Programtervező informatikus BSc (2021) záróvizsga tételek

Matematikai és számítástudományi ismeretek:

1.1. Diszkrét és folytonos valószínűségi eloszlás fogalma. Nevezetes eloszlások: binomiális, Poisson, egyenletes, exponenciális, normális.

Mateking epizódok:

Az eloszlásfüggvény

A sűrűségfüggvény

A binomiális eloszlás

A Poisson eloszlás

Egyenletes, exponenciális, normális eloszlások

Diszkrét valószínűségi eloszlás:
Olyan valószínűségi eloszlás, melyben a valószínűségi változók csak véges számú értéket vehetnek fel, azaz diszkrétet.

A valószínűségek összege **mindig 1**.

Valószínűségi mérő – Véges

Ω: Mintatér → Összes kimenetel

A: Ω bizonyos részhalmaza, a megtörtént események

Eseményalgebra: Események családja, Ω részhalmazainak egy σ algebrája

P: Valószínűség

Véges eloszlás esetén megfeleltetés van diszkrét eloszlások és megszámlálható valószínűségi mezők között.

Diszkrét valószínűségi változó:

Diszkrétnek nevezzük azokat a valószínűségi változókat, amik megszámlálhatóan sok értéket vesznek fel. Ez azt jelenti, hogy vagy véges sokat, vagy végtelent, de úgy, hogy fel tudjuk sorolni az értékeit.

Eloszlásfüggvény:

Az X valószínűségi változó eloszlásfüggvénye:

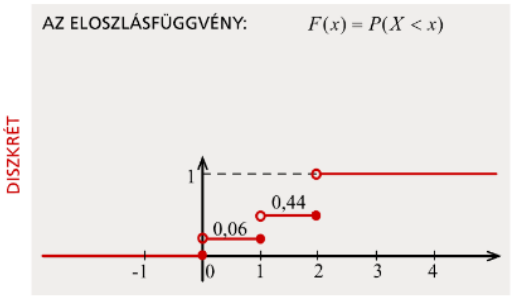
$$F(x) = P(X < x)$$

Ha az X valószínűségi változó diszkrét és értékei X = a, X = b, X = c, akkor az eloszlásfüggvény mindig egy lépcsőzetes függvény, ami minden számnál pontosan akkorát ugrik, mint az adott szám valószínűsége, amíg el nem érjük az 1-et.

$$F(x) = \{0,\ ha\ x \leq a\ P(X = a),\ ha\ a < x \leq b\ P(X = a) + P(X = b),\ ha\ b < x \leq c\ \ \ldots\ 1$$

ELOSZLÁS

találatok száma	valószínűség
0	0,06
1	0,38
2	0,56



Ez egy valós, monoton növekvő, balról folytonos függvény, ami minden tetszőleges változónak van.

Folytonos valószínűségi eloszlás:

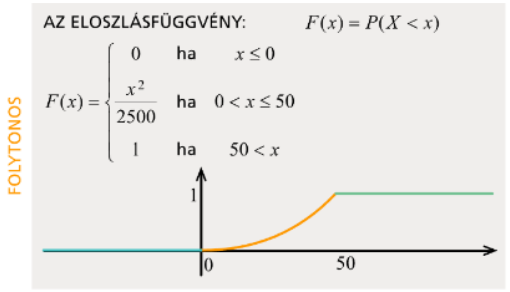
Olyan valószínűségi eloszlás, melyben a valószínűségi változók végtelen számú értéket vehetnek fel, vagyis megszámlálhatatlant.

Folytonos valószínűségi változó:

Folytonosnak nevezzük azokat a valószínűségi változókat, amik folytonos mennyiségeket mérnek, ilyen például az idő, a távolság. Ebben az esetben az eloszlás függvény is mindig folytonos függvény lesz.

Eloszlásfüggvény:

Ha az X valószínűségi változó folytonos, akkor az a és 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 végig egyet vesz fel. Vagyis ahol az X valószínűségi változó működik, ott a függvény életre kel, előtte és utána pedig hibernált állapotban van.



Sűrűségfüggvény:

Sűrűségfüggvénye csak folytonos valószínűségi változóknak van.

A sűrűségfüggvény úgy működik, hogy a valószínűségeket a görbe alatti területek adják meg. Az eloszlásfüggvény jele $F(x)$ volt, a sűrűségfüggvény jele $f(x)$. Az $a < X < b$ valószínűség éppen a görbe alatti terület a-tól b-ig.

$$P(a < X < b) = \int_a^b f(x)\,dx$$

Ha az $X < a$ valószínűséget szeretnénk kiszámolni:

$$P(X < a) = \int_{-\infty}^{a} f(x)\,dx$$

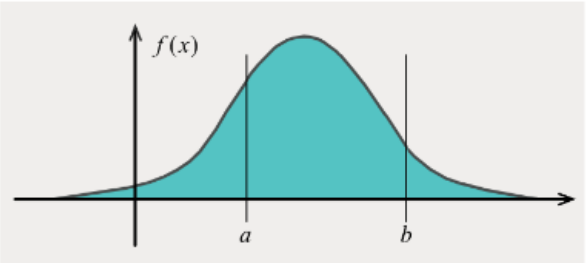
Ha a $b < X$ valószínűséget:

$$P(b < X) = \int_{b}^{+\infty} f(x)\,dx$$

A SŰRŰSÉGFÜGGVÉNY TULAJDONSÁGAI:

$$\int_{-\infty}^{+\infty} f(x)dx = 1$$

nem negatív



Ha ezt a három területet összeadjuk, akkor éppen a teljes görbe alatti területet kapjuk, ami a 100%-ot jelenti, így hát ez a terület éppen 1.

Binomiális eloszlás:

Egymástól teljesen független eseményekből álló diszkrét valószínűségi eloszlás.

Ezt a képletet hívjuk binomiális eloszlásnak:

$P(X = k) = \left(\frac{n}{k}\right) \times p^k \times (1 - p)^{n-k}$, ahol n a kísérletek száma, k a sikeres kísérletek száma, p pedig a sikeres kísérlet valószínűsége.

Visszatevéses mintavétel:

Visszatevéses mintavételről beszélünk, ha egy p valószínűségű elem többszöri kihúzásának esélyét vizsgáljuk úgy, hogy ha kihúzunk egy ilyen elemet, akkor ezt követően azt visszarakjuk.

Például ha azt vizsgáljuk, hogy egy kosárban van 8 piros és 5 kék golyó, és mennyi a valószínűsége, hogy háromszor húzva két piros és egy kék golyót húznánk úgy, hogy a kihúzott golyókat mindig visszatesszük, akkor az egy visszatevéses mintavétel.

A visszatevéses mintavételhez kapcsolódó eloszlás a **binomiális eloszlás**.

Várható érték: $E(X) = n \times p$

Szórás: $D(X) = \sqrt{n \times p \times (1 - p)}$

Hipergeometriai eloszlás:

A hipergeometriai eloszlás a visszatevés nélküli mintavételhez kapcsolódó diszkrét valószínűségi eloszlás, aminek képlete:

$$P(X = k) = \frac{\left(\frac{K}{k}\right) \times \left(\frac{N-K}{n-k}\right)}{\left(\frac{N}{n}\right)}$$

Visszatevés nélküli mintavétel

A visszatevés nélküli mintavétel tipikus példája, hogy van egy doboz, benne N darab elem. Közülük K darab valamilyen tulajdonságú, az egyszerűség kedvéért hívjuk selejtesnek. Mondjuk sárga vagy szép vagy ronda. Kihúzunk n darab elemet, és ez a képlet meg fogja nekünk mondani, hogy mekkora az esélye, hogy közülük k darab a vizsgált tulajdonságú.

Várható érték: $E(X) = n \times \frac{K}{N}$

Szórás: $D(X) = \sqrt{n \times \frac{K}{N} \times \left(1 - \frac{K}{N}\right) \times \frac{N-n}{N-1}}$

Poisson eloszlás:

A Poisson eloszlás egy diszkrét eloszlás, ahol előre ismert a várható érték, és a valószínűségi változó nem korlátos, vagyis tetszőleges bármilyen nagy érték is lehet.

Például valamilyen anyagban a hibák száma, vagy egy adott idő alatt bekövetkező események száma. A Poisson eloszlásos feladatokban általában valamilyen százalék vagy arány vagy várható érték vagy átlag vagy valószínűség van megadva. Mondjuk egy könyvben az oldalak 80%-ában nincs hiba, vagy az 20 méter hosszú ruhaszövetek harmadában nincs hiba, vagy egy üzletben óránként várhatóan 13 vevő érkezik, vagy egy bankban percenként átlag 24 tranzakció történik, vagy 0,2 a valószínűsége, hogy 10 perc alatt nem érkezik segélyhívás. Ezek mind Poisson eloszlások, ahol az *X* nem korlátos diszkrét valószínűségi változó.

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}$$

Várható érték: $E(X) = \lambda$

Szórás: $D(X) = \sqrt{\lambda}$

Egyenletes eloszlás:

Az egyenletes eloszlás egy folytonos eloszlás, amiben a megadott intervallumon belül a részintervallumba esés valószínűsége egyenlő a részintervallum és egész intervallum arányával.

Eloszlásfüggvénye:

$$F(x) = \{0,\ ha\ x \leq a\ \frac{x-a}{b-a},\ ha\ a < x \leq b\ 1,\ ha\ b < x$$

Sűrűségfüggvénye:

$$\{\frac{1}{b-a},\ ha\ a < x \leq b\ 0,\ különben$$

Várható érték: $E(X) = \frac{a+b}{2}$

Szórás: $D(X) = \frac{b-a}{\sqrt{12}}$

Exponenciális eloszlás:

Az exponenciális eloszlás egy folytonos eloszlás egymástól független események közötti időtartamok modellezésére, ahol az átlagos bekövetkezési gyakoriság állandó.

Eloszlásfüggvénye:

$$F(x) = \{0,\ ha\ x \leq 0\ 1 - e^{-\lambda x},\ ha\ 0 < \#x$$

Sűrűségfüggvénye:

$$f(x) = \{0,\ ha\ x \leq 0\ \lambda e^{-\lambda x},\ ha\ 0 < \#x$$

Várható érték: $E(X) = \frac{1}{\lambda}$

Szórás: $D(X) = \frac{1}{\lambda}$

Normális (Gauss) eloszlás:

A normális eloszlás egy folytonos eloszlás, amit másnéven haranggörbe eloszlásnak is neveznek, mert szimmetrikus.

Sűrűségfüggvénye (Nem lehet integrálni):

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Várható érték: $E(X) = \mu$

Szórás: $D(X) = \sigma$

Standard normális eloszlás (Normális eloszlás [0, 1] intervallumon):

Statisztika egyik alapköve.

Eloszlásfüggvénye (Táblázat):

$$F(X) = \Phi\left(\frac{x-\mu}{\sigma}\right)$$

z	$\Phi(z)$	z	$\Phi(z)$
0,0	0,5000	1,6	0,9452
0,2	0,5793	1,8	0,9641
0,4	0,6554	2,0	0,9772
0,6	0,7257	2,2	0,9861
0,8	0,7881	2,4	0,9918
1,0	0,8413	2,6	0,9953
1,2	0,8849	2,8	0,9974
1,4	0,9192	3,0	0,9987

Sűrűségfüggvénye:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

Várható érték: $E(X) = 0$

Szórás: $D(X) = 1$

1.2. Adatszerkezetekkel kapcsolatos alapfogalmak

Alapfogalmak és Osztályozás

- **Absztrakció:** Nyelvfüggő és implementációs részletek elrejtése azzal, hogy egy logikai modellel ábrázoljuk általánosan a problémát.
- **Absztrakt adatszerkezet:** Az adatelemek és a köztük lévő kapcsolatok logikai modellje, mely független az adattárolás fizikai megvalósításától.

Szükséges fogalmak:

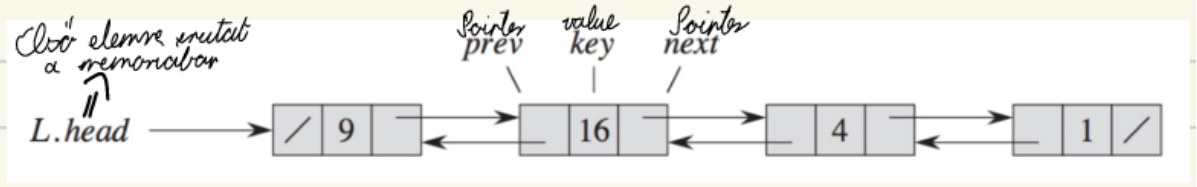
- **Adatszerkezetek részei:**
  - Adatelemek
  - Műveletek
    - Létrehozás
    - Elem elérése
    - Módosítás
      - ✔ Hozzáadás
      - ✔ Csere
      - ✔ Törlés
    - Opcionálisan
      - ✔ Rendezés
      - ✔ Keresés
      - ✔ Bejárás
  - Reprezentáció
  - Kapcsolatok.
- **Adatszerkezetek osztályozása:**
  - **Elemtípus alapján:** Homogén (azonos típusú elemek) / Heterogén (különböző típusú elemek).
  - **Elemszám alapján:** Statikus (fix, előre definiált) / Dinamikus (változhat).
  - **Kapcsolat alapján:** Szekvenciális (lineáris), Hierarchikus (szülő-gyerek), Struktúra nélküli (nincs explicit kapcsolat), Hálós (gráf-szerű), Asszociatív (kulcs-érték párok).
  - **Reprezentáció (memóriabeli elhelyezkedés) alapján:** Folytonos (memóriában egymást követően, pl. vektor) / Szétszórt (memóriában rendszertelenül, láncolt elemekkel, pl. pointerekkel).

Elemi adatszerkezetek

1. Lista

- **Tulajdonságok:** Homogén, szekvenciális, dinamikus, szétszórt .

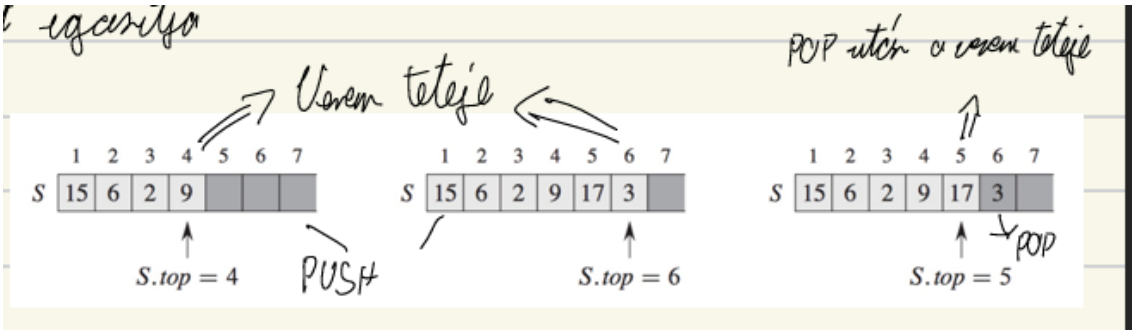
- **Felépítés:** Minden elemének van egy értéke és egy mutatója (pointere) a következő elemre, duplán láncolt listánál kettő (előző és következő elemre).



- **Műveletek:**
  - **Keresés / Elérés:** Lineáris kereséssel megkeresi a listában az első előfordulást.
  - **Beszúrás:** Az elemet beszúrja az adott helyre, a mutatókat (pointereket) átállítja.
  - **Törlés:** Megkeresi az elemet, a memóriát felszabadítja, a pointereket igazítja.

2. Verem (Stack)

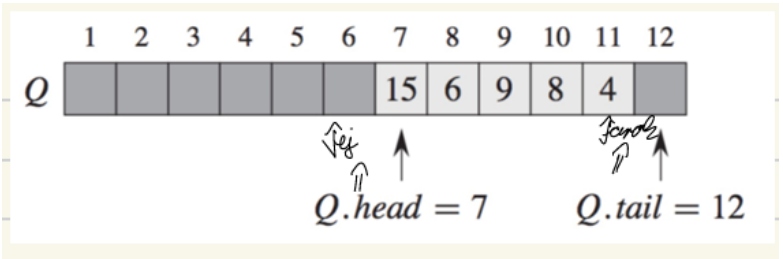
- **Tulajdonságok:** Homogén, dinamikus, szekvenciális, folytonos.
- **Működési elv: LIFO** (Last In First Out) – Mindig a legutoljára berakott elemet adja vissza. Csak a verem tetejéhez (Top) férünk hozzá.



- **Műveletek:**
  - **Push (Beszúrás):** Hozzáadjuk az elemet a verem tetejéhez.
  - **Pop (Kivétel/Törlés):** A verem tetején lévő elemet levesszük.

3. Sor (Queue)

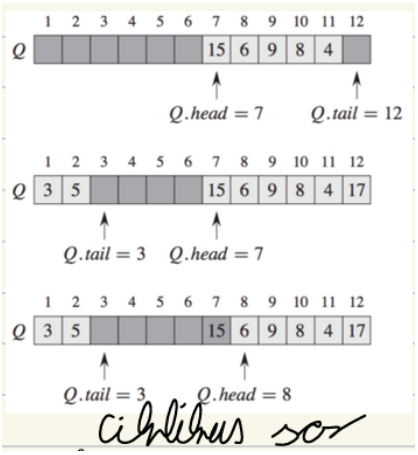
- **Tulajdonságok:** Homogén, dinamikus, szekvenciális, folytonos .
- **Működési elv: FIFO** (First In First Out) – Mindig a leghamarabb berakott elemet adja vissza. Két változót tart nyilván: Fej (Head - legrégebbi elem) és Farok (Tail - legújabb elem) .



- **Műveletek:**

- **Enqueue (Beszúrás):** Elem beszúrása a sor végére (ez lesz az új farok).
- **Dequeue (Kivétel/Törlés):** Sor elején lévő elemet (Fej) kiveszi.

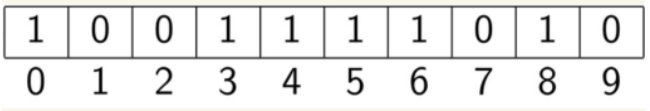
- **Típusok a megvalósítás szerint:**
  - **Naiv sor:** Vektorban tároljuk, minden elem fix helyen van (magas memóriaigény, nem hatékony).
  - **Sétáló sor:** Elem hozzáadásakor/törlésekor az elemeket mozgatjuk a vektorban.
  - **Ciklikus sor:** A vektort körkörösen (ciklikusan) használjuk, a mutatók körbeérnek.



Halmaz, Multihalmaz, Tömb

1. Halmaz

- **Tulajdonságok:** Homogén, struktúra nélküli, dinamikus, folytonos. Egymástól megkülönböztethető elemekből áll (nincs ismétlődés) .
- **Reprezentáció:** Bináris reprezentáció (bitvektor). Ha tudjuk a lehetséges értékeket, nagyon hatékony. (pl. egy 10 elemű univerzumban egy 10 bites szám jelzi, hogy mi van benne és mi nincs).



- **Műveletek (Logikai műveletekkel):**
  - **Unió:** Bitenkénti logikai VAGY.

A={0,3,4,5,6,8}, B={1,3,5,9}.

Az A halmaz reprezentációja:

1	0	0	1	1	1	1	0	1	0
0	1	2	3	4	5	6	7	8	9

A B halmaz reprezentációja:

0	1	0	1	0	1	0	0	0	1
0	1	2	3	4	5	6	7	8	9

Az A és B halmazok **uniójának** kiszámításához a logikai (bitenkénti) **vagy** műveletet használjuk:

1	1	0	1	1	1	1	0	1	1
0	1	2	3	4	5	6	7	8	9

- **Metszet:** Bitenkénti logikai ÉS.
- **Különbség:** A és NEM B (bitenkénti AND NOT).
- **Beszúrás:** Unió egyelemű halmazzal.
- **Törlés:** Különbség egyelemű halmazzal.

2. Multihalmaz

- **Tulajdonság:** Olyan halmaz, amiben egy elem többször is előfordulhat.
- **Reprezentáció:** Nem bináris. Azt tároljuk el, hogy az adott elem hányszor fordul elő a halmazban (darabszámok tömbje).

3. Tömb (Vektor és Mátrix)

- **Tulajdonságok:** Homogén, statikus, asszociatív, folytonos . Előre meghatározott méretű, minden elemet egy (vagy több) index azonosít egyértelműen.
- **Vektor (1D):** Egydimenziós tömb, egymást követő elemek a memóriában.
- **Mátrix (2D):** Kétdimenziós tömb.
  - **Sorfolytonos reprezentáció:** Soronként tároljuk egymás után az elemeket.

Legyen M a következő mátrix:

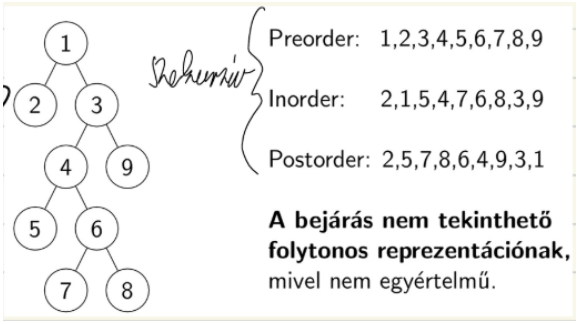
	1	2	3	4	5
1	6	11	8	5	22
2	12	9	2	21	6
3	1	5	24	2	9

Ekkor az M mátrix sorfolytonos reprezentációja:

1. sor					2. sor					3. sor				
6	11	8	5	22	12	9	2	21	6	1	5	24	2	9
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Tárolni kell továbbá a mátrix méretét: (3,5).

  - **Oszlopfolytonos reprezentáció:** Oszloponként tároljuk egymás után az elemeket.



Legyen M a következő mátrix:

	1	2	3	4	5
1	6	11	8	5	22
2	12	9	2	21	6
3	1	5	24	2	9

Ekkor az M mátrix oszlopfolytonos reprezentációja:

1. oszlop			2. oszlop			3. oszlop			4. oszlop			5. oszlop		
6	12	1	11	9	5	8	2	24	5	21	2	22	6	9
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Tárolni kell továbbá a mátrix méretét: (3,5).

- **Ritka mátrix:** Olyan mátrix, aminek az elemei nagyrészt nullák. Memóriaspórolás céljából csak a nem nulla elemeket tároljuk egy 3 oszlopos táblában: [Sor, Oszlop, Érték]. Akkor éri meg használni, ha a nem nulla elemek száma sokkal kisebb, mint a mátrix mérete.

Legyen M az alábbi ritka mátrix:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	0	0	3	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	11	0	0	0	0	0	9	0
3	0	5	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	25	0	0	0

Ekkor az M ritka mátrix 3 soros reprezentációja:

sor	1	2	2	3	4
oszlop	4	7	13	2	11
érték	3	11	9	5	25

Tárolni kell továbbá a mátrix méretét: (4,14).

Fák (Trees)

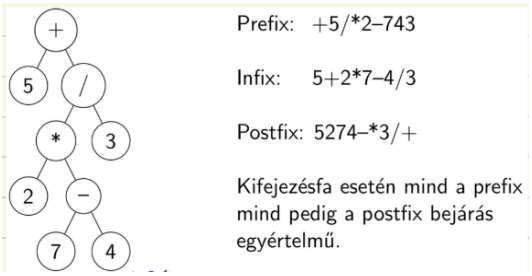
- **Tulajdonságok:** Homogén, dinamikus, hierarchikus. Csúcsok (csomópontok) és a köztük lévő élek alkotják.
- **Főbb fogalmak:** Gyökér, Levelek. Bármely csúcsba pontosan egy út vezet.
- **Reprezentáció:** Folytonos (hierarchikus lista) vagy Szétszórt (pointeres).

Bináris fa bejárások

A fa minden csúcsát pontosan egyszer látogatjuk meg.

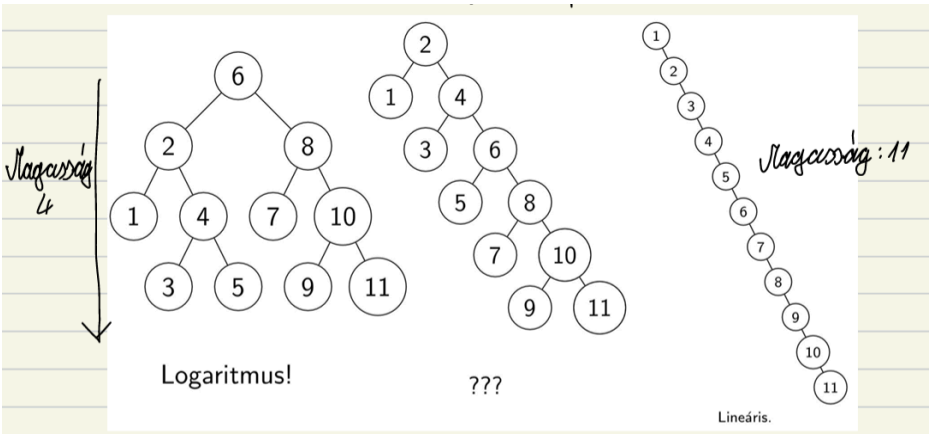
1. **Preorder:** Gyökér → Bal részfa → Jobb részfa.
2. **Inorder:** Bal részfa → Gyökér → Jobb részfa.
3. **Postorder:** Bal részfa → Jobb részfa → Gyökér.

- *Megjegyzés: Kifejezésfák (matematikai műveletek fái) esetén a bejárások adják meg a prefix, infix és postfix matematikai jelöléseket.*



Keresőfák

- **Bináris keresőfa:** Olyan bináris fa, ahol minden csúcsra igaz: a tőle balra lévő részfában kisebb, a tőle jobbra lévő részfában nagyobb értékek találhatók.
    - Műveletek:
        - Keresés: Ha a keresett szám kisebb, mint a csúcs – balra, ha nagyobb – jobbra.
        - Beszúrás: Ha a keresett szám kisebb, mint a csúcs – balra, ha nagyobb – jobbra, amíg nem találunk egy gyerek nélküli csúcsot.
        - Törlés
            - ✔ Ha nincs gyerek – Töröljük
            - ✔ Ha 1 gyerek van – A gyerek kerül az X helyére.
            - ✔ Ha 2 gyerek van – A bal részfa legnagyobb eleme kerül az X helyére.
    - *Probléma:* Kiegyensúlyozatlan lehet (pl. láncolt listává fajul), ekkor a keresés/beszúrás nem logaritmikus, hanem lineáris idejű lesz.

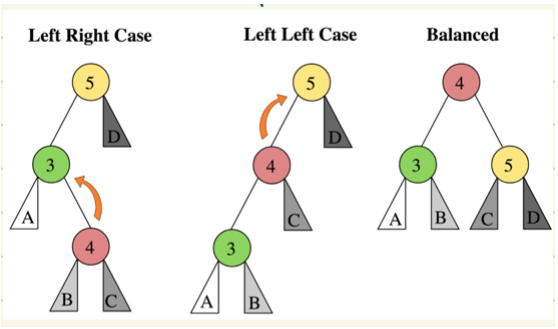


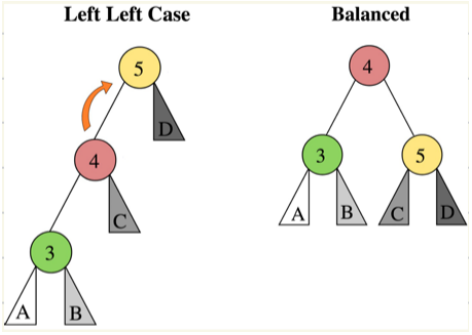
- **Kiegyensúlyozott keresőfa:**

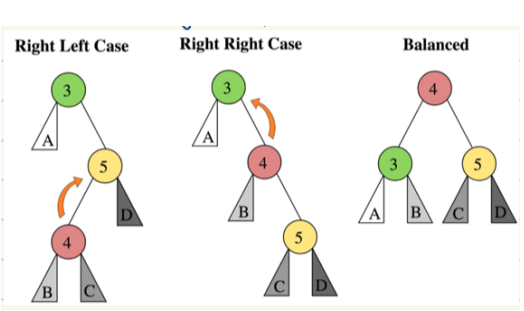
  Olyan fa, ahol bármely csúcs bal és jobb részfájának magasságkülönbsége maximum 1.

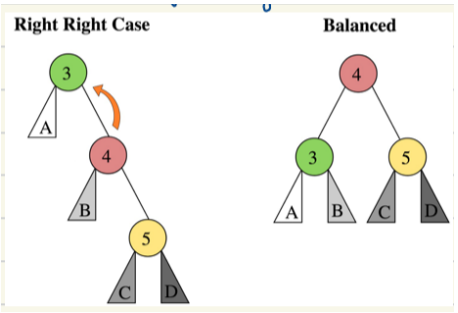
  A műveletek O(n) időben megvalósíthatóak, de a hozzáadás és törlés elronthatja az egyensúlyt.

- **AVL fa (Önkiegyensúlyozó fa):**
  - Folytonos, vagy láncolt reprezentáció.
  - Egyensúlyfaktor: jobboldali és baloldali részfa magasságának különbsége.
  - Ha beszúrás vagy törlés után megbomlik az egyensúly, újraszámoljuk és **forgatásokkal** (Left-Left, Right-Right, Left-Right, Right-Left) állítjuk helyre. Ezzel garantálja a gyors, logaritmikus műveleti időt.









- **Piros-Fekete fa:**
  - Bináris keresőfa, ahol minden csúcshoz 1 bit extra infó (szín: piros vagy fekete) tartozik.
  - *Szabályok:* 1. Minden csúcs piros vagy fekete. 2. A gyökér fekete. 3. Minden NIL levél fekete. 4. Minden piros csúcsnak mindkét gyereke fekete. 5. Bármely csúcstól a levelekig vezető utakon ugyanannyi fekete csúcs van.

- **B-fa**

  Olyan keresőfák, melyek merevlemezes háttértárakon történő adatok kezelésére alakítottak ki (Nem HDD-n is jók). A cél a lehető legkevesebb lemezművelet. Nem bináris fák, lapozások minimalizálása a cél. A B-fa magasságát az határozza meg, hányszor fordulunk háttértárhoz.

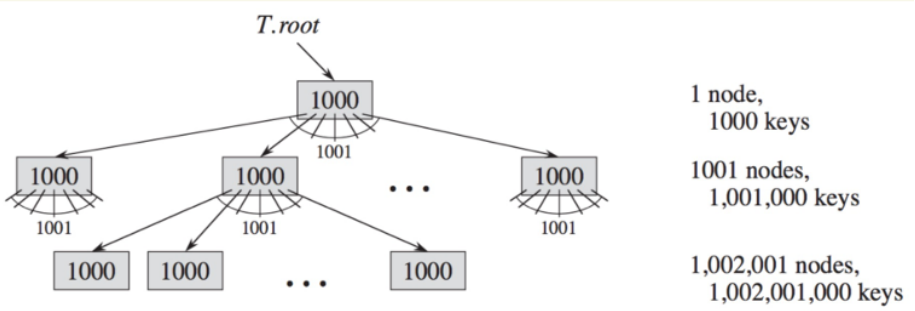
  Tulajdonságok:
  - Minden fának van foka – *t*, melyet mi adunk meg.
  - Minden csúcsnak gyökér kivételével *t-1* értéke van.
  - Minden csúcsnak *2t-1* értéke van.
  - Minden csúcsnak levelet kivéve pontosan kulcs+1 gyereke van, ahol a kulcs a csúcsban lévő elemek száma.
  - Minden levél ugyanazon a szinten van.

  Elágazási tényező:
  - Csúcsok gyerekeinek a száma → [50 - 2000]
  - Minél nagyobb, annál alacsonyabb (a fa) → Háttértáron történő olvasás száma csökken.

  B-fa magassága:

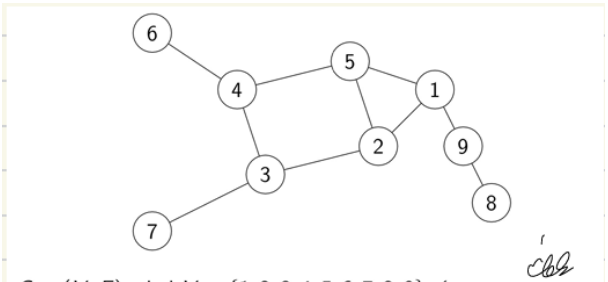
  Bármely n kulcsot tartalmazó (n≥1), t≥2 minimális fokszámú B-fa magassága:

$$h \leq \frac{n+1}{2}$$



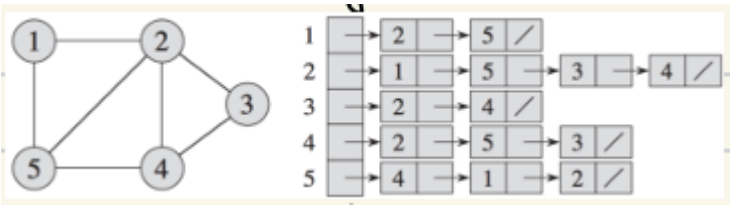
Gráfok

- **Gráf:** Csúcsok és élek véges halmaza.
- **Irányítatlan gráf:** Az éleknek nincs irányuk (a csúcspárok rendezetlenek).

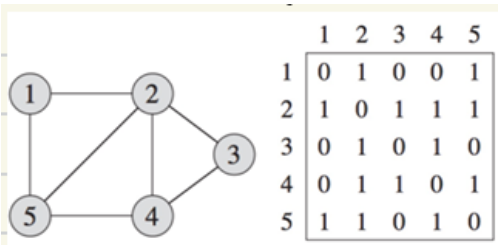


$G = (V, E)$, ahol $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, és
$E = \{(1, 2), (1, 5), (1, 9), (2, 3), (2, 5), (3, 4), (3, 7), (4, 6), (8, 9)\}$.

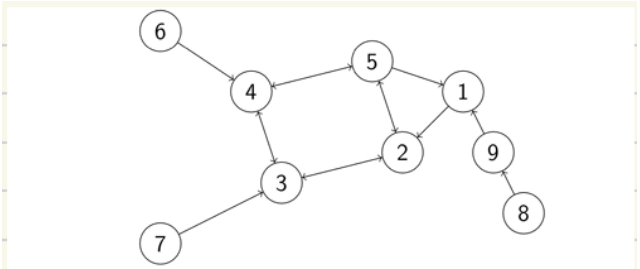
  - **Reprezentációs módok:**
    - **Szomszédsági lista:** Minden csúcshoz egy listában tároljuk, hogy mely csúcsokkal van összekötve.



    - **Szomszédsági mátrix:** Egy 2D-s tömb (M), ahol M[i][j] = 1, ha van él az *i* és *j* csúcs között, egyébként 0.



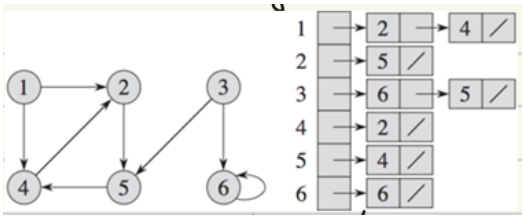
- **Irányított gráf:** Az éleknek van irányuk (a csúcspárok rendezettek).



