1. Adatbázisrendszerek. Adatbázis, adatbázisrendszer, adatbázis-kezelő rendszer (DBMS) fogalma és jellemzői. Egyed, tulajdonság és kapcsolat fogalma és tulajdonságai. Relációs, objektum-relációs és NoSQL adatbázisok jellemzése. A funkcionális függés fogalma. Koncepcionális adatbázis-tervezés, az ER modell és leképezése relációs modellre. Az SQL elemei: DDL, DML, DCL, egyszerű lekérdezések és táblák összekapcsolása.

1. ADATBÁZISRENDSZEREK. ADATBÁZIS, ADATBÁZISRENDSZER, ADATBÁZIS-KEZELŐ RENDSZER (DBMS) FOGALMA ÉS JELLEMZŐI.

Adatbázis: Egymással logikailag összefüggő, egymáshoz kapcsolódó, belső jelentéssel bíró adatok összessége. Speciális célra tervezett, felépített és közzétett adatok együttese.

Adat: Ismert tény, számszerűsíthető és implicit jelentése van.

Adatbázisrendszer: DBMS szoftver az adatokkal együtt, néha az alkalmazásokat is beleértjük. Magába foglalja az adatbázisokat, a számítógépes erőforrásokat, tágabb értelemben adatbázis adminisztrátorok (DBA). DBA: azok a személyek, akik az adatbázis kezelésért, tervezéséért felelősek. Metaadat leírásokat tartalmaz: információk az adatokról.

Adatbázis-kezelő rendszer: (DBMS) Olyan szoftvercsomag/rendszer, amely számítógépes adatbázisok létrehozását/karbantartását támogatja. Megkönnyítik az adatbázisok kezelését. Feladatuk az adatbázisban lévő adatok rögzítése, tárolása, kezelése.

- Adatbázisok létrehozása
- Adatbázisok tartalmának definiálása (adattípusok, szerkezet)
- Adatok tárolása/betöltése
- Adatok lekérdezése (keresés/kinyerés)
- Adatok módosítása (belső műveletek végrehajtása feldolgozáskor)
- Adatok védelme
- Adatok titkosítása
- Hozzáférési jogok kezelése, adatok szolgáltatása konzisztens módon
- Fizikai adatszerkezet szervezése
- Adatbázis karbantartása

Fajtái:

- Centralizált: egy rendszerbe egyesít mindent: szoftver, hardver, alkalmazói program. A felhasználók terminálon keresztül tudnak kapcsolódni
- Kliens-szerver alapú: több különböző célfeladatra dedikált szerverekből és kliensekből áll. Kliensek szükség szerint érik el a dedikált szervereket
- Három-rétegű architektúra: webalkalmzások is ezt használják. Előző két réteg kiegészül egy közbenső webszerver réteggel.

Szerver: futtatja a lekérdezéseket, módosítja az adatokat, stb…

Kliens: Interfészek használta a műveletek végrehajtásához

2. EGYED, TULAJDONSÁG ÉS KAPCSOLAT FOGALMA ÉS TULAJDONSÁGAI

Egyed (entitás): A valós világ azon elem, amelyet lemodellezünk az adatbázisunkban, jellemzőit el szeretnénk tárolni.

Egyedtípus: Azonos tulajdonságtípusokkal rendelkező egyedek ***absztrakciója***.

Egyed-előfordulás: A valós világ egy konkrét darabja, amely a tulajdonságai alapján valamely egyedtípusba besorolható, vagyis az adott egyed-típusnak egy ***konkrét példánya***.

Tulajdonság (attribútum): Az egyed olyan jellemzője, amely modellezés szempontjából lényeges. Az egyed az attribútumok összességével jellemezhető. Az egyed típus az egyedre vonatkozóan megadott tulajdonságok összessége. Például egy személy leírható a nevével, a születési dátumával, testmagasságával, haja és szeme színével együttesen.

Tulajdonságtípus: Azonos szerepű tulajdonságok absztrakciója

- egyszerű/atomi vagy összetett
- egyértékű vagy halmazértékű/többértékű
- tárolt vagy származtatott
- speciális: NULL

Tulajdonság-előfordulás: az egyed olyan valós tulajdonsága, amely megfeleltethető valamely tulajdonságtípusnak

Kapcsolat: az egyedek közötti összefüggést, viszonyt.

Kapcsolattípusok: Két (bináris) vagy több egyedtípus közötti jól meghatározott viszony
Bináris kapcsolatok fajtái:

- 1:1: Egy–egy típusú kapcsolat esetén az egyik egyed minden egyes előfordulásának a másik egyed pontosan egy előfordulása tartozik. (pl.: hallgató – Neptun kód kapcsolata: a Neptunkód egyedien azonosítja a hallgatót)
- 1:N: A következő csoportot az egy-sok típusú kapcsolatok alkotják. Ezeknél az egyik egyed minden előfordulásához a másik egyed több előfordulása tartozhat. (pl.: évfolyam – hallgató kapcsolata: egy évolyamba több hallgató tartozik, de egy hallgató csak egy évfolyamhoz tartozhat)
- N:M: A kapcsolatok legáltalánosabb formáját a sok-sok kapcsolatok jelentik. Sok-sok kapcsolat esetén mindkét egyed előfordulásaihoz a másik egyed több előfordulása tartozhat (pl.: hallgató – kurzus kapcsolata: egy hallgató járhat több kurzusra, ahogy egy kurzusra is több hallgató járhat)

Kapcsolat-előfordulás: Két valós egyed közt fennálló valós kapcsolat, mely megfeleltethető valamely kapcsolattípusnak

3. RELÁCIÓS ADATBÁZIS

Elemei:

- Reláció neve
- Attribútumok halmaza
- Attribútumok adattípusa
- Kulcsok meghatározása

- Érvényességi feltételek

Formális definíciója:

$R(A_1, A_2, \ldots A_N)$,

R: reláció neve

A_x: attribútumok

Reláció: Értékek egy táblázata, amely sorok (rekordok) egy halmazából áll.

A relációs adatmodell jellemzője, hogy az adatokat több, egymással összekapcsolt rendszerben ábrázolja. Az egyes relációkat egyedi névvel látjuk el. A relációk oszlopaiban azonos mennyiségre vonatkozó adatok jelennek meg. Az oszlopok névvel rendelkeznek, melyeknek a reláción belül egyedieknek kell lenniük, de más relációk tartalmazhatnak azonos nevű oszlopokat. Egy sor és oszlop metszésében található táblázat elemet mezőnek nevezzük, a mezők tartalmazzák az adatokat. A mezőkben oszloponként különböző típusú (numerikus, szöveges stb.) mennyiségek tárolhatók. A reláció helyett sokszor a tábla vagy táblázat, a sor helyett a rekord, az oszlop helyett pedig az attribútum elnevezés is használatos.

Tábla: A tábla, vagy adattábla egy kétdimenziós tábla, mely a logikailag szorosan összetartozó adatokat szemlélteti. A tábla oszlopokból és sorokból áll.

Rekord: A rekord, az adatbázis egy sora. Egy rekordban tároljuk az egymással összefüggő adatokat. A tábla sorai tartalmazzák az egyes tulajdonságok konkrét értékeit.

Attribútum: a tábla egy oszlopa. Minden egyes oszlop az adott dolog egy jellemzőjét jelenti, mely névvel és típussal van ellátva.

Elemi adat: Az elemi adatok, a tábla celláiban szereplő értékek, melyek az egyed konkrét tulajdonságai.

Reláció/tábla követelményei:

- Minden táblázat egyértelmű azonosítóval bír.
- A sorok és oszlopok metszéspontjában található adatok egyértékűek, ezeket nevezi elemi adatmezőknek.
- Az oszlopokban lévő adatok azonos jellegűek.
- Minden oszlopnak egyedi neve van.
- Ugyan annyi adat található a táblázat minden sorában.
- Nem lehet két azonos sor a táblázatban.
- A sorok és oszlopok sorrendje tetszőleges.
- Nem tartalmaz más adatokból kiszámítható információt

Kulcs: A reláció olyan adateleme, amely egyértelműen azonosítja a sort, egyértelműen definiál egy előfordulást. Egy egyednek több kulcsa is lehet, de a legtöbb esetben egyet szokás kiválasztani, amely a leginkább alkalmas az egyértelmű azonosításra. Ezt hívjuk elsődleges kulcsnak. Kulcsjellegű tulajdonság mindig található. Ha a tényleges adatok között nem lenne ilyen, akkor bevezethetünk egy elsődleges kulcsot, ez többnyire szám szokott lenni, mivel azok összehasonlítása gyorsabb, mint a stringeké. Szám típusú kulcs esetén az

auto-inkrementálás is egyszerűbb. Az azonosítók általában számsorok, vagy (egyedi) véletlen generált karaktersorok.

- **elsődleges (PK — egyedi azonosító):** amit kiválasztunk a fő azonosítónak. Általában szám, mert gyorsabb az összehasonlítás és könnyű auto-inkrementálni.
- **kulcsjelölt**: ha több attribútum is alkalmas lenne azonosítónak (pl. Neptun és személyi szám). Egyet választunk elsődlegesnek, a többi marad jelölt.
- **szuperkulcs:** olyan attribútum-csoport, amelyet vizsgálva nincs két megegyező rekord. (Minden kulcs szuperkulcs is, de nem minden szuperkulcs minimális kulcs.)
- **idegen kulcs (FK — másik tábla PK-jára hivatkozik):** egy olyan oszlop, ami egy másik tábla elsődleges kulcsára hivatkozik. Így kapcsoljuk össze a táblákat.

Relációs modell megszorításai:

- implicit megszorítás (matematikai relációk): Szemantikus megszorítások
- explicit megszorítás (sémában meghatározott megszorítások)
    - tartománymegszorítás: minden rekordban az adott attribútumhoz tartozó értéknek a megadott tartományba kell tartoznia (v. NULL). Ezen tartomány elemei atomi értékek.
    - kulcsmegszorítás: PRIMARY KEY, NOT NULL, UNIQUE
    - NULL értékre vonatkozó megszorítás
    - egyedintegritási megszorítás: egyetlen elsődleges kulcsérték sem lehet NULL
    - hivatkozásintegritási megszorítás: Két reláció közt értelmezzük a konzisztencia érdekében: idegenkulcsok meghatározása FOREIGN KEY (…) REFERENCES …
- alkalmazásszintű megszorítások

Relációs műveletek:

- Reláció algebra: procedurális nyelv, eljárást adunk meg, a kinyerni kívánt információ előállítására, halmazelméletre alapszik
    - szelekció: feltételek (WHERE)
    - projekció: attribútum lista (SELECT)
    - átnevezés (AS)
    - halmazműveletek:
        - unió: UNION
        - metszet: INTERSECT
        - különbség: EXCEPT
    - összekapcsolás (JOIN)
    - csoportképző műveletek: SUM, MIN, MAX…
- Reláció kalkulus: nemprocedurális nyelv, deklaratív kifejezésekkel adjuk meg a kinyerni kívánt információt, predikátum kalkulusra alapszik
    - logikai kifejezések: AND, NOT, OR

4. OBJEKTUM-RELÁCIÓS ADATBÁZIS

OO szemlélettel bővített relációs adatbázisrendszerek (pl.: Oracle)

Típus konstruktorok: összetett objektumok: row type, mely egy tuple/struktúrát határoz meg

Objektum azonosítás REFERENCE TYPE segítségével

UDT: felhasználó által definiált típusok, egységbezárt műveletekkel

- összetett objektumok létrehozása
- **ROW**: rekord típusú konstruktor
- kollekciók: ARRAY, MULTISET, LIST, SET
- **CREATE TYPE** nev AS komponens_deklaracio
- **OID**: objektum azonosítására szolgál, rendszer generálja
- példányosított relációk létrehozása
- egységbezárás:
    - private
    - protected
    - public

Öröklődési mechanizmus: UNDER kulcsszó

- NOT FINAL kulcsszó, ha altípust szeretnénk majd készíteni később
- minden attribútum öröklődik
- altípus használható mindenhol, ahol szupertípus használható
- altípusban felülírhatóak a mechanizmusok, meghíváskor a legjobban illeszkedő implementáció fut le

5. NoSQL

Nem relációs, rugalmas séma, horizontális skálázás. Nem azt jelenti hogy “nincs SQL”, hanem “nem csak SQL”. Elsősorban webes alkalmazások, Big Data és valós idejű alkalmazások igényeire jött létre.

- Volume, Variety: nagy mennyiségű, változatos adat kezelése, beleértve a strukturálatlan és félig strukturált adatokat.
- Velocity: nagy sebességű adatfeldolgozás — gyors írási és olvasási teljesítmény.
- Horizontális skálázhatóság: könnyű bővítés több szerver hozzáadásával (scale-out), szemben a relációs vertikális skálázásával (erősebb szerverre váltás).
- Rugalmasság: gyors fejlesztés és gyakori adatmodell-változtatások a merev séma kényszer nélkül.
- CAP-tétel: Konzisztencia + Availability + Partition tolerance — max 2 garantálható egyszerre.

NoSQL fajtái:

- **Kulcs-érték**: A legegyszerűbb modell: minden adat egy egyedi kulcshoz tartozó értékként van tárolva. Az érték bármilyen típusú lehet. Nagyon gyors írás és olvasás kulcs alapján. Skálázhatóság sharding (particionálás) segítségével. Használat: gyorsítótárazás (caching), session-kezelés, egyszerű konfigurációk. NEM JÓ, HA: adatok alapján kell lekérdezni vagy többműveletes tranzakciók kellenek.

- **Oszlopcsalád**: Az adatokat oszlopcsaládokba szervezett sorokban tárolják. Egy soron belül nem kell minden oszlopnak léteznie — hatékony ritka/sparse adatokra. Nagyon jó írási teljesítmény és skálázhatóság. Map-of-maps-of-maps felépítésű. Egy kulcshoz több adat is tartozik, mindig tartalmaz időbélyeget. Használat: Big Data analitika, idősoros adatok, blogok, számlálók. Pl. Cassandra, HBase.
- **Dokumentum**: Az adatokat „dokumentumokként" tárolják — gyakran JSON, BSON vagy XML formátumban. Egy dokumentum összetett struktúrájú lehet: beágyazott kulcs-érték párok, tömbök, más dokumentumok. A dokumentumok „gyűjteményekbe" (collections) vannak rendezve. Rugalmas séma — a gyűjteményen belüli dokumentumok eltérő struktúrájúak is lehetnek. Skálázás sharding segítségével. Használat: tartalomkezelők, katalógusok, felhasználói profilok, mobil háttérrendszerek. Pl. MongoDB, CouchDB.
- **Gráf**: Az adatokat nodeok (csomópontok) és edges (élek) hálózataként modellezik. A csomópontok entitásokat, az élek a köztük lévő kapcsolatokat reprezentálják. Nagyon hatékonyan lehet kezelni a bonyolult, összekapcsolt adatokat. Lekérdezés = bejárás. Nehezen skálázható. Használat: közösségi hálózatok, ajánlórendszerek, csalásfelderítés, hálózati topológiák, tudásgráfok. Pl. Neo4j. NEM JÓ, HA: csomópontokat kell módosítani, vagy teljes gráfot érintő műveleteket kellene végezni.

NoSQL modellezés sajátosságai: Alkalmazás-orientált modellezés, az alapján szervezed az adatokat, hogyan lesz rájuk szüksége az alkalmazásnak. Adatszerkezetek és algoritmusok mélyebb megértését igényli.

Denormalizáció: ugyanazon adat több helyen történő tárolása/átmásolása a lekérdezés-feldolgozás egyszerűsítése érdekében.

Soft Schema: lehetővé teszi a komplex, belső szerkezettel rendelkező egyedek osztályokba történő szervezését, merev séma nélkül.

6. FUNKCIONÁLIS FÜGGÉS

Olyan megszorítás, amely az adatbázis két attribútumhalmaza közt áll fent. Ennek a megszorításnak bármely két rekord esetén teljesülnie kell. (pl.: X és Y attribútum halmaz értékei párban megegyeznek: ahol X megegyezik, ott Y is, tehát Y funkcionálisan függ X-től)

Ha ez mindkét irányba fennáll: kölcsönös funkcionális függésről beszélünk. (egy az egyhez kapcsolat) A funkcionális függőség jobb oldalán több attribútum is állhat. Például az AUTÓ_RENDSZÁM -> TIPUS, TULAJDONOS funkcionális függőség azt fejezi ki, hogy az autó rendszámából következik a típusa és a tulajdonos neve, mivel minden autónak különböző a rendszáma, minden autónak egy tulajdonosa és típusa van.

Funkcionálisan független: ha ez előbbi viszony a két tulajdonságtípus között nem áll fenn. Például: tanuló szemének a színe és az iskola helye

Tranzitív funkcionális függőség: ha az egyedtípuson belül egy leíró tulajdonságtípus konkrét értékei meghatároznak más leíró tulajdonság értékeit

Reflexivitás: egy attribútumhalmaz meghatározza önmagát, és saját magának egy részhalmazát

Augmentivitás: mindkét oldal ugyanazzal az attribútumhalmazzal történő bővítése újabb érvényes funkcionális függést eredményez

Tranzitivitás: a funkcionális függések tranzitívak

Dekompozíció: jobb oldalról eltávolíthatunk attribútumokat

Additivitás: függések egy halmaza összevontható egy nagy halmazzá

Armstrong-axiómák (teljesek és helyesek): Meglévő függésekből új függéseket tudunk levezetni.

1. **Reflexivitás:** ha Y része X-nek ($X \supseteq Y$), akkor $X \rightarrow Y$. Triviális — egy halmaz meghatározza a részhalmazait.
2. **Augmentivitás:** ha $X \rightarrow Y$, akkor $XZ \rightarrow YZ$. Mindkét oldal ugyanazzal bővíthető.
3. **Tranzitivitás:** ha $X \rightarrow Y$ és $Y \rightarrow Z$, akkor $X \rightarrow Z$.
4. **Dekompozíció:** $X \rightarrow YZ$ esetén $X \rightarrow Y$. Jobb oldalról eltávolíthatunk attribútumokat.
5. **Additivitás:** $X \rightarrow Y$ és $X \rightarrow Z$ összevonható: $X \rightarrow YZ$.
6. **Pszeudotranzitivitás:** $X \rightarrow Y$ és $WY \rightarrow Z$ esetén $WX \rightarrow Z$.

7. KONCEPCIONÁLIS ADATBÁZIS-TERVEZÉS

A koncepcionális adatbázis tervezés az a folyamat, amely során azonosítjuk és leírjuk a rendszer számára releváns entitásokat (objektumokat vagy fogalmakat), azok tulajdonságait (attribútumait), valamint az entitások közötti kapcsolatokat és azok típusait (1:1, 1:N, N:M), a valós üzleti vagy rendszerkövetelmények alapján. Az eredmény egy magas szintű adatmodell, gyakran entitás-kapcsolat diagram (ER diagram) formájában ábrázolva.

Nem hivatalos iránynyelvek:

- Minden egyes rekordnak egy egyed-előfordulást vagy kapcsolat-előfordulást kell reprezentálnia: könnyen értelmezhető szemantikájú séma
    - Más egyedet attribútumai nem keverhetők
    - Hivatkozás csak külső kulcs segítségével történhet
    - Egyedek és kapcsolatok attribútumait el kell különíteni
- A sémát úgy kell megtervezni, hogy ne jöjjenek létre beszúrási/törlési/módosítási anomáliák: figyelni kell a redundáns adatábrázolásra
- NULL értékek minimalizálása, ha sok NULL értékű attribútum fordul elő, érdemes lehet külön relációba tenni ezeket
- A rekordokat úgy kell kialakítani, hogy kielégítsék a veszteségmentes összekapcsolás feltételét:
    - összekapcsolási műveletek értelmesek legyenek, ne legyenek hibák
    - nem jöhetnek létre „álrekordok”

ER modell: Az ER (Entitás-Kapcsolat) modell egy magas szintű, koncepcionális adatmodell, amely a valós világ objektumait (entitásait) és a köztük lévő kapcsolatokat írja le.

Gyenge egyedtípus: Nem rendelkeznek saját kulcsattribútummal - részleges kulcs: attribútumok halmaza, amely egyértelműen azonosítja a gyenge egyedeket, amelyek ugyanazon tulajdonos egyedhez tartoznak

Erős egyedtípus: Hagyományos egyedtípus, rendelkezik kulcsattribútummal, amely egyedileg azonosítja

Tulajdonság típusok: - egyszerű és összetett - egyértékű és halmaz értékű - tárolt és származtatott

Kapcsolattípusok: Tetszőleges fokszámú kapcsolattípus ábrázolható

8. ER MODELL LEKÉPEZÉSE RELÁCIÓS MODELLRE

Erős egyedtípusok: Egyetlen tábla jön létre, az attribútumok oszlopok lesznek, az elsődleges kulcs változatlanul a tábla PK-ja.

Gyenge egyedtípusok: Külön tábla jön létre, amely tartalmazza a saját attribútumokat és a tulajdonos erős egyed PK-ját FK-ként.
Az elsődleges kulcs összetett: tulajdonos PK + részleges kulcs.

Bináris 1:1 kapcsolatok

a. KÜLSŐ KULCS: A leggyakoribb módszer. Az egyik tábla PK-ja bekerül a másikba FK-ként, UNIQUE megszorítással a 1:1 biztosítására.

b. ÖSSZEVONÁS: Ha a két entitás szorosan összefügg, és a 1:1 kapcsolat részvételi megszorítása mindkét oldalon teljes (azaz minden előfordulás részt vesz a kapcsolatban), a két entitástípus összevonható egyetlen táblába. Ennek a táblának az egyik eredeti entitás elsőleges kulcsa lesz a saját elsőleges kulcsa.

c. KERESZTHIVATKOZÁS/KAPCSOLÓ RELÁCIÓ: Külön tábla jön létre két FK-val. PK = a két kulcs kombinációja (vagy mesterséges kulcs), mindkettő FK UNIQUE. Akkor hasznos, ha van saját kapcsolat attribútum vagy nem teljes a kapcsolat.

Bináris 1:N kapcsolatok: A "sok" oldalon lévő entitás táblájához hozzáadjuk a "egy" oldalon lévő entitás táblájának elsőleges kulcsát, mint külső kulcsot.

Bináris N:M kapcsolatok: Új kapcsolótábla jön létre, amely mindkét entitás PK-ját tartalmazza FK-ként. Az elsődleges kulcs ezek kombinációja, és ide kerülnek a kapcsolat attribútumai is.

Többértékű attribútumok: Külön tábla szükséges, amely tartalmazza az eredeti entitás PK-ját (FK) és az attribútum értékét. Az elsődleges kulcs ezek kombinációja.

N-ed fokú kapcsolatok: Külön kapcsolótábla jön létre, amely az összes résztvevő entitás PK-ját tartalmazza FK-ként. Az elsődleges kulcs ezek kombinációja, és a kapcsolat saját attribútumai is ide kerülnek.

9. Az SQL elemei

- **SQL:** Structured Query Language; adatbázis-lekérdező nyelv, melyekkel adatbáziskezelő műveleteket hajtunk végre.
- **DDL:** Data Definition Language; az adatbázis struktúrájának létrehozására, módosítására, törlésére használjuk. Hogyan nézzen ki az adatbázis. **CREATE, ALTER, DROP, TRUNCATE**

- **DML:** Data Manipulation Language; tárolt adatok kezelése. **SELECT, INSERT, UPDATE, DELETE**
- **DCL:** Data Control Language; az adatbázis objektumokhoz való hozzáférés és engedélyek kezelése. Ki és milyen műveletet hajthat végre**. GRANT, REVOKE**

Egyszerű lekérdezések: Az adatbázisból történő adatkinyerés alapvető módja. Egy egyszerű lekérdezés jellemzően egyetlen táblából választ ki adatokat, esetleg szűrési feltételek és rendezés megadásával. ***SELECT oszlopok FROM tábla WHERE feltétel ORDER BY oszlop.***

Táblák összekapcsolása: Táblák összekapcsolásával (JOIN művelettel) tudjuk ezeket az elszórt, de kapcsolódó információkat egyetlen eredményhalmazba egyesíteni. Az összekapcsolás egy vagy több oszlopban fennálló kapcsolat (általában elsődleges kulcs és külső kulcs) alapján történik.

- INNER JOIN: Csak azokat a sorokat adja vissza mindkét táblából, amelyeknél van egyezés a megadott kapcsolódási feltétel alapján.
- LEFT JOIN (vagy LEFT OUTER JOIN): Visszaad minden sort a "bal" oldali táblából, és a hozzá tartozó egyező sorokat a "jobb" oldali táblából.
- RIGHT JOIN (vagy RIGHT OUTER JOIN): Visszaad minden sort a "jobb" oldali táblából, és a hozzá tartozó egyező sorokat a "bal" oldali táblából.
- FULL OUTER JOIN: Visszaad minden sort mindkét táblából.

2. Adattípusok. Változó. Műveletek, operátorok, operandusok. Vezérlési szerkezetek. Kifejezések. Utasítások. Programegységek. Paraméterkiértékelés, paraméterátadás. Blokk. Hatáskörkezelés, láthatóság. Absztrakt adattípus. Kivételkezelés.

1. ADATTÍPUSOK

Adattípusok: Absztrakció első megjelenési formája, absztrakt programozási eszköz. Léteznek típusos (Java) és nem típusos (Python) nyelvek.

Adattípust meghatározza:

- **Tartomány**: azok az elemek, amelyeket a típus felvehet
- **Műveletek**: amiket végrehajthatunk a típusokon
- **Reprezentáció**: ábrázolási mód a fizikai táron

Fajtái definíció alapján:

- Beépített
- Felhasználó által definiált:
    - Meg kell adni: tartomány, műveletek, reprezentáció
    - Általában beépített adattípusok segítségével történik a definiálás
    - alaptípusból származtathatunk altípust a tartomány leszűkítésével

Primitív adattípusok: Atomi értékeket vehet fel, értékek nem bonthatóak

Numerikusok:

- Egész (int, byte, short, long): fixpontos, előjeles, előjel nélküli
- Valós (float és double): lebegőpontos

Karakteres (char): Tartomány elemek: karakterek

Karakterlánc (string): Tartomány elemek: karakterekből álló sorozatok

Logikai (bool): Igaz/hamis

Felsorolás (enum): Meg kell adni a tartomány elemeit, amik azonosítók lehetnek

Összetett adattípusok: Tartomány elemei is típussal rendelkeznek. Az elemek egy-egy értékcsoportot képviselnek, nem atomiak, az értékcsoport elemeihez külön-külön is hozzáférhetünk. Általában valamilyen absztrakt adatszerkezet programnyelvi megfelelői

Tömb: Absztrakt adatszerkezet megjelenése típusos szinten. Statikus és homogén elemek, melyek azonos típusúak, rendelkezik dimenziókkal, indexelt, meg kell adni az elemek típusait, sor- vagy oszlopfolytonosan reprezentált a memóriában.

Rekord (struct): Absztrakt adatszerkezet megjelenése típusos szinten. Különbség a tömbtől: heterogén elemek, ezeket az elemeket rekordonként mezőknek hívjuk. Mező: önálló neve és típusa van

Mutató: Tartomány elemei tárcímek, indirekt címzésre használható. Speciális eleme: NULL Szétszórt reprezentációjú adattípusok (pl. láncolt lista) esetén használjuk

2. VÁLTOZÓ

Nevesített konstans: név + típus + érték. Nem módosítható az értéke, deklarálni kell. Névvel hivatkozunk rá, viszont mindig az értékkomponenst kapjuk vissza.

Változó: egy elnevezett, absztrakt tárolóhely a számítógép memóriájában, amely adatokat tárol. Részei: név, memóriacím, típus, érték, élettartam, hatáskör.

- **Explicit deklaráció:** A programozó végzi deklarációs utasítás segítségével.
- **Automatikus deklaráció**: a deklaráció nem egy külön, dedikált utasítással történik, hanem a változó első használatakor vagy értékadásakor jön létre a változó, és a fordító vagy interpreter következtet a típusára.

Cím rendelése változóhoz:

- **Statikus**: futás előtt eldől, és a futás alatt nem változik
- **Dinamikus**: a cím hozzárendelést a futtató rendszer végzi. Amikor értéket kap, aktiválódik, és egy címet kap. Ha program befejeződik, akkor törlődik. Futás közben változhat
- Programozó által vezérelt:
    - abszolút címmel: megadja, hogy konkrétan hol helyezkedik el
    - relatív címmel: korábban elhelyezett programozási eszköz helyéhez képest
    - időintervallum megadásával: megadja, hogy mikortól létezzen, címkiosztást a futtató rendszer végzi
    - MINDIG KELL ESZKÖZ, AMI A CÍMEKET MEGSZÜNTETI

Értékkomponens meghatározása:

- **Értékadó utasítással**: leggyakoribb
- **Input**: perifériából kapott adat alapján
- **Kezdőértékadás**: explicit értékadáskor a programozó adja meg a deklarációs utasításban

Amíg nem adunk értéket, addig az értékkomponens határozatlan, nem használható a változó értéke. Vannak nyelvek, amelyek kezdőértéket rendelnek a változóhoz.

3. KIFEJEZÉSEK

Szintaktikai eszközök, amelyekkel ismert értékekből új értéket határozunk meg.

Formális összetevők: Operandusok (érték) + operátorok (művelet) + zárójelek.

- **Operandusok:** literál, nevesített konstans, változó, függvényhívás. Mindig értéket képvisel. Operandusok száma szerint lehet unáris (-x, !b), bináris (a+b), ternáris.
- **Operátorok:** műveleti jelek (+, -, *, /, ==, &&, !, stb.) Operátorok elhelyezkedése szerint lehet: prefix (*3 5), infix (3 * 5 — ezt használjuk a legtöbb nyelvben), postfix (3 5 *).
- **Kerek zárójelek:** a kiértékelési sorrendet befolyásolják.

Kifejezés kiértékelése: különböző sorrendben történhet. Precedencia táblázat: leírja a kiértékelés irányát

Típus egyenértékűség: amikor csak azonos típusú operandusok jelenhetnek meg a kifejezésben, nincs konverzió.

Típuskikényszerítés (koerzió): ha a kifejezés elvárja a típusegyenértékűséget, így kényszerítjük ki.

- Szűkítés: értékvesztés áll fent, pl.: kerekítés történik
- Bővítés: nincs értékvesztés, a konvertálandó típus tartományának minden eleme elme a céltípus tartományának is

4. UTASÍTÁSOK

- **Deklarációs:** Fordítóprogramnak szólnak, nem kerülnek lefordításra.
- **Végrehajtó:** Tárgykódot ebből generálja a fordítóprogram
    - **Értékadó:** Beállítja/Módosítja egy/több változó értékkomponensét
    - **Üres utasítás**
    - **Ugró utasítás:** feltétel nélküli vezérlésátadó utasítás. Kerülendő, spagetti kód. (GOTO)
    - **Elágaztató:** IF-ELSE (kétirányú, két tevékenység közül egyet választunk logikai feltétel alapján), SWITCH CASE DEFAULT (többirányú, egymást kölcsönösen kizáró tevékenységek közül egyet választunk).
    - **Ciklusszervező:** fej + mag + vég. Tevékenységek ismétlésére szolgálnak. Ismétlésre vonatkozó információk (ciklusparaméterek) fejben v. végben, ismétlendő utasítások a magban. Speciális esete a végtelen ciklus és az üres ciklus.
        - **Feltételes ciklus:** WHILE és DO-WHILE; az ismétlődés feltétel határozza meg, a WHILE kezdőfeltételes, a DO-WHILE pedig végfeltételes.
        - **Előírt lépésszámú:** FOR, A ciklusparaméter a fejben, ciklusváltozó által felvett értékekre fut le. Megadott tartomány, megadott lépésköz.
        - **Felsorolás:** FOREACH, az előírt lépésszámú ciklus általánosítása. Minden értékre lefut a mag, nem lehet üres vagy végtelen.
        - **Végtelen ciklus:** eseményvezérelt alkalmazásoknál hasznos, egyébként kerülendő. Nem adunk meg benne ismétlésre vonatkozó információt.
        - **Összetett ciklus:** Több ciklusfejet tartalmaz, az előzőekből állítjuk össze.
        - **Hívó:** Függvényhívás/eljáráshívás: elindítja az előre definiált kódrészletet, aktiváljuk az adott kódblokkot
    - **Vezérlésátadó utasítások**:
        - Continue: ciklusmagban alkalmazandó. Vagy újabb ciklusléphésbe kezd, vagy befejezi a ciklust.
        - Break: Kilép egy ciklusból, többszörös elágaztató utasításokból.
        - Return [kifejezés]: szabályosan befejezteti a függvényt és visszaadja a vezérlést és egy értéket a hívónak.

5. PROGRAMEGYSÉGEK

Eljárásorientált programnyelvekben a kód független programegységekre bontható. Háromféle szervezés:

1. Fizikailag önálló, külön-külön fordítható részek, melyek mélységükben nem strukturáltak.
2. A program egy egységben fordítható és mélységében strukturált.
3. Az előbbi kettő kombinációja: fizikailag független, belső struktúrával rendelkező programegységek.

Alprogram: A programegység egyik formája. Bemeneti adatcsoportot képez le kimeneti adatcsoportra. Ismerjük a specifikációt, de nem ismerjük az implementációt. Az újrafelhasználás alapvető eszköze: elég egyszer megírni, ezután csak meghívjuk. Komponensei: név, formális paraméterlista, törzs (deklarációs és végrehajtási utasítások szerepelnek), környezet (globális változók összessége).

Alprogram fajtái:

- **Eljárás**: tevékenységet hajt végre, megváltoztatja a környezetét a megadott paraméterek segítségével, nincs visszatérési értéke.
- **Függvény**: értéket ad vissza. Ennek az értéknek a típusa tetszőleges. Mellékhatásnak nevezzük azt, ha a függvény megváltoztatja a paramétereit vagy környezetét.

Hívási lánc: Egy programegység meghívhat egy másikat, így alakul ki a hívási lánc. A hív B-t, B hív C-t. Minden tagja aktív, de csak az utolsó dolgozik, a többi vár. A lánc első tagja, a főprogram fejeződik be utoljára.

Rekurzió: az aktív alprogram meghívja önmagát (közvetlen) vagy a láncban már szereplő aktív programot (közvetett). Átírható iteratív algoritmussá.

Blokk: Programegység, amely csak másik programegység belsejében helyezkedhet el, ott bárhol, ahol végrehajtó utasítás állhat. Formális részei: kezdet/vég jelzés + törzs (deklarációs és végrehajtó utasítások). Nincs paramétere, egyes nyelvekben lehet neve. Pl. C-ben a { ... } blokk.

6. PARAMÉTERKIÉRTÉKELÉS, PARAMÉTERÁTADÁS

Paraméterkiértékelés: Az a folyamat, amikor a formális és aktuális paraméterek egymáshoz rendelődnek. Formális paraméterlista egy adott alprogramhoz mindig ugyanaz, egyszer definiáljuk, míg az aktuális paraméterek változhatnak: más, akárhányszor meghívjuk az alprogramot.

Paraméterátadás:

- **Érték szerinti (pass by value):** Formális paraméterek rendelkeznek egy címkomponenssel az alprogram területén. érték másolódik. Az aktuális paraméter értékkomponense meghatározódik, és átmásolásra kerül a formális paraméter címkomponensére. Ezzel dolgozik az alprogram a saját területén. Ez a folyamat egyirányú. A két programegység egymástól függetlenül működik — mellékhatás nélküli. Az aktuális paraméter lehet kifejezés is.
- **Cím szerinti (pass by reference):** a formális paraméternek nincs saját címkomponense. Az aktuális paraméter címkomponense adódik tovább — ez lesz a formális paraméter címe. Az alprogram a hívó területén dolgozik. Kétirányú, mivel a hívott írhat a hívó területére. Gyors (nincs értékmásolás), viszont a hívott hozzáfér a

hívó információihoz, és nemkívánt mellékhatáshoz vezethet. Az aktuális paraméter csak változó lehet.

