

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

TARTALOMJEGYZÉK

| | |
|---|-----|
| Algoritmusok és adatszerkezetek I..... | 2 |
| Bonyolultságelmélet..... | 26 |
| Formális nyelvek | 38 |
| Közelítő és szimbolikus számítások..... | 51 |
| Logika és informatikai alkalmazásai / Logikai következtetési rendszerek | 60 |
| Mesterséges Intelligencia I..... | 73 |
| Operációkutatás I..... | 84 |
| Operációs rendszerek | 93 |
| Adatbázisok | 94 |
| Digitális képfeldolgozás | 109 |
| Programozás alapjai | 119 |
| Programozás I és II..... | 138 |
| Programozási nyelvek | 139 |
| Rendszerfejlesztés I..... | 143 |
| Számítógép-hálózatok | 163 |
| Számítógép architektúra | 164 |

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

TÖRZSTÁRGYAK I.

Algoritmusok és adatszerkezetek I.

1. Tétel

Időigény (Legrosszabb eset): Egy algoritmus időigénye $T(n)$, ha az algoritmus tetszőleges n méretű inputon $T(n)$ időben megáll. Egy algoritmus időigénye függ az input rendezettségtől és hosszától, ezért szeretünk egy biztos felső korlátot mondani (legrosszabb eset).

- Mohó algoritmus

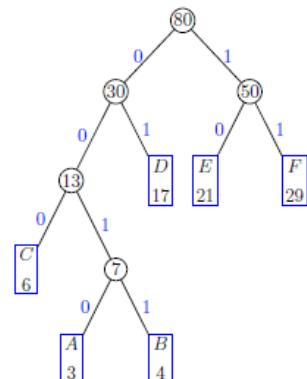
A mohó algoritmus minden lépésben optimálisnak látszó választást teszi, azaz lokálisan hoz legjobb döntést. Nem minden problémára adható mohó megoldás! De ha létezik mohó megoldás az nagyon hatékony, és két tulajdonsággal biztosan rendelkezik: optimális részstruktúrával és mohó választás tulajdonsággal. A mohóalgoritmus részfeladatokra bontja az adott feladatot, ilyenkor az a cél, hogy a mohó választás egyetlen részproblémát eredményezzen, amelynek optimális megoldásából következik az eredeti probléma optimális megoldása, ilyenkor optimális részstruktúráról beszélünk. A mohó választás tulajdonság azt jelenti, hogy a lokálisan optimális megoldások a globálisan optimális megoldáshoz vezetnek →lokális megoldás egyben a globális megoldás.

A mohó algoritmus tervezésének lépései a következők:

1. Fogalmazzuk meg az optimalizációs feladatot úgy, hogy minden egyes döntés hatására egy megoldandó részprobléma keletkezzen.
2. Bizonyítsuk be, hogy minden részprobléma az eredeti problémának, amely tartalmazza a mohó választást, tehát a mohó választás minden biztonságos.
3. Mutassuk meg, hogy a mohó választással minden részprobléma keletkezik, amelynek egy optimális megoldásához hozzájárul a mohó választást, az eredeti probléma egy optimális megoldását kapjuk.

Példák: Tankolás → adott egy út, a-tól b-ig, az üzemanyagtartály n liter, minimalizáljuk a tankolások számát, úgy, hogy ne fogyjon el az üzemanyag. Részprobléma: céltól vissza felé nézzük meg, hogy hol muszáj tankolni majd ellenőrizzük, hogy az előtt muszáj-e még valahol?

Huffman kódolás → Cél: egy adatállomány tömörítése adatvesztés nélkül. Bináris karakterkódokkal működik. Felírjuk a fát, úgy, hogy a betűértékeket sorra felvesszük, balra kisebb, jobbra nagyobb, a „szülő” pedig a gyerekek összege. Ha egy olyan érték következik, ami nem illik a fába, új fát indítunk vele. Mikor kész a fa, bal ágakra 0, jobb ágakra 1-t írunk, és leolvassuk a betűk útvonalát a gyökértől a betűig, ez lesz a kódjuk. 1. ábrán például a=0010, b=0011 stb.



1. ábra - Huffman kódolás

- Oszd meg és uralkodj

A feladatot több [diszjunkt = nem átfedő] részfeladatra osztjuk, amelyek hasonlóak az eredeti feladathoz, de méretük kisebb, rekurzív módon megoldjuk a részfeladatokat, majd összevonjuk ezeket a megoldásokat, hogy az eredeti feladatra megoldást adjanak. Felosztás: hogyan osztjuk a feladatot több részfeladatra. Uralkodás: a részfeladatokat rekurzív módon megoldjuk. Ha a részfeladatok mérete elég kicsi, akkor közvetlenül megoldja a részfeladatokat. Összevonás: a részfeladatok megoldásait összevonjuk az eredeti feladat megoldásával. (Elméletben) minden oszd-meg- és-uralkodj algoritmust le lehet iteratívan és rekurzívan is programozni. A rekurzió

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

olyan művelet, mely végrehajtásakor a saját maga által definiált műveletet, vagy műveletsort hajtja végre ezáltal önmagát ismétli. Egy rekurzív algoritmusnak két része van:

Alapeset: a megoldást már előre tudjuk vagy könnyen kiszámítható (ekkor nincs rekurzív hívás)

Rekurzív eset: a megoldást úgy kapjuk, hogy a probléma más, általában kisebb példányainak megoldását használjuk fel. Csinálunk belőle minden kisebbet, amíg el nem érjük az alapesetet.

Rekurzív implementáció előnye: olvashatóbb kód, intuitív megvalósítás

Rekurzív implementáció hátrányai: memóriaallokáció eljáráshívásokra (stack), nehezebb debugolni.

Példa:

Bináris keresés → Felosztás: n elemű sorozatot felosztjuk két $(n-1)/2$ elemű részsorozatra.

Uralkodás: elég az egyik részsorozatban kulcsot keresni, tegyük rekurzívan. Összevonás: megtalált indexet adjuk vissza. Két alapeset van, az egyik, hogy megtaláljuk az elemet: key ==

a [mid] → return mid, a másik pedig amikor nincs benne a keresett elem a tömbben: r <= 1 → return -1. Attól függően, hogy a középső elemről merre van a keresett elem, keressük abban a résztömbben rekurzívan (rekurzív eset). return binarySearch(a, alsohatar, felsohatar, key);

Futásidje a bináris keresésnek: $T(n) = \sum_{i=0}^{\log_2 n} c = \theta(\log n)$

Rendező algoritmus → adott egy n elemű tömb, amiben összevissza vannak a számok, addig osztjuk több részre, míg végül egyesével állnak az elemek, ekkor párban sorba tesszük őket, és végül összefeszüljük a listákat.

Ha $n = 1$ return A. Rekurzívan alkalmazzuk a lista két felére: MERGESORT ($A[1, \dots, n/2]$) és MERGESORT ($A[n/2 + 1, \dots, n]$). A 2 listát „fésüljük össze”.

- Dinamikus programozás

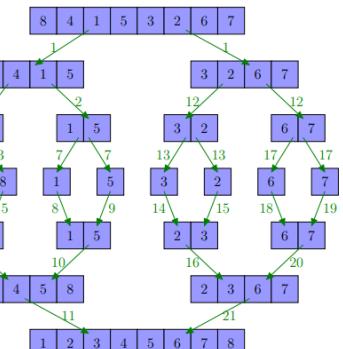
Részfeladatokra való osztással oldjuk meg a problémát.

Dinamikus Programozás, ha a részproblémák nem függetlenek,

azaz közös részproblémáik vannak (optimalizálási feladatoknál tipikus!). Alapgondolat: a már megoldott részproblémák optimális megoldásának értékét memorizáljuk és ha még egyszer fel kell használni felidézzük. DP minden egyes részfeladatot és annak minden részfeladatát pontosan egyszer oldja meg, az eredményt egy táblázatban tárolja, és ezáltal elkerüli az ismételt számítást, ha a részfeladat megint felmerül. Azaz gyorsabb, mint a rekurzív algoritmus, de nagyobb a tárígyene. Iteratív megvalósítás (táblázatkítoltés): Minden részmegoldást kiszámolunk, alulról felfele (bottom-up) építkezünk Rekurzív megvalósítás memorizálással: Részmegoldásokat kulcs-érték formájában tároljuk (feljegyzéses módszer), felülről lefelé (top-down) építkezünk. Csak akkor használjuk, ha nem kell minden megoldást kiszámolni!

Megoldás lépései:

- Adjuk meg a rekurzív megoldást (alap- és rekurzív eset).
- A rekurió alapján építsük fel „lentről felfelé”: az algoritmus kezdjen az alapesettel, majd a teljes megoldásig egy jó sorrendben számolja ki a részproblémákat.
 - **A részproblémák beazonosítása:** milyen különböző rekurzív hívások jöhettek szóba? Milyen inputokat kaphat a program?
 - **Az adatszerkezet kiválasztása a részproblémák megoldásainak memorizálásához:** A legtöbb esetben minden részprobléma azonosítható néhány egész számmal, így pl használhatunk egy több dimenziós tömböt. Néha viszont ennél bonyolultabb adatszerkezetre van szükség.
 - **Idő- és tárígyen:** A lehetséges részproblémák száma ad egy tárígyent. Az időigényt a részproblémák számának és a részproblémák kiszámolásához szükséges időnek a szorzataként kapjuk.



2. ábra - Rendező algoritmus

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- **Részproblémák közötti függőségek azonosítása:** Az alapesetek kivételével az összes rekurzív részprobléma függ a többiből. De melyektől? Jó taktika: rajzold le egy egyszerű példán!
- **Jó kiértékelési sorrend megadása:** Olyan sorrendbe kell rendezni a részproblémákat, hogy minden probléma mire sorra kerül, addigra minden, amitől függ, már ki legyen számolva.

A DP nem csak táblázatkitöltés! Ha az adatszerkezet megválasztása, a rekurzív összefüggés vagy a hívási sorrend rossz, az egész algoritmus rossz!

Példák:

Pénzváltási probléma → bontsuk részproblémáakra, azaz menjünk el egészen a legkisebb értékű érméig, és határozzuk meg, hogy minimum hány érme szükséges az adott összeghez, folyamatosan haladunk a nagyobb érték felé, amihez felhasználjuk a kisebb értékről meghatározott információkat. Futásidő: $O(Fn)$. 9 Ft legkevesebb hány érmével fizethető ki?

Iteratív megvalósítás (táblázatkitöltés):

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

3. ábra - összegekhez szükséges érmék darabszáma

Hátizsák probléma (ismétlődő tárgyakkal) → megvan a hátizsák kapacitása és a tárgyak súlya, illetve értékei. Cél minél nagyobb értéket belepakolni. Elindulunk a kisebb súlyuktól, és felépítjük, hogy adott súly, hogy lehet a legmagasabb értékű, és azt tároljuk. Végig megyünk az összes tárgyon, és összegzéssel megkapjuk, mi a legoptimálisabb bepakolás. futásidő: $O(Sn)$

| | | | | |
|-------|----|----|----|---|
| i | 1 | 2 | 3 | 4 |
| S_i | 6 | 3 | 4 | 2 |
| E_i | 30 | 14 | 16 | 9 |

S=10

5. ábra - tárgyak súlyai és értékei

| | | | | | | | | | | |
|---|---|---|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0 | 0 | 9 | 14 | 18 | 23 | 30 | 32 | 39 | 44 | 48 |

4. ábra - súlyok és maximális értékeik

- Rendező algoritmusok

Merge Sort

- **Algoritmus:**

$MERGESORT(A[1, \dots, n])$

1. Ha $n = 1$ return A

2. Rekurzívan alkalmazzuk a lista két felére: $MERGESORT(A[1, \dots, [n/2]])$ és $MERGESORT(A[[n/2] + 1, \dots, n])$

3. A 2 listát „fésüljük össze”

Példa: 2.ábra előző (4.) oldalon

- **Időigény:** $O(n \log n)$ (legrosszabb, legjobb és átlagos esetben is)

• **Tárigény:** $O(n)$ plusz tárat igényel, ha az adathalmazt tömbben tároljuk (ugyanannyit, mint a rendezendő input elemszáma) ezért **nagy elemszámú tömbben tárolt inputhoz nem ajánlott a használata**

• A lehető legjobb választás, ha láncolt listát kell rendezni, mert ekkor csak $O(1)$ a plusz tárigény

• Stabil rendezés, ami azt jelenti, hogy az inputban két egyenlő elem eredeti egymáshoz képesti sorrendje megmarad a rendezés után is

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Quick Sort

- **Paradigma:** Divide & Conquer

1. **Divide:** Osszuk fel a tömböt két résztömbre egy x pivot elem szerint a következőképpen: azok az elemek, amik kisebb egyenlőek a pivotnál legyenek előtte, amik nagyobb egyenlőek nála, legyenek utána.(A pivottal egyenlő elemek bármelyik résztömbbe kerülhetnek.)



2. **Conquer:** Rekurzívan rendezzük a két résztömböt.

3. **Combine:** Triviális.

- **Algoritmus:**

| | |
|--|--|
| <pre>QuickSort(A, p, r) if (p < r) q = Partition(A, p, r) QuickSort(A, p, q-1) QuickSort(A, q+1, r)</pre> | <pre>Partition(A, p, q) x=A[p] i=p j:=r while (true){ while (A[j]>=x&&(i<j)) j:=j-1 while (A[i]<=x&&(i<j)) i:=i+1 if (j<=i) break else csere(A[i],A[j]) } csere(A[p],A[j]) return j</pre> |
|--|--|

- Meghívása: $QuickSort(A, 1, n)$
- **Időigény:** $O(n^2)$ (legrosszabb eset), $O(n \log n)$ (legjobb és átlagos eset)
- **Tárigény:** $O(\log n)$ plusz tárat igényel
- Nem stabil rendezés
- A gyakorlatban általában kétszer olyan gyors, mint a Merge Sort

Példa: <http://www.inf.u-szeged.hu/~kgelle/sites/default/files/upload/alga-gyak-09.pdf>

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Counting Sort

- Akkor használjuk, ha a tömbben legfeljebb k -félé érték szerepel

- **Algoritmus:**

CountingSort($A[1, \dots, n]$, k)

1. Hozzunk létre egy k elemű tömböt és számoljuk bele össze, melyik elemből mennyi van
2. Módosítsuk úgy az értékeket, hogy minden tömbelem az ōt megelőzők összegét tartalmazza
3. Tegyük minden input elemet a helyére a kimenetben úgy, hogy végigmegyünk az inputon és a segédben található indexre tessük, majd növeljük a számlálóját

- **Időigény:** $O(n + k)$. Akkor érdemes használni, ha $k = O(n)$, mert akkor a futásidő $O(n)$

- **Tárigény:** $O(n + k)$

- Stabil rendezés

- Külső rendezés

Példa:

In:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 6 | 4 | 1 | 3 | 4 | 1 | 4 |
|---|---|---|---|---|---|---|---|

Ebből meghatározzuk, hogy mi a k , a következőképpen: Megszámoljuk, hogy melyik elemből mennyi van, tehát 3:2, 4:3, 1:2, 6:1 és 0-tól indexelve beírjuk nagyság szerint egy $k+1$ nagyságú tömbbe. Megkapjuk, hogy: C:

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 2 | 0 | 2 | 3 | 0 | 1 |
|---|---|---|---|---|---|---|

amit úgy módosítunk, hogy minden elem a nála kisebb elemek összegét tartalmazza.

Ekkor megkapjuk a végleges segédtömbünket:

C:

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 2 | 2 | 4 | 7 | 7 |
|---|---|---|---|---|---|---|

Ezt felhasználva rendezzük az inputot, úgy, hogy vesszük sorra a számokat, megnézzük a kimeneti tömbbe oda rakjuk ōket, ahova a segédtábla mutatja (0-6. indexek, pl. a 3-as oda kerül, amit a 3.indexű elem tartalmaz, tehát a második helyre), ezután pedig növeljük a számlálót egyel. Tömb sorrend végeredménye: 1,1,3,3,4,4,4,6.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Bináris kupac

- **Bináris fa:** Olyan fa, ahol minden csúcsnak legfeljebb 2 gyereke van
- **Bináris kupac:** majdnem teljes bináris fa, amely minden szintjén teljesen kitöltött kivéve a legalacsonyabb szintet, ahol balról jobbra haladva egy adott csúcsig vannak elemek
- A fát egy tömbben reprezentáljuk, minden elem a szint szerinti bejárás szerinti sorszámának megfelelő eleme a tömbnek.
- A kupacot reprezentáló A tömbhöz két értéket rendelünk: $hossz(A)$ a tömb mérete, $kupacmeret(A)$ a kupac elemeinek száma.
- A kupac gyökere $A[0]$, a szerkezeti kapcsolatok egyszerűen számolhatóak:
 - $A[i]$ bal fia $A[2i + 1]$
 - $A[i]$ jobb fia $A[2i + 2]$
 - $A[i]$ apja $A[\lfloor(i - 1)/2\rfloor]$
- A kupac minden gyökértől különböző elemére teljesül, hogy az értéke nem lehet nagyobb, mint az apjáé. Ennek következménye, hogy a kupac minden részfájára teljesül, hogy a gyökéreleme maximális.

Heap Sort

- **Algoritmus:**

$\text{HeapSort}(A[1, \dots, n])$

1. Építsünk egy maximum kupacot az inputból
2. Ekkor a legnagyobb elem a kupac gyökerében van. Cseréljük ezt ki a kupac utolsó elemével, és hívjuk meg az eljárást az egyetlen kisebb méretű kupacra. Ha a kupactulajdonság sérül az új gyökér miatt, állítsuk helyre.
3. Ismételjük a fenti lépéseket, amíg a kupac mérete nem lesz 1

- **Időigény:** $O(n \log n)$
- **Tárigény:** $O(1)$
- Nem stabil rendezés
- Helyben rendez

Példa

Felépítünk egy fát, aszerint, hogy az inputot sorban beillesztjük. Első elem a gyökér, második és harmadik elem a gyökér jobb és bal fia, negyedik ötödik elem a bal fiú jobb és bal fia, és így tovább, amíg el nem fogynak az elemek. A fa hossza = az elemek számával. Ha megvan a fa, akkor helyreállítjuk a kupacot a következőképpen:

Hasonlítsuk össze a gyökér és a két gyereke értékét. Ha a gyökér értéke a legnagyobb, nincs további dolgunk. Ha közülük a legnagyobb a jobb gyerek értéke, akkor cseréljük meg a gyökér és a jobb gyerek értékét, majd rekurzívan állítsuk helyre a jobb részfát. Ha a bal gyerek értéke a legnagyobb, cseréljük vele, majd rekurzívan állítsuk helyre a bal részfát. Végül már csak a gyökér (maximális elem) fogja sérteni a kupactulajdonságot, ezért azt kicseréljük a fa utolsó

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

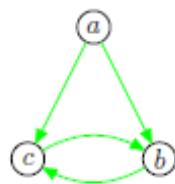
elemével (levél), és kitöröljük a levelet. Ezáltal bekerül a tömb utolsó helyére. Addig ismételjük a folyamatot, míg végül egy elem marad, az pedig a tömb első eleme lesz. (konkrét feladat: <http://www.inf.u-szeged.hu/~kgelle/sites/default/files/upload/alga-gyak-09.pdf>)

- Gráfalgoritmusok (szélességi- és mélységi keresés)

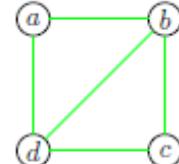
Gráf

Egy $G = (V, E)$ struktúrát gráfnak nevezünk, ahol:

- V a csúcsok halmaza.
- $E \subseteq V \times V$ az élek halmaza, vagyis csúcspárok
 - ha ez egy rendezett pár, akkor azt mondjuk, hogy a gráf irányított
 - ha nem rendezett (tehát ha (u, v) létezik, akkor (v, u) is), akkor pedig irányítatlan gráfról beszélünk
- Egy irányítatlan gráf összefüggő, ha bármely két csúcs között van út (u -ból v -be van út, ha nulla, egy vagy több egymás utáni él követésével u -ból eljuthatunk v -be).
- Egy irányított gráf erősen összefüggő, ha bármely két csúcs között van irányított út.
- Egy $G = (V, E, C)$ gráf súlyozott, ha minden élhez egy címkét/súlyt rendelünk $C : E \rightarrow$ súly, azaz C az élekhez súlyt rendel (a megengedett súlyok vagy címkék halmazából)
- Egy irányított $G = (V, E)$ gráf transzponáltja: $G^T = (V, E^T)$, ahol $E^T = \{(p, q) : (q, p) \in E\}$. (Tehát megfordítjuk az élek irányítását.)



Irányított gráf



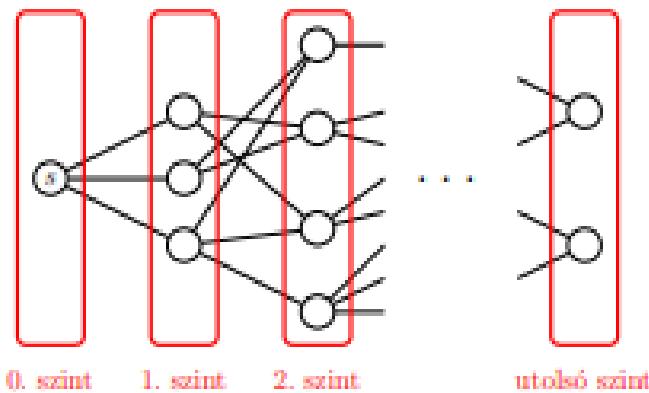
Irányítatlan gráf

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Szélességi keresés

A gráf szintről szintre való feltérképezése egy s kezdőcsúcsból.

- 0. szint: $\{s\}$
- i. szint: azok a csúcsok, amik s -ból i lépésből elérhetőek (de kevesebből nem)



Algoritmus:

```

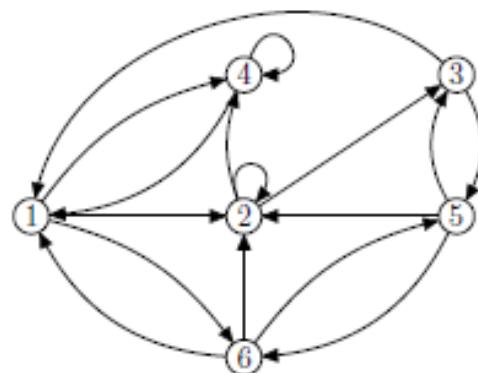
SZELETKERES(G, s)
    for (p in V){
        Szin(p):=feher
        Apa(p):=-1
        d(p):=INF}
        Szin(s):=szurke
        d(s):=0
        Apa(s):=0
        Letesit(S:Sor)
        Sorba(S, s)
        while (Elemszam(S)>0){
            Sorbol(S, u)
            for (v in Kiel(G, u)){
                if (Szin(v)==feher) {
                    Szin(v):=szurke
                    Apa(v):=u
                    d(v):=d(u)+1
                    Sorba(S, v)}}
            Szin(u):=fekete
        }
    }

```

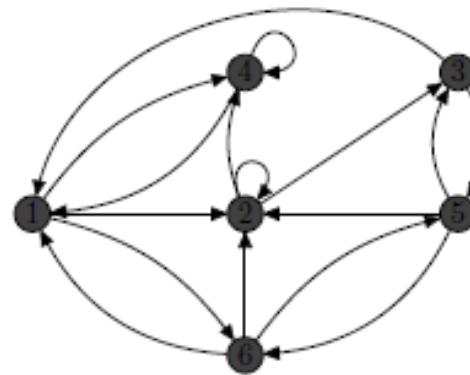
ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Példa:

Az 5-ös csúcsból indul a keresés.



Megoldás



melyik csúcsból jutunk oda →

hány lépésből jutunk oda →

| csúcs | 1 | 2 | 3 | 4 | 5 | 6 |
|-------|---|---|---|---|---|---|
| Apa | 3 | 5 | 5 | 2 | 0 | 5 |
| d | 2 | 1 | 1 | 2 | 0 | 1 |

6. ábra - Szélességi kererés feladat

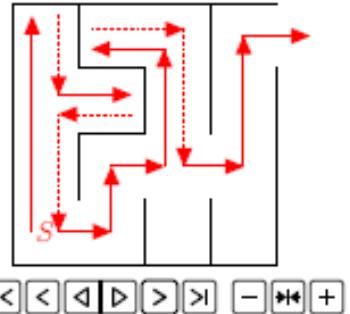
ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Mélységi keresés

Olyan, mint kijutni egy labirintusból:

- követed az utat, amíg nem jutsz el egy akadályig
 - backtrack visszafelé a kenyérmorzsák mentén, amíg egy eddig meg nem látogatott szomszédot nem találsz
 - rekurzívan feltárod ezt a szomszédot is
 - ügyelsz közben arra, hogy már meglátogatott pontba ne menj vissza még egyszer



Algoritmus:

Melykeres (G)

```

for(u in G){
    szin(u):=feher
    Apa(u):=0
}
ido:=0
for(u in G){
    if(szin(u)=feher)
        MBejar(u)
}

```

MBejar(u)

```

szin(u):=szurke
ido:=ido+1
d(u):=ido
for(v in Kiel(G,u)){
  if(szin(v)=feher){
    Apa(v):=u
    MBejar(v)}}
szin(u):=fekete
ido:=ido+1
f(u):=ido

```

Az elérési ($d(u)$) és elhagyási ($f(u)$) idők számítása nélkül is helyes az algoritmus. Utóbbit a mélyszegi keresést felhasználó algoritmusok esetében lehet szükség.

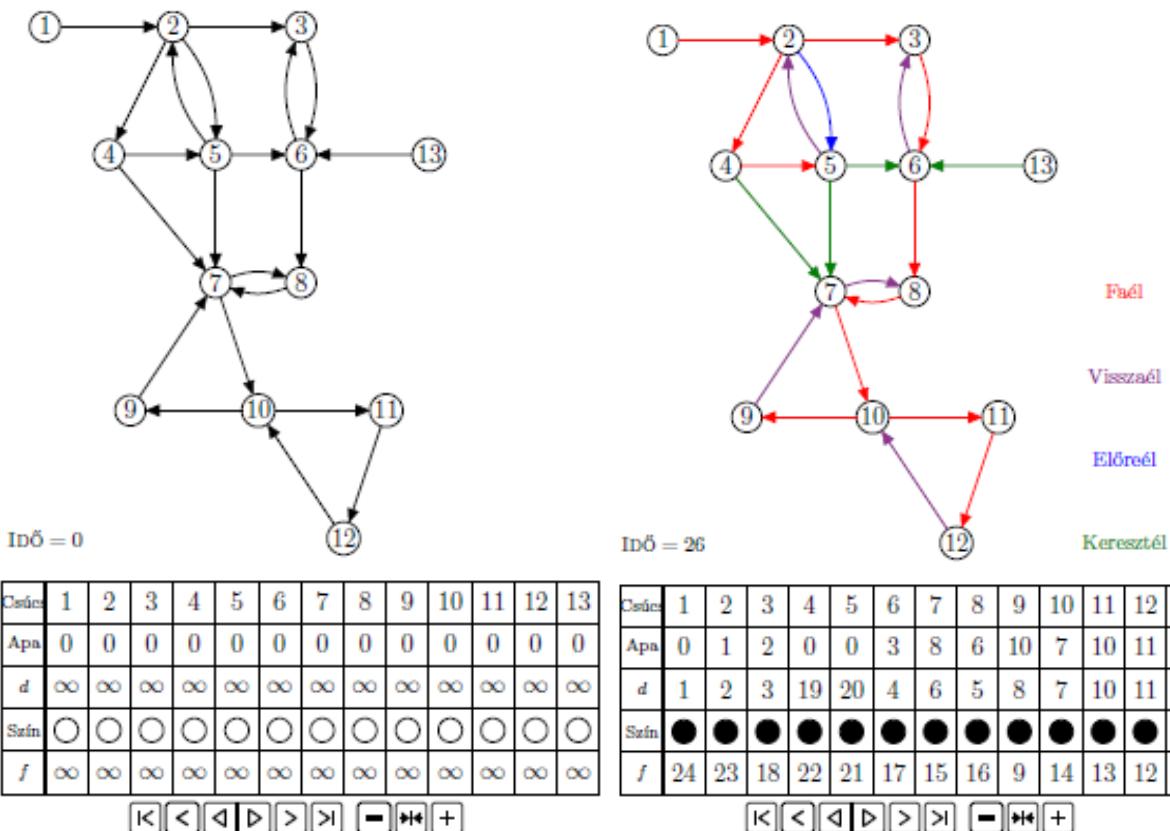
Az algoritmus egy ún. mélységi feszítőerdőt (MFE) ad eredményül.

Élek osztályozása

- **Faél:** $(u, v) \in E$ faél, ha bekerül a MFE élei közé, azaz $Apa(v) = u$.
 - **Visszaél:** $(u, v) \in E$ visszaél, ha u leszármaztatja v -nek a MFE-ben.
 - **Előreél:** $(u, v) \in E$ előreél, ha v leszármaztatja u -nak a MFE-ben és nem faél.
 - **Keresztél:** minden más esetben $(u, v) \in E$ keresztél.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Példa:



Elindulunk egynél, megyünk sorba az éleken, és ha olyan élbe jutunk, amit már egyszer feldolgoztunk, akkor annak az apját feketére színezzük, és bekerül az f-be (először a 9-nél történik ilyen). Addig lépünk vissza, ameddig olyan élhez nem jutunk, amiből még nem feldolgozott élbe juthatunk, de ekkora az apa 0 lesz, a lépésszám viszont növekszik.

Erősen összefüggő komponensek

Erősen összefüggő komponensek: a gráfban azok a maximális csúcshalmazok, amin belül bármelyik csúcsból el lehet jutni bármely másikba.

Meghatározása:

1. Számítsuk ki a MELYKERES algoritmussal az $f(u)$ elhagyási értékeket.
2. A G^T transzponált gráfra alkalmazzuk a MELYKERES eljárást úgy, hogy a pontokra a MBEJAR eljárást f szerint csökkenő sorrendben hívjuk.
3. A 2. pontban az egy mélységi feszítőfába kerülő pontok alkotnak egy erősen összefüggő komponenst.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Topologikus rendezés

Egy $G = (V, E)$ irányított körmentes gráf (Directed acyclic graph, DAG) topologikus rendezésén a V pontjainak egy olyan v_1, v_2, \dots, v_n ($n = |V|$) felsorolását értjük, amelyre teljesül, hogy minden $(u, v) \in E$ érte, u előbb van a felsorolásban, mint v .

Meghatározása:

1. A mélységi keresés algoritmusát hajtsuk végre a gráfra.
2. Az egyes csúcsok elhagyásakor beszűrjuk őket egy láncolt lista elejére.
3. A csúcsok láncolt listája adja meg a rendezési sorrendet.

Feladatokkal: <http://www.inf.u-szeged.hu/~kgelle/sites/default/files/upload/alga-gyak-10.pdf>

Minimális feszítőfák

Feszítőfa: minden csúcsot érintő, összefüggő, körmentes élhalmaz

Legyen $G = (V, E, c)$, $c : E \rightarrow \mathcal{R}^+$ egy súlyozott irányítatlan gráf. Terjessük ki a súlyfüggvényt a $T \subseteq E$ élhalmazokra: $C(T) = \sum_{(u,v) \in T} c(u, v)$. (Tehát egy élhalmaz súlya a benne lévő élek összsúlya.)

Az $F = (V, T)$ gráf minimális feszítőfája G -nek, ha

- F feszítőfája G -nek, és
- $C(T)$ minimális

Kruskal algoritmus

Algoritmus:

```
Kruskal(G,w)
    Letesit(A: halmaz)
    for(v in V)
        Halmazt-Keszit(v) //Kezdetben minden pont egy fa
    rendezzuk E eleit w szerint novekvo sorrendbe
    for((u,v) in E) a suly szerinti sorrendben
        if(Halmazt-Keres(u)!=Halmazt-Keres(v))
            A:=A ∪ {(u,v)}
            Egyesit(u,v)
```

- Minimális fák erdejét tároljuk
- minden lépésben a legkisebb, két fát összekötő élt húzzuk be (egyesítjük egyetlen fával a két fát)
- Mohó algoritmus
- Megvalósítása: Union-Find adattípussal.
- Futáridő: $O(|E| \log |E|)$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Prim algoritmus

Algoritmus:

```

Prim(G, c , r)
    for (v in V){
        d(v):=INF
        Apa(v):=0}
    d(r):=0
    Letesit(Q: ModPrisor)
    for (v in V){ SorBa(Q,v) }
    while (ElemSzam(Q)>0){
        SorBol(Q,u)
        for (v in KiEl(G,u)){
            if (c(u,v)<d(v)){
                Apa(v):=u
                d(v):=c(u,v)
                Modosit(v)}}}

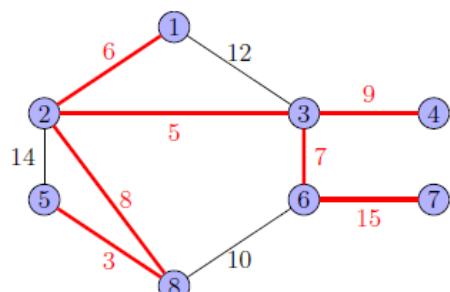
```

- Egyetlen fát növesztünk
- Tetszőleges gyökérpontból indulva
- minden lépésben új csúcsot kötünk be a fába
- Legolcsóbb éssel elérhető csúcsot választjuk
- Mohó algoritmus
- Megvalósítása: Bináris kupac adattípussal.
- Futásidő: $O(|E| \log |V|)$

Példák:

Prim algoritmus megoldása:

A 6-os csúcsból indul, megnézi hova mehet tovább, és a legkisebb költségű utat választja, így megy végig, míg végül „zsákutcába” ér, ekkor a fel nem dolgozott élük közül választ, hogy hova juthat el, mindenhol a minimális költésgű utat választva, az algoritmusnak akkor van vége, ha az összes csúcs feldolgozásra került.



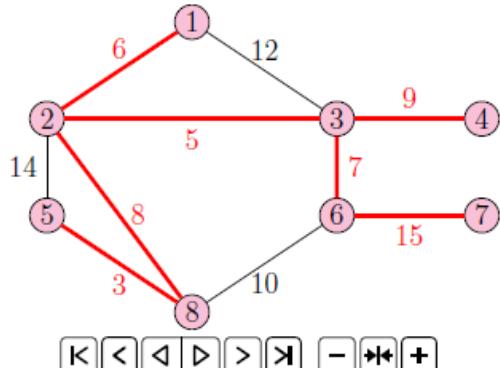
| Csúcs | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-------|---|---|---|---|---|---|----|---|
| Apa | 2 | 3 | 6 | 3 | 8 | 0 | 6 | 2 |
| d | 6 | 5 | 7 | 9 | 3 | 0 | 15 | 8 |

[◀◀◀▶▶▶] [−][+][+]

9. ábra - Prim algoritmus megoldása

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

A Kruskal algoritmus megoldása:



Az élek költségeit nézi, kiválasztja sorban a legkisebb költségűeket, és addig választja ki őket, még az összes csúcs feldolgozásra nem kerül. Akkor számít feldolgozottnak egy csúcs, ha valamilyen úton ellehet jutni hozzá, tehát összefüggő a gráf.

10. ábra - Kruskal algoritmus megoldása

Legrövidebb utak

A feladat egy súlyozott gráfból egy adott pontból kiinduló legrövidebb utak megkeresése. Az input a G súlyozott gráf és a kiindulási s pont. Outputként egy legrövidebb utak faját adunk vissza, egy Apa függvény által, továbbá a legrövidebb utak hosszait egy d függvény által. A feladatot nemnegatív élsúlyok esetén a következő Dijkstra algoritmussal oldhatjuk meg. A pontokat egy d érték szerinti Q módosítható prioritási sorban tároljuk.

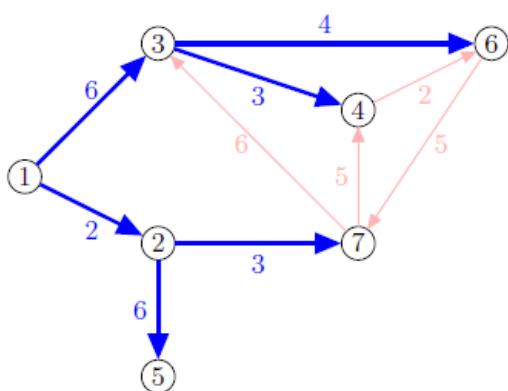
Algoritmus:

| | |
|---|---|
| <pre> Kezd(G, s) for (v in V){ d(v):=INF Apa(v):=0 Kesz(v):=0 } d(s):=0 } Kozelit(G, u, v, Q) if (d(v)>d(u)+c(u,v)){ d(v):=d(u)+c(u,v) Modosit(Q, v) Apa(v):=u } } </pre> | <pre> Dijkstra(G, s) Kezd(G, s) Letesit(Q: ModPrisor) for (v in V) SorBa(Q, v) while (Elemszam(Q)>0){ SorBol(Q, u) Kesz(u):=1 for (v in KiEl(G, u)){ if (Kesz(v)=0) Kozelit(u, v) } } } </pre> |
|---|---|

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Dijkstra algoritmus megoldása:



A feladat szerint az egyes csúcsból kell kiindulni. Oda lépünk, ahova „kevesebbe kerül” megtenni a lépést. A költségeket össze kell adni, ez lesz a d értéke. Tehát ha valahova 2 vagy több lépésből jutunk el, most nem a lépésszámokat kell számolni, hanem hogy mennyi az összköltsége az adott lépéseknek. Az algoritmusnak akkor van vége, ha az összes csúcsra megkaptuk, hogy hogy a legolcsóbb eljutni hozzá.

| Csúcs | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|----|---|
| Apa | 0 | 1 | 1 | 3 | 2 | 3 | 2 |
| d | 0 | 2 | 6 | 9 | 7 | 10 | 5 |
| Kész | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

2. Tétel
• Elemi adatszerkezetek

Tömb

- Ugyanolyan típusú elemeket tárol
- A mérete előre definiált kell legyen és nem lehet megváltoztatni futás során
- Legyen n a tömb mérete. Ekkor:
 - Elérési idő:** $O(1)$, mert az elemek egymás után folyamatosan tárolódnak a memoriában
 - Beszűrás:** $O(n)$ legrosszabb esetben, ha a tömb elejére akarunk beszűrni és minden eddigi elemet arrébb kell rakni.
 - Törlés:** $O(n)$ legrosszabb esetben, ha a tömb elejéről törlünk és minden további elemet egyelőrébb kell rakni
 - Keresés:** $O(\log n)$, ha rendezett a tömb (bináris keresés) és $O(n)$, ha nem (szekvenciális keresés)

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Láncolt listák

- minden elem egy adatból és egy (vagy több) mutatóból áll
- A mérete futás során módosítható
- Típusai:
 1. Egyirányú lista: minden elem egy adatot és egy rakkövetkező elemre mutató pointert/referenciát tárol. Az utolsó elem pedig egy NULL-ra mutat. Például:
 $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \text{NULL}$
 2. Kétirányú lista: minden elem két pointert/referenciát tárol az adat mellett. Egyet a rakkövetkező elemre, egy másikat a megelőzőre.
Előnye, hogy minden irányban bejárható és törlésnél nem kell tudnunk a megelőző csúcs címét.
Az első és utolsó eleme is NULL. Például: $\text{NULL} \leftarrow 1 \leftrightarrow 2 \leftrightarrow 3 \rightarrow \text{NULL}$
 3. Körben láncolt lista: az összes elem egy körbe van kötve. Nincs NULL elem a „végen”, az utolsó csúcs rakkövetkező pointere az első elemre mutat. Lehet egyirányú és kétirányú láncolt is.
Előnye, hogy bármelyik elemet kijelölhetjük kezdő elemnek.
- Legyen n a lista hossza. Ekkor:
 - Elérési idő: $O(n)$, mert a lista elejtől végig kell keresni
 - Beszúrás: $O(1)$, ha már azon a pozícióra vagyunk, ami után be akarunk szűrni
 - Törlés: $O(1)$, ha már a törölt elem pozícióján vagyunk és tudjuk a törölni kívánt csúcs megelőzőjének a címét (vagy ha a megelőző elemen vagyunk)
 - Keresés: $O(n)$

Verem

- A verem egy LIFO (last in, first out) adatszerkezet
- Két műveletet támogat:
 - push: egy elemet hozzáadunk az eddigiekhez úgy, hogy a verem tetejére tesszük
 - pop: az utoljára beszúrt elemet veszi ki a veremből (a tetejéről)
- Legyen n a verem mérete. Ekkor:
 - Elérési idő: $O(1)$, de csak a verem tetején lévő elemet tudjuk elérni
 - Beszúrás: $O(1)$, mert mindenkor a tetejére pakolunk
 - Törlés: $O(1)$, de csak a tetején lévő elemet tudjuk törölni

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Sor

- A sor egy FIFO (first in, first out) adatszerkezet
- Két műveletet támogat:
 - enqueue: egy új elemet adjunk hozzá így, hogy a sor végére szűrjuk be
 - dequeue: az első elemet töröljük a sorból
- Legyen n a sor mérete. Ekkor:
 - Elérési idő: $O(n)$ legrosszabb esetben
 - Beszúrás: $O(1)$
 - Törlés: $O(1)$

Prioritási Sor

- Absztrakt adatszerkezet, melyben az elemeket prioritásuk szerint tároljuk
- Hárrom műveletet támogat:
 - insert: beszűr egy elemet
 - pop: kiveszi a legkisebb prioritású elemet
 - min: a legkisebb prioritású elemet adja vissza (pl. int-ek esetén minimumot)
- Legyen n a PriSor mérete. Egy standard implementációban:
 - Min: $O(1)$
 - Beszúrás: $O(\log n)$
 - Törlés: $O(\log n)$

Példa:

Adott a 3, 6, 10, 8, 1, 9, 7 számsorozat, melyeket ebben a sorrendben tárolunk el. Adjuk meg milyen sorrendben vehetjük ki az elemeket verem, sor 'es prioritási sor adatszerkezet esetén!

Megoldás

Verem: Mivel egy LIFO adatszerkezetről van szó, így a legkésőbb berakott elem kerül ki legelőször. A sorrend tehát megfordul: 7, 9, 1, 8, 10, 6, 3

Sor: ez egy FIFO adatszerkezet, tehát az jön ki először, amit legelőször tettünk be. A sorrend megmarad: 3, 6, 10, 8, 1, 9, 7

Prioritási sor: az elemek rendezésre kerülnek az adatszerkezetben, tehát a sorrend: 1, 3, 6, 7, 8, 9, 10

Több példa: <http://www.inf.u-szeged.hu/~kgelle/sites/default/files/upload/alga-gyak-06.pdf>

- Bináris keresőfák

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Bináris fák

- **Bináris fa:** minden csúcsnak legfeljebb 2 gyereke van
- **Teljes (full) bináris fa:** olyan bináris fa, ahol minden szint teljesen ki van töltve
- **Majdnem teljes (complete) bináris fa:** olyan bináris fa, ahol maximum a legalsó szint nincs teljesen kitöltve, csak balról jobbra haladva kitöltött néhány elemig
- **Kiegyszűlyozott bináris fa:** olyan fa, ahol minden csúcs gyerekeinek részfáinak magassága maximum egyetér el.

Bináris keresőfák (BST)

A bináris keresőfa egy olyan adatszerkezet, amely olyan elemeket tárol, melyeknek kulcsa egy tejesen rendezett univerzumból való (pl. egészek). Feltessük, hogy minden elem kulcsa egyedi. Egy bináris keresőfa a következő műveleteket támogatja:

- `search(i)`: visszaadja azt az elemet, aminek a kulcsa *i*
- `insert(i)`: beszűrja az *i* kulcsú elemet a fába (ha még nem volt benne)
- `delete(i)`: törli az *i* kulcsú elemet a fából, ha az létezik

A bináris keresőfa **legfontosabb tulajdonsága**, hogy minden *x* csúcsra a bal részfájában lévő összes elem kisebb, mint az *x* kulcsa, míg a jobb részfájában lévő összes elem nagyobb, mint az *x* kulcsa.

Keresés bináris keresőfában

```
search(x, i){  
    if (key(x) == i)  return x  
    else if (i < key(x))  
        if (left(x) == NIL) return x  
        else return search(left(x), i)  
    else if (i > key(x))  
        if (right(x) == NIL) return x  
        else return search(right(x), i)}
```

Keressük az *x* gyökerű részfában az *i* kulcsú elemet.

1. Ha a gyökér az, adjuk vissza
2. Ha a *i* kisebb, mint a gyökér kulcsa, keressük a bal részfájában, ha van neki
3. Ha a *i* nagyobb, mint a gyökér kulcsa, keressük a jobb részfájában, ha van neki
4. Ha a részfa, amire lépnénk *x*-ről, nem létezik, adjuk vissza az *x*-et

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Beszúrás bináris keresőfába

```
insert(i){  
    x = search(i)  
    if(key(x)==i) return  
    y = new node()  
    key(y) = i  
    left(y)= NIL  
    right(y)= NIL  
    p(y)= x  
    if (i < key(x))  
        left(x) = y;  
    else  
        right(x) = y}
```

Az i csúcs beszúrásának lépései (továbbra is feltesszük, hogy a fában a kulcsok egyediek, így nem szűrhetjük be kétszer)

- Keressünk rá az i -re, ez ha még nincs benne a fában, vissza fogja adni azt a csúcsot, ami alá be kell szűrni az i -t (ő lesz a szülője).
- Hozzunk létre egy új csúcsot (y) és szűrjuk be:
 - Ha i kisebb, mint az x kulcsa, akkor bal gyereknek
 - Ha i nagyobb, mint az x kulcsa, akkor jobb gyereknek

Törlés bináris keresőfából

Három esetet különböztetünk meg az x csúcs törlésekor:

1. Ha x -nek nincs gyereke, töröljük, a szülő rá mutató pointerét NIL-re cseréljük.
2. Ha x -nek pontosan egy gyereke van c , mindegy, hogy bal vagy jobb gyerek volt, felemeljük az x helyére (x szülőjének gyereke c , c szülője az x szülője lesz).
3. Ha az x -nek két gyereke van (c_1 bal és c_2 jobb gyerek), megkeressük az x közvetlen rákövetkezőjét z -t, és őt tesszük x helyére a fában.
 - Ebben az esetben jegyezzük meg, hogy mivel z a c_2 gyökerű részfában van így egyszerűen rákereshetünk ($search(c_2, key(z))$).
 - Viszont mivel z az x rákövetkezője, így biztosan nincs bal gyereke, viszont jobb gyereke még lehet.
 - Ha van jobb gyereke, állítsuk rá z szülőjének pointerét (és z gyerekének szülője legyen z szülője).
 - Cseréljük ki x -et z -vel és állítsuk be a megfelelő pointereket, majd töröljük x -et, mint az 1. esetben.

(Egy másik megoldás, hogy x közvetlen megelőzőjét keressük meg és a fentieket tükrözve hajtjuk végre a cseréket.)

Futásidő

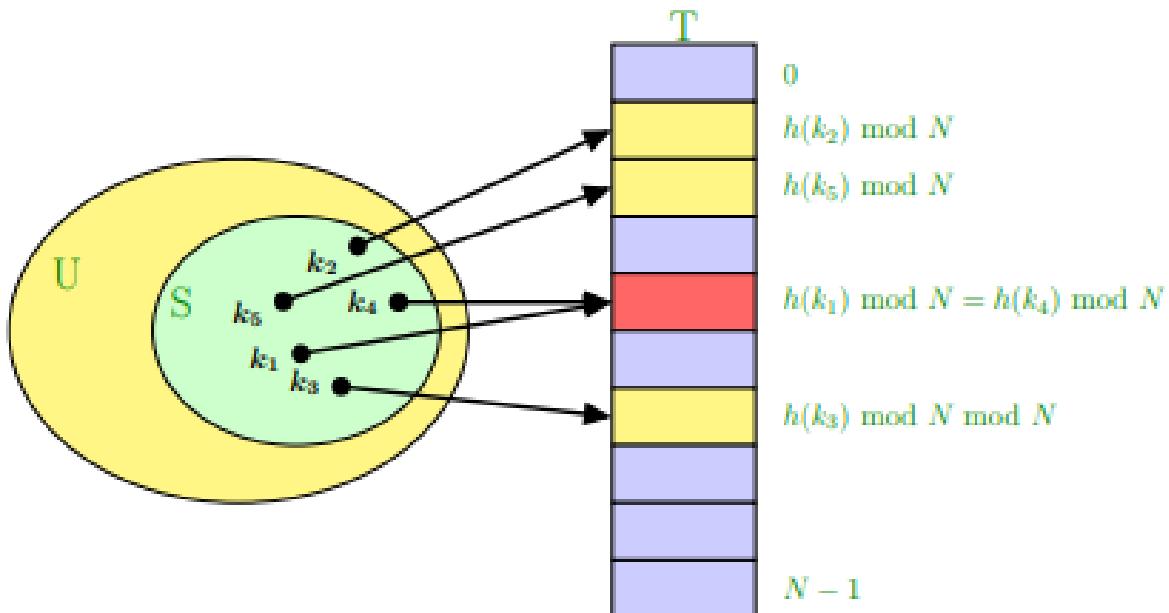
- Keresés: maximum annyit megyünk lefelé, amilyen magas a fa, tehát $O(\text{magasság})$.
- Beszúrás és törlés: mindenkettőben használjuk a keresést, a többi művelet (pointerek átállítása) konstans időben végezhető, így ez is $O(\text{magasság})$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- Hasító táblázatok

Hash táblák

- Cél: értékek egy halmazát eltárolni úgy, hogy $O(1)$ időben támogatjuk a következő műveleteket:
 - insert(i): hozzáad egy i értéket a halmazhoz
 - delete(i): törli az i értéket a halmazból
 - contains(i): megadja, hogy az i érték benne van-e a halmazban
- Hash függvény: egy objektumhoz rendeljünk egy intet úgy, hogy különböző objektumoknak általában (jó eséllyel) különböző legyen a hash kódja. Elvárás, hogy gyorsan ($O(1)$ időben) számítható legyen
- Hash tábla: egy tömb. Ha N elemű, akkor a kulcsok halmaza $0, \dots, N - 1$. Mérete futás közben változtatható
- Legegyszerűbb módszer a kulcsok $[0, N - 1]$ halmazban tartására: a hash függvény készít egy intet és ezt mod N vesszük.