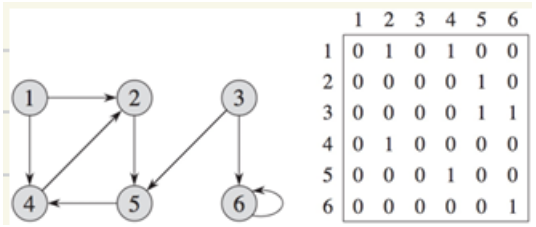
$G = (V, E)$, ahol
$V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, és
$E = \{(1, 2), (2, 3), (2, 5), (3, 2), (3, 4), (4, 3), (4, 5), (5, 1), (5, 2), (5, 4), (6, 4), (7, 3), (8, 9), (9, 1)\}$.

  - **Reprezentációs módok:**

- **Szomszédsági lista**



- **Szomszédsági mátrix**



- **Éllista (Edge List):** Ritka gráfok esetén érdemes használni, csak a meglévő éleket sorolja fel (hasonlóan a ritka mátrix logikájához).

2.1. Valószínűség fogalma és kiszámításának kombinatorikus módszerei (permutációk, variációk, kombinációk). Feltételes valószínűség, függetlenség, Bayes-formula.

Valószínűség fogalma

- **Véletlen tömegjelenség:** Olyan jelenség, melynek eredménye nem determinisztikus (nem jósolható meg előre pontosan), de sokszori megismétlés esetén statisztikai törvényszerűségeket mutat.
- **Klasszikus valószínűség:** A valószínűséget az összes lehetséges eset és a kedvező esetek számának hányadosaként kapjuk meg (feltéve, hogy az elemi események egyenlő valószínűségűek).
- **Valószínűségi mező (Ω, A, P):**
  - **Ω (Mintatér):** Az összes lehetséges kimenetel halmaza.
  - **A (Eseményalgebra):** Ω bizonyos részhalmazai, a megfigyelhető események.
  - **P (Valószínűség):** Minden eseményhez hozzárendel egy $0 \leq P(A) \leq 1$ közötti számot.
- **Axiómák (Kolmogorov):**
  1. Bármely A esemény valószínűsége: $0 \leq P(A) \leq 1$.
  2. A biztos esemény (mintatér) valószínűsége: P(Ω) = 1.
  3. Ha A_1, A_2, ...A_n egymást páronként kizáró (diszjunkt) események, akkor összegük valószínűsége megegyezik a valószínűségeik összegével:

$$P\left(A_1 \cup A_2 \cup ... \cup A_n\right) = P\left(A_1\right) + P(A_2) + ... + P(A_n)$$

Kombinatorikus módszerek

- **Permutáció (Sorbarendezés):**
  - *Ismétlés nélküli: n* darab különböző elem összes lehetséges sorrendje. Száma:

$$P_n = n!$$

  - *Ismétléses: n* darab elem sorrendje, ahol vannak megegyezőek (k_1, k_2 ... db azonos). Száma:

$$P_n^{(k_1, k_2, ...k_i)} = \frac{n!}{k_1! \times k_2! \times ... \times k_i!}$$

- **Variáció (Kiválasztás - számít a sorrend):**
  - *Ismétlés nélküli: n* elemből *k* darabot választunk ki sorrendben, visszatevés nélkül. Száma:

$$V_{n,k} = \frac{n!}{(n-k)!}$$

  - *Ismétléses: n* elemből *k* darabot választunk ki visszatevéssel (k > n). Száma:

$$V_{n,k}^{ism} = n^k$$

- **Kombináció (Kiválasztás - NEM számít a sorrend):**
    - *Ismétlés nélküli: n* elemből *k* darabot választunk ki, a sorrend mindegy. Száma:

$$C_{n,k} = \left(\frac{n}{k}\right) = \frac{n!}{k!(n-k)!}$$

    - *Ismétléses: n* elemből *k* elemet választunk ki visszatevéssel, sorrend függetlenül. Száma:

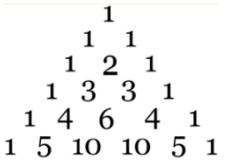
$$C_{n,k}^{ism} = \left(\frac{n+k-1}{k}\right)$$

Binomiális tétel

Binomiális együttható

$$(x+y)^n = \sum_{k=0}^{n} \left(\frac{n}{k}\right) x^k y^{n-k}$$

Pascal háromszög



Feltételes valószínűség és Függetlenség

- **Feltételes valószínűség:** Egy *A* esemény valószínűsége, feltéve, hogy egy *B* (pozitív valószínűségű) esemény már bekövetkezett. Képlete:

$$P(B) = \frac{P(A \cap B)}{P(B)}$$

- **Függetlenség:** *A* és *B* események függetlenek, ha egyik bekövetkezése nem befolyásolja a másikét. Vagyis $P(A|B) = P(A)$. Ebből következik a szorzási szabály:

$$P(A \cap B) = P(A) \cdot P(B)$$

- **Teljes eseményrendszer:** Olyan pozitív valószínűségű események halmaza $\left(A_1, A_2 \dots A_n\right)$, amelyek egymást páronként kizárják, és uniójuk kiadja a teljes eseményteret (Ω). (Vagyis mindig pontosan egy következik be közülük).

Bayes-formula és Teljes valószínűség tétele

- **Teljes valószínűség tétele:** Ha ismerjük egy *B* esemény feltételes valószínűségeit egy teljes eseményrendszer $\left(A_i\right)$ feltételei mellett, akkor *B* teljes valószínűsége:

$$P(B) = \sum_{i=1}^{n} P(B|A_i) \cdot P(A_i)$$

- **Bayes-tétel:** Visszafelé következtetés (okok valószínűségeinek visszaszámolása). Ha a *B* esemény bekövetkezett, mennyi az esélye, hogy ezt az A_j ok idézte elő:

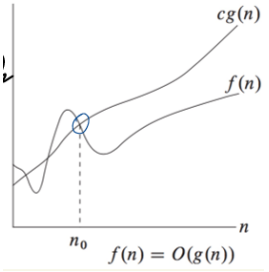
$$P(B) = \frac{P(B|A_j) \cdot P(A_j)}{\sum_{i=1}^{n} P(B|A_i) \cdot P(A_i)}$$

2.2. Algoritmusok lépésszáma, aszimptotikus jelölések. Beszúrásos rendezés, keresések lineáris és logaritmikus lépésszámmal. Táblázatok, hash függvények, hash táblák. Gráfok, szélességi és mélységi bejárás.

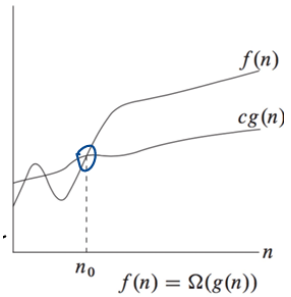
Algoritmusok lépésszáma és aszimptotikus jelölések

Az algoritmusok futási idejét (lépésszámát) a bemenet méretének (n) függvényében adjuk meg. Vizsgáljuk a legjobb, átlagos és legrosszabb esetet.

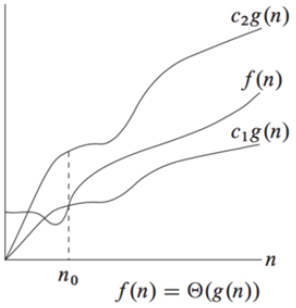
- **Felső korlát (*O* - Origo/Nagy O):** $f(n) = O(g(n))$, ha létezik egy $c > 0$ konstans és n_0 küszöb, hogy $n \geq n_0$ esetén $f(n) \leq c \cdot g(n)$. (A függvény legrosszabb esetben is ezen korlát alatt marad).



- **Alsó korlát (Ω - Omega):** $f(n) = \Omega(g(n))$, ha létezik $c > 0$ és n_0, hogy $n \geq n_0$ esetén $c \cdot g(n) \leq f(n)$.



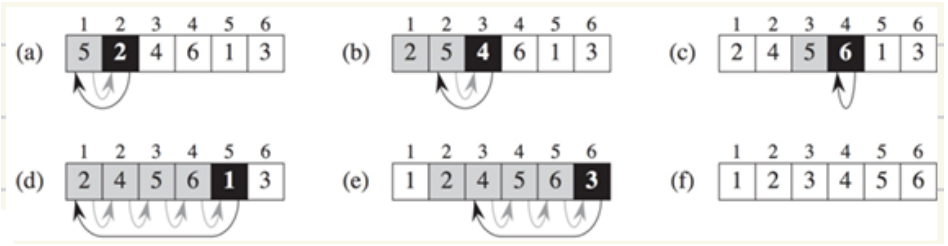
- **Éles korlát (Θ - Théta):** $f(n) = \Theta(g(n))$, ha az algoritmus felülről és alulról is ugyanazzal a függvénnyel becsülhető (konstans szorzó erejéig).



We assume a 30 day month and 365 day year. Egységnyi idő alatt megtett lépések száma - számítógép teljesítménye

	1 Second	1 Minute	1 Hour	1 Day	1 Month	1 Year	1 Century
$\lg n$	$2^{1\times10^6}$	$2^{6\times10^7}$	$2^{3.6\times10^9}$	$2^{8.64\times10^{10}}$	$2^{2.592\times10^{12}}$	$2^{3.1536\times10^{13}}$	$2^{3.15576\times10^{15}}$
$\sqrt{n}$	1×10^{12}	3.6×10^{15}	1.29×10^{19}	7.46×10^{21}	6.72×10^{24}	9.95×10^{26}	9.96×10^{30}
n	1×10^6	6×10^7	3.6×10^9	8.64×10^{10}	2.59×10^{12}	3.15×10^{13}	3.16×10^{15}
$n \lg n$	62746	2801417	133378058	2755147513	71870856404	797633893349	6.86×10^{13}
n^2	1000	7745	60000	293938	1609968	5615692	56176151
n^3	100	391	1532	4420	13736	31593	146679
2^n	19	25	31	36	41	44	51
$n!$	9	11	12	13	15	16	17

Rendezés és Keresés



- **Beszúrásos rendezés:**
  - Úgy működik, mint ahogy a kártyalapokat rendezzük a kezünkben: sorban haladunk, és minden elemet beszúrunk az előtte lévő, már rendezett rész megfelelő helyére.
  - *Legjobb eset: O(n)* (ha a tömb már rendezett).
  - *Legrosszabb eset:* $O(n^2)$ (ha fordítva van rendezve).

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

- **Keresések:**

	rendezett	rendezetlen
folytonos	van	van
láncolt	van	van

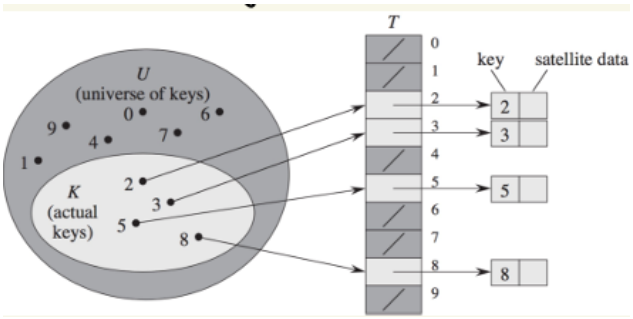
  - **Lineáris keresés:** Végigmegyünk az elemeken sorban. Lépésszám: *O(n)*.
  - **Logaritmikus (Bináris) keresés:** Csak rendezett adathalmazon működik! Mindig a középső elemet vizsgáljuk, és felezzük a keresési tartományt. Lépésszám: $O(\log log\, n)$.

Táblázatok és Hash táblák

Olyan összetett adatszerkezetek, amelyek **kulcs-érték párokat** tárolnak.

Típusok:

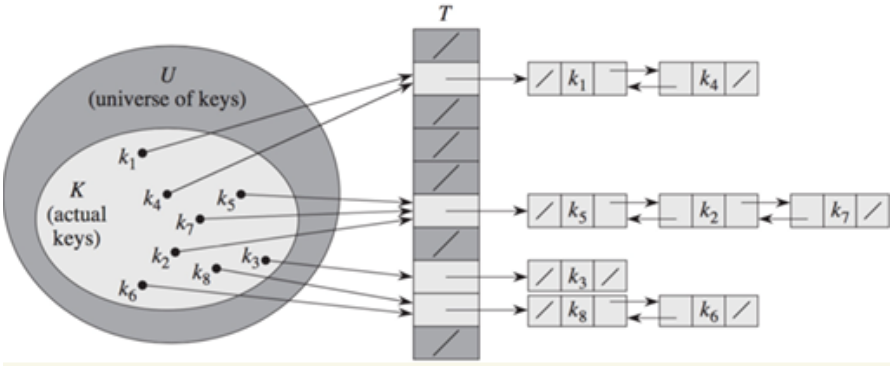
- **Soros táblázat**
- **Közvetlen címzésű táblázat:** A kulcs maga az index (pl. az 5-ös kulcsú adat az 5-ös indexen van a vektorban). Csak akkor jó, ha a lehetséges kulcsok halmaza (kulcstér) kicsi, különben rengeteg memória megy kárba.



- **Hash tábla (Kulcstranszformáció):**

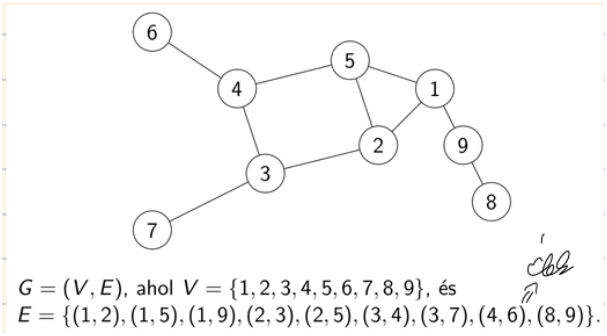
- A közvetlen címzés problémáját oldja meg. Egy **Hash függvény** ($h(k)$) a kulcsból generál egy indexet a táblázatban. Képezi a hatalmas kulcsteret egy sokkal kisebb táblaméretre *(m)*.

- **Ütközés (Collision):** Ha a hash függvény két különböző kulcsra ugyanazt az indexet adja.
- **Ütközés feloldása:**
  - *Láncolással:* A tábla adott indexén nem egy elemet, hanem egy láncolt listát tartunk fenn, amibe fűzzük az azonos hash-értékű elemeket.
  - *Nyílt címzéssel:* Ha a hely foglalt, a táblázaton belül keresünk egy új, szabad helyet (pl. lineáris próba - megnézzük a következőt; vagy kvadratikus próba).

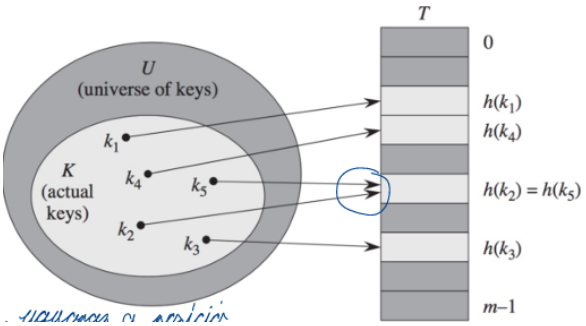


Gráfok

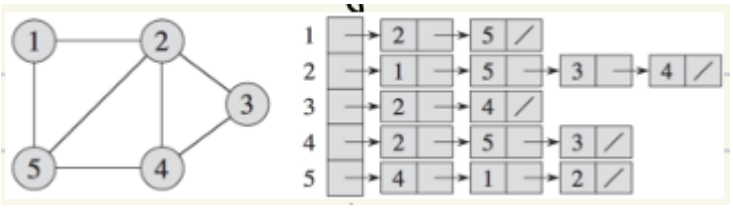
- **Gráf:** Csúcsok és élek véges halmaza.
- **Irányítatlan gráf:** Az éleknek nincs irányuk (a csúcspárok rendezetlenek).



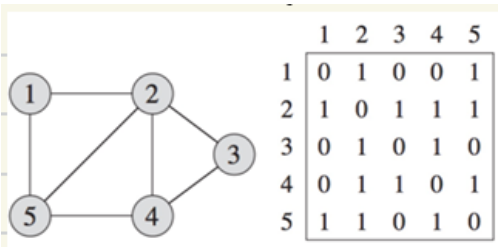
- **Reprezentációs módok:**



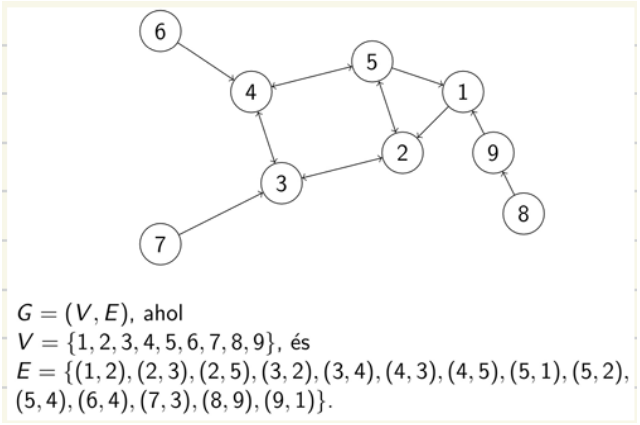
- **Szomszédsági lista:** Minden csúcshoz egy listában tároljuk, hogy mely csúcsokkal van összekötve.



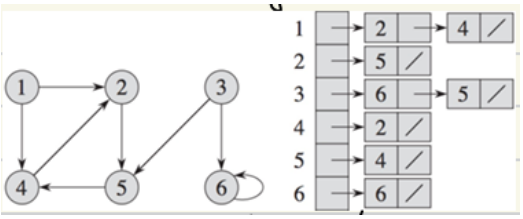
- **Szomszédsági mátrix:** Egy 2D-s tömb (M), ahol M[i][j] = 1, ha van él az *i* és *j* csúcs között, egyébként 0.



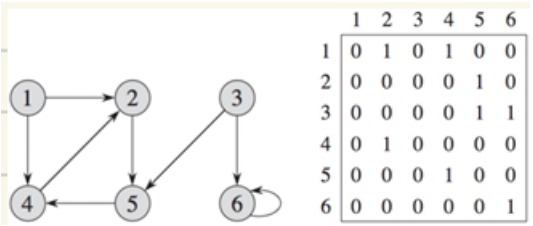
- **Irányított gráf:** Az éleknek van irányuk (a csúcspárok rendezettek).



  - **Reprezentációs módok:**
    - **Szomszédsági lista**



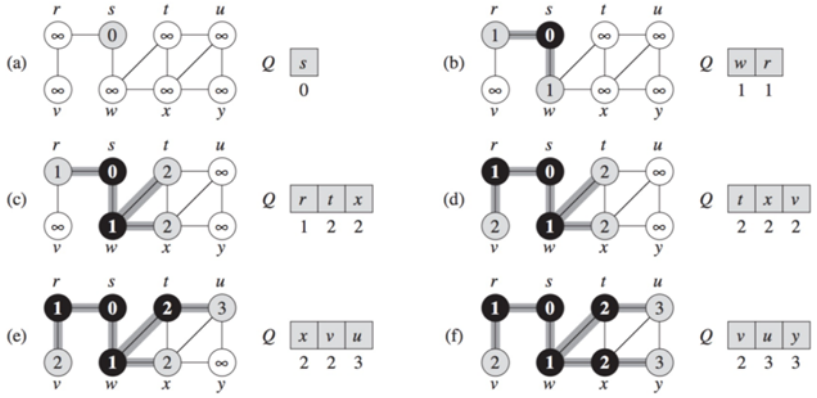
    - **Szomszédsági mátrix**



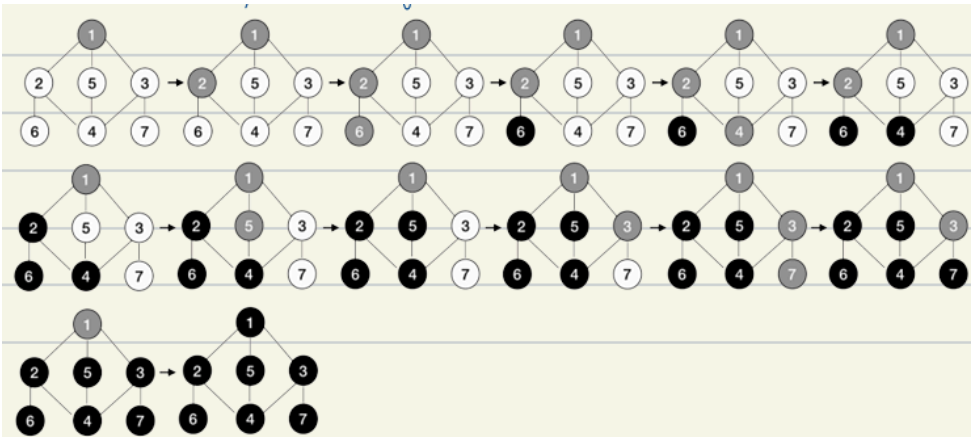
- **Éllista (Edge List):** Ritka gráfok esetén érdemes használni, csak a meglévő éleket sorolja fel (hasonlóan a ritka mátrix logikájához).

Gráfok bejárása

- **Szélességi bejárás (BFS - Breadth-First Search):**
  - **Sor (Queue)** adatszerkezetet használ (FIFO).
  - Hullámszerűen terjed: először a kezdőcsúcs közvetlen szomszédait látogatja meg, majd azok szomszédait, szinteken haladva.
  - Algoritmus: A kezdőpontot a sorba tesszük, majd amíg a sor nem üres, kiveszünk egy elemet, meglátogatjuk, és minden eddig nem látogatott szomszédját betesszük a sorba.



- **Mélységi bejárás (DFS - Depth-First Search):**
  - **Verem (Stack)** adatszerkezetet használ (LIFO - gyakran rekurzióval valósítják meg).
  - Elindul egy adott ágon, és olyan mélyre megy rajta, amennyire csak tud, mielőtt visszalépne (backtrack) az elágazásokhoz.



3.1. Függvények szélsőértéke, függvényvizsgálat. Legkisebb négyzetek módszere.

Függvények szélsőértéke

Naiv megfogalmazásban a függvény egy adott intervallumon belüli minimum vagy maximum értéke. Ha az intervallum nem a teljes értelmezési tartomány, akkor **lokális szélsőértékről** beszélünk.

Tételek (Feltételek):

1. **Szükséges feltétel (1. tétel):** Ha az *f* függvénynek egy x_0 pontban lokális szélsőértéke van, és a függvény itt differenciálható, akkor a deriváltja nulla: $f'(x_0) = 0$. *(Szemléletesen: a függvény érintőjének meredeksége itt vízszintes).*
2. **Elégséges feltétel I. (Előjelváltás):** Legyen $f'(x_0) = 0$.
   - Ha *f'* az x_0-ban pozitívból negatívba vált (+ → -), akkor x_0-ban **lokális maximum** van. *(Előtte nő, utána csökken).*
   - Ha *f'* az x_0-ban negatívból pozitívba vált (- → +), akkor x_0-ban **lokális minimum** van. *(Előtte csökken, utána nő).*
3. **Elégséges feltétel II. (Magasabbrendű deriváltak):** Tegyük fel, hogy $f'(x_0) = f''(x_0) = \ldots = f^{(n-1)}(x_0) = 0$, de $f^{(n)}(x_0) \neq 0$. Ekkor:
   - Ha *n* **páratlan**, akkor x_0-ban **nincs** lokális szélsőérték (ez egy inflexiós pont).
   - Ha *n* **páros**, és $f^{(n)}(x_0) > 0$, akkor x_0 **lokális minimum**.
   - Ha *n* **páros**, és $f^{(n)}(x_0) < 0$, akkor x_0 **lokális maximum**.

A teljes függvényvizsgálat lépései (12 pont)

1. **Értelmezési tartomány (D_f):** Hol van értelmezve a függvény? (pl. osztás nullával, gyök alatt negatív szám kizárása).
2. **Értékkészlet (R_f):** Milyen értékeket vehet fel a függvény?
3. **Paritás és Periodikusság:**
   - **Páros:** $f(x) = f(-x)$ (szimmetrikus az y-tengelyre).
   - **Páratlan:** $f(-x) = -f(x)$ (szimmetrikus az origóra).
   - **Periodikus:** Létezik $p > 0$, hogy $f(x) = f(x + p)$.
4. **Zérushelyek:** Ahol a függvény metszi az x-tengelyt ($f(x) = 0$).
5. **Előjelviszonyok:** Ahol $f(x) > 0$ (x-tengely felett van), és ahol $f(x) < 0$ (x-tengely alatt).
6. **Határértékek (Limeszek):** A függvény viselkedése az értelmezési tartomány szélein, illetve a végtelenben ($f(x)$ és $f(x)$).
7. **Monotonitás (Első derivált teszt):** * Ha $f'(x_0) \geq 0$, a függvény **monoton nő**.

- Ha $f'(x_0) \leq 0$, a függvény **monoton csökken**.

8. **Szakadási helyek:** Ahol a függvény nem folytonos (vizsgálat bal és jobb oldali határértékekkel).

9. **Deriváltak meghatározása:** Az $f'(x)$ (első) és $f''(x)$ (második) deriváltfüggvények kiszámítása.

10. **Szélsőértékek meghatározása:** Az $f'(x) = 0$ egyenlet megoldása, majd az $f''(x)$ előjelének vizsgálata a kapott pontokban (vagy az első derivált előjelváltásának vizsgálata).

11. **Konvexitás és Inflexiós pontok (Második derivált teszt):**
    - Ha $f''(x) > 0$, a függvény **konvex** (mosolyog ∪).
    - Ha $f''(x <)0$, a függvény **konkáv** (szomorú ∩).
    - **Inflexiós pont:** Ahol a görbületi irány (konvexitás) megváltozik. Feltétele: $f''(x_0) = 0$ és $f''(x_0) \neq 0$ (vagy f'' előjelet vált).

12. **Aszimptoták:**
    - *Vízszintes:* A határérték a végtelenben egy konstanshoz tart.
    - *Függőleges:* Ahol a határérték egy véges pontban (általában szakadási helyen) tart a végtelenbe.

Legkisebb négyzetek módszere

Cél: Megfigyelések (t_1, t_2, ..., t_m időpontokban kapott f_1, f_2, ..., f_m mérési értékek) alapján egy $F(t)$ függvény (modell) illesztése úgy, hogy a mérési hibák négyzetösszege minimális legyen.

Minimalizálandó célfüggvény:

$$J(x) = \sum_{i=1}^{m} (F(t_i) - f_i)^2$$

Modelltípusok:

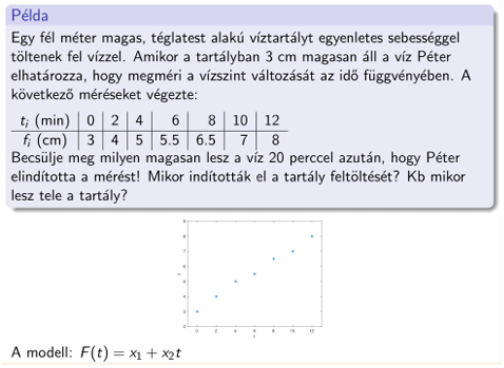
- **Egyenes illesztése:** $F(t) = x_1 + x_2 t$
- **Polinomiális modell:** $F(t) = x_1 + x_2 t + x_3 t^2 + \ldots + x_n t^{n-1}$
- **Trigonometrikus modell:** $F(t) = x_1 + x_2 \sin sin\,(\omega t) + x_3 \cos cos\,(\omega t)$

Matematikai megoldás (Gauss-féle normálegyenlet):

Ha az illesztendő egyenletrendszer felírható mátrixos formában: $Ax = f$, ahol az x az ismeretlen paraméterek vektora, akkor a legkisebb négyzetes megoldást a következő egyenlet adja:

$$A^T A x = A^T f$$

Ez az egyenlet mindig megoldható. Ha az A mátrix oszlopvektorai lineárisan függetlenek (determinánsa nem nulla), akkor pontosan egy (egyértelmű) megoldása van.



Példa

Egy fél méter magas, téglatest alakú víztartályt egyenletes sebességgel töltenek fel vízzel. Amikor a tartályban 3 cm magasan áll a víz Péter elhatározza, hogy megméri a vízszint változását az idő függvényében. A következő méréseket végezte:

t_i (min)	0	2	4	6	8	10	12
f_i (cm)	3	4	5	5.5	6.5	7	8

Becsülje meg milyen magasan lesz a víz 20 perccel azután, hogy Péter elindította a mérést! Mikor indították el a tartály feltöltését? Kb mikor lesz tele a tartály?

A modell: $F(t) = x_1 + x_2 t$

3.2. 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

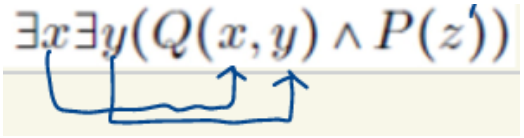
Elsőrendű nyelv szintaxisa

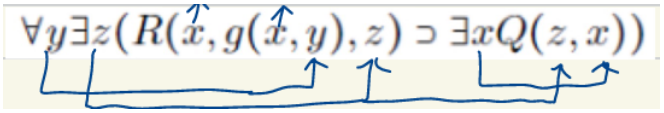
Az elsőrendű logikai nyelv ($L^{(1)}$) egy $\langle LC, Var, Con, Term, Form\rangle$ rendezett ötös, ahol:

- LC **(Logikai konstansok):** ¬ (nem), ∧ (és), ∨ (vagy), ⊃ (implikáció), ≡ (ekvivalencia), $=$ (egyenlőség), ∀ (univerzális kvantor/minden), ∃ (egzisztenciális kvantor/létezik).
- Var **(Változók):** Megszámlálhatóan végtelen halmaz (pl. $x,\ y,\ z$).
- Con **(Nem-logikai konstansok / Paraméterek):**
  - $F(0)$: Névkonstansok (0 argumentumú függvény).
  - $F(n)$: n-argumentumú függvényparaméterek.
  - $P(0)$: Állításkonstansok (0 argumentumú predikátum).
  - $P(n)$: n-argumentumú predikátumparaméterek (igaz/hamis értéket adnak vissza).
- $Term$ **(Termek):** Objektumokat megnevező kifejezések. Névkonstansok, változók, és ezekből függvényparaméterekkel felépített összetett nevek.
- $Form$ **(Formulák):** Állításokat megfogalmazó kifejezések. Predikátumok termeken alkalmazva, melyeket logikai operátorok (∧, ∨, ⊃, ¬) és kvantorok (∀, ∃) köthetnek össze.

Változók kötött és szabad előfordulása

- **Kötött változó:** Olyan változó-előfordulás egy formulában, amely egy kvantor ($\forall x$ vagy $\exists x$) hatáskörébe esik.
- **Szabad változó:** Olyan előfordulás, amely nem esik kvantor hatáskörébe.
- **Nyílt formula:** Tartalmaz legalább egy szabad változót.
- **Zárt formula:** Csak kötött változókat tartalmaz.





Interpretáció és Változókiértékelés

- **Interpretáció $\langle U, \rho \rangle$:** Egy U univerzum (objektumok nem üres halmaza, ami a kontextust adja) és egy ρ kiértékelő függvény, amely jelentést ad a nem-logikai konstansoknak (megmondja konkrétan ki kinek az édesanyja, ki a munkatárs stb.).
- **Változókiértékelés (v):** Egy függvény, amely minden szabad változóhoz (Var) hozzárendel egy konkrét elemet az U univerzumból.

Centrális logikai fogalmak

- **Törvény (Tautológia):** Olyan formula, ami *minden* interpretációban igaz (jelölése: $\models A$).
- **Ellentmondás:** Olyan formula, ami *minden* interpretációban hamis.
- **Kielégíthető:** Van legalább egy modellje (van olyan interpretáció, ahol igaz).
- **Kielégíthetetlen:** Nincs modellje.
- **Következmény ($A \models B$):** Ha A igaz, akkor szükségszerűen B is igaz (A minden modellje B-nek is modellje).
- **Ekvivalencia ($A \equiv B$):** Két formula logikailag ekvivalens, ha $A \models B$ és $B \models A$ is teljesül.

Normálformák

- **Literál:** Atomi formula vagy annak negáltja (pl. A vagy $\neg A$).
- **Elemi Konjunkció / Diszjunkció:** Literálok logikai ÉS / VAGY kapcsolata.
- **Diszjunktív Normálforma (DNF):** Elemi konjunkciók diszjunkciója (ÉS-ek vannak VAGY-gyal összekötve).
- **Konjunktív Normálforma (KNF):** Elemi diszjunkciók konjunkciója (VAGY-ok vannak ÉS-sel összekötve).

Prenex-alak: A formula összes kvantora a formula elejére van kihozva.

Szekvent kalkulus (Logikai levezetés)

A szekvent kalkulus egy formális bizonyítási rendszer. Egy szekvent alakja: $\Gamma \vdash \Delta$, ami azt jelenti, hogy a Γ halmaz formuláinak *konjunkciójából* (bal oldal, feltételek) következik a Δ halmaz formuláinak *diszjunkciója* (jobb oldal, állítások).

Axióma:

Egy szekvent axióma (azonosan igaz), ha ugyanaz a formula szerepel a bal és a jobb oldalon is:

$$\Gamma, A \vdash \Delta, A$$

Levezetési szabályok (Falevél felé építkezve):

- **Konjunkció (∧):**
  - Bal oldal: $\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}$
  - Jobb oldal: $\frac{\Gamma \vdash \Delta, A \qquad \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \wedge B}$
- **Diszjunkció (∨):**
  - Bal oldal: $\frac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}$
  - Jobb oldal: $\frac{\Gamma \vdash \Delta, A, B}{\Gamma \vdash \Delta, A \vee B}$
- **Implikáció (⊃):**
  - Bal oldal: $\frac{\Gamma \vdash \Delta, A \qquad \Gamma, B \vdash \Delta}{\Gamma, A \supset B \vdash \Delta}$
  - Jobb oldal: $\frac{\Gamma, A \vdash \Delta, B}{\Gamma \vdash \Delta, A \supset B}$
- **Negáció (¬):**
  - Bal oldal: $\frac{\Gamma \vdash \Delta, A}{\Gamma, \neg A \vdash \Delta}$
  - Jobb oldal: $\frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \Delta, \neg A}$

Bizonyítás menete:

1. Logikai törvény bizonyításánál a formulát a szekvent **jobb** oldalára tesszük ($\vdash A$).
2. Ellentmondás bizonyításánál a formulát a szekvent **bal** oldalára tesszük ($A \vdash$).
3. A levezetési fákon visszafelé (lentről felfelé) haladunk a szabályok alkalmazásával, amíg minden ág végén (falevél) egy axiómát nem kapunk. Ha minden ág axiómával zárul, a bizonyítás sikeres.