- **Eredmény szerinti:** mindkét paraméternek van címkomponense. Az aktuális paraméter címkomponense átadódik az alprogramnak, de az alprogram a saját területén dolgozik. A futás befejezésekor a paraméter értékét átmásolja a hívótól kapott címre. Egyirányú (kifelé), van értékmásolás. Az aktuális paraméter csak változó lehet
- **Érték-eredmény sz.:** a formális paraméternek van címkomponense, az aktuálisnak cím és értékkomponense is. Kiértékeléskor mindkettő átadásra kerül. A hívott a kapott értékkel kezd dolgozni a saját területén. Futás befejeződése után átmásolja az értéket az aktuális paraméter címére. Kétirányú, van értékmásolás. Az aktuális paraméter csak változó lehet
- **Név szerinti:** az aktuális paraméter egy tetszőleges szimbólumsorozat egy szövegkörnyezetben. A szövegkörnyezet rögzítésre kerül, az aktuális paraméter értelmezésre kerül, majd a szimbólumsorozat átírásra kerül: minden formális paraméter neve helyére az aktuális paraméter kerül. Ezután fut le az alprogram.

7. HATÁSKÖRKEZELÉS, LÁTHATÓSÁG

Határkör: A hatáskör (scope) a programszöveg azon része, ahol egy név ugyanarra a programozási eszközre hivatkozik. Szinonimája a láthatóság.

- **Lokális:** Programegységben deklarált név.
- **Szabad**: Olyan név, amelyet nem a programegységben deklarálunk, de hivatkozunk rá.
- **Globális**: nem lokális, de látható.

Statikus hatáskörkezelés: fordítási időben, fordítóprogram által történik, programszerkezet alapján. Befelé terjed a hatásköre. Forrásszöveg alapján megállapítható egy név hatásköre. Legtöbb modern nyelv ezt használja. A lokális név hatásköre az a programegység, melyben deklaráltuk, és minden olyan programegység, amelyet ez az egység tartalmaz

Dinamikus hatáskörkezelés: futási időben, hívási lánc alapján történik. Ha a rendszer talál egy szabad nevet, addig lépked vissza a hívási láncon, amíg meg nem találja egy programegység lokális neveként, vagy a lánc elejére nem ér. Ha nem találja: futásidejű hiba vagy automatikus deklaráció. A nevek hatásköre futásonként és futásidőben változhat. A név hatásköre a programegység, amelyben deklaráltuk, és minden olyan egység, amely ezen programegységből induló hívási láncon szerepel.

8. ABSZTRAKT ADATTÍPUS

Olyan adattípus, amely megvalósítja a bezárást/információrejtést. Nem ismerjük a reprezentációt és a műveletek implementációt (ezeket nem mutatja meg a külvilágnak). Értékeihez csak szabályozott módon lehet hozzáférni: interfészeken keresztül, műveleteinek specifikációi által.

9. KIVÉTELKEZELÉS

Lehetővé teszi, hogy az operációs rendszerektől átvegyük a megszakítások kezelését.

Kivételek: olyan események, amelyek megszakítást okoznak - van nevük és kódjuk. Egyes kivételek figyelése letiltható/engedélyezhető

Kivételkezelés: az a tevékenység, amelyet a program végez, ha egy kivétel következik be.

Kivételkezelő: olyan programrész, amely működésbe lép egy kivétel kezelése után. Lehetővé teszi az eseményvezérlést.

A legtöbb nyelv támogatja a kivételkezelést speciális blokkokkal:

- try: Ide kerül az a kódrészlet, amely kivételt dobhat.
- catch vagy except: Ide kerül az a kód (a kivételkezelő), amely akkor fut le, ha a try blokkban egy meghatározott típusú kivétel keletkezett. Több catch/except blokk is lehet különböző kivételtípusokra.
- finally: Ez a blokk mindig lefut, akár történt kivétel a try blokkban, akár nem, és akár kezelték azt, akár nem. Ideális hely a források felszabadítására.

A program nem csak "elkaphat" kivételeket, de explicit módon is "dobhat" (throw) kivételeket egy throw paranccsal, ha olyan hibaállapot vagy speciális eset következik be, amit a hívó kódnak kell kezelnie.

3. Az objektumorientált paradigma alapfogalmai. Osztály, objektum, példányosítás. Öröklődés, osztályhierarchia. Polimorfizmus, metódustúlterhelés. A bezárási eszközrendszer. Absztrakt osztályok és interfészek. Típustagok.

1. AZ OBJEKTUMORIENTÁLT PARADIGMA ALAPFOGALMAI

Az objektumorientált (OO) paradigma középpontjában a programozási nyelvek absztrakciós szintjének növelése áll. Ezáltal egyszerűbbé, könnyebbé válik a modellezés, a valós világ jobban leírható, a valós problémák hatékonyabban oldhatók meg. Az OO szemlélet szerint az adatmodell és a funkcionális modell egymástól elválaszthatatlan, külön nem kezelhető. Alapját az absztrakt adattípusok jelentik. (osztályok fogják megvalósítani az absztrakt adattípust az OO nyelvekben)

Az egységbezárás elve: A valós világot egyetlen modellben kell leírni és együtt kell kezelni a statikus és dinamikus jellemzőket.

- Statikus jellemzők: adat
- Dinamikus jellemzők: viselkedés

Absztrakt adattípus: Olyan nyelvi egység, amely megvalósítja az egységbe zárás elvét, elrejti a típus reprezentációját. Használatával a programozó lehetőséget kap arra, hogy magasabb szinten gondolkodjon a programról, ne az alacsony szintű megvalósítási kérdések legyenek a meghatározók.

Objektum: A változó fogalmának kiterjesztése. Komponensei és tulajdonságai:

- **Attribútumok (adatrész):** az objektum statikus része. Tetszőleges bonyolultságú adatszerkezet. Az objektum számára definiált tárterületen találhatóak a bitsorozatok.
- **Terminológia**: az objektumnak állapota van, minden állapot egy bitkombinációnak felel meg.
- **Metódusok (módszerek):** az objektum viselkedését írják le. Az eljárásorientált nyelvek függvényei, eljárásai OO kontextusban. Képesek lekérdezni és módosítani az objektum aktuális állapotát.
- **Azonosság:** bármely objektum csak önmagával azonos. Minden objektum rendelkezik OID-dal (Object ID).
- **Analógia:** A változó neve nem azonosítja az objektumot, csak hatáskörön belül jelent egyértelmű hivatkozást. Az OID alkalmazások közt is egyedi.

Objektumot létre kell hozni (explicit tevékenység), addig él amíg meg nem szűnik

- megszüntetés történhet nyelvi szinten (automatikusan) vagy végezheti a programozó (explicit törlés)
    - automatikus törlési folyamatért egy mechanizmus (garbage collector) felelős, amely automatikusan felszabadítja a tárhelyet
- OID mindig létezik, minden szinten

Osztály: Absztrakt eszköz, típusfogalmak általánosítása (típusnak is nevezzük). Azonos attribútumú és módszerű objektumok együttese, belőlük származtathatóak az objektumok.

Példányosítás: Az a folyamat amikor létrehozok egy osztályból egy objektumot, ekkor egy példány jön létre (egy adott osztály egy adott objektuma). Minden példány ugyanolyan attribútumokkal és módszerekkel rendelkezik, a módszerek mindig az aktuális példányra futnak le. Példányosításkor az adatszerkezet jelenik meg újra és újra a tárban, módszerek nem többszöröződnek.

Osztályattribútum, osztálymódszer: nem a példány viselkedését/tulajdonságát írja le, hanem az egész osztályét (pl.: hány példánya van az osztálynak)

Először az osztályt kell létrehozni (attribútumok, metódusok leírása), ez után hozhatunk létre példányokat (tehát konkrét objektumokat)

2. ÖRÖKLŐDÉS

Az újrafelhasználhatóságot valósítja meg. Az osztályok nem függetlenek egymástól, speciális viszony van köztük (öröklődés).

Szuperosztály és alosztály(ok): alosztály(ok) átveszi(k) a szuperosztály attribútumait, módszereit (mindet, amelyet megenged a láthatóság). az alosztály ezen felül:

- új attribútumokat, módszereket hozhat létre
- újra implementálhat módszereket
- törölhet attribútumokat, módszereket
- újra értelmezheti a láthatósági szabályokat
- átnevezhet attribútumokat
- duplikálhat attribútumokat és módszereket

Az öröklődés aszimmetrikus: szuper osztály nem láthatja/manipulálhatja alosztályait, de fordítva igen. Minden alosztály objektuma a szuperosztály objektuma is, fordítva nem igaz (tehát mindenhol, ahol szuperosztály példánya szerepelhet, ott szerepelhet alosztály példánya is). Egy szuperosztályhoz több alosztály tartozhat.

- **Egyszeres öröklődés:** egy alosztálynak egy szuperosztálya van (Java, C#).
- **Többszörös öröklődés:** egy alosztály több szuperosztálytól örökölhet (C++, Python). Problémák: attribútum-névütközés, gyémánt probléma.

Osztályhierarchia / öröklődési hierarchia: Egymásból örököltetett osztályok hierarchiája.

- **Öröklődési fa:** egyszeres öröklődés esetén.
- **Öröklődési körmentes gráf:** többszörös öröklődés esetén.
- **Ősosztály:** a fa gyökere, melyből minden más elem származik (Java-ban: Object).
- **Előd:** szülő osztály és azok szülőosztályai...
- **Leszármazott:** egy osztály alosztályai, és azok alosztályai...
- **Kliens osztályok:** osztályok, melyek közt nincs öröklődési kapcsolat.

3. POLIMORFIZMUS, METÓDUSTÚLTERHELÉS

Objektumpolimorfizmus: Minden objektum tudja magáról, hogy mely osztály példányaként jött létre. Egy objektum példánya a saját osztályának, de az öröklődési hierarchiában objektuma valamennyi elődosztálynak is egyben. - objektum szerepelhet minden olyan helyen, ahol az ősosztály objektumai is szerepelhetnek

Módszerpolimorfizmus: Egy alosztály újraimplementálhat módszereket: a specifikáció változatlan, de az implementáció megváltozik. Pl.: ZártAlakzat.terület() más, mint Háromszög.terület(), mindkettőnek van terület metódusa, de másképp számolják.

Kötés: egy függvény több implementációval

- **Statikus (korai) kötés:** fordítási időben történik a hozzárendelés. Gyors. A metódustúlterhelés (overloading) is ide tartozik.
- **Dinamikus (késői) kötés:** futási időben történik, attól függően, hogy melyik osztálykörnyezetben dolgozunk. Az aktuális osztályban definiált kód fut le — vagy ha ilyen nincs, a hierarchián legközelebb lévő.

Metódustúlterhelés: statikus (fordítási idejű) polimorfizmus forma. Azt jelenti, hogy egy azon osztályon belül több metódusnak is ugyanaz a neve, de a paraméterlistájuk különbözik (más a paraméterek száma vagy típusa). A fordítóprogram a híváskor használt argumentumok alapján dönti el, hogy a túlterhelt metódusok közül melyiket kell meghívni.

4. A BEZÁRÁSI ESZKÖZRENDSZER – ENCAPSULATION

Hatáskör fogalmának kiterjesztése, nem objektumhoz kapcsolódó fogalom

1. Definíció: olyan eszközrendszer, amellyel szabályozható, hogy egy osztályból mi látható kívülről és mi nem

2. Definíció: Az osztály olyan absztrakt adattípus, melynek két része van: interfész és implementáció. Az osztály objektumaihoz kizárólag az interfészen keresztül férünk hozzá, így az implementáció csak korlátozottan hozzáférhető (információ elrejtés elve). Ez az attribútumokat és a módszereket két részre osztja: nyilvános rész, privát rész.

Ezt láthatósági szintekkel valósítjuk meg:

- **public:** mindenhonnan elérhető.
- **protected:** csak az osztályból és leszármazottaiból.
- **private:** csak az osztályon belülről.

5. ABSZTRAKT OSZTÁLYOK ÉS INTERFÉSZEK

Absztrakt osztály: Olyan osztályok, melyeknek nincsenek példányaik, nem példányosíthatóak. Örököltetésre valóak, tervezéskor használatosak. Fő jellemzőjük, hogy tartalmaznak absztrakt módszereket is.

Absztakt módszerek: Nincs implementációjuk, csak specfikicációjuk. Későbbi örököltetett nem-absztrakt osztályokban implementálni kell őket.

Interfész: Teljesen absztrakt osztály, csak absztrakt függvényekkel rendelkezik.

6. TÍPUSTAGOK

Típustag: Ez egy gyűjtőfogalom azokra az elemekre, amelyek egy típus (pl. egy osztály vagy interfész) szerkezetét és viselkedését alkotják.

Mezők/attribútumok: Ezek a változók tárolják egy objektum állapotát. Minden egyes példány (objektum) rendelkezik a saját, külön mezőértékekkel (kivéve a statikus mezőket).

Metódusok: Ezek a függvények vagy eljárások definiálják egy objektum viselkedését. A mezőkön operálnak, vagy az objektummal kapcsolatos valamilyen műveletet végeznek.

Tulajdonságok: Olyan tagok, amelyek olvasási vagy írási hozzáférést biztosítanak egy mezőhöz, gyakran úgy, hogy közben valamilyen extra logikát is végrehajtanak (pl. validáció). Külsőleg úgy viselkednek, mint mezők, de valójában getter/setter metódusok vagy hasonló mechanizmus áll mögöttük.

Konstruktor: Osztály nevével megegyező függvény, amely a példányosítás során hívódik meg. Célja az objektum inicializálása, és adatokkal történő feltöltése. Nincs visszatérési értéke.

- **Default konstruktor**: üres, paraméter nélküli konstruktor.
- **Paraméterezett konstruktor**: paramétereket vár, melyek általában adatok.
- **Másoló konstruktor**: sekély másoláskor a két objektum ugyanazt a tárterületet használja, mély másoláskor két különböző tárterületet használ

Destruktor: Osztály nevével megegyező függvény, neve előtt ~ karakter áll. Objektum törlésekor fut le.

4. Operációs rendszerek fogalma, felépítése, osztályozásuk. Fájlok és fájlrendszerek. Speciális fájlok Unix alatt. Átirányítás, csővezetékek. Folyamatkezelés. Jelzések, szignálok. Ütemezett végrehajtás.

1. OPERÁCIÓS RENDSZEREK FOGALMA, FELÉPÍTÉSE, OSZTÁLYOZÁSUK

Operációs rendszer: egy rendszerszoftver, a hardver és alkalmazások közti réteg (Hardver > OS > Alkalmazások). Két fő szerepe:

- **Erőforrás-kezelő**: megadott erőforrások kezelése, feladatok priorizálása o CPU, memória, I/O
- **Irányító program**: programok végrehajtása, szolgáltatások/interfész nyújtása a felhasználónak

Felépítése: Hierarchikus felépítésű, rétegekből áll, egymással együttműködve biztosítják a hatékony működés.

1. **Kernel (rendszermag):** Az OS legfontosabb része közvetlenül kezeli a kritikus hardvererőforrásokat és kiszolgálja a felhasználói kéréseket. Feladatai: folyamatkezelés, memóriakezelés, eszközkezelés (I/O), fájlrendszer kezelése, rendszerhívások kezelése
2. **Eszközillesztők, könyvtárak (API):** Szoftvermodulok, melyek hardvereszközökkel való kommunikációt biztosítja a kernel és a UI között rendszerhívásokkal.
3. **Rendszer héj (Shell) és felhasználói felület (UI):** Feladata a parancsértelmezés. A UI feladata a felhasználói interakciók kezelése és az adatok megjelenése. A felhasználói felület lehet grafikus (GUI) illetve szöveg alapú (CLI – Command Line Interface).
4. **Segédprogramok (Utilities):** a felhasználói "élményt" fokozó kiegészítő programok (pl szövegszerkesztők, fordítóprogramok), amelyek nem képzik a rendszer elválaszthatatlan részét.

Osztályozásuk:

Felépítés szerint:

- **Monolitikus kernel**: Minden szolgáltatás (folyamatkezelés, memóriakezelés, fájlrendszer, eszközkezelők) egyetlen nagy kernel kódban fut privilegizált módban. Gyors, de nehéz fejleszteni és karbantartani.
- **Mikrokernel**: Minimális funkcionalitást (folyamatkommunikáció, alapvető memóriakezelés) tartalmaz a kernelben, a többi szolgáltatás (fájlrendszer, eszközkezelők, hálózat) felhasználói térben futó szerverfolyamatokként. Modulárisabb, megbízhatóbb lehet, de lassabb.
- **Hibrid kernel**: Monolitikus és mikrokernel elemeket kombinál. (Pl. Linux, Windows).

Felhasználói szám:

- **Egyfelhasználós**: Egyszerre csak egy felhasználó tud aktívan bejelentkezni és dolgozni a rendszeren. (DOS)

- **Többfelhasználós**: Lehetővé teszi több felhasználó számára, hogy egyidejűleg vagy közel egyidejűleg használják ugyanazt a rendszert. (Windows, Linux)

Feladatok száma:

- **Egyfeladatos**: Egyszerre csak egyetlen program vagy folyamat futhat a rendszeren a CPU-n. (DOS)
- **Többfeladatos (Multi-tasking):** Képes több program vagy folyamat futását látszólagosan (szimuláltan) párhuzamosan kezelni a CPU idő szeletelésével és megosztásával. (Unix)

Felhasználói felület:

- **Parancssori interfész/Shell**: A felhasználó szöveges parancsokat gépel be a rendszerrel való interakcióhoz. (DOS)
- **Grafikus (GUI):** A felhasználó vizuális elemek (ikonok, ablakok, menük) segítségével, jellemzően egérrel és billentyűzettel lép interakcióba a rendszerrel. (Windows)

Feldolgozók száma:

- **Egyprocesszoros:** A rendszer egyetlen központi feldolgozó egységet (CPU-t) használ.
- **Többprocesszoros (SMP – Symetric Multiprocessing):** A rendszer több, azonos képességű CPU-t használ, amelyek közös erőforrásokon osztoznak a feladatok párhuzamos futtatásához.
- **Elosztott rendszerek:** Több, hálózatba kötött, független számítógép működik együtt egyetlen logikai egységként a felhasználó számára.

Működés módja:

- **Batch (kötegelt) feldolgozás**: A rendszer egymás után, felhasználói interakció nélkül futtat előre összeállított feladatcsomagokat.
- **Időosztásos rendszerek**: A rendszer gyorsan váltogat a több, interaktív felhasználó vagy feladat között a CPU-n, válaszkész környezetet biztosítva.
- **Valós idejű rendszerek**: A rendszer szigorú, garantált időzítési korlátokon belül hajtja végre a feladatokat, ami kritikus rendszerekhez elengedhetetlen.

Terjesztési licensz:

- **Kereskedelmi**: A szoftver használatáért jellemzően fizetni kell, és a forráskód általában nem érhető el a felhasználók számára. (DOS, Windows)
- **Szabad**: A szoftver általában ingyenesen használható, terjeszthető, a forráskód elérhető, és gyakran módosítható is bizonyos feltételek mellett. (Unix)

2. FÁJLOK ÉS FÁJLRENDSZEREK

Fájl: A másodlagos tárolón tárolt információk logikai egysége. Az OS absztrakt nézetet nyújt róla (byte-ok sorozata), elrejtve az adatok fizikai elhelyezkedésének részleteit.

Attribútumai:

- Név: Emberileg olvasható
- Kiterjesztés: nem kötelező, DOS esetén maximum 3 karakter lehet. Általában a fájl jellegére utal.
- Hely: Hol van tárolva a lemezen
- Időbélyegek: fájl létrehozásának, utolsó módosításának és utolsó hozzáférésnek dátuma.
- Méret: a fájl mérete bájtokban.
- Védelem / jogosultság / hozzáférés

Műveletek: létrehozás, másolás, törlés, megnyitás/bezárás, olvasás/írás.

Fájltípusok:

- **Adatfájlok:** Ez a leggyakoribb típus, ide tartozik a legtöbb "normál" fájl. Tartalmazhatnak szöveget, képeket, zenét, videót, dokumentumokat, programok által generált adatokat stb. Ezek általában csak byte-ok sorozatai, az OS maga nem feltétlenül "érti" a tartalmuk belső struktúráját, a tartalom értelmezése az azokat használó alkalmazások feladata.
- **Programfájlok:** végrehajtható kódot tartalmaznak. Az operációs rendszer képes ezeket betölteni a memóriába, és elindítani őket, mint folyamatot.
- **Könyvtárfájlok:** Ezek speciális fájlok, amelyeket a fájlrendszer kezel. Információkat tartalmaznak más fájlokról és könyvtárakról, amelyek az adott könyvtárban találhatók. Ezek hozzák létre a fájlrendszer hierarchikus struktúráját (mappák).
- **Speciális / eszközfájlok:** Különösen Unix-szerű rendszerekben jellemző, de más OS-ek is hasonlóan kezelhetik a hardvereket. Hardvereszközöket (pl. merevlemez partíciók, nyomtatók, terminálok) vagy OS belső interfészeket reprezentálnak. I/O műveleteket hajt vég végre az adott eszközön.

Fájlrendszerek: Szűkebb értelemben az operációs rendszer azon része, amely a fájlok és a könyvtárak (mappák) kezeléséért felelős. Szélesebb értelemben absztrakt adattípusok halmaza adatok tárolására, hierarchikus rendezésére, kezelésére, megtalálására, navigálásra, hozzáférésre és visszakeresésre. Egy kialakított struktúra háttértároló eszközön.

Típusai: Helyi fájlrendszerek, Hálózati fájlrendszerek (pl. NFS, SMB), Virtuális fájlrendszerek (csak absztrakt hozzáférést biztosítanak, pl. Linux procfs).

A legtöbb fájlrendszer rögzített méretű blokkokból vagy szektorokból álló sorozatot használ a tárolásra. Ezeket szervezi fájlokká és könyvtárakká, és nyilvántartja a hozzárendeléseket. Gyakran bevezetik a cluster fogalmát is. Nem feltétlenül kell mindenre tárolóberendezést használni, lehet dinamikus elérés (pl. hálózaton keresztül).

A legtöbb fájlrendszer hierarchikus: egy gyökérkönyvtárból indul ki, fa-szerkezetben könyvtárakat, almappákat és fájlokat tartalmaz.

Könyvtárműveletek: létrehozás, törlés, tartalom listázása, átnevezés.

3. SPECIÁLIS FÁJLOK UNIX ALATT

Névcsövek (Named Pipes vagy FIFO - First-In, First-Out): Két folyamat közötti kommunikációra szolgáló speciális fájltípus. Létrehozható a fájlrendszerben, és az egyik folyamat beleír, a másik pedig kiolvassa belőle az adatot, mint egy csővezetékből.

Szoketek (Sockets - Unix domain sockets): Hasonlóan a névcsövekhez, folyamatok közötti kommunikációra szolgálhatnak ugyanazon a gépen belül. Fájlként jelennek meg a fájlrendszerben.

Hard Link: Egy új könyvtárbejegyzés, amely ugyanarra a fizikai adatra (inode-ra) mutat, mint az eredeti fájl. Az eredeti és a hard link megkülönböztethetetlen az OS számára.

Szimbolikus linkek (Symbolic Links vagy Soft Links): "parancsikon" vagy mutató, amely egy másik fájlra vagy könyvtárra hivatkozik a fájlrendszerben. Ez csak az elérési útra mutat, nem az eredeti fájlra.

Eszközfájlok (Device Files): A hardver elemeit reprezentálják a fájlrendszerben, lehetővé téve hozzáférési jogok beállítását és közvetlen használatukat utasításokban. Lehetnek karakteres vagy blokk elérésűek. Példák: billentyűzet, terminál, merevlemez.

Speciális, fizikailag nem létező eszközök (/dev alatt):

- **/dev/null:** Minden bejövő adatot elnyel, kimenetet nem produkál. Egy parancs kimenetének eltüntetésére használható.
- **/dev/random**: Véletlenszerűen generált karaktersorozatokat állít elő. Használható véletlenszám generálásra.
- **/dev/zero**: Csupa 0 karakterből álló sorozatot produkál. Fájlok kinullázására használható biztonsági törléshez.

4. ÁTIRÁNYÍTÁS, CSŐVEZETÉKEK

Ezek a parancssori értelmezők (shellek, pl. Bash) alapvető funkciói a folyamatok standard bemeneti/kimeneti adatfolyamainak manipulálására.

Standard I/O adatfolyamok: Minden futó program (folyamat) rendelkezik velük alapértelmezetten:

- **Standard kimenet (stdout):** A program normál kimenete (alapértelmezetten a terminál).
- **Standard bemenet (stdin):** A program bemenete (alapértelmezetten a billentyűzet).
- **Standard hiba (stderr):** A program hibaüzeneteinek kimenete (alapértelmezetten a terminál).

Átirányítás: Megváltoztatja a standard adatfolyamok célját vagy forrását.

- command > file: A standard kimenetet (stdout) a fájlba írja (felülírja a fájlt).
- command >> file: A standard kimenetet (stdout) a fájl végéhez fűzi.
- command < file: A standard bemenetet (stdin) a fájlból olvassa.
- command 2> file: A standard hiba kimenetet (stderr) a fájlba írja.

Csővezeték: Lehetővé teszi egy folyamat standard kimenetének közvetlen összekötését egy másik folyamat standard bemenetével a | operátorral.

command1 | command2: A command1 kimenete lesz a command2 bemenete. Ezzel "adatfeldolgozó láncokat" (pipeline-okat) lehet építeni.

5. FOLYAMATKEZELÉS

Folyamat (Process): Egy futó program példánya, taszknak is nevezik. Minden folyamatnak van saját:

- **Egyedi azonosítója**: PID.
- **Szülő azonosítója**: PPID.
- **Virtuális memória tere**.
- **Erőforrásai** (fájlleírók, hálózati kapcsolatok).
- **Végrehajtási kontextusa** (CPU regiszterek állapota, programszámláló): Minden process önálló entitás a saját programszámlálójával és kontextusával.
- **Folyamat állapotok**: A processzek életük során különböző állapotokban lehetnek (Új, Kész/Futásra vár - ready, Futó - running, Várakozó/Blokkolt - blocked, Leállt - terminated). Állapotátmenetek lehetségesek (pl. futóból blokkolt várakozásra, blokkoltból készre jelzés hatására).

Folyamat Vezérlő Blokk (PCB - Process Control Block): Az operációs rendszer által minden folyamatról nyilvántartott adatstruktúra, amely tartalmazza a folyamat állapotára és erőforrásaira vonatkozó információkat.

Folyamatkezelés: Folyamatok jól definiált hierarchiát alkotnak. Minden folyamatnak pontosan egy szülője van, gyermeke lehet több. A legfelső folyamat az init folyamat, ami a rendszer indításakor jön létre. Az árvává vált gyermekfolyamatokat az init örökli. A processzek futhatnak előtérben (interaktív), vagy háttérben. Előtérben futó folyamat felfüggeszthető (Ctrl+Z), majd háttérbe helyezhető vagy folytatható. Lehet közöttük együttműködési kapcsolat, szinkron (egyik kimenete a másik bemenete) vagy aszinkron. Szinkron esetben a fogadó folyamat blokkolódhat.

Speciális folyamatok:

- **Daemon**: Speciális háttérfolyamat, amely rendszerbetöltéskor indul el, és szolgáltatásokat nyújt (pl. hálózati belépők, nyomtatási kérelmek figyelése). Inaktív állapotban várakoznak kérésre.
- **Zombie**: Olyan folyamat, amely már leállt, de a szülője még nem fogadta a visszatérési értékét, így még szerepel a folyamattáblában.

OS felelősségei a folyamatkezelésben:

- Folyamat létrehozás (pl. fork() és exec() Unix/Linux alatt).
- Folyamat leállítás (exit(), kill()).
- Folyamat ütemezés (CPU idő elosztása a kész folyamatok között).
- Folyamat szinkronizáció (koordinálás megosztott erőforrások elérésénél).
- Folyamatok közötti kommunikáció (IPC - Inter-Process Communication, pl. csővezetékek, üzenetsorok, osztott memória).

Folyamatkezelő parancsok (Unix/Linux): ps (folyamatok állapotának listázása), pstree (folyamathierarchia fa-nézetben), nohup (folyamat leválasztása a terminálról,

kijelentkezés után is fut), top (folyamatosan frissülő lista aktív folyamatokról, rendszerterhelésről, szignálküldés lehetősége).

6. JELZÉSEK, SZIGNÁLOK

A folyamatok közötti kommunikáció alapmechanizmusa. A **jelzések** (szignálok) aszinkron értesítések, amelyeket az operációs rendszer vagy más folyamatok küldhetnek egy folyamatnak bizonyos eseményekről való tájékoztatásra Unix-szerű rendszerekben. Egyfajta szoftveres megszakítás. Körülbelül 60 féle szignál létezik. Célja: Kezelni váratlan eseményeket (pl. megszakítás a felhasználótól), hibákat (pl. nullával osztás), vagy rendszereseményeket (pl. gyermekfolyamat befejeződése)

Szignál kezelése:

- Figyelmen kívül hagyás
- Elfogadja és kezeli egy szignálkezelő függvény végrehajtásával
- Engedheti az alapértelmezett művelet végrehajtását

7. ÜTEMEZETT VÉGREHAJTÁS

Az operációs rendszerek egyik fontos feladata a folyamatok ütemezése, ami meghatározza, hogy mikor és milyen sorrendben kapják meg a CPU-t az egyes folyamatok. Biztosítani kell a rendszer erőforrásainak hatékony és fair elosztását, miközben figyelembe kell venni a különböző típusú folyamatok prioritásait, futási időt, és más teljesítményi szempontokat.

Nem megszakítható ütemezés: Miután egy folyamat megkapta a CPU-t, addig fut, amíg be nem fejeződik, vagy önként várakozó állapotba nem kerül (pl. I/O műveletre várva).

Megszakítható ütemezés: Az operációs rendszer megszakíthatja egy futó folyamat végrehajtását, és átadhatja a CPU-t egy másiknak (pl. lejárt az időkerete, vagy magasabb prioritású folyamat érkezett). A modern többfeladatos OS-ek többsége megszakítható ütemezésű.

- **First-Come, First-Served (FCFS):** Az elsőként érkező folyamatot a rendszer elsőként futtatja, azaz a folyamatokat érkezési sorrendben végzi el.
- **Shortest-Job-First (SJF):** Az az ütemezés, amely a legrövidebb következő CPU használati igénnyel (burst) rendelkező folyamatot választja. Optimalizálja az átlagos várakozási időt.
- **Round Robin (RR):** Legelterjedtebb ütemezési algoritmus a multitasking rendszerekben. Az operációs rendszer minden egyes folyamatnak egyenlő időt biztosít (time slice, általában 10-100 milliszekundum). Ha nem fejeződik be ezen belül, megszakítják és a kész sor végére helyezik. Jó válaszidőt biztosít interaktív felhasználóknak.
- **Priority Scheduling**: A folyamatok egy prioritási szinttel rendelkeznek. Az operációs rendszer mindig a legmagasabb prioritású folyamatot futtatja, és a következő legmagasabb prioritású folyamatot választja ki, miután az előző folyamat befejeződött. Alacsony prioritású folyamatok éheztetéséhez (starvation) vezethet, amit az érés (aging) technikával (a várakozó folyamatok prioritásának növelése idővel) lehet enyhíteni.

5. Verziókezelés, verziókezelő rendszerek. Szoftvertesztelési alapfogalmak (tesztszintek, teszttípusok, teszttervezési módszerek). Objektum orientált tervezési alapelvek (GoF, SOLID). Függőségbefecskendezés. Architekturális minták (MVC). Tervezési minták. Szabad és nem szabad szoftverek. Szoftverlicencek, szabad és nyílt forrású licencek fajtái

1. VERZIÓKEZELÉS, VERZIÓKEZELŐ RENDSZEREK

Verziókezelés: fájlok és dokumentumok változásainak nyomon követése az idő múlásával. Lehetővé teszi a korábbi állapotok visszaállítását, a változtatások összehasonlítását, és a párhuzamos munkavégzést.

Verziókezelő rendszerek: Szoftvereszközök, melyek lehetővé teszik a verziókezelést

- Változások nyomon követése: Rögzítik, hogy ki, mikor és milyen módosításokat végzett
- Korábbi verziók visszaállítása: Lehetővé teszik a fájlok vagy akár a teljes projekt korábbi állapotának előhívását
- Együttműködés támogatása: Több fejlesztő dolgozhat ugyanazon a kódbázison anélkül, hogy egymás munkáját felülírnák
- Ágak (branching) és egyesítés (merging): Lehetővé teszik különálló fejlesztési ágak létrehozását új funkciókhoz vagy hibajavításokhoz, majd ezek visszaintegrálását a fő vonalba
- Konfliktusok kezelése: Segítenek a párhuzamosan végzett, egymást átfedő változtatásokból adódó konfliktusok feloldásában

Verziókezelő rendszerek típusai:

- Központosított Verziókezelő Rendszerek (CVCS): Egyetlen központi szerver tárolja az összes verziót. Példák: Subversion (SVN), CVS. Előnyük az egyszerűség, hátrányuk a központi szerver meghibásodásának kockázata és a hálózati kapcsolat szükségessége a műveletekhez.
- Elosztott Verziókezelő Rendszerek (DVCS): Minden felhasználó rendelkezik a teljes tároló (repository) egy másolatával. Példák: Git, Mercurial. Előnyük a robosztusság (nincs egyetlen hibapont), a gyorsabb műveletek (legtöbb offline is végezhető), és a rugalmasabb munkafolyamatok.

Git: Elosztott (distributed) verziókezelő rendszer. Nem egy központi szerverre támaszkodik, minden klón a teljes tároló (repository) másolatát tartalmazza.

- **Repository**: a projekt összes fájlja és teljes változási története. Lokális vagy távoli (GitHub, GitLab).
- **Commit**: pillanatkép az adott állapotról. SHA-1 hash azonosítja (40 karakter). Tartalmazza a változásokat, szerzőt, időpontot, üzenetet.
- **Branch**: fejlesztés elágaztatása. Külön ágakon párhuzamosan fejleszthetsz anélkül, hogy a fő vonalat (main) befolyásolnád.
- **HEAD**: az aktuális ágra mutató pointer.
- **Merge**: két ág változásainak összevonása.
- **Working Directory**: a fájlok aktuális állapota a fájlrendszerben.

2. SZOFTVERTESZTELÉSI ALAPFOGALMAK (TESZTSZINTEK, TESZTTÍPUSOK, TESZTTERVEZÉSI MÓDSZEREK)

A szoftvertesztelés egy megoldási forma, melynek célja a hibák kiszűrése, és a szoftver meghibásodásának megakadályozása működés közben. A tesztelés csak a hibák jelenlétét mutatja meg, nem hiányukat. A szoftvertesztelés tágabb értelemben a verifikáció és validáció része:

- **Verifikáció**: Annak ellenőrzése, hogy a szoftver megfelel-e a követelményeknek.
- **Validáció**: Annak ellenőrzése, hogy a szoftver megfelel-e az ügyfél elvárásainak.

Tesztszintek:

- **Egységtesztelés** (Unit): Az alkalmazás legkisebb tesztelhető egységeinek (pl. függvények, metódusok, osztályok) izolált tesztelése. Általában a fejlesztők végzik.
    - **Fast**: Gyors lefutású tesztek
    - **Indedendent**: Egymászól független tesztek
    - **Repeatable**: Tesztek bármilyen környezetben megismételhetőek
    - **Self-Validating**: Logikai kimenettel rendelkeznek: sikeres vagy sikertelen
    - **Timely**: A szükséges időben kell a teszteket megírni
    - Léteznek automatizált egységteszt keretrendszerek, melyek megkönnyítik a tesztelést (Java: JUnit, Python: pytest)
- **Integrációs tesztelés**: Az egységek kombinációinak tesztelése, ellenőrizve, hogy a különböző modulok megfelelően működnek-e együtt.
    - **Komponens integrációs tesztelés**: komponensek közti kommunikációra és interfészekre koncentrál. Egységteszteléssel, automatizált módon a fejlesztők végzik.
    - **Rendszerintegrációs tesztelés**: rendszerek közti kommunikációra és interfészekre koncentrál. Általában a tesztelők felelőssége.
- **Rendszertesztelés**: teljes rendszer tesztelése a specifikáció alapján.
- **Átvételi tesztelés**: végfelhasználó/ügyfél teszteli annak ellenőrzésére, hogy a rendszer megfelel-e az üzleti követelményeknek. Alfa (belső) és béta (külső).