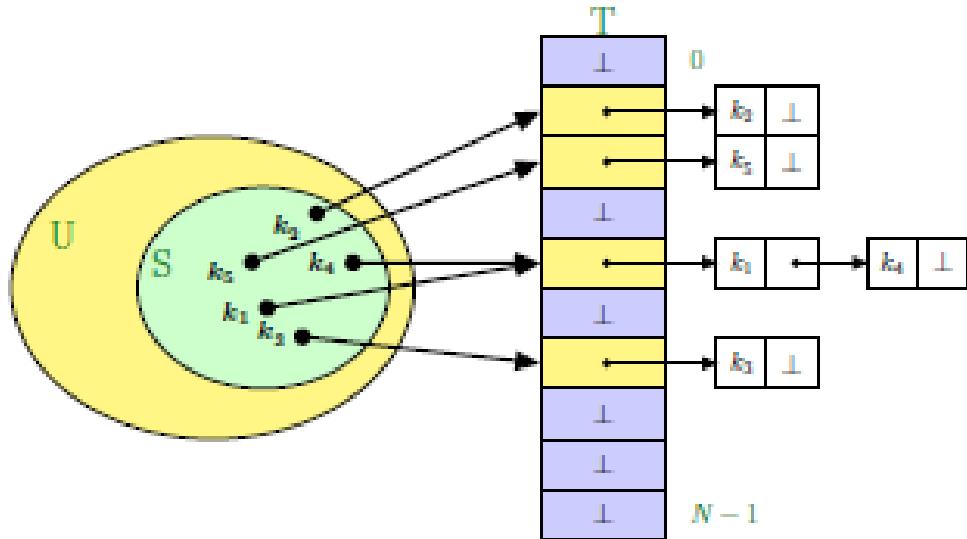


- Amikor egy elemet egy már foglalt cellába kellene rakjunk a hash kódja alapján, ütközésről beszélünk. Ezeket valahogyan fel kell oldjuk.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Chaining

- A tömb egy cellájában egy pointer van egy láncolt listára, ami az oda csatlakozókat tárolja



- Példa (a hash kód a Java String osztályának hashCode() metódusával készült):

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ |

- insert("bar") (hash: "bar".hashCode()=97299, 97299%8 = 3) és insert("red"), hash: "red".hashCode()=112785 → 1

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-----|---|---|---|---|---|-----|---|
| ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ |
| ↓ | | | | | | ↓ | |
| red | | | | | | bar | |
| ↓ | | | | | | ↓ | |

- insert("bab"), hash: 97293 → 3

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-----|---|---|---|---|---|-----|---|
| ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ |
| ↓ | | | | | | ↓ | |
| red | | | | | | bar | |
| ↓ | | | | | | ↓ | |
| bab | | | | | | ↓ | |

- Jó, ha rövidek a láncok
- Load factor: vödrök száma / elemek száma
- Ha a load factor túl nagy ⇒ több vödör és hash-eljük újra az összes eddigi elemet
- Kétszer (vagy konstansszor) annyi vödröt éri meg létrehozni
- Akkor éri meg, ha a rehash nem túl drága

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Amikor a több vödör nem segít

Ha ugyanaz a hash code

```
println( "AaBBAA" .hashCode() ) //1952508096
println( "BBAaAa" .hashCode() ) //1952508096
println( "BBBBAA" .hashCode() ) //1952508096
println( "AaAaAa" .hashCode() ) //1952508096
println( "AaBBBB" .hashCode() ) //1952508096
println( "BBAaBB" .hashCode() ) //1952508096
println( "BBBBBB" .hashCode() ) //1952508096
println( "AaAaBB" .hashCode() ) //1952508096
```

Java-ban a `String.hashCode()`:

$$s[0] \cdot 31^{n-1} + s[1] \cdot 31^{n-2} + \dots + s[n-1]$$

$$\begin{aligned} "Aa".hashCode() &= 31 * 65 + 97 = 2112 \\ &= 31 * 66 + 66 = "BB".hashCode() \end{aligned}$$

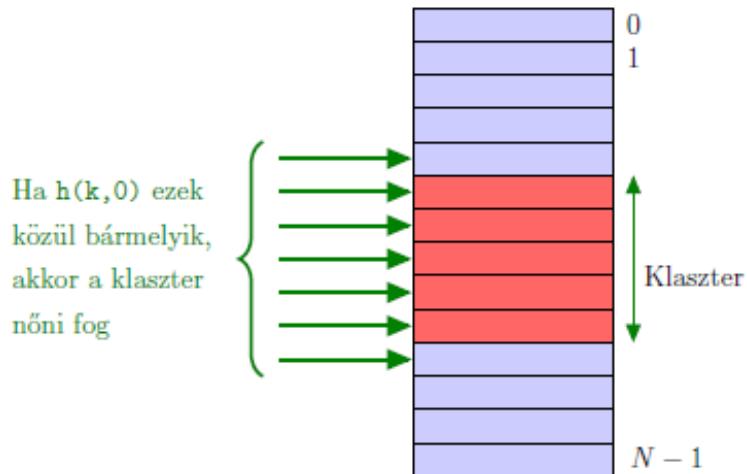
Open addressing

- minden vödörbe max egy kuleset teszünk
- A hash kód alapján számolt index csak kiindulópont
- Ha foglalt, akkor valamelyen módszerrel végigpróbálunk másokat.
Jelölés: `probe(h(k), i)`: a `h(k)` kules *i*. próbája
 - linear probing: $\text{probe}(h(k), i) = (h(k) + i) \bmod N$
 - quadratic probing: $\text{probe}(h(k), i) = h(k) + c_1 \cdot i + c_2 \cdot i^2 \bmod N$
 - double hashing: $\text{probe}(h(k), i) = h_1(k) + i \cdot h_2(k) \bmod N$
- A linear probingnél a „clustering” probléma, quadraticnál kevésbé, double hashingnál még kevésbé
- Ha a load factor kb. 0.7-re növekszik, mindegyik módszer meglassul

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Clustering (linear probing esetében)

Az egymás követő foglalt cellák csoportjai, melyek egyre nagyobbak lesznek. És minél nagyobbak, annál nagyobb eséllyel fognak tovább nőni. Vegyük a következő példát:



Ha egy ilyen klaszterbe beletalál egy hash kód, akkor végig kell próbálni a klaszter utolsó cellájáig a lehetséges helyeket. A klaszter vége után be fogjuk szúrni az elemet, így nőni fog a klaszter mérete, ami miatt pedig még nagyobb esélyünk lesz eltalálni a klaszter által fedett intervallumot.

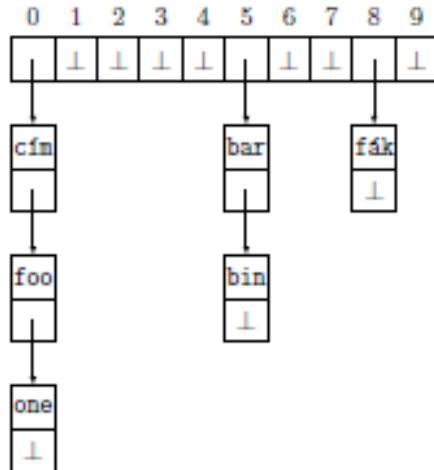
1. Feladat Szúrjuk be egy 10 hosszú hash táblába a következő objektumokat chaining, linear probing, quadratic probing ($c_1 = 0, c_2 = 1$) és double hashing módszerekkel. Az elemek két hash függvény (h_1, h_2) által adott hash kódjai a következők:

- $h_1("cím") = 210, h_2("cím") = 567$
- $h_1("foo") = 130, h_2("foo") = 132$
- $h_1("bar") = 65, h_2("foo") = 324$
- $h_1("fák") = 188, h_2("foo") = 111$
- $h_1("bin") = 785, h_2("foo") = 176$
- $h_1("one") = 960, h_2("foo") = 608$

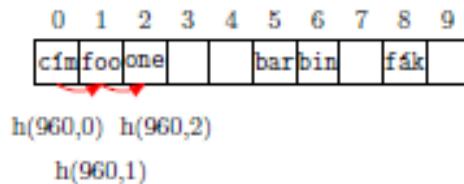
ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Megoldás chaining használatával:

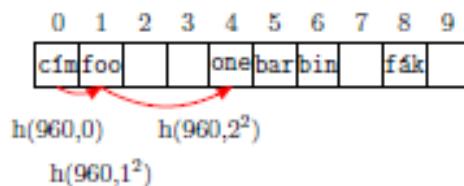


Megoldás linear probinggal:

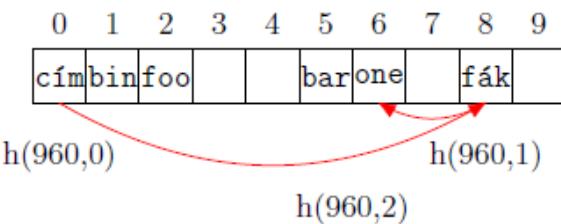


A clusteringet itt láthatjuk „akcióban”, ahogy a két részen elkezdenek „csomósodni” az elemek és ahányszor csak beletalál egy hash egy ilyen csomóba, akkor a csomó szélét fogja eggyel növelni, miután sok idő alatt kimegy a csomó széléig.

Megoldás quadratic probing-gal:



Megoldás double hashing módszerrel:



- Gráfok és fák számítógépes reprezentációja

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Bonyolultságelmélet

1. Tétel
 - Hatékony visszavezetés

Turing-visszavezetések

Legyenek A és B eldöntési problémák.

B legalább olyan nehéz, mint A , ha létezik olyan hatékony módszer, azaz f rekurzív függvény, mely az A tetszőleges x bemenetéhez hozzárendeli a B egy $f(x)$ bemenetét (példányát) úgy, hogy az A -nak x akkor és csak akkor „igen” példánya, ha B -nek $f(x)$ „igen” példánya.

Jelben: $A \leq_R B$, A Turing-visszavezethető (vagy rekurzívan visszavezethető) B -re.

Ekkor:

- ha van egy B -t eldöntő algoritmusunk, akkor A -ra is van:
function $A(x)$
 return $B(f(x))$;
- így tehát ha $A \leq_R B$ és A -ról tudjuk, hogy eldönthetetlen, akkor B is az kell legyen.

Hatékony visszavezetés

Az f függvényre további megszorításokat teszünk.

Definíció

Azt mondjuk, hogy az A probléma (polinomidőben) visszavezethető a B problémára, ha létezik olyan polinomidőben kiszámítható f függvény, hogy minden x bemenetre

$$x \in A \Leftrightarrow f(x) \in B.$$

Jelben: $A \leq_P B$.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

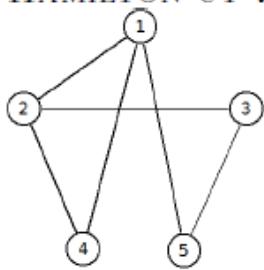
HAMILTON-ÚT $\leq_{\mathcal{P}}$ TSP(E)

HAMILTON-ÚT

- Input: G irányítatlan gráf.
- Output: van-e G -ben Hamilton-út (minden csúcsot pontosan egyszer érintő út)?

Példa

HAMILTON-ÚT visszavezethető TSP(E)-re.



| d | 1 | 2 | 3 | 4 | 5 |
|-----|---|---|---|---|---|
| 1 | 0 | 1 | 2 | 1 | 1 |
| 2 | 1 | 0 | 1 | 1 | 2 |
| 3 | 2 | 1 | 0 | 2 | 1 |
| 4 | 1 | 1 | 2 | 0 | 2 |
| 5 | 1 | 2 | 1 | 2 | 0 |

Tehát: $d_{i,j} = 1$, ha (i,j) él volt G -ben, 2 egyébként; a célérték $N + 1$, ahol N a városok száma. Ekkor ha van Hamilton-út G -ben, annak összköltsége $N - 1$; a két végpontját összekötve (ennek

PÁROSÍTÁS $\leq_{\mathcal{P}}$ SAT

PÁROSÍTÁS

- Input: $G = (V, E)$ páros gráf.
- Output: van-e G -ben teljes párosítás?

SAT

- Input: konjunktív normálformájú (ítéletkalkulus-beli) formula.
- Output: kielégíthető-e?

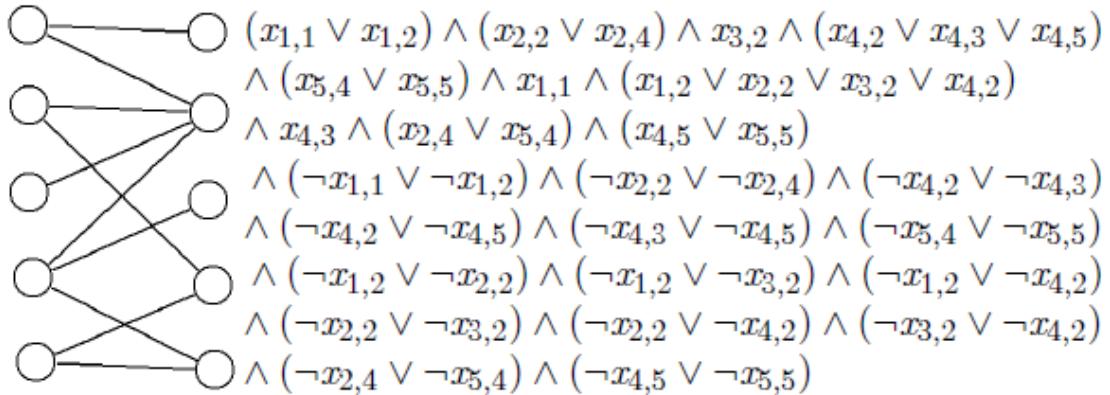
A visszavezetés

- minden $(u, v) \in E$ élhez rendelünk egy $x_{u,v}$ logikai változót.
- Intuíció: $x_{u,v}$ akkor 1, ha (u, v) párosításbeli él, 0 egyébként.
- minden u csúcsra felírjuk, hogy legalább egy rá illeszkedő élt kiválasztunk...
- ... és azt is, hogy legfeljebb egyet.
- Ez minden.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Példa



- Nemdeterminizmus

Nemdeterminizmus

Láttunk rá bizonyítékot, hogy minden „realisztikus” számítási modell szimulálható RAM-gépen, lényegi időigény-romlás nélkül.

Most bevezetünk egy nemrealisztikus számítási modellt.

Egy nemdeterminisztikus RAM-programban

- több különböző programsor is kaphatja ugyanazt a sorszámot
- egy lehetséges futás minden lépéseinben a PC által kijelölt sorszámú lehetséges utasítások egyike hajtódik végre
- (utána a PC eggyel nő vagy ugrás esetén a megfelelő értékre áll be)

A program elfogadja az inputot, ha van olyan lehetséges futás, mely ACCEPT utasítással terminál, egyébként elutasítja azt

<https://www.youtube.com/watch?v=lufECeWtN34>

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Nemdeterminisztikus generálás

Egy bit

1. $a := 0$
1. $a := 1$
2. ...

Egy n -bites szám

```
function ND( $n$ )  
1.  $r := 0$   
2. if  $n == 0$  return  $r$   
3.  $r := r + r$   
4.  $r := r + 1$   
4.  $r := r + 0$   
5.  $n := n - 1$   
6. goto 2
```

Példa: HAMILTON-ÚT

- Input: $\vec{G} = (V, E)$ (irányított) gráf. Feltehető, hogy $V = \{1, \dots, n\}$.
- Output: Van-e \vec{G} -ben Hamilton-út (azaz minden csúcsot érintő út)?

```
for  $i := 1 \dots n$  do  
     $W[i] := \text{ND}(1 + \lfloor \log n \rfloor)$   
for  $i := 1 \dots n - 1$  do  
    if  $(W[i], W[i + 1]) \notin E$  then REJECT  
for  $i := 1 \dots n$  do  
    for  $j := 1 \dots n$  do  
        if  $W[j] == i$  then BREAK  
        if  $j > n$  then REJECT  
ACCEPT
```

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Nemdeterminisztikus időigény

Egy nemdeterminisztikus program időigénye $f(n)$, ha bármely n méretű inputon tetszőleges lehetséges futás $f(n)$ lépések véget ér.

HAMILTON-ÚT időigénye

Az előző fólia algoritmusának időigénye például négyzetes:

- nemdeterminisztikusan generál n darab, egyenként $\approx \log n$ -bites számot, ez $n \log n$ lépés;
- (determinisztikusan) ellenőrzi, hogy az n szám ebben a sorrendben valóban egy séta-e a gráfban, ez n lépés;
- (determinisztikusan) ellenőrzi, hogy minden csúcs előfordul-e a sétában, ez n^2 lépés.

Megjegyzés

Az utolsó ellenőrzés hatékonyabban is elvégezhető, most a pszeudokód egyszerűségét tartottuk szem előtt.

- A P és NP osztályok

A P osztály

Definíció

P jelöli az összes polinomidőben eldönthető probléma osztályát, azaz melyek eldönthetők $\mathcal{O}(n^k)$ időkorlátos RAM-géppel, valamely konstans k -ra.

Polinom időben eldönthető problémák

ELÉRHETŐSÉG

- Adott: egy $G = (V, E)$ irányított gráf.
Feltehetjük, hogy $V = \{1, 2, \dots, n\}$.
- Kérdés: létezik-e 1-ből n -be vezető irányított út?
- Módszer:
 - $S := \{1\}$.
 - Jelöljük meg az 1 csúcson.
 - Amíg S nem üres:
 - Választunk egy $i \in S$ csúcson.
 - $S := S - \{i\}$.
 - $j = 1..n$:
Ha $(i, j) \in E$ és j nem megjelölt,
akkor $S := S \cup \{j\}$ és jelöljük meg j -t.
- Ha n megjelölt csúcs, adjunk ACCEPT választ,
különben adjunk REJECT választ.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Polinom időben eldönthető problémák

Néhány kimaradt részlet

- A bemenet megadása – pl. szomszédsági mátrix
(Függ a modelltől, de lényeges szerepet nem játszik.)
- S reprezentálása
 - sor: szélességi keresés
 - verem: egyfajta mélységi keresés

Hatókonyiság

- A mátrix minden elemét legfeljebb egyszer használjuk fel.
- Ha az egyszerű műveletek (S egy elemének kiválasztása, megjelölése, stb.) konstans időigényűek, akkor $\mathcal{O}(n^2)$.

Az **NP** osztály

Az **NP** osztályba mindenek szerint tartoznak a problémák, melyek eldönthetők polinom időigényű nemdeterminisztikus programmal.

Azaz, $L \in \mathbf{NP}$, ha van olyan M polinom időkorlátos nemdeterminisztikus program, melyre

- ha $x \in L$, akkor M -nek létezik elfogadó futása x -en, és
- ha $x \notin L$, akkor M -nek minden futása elutasítja x -et.

Példa

HAMILTON-ÚT, SAT és 3 – Színezés **NP**-beli problémák.

Mivel minden $f(n)$ időigényű program felfogható $f(n)$ időigényű nemdeterminisztikus programként is, így nyilván $\mathbf{P} \subseteq \mathbf{NP}$.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- NP teljes problémák

NP-teljes gráfelméleti problémák

FÜGGETLEN CSÚCSHALMAZ

Adott: $G = (V, E)$ irányítatlan gráf, K szám.

Kérdés: Létezik-e K elemű független csúcs halmaz?

Tétel

A FÜGGETLEN CSÚCSHALMAZ probléma NP-teljes.

Bizonyítás

Világos, hogy a probléma NP-ben van. Megmutatjuk, hogy 3SAT visszavezethető a FÜGGETLEN CSÚCSHALMAZ-ra.

Legyen φ a 3SAT egy példánya, $\varphi = c_1 \wedge \dots \wedge c_m$. A G gráf álljon m darab háromszögből, melyek a c_i tagoknak felelnek meg, s melyek csúcsai a c_i -ben lévő literáloknak felelnek meg. Ezen kívül még kössük össze érvel minden ellentétes literált. Végül legyen $K = m$.

φ kielégíthető $\Leftrightarrow G$ -ben létezik K elemű független csúcs halmaz.

FÜGGETLEN CSÚCSHALMAZ

Példa

Legyen

$\varphi = (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z)$.

Ekkor a G gráf:



NP-teljes gráfelméleti problémák

NP-teljes gráfelméleti problémák

KLIKK

Adott: $G = (V, E)$ irányítatlan gráf, K szám.

Kérdés: Létezik-e K elemű klikk (teljes részgráf)?

Tétel

KLIKK NP-teljes.

Bizonyítás (visszavezetéssel): gyakorlaton.

NP-teljes gráfelméleti problémák

Csúcslefedés

Adott: $G = (V, E)$ irányítatlan gráf, K szám.

Kérdés: Létezik-e olyan K elemű csúcs halmaz, hogy minden él illeszkedik a halmaz egy csúcsához?

Tétel

A CSÚCSLEFEDÉS probléma NP-teljes.

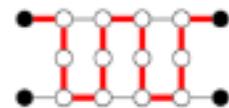
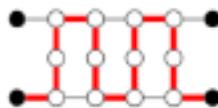
Bizonyítás (visszavezetéssel): gyakorlaton.

Ötlet

„XOR” gadget:



Minden Hamilton-út egy olyan gráfban, ami ezt a gadgetot feszített részgráfként tartalmazza, a következők egyike ezen a gráfon belül:



Hogy átláthatóbb legyen a rajzunk, ezt a gadgetet így rajzoljuk:



+ teljes 7. és 8. pdf

(előadás diákból)

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

2. Tétel

- A PSPACE osztály

A nevesített bonyolultsági osztályok

- **R, RE** – eldönthető ill. felismerhető problémák
- $\text{TIME}(f(n)), \text{NTIME}(f(n))$ – det/nemdet. $\mathcal{O}(f(n))$ időben eldönthető problémák
- $\text{SPACE}(f(n)), \text{NSPACE}(f(n))$ – det/nemdet. $\mathcal{O}(f(n))$ tárban eldönthető problémák
- $\mathbf{P} = \text{TIME}(n^k) = \bigcup_{k \geq 0} \text{TIME}(n^k)$ – polinomidőben eldönthető problémák
- $\mathbf{NP} = \text{NTIME}(n^k)$ – nemdet. polinomidőben eldönthető problémák
- $\mathbf{L} = \text{SPACE}(\log n)$ – logaritmikus tárban eldönthető problémák
- $\mathbf{NL} = \text{NSPACE}(\log n)$ – nemdet. logtárban eldönthetőek
- $\mathbf{PSPACE} = \text{SPACE}(n^k)$ – polinom tárban eldönthetőek
- $\mathbf{NPSPACE} = \text{NSPACE}(n^k)$ – nemdet. polinom tárban...
- $\mathbf{EXP} = \text{Time}(2^{n^k})\dots$

A „nevesített” osztályok sorrendje

Következmény

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP} (\subsetneq \mathbf{R})$$

Bizonyítás

$$\mathbf{L} \subseteq \mathbf{NL} \quad (\text{i})$$

$$\mathbf{NL} = \text{NSPACE}(\log n) \subseteq \text{TIME}(k^{\log n}) \subseteq \mathbf{P} \quad (\text{iii})$$

$$\mathbf{P} \subseteq \mathbf{NP} \quad (\text{i})$$

$$\mathbf{NP} = \text{NTIME}(n^k) \subseteq \text{SPACE}(n^k) = \mathbf{PSPACE} \quad (\text{ii})$$

$$\mathbf{PSPACE} = \text{SPACE}(n^k) \subseteq \text{TIME}(2^{n^k}) = \mathbf{EXP} \quad (\text{iii})$$

A **központi** kérdés, hogy $\mathbf{P} = \mathbf{NP}$ vagy sem, de a fentiek közül **egyiket** se tudjuk, hogy egyenlőség-e!

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- PSPACE-teljes problémák

További PSPACE-teljes problémák

HELYBEN EFOGADÁS

Adott: M determinisztikus program és x bemenő szó.

Kérdés: Elfogadja-e M az x -et legfeljebb $|x|$ tárban?

REGULÁRIS KIFEJEZÉSEK EKVIVALENCIÁJA

Adott: Két reguláris kifejezés.

Kérdés: Ugyanazt a nyelvet jelölik-e?

VÉGES AUTOMATÁK EKVIVALENCIÁJA

Adott: M_1 és M_2 véges **nemdeterminisztikus** automaták.

Kérdés: M_1 és M_2 ekvivalensek-e?

Tétel

A fenti három probléma mind PSPACE-teljes.

További közismert PSPACE-teljes problémák

- GÓ: input egy gó állás, melyik játékosnak van nyerő stratégiája?
- HEX: input egy hex állás, melyik játékosnak van nyerő stratégiája?
- REVERSI: input egy reversi állás...
- DÁMA: input egy dámaállás...
- AMÓBA: egy amőbaállás...
- SOKOBAN: megoldható-e az input Sokoban feladvány? (egyszemélyes!)
- RUSH HOUR: megoldható-e az input Rush Hour feladvány?
- ÉLETJÁTÉK: kipusztul-e minden cella egy adott kezdőállásból elindítva valahány lépésben? („nulla személyes”!)

QSAT és Földrajzi játék problémák vezetése a 11.pdf-ben (előadás dia).

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- Logaritmikus tárígényű visszavezetés

A logaritmikus tárban való visszavezetés

A polinomidejű visszavezetésre nézve **P** minden nemtriviális eleme „**P-teljes**”.

Bevezetünk egy (formailag) „gyengébb” visszavezetést, melyet a **P** osztályon belül alkalmazunk problémák egymáshoz viszonyított nehézségének definiálására.

Az f függvény az A eldöntési problémának a B eldöntési problémára való **logaritmikus tárígényű** visszavezetése, ha

- f kiszámítható logaritmikus tárban és
- tetszőleges I inputra $A(I) = B(f(I))$.

Ha létezik A-nak B-re való logaritmikus tárígényű visszavezetése, annak jele $\mathbf{A} \leq_{\mathcal{L}} \mathbf{B}$: A logtárban visszavezethető B-re.

Logaritmikus tárígényű (tehát mindenképp offline) algoritmusnak legfeljebb $2^{\mathcal{O}(\log n)}$, azaz polinom sok konfigurációja lehet adott input esetén; mivel a visszavezetés nem eshet végtelen ciklusba, így polinomidőben meg is kell álljon.

Vagyis ha $A \leq_{\mathcal{L}} B$, akkor $A \leq_{\mathcal{P}} B$ is.

Megjegyzés

Nem ismert, hogy $\leq_{\mathcal{L}}$ és $\leq_{\mathcal{P}}$ egybeesnek-e... (ha igen, akkor $\mathbf{L} = \mathbf{P}$)

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Nehézség, teljesség

Legyen \mathcal{C} problémák egy osztálya. Az A probléma **\mathcal{C} -nehéz** a logtáras visszavezetésre nézve, ha minden \mathcal{C} -beli probléma logárban visszavezethető A-ra.

Ha még $A \in \mathcal{C}$ is, akkor A egy \mathcal{C} -teljes probléma a logtáras visszavezetésre nézve.

Megjegyzés

Nem ismert, hogy a logtáras visszavezetésre nézve **NP-teljes** problémák ugyanazok-e, mint a polinomidejű visszavezetésre nézve **NP-teljesek**.

(A SAT egy **NP-teljes** probléma a logtáras visszavezetésre nézve is; az összes hatékony visszavezetés, melyet eddig láttunk, egyben logtáras is volt.)

A logtáras visszavezetés tranzitivitása

Tétel

A logárban való visszavezethetőség tranzitív: ha f az A-nak B-re, g pedig a B-nek C-re való logtáras visszavezetése, akkor az $f \circ g$ összetett függvény az A-nak C-re való logtáras visszavezetése.

A gond

Legyen M_f az f -et, M_g pedig a g -t kiszámító program.

Az nyilván igaz, hogy $I \in A \Leftrightarrow f(I) \in B \Leftrightarrow g(f(I)) \in C$.

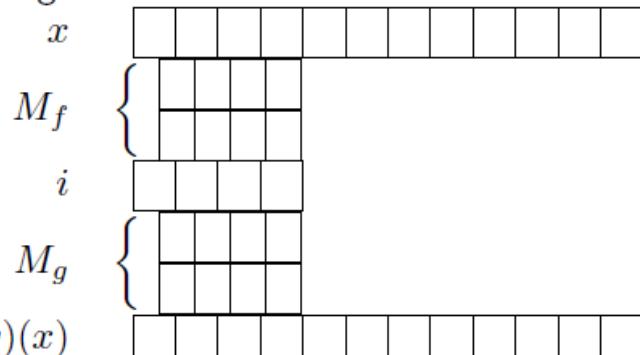
Csakhogy ha $M_f(I)$ -t munkaregiszterekbe írjuk, sokkal hosszabb lehet, mint $\log(|I|)!$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

A visszavezetés tranzitivitása

A trükk

Nem tároljuk el M_f teljes outputját, csak a legutolsóként írt output regiszter sorszámát és a belekerült értéket.



M_g -nek szüksége van egy nagyobb sorszámú regiszter tartalmára: folytatjuk M_f szimulálását.

M_g -nek szüksége van egy kisebb sorszámú regiszter tartalmára: előlről kezdjük M_f szimulálását.

- NL-teljes problémák

P-teljesség, **NL**-teljesség

Korábban láttuk, hogy **P** (így **NL**) bármely két nemtriviális problémája hatékonyan visszavezethető egymásra, így a polinomidejű visszavezetésre nézve ezen osztályokon belül a „teljesség” fogalma értelmetlenné válik.

Azt mondjuk, hogy egy probléma **NL/P-teljes/nehéz**, ha a **logtáras** visszavezetésre nézve az.

Tétel

A HÁLÓZAT-KIÉRTÉKELÉS probléma **P-teljes**.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Formális nyelvek

1. Tétel

- Véges automata fogalma és változatai

Az $M = (Q, \Sigma, \delta, q_0, F)$ rendszert **nemdeterminisztikus automatának** nevezük, ahol:

- Q egy nemüres, véges halmaz, az **állapotok** halmaza
- Σ egy ábécé, az **input** ábécé,
- $q_0 \in Q$ a **kezdő** állapot,
- $F \subseteq Q$ a **végállapotok** halmaza,
- $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ egy leképezés az **átmenetfüggvény**, pl: $\delta(q, a) = \{q_1, \dots, q_n\}$.

M konfigurációinak halmaza: $C = Q \times \Sigma^*$.

A $(q, a_1 \dots a_n)$ konfiguráció azt jelenti, hogy M a q állapotban van és az $a_1 \dots a_n$ szót kapja inputként.

Átmeneti reláció:

- $(q, w), (q', w') \in C$ esetén $(q, w) \vdash_M (q', w')$, ha $w = aw'$, valamely $a \in \Sigma$ betűre és $q' \in \delta(q, a)$.
- $(q, w) \vdash_M (q', w')$, egy lépés
- $(q, w) \vdash_M^n (q', w')$, $n \geq 0$ lépés
- $(q, w) \vdash_M^+ (q', w')$, legalább egy lépés
- $(q, w) \vdash_M^* (q', w')$, valamennyi (esetleg 0) lépés.

Az $M = (Q, \Sigma, \delta, q_0, F)$ automata által **felismert nyelven** az $L(M) = \{ w \in \Sigma^* \mid (q_0, w) \vdash_M^* (q, \lambda) \text{ valamely } q \in F \text{-re}\}$ nyelvet értjük.

Szavakkal: q_0 -ból a w hatására elérhető valamely $q \in F$ végállapot (ugyanakkor esetleg nem végállapotok is elérhetők).

Egy $M = (Q, \Sigma, \delta, q_0, F)$ automata **determinisztikus**, ha teljesül, hogy minden $q \in Q$ és $a \in \Sigma$ esetén a $\delta(q, a)$ halmaz legfeljebb egyelemű: $\forall q \in Q \forall a \in \Sigma \left(|\delta(q, a)| \leq 1 \right)$

Tétel. A **determinisztikus** automatákkal felismerhető nyelvek osztálya **megegyezik** a **nemdeterminisztikus** automatákkal felismerhető nyelvek osztályával.

⇒ irány: Ha egy nyelv felismerhető determinisztikus automatával akkor felismerhető nemdeterminisztikus automatával is, mivel a determinisztikus automata a nemdeterminisztikus automata speciális esete:

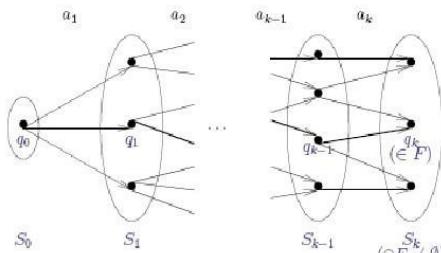
$|\delta(q, a)| \leq 1$, azaz egy betű hatására legfeljebb egy másik állapotba lehetünk.

⇐ irány (**determinizálás**): Legyen $M = (Q, \Sigma, \delta, q_0, F)$ egy nemdeterminisztikus automata. Megadunk egy $M' = (Q', \Sigma, \delta', q'_0, F')$ determinisztikus automatát, amelyre $L(M') = L(M)$.

$M' = (Q', \Sigma, \delta', q'_0, F')$, ahol

- $Q' = \mathcal{P}(Q) (= \{S \mid S \subseteq Q\})$
- $q'_0 = \{q_0\}$, az eredeti kezdőállapot (egyelemű halmazban, mert itt egy állapot állapothalmazt jelent)
- $F' = \{S \subseteq Q \mid S \cap F \neq \emptyset\}$, azaz azon állapothalmazok halmaza, melyek tartalmaznak legalább egy eredeti végállapotot,
- $\delta' : Q' \times \Sigma \rightarrow Q'$ az a leképezés melyre tetszőleges $S \in Q'$ és $a \in \Sigma$ esetén $\delta'(S, a) = \cup_{q \in S} \delta(q, a)$, azaz az új állapot minden (régi) részállapotából kiinduló átmenetek összessége.

$L(M') = L(M)$ bizonyítása: vegyük egy $w = a_1 \dots a_k$ szót.



Tehát a w szót végigolvassva a determinisztikus automatával, megkapunk egy q_0, \dots, q_k -kat tartalmazó felismerő állapot sorozatot: S_0, \dots, S_k . Ha ezen sorozatot végrehajtjuk ugyanazon input szón a nemdeterminisztikus automatával, akkor S_k végállapotba kerülünk, azaz felismertük a szót.

Az $M = (Q, \Sigma, \delta, q_0, F)$ automata teljesen definiált (vagy **teljes**), ha teljesül, hogy minden $q \in Q$ és $a \in \Sigma$ esetén $\delta(q, a)$ legalább egy elemű.

A **teljesen definiált** automaták minden szót **végig** tudnak **olvasni**, mivel minden (q, aw) alakú konfigurációjukra van legalább egy rákövetkező konfiguráció.

Tétel. Tetszőleges $M = (Q, \Sigma, \delta, q_0, F)$ **automatához megadható** olyan $M' = (Q', \Sigma, \delta', q'_0, F)$ **teljesen definiált** automata, melyre $L(M') = L(M)$:

Legyen $Q' = Q \cup \{q_c\}$, ahol $q_c \notin Q$, vagyis egy új („csapda”) állapot.

Továbbá minden $q \in Q$ és $a \in \Sigma$ esetén, legyen

Ha $\delta(q, a) = \emptyset$, akkor $\delta'(q, a) = \{q_c\}$ egyébként $\delta'(q, a) = \delta(q, a)$.

Azaz amelyik állapotból nem tudunk továbbmenni, azt átirányítjuk a csapda állapotba.

Végül, minden $a \in \Sigma$ betűre legyen $\delta'(q_c, a) = \{q_c\}$, azaz minden végigolvassuk a szót.

- A reguláris nyelvtanok, az automaták és a reguláris kifejezések ekvivalenciája

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Nyelvek megadása reguláris kifejezéssel. A reguláris kifejezéssel reprezentálható nyelvek megadása 3 típusú nyelvtanokkal

Egy Σ ábécé feletti reguláris kifejezések halmaza a $(\Sigma \cup \{\emptyset, (\cdot), +, *\})^*$ halmaz legszűkebb olyan U részhalmaza, amelyre az alábbi feltételek teljesülnek:

- $\emptyset \in U$, az üres reguláris kifejezés
- $\forall a \in \Sigma (a \in U)$, minden betű U -ban is van: $\Sigma \subset U$
- Ha $R_1, R_2 \in U$, akkor $(R_1) + (R_2)$, $(R_1)(R_2)$, és $(R_1)^*$ is elemei U -nak, azaz zárt az uniósra, konkatenációra.

Az R reguláris kifejezés által meghatározott (reprezentált) $|R|$ nyelvet a következőképpen definiáljuk:

$$\begin{array}{ll} R = \emptyset & \rightarrow |R| = \emptyset \\ R = a & \rightarrow |R| = \{a\} \\ R = (R_1) + (R_2) & \rightarrow |R| = |R_1| \cup |R_2| \\ R = (R_1)(R_2) & \rightarrow |R| = |R_1| |R_2| \\ R = (R_1)^* & \rightarrow |R| = |R_1|^* \end{array}$$

Megállapodás szerint a **precedencia**: $*$, konkatenáció, $+$; továbbá: $+$ és a konkatenáció asszociatív, $+$ kommutatív
Ekkor a **zárójelezés** egyszerűsíthető: $(\emptyset)^*$ helyett \emptyset^* , $((a) + (b))^*$ helyett $(a + b)^*$, $((a)(b))(a)^*$ helyett aba^*

Tétel. Minden véges nyelv reguláris:

Legyen $L = \{w_1, \dots, w_n\}$, $n \geq 1$. Ekkor $L = |R|$, ahol $R = R_1 + \dots + R_n$ és $R_i = w_i$ betűi, ha vannak és $R_i = \emptyset^*$, ha $w_i = \lambda$.

Ekvivalencia tétel / 1. lemma: $(1) \subseteq (2)$, azaz a Σ feletti reguláris nyelvek 3-típusúak:

Átfogalmazás: Tetszőleges Σ feletti L reguláris nyelv generálható 3-típusú nyelvtannal:

Az L -et reprezentáló R reguláris kifejezés struktúrája szerinti indukcióval:

Az indukció alapjai:

- $R = \emptyset$, ekkor $L = |R| = \emptyset$, mely generálható $G = (\{S\}, \Sigma, \emptyset, S)$ 3-típusú nyelvtannal.
- $R = a$, $a \in \Sigma$, ekkor $L = |R| = \{a\}$, mely generálható $G = (\{S\}, \Sigma, \{S \rightarrow a\}, S)$ 3-típusú nyelvtannal.

Indukciós lépések:

- $R = (R_1) + (R_2)$, ekkor $L = |R| = L_1 \cup L_2$, ahol $L_1 = |R_1|$ és $L_2 = |R_2|$.
Ha L_i generálható $G_i = (N_i, \Sigma, P_i, S_i)$ 3-típusú nyelvtannal, $i = 1, 2$. $(N_1 \cap N_2 = \emptyset)$, akkor L generálható a $G = (N_1 \cup N_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}, S)$ 3-típusú nyelvtannal, ahol S egy új szimbólum.
 $S \Rightarrow_G w \Leftrightarrow S_1 \Rightarrow_{G_1} w$ vagy $S_2 \Rightarrow_{G_2} w$
- $R = (R_1)(R_2)$, ekkor $L = |R| = L_1 L_2$, ahol $L_1 = |R_1|$ és $L_2 = |R_2|$.
Ha L_i generálható $G_i = (N_i, \Sigma, P_i, S_i)$ 3-típusú nyelvtannal, $i = 1, 2$. $(N_1 \cap N_2 = \emptyset)$, akkor L generálható a $G = (N_1 \cup N_2, \Sigma, P, S_1)$ 3-típusú nyelvtannal, ahol P a legszűkebb olyan szabályhalmaz amire teljesülnek:
 - $A \rightarrow wB \in P_1 \Rightarrow A \rightarrow wB \in P$
 - $A \rightarrow w \in P_1 \Rightarrow A \rightarrow wS_2 \in P$
 - $P_2 \subseteq P$ $S_1 \Rightarrow_{G_1}^* w_1$ és $S_2 \Rightarrow_{G_2}^* w_2 \Leftrightarrow S \Rightarrow_G^* w_1 w_2 \Rightarrow_G^* w_1 w_2$
- $R = (R_1)^*$, ekkor $L = |R| = L_1^*$, ahol $L_1 = |R_1|$.
Ha L_1 generálható $G_1 = (N_1, \Sigma, P_1, S_1)$ 3-típusú nyelvtannal, akkor L generálható a $G = (N_1 \cup \{S\}, \Sigma, P, S)$ 3-típusú nyelvtannal, ahol S egy új szimbólum, P pedig a legszűkebb olyan szabályhalmaz amire teljesülnek:
 - $S \rightarrow S_1, S \rightarrow \lambda \in P$
 - $A \rightarrow xB \in P_1 \Rightarrow A \rightarrow xB \in P$
 - $A \rightarrow x \in P_1 \Rightarrow A \rightarrow xS \in P$ $S \Rightarrow_G \lambda$
 $S \Rightarrow_G S_1 \Rightarrow_G^* w_1 S \Rightarrow_G w_1 (\in L_1 \subseteq L)$
 $w_1 S \Rightarrow_G w_1 S_1 \Rightarrow_G^* w_1 w_2 S \Rightarrow_G w_1 w_2 (\in L_1 L_1 \subseteq L)$
 $w_1 w_2 S \Rightarrow_G w_1 w_2 S_1 \dots$

A reguláris (3 típusú nyelvtannal megadható) nyelvek felismerhetők automatával

Tétel: A következő három nyelvosztály megegyezik:

(1) A nemdeterminisztikus automatákkal felismerhető nyelvek osztálya

(2) A determinisztikus automatákkal felismerhető nyelvek osztálya

(3) A determinisztikus és teljes automatákkal felismerhető nyelvek osztálya

Ezt a nyelvosztályt az **automatákkal felismerhető nyelvek osztályának** nevezzük.

Tétel: Tetszőleges Σ ábécé esetén

(1) a Σ feletti reguláris nyelvek osztálya, (2) a Σ feletti 3-típusú nyelvek osztálya,

(3) a Σ feletti automatával felismerhető nyelvek osztálya, egymással megegyeznek.

Bizonyítás:

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

1. Lemma: $(1) \subseteq (2)$

2. Lemma: $(2) \subseteq (3)$

3. Lemma: $(3) \subseteq (1)$

Akkor $(1) = (2) = (3)$.

1. Lemma: $(1) \subseteq (2)$: Tetszőleges, Σ feletti L reguláris nyelv generálható 3-típusú nyelvtannal.

Definíció: Egy nyelvtanban az $A \rightarrow B$ alakú szabályokat láncszabályoknak nevezzük.

Tétel: minden $G = (N, \Sigma, P, S)$, 3-típusú nyelvtanhoz van vele ekvivalens $G' = (N, \Sigma, P', S)$, láncszabálymentes 3-típusú nyelvtan.

Bizonyítás: Legyen $G = (N, \Sigma, P, S)$ egy 3-típusú nyelvtan.

a) minden $A \in N$ -re számoljuk ki az $NA = \{B \in N \mid A \Rightarrow^* B\}$ halmazt.

b) Konstruáljuk meg $G' = (N, \Sigma, P', S)$ -t:

$P' = \emptyset$;

Minden $A \in N$ -re,

minden $B \in NA$ -ra, /* $NA = \{B \in N \mid A \Rightarrow^* B\}$ */

minden $B \rightarrow \alpha \in P$ szabály esetén:

Ha $B \rightarrow \alpha$ nem láncszabály akkor

vegyük fel az $A \rightarrow \alpha$ szabályt P' -be;

Ekkor $L(G) = L(G')$ lesz.

2. Lemma: $(2) \subseteq (3)$: minden 3-típusú nyelv felismerhető automatával.

Bizonyítás: Legyen L egy 3-típusú nyelv és tegyük fel, hogy $L = L(G)$, ahol G láncszabálymentes 3-típusú nyelvtan.

2.1. Lemma: minden $G = (N, \Sigma, P, S)$, 3-típusú láncszabálymentes nyelvtanhoz megadható vele ekvivalens

$G' = (N', \Sigma, P', S)$, 3-típusú nyelvtan, úgy hogy P' -ben minden szabály $A \rightarrow aB$ vagy $A \rightarrow \lambda$ alakú, ahol $A, B \in N$ és $a \in \Sigma$.

Bizonyítás: Konstruáljuk meg P' -t a következőképpen:

(i) minden $A \rightarrow aB$ vagy $A \rightarrow \lambda$ alakú P -beli szabályt vegyük fel P' -be.

(ii) minden $A \rightarrow a_1 \dots a_n B$, P -beli szabály esetén (ahol $n > 1$, $a_1, \dots, a_n \in \Sigma$) vegyük fel P' -be az $A \rightarrow a_1 A_1, A_1 \rightarrow a_2 A_2, \dots, A_{n-1} \rightarrow a_n B$ szabályokat, ahol A_1, \dots, A_{n-1} új nemterminális szimbólumok.

(iii) minden $A \rightarrow a_1 \dots a_n$, P -beli szabály esetén (ahol $n \geq 1, a_1, \dots, a_n \in \Sigma$) vegyük fel P' -be az $A \rightarrow a_1 A_1, A_1 \rightarrow a_2 A_2, \dots, A_{n-1} \rightarrow a_n A_n, A_n \rightarrow \lambda$ szabályokat, ahol A_1, \dots, A_n új nemterminálisok. Legyen $N' = N \cup \{ \text{új nemterminálisok} \}$.

Minden $A \in N$ -re és $w \in \Sigma^*$ -ra $A \Rightarrow^* G$ w akkor és csak akkor, ha $A \Rightarrow^* G'$ w.

Ugyanis $A \rightarrow a_1 \dots a_n B \in P$ akkor és csak akkor, ha $A \Rightarrow G' a_1 A_1 \Rightarrow G' \dots \Rightarrow G' a_1 \dots a_{n-1} A_{n-1} \Rightarrow G' a_1 \dots a_n B$

és $A \rightarrow a_1 \dots a_n \in P$ akkor és csak akkor, ha $A \Rightarrow G' a_1 A_1 \Rightarrow G' \dots \Rightarrow G' a_1 \dots a_n A_n \Rightarrow G' a_1 \dots a_n$.

Az $A = S$ választással kapjuk, hogy $L(G) = L(G')$.

2.2. Lemma: minden olyan $G = (N, \Sigma, P, S)$, 3-típusú nyelvtanhoz melynek csak $A \rightarrow aB$ vagy $A \rightarrow \lambda$ alakú szabályai vannak megadható olyan $M = (Q, \Sigma, \delta, q_0, F)$ automata, amelyre $L(M) = L(G)$.

Bizonyítás: Konstruáljuk meg M -et a következőképpen:

- $\square Q = N$,
- $\square q_0 = S$,
- $\square F = \{B \in N \mid B \rightarrow \lambda \in P\}$,
- \square minden $A \in N$ és $a \in \Sigma$ esetén legyen $\delta(A, a) = \{B \in N \mid A \rightarrow aB \in P\}$.

Ekkor minden $n \geq 1, A, B \in N$ és $w \in \Sigma^*$ esetén $A \Rightarrow^n wB$

akkor és csak akkor ha $(A, w) \vdash^n (B, \lambda)$.

$A \Rightarrow G a_1 A_1 \Rightarrow G \dots \Rightarrow G a_1 \dots a_{n-1} A_{n-1} \Rightarrow G a_1 \dots a_n B$ akkor és csak akkor, ha

$(A, a_1 \dots a_n) \vdash M (A_1, a_2 \dots a_n) \vdash M \dots \vdash M (B, \lambda)$.

Az $A = S, B \in F$ választással adódik, hogy $L(M) = L(G)$.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- Környezetfüggetlen nyelvekkel kapcsolatos alapfogalmak

Egy $G = (N, \Sigma, P, S)$ nyelvtan környezetfüggetlen, ha minden szabálya $A \rightarrow \alpha$ alakú.

Példák:

1) Az $S \rightarrow aSb \mid \epsilon$ nyelvtan, amely az $\{anbn \mid n \geq 0\}$ nyelvet generálja.

2) A Gar nyelvtan, melynek szabályai

$K \rightarrow K + T \mid T$,

$T \rightarrow T * F \mid F$,

$F \rightarrow (K) \mid a$.

$L(\text{Gar})$ az a-ból valamint a $(,)$, $+$ és $*$ jelekből képezhető aritmetikai kifejezések halmaza. Pl a $*(a + a) \in L(\text{Gar})$.

3) Egy további példa a szabályos zárójelezéseket generáló nyelvtan:

$S \rightarrow SS \mid (S) \mid ()$

Például:

$S \Rightarrow SS \Rightarrow (S)S \Rightarrow (())S \Rightarrow ((())()$

A környezetfüggetlen nyelvtanok két legfontosabb alkalmazása:

természetes nyelvek feldolgozása (NLP),

programozási nyelvek szintaxisának megadása.

Az alkalmazástól függően kisebb kiterjesztések szükségesek („majdnem környezetfüggetlen” nyelvtan).

Rövidítés: cf nyelv(tan) := környezetfüggetlen nyelv(tan)

Backus-Naur forma

A gyakorlatban (pl programozási nyelvek megadásakor) a környezetfüggetlen nyelvtan egy makró-szerű változatát használják. A neve Backus-Naur forma, röviden BNF. A jelölés tömörebb lesz, de a nyelv generáló kapacitása nem változik.

1) A nemterminálisokat szöglletes zárójelbe tett szavakkal adjuk meg, pl: <program>, <ut.listai> a terminálisokat kövér kisbetűkkel: if, while.

2) $A \rightarrow$ helyett ::= jelet írunk.

3) ”egy vagy több” makró: $\alpha\dots$

pl <integer> ::= <digit>...

<digit> ::= 0|1|2|3|4|5|6|7|8|9

A mi jelölésünkkel: $\alpha\dots$ helyett A-t írunk és felvesszük az $A \rightarrow A\alpha|\alpha$ szabályokat.

4) ”egy vagy egy sem” (opcionális) makró: $[\alpha]$

pl.: <ifut> ::= if <relacio> then <út> [else<ut>]

A mi jelölésünkkel: $[\alpha]$ helyett A-t írunk és felvesszük az $A \rightarrow \alpha|\epsilon$ szabályokat.

5) ”több szimbólum egy egység” makró: $\{\alpha\}$

pl <ut.lista> ::= <ut> [{;<ut>} ...]

A mi jelölésünkkel: $\{\alpha\}$ helyett A-t írunk és felvesszük az $A \rightarrow \alpha$ szabályt.