Példa

konjunkció: K

$$\frac{\Gamma \vdash \Delta, A \quad 1 \qquad \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \wedge B} \qquad \frac{\Gamma, A, B \overset{2}{\vdash} \Delta}{\Gamma, A \wedge B \vdash \Delta}$$

diszjunkció: D

$$\frac{\Gamma \vdash \Delta \overset{1}{,} A, B}{\Gamma \vdash \Delta, A \vee B} \qquad \frac{\Gamma, A \vdash \Delta \quad 2 \quad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}$$

implikáció: I

$$\frac{\Gamma, A \overset{1}{\vdash} \Delta, B}{\Gamma \vdash \Delta, A \supset B} \qquad \frac{\Gamma \vdash \Delta, A \quad 2 \quad \Gamma, B \vdash \Delta}{\Gamma, A \supset B \vdash \Delta}$$

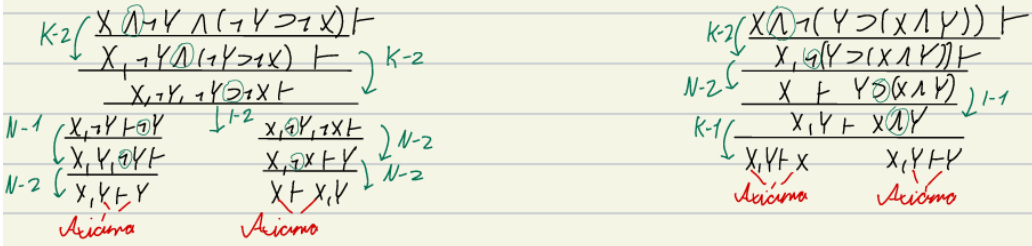
negáció: N

$$\frac{\Gamma, A \overset{1}{\vdash} \Delta}{\Gamma \vdash \Delta, \neg A} \qquad \frac{\Gamma \vdash \overset{2}{\Delta}, A}{\Gamma, \neg A \vdash \Delta}$$

11.I.14. Bizonyítsuk be, hogy az alábbi formulák ellentmondásosak!

(a) $X \wedge \neg Y \wedge (\neg Y \supset \neg X)$ (b) $X \wedge \neg(Y \supset (X \wedge Y))$

Jövel itt ellentmondást bizonyítunk, a szekvent bal oldalára helyezzük a formulát



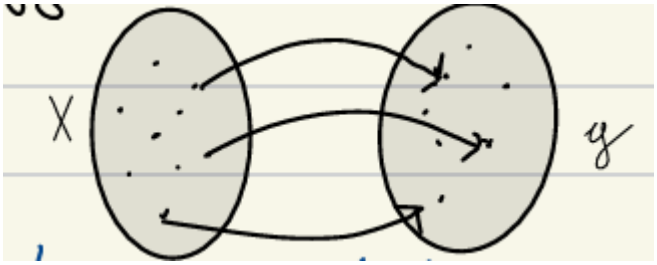
Indoklás:

Levezetési fa minden levele axióma, tehát a levezetés bizonyítás. Jövel a szekvent bal oldalára helyezve a formulát, a levezetés bizonyítás, ez pedig formula ellentmondás

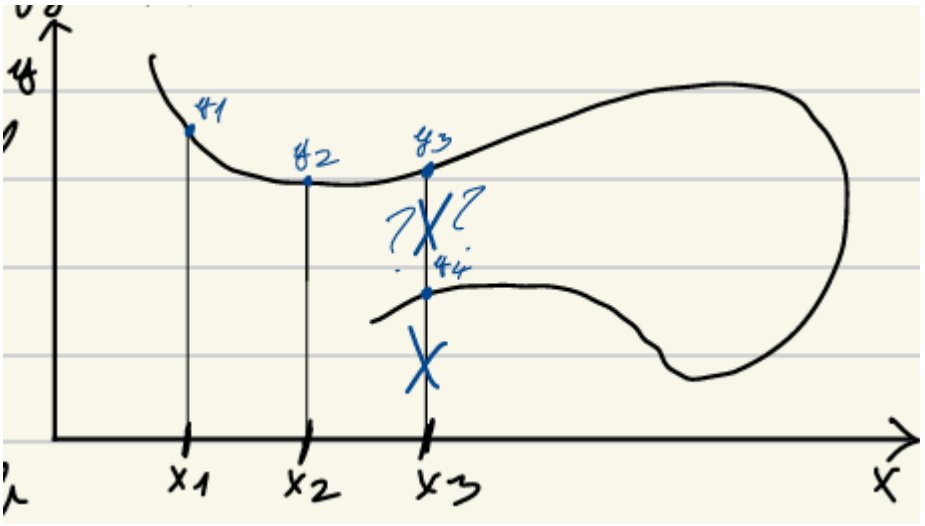
4.1. Függvények, görbék, felületek leírása és számítógépes ábrázolása

Függvények és Görbék megadási módjai a síkban (2D)

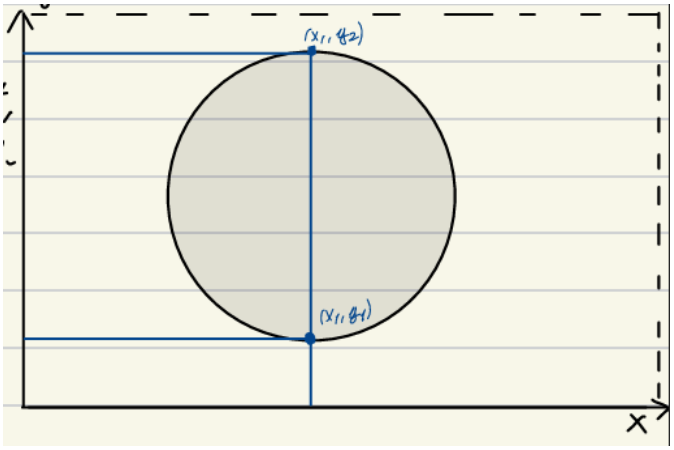
Egy függvény két halmaz (értelmezési tartomány és értékkészlet) közötti egyértelmű leképezés. Egy görbe ennél lazább fogalom, maga alá is hajolhat. A számítógépes grafikában három fő megadási módot használunk :



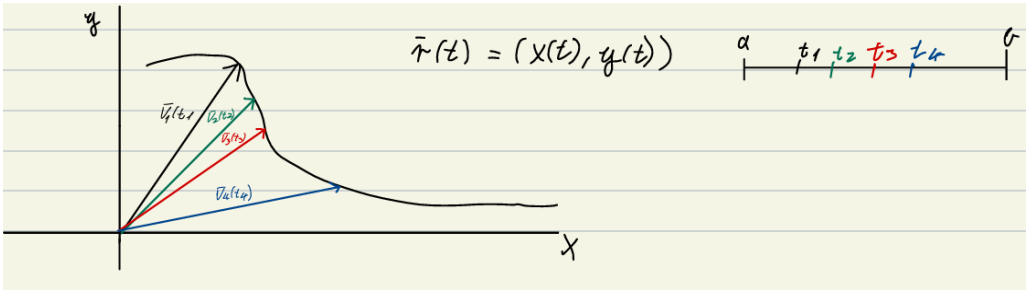
1. **Explicit megadás:**
    - **Képlet:** y = f(x) (pl. y = 3x + 5).
    - **Jellemzők:** Egy x értékhez csak egyetlen y érték tartozik (szigorú egyértelmű leképezés).
    - **Ábrázolás:** Könnyen és gyorsan ábrázolható, nem igényel sok erőforrást.
    - **Hátrány:** Nagyon korlátozott. Nem tud "maga alá görbülni" vagy zárt alakzatot (pl. kört) leírni. Számítógépes grafikában emiatt rugalmatlan.



2. **Implicit megadás:**
    - **Képlet:** f(x, y) = 0 (pl. $x^2 + y^2 - 9 = 0$) .
    - **Jellemzők:** Két valós számból álló párhoz rendel egy valós számot. Itt már egy x-hez több y is tartozhat (ez már görbe, nem függvény).
    - **Ábrázolás:** Kifejezetten nehéz és borzasztóan erőforrásigényes. A képernyő minden egyes pontját (x, y) be kell helyettesíteni az egyenletbe, hogy kiderüljön, rajta van-e a görbén.
    - **Verdikt:** Vizualizáció szempontjából zsákutca, a grafikában nem igazán használjuk.



3. **Paraméteres megadás (A nyertes módszer!):**
    - **Képlet:** r(t) = (x(t), y(t)).
    - **Jellemzők:** Koordinátafüggvényekkel adjuk meg. Van egy 't' paraméterünk (pl. idő), amely végigfut egy intervallumon, és minden 't' pillanathoz adunk egy x és egy y koordinátát .
    - **Ábrázolás:** Rendkívül rugalmas! Bármilyen alakzat, zárt görbe (hurok) is leírható vele. Csak a 't' sűrűségét kell megadni, és a gép pontról pontra kirajzolja. A grafikában szinte kizárólag ezt használjuk !



Görbék és Felületek a térben (3D)

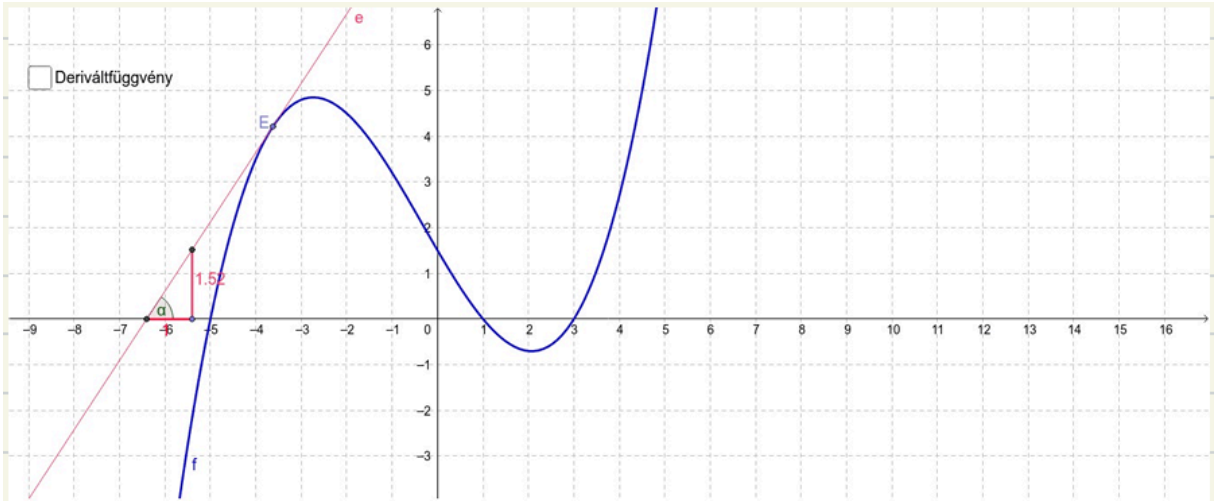
A térben is a paraméteres megadás a leghatékonyabb :

- **Térbeli görbe:** Egyetlen 't' paraméter kell. Képlete: r(t) = (x(t), y(t), z(t)).
- **Felületek ábrázolása (2 paraméter kell, pl. t és s):**
    - **Explicit:** z = f(x, y). Jól programozható, de felülről nézve nem tud "maga alá hajolni" (pl. egy barlang plafonját nem lehet vele leírni).
    - **Implicit:** f(x, y, z) = 0. Nagyon erőforrásigényes, úgynevezett "réteges" megjelenítés.
    - **Paraméteres felület:** F(t, s) = (x(t, s), y(t, s), z(t, s)). Dupla ciklussal lefedjük a paramétersíkot, hálószerű (rácsszerű) felületet alkotva. Gyors, hajlékony, maga alá tud görbülni – 3D grafikában ez a standard !

Interpoláció (Adott pontokra görbe illesztése)

Sokszor vannak fix pontjaink, amikre görbét akarunk illeszteni :

- **Lagrange-interpoláció:** Magas fokszámú polinomot illeszt. Fő hátránya, hogy sok pont esetén a polinom fokszáma is nagyon megnő, emiatt instabillá válik és oszcillálni (hullámzani) kezd a pontok között.
- **Hermite-interpoláció:** Megoldja az oszcillációt azzal, hogy darabonként (mindig csak 2 pont között) rajzol görbét. Viszont a felhasználótól megköveteli, hogy a pontok mellett az érintővektorokat (a görbe meredekségét/deriváltját) is adja meg a végpontokban, ami a való életben ritka .
- **Bézier-görbe:** A Hermite módszerből alakult ki. Két végpontot és (általában) két kontrollpontot használ az érintők meghatározására. Nagyon népszerű a grafikában.
- **Spline:** Alacsony fokszámú görbedarabok finom, törésmentes összekötése.



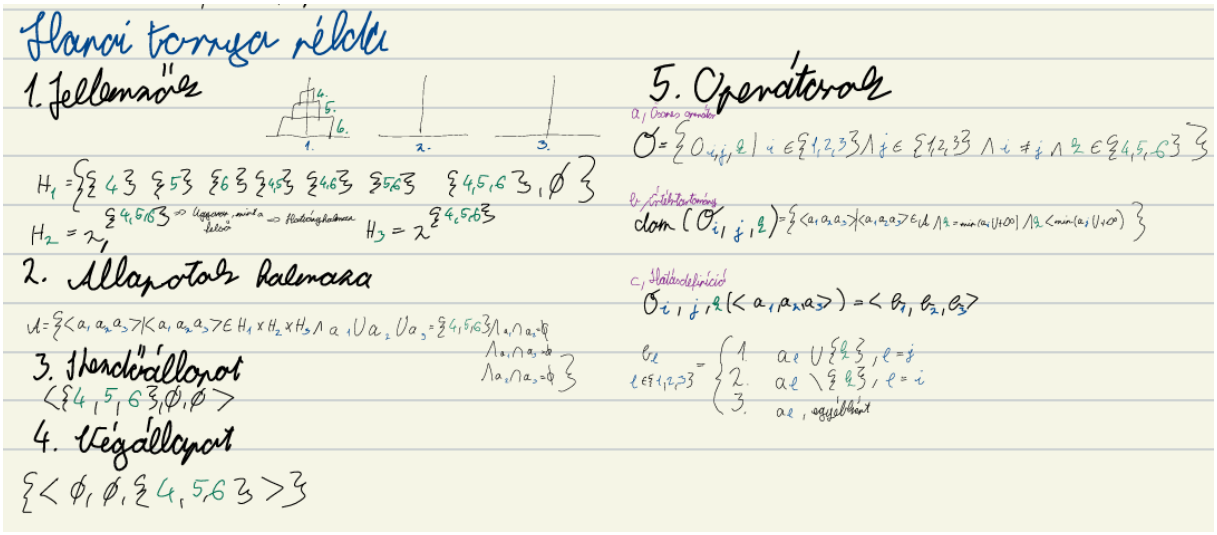
4.2. Problémák reprezentálása állapottéren. A megoldás keresése visszalépéssel. Szisztematikus és heurisztikus fa- és gráfkereső eljárások.

Állapottér reprezentáció

Egy probléma matematikai, elvont (absztrakt) megfogalmazása, amely figyelmen kívül hagyja a felesleges implementációs részleteket. Egy $\langle A, a_0, C, O$ négyessel írható le :

- A **(Állapotok halmaza):** A probléma érvényes fizikai konfigurációi.
- a_0 **(Kezdőállapot):** Innen indul a keresés.
- C **(Célállapotok halmaza):** A kívánt végállapotok.
- O **(Operátorok halmaza):** A végrehajtható cselekvések (szabályok). Rendelkeznek *előfeltétellel* (mikor alkalmazhatóak) és *hatásdefinícióval* (milyen új állapotot hoznak létre) .

(Fontos megjegyzés: Az "állapot" a feladat fizikai állása. A "csúcs" viszont egy memóriabeli adatszerkezet a fában, ami tárolja az állapotot, a szülőt, a mélységet és az útköltséget! A csúcs nem egyenlő az állapottal!)

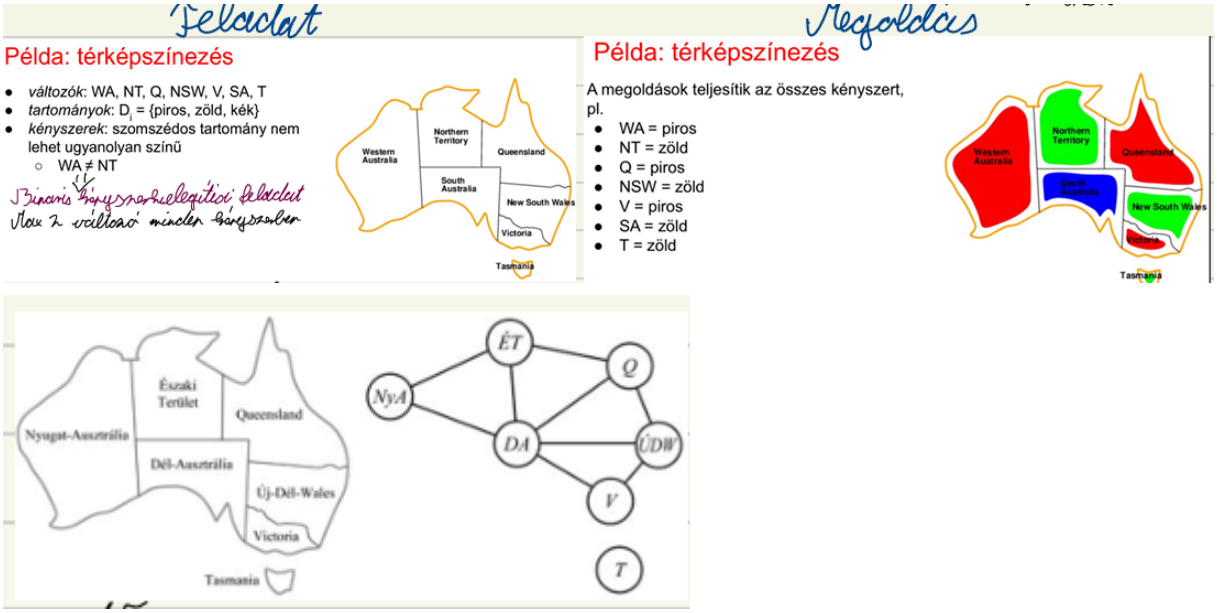


Kényszerkielégítési problémák (CSP - Constraint Satisfaction Problem)

Olyan problémák (pl. térképszínezés, órarendtervezés), ahol adottak:

- **Változók:** (X_i).
- **Tartományok:** (D_i) Minden változó csak innen vehet fel értékeket.
- **Kényszerek (Constraints):** Szabályok a változókra. Lehetnek unárisak (1 változót érint), binárisak (2 változót érint, pl. szomszédos terület nem lehet azonos színű) vagy puha kényszerek (preferenciák) .

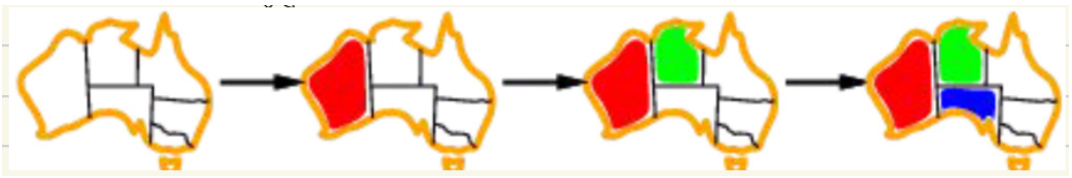
A problémát gyakran **Kényszergráffal** ábrázoljuk, ahol a csúcsok a változók, az élek pedig a kényszerek.



Visszalépéses keresés (Backtracking) és Heurisztikák (CSP gyorsítása)

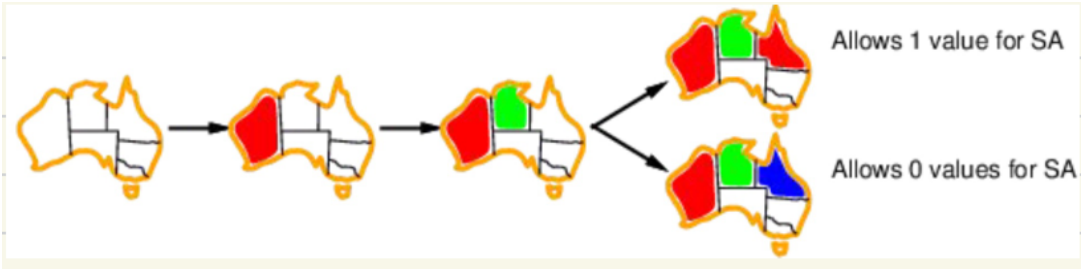
A visszalépéses keresés egy speciális mélységi keresés CSP-re, ami egyszerre csak egy változónak ad értéket. Ha megsért egy kényszert (zsákutca), visszalép . Hogy ne legyen lassú, az alábbi heurisztikákat alkalmazzuk a választásoknál :

- **MRV (Minimum Remaining Values - Legkevesebb fennmaradó érték):** Azt a változót választja ki következőnek, amelynek a legkevesebb megengedett értéke maradt a tartományában. (Gyorsan megtalálja a hibát) .

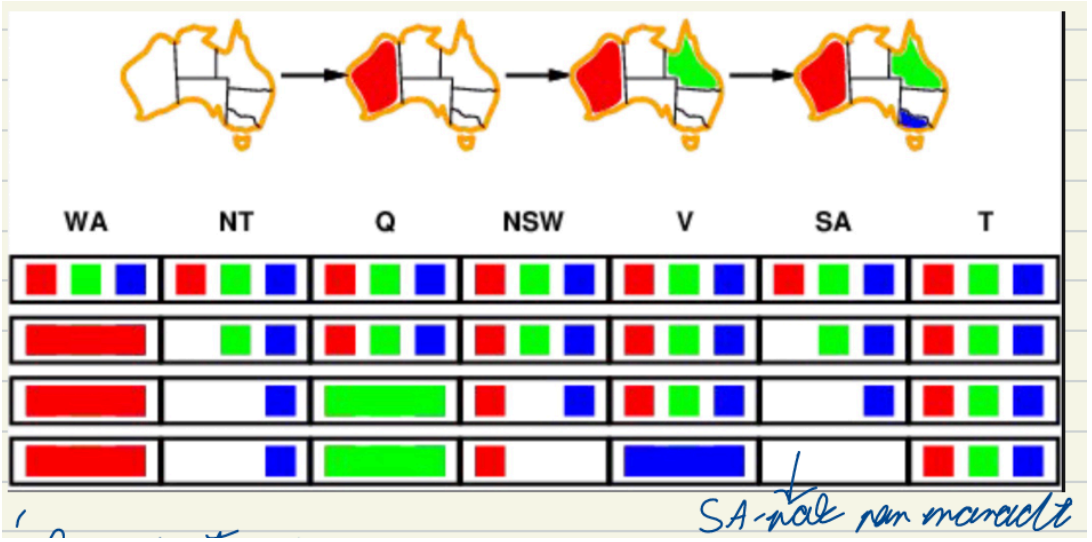


- **Fokszám heurisztika:** Ha az MRV alapján döntetlen van, azt a változót választja, amelyik a legtöbb kényszerben (élben) vesz részt a gráfban.

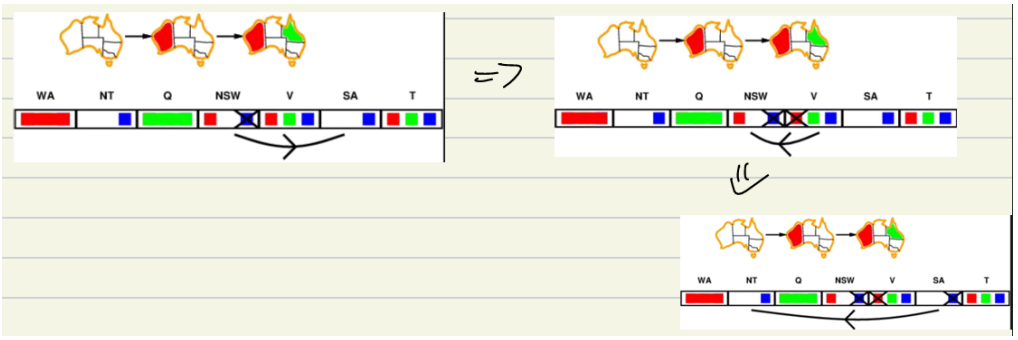
- **LCV (Least Constraining Value - Legkevésbé korlátozó érték):** Az értékek közül azt próbálja ki először, ami a lehető legkevesebb opciót zárja ki a többi (még üres) változó elől.



- **Előrenéző ellenőrzés (Forward checking):** Amikor egy változó értéket kap, azonnal törli a vele kapcsolatban lévő változók tartományából az ütköző értékeket. Ha valamelyik változó tartománya kiürül, azonnal visszalép .



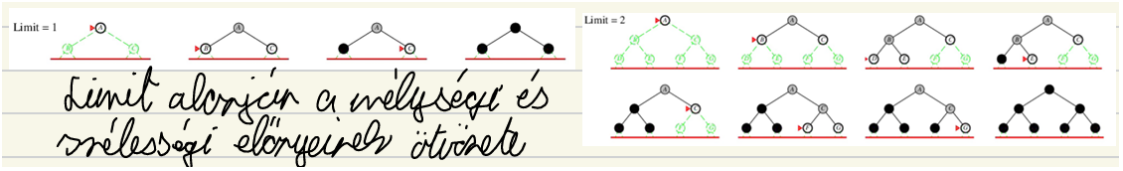
- **Élkonzisztencia (Arc consistency):** Erősebb, mint az előrenézés. Minden lépésben az összes kényszerélet konzisztenssé teszi a gráfban .



Keresőalgoritmusok (Szisztematikus / Neminformált)

Ezek az algoritmusok vakon keresnek, csak az állapottér struktúráját ismerik.

1. **Szélességi keresés (BFS):**
    - Sor (Queue) adatszerkezetet használ. Szintről szintre halad, mindig a legkisebb mélységű csúcsot bontja ki.
    - **Teljes?** Igen.
    - **Optimális?** Csak akkor, ha minden lépésköltség megegyezik (pl. 1).
    - **Hátrány:** Nagyon magas a memória- és időigénye: $O(b^d)$.
2. **Egyenletes költségű keresés:** Mindig a legkisebb eddigi összköltségű ($g(n)$) csúcsot terjeszti ki. Teljes és optimális .
3. **Mélységi keresés (DFS):**
    - Verem (Stack) adatszerkezet. Mindig a legmélyebb csúcsot fejti ki. Ha elakad, visszalép.
    - **Teljes?** Nem (végtelen ágon vagy körben beragadhat).
    - **Optimális?** Nem.
    - **Előny:** Kicsi a tárigénye: $O(b \cdot m)$.
4. **Iteratívan mélyülő keresés:** A mélységi és szélességi keresés legjobb tulajdonságait ötvözi. Növeli a mélységi limitet (0, 1, 2...), és minden limitnél lefuttat egy mélységi keresést. Teljes és optimális (mint a szélességi), de kicsi a tárigénye (mint a mélységinek) .



5. **Gráfkeresés (Körök elkerülése):** Tart egy "Explored" (látogatott) listát. Ezzel exponenciálisan gyorsítja a keresést, mert nem értékel ki egy állapotot többször .

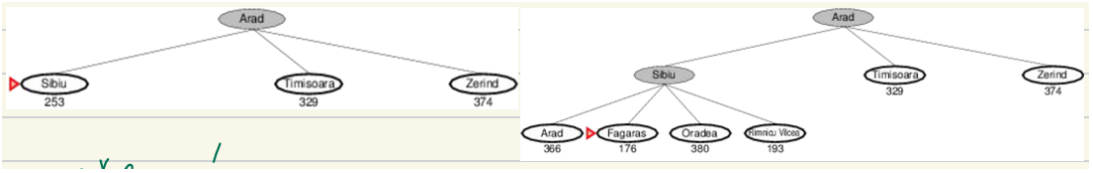
Algoritmusok összefoglalása

tulajdonság	szélességi	egyenletes költségű	mélységi	mélység-korlátozott	iteratívan mélyülő mélységi
teljes?	igen*	igen*	nem	igen, ha $l \geq d$	igen
idő-bonyolultság	b^{d+1}	$b^{\lceil C^*/\varepsilon \rceil}$	b^m	b^l	b^d
tár-bonyolultság	b^{d+1}	$b^{\lceil C^*/\varepsilon \rceil}$	bm	bl	bd
optimális?	igen*	igen	nem	nem	igen*

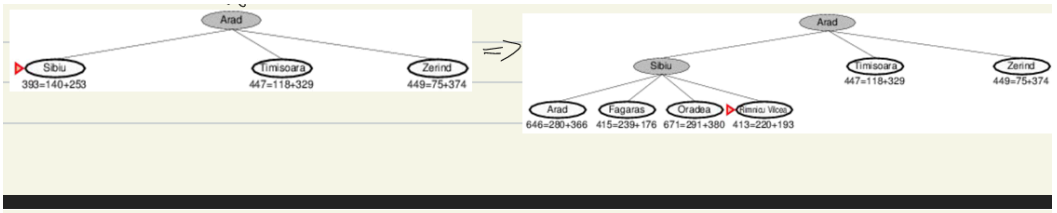
Heurisztikus / Informált keresők

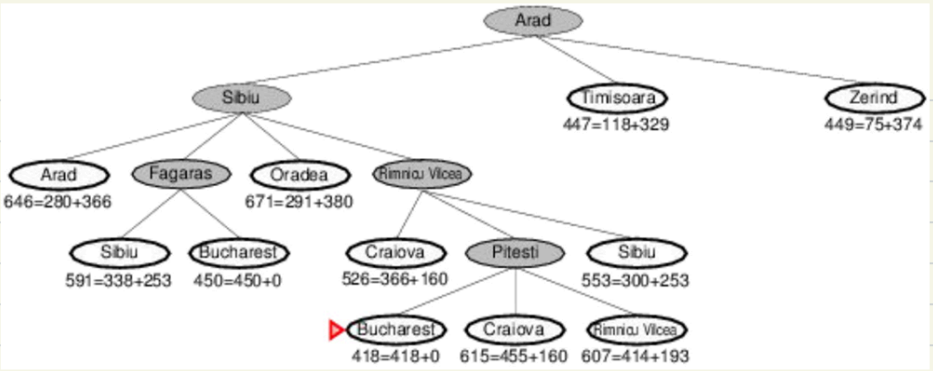
Egy $h(n)$ becslőfüggvényt (heurisztikát) használnak, ami megbecsüli, milyen messze van a csúcs a céltól (pl. légvonalbeli távolság) .

1. **Mohó legjobb először (Greedy):**
   - Kiértékelő függvény: $f(n) = h(n)$.
   - Mindig azt az utat választja, ami a "legközelebbinek tűnik" a célhoz.
   - **Teljes?** Nem, beragadhat.
   - **Optimális?** Nem .



2. *A (A-csillag) keresés:**
   - Kiértékelő függvény: $f(n) = g(n) + h(n)$.
   - $g(n)$: A starttól az aktuális 'n' csúcsig megtett tényleges útköltség.
   - $h(n)$: Becsült hátralévő költség a célig.
   - **Teljes és Optimális**, FELTÉVE, hogy a heurisztikája **elfogadható** (admissible).
   - *Elfogadható heurisztika:* $h(n) \leq h^*(n)$, azaz a heurisztika sosem becsülheti túl a tényleges költséget. (A legpesszimistább becslés is optimista vagy pontos kell, hogy legyen) . Tárigénye sajnos nagy, mert minden nyitott csúcsot a memóriában kell tartania.

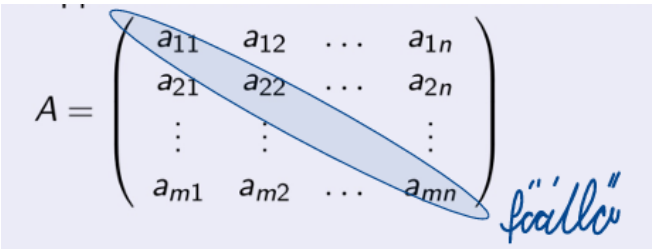

Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

5.1. Mátrix fogalma, műveletek, determináns, rang, sajátérték

Mátrix fogalma és speciális mátrixok

A mátrix egy számokból álló, sorokba és oszlopokba rendezett táblázat. Egy $m \times n$-es mátrixnak m sora és n oszlopa van.



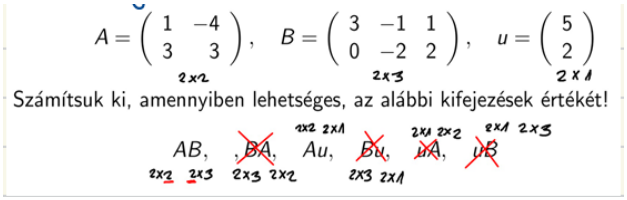
- **Kvadratikus (négyzetes) mátrix:** A sorok és oszlopok száma megegyezik ($n = m$).
- **Főátló:** A bal felső sarokból a jobb alsó sarokba tartó elemek ($a_{11}, a_{22}, a_{33},$).
- **Transzponált mátrix (A^T):** A mátrix sorait és oszlopait megcseréljük.
- **Szimmetrikus mátrix:** Megegyezik a saját transzponáltjával ($A = A^T$).
- **Egységmátrix (E vagy I):** Olyan kvadratikus mátrix, amelynek a főátlójában csupa 1-es, mindenhol máshol 0 áll.

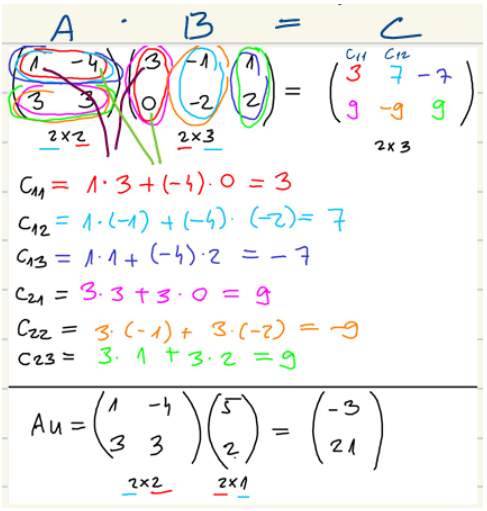
$$E_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad E_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- **Inverz mátrix (A^{-1}):** Csak négyzetes mátrixoknak lehet. Az a mátrix, amivel az eredetit megszorozva az egységmátrixot (E) kapjuk: $A \cdot A^{-1} = E$. Csak akkor létezik (invertálható/reguláris a mátrix), ha a determinánsa NEM nulla.