Teszttípusok:

- **Validációs tesztelés**: célja bizonyítani a fejlesztő és felhasználó számára, hogy a szoftver megfelel a követelményeknek, felhasználást tükröző teszteseteket használ
- **Hiányosság tesztelés**: olyan bemenetek keresése, melynél a szoftver viselkedése helytelen, nemkívánatos, nem felel meg a specifikációknak, speciális teszteseteket használ
- **Funkcionális tesztelés**: a rendszer funkcióit tesztelik.
- **Nem funkcionális tesztelés**: olyat tényezőket vizsgál, mint például: használhatóság, teljesítmény, biztonság.
- **Fehérdobozos tesztelés**: rendszer belső felépítésén, megvalósításán alapuló tesztelés.
- **Változással kapcsolatos tesztelés** (megerősítő, regressziós tesztelés): Ha a rendszer módosul, hibajavítás történik, új funkciót kap a rendszer. - Megerősítő tesztelés: célja megerősíteni az eredti hiba kijavításának sikerességét - Regressziós tesztelés: annak

ellenőrzése, hogy az újabb változtatások nem rontották-e el a meglévő funkciókat (mellékhatások észlelése).

Teszttervezési módszerek:

- **Fekete dobozos tesztelés**: specifikáció alapján, belső ismeret nélkül.
- **Fehér dobozos tesztelés**: A szoftver belső szerkezetének és kódjának ismeretében történik a tesztelés.
    - Útvonal-lefedettség (Path Coverage): Minden lehetséges végrehajtási útvonal tesztelése.
    - Elágazás-lefedettség (Branch Coverage): Minden elágazás mindkét ágának tesztelése.
    - Utasítás-lefedettség (Statement Coverage): Minden utasítás legalább egyszeri végrehajtásának tesztelése.
- **Szürke dobozos tesztelés**: részleges ismeretekkel, a kettő kombinációja.
- **Tesztdublőrök**: a tesztelt egység valódi függőségét helyettesítő objektumok. Célja, hogy kontrollálható és kiszámítható viselkedést biztosítson a függőség helyett, így lehetővé téve a tesztelt egység izolált és megbízható tesztelését.

3. OBJEKTUM ORIENTÁLT TERVEZÉSI ALAPELVEK (GOF, SOLID)

GoF – Gang of Four:

- **Programozz interfészre, ne implementációra**: Az osztályok közötti függőségeket interfészeken keresztül alakítsuk ki, nem konkrét implementációkon keresztül. Ez növeli a rugalmasságot és csökkenti a kötöttséget.
- **Részesítsd előnyben az objektum-összetételt az öröklődéssel szemben**: Új funkcionalitás hozzáadásához preferáljuk az objektumok "összerakását" (kompozíció), szemben az öröklődéssel. A kompozíció rugalmasabb kapcsolatokat tesz lehetővé.

SOLID: Robert C. Martin (becenevén "Uncle Bob") által népszerűsített öt alapelv, amelyek célja a szoftverek érthetőségének, rugalmasságának és karbantarthatóságának növelése.

- **S - Single Responsibility Principle (Egyetlen Felelősség Elve):** Egy osztálynak vagy modulnak csak egyetlen oka legyen a változásra, azaz csak egyetlen jól definiált feladatért feleljen.
- **Open/Closed Principle (Nyílt/Zárt Elv):** Egy szoftver egységnek (osztály, modul, függvény) nyitottnak kell lennie a bővítésre, de zártnak a módosításra. Új funkcionalitást kód módosítása nélkül, kiterjesztéssel (pl. öröklődés, interfész implementáció) lehessen hozzáadni.
- **L - Liskov Substitution Principle (Liskov Behelyettesítési Elv):** Egy alaposztály példányai helyettesíthetők legyenek az őt megvalósító származtatott osztályok példányaival anélkül, hogy a program helyessége megsérülne. (Ha S a T leszármazottja, akkor T típusú objektumok helyett S típusú objektumokat használhatunk a program helyességének sérülése nélkül.)
- **I - Interface Segregation Principle (Interfész Szegregációs Elv):** Az ügyfelek (clientek) ne függjenek olyan interfész tagoktól, amelyeket nem használnak. Jobb több, kisebb, specifikus interfész, mint egy nagy, általános interfész.

- **D - Dependency Inversion Principle (Függőség Megfordítási Elv):** Magas szintű modulok ne függjenek alacsony szintű moduloktól. Mindkettő absztrakcióktól (interfészek, absztrakt osztályok) függjön. Az absztrakciók ne függjenek a részletektől, a részletek függjenek az absztrakcióktól.

DRY (Don't Repeat Yourself): minden info egy helyen. Duplikáció = dupla hibaforrás.

KISS (Keep It Simple, Stupid): egyszerűségre törekvés. Túlbonyolítás = hibák forrása.

YAGNI (You Aren't Gonna Need It): ne építs olyat, amire most nincs szükség.

Demeter törvénye: egy objektum csak a „legközelebbi barátaival"; beszéljen. Tiltja a vonatszerencsétlenséget: a.getB().getC().doSomething(). A kód karbantarthatóságát és rugalmasságát hivatott növelni a **csatolás** (coupling) csökkentésével.

4. FÜGGŐSÉGBEFECSKENDEZÉS

A függőségbefecskendezés egy tervezési minta, amely a SOLID elvek, különösen a Függőség Megfordítási Elv megvalósítását segíti. Egy objektum (kliens) nem hozza létre maga a szükséges függőségeit, hanem azokat kívülről "fecskendezik be" neki: Az objektum függőségeit egy külső entitás (az "injektor" vagy DI konténer) hozza létre, és adja át a kliensnek (pl. konstruktoron, setter metóduson vagy interfészen keresztül).

- Csökkenti a kötöttséget (loose coupling): Az objektumok kevésbé függenek egymás konkrét implementációitól.
- Könnyebb tesztelés: A függőségek könnyen kicserélhetők teszt objektumokra (mock objects).
- Jobb karbantarthatóság és bővíthetőség: A függőségek változtatása vagy cseréje kisebb hatással van a klienst használó kódra.
- Újrahasznosíthatóság: Az objektumok könnyebben újrahasznosíthatók különböző kontextusokban.

5. ARCHITEKTURÁLIS MINTÁK (MVC)

Az architekturális minták magasabb szintű, a teljes alkalmazás szerkezetét érintő tervezési megoldások. Szerkezetek felépítésére adnak sémákat, szabályokat, irányelveket.

Model-View-Controller: A felhasználó interakciója (pl. kattintás) eljut a Controllerhez. A Controller feldolgozza a kérést, szükség esetén módosítja a Modelt, majd kiválasztja a View-t, amely a frissített adatokat megjeleníti a felhasználónak. A Model értesítheti a View-t a változásokról. Előnyök: felelősségek szétválasztása, könnyebb tesztelés, párhuzamos fejlesztés.

Model: Az alkalmazás adatainak és az ezeken végrehajtható üzleti logikának a reprezentációja. Nem függ a megjelenítéstől vagy a felhasználói interakciótól.

View: Felelős az adatok (Model) megjelenítéséért a felhasználó számára. Nem tartalmaz üzleti logikát. Több View is tartozhat ugyanahhoz a Modelhez.

Controller: Kezeli a felhasználói interakciókat, lekérdezi a Modelből a szükséges adatokat, frissíti a Modelt a felhasználói bevitel alapján, és kiválasztja a megfelelő View-t a válasz megjelenítéséhez. Összekapcsolja a Modelt és a View-t.

6. TERVEZÉSI MINTÁK

Bevált, általános megoldások gyakran előforduló problémákra a szoftvertervezésben. Nem kész kódot jelentenek, hanem útmutatást adnak a problémák elegáns és hatékony megoldásához. GoF szerint 3 kategóriája van.

Létrehozási minták: céljuk objektumok rugalmas létrehozása, a példányosítás részleteinek elrejtése.

- Factory Method: szuperosztály metódus definiálja a létrehozást, leszármazott a konkrét típust.
- Abstract Factory: interfész (absztrakt gyár) objektumcsalád létrehozására, konkrét gyárak valósítják meg.
- Singleton: az osztály maga hozza létre és adja vissza saját egyetlen példányát (privát konstruktor, statikus get metódus).
- Builder: külön építő objektum felelős a komplex objektum részenkénti felépítéséért.
- Prototype: új objektumok klónozással készülnek egy meglévő prototípusból.

Strukturális minták: céljuk objektumok és osztályok hatékony összerakása nagyobb struktúrákba.

- Adapter: adapter osztály "lefordítja" az egyik interfészt a másikra.
- Bridge: absztrakció és implementáció szétválasztása külön hierarchiákba.
- Composite: fa-szerű struktúra, közös interfész leveleknek és csomópontoknak.
- Decorator: dekorátor objektum becsomagolja az eredetit, extra viselkedést adva hozzá.
- Facade: egyetlen Facade objektum egyszerűsített interfészt nyújt a komplex alrendszerhez.
- Flyweight: közös állapot megosztása több objektum között.
- Proxy: helyettesítő (Proxy) objektum kezeli a hozzáférést a valódi objektumhoz.

Viselkedési minták: céljuk objektumok közötti kommunikáció és felelősségkiosztás

- Chain of Responsibility: kérés továbbítása egy objektum láncon, amíg valamelyik kezeli.
- Command: kérés egy Command objektumként reprezentálva.
- Iterator: gyűjtemény bejárása belső ismeret nélkül külön Iterátor objektum által.
- Observer: alany értesíti a regisztrált megfigyelőket állapotváltozásról.
- State: viselkedés állapottól függ, minden állapot külön objektum.

7. SZABAD ÉS NEM SZABAD SZOFTVEREK

A szoftvereket jogi szempontból két fő kategóriába sorolhatjuk: szabad szoftverek és nem szabad szoftverek. Ez a besorolás a szoftverek használati jogain, hozzáférhetőségén és módosíthatóságán alapul.

Szabad szoftver (Free Software): Olyan szoftver, amelyet a felhasználó szabadon használhat, tanulmányozhat, módosíthat és terjeszthet licenctől függően. A szoftver licenc feltételei jelentősen korlátozzák a felhasználó szabadságát.

Nem szabad szoftver (Proprietary Software / Closed-source Software): A használat, terjesztés és módosítás tilos, korlátozott vagy engedélyhez kötött

Nyílt forráskódú szoftver (Open-source Software): Bár nagyon hasonló a szabad szoftverhez és a két fogalom gyakran fedi egymást, van némi filozófiai különbség. A nyílt forráskódú mozgalom (Open Source Initiative - OSI) inkább a fejlesztési modell előnyeire fókuszál (gyorsabb fejlesztés, nagyobb biztonság a közösségi ellenőrzés miatt), míg a szabad szoftver mozgalom a felhasználói szabadságra helyezi a hangsúlyt. Minden szabad szoftver nyílt forráskódú, de nem minden nyílt forráskódú szoftver szabad.

8. SZOFTVERLICENCEK, SZABAD ÉS NYÍLT FORRÁSÚ LICENCEK FAJTÁI

A szoftverlicenc egy jogi dokumentum, amely meghatározza a szoftver felhasználásának feltételeit.

Szabad és nyílt forrású licencek fajtái: Ezek a licencek biztosítják a szabad és nyílt forráskódú szoftverek használatának, módosításának és terjesztésének jogát a felhasználók számára, de különböző feltételekkel:

- **Permisszív licencek:** Minimális korlátozásokkal rendelkeznek a szoftver felhasználására és terjesztésére vonatkozóan. Általában megengedik a szoftver zárt forráskódú projektekben való felhasználását.
    - **MIT License**: nagyon egyszerű, bármire használható, csak az eredeti licencet és a szerzői jogi nyilatkozatot kell feltüntetni.
    - **BSD License**: hasonlóan engedékeny, mint az MIT.
    - **Apache 2.0 License**: engedékeny licenc, amely széles körű felhasználást tesz lehetővé, beleértve a szabadalmi jogokat is.
- **Copyleft licencek**: Ezek a licencek biztosítják a szabad szoftver szabadságjogait, de megkövetelik, hogy a származtatott művek is ugyanolyan szabadságjogokkal rendelkezzenek. Céljuk, hogy a szoftver "szabad" maradjon.
    - **GNU General Public License (GPL):** erős copyleft licenc. Ha egy szoftver GPL alatt licencelt kódot használ, akkor a származtatott művet is GPL alatt kell terjeszteni.
    - **GNU Lesser General Public License (LGPL):** kevésbé szigorú, mint a GPL. Lehetővé teszi a szoftver linkelését zárt forráskódú alkalmazásokból anélkül, hogy az egész alkalmazást LGPL alá kellene helyezni.
    - **Mozilla Public License (MPL):** közepes copyleft licenc. Meghatározott fájlokat vagy modulokat érint a copyleft hatálya, nem feltétlenül a teljes származtatott művet.

6. Hagyományos szoftverfejlesztési módszertanok: vízesés modell, V-modell, spirális fejlesztési modell, prototípus alapú fejlesztés, iteratív és inkrementális módszertanok, gyors alkalmazásfejlesztés. Agilis szoftverfejlesztési módszertanok: az agilis szoftverfejlesztés alapjai, az agilis kiáltvány, valamint egy szabadon választott agilis módszertan részletes bemutatása.

1. HAGYOMÁNYOS SZOFTVERFEJLESZTÉSI MÓDSZERTANOK

Ezeket a módszertanokat gyakran nevezik szekvenciális vagy "nehézsúlyú" módszertanoknak is. Jellemzőjük a szigorú, egymást követő fázisok és az átfogó dokumentáció hangsúlyozása. Általában olyan projektekhez javasoltak, ahol a követelmények jól definiáltak és valószínűleg nem változnak a fejlesztés során.

Vízesés modell: A fejlesztés szigorúan egymást követő fázisokon megy keresztül, mint egy vízesés. Nincs visszatérés az előző fázisba. Legalapvetőbb modell, lépések lineárisan egymásra épülnek. Mindent előre definiálunk, fejlesztés nem alakítható a feladatok alapján.

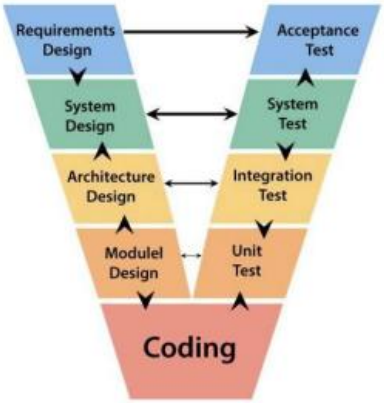
1. Követelmények elemzése és specifikáció
2. Rendszer- és szoftvertervezés
3. Megvalósítás és egységtesztelés
4. Integrációs és rendszertesztelés
5. Üzemeltetés és karbantartás

Előnyök: Egyszerű, jól érthető, könnyen kezelhető kis projektek esetén, ahol a követelmények stabilak. Erős hangsúly a dokumentáción.

Hátrányok: Nagyon merev, nehéz kezelni a változó követelményeket. A hibákat általában csak a folyamat későbbi fázisaiban fedezik fel, javításuk költséges lehet. Az ügyfél csak a végén látja a működő szoftvert.

V-modell: A vízesés modell kiterjesztése, ahol a fejlesztési fázisokhoz szorosan kapcsolódnak a megfelelő tesztelési fázisok, így egy "V" alakot formáznak.

- bal oldali szár a fejlesztés:
  - Követelmények
  - Rendszertervezés
  - Architektúrális tervezés
  - Modul tervezés
- Implementáció(kódolás)
- jobb oldali szár a tesztelés és validáció:
  - Átvételi tesztelés
  - Rendszertesztelés
  - Integrációs tesztelés
  - Egységtesztelés



Előnyök: Erős hangsúlyt fektet a tesztelésre már a korai fázisokban, növelve a minőséget. Jobb nyomon követhetőség a követelmények és a tesztek között.

Hátrányok: Még mindig viszonylag merev, nehézkes a változások kezelése a folyamat közben. A tesztelési fázisok csak a fejlesztési fázisok befejezése után kezdődnek meg.

Spirális fejlesztési modell: Kockázatvezérelt iteratív modell, amely a vízesés és a prototípus modellek elemeit ötvözi.

Fázisok (ciklikusan ismétlődnek):

- Célok meghatározása
- Kockázat elemzés és csökkentés
- Fejlesztés és validáció
- Tervezés

Előnyök: Jól kezeli a kockázatokat, rugalmasabb, mint a vízesés modell. Lehetővé teszi a prototípusok használatát.

Hátrányok: Kockázat elemzési szakértelemet igényel. Lehet drága és időigényes nagy kockázatú projektek esetén. Nem ideális alacsony kockázatú, kis projektekhez.

Prototípus alapú fejlesztés: Egy kezdeti, gyakran nem teljes funkcionalitású prototípus gyors elkészítése a követelmények tisztázására és a felhasználói visszajelzések gyűjtésére.

- Követelmények gyűjtése
- Ciklikusan:
    - Prototípus gyors tervezése
    - Prototípus felépítése
    - Ügyfél értékelése
    - Prototípus finomhangolása
- Ügyfél elfogadja a prototípust -> Végső rendszer implementálása: Fejlesztés, tesztelés, kiadás

Előnyök: Segít tisztázni a követelményeket, korai felhasználói visszajelzést biztosít. Csökkenti a félreértések kockázatát.

Hátrányok: A felhasználók ragaszkodhatnak a kezdeti prototípushoz, figyelmen kívül hagyva annak hiányosságait.

Iteratív és inkrementális módszertanok: A fejlesztés kisebb, kezelhetőbb részekre (inkrementumokra) bomlik, és ezeket egymást követő ciklusokban (iterációkban) fejlesztik. Minden iteráció végén egy működőképes, a korábbiakhoz képest új funkciókkal bővített (inkrementált) rendszerrész keletkezik. Működés: Ismétlődő ciklusok (iterációk), mindegyik tartalmazza a tervezés, megvalósítás és tesztelés fázisait az adott inkrementumra vonatkozóan.

Előnyök: Rugalmasabb a változások kezelésében, mint a vízesés modell. Korai működőképes szoftvert eredményez. Csökkenti a kockázatokat, mivel a problémák korábban kiderülnek.

Hátrányok: A rendszer architektúrája a projekt elején nem feltétlenül tiszta.

Gyors alkalmazásfejlesztés, RAD (Rapid Application Development): Iteratív, prototípus-vezérelt megközelítés, amely a gyors fejlesztésre és a felhasználói visszajelzésre helyezi a hangsúlyt. Gyakran használ speciális eszközöket és technikákat a sebesség növelésére.

- Üzleti modellezés

- Adat modellezés
- Folyamat modellezés
- Alkalmazás generálás
- Tesztelés és átadás

Előnyök: gyors fejlesztés, korai és folyamatos felhasználói bevonás. Csökkenti a fejlesztési időt.

Hátrányok: képzett fejlesztőket és erős felhasználói elkötelezettséget igényel. Nem alkalmas rendszerszintű komponensekhez vagy nagy, komplex projektekhez.

2. AGILIS SZOFTVERFEJLESZTÉSI MÓDSZERTANOK

Az agilis módszertanok a hagyományos megközelítésekre adott válaszként jelentek meg, hangsúlyozva a rugalmasságot, az ügyféllel való együttműködést és a gyors reagálást a változásokra. A középpontban a működő szoftver és az emberek állnak.

Az agilis szoftverfejlesztés alapjai:

- Alkalmazkodás a változásokhoz a szigorú tervezés helyett.
- Iteratív (ismétlődő) és inkrementális (növekményes) fejlesztés rövid ciklusokban (általában 1-4 hét).
- Fókuszban a működőképes szoftver.
- Ügyféllel és érintettekkel való szoros, folyamatos együttműködés.
- Önszerveződő és keresztfunkcionális csapatok.
- Folyamatos kommunikáció (gyakran személyes).
- Folyamatos fejlődés és tanulás.

Az agilis kiáltvány (agile manifesto): 2001-ben 17 szoftverfejlesztő fogalmazta meg. 4 alapelvet tartalmaz:

- Egyének és az interakciók a folyamatokkal és eszközökkel szemben.
- Működő szoftver az átfogó dokumentációval szemben.
- Ügyféllel való együttműködés a szerződéses egyeztetéssel szemben.
- Változásra való reagálás a tervek szigorú követésével szemben.

SCRUM: A Scrum egy agilis módszertan, amely segít a csapatoknak hatékonyan és rugalmasan kezelni a komplex projekteket. Rövid iterációkra (ún. sprintekre) osztja a munkát, miközben állandóan fókuszál a csapatmunkára, az ügyfélérték megteremtésére és a gyors alkalmazkodásra. Alapelvei az átláthatóság (Transparency), ellenőrzés (Inspection) és alkalmazkodás (Adaptation).

Scrum Csapat: Kicsi, önszerveződő és keresztfunkcionális. Nincs alcsapat vagy hierarchia. Három szerepköre van:

- **Product Owner** (Terméktulajdonos): Felelős a termék értékének maximalizálásáért. Kezeli a Termék Backlogot (mi készüljön, milyen prioritással).
- **Scrum Master**: Segíti a csapatot és a szervezet minden tagját a Scrum megértésében és alkalmazásában. Akadályokat hárít el. "Szolgáló vezető".

- **Development Team (Fejlesztői Csapat):** Keresztfunkcionális szakemberek csoportja, akik felelősek a Működő Inkrementum leszállításáért minden Sprint végén. Önszerveződőek a munka elvégzésében.

Scrum Események (Time-boxed, fix idejű események):

- **Sprint**: A Scrum szíve. Egy fix időtartam (általában 1-4 hét), amelyen belül a csapat befejezi a tervezést, megvalósítást, tesztelést és leszállít egy Működő Inkrementumot.
- **Sprint Planning** (Sprint Tervezés): A Sprint elején tartott esemény, ahol a csapat megtervezi a következő Sprintet (Mi készüljön el? Hogyan készüljön el?).
- **Daily Scrum** (Napi Scrum): Rövid, napi (általában 15 perces) megbeszélés a Fejlesztői Csapat számára, ahol szinkronizálják tevékenységüket és megtervezik a következő nap munkáját.
- **Sprint Review** (Sprint Áttekintés): A Sprint végén tartott esemény, ahol a csapat bemutatja az elkészült Működő Inkrementumot az érintetteknek, és visszajelzést gyűjtenek.
- **Sprint Retrospective** (Sprint Visszatekintés): A Sprint befejezése után tartott esemény, ahol a csapat megvizsgálja, hogyan dolgozott a Sprint során, és azonosítja a javítási lehetőségeket a következő Sprintre.

Scrum Munkadarabok (Artifacts):

- **Product Backlog (Termék Backlog):** Priorizált lista mindenről, ami szükséges a termékben. A Product Owner felelőssége. Folyamatosan változik, finomodik.
- **Sprint Backlog**: A Product Backlog azon elemeinek és a cél eléréséhez szükséges munka listája, amelyeket a Fejlesztői Csapat a Sprint során elkötelezett megvalósítani. A fejlesztői csapat kezeli.
- **Increment (Inkrementum):** A Sprint során elkészült, "Kész" állapotú, működőképes termékrész. Potenciálisan szállítható állapotban van.

7. A web működésének alapjai. Web szabványok és szabványügyi szervezetek. URI-k és felépítésük. HTTP: kérések és válaszok felépítése, metódusok, állapotkódok, tartalomegyeztetés, sütik. A web jelölőnyelvei: XML és HTML dokumentumok felépítése. Stíluslap nyelvek. JSON.

1. A WEB MŰKÖDÉSÉNEK ALAPJAI

A web működése az internet infrastruktúrájára, az adatok kérésére és továbbítására épül, a kliens-szerver modell és a TCP/IP protokollcsalád alapjain. Web alapötlete:

- Erőforrások azonosítása URI-kkal.
- Kliens-szerver alapú modell
- Hiperszöveg jelölőnyelv (HTML)

Világháló, web: információs tér, melynek erőforrásnak nevezett elemeit URI-k azonosítják

Erőforrás: bármi, ami URI-val azonosítható

Információ erőforrás: azzal a tulajdonsággal rendelkező erőforrás, hogy minden lényeges jellemzője továbbítható egy üzenetben

Kliens (Client): A kliens az a számítógép vagy alkalmazás, amely kéréseket küld egy kiszolgálónak. Jellemzően egy böngésző.

(Web)Szerver (Server): Egy számítógép vagy rendszer, amely tárolja a weboldalakat, alkalmazásokat és adatokat, és készen áll a kliensek kéréseinek fogadására és feldolgozására.

Web ágens: Weben személy, entitás vagy folyamat nevében cselekvő személy Felhasználói ágens: a webágensek egy fajtája, egy személy nevében cselekvő szofver

Web architektúrális alapja:

- **Azonosítás**: Erőforrások azonosítása URI-nak nevezett globális azonosítókkal
- **Kölcsönhatás**: Web ágensek szabványosított protokollok segítségével kommunikálnak, melyek alapja az üzenetcsere. Protokollok: HTTP, HTTPs. Üzenet: adatok + metaadatok az erőforrásról és magáról az üzenetről
- **Adatformátumok**: Protokollok határozzák meg az átvihető adatok formátumát: a két kommunikáló fél tudja kezelni az adatokat

Web böngészők: A webböngésző egy szoftveralkalmazás, amelynek elsődleges célja az információs erőforrások (elsősorban weboldalak) elérése, lekérése, megjelenítése és navigálása a Világhálón (World Wide Web). Ezek az erőforrások Uniform Resource Identifier (URI) vagy Uniform Resource Locator (URL) segítségével vannak azonosítva.
Feladatai:

- Lekérés: protokoll használatával tartalom lekérése
- Megjelenítés: webes technológiák értelemzése, vizuális megjelenítése
- Navigáció: weblapok közti mozgás
- Állapotkezelés: Sütik kezelése, helyi adatok tárolása az interakciók során az állapotkezelés céljából

Komponensei:

- **UI**: A böngésző látható része, amellyel a felhasználó interakcióba lép
- **Browser Engine**: Összehangolja a böngésző különböző részeit. Lekérdezi a weboldalakat a hálózatról, és a renderelő motort használja azok megjelenítésére.
- **Rendering Engine**: Felelős a HTML és CSS kód értelmezéséért és a weboldal képernyőn történő megjelenítéséért.
- **Networking**: Kezeli a HTTP/HTTPS kéréseket és válaszokat, valamint egyéb hálózati kommunikációt (DNS lekérdezések stb.).
- **JavaScript Engine**: Felelős a JavaScript kód értelmezéséért és végrehajtásáért, interaktivitásért felelős
- **UI Backend**: Rajzolja a böngésző felhasználói felületének alapvető widgetjeit (pl. gombok, beviteli mezők). Az operációs rendszer grafikus felületét használja.
- **Data Storage**: Kezeli a helyi adatokat, mint a sütik, localStorage, IndexedDB.

2. WEB SZABVÁNYOK ÉS SZABVÁNYÜGYI SZERVEZETEK

Szabvány: Dokumentum, mely követelményeket, előírásokat, irányelveket vagy jellemzőket fogalmaznak meg. Ezek következetesen használhatóak annak biztosítására, hogy anyagok, termékek folyamatok é szolgáltatások megfeleljenek a rendeltetésüknek. Fajtái:

- **De facto**: gyakori használatból, piaci elfogadottságból származnak (QWERTY billentyűzet)
- **De jure**: helyi, állami, nemzetközi szintű szabályozók által meghatározott (SI mértékegységek)
- **Önkéntes közmegegyezéses szabályok**: magánintézmények által meghatározott szabályok (Internetprotokollok: TCP, HTTP, …)

Webszabványok: A web nyitott és együttműködő jellege nagyrészt a nemzetközi szabványoknak köszönhető. Ezek a szabványok biztosítják, hogy a különböző böngészők, szerverek és webes technológiák képesek legyenek egymással kommunikálni és a tartalmat egységesen értelmezni.

IANA (Internet Assigned Numbers Authority): Feladata az internet alapjául szolgáló kódok és számok kiosztása, DNS-gyökérzóna felügyelete, IP-címek kiosztása, internetprotokollok által használt kódok nyilvántartása. Ez egy funkció, melyet egy non-profit cég működtet: Internet Corporation for Assigned Names and Numbers (ICANN).

IETF (Internet Engineering Task Force): Feladata az internet szabványainak fejlesztése egy nemzeti szabványügyi szervezetként, protokollkészlet fejlesztése)TCP/IP), RFC dokumentumok publikálása (szabványokhoz tartozó specifikációk). Nyílt részvételű szervezet, munkacsoportokban történik a fejlesztés

RFC dokumentumok: műszaki és szervezeti dokumentumok – Request for Comments

- minden RFC-t egy szám azonosít
- elérhetőek ASCII szövegként
- a kiadott RFC-k nem módosulnak, hibajegyzékekben rögzítik a hibákat. Változtatások új RFC írásával, amely az előzőt elavultá vagy frissítetté teheti
- Standard Track: érettségi szintek RFC-k esetén

- Proposed Standard – javasolt szabvány
- Draft Standard – szabványtervezet
- Internet Standard – internet szabvány o változhatnak, törlésre kerülhetnek, legfeljebb 6 hónapig érvényesek o RFC-vé válhatnak

W3C (World Wide Web Consortium): a webes technológiák legtöbb szabványáért felelős, mint például a HTML, CSS, XML, DOM. Céljuk a web teljes potenciáljának kiaknázása szabványok fejlesztésével. Nemzetközi közösség, ahol főállású alkalmazottak és a nyilvánosság együttműködve dolgozik a webszabványok fejlesztésén.

W3C ajánlások: webtechnológiákat meghatározó és webszabványoknak számító dokumentumok

Alapelvei:

- web mindenkinek legyen elérhető: függetlenül hardverektől, szoftverektől, hálózati infrastruktúrától, anyanyelvtől, kultúrától, földrajzi elhelyezkedéstől, fizikai/szellemi képességektől o akadálymentesített web, nemzetköziesítés
- web mindenhol legyen elérhető: legkülönfélébb eszközökről

Fejlesztést munkacsoportok végzik: W3C tagjai, tagszervezetek képviselői (Adobe, Amazone, Apple, CERN, Google…), meghívott szakértők. Nyitott részvétel is lehetséges: visszajelzések, csatlakozás közösségi csoportokhoz, workshopokon való részvétel.

W3C dokumentumok érettségi szintjei:

- Working Draft (WD) - Munka Vázlat: Ez a legkorábbi hivatalos érettségi szint, a munkacsoport által kiadott első nyilvános verzió. Célja a korai visszajelzések gyűjtése a szélesebb közösségtől.
- Candidate Recommendation (CR) - Jelölt Ajánlás: A munkacsoport úgy véli, hogy a specifikáció stabil, és készen áll a szélesebb közösség, különösen a fejlesztők általi implementálásra. A kritikus hibákat ebben a fázisban még lehet javítani, ami kisebb változásokhoz vezethet. Jelentős változások esetén visszatérhet Working Draft szintre. A W3C felhívja a figyelmet az implementációra, hogy teszteljék a specifikáció megvalósíthatóságát és interoperabilitását.
- Proposed Recommendation (PR) - Javasolt Ajánlás: A munkacsoport úgy véli, hogy a specifikációt elégségesen implementálták, és a tesztek igazolták annak működőképességét és interoperabilitását. A W3C tagság (tagvállalatok, szervezetek) hivatalos felülvizsgálatra bocsátják.
- W3C Recommendation (REC) - W3C Ajánlás: Ez a legmagasabb érettségi szint. A specifikáció hivatalos W3C szabvánnyá válik.

WHATWG (Web Hypertext Application Technology Working Group): Egy közösség, amely webes technológiák, különösen a HTML, a DOM (Document Object Model) és kapcsolódó API-k szabványainak fejlesztésével és karbantartásával foglalkozik. Főként a nagy böngészőgyártók (Apple, Mozilla, Google, Microsoft) képviselői alkotják. "Living Standard" fejlesztése és karbantartása: Ez a WHATWG legfontosabb hozzájárulása. Ahelyett, hogy fix verziókat adnának ki, a WHATWG szabványok folyamatosan frissülnek, tükrözve a web technológiák folyamatos fejlődését és a böngészőkben megvalósított funkciókat.

3. URI-K ÉS FELÉPÍTÉSÜK

URI: globális erőforrásazonosító a Weben, absztrakt vagy fizikai erőforrást azonosító tömör karaktersorozat

Hivatkozás feloldás: URI használata a hivatkozott erőforrás eléréséhez, aktuális állapot letöltése, létrehozása, módosítása.

séma:autoritás/útvonal?lekérdezés#erőforrásrész

Séma/protokoll: Meghatározza, hogyan lehet hozzáférni az erőforráshoz. Pl: http, https, ftp.

Autoritás: Az erőforrást szolgáltató szerver azonosítója, az utána lévő részek a meghatározott névtér alá tartoznak: hostnév (tartománynév vagy IP-cím), :port (ha nem az alapértelmezett portot használják)

Útvonal: Az erőforrás útvonala a szerveren belül, általában könyvtárakkal és fájlnévvel. / karakterrel elválasztott sorozat, mely lehet üres.

Lekérdezés: Opcionális. Kiegészítő paramétereket ad át a szervernek, gyakran névérték párok formájában, & jellel elválasztva. A ? jel vezeti be.

Erőforrásrész azonosító: Opcionális. Egy adott részét azonosítja az erőforrásnak az erőforráson belül. A # jel vezeti be, és a böngésző általában nem küldi el a szervernek, csak a kliens oldalon használja. A médiatípusok határozzák meg

Abszolút URI: Nem tartalmazza az erőforrásrész azonosítót

Relatív hivatkozás: URI séma specifikus része, csak adott környezetben értelmezhető. Feloldását egy algoritmus írja le a specifikációban

URL: Uniform Resource Locator, Az URL az URI egyik altípusa. Az erőforrást a helye alapján azonosítja a hálózaton, megadva azt is, hogyan (milyen protokollal) lehet hozzáférni.

4. HTTP: KÉRÉSEK ÉS VÁLASZOK FELÉPÍTÉSE, METÓDUSOK, ÁLLAPOTKÓDOK, TARTALOMEGYEZTETÉS, SÜTIK

HTTP: A HTTP az az alapvető protokoll, amelyen a web működik. A kliens és a szerver közötti kommunikációt szabályozza. Jellemzői:

- egységes interfészt biztosít az erőforrásokkal történő interakcióhoz olyan üzenetek révén, melyek reprezentációt manipulálnak, továbbítanak
- kliens-szerver modellen alapuló protokoll, melynek alapja az üzenetcsere
- állapotnélküli protokoll, minden egyes kérés jelentése a többiétől külön értelmezendő
- kiterjeszthető, újabb metódusok, állapotkódok, mezők, hitelesítési sémák, tartomány egységek és tartalomkódolások vezethetőek be újabb verzió bevezetése nélkül

Kapcsolat: a HTTP egy kliens-szerver protokoll, mely egy megbízható szállítási vagy viszony rétegbeli kapcsolat felett működik

Üzenet: a HTTP protokollon küldött adat állapotnélküli kérése/válasza

Küldő/fogadó: egy adott üzenetet elküldő vagy fogadó tetszőleges implementáció

Kliens: egy szerverrel, HTTP kérés(ek) küldése céljából kapcsolatot létesítő program

Szerver: HTTP kéréseket kiszolgálás céljából kapcsolatokat elfogadó program, mely HTTP válaszokat küld

Felhasználó ágens: HTTP kérést kezdeményező program (webböngésző), nem feltétlen emberi személy kezeli a programot a kérés elküldésekor

Közvetítő: lehetővé teszik a kérések kiszolgálását egy kapcsolatláncon keresztül - proxy, átjáró, alagút

Kérések és válaszok felépítése: HTTP főverziók saját szintaxist határoznak meg az üzenetváltáshoz (keretezési mechanizmus). Üzenetek önleíróak: mindent amire szüksége van a feladónak, megtalálható az üzenetben. Üzenet részei/absztrakciója:

- **vezérlő adatok**: üzenet célját írja le
- **fejléc szakasz**: fejlécmezőket tartalmaz
- **tartalom**: teljes vagy részleges reprezentációt tartalmazza, bájtfolyamok
- **lezáró szakasz**: lezárómezőket tartalmaz

<table>
<tr><th></th><th>Kérés</th><th>Válasz</th></tr>
<tr><td>Vezérlő Adatok</td><td>Metódus
Kérés célja
Protokoll verzió</td><td>Állapotkód
Opcionális indok frázis
Protokoll verzió</td></tr>
<tr><td rowspan="2">Fejléc szakasz</td><td colspan="2">Fejlécmezők sorozata</td></tr>
<tr><td><code>User-Agent</code>: felhasználói ágensről információk, válasz testreszabásához fontos</td><td>Reprezentációt leíró metaadatok: leírja, hogy hogyan kell értelmezni az erőforrást
<code>Content-Type</code>: reprezentáció formátuma (médiatípus)
<code>Content-Encoding</code>: reprezentáció kódolása, meghatározza, milyen dekódolási mechanizmusra lesz szükség: veszteségmentes tömörítés, végső fogadó dekódolja (gzip, br)</td></tr>
<tr><td rowspan="2">Tartalom</td><td colspan="2">Reprezentáció: részleges vagy teljes
Bájtsorok formájában</td></tr>
<tr><td>Metódus határozza meg, opcionális</td><td>Metódus, állapotkód, válaszmezők határozzák meg</td></tr>
<tr><td>Lezáró szakasz</td><td colspan="2">Lezárómezők sorozata:
Ellenőrző összegek, digitális aláírások, kézbesítési metrikák, utófeldolgozási állapotinformációk</td></tr>
</table>

Metódusok: Ezek határozzák meg a kérés típusát és a szerver által végrehajtandó műveletet.