Példa:

Fordítsuk le a sztenderd jelölésre az $L ::= S[\{;S\}\dots]$ definíciót.

a)

$$\begin{array}{l} L \rightarrow S[A\dots] \\ A \rightarrow ;S \end{array}$$

b)

$$\begin{array}{l} L \rightarrow SB \\ B \rightarrow A\dots|\epsilon \\ A \rightarrow ;S \end{array}$$

c)

$$\begin{array}{l} L \rightarrow SB \\ B \rightarrow C|\epsilon \\ C \rightarrow AC|A \\ A \rightarrow ;S \end{array}$$

Korlátozás nélküli, bal- és jobb oldali derivációk

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Korlátozás nélküli deriváció (levezetés):

$\alpha_0 \Rightarrow \alpha_1 \Rightarrow \dots \Rightarrow \alpha_n$ alakú kifejezések

Bal oldali deriváció (α_i legbaloldalibb nemterminálisát helyettesítjük):

$\alpha_0 \Rightarrow_l \alpha_1 \Rightarrow_l \dots \Rightarrow_l \alpha_n$

Amennyiben a fenti deriváció során minden $i = 1, \dots, n$ esetén α_i -t úgy kapjuk, hogy α_{i-1} -ben a bal oldalról nézve legelső nemtermináliszt helyettesítjük egy rá vonatkozó szabály jobb oldalával, akkor a derivációt bal oldali derivációnak hívjuk, és rá az $\alpha_0 \Rightarrow_l \alpha_1 \Rightarrow_l \dots \Rightarrow_l \alpha_n$ jelölést használjuk.

Jobb oldali deriváció (α_i legjobboldalibb nemterminálisát helyettesítjük):

$\alpha_0 \Rightarrow_r \alpha_1 \Rightarrow_r \dots \Rightarrow_r \alpha_n$

Amennyiben egy deriváció minden lépései a jobbról nézve legelső nemtermináliszt helyettesítjük jobb oldali derivációról beszélünk.

Példa:

Korlátozás nélküli, bal oldali és jobb oldali levezetések az aritmetikai kifejezéseket generáló G_{ar} nyelvtanban. (Az aláhúzott nemtermináliszt helyettesítjük.)

$$\begin{aligned} \underline{K} &\Rightarrow \underline{I} \Rightarrow \underline{I * F} \Rightarrow F * \underline{E} \Rightarrow F * (K) \\ \underline{K} &\Rightarrow_l \underline{I} \Rightarrow_l \underline{I * F} \Rightarrow_l E * F \Rightarrow_l a * F \\ \underline{K} &\Rightarrow_r \underline{I} \Rightarrow_r \underline{T * E} \Rightarrow_r T * (\underline{K}) \Rightarrow_r T * (K + T) \end{aligned}$$

A különböző levezetési módok kapcsolata

Ha $X \Rightarrow^* l \alpha$ (bal oldali deriváció) vagy $X \Rightarrow^* r \alpha$ (jobb oldali deriváció), akkor $X \Rightarrow^* \alpha$ is fennáll.

Fordítva nem igaz, például Gar-ban

$K \Rightarrow K + T \Rightarrow K + T + T \Rightarrow K + F + T$,

de sem $K \Rightarrow^* l K + F + T$ sem $K \Rightarrow^* r K + F + T$ nem teljesül.

Ellenben, ha α terminális szó ($\alpha \in \Sigma^*$), akkor az állítás már megfordítható, vagyis igaz lesz, hogy bal oldali (jobb oldali) levezetésekkel ugyanazok a terminális szavak kaphatók meg, mint korlátozás nélküli levezetésekkel.

Lemma. minden $X \in (N \cup \Sigma)$ és $w \in \Sigma^*$ esetén:

$X \Rightarrow^* w \Leftrightarrow X \Rightarrow^* l w \Leftrightarrow X \Rightarrow^* r w$

(X-ből korlátozás nélkül levezethető w, akkor és csak akkor, ha X-ból bal és jobb oldali derivációval is megkapjuk w-t)

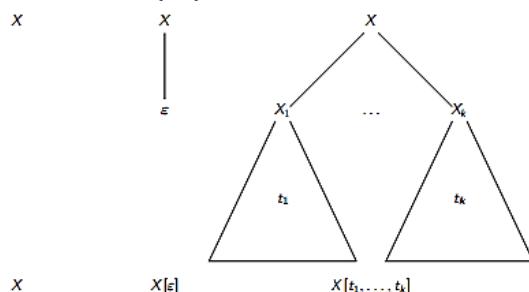
Az $X \in (N \cup \Sigma)$ gyökerű derivációs fák halmaza a legszűkebb olyan DX halmaz, amelyre

(i) Az a fa, amelynek egyetlen szög pontja (vagyis csak gyökere) az X, eleme DX -nek.

(ii) Ha $X \rightarrow \epsilon \in P$, akkor az a fa, amelynek gyökere X, a gyökerének egyetlen leszármazottja az ϵ , eleme DX -nek.

(iii) Ha $X \rightarrow X_1 \dots X_k \in P$, továbbá $t_1 \in DX_1, \dots, t_k \in DX_k$, akkor az a fa, amelynek gyökere X, a gyökérből k él indul rendre a t_1, \dots, t_k fák gyökeréhez, eleme DX -nek.

Megjegyzés: ha $X \in \Sigma$, akkor az (ii) és (iii) pontok nem eredményeznek további fákat, tehát ekkor $DX = \{X\}$.

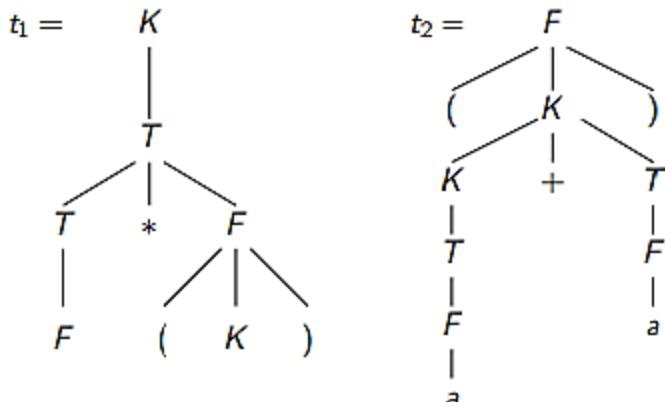


Legyen t egy X gyökerű derivációs fa. Akkor t magasságát $h(t)$ -vel, a határát pedig $fr(t)$ -vel jelöljük és az alábbi módon definiáljuk

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- (i) Ha t az egyetlen X szögpontból álló fa, akkor $h(t) = 0$ és $fr(t) = X$.
- (ii) Ha t gyökere X , aminek egyetlen leszármazottja ε , akkor $h(t) = 1$ és $fr(t) = \varepsilon$.
- (iii) Ha t gyökere X , amiből k él indul rendre a t_1, \dots, t_k közvetlen részfák gyökeréhez, akkor $h(t) = 1 + \max\{h(t_i) \mid 1 \leq i \leq k\}$ és $fr(t) = fr(t_1) \dots fr(t_k)$.

Informálisan: $h(t)$ a t -ben lévő olyan utak hosszának maximuma, amelyek t gyökerétől annak valamely leveléhez vezetnek. Továbbá, $fr(t)$ azon $(N \cup \Sigma)^*$ -beli szó, amelyet t leveleinek balról jobbra (vagy: preorder bejárással) történő leolvasásával kapunk.



$$fr(t_1) = F * (K)$$

$$h(t_1) = 3$$

$$fr(t_2) = (a + a)$$

$$h(t_2) = 5$$

A derivációs fák és a derivációk között a következő kapcsolat áll fenn:

Tétel Tetszőleges $X \in (N \cup \Sigma)$ és $\alpha \in (N \cup \Sigma)^*$ esetén $X \Rightarrow^* \alpha$ akkor és csak akkor, ha van olyan $t \in D_X$ derivációs fa amelyre $fr(t) = \alpha$.

Legyen $G = (N, \Sigma, P, S)$ egy környezetfüggetlen nyelvtan. Az előző tételeből azonnal kapjuk:

Következmény. Tetszőleges $w \in \Sigma^*$ esetén $S \Rightarrow^* w$ akkor és csak akkor, ha van olyan S gyökerű derivációs fa amelynek határa w .

Ebből és a terminális szóban végződő különböző levezetési módok ekvivalenciából pedig:

Következmény. Tetszőleges $w \in \Sigma^*$ esetén a következő állítások ekvivalensek:

- $w \in L(G)$,
- $S \Rightarrow^* w$,
- $S \Rightarrow^{*l} w$,
- van olyan S gyökerű derivációs fa amelynek határa w .

Más szóval, a $G = (N, \Sigma, P, S)$ cf nyelvtan által generált nyelv a most bevezetett fogalmak segítségével a következőképpen írható fel.

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\},$$

$$= \{w \in \Sigma^* \mid S \Rightarrow^{*l} w\},$$

$$= \{fr(t) \mid t \in DS, fr(t) \in \Sigma^*\}.$$

Definíció. Tetszőleges $x \in L(G)$ esetén x derivációs fájának nevezünk egy olyan S gyökerű derivációs fát, melynek határa x .

Egy $x \in L(G)$ szónak általában nem csak egy derivációs fája van. Tekintsük pl a szabályos zárójelezéseket generáló

$$S \rightarrow SS \mid (S) \mid ()$$

nyelvtant és az alábbi bal oldali derivációkat:

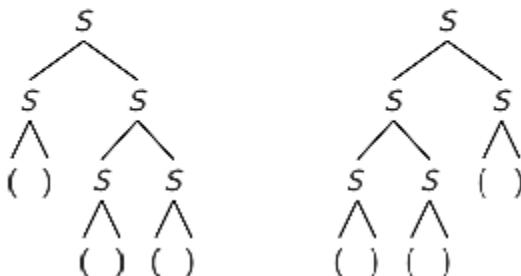
$$S \Rightarrow^1 SS \Rightarrow^1 ((S)) \Rightarrow^1 (((S))) \Rightarrow^1 (((((S))))))$$

$$S \Rightarrow^1 SS \Rightarrow^1 SSS \Rightarrow^1 (((S))) \Rightarrow^1 (((((S))))))$$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Ha megkonstruáljuk ezen derivációkhöz tartozó derivációs fákat, akkor a $()()$ szó két különböző derivációs fáját kapjuk.



A derivációs fák és a bal oldali levezetések kapcsolata

Vegyük egy $x \in L(G)$ szót.

Az x szó minden derivációs fája egyértelműen meghatározza az x egy bal oldali levezetését. Ha két derivációs fa különböző, akkor a bal oldali levezetések is különbözőek lesznek.

Fordítva, x minden bal oldali levezetése egyértelműen meghatározza az x egy derivációs faját. Különböző bal oldali levezetések különböző derivációs fákat eredményeznek. Tehát x derivációs fái és bal oldali levezetései között bijekció áll fenn

Egyértelmű nyelvtanok és nyelvek

Definíció. Egy G nyelvtan egyértelmű, ha minden $x \in L(G)$ szónak csak egy derivációs fája van. A feltétel ekvivalens azzal, hogy minden $x \in L(G)$ szónak csak egy bal oldali levezetése van.

A természetes nyelvek nem egyértelműek.

Pi "Láttam Ferit a távcsővel."

Programozási nyelveknek egyértelműeknek kell lenniük. Különben egy programnak több derivációs fája van, amiből problémák adódnak a szemantika szintjén.

Példa

A szabályos zárójelezéseket generáló $S \rightarrow SS | (S) | ()$ nyelvtan nem egyértelmű.

Mint láttuk, például a $()()$ szónak két bal oldali levezetése van:

- $S \Rightarrow l SS \Rightarrow l ()S \Rightarrow l ()SS \Rightarrow l ()()S \Rightarrow l ()()()$
- $S \Rightarrow l SS \Rightarrow l SSS \Rightarrow l ()SS \Rightarrow l ()()S \Rightarrow l ()()()$

Egy negatív eredmény:

Tétel. Nem létezik olyan algoritmus, amely tetszőleges cf nyelvtanról eldönti, hogy egyértelmű-e. Más szóval, nem tudunk olyan programot írni, melynek inputja egy cf nyelvtan, outputja pedig igen vagy nem, attól függően, hogy az input nyelvtan egyértelmű vagy nem.

Mindkét, eddig megismert, nem egyértelmű nyelvtanhoz adható vele ekvivalens (ugyanazon nyelvet generáló) egyértelmű nyelvtan.

Kérdés, hogy ez igaz-e általában is: megadható-e minden nem egyértelmű nyelvtanhoz vele ekvivalens egyértelmű nyelvtan.

Látni fogjuk, hogy nem: van olyan környezetfüggetlen nyelv, melyhez nem adható meg őt generáló egyértelmű nyelvtan.

Definíció. Egy L környezetfüggetlen nyelv egyértelmű, ha van olyan egyértelmű környezetfüggetlen G nyelvtan, melyre $L = L(G)$.

Tétel. Létezik nem egyértelmű környezetfüggetlen nyelv.

Például

Az $L = \{aibjck \mid i = j \text{ vagy } j = k\}$ nyelv nem egyértelmű.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

$L = \{a^i b^j c^k \mid i = j \text{ vagy } j = k\}$. Tekintsük az

$$\begin{array}{lcl} S & \rightarrow & AB \mid CD \\ A & \rightarrow & aAb \mid ab \\ B & \rightarrow & cB \mid c \\ C & \rightarrow & aC \mid a \\ D & \rightarrow & bDc \mid bc \end{array}$$

nyelvtant, ami L -et generálja.

A biztosítja, hogy "a-k száma" = "b-k száma". Ehhez B valamennyi c betűt hozzátesz. Hasonlóan, D biztosítja, hogy "b-k száma" = "c-k száma". Ehhez C valamennyi a betűt hozzátesz. A nyelvtan tehát az $\{a^i b^j c^k \mid i = j \text{ vagy } j = k\}$ nyelvet generálja. Az $a^n b^n c^n$ alakú szavak viszont "duplán jönnek létre".

Az $a^n b^n c^n$ alakú szavak viszont "duplán jönnek létre". Például

$$S \Rightarrow_I AB \Rightarrow_I abB \Rightarrow_I abc \text{ és } S \Rightarrow_I CD \Rightarrow_I aD \Rightarrow_I abc$$

Tehát nekünk sem sikerült egyértelmű nyelvtant megadni ;

Ez persze nem bizonyítás.

1. Nem produktív szimbólumok elhagyása:

$A \in N$ nemterminális **produktív**, ha létezik $A \Rightarrow^* x$ szabály, ahol $x \in \Sigma^*$.

A produktív nemterminálisok U halmazát kiszámító algoritmus:

- (i) Legyen $U_1 = \{A \in N \mid \exists(A \rightarrow x \in P) \text{ valamely } x \in \Sigma^*\text{-ra}\}$ és legyen $i = 1$.
- (ii) Legyen $U_{i+1} = U_i \cup \{A \in N \mid \exists(A \rightarrow \alpha \in P) \text{ úgy, hogy } \alpha \in (U_i \cup \Sigma)^*\}$.
- (iii) Ha $U_{i+1} = U_i$, akkor állunk meg (és ekkor $U = U_i$), különben legyen $i = i + 1$ és hajtsuk végre az (ii) lépést.

Mellesleg: $L(G) \neq \emptyset$ akkor és csak akkor, ha $S \in U$.

Kiszámoljuk a produktív nemterminálisok U halmazát, majd elhagyunk minden olyan szabályt, ami tartalmaz nem U -beli szimbólumot. (Az új nemterminálisok halmaza: $N_1 := U$)

2. Nem elérhető szimbólumok elhagyása:

$X \in (N \cup \Sigma)$ szimbólum **elérhető**, ha létezik $S \Rightarrow^* \alpha X \beta$ szabály, ahol $\alpha, \beta \in (N \cup \Sigma)^*$ és S a kezdőszimbólum.

Megjegyzés: Vegyük észre, hogy itt a terminálisokat (ezek Σ elemei) is belevesszük a buliba, és ennek megfelelően a nem elérhető terminálisokat majd kidobjuk a Σ ábécéből!

Az elérhető szimbólumok H halmazát kiszámító algoritmus:

- (i) Legyen $H_0 = \{S\}$ és legyen $i = 0$.
- (ii) Legyen $H_{i+1} = H_i \cup \{X \in (N \cup \Sigma) \mid \exists(A \rightarrow \alpha X \beta \in P) \text{ úgy, hogy } A \in H_i\}$.
- (iii) Ha $H_{i+1} = H_i$, akkor állunk meg (és ekkor $H = H_i$), különben legyen $i = i + 1$ és hajtsuk végre az (ii) lépést.

Kiszámoljuk az elérhető szimbólumok H halmazát, majd elhagyunk minden olyan szabályt, ami nem tartalmaz H -beli szimbólumot. Módosítjuk a nemterminálisok halmazát: $N_2 := H \cap N_1$, továbbá az ábécét is: $\Sigma_2 := H \cap \Sigma$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

1. ϵ -mentesítés

Cél, hogy a nyelvtan ne tartalmazzon $A \rightarrow \epsilon$ alakú szabályt, kivéve esetleg az $S \rightarrow \epsilon$ szabályt, de ekkor S nem lehet a szabályok jobb oldalán.

Lépései:

- I. $H = \{X \in N | X \Rightarrow^* \epsilon\}$ halmaz kiszámolása
- II. H elemeit az összes lehetséges módon töröljük az eredeti szabályokból (0 darabot, 1 darabot, 2 darabot,... törlünk), úgy hogy ne kapunk ϵ -szabályt
- III. Ha $S \in H$, akkor vegyük fel egy új S_1 kezdőszimbólumot és az $S_1 \rightarrow S|\epsilon$ szabályt

2. Láncszabály-mentesítés

Cél, hogy a nyelvtan ne tartalmazzon $A \rightarrow B$ alakú szabályt

Láncszabálygráfot írunk fel, melynek csúcsai a nemterminálisok és egy $A \rightarrow B$ élt behúzunk, ha van $A \rightarrow B$ szabályunk. Ezután az az új nyelvtanban A jobboldalára beírjuk az összes A -ból a láncszabálygráfban elérhető nemterminálisok jobboldalát, kivéve a láncszabályokat.

Chomsky normál alakra hozás:

1. ϵ -mentesítés

Cél, hogy a nyelvtan ne tartalmazzon $A \rightarrow \epsilon$ alakú szabályt, kivéve esetleg az $S \rightarrow \epsilon$ szabályt, de ekkor S nem lehet a szabályok jobb oldalán.

Lépései:

- I. $H = \{X \in N | X \Rightarrow^* \epsilon\}$ halmaz kiszámolása
- II. H elemeit az összes lehetséges módon töröljük az eredeti szabályokból (0 darabot, 1 darabot, 2 darabot,... törlünk), úgy hogy ne kapunk ϵ -szabályt
- III. Ha $S \in H$, akkor vegyük fel egy új S_1 kezdőszimbólumot és az $S_1 \rightarrow S|\epsilon$ szabályt

2. Láncszabály-mentesítés

Cél, hogy a nyelvtan ne tartalmazzon $A \rightarrow B$ alakú szabályt

Láncszabálygráfot írunk fel, melynek csúcsai a nemterminálisok és egy $A \rightarrow B$ élt behúzunk, ha van $A \rightarrow B$ szabályunk. Ezután az az új nyelvtanban A jobboldalára beírjuk az összes A -ból a láncszabálygráfban elérhető nemterminálisok jobboldalát, kivéve a láncszabályokat.

3. Terminálisok helyettesítése nemterminálisokkal

...úgy, hogy ne alakuljon ki láncszabály:

azokat a szabályokat megtartjuk, ahol egyetlen terminális van a jobboldalon ($A \rightarrow a$), viszont ahol 2-nél több nemterminális van a jobboldalon, oda a terminálisok helyére nemterminálisokat írunk: $A \rightarrow abaa$ helyett $A \rightarrow X_a X_b X_a X_a$ szabályt írunk és felveszük az $X_a \rightarrow a$, $X_b \rightarrow b$ szabályokat, továbbá a nemterminálisokhoz odarakjuk az újdonsült X_a és X_b nemterminálisokat.

4. a 2-nél hosszabb jobboldalak darabolása

Itt most az $A \rightarrow BC$ alakra gyűrünk, tehát célunk, hogy pontosan 2 nemterminális legyen a szabályok jobboldalán. Így például az előző $A \rightarrow X_a X_b X_a X_a$ szabályt így darabolom:

$$\begin{aligned} X_{a,a} &\rightarrow X_a X_a \quad (\text{az utolsó két } X_a\text{-t összevonom}) \\ X_{b,a,a} &\rightarrow X_b X_{a,a} \quad (X_b\text{-t és } X_{a,a}\text{-t összevonom}) \\ A &\rightarrow X_a X_{b,a,a} \end{aligned}$$

és a nemterminálisokhoz beveszem még $X_{a,a}$ -t és $X_{b,a,a}$ -t.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- Veremautomaták és környezetfüggetlen nyelvtanok ekvivalenciája

Veremautomatának (vagy pushdown automaton, röviden pda) nevezük a $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ rendszert, ahol:

- Q egy véges halmaz, az állapotok halmaza
- Σ az input ábécé
- Γ a verem ábécé
- $q_0 \in Q$ kezdeti állapot
- $Z_0 \in \Gamma$ a verem kezdeti szimbólum
- $F \subseteq Q$ a végállapotok halmaza
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow \mathcal{P}_f(Q \times \Gamma^*)$ az átmenet függvény.

Tetszőleges $q \in Q$, $a \in \Sigma \cup \{\lambda\}$ input- és $Z \in \Gamma$ veremszimbólum esetén

$\delta(q, a, Z) = \{(q_1, \alpha_1), \dots, (q_n, \alpha_n)\}$, ahol $n \geq 0$, $q_1, \dots, q_n \in Q$ és $\alpha_1, \dots, \alpha_n \in \Gamma^*$. (Az $n = 0$ esetben a képhalmaz az \emptyset lesz.)

A $C = Q \times \Sigma^* \times \Gamma^*$ halmazt a P konfigurációi halmazának nevezik.

Egy $(q, w, \gamma) \in C$ konfiguráció jelentése az, hogy P a $q \in Q$ állapotban van, a $w \in \Sigma^*$ input szót kapja és veremének tartalma γ . A konfigurációt (q, aw, Zy) alakban is megadhatjuk.

$A \vdash_p \subseteq C \times C$ átmeneti reláció: tetszőleges $p, q \in Q$, $a \in \Sigma \cup \{\lambda\}$, $w \in \Sigma^*$, $Z \in \Gamma$ és $\alpha, \gamma \in \Gamma^*$ -ra:

$$(q, aw, Zy) \vdash_p (p, w, \alpha\gamma) \Leftrightarrow (p, \alpha) \in \delta(q, a, Z).$$

A $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ által végállapotokkal felismert nyelv: $L_f(P) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash_p^* (q, \lambda, \gamma), q \in F, \gamma \in \Gamma^*\}$

A $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ által üres veremmel felismert nyelv: $L_0(P) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash_p^* (q, \lambda, \lambda), q \in Q\}$

Egy konkrét P veremautomatára általában $L_f(P) \neq L_0(P)$.

Tétel. A veremautomatákal végállapotokkal felismerhető nyelvek osztálya megegyezik a veremautomatákkal üres veremmel felismerhető nyelvek osztályával: $\{L \mid L = L_f(P)$ valamely P veremautomatára $\} = \{L \mid L = L_0(P)$ valamely P veremautomatára $\}$

Minden $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ pda-hoz megkonstruálható olyan $P' = (Q', \Sigma, \Gamma', \delta', q'_0, Z'_0, F')$ pda amelyre $L_0(P') = L_f(P)$

A P' automata egy „burkoló” automata P köré, mely lefuttatja P -t, majd kiüríti a vermet:

- $Q' = Q \cup \{q'_0, q_e\}$, ahol q'_0 és q_e (verem kiürítő /empty/ állapot) új állapotok
 - $\Gamma' = \Gamma \cup \{Z'_0\}$, ahol Z'_0 egy új szimbólum
 - F' tetszőleges részhalmaza Q' -nek
1. $\delta'(q'_0, \lambda, Z'_0) = \{(q_0, Z_0 Z'_0)\}$
 2. minden $q \in Q$, $a \in \Sigma \cup \{\lambda\}$, $Z \in \Gamma$ esetén
 - o ha $a = \lambda$ és $q \in F$, akkor $\delta'(q, a, Z) = \delta(q, a, Z) \cup \{(q_e, \lambda)\}$
azaz ha végállapot és végigolvasta a szót, akkor \vdash verem kiürítő állapot
 - o ha $a \neq \lambda$ vagy $q \notin F$, akkor $\delta'(q, a, Z) = \delta(q, a, Z)$
ellenkező esetben minden marad az eredeti
 3. minden $Z \in \Gamma$ -re $\delta'(q_e, \lambda, Z) = \{(q_e, \lambda)\}$, azaz a q_e állapot törli az összes veremszimbólumot a veremből

$L_0(P') = L_f(P)$, mert: $w \in L_f(P)$

\Leftrightarrow (def) $(q_0, w, Z_0) \vdash_p^* (q, \lambda, \gamma)$ [ahol $q \in F$]

$\Leftrightarrow (q'_0, w, Z'_0) \vdash_{P'} (1) (q_0, w, Z_0 Z'_0) \vdash_{P'} (2) (q, \lambda, \gamma Z'_0)$ [ahol $q \in F$] = $(q, \lambda, Z_1 \dots Z_n Z'_0) \vdash_{P'} (2) (q_e, \lambda, Z_2 \dots Z_k Z'_0) \vdash_{P'} (3) (q_e, \lambda, \lambda)$

\Leftrightarrow (def) $w \in L_0(P')$

Minden $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ pda-hoz megkonstruálható olyan $P' = (Q', \Sigma, \Gamma', \delta', q'_0, Z'_0, F')$ pda amelyre $L_f(P') = L_0(P)$

A P' automata egy olyan „burkoló” automata P köré, mely lefuttatja P -t, majd ha kiürült a verme (Z'_0 maradt), átvizsi q_f -be:

- $Q' = Q \cup \{q'_0, q_f\}$, ahol q'_0 és q_f (finish state) új állapotok
- $\Gamma' = \Gamma \cup \{Z'_0\}$, ahol Z'_0 egy új szimbólum
- $F' = \{q_f\}$
- δ' az alábbi módon definiált átmenetfüggvény:
 1. $\delta'(q'_0, \lambda, Z'_0) = \{(q_0, Z_0 Z'_0)\}$
 2. $\delta'(q, a, Z) = \delta(q, a, Z)$ minden $q \in Q$, $a \in \Sigma \cup \{\lambda\}$, $Z \in \Gamma$, az eredeti állapotátmenetek
 3. $\delta'(q, \lambda, Z'_0) = \{(q_f, \lambda)\}$ minden $q \in Q$ esetén, minden állapotból a végállapotba megy át, ha üres a verem

$L_f(P') = L_0(P)$, mert: $w \in L_0(P)$

\Leftrightarrow (def) $(q_0, w, Z_0) \vdash_p^* (q, \lambda, \lambda)$ [ahol $q \in Q$]

$\Leftrightarrow (1) (q'_0, w, Z'_0) \vdash_{P'} (1) (q_0, w, Z_0 Z'_0) \vdash_{P'} (2) (q, \lambda, Z'_0)$ [ahol $q \in Q$] $\vdash_{P'} (3) (q_f, \lambda, \lambda)$

\Leftrightarrow (def) $w \in L_f(P')$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Tétel. minden környezetfüggetlen nyelv felismerhető pda-val: Vegyünk egy $G = (N, \Sigma, R, S)$ környezetfüggetlen nyelvtant. Legyen $P = (\{q\}, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$ az a veremautomata amelyben:

- $\Gamma = N \cup \Sigma$
- $Z_0 = S$
- δ átmenetfüggvény pedig a következő módon van definiálva:
 1. $\delta(q, \lambda, A) = \{(q, \alpha) \mid A \rightarrow \alpha \in R\}$ minden $A \in N$ -ra, azaz alkalmaz egy bal oldali levezetési szabályt a veremben
 2. $\delta(q, a, a) = \{(q, \lambda)\}$ minden $a \in \Sigma$ -ra, azaz törli a terminális a veremből, ha azt olvassa

Megmutatjuk, hogy $L_0(P) = L(G)$. Elegendő igazolni, hogy minden $X \in \Gamma$ és $w \in \Sigma^*$ esetén:

$X \Rightarrow^* w$ akkor és csak akkor, ha $(q, w, X) \vdash^* (q, \lambda, \lambda)$:

- (\Rightarrow irány: ha $X \Rightarrow^* w$, akkor $(q, w, X) \vdash^* (q, \lambda, \lambda)$):
 - $Tfh X \Rightarrow^n w$
 - $n = 0: X = w = a \in \Sigma$, ezért a (2) pont szerint $(q, a, X) = (q, a, w) \vdash (q, \lambda, \lambda)$.
 - $n \Rightarrow n + 1: X \Rightarrow X_1...X_k \Rightarrow^n w_1...w_k = w$
 - $X \rightarrow X_1...X_k \in R$
 - $X_i \Rightarrow^n w_i$ minden $1 \leq i \leq k$ -ra, ahol $n_i \leq n$
 - Indukciós feltevés: minden $1 \leq i \leq k$ esetén $(q, w_i, X_i) \vdash^* (q, \lambda, \lambda)$.
 - Innen kapjuk, hogy $(q, w, X) = (q, w_1...w_k, X) \vdash (q, w_1...w_k, X_1...X_k) \vdash^* (q, w_2...w_k, X_2...X_k) ... \vdash^* (q, w_k, X_k) \vdash^* (q, \lambda, \lambda)$
 - A felismerés során P a w-nek egy bal oldali levezetését szimulálja.
 - (\Leftarrow irány: ha $(q, w, X) \vdash^* (q, \lambda, \lambda)$, akkor $X \Rightarrow^* w$):
 - $Tfh (q, w, X) \vdash^* (q, \lambda, \lambda)$. Az átmenetek során alkalmazott (1) típusú átmenetek száma szerinti indukcióval igazolhatjuk, hogy $X \Rightarrow^* w$

Tétel. minden pda-val felismerhető nyelv környezetfüggetlen:

- Legyen $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ egy pda. Konstruáljuk meg a $G = (N, \Sigma, R, S)$ környezetfüggetlen nyelvtant a következőképpen:
 - S egy új szimbólum
 - $N = \{S\} \cup \{[qZr] \mid q, r \in Q, Z \in \Gamma\}$
 - R szabályok olyan legszűkebb halmaza, amire:
 - minden $q \in Q$ -ra legyen $S \rightarrow [q_0Z_0q]$ szabály R-ben
 - minden $q \in Q, a \in \Sigma \cup \{\lambda\}, Z \in \Gamma$ -ra, ha $(s_0, Z_1...Z_k) \in \delta(q, a, Z)$, (ahol $k \geq 1, Z_1, ..., Z_k \in \Gamma$) akkor minden $s_1, ..., s_k \in Q$ sorozatra $[qZs_k] \rightarrow a[s_0Z_1s_1]...[s_{k-1}Z_ks_k] \in R$ -ben
 - minden $q \in Q, a \in \Sigma \cup \{\lambda\}, Z \in \Gamma$ -ra, ha $(s_0, \lambda) \in \delta(q, a, Z)$, akkor $[qZs_0] \rightarrow a \in R$ (előző eset $k = 0$ -ra)
- Elegendő megmutatni, hogy minden $q, r \in Q, Z \in \Gamma$ és $w \in \Sigma^*$ esetén $(q, w, Z) \vdash^* (r, \lambda, \lambda) \Leftrightarrow [qZr] \Rightarrow^* w$, ugyanis ezen ekvivalencia a $q = q_0$ és $Z = Z_0$ esetben épben az $L(G) = L_0(P)$ egyenlőséget jelenti:
 $w \in L_0(P) \Leftrightarrow (q_0, w, Z_0) \vdash^* (r, \lambda, \lambda) \Leftrightarrow [q_0Z_0r] \Rightarrow^* w \Leftrightarrow S \Rightarrow [q_0Z_0r] \Rightarrow^* w \Leftrightarrow w \in L(G)$.
 - $Tfh [qZr] \Rightarrow^n w$ valamelyen $n \geq 1$ -re.
 - $n = 1: [qZr] \Rightarrow w$, ez csak úgy lehet, ha $w = a \in \Sigma \cup \{\lambda\}$ és $(r, \lambda) \in \delta(q, a, Z)$. Tehát $(q, w, Z) = (q, a, Z) \vdash (r, \lambda, \lambda)$.
 - $n \Rightarrow n + 1: [qZr] \Rightarrow a[s_0Z_1s_1]...[s_{k-1}Z_ks_k] \Rightarrow^n aw_1...w_k = w$
 - $[qZr] \rightarrow a[s_0Z_1s_1]...[s_{k-1}Z_ks_k] \in R, r = s_k$, azaz $(s_0, Z_1...Z_k) \in \delta(q, a, Z)$
 - minden $1 \leq i \leq k$ -ra $[s_{i-1}Z_is_i] \Rightarrow^n w_i$, és $n_i \leq n$.
 - Indukciós feltevés: minden $1 \leq i \leq k$ -ra $(s_{i-1}, w_i, Z_i) \vdash^* (s_i, \lambda, \lambda)$.
- Kapjuk, hogy $(q, w, Z) = (q, aw_1...w_k, Z) \vdash (s_0, w_1...w_k, Z_1...Z_k) \vdash^* (s_1, w_2...w_k, Z_2...Z_k) ... \vdash^* (s_{k-1}, w_k, Z_k) \vdash^* (s_k, \lambda, \lambda) = (r, \lambda, \lambda)$

Az ekvivalencia megfordítását hasonló módon, a $(q, w, Z) \vdash^* (r, \lambda, \lambda)$ feltételben szereplő n szerinti indukcióval bizonyíthatjuk be.

• A Bar-Hillel lemma és következményei

Tétel.(Bar Hillel lemma) minden $L \subseteq \Sigma^*$ környezetfüggetlen nyelvhez megadhatók olyan (L -től függő) $p, q > 0$ egész számok, hogy minden $w \in L$ esetén, ha $|w| > p$, akkor vannak olyan $w_1, w_2, w_3, w_4, w_5 \in \Sigma^*$ szavak melyekre teljesülnek az alábbi feltételek:

1. $w = w_1w_2w_3w_4w_5$
2. $|w_2w_3w_4| \leq q$
3. $w_2w_4 \neq \lambda$, azaz w_2 vagy w_4 nemüres
4. minden $n \geq 0$ -ra $w_1w_2^n w_3w_4^n w_5 \in L$

Szükséges feltétele annak, hogy egy nyelv környezetfüggetlen legyen: ha L egy környezetfüggetlen nyelv, akkor teljesül rá;
ha nem teljesül egy L nyelvre, akkor az nem környezetfüggetlen nyelv.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Legyen $L = L(G)$, ahol $G = (N, \Sigma, P, S)$ láncszabály és λ -mentes környezetfüggetlen nyelvtan.

Legyen $n = |N|$, $k = \max\{|\alpha| \mid A \rightarrow \alpha \in P\}$ és $p := k^n$ és $q := k^{n+1}$.

Megmutatjuk, hogy ez a p és q megfelelő lesz.

Vegyük egy $w \in L$ szót, melyre $|w| > p$. Akkor van olyan $t \in D_s$ derivációs fa, melyre $fr(t) = w$ és $h(t) > n$.

Következésképpen t-ben van olyan, S-től valamely levélhez vezető út melynek hossza legalább $n + 1$ és amelyen ezért (egy terminális és) legalább $n + 1$ nemterminális szimbólum szerepel.

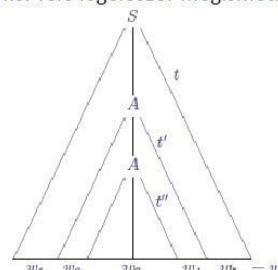
Mivel $n = |N|$, van olyan nemterminális ami ezen az úton legalább kétszer előfordul.

Jelölje A azt a nemterminális, amelyik ezen úton a terminális levéltől indulva a gyökér felé legelőször megismétlődik.

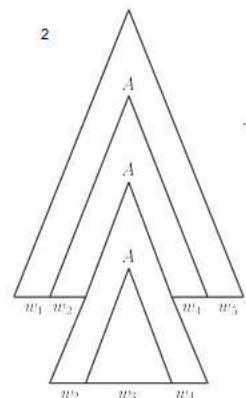
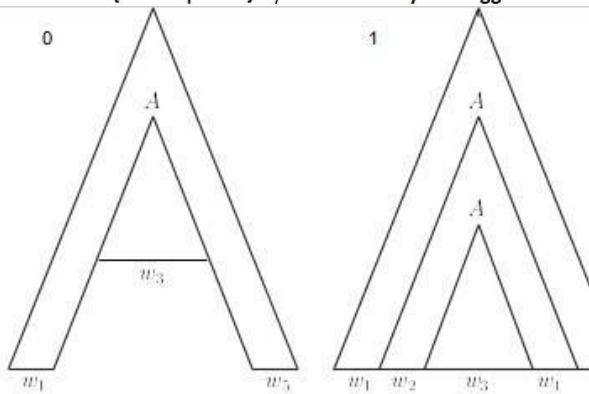
Definiálik w_1 , w_2 , w_3 , w_4 és w_5 szavakat az ábrán látható módon:

Behinde

1. $w = w_1 w_2 w_3 w_4 w_5$, mert t határát osztottuk fel
 2. $|w_2 w_3 w_4| \leq q$, mert a legelső ismétlődést vettük
 3. $w_2 w_4 \neq \lambda$, mert G láncszabály és λ -mentes
 4. minden $n \geq 0$ -ra $w_1 w_2^n w_3 w_4^n w_5 \in L$, mert a t' fa iterálható ábrák az $n = 0$, $n = 1$ és $n = 2$ esetre



Tétel. Az $L = \{a^n b^n c^n \mid n \geq 1\}$ nyelv nem környezetfüggetlen.



Tétel. Az $L = \{a^n b^n c^n \mid n \geq 1\}$ nyelv nem környezetfüggetlen:

Tehát igen. A Bar Hillel lemma szerint vannak olyan $p, q > 0$ számok, hogy minden $w \in L$ szóra, ha $|w| > p$, akkor teljesülnek ezen lemmában szereplő 1–4 feltételek.

Vegyük a $w = a^p b^p c^p \in L$ szót. Mivel $|w| = 3p$, az is igaz, hogy $|w| \geq p$.

Feltételek: $w = w_1 w_2 w_3 w_4 w_5$, $w_2 w_4 \neq \lambda$, ahol minden $n \geq 0$ -ra $w_1 w_2^n w_3 w_4^n w_5 \in L$

Hogyan helyezkedhet el w_2 és w_4 az $a^pb^pc^p$ szóban?

Sem w_2 sem w_4 nem tartalmazhat két különböző betűt, mert ekkor például a $w_1w_2w_2w_3w_4w_4w_5$ szóban a betűk sorrendje nem $a - b - c$ lenne!

Akkor n -et növelve a $w_1w_2^n w_3w_4^n w_5$ szóban csak legfeljebb két fajta

AKKOR II-ET HÖVÉVE A W₁W₂ W₃W₄ W₅ SZÓBAN ÉS KÉT LEGELJÉBB KÉT RAJTA BETÜNK A SZÁMÁ HÖVÉSZIK, TEHÁT W₁W₂ W₃W₄ W₅ € L

Tétel. $\mathcal{L}_2 \subset \mathcal{L}_1$: Azt már láttuk, hogy $\mathcal{L}_2 \subseteq \mathcal{L}_1$ (13. téTEL: λ -mentesítés).
 $L = \{a^n b^n c^n \mid n \geq 1\} \in \mathcal{L}_1$ (környezetfüggő), mert a következő nyelvtan generálja, viszont $L \notin \mathcal{L}_2$ (környezetfüggetlen)

$$B_1 B_2 \rightarrow BB_2$$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Közelítő és szimbolikus számítások

1. Tétel

- Eliminációs módszerek, mátrixok trianguláris felbontásai

Olyan mátrixot viszonylag egyszerű megadni, amellyel balról való beszorzás azt eredményezi, hogy egy a vektor k -adik alatti együtthatói eltűnnek:

$$M_k a = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & & -a_{k+1}/a_k & \\ & & & \vdots & \ddots \\ & & & -a_n/a_k & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \\ a_{k+1} \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Itt az M_k mátrixban minden ki nem írt együttható nulla, és az a_k együtthatót generálóelemek hívjuk, magát az M_k mátrixot pedig *eliminációs mátrixnak*, vagy *Gauss-transzformációnak*. Az eliminációs mátrixoknak érvényesek a következő tulajdonságai:

- M_k alsó háromszög mátrix, a főátlójában csupa egyessel, tehát reguláris (nem szinguláris) mátrix,
- $M_k = I - me_k^T$, ahol $m = (0, \dots, a_{k+1}/a_k, \dots, a_n/a_k)^T$, és e_k az egységmátrix k -adik oszlopa,
- $M_k^{-1} = I + me_k^T$, azaz M_k és az inverze csak a főátlón kívüli elemek előjelében különböznek. Az M_k^{-1} alsó háromszög mátrixot L_k -val is fogjuk jelölni.
- Két eliminációs mátrix szorzatát könnyű megadni a $k < j$ esetre:

$$M_k M_j = (I - me_k^T)(I - te_j^T) = I - me_k^T - te_j^T + me_k^T te_j^T = I - me_k^T - te_j^T,$$

mivel $e_k^T t = 0$. Két eliminációs mátrix szorzata tehát a kettő kompozíciója a fenti képlet értelmében.

- Hasonlóan az inverzekre is érvényes $L_k L_j = I + me_k^T + te_j^T$ (ha $k < j$).

PÉLDA. Legyen $a = (1, 2, -3)^T$, ekkor

$$M_1 a = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ -3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

Az előző oldal példájának Matlab megvalósítása:

```
>> a = [1 2 -3]';  

>> m = [0 a(2)/a(1) a(3)/a(1)]';  

>> M1 = eye(3);  

>> M1 = M1-m*M1(:,1)'  

M1 =  

1 0 0  

-2 1 0  

3 0 1  

>> M1*a  

ans =  

1  

0  

0
```

Itt A' az A mátrix transponáltját jelöli, a 3×3 -as egységmátrixot pedig az `eye(3)` utasítás adja. Az M_1 eliminációs mátrix inverze, L_1 a következő módon határozható meg:

```
>> L1 = M1^-1  

L1 =  

1 0 0  

2 1 0  

-3 0 1
```

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

LU FELBONTÁS, GAUSS ELIMINÁCIÓ

A lineáris egyenletrendszer megoldáshoz az $Ax = b$ egyenletrendszerből eliminációs mátrixokkal olyan szeretnénk kialakítani, amelynek baloldalán egy felső háromszögmátrix van, így a behelyettesítést végre tudjuk hajtani.

$$M_Ax = M_{n-1} \cdots M_1Ax = M_{n-1} \cdots M_1b = Mb.$$

Itt M_1 az A mátrix a_{11} elemére, mint generáló elemre támaszkodik, M_2 az M_1A mátrix 2., főátlöbeli elemére stb.:

$$\begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \end{pmatrix} M_1 \rightarrow \begin{pmatrix} x & x & x \\ 0 & x & x \\ 0 & x & x \end{pmatrix} M_2 \rightarrow \begin{pmatrix} x & x & x \\ 0 & x & x \\ 0 & 0 & x \end{pmatrix}.$$

$$A \quad M_1A \quad M_2 \quad M_2M_1A$$

Itt x az épp végrehajtott szorzás előtti mátrix elemeit jelöli, x pedig az utolsó lépéshén megváltozott, nem feltétlenül nulla együtthatókat.

Vegyük észre, hogy ez az eljárás az eredeti A mátrixot két új, háromszögmátrix szorzárára bontja: $A = LU$, ahol

$$L = M^{-1} = (M_{n-1} \cdots M_1)^{-1} = M_1^{-1} \cdots M_{n-1}^{-1} = L_1 \cdots L_{n-1}.$$

A konstrukcióból adódik, hogy $U = MA$ egy felső háromszögmátrix, az L diagonális elemei pedig egyesek.

Az eredeti feladatunkat most az LU felbontás segítségével egyszerűsítjük: $Ax = b$ helyett vegyük az $LUx = b$ egyenletrendszeret. Ezt két lépéshén oldjuk meg, először az

$$Ly = b$$

egyenletrendszert az új, mesterséges y változóra, majd az

$$Ux = y$$

egyenletet. Vegyük észre, hogy $y = Mb$.

Tekintsük a következő egyszerű lineáris egyenletrendszerét:

$$\begin{aligned} 2x_1 + 4x_2 - 2x_3 &= 2, \\ 4x_1 + 9x_2 - 3x_3 &= 8, \\ -2x_1 - 3x_2 + 7x_3 &= 10. \end{aligned}$$

Ennek szokásos, mátrix formájú alakja:

$$\begin{pmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 8 \\ 10 \end{pmatrix}.$$

Ahhoz, hogy az első oszlopban a főátló alatti elemek eltűnjenek, az első sor kétszeresét kell kivonni a második sorból, és egyszerűen hozzáadni a harmadik sorhoz:

$$M_1A = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{pmatrix} = \begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{pmatrix},$$

$$M_1b = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 8 \\ 10 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 12 \end{pmatrix}.$$

Már csak a harmadik sor második elemét kell nullává változtatni, ehhez vonjuk ki a második sort a harmadikból:

$$M_2M_1A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{pmatrix} = \begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{pmatrix},$$

$$M_2M_1b = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 4 \\ 12 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 8 \end{pmatrix}.$$

Ezzel megkaptuk az eredeti lineáris egyenletrendszerünk ekvivalens, felső háromszögmátrixos alakját:

$$\begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 8 \end{pmatrix}.$$

A kapott,

$$\begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 8 \end{pmatrix}$$

egyenletrendszer megoldása visszahelyettesítéssel: $x = (-1, 2, 2)^T$.

Az LU felbontásban az L mátrix:

$$L = L_1L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix}.$$

Ezzel az A mátrix felbontása:

$$A = \begin{pmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{pmatrix} = LU.$$

Az $Ax = b$ lineáris egyenletrendszer megoldására szolgáló Gauss elimináció formalisan a következő lépésekkel áll:

1. Az A mátrix LU felbontása (egy alsó és egy felső háromszögmátrixra),
2. $Ly = b$ megoldása y -ra,
3. $Ux = y$ megoldása x -re.

A megfelelő Matlab program az `LTriSol`, `UTriSol` és az `LU` eljárásokra építve:

```
>> [L, U] = LU(A); % LU felbontás
>> y = LTriSol(L, b); % előre behelyettesítés
>> x = UTriSol(U, y); % hatra behelyettesítés
```

A felhasznált LU eljárás Matlab kódja:

```
1 function [L,U] = LU(A)
2 [m,n] = size(A);
3 for k=1:n-1
4     A(k+1:n,k) = A(k+1:n,k)/A(k,k);
5     A(k+1:n,k+1:n) = A(k+1:n,k+1:n)-A(k+1:n,k)*A(k,k+1:n);
6 end
7 L = eye(n,n) + triu(A,-1);
8 U = triu(A);
```

Itt `tril` és `triu` az alsó-, illetve felső háromszögmátrixokat képezik az argumentum mátrixból kivágva. A második argumentum pedig a főátlótól való eltérés mértékét adja, tehát `tril(A, -1)` azt az alsó háromszögmátrixot adja, amelynek főátlójában is nullák vannak.

FŐELEMKIVÁLASZTÁS



Az LU felbontás persze csak akkor sikeres, ha az A mátrix nem szinguláris, és minden generáló elem nullától különboző. Ha a generálóelem (*főelem*, pivot elem) nulla, akkor még lehetséges lehet a felbontás – a mátrix átrendezése után, ami szintén ekvivalens feladathoz vezet. Ezt az eljárást *főelemkiválasztásnak* hívjuk.

Például az

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

mátrix reguláris, mégse hajtható végre az LU felbontás közvetlenül. Szabad viszont a sorokat felcserélni (természetesen a lineáris egyenletrendszerben a jobboldali vektor elemei együtt mozognak). A sorcserek utáni mátrix triviális LU felbontása:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = LU,$$

illetve a sorok felcserélését elvégző P , ún. *permutációs mátrixszal* formalisan:

$$PA = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = LU.$$

Jegyezzük meg, hogy ha a főátló alatt az LU felbontás egy fázisában csak nulla elem van, akkor $M_k = I$ választással tovább folytatjuk a felbontást. Ha ez olyankor következik be, amikor a főátlöbeli elem nulla volt, akkor az így kapott LU felbontás nem lesz reguláris. Ez nem csoda, hiszen az eredeti A mátrix is szinguláris kellett hogy legyen. Tekintsük például a következő esetet:

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = LU.$$

A csak a sorcserekkel alapuló főelemkiválasztást részleges főelemkiválasztásnak is nevezik. A hibaforrásoknál elmondottak miatt érdemes nagyabszolút értékű főelemeket választani.

Amikor a főelemet a teljes még szabadjó részmátrixból választjuk, *teljes főelemkiválasztásról* beszélünk.

Bár a permutáció elrontja az alsó- és felső háromszögmátrixokat, az egyenletrendszer megoldása szempontjából ezek teljes értékűek maradnak, így a jelölésben is szokás az eredeti LU alakot megtartani.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- Lineáris egyenletrendszer megoldása iterációs módszerekkel

EGYENLETRENDSZEREK ITERÁCIÓS MÓDSZEREI

Ahogy korábban már tárgyalunk, kis és közepes lineáris egyenletrendszerre, továbbá a nagyméretű sávmátrixokra a direkt módszerek ajánlhatók, a nagy méretű, de nem feltétlen ritka mátrixokkal rendelkező lineáris egyenletrendszer megoldására iterációs eljárások használatosak. Az utóbbi feladatokra a korábban bemutatott algoritmusok használhatatlanul nagy műveletigényük lehetnek.