Mátrixműveletek

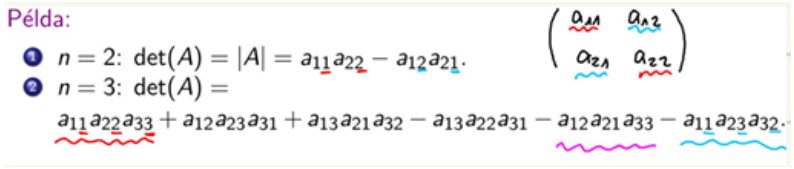
1. **Összeadás:** Csak azonos méretű (típusú) mátrixokat lehet összeadni, elemenként.
2. **Skalárral való szorzás:** A mátrix minden egyes elemét megszorozzuk az adott λ számmal.
3. **Mátrixszorzás (Sor-Oszlop szorzás):** * Csak akkor végezhető el ($A \cdot B$), ha az első mátrix (A) oszlopainak száma megegyezik a második mátrix (B) sorainak számával.
    - Kiszámítása: Az A mátrix i-edik sorát szorozzuk a B mátrix j-edik oszlopával elemenként, majd ezeket összeadjuk.
    - **NEM kommutatív:** Általában $A \cdot B \neq B \cdot A$. (De asszociatív és disztributív).



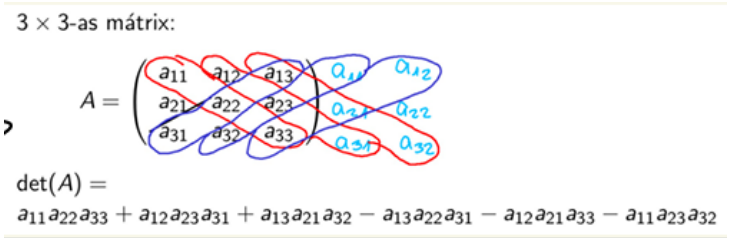


Determináns és Rang

- **Determináns ($de(A)$ vagy $|A|$):** Egy négyzetes mátrixokhoz rendelt speciális számérték.
  - Kiszámítása 2×2-es mátrixnál: Főátló elemeinek szorzata MÍNUSZ a mellékátló elemeinek szorzata.



  - Nagyobb mátrixoknál Kifejtési tétel vagy Sarrus-szabály alkalmazható.

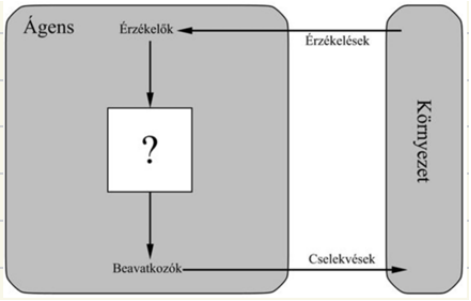


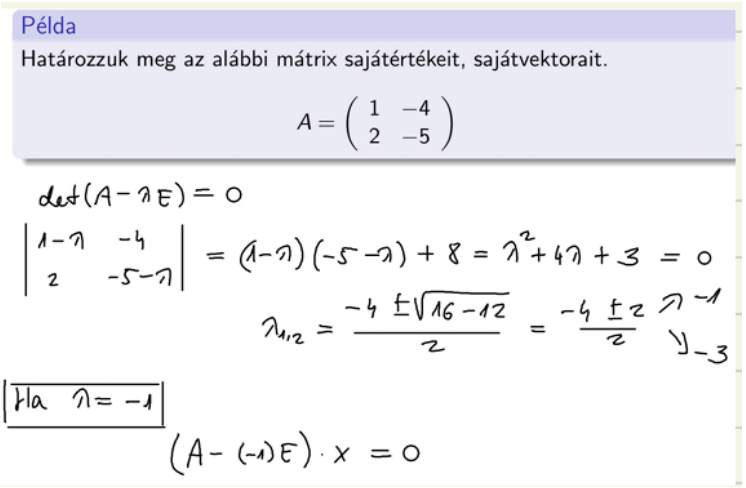
  - Ha a determináns 0, a mátrixot szinguláris nak hívjuk (nincs inverze, sorai/oszlopai lineárisan összefüggők).
  - Ha egy mátrix bármely sora csupa nulla, vagy két sora azonos, a determináns 0.
- **Rang:** A mátrix lineárisan független sorainak (vagy oszlopainak) száma. Gauss-eliminációval számolható.

Mátrix mint Lineáris Transzformáció, Sajátérték és Sajátvektor

- Minden A mátrix felfogható egy függvényként, ami egy v vektort egy másik vektorba visz át (leképez): $\varphi(v) = A \cdot v$.
- **Sajátvektor (x):** Egy olyan (nem nulla) vektor, aminek az iránya nem változik meg a mátrixszal való szorzás során, csak a hossza.
- **Sajátérték (λ):** Az a szám (skalár), amennyiszeresére nyúlik vagy zsugorodik a sajátvektor.

- **Képlete:** $A \cdot x = \lambda \cdot x$
- **Kiszámítása:** A karakterisztikus egyenlet megoldásával történik: det $det\ (A - \lambda \cdot E) = 0$.





5.2. Intelligens ágensek és Megerősítéses Tanulás (RL)

Ágensek és Racionalitás

Az ágens egy olyan entitás, amely érzékelőkön (szenzorokon) keresztül figyeli a környezetét, és beavatkozókon (aktuátorokon) keresztül cselekszik. Formálisan egy függvény, ami az észlelési sorozathoz rendel egy cselekvést.

- **Racionális ágens:** Mindig azt a cselekvést választja, amely az addigi érzékelések alapján maximalizálja a várható teljesítményértéket.
- **PEAS modell:** Egy ágens tervezésénél meg kell adni: Performance (Teljesítményérték), Environment (Környezet), Actuators (Beavatkozók), Sensors (Érzékelők). (Példa automata taxinál: Biztonság, Városi utak, Kormány/Gáz, Kamera/GPS).

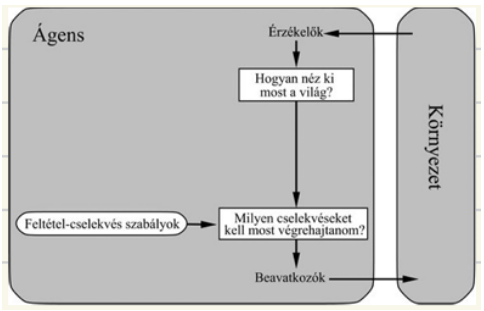
Környezet tulajdonságai

- **Megfigyelhetőség:** Teljesen (mindent lát a döntéshez) vagy Részben megfigyelhető.

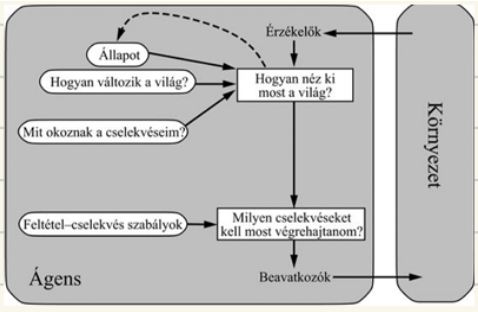
- **Determinisztikusság:** Determinisztikus (a cselekvés és a jelen állapot egyértelműen meghatározza a következőt) vagy Sztochasztikus (véletlenszerű, valószínűségi).
- **Időbeliség:** Epizodikus (egyszeri döntések) vagy Sorozatszerű (a jelenlegi döntés hat a jövőre).
- **Dinamika:** Statikus (nem változik gondolkodás közben) vagy Dinamikus (változik).
- **Értékek:** Diszkrét (véges számú állapot/lépés, pl. sakk) vagy Folytonos (pl. autóvezetés).

Ágensek Típusai

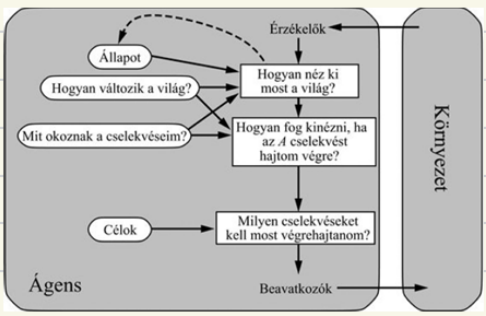
1. **Egyszerű reflex ágens:** Csak a pillanatnyi érzékelés alapján, "Feltétel-Cselekvés" szabályok (Ha-Akkor) alapján dönt.



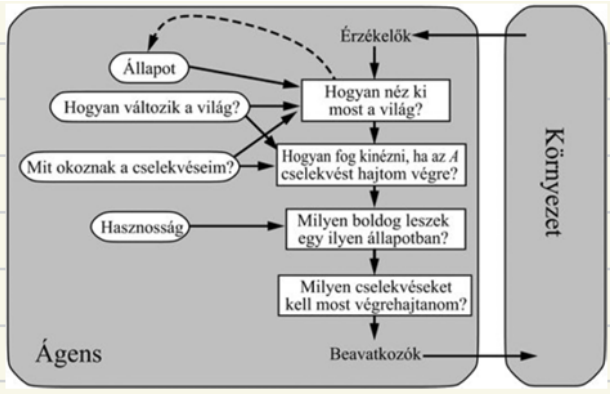
2. **Modell alapú reflex ágens:** Van memóriája, épít egy belső modellt a világ működéséről, így tudja, mi történt korábban.



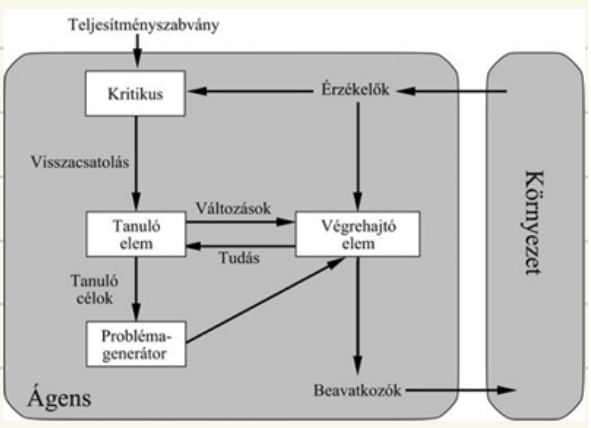
3. **Célorientált ágens:** A szabályok helyett egy konkrét célt próbál elérni, a cselekvések jövőbeli hatásait vizsgálva.



4. **Hasznosság alapú (Utility) ágens:** Nem csak el akar érni egy célt, de azt a leghatékonyabban ("legboldogabban") teszi. A teljesítményt maximalizálja.



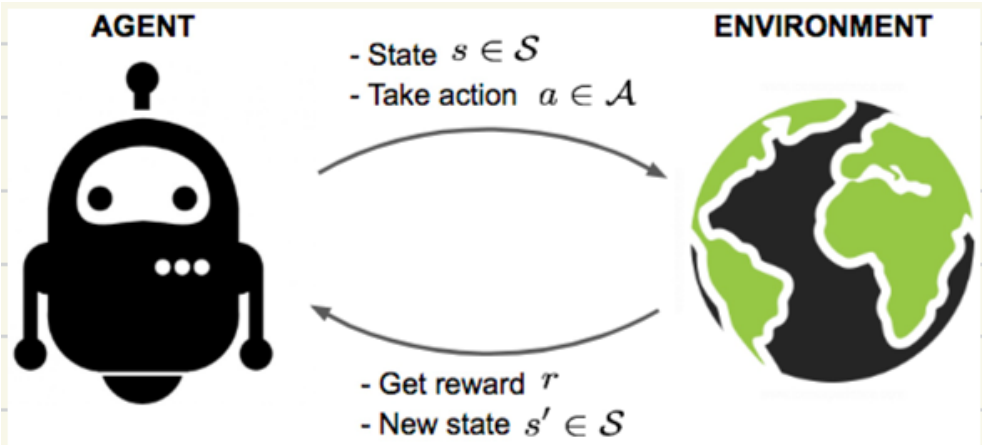
5. **Tanuló ágens:** A teljesítménye idővel javul. Részei: Tanuló elem, Végrehajtó elem, Kritikus (értékeli a sikert), Problémagenerátor (új helyzeteket fedez fel).



Markov Döntési Folyamat (MDP) és Megerősítéses Tanulás (RL)

A Megerősítéses Tanulás célja, hogy az ágens úgy cselekedjen a környezetében, hogy maximalizálja a hosszú távú összejutalmat. Ennek matematikai modellje a Markov Döntési Folyamat (MDP).

- **MDP elemei:** S (Állapotok), A (Cselekvések), R (Jutalom függvény), P (Átmenet-valószínűségek).
- **Markov-tulajdonság:** A következő állapot CSAK a jelenlegi állapottól és cselekvéstől függ, a múlttól nem.
- **Diszkont faktor (γ):** Egy 0 és 1 közötti szám. Azt szabályozza, mennyire fontos a jövőbeli (későbbi) jutalom a jelenlegihez képest.



Tanulási Stratégiák (Passzív vs. Aktív)

- **Passzív ágens:** A stratégiája (policy) rögzített. Csak végrehajtja a parancsokat, és közben próbálja megtanulni az állapotok jóságát (hasznosságát).
- **Aktív ágens:** Nincs rögzített stratégiája. Neki kell eldöntenie, mit lépjen, így meg kell találnia az optimális stratégiát.

Felfedezés vs. Kihasználás (Exploration vs. Exploitation)

Az aktív tanulás alapvető dilemmája:

- **Kihasználás (Exploitation):** Az ágens az eddigi tudása alapján a legjobbnak vélt lépést teszi (hogy sok jutalmat kapjon).
- **Felfedezés (Exploration):** Az ágens szándékosan ismeretlen, nem optimális lépéseket tesz, hogy új információkat (potenciálisan még jobb utakat) gyűjtsön.
- **Epsilon-greedy (ϵ-greedy) módszer:** Az ágens $1 - \epsilon$ valószínűséggel a legjobb lépést teszi (kihasználás), ϵ valószínűséggel pedig véletlenszerűen lép (felfedezés).

Tanulási Algoritmusok (ADP, TD, Q-learning)

1. **Adaptív Dinamikus Programozás (ADP):** Modell alapú. Az ágens megtanulja az átmenet-modellt (P) és a jutalmakat, majd a Bellman-egyenlet megoldásával kiszámolja az állapotok pontos értékét ($V(s)$).
2. **Időbeli különbség tanulás (TD - Temporal Difference):** Modell-mentes. Két egymást követő állapot értéke közötti különbség alapján, lépésről lépésre frissíti a hasznosságértéket (V) a folyamat közben, a tanulási ráta (α) paraméter segítségével.

3. **Q-tanuló ágens (Q-learning):** A TD tanulás egy fejlettebb, aktív változata. Ahelyett, hogy az állapotok értékét (V) tanulná meg, a "Cselekvés-Állapot" párok minőségét (Q-érték) tanulja. Közvetlenül megmondja, hogy egy adott állapotban egy adott lépés mennyire jó, így nem kell hozzá a környezet előzetes modellje.
    - **Képlete:** $Q_{új}(s,a) = Q_{régi}(s,a) + \alpha \cdot \left[R + \gamma \cdot \left(Q\left(s', a'\right) \right) - Q_{régi}(s,a) \right]$

6.1. Gráf fogalma, megadásának módjai, séta, út, összefüggőség és nevezetes gráfok

A gráf fogalma és megadási módjai

A gráf csúcsok (pontok) és élek halmaza. Matematikailag egy $G = (V, E)$ rendezett pár, ahol:

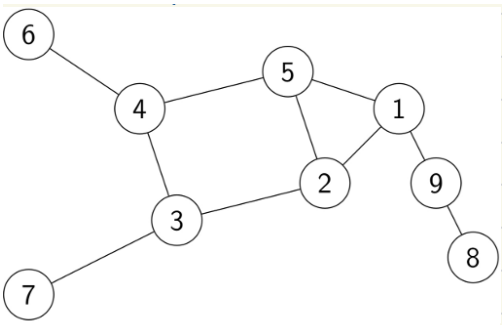
- V (Vertices): A csúcsok véges, nem üres halmaza.
- E (Edges): Az élek halmaza, ahol minden élhez egy vagy két csúcs (a végpontok) tartozik.

Megadási módjai:

1. **Rajz (Ábra):** Pontok és a köztük lévő vonalak.
2. **Szomszédsági lista (Adjacency list):** Minden csúcshoz felsoroljuk, hogy mely más csúcsokkal van összekötve.
3. **Szomszédsági mátrix (Adjacency matrix):** Egy $|V|\left[cite_s tart\right] \times |V|$ méretű mátrix, ahol az i-edik sor j-edik eleme 1, ha van él az i és j csúcs között, egyébként 0.

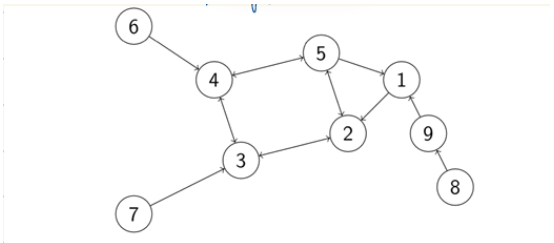
Irányított és Irányítatlan gráfok

- **Irányítatlan gráf:** Az élek halmaza a csúcshalmaz elemeiből alkotott *rendezetlen* párokból áll. (Az élnek nincs iránya, ha A-ból B-be van él, akkor B-ből A-ba is átjárható) .



$G = (V, E)$, ahol $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, és
$E = \{(1, 2), (1, 5), (1, 9), (2, 3), (2, 5), (3, 4), (3, 7), (4, 6), (8, 9)\}$.

- **Irányított gráf:** Az élek halmaza *rendezett* csúcspárokból áll. (Az élnek iránya van, nyíllal jelöljük) .



$G = (V, E)$, ahol
$V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, és
$E = \{(1, 2), (2, 3), (2, 5), (3, 2), (3, 4), (4, 3), (4, 5), (5, 1), (5, 2), (5, 4), (6, 4), (7, 3), (8, 9), (9, 1)\}$.

Séta, Út, Kör és Összefüggőség

- **Séta (Walk):** Csúcsok és élek váltakozó sorozata. Egy séta során csúcsokat és éleket is *többször* is érinthetünk .
    - *Zárt séta:* A kezdő és végpont megegyezik.

- **Vonal (Trail / Egyszerű séta):** Olyan séta, amelyben minden élet *pontosan egyszer* érintünk (de csúcsot érinthetünk többször is).
- **Út (Path):** Csúcsok és élek olyan sorozata, amelyben minden csúcsot (és így minden élet is) *legfeljebb egyszer* érintünk . Hosszát az élek száma adja.
- **Kör (Cycle / Zárt út):** Olyan út, amelynek kezdő és végpontja megegyezik (legalább 3 hosszú, és a kezdő/végpontot leszámítva nem ismétel csúcsot).
- **Összefüggőség:** Egy irányítatlan gráf összefüggő, ha bármely két csúcsa között létezik út .

Nevezetes gráfok

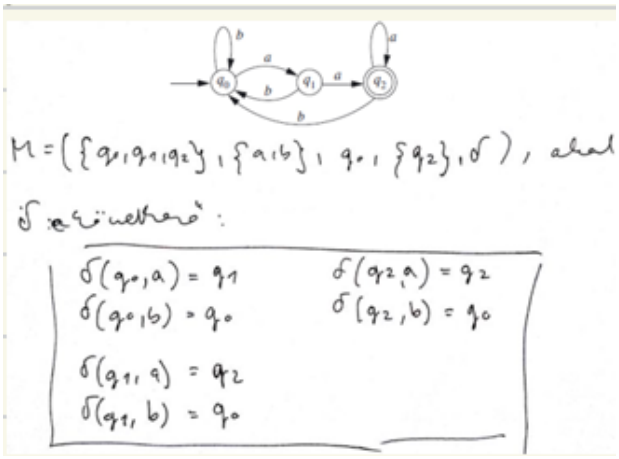
- **Egyszerű gráf:** Bármely két csúcs között legfeljebb egy él fut, és nincsenek benne hurokélek (önmagába visszatérő élek) .
- **Páros gráf (Bipartite graph):** A csúcshalmaz két diszjunkt halmazra bontható úgy, hogy minden él csak a két különböző halmazba tartozó csúcsokat köt össze (halmazon belül futó él nincs) .
- **Teljes gráf:** Bármely két különböző csúcsa között pontosan egy él fut (mindenki mindenkivel össze van kötve) .
- **Fa:** Olyan összefüggő gráf, amely nem tartalmaz kört. Egy n csúcsú fának mindig pontosan $n - 1$ éle van.
- **Súlyozott gráf:** Minden élhez hozzá van rendelve egy számérték (súly/költség) .

6.2. Automaták, Formális Nyelvek és Grammatikák

Determinisztikus Véges Automata (DFA)

A DFA egy matematikai modell, amely véges sok állapotot vehet fel, és egy bemeneti szalagot olvasva hoz döntést (elfogadja a szót vagy nem). Formálisan egy $M = \left(Q, \Sigma, q_0, A, \delta\right)$ ötös , ahol:

- Q: Az állapotok véges halmaza.
- Σ: A bemeneti ábécé.
- $q_0 \in Q$: A kezdőállapot.
- $A \subseteq Q$: Az elfogadó (vég)állapotok halmaza .
- δ: $Q \times \Sigma \rightarrow Q$: Az állapotátmenet függvény (megmondja, hogy adott állapotban, adott betűt olvasva melyik állapotba lépünk).

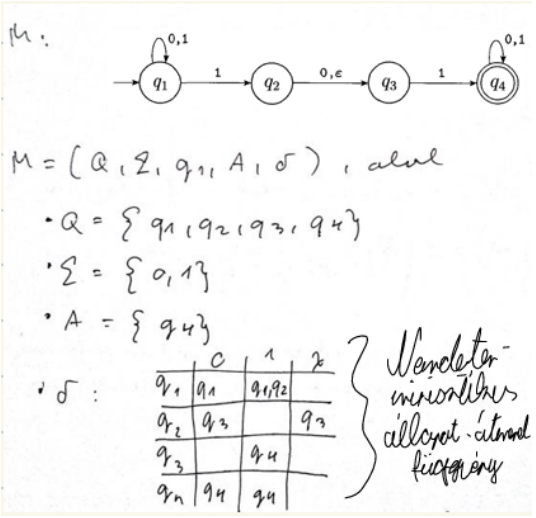


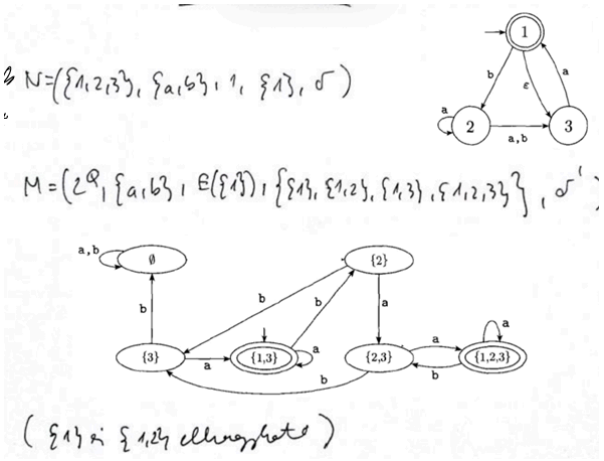
Elfogadott nyelv ($L(M)$): Azon szavak halmaza, amelyek betűit sorban kiolvasva a δ függvény egy elfogadó állapotba (A-ba) visz: $L(M) = \{x \in \Sigma^* | \delta^*(q_0, x) \in A\}$.

Nemdeterminisztikus Véges Automata (NFA)

Az NFA abban különbözik a DFA-tól, hogy egy állapotból egy adott betű hatására *több* különböző állapotba is léphet, illetve létezhet úgynevezett λ (vagy ϵ) átmenet, amellyel betű olvasása nélkül is állapotot válthat.

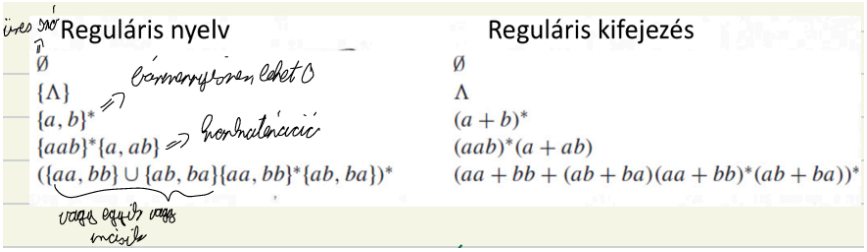
- Átmenetfüggvénye: $\delta: Q \times (\Sigma \cup \{\lambda\}) \rightarrow 2^Q$ (A lehetséges állapotok halmazába képez).
- *Tétel:* Minden NFA-hoz konstruálható egy vele ekvivalens (ugyanazt a nyelvet elfogadó) DFA (részhalmaz-konstrukcióval) .

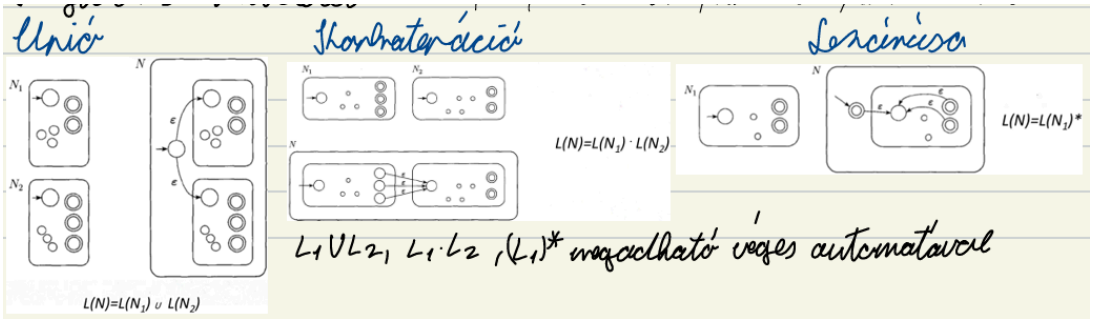




Reguláris Nyelvek és Kifejezések

- **Reguláris műveletek:** Unió (∪), Konkatenáció (·), Kleene-csillag (Iteráció, *) .
- Egy L nyelv pontosan akkor reguláris, ha leírható egy **reguláris kifejezéssel**, vagy ha elfogadja egy véges automata (DFA/NFA) .
- *Zártság:* A reguláris nyelvek halmaza zárt az unióra, metszetre, iterációra és komplementerképzésre .





Pumpálási Lemma reguláris nyelvekre

Ez a lemma szükséges feltétel a regularitáshoz, leggyakrabban arra használjuk, hogy **bebizonyítsuk egy nyelvről, hogy NEM reguláris** (indirekt bizonyítással) .

- *Tétel:* Ha L reguláris nyelv, akkor létezik egy p (pumpálási) hossz, hogy minden $x \in L$, amelyre $|x| \geq p$, felbontható $x = uvw$ alakra úgy, hogy:
  1. $|uv| \leq p$
  2. $|v| > 0$ (a felpumpálandó v rész nem üres)
  3. Minden $i \geq 0$-ra az $uv^i w \in L$ (a v rész tetszőlegesen ismételhető, a szó a nyelvben marad) .

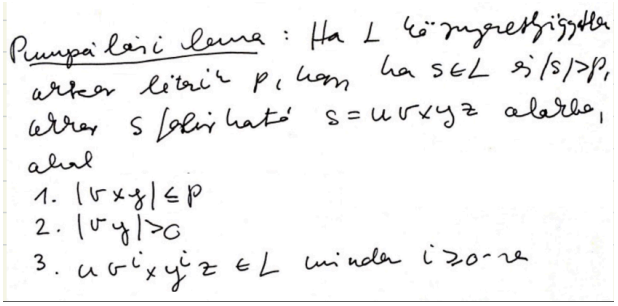
Generatív Grammatikák (Nyelvtanok)

A grammatika szabályokat ad meg, amikkel a nyelv szavait legenerálhatjuk (levezethetjük). $G = (N, \Sigma, S, P)$:

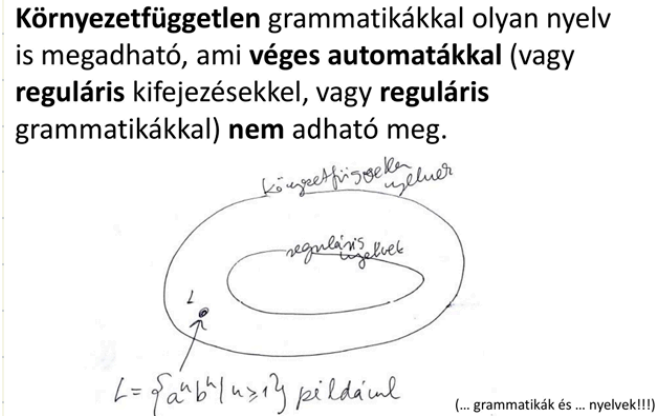
- N: Nemterminálisok halmaza (változók, pl. $S,\ A,\ B$).
- Σ: Terminálisok halmaza (az ábécé tényleges betűi, pl. $a,\ b$).
- $S \in N$: Kezdőszimbólum.
- P: Levezetési szabályok halmaza (pl. $S \to aSb$).

Környezetfüggetlen (Context-Free) Grammatika és Nyelv

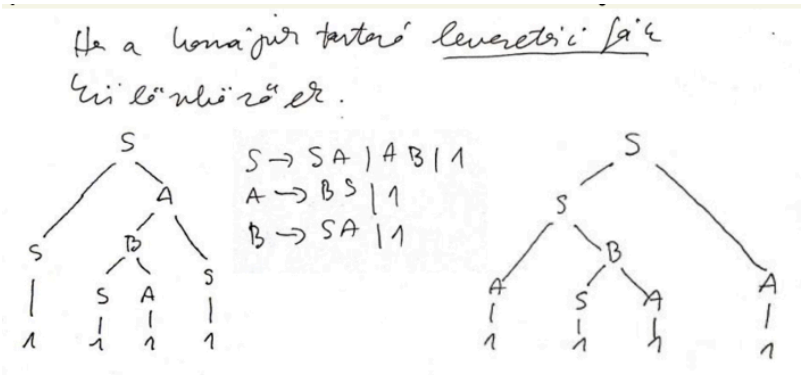
A hierarchiában a reguláris nyelvek felett állnak. Egy grammatika környezetfüggetlen, ha minden P -beli szabály bal oldalán **pontosan egyetlen nemterminális** áll: $A \to \alpha$, ahol $A \in N$ és $\alpha \in (N \cup \Sigma)^*$.



- **Reguláris grammatika:** Olyan környezetfüggetlen grammatika, ahol a szabályok jobb oldalán szigorúan csak egy terminális ($A \rightarrow a$) vagy egy terminális és egy nemterminális állhat ($A \rightarrow aB$) .



- **Egyértelműség:** Egy grammatika egyértelmű, ha minden általa generált szónak csak egyetlen baloldali levezetése (levezetési fája) van .



Chomsky Normálforma (CNF) és a CYK algoritmus

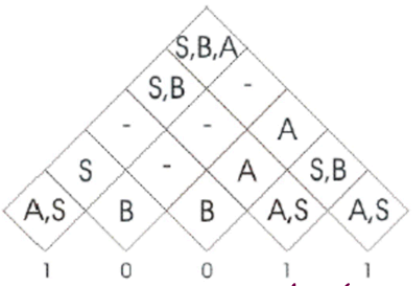
- Egy környezetfüggetlen grammatika Chomsky normálformában van, ha minden szabálya szigorúan a következő két alak valamelyike:

$A \rightarrow BC$ (Két nemterminálisra bomlik)

$A \rightarrow a$ (Egy terminálisra bomlik) .

- Erre az alakra azért van szükség, mert így a **Cocke-Younger-Kasami (CYK) algoritmus** dinamikus programozással, polinomiális időben el tudja dönteni egy tetszőleges w szóról, hogy benne van-e a generált nyelvben .

Tekintsük a következő grammatikát!
$G=(\{S,A,B\},\{0,1\},S,H)$,ahol H szabályai:
$\{S \to SA,\ S \to AB,\ A \to BS,\ B \to SA,\ A \to 1,\ S \to 1,\ B \to 0\}$
Bizonyítsuk be, hogy az 10011 szó benne van a grammatika által generált nyelvben,

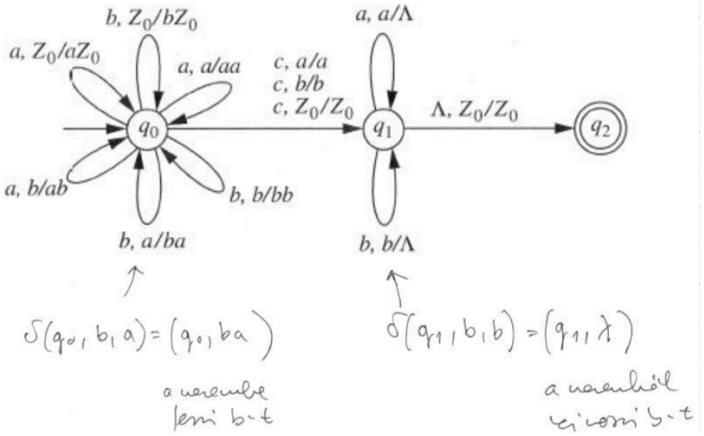


(Megjegyzés: A környezetfüggetlen nyelveknek is van pumpálási lemmája, ahol a felbontás $s = uvxyz$ alakú, és a v és y részeket pumpáljuk egyszerre .)

Veremautomata

Amíg a reguláris nyelveket a véges automaták, addig a környezetfüggetlen nyelveket a veremautomaták fogadják el. A véges automatát kibővítjük egy **LIFO memóriával (veremmel)**.

- Jele: $M = (Q, \Sigma, \Gamma, ..., \delta, F)$.
- Γ: A verem ábécéje.
- $Z_0 \in$: A kezdő veremszimbólum (jelzi a verem alját).
- δ átmenetfüggvény: Nem csak a bemeneti betűt és az aktuális állapotot nézi, hanem leolvassa a **verem tetején lévő szimbólumot is**, majd cserébe állapotot válthat, beolvashatja a betűt, és a verem tetejét kicserélheti egy új karaktersorozatra .



- *Fontos különbség:* Míg a DFA és NFA nyelvosztálya megegyezik (reguláris nyelvek), addig a Determinisztikus Veremautomata gyengébb, mint a Nemdeterminisztikus Veremautomata! A teljes környezetfüggetlen nyelvosztályhoz nemdeterminisztikus veremautomata szükséges .

2. Készítse el az $S \to aBb \mid Aa \mid \ldots$ nyelvtanhoz tartozó LL(1) elemző táblázatot!

	a	b	c
S	—	—	Aa , Bb
A	—	—	cA , c
B	—	—	cB , c

$S \to Aa$

$S \to Bb$

3. Készítse el az $S \to aA \mid bBc,\ A \to Bd \mid Cc,\ B \to e \mid \lambda,\ C \to f \mid \lambda$ nyelvtanhoz tartozó LL(1) elemző táblázatot!
Megoldás

	a	b	c	d	e	f	λ
S	$S \to aA$	$S \to Bc$	—	—	—	—	—
A	—	—	$A \to Cc$	$A \to Bd$	$A \to Bd$	$A \to Cc$	—
B	—	—	—	—	$B \to e$	—	$B \to \lambda$
C	—	—	—	—	—	$C \to f$	$C \to \lambda$

7.1. Lineáris egyenletrendszer fogalma és megoldása Gauss-eliminációval

Lineáris egyenletrendszer fogalma

Egy m darab egyenletből álló, n darab ismeretlent tartalmazó lineáris egyenletrendszer általános alakja:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2$$

$$\ldots$$

$$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m$$

Ahol:

- a_{ij}: az egyenletrendszer ismert együtthatói.
- b_k: a szabad tagok (konstansok).
- x_n: a keresett ismeretlenek .