- GET: Erőforrás lekérése a szerverről. Nincs kérés törzse, a paraméterek az URL-ben vannak. Ideális adatok lekérdezésére, nem módosít semmit a szerveren (idempotens).
- POST: Adatok küldése a szervernek egy új erőforrás létrehozásához vagy egy meglévő frissítéséhez. Az adatok a kérés törzsében vannak. Nem idempotens (többszöri hívás többszöri eredményt hozhat).
- PUT: Egy erőforrás frissítése vagy létrehozása a megadott URI-n. Az adatok a kérés törzsében vannak. Idempotens (többszöri hívás ugyanazt az eredményt adja).
- DELETE: Egy erőforrás törlése a megadott URI-n. Idempotens.
- PATCH: Egy erőforrás részleges módosítása. Nem idempotens.
- HEAD: Ugyanaz, mint a GET, de a szerver nem küld vissza kérés törzset, csak a fejlécet. Hasznos az erőforrás metaadatainak ellenőrzésére.

- OPTIONS: Megtudható vele, milyen metódusok engedélyezettek egy adott erőforráson.
- TRACE: A kérés visszaküldése, biztonsági okokból a szervereken nem engedélyezett

Állapotkódok: A szerver által visszaadott 3 jegyű számok, amelyek a kérés eredményét mutatják. Kliensnek minden regisztrált állapotkódot tudnia kell értelmeznie.

- **1xx (Informational):** A kérés előrehaladását, kapcsolat állapotát jelzi, gyakorlatban ritkán használt
  - 100 Continue: A kérés nem került elutasításra, szerver feldolgozza a kérést
  - 101 Switching Protocols: A szerver készenáll a használt alkalmazási protokoll lecserélésére
- **2xx (Success):** A kérést sikeresen fogadta, értelmezte és elfogadta a szerver.
  - 200 OK: A kérés sikeres volt.
  - 201 Created: A kérés eredményeként új erőforrás jött létre.
  - 204 No Content: A kérés sikeres volt, de nincs visszaküldendő tartalom.
  - 206 Partial Content: A szerver teljesítette a kérést a reprezentáció egy vagy több részével történő átvitelével.
- **3xx (Redirection):** További művelet szükséges a kérés teljesítéséhez (átirányítás).
  - 300 Multiple Choices: A kéréshez tartozó erőforrás többféle reprezentációban érhető el.
  - 301 Moved Permanently: Az erőforrás véglegesen új helyre került.
  - 302 Found: Az erőforrás ideiglenesen új helyre került.
  - 304 Not Modified: Az erőforrás nem változott az utolsó kérés óta (kliens használhatja a gyorsítótárazott verziót).
- **4xx (Client Error)**: A kliens hibát követett el.
  - 400 Bad Request: A szerver nem tudta értelmezni a kérést.
  - 401 Unauthorized: A kérés hitelesítést igényel.
  - 403 Forbidden: A szerver megtagadja a kérést, még ha a kliens hitelesített is.
  - 404 Not Found: A kért erőforrás nem található a szerveren.
  - 405 Method Not Allowed: A kéréshez használt metódus nem engedélyezett az erőforráson.
- **5xx (Server Error):** A szerver hibát követett el a kérés feldolgozása során
  - 500 Internal Server Error: Általános szerver hiba.
  - 501 Not Implemented: A szerver nem támogatja a kérés teljesítéséhez szükséges funkcionalitást. Leggyakrabban akkor fordul elő, ha a kliens olyan HTTP metódust vagy más olyan funkciót kért, amelyet a szerver nem ismer vagy nem implementált.
  - 503 Service Unavailable: A szerver átmenetileg nem elérhető (túlterhelés vagy karbantartás miatt).

Tartalomegyeztetés: Egy erőforráshoz több, alternatív reprezentáció kínálása és ezek közül a legmegfelelőbb kiválasztás, amikor egy reprezentációt kell szolgáltatni.

Proaktív egyeztetés: A kliens a kérés fejléceiben jelzi a felhasználói ágensnek megfelelő preferenciáit (pl. User-Agent, Accept (médiatípus meghatározása), AcceptLanguage, Accept-Encoding, Accept-Charset (karakterkódolás)), a szerver pedig a lehetőségei közül kiválasztja

a legjobban illeszkedőt, és a válasz fejléceiben jelzi a választását (pl. Content-Type, Content-Language, Content-Encoding).

Reaktív egyeztetés: A szerver egy listát küld, melyből a felhasználói ágens választ.

Sütik: Egy kis méretű szöveges adatcsomag, név-érték pár és kapcsolódó metaadatok, melyeket egy eredet szerver küld a felhasználói ágensnek, melyet a kliens eltárol. A sütik lehetővé teszik, hogy az alkalmazások tárolják a felhasználóval kapcsolatos információkat (pl. bejelentkezési adatok, preferenciák) a böngészés során. Állapotmentes web esetén eléri, hogy állapotot tartson fel a kliens számára. Felhasználási területek:

- Munkamenet-kezelés (Session Management): felhasználók bejelentkezett állapotának fenntartása
- Személyre szabás (Personalization): felhasználói preferenciák, beállítások, mint például nyelvválasztás, vagy akár személyre szabott tartalom megjelenítése
- Követés (Tracking): Tevékenységek követése több oldalon keresztül

Fontos attribútumok süti esetén:

- **Expires, Max-Age**: Meghatározza, hogy meddig érvényes a süti. Ha nincs megadva, a süti a böngésző bezárásakor törlődik (munkamenet süti). Ha meg van adva, perzisztens süti lesz.
- **Domain**: mely tartományok számára hozzáférhető a süti. Alapértelmezés szerint csak az a tartomány, amelyik beállította.
- **Path**: Meghatározza, mely útvonalak (path-ok) alatt küldje vissza a böngésző a sütit az adott tartományon belül.
- **HttpOnly**: Ha be van állítva, a süti nem hozzáférhető a kliens oldali scriptekből (pl. JavaScript).
- **Secure**: Ha be van állítva, a süti csak HTTPS kapcsolaton keresztül kerül elküldésre a szervernek, soha nem titkosítatlan HTTP-n keresztül.
- **SameSite**: kontrollálja, hogy a böngésző mikor küldje el a sütit a harmadik fél weboldalaknak (cross-site kérések esetén).

Munkamenet süti: Ideiglenes sütik, amelyek a böngésző bezárásakor törlődnek. Elsősorban munkamenet-kezelésre használják.

Perzisztens süti: Meghatározott lejárati dátummal vagy időtartammal rendelkeznek. A böngésző bezárása után is megmaradnak a felhasználó gépén a lejáratig.

5. A WEB JELÖLŐNYELVEI: XML ÉS HTML DOKUMENTUMOK FELÉPÍTÉSE

XML: Általános célú jelölőnyelv. Elsősorban adatok struktúrált tárolására és szállítására tervezték. Nem a megjelenítés a fő célja. Az "Extensible" (kiterjeszthető) szó azt jelenti, hogy a felhasználók maguk definiálhatják a saját tag-jeiket az adatok leírására.

Alapfogalmak

- **Elemek:** nyitó tag + tartalom + záró tag
- **Tagek:** az elemek határait jelölik (<termek> </termek>)
- **Attribútumok:** kiegészítő adatok az elemekhez (id="123")

XML dokumentum részei

1. XML deklaráció (<?xml version="1.0"?>)
2. Egyetlen gyökérelem
3. Hierarchikus fészkelés

Jól formázott XML: Egy gyökéreleme van, minden tag le van zárva, helyes egymásba ágyazás, attribútumok idézőjelben, speciális karakterek kezelése (<, >)

Érvényes XML: A jól formázott XML akkor érvényes, ha megfelel egy sémának (DTD vagy XSD).

DTD (dokumentumtípus-definíció): Definiálja, milyen elemek szerepelhetnek a dokumentumban, milyen attribútumaik lehetnek, és hogyan ágyazhatók egymásba. Korlátozott lehetőségeket kínál az adattípusok definiálására. Általában a dokumentum elején lévő deklaráció hivatkozik a DTD-re.

HTML: a web elsődleges leíró nyelve. A HTML-ben elemek, attribútumok és attribútum-értékek meghatározott szemantikája van. Ezen elemeket csak rendeltetésszerűen szabad használni. A HTML dokumentum fő részei:

- **DOCTYPE deklaráció**: Meghatározza, hogy a dokumentum melyik HTML szabványt használja.
- **html elem**: A teljes dokumentum gyökéreleme. Minden más HTML elem ezen belül helyezkedik el.
    - **head rész**: A dokumentum metaadatait tartalmazza. Ide tartozik például: az oldal címe, karakterkódolás, stíluslapok, külső fájlokra való hivatkozások, JavaScript kapcsolatok.
    - **body rész**: A böngészőben megjelenő tartalmat tartalmazza. Ide kerülnek: szövegek, címsorok, képek, hivatkozások, táblázatok, listák, űrlapok.

6. Stíluslap nyelvek

Stíluslap nyelv: A stíluslap nyelvek célja, hogy elválasztsák a dokumentum tartalmának struktúráját (amit jelölőnyelvek, mint a HTML vagy XML definiálnak) annak megjelenítésétől. Leírják, hogyan formázza és jelenítse meg a böngésző vagy más megjelenítő alkalmazás a dokumentum elemeit.

CSS (Cascading Style Sheets): Ez a legelterjedtebb stíluslap nyelv a weben. A HTML és az XML dokumentumok vizuális megjelenésének befolyásolására szolgál. CSS szabályokból (rules) állnak, amelyek a következőt tartalmazzák:

- **Szelektor** (Selector): Meghatározza, mely HTML (vagy XML) elemekre vonatkozik a szabály
    - **Univerzális Szelektor** (*): Kiválaszt minden elemet a dokumentumban.
    - **Típus/Elem Szelektor** (Type/Element Selector): Kiválaszt minden elemet egy adott HTML tag neve alapján.

    - **Osztály Szelektor** (Class Selector .): Kiválaszt minden elemet, amely rendelkezik egy adott class attribútummal. Egy oldalon több elemnek is lehet ugyanaz az osztálya.
    - **Azonosító Szelektor** (ID Selector #): Kiválaszt egy egyedi elemet azonosító (id attribútum) alapján. Az id-nek egyedinek kell lennie az egész HTML dokumentumon belül.
    - **Attribútum Szelektor** (Attribute Selector []): Kiválaszt elemeket meglévő attribútumaik vagy attribútum-értékeik alapján.
- **Kombinátorok**: két vagy több szelektor összekapcsolására szolgálnak, hogy kiválasszanak elemeket más elemekhez viszonyított kapcsolatuk alapján.
    - **Leszármazott Kombinátor** (Descendant Combinator - szóköz): Kiválaszt minden olyan E elemet, amely egy A elem leszármazottja (bárhol benne a hierarchiában).
    - **Gyermek Kombinátor** (Child Combinator >): Kiválaszt minden olyan E elemet, amely egy A elem közvetlen gyermeke.
    - **Szomszéd Testvér Kombinátor** (Adjacent Sibling Combinator +): Kiválaszt minden olyan E elemet, amely közvetlenül követi egy A elemet, és ugyanazon a szinten vannak (ugyanaz a szülőjük).
    - **Általános Testvér Kombinátor** (General Sibling Combinator ~): Kiválaszt minden olyan E elemet, amely követi egy A elemet a kódban, és ugyanazon a szinten vannak (ugyanaz a szülőjük), függetlenül attól, hogy közvetlenül követi-e.
- **Pseudo-osztályok** (Pseudo-classes :): speciális kulcsszavak, amelyek olyan elemeket választanak ki, amelyek egy bizonyos állapotban vannak, vagy amelyek a dokumentum struktúrájában egy bizonyos pozícióban helyezkednek el.
    - Állapot alapú pseudo-osztályok: pl.: :hover, :focus, :active …
    - Struktúra/Pozíció alapú pseudo-osztályok: pl.: :first-child, :nth-child(n), :empty, …
- **Pseudo-elemek** (Pseudo-elements ::): "virtuális" elemeket vagy az elem bizonyos részeit célozzák meg, amelyek nem léteznek explicit módon a HTML struktúrában. o ::before, ::after, ::first-line, …
- **Deklarációs blokk** (Declaration Block): Kapcsos zárójelek ({}) közé zárt egy vagy több deklarációból áll.
- **Deklaráció** (Declaration): Egy tulajdonság-érték párból áll, pontosvesszővel elválasztva
- **Tulajdonság** (Property): A stílus-jellemző neve
- Érték (Value): A tulajdonsághoz rendelt érték

"Cascading" (Egymásra Halmozódás) elv: Ez a CSS egyik legfontosabb jellemzője. Meghatározza, hogy ha több stílus definíció is vonatkozik ugyanarra az elemre. Szabályai:

- **Eredet** (Origin): Böngésző alapértelmezett stílusai < Felhasználói stílusok < Szerzői stílusok (az oldal készítője) < Szerzői !important szabályok < Felhasználói !important szabályok < Inline stílusok.
- **Specificitás** (Specificity): Egy ID szelektor (#azonosító) specifikusabb, mint egy osztály szelektor (.osztalynev), ami specifikusabb, mint egy elem szelektor (p). A specifikusabb szabályok felülírják az általánosabbakat.

- **Sorrend** (Order): Ha az eredet és a specificitás megegyezik, a később deklarált szabály nyer.
- **Öröklődés** (Inheritance): Bizonyos stílusok (pl. betűszín, betűtípus) öröklődnek a szülő elemről a gyermek elemekre, hacsak nincs explicit módon felülírva.

7. JSON

JSON (Javascript Object Notation): A JSON egy könnyűsúlyú, ember által olvasható és könnyen értelmezhető formátum strukturált adatok cseréjére. Elsősorban webalkalmazásokban használják a kliens (böngésző) és a szerver közötti adatkommunikációra (API-k válaszformátuma), de konfigurációs fájlokban és adat tárolására is népszerű. XML előnyei a hátrányai nélkül. Alapvető struktúrái:

- **Objektum** (Object): Kulcs-érték párok rendezetlen gyűjteménye. Kapcsos zárójelek ({}) közé van zárva. Minden kulcsnak string-nek kell lennie (kétszeres idézőjelben), amelyet egy kettőspont (:) követ, majd az érték. A kulcs-érték párok vesszővel (,) vannak elválasztva.
- **Tömb** (Array): Rendezett értéklisták. Szögletes zárójelek ([]) közé van zárva. Az elemek vesszővel (,) vannak elválasztva. A tömb elemei bármilyen érvényes JSON típusúak lehetnek, beleértve objektumokat vagy más tömböket is.

Támogatott adattípusok:

- String: Karakterlánc, kétszeres idézőjelben
- Number: Szám (egész vagy lebegőpontos), nincs idézőjelben
- Boolean: Logikai érték, true vagy false
- Null: Üres érték, null
- Object: Beágyazott JSON objektum ({...})
- Array: Beágyazott JSON tömb ([...])

Az JSON előnye, hogy egyszerű, könnyen olvasható és könnyen feldolgozható adatcsere-formátum. Az XML-hez képest kisebb méretű, ezért gyorsabb adatátvitelt és hatékonyabb feldolgozást tesz lehetővé. Nyelvfüggetlen formátum, így szinte minden programozási nyelv támogatja. Különösen előnyös webalkalmazásokban, mert JavaScript-ben natívan kezelhető objektumstruktúrát biztosít, ezért a webes API-k legelterjedtebb adatcserélő formátuma lett.

Hátránya, hogy nem rendelkezik szabványos, beépített sémamechanizmussal, így az adatok szerkezetének ellenőrzése nehezebb lehet. Az XML-lel összehasonlítva korlátozottabb adattípusokat támogat, például nincs natív dátum- vagy bináris adattípus kezelése. A szabvány szerint kommenteket sem lehet elhelyezni benne, ami megnehezítheti a dokumentálást. Emellett nem támogat névtereket és külön attribútumfogalmat, ezért kevésbé alkalmas nagyon összetett dokumentumszerkezetek leírására.

8. Számítógép-hálózatok osztályozási szempontjai. Hálózati rétegmodellek. IP technológia címzési rendszere, és vezérlése. Forgalomirányítás elve és az útválasztási kategóriák jellemzése. TCP és UDP mechanizmusok.

1. Számítógép-hálózatok osztályozási szempontjai

Méret és kiterjedés alapján:

- **PAN (Personal Area Network):** Nagyon kis területű hálózat, egyetlen személy közvetlen környezetében (pl. Bluetooth eszközök, vezeték nélküli fejhallgató). Néhány méteres hatótáv.
- **LAN (Local Area Network):** Helyi hálózat, kisebb földrajzi területet fed le (pl. egy épület, iroda, iskola). Tipikusan nagy átviteli sebesség jellemzi, kisebb késleltetéssel. Néhány km-es hatótáv
- **MAN (Metropolitan Area Network):** Városi hálózat, egy város vagy nagyobb kampusz területét fedi le. LAN-ok összekapcsolásával jön létre. Hatótávolság 10-50 km.
- **WAN (Wide Area Network):** Nagy kiterjedésű hálózat, városokat, országokat, kontinenseket köt össze (pl. az Internet). Általában alacsonyabb átviteli sebesség jellemzi, mint a LAN-t, és bérelt vonalakat vagy publikus hálózatokat használ. Hatótávolság több száz vagy ezer km.
- **GAN (Global Area Network):** Világszintű hálózat (gyakorlatilag az Internet maga, mint globális WAN).

Adatátviteli ráta alapján:

- **Klasszikus**: kbps…Mbps
- **Nagysebességű**: 100Mbps … Tbps

Tulajdonjog alapján:

- **Nyilvános hálózatok** (Public Networks): Szolgáltatók (pl. távközlési cégek, internetszolgáltatók - ISP-k) tulajdonában és üzemeltetésében lévő hálózatok, amelyeket a nagyközönség vagy több szervezet használhat díj ellenében. A legnagyobb és legismertebb példa az Internet.
- **Magánhálózatok** (Private Networks): Egyetlen szervezet (cég, intézmény, magánszemély) tulajdonában és üzemeltetésében lévő hálózatok, amelyek kizárólag az adott szervezet/személy saját használatára szolgálnak. Példák: Vállalati LAN-ok, magán WAN-ok, VPN-ek (Virtual Private Network) amelyek publikus hálózaton hoznak létre titkosított, privát kapcsolatot.

Topológia alapján:

- **Busz** (Bus): Minden eszköz egy közös kommunikációs vonalra csatlakozik.
- **Csillag** (Star): Minden eszköz egy központi csomóponthoz (hub, switch) csatlakozik.
- **Gyűrű** (Ring): Az eszközök kör alakban csatlakoznak egymáshoz, az adat egy irányban halad.
- **Fa** (Tree): A csillag topológia hierarchikus kiterjesztése.
- **Háló** (Mesh): Minden eszköz közvetlenül csatlakozik több másik eszközhöz. Lehet teljes (minden mindennel) vagy részleges.
- **Hibrid** (Hybrid): Több alap topológia kombinációja.

Kommunikáció iránya szerint:

- Simplex (Egyirányú): Az adatátvitel csak egyetlen irányban lehetséges. A **kommunikációban** részt vevő egyik fél mindig csak küld, a másik pedig mindig csak fogad. Pl.: hagyományos rádió
- **Half-Duplex** (Fél-duplex): Az adatátvitel mindkét irányban lehetséges a kommunikációs csatornán, de nem egyidejűleg. Amikor az egyik fél küld, a másiknak fogadnia kell, és fordítva. Pl.: walkie-talkie
- **Full-Duplex** (Teljes-duplex): Az adatátvitel mindkét irányban, egyidejűleg lehetséges a kommunikációs csatornán. A kommunikációban részt vevő mindkét fél egyszerre tud adatot küldeni és fogadni. Pl.: hagyományos telefonbeszélgetés

2. Hálózati rétegmodellek

A hálózati kommunikáció komplexitásának kezelésére rétegmodelleket használnak. Ezek absztrakciós szintekre osztják a kommunikációs folyamatot, ahol minden réteg a felette lévő rétegnek nyújt szolgáltatásokat, és az alatta lévő réteg szolgáltatásait használja. A két legfontosabb modell az OSI és a TCP/IP. Rétegelés céljai:

- **Komplexitás csökkentése**: A hálózati kommunikáció összetett feladatát kisebb, kezelhetőbb részekre bontja
- **Modularitás**: Az egyes rétegek függetlenül fejleszthetők és módosíthatók anélkül, hogy ez befolyásolná a többi réteget
- **Szabványosítás és Interoperabilitás**: Tiszta interfészeket definiál a rétegek között, lehetővé téve különböző gyártók eszközeinek és szoftvereinek együttműködését
- **Könnyebb karbantartás és hibaelhárítás**: A problémák könnyebben lokalizálhatók egy adott rétegre
- **Új technológiák bevezetésének megkönnyítése**: Egy réteg technológiájának cseréje vagy frissítése nem igényel változtatást a többi rétegben

Beágyazás (Encapsulation): Amikor adat halad lefelé a rétegmodellben (a feladónál), minden egyes réteg hozzáadja a saját protokolljához tartozó vezérlőinformációt (fejlécet/láblécet) a felsőbb rétegből kapott adatblokkhoz (SDU). Ez a hozzáadott információ a PDU részévé válik, és az alatta lévő réteg számára szolgáltatásokat nyújt.	↔	**Kiemelés (Decapsulation):** Amikor adat halad felfelé a rétegmodellben (a fogadónál), minden egyes réteg feldolgozza és eltávolítja a saját protokolljához tartozó vezérlőinformációt a kapott PDU-ból, majd a fennmaradó adatot továbbítja a magasabb rétegnek. Ez a folyamat addig tart, amíg az eredeti alkalmazási adat el nem éri a legfelső réteget.

Darabolás (Fragmentation): Egy nagyméretű adatcsomagot (PDU-t) kisebb darabokra osztásának folyamata, ha az meghaladja az adott hálózati kapcsolaton maximálisan átvihető egység méretét (Max. Transmission Unit). Minden létrejött kisebb darab önálló csomagként kerül továbbításra.	↔	**Visszaállítás (Reassembly):** A darabolás során létrejött kisebb csomagok újra egyesítésének folyamata az eredeti, nagyméretű adatcsomaggá a célállomáson. Ez biztosítja, hogy a felsőbb réteg az eredeti, teljes adatot kapja meg.

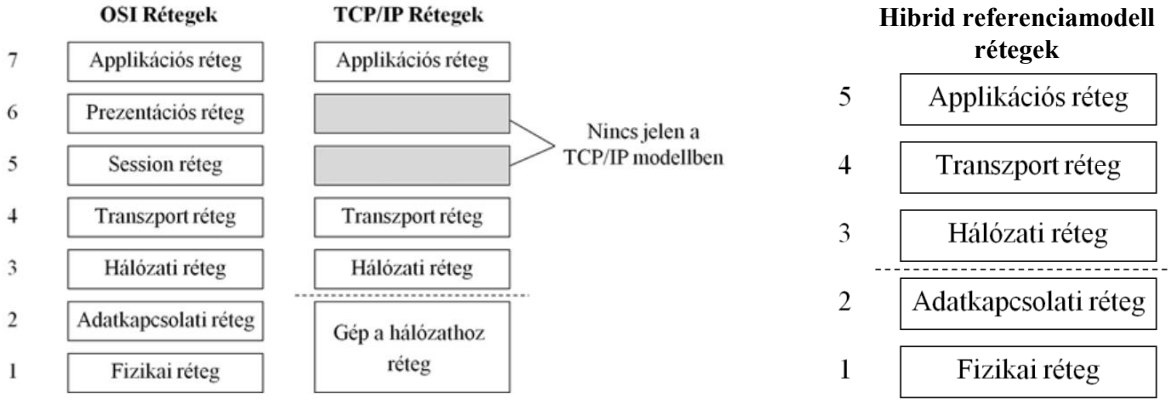
OSI modell (Open Systems Interconnection Model): Egy referencia modell, nem maga a protokollkészlet. Hét rétegből áll.

1. **Fizikai réteg** (Physical Layer): Meghatározza a fizikai átviteli közeg (kábelek, vezeték nélküli jelek) és az elektromos/optikai jelek specifikációit. A bitek átvitelével foglalkozik.
2. **Adatkapcsolati réteg** (Data Link Layer): Kezeli az adatátvitelt a hálózat közvetlenül összekapcsolt pontjai között. Felelős a fizikai címzésért (MAC címek), hibafelismerésért és néha javításért.
3. **Hálózati réteg** (Network Layer): Felelős a logikai címzésért (IP-címek) és a forgalomirányításért (routing) a hálózatok között (forrástól célig). Itt található az IP és ICMP.
4. **Szállítási réteg** (Transport Layer): Biztosítja a megbízható vagy nem megbízható adatátvitelt a forrás és a cél folyamatok között (végponttól végpontig). Itt található a TCP és az UDP.
5. **Viszony réteg** (Session Layer): Kezeli a kommunikációs munkameneteket, a kapcsolatok felépítését, fenntartását és bontását.
6. **Megjelenítési réteg** (Presentation Layer): Felelős az adatok formátumáért, kódolásáért, titkosításáért/dekódolásáért.
7. **Alkalmazási réteg** (Application Layer): Közvetlenül az alkalmazások számára nyújt hálózati szolgáltatásokat (pl. HTTP, FTP, SMTP)

TCP/IP: Ez az Internet alapjául szolgáló protokollkészlet modellje. Gyakorlat-orientáltabb, mint az OSI modell. Általában négy rétegből áll:

1. **Hálózati hozzáférési réteg** (Network Access Layer): Tartalmazza az OSI 1. és 2. rétegének funkcionalitását (pl. Ethernet, Wi-Fi protokollok, eszközmeghajtók). Kezeli a fizikai átvitelt és a helyi hálózati hozzáférést.
2. **Internet réteg** (Internet Layer): Megfelel az OSI 3. rétegének (IP, ICMP, IGMP). Felelős a csomagok továbbításáért hálózatok között.
3. **Szállítási réteg** (Transport Layer): Megfelel az OSI 4. rétegének (TCP, UDP).
4. **Alkalmazási réteg** (Application Layer): Tartalmazza az OSI 5., 6., és 7. rétegének funkcionalitását (pl. HTTP, FTP, SMTP, DNS).

Hibrid referenciamodell: A hibrid referenciamodell a számítógépes hálózatok rétegelt szerkezetének egy olyan szemléleti megközelítése, amely az OSI-modell és a TCP/IP modell előnyeit ötvözi. Gyakran 5 rétegből áll, egyesítve az OSI modell alsóbb rétegeinek (fizikai, adatkapcsolati) részletességét a TCP/IP modell felsőbb rétegeinek (hálózati, szállítási,



alkalmazási) gyakorlatiasságával. Célja egy jól használható, mégis kellően részletes keret biztosítása a hálózati kommunikáció megértéséhez és tanításához.

3. IP TECHNOLÓGIA CÍMZÉSI RENDSZERE, ÉS VEZÉRLÉSE

IP protokoll: Az IP az Internet réteg (OSI 3., TCP/IP Internet réteg) legfontosabb protokollja. Csomagkapcsolt hálózatokban biztosítja az adatok továbbítását a forrástól a célig a hálózatok között. Az IP egy kapcsolat nélküli (connectionless) és nem megbízható (unreliable) protokoll. Az IP a logikai címzést használja az eszközök azonosítására a hálózaton.

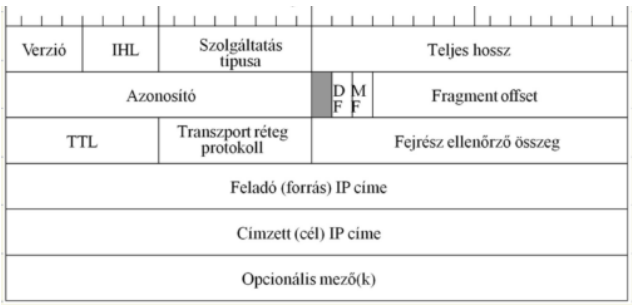
IPv4: 32 bites címeket használ (2^32, azaz kb. 4.3 milliárd egyedi cím). Pontokkal elválasztott négy decimális, 8 bites számként írják. Minden IPv4 cím két részből áll: Hálózati rész (Network Part) és Host rész (Host Part). Az alhálózati maszk (Subnet Mask) határozza meg, mely bitek tartoznak a hálózati részhez és melyek a host részhez. Az alhálózat (Subnetting) lehetővé teszi egy nagyobb hálózat felosztását kisebb alhálózatokra a címterület hatékonyabb kihasználása és a forgalom csökkentése érdekében. Az, hogy milyen címosztályba osztályba tartozik a hálózatunk, meghatározza, hogy hány szám a négyből a hálózatszám. **Címosztályok**:

- **A Osztály**: Nagyon nagy hálózatokhoz (első bájt 1-126). Kevés hálózat, sok állomás hálózatonként.
- **B Osztály**: Közepes hálózatokhoz (első bájt 128-191). Közepes számú hálózat és állomás.
- **C Osztály**: Kis hálózatokhoz (első bájt 192-223). Sok hálózat, kevés állomás hálózatonként.

IPv6: 128 bites címeket használ, hogy a címhiányt megoldja. Hexadecimális számokkal írják, kettősponttal elválasztva (pl. 2001:0db8:85a3:0000:0000:8a2e:0370:7334). Sokkal több címet biztosít, és egyszerűsíti a hálózatok kezelését is.

IPv4 adatcsomag (datagram) fejlécének felépítése:

- **Verzió** (Version): 4 bit. Az IP protokoll verzióját jelzi, IPv4 esetén értéke mindig 4.
- **Internet Fejléc Hossz** (Internet Header Length - IHL): 4 bit. A fejléc hosszát adja meg 32 bites szavakban. A minimum érték 5 (20 bájt), a maximum 15 (60 bájt).
- **Szolgáltatás Típusa** (Type of Service - ToS): 8 bit. Meghatározza a csomag prioritását és a kívánt szolgáltatás minőségét (QoS).
- **Teljes Hossz** (Total Length): 16 bit. A teljes IP csomag méretét (fejléc + adat) bájtban adja meg. Maximális mérete 65 535 bájt.
- **Azonosító** (Identification): 16 bit. Egyedi azonosító az eredeti IP datagramhoz. Ezt a mezőt a fragmentáció (darabolás) és újraösszerakás során használják
- **Vezérlőbitek** (Flags): 3 bit
    - Foglalva (Reserved): 1 bit (mindig 0)



    - Ne Darabold (Don't Fragment - DF): 1 bit. Ha be van állítva (1), a csomag nem darabolható.
    - További Darabok (More Fragments - MF): 1 bit. Ha be van állítva (1), az azt jelenti, hogy további darabok következnek az eredeti csomagból
- **Töredék Eltolás** (Fragment Offset): 13 bit. A jelenlegi töredék helyét adja meg az eredeti, nem fragmentált csomag adataihoz képest, 8 bájtos blokkokban
- **Élettartam** (Time To Live - TTL): 8 bit. Meghatározza, hogy a csomag hány hopon (routeren) mehet keresztül, mielőtt eldobásra kerül. Minden router csökkenti az értékét eggyel
- **Protokoll** (Protocol): 8 bit. Az IP adatrészében található felsőbb rétegbeli protokoll típusát azonosítja (pl. TCP, UDP, ICMP)
- **Fejléc Ellenőrző Összeg** (Header Checksum): 16 bit. A fejléc integritásának ellenőrzésére szolgál. Minden hopon újra kell számolni a TTL mező változása miatt
- **Forrás IP Cím** (Source IP Address): 32 bit. A csomagot küldő eszköz IP címe
- **Cél IP Cím** (Destination IP Address): 32 bit. A csomag címzettjének IP címe.
- **Opciók** (Options): Változó méretű (0-40 bájt). Opcionális információkat tartalmazhat, például útvonalrögzítésre vagy biztonságra vonatkozóan. Kitöltő bájtokkal (Padding) egészül ki, hogy a fejléc mérete mindig 32 bites szavak többszöröse legyen

Vezérlés (ICMP - Internet Control Message Protocol): Az ICMP az IP kísérő protokollja (az Internet réteg része). Nem adatok szállítására szolgál, hanem hibaüzenetek küldésére és diagnosztikai funkciók ellátására az IP forgalommal kapcsolatban. ICMP csomag felépítése:

- **Típus** (Type): 8 bit. Meghatározza az ICMP üzenet általános kategóriáját (pl. hibaüzenet vagy információs üzenet). Az értéktartomány 0-255.
- **Kód** (Code): 8 bit. Azonos típuson belül további finomítást ad az üzenet specifikus jelentésére vonatkozóan.
- **Ellenőrző** Összeg (Checksum): 16 bit. Az ICMP üzenet teljes egészének (fejléc és adatrész) integritását ellenőrzi.
- **Változó Rész** (Variable Part / Rest of Header): 32 bit. Ez a mező az üzenet típusától és kódjától függően eltérő információkat tartalmaz. Lehet például egy azonosító és sorozatszám (pl. Echo Request/Reply üzeneteknél), vagy mutató a hibás IP fejléc mezőre, vagy más diagnosztikai adat.

4. Forgalomirányítás elve és az útválasztási kategóriák jellemzése

Forgalomirányítás: A forgalomirányítás (routing) az a folyamat, amely során egy hálózati eszköz (az útválasztó vagy router) meghatározza a legjobb útvonalat egy adatcsomag számára a forrástól a célig a hálózatok hálózatán keresztül.

Útválasztási algoritmusok: Meghatározzák, hogyan építik fel és tartják karban az útválasztók az útválasztási tábláikat.

- **Távolság-vektor** (Distance-Vector): A routerek minden elérhető célra (gép vagy hálózat) nyilvántartják, hogy a legjobb úton milyen irányban milyen távolsággal érhető el az adott cél (távolságvektor). A forgalomirányítók ezen információkat meghatározott időközönként kicserélik egymással. Az új információk birtokában a

routerek ellenőrzik, hogy szükséges-e változás valamelyik eddig ismert legjobb úttal kapcsolatban.
- **Link állapot** (Link-State): Minden útválasztó információt gyűjt a hálózat topológiájáról és a linkek állapotáról, majd saját maga számolja ki a legjobb útvonalakat minden célhoz

Útválasztási kategóriák:

- **Minimális útválasztás:** Teljesen izolált (router nélküli) hálózati konfiguráció. Forgalomirányítási döntés nem csak a forgalomirányítókon történik, hanem minden csomóponton a csomag küldése előtt.
- **Statikus útválasztás** (Static Routing): Az útválasztó táblákat manuálisan konfigurálja a hálózati adminisztrátor. Egyszerű, de nem reagál automatikusan a hálózat változásaira vagy hibáira. Kis, stabil hálózatokban vagy alapértelmezett útvonalak beállítására használják.
- **Dinamikus útválasztás** (Dynamic Routing): Az útválasztók útválasztási protokollok segítségével automatikusan megosztják egymással az útválasztási információkat, és dinamikusan frissítik a táblájukat a hálózat változásainak megfelelően. Nagy, komplex, gyakran változó hálózatokban használják (pl. az Internet).
- **Alapértelmezett útválasztás** (Default Routing): Egy speciális statikus útvonal, amely meghatározza, hová küldje az útválasztó azokat a csomagokat, amelyek céljához nincs specifikus bejegyzés az útválasztási táblában.