PÉLDA. Tekintsünk egy egyszerű lineáris egyenletrendszer:

$$\begin{aligned} 4x_1 - x_2 + x_3 &= 4 \\ 2x_1 - 4x_2 + x_3 &= -1 \\ -2x_1 + x_2 + 5x_3 &= 4 \end{aligned}$$

Az egyenletrendszer megoldása az $x = (1, 1, 1)^T$ vektor. Ezeket az egyenleteket írhatjuk olyan alakban, hogy minden egyenlet bal oldalára rendezünk egy-egy változót:

$$\begin{aligned} x_1 &= \frac{4 + x_2 - x_3}{4} \\ x_2 &= \frac{1 + 2x_1 + x_3}{4} \\ x_3 &= \frac{4 + 2x_1 - x_2}{5} \end{aligned}$$

Most vegyük egy $x_0 = (1, 2, 3)^T$ indulóvektort, és azt az utóbbi egyenletrendszer jobboldalába helyettesítve határozzuk meg a baloldalon álló változók új értékeit, a következő iterált vektort. Az iteráció a következő vektorokat adja:

| k | x_1 | x_2 | x_3 |
|-----|--------|--------|--------|
| 0 | 1.0000 | 2.0000 | 3.0000 |
| 1 | 0.7500 | 1.5000 | 0.8000 |
| 2 | 1.1750 | 0.8250 | 0.8000 |
| 3 | 1.0063 | 1.0375 | 1.1050 |
| 4 | 0.9831 | 1.0294 | 0.9950 |
| 5 | 1.0086 | 0.9903 | 0.9874 |
| 6 | 1.0007 | 1.0011 | 1.0054 |
| 7 | 0.9989 | 1.0017 | 1.0001 |
| 8 | 1.0004 | 0.9995 | 0.9992 |

JACOBI ITERÁCIÓ

Ezt a bevezetett eljárást Jacobi iterációnak nevezzük. Bár az előző példa igéréses, mégis vannak esetek, amikor a Jacobi iteráció nem konvergál. Tekintsük például a vizsgált lineáris egyenletrendszer következő átrendezését (csak annyi történt, hogy az utolsó sort vettük előre):

$$\begin{aligned} -2x_1 + x_2 + 5x_3 &= 4 \\ 4x_1 - x_2 + x_3 &= -1 \\ 2x_1 - 4x_2 + x_3 &= -1 \end{aligned}$$

Az ehhez tartozó iterációs egyenletek:

$$\begin{aligned} x_1 &= \frac{-4 + x_2 + 5x_3}{2} \\ x_2 &= -4 + 4x_1 + x_3 \\ x_3 &= -1 - 2x_1 + 4x_2. \end{aligned}$$

Használjuk ismét az $x_0 = (1, 2, 3)^T$ indulóvektort, ekkor az

```
>> y = [ (-4+x(2)+5*x(3))/2, -4+4*x(1)+x(3), -1-2*x(1)+4*x(2) ]
>> x = y
```

iterációs lépések ismétlése a következő divergáló vektorsorozatot adja:

| k | x_1 | x_2 | x_3 |
|-----|----------|--------|--------|
| 0 | 1.0000 | 2.0000 | 3.0000 |
| 1 | 6.5000 | 3.0000 | 5.0000 |
| 2 | 12 | 27 | -2 |
| 3 | 6.5 | 42 | 83 |
| 4 | 226.5 | 105 | 154 |
| 5 | 435.5 | 1056 | -34 |
| 6 | 441 | 1704 | 3352 |
| 7 | 9230 | 5112 | 5933 |
| 8 | 17387 | 42849 | 1987 |
| 9 | 26390 | 71529 | 136622 |
| 10 | 377317.5 | 242178 | 233335 |

A JACOBI ITERÁCIÓ KONVERGENCIÁJA

Irjuk fel az $Ax = b$ lineáris egyenletrendszer Jacobi iterációját a következő alakban (feltéve, hogy a diagonális elemek nem nullák):

$$x^{(k+1)} = D^{-1}b - D^{-1}(A - D)x^{(k)},$$

ahol D az A mátrix diagonális elemeiből álló mátrix. Ugyanerre az egyenletre használjuk a következő, egyszerűbb jelölést:

$$x^{(k+1)} = Bx^{(k)} + c,$$

ahol tehát $B = -D^{-1}(A - D)$ és $c = D^{-1}b$. Az általános iterált vektor ekkor felirható:

$$\begin{aligned} x^{(k)} &= c + B(c + B(c + \dots x^{(0)}) \dots) = c + Bc + B^2c + \dots + B^{k-1}c + B^kx^{(0)} \\ \text{alakban. Vizsgáljuk meg két egymást követő iterált vektor eltérését:} \end{aligned}$$

$$\|x^{(k+1)} - x^{(k)}\| = \|B^k c + (B^{k+1} - B^k)x^{(0)}\| \leq \|B^k\| \|c + (B - I)x^{(0)}\|.$$

Innen látszik, hogy az $\{x^{(k)}\}$ vektorsorozat konvergenciája és a $\|B\|$ norma egynél kisebb volta összefügg. Utóbbi azt jelenti, hogy a konvergenciához

$$\|D^{-1}(A - D)\| < 1$$

sükséges. Vegyük most a végtelen normát.

$$\|B\|_\infty = \|D^{-1}(A - D)\|_\infty = \max_i \sum_{j=1}^n |b_{ij}| = \max_i \sum_{j=1, j \neq i}^n |a_{ij}/a_{ii}| < 1$$

összefüggésekkel következik, hogy

$$\sum_{j=1, j \neq i}^n |a_{ij}| < |a_{ii}|$$

teljesül minden $i = 1, 2, \dots, n$ -re (ahol A az $n \times n$ -es mátrix). Ha az utóbbit egyenlőtlenség teljesül egy mátrixra, akkor azt mondjuk, hogy az *szigorúan domináns*.

A fentiek alapján a diagonális dominancia elegendő a Jacobi iteráció konvergenciájához. Nézzük az összefüggéseket részletesebben.

AZ ITERÁCIÓS MÓDSZEREK KONVERGENCIÁJA

Vizsgáljuk meg az $x^{(k+1)} = Bx^{(k)} + c$ iteráció által definiált $\{x^{(k)}\}$ sorozat konvergenciáját. Jelöljük az eredeti egyenletrendszerünk megoldását x^* -al. Az $e_k = x^{(k)} - x^*$ eltérésre a következő állítás érvényes:

TÉTEL. Tetszőleges $x^{(0)}$ kezdővektor esetén a k -adik közelítés eltérése az x^* megoldástól $e_k = B^k e_0$.

BIZONYÍTÁS. Használjunk teljes indukciót. $k = 0$ -ra nyilvánvalóan igaz az állítás, mert ekkor $e_0 = B^0 e_0 = Ie_0$ adódik. Tegyük fel most, hogy egy adott k -ra teljesül az összefüggés, és vizsgáljuk meg a $k + 1$ esetet.

$$\begin{aligned} e_{k+1} &= x^{(k+1)} - x^* = (Bx^{(k)} + c) - (Bx^* + c) = B(x^{(k)} - x^*) \\ &= Be_k = B(B^k e_0) = B^{k+1} e_0 \end{aligned}$$

Ami igazolja a téTEL állítását. \square

A téTEL következménye, hogy ha a B mátrix *nilpotens* (azaz van olyan j index, amire $B^j = 0$), akkor $B^j e_0 = 0$, tehát az iterációs eljárás véges sok lépésben megtalálja a megoldást. Ez a feltétel teljesül pl. akkor, ha B *szigorúan trianguláris*, tehát olyan háromszögmátrix, amelynek főátlójában csak nullák vannak.

Akkor mondjuk, hogy egy iterációs sorozat *globálisan konvergens*, ha minden indulóvektorral ugyanazt a megoldást kapjuk.

Mivel a globális konvergencia esetén tetszőleges kezdővektorból indulva megkapjuk a megoldást, ezért szokásos kezdővektornak a c vektort használni.

Iterációs eljárásokat használunk olyan esetben is, amikor az eliminációs eljárással kapott, kerekítési hibákkal terhelt közelítő megoldást kell pontositani.

- Bizonyítsuk be $\rho(B) < 1$ esetén a B mátrixhoz tartozó iteráció globális konvergenciáját a sajátértékek, sajátvektorok tulajdonságai alapján!

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

AZ ITERÁCIÓS MÓDSZEREK KONVERGENCIÁJA / 2.

A következő elméleti állítás azon a lineáris algebrai tényen alapul, hogy egynél kisebb spektrálsgarájú mátrixhoz létezik olyan mátrixnorma, amelyik a mátrixra egynél kisebb értéket ad.

TÉTEL. Az $x^{(k+1)} = Bx^{(k)} + c$ iteráció akkor és csak akkor globálisan konvergens, ha $\rho(B) < 1$.

BIZONYÍTÁS. Tegyük fel először, hogy az eljárás globálisan konvergens, és mégis $\rho(B) \geq 1$. Ekkor a maximális abszolút értékű λ sajátértékre $|\lambda| \geq 1$. Válasszuk ezután az $x^{(0)}$ indulóvektort úgy, hogy a λ sajátértékhez tartozó x normált sajátvektor legyen az e_0 eltérésvektor. Mivel az előző tétel szerint ebben az esetben $e_k = B^k e_0 = \lambda^k e_0$, ezért

$$\|e_k\| = \|\lambda^k e_0\| = |\lambda^k| \|e_0\| = |\lambda|^k 1 \geq 1$$

azaz e_k nem konvergal nullához. Ez pedig ellentmond annak, hogy a módszer globálisan konvergens.

Indulunk ki most abból, hogy $\rho(B) < 1$. Ekkor van olyan $\|\cdot\|$ mátrixnorma, hogy $\|B\| < 1$. Ezzel pedig tetszőleges kezdővektorra és vektornormára igaz, hogy

$$\|e_k\| = \|B^k e_0\| \leq \|B^k\| \|e_0\| \leq \|B\|^k \|e_0\| \rightarrow 0.$$

Tehát az eljárás globálisan konvergens. \square

Ha megadható olyan $\|\cdot\|$ mátrixnorma, amelyre $\|B\| < 1$, akkor érvényesek a következő hibabecslések:

$$\begin{aligned} \|e_{k+1}\| &\leq \|B\| \|e_k\|, \\ \|e_k\| &\leq \|B\|^k \|e_0\|, \\ \|e_k\| &\leq \frac{\|B\|}{1 - \|B\|} \|x^{(k)} - x^{(k-1)}\|, \\ \|e_k\| &\leq \frac{\|B\|^k}{1 - \|B\|} \|x^{(1)} - x^{(0)}\|, \\ \|e_k\| &\leq \frac{\|B\|^{k+1}}{1 - \|B\|} \|c\|, \text{ ha } x^{(0)} = c. \end{aligned}$$

Ha az A mátrix diagonális domináns, akkor a hozzá tartozó B mátrixra a spektrálsgarájú kisebb mint egy, tehát a belőle adódó Jacobi iteráció globálisan konvergens.

• Mátrixok sajátértékeinek és sajátvektorainak numerikus meghatározása

MÁTRIXOK SAJÁTÉRTÉKEI ÉS SAJÁTVEKTORAI

Legyen adott egy A négyzetes mátrix. Adjuk meg a λ számot és az $x \neq 0$ vektort úgy, hogy

$$Ax = \lambda x.$$

Itt λ az A mátrix sajátértéke, x pedig a sajátvektora. Egy hasonló definícióval értelmezhető a baloldali sajátérték és sajátvektor:

$$y^T A = \lambda y^T.$$

Mivel y sajátvektora az A^T mátrixnak, ezért a tövábbiakban csak az első feladatot vizsgáljuk. Egy mátrix sajátértékeit halmazát a mátrix spektrumának hívjuk, és $\lambda(A)$ -val jelöljük. A spektrálrádiusszám pedig a $\max\{|\lambda| : \lambda \in \lambda(A)\}$ mennyiséget jelenti, és $\rho(A)$ -val jelöljük.

A sajátvektorok irányába eső vektorokat az A mátrix megnagyítja az illető sajátvektorhoz tartozó sajátértéknak megfelelően. Igy egy mátrixszal reprezentált lineáris transzformáció hatását egyszerűen megadhatjuk a sajátértéki és sajátvektorai ismeretében. Számos fizikai és matematikai feladat (pl. rezonancia-vizsgálat, illetve differenciálegyenletek stabilitási kérdése) sajátértek számítással jól megoldható.

A sajátértékek vizsgálata komplex számokra is kiterjeszthető. Az egyetlen lényeges különbség, hogy akkor az A^T transponált mátrix helyett a konjugált transponáltat, A^H -t kell használni.

A sajátértékek és a sajátvektorok nem egyértelműek:

- például az egységmátrixnak az 1-n-szeres sajátértéke,
- egy sajátvektorral együtt annak minden nem nulla számmal szorzottja is ugyanahoz a sajátértékhez tartozó sajátvektora.

A sajátértékeket, sajátvektorokat meghatározhatjuk az

$$(A - \lambda I)x = 0$$

homogén lineáris egyenletrendszerből. Ennek pontosan akkor van a nulla vektortól különböző megoldása, ha az $A - \lambda I$ mátrix szinguláris. Emiatt a sajátértékeket leírja a

$$\det(A - \lambda I) = 0$$

egyenlet. Ennek baloldalán levő n -edfokú polinomot az A mátrix karakterisztikus polinomjának nevezzük.

GAUSS-SEIDEL ITERÁCIÓ

A Jacobi iteráció tanulmányozása során felmerülhet, hogy mivel egy a megoldáshoz közelítő vektorsorozatot akarunk generálni, ezért talán javítani lehet a konvergencia sebességén, ha az első új vektorkomponensek meghatározása után az iterációs képlet jobb oldalán már ezeket az új értékeket használjuk. Ez a Gauss-Seidel iteráció. A kapcsolódó képletek az eredeti feladatra:

$$\begin{aligned} x_1^{k+1} &= \frac{4 + x_2^k - x_3^k}{4} \\ x_2^{k+1} &= \frac{1 + 2x_1^{k+1} + x_3^k}{4} \\ x_3^{k+1} &= \frac{4 + 2x_1^{k+1} - x_2^{k+1}}{5}. \end{aligned}$$

Ismételjük meg most a Jacobi iterációval végzett első kísérletet. Az eredménye:

| k | x_1 | x_2 | x_3 |
|-----|--------|--------|--------|
| 0 | 1.0000 | 2.0000 | 3.0000 |
| 1 | 0.7500 | 1.3750 | 0.8250 |
| 2 | 1.1375 | 1.0250 | 1.0500 |
| 3 | 0.9938 | 1.0094 | 0.9956 |
| 4 | 1.0034 | 1.0006 | 1.0013 |
| 5 | 0.9998 | 1.0002 | 0.9999 |
| 6 | 1.0001 | 1.0000 | 1.0000 |
| 7 | 1.0000 | 1.0000 | 1.0000 |

Bár néhány iterált érték ből nemigen lehet megállapítani a konvergencia sebességét, azt mégis leszögezhetjük, hogy a kapott eredményünk lényegesen jobb, mint ami a Jacobi iterációval adódott.

Vigyázat, az

$$y = [(4+x(2)-x(3))/4 \quad (1+2+y(1)+x(3))/4 \quad (4+2+y(1)-y(2))/5]$$

Matlab utasítás ugyanazt az eredményt adja, mint amit a Jacobi iterációval kaptunk, mert a Matlab a jobboldal kiértékelése során az y vektor korábbi értékeivel számol, nem egyszerűen balról jobbra határozza meg az x vektort!

MÁTRIXOK SAJÁTÉRTÉKEI ÉS SAJÁTVEKTORAI / PÉLDÁK

Néhány mátrix a sajátértékeivel, az azokhoz tartozó sajátvektorokkal és a karakterisztikus polinommal:

$$\begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} : \quad \lambda = 1, x = (1, 0)^T, \quad \lambda = 2, x = (0, 1)^T, \quad \text{és} \quad (1 - \lambda)(2 - \lambda),$$

$$\begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix} : \quad \lambda = 1, x = (1, 0)^T, \quad \lambda = 2, x = (1, 1)^T, \quad \text{és} \quad (1 - \lambda)(2 - \lambda),$$

$$\begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix} : \quad \lambda = 2, x = (1, 1)^T, \quad \lambda = 4, x = (1, -1)^T, \quad \text{és} \quad \lambda^2 - 6\lambda + 8,$$

$$\begin{pmatrix} 1.5 & 0.5 \\ 0.5 & 1.5 \end{pmatrix} : \quad \lambda = 2, x = (1, 1)^T, \quad \lambda = 1, x = (-1, 1)^T, \quad \text{és} \quad \lambda^2 - 3\lambda + 2$$

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} : \quad \lambda = 1, x = (1, 1)^T, \quad \lambda = -1, x = (1, 1)^T, \quad \text{és} \quad \lambda^2 + 1,$$

ahol $i = \sqrt{-1}$. Ha egy valós mátrixnak van komplex sajátértéke, akkor a sajátértek komplex konjugáltja is sajátértek.

Azt mondjuk, hogy a $\|\cdot\|_{(v)}$ vektornorma kompatibilis az $\|\cdot\|_{(m)}$ mátrixnormával, ha tetszőleges A négyzetes mátrixra igaz

$$\|Ax\|_{(v)} \leq \|A\|_{(m)} \|x\|_{(v)}.$$

A származtatott mátrixnormák nyilvánvalóan kompatibilisek az össükkel, azzal a vektornormával, amellyel definiáltuk őket.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

A SAJÁTÉRTÉKEK KORLÁTAI

TÉTEL. Tetszőleges A mátrixra és bármely mátrixnormára

$$\rho(A) \leq \|A\|,$$

azaz a spektralsugár nem nagyobb mint a mátrix normája.

BIZONYÍTÁS. Legyen $\|\cdot\|_v$ az adott mátrixnormával kompatibilis vektornorma, λ_m az A mátrix maximális abszolt értékű sajátértéke, és x_m a hozzá tartozó sajátvektor. Ekkor

$$\rho(A) = \max_{1 \leq i \leq n} |\lambda_i| = |\lambda_m|,$$

tehát

$$\rho(A) \|x_m\|_v = |\lambda_m| \|x_m\|_v = \|\lambda_m x_m\|_v = \|Ax_m\|_v \leq \|A\| \|x_m\|_v.$$

Innen a nem nulla $\|x_m\|_v$ -vel leosztva kapjuk az igazolandó egyenlőtlenséget. \square

TÉTEL. (Gersgorin) Az A mátrix összes sajátértéke benne van a

$$K_i = \left\{ z \in C \mid |z - a_{ii}| \leq \sum_{k=1, k \neq i}^n |a_{ik}| \right\}$$

körök $K = \cup_{i=1}^n K_i$ egyesítésében.

Érvényes ennek a tételnek egy olyan előírása, hogy amennyiben a definiált körök egy m , és egy $n - m$ körből álló halmazra bonthatók úgy, hogy a két részhalmaz pontjai diszjunktak, akkor az első körhalmaz multiplicitással számolva pontosan m darab sajátértéket tartalmaz.

PÉLDA. Alkalmazzuk a fenti elmeleti eredményeket az alábbi mátrixra:

$$A = \begin{pmatrix} 1 & 1 & -1 \\ 2 & -2 & 0 \\ 1 & 0 & 5 \end{pmatrix}.$$

Ezek szerint minden sajátérték abszolut értéke legfeljebb $\|A\|_1 = 6$, $\|A\|_\infty = 6$, és $\|A\|_F = \sqrt{37}$, azaz $\rho(A) \leq 6$. A sajátértékekkel pedig kettő van az 1 középpontú, és 2 sugarú, valamint a -2 középpontú, 2 sugarú körök uniójában, és egy sajátérték van az 5 középpontú és 1 sugarú körben (a komplex síkon).

MÁTRIXOK SAJÁTÉRTÉKEI, HASONLÓSÁGI TRANSZFORMÁCIÓ / 2

TÉTEL. Az A mátrix akkor és csak akkor diagonalizálható, ha létezik n elemű lineárisan független sajátvektor-rendszere.

A legtöbb, amit általános esetben is elérhetünk, az az, hogy az A mátrixot hasonlósági transzformációkkal ún. *Jordan* alakra hozzuk: olyan mátrixot kapunk, amelyben a föátló mellett a felső mellékátló tartalmazhat még nullától különböző elemeket. Sajnos a Jordan alakra hozás numerikusan nem stabilis, ezért nem szokás használni. Emiatt nincs a Matlabban *Jordan* nevű algoritmus.

Az viszont stabil eljárás, amely hasonlósági transzformációkkal egy mátrixot felső háromszög mátrixszá alakít (ez az ún. *Schur-féle normálalak*), és ekkor is a föátló elemei adják a sajátértékeket.

Az alábbi táblázat egyes mátrix típusokat tartalmaz a hozzájuk tartozó hasonlósági mátrix fajtával és a transzformált mátrix típusával.

| A | T | $B = T^{-1}AT$ |
|-------------------------------|-------------|------------------------|
| páronként eltérő sajátértékek | reguláris | diagonális |
| szimmetrikus | ortogonális | valós diagonális |
| Hermite-szimmetrikus | unitér | valós diagonális |
| normális | unitér | diagonális |
| tetszőleges | unitér | felső háromszög mátrix |
| tetszőleges | reguláris | Jordan alakú |

Minden esetben a B diagonális elemei a sajátértékek, és az első négy esetben a T oszlopai sajátvektorok.

A Matlabban a sajátértékszámításra a következő fontosabb eljárások állnak rendelkezésre:

| | |
|---------|--|
| eig | az összes sajátérték és sajátvektor meghatározása |
| hess | Hessenberg alakra hozás |
| schur | Schur-féle normálalakra (felső háromszög mátrix) hozás |
| eigs | néhány sajátérték és sajátvektor |
| condeig | a sajátértékek kondíciósáma |
| poly | a mátrix karakterisztikus polinomját adja |
| eigshow | illusztrálja a 2×2 -es mátrixok sajátértékeit |

MÁTRIXOK SAJÁTÉRTÉKEI, A HASONLÓSÁGI TRANSZFORMÁCIÓ

Azt mondjuk, hogy az A mátrix *hasonló* a B mátrixhoz, ha létezik olyan T reguláris *hasonlósági mátrix*, hogy

$$B = T^{-1}AT.$$

Tekintsük most a B mátrix egy λ sajátértékét. Erre érvényes, hogy:

$$By = \lambda y \quad \Rightarrow \quad T^{-1}ATy = \lambda y \quad \Rightarrow \quad A(Ty) = \lambda(Ty),$$

tehát λ sajátértéke A -nak is (még ha más, Ty sajátvektorral is). Eszerint a hasonlósági transzformációk azok, amelyekkel egy mátrixot átalakíthatunk olyan kedvezőbb alakra, amelyről a sajátértékeket már könnyen le tudjuk olvasni.

A fordított összefüggés nem érvényes: ha két mátrixnak megegyeznek a sajátértékei, akkor még nem feltétlenül hasonló mátrixok. Például ez érvényes a következő mátrixokra (ahogy könnyen belátható):

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \text{és} \quad \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

Ebből az is következik, hogy hasonlósági mátrixokkal nem minden tudunk egy mátrixból hasonlósági transzformációkkal egy diagonális mátrixt kapni. Márpedig nyilvánvaló, hogy a diagonális mátrixok sajátértékei a diagonálisból számos, és a sajátvektorok pedig az ezeknek megfelelő egységvektorok.

A sajátvektor definíciójából adódik, hogy ha az A mátrix sajátvektoraiból állítjuk össze az X mátrixot, akkor például

$$AX = \begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 4 \end{pmatrix} = XD,$$

ahol D az a diagonális mátrix, amely A sajátértékeit tartalmazza. Tekintsük általános esetben a következő mátrixokat:

$$D = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n), \quad \text{és} \quad X = (x_1, x_2, \dots, x_n),$$

ahol λ_i az A mátrix i -edik sajátértéke, és x_i a hozzá tartozó sajátvektor. Ha az X mátrix reguláris, akkor

$$A = XDX^{-1}, \quad \text{és} \quad D = X^{-1}AX.$$

Ebből azonnal adódik a következő

SAJÁTÉRTÉKEK KONDÍCIONÁLTSÁGA

A sajátértékek kondícionáltsága azt mutatja meg, hogy az adott mátrix együtthatóiban való kis változások milyen hatásuk a sajátértékekre. Ez különbözik attól, hogy az adott mátrix kis eltérései mennyire hatnak a vele felírt lineáris egyenletrendszer megoldásairra. Egy adott mátrix sajátértékei is eltérhetnek a mátrix hibáira vonatkozó érzékenységek szerint.

Az A mátrix λ sajátértéke kondíciósáma az

$$\frac{1}{|y^H x|}$$

képlet határozza meg, ahol x és y az A mátrix λ sajátértékéhez tartozó bal- és jobboldali normalizált sajátvektorai, tehát

$$x^H x = y^H y = 1.$$

Más szóval a λ sajátérték kondíciósáma a bal- és a jobb sajátvektorai által bezárt szög koszinuszának reciproka. Ha ez a szám nagy, akkor λ rosszul kondícionált.

Egy szimmetrikus, vagy Hermite-szimmetrikus mátrix azonos sajátértékekhez tartozó bal- és jobb sajátvektorai megegyeznek. Emiatt

$$y^H x = x^H x = 1$$

adódik a normalizált sajátvektorokra, tehát a sajátértékek minden jól kondícionáltak. Általánosabban az is igaz, hogy a normális mátrixok (amelyekre $A^T A = AA^T$) sajátértékei is jól kondícionáltak.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

SAJÁTÉRTÉKEK KONDICIONÁLTSÁGA / PÉLDA

Vizsgáljuk meg az

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 0.999 & 1 \\ 0 & 0 & 2 \end{pmatrix}$$

mátrix sajátértékei kondicionáltságát. A hozzá tartozó normalizált (szokásos, jobb) sajátvektorok 4 jegyre:

$$x_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad x_2 = \begin{pmatrix} -1 \\ 0.0005 \\ 0 \end{pmatrix}, \quad x_3 = \begin{pmatrix} 0.9623 \\ 0.1923 \\ 0.1925 \end{pmatrix}.$$

Az ezekhez tartozó bal sajátvektorok:

$$y_1 = \begin{pmatrix} 0.0004 \\ 0.7066 \\ -0.7066 \end{pmatrix}, \quad y_2 = \begin{pmatrix} 0 \\ 0.7075 \\ -0.7068 \end{pmatrix}, \quad y_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

A sajátértékek persze a diagonálisbeli elemek:

$$\lambda_1 = 1, \quad \lambda_2 = 0.999 \quad \text{és} \quad \lambda_3 = 2.$$

A sajátértékek kondíciósáma a következő értékeket kapjuk rendre:

$$\frac{1}{|y_1^T x_1|} = 2.5 \cdot 10^4,$$

$$\frac{1}{|y_2^T x_2|} = 2.8269 \cdot 10^3,$$

$$\frac{1}{|y_3^T x_3|} = 5.1948.$$

Ezrinyen a $\lambda_1 = 1$ és a $\lambda_2 = 0.999$ sajátértékek viszonylag rosszul kondicionáltak, míg $\lambda_3 = 2$ jól kondicionált. A tényleges változások szemléltetéséhez változtassuk meg a mátrixban a_{31} értékét 0.000001-re. Az új mátrix sajátértékei rendre

$$\tilde{\lambda}_1 = 0.9995 + 0.0013i, \quad \tilde{\lambda}_2 = 0.9995 - 0.0013i \quad \text{és} \quad \tilde{\lambda}_3 = 2.$$

Ez azt jelenti, hogy az első két sajátérték a mátrixbeli kis eltéréshez képest nagyon megváltozott, ráadásul ezek komplex számokká váltak, míg a harmadik sajátérték nem változott.

A mátrixok sajátértékei kondíciósáma a `condeig` Matlab utasítás adja.

MÁTRIXOK SAJÁTÉRTÉKEI, LR TRANSZFORMÁCIÓ

Az *LR transzformáció* a H. Rutishauser által 1959-ben megadott eljárás, amely az LR felbontáson alapulva egy olyan mátrixsorozatot ad meg, amellyel meghatározhatók az A mátrix sajátértékei.

Az eljárás az $A_1 = A$ mátrixból indul, és a következő két lépésből áll:

$$A_k = L_k R_k,$$

$$A_{k+1} = R_k L_k.$$

Tehát az aktuális mátrixot felbontjuk egy alsó- és egy felső háromszögmátrixra, majd a következő iterált mátrixot úgy kapjuk, hogy ezeket fordított sorrendben összeszorozzuk. Az L_k alsó háromszögmátrix főátlójában egyesek vannak.

Legyen $T_k = L_1 L_2 \cdots L_k$ és $U_k = R_k R_{k-1} \cdots R_1$. Ekkor érvényes a következő

SEGÉDTÉTEL. Ha minden $k = 1, 2, \dots$ indexre létezik a fent definiált A_k mátrix, akkor

i, az A mátrix hasonló A_k -hoz,

$$\text{ii, } A_k = (L_1 L_2 \cdots L_{k-1})^{-1} A_1 (L_1 L_2 \cdots L_{k-1}) = T_{k-1}^{-1} A_1 T_{k-1},$$

$$\text{iii, } A_k^k = T_k U_k.$$

BIZONYÍTÁS. Az i, állítás adódik az ii.-ból. A többire alkalmazzunk teljes indukciót. A $k = 2$ esetben mindenkor nyilvánvalónak teljesül. Vegyük először az ii., állítást. Mivel L_k reguláris, ezért $A_{k+1} = L_k^{-1} A_k L_k$, és az indukciós feltevés miatt

$$\begin{aligned} A_{k+1} &= L_k^{-1} (L_1 L_2 \cdots L_{k-1})^{-1} A_1 (L_1 L_2 \cdots L_{k-1}) L_k \\ &= (L_1 L_2 \cdots L_k)^{-1} A_1 (L_1 L_2 \cdots L_k) \\ &= T_k^{-1} A_1 T_k. \end{aligned}$$

Tehát az ii., állítás igaz a $k + 1$ esetre is. Tekintsük most az iii., pontot hasonló gondolatmenetet követve:

$$\begin{aligned} A_1^{k+1} &= A_1 A_1^k = A_1 T_k U_k = A_1 (L_1 L_2 \cdots L_k) (R_k R_{k-1} \cdots R_1) \\ &= (L_1 L_2 \cdots L_k) A_{k+1} (R_k R_{k-1} \cdots R_1) \\ &= (L_1 L_2 \cdots L_k) L_{k+1} R_{k+1} (R_k R_{k-1} \cdots R_1) = T_{k+1} U_{k+1}. \end{aligned}$$

Amivel a harmadik részt is igazoltuk. \square

SAJÁTVEKTOROK KONDICIONÁLTSÁGA

A sajátvektorok kondicionáltsága a hozzájuk tartozó sajátérték kondicionáltságától és a többi sajátértéktől mért eltérésétől függ. A többszörös, vagy egymáshoz közeli sajátértékek sajátvektorai rosszul kondicionáltak lehetnek, különösen, ha a mátrix nem reguláris. Utóbbi esetben a mátrix sajátértékfeladatának kondicionáltságát diagonális hasonlósági transzformációkkal lehet javítani. A Matlab ebből a célból a `balance` utasítást ajánlja.

PÉLDA. Vizsgáljuk meg most az

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.99 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

diagonális mátrix sajátvektorai kondicionáltságát. A sajátértékek nyilvánvalóan a főátlöbeli $\lambda_1 = 1$, $\lambda_2 = 0.99$ és $\lambda_3 = 2$. Mivel a mátrix szimmetrikus, és minden sajátérték egyszeres, ezért a sajátértékek jól kondicionáltak. A sajátvektorok:

$$x_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad x_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad x_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Tekintsük most a kissé megváltoztatott újabb mátrixot:

$$\bar{A} = \begin{pmatrix} 1 & 0.0001 & 0 \\ 0.0001 & 0.99 & 0 \\ 0 & 0 & 2 \end{pmatrix}.$$

Ennek sajátértékei ismét 1, 0.99 és 2. Az új sajátvektorok viszont:

$$\bar{x}_1 = \begin{pmatrix} -1 \\ -0.01 \\ 0 \end{pmatrix}, \quad \bar{x}_2 = \begin{pmatrix} 0.01 \\ -1 \\ 0 \end{pmatrix}, \quad \bar{x}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

A harmadik sajátvektor nem változott, de a többi kettő meglehetősen. Ennek az oka, hogy a megfelelő sajátértékek közel vannak egymáshoz.

MÁTRIXOK SAJÁTÉRTÉKEI, LR TRANSZFORMÁCIÓ / POLYTATÁS

Az LR algoritmus konvergencia-tétele:

TÉTEL. Ha az A valós négyzetes mátrix teljesít az alábbi feltételeket:

1. az $A_k = L_k R_k$ trianguláris felbontás létezik minden $k \geq 1$ -re,

2. az A sajátértékeit alkalmasan indexelve érvényes

$$|\lambda_1| > |\lambda_2| > \dots > |\lambda_n| > 0$$

(amiből adódik, hogy A_1 reguláris és sajátértékei mindenkor valósak és egyszerek), és

3. megadható az A mátrixot diagonalizáló olyan X mátrix, amellyel egyszerű

$$X^{-1} A_1 X = D = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n),$$

mársrészt létezik mindenkor $X = \bar{L} \bar{R}$, mindenkor $X^{-1} = \bar{L} \bar{R}$ trianguláris felbontás,

akkor az $\{A_k\}$, $\{R_k\}$ és az $\{L_k\}$ mátrix sorozatok minden konvergensek:

$$\lim_{k \rightarrow \infty} A_k = \lim_{k \rightarrow \infty} R_k = \begin{pmatrix} \lambda_1 & x & \dots & x \\ 0 & \lambda_2 & \dots & x \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix} \quad \text{és} \quad \lim_{k \rightarrow \infty} L_k = I.$$

Az első két sorozat tehát egy olyan mátrixhoz tart, amelynek főátlójában az A_1 mátrix sajátértékei vannak nagyság szerint rendezve. Az LR algoritmus legfőbb hátránya, hogy még erős feltételek teljesülése esetén is előfordulhat, hogy valamely k -ra az A_k iterált mátrix nem bontható fel $L_k R_k$ alakban.

Az iteráció megállási feltétele lehet például, hogy az iterált A_k mátrix főátló alatti elemei négyzetösszege kisebb egy előre megadott kis $\epsilon > 0$ számnál.

Általános A mátrixra egy LR iteráció műveletigénye $\mathcal{O}(n^3)$, felső Hessenberg alakú A mátrixra $\mathcal{O}(n^2)$, és tridiagonális kiindulási mátrixra (amelynek a főátló mellett legfeljebb a mellékátlókban vannak nem nulla elemei) pedig $\mathcal{O}(n)$.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

2. Tétel

- Érintő, szelő és húr módszer, a konjugált gradiens eljárás

A SZELŐMÓDSZER

Legyen x^* az $f(x) = 0$ egyenlet egyszeres gyöke. Válasszunk alkalmas x_0 és x_1 kezdőértékeket, és ezekből indulva hajtsuk végre azt az iterációt, amit a következő képlet definíál:

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f'(x_k) - f'(x_{k-1})} = \frac{f(x_k)x_{k-1} - f(x_{k-1})x_k}{f(x_k) - f(x_{k-1})} \quad k = 1, 2, \dots$$

Könnyen belátható, hogy x_{k+1} nem más, mint az $(x_k, f(x_k))$ és az $(x_{k-1}, f(x_{k-1}))$ pontokon átmenő egyenes és az x tengely metszéspontja x koordinátája. Az iterációs képlet első alakja azt is megmutatja, hogy itt a Newton-módszer olyan módosításáról van szó, amikor $f'(x_k)$ helyett annak közelítéseként a numerikus derivált,

$$\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

szerepel. Ez alapján a szelőmódszer tehát olyan iterációs eljárás, amely az $f(x) = 0$ egyenlet megoldásához csak az $f(x)$ függvényt kiszámító szubrutinra támaszkodik.

Amennyiben az $f(x)$ függvény kétszer folytonosan differenciálható az x^* gyök egy $S_\delta(x^*)$ környezetében, akkor vannak olyan indukciók, amelyekkel a szelőmódszer az x^* megoldáshoz konvergál, és a közelítés hibájára érvényes, hogy

$$|e_k| \leq \frac{2\mu_1}{\mu_2} |e_{k-1}| |e_{k-2}|,$$

ahol $\mu_1 \leq |f'(x)|$ és $f''(x) \leq \mu_2$ az $S_\delta(x^*)$ környezetbeli minden pontra.

A szelőmódszert szokás olyan kezdőértékekkel indítani, amelyek közrefogják a keresett x^* gyököt. Másrészt ha $f'(x^*) > 0$, és $f''(x^*) > 0$, akkor x^* -nál nagyobb (de ahhoz közelebb) kezdőértékekkel szigorúan monoton konvergencia érhető el.

Azt az algoritmus-változatot, amely felteszi, hogy a kezdeti x_0, x_1 pontokban az $f(x)$ függvény ellentétes előjelű, és $f'(x_{k+1})$ előjelle függvényében a megelőző két pontból azt választja a következő iterációs lépéshez, amelyikkel ez a tulajdonság fennmarad, húrmódszernek hívjuk.

A NEWTON-MÓDSZER

Tegyük fel, hogy az $f(x) = 0$ egyenlet x^* egyszeres, izolált zérushelyét akarjuk meghatározni, és hogy ennek egy környezetében $f(x)$ differenciálható. Válasszunk ki ebből a környezetből egy x_0 kezdőértéket, majd képezzük az

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

iterációs sorozatot. Ezt az eljárást érintő-, vagy Newton-módszernek nevezünk. Az iterációs képlet geometriai értelmezése az, hogy az aktuális x_k pontban meghatározzuk az $f(x)$ függvény és deriváltja értékét, ezekkel képezzük az adott ponthoz húzott érintőt, és következő iterációs pontnak azt határozzuk meg, amelyben az érintőnek zérushelye van.

SEGÉDTÉTEL. Ha az $f(x) = 0$ egyenlet x^* egyszeres zérushelyének egy környezetében $f(x)$ kétszer differenciálható, akkor megadható olyan $0 < \delta$, hogy ha $x_k \neq x^*$ benne van az x^* körül δ sugarú $S_\delta(x^*)$ környezetben (intervallumban), akkor létezik az $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$ közelítés, és érvényes az

$$|e_{k+1}| = \left| \frac{f''(\eta_k)}{2f'(x_k)} \right| e_k^2$$

hibabecslés valamely olyan η_k -ra, amely benne van az x_k és x^* nyitott burkában.

BIZONYÍTÁS. Fejtsük $f(x)$ -et x_k körül Taylor-sorba:

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{f''(\eta_k)}{2!}(x - x_k)^2.$$

Helyettesítünk most be az x^* értéket:

$$0 = f(x^*) = f(x_k) + f'(x_k)(x^* - x_k) + \frac{f''(\eta_k)}{2!}(x^* - x_k)^2.$$

Az iterációs képletből

$$f(x_k) = f'(x_k)(x_k - x_{k+1}).$$

Az utolsó két egyenletből pedig az adódik, hogy

$$0 = f'(x_k)(x^* - x_{k+1}) + \frac{f''(\eta_k)}{2!}(x^* - x_k)^2.$$

Ez az összefüggés már közvetlenül a segédtétel állítását igazolja. \square

A SZELŐMÓDSZER – PÉLDA

Tekintsük ismét a Newton-módszerrel már vizsgált feladatot: $f(x) = x^2 - x$. Két kezdőpont párral indítsuk az eljárást: 2 és 1.3 legyen az egyik, és 1.5, 0.5 a másik.

A Matlabbal kapott eredményeket a következő táblázat tartalmazza.

| iteráció | 1. közelítés | 2. közelítés | húrmódszer |
|----------|------------------|------------------|------------------|
| 0 | 2.00000000000000 | 1.50000000000000 | 1.50000000000000 |
| 1 | 1.33333333333333 | 0.50000000000000 | 0.50000000000000 |
| 2 | 1.14285714285714 | 0.75000000000000 | 0.75000000000000 |
| 3 | 1.03225806451613 | 1.50000000000000 | 0.90000000000000 |
| 4 | 1.00392156862745 | 0.90000000000000 | 0.96428571428571 |
| 5 | 1.00012208521548 | 0.96428571428571 | 0.98780487804878 |
| 6 | 1.00000047683739 | 1.00413223140496 | 0.99590163934426 |
| 7 | 1.00000000005821 | 0.99984760743676 | 0.99863013698630 |
| 8 | 1 | 0.9999937277492 | 0.99954296160878 |
| 9 | | 1.00000000009560 | 0.99984760743676 |
| 10 | | 1 | 0.99944919731762 |
| 11 | | | 0.9998306519898 |
| 12 | | | 0.9999435500260 |
| 13 | | | 0.9999811832712 |
| 14 | | | 0.9999937277492 |
| 15 | | | 0.9999979092489 |
| 16 | | | 0.9999993030829 |
| 17 | | | 0.9999997676943 |
| 18 | | | 0.9999999225648 |
| 19 | | | 0.9999999741883 |
| 20 | | | 0.9999999913961 |

A pontos jegyek számának duplázódását később ugyan, de meg lehet figyelni a szelőmódszer esetén is. Mivel a Newton-módszer értékesebb információt is igényel, ezért természetes, hogy annak konvergienciája gyorsabb volt. A húrmódszer lassabb konvergienciája az egyik alappontról változatlanul maradása miatt adódott.

A NEWTON-MÓDSZER / 2

A előző segédtétel alapján bizonyítható, hogy ha az $f(x)$ függvény kétszer folytonosan differenciálható az x^* zérushely egy környezetében, akkor van olyan pont, ahonnan indulva a Newton-módszer kvadratikusan konvergens sorozatot ad meg:

$$|x^* - x_{k+1}| \leq C|x^* - x_k|^2$$

valamely pozitív C konstansossal.

Ha az eljárást az $f(x)$ függvény deriváltjára alkalmazzuk, akkor optimalizálási módszert kapunk. Ennek az algoritmusnak van többváltozós változata is. Ott a deriválttal való osztás helyett a második parciális deriváltakat tartalmazó Hesse mátrix inverzével kell szorozni:

$$x_{k+1} = x_k - H^{-1}(x_k) \nabla f(x_k).$$

PÉLDA. A zérushely keresésre való Newton-módszer sebességének bemutatására tekintsük az $f(x) = x^2 - x$ függvényt. Ennek két zérushelye van, a nulla és az egy. Válasszuk indulópontnak az $x_0 = 2$ -t. Az $f(x)$ függvény deriváltja $f'(x) = 2x - 1$. Ezzel az iterációs egyenlet:

$$x_{k+1} = x_k - \frac{x_k^2 - x_k}{2x_k - 1}.$$

Az iterációs sorozatot az alábbi táblázat tartalmazza:

| iteráció | közelítés |
|----------|------------------|
| 0 | 2.00000000000000 |
| 1 | 1.33333333333333 |
| 2 | 1.06666666666667 |
| 3 | 1.00392156862745 |
| 4 | 1.00001525902190 |
| 5 | 1.00000000023283 |
| 6 | 1 |

Figyeljük meg, hogy az eredményen szépen látszik, ahogy a pontos jegyek száma minden lépésben közel megduplázódik, és az is, hogy az utolsó iterált már megkülönböztethetetlen a pontos megoldástól.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

KONJUGÁLT GRADIENS MÓDSZER

A konjugált gradiens módszer az optimalizálás elvein alapul, és szimmetrikus pozitív definit mátrixú lineáris egyenletrendszer megoldására alkalmas. Pontos aritmetikával ugyan véges sok lépében megtalálná a megoldást, de a kerekítési hibák miatt mégis iterációs eljárásnak kell tekinteni. Számos variánsa ismert.

Legyen A egy szimmetrikus, pozitív definit mátrix, akkor a

$$q(x) = \frac{1}{2}x^T Ax - x^T b$$

kvadratikus függvénynek egyetlen x^* minimumpontja van, és erre $Ax^* = b$ teljesül. Más szóval az $Ax = b$ lineáris egyenletrendszer megoldása ekvivalens a $q(x)$ kvadratikus függvény minimumpontjának meghatározásával.

A többdimenziós optimalizálási eljárások rendszerint az

$$x_{k+1} = x_k + \alpha s_k$$

alakban keresik az új közelítő megoldást, ahol s_k egy keresési irány, és α a lépésköz. A kvadratikus függvények optimalizálása során a következő észrevételeket tehetjük:

- i. A negatív gradiens (amelyik irányában a célfüggvény csökken) a reziduális vektor: $-\nabla q(x) = b - Ax = r$.
- ii. Adott keresési irány mentén nem kell adaptív módon meghatározni a lépésközt (mint általános nemlineáris minimalizálás esetén kellene), mert az optimális α közvetlenül megadható. A keresési irány mentén ott lesz a célfüggvény minimális, ahol az új reziduális vektor merőleges s_k -ra:

$$0 = \frac{d}{da} q(x_{k+1}) = \nabla q(x_{k+1})^T \frac{d}{da} x_{k+1} = (Ax_{k+1} - b)^T \left(\frac{d}{da} (x_k + \alpha s_k) \right) = -r_{k+1}^T s_k.$$

Az új reziduális vektort ki lehet fejezni a régvivel és a keresési iránnyal:

$$r_{k+1} = b - Ax_{k+1} = b - A(x_k + \alpha s_k) = (b - Ax_k) - \alpha As_k = r_k - \alpha As_k.$$

Behelyettesítve r_{k+1} -et, és megoldva az egyenletet α -ra azt kapjuk, hogy

$$\alpha = \frac{r_k^T s_k}{s_k^T A s_k}.$$

• Lagrange interpoláció

Tétel: Legyenek adottak az x_0, \dots, x_n ($n+1$ db) páronként különböző alappontok. Ekkor az $f(x_i)$, $i = 0, \dots, n$ függvényérékekhez egyértelműen létezik olyan legfeljebb n -edfokú interpoláló polinom, amely megegyezik a Lagrange interpolációs polinommal.

Bizonyítás: A definíció alapján az L_i polinomok mindegyike n -edfokú polinom (mert $n+1$ alappont van), amelyekre

$$L_i(x_j) = \begin{cases} 0 & \text{ha } i \neq j \\ 1 & \text{ha } i = j. \end{cases}$$

Ebből adódik, hogy $p_n(x) = \sum_{i=0}^n f(x_i)L_i(x)$ egy legfeljebb n -edfokú polinom, amelyik kielégíti a $p_n(x_j) = f(x_j)$ feltételt.

Az unicitás bizonyításához feltesszük, hogy p egy legfeljebb n -edfokú polinom, amelyre $p(x_j) = f(x_j)$, $j = 0, \dots, n$. Ekkor a $p_n - p$ polinom foka is $\leq n$ (mindkettő legfeljebb n -edfokú, ha kivonjuk öket egymásból, nagyobb nem lehet), és legalább $n+1$ darab páronként eltérő gyöke van: x_0, \dots, x_n (mert amikor kivontuk öket egymásból, mivel mindenkettő teljesíti, hogy az alappontokban felveszik a helyes y értéket, így pont ezekben a pontokban a különbségük 0). Ebből az következik, hogy $p_n - p$ osztható az $(x - x_0)(x - x_1) \dots (x - x_n)$ szorzattal. Mivel az osztó fokszáma $> n$, így ebből az adódik, hogy $p_n(x) - p(x)$ azonosan nulla, vagyis $p = p_n$.

KONJUGÁLT GRADIENS MÓDSZER / 2

Ezzel megkaptuk a szimmetrikus, pozitív definit mátrixú lineáris egyenletrendszer megoldására szolgáló konjugált gradiens módszert. Egy adott x_0 indulópontra legyen $s_0 = r_0 = b - Ax_0$, és iteráljuk $k = 1, 2, \dots$ értékekre az alábbi lépésekkel, amíg a megállási feltételek nem teljesülnek:

1. $\alpha_k = (r_k^T r_k) / (s_k^T A s_k)$ (a lépéshossz meghatározása)
2. $x_{k+1} = x_k + \alpha_k s_k$ (iterált közelítő megoldás)
3. $r_{k+1} = r_k - \alpha_k A s_k$ (az új reziduális vektor)
4. $\beta_{k+1} = (r_{k+1}^T r_{k+1}) / (r_k^T r_k)$ (segédváltozó)
5. $s_{k+1} = r_{k+1} + \beta_{k+1} s_k$ (az új keresési irány)

Vegyük észre, hogy az α értékét most kicsit más formában határoztuk meg ($r_k^T s_k$ helyett $r_k^T r_k$ áll). Érvényes viszont, hogy

$$r_k^T s_k = r_k^T (r_k + \beta_k s_{k-1}) = r_k^T r_k + \beta_k r_k^T s_{k-1} = r_k^T r_k,$$

mivel az r_k reziduális vektor merőleges az s_{k-1} keresési irányra.

A korábbi gradiensmódszerek egyszerűen a negatív gradienst követték minden iterációs lépésben, de felismerték, hogy ez a meredek falú enyhén lejtő völgy-színen függvények esetén szükségtelenül sok iterációs lépést követelt a völgy két oldalán való oda-vissza mozgással. A kisebb meredekséggel rendelkező irányban viszont lényegesen gyorsabban lehetett volna haladni. A konjugált gradiens módszer ezzel szemben a lépésekbeni megfelelő irányváltoztatással kikúszobölli ezt a hátrányt (innen a neve).

A megállási feltétel szokás szerint az, hogy a felhasználó előírja, hogy az utolsó néhány iterált közelítés eltérése és a lineáris egyenletrendszer két oldala különbsége normája ezekben a pontokban adott kis pozitív értékek alatt maradjanak.

A konjugált gradiens módszer nemlineáris optimalizálásra is alkalmas, ha minden iterációs lépésben az eredeti célfüggvény kvadratikus modelljére alkalmazzuk (az adott pontbeli függvényértékre, a gradiensre és a Hesse mátrixra vagy ezek közelítésére támaszkodva).

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- Numerikus integrálás

NUMERIKUS INTEGRÁLÁS, KVADRATÚRA-FORMULÁK

A kvadratúra a numerikus integrálás szinonimája, amikor az

$$\int_a^b f(x) dx = F(b) - F(a)$$

határozott integrál közelítése a feladat. Itt $F(x)$ az $f(x)$ integrálálandó függvény primitív függvénye. Ez utóbbi nem minden esetben áll rendelkezésre, sőt sokszor (mint például az $f(x) = e^x$ esetben) nem is elemi függvény, nem adható meg zárt alakban.

A határozott integrálokat szokás

$$\int_a^b f(x) dx \approx Q_n(f) = \sum_{i=1}^n w_i f(x_i)$$

alakban közelíteni, ahol $Q_n(f)$ -et kvadratúra-formulának nevezzük. Általában feltesszük, hogy $x_i \in [a, b]$ teljesül az x_i alappontokra, és ezek páronként különbözök. A w_i számokat súlyoknak hívjuk.

Az $\int_a^b f(x) dx$ integrál és a $Q_n(f)$ kvadratúra-formula homogén és additív leképezés, azaz érvényesek

$$\begin{aligned} \int_a^b f(x) + g(x) dx &= \int_a^b f(x) dx + \int_a^b g(x) dx, \\ Q_n(f+g) &= \sum_{i=1}^n w_i (f(x_i) + g(x_i)) = \sum_{i=1}^n w_i f(x_i) + \sum_{i=1}^n w_i g(x_i) = Q_n(f) + Q_n(g), \\ \int_a^b \alpha f(x) dx &= \alpha \int_a^b f(x) dx, \\ Q_n(\alpha f) &= \sum_{i=1}^n w_i \alpha f(x_i) = \alpha \sum_{i=1}^n w_i f(x_i) = \alpha Q_n(f). \end{aligned}$$

Az integrál fontos további tulajdonsága a határök szerinti additivitás:

$$\int_a^b f(x) dx = \int_a^{z_1} f(x) dx + \cdots + \int_{z_{m-1}}^{z_m} f(x) dx + \int_{z_m}^b f(x) dx,$$

és az, hogy ha az $f(x)$ folytonos függvény nem azonosan nulla, és jeltartó az $[a, b]$ intervallumon, akkor $\int_a^b f(x) dx \neq 0$.