Mátrixos alak: A fenti rendszer felírható tömören $A \cdot x = b$ alakban, ahol A az együtthatómátrix, x az ismeretlenek vektora, b pedig a szabad tagok vektora. A számolásokhoz a **kibővített mátrixot** (b) használjuk, ahol az A mátrix utolsó oszlopaként hozzácsatoljuk a b vektort .

Az egyenletrendszer alapmátrixa, ill. kibővített mátrixa:

$$A = \begin{pmatrix} a_{11} & \ldots & a_{1n} \\ a_{21} & \ldots & a_{2n} \\ \vdots & & \vdots \\ a_{m1} & \ldots & a_{mn} \end{pmatrix} \in \mathbb{R}^{m \times n} \quad \text{és} \quad (A|b) = \left(\begin{array}{ccc|c} a_{11} & \ldots & a_{1n} & b_1 \\ a_{21} & \ldots & a_{2n} & b_2 \\ \vdots & & \vdots & \vdots \\ a_{m1} & \ldots & a_{mn} & b_m \end{array}\right)$$

Az egyenletrendszer jobboldali vektora és ismeretlen vektora

$$b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} \in \mathbb{R}^m \quad \text{és} \quad x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$$

A lineáris egyenletrendszer mátrixos alakja: $Ax = b$.

$m \times n \quad n \times 1 \quad m \times 1$

Megoldhatóság és a Rang-kritérium

Egy mátrix **rangja** a lineárisan független sorainak (vagy oszlopainak) száma . Egy lineáris egyenletrendszer megoldhatóságát a rangok határozzák meg:

- **Megoldható**, ha $Rang(A) = Rang(b)$. A megoldás a b vektor benne van az A oszlopvektorai által kifeszített térben .
    - *Határozott (Pontosan 1 megoldás):* Ha a rang megegyezik az ismeretlenek számával (n).
    - *Határozatlan (Végtelen sok megoldás):* Ha a rang kisebb, mint az ismeretlenek száma.
- **Nem megoldható (Ellentmondásos):** Ha $Rang(A) \neq Rang(b)$ (vagyis nincs megoldás).

Homogén és Inhomogén rendszerek

- **Homogén:** Ha a szabad tagok vektora nulla ($A \cdot x = 0$). Ennek mindig van legalább egy "triviális" megoldása (amikor minden $x_i = 0$) . Pontosan akkor van a triviálistól eltérő megoldása, ha az A oszlopvektorai lineárisan függőek.
- **Inhomogén:** Ha $b \neq 0$. Az inhomogén egyenletrendszer általános megoldása felírható $x_{ih} = x^* + x_{hom}$ alakban, ahol x^* az inhomogén rendszer egy konkrét (partikuláris) megoldása, x_{hom} pedig a hozzá tartozó homogén rendszer ($A \cdot x = 0$) általános megoldása .

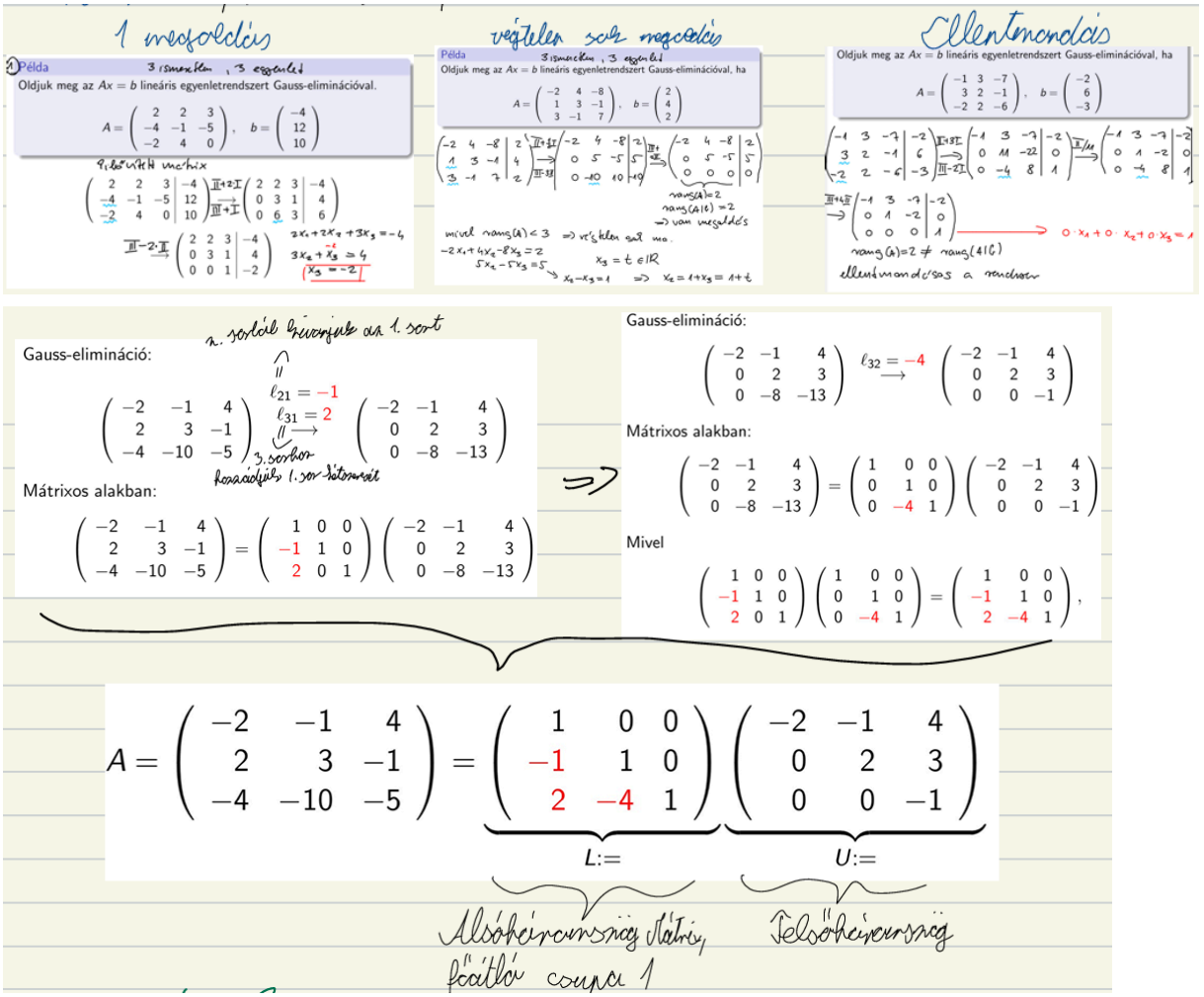
Gauss-elimináció

Olyan algoritmus, amely ekvivalens átalakításokkal (amelyek nem változtatják meg a megoldáshalmazt) az (b) kibővített mátrixot felső háromszögmátrixszá (lépcsős alakra) hozza.

Megengedett (ekvivalens) átalakítások :

1. Egy sor szorzása egy nem nulla skalárral.
2. Egy sorhoz hozzáadhatjuk egy másik sor konstansszorosát.
3. Két sor felcserélése.
4. Csupa nulla sor elhagyása.

A cél: A főátló alatti elemeket nullává tenni. Ha ezt elértük, lentről felfelé haladva, visszahelyettesítéssel könnyedén kiszámolhatók az x_n ismeretlenek. (Megjegyzés: Az LU-felbontás egy sorcsere nélküli Gauss-elimináció, amely az A mátrixot egy alsó (L) és egy felső (U) háromszögmátrix szorzatára bontja, ami nagyon hasznos, ha ugyanazt az A mátrixot több különböző b vektorral is meg kell oldani).



7.2. Turing-gépek, Algoritmikus eldönthetőség és Nyelvosztályok

A Turing-gép fogalma

A Turing-gép egy absztrakt, matematikai számítási modell, a modern számítógépek elméleti alapja. Képes szimulálni bármilyen algoritmikus számítást. Részei egy mindkét irányban végtelen, cellákra osztott szalag, egy író/olvasó fej, és egy vezérlőegység, amely véges sok állapotot vehet fel .

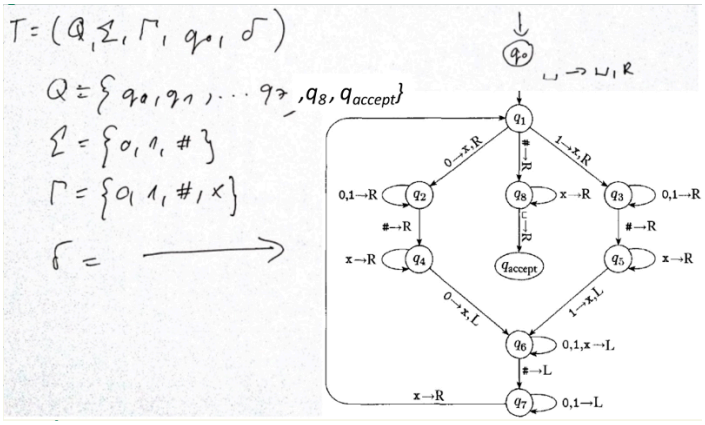
Formálisan egy $T = (Q, \Sigma, \Gamma, q_0, \delta)$ rendszer, ahol :

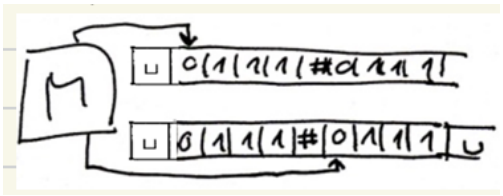
- Q: A véges állapothalmaz (tartalmazza az elfogadó és elutasító állapotokat is).
- Σ: Bemeneti ábécé.
- Γ: Szalag ábécé (tartalmazza az üres cella Δ jelét is, $\Sigma \subseteq \Gamma$).
- q_0: Kezdőállapot.
- δ: Állapotátmenet-függvény (Megmondja, hogy adott állapotban, adott karaktert olvasva milyen állapotba lépjen, mit írjon a szalagra, és merre lépjen a fej: Jobbra/Balra).

Működése: A gép a kezdeti konfigurációból indul. Ha a lépések során elér egy "elfogadó" (accept) állapotot, akkor a gép megáll, és azt mondjuk, hogy a gép **elfogadta** a bemeneti szót . Ha nem fogadja el, akkor vagy egy elutasító állapotban áll meg, vagy végtelen ciklusba esik és sosem áll meg .

Tehát: A T Turing gép által elfogadott nyelv L(T).

- Ha w ∈ L(T), akkor w bemenetre T **elfogadó állapotban áll meg**,
- Ha w ∉ L(T), akkor w bemenetre T **nem-elfogadó állapotban** áll meg, vagy **egyáltalán nem áll meg**.





Church-Turing tézis

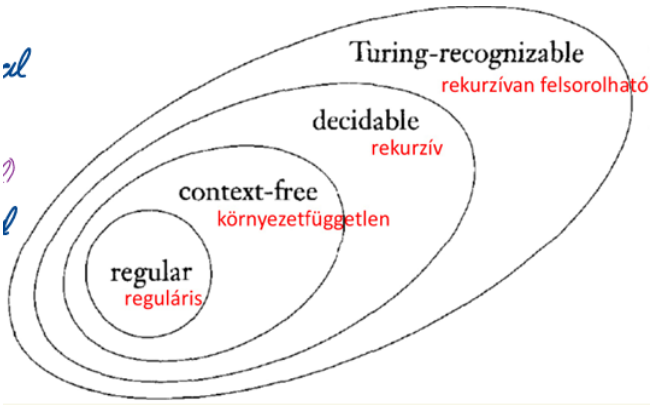
Bármely probléma, amelyre létezik mechanikus (algoritmikus) megoldási eljárás, az megoldható (szimulálható) egy Turing-géppel. Ha egy problémát Turing-géppel nem lehet megoldani, arra nincs semmilyen algoritmus a világon .

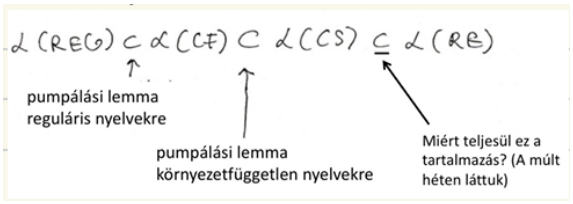
Nyelvosztályok és a Chomsky-hierarchia

A generatív grammatikák (nyelvtanok) és az őket felismerő automaták egy egyre bővülő halmazrendszert (hierarchiát) alkotnak :

1. **Reguláris nyelvek (REG):** Véges automatákkal (DFA, NFA) felismerhető nyelvek .
2. **Környezetfüggetlen nyelvek (CF):** Veremautomatákkal (PDA) felismerhető nyelvek .
3. **Környezetfüggő nyelvek (CS).**
4. **Rekurzív nyelvek (REC / Eldönthető nyelvek):** Létezik olyan Turing-gép, amely a nyelvhez tartozó szavakat elfogadja, a *nem a nyelvhez tartozó szavakra pedig biztosan elutasítással megáll*. Vagyis **mindig megáll** és helyes választ ad .
5. **Rekurzívan felsorolható nyelvek (RE / Felismerhető nyelvek):** Létezik olyan Turing-gép, amely a nyelvhez tartozó szavakon elfogadással megáll, de a nem a nyelvhez tartozó szavakon *lehet, hogy végtelen ciklusba esik* .

Képletesen: $REG \subset CF \subset CS \subset REC \subset RE$.





Algoritmikus eldönthetőség és Eldönthetetlen problémák

Egy *P* eldöntési probléma (ahol a válasz IGEN vagy NEM lehet) **algoritmikusan eldönthető** (megoldható), ha a hozzá tartozó nyelv **rekurzív** (azaz van olyan Turing-gép, ami mindig megáll és kiadja a helyes IGEN/NEM választ) .

Egy probléma algoritmikusan megoldható, ha van olyan Turing, ami **eldönti** a hozzá tartozó nyelvet (mindig megáll és IGEN-t és NEM-et válaszol a megfelelő módon).

Azaz:

Egy probléma algoritmikusan megoldható, ha a hozzá tartozó nyelv **rekurzív**.

Eldönthetetlen problémák (A megállási probléma): Vannak problémák, amelyek bizonyítottan algoritmikusan megoldhatatlanok (a nyelvük csak rekurzívan felsorolható, vagy még az sem). A leghíresebb ilyen a **Megállási probléma (Halting problem)**: Nem létezik olyan algoritmus (Turing-gép), amely egy tetszőleges programról (Turing-gépről) és egy tetszőleges bemenetről előre el tudná dönteni, hogy a program valaha is megáll-e a bemeneten, vagy végtelen ciklusba esik.

8.1. Statisztikai minta, becslések, átlag, szórás. Konfidenciaintervallumok. Az u-próba.

Statisztikai minta és alapfogalmak

A matematikai statisztika célja, hogy a valóságban megfigyelt adatokból (mintából) következtetéseket vonjunk le a teljes sokaságra vonatkozóan.

- **Statisztikai minta:** Az $X_1, X_2, ..., X_n$ független, azonos eloszlású (FAE) valószínűségi változókat n elemű mintának nevezzük.
- **Minta realizációja:** A megfigyelés (kísérlet) során kapott konkrét számértékek sorozata: $x_1, x_2, ..., x_n$.
- **Statisztika:** Bármely olyan függvény, ami a mintából kiszámolható (azaz csak a megfigyelt értékektől függ, ismeretlen paraméterektől nem). Függvénye: $T\left(X_1, X_2, ..., X_n\right)$.

Becslések, Átlag és Szórás

Sokszor a vizsgált jelenség (eloszlás) pontos paramétereit (pl. várható érték, szórás) nem ismerjük, ezeket a mintából kell "megbecsülni". Egy jó becslés elvárása, hogy **torzítatlan** legyen (azaz a becslés várható értéke megegyezzen a valódi paraméterrel).

- **Mintaátlag (Tapasztalati várható érték):**

Az ismeretlen μ (várható érték) torzítatlan becslésére szolgál. A minta elemeinek számtani közepe.

Képlete: $\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$

- **Tapasztalati szórásnégyzet és Szórás:**

Az eloszlás ismeretlen σ^2 (variancia/szórásnégyzet) paraméterét becsüljük vele.

  - *Korrigált tapasztalati szórásnégyzet:* Ez a torzítatlan becslés!

$$s_n^{*2} = \frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2$$

  - *Szórás (s_n):** Ennek a négyzetgyöke adja meg a minta tapasztalati szórását.

6.1.3. Example. Legyen $X_1, \dots, X_n$ minta. Tegyük fel, hogy $\mathbb{E}X_1 = m$ véges. Ekkor $\overline{X}$ az m-nek torzítatlan és konzisztens becslése. Valóban, $\mathbb{E}\overline{X} = m$ nyilvánvaló.

$$\overline{X} = \frac{1}{n}(X_1 + \cdots + X_n) \to m$$

majdnem biztosan teljesül a nagy számok erős törvénye miatt.
Ha $\mathbb{E}X_1^2 < \infty$ és $\sigma^2 = \mathbb{D}^2X_1$, akkor s_n^{*2} a σ^2 torzítatlan és (erősen) konzisztens becslése. Valóban, $\mathbb{E}s_n^{*2} = \sigma^2$-et már korábban láttuk. Továbbá, a Steiner-formulát használva

$$s_n^{*2} = \frac{n}{n-1}\frac{1}{n}\sum_{i=1}^{n}(X_i - m)^2 - \frac{n}{n-1}(\overline{X} - m)^2 \to \sigma^2 - 0 = \sigma^2$$

a nagy számok törvénye miatt.
Szavakban a fenti képletek az alábbit jelentik. Az empirikus közép az ismeretlen várható érték torzítatlan és konzisztens becslése. A korrigált empirikus szórásnégyzet pedig az ismeretlen elméleti szórásnégyzet torzítatlan és konzisztens becslése.

6.1.1. A maximum-likelihood-becslés A *maximum-likelihood elv* szerint az ismeretlen paraméter azon értékét fogadjuk el, amely mellett a bekövetkezett eredmény maximális valószínűségű.

6.1.4. DEFINITION. Legyen $X_1, \ldots, X_n$ minta egy diszkrét eloszlásból, $x_1, \ldots, x_n$ pedig a minta realizáció. Legyen ϑ az ismeretlen paraméter. Az

$$L(x_1, \ldots, x_n; \vartheta) = P(X_1 = x_1, \ldots, X_n = x_n) = \prod_{i=1}^{n} P(X_i = x_i)$$

függvényt *likelihood-függvénynek* nevezzük. Az

$$l(x_1, \ldots, x_n; \vartheta) = \log L(x_1, \ldots, x_n; \vartheta)$$

függvényt pedig *loglikelihood-függvénynek* hívjuk.

A maximum-likelihood elv szerint L-et kellene maximalizálni ϑ szerint. A maximum hely azonban pontosan egybeesik l maximumhelyével, hiszen a természetes alapú logaritmus függvény szigorúan monoton növekvő. Így elegendő az l maximumhelyét meghatározni.

Konfidenciaintervallumok

Nem egyetlen számot (pontbecslést) adunk meg az ismeretlen paraméterre, hanem egy **intervallumot**, amelybe a keresett paraméter egy előre megadott, magas valószínűséggel (úgynevezett $1 - \alpha$ megbízhatósági vagy konfidenciaszinttel, pl. 95%-kal vagy 99%-kal) beleesik.

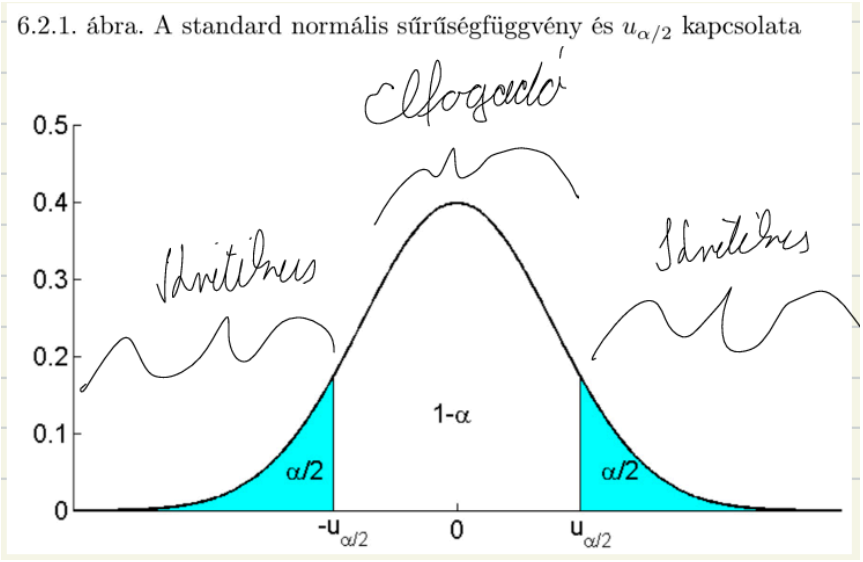
- Például egy 95%-os konfidenciaintervallum azt jelenti, hogy ha nagyon sokszor vennénk mintát és készítenénk egy ilyen intervallumot, az esetek 95%-ában a valódi, ismeretlen paraméter (pl. az országos átlagfizetés) benne lenne ebben az intervallumban.

Statisztikai Próbák: Az u-próba

Hipotézisvizsgálat során egy előzetes feltevést (Nullhipotézis, H_0) tesztelünk az adatok alapján az Ellenhipotézissel (H_1) szemben.

Az **u-próba** (vagy Z-próba) egy konkrét statisztikai teszt, amelyet arra használunk, hogy egy **normális eloszlású**, **ISMERT szórású** sokaság várható értékét (μ) megvizsgáljuk.

- *Nullhipotézis (H_0):* $\mu = \mu_0$ (A várható érték egyenlő egy feltételezett értékkel).
- *Próbastatisztika:* $u = \frac{\bar{X} - \mu_0}{\sigma/\sqrt{n}}$
- *Döntés:* Kiszámoljuk az u értéket a mintából. Ha ez az érték beleesik a kritikus tartományba (ami a megadott α szignifikanciaszinttől függ, pl. az $|u| > 1,96$ a 95%-os szinthez), akkor a nullhipotézist elvetjük.

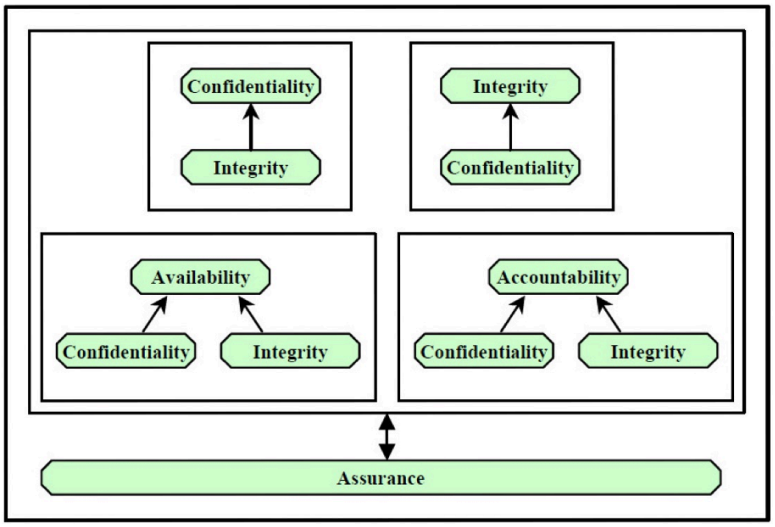


- **8.2. Informatikai biztonság, kártékony programok, algoritmikus védelem**

Az informatikai biztonság fogalma és a CIA-triád

Az információbiztonság célja az adatok és rendszerek védelme. A három legfontosabb biztonsági cél (CIA-triád):

1. **Confidentiality (Bizalmasság):** Csak az férhet hozzá az információhoz, akinek jogosultsága van rá.
2. **Integrity (Sértetlenség):** Az adatokat nem módosíthatják jogosulatlanul; garantálni kell, hogy az információ pontos és teljes.
3. **Availability (Rendelkezésre állás):** A jogosult felhasználók számára az adatok és szolgáltatások mindig, megszakítás nélkül elérhetők legyenek. +1. **Non-repudiation (Letagadhatatlanság):** A küldő nem tagadhatja le, hogy ő küldte az üzenetet.



Fizikai védelem

A hardverek, hálózatok és épületek védelme a fizikai behatolás, lopás vagy természeti katasztrófák ellen. (Eszközei: biztonsági őrség, zárak, kamerák, beléptető rendszerek, tűzvédelem, szünetmentes tápegységek).

Kártékony programok (Malware) és osztályozásuk

Olyan szoftverek, amelyeket informatikai rendszerek megzavarására, adatgyűjtésre vagy károkozásra terveztek.

Terjedési mód szerint:

- **Vírus (Virus):** Olyan kód, amely egy létező, futtatható fájlhoz (gazdaprogramhoz) kapcsolódik. A terjedéséhez emberi interakció (pl. a fertőzött program elindítása) szükséges.
- **Féreg (Worm):** Önálló program, amely emberi beavatkozás nélkül képes másolni és terjeszteni magát a hálózaton keresztül (gyakran biztonsági réseket kihasználva).
- **Trójai faló (Trojan):** Hasznosnak, ártalmatlannak álcázott szoftver, amely a háttérben kártékony funkciókat hajt végre. Nem másolja önmagát.

Büntető rutin (Payload / Károkozási mód) szerint:

- **Zsarolóvírus (Ransomware):** Titkosítja a felhasználó adatait, és a feloldókulcsért cserébe váltságdíjat követel.
- **Kémprogram (Spyware):** Titokban adatokat (jelszavakat, bankkártyaadatokat, böngészési szokásokat) gyűjt a felhasználóról (pl. Keylogger - billentyűzetleütés-rögzítő).
- **Rootkit:** A rendszer mélyén (akár kernel szinten) elrejtőző kártevő, ami teljes adminisztrátori (root) jogot biztosít a támadónak, miközben elrejti a saját folyamatait a vírusirtók elől.
- **Backdoor (Hátsó ajtó):** Egy titkos belépési pont a rendszerbe, amely megkerüli a normál hitelesítési mechanizmusokat.

Algoritmikus védelem: Titkosítás, Hash és Digitális Aláírás

A szoftveres/matematikai védekezés alapkövei a kriptográfiai eljárások.

1. Hash függvények (Lenyomatok): Tetszőleges hosszúságú bemeneti adatból egy fix hosszúságú (pl. 256 bites) bitsorozatot ("ujjlenyomatot") generálnak.

- *Tulajdonságai:* Egyirányú (a lenyomatból nem állítható vissza az eredeti adat), és ütközésmentes (két különböző fájlnak nem lehet ugyanaz a hash-e).
- *Célja:* A jelszavak biztonságos tárolása és az adatok **Sértetlenségének** ellenőrzése.

2. Titkosítás (Encryption):

- **Szimmetrikus titkosítás:** Ugyanazt az egyetlen titkos kulcsot használják a szöveg titkosítására és visszafejtésére is.
    - *Előnye:* Nagyon gyors, nagy adatmennyiség titkosítására alkalmas.
    - *Hátránya:* A kulcsot biztonságosan meg kell osztani a két fél között (kulcselosztási probléma).
    - *Példa algoritmus:* **AES** (Advanced Encryption Standard) – Jelenleg a legelterjedtebb, blokkalapú, szimmetrikus algoritmus, amit bankok és kormányok is használnak.

      AES (Advanced Encryption Standard)

      Az AES jelenleg a világ legelterjedtebb szimmetrikus titkosítási szabványa, amelyet kormányok, bankok és katonai szervezetek is használnak.

- **Típus:** Szimmetrikus kulcsú titkosítás.
- **Működési elv:** Blokktitkosító algoritmus, amely az adatokat fix méretű, 128 bites blokkokban kezeli.
- **Kulcshasználat:** Ugyanazt a titkos kulcsot használja a titkosításhoz és a visszafejtéshez is.
    - Képlete: $C = E_k(P) \text{és} P = D_k(C)$, ahol k a közös titkos kulcs.
- **Kulcshossz:** Választhatóan 128, 192 vagy 256 bit.
- **Jellemzők:**
    - **Gyorsaság:** Rendkívül gyors, nagy adatmennyiség (pl. merevlemezek, fájlok) titkosítására kiválóan alkalmas.
    - **Biztonság:** Eddig nem ismertek hatékony támadások ellene; a 256 bites verzió még a kvantumszámítógépekkel szemben is ellenállónak tekinthető.
- **Fő hátránya:** A **kulcselosztási probléma**. Mivel mindkét félnek ugyanazzal a kulccsal kell rendelkeznie, a kulcs biztonságos célba juttatása nehézkes egy nyilvános hálózaton keresztül.

- **Aszimmetrikus (Nyilvános kulcsú) titkosítás:** Minden félnek két kulcsa van: egy *Nyilvános* (amit bárki ismerhet) és egy *Privát* (amit szigorúan titokban kell tartani). Amit az egyik kulccsal titkosítanak, azt csak a másikkal lehet visszafejteni.

	Szimmetrikus	Aszimmetrikus
Kulcsok titkossága	kulcsok titkosak (K)	(PK, SK) nyilvános (public) és titkos (secret) kulcs
Kulcsok kezelése	kulcscsere algoritmusok	Nyilvános Kulcs Infrastruktúra (Public Key Infrastructure)
Időigény	gyors algoritmusok	lassú algoritmusok
Üzenetek mérete	nagy méretű	kis méretű
Példák	TDES, AES	RSA, ElGamal, elliptikus görbe titkosítás

    - *Példa:* Ha Aliz titkos üzenetet küld Bobnak, akkor Bob *nyilvános* kulcsával titkosítja. Ezt az üzenetet ezután kizárólag Bob tudja elolvasni a saját *privát* kulcsával.
    - *Példa algoritmus:* **RSA** (Rivest-Shamir-Adleman) – Működése a nagy prímszámok szorzatának felbontási nehézségén (faktorizáció) alapul. Mivel lassú, általában nem az egész üzenetet, csak a szimmetrikus kulcsokat titkosítják vele.

RSA (Rivest-Shamir-Adleman)

Az RSA az aszimmetrikus (nyilvános kulcsú) kriptográfia úttörője és legfontosabb algoritmusa.

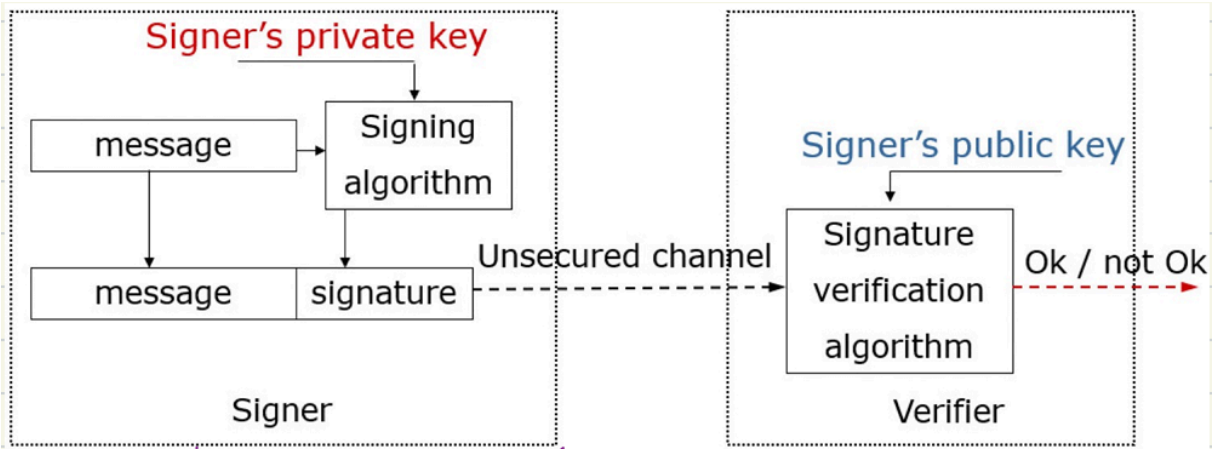
- **Típus:** Aszimmetrikus titkosítás.
- **Kulcspár:** Minden felhasználó két kulccsal rendelkezik:
    1. **Nyilvános kulcs (e, n):** Bárki számára elérhető, ezzel titkosítják az üzenetet.

2. **Privát kulcs (d, n):** Csak a tulajdonos ismeri, ezzel fejti vissza a titkosított szöveget.

- **Matematikai alap:** A **nagy számok faktorizációjának** (tényezőkre bontásának) nehézségén alapul. Két hatalmas prímszám (p és q) szorzatát ($n = p \cdot q$) könnyű kiszámolni, de a szorzatból visszafejteni az eredeti prímeket gyakorlatilag lehetetlen ésszerű időn belül.
- **Folyamat:**
  1. Választunk két nagy prímet: p, q
  2. Kiszámítjuk: $n = p \cdot q$
  3. Titkosítás: $C = M^e \ (mod\ n)$
  4. Visszafejtés: $M = C^d \ (mod\ n)$
- **Jellemzők:**
  - **Lassúság:** Sokkal lassabb, mint az AES, ezért közvetlenül nem használják nagy fájlok titkosítására.
  - **Felhasználás:** Főként **digitális aláírásokhoz** (hitelesítés) és **szimmetrikus kulcsok (pl. AES kulcs) átküldésére** használják.
    - **Biztonság:** A biztonság a kulcs hosszától függ (ma a 2048 vagy 4096 bit az elvárt).

3. Digitális Aláírás: Az aszimmetrikus kriptográfia fordított felhasználása a hitelesség és a **letagadhatatlanság** biztosítására.

- Működése: A feladó a dokumentum Hash-lenyomatát titkosítja a saját *Privát* kulcsával. Ez a digitális aláírás. Bárki ellenőrizheti ezt a feladó *Nyilvános* kulcsával: ha a dekódolt Hash megegyezik a dokumentum tényleges Hash-ével, akkor biztos, hogy a dokumentumot a feladó írta alá (hiszen csak neki van meg a privát kulcs), és azóta nem is módosították azt.



Informatikai ismeretek:

Adatbázisrendszerek

1.1. Alapfogalmak

- **Adatbázis (Database - DB):** Logikailag összefüggő, egyedeket és a köztük lévő kapcsolatokat leíró, belső jelentéssel bíró adatok összessége. A valós világ egy részének (minimodelljének) számítógépes leképezése.
- **Adatbázis-kezelő rendszer (DBMS - Database Management System):** Olyan speciális szoftverrendszer (vagy szoftvercsomag), amely lehetővé teszi a felhasználók számára az adatbázisok létrehozását, karbantartását, lekérdezését és a hozzáférés-szabályozást (pl. MySQL, Oracle, PostgreSQL).
- **Adatbázisrendszer:** Az adatbázis (a tényleges adatok) és a DBMS (a szoftver) együttesét, valamint a futtató hardvert és a felhasználókat együttesen adatbázisrendszernek nevezzük.