5. TCP és UDP mechanizmusok

Ezek a Szállítási réteg (OSI 4., TCP/IP Szállítási réteg) legfontosabb protokolljai, amelyek a forrás és cél folyamatok közötti kommunikációt kezelik a portszámok segítségével.

TCP (Transmission Control Protocol):

- **Kapcsolat-orientált** (Connection-oriented): Az adatátvitel megkezdése előtt egy megbízható kapcsolatot épít ki a két végpont között a háromutas kézfogás (three-way handshake) segítségével.
    - 1. Kliens -> Szerver: SYN (Synchronize) üzenet a kapcsolatlétesítési kéréssel.
    - 2. Szerver -> Kliens: SYN-ACK (Synchronize-Acknowledge) üzenet a nyugtázással és a szerver szinkronizálási kérésével.
    - 3. Kliens -> Szerver: ACK (Acknowledge) üzenet a szerver kérésének nyugtázásával. Ekkor létrejön a kapcsolat.
- **Megbízható** (Reliable): Garantálja az adatok célba érkezését és a helyes sorrendet. Újraküldi az elveszett vagy hibás csomagokat, és sorba rendezi a sorrenden kívül érkezett csomagokat. Nyugtázásokat (Acknowledgements - ACK) használ az átvitel visszaigazolására.
- **Adatfolyam-orientált** (Byte Stream-oriented): Az adatokat folyamként kezeli, nem különálló üzenetekként.
- **Áramlásvezérlés** (Flow Control): Megakadályozza, hogy a gyors küldő túlterhelje a lassú fogadót (ablakméretekkel szabályozva).
- **Torlódásvezérlés** (Congestion Control): Kezeli a hálózati torlódást, csökkentve az adatküldési sebességet, ha a hálózat túlterhelt.

- **Felhasználás**: Olyan alkalmazások, ahol a megbízhatóság kritikus (pl. webböngészés - HTTP/HTTPS, fájlátvitel - FTP, e-mail - SMTP).

UDP (User Datagram Protocol):

- **Kapcsolat nélküli** (Connectionless): Nem épít ki előre kapcsolatot, egyszerűen elküldi az adatcsomagokat (datagramokat).
- **Nem megbízható** (Unreliable): Nem garantálja az adatok célba érkezését, a sorrendet vagy az ismétlődések elkerülését. Nincs nyugtázás vagy újraküldés.
- **Üzenet-orientált** (Message-oriented): Különálló üzeneteket kezel.
- **Minimális többletköltség** (Low Overhead): Egyszerűsége miatt kisebb a protokoll által generált többletforgalom és gyorsabb a feldolgozása.
- **Felhasználás**: Olyan alkalmazások, ahol a sebesség fontosabb a megbízhatóságnál, és elfogadható némi adatvesztés vagy sorrendi eltérés (pl. valós idejű streaming média - videó, hang, online játékok, DNS).

1. Diszkrét és folytonos valószínűségi eloszlás fogalma. Nevezetes eloszlások: binomiális, Poisson, egyenletes, exponenciális, normális.
Adatszerkezetekkel kapcsolatos alapfogalmak: absztrakció, absztrakt adatszerkezetek. Elemi adatszerkezetek: lista, verem, sor. Halmaz, multihalmaz, tömb. Fák ábrázolása, bejárások, keresés, beszúrás, törlés.

1. DISZKRÉT ÉS FOLYTONOS VALÓSZÍNŰSÉGI ELOSZLÁS FOGALMA

Diszkrét valószínűségi eloszlás: Olyan valószínűség eloszlás, ahol a valószínűségi változó megszámlálhatóan sok értéket vehet fel (diszkrét számú lehetséges értéke a valószínűségi változónak): lehet véges, vagy megszámlálhatóan végtelen. Eloszlás függvénye: x_i lehetséges értékeket és a hozzájuk tartozó p_i valószínűségek. Ezek a pi valószínűségek összege 1. A függvény képe lépcsőzetes. Ha az X valószínűségi változó diszkrét, akkor az eloszlásfüggvény mindig lépcsőzetes, ami minden számnál pont annyit ugrik, mint az adott szám valószínűsége. Az ugrások addig folytatódnak, amíg el nem érjük az 1-et.

Folytonos valószínűségi eloszlás: Olyan valószínűség eloszlás, ahol a valószínűségi változó folytonos értékű. (Olyan valószínűségi változó, amely folytonos mennyiséget reprezentál: idő, távolság…). Az eloszlás függvény értéke 0 és 1 közé esik, határértékei 0 és 1. A függvény képe folytonos. Ha az X valószínűségi változó folytonos, akkor az [a, b] számok között bármilyen valós értéket felvehet. Ilyenkor az eloszlásfüggvény is folytonos, ami a-ig nullát vesz fel, a és b közt növekszik, és b után 1 lesz. Az a és b közt a függvény "működik", a többi helyen 1 vagy 0 lesz.

2. NEVEZETES ELOSZLÁSOK: BINOMIÁLIS, POISSON, EGYENLETES, EXPONENCIÁLIS, NORMÁLIS.

Binomiális (diszkrét): Ha egy megfigyelés során n független kísérletet hajtunk végre és a sikeres kimenetel (vagyis az A esemény) bekövetkezésének valószínűsége minden kísérletnél p = P(A), akkor a sikeres kimenetelek (A esemény bekövetkezéseinek) száma binomiális eloszlású valószínűségi változó. *Megadja annak valószínűségét, hogy n darab egymástól független kísérletből pontosan k siker következik be, ha egy kísérlet sikerének valószínűsége p.*

Poisson (diszkrét): Ha egy esemény bizonyos intervallumban (ami időbeli és térbeli intervallum is lehet) véletlenszerűen következik be és az intervallumban ismerjük az átlagos bekövetkezésének számát (λ - lambda), akkor az adott intervallumban az esemény bekövetkezéseinek száma paraméterű, Poisson-eloszlású valószínűségi változónak tekinthető. Tehát véletlenül bekövetkező esetek egyenletesen oszlanak el. *Azt adja meg, hogy egy adott idő- vagy területegységen belül egy esemény hányszor következik be, ha az események egymástól függetlenek és átlagosan állandó (λ) gyakorisággal történnek.*

Egyenletes (folytonos): Az esemény bekövetkezése egy adott intervallumon azonos valószínűségű.

Exponenciális (folytonos): Az exponenciális eloszlás folytonos valószínűségi eloszlás, amely két egymást követő esemény közötti várakozási idő modellezésére szolgál, ha az események egymástól függetlenül, állandó átlagos gyakorisággal következnek be. Paramétere a λ (lambda), amely az események átlagos gyakoriságát jelenti. *Megadja, mekkora a valószínűsége annak, hogy egy esemény bekövetkezéséig mennyi idő telik el.*

Normális (folytonos): Leggyakrabban használt eloszlás a statisztikában. Azt írja le, hogyan helyezkednek el az adatok egy átlagérték körül. Az eloszlásra jellemző, hogy a legtöbb adat az átlag közelében található, és az átlagtól távolodva az értékek előfordulási valószínűsége fokozatosan csökken. Grafikonja szimmetrikus, harang alakú görbe.

3. ADATSZERKEZETEKKEL KAPCSOLATOS ALAPFOGALMAK: ABSZTRAKCIÓ, ABSZTRAKT ADATSZERKEZETEK.

Absztrakció: Egy adott objektumra/elemre vonatkozó lényeges tulajdonságokat vesszük figyelembe, a lényegtelen részleteket pedig elhanyagoljuk. (Pl.: attribútumok értékeinek elhagyása). Nagyobb jelentőségű részletek kiemelése, hasonlít az általánosítás folyamatára. Számítástechnikában/szoftverfejlesztésben: alapvető fogalom, szorosan kapcsolódik az elmélethez és a tervezéshez, modellezésnek is nevezhető. A modellek absztrakciótípusoknak is tekinthetők, a valóság szempontjainak általánosítása alapján. (Logikai modell)

Absztrakt adatszerkezetek: Az adatelemek és a köztük lévő kapcsolat logikai modellje, független az adattárolástól: nem tartalmazza a műveletek implementációját, sem a fizikai adattárolást

Absztrakt adatszerkezetek csoportosítása:

- Elemek típusa és szerkezete szerint:
  - **Homogén**: azonos típusú elemek
    - **Struktúra nélküli**: adatelemek közt nincs kapcsolat, függetlenek pl.: halmaz, multihalmaz
    - **Asszociatív**: pl.: tömb
    - **Szekvenciális**: Az adatelemek mindig két másik elemmel vannak kapcsolatban pl.: lista, sor, verem
    - **Hierarchikus**: Lista általánosítása, az elemeknek akárhány rákövetkezője lehet, de az elemnek csak egy mezője létezik. Tartalmaz egy kitüntetett elemet, melynek nincs megelőzője, illetve olyat, amelyeknek nincs rákövetkezője pl.: fa, hierarchikus lista
    - **Hálós**: Fa általánosítása, melynek nincs kitüntetett eleme pl.: gráf
  - **Heterogén**: változó típusú elemek: rekord
- Elemek száma szerint:
  - **Statikus**: állandó számú elemek
  - **Dinamikus**: változhat az elemek száma, lehet 0 is
  - Minden adatszerkezet véges számú elemet tartalmaz!
- Tárolás/Reprezentáció szerint:
  - **Folytonos**: vektor szerű, összefüggő tárhely, kezdőérték és tárhelyméret ismerete szükséges
  - **Szétszórt**: láncolt, adatrész + mutató

4. ELEMI ADATSZERKEZETEK: LISTA, VEREM, SOR.

Lista: Homogén elemű, szekvenciális szerkezetű, dinamikus elemszámú adatszerkezet. Létezik első és utolsó eleme. Minden nem utolsó elemnek van rákövetkezője, minden nem első elemnek van megelőzője. Minden elemnek van értéke és pointere. A lista feje (head) a

lista első eleme. A lista farka (tail) a lista az első elem nélkül. A lista vége (end) a lista utolsó eleme. A lista mérete a lista elemeinek száma.

A listán végezhető műveletek:

- létrehozás, bővítés (beszúrás), törlés, csere, rendezés, keresés, bejárás és feldolgozás.

Speciális műveletek:

- **Push:** elem beszúrása a lista elejére,
- **Pop:** az első elem törlése,
- **Inject:** elem beszúrása a lista végére,
- **Eject:** az utolsó elem törlése.

A lista reprezentációja lehet:

- **folytonos**, amikor tömbként tároljuk,
- **szétszórt**, vagyis láncolt lista, ahol az adatelem tartalmazza az értéket és egy vagy két mutatót. A láncolt lista lehet egyszeresen vagy kétszeresen láncolt, illetve ciklikus is. A lista elejét a HEAD mutató jelöli, üres lista esetén HEAD = NIL.

Verem: Homogén elemű, szekvenciális szerkezetű, dinamikus elemszámú adatszerkezet. A verem a leggyakrabban alkalmazott adatszerkezet a tömb után. A verem egy olyan speciális lista, ahol az előbb definiált műveletek közül csak az létrehozás, módosítás, törlés és két módosító művelet van megengedve. LIFO (Last In First Out) elv alapján működik: minden új elemet a sor végére helyezünk, az utoljára betett elem kerül ki először.

Alapműveletek

- **Push:** elem hozzáadása a verem tetejére
- **Pop:** a legfelső elem eltávolítása
- **Top:** a legfelső elem elérése

Műveletek

- létrehozás, bővítés (Push), törlés (Pop), elérés csak a felső elemhez, feldolgozás LIFO elv szerint

Csere, rendezés, keresés és bejárás nem értelmezett.

Reprezentáció

- **Folytonos:** vektorként kezeljük, az első elem a verem alja. Használunk egy mutatót, ami a verem tetejét mutatja. Bővíteni csak akkor tudunk, ha van hely a vetem tetején
- **Szétszórt:** egyszeresen láncolt listával valósítható meg.

Sor: Homogén elemű, szekvenciális szerkezetű, dinamikus elemszámú adatszerkezet. A sor egy speciális lista. FIFO (First In First Out) elven működik: minden új elemet a sor végére helyezünk, az elsőként betett elem kerül ki először. Létezik üres és teli sor.

Alapműveletek

- **Enqueue (Put):** elem hozzáadása a sor végére
- **Dequeue (Get):** első elem elérése és törlése
- **Front:** első elem elérése

Műveletek

- létrehozás, bővítés (Enqueue), törlés (Dequeue), elérés az első elemhez, feldolgozás FIFO elv szerint.

Csere, rendezés, keresés és bejárás nem értelmezett.

Speciális sorok

- **Kettős sor:** mindkét végén lehet beszúrni és törölni.
  - **Input korlátozott kettős sor:** csak a végén lehet bővíteni, viszont eltávolítani mindkét végéről.
  - **Output korlátozott kettős sor:** csak első elemhez lehet hozzáférni, de mindkét végéről bővíthető

Reprezentáció

- **Folytonos (tömb):**
  - fix kezdetű: Sor első elemének indexe rögzített. Indexmutató jelöli a sor végét. Elérhető elem indexe MINDIG 1, elem törlésekor elemeket balra toljuk
  - vándorló: Két indexmutató: sor eleje és sor vége. Bővíteni a sor vége index után lehet (sor vége mutató), hozzáférni az sor eleje indexű elemhez lehet (sor eleje mutató). Ha a rendelkezésre álló tárhely végére ér, eltolódik az egész vektor (frissül a két mutató).
  - ciklikus sor: Két indexmutató: sor eleje és vége. A sor ciklikusan vándorol körbe, adatokat nem mozgatjuk. Ha vektor utolsó eleme foglalt, de első nem, akkor az elején folytatom a bevitelt.
- **Szétszórt:** egy irányba láncolt listaként

5. HALMAZ, MULTIHALMAZ, TÖMB.

Halmaz: Homogén elemű, struktúra nélküli, dinamikus elemszámú adatszerkezet. A halmaz a matematikai halmaz fogalmát valósítja meg: véges számú, egyedi és rendezetlen elemeket tartalmaz.

Alapfogalmak és műveletek

- **Eleme:** megadja, hogy egy elem benne van-e a halmazban ($x \in S$)
- **Unió:** két halmaz összes eleme ($A \cup B$)
- **Metszet:** két halmaz közös elemei ($A \cap B$)
- **Különbség:** az egyik halmaz elemei a másik nélkül ($A \setminus B$)

Tulajdonságok: A halmaz megfelel a matematikai halmazműveletek szabályainak:

- kommutativitás,
- asszociativitás,
- disztributivitás,
- idempotencia,
- DeMorgan-azonosságok stb.

Műveletek:

- létrehozás, bővítés (unióval), törlés (különbségképzéssel), elérés (eleme-e), feldolgozás alapműveletekkel.

Csere, rendezés, keresés és bejárás nem értelmezett.

Reprezentáció: Folytonosan, karakterisztikus függvénnyel ábrázolható. Lehetséges elemeket sorba rendezzük, mindegyikhez hozzárendelünk 1-1 bitet. A bit jelzi, hogy az adatelem eleme-e a halmaznak: 0 – nem eleme, 1 – eleme.

Multihalmaz: ugyanaz, mint halmaz, viszont adatelemek ismétlődhetnek (nem egyediek az elemek). Reprezentációja ennek is folytonos, karakterisztikus függvény segítségével történik: lehetséges elemeket sorba rendezzük, mindegyikhez tárhelyet rendelünk (ált. 1 byte), és a tárhelyen tároljuk az elemek előfordulásának számát.

Tömb / Mátrix: Homogén elemű, asszociatív, statikus elemszámú adatszerkezet. Az elemek indexek segítségével érhetők el, a dimenziók száma pedig meghatározza a tömb típusát.

- **Egydimenziós tömb:** vektor
- **Többdimenziós tömb:** olyan tömb, amelynek elemei is tömbök (pl. mátrix)

Fizikailag minden tömb egydimenziós tárolású, sor- vagy oszlopfolytonosan.

Műveletek

- létrehozás, csere (felülírás), logikai törlés, közvetlen elérés index alapján, rendezés, keresés, bejárás, feldolgozás.

Bővítés nincs, mert statikus méretű. Rendezés, keresés, bejárás nem értelmezett.

Reprezentáció: folytonos, tömbként tároljuk memóriában. K kezdőcímke alapján bármely másik elem tárbeli címe meghatározható a címfüggvény segítségével. Többdimenziós tömb esetén tárolni kell a méreteket is.

Ritka mátrix: olyan mátrix, amelyben sok a 0 érték, ezért speciális tárolást alkalmazunk.

- **Folytonos reprezentáció**
  - **3 soros:** sorindex, oszlopindex, érték
  - **4 soros:** plusz mutató a következő ritka elemre
  - **4+2 soros:** további mutatók az első sor- és oszlopelemekhez
- **Szétszórt reprezentáció**
  - **Láncolt lista:** ritka elemek és indexei tárolása láncolt listaként. Szerkezete: sorindex és oszlopindex (növekvő sorrendben), érték, mutató
  - **Multilista:** sorindex, oszlopindex, érték, és két mutató: sor következő ritkaeleme, oszlop következő ritkaeleme.

6. FÁK ÁBRÁZOLÁSA, BEJÁRÁSOK, KERESÉS, BESZÚRÁS, TÖRLÉS

Fa: Homogén elemű, hierarchikus szerkezetű, dinamikus elemszámú adatszerkezet. Minden elem megmondja a rákövetkezőjét, szülő-gyermek kapcsolaton alapszik.

Gyökérelem: elem, amelynek nincs megelőzője, ha fa nem üres, 1 van belőle

Levélelem: elem, amelynek nincs rákövetkezője, bármennyi lehet belőle

Közbenső elem: elem, amelynek van megelőzője és rákövetkezője is: nem gyökér és nem levél elem

Út: gyökérelemből kiinduló, különböző szinteken átmenő, és levélben véget erő, egymáshoz kapcsolódó élek sorozata. Út hossza: az élek száma. Minden levél pontosan egy úton érhető el.

Szintek: egy elem szintje a gyökértől vett távolságával egyezik meg. A gyökér szintje 0.

Magasság/Mélység: A maximális szintszám egy fában.

Részfa: minden közbenső elem egy részfa gyökerének tekinthető. Fa részfákra bontható.

Rendezettlen fa: elemek felcserélhetőek benne, sorrendnek nincs jelentősége

Rendezett fa: ha rendezettnek tekintjük számít az elemek sorrendje a fában

Bináris fa: Olyan fa adatszerkezet, melyben minden adatelemnek legfeljebb két rákövetkezője van. Szigorú bináris fa: minden elemnek 0 vagy 2 rákövetkezője van.

Műveletek:

- létrehozás (üres faként), bővítés egy elemmel vagy részfával, törlés, csere, keresés, elérés és feldolgozás bejárással.

Bejárások:

- **Preorder:** gyökér → bal → jobb
- **Inorder:** bal → gyökér → jobb
- **Postorder:** bal → jobb → gyökér

A bejárás célja, hogy a fa elemeit sorozattá alakítsuk.

Reprezentáció

- **Folytonos:** 3 vektorral:
  - adatvektor: adatelem értéke
  - bal vektor: bal oldali rákövetkező vektor béli indexe
  - jobb vektor: jobb oldali rákövetkező vektor béli indexe

  Fa gyökérelemét a vektorok első eleme írja le
- **Szétszórt:** Láncolt reprezentáció, ahol minden elem:
  - bal mutatót,
  - egy értéket,
  - jobb mutatót tartalmaz.

  A gyökérelemre külön mutató hivatkozik.

Kifejezés fa: Olyan bináris fa, melynek minden közbenső eleme művelet, leveli pedig operandusok.

Bináris keresőfa: A bináris keresőfa olyan rendezett bináris fa, melyben az adatelemek mindegyike rendelkezik egy kulccsal, és minden adatelemre igaz az, hogy az adatelem bal oldali részfájában lévő elemek kulcsai kisebbek, a jobb oldali részfájában lévő elemek kulcsai pedig nagyobbak az elem kulcsánál.

Bináris keresőfa bővítése rekurzívan: Ha üres a fa, akkor a beszúrandó elem lesz a fa egyetlen eleme (levéleleme), és ezzel az algoritmus sikeresen véget ér. Egyébként

összehasonlítjuk a gyökérelem értékét a beszúrandó elemmel. Ha a két elem egyenlő, akkor a beszúrandó elemet nem helyezhetjük el a fában (mert nem szerepelhet benne két azonos értékű elem), és ezzel az algoritmus sikertelenül véget ér. Ha a beszúrandó elem kisebb a gyökérelemnél, akkor a gyökérelem bal oldali részfáját bővítjük a beszúrandó elemmel. Egyébként a gyökérelem jobb oldali részfáját bővítjük a beszúrandó elemmel.

Bináris keresőfa elemeinek törlés rekurzívan: Összehasonlítjuk a gyökérelem értékét a törlendő elemmel. Ha a törlendő elem kisebb a gyökérelemnél, akkor a gyökérelem bal oldali részfájából töröljük az elemet. Ha a törlendő elem nagyobb a gyökérelemnél, akkor a gyökérelem jobb oldali részfájából töröljük a törlendő elemet. Ha a két elem egyenlő, akkor megnézzük, hogy a gyökérelemnek hány rákövetkezője van. Ha a gyökérelemnek egy rákövetkezője sincs (azaz levélelem), akkor egyszerűen törölhető. Ha a gyökérelemnek egy rákövetkezője van, akkor felülírjuk a gyökérelemet azzal a rákövetkező elemmel (azaz egy szinttel feljebb csúsztatjuk a gyökérelem nem üres részfáját). Ha a gyökérelemnek két rákövetkezője van, akkor a gyökérelem értékét felülírjuk a gyökérelem bal oldali részfája legnagyobb (legjobboldalibb) elemének az értékével, majd a gyökérelem bal oldali részfájából töröljük ezt a legnagyobb (legjobboldalibb) elemet.

Bináris keresőfa elemeinek keresése: Bináris keresőfában az elemek rendezettek, bal oldalon a kisebbek, jobb oldalon a nagyobbak, igy a keresés elég egyszerű, vizsgáljuk a gyökért, majd, ha nagyobb a keresett elem jobbra, ha kisebb balra indulunk el, rekurzívan

Kiegyensúlyozott keresőfa: Olyan bináris keresőfa, mely esetén bármely csúcs esetén a baloldali és a jobboldali részfa magassága közti különbség maximum 1.

AVL fa: Olyan kiegyensúlyozott keresőfa, mely beszúrás és törlés után visszaállítja a kiegyensúlyozottságot (Rebalance művelet).

2. Valószínűség fogalma és kiszámításának kombinatorikus módszerei (permutációk, variációk, kombinációk). Feltételes valószínűség, függetlenség, Bayes-formula. Algoritmusok lépésszáma, aszimptotikus jelölések. Beszúrásos rendezés, keresések lineáris és logaritmikus lépésszámmal. Táblázatok, hash függvények, hash táblák. Gráfok, szélességi és mélységi bejárás.

1. Valószínűség fogalma és kiszámításának kombinatorikus módszerei (permutációk, variációk, kombinációk)

Valószínűség: egy esemény bekövetkezésének esélye, melynek értéke a [0, 1] intervallumon van.

Elemi események: a kísérlet lehetséges kimenetelei, mindig egyféleképpen következhetnek be, egymást kizáró események.

Eseménytér: elemi események teljes halmaza egy kísérlet esetén, jele: **Ω**

A: esemény

n: független események száma

k: a bekövetkezésének száma

$k_{A/n}$: A relatív gyakorisága. A relatív gyakoriság fix szám körül ingadozik: P(A), azaz A esemény valószínűsége

- Bármely esemény valószínűsége mindig nem negatív $P(A) \geq 0$
- A biztos esemény mindig bekövetkezik, tehát valószínűsége 1: $P(\Omega) = 1$
- Ha A és B egymást kizáró események: $P(A + B) = P(A) + P(B)$
- Lehetetlen esemény valószínűsége 0. $P(\oslash) = 0$
- Ha A1, A2, … An páronként kizáróak (bármely kettőt tekintve), a valószínűség végesen additív $P(A_1 + A_2 + \ldots + A_n) = P(A_1) + P(A_2) + \cdots + P(A_n)$

Klasszikus valószínűségi mező: Véges sok kimenetel (N), minden kimenetel egyenlő esélyű ($p_i = 1/N$). $P(A) = \frac{kedvező\ eset}{összes\ eset}$

Kombinatorikus módszerek: A kombinatorika azzal foglalkozik, hogy megszámolja, hányféleképpen lehet kiválasztani vagy sorba rendezni elemeket egy adott halmazból, bizonyos feltételek mellett.

Permutációk: A halmaz összes elemének sorba rendezése. Itt számít az elemek sorrendje, és minden elemet pontosan egyszer használunk fel. Elemek száma: n

- **Ismétlés nélküli**: n db különböző elem $P_n = n! = 1 * 2 * \ldots * n$
- **Ismétléses**: n db elem, ahol k_1 , k_2 , … k_r egymással megegyező elemet rendezünk sorba $P_n^{k1\ldots kr} = \frac{n!}{k_1! * k_2! * \ldots k_r!}$

Variációk: A halmaz néhány elemének kiválasztása és sorba rendezése. n db elemből kiválasztunk k db-ot (k ≤ n), számít a kiválasztott elemek sorrendje.

- **Ismétlés nélküli**: n db különböző elem $V_n^k = \frac{n!}{(n-k)!}$

- **Ismétléses**: n db elem, melyek közül többször is választunk (visszatevéses) $V_n^{k,ism} = n^k$

Kombinációk: A halmaz néhány elemének kiválasztása, anélkül, hogy számítana a kiválasztott elemek sorrendje. n db elemből kiválasztunk k db-ot (k ≤ n), sorrendjük nem számít.

- **Ismétlés nélküli**: n db különböző elem $C_n^k = \binom{n}{k} = \frac{n!}{k! * (n-k)!}$
- **Ismétléses**: n db elem, melyek közül többször is választunk (visszatevéses) $C_n^{k,ism} = \binom{n+k-1}{k}$

2. Feltételes valószínűség, függetlenség, Bayes-formula.

Feltételes valószínűség: A esemény valószínűsége, ha B biztosan bekövetkezik. A B esemény bekövetkezése befolyásolja az eseményteret, amiben A bekövetkezésének esélyét vizsgáljuk.

$$P(A|B) = \frac{P(A*B)}{P(B)}$$

Valószínűségek függetlensége: Két esemény akkor és csak akkor független, ha az egyik bekövetkezése nem befolyásolja a másik bekövetkezésének valószínűségét. A és B független események, ha $P(A*B) = P(A)*P(B)$

Több esemény függetlensége: Páronként függetlenek, ha $P(A_iA_j) = P(A_i)*P(A_j)$ *és* $i \neq j$

Teljes függetlenség: Az események páronként függetlenek és bármely események halmazára igaz, hogy $P(A_{i_1}A_{i_2} \dots A_{i_k}) = P(A_{i_1})*P(A_{i_2})*\dots* P(A_{i_k})$

Teljes eseményrendszer: A1, A2, … An, páronként kizáróak és összegük az egész eseménytér.

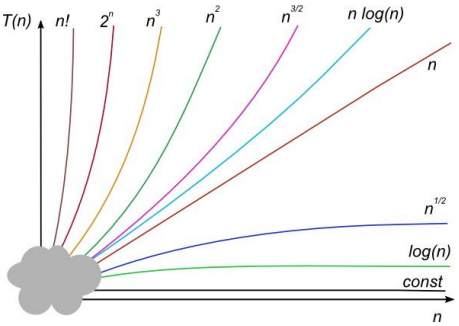
Teljes valószínűség tétele: Ha B1, B2, … Bn, teljes eseményrendszer, valamint A tetszőleges esemény, akkor: $P(A) = P(A|B_1) - P(B_1) + P(A|B_2) - P(B_2) + \cdots P(A|B_n) - P(B_n)$

Bayes-formula: Legyen A és B két, pozitív valószínűségű esemény. A feltételes valószínűség definíciójából: $P(A|B) = \frac{P(A)*P(B|A)}{P(B)}$

Bayes-tétel: Ha A1, A2, … An teljes eseményrendszer, valamint B tetszőleges esemény, akkor: $P(A_k|B) = \frac{P(B|A_j)*P(A_j)}{\sum P(B|A_i)*P(A_i)}$

3. Algoritmusok lépésszáma, aszimptotikus jelölések

Algoritmus lépésszáma: Azt méri, hogy egy algoritmus hány elemi műveletet hajt végre a bemenet méretének (n) függvényében. Elemi művelet lehet például egy összehasonlítás, egy értékadás, egy aritmetikai művelet stb. A lépésszám azt jelzi, hogy az algoritmus futási ideje hogyan függ a bemenet méretétől. Lépésszámok osztályozása:



- **Legjobb eset**: Nem releváns.
- **Átlagos eset**: Nem rossz, de nem sokat használt.

- **Legrosszabb eset**: Ez a mérvadó, mivel ez a legrosszabb eset.

Aszimptotikus jelölések: Az aszimptotikus jelölések az algoritmusok teljesítményének jellemzésére szolgálnak nagy bemeneti méretek mellett. Egy olyan függvényt adnak meg, amely közelíti a lépésszám növekedési ütemét.

Aszimptotikus felső korlát - nagy ordo jelölés (O): Egy algoritmus futási ideje $O(g(n))$, ha létezik olyan pozitív c konstans és n_0 szám, hogy minden $n \geq n_0$ esetén:

$$f(n) \leq c \cdot g(n), \text{ ahol:}$$

- $f(n)$ az algoritmus futási idejét leíró függvény,
- $g(n)$ az aszimptotikus felső korlát,
- c pozitív konstans.

Felső korlátot ad meg. Azt jelenti, hogy az algoritmus lépésszáma legfeljebb olyan gyorsan nő, mint a jelölésben szereplő függvény. Ez gyakran a legrosszabb eset időkomplexitását írja le.

Aszimptotikus alsó korlát – nagy omega jelölés (Ω): Egy algoritmus futási ideje $\Omega(g(n))$, ha létezik olyan pozitív c konstans és $n_0$szám, hogy minden $n \geq n_0$esetén:

$$f(n) \geq c \cdot g(n), \text{ ahol}$$

- $f(n)$ az algoritmus futási ideje,
- $g(n)$ az alsó korlátot adó függvény,
- c pozitív konstans.

Alsó korlátot ad meg. Azt jelenti, hogy az algoritmus lépésszáma legalább olyan gyorsan nő, mint a jelölésben szereplő függvény. Ez gyakran a legjobb eset időkomplexitását írja le.

Aszimptotikus éles korlát – nagy theta jelölés (Θ): Egy algoritmus futási ideje $\Theta(g(n))$, ha léteznek pozitív c_1, c_2 konstansok és n_0 szám, hogy minden $n \geq n_0$ esetén:

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n), \text{ ahol:}$$

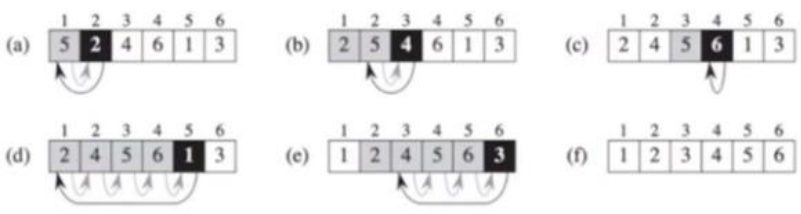
- $f(n)$ az algoritmus futási ideje,
- $g(n)$ az éles korlátot adó függvény.

Szoros korlátot ad meg. Azt jelenti, hogy az algoritmus lépésszáma pontosan olyan gyorsan nő, mint a jelölésben szereplő függvény (azaz a felső és alsó korlát is megegyezik). Ez gyakran az átlagos eset időkomplexitását írja le.

Bármely két f(n) és g(n) függvény esetén f(n)= Θ(g(n)) akkor és csak akkor, ha f(n)=O(g(n)) és f(n)= Ω(g(n)).

4. Beszúrásos rendezés, keresések lineáris és logaritmikus lépésszámmal

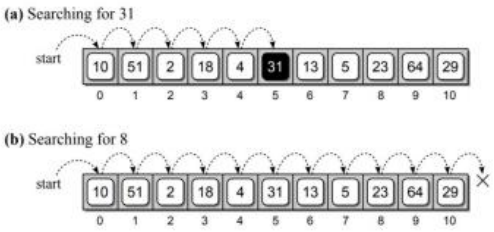
Beszúrásos rendezés – polinomiális lépésszámú rendezés: Tömböt v. listát rendezett és rendezetlen részre osztja. A rendezett rész kezdetben az első elemet tartalmazza (a), majd sorban veszi a rendezetlen rész elemeit, és azokat beszúrja a megfelelő helyre, a rendezett részbe



(b-e). Így egy rendezett tömböt/listát kapunk (f).

- Legjobb eset: Θ(n) (ha a bemenet már rendezett) – lineáris futási idő
- Legrosszabb eset: Θ(n2) (ha a bemenet fordított sorrendben van) –
- Átlagos eset: Θ(n2) - polinomiális futási idő

Lineáris keresés – lineáris lépésszámú keresés:
Egy elemet keres egy adatszerkezetben úgy, hogy az elemeket egymás után, sorban vizsgálja, kezdetétől a végéig. Nem igényel rendezett adatokat.
Bármilyen lista vagy tömb esetén használható. Kis méretű adathalmazokra hatékony, helyben rendez (nem igényel sok extra memóriát).



- Legjobb eset: Ω(1) (az első elem a keresett) – konstans futás idő
- Legrosszabb eset: Θ(n) – lineáris futási idő
- Átlagos eset: Θ(n) – lineáris futási idő

Bináris keresés – logaritmikus lépésszámú keresés: Rendezett adatszerkezetben keres egy adatot felező módszerrel: megvizsgálja a középső elemet, ha az a keresett elem, algoritmus befejeződik. Ha keresett elem kisebb, akkor a bal oldali félben folytatja a keresést, ha nagyobb, akkor jobb oldali félben: veszi a középső elemet stb. Addig folytatja amíg nem találja az elemet, vagy üres tartományhoz nem ér. Lényegében felező módszer. Rendezett adatokat igényel.

- Legjobb eset: Ω(1) (a középső elem a keresett) – konstans futásidő
- Legrosszabb eset: Θ(log2n) – logaritmikus futási idő
- Átlagos eset: Θ(log2n) – logaritmikus futási idő

5. Táblázatok, hash függvények, hash táblák

Táblázatok: Homogén eleműek, dinamikus elemszámúak és asszociatív szerkezetűek. Reprezentációja lehet folytonos/szétszórt, valamint rendezett/rendezetlen. Összetett adatelemeket tartalmaznak, melyek két elemből állnak:

- kulcs: minden esetben egyedi, ez alapján azonosítjuk az elemeket
- érték: maga is lehet összetett, nem szükséges egyedinek lennie

Önátrendező táblázat: Előnyös, ha elemek feldolgozási gyakorisága változó, nagy mértékben eltérő. SEARCH művelet használatakor a megtalált elemet mindig a táblázat elejére szúrjuk be: a leggyakrabban használt elemek a táblázat tetején lesznek. Emiatt a lineáris keresés az ideális. Gyorsan reagál a feldolgozási gyakoriság miatt bekövetkező változásokra.

Közvetlen címzésű táblázat: Olyan táblázat, melyben az elemekhez a kulcs alapján közvetlenül hozzáférünk. Csak akkor hatékony, ha a kulcstér (U) kicsi, ekkor minden elem pozíciója megegyezik a kulcsával.

Hash tábla: Egy adatszerkezet, amely kulcs-érték párokat tárol. A közvetlen címzésű tábla továbbfejlesztett változata: ha túl nagy a kulcstér, akkor hash függvényt alkalmazunk a kulcsból egy belső tömb indexének meghatározására.