NUMERIKUS INTEGRÁLÁS, KVADRATÚRA-FORMULÁK / 3

TÉTEL. A Q_n n alappontos kvadratúra-formula rendje legfeljebb $2n - 1$ lehet.

BIZONYÍTÁS. Tekintsük a $q(x) = (\omega_{n-1}(x))^2$ 2n-edfokú polinomot, ahol $\omega_{n-1}(x)$ az n alappontra felírt $(x - x_1)(x - x_2) \cdots (x - x_n)$ polinom. Erre

$$0 < \int_a^b q(x) dx = \int_a^b (\omega_{n-1}(x))^2 dx \neq Q_n(q) = \sum_{i=1}^n w_i (\omega_{n-1}(x_i))^2 = 0,$$

tehát erre a polinomra nem pontos a formula. \square

Amennyiben legalább 0 rendű formulát akarunk előállítani, akkor annak az $f(x) = 1$ függvényre pontosnak kell lennie, azaz a

$$w_1 + w_2 + \cdots + w_n = \int_a^b dx = b - a$$

feltétel adódik a súlyokra. A továbbiakban ezt minden feltételezzük.

Nagyszámú alappont esetén gondot jelenthet a kvadratúra-formulák öröklött hibája. Ha a pontos $f(x_i)$ függvényértékek helyett az $f^*(x_i)$ közelítő értékeket használjuk, akkor a velük adódó $Q_n^*(f) = \sum_{i=1}^n w_i f^*(x_i)$ öröklött hibáját:

$$\begin{aligned} |Q_n(f) - Q_n^*(f)| &= \left| \sum_{i=1}^n w_i f(x_i) - \sum_{i=1}^n w_i f^*(x_i) \right| = \left| \sum_{i=1}^n w_i (f(x_i) - f^*(x_i)) \right| \\ &\leq \sum_{i=1}^n |w_i| |(f(x_i) - f^*(x_i))| \leq \epsilon \sum_{i=1}^n |w_i|. \end{aligned}$$

Itt $\epsilon = \max_{1 \leq i \leq n} |(f(x_i) - f^*(x_i))|$ a függvények abszolút hibakorlátja. Mivel Q_n pontossági rendje legalább 0, és $\sum_{i=1}^n w_i = b - a$, ezért

$$\epsilon(b - a) \leq \epsilon \sum_{i=1}^n |w_i|,$$

és az egyenlőség pontosan akkor teljesül, ha minden w_i súly pozitív. Az ilyen tulajdonságú kvadratúra-formulákat pozitív kvadratúra-formuláknak hívjuk. Az öröklött hiba abszolút hibakorlátja tehát ilyen formulákra minimális. Ezek a korlátok elérnek abban az értelemben, hogy bizonyos esetekben a tényleges öröklött hiba megegyezik a megadottakkal.

NUMERIKUS INTEGRÁLÁS, KVADRATÚRA-FORMULÁK / 2

Érvényesek továbbá a következő állítások is az integrálokra:

1. Legyen $f(x)$ és $g(x)$ két, ahol $[a, b]$ intervallumon integrálható függvény. Ha $g(x)$ jeltartó $[a, b]$ -n, akkor

$$\int_a^b f(x)g(x) dx = \mu \int_a^b g(x) dx, \text{ ahol } \inf_{x \in [a, b]} f(x) \leq \mu \leq \sup_{x \in [a, b]} f(x).$$

2. Ha még azt is tudjuk, hogy $f(x)$ -nek megvan a Darboux-tulajdonsága (például mert folytonos) $[a, b]$ -n, akkor érvényes

$$\int_a^b f(x)g(x) dx = f(\gamma) \int_a^b g(x) dx$$

valamely $\gamma \in [a, b]$ -re.

A kvadratúra-formula képlethebibját az $R_n(f) = \int_a^b f(x) dx - Q_n(f)$ kifejezéssel definiáljuk. Akkor mondjuk, hogy egy kvadratúra-formula pontos $f(x)$ -re, ha $R_n(f) = 0$. Amennyiben az $f(x)$ függvény, vagy olyan tulajdonságai, mint a deriválhatóság, illetve a deriváltak korlátai ismertek, a kvadratúra-formula hibája jól becsültethető. Sokszor viszont az $f(x)$ függvénynek csak az értékei ismertek bizonyos helyeken. A kvadratúra-formulák pontosságának jellemzésére azoknak a polinomokon való képlethebját szokás vizsgálni.

Azt mondjuk, hogy a $Q_n(f)$ kvadratúra-formula (pontossági) rendje az r természetes szám, ha az pontos az $1, x, x^2, \dots, x^r$ hatványfüggvényekre (azaz $R_n(x^k) = 0$ minden $0 \leq k \leq r$ -re), de nem pontos x^{r+1} -re.

A rend meghatározása ekvivalens a

$$\begin{aligned} w_1 + w_2 + \cdots + w_n &= \int_a^b dx, \\ w_1 x_1 + w_2 x_2 + \cdots + w_n x_n &= \int_a^b x dx, \\ &\vdots \\ w_1 x_1^r + w_2 x_2^r + \cdots + w_n x_n^r &= \int_a^b x^r dx \end{aligned}$$

egyenletek megoldásával. Ha az alappontokat és a súlyokat ismeretlennek tekintjük, akkor ez egy $r + 1$ egyenletből álló algebrai-, de nem lineáris egyenletrendszer $2n$ darab ismeretlennel. Ha viszont rögzítjük az alappontokat, akkor a súlyokra már lineáris egyenletrendszerrel kapunk.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

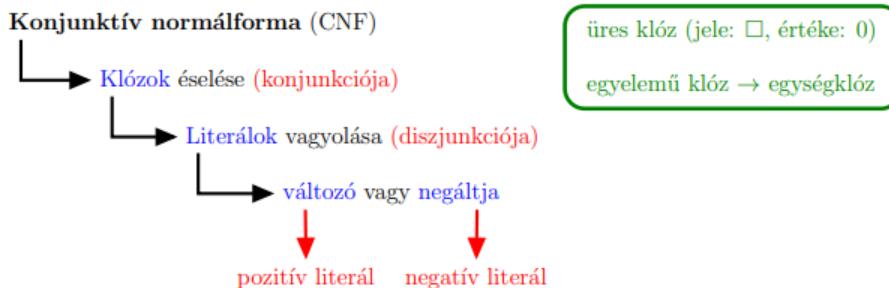
Logika és informatikai alkalmazásai / Logikai következtetési rendszerek

1. Tétel

- Normálformák az ítéletkalkulusban, teljes rendszerek

Ítéletkalkulus / nulladrendű/propozicionális logika ítéletváltozók: $p, q, r, p_1, p'_1, \dots$ változók értéke: 0 vagy 1 igazságértékek. formulák: változókból épülnek fel konnektívákkal (összekötő jelek), pl. $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ és zárójelekkel.

Precedencia sorrend $\neg > \wedge > \vee > \rightarrow > \leftrightarrow$



CNF-re hozás (A sorrend fontos!)

1. \rightarrow és \leftrightarrow eliminálása

- $F \rightarrow G \equiv \neg F \vee G$
- $F \leftrightarrow G \equiv (\neg F \vee G) \wedge (F \vee \neg G)$

2. Negálások bevitеле zárójelekbe

- $\neg\neg F \equiv F$
- $\neg(F \vee G) \equiv (\neg F \wedge \neg G)$
- $\neg(F \wedge G) \equiv (\neg F \vee \neg G)$

3. Disztributivitás (*Akkor van „baj”, ha két CNF-et vagyolunk*)

- $(F \wedge G) \vee H \equiv (F \vee H) \wedge (G \vee H)$
- $F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$
- $(C_1 \wedge \dots \wedge C_n) \vee (D_1 \wedge \dots \wedge D_m) \equiv \bigwedge_{i,j} (C_i \vee D_j), i = 1, \dots, n \text{ és } j = 1, \dots, m$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK
Teljes rendszerek

Teljes rendszer: olyan Boole függvény(ek) (logikai művelet(ek)), amelyekkel kifejezhető az összes többi is.

Az alábbiak például teljes rendszerek:

- $\{\vee, \wedge, \neg\}$, mert minden CNF-re lehet hozni
- $\{\wedge, \neg\}$, mert ha már egy ismert teljes rendszer összes jelét ki tudjuk velük fejezni, õ is teljes rendszert alkot.

$\{\vee, \wedge, \neg\}$ teljes:

$$\begin{aligned}x \wedge y &\equiv x \wedge y \\ \neg x &\equiv \neg x \\ x \vee y &\equiv \neg(\neg x \wedge \neg y)\end{aligned}$$

- $\{\vee, \neg\}$ is teljes, mert:

$\{\vee, \wedge, \neg\}$ teljes:

$$\begin{aligned}x \vee y &\equiv x \vee y \\ \neg x &\equiv \neg x \\ x \wedge y &\equiv \neg(\neg x \vee \neg y)\end{aligned}$$

- $\{\rightarrow, \neg\}$ is teljes, mert:

$\{\vee, \neg\}$ -et fejezzük ki:

$$\begin{aligned}\neg x &\equiv \neg x \\ x \vee y &\equiv (\neg x) \rightarrow y\end{aligned}$$

- $\{\rightarrow, \downarrow\}$ is teljes, mert:

$\{\rightarrow, \neg\}$ -et fejezzük ki:

$$\begin{aligned}x \rightarrow y &\equiv x \downarrow y \\ \neg x &\equiv x \rightarrow \downarrow\end{aligned}$$

- $\{\odot\}$ (NAND) is teljes, mert: $//x \odot y \equiv \neg(x \wedge y)$

$\{\wedge, \neg\}$ teljességből:

$$\begin{aligned}\neg x &\equiv \neg(x \wedge x) \equiv x \odot x \\ x \wedge y &\equiv \neg(\neg(x \wedge y)) \equiv (x \odot y) \odot (x \odot y)\end{aligned}$$

- $\{\oslash\}$ (NOR) is teljes, mert: $//x \oslash y \equiv \neg(x \vee y)$

$\{\vee, \neg\}$ teljességből:

$$\begin{aligned}\neg x &\equiv \neg(x \vee x) \equiv x \oslash x \\ x \vee y &\equiv \neg(\neg(x \vee y)) \equiv (x \oslash y) \oslash (x \oslash y)\end{aligned}$$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- Következtető módszerek: Hilbert-kalkulus és rezolúció

Rezolúció

A rezolúciós algoritmus:

- **Input:** egy CNF //ha nem ílyet kapunk, akkor előbb CNF-re hozzuk
- **Output:** Kielégíthetetlen-e?
- **Módszer:**
 - A CNF-et klözök halmazaként, a klözököt literálok halmazaként fogjuk fel
 - Listát vezetünk a klözökről. Egy klózt felvehetünk, ha:
 - * az input CNF egy klóza VAGY
 - * két, a listán már szereplő klóz *rezolvense*:
$$p \in C_1 \text{ és } \neg p \in C_2, \text{ akkor felvehetjük } C_1 \text{ és } C_2 \text{ (} p \text{ menti) rezolvensét:}$$
$$(C_1 - \{p\}) \cup (C_2 - \{\neg p\}).$$

Rezolúciós következtetés: $\{F \vee G, \neg F \vee H\} \vDash G \vee H$
 - Ha az üres klöz rákerül a listára, akkor **kielégíthetetlen**
 - Ha nem, és többféle klózt nem tudunk felvenni a listára, akkor **kielégíthető**

1. Feladat Igazoljuk rezolúcióval, hogy kielégíthetetlen:

$$(((p \rightarrow q) \wedge \neg q) \vee ((r \rightarrow \neg p) \wedge r)) \wedge s \wedge (s \rightarrow p)$$

Megoldás

CNF-re hozás:

1. Nyilak eliminálása:

$$(((\neg p \vee q) \wedge \neg q) \vee ((\neg r \vee \neg p) \wedge r)) \wedge s \wedge (\neg s \vee p)$$

2. Negáció bevitele: Ez most készen van.

3. Disztributivitás:

$$\begin{aligned} & (((\neg p \vee q) \wedge \neg q) \vee ((\neg r \vee \neg p) \wedge r)) \wedge s \wedge (\neg s \vee p) \\ & \equiv ((\neg p \vee q \vee \neg r \vee \neg p) \wedge (\neg p \vee q \vee r) \wedge (\neg q \vee \neg r \vee \neg p) \wedge (\neg q \vee r)) \wedge s \wedge (\neg s \vee p) \end{aligned}$$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Rezolúció:

- Vegyük fel a halmazt a klózokról:

$$\Sigma = \{\{\neg p, q, \neg r\}, \{\neg p, q, r\}, \{\neg p, \neg q, \neg r\}, \{\neg q, r\}, \{s\}, \{p, \neg s\}\}$$

Ha egy klózban egy változó ismétlődik, elég egyszer feltüntetnünk, hiszen többszörösen felvéve sem változtat a klóz értékén.

- Lista:

| | | |
|----------------------------------|--------------|--|
| 1. $\{\neg p, q, \neg r\}$ | $\in \Sigma$ | |
| 2. $\{\neg p, q, r\}$ | $\in \Sigma$ | |
| 3. $\{\neg p, q\}$ | Res(1, 2) | Még egyszer: az 1. klózban szerepel $\neg r$, a 2. klózban r , így rezolvensük az a klóz, amit úgy kaphunk, hogy 1-ből kiveszünk $\neg r$ -t, 2-ből kiveszük r -t, és minden egyéb literált, ami marad a két klóz valamelyikében, felveszük a rezolvensbe. Persze mindenekkel csak egyszer. |
| 4. $\{p, \neg s\}$ | $\in \Sigma$ | |
| 5. $\{q, \neg s\}$ | Res(3, 4) | |
| 6. $\{\neg q, r\}$ | $\in \Sigma$ | |
| 7. $\{\neg s, r\}$ | Res(5, 6) | |
| * | 8. $\{s\}$ | Egységláz! Heurisztika: amit tudunk, rezolválunk egységlázossal. |
| * | 9. $\{r\}$ | Res(7, 8) |
| 10. $\{p\}$ | Res(8, 4) | |
| 11. $\{q\}$ | Res(3, 10) | * Ha egy klóz valódi részhalmaza egy másik klóznak (mint a 9. a 7.-nek), akkor a bővebbet elhagyhatjuk. Vele később már nem érdemes rezolválnunk. |
| 12. $\{\neg p, \neg q, \neg r\}$ | $\in \Sigma$ | |
| 13. $\{\neg q, \neg r\}$ | Res(10, 12) | |
| 14. $\{\neg r\}$ | Res(11, 13) | |
| 15. \square | Res(9, 14) | |

Mivel $\square \in \text{Res}^*(\Sigma)$, így $\Sigma \vDash \perp$.

(Ahol $\text{Res}^*(\Sigma)$ a Σ -ból levezethető összes klóz halmaza.)

Ami **TILOS**: NEM rezolválhatunk egyszerre két literál mentén!

Jó ötletnek tűnhet, de nem az:

| | | | | | | | | | | | |
|----------------------------|--------------|---|---------------|--------------|--|-------------------------|--------------|--|--------------|-----------|--|
| 1. $\{\neg p, q, \neg r\}$ | $\in \Sigma$ | | | | | | | | | | |
| 2. $\{\neg p, \neg q, r\}$ | $\in \Sigma$ | Ezért: $(p \vee q) \wedge (\neg p \vee \neg q)$ -ból kapjuk: | | | | | | | | | |
| 3. $\{\neg p\}$ | Res(1, 2) | <table border="0"> <tbody> <tr> <td>1. $\{p, q\}$</td> <td>$\in \Sigma$</td> <td></td> </tr> <tr> <td>2. $\{\neg p, \neg q\}$</td> <td>$\in \Sigma$</td> <td></td> </tr> <tr> <td>3. \square</td> <td>Res(1, 2)</td> <td></td> </tr> </tbody> </table> | 1. $\{p, q\}$ | $\in \Sigma$ | | 2. $\{\neg p, \neg q\}$ | $\in \Sigma$ | | 3. \square | Res(1, 2) | |
| 1. $\{p, q\}$ | $\in \Sigma$ | | | | | | | | | | |
| 2. $\{\neg p, \neg q\}$ | $\in \Sigma$ | | | | | | | | | | |
| 3. \square | Res(1, 2) | | | | | | | | | | |
| | | Pedig ez a formula kielégíthető: pl: $p = 1$ és $q = 0$ értékkadással! | | | | | | | | | |

Felhasználva azt, hogy $\Sigma \models F$ pontosan akkor, ha $\Sigma \cup \{\neg F\} \vDash \perp$, a rezolúciós algoritmus következetetések igazolására is használható a következő módon:

- **Input:** formulák egy Σ halmaza és egy F formula.

- **Output:** Igaz-e, hogy $\Sigma \models F$?

- **Algoritmus:**

- CNF-re hozzuk Σ összes elemét és a $\neg F$ formulát is. Az összes kapott klóz halmazát jelölje Σ' .
- Hajtsunk végre Σ' -n rezolúciót. Ha $\square \in \text{Res}^*(\Sigma')$ akkor $\Sigma \models F$, különben $\Sigma \not\models F$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Hilbert rendszere

A formulákban most csak:

- változók
- \rightarrow
- \downarrow

szerepelhetnek.

Pl: $(p \rightarrow (p \rightarrow \downarrow)) \rightarrow q$

Minden formulát ilyen alakra lehet hozni, mivel $\{\rightarrow, \downarrow\}$ teljes rendszer

Hilbert rendszere:

Input: egy Σ formulahalmaz és egy \mathcal{F} célfórmula

Output: igaz-e, hogy $\Sigma \vdash \mathcal{F}$?

Lépések: Listát vezetünk formulákról. A listára felkerülhetnek:

- Σ elemei
- Axiómapéldányok ízlés szerint: a három közül kiválasztunk egyet, aztán F, G, H helyére tetszőleges formulákat írhatunk
- *Modus ponens:* ha F és $F \rightarrow G$ is megvan a listán, akkor felvehetjük G -t is
Leválasztási következtetés: $\{F, F \rightarrow G\} \vdash G$

Axiómák:

1. $(F \rightarrow (G \rightarrow H)) \rightarrow ((F \rightarrow G) \rightarrow (F \rightarrow H))$
2. $F \rightarrow (G \rightarrow F)$
3. $((F \rightarrow \downarrow) \rightarrow \downarrow) \rightarrow F$

Folytatjuk a formulák felvételét, amíg \mathcal{F} , a célfórmula a listán nem lesz (jele: $\Sigma \vdash \mathcal{F}$), amivel igazoltuk, hogy $\Sigma \vdash \mathcal{F}$.

Tétel: $\Sigma \vdash \mathcal{F} \Leftrightarrow \Sigma \models \mathcal{F}$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

1. Feladat Mutassuk meg, hogy $\vdash \downarrow \rightarrow p$. Az első lépés: $\text{Ax1}[F/\downarrow, G/(p \rightarrow \downarrow) \rightarrow \downarrow, H/p]$.

Megoldás

Σ üres, szóval csak axiómapéldányt és modus ponens fogunk használni.

Mindig nézzük meg, mi az, amit le akarunk vezetni:

- Σ -beli-e?
- axiómapéldány?
- $G \rightarrow F$ alakú-e, ahol F -et le tudjuk vezetni?
 - F -et levázzunk
 - $F \rightarrow (G \rightarrow F)$ // a 2. axiómába behelyettesítéssel
 - modus ponens ezeket: $G \rightarrow F$ // a 2. axióma pont erre lesz jó

1. $\text{Ax1}[F/\downarrow, G/(p \rightarrow \downarrow) \rightarrow \downarrow, H/p]$

$$\frac{(\downarrow \rightarrow (((p \rightarrow \downarrow) \rightarrow \downarrow) \rightarrow p)) \rightarrow ((\downarrow \rightarrow ((p \rightarrow \downarrow) \rightarrow \downarrow)) \rightarrow (\downarrow \rightarrow p))}{\text{előzőr ezzel vágunk} \quad \downarrow \quad \text{majd ezzel} \quad \text{célformula}}$$

csak kiülő jel mentén vágunk!

2. A formula közepre most épp egy axiómapéldány. Levezethetjük, később jól jön, de középről nem vágunk!

$$\downarrow \rightarrow ((p \rightarrow \downarrow) \rightarrow \downarrow)$$

$\text{Ax2}[F/\downarrow, G/p \rightarrow \downarrow]$

$$3. ((p \rightarrow \downarrow) \rightarrow \downarrow) \rightarrow p$$

$\text{Ax3}[F/p]$

// Ez az 1. elejének jobb oldala.

$$4. (((p \rightarrow \downarrow) \rightarrow \downarrow) \rightarrow p) \rightarrow (\downarrow \rightarrow (((p \rightarrow \downarrow) \rightarrow \downarrow) \rightarrow p)) \quad \text{Ax2}[F/(3), G/\downarrow]$$

Megvan a 3. formula, de kéne elő egy \downarrow , hogy pont az 1. eleje legyen, amivel vágni szeretnénk. Erre jó az Ax2.

$$5. \downarrow \rightarrow (((p \rightarrow \downarrow) \rightarrow \downarrow) \rightarrow p)$$

$\text{MP}(3, 4)$

Az előző elejét leválaszthatjuk (az volt a 3.), így pont azt a részformulát kapjuk, amit szerettünk volna legyártani.

$$6. (\downarrow \rightarrow ((p \rightarrow \downarrow) \rightarrow \downarrow)) \rightarrow (\downarrow \rightarrow p)$$

$\text{MP}(1, 5)$

$$7. \downarrow \rightarrow p$$

$\text{MP}(2, 6)$

Tehát $\vdash \downarrow \rightarrow p$

Sokszor segít a dedukciós téTEL:

$$\Sigma \vdash F \rightarrow G \Leftrightarrow \Sigma \cup \{F\} \vdash G$$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

2. Tétel

- Normálformák a predikátumkalkulusban

Mi micsoda a formulákban?

- \forall : univerzális kvantor („bármely”)
- \exists : egzisztenciális kvantor („van olyan”)
- $x, y : \text{változók}$ (valamelyen alaphalmazból / univerzumból. Ennek jele általában A lesz.)
- $p(x, y) : \text{predikátumjel}$ (ez épp bináris) és a „jelentése” objektumból igazságértéket képez
- lesz még függvényjel is: $f(x)$, aminek a „jelentése” objektumból objektumot képez

Ennek a kiértékeléséhez szükségünk van egy struktúrára!

Az elsőrendű logika szemantikája:

Struktúra: $\mathcal{A} = (A, I, \varphi)$ hármas, ahol:

- A az alaphalmaz, nem üres
- I az interpretációs függvény:
 - f/n -hez egy $A^n \rightarrow A$ függvényt rendel
 - p/n -hez egy $A^n \rightarrow \{0, 1\}$ predikátumot rendel
- φ minden változóhoz egy A -beli elemet rendel:
 - változóértékadás: $X \rightarrow A$
 - $\forall x, \exists x$ -el felülírjuk

Elsőrendű logika két szintaktikus kategóriája:

Termek: (Objektumértékek lesz.)

- az összes változó; (Rögzített halmaz $X = \{x, y, z, x_1, x_2, x_{127}, y_4, y', z'', \dots\}$.)
- ha f/n egy n -változós függvényjel, t_1, \dots, t_n pedig termek, akkor $f(t_1, \dots, t_n)$ is term
- másféle term nincs.

Formulák: (Igazságértékűek.)

- ha p/n egy n -változós predikátumjel, t_1, \dots, t_n pedig termek, akkor $p(t_1, \dots, t_n)$ egy atomi formula;
- ha F, G formulák, $x \in X$ pedig változó, akkor formula még $(F \vee G), (F \wedge G), (F \rightarrow G), (F \leftrightarrow G), (\neg F), (\forall x F)$ és $(\exists x F)$ is;
- \downarrow és \uparrow is formulák.
- másféle formula nincs.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Egy elsőrendű logikai formula például a következő:

$$\begin{aligned}
 F &= \forall x (x = c \vee \exists y (f(y) = x)) \\
 &= (x, c) \quad = (f(y), x) \\
 &= (x, c())
 \end{aligned}$$

kötő
konstans

Mivel a formulában minden változó előfordulás kötött, ilyenkor mondatnak nevezzük.
Ekkor nem kell megadunk φ -t.

Viszont ez így önmagában nem jelent semmit. Ahhoz, hogy ki tudjuk értékelni, szükségünk van egy struktúrára.

Ez a struktúra lehet például: $\mathcal{A} = (\mathbb{N}_0, I, \varphi)$, ahol:

- $I(c) = 0$
- $I(f)(n) = n + 1$
- $I(=)$: minden az egyenlőség

Ha ebben a struktúrában értékeljük ki az F formulát, akkor a jelentése a következő:

$\mathcal{A}(F) = „minden $a \in \mathbb{N}_0$ -ra igaz, hogy $a = 0$ vagy van olyan $b \in \mathbb{N}_0$, amire: $b + 1 = a$ ”$

Ez épp igaz.

A gond: automatikusan kéne ezt csinálni

A nagy gond: Nincs olyan algoritmus, ami KIÉRTÉKELNE egy formulát

Olyan van, ami megmondja egy formuláról, hogy kielégíthetetlen (minden struktúrában hamis.)

Normálformák elsőrendben

Egy formula **kiigazított**, ha:

- Különböző kvantorok különböző változókat kötnek
- Nincs olyan változó, amely szabadon és kötötten is előfordul.

Minden formulát kiigazíthatunk, pl. átnevezéssel. (Ekvivalencia-tartó módon.)

1. Feladat Igazitsuk ki: $F = \forall x ((\exists y p(x, y)) \rightarrow q(x)) \wedge \exists y \forall x p(x, y) \wedge \neg q(x)$

Megoldás

Keressük ki a kötött változókat:

$$F = \forall x ((\exists y p(x, y)) \rightarrow q(x)) \wedge \exists y \forall x p(x, y) \wedge \neg q(x)$$

Indexeljük le a **kötött** változókat! (Válasszunk minden kvantor mellé egy tetszőleges indexet: pl. $x_1, y_{007}, z_{másolás}$, majd írjuk ezt be a kvantor mellett a változó helyére, és oda, amelyik változót az adott kvantor köti.)

$$F = \forall x_1 ((\exists y_2 p(x_1, y_2)) \rightarrow q(x_1)) \wedge \exists y_3 \forall x_4 p(x_4, y_3) \wedge \neg q(x)$$

Itt most épp minden kvantorhoz azt a sorszámot választottuk, ahányadjára épp találkoztunk vele a formula „olvasása” során. (Feltételezés: jobbra olvasunk!)

A szabad változót NE bántsonk!

Észrevehetjük: így már minden változóról rögtön el tudjuk dönteneni, kvantor köti-e! Mivel a szabad változókhoz nem nyúltunk, viszont a kvantorok által kötött változókhoz indexet rendeltünk, így a szabad változók nevei biztosan nem fognak kötötten se előfordulni. Ez nekünk jó: a formula kiigazított lesz.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Egy formula prenex alakú, ha „a kvantorok az elején vannak”:

Átalakítás (előadás szerint), ha már ki van igazítva:

- $\rightarrow, \leftrightarrow$: még előtte elimináljuk!
(Célszerű kiigazítás előtt!)
 - $(\exists x F) \vee G \Rightarrow \exists x(F \vee G)$
 - $(\forall x F) \vee G \Rightarrow \forall x(F \vee G)$
 - $(\exists x F) \wedge G \Rightarrow \exists x(F \wedge G)$
 - $(\forall x F) \wedge G \Rightarrow \forall x(F \wedge G)$
 - $\neg \exists x F \Rightarrow \forall x \neg F$
 - $\neg \forall x F \Rightarrow \exists x \neg F$

Előtte igazítsuk ki a formulát, mivel ezek a szabályok csak akkor igazak, ha G -ben NINCS x !

2. Feladat Folytassuk az előző formula átalakítását!

Megoldás

Az \rightarrow és \leftrightarrow eliminálása után:

$$F = \forall x_1 ((\neg \exists y_2 p(x_1, y_2)) \vee q(x_1)) \wedge \exists y_3 \forall x_4 p(x_4, y_3) \wedge \neg q(x)$$

6. szabály $\underbrace{\forall y_2 \neg p(x_1, y_2)}_{\text{3. szabály}}$ 4-5. szabály $\underbrace{\exists y_3 \forall x_4 (p(x_4, y_3) \wedge \neg q(x))}_{\text{4-5. szabály}}$
3. szabály $\underbrace{\forall x_1 \forall y_2 (\neg p(x_1, y_2) \vee q(x_1))}_{\text{4-5. szabály}}$ 4-5. szabály
 $\forall x_1 \forall y_2 \exists y_3 \forall x_4 ((\neg p(x_1, y_2) \vee q(x_1)) \wedge p(x_4, y_3) \wedge \neg q(x))$

Ugyanezt kapjuk, ha:

- az eredeti sorrendben kvantáljuk a változókat (azaz felvesszük a kvantorokat és a hozzájuk tartozó változókat az eredeti formula sorrendjének megfelelően a formulánk elejére!)
 - a kvantor fordul (\forall -ból \exists -be és fordítva), ha páratlan sok \neg belsőjében van (Fontos: mire ide eljutunk, már NEM LEHET a formulánkban \rightarrow és \leftrightarrow)
 - a formula magját (azaz, ami a kvantorok után szerepel) úgy kapjuk, hogy az eredeti formulából kitöröljük a kvantorokat (pl $(\exists x p(x)) \vee q(y)$ -ból csak $p(x) \vee q(y)$ -t kell, leírjuk).

Egy formula Skolem alakú, ha prenex és csak univerzális kvantor (\forall) van benne.

Skolem alakra hozás:

- prenex alakra hozzuk
 - az összes $\exists x$ változóra:
 - töröljük a $\exists x$ -et
 - a magbeli x -ek helyére mindenhol $f(x_1, \dots, x_n)$ kerül, ahol f új függvényel és x_1, \dots, x_n pedig az x előtt deklarált \forall változók

Ha **zárt Skolem** alak kell, akkor még: a szabad változók helyére **új konstansjelek** kerülnek.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- Egyesítési algoritmus

Helyettesítés, egyesítés

Helyettesítés és egyesítés:

Ha $s = [x_1/t_1][x_2/t_2] \dots [x_n/t_n]$ egy helyettesítés, C egy literálhalmaz, akkor $C \cdot s$ -t így kapjuk, hogy C -ben az összes x_i -et t_i -re, aztán az összes x_j -et t_j -re, stb. cseréljük.

A sorrend számít:

$$\boxed{C = \{p(x, y), p(f(y), z)\} [x/f(y)][y/c] = \{p(f(c), c), p(f(c), z)\}}$$

$$C = \{p(x, y), p(f(y), z)\} [y/c][x/f(y)] = \{p(f(y), c), p(f(c), z)\}$$

Egy s helyettesítés akkor egyesíti C -t, ha $|C \cdot s| \leq 1$ // mindenkinél ugyanaz a képe

Egyesítési algoritmus:

Input: literálok C halmaza

Output: Egyesíthető-e C ? (Változók helyére termék írásával lehetnek-e egyformák C elemei?)
Ha igen, hogyan?

$$\boxed{\begin{aligned} \text{Pl: } C &= \{p(x, f(x)), p(a, z)\} \\ \text{Egy egyesítője: } &[x/a][z/f(a)] \\ \text{Eredmény: } &\{p(a, f(a))\} \end{aligned}}$$

Algoritmus:

- $s := []$ (üres helyettesítés)
- amíg $|C| > 1$:
 - válasszunk ki két literált C -ból (ha kettő nem egyesíthető, akkor ha több literál van sem lesz az)
 - keressük az első eltérést
 - ha itt az egyikben egy x változó, a másikban egy t term kezdődik, amiben nincs x :
- $s := s[x/t], C := C[x/t],$
- különben NEM egyesíthető
- Ha sikerült, return s

1. Feladat Egyesítsük $C = \{p(x, f(x)), p(a, z)\}$

Megoldás

Az első eltérés: $\{p(x, f(x)), p(a, z)\}$

- $s = [x/a]$
- $C = \{p(a, f(a)), p(a, z)\}$

Az $[x/a]$ jelentése: a formulahalmazban minden x helyére írunk a -t

A második eltérés az új C -ben: $\{p(a, f(a)), p(a, z)\}$

- $s = [x/a][z/f(a)]$
- $C = \{p(a, f(a))\}$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- Következtető módszerek: Alap rezolúció, elsőrendű rezolúció

Alap rezolúció

Alap rezolúció:

- **Input:** elsőrendű logikai formulák halmaza
- **Output:** Kielégíthetetlen-e az input?
- **Algoritmus:**
 - a formulákat zárt Skolem alakra, a magokat CNF-re hozzuk
 - a klózokat egy Σ halmazba gyűjtjük
 - Listát vezetünk a klózokról. A listára felvehetjük:
 - + Σ klózainak **alap párányait**
 - A változók helyére **alaptermeket (T_0)** helyettesítünk
 - Konstansokból és függvényelekből képezhetjük
 - + két korábbi, már a listán lévő klóz rezolvensét
 - ha kijön az üres klóz, akkor az input kielégíthetetlen

1. Feladat Döntsük el alaprezolúcióval, hogy kielégíthetetlen-e:

$$F = \exists x \forall y (p(x, y) \leftrightarrow \neg p(y, y))$$

Megoldás

A Skolem alakra hozáshoz előbb hozzuk CNF-re a magot:

$$F = \exists x \forall y ((\neg(p(x, y) \vee \neg p(y, y))) \wedge ((p(x, y) \vee p(y, y))))$$

A $\exists x$ eliminálásához egy új konstanst (pl. a) kell bevezetni:

$$F = \forall y ((\neg(p(a, y) \vee \neg p(y, y))) \wedge ((p(a, y) \vee p(y, y))))$$

Gyűjtsük össze a klózokat:

$$\Sigma = \{(\{\neg p(a, y), \neg p(y, y)\}, \{p(a, y), p(y, y)\})\}$$

Gyűjtsük össze az alaptermeket:

$$T_0 = \{a\}$$

Rezolúció:

1. $\{\neg p(a, a)\} \quad // 1. \text{ klóz } [y/a] \in E'(\Sigma)$
2. $\{p(a, a)\} \quad // 2. \text{ klóz } [y/a] \in E'(\Sigma)$
3. $\square \Rightarrow F \vdash \perp \quad \text{Res}(1, 2)$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

HELP Box

Az alaptermek (T_0) legyártása:

- Kikeressük a klózokban szereplő az **összes konstanst** (a, b, c, d, \dots) és felvesszük T_0 -ba. Ha nincs a formulában konstans, választunk egy tetszőlegeset, és felvesszük azt.
- Kikeressük a klózokban szereplő **összes nem-konstans függvényjelet** (f, g, h, i, \dots) és az argumentumokba tetszőleges, már T_0 -ban szereplő termeket frunk. (Ha nincs függvényjel a klózokban, nem baj.)
- Hint: ha a klózokban van legalább egy függvényjel, az alaptermek halmaza végtelen. Ne próbáljuk meg felsorolni az összeset!

Klózok alappéldányainak gyártása:

- Választunk egy tetszőleges klózt
- A klózban szereplő minden változó helyére tetszőleges T_0 -beli alaptermet helyettesítünk (az egyformákba ugyanazt)

How to win:

- ha hamar meg akarjuk kapni az üres klózt, célszerű megkeresni azokat a literálokat, amelyeknek pozitív, vagy negatív alakjára valamelyen „korlátozásunk” van, pl: van benne valahol függvényjel
- próbálunk rá a legegyszerűbb helyettesítéssel, ami eleget tesz ezeknek
- Pl 3. példában
 - $\{\{p(x, f(y))\}, \{\neg p(f(x), z), r(x, g(z))\}, \{\neg r(f(y), g(y)), \neg p(y, y)\}\}$
 - $r(f(\dots), g(\dots))$ alak kell mindenképp (2. és 3. klóz)
 - de akkor a 3. klózban $p(\dots, f(\dots))$ alakot kell legyártanunk (1. klóz miatt)
 - válasszuk a legegyszerűbbet: a 3. klózban y helyére $f(c)$ -t (f -et muszáj, egyváltozós, a legegyszerűbb alakja ez)
 - el is jutottunk a célravezető megoldásunkhoz

Elsőrendű rezolúció:

Input: Σ formulahalmaz

Output: Kielégíthetetlen-e Σ ?

Algoritmus:

- a formulákat zárt Skolem-re, a magokat CNF-re hozzuk
- a klózokat egy Σ' formulahalmazba gyűjtjük
- Listát vezetünk a klózokról. A listára felvehetjük:
 - Σ' halmaz elemeit
 - két korábbi, már a listán lévő klóz (C_1 és C_2) elsőrendű rezolvensét

1. A változókat átnevezzük, hogy C_1 -ben és C_2 -ben ne legyen két egyforma
2. C_1 -ből egy vagy több pozitív literált (pl $\{l_1, \dots, l_n\}$, ahol $n \geq 1$), C_2 -ből egy vagy több negatív literált (pl $\{\neg k_1, \dots, \neg k_m\}$, ahol $m \geq 1$), választunk, ahol a predikátumnak egyformának kell lennie!
3. Futtatjuk az egyesítési algoritmust a kiválasztott literálok előjel nélküli változataira.
4. Ha az egyesítés sikeres, felvesszük:

$$((C_1 - \{l_1, \dots, l_n\}) \cup (C_2 - \{\neg k_1, \dots, \neg k_m\})) \cdot s$$

ahol s az egyesítési algoritmus által visszaadott egyesítő

- ha kijön az üres klóz, akkor Σ kielégíthetetlen

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

1. Feladat Döntsük el elsőrendű rezolúcióval, hogy kielégíthetetlen-e

$$F = \forall x \forall y (p(x) \wedge (p(f(y)) \rightarrow r(y)) \wedge \neg r(g(x, y)))$$

Megoldás

A formula már Skolemben van. Hozzuk CNF-re a magot:

$$\forall x \forall y (p(x) \wedge (\neg p(f(y)) \vee r(y)) \wedge \neg r(g(x, y)))$$

Gyűjtsük össze a klózokat:

$$\Sigma' = \{\{p(x)\}, \{\neg p(f(y)), r(y)\}, \{\neg r(g(x, y))\}\}$$

Rezolúció: (célszerű azzal kezdeni, hogy felvesszük Σ' összes elemét)

1. $\{p(x)\} \quad \in \Sigma'$
2. $\{\neg p(f(y)), r(y)\} \quad \in \Sigma'$
3. $\{\neg r(g(x, y))\} \quad \in \Sigma'$
4. //próba: 1. klóz és 2. klóz:

- átnevezés: nincs változónév ütközés, nem kell átnevezni
- kiválasztás: Egyforma predikátumot választunk: 1. klóz teljesen, 2. klóz 1. literál
- $C = \{p(x), p(f(y))\}$
- egyesítő: $s = [x/f(y)]$
- megmarad: $\{p(x)\} - \{p(x)\} \cup \{\neg p(f(y)), r(y)\} - \{\neg p(f(y))\} = \{r(y)\}$
- rezolvens: $\{r(y)\} \cdot s = \{r(y)\} \quad \text{Res}(1, 2)$

5. //próba: 4 – 3 klózok:

- átnevezés: mindkettőben van y ! Nevezzük át a 4. klózban y -t. $\{r(y)\} \cdot [y/z] = \{r(z)\}$
- kiválasztás: Mindkét klózból az egészet választjuk
- $C = \{r(z), r(g(x, y))\}$
- egyesítő: $s = [z/g(x, y)]$
- megmarad: $\{r(y)\} - \{r(y)\} \cup \{\neg r(g(x, y))\} - \{\neg r(g(x, y))\} = \emptyset$
- rezolvens: $\square \Rightarrow F \models \downarrow \quad \text{Res}(3, 4)$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Mesterséges Intelligencia I.

1. Tétel

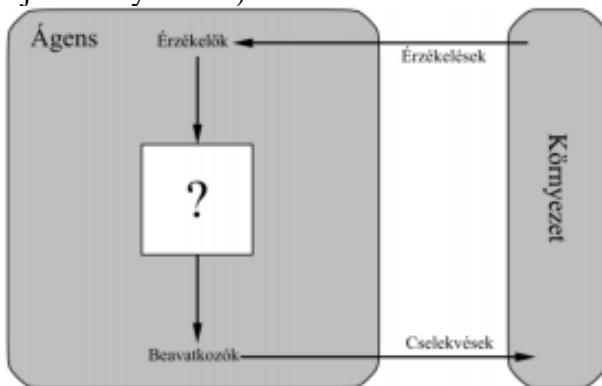
- Keresési feladat: feladatrepräsentáció, vak keresés, informált keresés, heurisztikák

Feladatrepräsentáció: A megoldandó probléma egy, az adott problémakörrel kapcsolatos “világot”, “környezetet” határoz meg az ágens körül. Ez lesz a feladatkörnyezet, amely minden pillanatban valamelyen állapotban van.

A feladatkörnyezet jellemzői:

- Teljesen vagy részlegesen megfigyelhető: adott absztraktiós szinten.
- Determinisztikus vagy sztochasztikus: Determinisztikus akkor, ha az aktuális állapot és bármely cselekvés együtt egyértelműen meghatározza a következő állapotot. Egyébként sztochasztikus.
- Epizodikus vagy sorozatszerű: Egymástól független epizódok (érzékelés-akció), vagy folyamatos feladat.
- Statikus vagy dinamikus: Akkor dinamikus, ha a környezet változik, míg az ágens tevékenykedik. Azaz bármely komponens (állapot, teljesítmény, cselekvés hatása, stb.) függhet az időtől.
- Diszkrét vagy folytonos
- Egy- vagy többágenses, stb.

A feladatkörnyezet négy komponense a teljesítmény mértéke, a környezet, a beavatkozók és az érzékelők. A feladatkörnyezetben cselekvő és a feladatot megoldani hivatott gépet ágensnek nevezzük. Az ágens az érzékelői segítségével érzékeli a környezetet és beavatkozó szervei segítségével megváltoztatja azt. Az ágens célja, hogy egy függvényt keressen (ágensfüggvény), illetve az azt megvalósító programot (ágensprogram), ami valamely megadott módon optimális (teljesítménymérték).



Ágenstípusok:

- Reflex ágensek: A cselekvés csak a környezetről rendelkezésre álló információktól (memória, világmodell, aktuális érzékelés) függ.
- Egyszerű reflex ágens: nincs belső állapot, a cselekvés csak az aktuális érzékeléstől függ.
- Célorientált és hasznosságorientált ágensek: a memóriát és a világmodellt felhasználják arra, hogy megjósolják a saját cselekvéseik következményeit, és olyan cselekvést választanak, ami valamely előre adott célhoz vezet, vagy valamely hasznossági függvényt optimalizál.
- Tanuló ágensek: bármelyik fenti típus lehet (reflex stb.), lényeg, hogy van visszacsatolás az értékelő függvénnyel, aminek a hatására az ágens programja módosul.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Az általunk elképzelt eredményt nevezzük célállapotnak. Ilyen megközelítésben a feladatot akkor tekinthetjük megoldottnak, ha a pillanatnyi állapot és a célállapot megegyezik, vagy legalábbis nincs közöttük számunkra fontos különbség. A feladatmegoldás pedig azt jelenti, hogy keressük azt a műveletsorozatot, amely segítségével a kezdőállapotból a végállapotba juthatunk. Ahhoz azonban, hogy egy feladatot jól és hatékonyan tudunk megoldani, fontos, hogy magát a feladatot, a feladathoz tartozó "világot" jól és pontosan tudjuk leírni. Ezt nevezzük a feladat specifikálásának. A hagyományos programozási feladatok megoldása is a feladatspecifikációval kezdődik. Ha szabályosan járunk el, akkor először megadjuk a feladat állapotterét.

Az állapottér megadásánál a feladatkörnyezet tökélesen modellezhető a következőkkel:

- lehetséges állapotok halmaza
- lehetséges kezdőállapotok halmaza
- lehetséges cselekvések halmaza, és egy állapotátmenet függvény, amely minden állapothoz hozzárendel egy (cselekvés,állapot) típusú, rendezett párokból álló halmazt
- állapotátmenet költségfüggvénye, amely minden lehetséges állapot-cselekvés-állapot hármashoz egy $c(x,a,y)$ valós nemnegatív költségértéket rendel
- célállapotok halmaza (lehetséges állapotok részhalmaza)

A fenti modell egy súlyozott gráfot definiál, ahol a csúcsok az állapotok, az élek cselekvések, a súlyok pedig a költségek. Ez a gráf az állapottér. A továbbiakban feltesszük, hogy az állapotok számosága véges vagy megszámlálható. Egy állapotnak legfeljebb véges számú szomszédja lehet. Úton, azaz a feladat megoldásához vezető lépéseken, állapotok cselekvésekkel összekötött sorozatát értjük (pl. $x_1; a_1; x_2; a_2; : : : ; x_n$, melynek költsége Ebben a feladatkörnyezetben ha az ágens ismeri a világ modelljét, akkor nem is kell neki szenzor! Az állapotteret reprezentáló gráf sokszor nagy méretű, esetleg végtelen, ezért vagy nem célszerű, vagy nem is lehet explicit módon fölrajzolni, hanem csak implicit módon megadni. A gráf azonban roppant szemléletes eszköz a megoldó algoritmusok létrehozásában. Egy általános gráfban sokszor nehéz megtalálni a megoldást, egy fa gráfban azonban általában jóval könnyebb. Ezért a reprezentációs gráfot gyakran fává alakítják.

Vak keresés (nem informált keresés):

Informálatlan keresés definíciója: ezen stratégiáknak semmilyen információjuk nincs az állapotokról a probléma definíciójában megadott információin kívül. Működéstük során más nem tehetnek, mint a következő állapotok generálása és a célállapot megkülönböztetése a nem célállapottól.

Fa keresés: egy FIFO sor, mellyel biztosítja, hogy a korábban meglátogatott csomópontokat az algoritmus korábban fejti ki. Azaz a Fa-keresés szélességi keresést eredményez, mivel az összes újonnan generált követőt a sor végére teszi, azaz a sekelyebb csomópontok korábban kerülnek kifejtésre, mint a mélyebben fekvők.

Szélességi keresés jellemzése:

teljesség: Ha a legsekelyebb célcsomópont valamilyen véges d mélységben fekszik, a szélességi keresés eljut hozzá az összes nála sekelyebben fekvő csomópontot kifejtve.

optimalitás: A szélességi keresés optimális, ha az útköltség a csomópont mélységének nem csökkenő függvénye.

komplexitás: Az exponenciális komplexitású keresési problémák közül csak a legkisebb problémapéldányok oldhatók meg.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Egyenletes költségű keresés: minden a legkisebb költségű n csomópontot fejti ki először, nem pedig a legkisebb mélységű csomópontot.

teljesség: csak úgy garantálható, hogy minden lépés költsége egy kis pozitív ε konstansnál nagyobb vagy azzal egyenlő

optimalitás: megegyezik a teljesség feltételével, ami azt jelenti, hogy egy út költsége az út mentén minden növekszik. Így az első kifejtésre kiválasztott célcsomópont egyben az optimális megoldás is.

komplexitás: legrosszabb esetben $O(b^{1+[C^*/\varepsilon]})$

Mélységi keresés: minden a keresési fa aktuális peremében a legmélyebben fekvő csomópontot fejti ki. Kifejtésüköt követően kikerülnek a peremből és a keresés „visszalép” aholhoz a következő legmélyebben fekvő csomóponthoz, amelynek vannak még ki nem fejtett követői.

teljesség: igaz, ha a fa véges mélységű

optimalitás: nem optimális

komplexitás: $O(b^m)$, ahol b az elágazási tényező és m a csomópontok maximális mélysége

Iteratívan mélyülő keresés fában: egy általános stratégia, amit sokszor a mélységi kereséssel együtt alkalmaznak a legjobb mélységekről megtalálására. Az algoritmus képes erre, mert fokozatosan növeli a mélységekről, amíg a célt meg nem találja. Amikor az megvan, a mélységekről elérte a d -t, a legkésőbb fekvő célcsomópont mélységét.

teljesség: teljes, ha elágazási tényezője véges

optimalitás: optimális, ha az útköltség a csomópontok mélységének nem csökkenő függvénye

komplexitás: $O(bd)$, ahol b az elágazási tényező és d a mélység

Heurisztikák előállításának módszerei:

Relaxálás: a probléma egy, egyszerűbb változatának a költségének a megadása. Az így kapott költséget felhasználhatjuk heurisztikának az eredeti feladat megoldásához, mivel ez a költség kisebb lesz, mint az eredeti feladat költsége. Relaxált probléma optimális megoldásának költsége a $h()$ egy **elfogadható** heurisztika az eredeti problémára. relaxáció = a feltételek elhagyása.

Relaxált probléma optimális költsége minden kisebb, vagy egyenlő mint az eredeti. Ezt egy elfogadható heurisztikának tekintjük az eredeti problémára. Mivel a számított heurisztika a relaxált problémára egy pontos költség, teljesítenie kell a háromszög egyenlőtlenséget is, és ebből kifolyólag **konzisztsz** is.

Automatizált relaxálás: ha a feltételek formális nyelven is adottak, akkor a relaxált problémákat automatikusan is előállíthatjuk.

Heurisztikák kombinálása: $h(n) = \max \{h_1(n), \dots, h_m(n)\}$

Az így megkonstruált összetett heurisztikus függvény minden azt a függvényt használja, amelyik az adott csúcspontra a legfontosabb. Mivel az alkotóelemként felhasznált heurisztikus függvények minden elfogadhatóak, ezért h is elfogadható. Továbbá h konzisztsz és dominálja az összes, benne alkotóelemként felhasznált heurisztikus függvényt.

Mintaadatbázisok: tároljuk az egyes részprobléma esetekhez tartozó pontos megoldási költségeket.

Független részproblémák: a költségek összeadhatóak.

Elfogadhatóság: kifejtve a relaxálásnál

Konzisztsz: kifejtve a relaxálásnál

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Informált keresés: amely a probléma definícióján túlmenően problémáspecifikus tudást is felhasznál, azaz hogyan képes hatékonyabban megtalálni a megoldást.

A* algoritmus és tulajdonságai: A csomópontokat úgy értékeli ki, hogy összekombinálja a $g(n)$ értéket – az aktuális csomópontig megtett út költsége – és a $h(n)$ értéket – vagyis az adott csomóponttól a célhoz vezető út költségének becslójét. Igy kapjuk a $f(n) = g(n) + h(n)$ összefüggést. $f(n)$ a legolcsóbb, az n csomópontron át vezető megoldás becsült költsége teljesség:

optimalitás: a Gráf-keresést használó A* algoritmus optimális, ha $h(n)$ konzisztens, azaz az $f(n)$ akármilyen út mentén nem csökken

komplexitás:

optimális hatékonyúság: Azt jelenti, hogy egyetlen más optimális algoritmus sem fejt ki garantáltan kevesebb csomópontot, mint az A*. Ez azért van, mert az összes olyan algoritmus, amelyik nem fejti ki az összes csomópontot, melyre $f(n) < C^*$, kockázatja az optimális megoldás elkerülését.