1.2. Koncepcionális adatbázis-tervezés és az ER modell

A koncepcionális tervezés célja a valós világ követelményeinek egy magas szintű, implementációfüggetlen modellbe való átültetése. Ennek legelterjedtebb eszköze az **Egyed-Kapcsolat (Entity-Relationship - ER) modell**.

Az ER modell elemei:

1. **Egyed (Entity):** A valós világ egy önállóan létező, másoktól megkülönböztethető dolga vagy fogalma (pl. *Dolgozó*, *Osztály*). ER diagramon **téglalappal** jelöljük.
2. **Tulajdonság (Attribútum):** Az egyedeket leíró jellemzők (pl. *Név*, *Fizetés*). ER diagramon **ellipszissel** jelöljük.
    - *Kulcs attribútum:* Egyértelműen azonosítja az egyedet (aláhúzva jelöljük).
    - *Többértékű attribútum:* Egy egyedhez több érték is tartozhat (pl. *Telefonszámok* - dupla ellipszis).
    - *Származtatott attribútum:* Más adatokból kiszámolható (pl. *Életkor* a születési dátumból - szaggatott ellipszis).
3. **Kapcsolat (Relationship):** Két vagy több egyed közötti logikai viszony (pl. *Dolgozik_rajta*). ER diagramon **rombusszal** jelöljük.

	absztrakt	konkrét
egyed	egyedtípus	egyed-előfordulás
tulajdonság	tulajdonságtípus	tulajdonság-előfordulás
kapcsolat	kapcsolattípus	kapcsolat-előfordulás

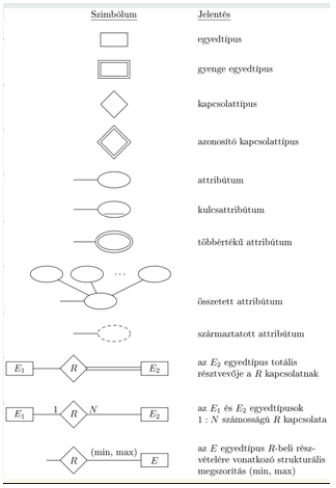
Kapcsolatok számossága (Kardinalitás):

- **1:1 (Egy-az-egyhez):** Egy A egyedhez pontosan egy B egyed tartozik, és fordítva (pl. *Osztály* és *Vezető*).
- **1:N (Egy-a-többhöz):** Egy A egyedhez több B egyed tartozhat, de egy B egyedhez csak egy A (pl. *Osztály* és *Dolgozó*).

- **M:N (Több-a-többhöz):** Egy A egyedhez több B egyed is tartozhat, és fordítva (pl. *Dolgozó* és *Projekt*).

1.3. Az ER modell leképezése Relációs modellre

Ahhoz, hogy az ER modellből fizikai adatbázis legyen, relációs sémává (táblákká) kell alakítani a következő szabályok szerint:



1. **Egyedek leképezése:** Minden erős (önálló) egyedből egy tábla (reláció) lesz. Az egyed attribútumai lesznek a tábla oszlopai, a kulcs attribútum lesz az **Elsődleges kulcs (Primary Key - PK)**.
2. **1:1 kapcsolat leképezése:** Az egyik tábla elsődleges kulcsát beletesszük a másik táblába **Idegen kulcsként (Foreign Key - FK)**. Érdemes abba tenni, amelyik a kapcsolatban kötelezően részt vesz.
3. **1:N kapcsolat leképezése:** A kapcsolat "több" (N) oldalán lévő táblába betesszük az "egy" (1) oldal táblájának elsődleges kulcsát idegen kulcsként.
4. **M:N kapcsolat leképezése:** Ehhez kötelező létrehozni egy **új kapcsolótáblát**. Ennek a táblának az elsődleges kulcsa a két összekapcsolt tábla elsődleges kulcsaiból fog állni (összetett kulcs).

1.4. Adatbázisok típusai / jellemzése

- **Relációs adatbázis:** Az adatok kétdimenziós táblákban (relációkban) helyezkednek el. A táblák sorokból (rekordok/n-esek) és oszlopokból (attribútumok) állnak. Szigorú sémával és relációkkal (PK, FK) rendelkezik. Matematikai alapja a relációs algebra.
- **Objektum-relációs adatbázis:** A hagyományos relációs modell kiterjesztése objektumorientált fogalmakkal. Támogatja a komplex adattípusokat (pl. tömbök), az öröklődést, és lehetővé teszi metódusok (függvények) tárolását az adatbázisban.
- **NoSQL (Not Only SQL) adatbázis:** Nem relációs elvű adatbázisok, amelyek rugalmas (schema-less) adatmodellt használnak. Kifejezetten a horizontálisan skálázhatóságra és a "Big Data" kezelésére lettek kitalálva.
    - *Típusai:* Dokumentum alapú (pl. MongoDB - JSON dokumentumok), Kulcs-érték tárak (pl. Redis), Gráf alapúak (pl. Neo4J), Oszlopcsalád alapúak (pl. Cassandra).

```
{
  "lastVisit":1324669989288,
  "user":{
    "customerId":"91cfdf5bcb7c",
    "name":"Márton Ispány",
    "countryCode":"HUN",
    "tzOffset":"CET"
  }
}
```

1.5. A Funkcionális függés fogalma

A relációs adatbázis-tervezés (és a normalizálás) matematikai alapja.

- **Definíció:** Egy R relációban az Y attribútumhalmaz funkcionálisan függ az X attribútumhalmaztól (jelölése: $X \rightarrow Y$), ha az X értéke egyértelműen meghatározza az Y értékét.
- **Jelentése a gyakorlatban:** Ha egy táblában két sornak megegyezik az X értéke, akkor kötelezően meg kell egyeznie az Y értéküknek is. (Például: Személyi_szám → Név, Születési_idő). Ha az X egyértelműen meghatározza a tábla összes többi oszlopát, akkor X egy **szuperkulcs**.

1.6. Az SQL elemei és lekérdezések

Az SQL (Structured Query Language) a relációs adatbázisok szabványos nyelve. Három fő részre oszlik:

1. DDL (Data Definition Language - Adatdefiníciós nyelv)

Az adatbázis szerkezetének (sémájának, tábláinak, nézeteinek) létrehozására és módosítására szolgál.

- CREATE: Új tábla/adatbázis/index létrehozása. (pl. CREATE TABLE Dolgozo (...))
- ALTER: Meglévő struktúra módosítása (pl. oszlop hozzáadása).
- DROP: Tábla vagy adatbázis végleges törlése.

2. DML (Data Manipulation Language - Adatmanipulációs nyelv)

A táblákban lévő tényleges adatok (sorok) kezelésére, lekérdezésére és módosítására szolgál.

- SELECT: Adatok lekérdezése (A relációs algebra **szelekció (σ)** és **projekció (π)** műveleteinek felel meg).
- INSERT: Új sor (rekord) beszúrása.
- UPDATE: Meglévő adatok módosítása.
- DELETE: Sorok törlése a táblából.

3. DCL (Data Control Language - Adatvezérlő nyelv)

A jogosultságok és a hozzáférés szabályozására szolgál.

- GRANT: Jogosultság (pl. olvasási, írási jog) adása egy felhasználónak.
- REVOKE: Megadott jogosultság visszavonása.

Egyszerű lekérdezések és Táblák összekapcsolása (JOIN)

Egy alapvető lekérdezés szerkezete:

```
SQL
SELECT oszlop1, oszlop2   -- Projekció (mely oszlopokat kérjük)
FROM tabla                -- Melyik táblából
WHERE  feltétel;                  -- Szelekció (milyen feltételnek
megfelelő sorokat)
```

Táblák összekapcsolása: Mivel az adatok normalizált formában, több táblában vannak szétszórva, a JOIN műveletekkel (relációs algebrai Descartes-szorzat és szelekció kombinációjával) tudjuk őket újra egyesíteni a kulcsok (PK-FK) mentén.

- INNER JOIN: Csak azokat a sorokat adja vissza, amelyeknek mindkét táblában van párjuk a megadott feltétel alapján.
- LEFT (OUTER) JOIN: Visszaadja a bal oldali tábla **összes** sorát, és a jobb oldali táblából azokat, amik illeszkednek. (Ahol nincs pár, ott NULL értékek lesznek).
- RIGHT (OUTER) JOIN: Ugyanaz, csak a jobb oldali tábla minden sorát adja vissza.
- FULL (OUTER) JOIN: Visszaadja mindkét tábla összes sorát, párosítva, ahol lehet, máshol NULL-al kitöltve.

Programozási alapfogalmak

2.1. Adattípusok és Változók

Adattípus fogalma

Az adattípus egy programozási koncepció, amelyet három fő dolog határoz meg :

1. **Értékkészlet (Tartomány):** Milyen értékeket vehet fel (pl. egész számok halmaza).
2. **Műveletek halmaza:** Milyen operációkat végezhetünk rajta (pl. összeadás, szorzás).
3. **Reprezentáció:** Hogyan tárolódik a memóriában (pl. 32 biten, kettes bókoló alakban).

Csoportosításuk:

- **Beépített (Primitív) típusok:** * *Egyszerű típusok:* Numerikus (int, float), Karakteres (char, string), Logikai (bool) .
    - *Összetett típusok:* Vektor (tömb), pointer, felsorolás (enum) .
- **Felhasználói típusok:** Osztályok (class), struktúrák (struct), amiket a programozó hoz létre.

Változó

A változó egy névvel azonosított memóriaterület, amely az adatok tárolására szolgál . **Egy változó 6 fő jellemzője:**

1. **Név (Azonosító):** Ezzel hivatkozunk rá a kódban.
2. **Cím (L-value):** Hol helyezkedik el a memóriában.
3. **Típus:** Meghatározza a méretét és a rajta végezhető műveleteket.
4. **Érték (R-value):** A memóriacímen tárolt tényleges adat.
5. **Élettartam (Lifetime):** Az az időtartam a program futása során, amíg a változó a memóriában létezik (pl. egy függvény futása alatt).
6. **Hatáskör (Scope):** A programkód azon része, ahol a változó neve érvényes és hivatkozni lehet rá.

2.2. Műveletek, Operátorok, Kifejezések és Utasítások

- **Operátor (Műveleti jel):** A műveletet szimbolizálja (pl. +, -, ==, &&).
- **Operandus:** Az az adat (változó vagy konstans), amin a műveletet elvégezzük.
    - *Unáris:* 1 operandusa van (pl. i++, -x).
    - *Bináris:* 2 operandusa van (pl. a + b).
    - *Ternáris:* 3 operandusa van (pl. feltételes operátor: feltétel ? igaz_ág : hamis_ág).
- **Kifejezés:** Operátorok és operandusok szabályosan felépített sorozata, amelynek kiértékelése egy **értéket** eredményez. A kiértékelést két dolog szabályozza:
    - **Precedencia:** A műveletek végrehajtási sorrendje (kötési erősség, pl. a szorzás erősebb az összeadásnál).
    - **Asszociativitás:** Azonos precedenciájú műveletek esetén a kiértékelés iránya (általában balról jobbra, de pl. az értékadás jobbról balra köt).

- **Utasítás:** A program egy önálló, végrehajtható egysége (pl. értékadó utasítás, elágazás, függvényhívás). Nem feltétlenül ad vissza értéket, inkább egy cselekvést hajt végre.

2.3. Vezérlési szerkezetek és Blokk

A vezérlési szerkezetek határozzák meg az utasítások végrehajtásának sorrendjét. (A Böhm-Jacopini tétel kimondja, hogy minden algoritmus megírható 3 alapvető szerkezettel).

1. **Szekvencia:** Az utasítások egyszerű, egymás utáni (sorrendi) végrehajtása.
2. **Szelekció (Elágazás):** Feltétel alapján dől el, hogy melyik kódág fut le.
    - *Egy-/kétirányú:* if / if-else.
    - *Többirányú:* switch-case.
3. **Iteráció (Ciklus):** Egy utasításblokk ismételt végrehajtása.
    - *Elöltesztelős:* Először vizsgálja a feltételt, lehet, hogy egyszer sem fut le (while).
    - *Hátultesztelős:* Először lefut a mag, majd utána tesztel. **Legalább egyszer** biztosan lefut! (do-while).
    - *Számlálós (diszkrét):* Előre ismert az ismétlések száma (for).

- **Blokk:** Utasítások sorozata, amelyeket egy logikai egységbe foglalunk (pl. { és } jelek közé C-szerű nyelvekben). A blokk egyben új hatáskört is nyit.

2.4. Hatáskörkezelés és Láthatóság

- **Hatáskör (Scope):** A programkód azon lexikális szakasza, ahol egy adott azonosítóhoz (névhez) a deklarációja érvényes.
    - *Fajtái:* Lokális (csak a blokkon/függvényen belül él), Globális (mindenhonnan elérhető), Osztályszintű.
- **Láthatóság:** Ahol a változó egyértelműen elérhető a nevén keresztül.
    - *Árnyékolás (Shadowing):* Ha egy belső blokkban deklarálunk egy ugyanilyen nevű változót, az eltakarja (árnyékolja) a külső változót; a hatásköre ott van, de nem látható.

2.5. Programegységek, Paraméterátadás

- **Programegység / Alprogram:** Kód egy logikailag elkülönített, újrahasználható része. Lehet **Függvény** (van visszatérési értéke) vagy **Eljárás** (nincs visszatérési értéke, pl. void C-ben).
- **Formális paraméter:** Az a változó, ami az alprogram definíciójában (fejlécében) szerepel.
- **Aktuális paraméter:** Az az érték (kifejezés), amit az alprogram *hívásakor* konkrétan átadunk.

Paraméterátadási módok :

1. **Érték szerinti (Call by Value):** Az aktuális paraméter értéke *lemásolódik* a formális paraméterbe. A függvény a másolaton dolgozik, az eredeti változó nem módosul .
2. **Cím szerinti (Call by Reference / Pointer):** A változó memóriacíme adódik át. A függvény közvetlenül az eredeti változót manipulálja, így a változás tartós marad .
3. **Eredmény szerinti:** Csak "kifelé" kommunikál, a függvény futása végén az eredmény bemásolódik a megadott változóba .

4. **Név szerinti (Macro):** Szöveges helyettesítés híváskor (inkább történelmi jelentőségű vagy makróknál használatos) .

2.6. Absztrakt Adattípus (ADT)

Egy olyan adattípus, amely megvalósítja az **adatbezárást (encapsulation)** és az **információrejtést (information hiding)** .

- Lényege: Szétválasztja az *interfészt* (mit lehet csinálni az adattal - a publikus metódusok) és az *implementációt* (hogyan van ez belsőleg leprogramozva és tárolva) .
- A felhasználó csak a definiált műveleteken (függvényeken) keresztül férhet hozzá az adatokhoz, közvetlenül a memóriareprezentációhoz nem (pl. privát adattagok egy osztályban). Ezzel garantálható az adatok konzisztenciája és biztonsága.

2.7. Kivételkezelés (Exception Handling)

A futási idejű (runtime) hibák elegáns és strukturált kezelésére szolgáló nyelvi mechanizmus.

- **Probléma:** Ha kivételkezelés nélkül futunk hibára (pl. nullával osztás, nem létező fájl megnyitása), a program azonnal "eláll" (összeomlik).
- **Megoldás:** A gyanús kódrészt betesszük egy **védett blokkba** (try). Ha itt hiba történik, a rendszer "eldob" egy kivétel-objektumot (throw). A program futása megszakad a try blokkban, és a vezérlés átkerül a **hibakezelő blokkba** (catch), ami a hibát lekezeli (pl. hibaüzenetet ír ki), majd a program futása normálisan folytatódik.
- *Opcionális rész:* A finally blokk mindenképpen lefut (akár volt hiba, akár nem), ideális a megnyitott erőforrások (pl. fájlok, adatbázis kapcsolatok) biztonságos lezárására.

Az objektumorientált paradigma alapfogalmai

3.1. Az OOP paradigma, Osztály, Objektum és Példányosítás

- **Objektumorientált paradigma (OOP):** Olyan programozási megközelítés (módszertan), amely a valós világot egymással kommunikáló objektumok halmazaként modellezi. A fókusz a funkciók helyett az adatokon (és az azokon végezhető műveleteken) van.
- **Osztály (Class):** * Egy elvont "tervrajz" vagy sablon (felhasználói típus).
    - Két fő részből áll: **Adattagokból** (tulajdonságok/állapot) és **Metódusokból** (viselkedés/függvények).
    - Önmagában nem foglal memóriát az adatoknak, csak leírja, hogy az ezen alapuló objektumok hogyan fognak kinézni.
- **Objektum (Példány / Instance):**
    - Az osztály alapján memóriában létrejött, futásidőben létező konkrét entitás.
    - Minden objektumnak saját *állapota* van (az adattagok aktuális értékei), de a *viselkedésükön* (metódusokon) osztoznak az osztály többi példányával.
- **Példányosítás (Instantiation):**
    - Az a folyamat, amikor az osztályból egy konkrét objektumot hozunk létre (memóriát foglalunk neki).
    - Ezt a feladatot a **Konstruktor** nevű speciális metódus végzi el, amely inicializálja az objektum alapállapotát. (Általában a new kulcsszóval történik).

3.2. A Bezárási eszközrendszer (Encapsulation)

Az adatok elrejtésének (information hiding) és egységbe zárásának elve.

- **Célja:** Az objektum belső állapota rejtve maradjon a külvilág elől, és azt csak ellenőrzött módon, a publikus metódusokon (Getterek és Setterek) keresztül lehessen módosítani vagy lekérdezni. Ez védi az adatok konzisztenciáját (pl. ne lehessen egy ember életkora negatív).

Láthatósági (hozzáférési) módosítók:

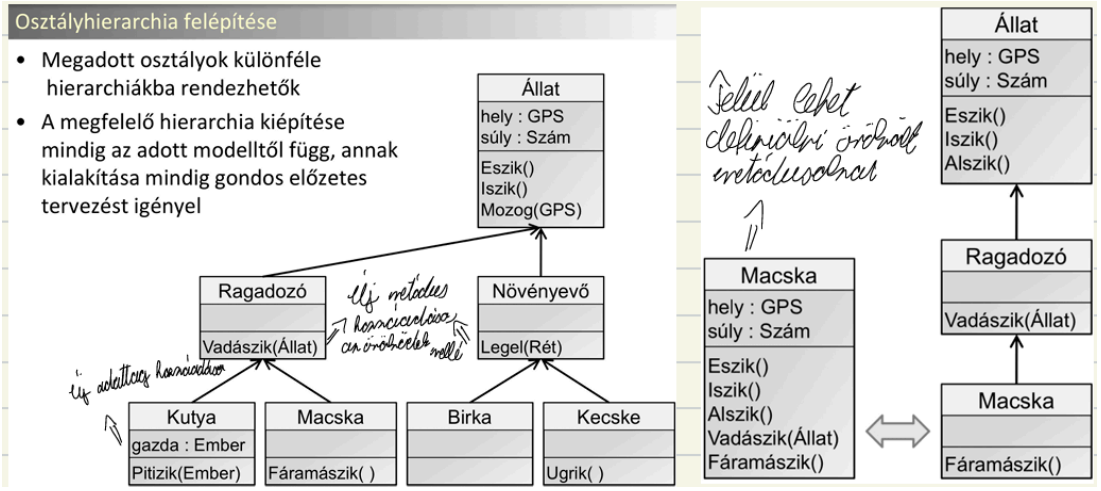
1. **Private (Privát):** A tag csak és kizárólag az adott osztályon *belülről* érhető el. Ez a bezárás alapja.
2. **Public (Publikus):** A tag bárhonnan, bármilyen más osztályból elérhető (ez adja az osztály "interfészét", a külvilág felé mutatott arcát).
3. **Protected (Védett):** A tag elérhető az osztályon belül, ÉS az ebből az osztályból származó (öröklő) gyermekosztályokban is (de kívülről nem).

3.3. Öröklődés (Inheritance) és Osztályhierarchia

- **Öröklődés:** Olyan mechanizmus, amellyel egy meglévő osztályból (Ős/Szülő osztály) egy új osztályt (Gyermek/Leszármazott osztály) hozhatunk létre.
- **Jellemzői:**
    - A gyermek megörökli a szülő összes nem privát adattagját és metódusát.

- A gyermek *kibővítheti* az őst új tulajdonságokkal, vagy *felülírhatja* (override) az ős metódusait.
- Ez egy **"IS-A" (egyfajta)** kapcsolat. (Pl. a Kutya egyfajta Állat). Célja a kódújrahasznosítás.

- **Osztályhierarchia:** Az öröklődések sorozatából kialakuló fa struktúra (pl. Object → Jármű → Autó → Sportautó).



3.4. Polimorfizmus és Metódustúlterhelés

A többalakúság (polimorfizmus) azt jelenti, hogy azonos nevű vagy interfészű dolgok különbözőképpen viselkedhetnek. Két fő fajtája van:

1. Metódustúlterhelés (Overloading) – Statikus / Fordítási idejű polimorfizmus:

- Egy osztályon belül több metódus is **létezhet ugyanazzal a névvel**, feltéve, hogy a **szignatúrájuk** (a paraméterek száma, típusa vagy sorrendje) eltérő.
- A fordító már a kód megírásakor (fordítási időben) tudja, hogy a paraméterek alapján pontosan melyik függvényt kell meghívni.

2. Polimorfizmus (Dinamikus kötés) – Futási idejű többalakúság:

- Lényege: Egy ősosztály típusú referencia (változó) mutathat egy leszármazott osztály példányára (Pl. Allat a = new Kutya();).
- Ha meghívunk egy felülírható (virtuális) metódust az ős referenciaváltozóján, a program csak **futási időben** dönti el az objektum *tényleges típusa* alapján, hogy melyik metódus fusson le (az ősé, vagy a gyermek által felülírt változat).

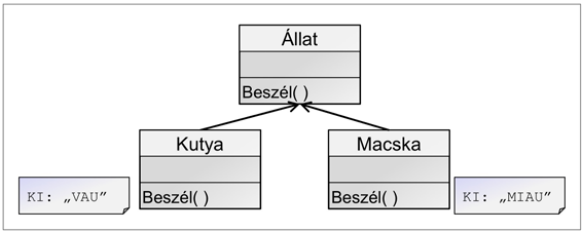
3.5. Absztrakt osztályok és Interfészek

Mindkettő a magas szintű tervezést és az öröklődés kikényszerítését szolgálja.

Absztrakt osztály (Abstract Class):

- Olyan osztály, amelyet **NEM lehet példányosítani** (nem hozható létre belőle objektum a new kulcsszóval).
- Tartalmazhat normál (kidolgozott) metódusokat és **absztrakt metódusokat** is.

- *Absztrakt metódus:* Csak deklarációja van (fejléce), de nincs törzse (implementációja). A gyermekosztálynak **kötelező** ezt megírnia, különben a gyermek is absztrakt marad.



Interfész (Interface):

- Egy teljesen absztrakt "szerződés" vagy viselkedésleírás.
- **Kizárólag** absztrakt metódusok deklarációit tartalmazhatja (bár újabb nyelvekben már lehet alapértelmezett implementáció). **NEM tartalmazhat adattagokat** (állapotot).
- Egy osztály *implementálja* az interfészt (akár többet is egyszerre, ezzel megoldva a **többszörös öröklődést**, ami sima osztályoknál sok nyelvben, pl. C# vagy Java, tilos).

Tulajdonság	**Absztrakt Osztály**	**Interfész**
Példányosítható?	Nem	Nem
Tartalmazhat adattagot (állapotot)?	Igen	Nem
Tartalmazhat kifejtett metódust?	Igen	(Általában) Nem
Többszörös "öröklődés"	Nem (csak 1 szülő lehet)	Igen (több is implementálható)

3.6. Típustagok (Statikus tagok)

- A normál adattagok és metódusok az *objektumhoz* (példányhoz) tartoznak (példány szintűek). Minden objektumnak saját másolata van az adattagokból.
- A **Típustagok (Static / Statikus tagok)** magához az **osztályhoz** (a típushoz) tartoznak, nem a példányokhoz.
- Jellemzői:
    - A statikus változóból csak egyetlen egy darab létezik a memóriában, ezen osztozik az osztály összes példánya.
    - A statikus metódusok meghívásához nem kell objektumot létrehozni (pl. Math.Sqrt()).
    - Egy statikus metódusból nem lehet hivatkozni példányszintű (nem statikus) tagokra vagy a this kulcsszóra, hiszen nem tudni, melyik objektum nevében fut.

Operációs rendszerek és Folyamatkezelés

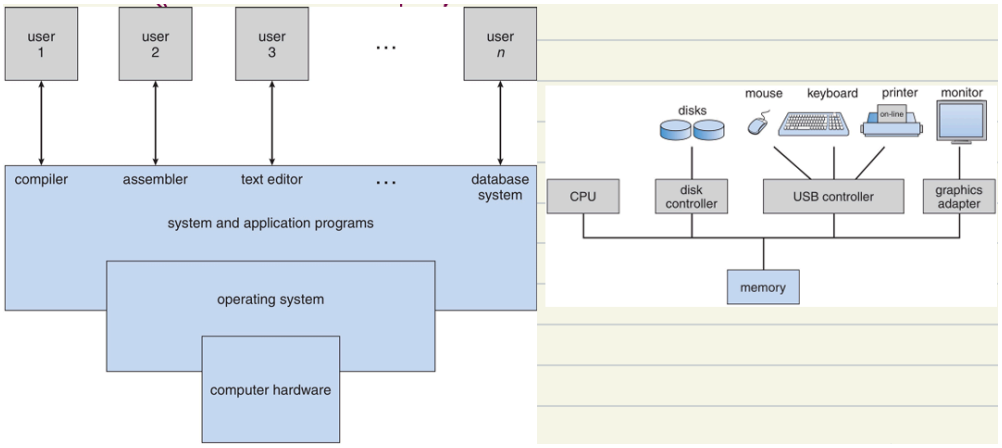
4.1. Operációs rendszerek fogalma, felépítése, osztályozásuk

Az Operációs rendszer (OS) fogalma:

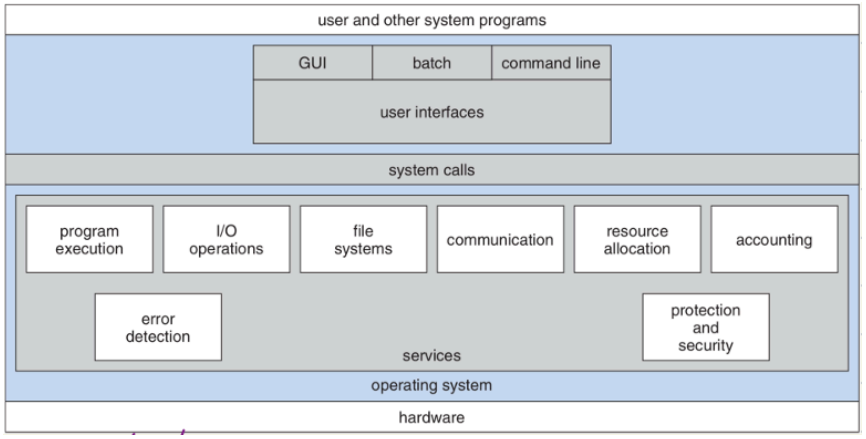
Nincs egyetlen, univerzális definíció, de funkcionálisan az OS egy olyan alapszoftver, amely a számítógép hardverét és szoftvereit kezeli, absztrakciót nyújt a hardver felett a felhasználói programok számára, és közvetítőként szolgál a felhasználó és a gép között.

Két fő megközelítésből írható le:

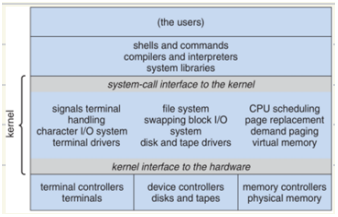
1. **Erőforrás-kezelő (Resource allocator):** Kezeli a CPU-t, memóriát, I/O eszközöket, és igazságosan/hatékonyan osztja el azokat a programok között.
2. **Vezérlő program (Control program):** Felügyeli a felhasználói programok végrehajtását a hibák és a számítógép helytelen használatának elkerülése végett.



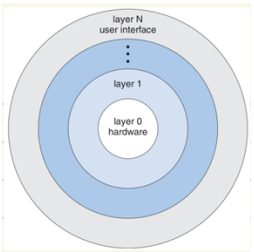
Felépítésük (Architektúrák):



- **Monolitikus:** Az operációs rendszer összes funkciója (fájlkezelés, folyamatkezelés, memóriakezelés) egyetlen nagy kernelben fut, kernel módban. Gyors, de ha egy rész összeomlik, a teljes rendszer leállhat (pl. korai UNIX, Linux).



- **Mikrokernel:** A kernel csak a legszükségesebb funkciókat (pl. memóriakezelés alapjai, IPC - folyamatok közötti kommunikáció) tartalmazza. A többi szolgáltatás (pl. fájlrendszer) felhasználói (user) módban fut. Lassabb, de sokkal biztonságosabb és stabilabb (pl. Minix, Mach).
- **Réteges (Layered):** A rendszer szigorú hierarchikus rétegekből áll, ahol az N. réteg csak az $N - 1$. réteg szolgáltatásait használhatja.



- **Hibrid:** Több struktúra előnyeit ötvözi (pl. Windows NT, macOS).

Osztályozásuk:

- *Batch (Kötegelt):* Nincs interakció, a feladatok (jobok) sorban hajtódnak végre.
- *Time-sharing (Időosztásos):* A CPU gyorsan váltogat a feladatok között, így több felhasználó interaktívan, egyidejűleg használhatja a gépet.
- *Real-time (Valós idejű):* Szigorú időkorlátok vannak (pl. orvosi műszerek, autóvezérlés). Ha a határidőn belül nem fut le a folyamat, a rendszer kudarcot vallott.

4.2. Fájlok és Fájlrendszerek. Speciális fájlok UNIX alatt.

- **Fájl:** Logikai tárolási egység, egymáshoz kapcsolódó adatok névvel ellátott halmaza a háttértáron.
- **Fájlrendszer:** A fájlok és könyvtárak (mappák) tárolásának, elnevezésének és rendszerezésének módszere az adathordozón (pl. NTFS Windows-on, ext4 Linux-on).

Speciális fájlok UNIX alatt:

A UNIX filozófiája: *"Minden fájl."* Ez azt jelenti, hogy a hardvereszközöket és a folyamatok közötti kommunikációs csatornákat is fájlként látja a rendszer.

1. **Normál fájl (-):** Szöveges vagy bináris adatok.
2. **Könyvtár (d - directory):** Fájlok és almappák neveit és inode (index csomópont) hivatkozásait tartalmazza.
3. **Karakteres eszközfájl (c - character device):** Soros hozzáférésű hardverek, bájtonként kommunikálnak (pl. billentyűzet, egér, soros port).
4. **Blokkos eszközfájl (b - block device):** Véletlenszerű hozzáférésű hardverek, blokkokban kommunikálnak (pl. merevlemez, SSD).

5. **Csővezeték (p - pipe/FIFO):** Folyamatok közötti kommunikációra szolgáló speciális fájl.
6. **Socket (s):** Hálózati (vagy gépen belüli) kommunikációs végpont fájlként reprezentálva.
7. **Szimbolikus link (l - symlink):** Olyan fájl, amely egy másik fájl vagy könyvtár elérési útját (pointerét) tartalmazza (hasonló a Windows parancsikonhoz).

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

4.3. Átirányítás, csővezetékek (Pipes)

A parancssoros (Shell) környezetben minden futó program alapvetően 3 standard I/O (Input/Output) csatornával rendelkezik:

- **stdin (0):** Standard bemenet (alapértelmezetten a billentyűzet).
- **stdout (1):** Standard kimenet (alapértelmezetten a képernyő).
- **stderr (2):** Standard hibakimenet (alapértelmezetten a képernyő).

Átirányítás operátorai:

- > : A kimenetet egy fájlba irányítja (felülírja a fájlt). Pl: ls > lista.txt
- >> : A kimenetet egy fájl végéhez fűzi (hozzáírás).
- < : A bemenetet egy fájlból veszi a billentyűzet helyett. Pl: sort < nevek.txt
- 2> : Csak a hibakimenetet (stderr) irányítja át fájlba.

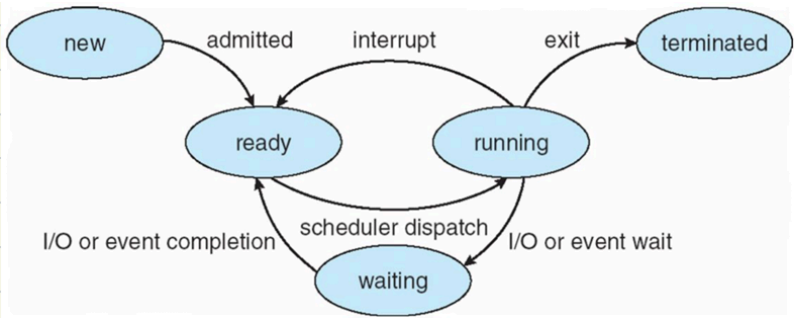
Csővezeték (Pipe |):

A pipe operátor lehetővé teszi, hogy két programot láncba kössünk úgy, hogy az **első program standard kimenete (stdout) közvetlenül a második program standard bemenetévé (stdin) váljon**.

- *Példa:* ls -l | grep "txt" (Listázza a fájlokat, de a grep szűri a kimenetet, és csak a "txt"-t tartalmazó sorokat írja ki).

4.4. Folyamatkezelés (Process management) és Ütemezés

- **Folyamat (Process):** Egy végrehajtás alatt álló program. (A program a diszken lévő passzív kód, a folyamat a memóriában lévő aktív entitás).
- **Folyamat állapotai:** Új (New), Futásra kész (Ready), Futó (Running), Várakozó / Blokkolt (Waiting), Befejezett (Terminated).



CPU Ütemezési algoritmusok:

Az operációs rendszer feladata eldönteni, hogy a "Ready" sorban várakozó folyamatok közül melyik kapja meg a processzort (CPU-t).