Hash függvény: Egy függvény, amely egy tetszőleges méretű bemenetből (kulcsból) egy fix méretű kimenetet generál (hash érték, hash kód). Jó hash függvények esetén a különböző bemenetekhez különböző kimeneteket rendel nagy valószínűséggel, és a kimenet egyenletesen oszlik el a lehetséges kimeneti tartományon. A cél, hogy gyorsan lehessen belőle indexet számolni. A h hash függvény az U kulcstér elemihez rendeli egy adott T hash táblázat pozícióit. Problémája a **kulcsütközés**, amikor több kulcshoz ugyanaz a pozíció tartozik a hash függvény alapján.

Ütközés feloldása:

- **Osztásos módszer**: h hash függvény a k kulcshoz az m különböző pozíció egyikét rendeli hozzá maradékos osztás segítségével: h(k) = k mod m
- **Láncolással**: azonos hash értékkel rendelkező elemeket egy láncolt listában tároljuk. Ha nincs következő elem / nincs a tábla pozícióján érték, akkor NIL elem.
- **Nyílt címzéssel**: minden elem a táblázatban tárolódik, tehát egy pozíció értéke vagy maga az érték, vagy NIL. Kereséskor végig haladunk az elemeken. Előnye: nincs szükség más adatszerkezetre a táblázaton kívül, és nincs szükség mutatók kezelésére. Ha egy adott hash értéke foglalt, a következő szabad helyre szúrjuk be az értéket
- **Láncolt nyílt címzéssel**: érték mellett tárolunk egy mutatót, amely a következő megegyező hash értékű elem valódi pozíciójára mutat. A keresés ezáltal még hatékonyabb, hiszen tudjuk a hash értéket, és végig megyünk a közvetett láncolt listán

6. Gráfok, szélességi és mélységi bejárás

Más tételekben részletezett.

3. Függvények szélsőértéke, függvényvizsgálat. A legkisebb négyzetek módszere. Az elsőrendű logika nyelvének szintaxisa. Változók kötött és szabad előfordulása. A nyelv interpretációja, változókiértékelés. Termek és formulák értéke interpretációban, változókiértékelés mellett. Törvény, ellentmondás, ekvivalencia, következmény. Normálformák, prenex formulák. Logikai kalkulusok

1. FÜGGVÉNYEK SZÉLSŐÉRTÉKE, FÜGGVÉNYVIZSGÁLAT

Szélsőérték: az értelmezési tartománynak azok a pontjai, ahol a függvénynek lokális (egy adott környezetben a legnagyobb vagy legkisebb) vagy globális (az egész értelmezési tartományon a legnagyobb vagy legkisebb) maximuma vagy minimuma van.

Adott $]a, b[$ intervallumon értelmezett x_0, f valós függvény. Ha $f'(x_0) = 0$ akkor x_0 f függvény kritikus pontja/stacionárius pontja (lehetséges szélsőértékhely).

Kritikus pont vizsgálata, hogy valódi szélsőérték-e:

- Első derivált próba: Megvizsgáljuk az első derivált előjelét a kritikus pont előtt és után.
    - Ha az előjel pozitívról negatívra változik, a kritikus pontban lokális maximum van.
    - Ha az előjel negatívról pozitívra változik, a kritikus pontban lokális minimum van.
    - Ha az előjel nem változik (pozitív marad vagy negatív marad), akkor a pontban nincs lokális szélsőérték
- Második derivált próba: Ha a függvény kétszer differenciálható a kritikus pont (x0) környezetében, megnézzük a második derivált ($f''(x_0)$) értékét ebben a pontban.
    - Ha $f''(x_0) > 0$, a pontban lokális minimum van.
    - Ha $f''(x_0) < 0$, a pontban lokális maximum van.
    - Ha $f''(x_0) = 0$, a próba eredménytelen

Függvényvizsgálat:

Értelmezési tartomány: Azoknak a valós számoknak a halmaza, amelyekre a függvény értéke kiszámítható, "értelmezve van". Ez határozza meg, hogy milyen x értékeket helyettesíthetünk a függvénybe.

Értékkészlet: A függvény által felvehető értékek halmaza.

Paritás:

- Páros: Ha a függvény grafikonja szimmetrikus az y-tengelyre. Matematikailag: $f(-x) = f(x)$ minden x esetén az értelmezési tartományból.
- Páratlan: Ha a függvény grafikonja szimmetrikus az origóra. Matematikailag: $f(-x) = -f(x)$ minden x esetén az értelmezési tartományból.

Periodikusság: Azt vizsgáljuk, hogy a függvény grafikonja (és így az értékei) ismétlődnek-e szabályos időközönként (periódusonként). Ha van olyan pozitív p szám (periódus), amire $f(x + p) = f(x)$ igaz az értelmezési tartomány minden x-ére, akkor a függvény periodikus.

Zérushelyek (függvény gyökei): Az értelmezési tartománynak azok az x értékei, ahol a függvény grafikonja metszi az x-tengelyt, vagyis ahol a függvény értéke nulla: $f(x) = 0$.

Értelemzési tartomány azon halmazai, ahol függvény értékének előjele állandó: Azokat az intervallumokat keressük az értelmezési tartományon belül, ahol a függvény értéke vagy mindig pozitív ($f(x) > 0$) vagy mindig negatív ($f(x) < 0$). Ezeket az intervallumokat a zérushelyek és a szakadási helyek határolják.

F határértéke az értelemzési tartomány határpontjaiban: A függvény viselkedését vizsgáljuk az értelmezési tartomány "széleinél", vagyis ahogy az x értékek közelednek az értelmezési tartomány véges határpontjaihoz (akár balról, akár jobbról), vagy ahogy $x \to \infty$ vagy $x \to -\infty$. Ez segít megérteni, hova "tart" a függvény a tartomány szélein.

$$\lim_{x \to inf} f(x) \text{ és } \lim_{x \to -inf} f(x)$$

Monotonitás: Azokat az intervallumokat határozzuk meg az értelmezési tartományon belül, ahol a függvény értéke az x növekedésével mindig növekszik (monoton nő), vagy mindig csökken (monoton csökken). Ezt a viselkedést a függvény első deriváltjának előjele alapján lehet vizsgálni.

Szakadáshelyek: Azok a pontok, ahol a függvény grafikonja "megszakad", vagyis a függvény nem folytonos ezeken a helyeken.

2. A LEGKISEBB NÉGYZETEK MÓDSZERE

Alapvető statisztikai és adatelemzési technika, amelyet arra használnak, hogy a legjobban illeszkedő görbét (leggyakrabban egyenest) találják meg egy adott adathalmazhoz. Folyamatot figyelünk meg és méréseket végzünk (melyek esetleg hibával terheltek). Adott alakú modellek közül kiválasztjuk a mérésre legjobban illeszkedőt.

Adott $t1$,$t2$, … ,tn időpillanatokban az $f1$, $f2$, … , fm megfigyelések. A folyamatot leíró $F(x) = \sum_{j=1}^{n} x_j * l_j(t)$ modell paramétereit keressük úgy, hogy $J(x) = \sum_{i=1}^{m} (F(t_j) - f_j)^2$ minimális legyen (négyzetes eltérés összege). x_j ismeretlen paraméterek, $l_j(t)$ adott függvények.

A modell lehet:

- Egyenes: $F(t) = x_1 + x_2 t$
- Polinomiális: $F(t) = x_1 + x_2 t + x_3 t^2 + \cdots + x_n t^{n-1}$
- Trigonometrikus: $F(t) = x_1 + x_2 \sin(\pi t) + x_3 \cos(\pi t)$

A minimalizálandó függvény a Gauss-féle normálegyenletre vezet: $A^T A x = A^T f$

A: tervezés mátrix, x= ismeretlenek vektora, f: megfigyelt értékek vektora (célunk az x meghatározása). Mindig megoldható: lineáris egyenletrendszerként (pl.: Gauss-eliminációval). A megoldás a legkisebb négyzetes értelemben legjobban közelítő modell paramétereit adja. Ha A mátrix oszlopvektorai lineárisan függetlenek, akkor egyetlen megoldása van, ha függőek, akkor végtelen sok.

Legyen

$$A = \begin{pmatrix} \varphi_1(t_1) & \varphi_2(t_1) & \cdots & \varphi_n(t_1) \\ \varphi_1(t_2) & \varphi_2(t_2) & \cdots & \varphi_n(t_2) \\ \vdots & & & \\ \varphi_1(t_m) & \varphi_2(t_m) & \cdots & \varphi_n(t_m) \end{pmatrix} \in \mathbb{R}^{m \times n},$$

$$f = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{pmatrix} \in \mathbb{R}^m, \qquad x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n,$$

Ekkor

$$Ax = \begin{pmatrix} F(t_1) \\ F(t_2) \\ \vdots \\ F(t_m) \end{pmatrix}$$

3. AZ ELSŐRENDŰ LOGIKA NYELVÉNEK SZINTAXISA

Klasszikus elsőrendű logika nyelve:

$L1 = <LC, Var, Con, Term, Form>$ rendezett ötös

LC: nyelv logikai konstansainak halmaza (logikai összekötők és kvantorok) { ¬, ⊃, ⋀, ⋁, ≡, ∀, ∃, =, (,) }

- negáció (nem): ¬
- implikáció (ha … akkor): ⊃
- konjunkció (és): ⋀
- diszjunkció (vagy): ⋁
- ekvivalencia (akkor és csak akkor): ≡
- univerzális kvantor (minden): ∀
- egzisztenciális kvantor (létezik): ∃
- zárójelek, egyenlőségjel

Var: nyelv változóinak megszámlálhatóan véges halmaza $\{x1, x2 \ldots xn\}$. A nyelv interpretációjakor az értelmezési tartomány elemeire hivatkozhatnak

Con: a nyelv nem logikai konstansainak legfeljebb megszámlálhatóan végtelen halmaza

- F(0): Argumentum nélküli függvények (név konstansok)
- F(n): n argumentumú függvények (függvényjelek)
- P(0): Argumentum nélküli predikátumfüggvény (állítás konstans)
- P(n): n argumentumú predikátumfüggvények (predikátum konstansok) **– boolean kimenet**
- csak elnevezések, hogy mit jelölnek, az csak az interpretáció megadásakor derül ki

Term: terminusok halmaza, változók és függvények uniója. Nevek és névből nevet képző függvények - **egyedet jelöl ki**

Form: formulák halmaza, predikátumok és nyelvi konstansok uniója - **állítást tesz, amely igaz vagy hamis lehet**

Példa:

- F(0) = {Péter, én}
- F(1) = {édesanyja(x)}
- P(0) = {havazik} P(2) = {munkatársak(x,y)}
- ***Péter édesanyja és én munkatársak***: munkatársak(édesanyja(Péter),én)
- Terminusok:
  - Péter
  - édesanyja(Péter)
  - én
- Formulák:
  - havazik
  - munkatársak(édesanyja(Péter),én)

4. VÁLTOZÓK KÖTÖTT ÉS SZABAD ELŐFORDULÁSA.

Egy formulában egy változószimbólum (pl. x) előfordulása a változó konkrét megjelenését jelenti a formula szövegében.

Kötött előfordulás: Akkor beszélünk kötött előfordulásról, ha a változó előfordulása egy őt tartalmazó kvantor (∀x vagy ∃x) hatókörén belül van, ÉS a kvantor éppen ezt a változót köti. A kvantor (pl. ∀x) a hatókörében lévő összes olyan változó-előfordulást köti, amelynek neve megegyezik a kvantor melletti változó nevével (x).

Szabad előfordulás: Akkor beszélünk szabad előfordulásról, ha a változó előfordulása nem kötött. Ez azt jelenti, hogy nincs olyan kvantor, amelynek a hatókörén belül lenne, és amely ezt a változót kötné. Egy szabad előfordulás "független" a kvantoroktól az adott formulán belül.

5. A NYELV INTERPRETÁCIÓJA, VÁLTOZÓKIÉRTÉKELÉS

Az <U, ϱ> univerzum – interpretációs függvény párt az $L1 = \langle LC, Var, Con, Term, Form \rangle$ elsőrendű nyelv egy interpretációjának nevezzük, ha

- U != ∅, azaz U univerzum nemüres halmaz;
- A ϱ (ró) interpretációs függvény a Con halmazon értelmezett függvény, amelyre teljesülnek a következők:
    - Minden argumentum nélküli függvényhez hozzárendeli az U egy elemét
    - Minden n-változós függvényhez hozzárendel egy n-változós függvényt az U felett
    - Minden argumentum nélküli predikátumfüggvényhez igaz vagy hamis kimenetet rendel.
    - Minden n-változós predikátumfüggvényhez hozzárendel egy n változós relációt az U felett

Példa:

- U: város lakóinak halmaza: {lakó0, lakó1, lakó2}
- p: p(Péter) = lakó0, p(én) = lakó1, p(Mari néni) = lakó2
- p(édesanyja(_)) olyan függvény ahol, lakó0 -> lakó2
- p(munkatársak(_,_)) = (lakó0, lakó1)

Változókiértékelés: Legyen $L1 = \langle LC, Var, Con, Term, Form \rangle$ egy elsőrendű nyelv, <U, ϱ> pedig a nyelv egy interpretációja. Erre az interpretációra támaszkodó v értékelésen egy olyan függvényt értünk, mely teljesíti a következőket:

- értelmezési tartománya a nyelv változóinak halmazával egyenlő
- ha x eleme a nyelv változóinak halmazának, akkor v(x) eleme az U halmaznak

A változókiértékelés egy leképezés (függvény), amely a nyelv változóinak halmazából (Var) az interpretáció univerzumába (U) képez.

Példa folytatása: v: értékelés

- v(x)=lakó2, ekkor munkatársak(Péter,x) hamis

6. TERMEK ÉS FORMULÁK ÉRTÉKE INTERPRETÁCIÓBAN, VÁLTOZÓKIÉRTÉKELÉS MELLETT

Ez a folyamat ad precíz értelmet a logikai kifejezéseknek: mit jelentenek, és igazak vagy hamisak-e egy "világban" (interpretációban) a változók aktuális "értékei" mellett (változókiértékelés). Legyen $L^1 =< LC, Var, Con, Term, Form >$ egy elsőrendű nyelv, $< U, \rho >$ a nyelv interpretációja, v pedig az interpretációra támaszkodó értékelés. Ekkor:

- Konstansszimbólum értéke: Ha a egy konstansszimbólum, akkor az az az elem a domainből, amit az interpretáció hozzárendelt az a konstanshoz.
- Változó értéke: Ha x egy változó, akkor az értéke az az elem a domainből, amit a p változókiértékelés hozzárendelt az x változóhoz.
- Függvényalkalmazás term értéke: Ha f egy n-változós függvényszimbólum, és t1,…,tn termek, akkor az f(t1,…,tn) term értéke úgy adódik, hogy vesszük az f szimbólumhoz az interpretáció által rendelt függvényt, és ezt a függvényt alkalmazzuk a t1,…,tn részelemek értékeire.
- 0-változós predikátum igazságértéke: Ha p egy 0-változós predikátumszimbólum (egy egyszerű állítás, ami önmagában igaz vagy hamis lehet, változók nélkül), akkor az igazságértéke az, amit az interpretáció hozzárendelt a p szimbólumhoz.
- Predikátum formula igazságértéke: Ha P egy n-változós predikátumszimbólum, és t1,…,tn termek, akkor a P(t1,…,tn) formula igazságértéke 1 (Igaz) akkor és csak akkor, ha a t1,…,tn termek értékeiből képzett rendezett n-es benne van abban a relációban, amit az interpretáció hozzárendelt a P predikátumszimbólumhoz.
- Negáció formula igazságértéke: Ha A egy formula, akkor a ¬A formula igazságértéke 1 (Igaz), ha A igazságértéke 0 (Hamis), és 0 (Hamis), ha A igazságértéke 1.
- Bináris logikai összekötők formula igazságértéke (Implikáció, Konjunkció, Diszjunkció, Ekvivalencia): Ha A és B formulák, akkor az összetett formulák igazságértéke az A és B formulák igazságértékéből adódik a logikai igazságtáblák szerint:
    - A→B (A-ból következik B): Igaz (1), kivéve ha A Igaz (1) és B Hamis (0).
    - A∧B (A és B): Igaz (1) akkor és csak akkor, ha A is Igaz (1) és B is Igaz (1).
    - A∨B (A vagy B): Hamis (0) akkor és csak akkor, ha A is Hamis (0) és B is Hamis (0). Egyébként Igaz (1).
    - A↔B (A akkor és csak akkor B): Igaz (1) akkor és csak akkor, ha A és B igazságértéke megegyezik.
- Kvantoros formulák igazságértéke: Ha A egy formula és x egy változó:
    - ∀xA (Minden x-re A): A formula igazságértéke 1 (Igaz) akkor és csak akkor, ha az A formula igaz bármely olyan esetben, amikor az x változóhoz a domain összes elemét rendre hozzárendeljük. Ha van a domainben legalább egy olyan elem, amelyhez az x-et hozzárendelve az A formula hamissá válik, akkor az egész ∀xA formula Hamis (0).
    - ∃xA (Létezik olyan x, amire A): A formula igazságértéke 1 (Igaz) akkor és csak akkor, ha létezik a domainben legalább egy olyan u elem, amelyhez az x változót hozzárendelve az A formula igazságértéke 1 (Igaz) lesz. Ha a domain összes eleméhez rendelve az x-et az A formula hamis marad, akkor az egész ∃xA formula Hamis (0).

7. TÖRVÉNY, ELLENTMONDÁS, EKVIVALENCIA, KÖVETKEZMÉNY.

Törvény: Egy formula, mely minden interpretációban igaz

Ellentmondás: Egy formula, mely minden interpretációban hamis, a törvény ellentéte

Ekvivalencia: Két formula logikailag ekvivalens, ha A ⊨ B és B⊨A.

Modell: A elsőrendű nyelv formulahalmazának egy olyan interpretációja és változókiértékelése, melyben a formulahalmaz minden eleme igaz.

Következmény: Az A formulának akkor következménye B formula, ha A minden modellje B-nek is modellje.

8. NORMÁLFORMÁK, PRENEX FORMULÁK

Literálok: Az atomi formulák és a negáltjaik

Elemi konjunkció: literálok konjunkciója. $L_1 \wedge L_2 \wedge \ldots \wedge L_n$

Elemi diszjunkciók: literálok diszjunkciója. $L_1 \vee L_2 \vee \ldots \vee L_n$

Konjunktív normálforma: Elemi konjunkciók diszjunkciója. $C_1 \wedge C_2 \wedge \ldots \wedge C_n$, ahol minden C_j egy elemi diszjunkció.

Diszjunktív normálforma: Elemi konjunkciók diszjunkciója. $D_1 \vee D_2 \vee \ldots \vee D_n$, ahol minden C_j egy elemi diszjunkció.

Prenex formula: A prenex normálforma egy olyan alak, ahol az összes kvantor (univerzális ∀ és egzisztenciális ∃) a formula elején helyezkedik el, és ezt követi egy kvantormentes rész. Egy prenex formulának az általános alakja: $Q_1x_1Q_2x_2 \ldots Q_nx_nB(n = 1,2 \ldots)$ Ahol:

- Q_n lehet ∀ vagy ∃ kvantor.
- x_n különböző változók.
- B egy kvantormentes formula.

Prenexformára hozás:

1. A kötött változók szabályos átnevezésével állítsuk elő a formulánknak az eredeteivel kongruens, így az eredetivel ekvivalens változóiban tiszta alakját.
2. 2 Alkalmazzuk De Morgan kvantoros törvényeit és a kvantorkiemelésre vonatkozó logikai ekvivalenciákat a formula részformuláira, amíg a formulánk prenex alakú nem lesz.

Hozzuk a

$$\forall x(\forall y Q(x,y) \supset \neg\exists x P(x)) \supset \forall y Q(x,y)$$

formulát prenex alakúra.

1. Változóiban tiszta alakra hozás:
$$\forall v(\forall w Q(v,w) \supset \neg\exists z P(z)) \supset \forall y Q(x,y)$$
2. De Morgan törvényeinek alkalmazása:
$$\forall v(\forall w Q(v,w) \supset \forall z \neg P(z)) \supset \forall y Q(x,y)$$
3. Kvantorkiemelés:
$$\forall v \exists w \forall z(Q(v,w) \supset \neg P(z)) \supset \forall y Q(x,y)$$
4. Kvantorkiemelés:
$$\exists v \forall w \exists z \forall y((Q(v,w) \supset \neg P(z)) \supset Q(x,y))$$

9. LOGIKAI KALKULUSOK

Szekvent: Ha Γ és Δ két (esetleg üres) formulahalmaz, akkor a Γ ⊢ Δ egy szekvent.

Érvényes szekvent: Legyen S a Γ ⊢ Δ szekvent, ahol Γ = {$A_1, \ldots, A_n$} és Δ = {$B_1, \ldots, B_m$}; Az S szekvent érvényes, ha minden olyan ϱ interpretáció esetén, ahol $A_1 \ldots A_n$ értékelése

igaz eredményű, B_i értékelése igaz valamely i-re *(ha a bal oldalon minden igaz és a jobb oldalon legalább egy igaz)*.

Szekvens kalkulus: A szekvens kalkulus formális bizonyítási rendszer, amelyben a bizonyítás szekvensekből és levezetési szabályok alkalmazásából épül fel. A levezetési szabályok meghatározzák, hogyan lehet új szekvenseket levezetni korábbiakból. Mivel minden logikai operátor megjelenik a levezetési szabályok egyikében, és a szabály eltávolítja őket, a folyamat akkor fejeződik be, ha nem maradnak logikai operátorok: A formula *lebomlott*. Így a fák leveleinek szekvensei csak atomi szimbólumokat tartalmaznak, amelyeket vagy az axióma bizonyíthat, vagy sem, aszerint, hogy a jobb oldalon látható szimbólumok egyike is megjelenik-e a bal oldalon.

Axióma: formulahalmaz. Egy szekvent akkor axióma, ha ugyanaz a formula szerepel mindkét oldalán. Ekkor a levezetési fa adott ága sikeres. $\Gamma, A \vdash \Delta, A$, ahol A atomi formula, Γ és Δ formulahalmazok.

Levezetési fa: Olyan fa, melynek csúcsaiban szekventek állnak, és a levezetési szabályok szerint épül fel. Gyökere: szekvent amelynek érvényességét bizonyítani szeretnénk A levezetési fa **bizonyítás**, ha minden levele axióma.

4. Függvények, görbék, felületek leírása és számítógépes ábrázolása.
Problémák reprezentálása állapottéren. A megoldás keresése visszalépéssel. Szisztematikus és heurisztikus fa- és gráfkereső eljárások.

1. FÜGGVÉNYEK, GÖRBÉK, FELÜLETEK LEÍRÁSA ÉS SZÁMÍTÓGÉPES ÁBRÁZOLÁSA.

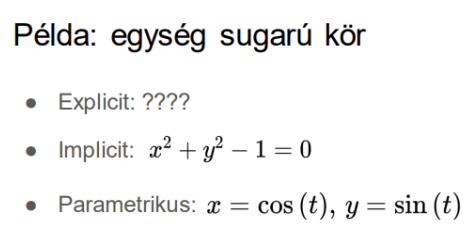
Függvény: Matematikailag egy függvény két halmaz (egy értelmezési tartomány és egy képhalmaz) elemei közötti olyan hozzárendelés, ahol az értelmezési tartomány minden eleméhez pontosan egy képhalmazbeli elem van hozzárendelve. Ha az értelmezési tartomány A, a képhalmaz pedig B, a függvényt jelölhetjük f: $A \rightarrow B$. Minden $x \in A$ elemhez létezik egy egyértelmű $y \in B$ elem, amit f(x)-szel jelölünk. Két dimenzióban a függvény képe egy **görbe**, háromban pedig rendszerint egy **felület**.

Tényleges grafikont (görbét vagy felületet) nem tudunk a képernyőn kirajzoltatni, hiszen végtelen sok pontból áll. Valójában véges sok értéket választunk az értelmezési tartományból, ezekre kiszámítjuk a függvény értékét, és ábrázoljuk. Ez függ a paraméterek számától, mely a dimenzióra is hatással van. Az így meghatározott pontokat páronként összekötjük.

Explicit megadási mód: $y = f(x)$, $ahol\ \mathbb{R} \rightarrow \mathbb{R}$
Vesszük az x értékeket az értelmezési tartományból, behelyettesítjük az f(x) függvénybe, és a kapott y értékek segítségével (x,y) párokat alkotunk, melyeket ábrázolunk. Ezeket a pontokat kötjük össze. Egyértelmű hozzárendelést ad, így függvény megadására is használható.

A függvény által kirajzolt görbe minden koordinátája egy kiszámított pixel, viszont a sok számolás elkerülésére elég minden n-edik értéket kiszámítani, majd az így kapott pontokat összekötni. Könnyen, kevés erőforrás használatával ábrázolható. Hátránya, hogy a leképezés egyértelműsége miatt nem rendelhetünk x-hez több értéket. Emiatt az explicit függvény nem görbülhet maga alá.

Implicit megadási mód: $0 = F(x, y)$, $ahol\ \mathbb{R}^2 \rightarrow \mathbb{R}$
Minden változóra egyszerre számolunk értéket, emiatt nem egyértelműen történik a hozzárendelés (egy helyhez több érték is tartozhat), azaz nem függvény. Minden explicit függvényt át tudunk alakítani implicit alakba, de ez fordítva nem igaz, mivel x és y szerepelhet egy tagban. Ábrázolás: veszünk egy (x,y) pontot és behelyettesítjük F(x,y) függvénybe. Ha 0 értéket kapunk, rajta van a függvényen. Mindig kérdés, hogy milyen sorrendben kell őket összekötni.



Az implicit görbe megjelenítéséhez az $F(x, y) = 0$ egyenletbe kell számpárokat behelyettesíteni. Mivel ez nagyon sok számítást igényel, így nem ideális görbék megjelenítésére.

Paraméteres megadási mód: $p(t) = (x(t), y(t))$, $ahol\ \mathbb{R} \rightarrow V$
A görbe pontjai egy vagy több független változó (paraméter) függvényeként vannak megadva. Ábrázolás hasonlóan az explicithez, viszont nem csak egyértelműen rendel étékeket a pontokhoz („maga alá görbülhet”). Tudjuk a pontok összekötésének sorrendjét, hiszen ezt a paraméter meghatározza. A paraméteres görbéket két külön függvény, $x(t)$ és $y(t)$

segítségével adjuk meg, ahol az egyes függvények a megfelelő *x* és *y* koordinátákat jelölik, és egy bevezetett *t* paramétertől függnek. Ahogyan *t* változik, úgy változnak a görbe pontjainak *x* és *y* koordinátái.

A paraméteres görbék megjelenítéséhez a paraméter különböző értékeihez kiszámítjuk az $(x(t), y(t))$ pontokat, majd ezeket a paraméter növekedésének sorrendjében összekötjük. Ez hatékonyabb és egyszerűbb ábrázolást tesz lehetővé, mint az implicit forma, miközben bonyolultabb vagy zárt görbék leírására is alkalmas. A számítógépes grafikában azért különösen fontos, mert egyértelműen meghatározza a görbe bejárási irányát, valamint 3D görbék és felületek kezelésére is jól használható.

Lagrange-interpoláció: Abban az esetben használjuk, ha polinommal szeretnénk közelíteni egy függvényt. A módszer n alappontból álló sorozatot egy n-1-edfokú polinommal közelít. Két pontból egyenest, három pontból parabolát, efelett hullámos függvényt ad vissza. Sok pontra nagyon magas fokszámú polinom, valamint erőteljes hullámzás.

Hermite-ív: Adott a kezdő és végpont koordinátái és ezek érintővektorai (a függvény pontbeli deriváltjai). A görbe mindig két pont között húzódik, és az érintővektorok meghatározzák a görbe alakját és irányát. Előnye, hogy csökkenti az interpoláció során fellépő oszcillációs problémákat.

Bézier-görbe: A Bézier-görbe a Hermite-ív továbbfejlesztett változata, ahol az érintővektorokat további vezérlőpontokkal helyettesítjük. Adottak a: $P_0, P_1, \dots, P_n$ pontok, ahol az első és az utolsó pont a kezdő- és végpont, a közbülső pontok pedig a görbe alakját befolyásoló kontrollpontok. Ezek alkotják a kontrollpoligont. Az $n + 1$darab ponttal meghatározott Bézier-görbe fokszáma n. A görbe nem feltétlenül halad át a közbülső kontrollpontokon, de azok „húzzák" a görbét. Előnye, hogy könnyen kezelhető, jól szabályozható, és széles körben használják számítógépes grafikában

Felületek: A függvényekben megjelenik a z koordináta is: görbéből felület lesz. Grafikonja: drótháló megjelenítés, bizonyos lépésközönként térbeli görbéket ábrázolunk (paraméter vonalak).

Explicit felületmegadás: $z = f(x, y)$, $ahol\ \mathbb{R}^2 \rightarrow \mathbb{R}$
Minden (x,y) pont esetén meghatároz egy z értéket, mely alapján felírható az adott pont: x-y síkon keresek pontot, majd z irányba tolom el. Könnyű ábrázolás paramétervonalakkal, viszont egyértelmű hozzárendelés, nem tud maga alá görbülő felületet leírni.

Implicit felületmegadás: $0 = F(x, y, z)$, $ahol\ \mathbb{R}^3 \rightarrow \mathbb{R}^2$
Egyszerre számítjuk ki mindhárom ismeretlent. Előny: Maga alá görbülhet a felület. Hátrány: Kiértékelés pontoként lassú lehet.

Paraméteres felületmegadás: $p(u, v) = (x(u, v), y(u, v), z(u, v), ahol\ \mathbb{R}^2 \rightarrow V^2$
t, s: paraméterek, mivel térben vagyunk kettőre van szükségünk. A paraméterek egy paraméter síkot határoznak meg. (más néven paraméter tér). A paramétersíkot dupla ciklussal fedjük le. Ábrázolás: hasonlóan az explicithez, viszont nem csak egyértelműen rendel étékeket a pontokhoz („maga alá görbülhet"). Tudjuk a pontok összekötésének sorrendjét, hiszen ezt a paraméter meghatározza. Ez a leggyakrabban használt felületmegadási módszer.

2. PROBLÉMÁK REPREZENTÁLÁSA ÁLLAPOTTÉREN

A mesterséges intelligencia problémáinak megoldása a probléma megfogalmazásával kezdődik: leírjuk a problémát, reprezentáljuk. Egyik legelterjedtebb reprezentációs technika az állapottér-reprezentáció: a probléma formális leírására szolgál.

p: probléma, p állapotreprezentált, ha megadjuk a p<A, a_0, C, O> négyest, ahol:

- **A: a probléma állapottere**: összes állapot véges, nemüres halmaza
- **a0: kezdőállapot**: amelyből a problémát indítjuk
- **C: célállapotok halmaza**: célállapot, ahová el akarunk jutni
- **O: operátorok véges, nemüres halmaza**: összes végrehajtható cselekvés: állapotok megváltoztatására szolgáló cselekvések. Nem minden operátor alkalmazható egy állapotra, ezért meg szoktuk adni operátor értelmezési tartományát (alkalmazási előfeltételnek is nevezzük). **Hatásdefiníció**: megadja azt az új állapotot, amelyet az operátor végrehajtásával kapunk

3. A MEGOLDÁS KERESÉSE VISSZALÉPÉSSEL

Az algoritmus lényege, hogy a megoldást egy adott úton haladva próbálja megtalálni, és ha zsákutcába kerül, visszalép, hogy más lehetőséget keressen. Nem tároljuk a teljes gráfot, csupán az aktuálisan vizsgált utat tartjuk nyilván: csomópontot tárolunk mely tartalmazza:

- egymáshoz kapcsolt csúcsokat, melyek az állapotokat reprezentálják
- a szülő csúcsot (előző csúcsot)
- az operátort, mellyel a szülőből az aktuális csúcsba jutottunk
- az operátorok teljes halmazát, és jelöli, melyeket próbáltunk már ki

Az algoritmus mindig az utat hosszabbítja egy még nem használt operátorral (az aktuális úton). Ha megfelel az alkalmazási előfeltételnek, akkor végrehajtja: az aktuális csúcs lesz az új szülőcsúcs, az új csúcs pedig az új aktuális csúcs.

Visszalépés: speciális művelet, melyet akkor hajtjuk végre, ha az aktuális csúcsból nem tudunk továbblépni más csúcsra, és az adott csúcs nem célállapotot reprezentál. Visszatérünk a szülő csúcshoz és más operátort hajtunk végre vagy ismét visszalépünk

Vezérlő: algoritmus motorja, ez az egység dönti el, mikor melyik műveletet kell végrehajtani. Vagy operátor műveletet végez, vagy visszalép. Addig végzi ezeket a folyamatokat, amíg célállapot nem teljesül, vagy a megállási feltételek nem teljesülnek (pl.: megengedett futási idő, kifut a lehetséges utakból, stb…). Ha a reprezentációs gráf köröket nem tartalmazó véges gráf, akkor az alap visszalépéses megoldáskereső véges sok keresőlépés megtétele után befejezi a keresést. Ha van megoldás, előállít egy lehetséges megoldást, ha nincs megoldás, azt felismeri.

4. SZISZTEMATIKUS ÉS HEURISZTIKUS FA- ÉS GRÁFKERESŐ ELJÁRÁSOK

Ezek az algoritmusok az állapottér gráfban (vagy keresési fában) keresik a kezdőállapotból a célállapotba vezető utat.

Szisztematikus eljárások: Neminformált eljárásnak is nevezik őket, mivel nem tárolnak semmilyen plusz ’tudást’, heurisztikát a probléma megoldásához. Csak az állapottér-

reprezentációra támaszkodnak, előre meghatározott (szisztematikus) sorrendben járják be az állapotteret (fát vagy gráfot).

Szélességi keresés: Szintek szerint végzi a keresést: feltárja a kezdő állapot összes közvetlen szomszédját, majd szomszédonként, azok szomszédját, szintről szintre. Sor adatszerkezetet használ, az új feltárt szomszédokat sor végére szúrja sorban, a következő vizsgálandó elem a sor elejéről kerül ki Tulajdonságai:

- teljes: megtalálja a megoldást, ha az létezik
- optimális: garantáltan a legrövidebb utat találja meg, ha élek költsége egységes
- magas memóriaigény, különösen, ha sok az elágazás

Egyenletes költségű keresés: mindig a legkisebb útköltségű csomópontot fejti ki először, nem pedig a legkisebb mélységű csomópontot. Egyszerűen belátható, hogy a szélességi keresés is egyenletes költségű keresés, amennyiben minden lépésköltség azonos.

Mélységi keresés: Mindig egy úton halad lefelé, ha zsákutcába jut, a legutolsó elágazás más ágán folytatja. Verem adatszerkezetet használ, az új csúcsot a verem tetejére pakolja, kiveszi és megvizsgálja. Ha ennek nincs elágazása, akkor ismét kiveszi a legfelsőt, és ennek másik elágazását vizsgálja. Tulajdonságai:

- nem teljes: végtelen mélységű ágakban ragadhat
- véges gráfon, ciklusdetektálással teljes
- nem optimális: nem a legrövidebb utat találja meg elsőként/garantáltan
- memória igény alacsonyabb, mint szélességi gráfon
- találhat megoldást, ha az mélyen van.

Mélységkorlátolt keresés: egy mélységi keresés adott mélységkorláttal, ami megakadályozza a túl mélyre hatolást, de nem garantálja a megoldás megtalálását, ha az a korláton kívül esik.

Iteratívan mélyülő mélységi keresés: sorozatos, egyre növekvő mélységkorlátú mélységi kereséseket végez, kombinálva a mélységi keresés alacsony memóriaigényét a szélességi keresés teljességével és optimalitásával.

Heurisztikus eljárások: Ezek az eljárások problémamonospecifikus tudást (heurisztikát) használnak a keresés irányítására, hogy hatékonyabban találják meg a megoldást, különösen nagy állapotterű problémák esetén. **Heurisztikus függvény (h(n))**: Egy függvény, amely minden n állapotra egy nemnegatív értéket ad, ami megbecsüli a költséget az n állapotból a legközelebbi célállapotig. Egy jó heurisztika "közelebb" vezeti a keresést a célhoz. Segít az algoritmusnak eldönteni melyik csúcsot fejtse ki hamarabb.