Memóriakorlátozott A*: Mivel minden esetben korlátos a memória, amit felhasználhatunk, ezért az A* algoritmust is erre kellett módosítani. Így a módosult algoritmus működése:

- amíg van memória \rightarrow A*
- ha elfogy, töröljük a legrosszabb levelet

Ha betlik a memória az éppen tárolt csomópontok halmazából kitörli azt, amely a legrosszabb költségbecsléssel rendelkezik, tehát a legkevésbé igéretes csomópontot egyszerűen kitörli, ezáltal memóriát szabadít fel. Azt a költségbecslést, ami az érintett csomópontban volt propagálja felfelé a szülőknek, a szülők költségét módosítja úgy, hogy tükrözze azt a tudást, amit közben megszerzett az adott csomópontok kiterjesztésével.

- Kétszemélyes zéró összegű játékok: minimax, alfa-béta eljárás

Többágenses környezeteket vezetünk be, ahol minden ágensnek számolnia kell a többi ágens cselekvésével, mivel azok hatással vannak rá is (pl. sakk, reversi).

Az MI-ben a játékok általában igen specializáltak – amit a játékelméleti szakemberek determinisztikus, váltott lépésű kétszemélyes, **zérusösszegű** teljes információjú játékoknak neveznek. Vagyis:

- Egy determinisztikus, megfigyelhető világban az ágensek cselekvései váltják egymást
- A játék végén a hasznosságértékeik minden azonosak, és ellentétes előjelűek.

A játékot formálisan egyfajta keresési problémának is fel lehet fogni az alábbi komponensekkel:

- Kiinduló állapot: tábla-állás + ki fog lépni
- Állapotámenet-függvény: megadja a legális lépéseket, és az azokból következő lépéseket
- Végeszt: meghatározza, hogy mikor van vége a játéknak, ezek az állapotok a végállapotok
- Hasznossági függvény: a játék végeredményéhez egy számértéket rendel hozzá, pl.: -1,0,+1

Optimális stratégia:

Egy optimális stratégia olyan kimenetekhez vezet, amelyek legalább olyan jók, mintha egy tévedhetetlen ellenféllel játszanánk. Az optimális stratégia lényege, hogy a kezdőállapotból a lehetséges lépéseken át eljussunk a nyerésig. Természetesen ebbe az ellenfél is beleszól, vagyis a lépéssorozatban figyelembe kell vennünk, hogy az ellenfél is próbál törekedni az optimális stratégiára.

MINMAX algoritmus:

Alkalmazzuk a minmax fa minden csomópontjára az alábbi függvényt: MINMAX-ÉRTÉK(n)=

Hasznosság(n), ha n egy végállapot

MAX_s $\hat{\cup}$ Követők(n)MINIMAX-ÉRTÉK(s), ha n egy MAX csomópont

MIN_s $\hat{\cup}$ Követők(n)MINIMAX-ÉRTÉK(s), ha n egy MIN csomópont

Az optimális döntést az aktuális állapotból számítja ki, felhasználva az egyes követő állapotok minmax értékeinek kiszámítására a definiáló egyenletekből közvetlenül származtatott, egyszerű rekurzív formulát. A rekurzió egészen a levekig folytatódik, majd a minmax értékeket a fa mentén visszafelé terjesztjük a rekurzió visszalépésekör.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Idő-komplexitása $O(b^m)$ és tár-komplexitása $O(bm)$, ahol m a fa maximális mélysége és b a legális lépések száma.

A MINMAX algoritmus bejárja az egész fát, ezért nagy fa esetén sok erőforrást használ. Ezt elkerülendő, bizonyos feltételek alapján a fa ágainak egy részét elnyeshetjük, és ezáltal ki sem kell értékelni azokat.

Ha a Játékosnak van az n szülőcsomópontjánál vagy ettől feljebb egy jobb választása, akkor az n csomópontot soha nem fogjuk elérni. Ezek alapján az n csomópontot nyugodtan lenyeshetjük.

Az alfa-béta nyesés két paraméterről függ:

- α : az út mentén tetszőleges döntési pontban a MAX számára eddig megtalált legjobb(vagyis legmagasabb értékű) választás értéke
- β : az út mentén tetszőleges döntési pontban a MIN számára eddig megtalált legjobb(vagyis legalacsonyabb értékű) választás értéke.

Az alfa-béta keresés az α és a β értékeit munka közben frissíti, és a csomópontnál a megmaradó ágakat lenyeli. Az algoritmus hatékonysága alapvetően függ a csomópontok vizsgálatának sorrendjétől. Komplexitása: $O(b^{(m^2)})$.

Algoritmus:

alphabeta(állapot, alpha, beta)

- 1) if állapot = végállapot then return hasznosság(állapot)
- 2) for minden s eleme lehetséges következő állapot
- 3) alpha = max (alpha, -alphabeta(s, -beta, -alpha))
- 4) if beta <= (kisebb egyenlo) alpha then return alpha
- 5) return alpha

- Korlátozás kielégítési feladat

Korlátozott kielégítési feladatok:

Definíció: az állapottér: $D = D_1 \times D_2 \times \dots \times D_n$ (D_i az i. változó (x_i) lehetséges értékei), korlátozások: C_1, \dots, C_m , ahol $C_i \subseteq D_i$, konzisztsz (megengedett) hozzárendelések: $C_1 \subseteq C_2 \subseteq \dots \subseteq C_m$.

Egy hozzárendelést konzisztsnek nevezünk, ha nem sért meg egyetlen korlátozást sem. A célállapotok a megengedhető állapotok is egyben és néhány korlátozás kielégítési probléma azt is igényli, hogy a megoldás célfüggvényt maximalizáljon. Általában egy C_i , a változók egy részhalmazára tartalmaz megszorítást, gyakran változópárra.

Algoritmusok:

Kényszergráf: ha C_i párokra vonatkozik, akkor adott, ha nem, akkor segédváltozók bevezetésével átalakítható. A gráf csomópontjai a probléma változóinak, élei pedig a korlátoknak felelnek meg.

A keresési teret inkrementálisan is definiálhatjuk:

kezdeti állapot: $\{\}$ (üres halmaz), azaz egyik változónak sincs értéke.

szomszédok: valamely hiányzó változóhoz rendeljünk értéket, amely nem okoz konfliktust

út költség: konstans költség minden egyik lépésre

2. Tétel

- Teljes együttes eloszlás tömör reprezentációja, Bayes hálók

Ha világot leíró véletlen változók teljes halmazáról gondolkozunk. Az olyan együttes valószínűség-eloszlást, amely lefedi a teljes halmazt teljes együttes valószínűség-eloszlásnak (full joint probability distribution) nevezzük. Például, ha a világ csak a Lyuk, a Fogfajás és az Időjárás változókból áll, akkor a teljes együttes valószínűség-eloszlást a: $P(\text{Lyuk}, \text{Fogfajás}, \text{Időjárás})$ adja meg. Ez az együttes valószínűség-eloszlás egy 16 elemű, $2 \times 2 \times 4$ -es táblázattal reprezentálható. A teljes együttes valószínűség-eloszlás minden egyes elemi esemény valószínűségét, és így a kérdéses világgal kapcsolatos összes bizonytalanságot meghatározza. A teljes együttes valószínűség-eloszlás alapján bármely valószínűségi kérdés megválaszolható.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Feltételes függetlenségről:

Lehet, hogy a és b általában nem függetlenek, viszont létezik c úgy, hogy $P(a „és” b|c) = P(a|c)P(b|c)$.

Ekkor a és b kijelentések feltételesen függetlenek (c feltevésével), például akkor, ha a és b közös oka c . Általában A és B változó függetlenek, akkor és csak akkor, ha létezik C úgy, hogy $P(A|B,C) = P(A|C)P(B|C)$.

A definícióval ekvivalens két másik definíció: $P(A|B,C) = P(A|C)$ és $P(B|A,C) = P(B|C)$.

Bayes - szabály:

$$\text{Állítás: } P(b|a) = \frac{P(a|b)P(b)}{P(a)}$$

$$\text{Bizonyítás: } P(a „és” b) = P(a|b)P(b) = P(b|a)P(a)$$

$$\text{Általában is: } P(B|A) = \frac{P(A|B)P(B)}{P(A)}, \text{ ahol } A \text{ és } B \text{ véletlen változók halmaza}$$

1. Feltételes függetlenség kompakt reprezentációja használható.

2. A gyakorlatban sokszor a jobb oldalon lévő példák ismertek, míg a bal oldaliak nem.

Másik alakja (α normalizálási konstans használatával): $P(B|A) = \alpha P(A|B)P(B)$, itt α adódik az 1-re normalizálásból, de a teljes eloszlást ki kell hozzá számolni.

Bayes háló:

Olyan adatstruktúra, mely a feltételes függetlenségeket ragadja meg, és egy teljes együttes eloszlás tömör reprezentációja.

Részei:

1. Csúcsok: véletlen változók
2. Élek: függési reláció (ha $X \rightarrow Y$ él, akkor X az Y szülője)
3. Feltételes eloszlás: $P(X | X \text{ szülői})$
4. A háló DAG azaz körmentes irányított gráf

A háló nem egyértelmű, az eloszlást több háló is leírhatja!

- Gépi tanulás: felügyelt tanulás problémája, döntési fák, naiv Bayes módszer, modellillesztés, mesterséges neuronhálók, k-legközelebbi szomszéd módszere

Felügyelt tanulás (induktív tanulás - inductive learning):

Egyszerű függvényillesztéstől a fizika tudományáig sok minden lefed.

Adott egy ismeretlen $f: A \rightarrow B$ függvény és $(x_1, f(x_1)), \dots, (x_n, f(x_n))$ példák. Keresünk egy $h: A \rightarrow B$ függvényt, ami f -et közelíti. A h függvény a H hipotézistér eleme, H lehet pl. maximum k fokú polinomok, vagy bármi más.

Indukció problémája:

Az f jól közelítése olyan h , amely ismeretlen x -re is jól becsüli $f(x)$ -et (nem csak az adott példákra), azaz jól általánosít.

$f(x)$, ha ismert a példa
0, egyébként

Pl.: Ha $h(x) = \begin{cases} \textcolor{red}{x} & \text{ha } x \in \textcolor{red}{\{x_1, x_2, x_3\}} \\ \textcolor{brown}{x} & \text{ha } x \in \textcolor{brown}{\{x_1, x_2, x_3\}} \end{cases}$

akkor h egyáltalán nem általánosít, de a tanító példákat tökéletesen tudja. Ebből következik, hogy:

- 1) Törekünk tömörebb reprezentációra, mint az adatok:

Ockham borotvája (Ockham's razor): a lehető legtömörebb leírás, ami még elég
Kolmogorov-komplexitás (algoritmikus komplexitás):

- 2) Törekünk egyszerű reprezentációra a hatékonyság miatt is, mivel egyszerűbb reprezentációban könnyebb keresni.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Döntési fák (decision tree):

Bemenetként egy attribútumokkal leírt objektumot vagy szituációt kap, és egy döntést ad vissza eredményként – a bemenetre adott válasz jóolt értékét. Mind a bemeneti attribútumok, mind pedig a kimeneti érték lehet diszkrét vagy folytonos.

Feltesszük, hogy $x \in A$ diszkrét változók értékeinek vektorá, és $f(x) \in B$ diszkrét változó egy értéke (pl. $B = \{\text{igen, nem}\}$)

Ha a függvény értékkészlete, $f(x)$:

- diszkrét, a függvény tanulását osztályozás (classification) tanulásnak,
- folytonos, a függvény tanulását regressziónak (regression) nevezzük.

Döntési fák kifejezőreje:

Várakozás $\leftrightarrow (A_1 \subseteq \dots \subseteq A_m) \wedge (B_1 \subseteq \dots \subseteq B_n) \dots$

Útvonalak „igen” levélbe, A_i, B_i pedig az elágazások.

A formulában a diszjunkciós tagok mindegyike azon tesztek konjukciójának felel meg, amelyeket a gyökértől egy pozitív kimenetet jelentő levélig megtett út során végeztünk. Ez éppen egy ítéletkalkulusbeli kifejezés, mivel csak egyetlen változót tartalmaz és minden predikátum unáris.

- 1.) Ítéletek, pl.: Vendégek = Senki, Vendégek = Néhány, stb.
- 2.) Az x egy modell.
- 3.) A döntési fa pedig logikai formula, $f(x)$ a formula kiértékelése.

Milyen tömör lehet?

Láttuk, hogy ha véletlen, nem tömöríthető. De nem tömöríthető a paritásfüggvény (akkor és csak akkor ad 1-et, ha páros számú bemenet 1 értékű), ekkor egy exponenciálisan nagy döntési fára lesz szükség. Vagy a többségfüggvény (akkor ad 1-et, ha a bemeneteinek több, mint a fele 1 értékű). Az összes változót tudni kell - illetve $O(n)$ - et.

Döntési fa építése:

Egy logikai döntési fa által kezelhető példa a bemeneti attribútumok X vektorából és egyetlen logikai kimeneti értékből, y -ból áll.

Heurisztika:

A lehető legjobban szétválogatjuk a pozitív és negatív példákat, így a gyökérbe kerül azaz attribútum, amely a legjobban szeparál. (rekurzív algoritmus):

- 1.) Válasszuk ki a gyökeret és szeparáljuk a példákat,
- 2.) minden ágon a maradék attribútumokra és az oda eső példákra rekurzívan ugyanez.

Az esetek, amiket vizsgálni kell a rekurzióban:

- 1.) Pozitív és negatív példa is van: szeparáljuk őket a legjobban egy attribútumot választva.
- 2.) Csak pozitív vagy csak negatív példa van: leáll, ez az ág kész.
- 3.) Nincs példa, de attribútum még van, ekkor „default” érték, és heurisztikát alkalmazunk.
- 4.) Ha pozitív és negatív példa is van, de nincs több attribútum, azaz hiba (zaj) a példákban. Ekkor heurisztikát alkalmaz, pl. többségi szavazat (ha több volt a pozitív, akkor pozitív lesz.)

A legjobban szeparáló attribútum:

Adott változó ismerete mennyivel növeli az információt arról, hogy egy példa „igen” vagy „nem”.

Információ – információelmélet:

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

A p valószínűségű esemény információtartalma: $-\log_2(p)$. Ha \log_2 -t használunk, mértékegysége a bit. Véletlen változó átlagos információtartalma (entrópia): $\sum_i p_i \log_2(p_i)$ (p_1, \dots, p_n , valamint az eloszlás: $1 = \sum p_i$).

Döntési fa egy csomópontjának információtartalma:

Legyen p a csomópont alatti pozitív, n pedig a negatív példák száma, a csomópont A .

$$I(A) = \frac{-p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}$$

Információ-nyereség A-ban:

A gyerek csúcsainak az átlagos információtartalma azt fejezi ki, hogy A - szerint felbontva mennyi „bizonytalanság” marad.

Legyenek B_1, \dots, B_v A gyerek csúcsai. (A -nak v értéke van, ami a részhalmazra osztja a példákat.). Legyenek E_1, \dots, E_v a megfelelő példa részhalmazok, ahol $|E_i| = n_i + p_i$.

$$\text{Nyeréség } (A) = I(A) - \sum_{i=1}^v \frac{p_i}{p+n} I(B_i), \text{ ahol a maximális nyereségű attribútumot választjuk.}$$

A tanító halmazon építjük a modellt (pl. döntési fát), a teszt halmazon értékeljük ki.

„Kukucs kálás” (peeking): Ha a teszteredmények alapján finomhangoljuk az algoritmus paramétereit, akkor a modell flügg a teszttől, és a teszthalmazra van optimalizálva.

Túllillesztés (overfitting): a „magolás” egy általánosabb formája a túllillesztés, amikor túl messzemenő következeteteket vonunk le kevés adatból. Kisszámú példák miatt nem igazi minták alakulhatnak ki.

Keresztsvalidáció (cross-validation): A tanító / teszt felosztás általánosítása, ez a módszer alaposabban szeparál:

- 1.) Osszuk fel a példákat K egyenlő részre.
- 2.) Építünk K modellt a következőképpen: vegyük valamely részhalmazt, mint teszt halmazt és a többi $K-1$ – et tanítsuk.
- 3.) Értékeljük ki a K modellt a megfelelő teszthalmazon és vegyük a K értékelés átlagát, ez lesz az értékelés.

Megbízhatóbb kiértékelés, hiszen az összes elemet felhasználtuk tesztadatnak.

Alkalmazás: Ha van m darab algoritmusunk, minden kiértékeljük keresztsvalidációval, és a legjobb algoritmus segítségével építünk modellt az egész példahalmazon. Ez lesz a végeredmény.

Döntési fa specifikus technika (keresztsvalidációs algoritmus független):

Azért, hogy az irreleváns attribútumokat ne építse be az algoritmusba. Pl. statisztika alkalmazásával. => Valamit tanulni fog, de az biztos tévedés lesz!

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Naiv

Bayes

módszer:

$$\text{Állítás: } P(b|a) = \frac{P(a|b)P(b)}{P(a)}$$

$$\text{Általában is: } P(B|A) = \frac{P(A|B)P(B)}{P(A)}, \text{ ahol A és B véletlen változók halmaza}$$

Feltételes függetlenség kompakt reprezentációja használható.

A Bayes - szabály során az okból az okozatra következtető valószínűséget használjuk fel arra, hogy okozatból okra következtető valószínűséget kapunk.

Másik alakja (α normalizálási konstans használatával): $P(B|A) = \alpha P(A|B)P(B)$, itt α adódik az 1-re normalizálásból, de a teljes eloszlást ki kell hozzá számolni. Naiv Bayes - következtetés:

Adottak:

- Y megfigyelési vektor (y_1, \dots, y_n)
- V osztályok halmaza $\{v_1, \dots, v_k\}$

$$v_{max} = argmax_{v \in V} P(v|y) =_{\text{Bayes tétel}} argmax_{v \in V} \frac{P(Y|v) \cdot P(v)}{P(Y)} \\ = argmax_{v \in V} P(v) \cdot P((y_1, \dots, y_n)|v)$$

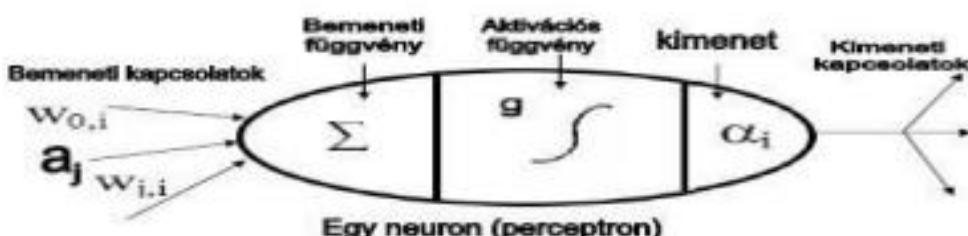
Naiv Bayes tanulás:

o itt $O(N)$ táblázat van: adatokból becsülhetőek a paraméterek maximális likelihood szerint.

o $O(N)$ db osztás kell csak, és a $P(c | x_1, \dots, x_n) = \alpha P(c) \prod_i P(x_i | C)$ becsülhető
o megjegyzés: olcsó, és ehhez képest elég jó

Mesterséges neuronhálók (perceptron)

Neuron absztrakt modellje:



Inputok: tipikus a kétértékű input.

Minden inphothoz (a_i) tartozik súly, $w_{j,i}$, ahol i a neuron indexe.

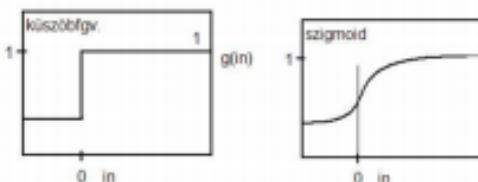
ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Összeszorozzuk az inputokat a súlyokkal, aztán az egészet össze szummázzuk. (belsı szorzat) $w_{0,i}$: eltolássúly (bias weight)

a₀ lineáris bemenet az eltolás kapcsolaton (-1) fix input, nagy jelentőségű. A lineáris függvények eltolásához hasznos lesz. Inputtól nem függ. (Meghatározza, hol lesz a kùszöbérték)

g: (0,1) közé képezi le. Leggyakoribb, lépcsös vagy sigmoid



Aktivációs függvény:

1. Ha jó input jön, adjon 1-korú értéket, ha rossz, akkor 0 körülíti
2. Nemlineáris legyen kùszöbfüggvény vagy sigmoid függvény ($1/(1+e^{-x})$)
3. g(in) éppen $w_{0,i}$ -nél fog változni 0-ból 1-be
4. sigmoid deriválható tanuló algoritmusnak jobb

Általában osztályozási feladatot adunk a neuronoknak. Nem tipikus használat: logikai kapuk. Következmény: bármilyen logikai függvény modellezhető megfelelő számú neuronnal. A neurális háló általában logikailag nem (vagy nehezen) értelmezhető konstrukció. Teljesítmény szempontjából megfelelő. Gyakran túlszámyalja a döntési fát. A neuronokból hálózatot szoktunk építeni. Az egyik neuron outputját hozzácsatoljuk a másik neuron inputjához.

Tanuló algoritmus:

Numerikus optimalizálás: súlyok terében gradiens módszer. Adott pontban megnézzük a függvény deriváltját, megnézzük nö-e vagy csökken, és attól függően, hogy min-t vagy max-ot keresünk, ellépünk valamerre. Hasonlit a hegemászóra. A hibafüggvényt akarjuk minimalizálni. Mintapéldától és súlyvektortól függ (1-1 db). A hibának a gradiensét határozzuk meg w függvényében, ezután ellépünk a minimalizálás irányába. Technikai megoldás: hiba négyzetre emelése, hogy minden pozitív legyen.

$$E(w, x) = \frac{1}{2} Err^2(w, x) = \frac{1}{2} (y - h_w(x))^2$$

Lényeg:

- veszünk egy (x,y) tanítópéldát és
- erre az egy példára csökkentjük Bizonyítás: $P(a \text{ „és” } b) = P(a|b)P(b) = P(b|a)P(a)$ az E hibafüggvényt, és
- ezt ismételjük többször az összes példára, amíg w konvergál.

Példányalapú tanulás – k legközelebbi szomszéd módszere

A tanulás arról szól, hogy modellt húzunk az adatokra, és a modell alapján a meg nem látott dolgokat szeretnénk általánosítani, megjósolni előre. A példa alapú tanulás is hasonlóról szól.

Adott $(x_1, y_1), \dots, (x_n, y_n)$ példák.

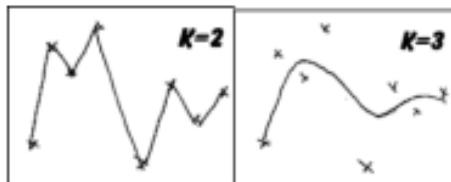
Klaszterezési feladatnál például x alapján kell y-t megmondani. A tanítópéldáknál ezt előre megmondjuk.

Ötlet: ismeretlen x kiértékelése „hasonló” ismert példák alapján. Ezekből a tapasztalatokból állitsuk elő y-t.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Nincs explicit modell, vagy függvény, hanem úgymond halmazzal tanítunk, de egyáltalán nem „magolásról” van szó.



Legközelebbi szomszéd modellek:

1. Keressük meg x -nek a k legközelebbi szomszédját. Globális heurisztikának tekinthető, nagyon általános érvényű („hasonló dolgok hasonlóképp viselkednek”). Ha ezt a Touring-gépek halmazán tekintjük, ez egy merész heurisztika.
 2. $h(x)$ a szomszédok y -jainak átlaga (esetleg távolság szerint súlyozott) ha folytonos a probléma, v a szomszédok y -jainak többségi szavazata (diszkrét). Ha csak x_1, \dots, x_n adott és y -ok nem, akkor a $p(x)$ sürűség közelítése is jó, azaz milyen valószínűsséggel jönnek adott inputok a továbbiakban. Nézzük meg, hogy k legközelebbi szomszéd mekkora területen oszlik el? (sürűség: pontok / terület) fontos: távolságfüggvény $D(x_1, x_2)$
 1. folytonos változónál normalizálni kell a dimenziókat (pl. súly, magasság nem ugyanaz a skála, ellipszkisként képzelhetjük el a vizsgált területet). Ekkor vehetjük például minden jellemző szórását, és a jellemző értéket a szórás többszöröseiként fejezzük ki.
Folytonos esetben érdemes valamelyen euklideszi távolságot venni.
 2. diszkrét esetben Hamming távolság: különböző jellemzők száma (hány változóban különbözik), pl. $H(001,111) = 2$ (2 helyen különbözik) egyszerű, de sokszor jó tanuló algoritmus. Problémái:
 - érzékeny a távolság-függvényre (holott a módszer egyszerű ötleten alapszik)
 - nagy adathalmazokon költséges a k szomszédot megtalálni (néha még végigmenni is költséges egy adathalmazon, internet), ilyenkor a modellépítés célravezetőbb lesz
 - ha sok jellemző van (sokdimenziós tér) akkor kb. „ mindenki közel van mindenkihez”.
- Kernel módszerek:**
- Módszer: minden példát saját jogán veszek figyelembe (szomszéd nincs). minden példához hozzárendelek egy saját függvényt (pl. harang-alakú függvényt), ami azt mutatja, milyen súlyjal vesszük a példát figyelembe. minden x_i példához veszünk egy $K(x, x_i)$ kernel - függvényt, pl x_i várható értékű, szigma normális eloszlás. Általában a K függvény egy sürűség-(illetve diszkrét esetben eloszlás-) függvény.
- sürűség közelítése egy x pontra. $p(x) = (1/n)\sum K(x, x_i)$
 - $h(x)$ közelítése: K -val súlyozott átlag
- Megjegyzés: itt minden tanítópéldát figyelembe veszünk

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Operációkutatás I

1. Tétel

- LP alapfeladat, példa

Definíció (LP alapfeladat): Keressük adott lineáris \mathbb{R}^n értelmezési tartományú függvény (célfüggvény) szélsőértékét (min/max) értelmezési tartományának adott lineáris korlátokkal (feltételekkel) meghatározott részében.

A standard alakú lineáris programozási feladat (maximalizálás) felírása a következő:

$$\begin{aligned} \sum_{j=1}^n a_{ij}x_j &\leq b_i & i = 1, 2, \dots, m \\ x_j &\geq 0 & j = 1, 2, \dots, n \\ \max \sum_{i=1}^n c_i x_i &= z \end{aligned}$$

Részletesen kiírva az együtthatókat:

$$\begin{array}{ll} \text{Max} & c_1x_1 + c_2x_2 + \dots + c_nx_n = z \\ \text{Feltételek} & a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq b_1 \\ & a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \leq b_2 \\ & \dots \\ & a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq b_m \\ & x_1, \dots, x_n \geq 0 \end{array}$$

Az első sor a célfüggvény, amelyet z -vel jelölünk, az ezt követő sorok az általános feltételek, az utolsó sor pedig a változókra vonatkozó egyedi nemnegativitási feltételek kifejezése.

Két alakot különböztetünk meg a lineáris programozási feladatoknál, a standard és a kanonikus alakot. Ha a feltételrendszer csak \leq relációt tartalmaz, a változók értékei csak nemnegatívak, és a célfüggvénynek a maximumát keressük, akkor standard alakról beszélünk.

- Szimplex algoritmus

A lineáris programozási feladatokat számos módon meg lehet oldani, de a legfontosabb megoldó algoritmusa a szimplex módszer, és annak különböző variánsai. Ahhoz, hogy a szimplex módszert legegyszerűbben alkalmazni tudjuk, szükségünk van egy kezdőszótárra. Egy LP kezdőszótárát úgy kapjuk meg, hogy bevezetünk új úgynevezett mesterséges változókat a már meglévő természetes változók mellé, amelyek nemnegatívak: $x_{n+1}, x_{n+2}, \dots, x_{n+m}$, majd egyenleteket írunk fel:

$$\begin{aligned} x_{n+1} &= b_1 - a_{11}x_1 - a_{12}x_2 - \dots - a_{1n}x_n \\ x_{n+2} &= b_2 - a_{21}x_1 - a_{22}x_2 - \dots - a_{2n}x_n \\ &\dots \\ x_{n+m} &= b_m - a_{m1}x_1 - a_{m2}x_2 - \dots - a_{mn}x_n \\ z &= c_1x_1 + c_2x_2 + \dots + c_nx_n \end{aligned}$$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Az egyenlet bal oldalán a bázisváltozók állnak, a jobb oldalán pedig a nem bázisváltozók. A szimplex algoritmus egyes lépéseiit iterációknak nevezzük. A szimplex algoritmus egy iterációja során új szótárt adunk meg egy bázis (kilépőváltozó) és nem bázisváltozó (belépőváltozó) felcserélésével.

Az algoritmus lépései:

1. Megnézzük a célfüggvény együtthatókat, $c_j \leq 0$ minden $j = 1, 2, \dots, n - re?$ \rightarrow amennyiben az összes célfüggvény együttható negatív, megtaláltuk az optimális megoldást, melyet úgy olvasunk le, hogy a nem bázisváltozók értéke = 0, a bázisváltozók értéke a megfelelő konstans, a célfüggvény érétekét pedig egyszerűen leolvassuk. Ha viszont van egy vagy több pozitív együttható, akkor megyünk a következő lépére.
2. Válasszunk egy pozitív együtthatós nem bázisváltozót, azaz valamely $x_k - t$, amelyre $c_k > 0$, ez lesz a belépőváltozónk.
3. A célfüggvényben kiválasztott együttható oszlopát megnézzük, hogy az összes együttható pozitív-e, azaz $-a_{ik} \geq 0$ minden $i = 1, 2, \dots, m - re?$ \rightarrow amennyiben igen, az LP feladat nem korlátos, és az algoritmus megáll, ha viszont nem, akkor megyünk a következő lépére.
4. Határozzuk meg a legszűkebb korlátot a következőképpen: legyen l valamely index, amelyre $[b_l/a_{lk}]$ minimális és $-a_{lk} < 0$, ez lesz a kilépőváltozónk, az eljárást pedig hárnyadostesztnek nevezzük.
5. Hajtsunk végre egy pivot lépést, a megkapott belépő és kilépőváltozóinkkal, majd térjünk vissza az 1. lépére.
- Az LP geometriája

Lehetséges megoldások halmazának megkeresése:

- A lehetséges megoldások halmaza ebben az esetben egy **kétdimenziós tartomány** a síkban.
- **Ábrázoljuk** a korlátozó feltételeket egy koordinátarendszerben, amelynek a vízszintes tengelyen x_1 döntési változót, a függőleges tengelyén pedig az x_2 döntési változót vesszük fel. Az egyenlőtlenségekkel megadott korlátozó feltételek egy félsíket, az egyenlőséggel megadott feltételek pedig egy egyenest határoznak meg.

A $5x_1 + 5x_2 \leq 200$ feltétel egy olyan félsíket határoz meg, amely határegyenésének egyenlete $5x_1 + 5x_2 = 200$. Megrajzoljuk ezt az egyenest a tengelyekkel való metszéspontok segítségével. Legyen $x_1 = 0$. Ekkor $5 \times 0 + 5x_2 = 200$ egyenlet megoldása $x_2 = 40$. Ha $x_2 = 0$, akkor a $5x_1 + 5 \times 0 = 200$ egyenlet megoldása $x_1 = 40$. A metszéspontokat a következők:

| | | |
|-------|----|----|
| x_1 | 0 | 40 |
| x_2 | 40 | 0 |

A metszéspontokat jelöljük $A (0, 40)$ -val és $B (40, 0)$ -vel. Megvizsgáljuk, hogy az $O (0, 0)$ pont a lehetséges pontokat tartalmazó félsíkhoz tartozik-e, behelyettesítve az O pont koordinátait az előző feltételbe. Mivel az $5 \times 0 + 5 \times 0 \leq 200$ feltétel teljesül, ezért a lehetséges megoldások halmaza az AB egyenesnek az O irányába eső félsíkjában van.

Az $x_2 \leq 24$ feltétel határegyenese az $x_2 = 24$ vízszintes egyenes, amely a függőleges tengelyt az $E (0, 24)$ pontban metszi. Mivel $0 \leq 24$, a lehetséges megoldások halmaza az egyenes origó felőli félsíkjába esik.

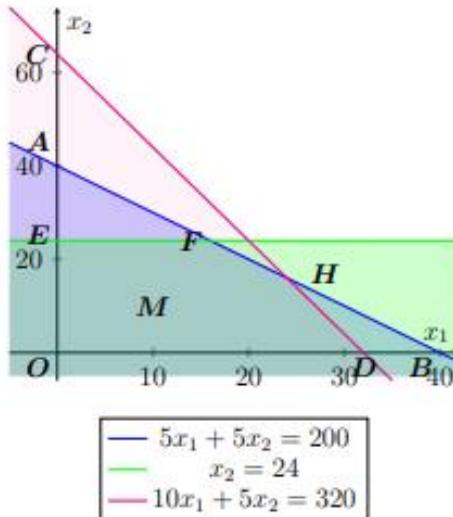
A $10x_1 + 5x_2 \leq 320$ feltétel határegyenese $10x_1 + 5x_2 = 320$.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

A tengelyekkel való metszéspontok $C(0, 64)$ és $D(32, 0)$. Mivel a $10x_1 + 5x_2 \leq 320$ feltétel teljesül, ezért a lehetséges megoldások halmaza a CD egyenesnek az O irányába eső félsíkjában van.

A feltételeket a következő ábra szemlélteti:



Jól látható, hogy az általunk meghatározott egyenleteket ábrázolva kirajzolódik a **lehetséges megoldások halmaza**, (jele: M) azaz az a terület, amelyik minden függvény feltételeinek eleget tesz.

Ha a döntési változókra csak $x_1 \geq 0, x_2 \geq 0$ feltételek volnának felírva, akkor a lehetséges megoldások M halmaza a három korlátos feltétel által meghatározott tartományok metszetének a koordináta-rendszer első negyedébe eső része, de mivel $x_1, x_2 \in \mathbb{Z}$, a feladat lehetséges megoldásainak halmaza az M -beli egész koordinátájú pontokat tartalmazza. Az M egy olyan poliéder, amelynek csúcsPontjai: O, E, F, H, D . Ezeket a csúcsPontokat, amelyeket két feltétel egyenesének metszéspontjában kapunk, extremális pontoknak nevezzük.

Tétel

Ha egy LP feladatnak van optimális megoldása, akkor olyan optimális megoldása is van, ami a lehetséges megoldási tartomány csúcsPontja.

Általánosan: egy **lehetséges megoldás** egy olyan $p = (p_1, \dots, p_n) \in \mathbb{R}^n$ vektor, hogy x_i -be p_i -t helyettesítve kielégíti a feltételrendszeret ($\forall i \in (1, 2, \dots, n)$).

Az optimális megoldás keresése: Az optimális megoldás az az (x_1, x_2) pont a lehetséges megoldások halmazából, amelyre a $z = 50x_1 + 40x_2$ célfüggvény értéke a legnagyobb.

Általánosan: az **optimális megoldás** olyan lehetséges megoldás, amelyben a célfüggvény felveszi a maximumát/minimumát.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- Generálóelem választási szabályok

A Pivot stratégiák:

- Klasszikus szimplex algoritmus pivot szabálya:
- A lehetséges belépőváltozók közül a legnagyobb c_k értékű válasszuk, amennyiben több ilyen is van, akkor a legkisebb indexűt. A lehetséges kilépőváltozók közül pedig a legkisebb 1 indexű egyenlet változóját válasszuk.
- Bland szabály, azaz a legkisebb indexek módszere:
- A lehetséges belépőváltozók közül a legkisebb indexűt és a lehetséges kilépőváltozók közül is a legkisebb indexűt válasszuk. Ez a szabály garantálja, hogy az algoritmus véges számú lépéssben véget ér.
- Legnagyobb növekmény módszere:
- A lehetséges belépőváltozók közül olyat választunk, amelyik a célfüggvényt legnagyobb mértékben növeli, ennek elérése érdekében a következő algoritmust alkalmazzuk:
- $\max(c_i \times \min(\text{lehetséges } \left\lfloor \frac{b_j}{a_{ij}} \right\rfloor \text{ korlátok}))$, ahol $c_i > 0$ és c_i az x_i célfüggvény együtthatója, valamint a_{ij} negatív, az x_i j. egyenletben levő együtthatója és b_j a j. egyenlet konstansa.

- Kétfázisú szimplex módszer

Két fázisú szimplex módszerről beszélünk, amennyiben a mesterséges változók valamelyikének értéke negatív, ugyanis az nem egy lehetséges kezdőszótár, ilyenkor segédfeladatra van szükségünk, és a segédfeladat felírása lesz a módszer első fázisa, a második fázis pedig a standard szimplex módszer. Duális feladatnál elsőként be kell szoroznunk -1 -el az egész feladatot, hogy standard alakra hozzuk. Ha ez megtörtént, egyértelműen láthatjuk, hogy kétfázisú szimplex módszerre lesz szükségünk a feladat megoldásához, mivel a feltételek konstansai negatívak.

- Speciális esetek (ciklizáció- degeneráció, nem korlátos feladat, nincs lehetséges megoldás)

Ha egy iteráció során a bázismegoldás nem változik, degenerált iterációs lépésről beszélünk. Ha egy vagy több bázisváltozó értéke nulla a bázismegoldásban, akkor degenerált bázismegoldásról beszélünk. A degeneráció miatt ciklizáció alakulhat ki, ami azt jelenti, hogy az algoritmus valamely iterációja végén egy korábbi iteráció szótárát kapjuk meg újra, és ilyenkor az algoritmus ciklizál. Nem korlátos LP: A célfüggvényben kiválasztott együttható oszlopát megnézzük, hogy az összes együttható pozitív-e, azaz $-a_{ik} \geq 0$ minden $i = 1, 2, \dots, m - re? \rightarrow$ amennyiben igen, az LP feladat nem korlátos, és az algoritmus megáll.

2. Tétel

- Primál-duál feladatpár

A dualitás egy általános elv, mely számos területen van jelen, így a lineáris programozási feladatoknál is nagy jelentősége van. Az általános LP feladatot nevezzük most primálnak, melynek van egy úgynévezett duál párja, ez a primál- duál feladatpár.

Amennyiben a következő a primál feladatunk általános alakja:

$$\sum_{j=1}^n a_{ij}x_j \leq b_i \quad i = 1, 2, \dots, m$$

$$x_j \geq 0 \quad j = 1, 2, \dots, n$$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

$$\max \sum_{i=1}^n c_i x_i = z$$

akkor a duál feladat általános alakja így néz ki:

$$\begin{aligned} \sum_{i=1}^m a_{ij} y_i &\geq c_j & j = 1, 2, \dots, n \\ y_i &\geq 0 & i = 1, 2, \dots, m \end{aligned}$$

$$\min \sum_{i=1}^m b_i y_i = w$$

Ebből megállapíthatjuk, hogy a duál duálisa az a primál.

Valamint azt is, hogy míg a primál feladatban a célfüggvényt maximalizáltuk, duálban minimalizálunk, ezért az egyenlőtlenségeket $\geq -ra$ cseréljük, a duál célfüggvény együtthatói pedig nem mások, mint a primál feladat jobb oldalai. Duálban pontosan annyi változó lesz, mint amennyi feltétel volt primálban, valamint a primál célfüggvény együtthatói jelennek meg a duál feladat feltételrendszerében.

- Dualitási komplementaritási tételek

Tétel (Gyenge dualitás). Ha $x = (x_1, \dots, x_n)$ lehetséges megoldása a primál feladatnak, és $y = (y_1, \dots, y_m)$ lehetséges megoldása a duál feladatnak, akkor:

$$\sum_{j=1}^n c_j x_j \leq \sum_{i=1}^m b_i y_i$$

Tehát a duális feladat bármely lehetséges megoldása felső korlátot ad a primál bármely lehetséges megoldására, vagyis az optimális megoldásra is.

Tétel (Erős dualitás). Ha $x^* = (x_1, \dots, x_n)$ optimális megoldása a primál feladatnak, és $y^* = (y_1, \dots, y_m)$ optimális megoldása a duál feladatnak, akkor:

$$\sum_{j=1}^n c_j x_j = \sum_{i=1}^m b_i y_i$$

Komplementáris lazaság: Valamint az is igaz, hogy ha valamelyik i -edik feltétel egyenletnél nincs egyenlőség a primál optimumban, akkor a kapcsolódó duál y_i változó 0 kell legyen. A másik oldalról tekintve pedig, ha egy primál x_i változó szigorúan pozitív, akkor a kapcsolódó duális feltétel egyenletnél egyenlőség kell legyen.

- Egész értékű feladatok és jellemzőik

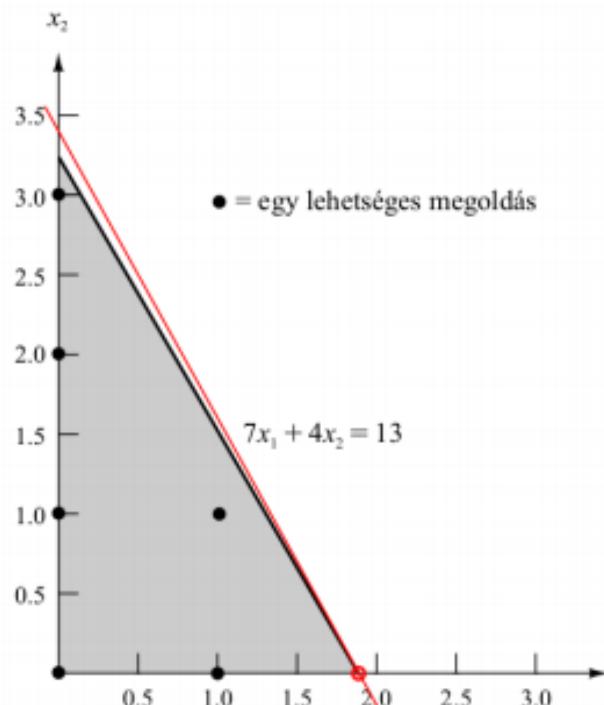
ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Az egészértékű lineáris feladat NP nehéz, nincs polinom futási idejű algoritmus a megoldására. Az egészértékű programozási feladatoknál a változók csak egész értékét vehetnek fel. Tiszta egészértékű programozásról akkor beszélünk, ha a modellben az összes változónak egészértékűnek kell lennie, ha viszont csak valamely változóra érvényes az egészértékűség, akkor vegyes egészértékű programozási feladatról beszélünk. Ha azt is meghatározzuk, hogy a változók értéke csak 0 vagy 1 lehet, akkor pedig 0-1 egészértékű vagy bináris feladatról beszélünk.

Állítások

- Bármelyik IP lehetséges megoldáshalmaza része az LP-lazítása lehetséges megoldástartományának.
- Maximalizálnál az LP-lazítás optimum értéke \geq az IP optimum értékénél.
- Ha az LP-lazítás lehetséges megoldáshalmazának minden csúcspontja egész, akkor van egész optimális megoldása ami az IP megoldása is egyben.
- Az LP-lazítás optimális megoldása bármilyen messze lehet az IP megoldásától.



- A branch and bound módszer (korlátozás és szétválasztás)

A korlátozás és szétválasztás módszer alapelve az, hogy ha egy feladat túl bonyolult, akkor azt részfeladatokra kell bontani, és úgy megoldani. A feladatot úgy bontjuk részfeladatokra, hogy a részfeladatok lehetséges megoldásainak halmaza az eredeti feladat megoldási halmazának részhalmaza legyen, a célfüggvénye pedig azonos az eredeti feladatéval, erre utal a szétválasztás. Ha a célfüggvényt maximalizáljuk, akkor a korlátozás azt jelenti, hogy a részfeladatok optimális célfüggvény értékére korlátokat szabunk meg valamilyen módszerrel.

Hogyan néz ki a különböző feladatok lehetséges megoldásainak a halmaza?

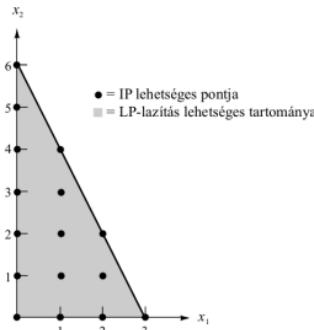


A) LP (szürke háromszög)

B) Vegyes egészértékű programozási feladat (sárga szakaszok)

C) IP, azaz tiszta egészértékű programozási feladat (fekete pontok)

LP-lazítás
Egy egészértékű programozási feladat **LP-lazítása** az az LP, amelyet úgy kapunk az IP-ből, hogy a változóra tett minden egészértékű vagy 0-1 megkötést eltörölünk.



ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Példa⁴

$x_1, x_2 \geq 0$, és egész

$$4x_1 + 2x_2 \leq 20$$

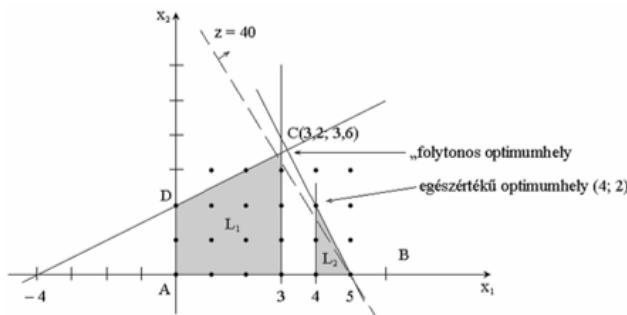
$$-x_1 + 2x_2 \leq 4$$

$$z = 8x_1 + 5x_2 \rightarrow \max$$

Ha megoldjuk a feladatot, megkapjuk, hogy a célfüggvény értéke:

$$z = 8 \times 3,2 + 5 \times 3,6 = 25,6 + 18 = 43,6$$

Amiből látjuk, hogy a részfeladat célfüggvényének felső korlátja 43.



$15 = 39$. Mivel ez kisebb, mint az előző megoldás, ezért az L_2 részfeladat optimális megoldása egyben az eredeti feladat optimális megoldása is.

- A hátizsák feladat

Hátizsák probléma

Egy olyan IP-t, amelyben csak egy feltétel van, **hátizsák feladatnak** hívunk.

Példa

Josie Camper egy kétnapos túrára készül. Négy tárgy van, amire szüksége lehet, de csak 1400 grammot tud ezekből elvinni. A tárgyak súlya, illetve haszná Josie szerint

| Tárgy | Súly(100g) | Haszon | Relatív hasznosság | Sorrend |
|-----------|------------|--------|--------------------|---------|
| Tablet | 5 | 16 | 3.2 | 1. |
| Laptop | 7 | 22 | 3.1 | 2. |
| Okostelő | 4 | 12 | 3 | 3. |
| Elemlámpa | 3 | 8 | 2.7 | 4. |

Ha az eredeti feladat ismert lehetséges megoldásaihoz tartozó célfüggvény értékeknél nem nagyobb a részfeladat célfüggvényének felső korlátja, akkor nincs szükség a részfeladat további vizsgálatára.

Láthatjuk az ábrán, hogy az L_2 részfeladat optimális megoldása egészértékű optimális megoldás: 4,2. Ha ezeket az értékeket behelyettesítjük a célfüggvényünkbe, akkor:

$$z(L_2) = 8 \times 4 + 5 \times 2 = 32 + 10 = 42$$
. Az L_1 részfeladatnál egészértékű optimális megoldás a 3,3 lehet, ekkor a célfüggvényünk:

$$z(L_1) = 8 \times 3 + 5 \times 3 = 24 +$$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Matematikai modell

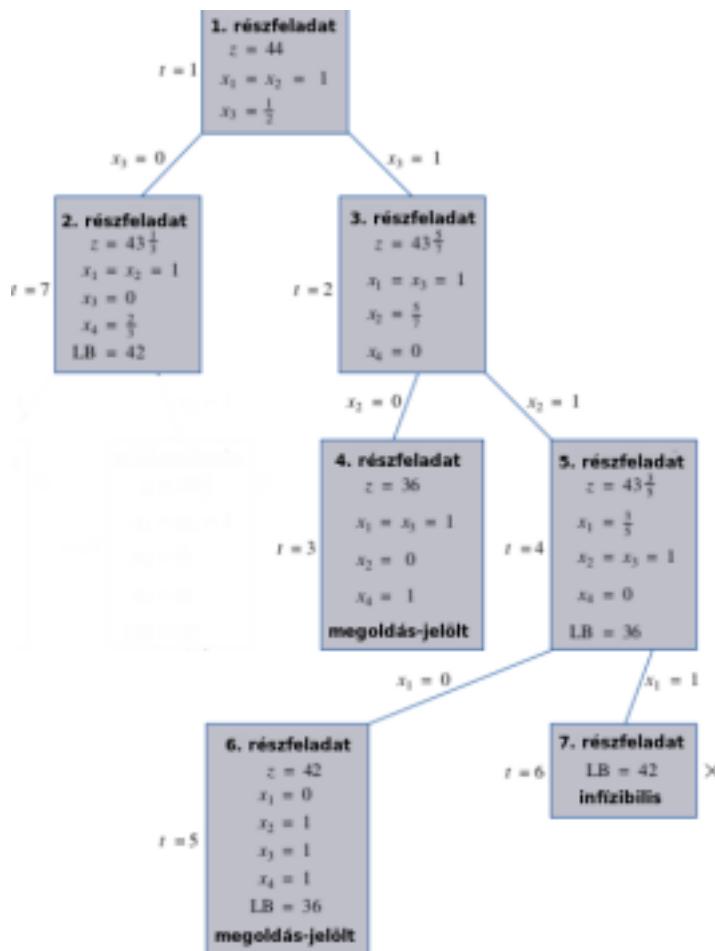
Legyen $x_i = 1$ ha az i . tárgyat viszi, $x_i = 0$ ha marad. Ekkor a feladat

$$\begin{aligned} \max \quad & z = 16x_1 + 22x_2 + 12x_3 + 8x_4 \\ \text{f.h.} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \\ & x_i \in \{0, 1\} \end{aligned}$$

Az LP-lazítás megoldása könnyen számítható: a relatív hasznosság szerint tesszük be sorba a tárgyakat a hátizsákba, ami nem fér, annak csak tört részét.

A tablet meg a laptop bekerül, ezzel 1200 g a súlya, a telő 400 g, ennek a fele fér bele: $x_1 = 1, x_2 = 1, x_3 = 0.5, z = 16 + 22 + 0.5 * 12 = 44$.