1. **FCFS (First-Come, First-Served):** Érkezési sorrendben hajtja végre a feladatokat. (FIFO). Hátránya: Egy hosszú folyamat megakaszthatja a többit (Konvoj effektus).
2. **SJF (Shortest Job First):** A legrövidebb becsült futási idejű folyamatot veszi előre. Lehet megszakításos (Preemptive) vagy megszakítás nélküli (Non-preemptive).
3. **Priority Scheduling (Prioritásos):** Minden folyamat kap egy prioritási számot. A legmagasabb prioritású fut először.
    - *Probléma: Kiéheztetés (Starvation)* - Az alacsony prioritású folyamatok sosem kerülnek sorra.
    - *Megoldás: Öregítés (Aging)* - A várakozási idővel arányosan folyamatosan növeljük a prioritásukat.
4. **Round-Robin (RR - Körforgó):** Kifejezetten időosztásos (Time-sharing) rendszerekhez. Minden folyamat kap egy kis időkvantumot (q, pl. 10-100 ms). Ha lejár az idő, a rendszer megszakítja (preemptálja) a folyamatot, és a sor végére teszi.
    - Ha a q túl nagy → FCFS lesz belőle.
    - Ha a q túl kicsi → A kontextusváltások (Context switch) miatti adminisztrációs teher (overhead) lelassítja a gépet.
5. **Multilevel Queue (Többszintű sorok):** A folyamatokat különböző sorokba osztja (pl. rendszerfolyamatok, interaktív, batch), és a sorok között is van ütemezés (prioritás).

Többprocesszoros ütemezés (Multiprocessor Scheduling):

- *SMP (Symmetric Multiprocessing):* Minden CPU-nak (magnak) saját ütemezője van, maguk döntik el, mit futtatnak a közös (vagy saját) várakozási sorból.
- *Processzor affinitás:* A rendszer igyekszik a folyamatot ugyanazon a CPU-magon tartani, ahol korábban futott, hogy a gyorsítótár (Cache) adatai ne vesszenek el.

4.5. Jelzések (Szignálok) és Ütemezett végrehajtás

Jelzések (Signals):

Szoftveres megszakítások (Software interrupts), a folyamatok közötti kommunikáció (IPC) legrégebbi és legegyszerűbb formája UNIX rendszerekben. A rendszer aszinkron eseményekről értesíti a folyamatot.

- Ha egy folyamat kap egy szignált, válaszolhat rá alapértelmezett módon (pl. kilép), ignorálhatja (ha a szignál engedi), vagy lekezelheti egy saját függvénnyel (Signal handler).
- *Gyakori szignálok:*
    - SIGINT (2): Megszakítás a billentyűzetről (Ctrl+C).
    - SIGTERM (15): Kérés a folyamat szabályos, "szép" befejezésére (Alapértelmezett a kill parancsnál).
    - SIGKILL (9): Azonnali kilövés. Ezt a folyamat NEM tudja elkapni, ignorálni vagy felülírni, az OS azonnal megöli.

Ütemezett végrehajtás:

Lehetőség parancsok vagy scriptek automatikus lefutására egy megadott jövőbeli időpontban.

- at parancs: Egyszeri, jövőbeli időpontban történő végrehajtásra (pl. at 14:00 tomorrow).
- cron daemon (szolgáltatás): Rendszeresen ismétlődő feladatok végrehajtására szolgál. A feladatokat a crontab nevű fájlban definiáljuk.
    - A crontab szintaxisa 5 időmezőből és a parancsból áll: [perc] [óra] [nap] [hónap] [hét napja] [parancs]
    - Pl. 0 5 * * 1 tar -zcf backup.tar.gz /home (Minden hétfőn hajnali 5 órakor lefut a biztonsági mentés).

Szoftverfejlesztési elvek, Tesztelés és Minták

5.1. Verziókezelés és Verziókezelő rendszerek (VCS)

A verziókezelés olyan eljárások és szoftveres eszközök összessége, amelyek nyomon követik és rögzítik a forráskódban (vagy dokumentumokban) végzett változtatásokat. Célja a csapatmunka támogatása, a kód történetének megőrzése és a korábbi állapotok visszaállításának lehetősége.

Alapfogalmak:

- **Repository (Tároló):** Az a központi vagy helyi adatbázis, ahol a fájlok és a változások története tárolódik.
- **Commit:** Egy adott állapot (változáscsomag) véglegesítése és mentése a repository-ba.
- **Branch (Elágazás):** A fő fejlesztési vonaltól (általában main vagy master) való eltérés, amely lehetővé teszi új funkciók párhuzamos fejlesztését anélkül, hogy a stabil kódot elrontanánk.
- **Merge (Összeolvasztás):** Két ág (pl. egy fejlesztői ág és a fő ág) módosításainak egyesítése.
- **Központosított vs. Elosztott:** A modern rendszerek (mint a **Git**) elosztottak, ami azt jelenti, hogy minden fejlesztő gépén ott van a teljes repository egy teljes másolata, nem csak a központi szerveren.

5.2. Szoftvertesztelési alapfogalmak

A szoftvertesztelés célja a hibák (bugok) megtalálása, a minőség biztosítása és annak igazolása, hogy a szoftver megfelel a specifikációknak.

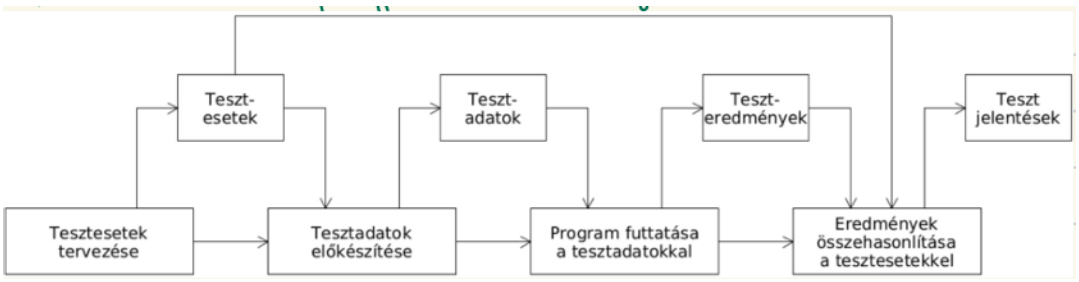
A szoftvertesztelés hét alapelve:

1. A tesztelés a hibák jelenlétét mutatja meg, nem a hiányukat
2. Lehetetlen a kimerítő tesztelés
3. A korai tesztelés időt és pénzt takarít meg
4. A hibák csoportosulnak
5. Óvakodj a kártevőírtó paradoxontól
6. A tesztelés környezetfüggő
7. A hibamentesség egy tévhit

Egy tesztfolyamat az alábbi fő tevékenységcsoportokból áll:

- Teszttervezés (*test planning*)
- Tesztfelügyelet és -irányítás (*test monitoring and control*)
- Tesztelemzés (*test analysis*)
- Teszttervezés (*test design*)
- Teszt megvalósítás (*test implementation*)
- Teszt végrehajtás (*test execution*)
- Tesztlezárás (*test completion*)

Modellje:



1. Tesztszintek (Milyen mélyen tesztelünk?):

- **Egységteszt (Unit test):** A legkisebb, önállóan tesztelhető kódrészek (függvények, metódusok, osztályok) izolált tesztelése. (Fejlesztők végzik).
- **Integrációs teszt:** A már letesztelt egységek közötti kommunikáció és adatcsere helyességének ellenőrzése.
- **Rendszerteszt:** A szoftver egészének, mint teljes, integrált rendszernek a tesztelése a specifikáció alapján.
- **Elfogadási teszt (Acceptance test):** A megrendelő vagy a felhasználó végzi, célja annak eldöntése, hogy a rendszer megfelel-e az üzleti igényeknek.

2. Teszttervezési módszerek (Teszttípusok):

- **Feketedoboz-tesztelés:** A tesztelő nem ismeri a szoftver belső kódját. Csak a bemeneteket (input) adja meg, és a kimeneteket (output) vizsgálja az elvárásokhoz képest.
- **Fehérdoboz-tesztelés:** A tesztelő látja és ismeri a forráskódot. A kód belső szerkezetét, elágazásait és ciklusait teszteli (pl. kódlefedettség - code coverage mérése).

5.3. Objektumorientált tervezési alapelvek: SOLID, DRY

A SOLID öt alapelv rövidítése, amelyek betartásával a kód rugalmasabb, karbantarthatóbb és könnyebben bővíthető lesz.

1. **S - Single Responsibility Principle (Egyetlen felelősség elve):** Egy osztálynak csak egyetlen feladata (felelőssége) lehet. Csak egyetlen oka lehet a változásra. (Pl. ne ugyanaz az osztály nyomtasson a képernyőre, ami az adatbázis-kapcsolatot kezeli).
2. **O - Open/Closed Principle (Nyílt/Zárt elv):** A szoftver entitásainak (osztályok, modulok) **nyíltnak** kell lenniük a kiterjesztésre (öröklődés, interfészek), de **zártnak** a módosításra. Új funkciót új kód hozzáadásával érjünk el, ne a már működő, régi kód átírásával.
3. **L - Liskov Substitution Principle (Liskov-féle behelyettesítési elv):** Egy ősosztály típusú referenciának anélkül kell behelyettesíthetőnek lennie bármelyik leszármazottjával, hogy a program helyes működése megváltozna. (A gyermek ne szűkítse le az ős képességeit).
4. **I - Interface Segregation Principle (Interfész-elkülönítési elv):** A klienseket nem szabad rákényszeríteni olyan interfészek implementálására, amelyeket nem használnak. Inkább sok kicsi, specifikus interfészt csináljunk, mint egy nagy "mindenes" (fat) interfészt.
5. **D - Dependency Inversion Principle (Függőség-megfordítási elv):** A magas szintű modulok ne függjenek az alacsony szintű moduloktól. Mindkettőnek absztrakcióktól (interfészektől) kell függnie.

DRY – Don’t Repeat Yourself – Kerüljük a kódismétlést

- Az ismétlések fajtái:
  - **Kényszerített ismétlés** (***imposed duplication***): a fejlesztők úgy érzik, hogy nincs választásuk, a környezet láthatólag megköveteli az ismétlést.
  - **Nem szándékos ismétlés** (***inadvertent duplication***): a fejlesztők nem veszik észre, hogy információkat duplikálnak.
  - **Türelmetlen ismétlés** (***impatient duplication***): a fejlesztők lustaságából fakad, az ismétlés látszik a könnyebb útnak.
  - **Fejlesztők közötti ismétlés** (***interdeveloper duplication***): egy csapatban vagy különböző csapatokban többen duplikálnak egy információt.
- Kapcsolódó fogalom: kódismétlés (*code duplication*, *duplicate code*), *copy-and-paste programming*

Ellenkezője – **Write Everything Twice**

KISS – Keep It Simple Stupid

Minél egyszerűbben oldjunk meg mindent.

YAGNI – You Aren't Gonna Need It

Csak akkor implementáljunk valamit, ha szükséges.

Csatoltság – Modul függése egy másiktól (Laza(jobb) vagy Szoros)

Függőség-befecskendezés (Dependency Injection - DI)

A DIP elv megvalósításának egyik módszere. Lényege, hogy ha egy A osztálynak szüksége van egy B osztályra a működéséhez, akkor az A osztály **nem maga hozza létre** (nem new-ozza le) a B osztályt, hanem kívülről "kapja meg" (injektálják neki) paraméterként – általában a konstruktorában, egy interfészen keresztül. Ez nagymértékben megkönnyíti a tesztelést.

5.4. Architekturális és Tervezési minták

A tervezési minták (Design Patterns) gyakran ismétlődő programozási problémákra adott, bevált, újrahasznosítható megoldássablonok.

1. GoF (Gang of Four) Tervezési minták: Három fő kategóriába soroljuk őket:

- **Létrehozási minták (Creational):** Objektumok példányosítását szabályozzák (pl. *Singleton / Egyke* - garantálja, hogy egy osztályból csak egyetlen példány létezzen; *Factory / Gyár*).
- **Szerkezeti minták (Structural):** Osztályok és objektumok nagyobb struktúrákba rendezését segítik (pl. *Adapter*, *Decorator*).
- **Viselkedési minták (Behavioral):** Objektumok közötti kommunikációt és felelősségmegosztást írják le (pl. *Observer / Megfigyelő*, *Strategy*).

2. Architekturális minta: MVC (Model-View-Controller) Interaktív szoftverek (pl. weboldalak, asztali alkalmazások) felépítésének alapszabványa. Célja a felhasználói felület (UI) és az üzleti logika szigorú szétválasztása.

- **Model (Modell):** Az alkalmazás adatai és a hozzájuk tartozó üzleti logika (pl. adatbázis-lekérdezések, számítások). Nem tud a felületről.
- **View (Nézet / Megjelenítés):** A grafikus felület, amit a felhasználó lát. Megjeleníti a Modell adatait.
- **Controller (Vezérlő):** A közvetítő. Fogadja a felhasználói bemeneteket (kattintás, gépelés), értelmezi azokat, utasítja a Modellt az adatok frissítésére, majd frissíti a Nézetet az új adatokkal.

5.5. Szoftverlicencek: Szabad és nem szabad szoftverek

A szoftverlicenc egy jogi szerződés a készítő és a felhasználó között, amely meghatározza a szoftver használatának, módosításának és terjesztésének feltételeit.

1. Proprietary (Zárt / Tulajdonosi) szoftverek: A forráskód titkos (nem elérhető a felhasználók számára). A licenc csak a használat jogát adja meg, a módosítás, visszafejtés (reverse engineering) és a továbbadás szigorúan tilos. (Pl. Microsoft Windows, MS Office).

2. Szabad szoftver (Free Software) és Nyílt forráskód (Open Source): Bár a két mozgalom filozófiája kicsit eltér (a Szabad szoftver mozgalom a felhasználó szabadságjogaira, az Open Source a fejlesztési modell hatékonyságára fókuszál), a gyakorlatban a licenceik nagyon hasonlók. A Szabad szoftver alapja a **4 alapszabadság**:

1. A program tetszőleges célból történő **futtatásának** szabadsága.
2. A program működésének **tanulmányozási** és módosítási szabadsága (ehhez a forráskód elérése előfeltétel).
3. A másolatok **továbbadásának** szabadsága (segíthetsz a felebarátodon).
4. A módosított verziók **terjesztésének** szabadsága.

Szabad licencek fajtái:

- **Copyleft licencek (pl. GNU GPL):** Szigorúak (vírusszerűek). Ha felhasználsz vagy módosítasz egy ilyen kódcsipetet a saját szoftveredben, akkor a te szoftveredet is **kötelező** ugyanezzel a szabad licenccel, nyílt forráskóddal kiadnod.
- **Megengedő (Permissive) licencek (pl. MIT, BSD, Apache):** Nagyon kevés korlátozást tartalmaznak. Lehetővé teszik, hogy a nyílt forráskódot zárt, üzleti (proprietary) szoftverekben is felhasználd anélkül, hogy a saját kódodat nyílttá kellene tenned (általában csak a szerzők nevét kell feltüntetni).

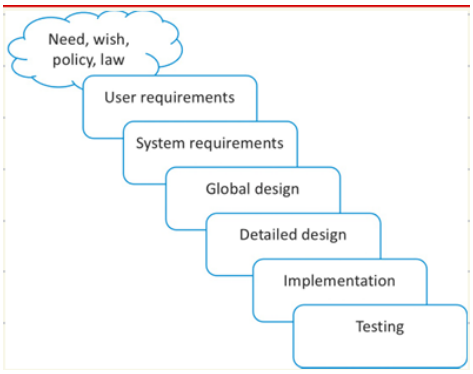
Szoftverfejlesztési Módszertanok

6.1. Hagyományos (Klasszikus) Szoftverfejlesztési Módszertanok

Ezek a modellek a szoftverfejlesztés hőskorából származnak. Jellemzőjük, hogy a folyamatokat szigorúan elkülönülő fázisokra bontják (pl. Követelményelemzés → Tervezés → Implementáció → Tesztelés → Karbantartás).

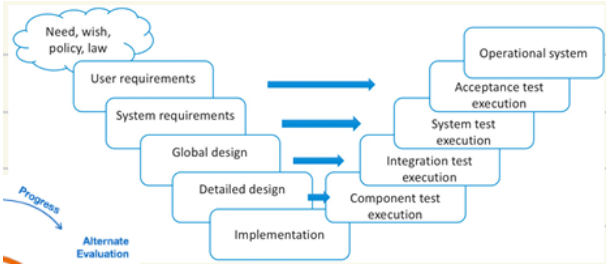
1. Vízesés modell (Waterfall)

- A legrégebbi és legegyszerűbb módszertan. Lineáris, a lépések szigorúan egymás után következnek.
- Egy fázisnak teljesen le kell zárulnia ahhoz, hogy a következő elkezdődhessen.
- **Előnye:** Könnyen menedzselhető, jól dokumentált.
- **Hátránya:** Teljesen rugalmatlan. Mindent előre definiálni kell az elején. A megrendelő csak a folyamat legvégén lát működő szoftvert. Ha menet közben megváltozik az igény, szinte lehetetlen visszalépni.
- **Mikor jó?** Ha a követelmények kőbe vannak vésve és nem változnak (pl. repülésirányítás, NASA).



2. V-modell

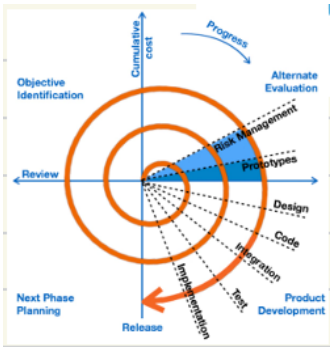
- A Vízesés modell továbbfejlesztése. A különbség az, hogy a tervezési/implementációs szakaszokkal (a V betű bal szára) **párhuzamosan** már megtervezik a hozzájuk tartozó tesztelési szakaszokat is (a V betű jobb szára).
- Pl. a Követelményelemzéshez tartozik az Elfogadási teszt, a Rendszertervezéshez a Rendszerteszt, a Kódoláshoz az Egységteszt.



3. Spirális modell

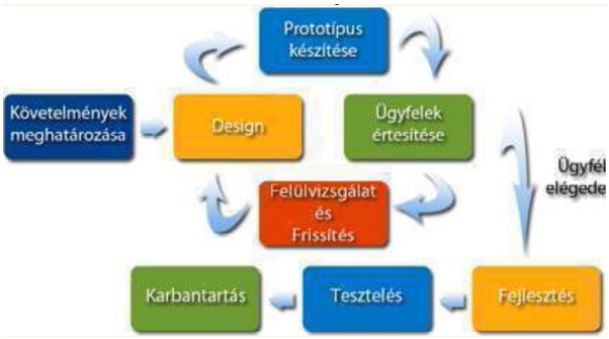
- A legfőbb kulcsszava a **kockázatelemzés**.

- Ciklusos (iteratív) modell. A folyamat egy spirál mentén halad, minden körben fázisokon (tervezés, kockázatelemzés, fejlesztés, értékelés) megy keresztül.
- Minden ciklus végén elkészül egy működő verzió (prototípus).
- **Mikor jó?** Hatalmas, drága, komplex és nagyon kockázatos rendszerek (pl. banki alaprendszerek) fejlesztésénél.



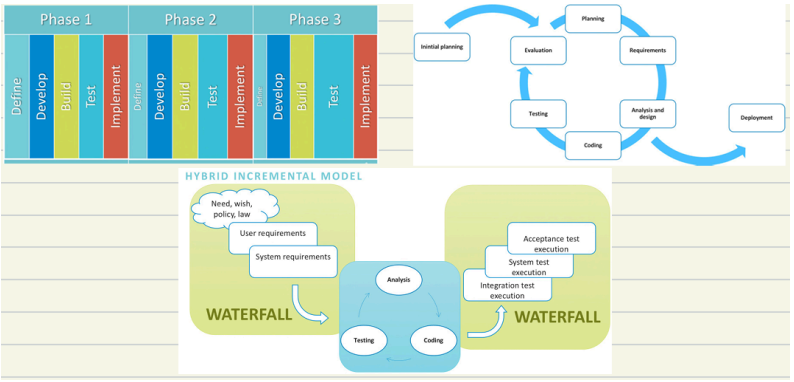
4. Prototípus alapú fejlesztés

- Ha a megrendelő nem tudja pontosan megfogalmazni, mit akar, a fejlesztők készítenek egy gyors, "működőnek látszó" prototípust (pl. csak a grafikus felületet, amire lehet kattintani, de nincs mögötte kód).
- A megrendelő kipróbálja, elmondja a véleményét, és csak a pontosított követelmények után kezdik el a valódi, robusztus rendszer fejlesztését (a prototípust gyakran eldobják).



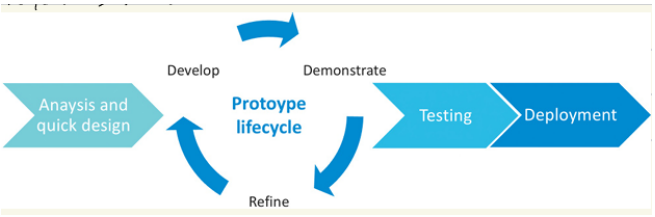
5. Iteratív és Inkrementális módszertanok

- **Inkrementális (Növekményes):** A szoftvert kisebb, önállóan működő darabokra (inkrementumokra) bontják, és ezeket lépésenként adják ki.
- **Iteratív (Ismétlődő):** A fejlesztési fázisokat ciklusokban (iterációkban) ismétlik, a szoftver folyamatosan finomodik, bővül.



6. Gyors alkalmazásfejlesztés (RAD - Rapid Application Development)

- Az iteratív modell egy speciális formája. A kódolás helyett a **tervezésen és a gyors prototípusgyártáson** van a hangsúly. Kódgenerátorokat, meglevő komponenseket használnak összeollózva, hogy minél előbb működő programot mutassanak.



6.2. Agilis Szoftverfejlesztési Módszertanok és a Kiáltvány

A 2000-es évek elején a szoftverfejlesztők rájöttek, hogy a hagyományos (Vízesés) modellek bürokratikusak, és mire a szoftver elkészül, a piac és a megrendelői igények már megváltoztak. Ekkor hozták létre az **Agilis Kiáltványt (Agile Manifesto)**.

Az Agilis Kiáltvány 4 alapértéke:

1. **Indivídumok és interakciók** fontosabbak, mint a *folyamatok és eszközök*. (A jó kommunikáció a csapatban mindennél többet ér).
2. **Működő szoftver** fontosabb, mint az *átfogó dokumentáció*.
3. **Megrendelővel való folyamatos együttműködés** fontosabb, mint a *szerződéses egyeztetések*.
4. **Változásokra való reagálás** fontosabb, mint *egy eredeti terv merev követése*.

(Megjegyzés: Ez nem azt jelenti, hogy a dőlt betűs dolgok nem fontosak, csak a vastagon szedettek sokkal értékesebbek az agilis szemléletben).

12 alapelv:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity–the art of maximizing the amount of work not done–is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

6.3. Egy szabadon választott agilis módszertan: a SCRUM

A Scrum nem egy hagyományos folyamat, hanem egy "keretrendszer" (framework), amelyben a komplex szoftverfejlesztést menedzselni lehet. Erősen iteratív és inkrementális. Három fő pillérre épül: Átláthatóság, Vizsgálat (ellenőrzés) és Alkalmazkodás.

1. Scrum Szerepkörök (Roles)

A Scrumban nincsenek hagyományos projektmenedzserek, a felelősség szét van osztva:

- **Product Owner (Terméktulajdonos):** A megrendelő és az üzleti oldal képviselője a csapatban. Ő dönti el, hogy MIT fogunk csinálni. Feladata a követelmények listájának (Product Backlog) összeállítása és a feladatok fontossági sorrendjének (prioritásának) meghatározása.
- **Scrum Master:** A folyamat felelőse. Ő egy "szolgáló vezető" (facilitátor). Nem főnöke a csapatnak! Az a dolga, hogy elhárítsa a csapat elől a munkát gátló akadályokat, és biztosítsa, hogy mindenki betartsa a Scrum szabályait.
- **Development Team (Fejlesztőcsapat):** Önszerveződő és keresztfunkcionális (van benne fejlesztő, tesztelő, dizájner, aki csak kell a munkához). Általában 3-9 fő. Ők döntik el, hogy HOGYAN oldják meg a feladatokat.

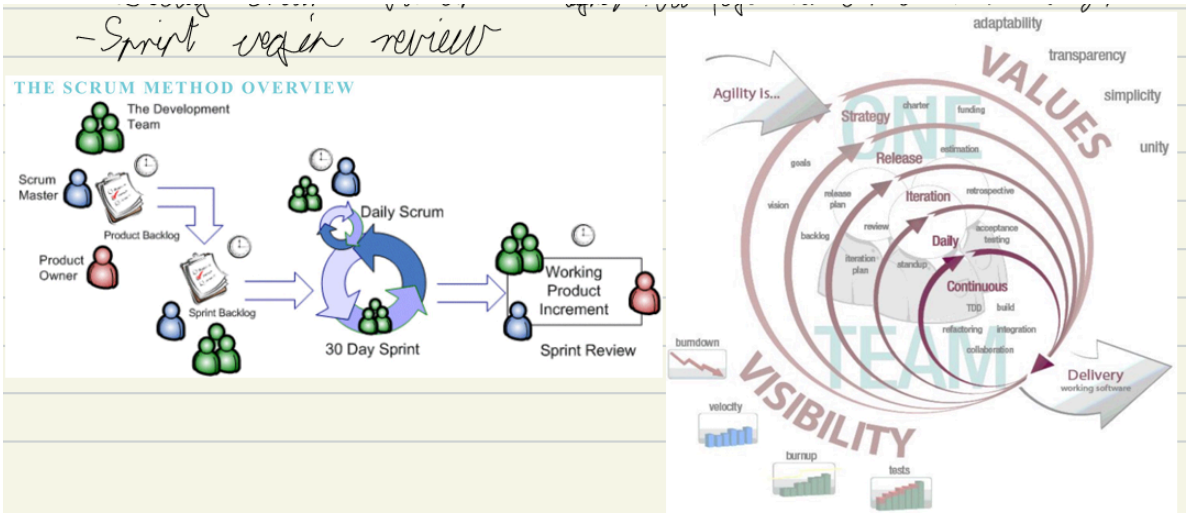
2. Scrum Dokumentumok / Eszközök (Artifacts)

- **Product Backlog (Termék Teendőlista):** Az összes olyan dolog (követelmény, funkció, hiba, javítás) listája, amire a terméknek valaha szüksége lehet. A Product Owner kezeli. Az elemei gyakran "User Story" (Felhasználói történet) formátumban vannak felírva.
- **Sprint Backlog (Sprint Teendőlista):** Az aktuális fejlesztési ciklusra (Sprintre) a fejlesztőcsapat által kiválasztott feladatok listája.
- **Increment (Inkrementum / Növekmény):** Az aktuális Sprint végén elkészült, letesztelt, **potenciálisan kiadható** (működőképes) szoftververzió.

3. Scrum Események (Ceremonies)

Minden esemény fix időkeretű (timeboxed).

- **Sprint:** A Scrum szíve. Egy 1-től 4 hétig terjedő, fix hosszúságú iteráció, amelynek során a csapat létrehozza a Működő Szoftver-inkrementumot.
- **Sprint Planning (Sprint Tervezés):** A Sprint legelején tartják. A Product Owner felvázolja a legfontosabb feladatokat, a Csapat pedig kiválasztja, hogy ebből mennyit tud biztonsággal megcsinálni a Sprint alatt (ez lesz a Sprint Backlog).
- **Daily Scrum (Napi Állóértekezlet):** Minden nap ugyanakkor, ugyanott tartott max. 15 perces megbeszélés a Fejlesztőcsapatnak. Csak 3 kérdésre válaszolnak:
    1. *Mit csináltam tegnap, ami segítette a Sprint célját?*
    2. *Mit fogok csinálni ma?*
    3. *Van-e valamilyen akadály (blocker) előttem?*
- **Sprint Review (Sprint Áttekintő / Demo):** A Sprint végén a csapat megmutatja (demózza) a megrendelőnek (és a Product Ownernek) az elkészült, működő szoftvert.
- **Sprint Retrospective (Sprint Visszatekintő):** A csapat (a Scrum Master vezetésével) megbeszéli a Sprint során szerzett tapasztalatokat. *Mi ment jól? Min kellene javítani a következő Sprintben a hatékonyabb munkáért?*



A Web működésének alapjai és Technológiái

7.1. A Web alapfogalmai és Szabványügyi szervezetek

Alapfogalmak

- **Erőforrás (Resource):** Bármi, ami azonosítható egy URI-val a weben.
- **Információ erőforrás:** Olyan erőforrás, amelynek minden lényeges jellemzője előállítható és átvihető egy üzenetben.
- **Web:** Az információs tér, melynek lényeges elemeit globális azonosítók (URI-k) azonosítják.
- **Reprezentáció:** Egy erőforrás aktuális állapotáról információkat kódoló adat.
- **Tartalomegyeztetés (Content Negotiation):** Ha egy erőforráshoz több reprezentáció is tartozik (pl. angol és magyar nyelvű oldal, vagy PDF és HTML formátum), ez az a mechanizmus, amely kiválasztja a felhasználónak legmegfelelőbbet.
- **Web ágens (Web agent):** A weben egy entitás nevében cselekvő szoftver. A **User-agent** kifejezetten egy emberi személy nevében eljáró szoftver (pl. a webböngészőnk).

Web Szabványok és Szervezetek

A weben a kommunikáció protokolljait és formátumait szabványok rögzítik.

- **De-Facto szabvány:** Gyakori használatból, piaci elterjedtségből fakadó (nem hivatalos) szabvány (pl. QWERTY billentyűzet, régi PDF).
- **De-Jure szabvány:** Állami vagy nemzetközi szabványügyi testületek által előírt szabvány.

Főbb szervezetek:

1. **IANA (Internet Assigned Numbers Authority):** IP címek kiosztását végzi, felügyeli a DNS gyökérzónákat, és nyilvántartja a különböző protokollokat és portszámokat .
2. **IETF (Internet Engineering Task Force):** Az internet mérnöki munkacsoportja, ők fejlesztik a TCP/IP alapú internet szabványokat, amelyeket **RFC** (Request for Comments) dokumentumokban publikálnak .
3. **W3C (World Wide Web Consortium):** A legfontosabb webes szabványosító testület (ők felelnek az XML, CSS szabványokért). Nyílt ajánlásokat (Recommendation) publikálnak a "Web mindenkinek" elv alapján .
4. **WHATWG:** A W3C-ből kivált (majd azzal újra együttműködő) szervezet, amely a modern webes technológiákat (HTML5, DOM, URL) gondozza.

7.2. URI, URL és URN

Az **URI (Uniform Resource Identifier)** egy egységes erőforrás-azonosító. A weben minden erőforrásnak van URI-ja. Két fő alcsoportja van:

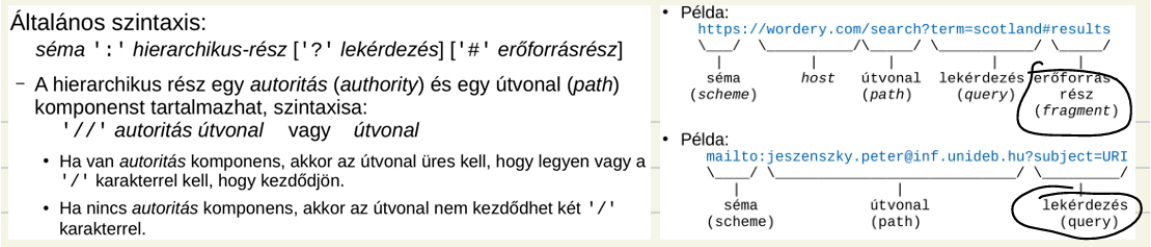
- **URL (Locator):** Nemcsak azonosítja az erőforrást, hanem megadja az **elérésének módját és helyét is** (pl. https://www.unideb.hu).
- **URN (Name):** Helytől független, tartós azonosító, amely csak megnevezi a dolgot, de nem mondja meg, hol van (pl. egy könyv ISBN száma: urn:isbn:0451450523).

Az URI szintaxisa:

séma: *hierarchikus-rész [? lekérdezés] [# erőforrásrész]*

Példa: *https://example.com:80/search?term=scotland#results*

- **Séma (Scheme):** https (pl. http, mailto, file)
- **Hierarchikus rész:** //example.com:80/search (Tartalmazza az autoritást/hosztot és az útvonalat)
    - *Autoritás (Host + Port):* example.com:80
    - *Útvonal (Path):* /search
- **Lekérdezés (Query):** term=scotland
- **Erőforrásrész / Fragmens (Fragment):** results (Az oldalon belüli horgony)



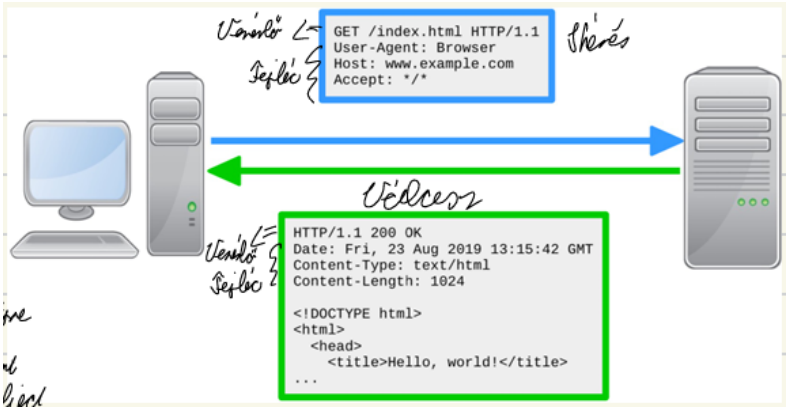
7.3. A HTTP Protokoll (Hypertext Transfer Protocol)

Állapotmentes (stateless), alkalmazási rétegbeli kérés-válasz (client-server) alapú protokoll.

Üzenetek felépítése:

Mind a kérés (Request), mind a válasz (Response) két fő részből áll:

1. **Fejléc (Header / Vezérlő adatok):** Tartalmazza a metódust/állapotkódot, a szoftver verzióját, és a metaadatokat (pl. Content-Type, User-Agent) .
2. **Tartalom (Body):** A tényleges adat (pl. a HTML kód vagy a letöltött kép).



HTTP Metódusok:

- **GET:** Erőforrás lekérése. Csak adatot olvas, a szerveren nem változtat semmit.
- **HEAD:** Ugyanaz, mint a GET, de a szerver csak a fejléceket küldi vissza, tartalmat (body) nem. Jó metaadatok ellenőrzésére sávszélesség spórolásával.

- **POST:** Adat küldése a szervernek feldolgozásra (pl. űrlap beküldése, új erőforrás létrehozása) .
- **PUT:** Egy meglévő erőforrás teljes lecserélése/frissítése (ha nem létezik, létrehozza) .
- **DELETE:** Erőforrás törlése.
- **OPTIONS:** Lekérdezi, hogy az adott erőforráson milyen metódusok (kommunikációs opciók) engedélyezettek .

HTTP Állapotkódok (Válaszkódok) :

- **1xx (Információs):** A kérés megkapva, a folyamat folytatódik.
- **2xx (Siker):** A kérés sikeresen megérkezett és feldolgozva (pl. 200 OK, 201 Created).
- **3xx (Átirányítás):** További műveletre van szükség a kérés befejezéséhez (pl. 301 Moved Permanently).
- **4xx (Kliens hiba):** A kérés hibás, vagy a kliensnek nincs jogosultsága (pl. 400 Bad Request, 403 Forbidden, 404 Not Found).
- **5xx (Szerver hiba):** A szerver hibát észlelt a szerver oldalon (pl. 500 Internal Server Error).