A mohó legjobbat-először keresés: azt a csomópontot fejti ki a következő lépésben, amelyiknek az állapotát a legközelebbinek ítéli a célállapothoz, abból kiindulva, hogy így gyorsan megtalálja a megoldást. A csomópontokat az algoritmus a $h(n)$ heurisztikus függvénnyel értékeli ki. Az algoritmus „mohó”, mivel minden lépésben igyekszik annyira közel kerülni a célhoz, ahogy csak lehet. Emiatt gyakran gyors, de nem garantálja az optimális megoldás megtalálását, és könnyen zsákutcába juthat.

A* keresés: Egyes csomópontokat értékelő függvénnyel f(n) rendszerezi és mindig a legkisebb értékűt vizsgálja tovább. f(n) figyelembe veszi az aktuálisan megtett út költségét és a célig vezető út költségét. Teljes, valamint megtalálja a legoptimálisabb utat.

5. Mátrix fogalma, műveletek, determináns, rang. Speciális mátrixok, inverz. Mátrix, mint lineáris transzformáció. Sajátérték, sajátvektor.
Intelligens ágensek, ágensek típusai, ágensek környezetét leíró tulajdonságok. Markov döntési folyamatok és adaptív dinamikus programozás alapú ágens, Időbeli különbség (TD-temporal difference) alapon hasznosságot tanuló ágens. Aktív megerősítéses tanuló ágens, felfedezés és kihasználás (exploration és exploitation) módszere. Q-tanuló ágens.

1. MÁTRIX FOGALMA, MŰVELETEK, DETERMINÁNS, RANG

Mátrix: m sorral n oszloppal rendelkező számtáblázatot m*n-es mátrixnak nevezünk. Két mátrix egyenlő, ha dimenziójuk megegyezik és elemeik egyenlőek.

- Dimenzió: m*n
- Elemei: aij
- Főátló: azok az elemek, ahol i=j
- Sorvektorok: m = 1, Oszlopvektorok: n = 1

Összeadás: csak azonos dimenziójú mátrixokon végezhető, elemenkénti művelet.

Skalárral való szorzás: egy skalár (szám) és mátrix szorzata. Minden elemet felszorzunk a skalárral

Mátrix-szorzás: A: m*k és B: k*n dimenziójú, ekkor a szorzatuk egy m*n dimenziójú C mátrix (A oszlopszáma = B sorainak száma). Nem kommutatív (tagok nem felcserélhetőek). Az eredmény i-edik sorának j-edik oszlopában lévő eleme (cij) az A mátrix i-edik sorvektorának és a B mátrix j-edik oszlopvektorának skaláris szorzata.

Transzponálás: m*n-es A mátrix transzponáltja (A^T) egy n*m-es mátrix. elemeit úgy, kapjuk, hogy A sorait oszloponként átrendezzük $(AB)^T=B^T*A^T$

Determináns: Kvadratikus mátrixhoz rendelt skalárérték, mely információt hordoz a mátrixról. Jele: det(A) , $|A|$. A transzponálás nem változtatja meg a determinánst: det(A)=det(A^T). Ha egy mátrix valamely sora (vagy oszlopa) csupa 0, akkor a determináns 0. Ha két sorát (vagy oszlopát) felcseréljük, a determináns előjelet vált (-1- szeresére változik). Ha két sora (vagy oszlopa) megegyezik, akkor a determináns 0. Ha egy sorát (vagy oszlopát) megszorozzuk egy γ skalárral, a determináns is γ-szorosára változik. Ha egy n×n-es mátrix minden sorát (vagy oszlopát) megszorozzuk egy c skalárral (azaz a c·A mátrix determinánsát nézzük), akkor det(cA)=cn det(A). Ha egy sorhoz (vagy oszlophoz) hozzáadjuk egy másik sor (vagy oszlop) skalárszorosát, a determináns nem változik. Ha egy mátrix sorai (vagy oszlopai) lineárisan összefüggők (azaz valamelyik sor/oszlop kifejezhető a többi lineáris kombinációjaként), akkor a determináns 0. Kiszámítási mód:

- **Sarrus-szabály**: 2*2, 3*3 mátrixok determinánsának kiszámítása
- **Gauss-elimináció**: Mátrixot felső háromszög alakra hozzuk: ekkor a főátló elemeinek szorzata megadja a determinánst

Rang: A mátrix sorai (vagy oszlopai), mint vektorok által alkotott vektorrendszer rangját értjük: lineárisan független sorvektorainak maximális száma. Jele: $rang(A)$. A mátrix rangja tehát k, ha oszlopai vagy sorai közül kiválasztható k db lineárisan független, de $k + 1$ db már nem. Kiszámításához Gauss-eliminációval felsőháromszög alakra hozzuk a mátrixot, a nem null sorok száma megadja a mátrix rangját. Jelentősége:

- **Lineáris egyenletrendszerek megoldhatósága**: Ax = b lineáris egyenlet pontosan akkor megoldható, ha rang(A) = rang([A|b])
- **Mátrix invertálhatósága**: akkor invertálható egy n*n-es mátrix, ha A mátrix esetén rang(A) = n (teljes rangú)

2. SPECIÁLIS MÁTRIXOK, INVERZ

Kvadratikus mátrix: Ha m*n-es A mátrix esetén n=m, akkor négyzetes/kvadratikus mátrixról beszélünk

Null mátrix: Minden eleme 0

N dimenziós egységmátrix: n*n-es mátrix, ahol a főátló elemei 1-esek, minden más elem 0

Diagonális mátrix: Szimmetrikus négyzetmátrix, a mátrix megegyezik a transzponáltjával $A = A^{\mathrm{T}}$

Háromszögmátrix: Felső: főátló alatti elemek 0-k. Alsó: főátló feletti elemek 0-k

Inverz mátrix: n*n-es négyzetes mátrix esetén értelmezett Jele: A^{-1}. Inverz akkor létezik, ha a mátrix négyzetes, valamint determinánsa nem 0. Az a mátrix, melyre teljesül: $A * A^{(-1)} = A^{-1} * A = \mathrm{E}$ $(egységmátrix)$

Kiszámítása:

- 2*2-es mátrix: $A^{-1} = \frac{1}{\det(A)} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$
- n*n-es mátrix: Gauss-eliminációval: $[A|\mathrm{E}] \rightarrow [\mathrm{E}|A^{-1}]$

3. MÁTRIX, MINT LINEÁRIS TRANSZFORMÁCIÓ

A: $\mathrm{R}^m \rightarrow \mathrm{R}^n$ típusú függvényt **lineáris leképezésnek** nevezzük, ha bármely $x, y \in R^m, \lambda \in R$ esetén:

- $A(x+y) = A(x) + A(y)$ additív
- $A(\lambda x) = \lambda * A(x)$ homogén

Ha speciálisan m = n, akkor **lineáris transzformációról** beszélünk. Ekkor A minden n elemű oszlopvektorhoz hozzárendel egy m elemű oszlopvektort.

4. SAJÁTÉRTÉK, SAJÁTVEKTOR

Legyen A egy n*n-es mátrix. Egy nemnulla x vektor az A **sajátvektora**, ha létezik olyan λ skalár, hogy Ax = λx. Ekkor λ az A mátrix x sajátvektorához tartozó **sajátértéke**. Meghatározásuk karakterisztikus egyenlettel történik (λ-ban kifejezett polinom):

$$det(A - \lambda \mathrm{E}) = 0$$

Határozzuk meg az alábbi φ sajátértékeit és sajátvektorait!

$$\varphi\colon \mathbb{R}^2 \to \mathbb{R}^2,\ (x,y) \mapsto \varphi(x,y) = (2x - y, -12x + 3y)$$

Már láttuk, hogy φ mátrixa a természetes bázisban $\begin{pmatrix} 2 & -1 \\ -12 & 3 \end{pmatrix}$. Így φ karakterisztikus polinomja

$$\det(A - \lambda E_n) = \left| \begin{pmatrix} 2 & -1 \\ -12 & 3 \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right| = \left| \begin{pmatrix} 2-\lambda & -1 \\ -12 & 3-\lambda \end{pmatrix} \right| =$$
$$= (2-\lambda)(3-\lambda) - (-1)(-12) = \lambda^2 - 5\lambda - 6 = (\lambda+1)(\lambda-6).$$

Így a sajátértékek $\lambda_1 = -1$ és $\lambda_2 = 6$. Például a $\lambda_2 = 6$-hoz tartozó sajátvektorok:

$$\begin{cases} 2x - y = 6x \\ -12x + 3y = 6y \end{cases} \Rightarrow \begin{cases} -4x - y = 0 \\ -12x - 3y = 0 \end{cases} \Rightarrow \begin{pmatrix} x \\ y \end{pmatrix} = t \cdot \begin{pmatrix} 1 \\ -4 \end{pmatrix}, t \in \mathbb{R}.$$

Ha megvan λ, akkor meg tudjuk határozni ezekhez az értékekhez tartozó sajátvektorokat is: $(A - \lambda \mathrm{E})v = 0$ homogén lineáris egyenletrendszer megoldásával.

5. INTELLIGENS ÁGENSEK, ÁGENSEK TÍPUSAI, ÁGENSEK KÖRNYEZETÉT LEÍRÓ TULAJDONSÁGOK

Ágens: egy függvény, mely egy érzékeléssorozatot cselekvésre képez le. $f: P* \rightarrow A$. Fizikai architektúrám fut, hogy ezt a függvényt megvalósítsa.

Intelligens ágens: egy olyan autonóm entitás, amely érzékeli a környezetét érzékelők (szenzorok) segítségével, és cselekvéseket hajt végre a környezetben. Célja: racionálisan cselekedjen, azaz a rendelkezésre álló információk alapján a lehető legjobb eredményt érje el a teljesítmény mérője szerint.

Racionális ágens: a várható teljesítményét maximalizálja a rendelkezésére álló információk (környezet, előzetes tudás) alapján, egy meghatározott teljesítménymérték alapján. A racionális ágens az érzékelésekre támaszkodik, de nem feltétlenül mindentudó. Racionalitás szükséges feltételei: Felfedezés, tanulás, autonómia, Racionális ágens leírása: (pl.: automata taxi):

- **teljesítményérték**: biztonság, szabálykövetés, komfort, útvonal hossza/ideje
- **környezet**: városi közlekedés, autópálya, gyalogosok, időjárás
- **beavatkozók**: kormánykerék, gázpedál, fék, duda, hangszóró, képernyő
- **érzékelők**: videókamera, gyorsulásmérő, mozgásérzékelő, motorszenzorok, GPS 1.1

Ágensek típusai:

- **Egyszerű Reflex Ágens**: Feltétel-cselekvés szabályokon alapul. Az érzékelők által kapott információk alapján értékel ki egy szabálylistát, és annak alapján cselekszik. Nincs memóriája vagy belső modellje a környezetről. Egyszerű "HA-AKKOR" szabályokon alapul. Hátránya: Csak teljes mértékben megfigyelhető környezetben működhet jól.
- **Modell Alapú Reflex Ágens**: Hasonlít a reflex ágenshez, de a memóriájában épít egy modellt a környezetről és arról, hogy a cselekvései milyen hatással vannak rá. Ezáltal az aktuális érzékelést és a modellben felállított állapotot is figyelembe tudja venni a döntésnél.
- **Cél Alapú Ágens** (Goal-Oriented Agent): A modell alapján nemcsak az aktuális állapotot becsüli, hanem a célokat is figyelembe véve választja meg a következő cselekvést. Megvizsgálja, hogyan fog kinézni a környezet egy adott cselekvés után, és célok alapján választ. Keresési vagy tervezési algoritmusokat használ a modellben, hogy olyan cselekvéssorozatot találjon, amely elvezeti a célhoz.
- **Hasznosság Alapú Ágens** (Utility-Oriented Agent): Cél-orientált ágens alapú, de célok helyett a hasznosságot vagy teljesítményértéket veszi alapul a döntésnél. A legmagasabb várható hasznosságú cselekvést választja Akkor fontos, ha több cél is van, vagy a célállapotok különböző mértékben kívánatosak, vagy ha a cél elérése valószínűségi.
- **Tanuló Ágens**: Képes a környezetből tanulni a tapasztalatai alapján. Van egy tanuló eleme, végrehajtó eleme, kritikus eleme és problémagenerátor eleme, amelyek visszacsatolással segítik a tanulási folyamatot. Ez teszi az ágenst adaptívvá és képessé olyan környezetekben is működni, amelyekről kezdetben nincs teljes ismerete.

Ágensek környezetét leíró tulajdonságok:

- **Megfigyelhetőség**: Az ágens szenzorai képesek-e a teljes környezetet megismerni adott pillanatban (teljesen megfigyelhető) vagy sem (részlegesen megfigyelhető)
- **Determinisztikusság/sztochasztikusság**: A környezet következő állapota egyértelműen meghatározott az aktuális állapot és a cselekvés alapján (determinisztikus) vagy véletlenszerű (sztochasztikus)
- **Epizodikusság/sorozatsszerűség**: Az ágens tapasztalata különálló részekre bontható, ahol minden rész független (epizodikus), vagy a jelenlegi cselekvések befolyásolják a jövőbelieket (sorozatszerű)
- **Statikusság/dinamikusság**: A környezet állapota, az idő vagy a cselekvések diszkrétek vagy folytonosak.
- **Diszkrét/folytonos**: A környezet nem változik, amíg az ágens cselekszik (statikus), vagy változhat (dinamikus). Ha a környezet változik, de az ágens nem törődik vele, félig dinamikus.
- **Ágensek száma**: Egyágenses (csak egy ágens van a környezetben) vagy többágenses (több ágens működik együtt vagy verseng)

6. MARKOV DÖNTÉSI FOLYAMATOK ÉS ADAPTÍV DINAMIKUS PROGRAMOZÁS ALAPÚ ÁGENS, IDŐBELI KÜLÖNBSÉG (TD-TEMPORAL DIFFERENCE) ALAPON HASZNOSSÁGOT TANULÓ ÁGENS

Markov döntési folyamatok: Az MDP gy formális keretrendszer a szekvenciális döntéshozatal modellezésére sztochasztikus környezetben. Feltételezi a Markov tulajdonságot: a következő állapot csak a jelenlegi állapottól és a cselekvéstől függ, nem a teljes előzménytől. Komponensei:

- **Állapotok** (S)
- **Akciók** (A)
- **Átmenet valószínűségek** (P) – $P(s' \mid s, a)$ annak valószínűsége, hogy s′ állapotba kerülünk s-ből a akcióval
- **Jutalom függvény** (R) – a jutalom, amit kapunk egy átmenetért
- **Diszkont faktor** (γ) – a jövőbeli jutalmak súlya

Cél: Megtalálni az optimális politikát, ami maximalizálja a várható kumulatív jutalmat.

Állapot-átmenet függvény: Ez írja le, hogy hogyan változik a környezet, ha az ágens egy adott állapotban egy adott cselekvést hajt végre. Ez egy valószínűségeloszlás, Markov döntési folyamatok esetén ez az átmeneti valószínűségnek felel meg.

Megerősítéses tanulás (reinforcement learning - rl): az intelligens ágens cselekedetek végrehajtásán és az arra kapott jutalmakon/büntetéseken keresztül tanul a környezetével való interakcióból. Az ágens célja, hogy megtanuljon egy olyan viselkedési stratégiát (politikát), amely maximalizálja a hosszú távon összegyűjtött jutalmakat. Az ágens nem kap explicit utasításokat, hanem próbálgatással és a megerősítések alapján jön rá a "helyes" viselkedésre. Releváns az aktív ágensek számára, amelyek szekvenciális, sztochasztikus és kezdetben ismeretlen környezetekben működnek. **Aktív ágens**: *maga választja meg a döntéseit/akcióit, gyakran a felfedezés és kihasználás egyensúlyával, hogy megtanulja az optimális kiértékelési*

algoritmust/politikát. Gyakran az MDP-k keretrendszerében modellezik. Az ágens megtanulhatja az állapotok értékét (V(s)) vagy az akciók értékét adott állapotokban (Q(s,a)), vagy közvetlenül a politikát

Adaptív dinamikus programozás alapú ágens: Az ADP olyan módszereket foglal magában a MDP-k megoldására, ahol az ágens megtanulja vagy becsli a környezet modelljét (az átmeneti valószínűségeket és a jutalmakat) a tapasztalataiból. Miután a modellt megtanulta, dinamikus programozási algoritmusokat használ ezen a modellen. Ez egy modell-alapú RL (Reinforcement learning) megközelítés. Hátránya: Meg kell tanulnia és tárolnia kell a teljes modellt, ami nagy állapoterű MDPknél nehéz lehet. **Passzív ágensek** használják főként: olyan ágens, amely nem hoz saját döntéseket a cselekvéseiről. Ehelyett egy rögzített politikát vagy egy előre meghatározott cselekvéssorozatot követ. A tanulási feladata általában nem az optimális politika megtalálása, hanem a saját, rögzített politikájának kiértékelése.

Időbeli különbség (td-temporal difference) alapon hasznosságot tanuló ágens: Az Időbeli Különbség (TD) Tanulás a megerősítéses tanulás egyik alapvető, modellmentes koncepciója. Értékbecsléseket frissít más, jövőbeli értékbecslések alapján (bootstrapping), nem várja meg a teljes epizód végét. A kiértékelő függvény várható értéke kerül frissítésre minden időpillanatban.

TD(0) alapú ágens: Megfigyelt állapotátmeneteket és jutalmakat használ az állapotok értékének módosítására, hogy azok jobban megfeleljenek a Bellman egyenleteknek. Állapotértékeket (V(s)) becsül.

7. AKTÍV MEGERŐSÍTÉSES TANULÓ ÁGENS, FELFEDEZÉS ÉS KIHASZNÁLÁS (EXPLORATION ÉS EXPLOITATION) MÓDSZERE

Aktív megerősítéses tanuló ágens: Olyan ágens, amely választ akciókat a környezetben, nem csak passzívan megfigyel. Saját maga generálja a tanuláshoz szükséges tapasztalatokat.

Kihasználás: Az ágens a pillanatnyilag ismert legjobb akciót választja (a jelenlegi hasznosságbecslések alapján), maximalizálva a várható azonnali jutalmat.

Felfedezés: Az ágens új, még nem próbált akciókat vagy állapotokat látogat meg, hogy további információkat gyűjtsön a környezetről és javítsa a becsléseit, potenciálisan jobb stratégiákat találva.

Dilemma: Egyensúlyt kell találni a jelenlegi legjobb kihasználása és az új, esetleg jobb lehetőségek felfedezése között. Kezelésére számos módszer létezik, pl. ϵ-greedy stratégia.

Epszilon-greed stratégia: Az ágens véletlenszerűen választ felfedezésre (ϵ valószínűséggel), vagy az ismert legjobb megoldást választja kihasználásra ($1 - \epsilon$ valószínűséggel). Az értéket (ϵ) a tanulás előrehaladtával csökkenteni lehet, így az ágens kezdetben többet fedez fel, majd egyre inkább kihasználja az eredményeket.

8. Q-TANULÓ ÁGENS

A Q-tanulás (Q-learning) egy népszerű, modellmentes, off-policy időbeli különbségtanuláson (TD-learning) alapuló megerősítéses tanulási algoritmus. Célja, hogy közvetlenül megtanulja az optimális akció–érték függvényt, vagyis azt, hogy egy adott állapotban melyik akció végrehajtása eredményezi a legnagyobb várható hosszú távú jutalmat. A módszer

modellmentes, mert nem szükséges ismerni a környezet működését vagy átmeneti valószínűségeit. Off-policy jellege miatt az ágens tanulhat úgy is az optimális stratégiáról, hogy közben más stratégiát használ a felfedezéshez.

A Q-tanuló ágens a környezettel való interakció során tapasztalatokat gyűjt. Az egyes állapot–akció párokhoz tartozó $Q(s, a)$értékeket becslésként tárolja (például egy táblázatban), majd ezeket folyamatosan frissíti a kapott jutalmak és a jövőbeli várható jutalom alapján. A frissítés képlete:

$$Q(s_t,a_t)=Q(s_t,a_t)+\alpha[r_{t+1}+\gamma*max_aQ(s_{t+1},a)-Q(s_t,a_t)]$$

ahol:

- $Q(s_t,a_t)$: az aktuális állapot–akció pár régi becslése,
- α: tanulási ráta (learning rate),
- r_{t+1}: a végrehajtott akció után kapott jutalom,
- γ: diszkontfaktor, amely a jövőbeli jutalmak fontosságát szabályozza,
- $\max_a Q(s_{t+1}, a)$: a következő állapotban elérhető legjobb becsült jutalom.

6. Gráf fogalma és megadásának módjai. Egyszerű, irányított és irányítatlan gráfok. Séta, út, összefüggőség. Nevezetes gráfok: páros gráf, teljes gráf, fa, kör, súlyozott gráf. Determinisztikus és nemdeterminisztikus véges automaták, reguláris nyelvek. Környezetfüggetlen grammatikák és környezetfüggetlen nyelvek, veremautomaták.

1. GRÁF FOGALMA ÉS MEGADÁSÁNAK MÓDJAI

Gráf: általánosított fa. Pontok (csúcsok) és élek halmaza, ahol az élek pontokat kötnek össze úgy, hogy minden élre legalább egy, legfeljebb két pont illeszkedik. Minden adatelemnek tetszőleges számú megelőzője és rákövetkezője van, hálós adatszerkezetet alkotva.

Megadási módjai:

- **Szomszédsági lista**: Az egyes i csúcspontokhoz tároljuk a vele szomszédos csúcspontok tömbjét. Ez egy olyan tömböt jelent, amiben n darab szomszédsági listát tartalmazó tömb van, ami csúcspontonként egy szomszédsági listát jelent.
- **Szomszédsági mátrix**: Egy n csúcsponttal rendelkező gráf esetében a szomszédsági mátrix egy n*n méretű, 0-kat és 1-eket tartalmazó mátrix, ahol az i-edik sor j-edik oszlopában akkor és csak akkor van 1, ha az ij él szerepel a gráfban.
- **Éllista**: soronként megadjuk az adott élre illeszkedő két csúcsot

2. EGYSZERŰ, IRÁNYÍTOTT ÉS IRÁNYÍTATLAN GRÁFOK

Egyszerű: Egyszerű gráfban bármely két csúcs között legfeljebb egy él lehet (kizárjuk a többszörös éleket: több él ugyanazon csúcspár között) és nem engedünk meg köröket (nem tartalmaz hurkokat), azaz olyan éleket, amelyek kezdő- és végpontja azonos.

Irányított: Irányított gráf esetében az éleknek irányuk van. Az élek rendezett csúcspárok, pl. (u,v). Jelölése u→v. Az élnek van iránya, csak u-ból v-be lehet közvetlenül "menni".

Irányítatlan: Irányítatlan gráf esetében az élekhez nincs irány rendelve. Az élek rendezetlen csúcspárok, pl. {u,v}. Az él mindkét irányban "járható". Ha van él u és v között, az ugyanaz, mintha v és u között lenne.

3. SÉTA, ÚT, ÖSSZEFÜGGŐSÉG

Séta Csúcsok és őket összekötő élek sorozata. Nem feltétlenül érinti a gráf minden csúcsát, valamint egy élen többször is áthaladhat.

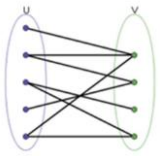
Út: Olyan séta, amelyben minden csúcs különböző. Ebből következik, hogy minden él is különböző.

Kör: Olyan zárt séta ($v_0 = v_k$), amelyben a kezdő/végcsúcson kívül minden csúcs (és él) különböző.

Összefüggőség: Bármely két csúcs (u,v) között létezik út. Azaz el lehet jutni bármely csúcsból bármely másik csúcsba az élek mentén.

4. NEVEZETES GRÁFOK: PÁROS GRÁF, TELJES GRÁF, FA, KÖR, SÚLYOZOTT GRÁF

Páros gráf: A G gráf csúcsok halmaza felbontható két diszjunkt részhalmazra (U és V), úgy, hogy minden él pontosan egy U-beli és egy V-beli csúcsot köt össze. Nincs él U-n vagy V-n belül.



Teljes gráf: Egy irányítatlan gráf, amelyben minden csúcspár között pontosan egy él húzódik. Jelölése K_n, ahol n a csúcsok száma. Minden csúcs fokszáma n−1.

Fa gráf: Egy összefüggő, irányítatlan gráf, amely nem tartalmaz kört és bármely két csúcsát pontosan egy út köti össze.

Kör gráf: Egy n≥3 csúcsból álló gráf, amely pontosan egy körből áll. Jelölése C_n. Minden csúcs fokszáma 2.

Súlyozott: Egy gráf (irányított vagy irányítatlan), amelynek éleihez számértékek (súlyok, költségek) vannak rendelve. Ezek a súlyok reprezentálhatnak távolságot, időt, költséget stb., és fontosak például a legrövidebb út algoritmusoknál. w: súlyfüggvény, mely megadja a súlyokat két csúcs közti élhez

5. DETERMINISZTIKUS ÉS NEMDETERMINISZTIKUS VÉGES AUTOMATÁK, REGULÁRIS NYELVEK

Véges automata: A véges automata egy olyan eszköz, amely képes egy véges számú állapot között váltogatni és szabályok szerint feldolgozni a bemeneti szimbólumokat, amíg el nem éri a döntést: elfogadja vagy elutasítja a bemenetet. A legegyszerűbb absztrakt gépek, véges memóriával.

$M = (Q, \Sigma, \delta, q_0, F_Q)$

- Q: állapotok véges halmaza, tartalmazza az összes állapotot, amiben az automata lehet a számítás során. Például Q={q0,q1,q2}, ahol q0 az induló állapot.
- Σ: bemeneti ábécé, tartalmazza azokat a szimbólumokat, amelyeket az automata képes feldolgozni. Például Σ = {a,b}
- δ: átmeneti függvény. Ha az állapot q0 és a bemenet 'a', akkor az automata az q1 állapotba léphet: δ(q0,a)=q1
- q0∈Q: kezdeti állapot
- F⊆Q: elfogadó állapotok halmaza

Elfogadott szó: Ha egy bemenet feldolgozása után egy automata olyan állapotba kerül, amely az elfogadó állapotok halmazába tartozik, az automata elfogadta azt a bemenetet, más szóval az a szó elfogadott.

Elfogadott nyelv: Egy nyelv egy olyan szavak halmaza, amelyeket egy adott automata elfogad.

Determinisztikus véges automaták: Egy adott bemeneti szimbólumnak mindig egyértelmű hatása van: az automata nem képes "választani" két lehetséges átmenet között. Átmeneti függvény: δ: $Q \times \Sigma \rightarrow Q$

Minden állapothoz és minden bemeneti szimbólumhoz pontosan egyetlen következő állapotot rendel. A gép működése minden lépésben egyértelmű. Egy DFA akkor fogad el (ismer fel) egy bemeneti stringet, ha a string végigolvasása után az elfogadó állapotok valamelyikébe jut.

Nemdeterminisztikus véges automaták: Egy adott állapotból ugyanazzal az inputtal több állapotba is eljuthatunk. Átmeneti függvény: δ: $Q \times (\Sigma \cup \{\epsilon\}) \rightarrow P(Q)$

Egy állapothoz és egy bemeneti szimbólumhoz (vagy az üres szimbólumhoz - ϵ) nulla, egy vagy több lehetséges következő állapotot is rendelhet. Lehetnek ϵátmenetek (átmenet bemeneti szimbólum olvasása nélkül). A gép működése nem egyértelmű. Egy NFA akkor fogad el egy bemeneti stringet, ha létezik legalább egy olyan út a kezdeti állapotból az elfogadó állapotok egyikébe a string végigolvasása után (beleértve az ϵ-átmeneteket). Minden NFA-hoz létezik egy vele ekvivalens DFA, amely pontosan ugyanazt a nyelvet ismeri fel. A nemdeterminizmus nem növeli a véges automaták felismerő képességét. Nemdeterminisztikus véges automatát alakíthatunk determinisztikus automatává.

Reguláris nyelv: Egy Σ ábécé feletti L nyelv (stringek halmaza) pontosan akkor reguláris, ha van olyan véges automata (DFA vagy NFA), amely ezt a nyelvet felismeri.

Pumpálási lemma: Célja bebizonyítani, hogy egy adott nyelv NEM reguláris. Bármely L reguláris nyelv esetében létezik olyan n ≥ 1 természetes szám (amely csak L-től függ), hogy L bármely legalább n hosszúságú u szava felírható u = xyz alakban úgy, hogy

- |xy| ≤ n,
- |y| ≥ 1,
- $xy^iz \in L$ minden i = 0, 1, 2, . . . értékre.

Tehát ha L egy reguláris nyelv, akkor létezik egy olyan hossz, aminél hosszabb tetszőleges L-beli szóban tetszőlegesen kiválasztva egy ennél a hossznál nagyobb részszót igaz, hogy ezen részszóban van egy olyan rész valahol, melyet tetszőlegesen sokszor megismételve (vagy elhagyva) még mindig L-beli szavakat kapunk.

6. KÖRNYEZETFÜGGETLEN GRAMMATIKÁK ÉS KÖRNYEZETFÜGGETLEN NYELVEK, VEREMAUTOMATÁK

Környezetfüggetlen grammatika: Egy környezetfüggetlen nyelvtan alatt egy olyan G = (V, Σ, S, P) rendszert értünk, ahol

- V egy véges, nem üres halmaz, a változók (vagy nemterminálisok) halmaza,
- Σ egy ábécé, amire V ∩ Σ = ∅, a karakterek (vagy terminálisok) halmaza
- S ∈ V a kezdőváltozó,
- P egy véges halmaz, az ún. levezetési (vagy produkciós, illetve átírási) szabályok halmaza.

Ha a G nyelvtan minden szabálya A → α alakú, akkor az L(G) nyelv környezetfüggetlen. A ∈ V egy változó (nemterminális), α ∈ $(V \cup \Sigma)^*$ egy változókból és a Σ elemeiből (terminálisokból és nemterminálisokból) álló tetszőleges, véges hosszú sorozat.

A átírása α-ra független attól, hogy milyen szimbólumok veszik körül A-t a stringben. Környezetfüggetlen grammatikákkal olyan nyelv is megadható, ami véges automatákkal (vagy reguláris kifejezésekkel, vagy reguláris grammatikákkal) nem adható meg. Egy grammatika nem egyértelmű, ha van olyan szó, aminek lényegesen különböző levezetése van. Két levezetés lényegében különböző, ha levezetési fájuk különbözik. Egy nyelv

környezetfüggetlen, ha létezik egy környezetfüggetlen grammatika, amely az összes nyelvi elemet generálni tudja.

Környezetfüggetlen nyelv: A G nyelvtan által generált nyelv: $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$. Egy L nyelv környezetfüggetlen, ha van olyan G környezetfüggetlen nyelvtan, amelyre L = L(G). Környezetfüggetlen nyelvnek hívunk minden környezetüggetlen nyelvtannal előállítható nyelvet.

Például: Az $S \rightarrow aSb \mid \varepsilon$ nyelvtan, amely az $\{a^n b^n \mid n \geq 0\}$ nyelvet generálja.

Pumpálási lemma környezetfüggetlen nyelvekre: Környezetfüggetlen nyelvekre is létezik a reguláris nyelveknél megismert pumpáló lemmához hasonló lemma. Tetszőleges L környezetfüggetlen nyelvhez megadható egy olyan n természetes szám (amely csak az adott nyelvtől függ) úgy, hogy a nyelv minden n-nél hosszabb z szavát fel lehet írni uvwxy alakban, és igazak a következők:

- $|vx| > 0$,
- $|vwx| \leq n$,
- uv^iwx^i y is eleme L-nek, minden $i \geq 0$ értékre.

Veremautomaták: Véges automatákkal csak reguláris környezetfüggetlen grammatikát tudunk elfogadni, viszont veremautomatákkal nem reguláris nyelv is elfogadható. Egy véges automatára hasonlító absztrakt gép, de rendelkezik egy veremmel (stack), amely LIFO (Last-In, First-Out) elv szerint működik. A verembe lehet elemeket betenni (push) és kivenni (pop), és csak a verem tetején lévő elemet lehet olvasni.

$M = \{Q, \Sigma, \Gamma, q0, z0, \delta, F\}$ - Memória verem

- Q: Állapothalmaz
- Σ: bemeneti ábécé
- Γ: veremábécé
- q0: kezdőállapot
- z0: verem kezdő szimbóluma
- δ: állapot-átmeneti függvény. Az átmenet függ az aktuális állapottól, a bemeneti szimbólumtól és a verem tetején lévő szimbólumtól.
- F: végállapotok halmaza

A veremautomata egy szót kap bemenetként, a kezdőállapotból indul ki, és a veremben egy speciális szimbólum, a verem kezdőszimbóluma áll. A működése során az aktuális állapot, a következő bemeneti szimbólum (amely üres szó is lehet), és a verem tetején levő szimbólum ismeretében állapotot vált, és a verem tetején levő szimbólum helyére egy szót ír be (amely szintén lehet üres is).

7. Lineáris egyenletrendszer fogalma és megoldása Gauss eliminációval. Turing-gépek, algoritmikus eldönthetőség, eldönthetetlen problémák, rekurzív és rekurzívan felsorolható nyelvek. Generatív grammatikák és nevezetes nyelvosztályok, a Chomsky hierarchia.

1. LINEÁRIS EGYENLETRENDSZER FOGALMA ÉS MEGOLDÁSA GAUSS ELIMINÁCIÓVAL

Lineáris egyenlet: Egy olyan egyenlet, amelyben a változók csak első hatványon szerepelnek, és csak konstansokkal szorozva és összeadva vannak. Nincs benne változók szorzata, hatványa (az elsőn kívül), gyöke, trigonometrikus vagy más nem-lineáris függvénye.

$$a_1x_1 + \cdots + a_nx_n = b$$

x_n: változók

a_n: konstans együtthatók

b: konstans

n: ismeretlenek száma

Lineáris egyenletrendszer: Egy vagy több lineáris egyenlet halmaza, amelyek ugyanazokat a változókat tartalmazzák. Ha mindegyik egyenlet konstans tagja 0, homogén lineáris egyenletrendszerről beszélünk. Ha csak egy is különbözik 0-tól, inhomogénnek nevezzük a lineáris egyenletrendszert.

Mátrixos alak: Egyenlet bal oldala az alap mátrix, jobb oldala a konstansokat tároló vektor. Kettő együttese a kibővített mátrix. Egyenletrendszer mátrixos alakja: Ax = b. Kibővített mátrix: A|b.

Egy lineáris egyenletrendszer pontosan akkor oldható meg, ha b vektor előállítható az a vektor oszlopvektorainak lineáris kombinációjaként, azaz b vektor benne van az A oszlopvektorai által felfeszített térben. Ekkor A rangja megegyezik A|b (a kibővített mátrix rangjával (lineárisan független sorvektorainak maximális száma).

Ellentmondásos lineáris egyenletrendszer: Az egyenletrendszernek pontosan akkor nincs megoldása, ha a bővített mátrixának lépcsős alakjában van ellentmondó sor:

(0 0 … 0 | c), ahol c != 0.

Határozatlan lineáris egyenletrendszer: Az egyenletrendszernek pontosan akkor van végtelen sok megoldása, ha van szabad változója. Ekkor A oszlopvektorai lineárisan függőek.

Határozott lineáris egyenletrendszer: Az egyenletrendszernek pontosan akkor van egyetlen megoldása, ha a bővített mátrixának lépcsős alakjának ugyanannyi sora van, mint ahány ismeretlen, azaz nincs szabad változó. Ekkor A oszlopvektorai lineárisan függetlenek.

Gauss-elimináció: Lineáris egyenleteket ezzel a módszerrel tudjuk megoldani. A cél a kibővített mátrix átalakítása egy olyan alakra (felső háromszög alak, azaz minden főátló alatti elem nulla) elemi sorműveletek segítségével, amelyből a megoldás könnyen leolvasható vagy kiszámítható. Az elemi sorműveletek nem változtatják meg az egyenletrendszer megoldáshalmazát. Elemi sorműveletek:

- Sorcsere: Két sor felcserélése.
- Skalárral szorzás: Egy sor minden elemének szorzása egy nem-nulla konstanssal.
- Sorhoz hozzáadás: Egy sorhoz hozzáadni egy másik sor konstansszorosát.

Az algoritmus lépései:

1. Írjuk fel az egyenletrendszer kibővített mátrixát [A|b].