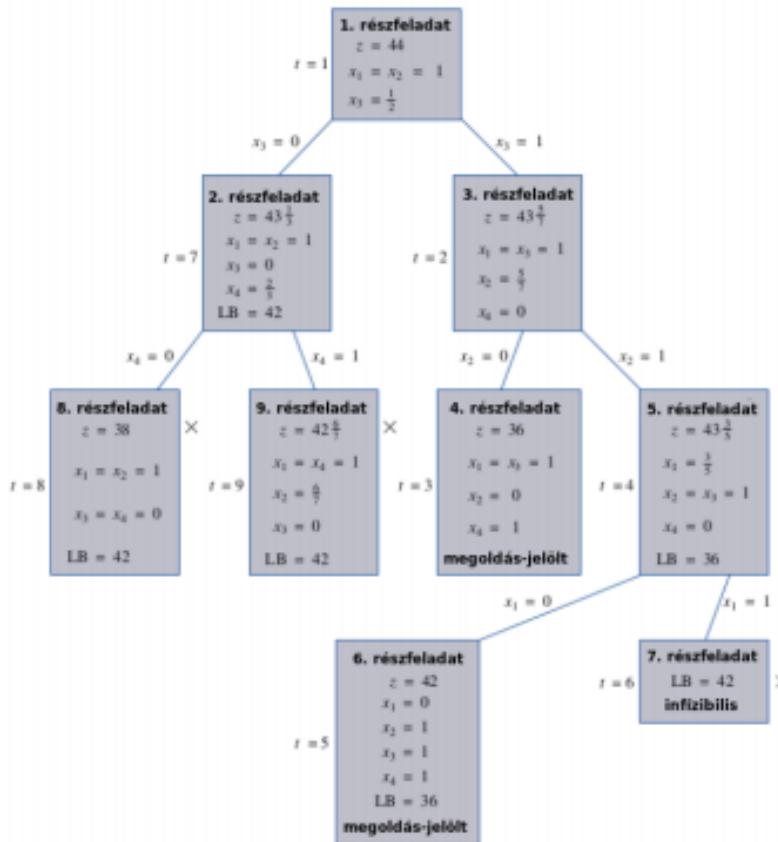
Az egyetlen tört változó szerint szébtontjuk: $x_3 = 1$ (már csak 1000 g-nyi tárgyat keresünk) vagy $x_3 = 0$ (csak 3 tárgyból keresünk).



| Tárgy | Súly | Haszon |
|----------|------|--------|
| Tablet | 5 | 16 |
| Laptop | 7 | 22 |
| Okostelő | 4 | 12 |
| Lámpa | 3 | 8 |

2. lépésben az $x_3 = 1$ feltétellel kiegészített feladatot oldjuk meg. Majd hozzávesszük a $x_2 = 0$ feltételt, így kapunk egy egész megoldást, ami alsó korlát a feladat megoldására: LB=36. A 6. részfeladat megoldása egész, LB=42-re módosul. A 7. részf. nem lehetséges, mind a 4 tárgy nem fér bele a hátizsákba.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK



| Tárgy | Súly | Haszon |
|----------|------|--------|
| Tablet | 5 | 16 |
| Laptop | 7 | 22 |
| Okostelő | 4 | 12 |
| Lámpa | 3 | 8 |

A 8. részf. elvethető, mert $z \leq LB$.
A 9. részf. elvethető, hiszen a célfüggvény együtthatók miatt minden egész megoldáshoz egészértékű hasznosság tartozik, így az $LB=42$ -nél jobb megoldás nem lehet ezen az ágon sem.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Operációs rendszerek

1. Tétel
 - Processzusok, szálak/fonalak, processzus létrehozása/befejezése, processzusok állapotai, processzus leírása
 - Ütemezési stratégiák és algoritmusok kötegelt, interaktív és valós idejű rendszereknél, ütemezési algoritmusok céljai.
 - Kontextus-csere
2. Tétel
 - Processzusok kommunikációka, versenyhelyzetek, kölcsönös kizárási mechanizmusai
 - Konkurens és kooperatív processzusok
 - Kritikus szekciók és megvalósítási módszereik: kölcsönös kizárási meccanismusok (megszakítások tiltása, változók zárolása, szigorú válogatás, Peterson megoldása, TSL utasítás)
 - Altatás és ébresztés: termelő- fogyasztó probléma, szemaforok, mutex-ek, monitorok, üzenet, adás, vétel
 - Írók és olvasók problémája
 - Sorompók

os1_1, os1_2, os2 → pdf-ek

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

TÖRZSTÁRGYAK II.

Adatbázisok

1. Tétel

- Adatbázis tervezés: A relációs adatmodell fogalma

Egy adatbázis-alkalmazásnál az alábbi szinteket különböztethetjük meg:

-Felhasználói felület

Célalkalmazásként készített program

-Adatmodell (logikai adatstuktúra)

DBMS

-Fizikai adatstruktúra

Adatmodellek

Adatbázisséma: az adatbázis struktúrájának leírása. Erre különféle adatmodellek használatosak. Relációs adatmodell: Az egyedeket, tulajdonságokat és kapcsolatokat egyaránt táblázatok, úgynevezett adattáblák segítségével adja meg. Az adattábla (vagy egyszerűen csak tábla) sorokból és oszlopokból áll. Egy sorát rekordnak nevezzük, amely annyi mezőből áll, ahány oszlopa van a táblának.

- Az egyed-kapcsolat diagram és leképezése relációs modellre, kulcsok fajtái

Egyed-kapcsolat modell

Grafikus leíró eszköz, diagram segítségével szemléletesen adja meg az adatbázis struktúráját. Az adatbázis implementálásához a diagramot transzformálni kell valamelyen adatmodellre, ill. annak megfelelő nyelvi leírásra (pl. SQL).

Példa. Tegyük fel, hogy egy könyvtár kölcsönzési nyilvántartását szeretnénk adatbázissal megoldani. Ehhez nyilvántartást kell vezetni a könyvekről, az olvasókról, valamint a kikölcsönzési és visszahozási időpontokról. A modell megalkotásához néhány alapfogalmat meg kell ismernünk.

Egyed/entitás: egy, a valós világban létező dolgot, amit tulajdonságokkal akarunk leírni. Például: könyv, olvasó.

Tulajdonság/attribútum: az egyed egy jellemzőjét. Például: könyv címe, szerzőjének neve.

Az attribútumokat úgy célszerű megválasztani, hogy azok egyértelműen meghatározzák az egyedet. Mivel adott szerző adott című könyve több kiadásban is megjelenhet, sőt adott kiadásból is több példány lehet a könyvtárban, így minden könyvhöz egy egyedi azonosítót, könyvszámot (könyvtári számot) célszerű felvenni. Ekkor a "könyv" egyed tulajdonságai: könyvszám, szerző, cím. Hasonló meggondolások alapján az "olvasó" egyedhez olvasószám, név, lakcím tulajdonságokat rendelhetünk.

Szuperkulcs: egy attribútumhalmaz, mely egyértelműen azonosítja a tábla sorait. Pontosabban: egy $R(A_1, \dots, A_n)$ relációséma esetén az $A = \{A_1, \dots, A_n\}$ attribútumhalmaz egy K részhalmaza szuperkulcs, ha bármely R feletti T tábla bármely két sora K-n különbözik. Példa. A Könyv (könyvszám, szerző, cím) sémában a {könyvszám} szuperkulcs, de szuperkulcs például a {könyvszám, szerző} vagy a {könyvszám, cím} attribútumhalmaz is. A teljes A attribútumhalmaz minden sora különböző. **Kulcs:** az A attribútumhalmaz K részhalmaza, ha

11. ábra - EK példa



ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

minimális szuperkulcs, vagyis egyetlen valódi részhalmaza sem szuperkulcs. Ha K egyetlen attribútumból áll, akkor egyszerű, egyébként összetett kulesról beszélünk. Ha egy relációsémának több kulcsa is van, egyet kiválasztunk közülük, ez lesz az elsődleges kulcs. Egy relációséma attribútumainak valamely részhalmaza külső kulcs (AKA idegen kulcs, foreign key), ha egy másik séma elsődleges kulcsára hivatkozik. A külső kulcs értéke a hivatkozott táblában előforduló kulcsérték vagy NULL lehet. Egy egyed attribútumainak azt a minimális részhalmaza, amely egyértelműen meghatározza az egyedet. Jele: aláhúzás. Esetünkben a „könyv” egyed kulcsa a könyvszám, az „olvasó” egyedé az olvasószám. A "könyv" és "olvasó" egyedek között ugyanis egy sajátos kapcsolat léphet fel, amelyet kölcsönöznek nevezünk. Ezen kapcsolathoz a kivétel és visszahozás időpontját rendelhetjük tulajdonságként. A valós világ jelenségeit egyedekkel, tulajdonságokkal és kapcsolatokkal leíró modellt egyed-kapcsolat modellnek, az ezt ábrázoló diagramot egyed-kapcsolat diagramnak nevezik. (AKA: E-K modell & E-K diagram, vagy E-K helyett E-R (entity-relationship)).

Az egyed-kapcsolat diagramoknak sajátos jelölésrendszerük van:

- az egyedekeket téglalappal,
- az attribútumokat ellipszissel,
- a kapcsolatokat rombusszal

szokták jelölni. A 11. ábra a fentiekben tárgyalt könyvtári nyilvántartás E-K diagramját ábrázolja. A tervezés kezdeti szakaszában, illetve bonyolult E-K diagramok esetén az attribútumok ábrázolását el szokták hagyni. Az eddig leírtaknál kissé pontatlanul fogalmaztunk, ugyanis meg kell különböztetni egyedpéldányt, egyedtípus és egyedhalmazt. Példánkban az egyedpéldány egy adott könyvet, az egyedtípus a könyv fogalmat jelenti. Egy valós adatbázisban minden egyedtípusnak egy konkrét egyedhalmaz (egyedpéldányok halmaza) felel meg. A kissé nehézkes terminológia elkerülésére az egyedpéldány, egyedtípus és egyedhalmaz helyett egyszerűen egyedet mondunk, ha ez nem értelemezvaró.

Tulajdonságpéldány: egy egyedpéldány adott tulajdonsága (például adott könyv szerzőjének nevét).

Tulajdonságtípus: adott egyedtípus adott tulajdonságát, mint fogalmat jelöli (például könyvek esetén a "szerző" fogalmat). Ugyanígy meg lehet különböztetni kapcsolatpéldányt, amely két egyedpéldány közötti konkrét kapcsolatot jelent (például X olvasó kikölcönözte Y könyvet), kapcsolattípushat és kapcsolathalmazt, ez utóbbi a két egyedtípus közötti kapcsolatok összességét jelenti.

Kapcsolattípusok

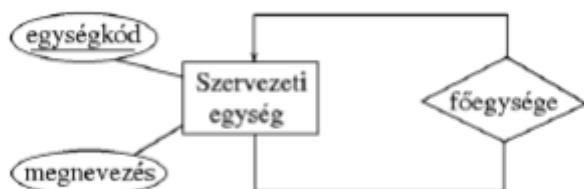
a). Két egyed közötti (bináris) kapcsolat, mint a könyvtári példa esetében.

- 1:1 kapcsolat
- 1:N kapcsolat (egy-a-sokhoz kapcsolat)
- N:M kapcsolat (sok-a-sokhoz kapcsolat)

b). Kettőnél több egyed közötti (sokágú) kapcsolat. Ez a típus ritkábban lép fel, szükség esetén visszavezethető bináris kapcsolatokra.

Jelölés: az „1” oldalon nyílfej van. Egy egyedtípus teljesen részt vesz egy kapcsolatban, ha minden egyedpéldány kapcsolatban áll valamely másik egyeddel. Ha ezt hangsúlyozni akarjuk, akkor az egyed és a kapcsolat közötti kettős vonalat húzunk.

Példa. Előfordul, hogy egy egyedtípus önmagával áll kapcsolatban. Az ábra egy hierarchikus felépítésű intézmény szervezeti egységeit modellezzi (például egyetemi karok, tanszékcsoportok stb.). Itt 1:N kapcsolatról van szó, ahol egy kapcsolatpéldány azt jelenti, hogy X egységnek Y egység a főegysége. Ez a modell nem zárja ki a körkörös hivatkozásokat.



ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

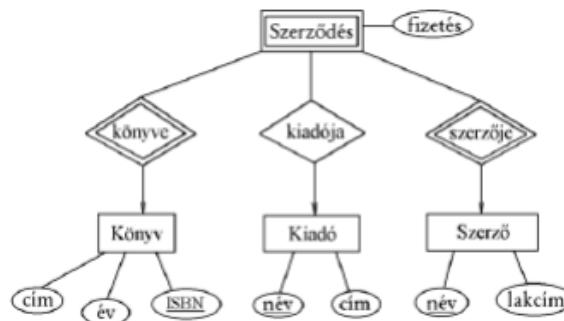
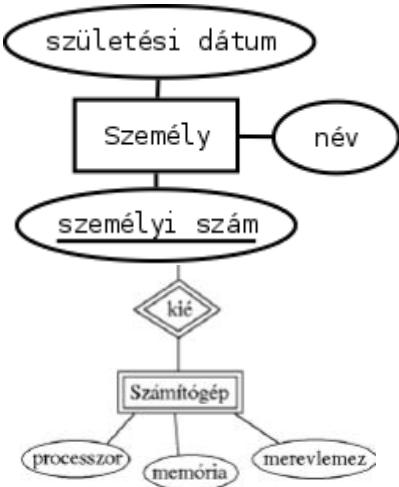
Összetett attribútum (struktúra): maga is attribútumokkal rendelkezik. Például a lakcím attribútumhoz az irányítószám, helység, utca, házszám részattribútumok tartoznak. Jelölése: attribútumhoz kapcsolódó attribútumok.

Többértékű attribútum: aktuális értéke halmaz (nem fontos a sorrend) vagy lista (fontos a sorrend) lehet. Használhatjuk ezt, ha pl. egy könyvnek több szerzője van. Jele: kettős ellipszis. **Gyenge entitás:** az attribútumai nem határozzák meg egyértelműen, csak a kapcsolatai. Jele: kettős téglalap.

Meghatározó kapcsolat: gyenge entitást határoz meg. Jele: kettős rombusz.

Példa. Egy számítógép szerviz nem bajlódik azzal, hogy egyedi azonosítót rendeljen a javított gépekhez, hanem azokat a tulajdonosaik szerint tartja nyilván. Itt a számítógép gyenge entitás, mivel a műszaki paraméterek nem határozzák meg egyértelműen a gépet. Ha előfordulhat, hogy egy tulajdonosnak több, azonos paraméterrel rendelkező gépe van, akkor a számítógép egyedhez egy sorszám attribútum felvétele is szükséges a megkülönböztetésre.

N:M típusú és sokágú kapcsolat minden helyettesíthető gyenge entitással és több bináris kapcsolattal.



Specializáló általános

altípusait külön szeretnénk modellezni, akkor a főtípus és az altípusok viszonyát specializáló kapcsolattal írhatjuk le. Jelölés: háromszög, amelynek csúcsa a főtípus felé mutat. Példa: helyiség egyed altípusai: tanterem, gépterem, iroda. Egy egyed egyszerre kettőnél több egyedhalmazban is előfordulhat, egy számítógépes oktatáterem például tanterem és gépterem egyszerre. Az altípusok öröklik a főtípus attribútumait és rendelkezhetnek újabb, saját attribútumokkal. Egy konkrét előadóterem egyaránt része a Helyiség és Tanterem egyedhalmazoknak. A specializáló kapcsolat lényegében 1:1 kapcsolatot jelent egy főtípus és egy altípus között, de sajátos módon nem különböző egyedeket, hanem ugyanazon egyed két előfordulását kapcsolja össze.

kapcsolat: Ha valamely egyednek bizonyos

Az egyed kapcsolat diagram leképezése relációs modellre:

Egyed: Az egyedek leképezésénél a relációséma neve az egyed neve lesz, majd zárójelben felsoroljuk az attribútumait. A relációséma kulcsa az egyed kulcsa lesz.

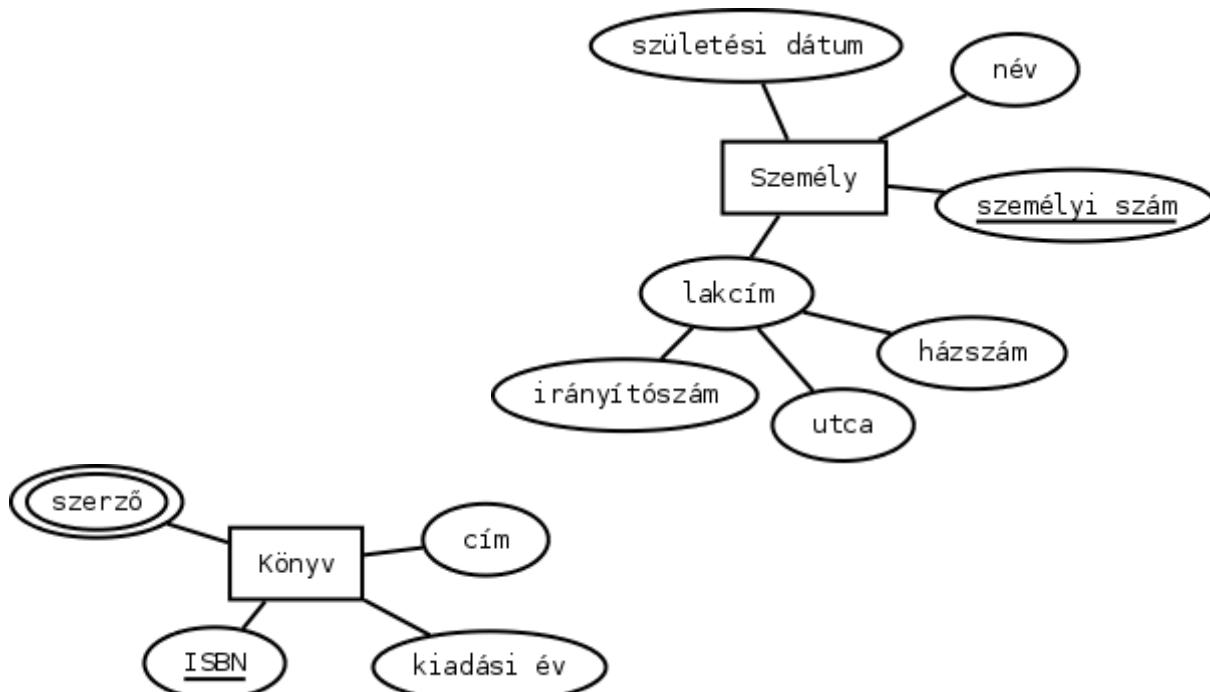
Személy(személyi szám, név, születési dátum)

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Összetett attribútum: Az összetett attribútumot a részattribútumaival helyettesítjük a leképezés során.

Személy (személyi szám, név, születési dátum, irányítószám, utca, házszám)

A többértékű attribútum leképezésének három lehetséges módja van:



- I. A többértékű attribútumot egy attribútumnak tüntetjük fel a relációsémában és értékeit, egy hosszú sztringként tároljuk. Ekkor az egyes elemekre nem lehet külön keresni. Nem jó megoldás.
Könyv (ISBN, cím, szerzők, kiadási év)
- II. A többértékű attribútumot egy attribútumnak tüntetjük fel a relációsémában és a táblában minden egyes értékéhez egy új sort veszünk fel. Nem jó megoldás.
Könyv (ISBN, cím, szerzők, kiadási év)
- III. A többértékű attribútum számára egy külön relációsémát hozunk létre, amelyben feltüntetjük az attribútumot és az egyed kulcsát (valamint, ha szükséges, akkor az attribútum értékének sorszámát is hozzávehetjük, ha az értékek sorrendje meghatározott).
Könyv (ISBN, cím, kiadási év)
Szerző (ISBN, szerző)
Könyv (ISBN, cím, kiadási év)
Szerző (ISBN, sorszám, szerző)

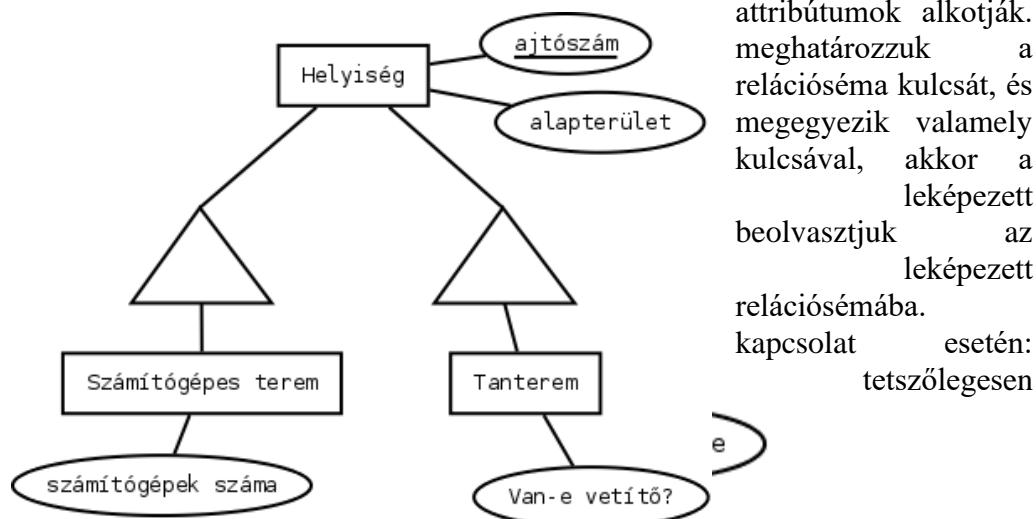
ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

A **kapcsolatot** leképezzük egy külön relációsémába oly módon, hogy a kapcsolat neve lesz a relációséma neve, a relációséma attribútumait pedig a kapcsolódó egyedek kulesai és a attribútumok alkotják.

Ezután leképezett ha az egyed kapcsolatból relációsémát egyedből

1:1



választhatunk, hogy melyik egyedhez ágyazzuk be.

1:N kapcsolat esetén: általában az N-oldali egyedből leképezett relációsémával vonható össze.

N:M kapcsolat esetén nem vonható össze egyik egyedből leképezett relációsémával sem.

Hallgató (EHA, név, szak)

Kurzus (kurzuskód, kurzus neve, férőhelyek száma)

Jelentkezett (EHA, kurzuskód)

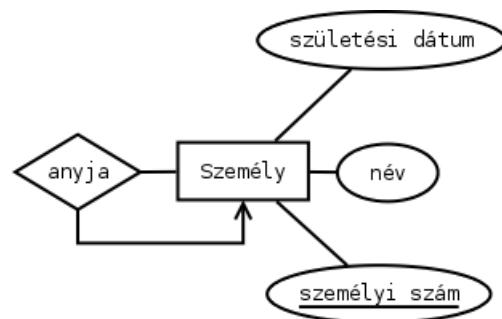
Rekurzív kapcsolatnál hasonlóképpen járunk el, mint a kapcsolat leképezésénél, de az egyedet két külön egyednek tekintjük.

Személy (személyi szám, név, születési dátum) Anyja(gyermek személyi száma, anya személyi száma)

Összevonás után:

Személy (személyi szám, név, születési dátum, anya személyi száma)

A **specializáló kapcsolatok** egy hierarchiát, "típusoltságot" (főtípush, altípush) jelölnek. Ezeket a kapcsolatokat kétféleképpen lehet leképezni. Lássuk az alábbi példát:



ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

I. Leképezzük a főtípus egyedet egy relációsémába, majd leképezzük az altípus egyedeiket egy-egy relációsémába úgy, hogy feltüntetjük a főtípus összes attribútumát.

Helyiség (ajtószám, alapterület)

Számítógépes terem (ajtószám, alapterület, számítógépek száma)

Tanterem (ajtószám, alapterület, van-e vetítő)

II. Leképezzük a főtípus egyedet egy relációsémába, majd leképezzük az altípus egyedeiket egy-egy relációsémába úgy, hogy a főtípus egyed kulcsát vesszük hozzá az egyedeik attribútumaihoz.

Helyiség (ajtószám, alapterület)

Számítógépes terem (ajtószám, számítógépek száma)

Tanterem (ajtószám, van-e vetítő)

Hogy melyik a jobb megoldás, arról lehet vitatkozni, leginkább azt kell végig gondolni, hogy a minden adatra vonatkoznak majd a leggyakoribb lekérdezések és módosítások, és mennyi adatot fognak tárolni ezek a táblák. Ha kevés beszúrás és módosítás érinti ezeket a táblákat és sok lekérdezést hajtunk rajtuk végre, akkor az első megoldás a jobb, mert többnyire elegendő lesz csak az egyik táblát beolvasni. Ha viszont - tegyük fel - lenne a helyiségekhez egy "kulcsot vitte", vagy utolsó "tűzvédelmi ellenőrzés időpontja", attribútum, akkor a második megoldás lenne célravezetőbb, mert akkor ezek az értékek csak Helyiség táblában játszanak fontos szerepet, viszonylag gyakran módosulnak, és akkor nem kellene minden táblában módosítani ezeket az értékeket. Ezen kívül a lekérdezésnél sem fontos az adott terem tanerem vagy számítógépes terem.

- Funkcionális függőség, a normalizálás célja, normálformák

Funkcionális függőség:

Legyen $R(A_1, \dots, A_n)$ egy relációséma, és P, Q az $\{A_1, \dots, A_n\}$ attribútumhalmaz részhalmazai. P -től funkcionálisan függ Q (jelölésben $P \rightarrow Q$), ha bármely R feletti T tábla esetén valahányszor két sor megegyezik P -n, akkor megegyezik Q -n is, vagyis bármely $t_i \in T$ és $t_j \in T$ esetén ha $t_i(P) = t_j(P)$ akkor $t_i(Q) = t_j(Q)$.

Elnevezések:

$A \rightarrow Q$ függést triviálisnak nevezzük, ha $Q \subseteq P$, ellenkező esetben nem triviális.

$A \rightarrow Q$ függést teljesen nemtriviálisnak nevezzük, ha $Q \cap P = \emptyset$.

A gyakorlatban általában teljesen nemtriviális függőségeket adunk meg.

Tekintsük a Dolgozói nyilvántartást és keressünk benne függőségeket!

| Név | Adószám | Cím | Osztálykód | Osztálynév | VezAdószám |
|--------|---------|------------------|------------|------------|------------|
| Kovács | 1111 | Pécs, Vár u. 5 | 2 | Tervezési | 8888 |
| Tóth | 2222 | Tata, Tó u. 2 | 1 | Munkaügyi | 3333 |
| Kovács | 3333 | Vác, Róka u. 1 | 1 | Munkaügyi | 3333 |
| Török | 8888 | Pécs, Sas u. 8 | 2 | Tervezési | 8888 |
| Kiss | 4444 | Pápa, Kő tér 2. | 3 | Kutatási | 4444 |
| Takács | 5555 | Győr, Pap u. 7. | 1 | Munkaügyi | 3333 |
| Fekete | 6666 | Pécs, Hegy u. 5. | 3 | Kutatási | 4444 |
| Nagy | 7777 | Pécs, Cső u. 25. | 3 | Kutatási | 4444 |

$\{osztálykód\} \rightarrow \{osztálynév, vezAdószám\}$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

{adószám} → {név, lakkím, osztálykód, osztálynév, vezAdószám}

Normalizálás célja: A relációs adatbázisok tervezésekor fontos tényező a redundancia csökkentése, vagyis, hogy ne tároljunk ismétlődő adatokat több példányban. Természetesen vannak helyzetek, amikor egyes adatokat nem érdemes külön táblában tárolni, mert az összetartozó adatok összekapcsolása nagyon sok számítással jár. Ezt minden esetben mérlegelni kell.

Az adatokat akkor kell normalizálni, ha

-egy adat gyakran fordul elő, és minden ugyanabban a formában kell bevinnünk/lekérdeznünk
-külön szeretnénk választani a kódokat/azonosítókat az információs adattól

Normálformák:

- 1NF: Egy $R(A,F)$ relációséma 1. normálformában van, ha az A attribútum-halmaz minden eleme atomi, vagyis nem többértékű és nem összetett attribútum.

Az összetett attribútumokat a részattribútumaikkal helyettesítjük.

A többértékű attribútumokat célszerűen új relációséma váltásával vesszük fel, amelyhez külső kulcsként hozzávesszük az őt tartalmazó relációséma kulcsát.

A továbbiakban feltételezzük, hogy minden relációséma teljesíti az 1NF feltételeit.

- 2NF: Legyen $R(A)$ relációséma, $X,Y \subseteq A$, és $X \rightarrow Y$. Azt mondjuk, hogy X -től teljesen függ Y , ha X -ból bármely attribútumot elhagyva a függőség már nem teljesül, vagyis bármely $X_1 \subset X$ esetén $X_1 \rightarrow Y$ már nem igaz. Megjegyzés: Ha K kulcs, akkor A teljesen függ K -tól.

Egy attribútumot elsődleges attribútumnak nevezünk, ha szerepel a relációséma valamely kulcsában, ellenkező esetben másodlagos attribútum. Vagyis, ha a séma kulcsai K_1, \dots, K_r , akkor $K = K_1 U \dots U K_r$ az elsődleges attribútumok halmaza, $A - K$ a másodlagos attribútumok halmaza. Egy relációséma 2NF-ben van, ha minden másodlagos attribútum teljesen függ bármely kulcsról.

Következmények: Ha minden kulcs egy attribútumból áll, akkor a séma 2NF-ben van. Ilyen például a

Dolgozó (név, adószám, cím, osztálykód, osztálynév, vezAdószám) tábla.

Ha a sémben nincs másodlagos attribútum, akkor 2NF-ben van. Ilyen például a Fuvar (gkvez, rendszám, indul, érkezik) tábla, mivel a következő kulcsok vannak: {gkvez, indul}, {gkvez, érkezik}, {rendszám, indul}, {rendszám, érkezik}.

2NF-re hozás

A sémet felbontjuk Heath tétele szerint, a normálformát sértő függőség mentén.

Ha valamely K kulcsra $L \subset K$ és $L \rightarrow B$ (itt B legyen az összes L -től függő attribútum halmaza), akkor a sémet felbontjuk az $L \rightarrow B$ függőség szerint. Legyen $C = A - (L \cup B)$, ekkor az $R(A)$ sémet az $R_1 (C \cup L)$ és $R_2 (L \cup B)$ sémekkel helyettesítjük. Heath tétele alapján a felbontás hűséges.

- 3NF: Legyen $X, Z \subseteq A$, és $X \rightarrow Z$. Azt mondjuk, hogy X -től tranzitívan függ Z , ha van olyan $Y \subseteq A$, amelyre $X \rightarrow Y$ és $Y \rightarrow Z$, de X nem függ Y -tól, és az $Y \rightarrow Z$ függés teljesen nemtriviális. Ellenkező esetben Z közvetlenül függ X -től. Megjegyzés: Az "X nem függ Y-tól" és az "Y → Z függés teljesen nemtriviális" kiegészítő feltételek nem csak a triviális esetek kiszűréséhez kellenek, hanem a későbbi állítások szempontjából is lényegesek.

Egy relációséma 3NF-ben van, ha minden másodlagos attribútuma közvetlenül függ bármely kulcsról.

Következmény: Ha a sémben nincs másodlagos attribútum, akkor 3NF-ben van.

3NF-re hozás: Ha másodlagos attribútumok egy B halmazára és valamely K kulcsra $K \rightarrow Y \rightarrow B$ tranzitív függés fennáll, akkor a sémet felbontjuk Heath tétele szerint az $Y \rightarrow B$ függés mentén.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

B legyen az összes Y-tól függő attribútum halmaza. Legyen C = A – (Y U B), ekkor az R(A) sémát az R1 (C U Y) és R2 (Y U B) sémákkal helyettesítjük. Heath tétele alapján a felbontás hűséges.

Példa:

http://www.inf.u-szeged.hu/~gnemeth/adatbgyak/exe/Normalizalas_web/plda_knyvtri_adatbzis.html

2. Tétel

- Az SQL adatbázisnyelvek: Az adatdefiníciós nyelv (DDL) és az adatmanipulációs nyelv (DML)

SQL = Structured Query Language (= strukturált lekérdező nyelv). A relációs adatbáziskezelés szabványos nyelve. Nem algoritmikus nyelv, de algoritmikus nyelvekbe beépíthető (beágazott SQL).

Az SQL utasításait két fő csoportba szokták sorolni:

- DDL (= Data Definition Language): adatstruktúra definiáló utasítások.
- DML (= Data Manipulation Language): adatokon műveletet végző utasítások.

Jelen anyagban - az RDBMS fő feladatai alapján - az alábbi csoportokban tárgyaljuk az SQL utasításokat:

- adatbázisséma definiálása (DDL),
- adatok aktualizálása (DML),
- lekérdezési lehetőségek (DML).

Szintaxis: Kisbetű és nagybetű a nyelv alapszavaiban egyenértékű. Utasítások sorfolytonosan írhatók, lezárás pontosvesszővel.

Változó nincs, csak tábla- és oszlopnevekre lehet hivatkozni. Kifejezésben hivatkozás egy tábla adott oszlopára: tábla.oszlop (ha a tábla egyértelmű, akkor elhagyható).

Alias név: név AS másodnév (egyes implementációkban AS elhagyható).

Szövegkonstans: 'szöveg'

Dátum: DATE '1968-05-12'. Egyes rendszerek az SQL szabványtól eltérő konvenciót alkalmaznak, például 13-NOV-94 (Oracle).

Idő: TIME '15:31:02.5' (óra, perc, másodperc).

Stringek konkatenációja: + vagy || .

Relációjelek: =, <=, >=, !=, <>

Logikai műveletek: AND, OR, NOT.

Az SQL rendszerek "háromértékű logikát" használnak, vagyis a TRUE és FALSE mellett a NULL (definiálatlan).

Speciális logikai kifejezések

x IS NULL: igaz, ha az x mező értéke NULL. Ez nem egyenértékű az "x = NULL" kifejezéssel, ugyanis ennek értéke definiálatlan, mivel definiálatlan komponenst tartalmaz.

x BETWEEN a AND b: igaz, ha a ≤ x ≤ b.

x IN halmaz: igaz, ha x megegyezik a megadott halmaz egy elemével. A halmazt explicit módon vagy lekérdezéssel lehet megadni. Példa: város IN ('Szeged','Szolnok','Pécs')

x relációjel ALL halmaz: igaz, ha x a halmaz minden elemével a megadott relációban van. Példa: fizetés != ALL (81000, 136000, 118000)

x relációjel ANY halmaz: igaz, ha a halmaznak van olyan eleme, amellyel x a megadott relációban van. Példa: fizetés < ANY (81000, 136000, 118000)

EXISTS halmaz: igaz, ha a halmaz nem üres.

x LIKE minta: igaz, ha az x karaktersorozat megfelel a megadott mintának. Ha a mintában "%" illetve "_" jel szerepel, az tetszőleges karaktersorozatot illetve tetszőleges karaktert jelent.

Példa: lakcím LIKE '%Vár u.%' igaz minden olyan lakcímre, amelyben szerepel a "Vár u." részlet. A fentiekben általában a NOT is használható, például x IS NOT NULL, x NOT IN halmaz, stb.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Az adattípusok (rendszerenként eltérők lehetnek):

CHAR(n) n hosszúságú karaktersorozat

VARCHAR(n) legfeljebb n hosszúságú karaktersorozat

INTEGER egész szám (röviden INT)

REAL valós (lebegőpontos) szám, másnéven FLOAT

DECIMAL(n[,d]) n jegyű decimális szám, ebből d tizedesjegy

DATE dátum (év, hó, nap)

TIME idő (óra, perc, másodperc)

Az adattípushoz "DEFAULT érték" megadásával alapértelmezett érték definiálható. Ha illet nem adunk meg, az alapértelmezett érték NULL.

- Relációsémák definiálása, megszorítások típusai és létrehozásuk

Relációsémák definiálása (DDL)

Relációséma létrehozására a CREATE TABLE utasítás szolgál, amely egyben egy üres táblát is létrehoz a sémához. Az attribútumok definiálása mellett a kulcsok és külső kulcsok megadására is lehetőséget nyújt.

```
CREATE TABLE táblanév  
  ( oszlopnev adattípus [feltétel],  
    ... ...,  
    oszlopnev adattípus [feltétel]  
    [, táblaFeltételek]  
  );
```

Feltételek (egy adott oszlopra vonatkoznak):

PRIMARY KEY:

elsődleges kulcs (csak egy lehet)

UNIQUE:

kulcs (több is lehet)

REFERENCES tábla(oszlop) [ON-feltételek]:

külső kulcs

Táblafeltételek:

PRIMARY KEY (oszloplista):

elsődleges kulcs

UNIQUE (oszloplista):

kulcs

FOREIGN KEY (oszlop) REFERENCES tábla(oszlop) [ON-feltételek]:

külső kulcs

A tábla módosításakor a definiált kulcsfeltételek automatikusan ellenőrzésre kerülnek. PRIMARY KEY és UNIQUE esetén ez azt jelenti, hogy a rendszer nem enged olyan módosítást illetve új sor felvételét, amely egy már meglévő kulccsal ütközne. REFERENCES (külső kulcs hivatkozás) esetén ON-feltételek megadásával szabályozhatjuk a rendszer viselkedését (jelölje T1 a hivatkozó és T2 a hivatkozott táblát):

- Alapértelmezés (ha nincs ON-feltétel): T1-ben nem megengedett olyan beszúrás és módosítás, amely T2-ben nem létező kulcs értékre hivatkozna, továbbá T2-ben nem megengedett olyan kulcs módosítása vagy sor törlése, amelyre T1 hivatkozik.

- ON UPDATE CASCADE: ha T2 egy sorában változik a kulcs értéke, akkor a rá való T1-beli hivatkozások is megfelelően módosulnak (módosítás továbbgyűrűzése).

- ON DELETE CASCADE: sortörlés továbbgyűrűzése.

- ON UPDATE SET NULL: ha T2 egy sorában változik a kulcs értéke, akkor T1-ben a rá való külső kulcs hivatkozások értéke NULL lesz.

- ON DELETE SET NULL: a hivatkozott kulcs értéke NULL lesz.

Megszorítások

Aktív elem: olyan programrész, amely bizonyos szituációban automatikusan végrehajtódik. Ennek speciális esete a megszorítás, ami bizonyos feltételek ellenőrzését jelenti bizonyos helyzetekben.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Attribútumok megszorításai: A CREATE TABLE-ben valamely attribútum deklarációja után adhatók meg. Kulcs feltételek: a CREATE TABLE utasításban adhatók meg a PRIMARY KEY, UNIQUE, REFERENCES kulcsszavakkal. Aktualizálási műveleteknél a megfelelő feltétel automatikus ellenőrzését váltják ki.

További megszorítások: NOT NULL: Adott attribútum értéke nem lehet NULL. CHECK (feltétel): Az adott attribútum módosítását a rendszer csak akkor engedi meg, ha a feltétel teljesül.

Példa: A dolgozók nemét is nyilvántartjuk (F=férfi, N=nő):

```
CREATE TABLE Dolgozó (
    adószám DECIMAL(10) PRIMARY KEY,
    nem CHAR(1) CHECK (nem IN ('F', 'N'))
);
```

Értéktartomány definiálása: CREATE DOMAIN név típus [DEFAULT érték] [CHECK (feltétel)];

Értéktartomány módosítása ALTER DOMAIN, törlése DROP DOMAIN utasítással történik.

Példa: CREATE DOMAIN NemÉrték CHAR(1) CHECK (VALUE IN ('F', 'N'));

```
CREATE TABLE Dolgozó (
    adószám DECIMAL(10) PRIMARY KEY,
    nem NemÉrték);
```

Táblára vonatkozó megszorítások: A CREATE TABLE végére, a táblaFeltételeknél helyezendők el. Kulcs feltételek: PRIMARY KEY, UNIQUE, FOREIGN KEY kulcsszavakkal. Ha a CHECK feltétel egyszerre több attribútumot érint, akkor szintén a táblafeltételeknél helyezendő el.

Példa: CREATE TABLE Könyv (
 könyvszám DECIMAL(6) PRIMARY KEY,
 kivétel DATE,
 vissza DATE,
 CHECK (kivétel < vissza));

Általános megszorítások: Több táblára (általában, a teljes adatbázissémára) vonatkozhatnak.

Megadásuk: CREATE ASSERTION név CHECK (feltétel);

A feltételben szereplő táblák bármelyikének módosításakor a feltétel ellenőrzésre kerül.

Példa: CREATE ASSERTION VezetőFizetés
 CHECK (NOT EXISTS
 (SELECT * FROM Dolgozó, Osztály
 WHERE Dolgozó.adószám = Osztály.vezAdószám
 AND fizetés < 100000));

A feltétel két esetben sérülhet: ha egy dolgozó fizetését változtatjuk, vagy ha egy dolgozót vezetőnek nevezünk ki. Ezért a fenti önálló megszorítás nem helyettesíthető egyetlen táblára vonatkozó megszorítással.

Az önálló megszorítás törlése: DROP ASSERTION név;

A megszorításokat célszerű elnevezni a "CONSTRAINT név" előtag segítségével.

Például a Dolgozó tábla név attribútuma esetén:

név CHAR(30) CONSTRAINT NévKulcs UNIQUE

Ezután a kulcsfeltétel elvethető a következő utasítással:

```
ALTER TABLE Dolgozó DROP CONSTRAINT NévKulcs;
```

A kulcsfeltétel újra érvényesíthető táblafeltételként:

```
ALTER TABLE Dolgozó ADD CONSTRAINT NévKulcs UNIQUE (név);
```

Értéktartományra vonatkozó megszorítás esetén:

```
CREATE DOMAIN NemÉrték AS CHAR(1) CONSTRAINT FérfiVagyNő CHECK (VALUE IN ('F', 'N'));
```

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Értéktartományra vonatkozó megszorítás hasonlóan módosítható:

ALTER DOMAIN NemÉrték DROP CONSTRAINT FérfiVagyNő;

A trigger egy aktualizáció művelet esetén végrehajtandó programrészletet definiál:

CREATE TRIGGER név

{ BEFORE | AFTER | INSTEAD OF }

{ DELETE | INSERT | UPDATE [OF oszlopok] }

ON tábla

[REFERENCING [OLD AS régi] [NEW AS új]

[FOR EACH ROW]

[WHEN (feltétel)] programblokk;

név: a trigger neve.

BEFORE, AFTER, INSTEAD OF: az aktualizáció művelet előtt, után, vagy helyette lép működésbe.

DELETE, INSERT, UPDATE OF: az aktualizáció művelet neve.

ON tábla: ezen tábla aktualizálásakor lép működésbe a trigger.

REFERENCING: lehetővé teszi, hogy a tábla aktuális sorának aktualizálás előtti és utáni állapotára névvel hivatkozzunk.

FOR EACH ROW: ha megadjuk, akkor a trigger a tábla minden egyes sorára lefut, amelyet az aktualizáció művelet érint (sor szintű trigger). Ha nem adjuk meg, akkor egy aktualizáció művelet esetén csak egyszer fut le a trigger (utasítás szintű trigger).

WHEN feltétel: a megadott feltétel teljesülése esetén hajtódkik végre a trigger. programblokk: egy vagy több SQL utasításból álló, vagy valamely programozási nyelven írt blokk.

Példa sor szintű triggerre. Az alábbi trigger egy FizetésNapló (dátum, adószám, régifiz, újfiz) táblában gyűjti a fizetés-módosítások adatait:

CREATE TRIGGER fiz_napló

AFTER UPDATE OF fizetés ON Dolgozó

REFERENCING OLD AS régi NEW AS új

FOR EACH ROW

INSERT INTO FizetésNapló

VALUES (SYSDATE, régi.adószám,

régi.fizetés, új.fizetés);

A trigger engedélyezett vagy letiltott állapotban lehet. Létrehozáskor engedélyezett, változtatás ALTER TRIGGER utasítással lehetséges

- Adatmanipulációs lehetőségek és lekérdezések

Relációséma törlése:

DROP TABLE táblanév;

Relációséma módosítása:

ALTER TABLE táblanév

[ADD (újelem, ..., újelem)]

[MODIFY (módosítás, ..., módosítás)]

[DROP (oszlop, ..., oszlop)];

újelem: egy "oszlopnév adattípus [feltétel]", vagy egy "táblafeltétel", mint a CREATE TABLE-nél.

módosítás: "oszlopnév adattípus [feltétel]".

Például ALTER TABLE Dolgozó MODIFY (lakcím VARCHAR(60));

Adattábla aktualizálása (DML)

A táblába új sor felvétele az

INSERT INTO táblanév [(oszloplista)] VALUES (értéklista);

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

utasítással történik. Ha oszloplista nem szerepel, akkor valamennyi oszlop értéket kap a CREATE

TABLE-ben megadott sorrendben. Egyébként csak az oszloplistában megadott mezők kapnak értéket, a többi mező értéke NULL lesz.

Sor(ok) módosítása az

UPDATE táblanév
SET oszlop = kifejezés, ..., oszlop = kifejezés
[WHERE feltétel];

utasítással történik. Az értékadás minden olyan soron végrehajtódik, amely eleget tesz a WHERE feltételnek. Ha WHERE feltétel nem szerepel, akkor az értékadás az összes sorra megtörténik.

Sor(ok) törlése a

DELETE FROM táblanév
[WHERE feltétel];

utasítással lehetséges. Hasonlóképp működik, mint az UPDATE.

Lekérdezés (DML)

Lekérdezésre a SELECT utasítás szolgál, amely egy vagy több adattáblából egy eredménytáblát állít elő. Az eredménytábla a képernyőn listázásra kerül, vagy más módon használható fel.

SELECT [DISTINCT] oszloplista
FROM táblanévlista
[WHERE feltétel];

A "SELECT DISTINCT A1,...,An FROM T1,...,Tm WHERE feltétel" utasítás egyenértékű a következő relációs algebrai kifejezéssel:

$E = \pi_{A1, \dots, An}(\sigma_{\text{feltétel}}(T1 \times \dots \times Tm))$

Vagyis, a felsorolt táblák Descartes-szorzatából szelektáljuk a feltételnek eleget tévő sorokat, majd ezekből projekcióval választjuk ki az E eredménytábla oszlopait. A DISTINCT opciót akkor kell kiírni, ha az eredménytáblában az azonos sorokból csak egyet kívánunk megtartani. Ha oszloplista helyére * karaktert írunk, ez valamennyi oszlop felsorolásával egyenértékű. A SELECT legegyszerűbb változatával adattábla listázását érhetjük el: SELECT * FROM T;

Projekció:

SELECT [DISTINCT] A1,...,An FROM T;

Példa: SELECT DISTINCT szerző, cím FROM Könyv;

Szelekció:

SELECT * FROM T WHERE feltétel;

Példa: SELECT * FROM Könyv WHERE kivétel < 2013.01.01;

Descartes-szorzat: T1 x T2

SELECT * FROM T1,T2;

Természetes összekapcsolás. Állítsuk elő például az Áru (cikkszám, megnevezés) és Vásárlás (cikkszám, mennyiség) táblák természetes összekapcsolását:

SELECT Áru.cikkszám, megnevezés, mennyiség
FROM Áru, Vásárlás
WHERE Áru.cikkszám = Vásárlás.cikkszám;

A fentivel egyenértékű, szintén gyakran használt szintaxis:

SELECT Áru.cikkszám, megnevezés, mennyiség
FROM Áru INNER JOIN Vásárlás ON Áru.cikkszám = Vásárlás.cikkszám;

Külső összekapcsolás. A fenti példát alapul véve, ha az eredménytáblában valamennyi áru adatait szerepeltetni szeretnénk, akkor ez az alábbi módon adható meg:

SELECT Áru.cikkszám, megnevezés, mennyiség
FROM Áru, Vásárlás
WHERE Áru.cikkszám (+)= Vásárlás.cikkszám;

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Az SQL szabvány szerint a LEFT, RIGHT vagy FULL OUTER JOIN kulcsszavakkal adható meg külső összekapcsolás, például:

```
SELECT Áru.cikkszám, megnevezés, mennyiség  
FROM Áru LEFT OUTER JOIN Vásárlás  
ON Áru.cikkszám = Vásárlás.cikkszám;
```

Théta join (két tábla Descartes-szorzatából tetszőleges feltétel szerint választunk sorokat):

```
SELECT * FROM T1,T2 WHERE feltétel;
```

Unió/metszet/különbség (két SELECT eredménytáblája kompatibilis kell, hogy legyen):

```
(SELECT * FROM T1) UNION|INTERSECT|EXCEPT (SELECT * FROM T2);
```

A SELECT után megadott oszloplista valójában nem csak oszlopneveket, hanem tetszőleges kifejezéseket is tartalmazhat, és az eredménytábla oszlopainak elnevezésére alias neveket adhatunk meg:

Példa: a Raktár(cikkszám, név, egységár, mennyiség) táblából egy E(áru, érték) tábla létrehozása:

```
SELECT név AS áru, egységár*mennyiség AS érték FROM Raktár;
```

A FROM után megadott táblák esetén is használhatók alias nevek, és erre szükség is van akkor, ha egy táblának önmagával való összekapcsolását képezzük.

Függvények

ABS(n), LOWER(char), UPPER(char)

LTRIM(char), RTRIM(char): balról és jobbról szóközök eltávolítása.

SUBSTR(char, m[, n]): a char string m-edik karakterétől n hosszú részstringet ad vissza.

TO_CHAR(n): konverzió numerikusról vagy dátumról karakteresre.

TO_DATE(char): konverzió karakteresről dátumra.

TO_NUMBER(char): konverzió karakteresről numerikusra.

Összesítő függvények

Egy oszlop értékeiből egyetlen értéket hoznak létre (például átlag). Általános alakjuk:

függvénynév ([DISTINCT] oszlopnév)

A számításnál a NULL értékek figyelmen kívül maradnak.

Az egyes függvények: AVG, SUM, MAX, MIN, COUNT.

Példa: SELECT AVG(fizetés) FROM Dolgozó

Csoportosítás (GROUP BY, HAVING)

Ha a tábla sorait csoportonként szeretnénk összesíteni, akkor a SELECT utasítás a GROUP BY oszloplista alparancsal bővítendő. Egy csoportba azok a sorok tartoznak, melyeknél oszloplista értéke azonos.

Az eredménytáblában egy csoportból egy rekord lesz. Az összesítő függvények csoportonként hajtódnak végre.

Csoportosítási szabály: A SELECT után összesítő függvényen kívül csak olyan oszlopnév tüntethető fel, amely a GROUP BY-ban is szerepel!

Példa: A Dolgozó táblából osztályonként az átlagfizetést számoljuk. Az eredménytáblának annyi sora lesz, ahány osztály van: SELECT osztkód, AVG(fizetés) FROM Dolgozó GROUP BY osztkód;

A GROUP BY által képezett csoportok közül válogathatunk a HAVING feltétel

alparancs segítségével: csak a feltételnek eleget tevő csoportok kerülnek összesítésre az eredménytáblába.