Tartalomegyeztetés és Sütik (Cookies)

- **Proaktív / Szerver vezérelt egyeztetés:** A böngésző a kérés fejlécében elküldi a preferenciáit (pl. Accept-Language: hu), és a szerver ez alapján választja ki a megfelelő tartalmat. Előnye a kényelem, hátránya, hogy a szervernek kell találgatnia a kliens igényeit .
- **Reaktív / Ágens vezérelt egyeztetés:** A szerver visszaküld egy listát a választható opciókról, és a böngésző (vagy a felhasználó) választ. Hátránya, hogy plusz egy kérés-válasz kört (időt) igényel .

Sütik (Cookies): Mivel a HTTP állapotmentes (nem emlékszik a korábbi kérésekre), a munkamenetek (pl. bejelentkezés) kezelésére Sütiket használunk . A szerver a válaszban küld egy Set-Cookie fejlécet (név-érték párral). A böngésző ezt elmenti, és minden további kérésnél visszaküldi a szervernek, így a szerver "felismeri" a felhasználót . Használják még testreszabásra és követésre is. (Attribútumai: Expires, Secure, HttpOnly) .

7.4. Jelölőnyelvek és a DOM (XML és HTML)

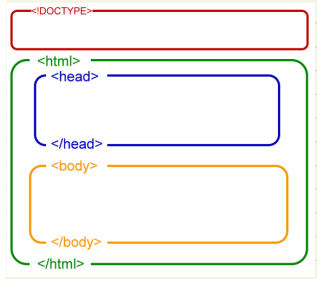
A webes adatok strukturálására jelölőnyelveket használunk.

- **XML (eXtensible Markup Language):** Általános célú, adatok leírására és adatcserére kitalált nyelv. Nincs előre definiált címkekészlete (bármilyen taget kitalálhatunk benne). Szigorú szintaxisú, adatorientált .
- **HTML (HyperText Markup Language):** Prezentációs célú nyelv, az információk webböngészőben való megjelenítésére. Előre definiált címkéi vannak (pl. <h1>, <p>, <a>). A web elsődleges nyelve (jelenleg HTML5) .

DOM (Document Object Model): Amikor a böngésző beolvassa a HTML kódot, a memóriában felépít belőle egy fa-struktúrát, a DOM-fát . Ez egy API (programozási felület), amelyen keresztül (pl.

JavaScript-tel) a dokumentum elemei elérhetők és manipulálhatók (módosíthatók, törölhetők).

- Minden ilyen dokumentumot egy fa ábrázol, mely az alábbi fajta csomópontokból áll:
  - `Document`, `DocumentType`, `DocumentFragment`, `Element`, `Text`, `ProcessingInstruction` és Comment.

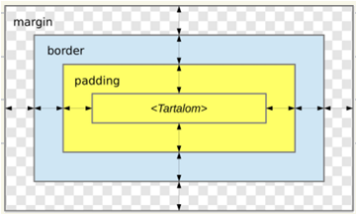


7.5. Stíluslapok: CSS (Cascading Style Sheets)

A CSS felel a strukturált HTML dokumentumok **megjelenítéséért** (dizájn). A legfőbb elv: a tartalom (HTML) és a megjelenés (CSS) szigorú szétválasztása.

Dobozmodell (Box Model): A weblapon minden elem egy téglalap alakú doboz. Ennek rétegei befelé haladva :

1. margin (Külső margó: az elemek közötti távolság)
2. border (Keret/Szegély)
3. padding (Belső margó: a keret és a tartalom közötti tér)
4. Content (Maga a tényleges tartalom)



Szelektorok (Kiválasztók): Megadják, hogy a CSS szabály melyik HTML elemre vonatkozzon.

- Elem szelektor: p { ... } (Minden bekezdésre)
- Osztály szelektor: .nav { ... } (Ahol class="nav")
- ID szelektor: #main { ... } (Ahol id="main")
- Kombinátorok: div > p (A div közvetlen gyermek p elemeire)

```
thead th { background-color: darkgrey }
nav > div { display: inline }
p > img:only-child { margin-left: 0 }
h1 + p { text-indent: 0 }
img ~ span { color: red }
```

Specifikusság (Prioritás): Ha egy elemre több, egymásnak ellentmondó CSS szabály is illeszkedik, a böngésző egy pontszám (a, b, c) alapján dönti el, melyik nyer:

- a **pont:** ID szelektorok száma a szabályban.
- b **pont:** Osztály (class), attribútum és pszeudo-osztály szelektorok száma.

- *c* **pont:** Elem és pszeudo-elem szelektorok száma. *(Megj: Az "inline" stílus, amit egyenesen a HTML tagbe írnak, minden fenti szabályt felülír).*

Kiválasztó	Specifikusság
*	(*a* = 0, *b* = 0, *c* = 0)
div	(*a* = 0, *b* = 0, *c* = 1)
p::first-letter	(*a* = 0, *b* = 0, *c* = 2)
a img	
h1 + p	
div[class=nav]	(*a* = 0, *b* = 1, *c* = 1)
div.nav	
#main *:lang(en)	(*a* = 1, *b* = 1, *c* = 0)

7.6. JSON (JavaScript Object Notation)

Könnyűsúlyú, szövegalapú adatcsere-formátum. Az XML modern alternatívája (főleg webes API-knál).

- **Előnye az XML-lel szemben:** Sokkal rövidebb, ember és gép számára is könnyebben olvasható, JavaScript-ben natívan feldolgozható, és jobban tükrözi az objektumorientált adatszerkezeteket (tömbök, kulcs-érték párok) .
- **Adattípusai:** Szöveg (String), Szám (Number), Logikai (Boolean), Null, Tömb (Array - []), Objektum (Object - {}).

Számítógép-hálózatok

8.1. Hálózati alapfogalmak és Osztályozási szempontok

Alapfogalmak

- **Számítógép-hálózat:** Autonóm (önálló) számítógépek valamilyen adatátviteli módszerrel megvalósított, összekapcsolt rendszere.
- **Célja:** Kommunikáció, erőforrás-megosztás, skálázhatóság, megbízhatóság növelése.
- **Csomópont (Node):** Önálló, kommunikációra képes, saját hálózati címmel rendelkező eszköz. Lehet végpont (End-node, pl. PC, szerver) vagy köztes csomópont (Network node, pl. router, switch).
- **Adatátvitel elemei:** Közeg (kábel, levegő/rádióhullám), Jel (analóg vagy digitális fizikai mennyiség), Csatorna.

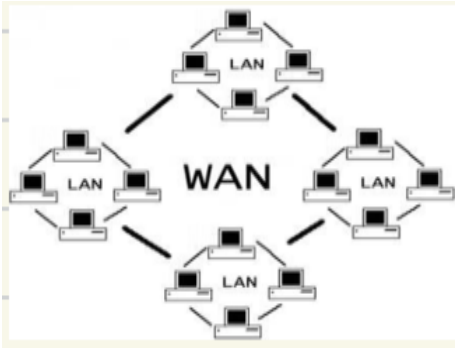
Kapcsolási módok (Hogyan jut el az adat A-ból B-be?)

1. **Vonalkapcsolt (Circuit-switched):** A kommunikáció idejére egy dedikált, fizikai vonal épül fel a két fél között (pl. hagyományos telefonhálózat).
2. **Csomagkapcsolt (Packet-switched):** Az adatokat kisebb csomagokra bontják, amelyek egymástól függetlenül, a hálózat aktuális terheltségétől függően keresnek utat a célig (pl. Internet).
3. **Üzenetkapcsolt:** A teljes üzenet egyben utazik csomópontról csomópontra (Store-and-forward elv).

Hálózatok osztályozása kiterjedés szerint

1. **BAN (Body Area Network):** Az emberi testen vagy testben lévő eszközök (pl. okosóra, pacemaker).
2. **PAN (Personal Area Network):** Személyi hálózat (pár méter, pl. Bluetooth eszközök).
3. **LAN (Local Area Network - Helyi hálózat):**
    - *Méret:* Néhány km, egy épület vagy intézmény (pl. SOHO - Small Office/Home Office).
    - *Jellemzők:* Állandó hozzáférés, nagy sebesség (10 Mbps - 10 Gbps), a hálózat az intézmény saját tulajdona.
    - *Eszközök:* Switch, Access Point, UTP kábel.
4. **MAN (Metropolitan Area Network):** Városi hálózat. Egy városrészt vagy várost fed le (pl. egyetem különböző telephelyei). LAN technológiákra épül.

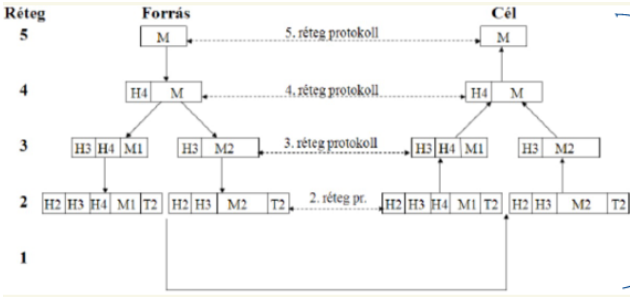
5. **WAN (Wide Area Network - Kiterjedt hálózat):**
    - *Méret:* Hatalmas, földrajzilag elkülönített területek (országok, kontinensek).
    - *Jellemzők:* Szolgáltatótól (ISP) bérelt vonalak, összetettebb technológia.



8.2. Hálózati rétegmodellek (OSI és TCP/IP)

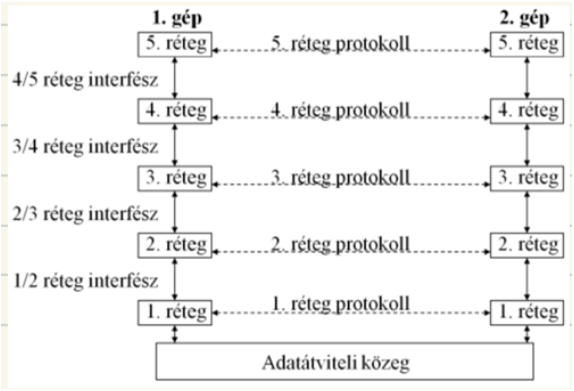
A hálózati kommunikációt rétegekre bontjuk. Minden réteg csak az alatta lévő szolgáltatásait használja, és a felette lévőnek nyújt szolgáltatást.

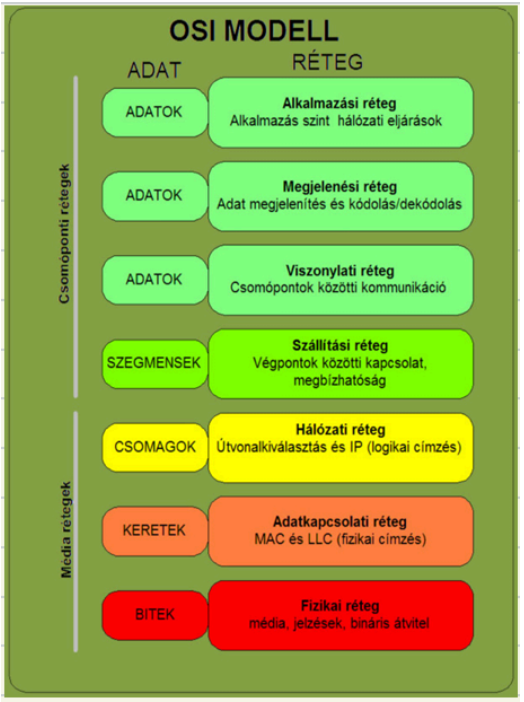
- **Beágyazás (Encapsulation):** Lefelé haladva minden réteg hozzáteszi a saját fejlécét (Header) az adathoz. Felfelé haladva (Decapsulation) ezeket a fejléceket a fogadó gép rétegei leolvassák és eltávolítják.



Az OSI (Open Systems Interconnection) 7 rétegű modellje:

1. **Fizikai réteg (Physical):** Bitek (0 és 1) továbbítása a fizikai közegen. Elektromos, mechanikai jellemzők (pl. kábelek, feszültségszintek). *PDU (Protokoll adategység): Bit.*
2. **Adatkapcsolati réteg (Data Link):** Fizikai (MAC) címzés, hibajavítás a szomszédos csomópontok között. Két alrétege van: LLC és MAC. *PDU: Keret (Frame).*
3. **Hálózati réteg (Network):** Logikai (IP) címzés és útvonalválasztás (Routing) a hálózatok között. *PDU: Csomag (Packet).*
4. **Szállítási réteg (Transport):** Végpontok (alkalmazások) közötti megbízható vagy megbízhatatlan adatátvitel, portszámok kezelése. *PDU: Szegmens (Segment).*
5. **Viszony réteg (Session):** Felépíti, kezeli és lebontja a kommunikációs dialógusokat (munkameneteket). *PDU: Üzenet.*
6. **Megjelenítési réteg (Presentation):** Adatkonverzió, kódolás (pl. ASCII), titkosítás és tömörítés. *PDU: Üzenet.*
7. **Alkalmazási réteg (Application):** Hálózati szolgáltatásokat nyújt a felhasználói programoknak (pl. HTTP böngészéshez, SMTP e-mailhez). *PDU: Üzenet.*

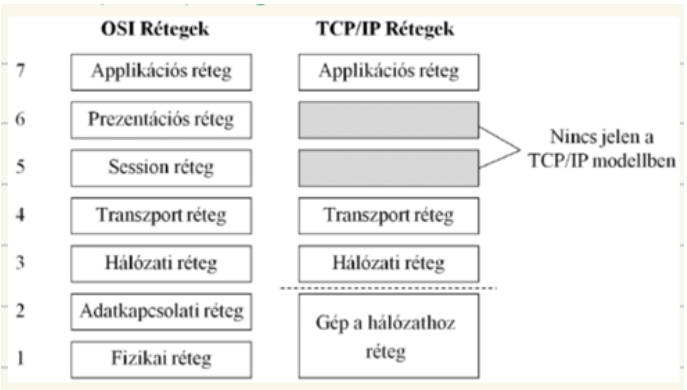




A TCP/IP (Internet) 4 rétegű modellje:

A gyakorlatban használt modell, amely összevonja az OSI rétegeit:

1. **Hálózat-elérési réteg** (OSI 1-2. réteg)
2. **Internet réteg** (OSI 3. réteg - IP protokoll)
3. **Transzport réteg** (OSI 4. réteg - TCP, UDP)
4. **Alkalmazási réteg** (OSI 5-6-7. rétegek egybevonva)



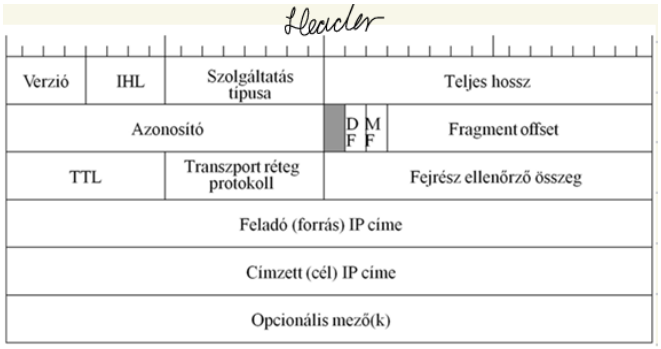
Hibrid referenciamodell - Ezt használjuk

8.3. IP technológia és Címzési rendszere

Az IP (Internet Protocol) egy hálózati rétegbeli, összeköttetés-mentes (datagram alapú) protokoll.

IP Csomag (Datagram) fejléce (Header):

Tartalmazza: Verzió (IPv4/IPv6), Teljes hossz, TTL (Time To Live - Élettartam, nehogy végtelen ciklusba essen a csomag), Protokoll (pl. TCP vagy UDP kódja), **Forrás IP cím**, **Cél IP cím**.



IP Címosztályok (IPv4 - 32 bit):

A 32 bites IP címet 4 darab 8 bites (0-255) decimális számként írjuk fel. A cím két részből áll: Hálózat azonosító (Network ID - N) és Gép azonosító (Host ID - H).

- **A Osztály:** Kezdőbit 0. Első bájt tartománya: 0-127. Alap maszk: /8 (N.H.H.H). Hatalmas hálózatoknak (16 millió gép).
- **B Osztály:** Kezdőbitek 10. Első bájt tartománya: 128-191. Alap maszk: /16 (N.N.H.H). Közepes hálózatoknak (65 ezer gép).
- **C Osztály:** Kezdőbitek 110. Első bájt tartománya: 192-223. Alap maszk: /24 (N.N.N.H). Kis hálózatoknak (254 gép).

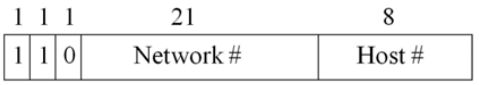
/n	Decimális	/n	Decimális
/1	128.0.0.0	/17	255.255.128.0
/2	192.0.0.0	/18	255.255.192.0
/3	224.0.0.0	/19	255.255.224.0
/4	240.0.0.0	/20	255.255.240.0
/5	248.0.0.0	/21	255.255.248.0
/6	252.0.0.0	/22	255.255.252.0
/7	254.0.0.0	/23	255.255.254.0
/8	255.0.0.0	/24	255.255.255.0
/9	255.128.0.0	/25	255.255.255.128
/10	255.192.0.0	/26	255.255.255.192
/11	255.224.0.0	/27	255.255.255.224
/12	255.240.0.0	/28	255.255.255.240
/13	255.248.0.0	/29	255.255.255.248
/14	255.252.0.0	/30	255.255.255.252
/15	255.254.0.0	/31	255.255.255.254
/16	255.255.0.0	/32	255.255.255.255

IP címosztályok:

A osztály:

B osztály:

C osztály:



- Osztályozási (első bájt) szabály:

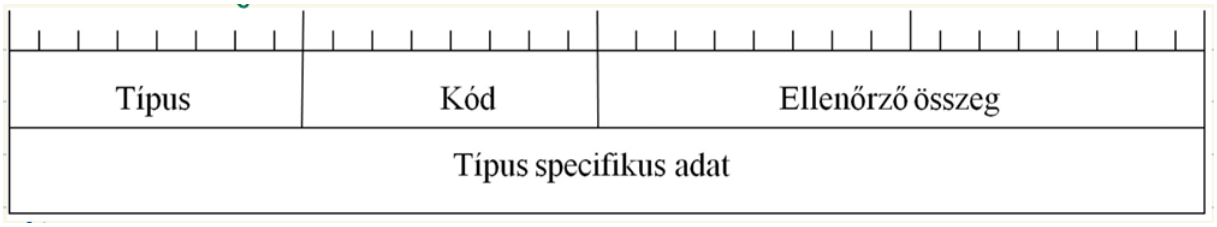
Osztály	Kezdőbit(ek)	1. Bájt értéke	Prefix	Hálózat	Bcast	N számossága	H számossága
A	0	0 - 127	/8	N.0.0.0	N.255.255.255	126 !!!	256 * 256 *256 – 2
B	01	128 - 191	/16	N.N.0.0	N.N.255.255	64 * 256	256 * 256 - 2
C	110	192 - 223	/24	N.N.N.0	N.N.N.255	32 * 256 * 256	256 - 2

IP címkezelés (példák, IPv4):

	IP	M	N	H	Megjegyzés
a)	13.194.58.11	255.0.0.0	13.0.0.0	0.194.58.11	N és H
b)	13.194.58.12	/8	13.0.0.0	0.194.58.12	N és H
c)	13.194.58.255	/24	13.194.58.0	0.0.0.255	N és speciális H
d)	13.194.58.16	/28	13.194.58.16	0.0.0.0	N és speciális H
e)	192.168.31.107	255.192.0.0	192.128.0.0	0.40.31.107	N és H
f)	192.169.31.108	/10	192.128.0.0	0.41.31.108	N és H
g)	192.170.31.255	/20	192.170.0.0	0.0.31.255	N és speciális H
h)	192.170.32.0	255.255.224.0	192.170.32.0	0.0.0.0	N és speciális H
i)	201.69.254.255	255.255.128.0	201.69.128.0	0.0.126.255	N és H
j)	223.69.255.255	/16	223.69.0.0	0.0.255.255	N és speciális H

ICMP (Internet Control Message Protocol):

Az IP vezérlő mechanizmusa. Mivel az IP önmagában nem garantálja a célba érést, az ICMP hibaüzeneteket és diagnosztikai adatokat küld (pl. Destination Unreachable vagy a jól ismert ping parancs visszhang kérése).



8.4. Forgalomirányítás (Routing)

A csomagok célba juttatása hálózatokon keresztül a legoptimálisabb útvonalon. A routerek (útválasztók) egy **Routing táblát** használnak, amely megmondja, hogy egy adott célhálózat felé melyik kimenő interfészen (Next hop) kell továbbítani a csomagot.

Célhálózat	Netmask	Kimenő interfész	Következő csomópont (next hop)	Metrika

Forgalomirányítás fajtái:

1. **Statikus Routing:** A rendszergazda kézzel, manuálisan tölti fel a routing táblát. Kicsi, nem változó hálózatoknál jó.
2. **Dinamikus Routing:** A routerek egy routing protokoll segítségével automatikusan megtanulják a hálózat szerkezetét egymástól.

- *Belső (IGP):* Egy autonóm rendszeren (pl. egy cég hálózatán) belül. (Protokollok: RIP, OSPF, EIGRP).
- *Külső (EGP):* Különböző autonóm rendszerek (pl. internetszolgáltatók) között. (Protokoll: BGP).

Dinamikus Routing Algoritmusok:

- **Távolságvektor alapú (Distance Vector - pl. RIP):** A Bellman-Ford algoritmust használja. A routerek csak a közvetlen szomszédjaiknak küldik el a teljes táblájukat. Lassan reagál a változásokra, előfordulhat "végtelenig számolás" hibája. (Metrika: ugrások száma/Hop count).

- **Algoritmus lépései:**
  1. Csomópont$_i$ mindegyik k szomszédjától megkapja a $D(k, j)$ értéket.
  2. Mindegyik csomópont$_k$ szomszédtól érkezett $D(k, j)$ érték alapján távolság metrika számolás: $X_{i,k,j} := d(i,k) + D(k, j)$
  3. Ha $X_{i,k,j} < D(i, j)$, akkor csomópont$_k$ előnyösebb úton van, ezért $D(i, j) := X_{i,k,j}$
  4. Ciklusidő várakozás, majd folytatás 1. lépésnél.

- Az eljárás véges számú lépés után optimális utat szolgáltat a forrás és cél csomópontok között.

- **Link-állapot alapú (Link-State - pl. OSPF, IS-IS):** A Dijkstra (SFP) algoritmust használja. Minden router ismeri a hálózat *teljes* topológiáját. Gyorsan reagál a hálózati változásokra, megbízhatóbb, de a számítás komplexebb.

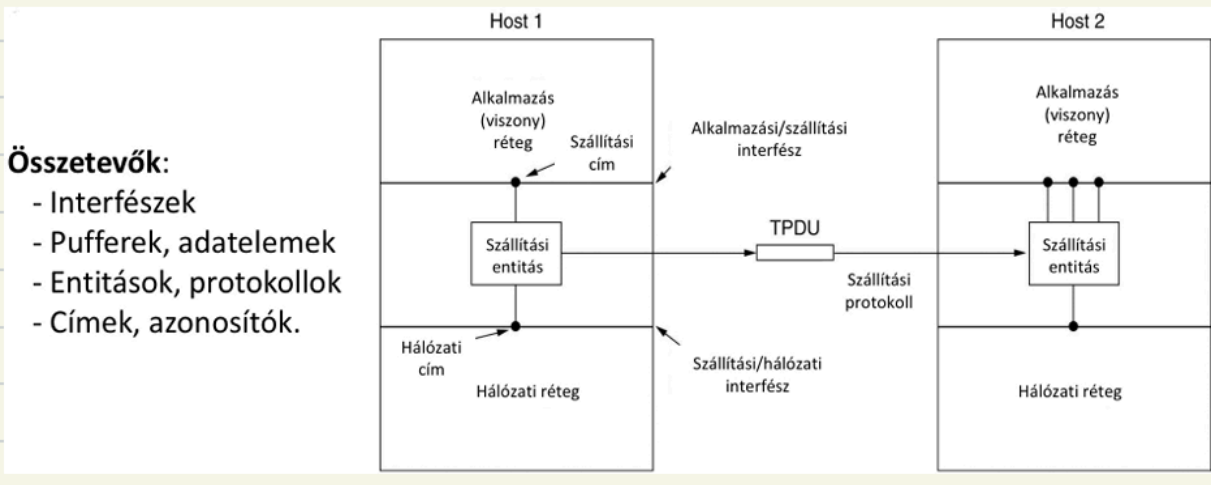
Mechanizmus ismételt lépései:

1. Szomszédok felfedezése.
2. A szomszédok felé vezető kapcsolat (link) költségének mérése (érzékelés és számolás).
3. PDU készítése a mérési eredményekről.
4. A készített PDU küldése (update) a hálózati forgalomirányítási tartomány összes útválasztójának, többes formájában (elárasztás).
5. Minden router ismeri a teljes hálózati topológiát, és ki tudja számítani (pl. Dijkstra algoritmussal) a többi routerhez vezető optimális utat (feszítőfa, spanning tree számítás).

Tulajdonság	OSPF	IS-IS
Routing protokoll	Dijkstra SFP	
VLSM/CIDR	Igen	
Update	Elárasztásos	
Area	Link-ek	Útválasztók
Update szállítás	IP csomagban	Adatkapcsolati keretben
Gerinc	Area 0	Több router közösen
Rugalmasság	Alacsony	Magas
Komplexitás	Magas	Nagyon magas
Security	Magas	Nagyon magas
Megbízhatóság	Magas	Nagyon magas
IPv6 támogatás	OSPFv3	Alapból, bármit

8.5. Szállítási réteg: TCP és UDP

A szállítási réteg feladata az alkalmazások közötti adatcsere biztosítása **Portszámok** (címek) segítségével. Két fő protokollja van:



1. UDP (User Datagram Protocol)

- **Jellemzői:** Összeköttetés-mentes, **NEM megbízható**. Nem garantálja a célba érést és a sorrendiséget. Nincs nyugtázás.
- **Előnye:** Nagyon gyors, kicsi az adminisztrációs terhe (Overhead: csak 8 bájtos a fejléce).
- **Hol használjuk?** Ahol a sebesség fontosabb a pontosságnál, és belefér egy-egy adatcsomag elvesztése (pl. VoIP hanghívások, videostreaming, online játékok, DNS).
- **Fejléce:** Forrás portszám, Cél portszám, Hossz, Ellenőrző összeg.

UDP szegmens fejrész szerkezete:

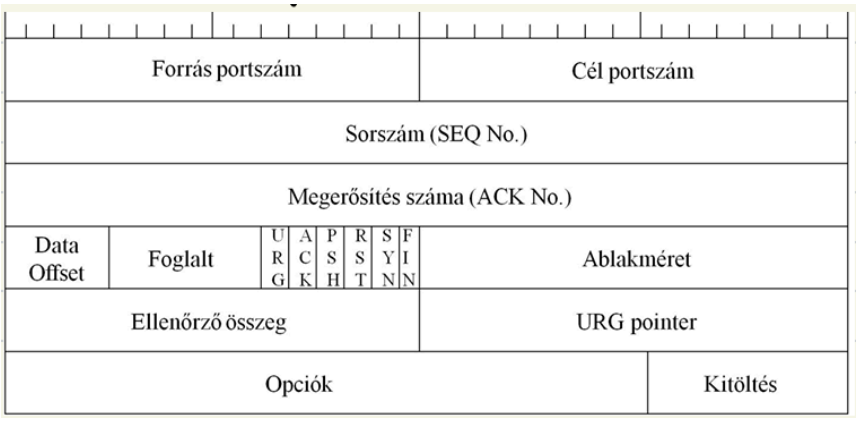
Forrás portszám	Cél portszám
Hossz (bájt)	Ellenőrző összeg

1. szó: Forrás portszám (16 bit), Cél portszám (16 bit).
2. szó: UDP szegmens hossza (16 bit), Fejrész ellenőrző összeg (16 bit).

UDP szegmens rakrész:
- Tetszőleges tartalom (bármilyen réteg PDU).
- Előnyös tűzfalakon továbbított protokollok áthidalására.
- Bithiba javítását, illetve egynél több vagy kevesebb példányos kézbesítés kezelését a rakrészben szállított PDU protokolljára bízza.

2. TCP (Transmission Control Protocol)

- **Jellemzői:** Összeköttetés alapú, **megbízható**. Garantálja az adatok sorrendhelyes és hiánytalan megérkezését. Flow control (adatfolyam-szabályozás) és Congestion control (torlódáskezelés) is van benne.
- **Fejléce:** Jóval nagyobb (20-60 bájt). Tartalmazza a **Sorszámot (SEQ)** az adatok sorrendbe rakásához, a **Nyugtázás számát (ACK)** a visszaigazoláshoz, és az **Ablakméretet** a forgalomszabályozáshoz.
- **Hol használjuk?** Amikor minden egyes bitnek pontosan és sorrendben meg kell érkeznie (pl. HTTP weboldalak, FTP fájlátvitel, e-mail).



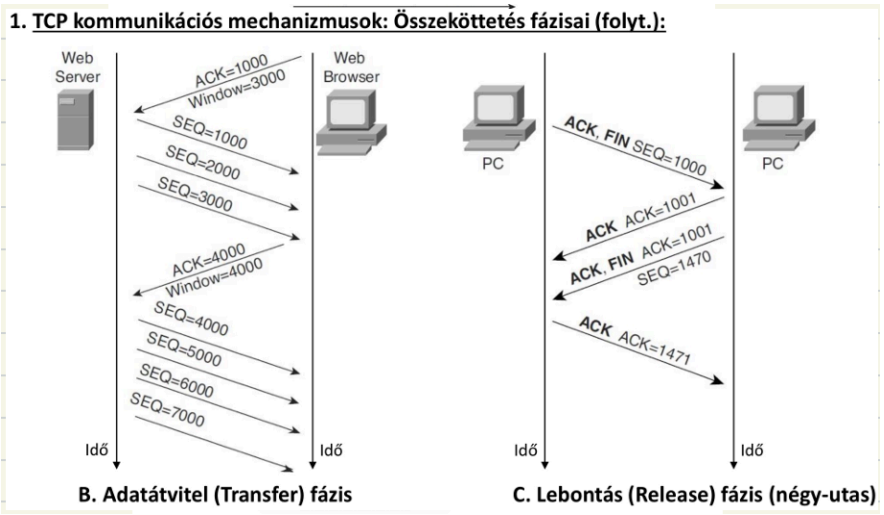
TCP Kommunikációs Fázisok:

1. **Felépítés (Háromutas kézfogás / 3-way handshake):**
    - *Kliens → Szerver:* **SYN** (Szeretnék kapcsolódni, kezdősorszámom: X).
    - *Szerver → Kliens:* **SYN + ACK** (Rendben, vettem az X-et. Én is kapcsolódnék, kezdősorszámom: Y).
    - *Kliens → Szerver:* **ACK** (Rendben, vettem az Y-t. Kezdődhet az adatátvitel).
2. **Adatátvitel:** Folyamatosan növekvő sorszámokkal küldik a szegmenseket, amire a fogadó folyamatosan ACK nyugtákat küld. Ha nincs nyugta, a küldő újra elküldi a csomagot.
3. **Lebontás (Négyutas kézfogás):** A felek **FIN** (Finish) csomagokkal jelzik, hogy nincs több küldendő adat, amit a másik fél **ACK**-val nyugtáz.

1. TCP kommunikációs mechanizmusok: Összeköttetés fázisai:

A. Felépítés (Setup) fázis: háromutas kézfogás (three-way handshake): SYN - SYN/ACK - ACK

1. Kliens: kapcsolat kiépítésének kezdeményezése:
   - Portszámok és kezdősorszám beállítása (pl. SEQ_No = 200); Jelzőbitek: SYN=1, ACK=0.
2. Szerver: jóváhagyó válasz-üzenetet küldése:
   - Portszámok felcserélése; Saját kezdősorszám beállítása (pl. SEQ_No = 1450).
   - Nyugta sorszám beállítása: kapott SEQ érték+1 (pl. ACK_No = 201). SYN = 1; ACK = 1.
3. Kliens: jóváhagyás küldése:
   - Portszámok felcserélése; Saját szegmens-sorszám beállítása (pl. SEQ_No = 201).
   - Nyugta sorszám beállítása: kapott SEQ érték+1 (pl. ACK_No = 1451); SYN = 0; ACK = 1.



TCP és UDP Összehasonlító Táblázat

Szempont	TCP	UDP
Kapcsolat típusa	Összeköttetés alapú	Összeköttetés mentes
Felhasználó alkalmazás	Magas megbízhatóság, korlátos időérzékenység	Rövid kérelmek, időérzékeny jelleggel
Tipikus alkalmazások	HTTP, HTTPs, FTP, SMTP, Telnet, SSH	DNS, DHCP, TFTP, SNMP, RIP, VoIP
Szegmens sorrend tartás	Igen	Nem
Átviteli sebesség	Alacsonyabb, mint UDP esetén	Magasabb, mint TCP esetén
Szegmens overhead	20 … 60 B	8 B
Hasonló mezők fejrészben	Forrás port, Cél port, Ellenőrző összeg	Forrás port, Cél port, Ellenőrző összeg
Adatok stream-elése	Bájtsorozat szinten	Üzenet szinten
Költség	Kapcsolat felépítés miatt magas	Alacsony (datagram-szerű)
Adatfolyam szabályozás	Igen	Nem
Nyugtázás	Igen	Nem
Kézfogás	Három-utas (Setup), Négy-utas (Release)	Nincs
IPv6 felhasználás	Igen	Igen
Tárgyak Internete (IoT)	Korlátozottan	Igen

(Megjegyzés: A szegmens fejrész ellenőrző összegének számításánál mindkét protokoll használ egy úgynevezett "álfejrészt" (Pseudo Header), amihez az IP fejrészt (Forrás és Cél IP), a Protokoll ID-t és az L4 PDU hosszát vonják be a biztonság növelése érdekében).

Szegmens fejrész ellenőrző összeg számolási algoritmusa (UDP és TCP esetén):

- Socket = (IP cím, Port) azonosítópár.
- Biztonsági funkció célja: a kapcsolathoz tartozó Socket pár ellenőrzése
  Forrás_Socket(Forrás IP cím, Forrás Port) és Cél_Socket(Cél IP cím, Cél Port).
- Ellenőrző kód kiszámolása álfejrész segítségével.

Álfejrész (Pseudo Header) szerkezete:

- IP fejrészből származó mezők:
  - Forrás IP cím (32 bit)
  - Cél IP cím (32 bit)
  - Töltelék (0x00)
  - Protokoll ID (TCP: 0x06, UDP: 0x17)
- Szegmens fejrészből származó mezők:
  - L4 PDU hossz (fejrész + rakrész)

- Ellenőrző összeg: 12 B összege