2. Használjunk elemi sorműveleteket a mátrix lépcsős alakra (vagy redukált lépcsős alakra) hozásához
3. Értelmezzük az eredményt a lépcsős alak alapján: Ha olyan sor keletkezik, amelynek a bal oldalán minden elem 0, de a jobb oldalán lévő konstans nem nulla, akkor az egyenlet ellentmondás. Ekkor az egyenletrendszernek nincs megoldása. Ha nincs ilyen ellentmondó sor, és a nem-null sorok (vezéregyesek) száma megegyezik a változók számával (n), akkor az egyenletrendszernek egyetlen megoldása van. Ha nincs ellentmondó sor, és a nem-null sorok (vezéregyesek) száma kevesebb, mint a változók száma (n), akkor az egyenletrendszernek végtelen sok megoldása van. A vezéregyesekhez tartozó változókat (vezérváltozók) kifejezzük a vezéregyes nélküli oszlopokhoz tartozó változók (szabad változók) segítségével.

Visszahelyettesítés (Back Substitution): Miután a mátrix lépcsős alakban van, az utolsó nem-null sorból kiszámítjuk az abban szereplő vezérváltozó értékét, majd ezt visszahelyettesítjük az eggyel felette lévő egyenletbe, és így haladunk felfelé, amíg minden vezérváltozó értékét meg nem kapjuk. (Redukált lépcsős alak esetén ez a lépés sokkal egyszerűbb, a megoldás közvetlenül leolvasható.)

2. TURING-GÉPEK, ALGORITMIKUS ELDÖNTHETŐSÉG, ELDÖNTHETETLEN PROBLÉMÁK, REKURZÍV ÉS REKURZÍVAN FELSOROLHATÓ NYELVEK

Turing-gép: A Turing-gép (TM) egy absztrakt számítási modell. Egy gép, melynek alapja egy végtelen hosszúságú szalag, ami felett balra-jobbra mozoghat egy olvasó-író fej, és speciális állapota az elfogadás. A Turing-gép tekinthető az algoritmus precíz matematikai definíciójának.

$$T = (Q, \Sigma, \Gamma, q_0, \delta)$$

- Q: Állapotok véges halmaza.
- Σ: Bemeneti ábécé
- Γ: Szalagábécé, amely tartalmazza a bemeneti ábécét (Σ⊆Γ) és egy üres szimbólumot (#)
- q0: Kezdeti állapot
- δ: Állapotátmenet függvény. Meghatározza, hogy az aktuális állapot és a fej által olvasott szimbólum alapján mi legyen a következő állapot, mit írjon a szalagra, és merre mozdítsa a fejet (balra 'L', jobbra 'R', vagy helyben marad 'S')

Egy Turing-gép konfigurációját az aktuális állapot, a szalag tartalma és a fej pozíciója határozza meg. A gép működése konfigurációk sorozata.

Elfogadás: a Turing-gép elfogadja a w∈Σ* szót, ha létezik olyan konfiguráció sorozat, amely a kezdeti konfigurációból (q0 állapot, szalag tartalma #w, fej a w elején) indulva elfogadó konfigurációba jut. Egy elfogadó konfiguráció C=(u,q_{accept},v), ahol q_{accept} az elfogadó állapot

Elutasítás: a Turing-gép nem fogad el egy szót, ha nem elfogadó állapotban áll meg, vagy ha soha nem áll meg (végtelen ciklusba kerül).

Algoritmikus eldönthetőség: Egy probléma algoritmikusan eldönthető, ha létezik olyan algoritmus (vagy Turing-gép), amely véges idő alatt minden bemenetre eldönti, hogy az adott probléma megoldható-e.

Church-Turing tézis: Egy Turing-gép meg tud oldani bármilyen problémát, ha a problémára létezik mechanikus eljárással véges számú lépésben megoldás. Ha egy problémát Turing-géppel nem lehet megoldani, akkor nincs sem mechanikus eljárás, sem algoritmus, ami megoldja.

Rekurzív nyelvek: Egy L nyelv rekurzív, ha van olyan Turing-gép, ami eldönti L-t, azaz minden bemeneten megáll, és ha a szó L-ben van, elfogadó állapotba kerül, ha nincs, nem elfogadó állapotba kerül. Ezek az eldönthető nyelvek.

Rekurzívan felsorolható nyelvek: Egy L nyelv rekurzívan felsorolható, ha van olyan Turing-gép, ami minden w ∈ L szó bemenetre elfogadó állapotban megáll. Ha $w \notin L$, akkor vagy nem elfogadó állapotban áll meg, vagy egyáltalán nem áll meg (végtelen ideig fut). Ezek a felismerhető nyelvek. Minden rekurzív nyelv rekurzívan felsorolható, de nem minden rekurzívan felsorolható nyelv rekurzív. A rekurzív nyelveknél mindig fix, hogy megáll-e a szó elfogadásával, míg a rekurzívan felsorolható nyelveknél nem biztos.

Eldönthető: Azok a problémák, amelyekre létezik eldöntő Turing-gép, azaz a hozzá tartozó nyelv rekurzív.

Eldönthetetlen: Olyan problémák, amelyekre nincs eldöntő Turing-gép. A nyelvük rekurzívan felsorolható, de nem rekurzív. Nincs olyan algoritmus, ami minden esetben véges időben leállna és helyes igennel vagy nemmel válaszolna.

Megállási probléma: Az egyik leghíresebb eldönthetetlen probléma. A kérdés az, hogy egy adott Turing-gép megáll-e egy adott bemeneten.

3. GENERATÍV GRAMMATIKÁK ÉS NEVEZETES NYELVOSZTÁLYOK, A CHOMSKY HIERARCHIA.

Grammatika (G): formális szabályrendszerek. Szabályok halmaza egy nyelv stringjeinek generálására. G = (V, Σ, P, S)

V: nemterminálisok/változók véges halmaza: $\{S, A, B\}$

Σ: terminálisok/az ábécé szimbólumai. Generálható nyelv ábécéje: (a, b)

S∈N: kezdő szimbólum

P: produkciós/átírási/helyettesítési szabályok: $S \rightarrow SA$, $S \rightarrow \lambda$ $(törlőszabály)$, $S \rightarrow aSb$

$L(G)$: G által generált nyelvek halmaza, a generált nyelvek különböző nyelvosztályokba sorolhatók, amelyet a Chomsky hierarchia ír le. Egy grammatika által generált nyelvben Σ felett álló w szó levezethető a kezdőszimbólumból.

Adott G generatív grammatika, valamint u, v szavak, melyek terminálisokból és/vagy nemterminálisokból épülnek fel (u, v ∈ (N ∪ T)*). A v szó **közvetlenül levezethető** az u szóból G-ben, ha:

- u = v, vagy

- $u = u_1xu_2$ és $v = u_1yu_2$, ahol u_1 és u_2 terminálisokból és/vagy nemterminálisokból épülnek fel ($u_1, u_2 \in (N \cup T)^*$) és $x \rightarrow y$ része a levezetési szabályok halmazának ($x \rightarrow y \in P$)

Chomsky hierarchia: A formális grammatikák és az általuk generált nyelvek osztályozása a produkciós szabályok formája és a nyelvek felismeréséhez szükséges automaták komplexitása alapján.

	Nyelvtan, nyelv	Automata	Produkciós szabályok
0-típusú	rekurzívan felsorolható	Turing-gép	$\gamma \rightarrow \alpha$
1-típusú	környezetfüggő	korlátos nemdeterminisztikus Turing-gép	$\alpha A \beta \rightarrow \alpha \gamma \beta$
2-típusú	környezetfüggetlen	nemdeterminisztikus veremautomata	$A \rightarrow \alpha$
3-típusú	reguláris	véges automata	$A \rightarrow a$ $A \rightarrow aB$
a = terminális $a \in T$ *A, B = nemterminálisok* $A, B \in N$ *α, β, γ = terminálisok és/vagy nemterminálisok sztringje* $\alpha, \beta, \gamma \in (N \cup T)^*$			

A hierarchia kapcsolata: Az egyes nyelvosztályok szigorúan tartalmazzák az alattuk lévőt: Reguláris $\subset$ Környezetfüggetlen $\subset$ Környezetfüggő $\subseteq$ Rekurzívan Felsorolható. A környezetfüggő nyelvek lehetnek egyenlőek is a rekurzív nyelvekkel. A rekurzív nyelvek a rekurzívan felsorolható nyelvek egy szigorú részhalmazát alkotják.

8. Statisztikai minta és becslések, átlag és szórás. Konfidenciaintervallumok. Az u-próba. Az informatikai biztonság fogalma, legfontosabb biztonsági célok. Fizikai védelem, kártékony programok, osztályozásuk terjedési módjuk és büntető rutinjuk szerint. Algoritmikus védelem eszközei: titkosítás, digitális aláírás, hash függvények. Az AES és RSA algoritmusok.

4. Statisztikai minta és becslések, átlag és szórás

Populáció: A vizsgálat tárgyát képező összes elem összessége (pl. egy ország összes lakosa, egy gyártósor összes terméke). Általában túl nagy vagy elérhetetlen ahhoz, hogy teljes egészében vizsgáljuk.

Statisztikai minta: A populáció egy kisebb, kiválasztott részhalmaza, amelyet ténylegesen megvizsgálunk. A minta kiválasztása gyakran véletlenszerűen történik, hogy a lehető legjobban reprezentálja a populációt. Valamely valószínűségi változóra vonatkozó véges számú független kísérlet vagy megfigyelés eredménye: véges sok azonos eloszlású valószínűségi változó. Tekintsük a valószínűségi változót, ekkor a X-re vonatkozó n elemű minta X1, X2,....., X n. Az n számú kísérlet elvégzése során minden mintaelem egy-egy konkrét számértéket vesz fel: X1 = x1, X2 = x2 ,..., Xn = xn. A statisztikai minta reprezentatív, a mintaelemek eloszlása megegyezik a vizsgált valószínűségi változó eloszlásával, hiszen mindegyik kísérletnél magát a valószínűségi változót figyeljük meg. A statisztikai minta elemei független valószínűségi változók, mivel a kísérleteket egymástól függetlenül végezzük. A mintaelemekből tapasztalati jellemzőket, ún. **statisztikát** konstruálunk.

Becslés: A minta adatainak felhasználásával következtetünk a populáció ismeretlen jellemzőire (paramétereire): átlag és szórás.

Torzítatlan becslés: Egy becslés torzítatlan, ha a becslő várható értéke megegyezik a becsülni kívánt paraméter valódi értékével. Ez azt jelenti, hogy sok mérés vagy mintavétel átlagában a becslés nem tér el szisztematikusan a valódi értéktől.

Konzisztens becslés: Egy becslés konzisztens, ha a mintaelemszám növelésével a becslés egyre jobban közelíti a valódi paraméter értékét. Nagy elemszám esetén a becslés valószínűséggel tetszőlegesen közel kerül a keresett értékhez.

Átlag (mintaközép): Összes elem összege elosztva az elemek számával. Populáció átlaga ismeretlen, de a minta átlaga jó becslést ad rá.

Szórás: Átlagtól való átlagos eltérés. Populáció szórása ismeretlen, de a minta átlaga jó becslést ad rá. Ha egy minta elég nagy, akkor a mintaátlag eloszlása közelítőleg normális eloszlású lesz.

$$\sigma = \sqrt{\frac{(x_1 - \bar{x})^2 + \cdots + (x_n - \bar{x})^2}{n}}$$

5. Konfidenciaintervallumok

Konfidenciaintervallumok: Becslések bizonytalanságát méri. Minél szélesebb a konfidencia intervallum, annál kevésbé bízhatunk a becslésekben. Egy minta adatai alapján számított értéktartomány, amely a populáció ismeretlen paraméterére (pl. μ (populáció átlag)) ad becslést. Például: 0.05-ös szignifikanciaszint azt jelenti, hogy a konfidenciaintervallum 95% eséllyel tartalmazza az ismeretlen paramétert.

Konfidenciaszint: Egy százalékos érték, mely megadja, hogy ha sok mintát vennénk, és minden mintából számolnánk egy konfidenciaintervallumot, akkor az intervallumok hány százaléka tartalmazná a populáció valódi paraméterét. $1 - \alpha$ valószínűség adja meg hogy a sokasági átlagot milyen valószínűséggel esik a konfidenciaintervallumba.

6. Az u-próba

Az U-próba egy statisztikai próba, melyet statisztikai hipotézisekre alkalmazunk. Célja annak statisztikai vizsgálata, hogy egy minta átlaga ($\overline{x}$) szignifikánsan eltér-e egy ismert populáció átlagától (q_0), VAGY két minta átlaga szignifikánsan eltér-e egymástól. Feltétele: Az u-próba használatának legfontosabb feltétele, hogy a populáció szórása ismert legyen. Továbbá feltételezi az adatok normális eloszlását vagy elegendően nagy mintaméretet. Ha a populáció szórása (σ) ismeretlen, helyette t-próbát kell alkalmazni.

Működése:

1. Felállítunk egy nullhipotézist (H0) (pl. a minta átlaga megegyezik az ismert populáció átlagával, $m = m_0$).
2. Felállítunk egy alternatív/ellen hipotézist (H1) (pl. a minta átlaga nem egyezik meg, $m \neq m_0$, vagy kisebb/nagyobb).
3. Kiszámítjuk a u-statisztika értékét: $u = \frac{\overline{x} - m_0}{\sigma}\sqrt{n}$, ahol $\overline{x}$ a minta átlaga, m_0 az elére adott érték, σ minta szórása, n pedig a minta elemszáma
4. Összehasonlítjuk a kiszámított Z-statisztikát egy kritikus Z-értékkel, tehát Z bele esik $[\frac{\alpha}{2}; -\frac{\alpha}{2}]$ intervallumba
5. Ha a Z-statisztika meghaladja a kritikus értéket, elutasítjuk a nullhipotézist, és azt mondjuk, a különbség statisztikailag szignifikáns. Ellenkező esetben nem utasítjuk el a nullhipotézist.

7. Az informatikai biztonság fogalma, legfontosabb biztonsági célok

Az informatikai biztonság fogalma: Az informatikai biztonság az informatikai rendszer olyan kielégítő állapota, amely az informatikai rendszerben kezelt **adatok bizalmassága**, **sérthetetlensége** és **rendelkezése állása**, és a **rendszerelemek sérthetetlensége** és **rendelkezése állása** szempontjából zárt, teljeskörű, folytonos és a kockázatokkal arányos.

- **zárt**: összes releváns támadást figyelembe veszi
- **teljeskörű**: a védelmi intézkedések a rendszer egészére kiterjednek (összes rendszerelemre)
- folytonos: a védelem a dinamikusan változó körülmények és viszonyok ellenére is **folyamatosan** fennáll
- **kockázatokkal arányos**: a védelem költségei arányosak a potenciális kárérték arányával

Bizalmasság: Titkos/személyes információkat jogosulatlan személyek nem ismerhetik meg. A bizalmasságot az adatok tárolásánál, feldolgozásánál és továbbításánál garantálni kell.

Sérthetetlenség:

- Adatintegritás: Annak garantálása, hogy az adat nem módosult tárolás, feldolgozás és továbbítás során.

- Rendszer sértetlensége: A rendszer működése az elvártnak megfelelő, jogosulatlan módosításoktól mentes.

Rendelkezésre állás: A szolgáltatás az arra jogosultak számára a szükséges időben és a szükséges időtartamra elérhető és használható

Nyomon követhetőség: Egy tevékenység visszakövethető legyen a tulajdonos entitáshoz, amely a tevékenységet végezte. Célja: ellenőrzés, letagadhatatlanság biztosítása, jogtalan behatolások detektálása.

Garancia, biztosíték: Bizalom abban, hogy az összes többi cél teljesül, a biztonsági alrendszer megfelelően működik.

Hitelesség: Biztosítja a digitális információk eredetének és sértetlenségének igazolhatóságát, megteremtve ezzel a bizalmat a digitális interakciókban.

8. Fizikai védelem, kártékony programok, osztályozásuk terjedési módjuk és büntető rutinjuk szerint

Fizikai védelem: Fizikai erőforrások védelme, amely szerepet játszik az adatok tárolásában, feldolgozásában és továbbításában. Fizikai infrastruktúra elemei, melyeket védeni kell:

- Informatikai rendszer hardver elemei: adatfeldolgozó és tároló egységes, adatátviteli és hálózati elemek, offline tároló eszközök, dokumentációk
- Épületek: ahol az informatikai rendszer elemei megtalálhatóak
- Kiszolgáló rendszerek: elektromos vezetékek, kommunikációs hálózatok, víz/gázvezetékek
- Személyzet: informatikai rendszer működtetői/fenntartói

Kártékony program: Olyan program, mely valamely rendszerbe beépülve az áldozat adatainak, szoftvereinek bizalmasságát, sérthetetlenségét, rendelkezésre állását veszélyezteti vagy erkölcsi károkat okoz: pénzt, időt, embert köt le.

Osztályozásuk terjedési mód szerint:

- **Vírusok**: fertőzött fájl futtatásával terjednek. Gazdaprogrammal rendelkeznek, mely lehet végrehajtható program vagy dokumentum. Más alkalmazásokat, rendszerfájlokat, programkódokat is megfertőzhet, célja a károkozás. Részei a fertőző mechanizmus (a vírus terjedéséért felelő), az indíték/logikai bomba (megadja a feltételt, amely aktiválja a büntető rutint) és a büntető rutin (kárt okozó tevékenység). Csoportosításuk célpont szerint:
    - **boot vírusok**: bootszektorba ágyazódnak, OS előtt aktiválódik, egész tárolót megfertőzi
    - **alkalmazás vírusok**: Fertőzött állomány forráskódjába írják a saját kódjukat. Ha elindul, megfertőz minden futó alkalmazást
        - felülíró: fertőzött alkalmazás adatot veszít, az eredeti nem állítható vissza
        - hozzáfűző: fertőzött alkalmazás kódjának végére íródik, viszont így is a program előtt fut le

    - **macrovírusok**: makrókat támogató dokumentumszerkeztő programokat támad meg (pl.: MS Office), megfertőz minden később megnyitott dokumentumot is. Jellemző fajtája a levelező vírus

  Csoportosításuk rejtőzési stratégia szerint.
    - **titkosított vírus**: titkosítással rejti el magát, véletlen generált titkosító kulcs segítségével. Bitminta segítségével sem ismerhető fel
    - **lopakodó vírus**: megkerüli az OS-t és antivírus szoftvereket, a memóriában marad. Módosíthatja az OS jellemzőit
    - **polimorf vírus**: bizonyos ciklusonként változtatja a megjelenési formáját, bitminta alapú felismerést megnehezíti. A kulcsgenerálásért és mutációért a mutációs motor felelős
    - **metamorfózisra képes vírus**: teljesen felül írják magukat (nem csak megjelenést hanem viselkedést is)
- **Férgek**: szoftver sérülékenységét kiaknázva sokszorosítja magát. Kliens-szerver oldali sebezhetőségek kihasználásával terjed. Terjedhet hálózaton keresztül, megszokott médiákon (USB, CD), email-ben (email férgek). A férgek önsokszorosítóak, nincs szükség gazdaprogramra. A terjedéshez hozzáféréseket keres host táblákban, címtáblákban, cserélhető médiákat keres, majd hozzáférési mechanizmus segítségével lemásolja saját magát. A modern férgek több platformot is fertőznek (főleg UNIX variánsok), többféle módon tud bejutni a rendszerbe (multi-kihasználás), optimalizálják a terjedést, hogy minél kevesebb idő alatt minél több sebezhető gépet próbáljanak elérni, valamint polimorfok, metamorfok és többféle büntetőrutint is szállíthatnak. A mobil férgek Bluethooth-on vagy MMS-en keresztül terjednek, lassítják a telefont. valamint nagy költségű SMS-eket küldenek ki és adatokat törölnek.
- **Pszichológiai támadás** (social engineering): trójai programok, spam email-ek. Célja az, hogy felhasználót rávegye egy kívánt interakcióra, tehát aktív részvétel szükséges hozzá
    - **spam emailek**: kb email forgalom 50%-a, viszont manapság detektálhatóak. Botnet hálózat küldi őket, kártékony programot hordoznak, vagy szimplán reklámot tartalmaznak. Adathalászatra is használják.
    - **trójai**: hasznos programnak / felületnek adja ki magát. Nem sokszorozódik. Backdoor-t tartalmaz, ezen keresztül jutnak be a hackerek. Jellemzően kártékony programokat, kémprogramokat telepít o modellei:
        - megtartja a funkcionalitást, de büntető rutint futtat
        - módosítja a működését, hogy végrehajtsa a büntető rutint (pl.: jelszavakat gyűjt)
        - lecseréli az eredeti program funkcióját

Osztályozásuk büntető rutinjuk szerint: Tevékenységek, melyek végrehajtódnak a fertőzött rendszeren, általában rendelkeznek vele.

- **Rendszer károsítása**: célja az adatmegsemmisítés, fizikai károkozás
    - **ransomware**: lezárja a felhasználó fájljait, blokkolja az adathozzáférést, váltságdíjat követelnek a visszafordításért
        - **fájltitkosító ransomwate**: trójaiak, email mellékletek, programok sebezhetőségét használják ki, titkosítja a személyes fájlokat

        - **nem titkosító ransomware**: teljes rendszert lezárja, illegális fájlokat keres, és fenyegetőzik egy hatóság nevében, hogy fizessen, különben pl börtönbe kerülhet a felhasználó
        - **böngészőlezáró ransomware**: blokkolja a böngészőt és figyelmeztető üzeneteket jelenít meg. Illegális cselekvéseket sorol fel
    - **fizikai sérülés**: pl BIOS kódot írja újra: Csernobil vírus
    - általában **logikai bomba**: feltétel teljesülésével „robbanásszerűen" akcióba lép az adatok/fájlok törlése, módosítása. Indíték: a feltétel amely elindítja a folyamatot
- **Támadó ügynökök (Botnet):** a támadó irányítás alá veszi az erőforrásokat: zombie gépeket csinál. Célja a koordinált működés botnet hálózatokban. Ez integritást és rendelkezésre állást támad. Zombie-k használata:
    - **DDoS támadás**: lényege a szolgáltatás megbénítása a hálózat leterhelésével
    - Spamek küldése
    - **forgalomfigyelés**: nyílt üzenetek megfigyelése, érzékeny infók megszerzése
    - **billentyűzetfigyelés:** érzékeny infók megszerzése
    - **kártékony programok terjesztése**: FTP-n vagy HTTP-n keresztül
    - **reklámok**: böngésző átirányítása, hamis website-okra történő átirányítás
- **Információ lopás** (kémprogram, adathalászat): a fertőzött gépen tárolt adatokat gyűjti össze. Az információ lopás fajtái:
    - **credential** (hitelesítő) **adatok ellopása**, **billentyűzet lehallgatás**, **kémprogramok**:
        - szűrő dolgozza fel a bejött adatokat (kulcsszó alapján)
        - grafikus megtévesztés, appletek
        - böngésző előzmények megfigyelése, weboldalak eltérítése
    - **adathalászat és személyazonosság ellopása**
        - szintén belépési adatok megszerzése a cél
        - megbízható felet személyesít meg
    - **felderítés, kémkedés**
        - meghatározott információkra kíváncsi
        - célzott adathalászattal felderítést végeznek.
    - Célzott adathalászat: olyan cég nevében küldenek levelet, melyben a címzett megbízik. Célpontja megválasztott, így a támadás testreszabott.
- **Lopakodás** (gyökércsomagok/rootkitek, backdoors): elrejtik magukat, integritást céloznak
    - **backdoor**: titkos belépési pont, amely hozzáférést biztosít hitelesítés nélkül. Legálisan használhatóak pl teszteléskor
    - **rootkitek**: programcsomag, mely installálás után fedett hozzáférést biztosít és tart fent a már fertőzött géphez. Ez admin jogosultságot biztosít, OS funkciókhoz ad hozzáférést.
        - Tárolása:
            - lehet perzisztensen tárolt (registerben)
            - lehet memória alapú, nehezebb detektálni, de újraindításkor elveszlik
        - Módja:

- **User mód**: felhasználói szinten működik, API hívások válaszait módosítja
- **Kernel mód**: úgy rejti el magát, hogy módosítja a kernelt. Technikái:
    - **rendszerhívási tábla módosítása**: hívások címét módosítja, hogy a rootkit rutinja fusson
    - **rendszerhívási tábla átirányítása**: másik kernel beli táblára irányítja a hivatkozást

9. Algoritmikus védelem eszközei: titkosítás, digitális aláírás, hash függvények

Szimmetrikus titkosítási séma: Titkosító és visszafejtő kulcs megegyezik, vagy a titkosító kulcsból polinomiális időn belül kiszámítható a másik kulcs.

$SE = (Key, Enc, Dec)$ egy szimmetrikus titkosítási séma, ha

- Key: kulcsgeneráló algoritmus, mely egy k biztonsági paraméterhez (kulcs méretére utal) megad egy $K \in \mathcal{K}$ titkos kulcsot
- Enc: titkosító algoritmus mely $\forall m \in \mathcal{P}$ nyílt üzenethez és $\forall K \in \mathcal{K}$ titkos kulcshoz generál egy $c \in \mathcal{C}$ titkos üzenetet. $\boldsymbol{c = Enc_K(m)}$
- Dec: determinisztikus visszafejtő algoritmus mely $c \in \mathcal{C}$ titkosított üzenethez és $\forall K \in \mathcal{K}$ titkos kulcshoz megad egy $m \in \mathcal{P}$ nyílt üzenetet. $\boldsymbol{m = Dec_K(c)}$

$SE = (Key, Enc, Dec)$ korrekt, ha: $\forall m \in \mathcal{P}$ és $\forall K \in \mathcal{K}$ esetén $Dec_K(Enc_K(m)) = m$ (tehát eredeti üzenetet kapjuk vissza)

Aszimmetrikus titkosítási séma: Titkosító és visszafejtő kulcs különbözik annyira, hogy a titkosító kulcsból nehezen (nincs rá polinomiális idejű algoritmus) kiszámítható a másik kulcs.

$AE = (Key, Enc, Dec)$ egy szimmetrikus titkosítási séma, ha

- Key: kulcsgeneráló algoritmus, mely egy k biztonsági paraméterhez (kulcs méretére utal) megad egy $(PK, SK) \in \mathcal{K}$ nyilvános és titkos kulcsból álló párt
- Enc: titkosító algoritmus mely $\forall m \in \mathcal{P}$ nyílt üzenethez és PK nyilvános kulcshoz generál egy $c \in \mathcal{C}$ titkos üzenetet $c = Enc_{PK}(m)$
- Dec: visszafejtő algoritmus mely $c \in \mathcal{C}$ titkosított üzenethez és SK titkos kulcshoz megad egy $m \in \mathcal{P}$ nyílt üzenetet $m = Dec_{SK}(c)$

$AE = (Key, Enc, Dec)$ korrekt, ha: $\forall m \in \mathcal{P}$ és $\forall K \in \mathcal{K}$ esetén $Dec_{SK}(Enc_{PK}(m)) = m$ (tehát eredeti üzenetet kapjuk vissza)

Digitális aláírás: Hitelességet és letagathatatlanságot biztosít. Digitális aláírási séma $DS = (Key, Sign, Ver)$ hármas, ahol

- Key: kulcsgeneráló algoritmus, mely egy k biztonsági paraméterhez (kulcs méretére utal) megad egy $(PK, SK) \in \mathcal{K}$ nyilvános és titkos kulcsból álló párt
- $Sign$: aláíró algoritmus az SK titkos kulcshoz és $m \in \{0,1\} *$ üzenetre generál egy $s = Sign_{SK}(m)$ aláírást.

- Ver_{PK}: ellenőrző algoritmus a PK nyilvános kulcsra, az m üzenetre és az s aláírásra $IGAZ$ vagy $HAMIS$ értéket ad vissza
    - $IGAZ$: érvényes aláírás
    - $HAMIS$. érvénytelen aláírás

Hash függvény: tetszőleges véges hosszú üzenethez n hosszú üzenetet rendelünk. Az n attól függ, hogy milyen hashelő algoritmus használunk. Az adatintegritást védi, mivel küldés előtt az üzenetet hasheljük, a hash-t is elküldjük, az adat megérkezése után pedig újra hasheljük és a két lenyomatot összehasonlítjuk. **Lenyomatnak** a hashelés eredményeként kapott hash értéket hívjuk. **Lavinahatás**: egy bit változása jelentős hash-beli változáshoz vezet. A hash függvény:

- **Egyirányú**: egy y hash érték (lenyomat) ismeretében nehéz feladat olyan x' üzenetet (ősképet) kiszámítani, amelyre h=H(x')
- **Őskép ellenálló**: adott y lenyomathoz nehéz olyan x üzenetet találni, aminek lenyomata y, azaz H(x) = y
- **Második őskép ellenálló**: adott x üzenethez nehéz olyan x'-et találni, ahol x és x' üzenetek nem egyeznek meg, viszont lenyomatuk igen, azaz x != x' és H(x) = H(x')
- **Ütközésmentes**: nehéz olyan x, x' üzenetpárt megadni, amelyeknek azonos a lenyomata, azaz H(x) = H(x')

10. Az AES és RSA algoritmusok

AES (Advanced Encryption Standard): szimmetrikus titkosítási séma. Az üzenetek fix hosszúak (128 bit), hosszabb üzeneteket ekkorára osztja, és ezeket ismétlődő transzformációs körök segítségével a kulcs alapján, amely 128/192/256 bites. Gyors és biztonságosnak számító algoritmus .

$SE = (Key, Enc, Dec)$

$\mathcal{P} = \{0,1\}^{128}$, $\mathcal{C} = \{0,1\}^{128}$

$\mathcal{K} = \{0,1\}^{k}$, $k \in \{128,192,256\}$

Key: $K \in \mathcal{K}$, $véletlenül\ választott$

Körök száma különböző kulcshosszok esetén: $k = 128 \rightarrow 10$, $k = 192 \rightarrow 12$, $k = 256 \rightarrow 14$

AES algoritmus működése:

1. Adat felosztása blokkokra: A titkosítani kívánt eredeti adatot (nyílt szöveg) 128 bites blokkokra osztják. Ha az utolsó blokk nem ér el 128 bitet, kiegészítik (padding). Ezeket a blokkokat 4x4-es (bájt)mátrixként kezeli.
2. A felhasználó által megadott 128, 192 vagy 256 bites titkos kulcsból több részkulcsot állít elő. Ezek közül minden titkosítási kör egy saját körkulcsot használ.
3. A titkosítás kezdetén a plaintext blokk XOR művelet segítségével összekeveredik az első körkulccsal. Ezt a lépést AddRoundKey műveletnek nevezzük.
4. Ismétlődő körök (Rounds): Ezt követően az algoritmus több körön (meneten) megy keresztül. A körök száma attól függ, milyen hosszú a titkos kulcs. Minden kör az alábbi műveletekből áll:

- **SubBytes:** a blokk minden bájtját egy előre definiált helyettesítési táblázat alapján lecseréli az algoritmus. Ez egy nemlineáris helyettesítési folyamat, amely növeli a titkosítás biztonságát.
- **ShiftRows:** az állapotmátrix sorai különböző mértékben ciklikusan eltolódnak balra. Az első sor nem változik, a második sor egy bájttal, a harmadik sor két bájttal, a negyedik sor három bájttal tolódik el. Ez az adatok összekeverését biztosítja.
- **MixColumns:** az állapot oszlopait, mint vektorokat megszorozza egy mátrixszal. Ez tovább növeli az adatok diffúzióját és összekeveri az oszlopok elemeit.
- **AddRoundKey:** az aktuális állapot XOR művelet segítségével összekeveredik a körkulccsal.

5. Utolsó kör (Final Round): Az utolsó kör hasonló az előzőekhez, de kihagyja a MixColumns lépést az átalakítások sorozatából.
6. Az utolsó kör végrehajtása után az algoritmus előállítja a titkosított adatblokkot, vagyis a ciphertextet.
7. A visszafejtés folyamata a titkosítás fordítottja. A dekódolás során az algoritmus az egyes műveletek inverzét hajtja végre fordított sorrendben, ugyanannak a titkos kulcsnak a felhasználásával, így visszaállítható az eredeti plaintext.

RSA: aszimmetrikus titkosító és digitális aláíró algoritmus

RSA titkosítási séma:

$AE = (Key, Enc, Dec)$

Key (kulcsgenerálás):

1. Véletlenszerűen válasszunk két nagy, egymástól különböző p és q prímet.
2. Ezekre kiszámítjuk az RSA modulust: $n = p * q$
3. Kiszámítjuk az Euler-féle φ-függvényt (fí-függvény): $\varphi(n) = (p-1) * (q-1)$
4. Választunk egy véletlen e egészt (a titkosító tényezőt), ahol $1 < e < \varphi(n)$ és $(e, \varphi(n)) = 1$, azaz e és $\varphi(n)$ relatív prímek (egyetlen közös osztójuk 1)
5. Kiszámítjuk d-t (e multiplikatív inverze modulo φ(n)), ahol 1 < d < $\varphi(n)$ ahol $e * d \equiv 1 \ (mod \ \varphi(n))$
6. Ekkor: $PK = (n, e)$, $SK = (d, n)$ és titkosparaméterek = $\phi(n), p, q$

$Enc_{pk}(m) = m^e \ (mod \ n)$ - titkosítás

$Dec_{sk}(c) = c^e \ (mod \ n)$ – visszafejtés

Titkosítási séma során használt algoritmusok:

- Prímtesztek p, q meghatározásához **Miller-Rabin prímteszt**: véletlen a szám alapján (bázis) eldönti, hogy a valószínűleg prím-e
- **Euklideszi algoritmus** titkosító e tényező kiválasztásához. Ez a legnagyobb közös osztó meghatározására szolgáló algoritmus, melyet az alábbi két tényezőre alkalmazunk: $e, \phi(n)$. Ez biztosítja, hogy a két tényező relatív prím.

- **Kibővített Euklideszi algoritmus** privát d paraméter meghatározására. Megfelelő Diophantikus egyenlet megoldását jelenti, ha a két együttható relatív prím (ez teljesül az RSA kikötése miatt): $d * e + k * \phi(n) = 1$
- **Gyorshatványozás** alkalmazása a titkosításhoz. Nagy számok hatványra emelése nem lehetséges, ezért lépésenként elvégezzük a moduláris maradékképzést, így megkapjuk $c^d\ (mod\ n)$ értékét
- **Kínai maradéktétel** alkalmazása a visszafejtéshez. Kimondja, hogy $x_1 \equiv C^d\ (mod\ p)$ és $x_2 \equiv C^d\ (mod\ q)$ osztható, melyet gyorsabb kiszámolni

RSA aláírási séma:

$AE = (Key, Sign, Ver)$

Key (kulcsgenerálás):

1. Véletlenszerűen válasszunk két nagy, egymástól különböző *p* és *q* prímet.
2. Ezekre kiszámítjuk az RSA modulust: $n = p * q$
3. Kiszámítjuk az Euler-féle φ-függvényt (fí-függvény): $\varphi(n) = (p-1) * (q-1)$
4. Választunk egy véletlen *e* egészt, ahol $1 < e < \varphi(n)$ és $(e, \varphi(n)) = 1$, azaz e és $\varphi(n)$ relatív prímek
5. Kiszámítjuk d-t (e multiplikatív inverze modulo φ(n)), ahol $1 < d < \varphi(n)$ ahol $e * d \equiv 1\ (mod\ \varphi(n))$
6. Ekkor: $PK = (n, e)$, $SK = d$ és titkosparaméterek = $\phi(n), p, q$

$Sign_{sk}(m) = m^d\ (mod\ n)$ – aláírás

$Ver_{pk}(m, s) = TRUE, s^e \equiv m\ (mod\ n)$
$FALSE$ egyébként

RSA biztonságosság oka: Nehéz kiszámolni a titkos paramétereket bármely publikus információ alapján. A prímek szorzata könnyen meghatározható, de a szorzat alapján nehéz meghatározni a két prímet, egyirányú függvénynek bizonyul. Nehéz a titkosított üzenet alapján visszafejteni a hatványalapot (m-et). Gyorsan meghatározható a hatványérték modulusa n-nel, viszont egy értékről nehezen megállapítható, hogy mit emeltünk hatványra