Példa: Azon osztályok lista, ahol az átlagfizetés > 180 000 Ft:

```
SELECT osztkód, AVG(fizetés) FROM Dolgozó  
GROUP BY osztkód  
HAVING AVG(fizetés) > 180000;
```

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Az eredménytábla rendezése

Bár a relációs modell nem definiálja a rekordok sorrendjét, a gyakorlatban rendszerint valamilyen rendezettségen kívánjuk látni az eredményt. Erre szolgál az

ORDER BY oszlopnév [DESC], ..., oszlopnév [DESC]

alparancs, amely a SELECT utasítás végére helyezhető, és az eredménytáblának a megadott oszlopok szerinti rendezését írja elő. Az oszlopnév után írt ASC (ascending) növekvő (default), DESC (descending) csökkenő sorrendben való rendezést jelent.

A SELECT utasítás általános alakja

```
SELECT [DISTINCT] oszloplista projekció
FROM táblanévlista Descartes-szorzat
[WHERE feltétel] szelekció
[GROUP BY oszloplista csoportonként összevonás
[HAVING feltétel] ] csoport-szelekció
[ORDER BY oszloplista]; rendezés
```

Az SQL nyelv ismertetésének elején láttunk halmazokat tartalmazó logikai kifejezéseket. Egy ilyen halmaz SELECT utasítással is előállítható, például a

'Tóth Pál' IN (SELECT név FROM Dolgozó WHERE osztálykód='015')

logikai kifejezés akkor igaz, ha Tóth Pál a 015 kódú osztály dolgozója.

Ha egy SELECT utasítás WHERE vagy HAVING feltételében olyan logikai kifejezés szerepel, amely SELECT utasítást tartalmaz, ezt alkérdeznek vagy belső SELECT-nek is nevezik. Általában, valamely SQL utasítás belsejében szereplő SELECT utasítást alkérdeznek nevezünk. Példa: Az alábbi utasítás azon dolgozók listáját adja, amelyek fizetése kisebb, mint az átlagfizetés:

```
SELECT név, fizetés FROM Dolgozó
WHERE fizetés < ( SELECT AVG(fizetés) FROM dolgozó );
```

Példa. A Dolgozó(név, cím, osztálykód, fizetés) táblából azon dolgozók listáját kérjük, akiknek az osztályon belül a legnagyobb a fizetése (ha több ilyen van, mindegyiket ki kell listázni). A Dolgozó tábla két példányát a D1 és D2 alias nevek különböztetik meg:

```
SELECT osztálykód, név, fizetés FROM Dolgozó AS D1
WHERE fizetés = ( SELECT MAX(fizetés) FROM Dolgozó AS D2
WHERE D1.osztálykód = D2.osztálykód );
```

Az első példánál csak egyszer értékelődik ki az alkérdezés, míg a másodiknál minden külső SELECT-es rekordra egyszer.

Ügyelni kell a típuskompatibilitásra: a belső SELECT által visszaadott érték típusa egyezzen meg a WHERE utáni mező típusával.

Bizonyos esetekben az alkérdezés join-műveletet helyettesít:

```
SELECT szerző, cím FROM Könyv WHERE olvasószám IN
(SELECT olvasószám FROM Olvasó WHERE lakkód LIKE '%Pécs%');
SELECT szerző, cím FROM Könyv, Olvasó
WHERE Könyv.olvasószám = Olvasó.olvasószám AND lakkód LIKE '%Pécs%';
```

Nem csak SELECT utasításban alkalmazható alkérdezés:

```
UPDATE Dolgozó SET fizetés=fizetés+10000
WHERE adószám IN ( SELECT adószám FROM Projekt WHERE pkód='A12' );
INSERT INTO Készlet SELECT név, egységár*mennyiség FROM Raktár;
```

- A lekérdezések megvalósítása beágyazott SQL utasításokkal

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Beágyazott SQL Az SQL lehetőségeivel nem oldható meg minden adatbázis kezelési feladat. SQL-ben például nem használhatók változók és vezérlési szerkezetek, így az adatbázis algoritmikus kezelése sem lehetséges. Ezért az SQL utasításokat általában egy hagyományos algoritmikus programnyelv (C, Java, stb.) utasításaival keverten használjuk, és az SQL utasításokban felhasználhatók a befogadó programnyelv változói is. Ezt a megoldást nevezzük beágyazott SQL-nek (embedded SQL).

Jellemző megoldási módok:

- Precompiler alkalmazása, amely a forráskódban felismeri az SQL utasításokat, és lecseréli azokat a befogadó nyelv függvényhívásaira (például Oracle Pro*C).
- Az SQL nyelvet algoritmikus lehetőségekkel bővíti. Itt valójában nincs befogadó nyelv, az algoritmikus nyelv és az SQL szerves egységet képez. (Ilyen például az Oracle rendszer PL/SQL nyelve.)
- A befogadó nyelvben beágyazott SQL utasítások helyett csak a nekik megfelelő függvényhívások használhatók (például ODBC, JDBC, PHP).

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Digitális képfeldolgozás

1. Tétel

- Simítás/ szűrés képtérben (átlagoló szűrők, Gauss simítás és mediánszűrés)

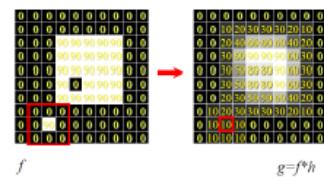
- Egy képen különböző zajok fordulhatnak elő, melyeket szükséges lehet korrigálni:



- Szűrés a környezet súlyozott átlagával:

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

- Átlagolás: a környezetbe eső valamennyi pont egyforma súlyal számít
- Súlyozott átlagolás: a környezet intenzitásaihoz (általában a távolsággal arányosan csökkenő) súlyokat rendelünk
- zajszűrő/simító maszkoknál a maszelemek összege minden 1!
- minél nagyobb a környezet, annál inkább elmosunk minden a szűrővel



- Átlag szűrő:

$$g(i, j) = \frac{1}{|S_{(i,j)}|} \cdot \sum_{m, n \in S_{(i,j)}} f(m, n),$$

ahol f a kiindulási kép, g a szűrt kép, $S_{(i,j)}$ az (i, j) pont környezete, $|S_{(i,j)}|$ pedig a környezetbe tartozó képpontok száma

- Átlag szűrés 3x3-mas környezetre, konvolúcióval:

$$\begin{aligned} g(i, j) &= \frac{1}{9} \cdot \sum_{u=-1}^1 \sum_{v=-1}^1 f(i+u, j+v) \\ &= \sum_{u=-1}^1 \sum_{v=-1}^1 f(i+u, j+v) \cdot h(u, v) \\ &= (f * h)(i, j), \end{aligned}$$

ahol:

$$h = \frac{1}{9} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

a konvolúciós maszk.

- Átlagszűrés hatása:

- a képpontok közelebb kerülnek a környezet átlagához, a kép simább lesz
- a szűrt kép intenzitásértékei a kiindulási kép intenzitástartományában maradnak
- lineáris operátor (a konvolúció is az)
- csökkenti a zajt (főleg gauss-zaj)
- kára: gyengíti az éleket, homályosít

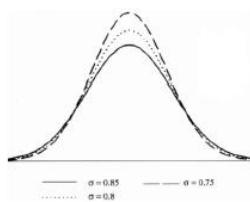
- Módosított környezeti átlagolás: csak akkor simítunk, ha az adott képpont intenzitásának a környezeti átlagtól való eltérése meghalad egy előre megadott T küszöbértéket:

$$g'(i, j) = \begin{cases} g(i, j), & \text{ha } |g(i, j) - f(i, j)| > T, \\ f(i, j), & \text{különben} \end{cases}$$

- Gauss függvény:

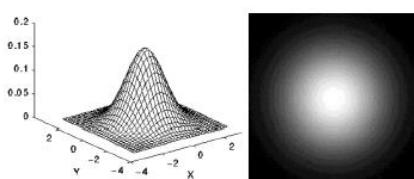
– 1D:

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



– 2D:

$$\begin{aligned} G_\sigma(x, y) &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \\ G_\sigma(x, y) &= G_\sigma(x) \cdot G_\sigma(y) \end{aligned}$$



ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- Gauss-szűrés:

$$h(x, y) = f(x, y) * G_\sigma(x, y),$$

ahol h a simított fgv, $*$ pedig a konvolúció jele (az egész anyag során)

- A Gauss-függvényt a képfeldolgozásban diszkrét értékekkel közelítjük, így határozzuk meg a szűrőmaszkokat:

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

- Minél nagyobb a maszk, annál homályosabb a kép, minél nagyobb σ , annál erősebb simítás
- **Medián szűrés:** az $a_1, a_2, \dots, a_{2n+1}$ számok mediánja a nagyság szerint rendezett számsorozat középső ($n + 1$ -edik) eleme, jelölés:

$$\text{med}\{a_1, a_2, \dots, a_{2n+1}\}$$

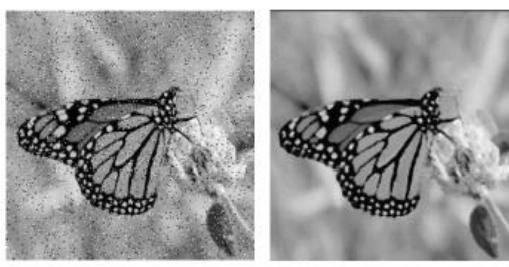
- Medián tulajdonságai:

- $\min\{a_i\} \leq \text{med}\{a_i\} \leq \max\{a_i\}$
- nem lineáris, de
- $\text{med}\{a_i + c\} = \text{med}\{a_i\} + c$
- $\text{med}\{c \cdot a_i\} = c \cdot \text{med}\{a_i\}$

- Mediánszűrés:

$$g(i, j) = \text{med}\{f(i + u, j + v) | (u, v) \in S\}$$

- A mediánszűrés eredményét az S környezet mérete (és alakja) határozza meg



só-bors zajos

medián szűrt

- Élek detektálása (gradiens-operátorokkal és Marr-Hildreth módszerrel)

- **Pont detektálás:** A feladat kiugró inétnizitású pontokat keresni, amely az alábbi maszkkal jól elvégezhető (ezzel a maszkkal történő konvolúció 0 értéket ad homogén területen, nagy abszolútértéket a kiugró helyeken):

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- **Vonal detektálás:** Célja az egységesi vastagságú görbek kijelölése. A probléma megoldható az adott irányú vonalpontokra érzékeny g_1, \dots, g_n maszkokkal történő konvolúcióval:

$$l(i, j) = \max\{|(f * g_1)(i, j)|, \dots, |(f * g_n)(i, j)|\}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix}$$

0° 45°

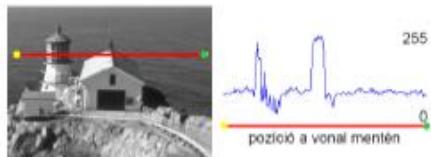
$$\begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix} \quad \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

90° 135°

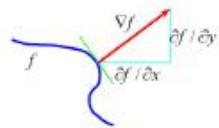
ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

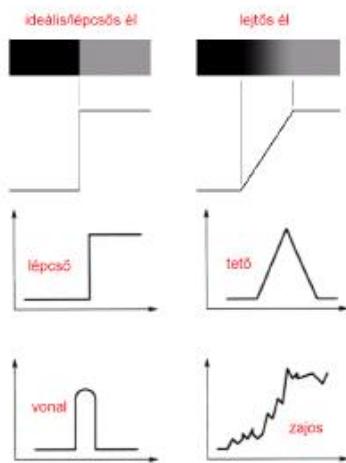
- **Éldetektálás:** A képen ott található él, ahol a képfüggvény valamely irány mentén hirtelen változik:
- Gradiens: $\nabla f = \nabla f(x_1, \dots, x_n) = (\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n})^T$



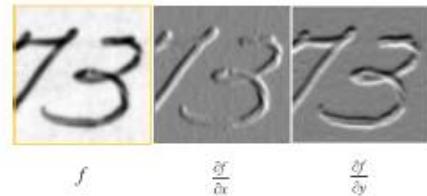
$$\nabla f = \nabla f(x_1, \dots, x_n) = (\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n})^T$$



- Tipikus él/intenzitásprofilok:



- Gradiens komponensek:



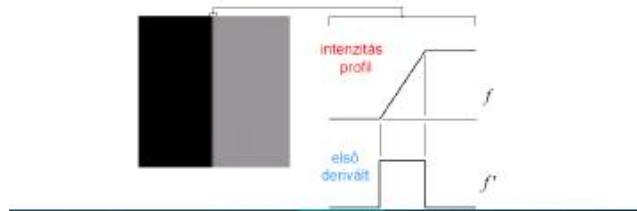
- Gradiens nagysága (magnitúdója):

$$|\nabla f(x_1, \dots, x_n)| = \|\nabla f(x_1, \dots, x_n)\|_2 = \sqrt{\sum_{i=1}^n \left(\frac{\partial f}{\partial x_i}\right)^2}$$

$$\approx \|\nabla f(x_1, \dots, x_n)\|_1 = \sum_{i=1}^n \left| \frac{\partial f}{\partial x_i} \right|$$

$|\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$

- Intenzitásprofilok elsőrendű deriváltja:



- Gradiens irány:

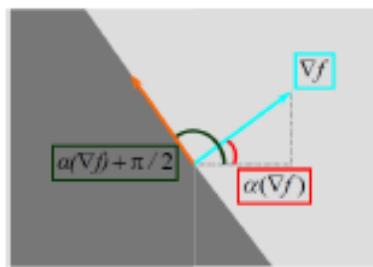
$$\alpha(\nabla f(x, y)) = \arctan \left(\frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}} \right)$$

$\alpha(\nabla f) = \arctan \left(\frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}} \right)$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- El iránya (merőleges a gradiensre):



- Diszkrét gradiens operátorok:

 - Roberts

$$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$\frac{\partial f}{\partial x} \qquad \frac{\partial f}{\partial y}$

könnyen számítható, de zajérzékeny

 - Prewitt

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$\frac{\partial f}{\partial x} \qquad \frac{\partial f}{\partial y}$

- Gradiens maszk tervezése x-irányban:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Feltételek:

1. Szimmetria: $a_{ij} = a_{ji}$ (ha $j = 1, 2, 3$)
2. Antiszimmetria: $a_{ii} = -a_{jj}$, $a_{ii} > 0$ és $a_{ij} = 0$
3. Nem reagál konstansra: $\sum_{i=1}^3 \sum_{j=1}^3 a_{ij} = 0$

Ezen feltételeket kielégítő maszk:

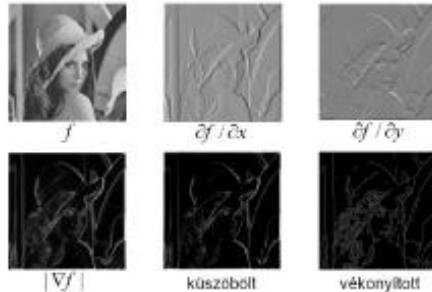
$$\begin{bmatrix} p & 0 & -p \\ q & 0 & -q \\ p & 0 & -p \end{bmatrix}$$

- Prewitt: $p = 1, q = 1$

- Sobel: $p = 1, q = 2$

- Frei-Chen: $p = 1, q = \sqrt{2}$

- Gradiens alapú éldetektálás lépései:



 - Sobel

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$\frac{\partial f}{\partial x} \qquad \frac{\partial f}{\partial y}$

simító hatása van

 - Frei-Chen (izotropikus)

$$\begin{bmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}$$

$\frac{\partial f}{\partial x} \qquad \frac{\partial f}{\partial y}$

- 8 irányban élő kereső gradiens operátorok:

 - Prewitt

$$\begin{bmatrix} 1 & 1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & -1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ 1 & -2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & 1 \\ 1 & -2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

E NE N NW

$$\begin{bmatrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & -1 \\ 1 & 1 & -1 \end{bmatrix}$$

W SW S SE

 - Robinson (3 elemű)

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

NE N NW

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix}$$

SW S SE

 - Robinson (5 elemű)

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

NE N NW

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix}$$

SW S SE

 - Kirsh

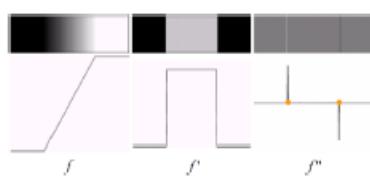
$$\begin{bmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{bmatrix} \quad \begin{bmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{bmatrix} \quad \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{bmatrix} \quad \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{bmatrix}$$

NE N NW

$$\begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix} \quad \begin{bmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{bmatrix} \quad \begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \quad \begin{bmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}$$

SW S SE

- A másodrendű derivált az előjelváltást vizsgálja:



- A Laplace-operátor egy lineáris differenciál-operátor a másodrendű derivált közelítésére (a gradiens operátor önmagával vett belső szorzata):

$$\nabla^2 = \nabla \cdot \nabla = \left[\frac{\partial}{\partial x_1}, \dots, \frac{\partial}{\partial x_n} \right] \cdot \begin{bmatrix} \frac{\partial}{\partial x_1} \\ \vdots \\ \frac{\partial}{\partial x_n} \end{bmatrix} = \sum_{i=1}^n \frac{\partial^2}{\partial x_i^2}$$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

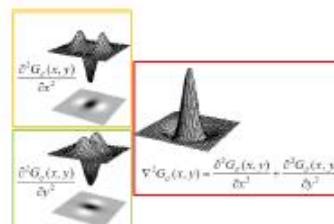
- Kétváltozós fgy Laplace-operátora:

$$\nabla f(x, y) = \frac{\delta^2 f(x, y)}{\delta x^2} + \frac{\delta^2 f(x, y)}{\delta y^2}$$

- Laplace-operátor diszkrét közelítése:

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -1 & 0 \\ 0 & 2 & 0 \\ 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

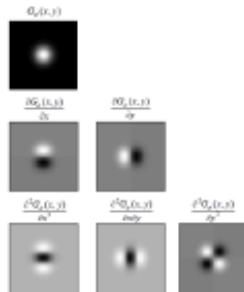
$$\approx \frac{\delta^2 f(x, y)}{\delta x^2} \quad \approx \frac{\delta^2 f(x, y)}{\delta y^2} \quad \approx \nabla^2 f(x, y)$$



- Laplace-operátor tulajdonságai:

- forgásinvariáns
- egyetlen maszzal számítható
- csak magnitúdó számolható vele
- duplán érzékelhet éleket
- zajérzékeny

- 2D Gauss-függvény deriváltjai:



- A LoG diszkrét közelítése:

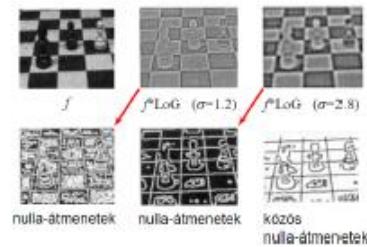
$$\nabla^2 G \approx \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

- Konvolúció LoG függvényivel:

$$\nabla^2(G * f) = (\nabla^2 G) * f$$

- Marr-Hildreth éldetektor:

- Konvolváljuk a képet egy (vagy több) alkalmas LoG függvényel
- Keressük (közös) nulla-átmeneteket (ilyenek ott vannak, ahol egy adott pozitív pont kis környezetében pozitív és negatív értékek is előfordulnak)



- 2D Gauss-függvény Laplace transzformáltja (LoG):

2. Tétel

- Alakreprezentáció

- Az alakreprezentáció fő módszerei:

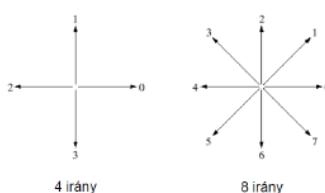
- Az objektumot körülvevő határ leírása
- Az objektum által elfoglalt régió leírása
- **Transzformációs** megközelítés

- Határvonal alapú tulajdonságok:

- lánckód, alakleíró szám
- kerület, terület, kompaktság, cirkularitás
- közelítés poligonnal
- parametrikus kontúr, határvonal leíró fgy
- meredekségi hisztogram
- görbület, energia
- struktúrális leírás

- **Lánckód:** Az alakzat határpontjait követi, láncolja az óramutató járásával ellenálléster irányban

- Határpont: Az alakzatnak olyan pontja, melynek van az alakzathoz nem tartozó 8 ill. 4 szomszédja:

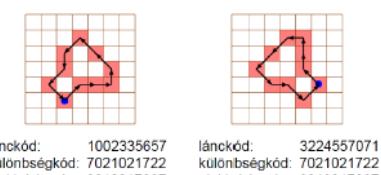


- Normalizált lánckód: Különböző kezdőpontok választása a lánckód ciklikusan permutált változatait eredményezi

- **Különbözkód:** a lánckód első deriváltja, a szomszédos elemek közötti elmozdulások száma

- **Normalizálás:** addig permutáljuk ciklikusan a különbözkódöt, amíg a legkisebb értékű kódot (a legkisebb 4-es ill. 8-as számrendszerbeli számot) kapjuk.

- **Alakleíró szám:** a normalizált különbözkód (nem függ a kezdőpont választásától)



Invariáns a forgatásra, ha a forgatási szög $k \cdot \pi/2$.

- Példa 8-as lánckódra:

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- Lánckód előnyei:

- kompakt
- eltolás-invariáns
- gyors algoritmus
- gyorsan rekonstruálható belőle az alakzat

- Hártrányok:

- nem forgás-invariáns
- nem skála-invariáns
- a pontosság legfeljebb pixelnyi lehet
- zajérzékeny

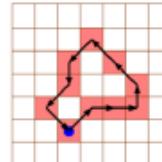
Kerület számítása (8-as) lánckódóból:

```

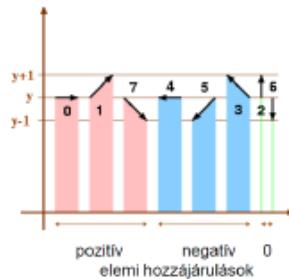
lánckód: 1 0 0 2 3 3 5 6 5 7
rend (a lánckód hossza): 10
páros elemek száma: 4
páratlan elemek száma: 6
kerület: 12.485

```

$$\text{kerület} = \frac{1}{2} \cdot (\text{páros elemek száma}) + \sqrt{2} \cdot (\text{páratlan elemek száma})$$



Terület számítása (8-as) lánckódóból:

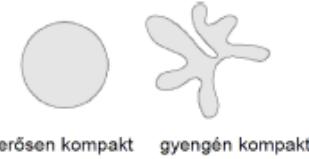


```

area = 0;
y = 0;
for each element code
    if code=0 then Δarea=y;      y=y;
    if code=1 then Δarea=y+0.5;  y=y+1;
    if code=2 then Δarea=0;      y=y+1;
    if code=3 then Δarea=-(y+0.5); y=y+1;
    if code=4 then Δarea=-y;     y=y;
    if code=5 then Δarea=-(y-0.5); y=y-1;
    if code=6 then Δarea=0;      y=y-1;
    if code=7 then Δarea=y-0.5;  y=y-1;
    area = area + Δarea;
endfor
area = abs(area);

```

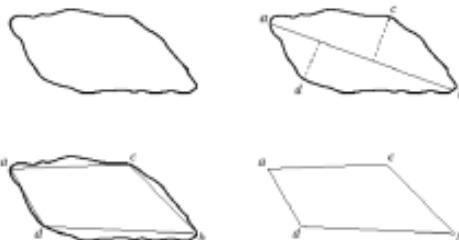
Kompaktság: kerület²/terület



- Cirkularitás: $1/\text{kompaktság} = \text{terület}/\text{kerület}^2$

- Maximális a körre: $1/4\pi \approx 0.08$

- Közelítés poligonnal:



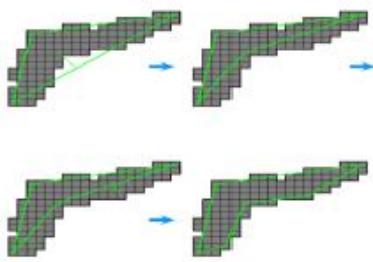
a kezdeti közelítést adó négyzet konstruálása

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

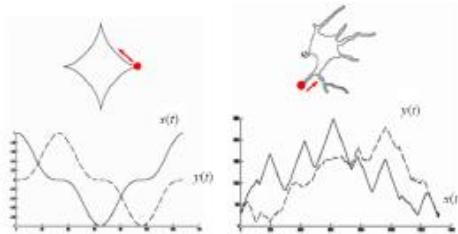
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- Határ- és régió-alapú alakleíró jellemzők

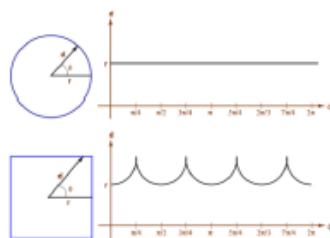
- Határfelosztás:



- Parametrikus kontúr: $(x(t), y(t))$

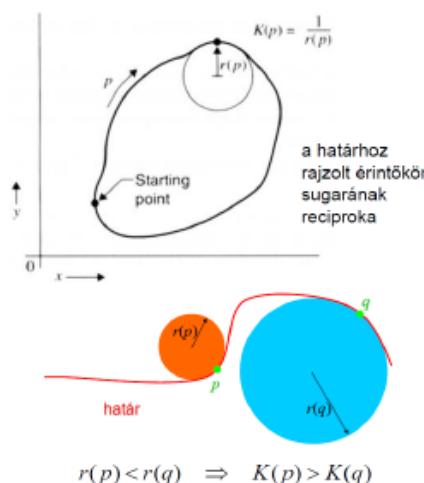


- Leírás egy változós fgv-ekkel: Pl. súlypontnak a határtól vett távolságát a szög függvényében fejezzük ki. Mivel ez az érték függ az alakzat méretétől és a határon vett kezdőpont megválasztásától, ezért a jellemző normalizálásra szorul



- Meredekség - a meredekség változása: az $s(t)$ görbével adott határ t pontjában a meredekség $\frac{ds}{dt}$

- Görbület - a görbület változása:



- Energia:

- A határ energiája:

$$E = \frac{1}{L} \int_0^L K(p)^2 dp,$$

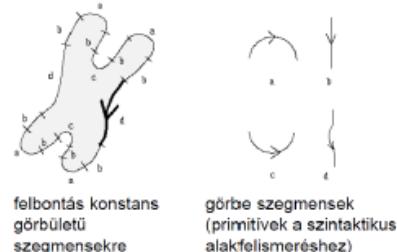
ahol L a határ hossza

- Rögzített terület mellett a kör energiája a minimális:

$$E = \left(\frac{1}{r}\right)^2,$$

ahol r a kör sugara.

- Strukturális leírás:



- Leírás generatív nyelvtanokkal:

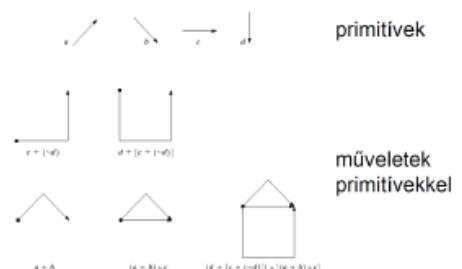
lépcső és kódolt szerkezet:



a CF-nyelvtan:

$$G = (\{S, A\}, \{a, b\}, \{S \rightarrow aA, A \rightarrow bS, A \rightarrow b\}, S)$$

- Leírás összefűzött primitívekkel:



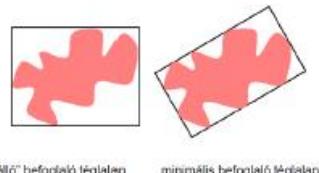
műveletek
primitívekkel

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

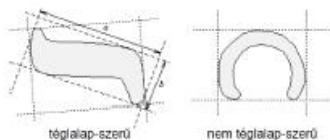
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- Régióalapú leírás:

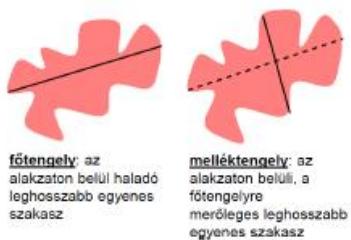
- befoglaló téglalap, rektangularitás
 - főtengely, melléktengely, átmérő, excentricitás, főtengely szöge
 - konvex burok, konvex kiegészítés, konkavitási fa, particionált határ
 - vetületek, törés-költség
 - topológiai leírások, Euler-szám, szomszedsági fa
 - váz
 - momentumok, invariáns momentumok
- Befoglaló téglalap:



- Rektangularitás: az alakzat területének és a minimális befoglaló téglalap területének a hányszáosa



- Fő- és melléktengely:

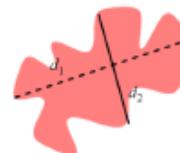


- Átmérő:

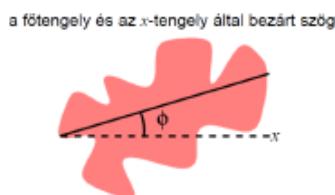
$$Diam(S) = \max_{p, q \in \partial S} \{d(p, q)\},$$

ahol:

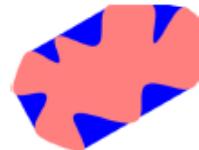
- S : alakzat,
 - ∂S : alakzat határa,
 - d : távolság
- Excentricitás: a fő- és a melléktengely hosszaránya: d_1/d_2



- Főtengely szöge (alakzat irányába):



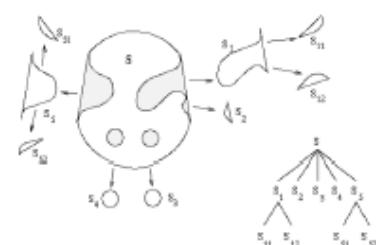
- Konvex burok: az alakzatot tartalmazó minimális konvex alakzat



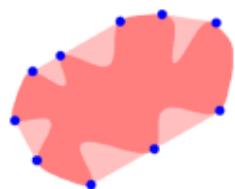
- Konvex kiegészítés: a konvex burok és az alakzat különbsége

- Konkavitási fa:

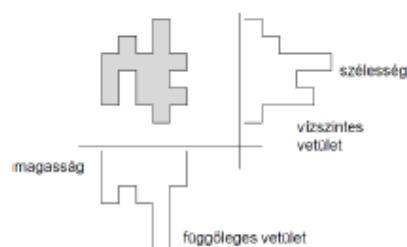
- A fa gyökere a kiindulási alakzat, az első szinten a konvex különbség alakzatai helyezkednek el, melyekre a faépítést rekurzív módon folytatjuk
- A fa elágazási pontjaiban lévő alakzatok nem konvexek, míg minden levélalakzat konvex



- Particionált határ: a határ partcionálható szerint, hogy hol kezdődik, ill. fejeződik be a konvex kiegészítés valamely komponense



Vetületek:



a függőleges vetület minimumai használhatók pl karakterek elkülönítésére
Törés költség: ennek meghatározásakor az egik vetületet vesszük figyelembe (pl. függőleges). Ekkor egy oszlop költsége a benne található olyan egyesek száma, melyekhez az előző oszlop ugyanazon sorában is egyes található

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

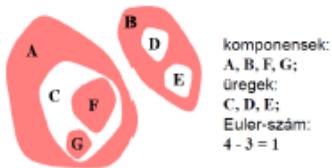
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

| | |
|--------------------------------------|-------------|
| | bináris kép |
| 0 4 3 1 0 2 4 3 1 0 törés költség | |

- Topológiai lefrás:

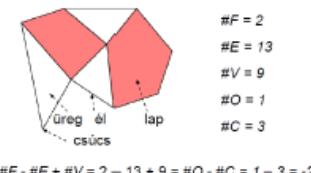
- Bináris kép: kétfélé érték (1: fekete, alakzat, komponens; 0: fehér, lyuk, háttér)
- Komponens: maximálisan összefüggő fekete halmaz (bármely két pontja összeköthető a halmazon belül 4-, ill. 8-tíittel) régió
- Üreg: a komplement/negált kép egy véges komponense

- Euler-féle szám: #(komponensek)-#(üregek)

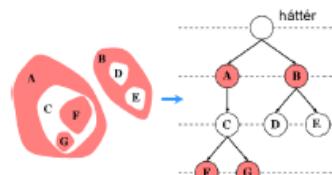


- Euler szám poligonhálózatokra:

- lapok: #F
- élek: #E
- csúcsok: #V
- régiók, objektumok: #O
- üregek: #C
- Euler szám: #F - #E + #V = #O - #C



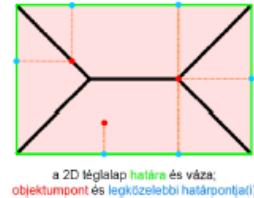
- Összefüggőségi fa: a bináris képekre rendelt irányított gráf, ahol
 - minden egyes szögpontról megfelel a kép egy fehér vagy fekete komponensének
 - A gráf tartalmazza az (X, Y) életet, ha az X komponens körülveszi a vele szomszédos Y komponensem



Az Euler szám nem kódolja a képeket!

- Váz: gyakran alkalmazott régió-alapú alaklefrő jellemző, mely leírja az objektumok általános formáját

- A váz a középtengely transzformáció eredménye: a vízat az objektum azon pontjai alkotják, melyekre kettő, vagy több legközelebbi határpontról található



- Prérít fiz-hasonlat: Az objektum határát minden pontjában egyidejűleg felgyűjtük. A víz azokból a pontokból áll, ahol a tűzfrontok találkoznak és kioltják egymást (tfh a tűzfrontok minden irányban egyenletes sebességgel: izotropikusan terjednek).

- A vázat az objektumba beírható maximális nyílt hipergömbök középpontjai alkotják. Egy beírható hipergömb maximális, ha öt nem tartalmazza egyetlen másik beírható hipergömb sem.

- A váz:

- Reprezentálja az objektum általános formáját, topológiai szerkezetét, lokális objektum szimmetriákat
- Invariáns az eltolásra, elforgatásra, uniform skálázásra
- Egyszerűbb szerkezet (vékony, csökkenti a dimenziót)

- Vázszervű jellemzők:

- Középfelszín
- Középvonal
- Topológiai mag

- Középvonal pontjai: vonal-pont, elágazási-pont, vég-pont

- Momentumok: előnyei:

- számok
- többszintű képekre is értelmezhetők
- invariánsak a fontosabb geometriai transzformációkra

Az I kép $(p+q)$ -adrendű centrális momentumai:
 $(p,q=0,1,2,\dots)$: $m_{p,q} = \sum_x \sum_y x^p \cdot y^q \cdot I(x,y)$

Az S bináris alakzat $(p+q)$ -adrendű centrális momentumai:

$$m_{p,q} = \sum_{(x,y) \in S} x^p \cdot y^q$$

Súlypont:
 $(\bar{x}, \bar{y}) = \left(\frac{m_{1,0}}{m_{0,0}}, \frac{m_{0,1}}{m_{0,0}} \right)$

- Centrális momentumok:

Az I kép $(p+q)$ -adrendű centrális momentumai:

$$\mu_{p,q} = \sum_x \sum_y (x - \bar{x})^p \cdot (y - \bar{y})^q \cdot I(x,y)$$

Az S bináris alakzat $(p+q)$ -adrendű centrális momentumai:

$$\mu_{p,q} = \sum_{(x,y) \in S} (x - \bar{x})^p \cdot (y - \bar{y})^q$$

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

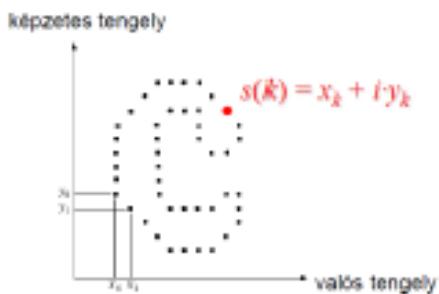
- Fourier leírás

- **Transzformáció alapuló alakleírás:**

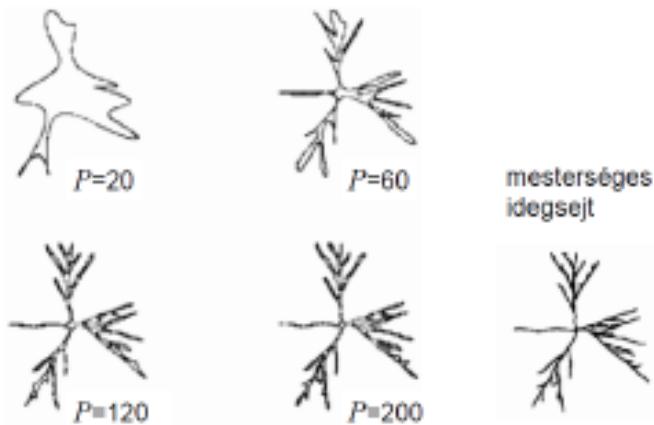
- multi-skálás transzformációk
- Fourier leírás

- Fourier leírás:

- Transzformáljuk a határ K darab mintavételezett pontjából (mint komplex $s(k)$ számokból) képzett s vektort. Az eredményül kapott a vektor (komplex $a(k)$ együtthatók) adják a Fourier leírást.
- Az alakzat rekonstrukciójához az inverz Fourier-transzformációt kell végrehajtani.



- kevés mintavételezési pont felhasználásával is nagyon jó közelítését tudjuk meghatározni az eredeti objektumnak:



ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Programozás alapjai

1. Tétel

- Algoritmusok vezérlési szerkezetei és megvalósításuk C programozási nyelven

Az algoritmus vezérlése:

Az az előírás, amely az algoritmus minden lépésére (részműveletére) kijelöli, hogy a lépés végrehajtása után melyik lépés végrehajtásával folytatódjék (esetleg fejeződjék be) az algoritmus végrehajtása.

Az algoritmusnak, mint műveletnek a vezérlés a legfontosabb komponense.

Vezérlési szerkezetnek nevezünk egy olyan programnyelvi konstrukciót, ami az összetevői végrehajtási sorrendjét adja meg. A elemei elemiutasítások, vagy további kialakítva a programrész vezérlési

Négy fő vezérlési szerkezetet

- Szekvenciális:* Véges sok adott sorrendben egymás után
- Szelekciós/elágazásos:* Véges közül adott feltétel alapján végrehajtása.
- Ismétléses/iterációs:* Adott szerinti ismételt végrehajtása.
- Eljárás:* Adott művelet alkalmazása adott argumentumokra, ami az argumentumok értékének pontosan meghatározott változását eredményezi.

Sok programozási nyelvben nincs rá eszköz, de bizonyos párhuzamos programozási helyzetek igényelhetnek további szerkezeteket, például a párhuzamos végrehajtás, kölcsönös kizáras, egy vagy több eseményre várakozás.

Az algoritmusok leírására többféle módszer használható:

- Természetes nyelvi leírás:* Nagyon távol állhat a „géptől”.
- Pszeudokód:* „Majdnem” programozási nyelv, strukturált, de sokkal szabadabb, mint egy valódi programozási nyelv.
- Folyamatábra:* Grafikus, kevésbé strukturált (tetszőleges vezérlésátadás), a működési folyamatra koncentrál.
- Szerkezeti ábra:* Grafikus, strukturált, az algoritmus felépítését és a működési folyamatot is leírja.

- A szekvenciális, iterációs, elágazásos, és az eljárás vezérlés

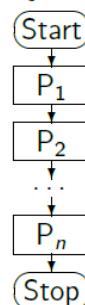
Szekvenciális vezérlés

Szekvenciális vezérlésről akkor beszélünk, amikor a P probléma megoldását úgy kapjuk, hogy a problémát P₁, . . . , P_n részproblémákra bontjuk, majd az ezekre adott megoldásokat (rézalgoritmusokat) sorban egymás után hajtjuk végre. P₁, . . . , P_n lehetnek elemi műveletek, vagy nem elemi részproblémák megnevezései. Utóbbi esetben a részproblémát tovább kell bontani.

Megvalósítása:

```
{  
    P1;  
    ...  
    Pn;  
}
```

Folyamatábrája:



vezérlési szerkezetek lehetnek, felépítését.

különböztetünk meg: művelet rögzített történő végrehajtása. sok rögzített művelet valamelyik

művelet adott feltétel

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Példa

Eltelt idő kiszámítása

Problémavezetés: Számítsuk ki a nap két időpontja között eltelt időt.

Specifikáció

- Perc pontossággal dolgozunk.

- A bemenő adat két időpont óra és perc formában, jelöljük ezeket O1, P1 illetve O2, P2-vel.

- A bemeneti feltétel:

$$0 \leq O_1 < 24 \text{ és } 0 \leq P_1 < 60$$

$$0 \leq O_2 < 24 \text{ és } 0 \leq P_2 < 60$$

$$O_1 < O_2 \text{ vagy } (O_1 = O_2 \text{ és } P_1 \leq P_2)$$

- A kimenő adat a két bemenő időpont között eltelt idő óra, perc formában, jelöljük ezeket O és P-vel.

- A kimeneti feltétel:

$$0 \leq O < 24 \text{ és } 0 \leq P < 60$$

Számolhatunk kétféleképpen:

-külön kiszámítjuk az eltelt órákat és perceket, de akkor a végén esetleg korrigálni kell a számítást a negatív percek miatt, vagy

-eleve percben számolunk, és csak a kiíratáskor alakítjuk át az eredményt óra:perc formátumra.

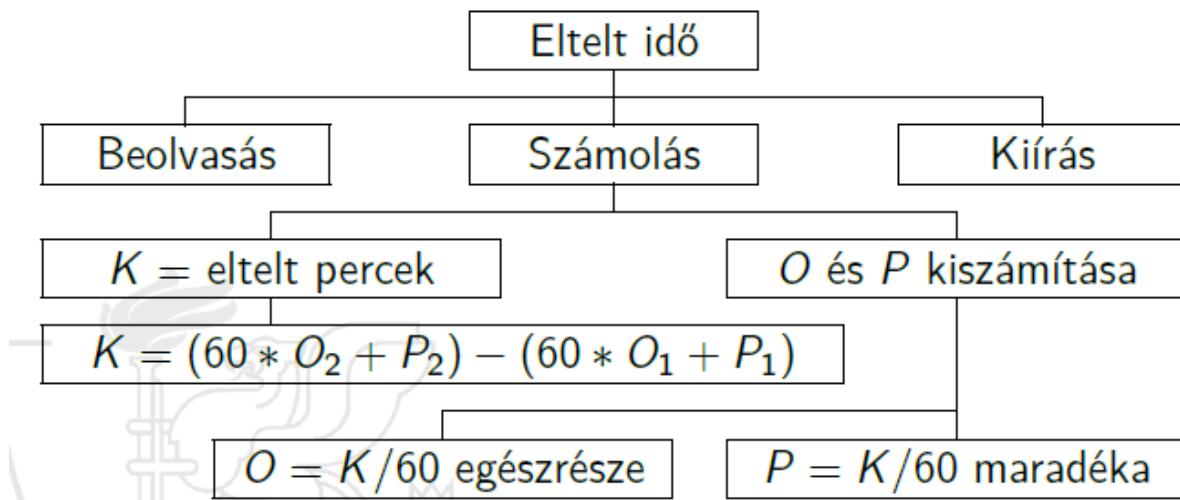
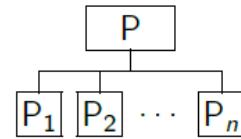
Az eltelt idő percben kifejezve $(60 \cdot O_2 + P_2) - (60 \cdot O_1 + P_1)$.

Tehát O, P akkor és csak akkor megoldás, ha

$$60 \cdot O + P = (60 \cdot O_2 + P_2) - (60 \cdot O_1 + P_1), \text{ és}$$

$$0 \leq P < 60$$

Az eltelt idő probléma megoldásának szerkezeti ábrája:



Az algoritmusban használt összes változó egész típusú, értékük tetszőleges egész szám lehet.

Szelekciós/elágazásos vezérlés

Szelekciós vezérlésről akkor beszélünk, amikor véges sok rögzített művelet közül véges sok feltétel alapján választjuk ki, hogy melyik művelet kerüljön végrehajtásra.

Típusai:

- Egyszerű szelekciós vezérlés
- Többszörös szelekciós vezérlés

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- Esetkválasztásos szelekciós vezérlés

- A fenti három „egyébként” ággal

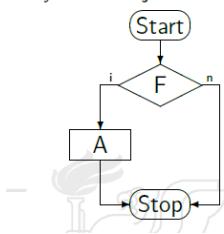
Egyszerű szelekciós vezérlés:

Egyszerű szelekció esetén egyetlen feltétel és egyetlen művelet van (ami persze lehet összetett).

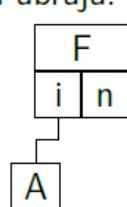
Legyen F logikai kifejezés, A pedig tetszőleges művelet. Az F feltételből és az A műveletből képzett egyszerű szelekciós vezérlés a következő vezérlési előírást jelenti:

1. Értékeljük ki az F feltételt és folytassuk a 2.) lépéssel.
2. Ha F értéke igaz, akkor hajtsuk végre az A műveletet és fejezzük be az összetett művelet végrehajtását.
3. Egyébként, vagyis ha F értéke hamis, akkor fejezzük be az összetett művelet végrehajtását.

Folyamatábrája:



Szerkezeti ábrája:



• Megvalósítás:

```

if (F) {
  A;
}
  
```

+ else ággal

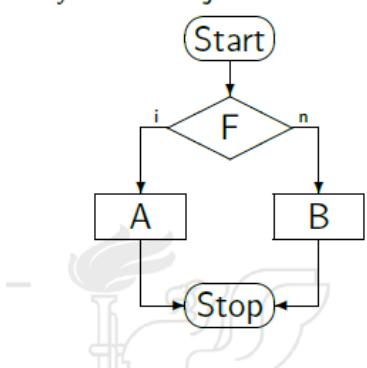
Egyszerű szelekció esetén egyetlen feltétel és egyetlen művelet van (ami persze lehet összetett).

Legyen F logikai kifejezés, A pedig tetszőleges művelet.

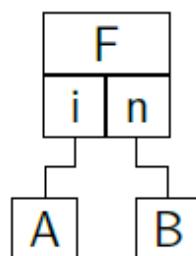
Az F feltételből és az A műveletből képzett egyszerű szelekciós vezérlés a következő vezérlési előírást jelenti:

1. Értékeljük ki az F feltételt és folytassuk a 2.) lépéssel.
2. Ha F értéke igaz, akkor hajtsuk végre az A műveletet és fejezzük be az összetett művelet végrehajtását.
3. Egyébként, vagyis ha F értéke hamis, akkor fejezzük be az összetett művelet végrehajtását.

Folyamatábrája:



Szerkezeti ábrája:



• Megvalósítás:

```

if (F) {
  A;
} else {
  B;
}
  
```

Többszörös szelekciós vezérlés

Ha több feltételünk és több műveletünk van, akkor többszörös szelekcióról beszélünk.

Legyenek F_i logikai kifejezések, A_i pedig tetszőleges műveletek $1 \leq i \leq n$ -re. Az F_i feltételekből és A_i műveletekből képzett többszörös szelekciós vezérlés a következő vezérlési előírást jelenti:

1. Az F_i feltételek sorban történő kiértékelésével adjunk választ a következő kérdésre: Van-e olyan i ($1 \leq i \leq n$), amelyre teljesül, hogy az F_i feltétel igaz és az összes F_j ($1 \leq j < i$) feltétel hamis?

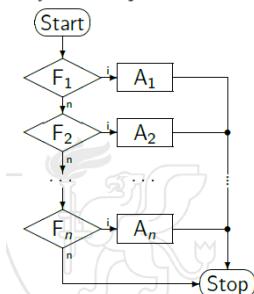
2. Ha van ilyen i , akkor hajtsuk végre az A_i műveletet és fejezzük be az összetett művelet végrehajtását.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

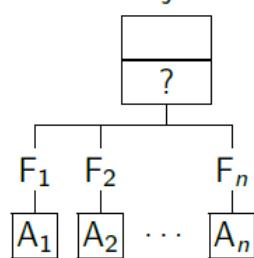
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

3. Egyébként, vagyis ha minden F_i feltétel hamis, akkor fejezzük be az összetett művelet végrehajtását.

Folyamatábrája:



Szerkezeti ábrája:



Megvalósítás:

```

if (F1) {
  A1;
} else if (F2) {
  A2;
...
} else if (Fn) {
  An;
}
  
```

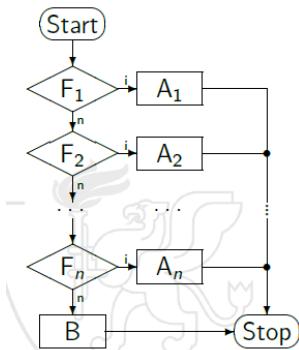
+ else ággal

A többszörös szelekció is bővíthető egyébként ággal úgy, hogy egy nemüres B műveletet hajtunk végre a 3. lépésben.

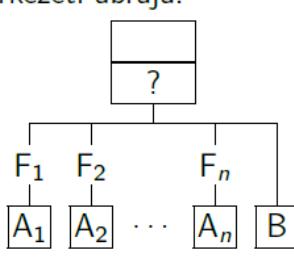
Legyenek F_i logikai kifejezések, A_i és B pedig tetszőleges műveletek $1 \leq i \leq n$ -re. Az F_i feltételekből, A_i és B műveletekből képzett többszörös szelekciós vezérlés a következő vezérlési előírást jelenti:

1. Az F_i feltételek sorban történő kiértékelésével adjunk választ a következő kérdésre: Van-e olyan i ($1 \leq i \leq n$), amelyre teljesül, hogy az F_i feltétel igaz és az összes F_j ($1 \leq j < i$) feltétel hamis?
2. Ha van ilyen i , akkor hajtsuk végre az A_i műveletet és fejezzük be az összetett művelet végrehajtását.
3. Egyébként, vagyis ha minden F_i feltétel hamis, akkor hajtsuk végre B -t és fejezzük be az összetett művelet végrehajtását.

Folyamatábrája:



Szerkezeti ábrája:



Megvalósítás:

```

if (F1) {
  A1;
} else if (F2) {
  A2;
...
} else if (Fn) {
  An;
} else {
  B;
}
  
```

C nyelvben nincs külön utasítás a többszörös szelekció megvalósítására, ezért az egyszerű szelekció ismételt alkalmazásával kell azt megvalósítani. Ez azon az összefüggésen alapszik, hogy a többszörös szelekció levezethető egyszerű szelekciók megfelelő összetételével.

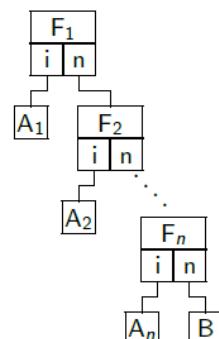
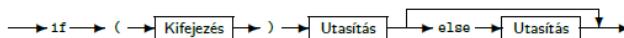
Esetkválasztásos szelekciós vezérlés

Ha a többszörös szelekciós vezérlésben minden F_i feltételünk $K \in Hi$ alakú, akkor esetkválasztásos szelekcióról beszélünk.

Legyen K egy adott típusú kifejezés, legyenek Hi ilyen típusú halmazok, A_i pedig tetszőleges műveletek $1 \leq i \leq n$ -re. A K szelektor kifejezésből, Hi

Ha valamelyen feltétel alapján egyik vagy másik utasítást akarjuk végrehajtani.

Az **if** utasítás szintaxisa C-ben



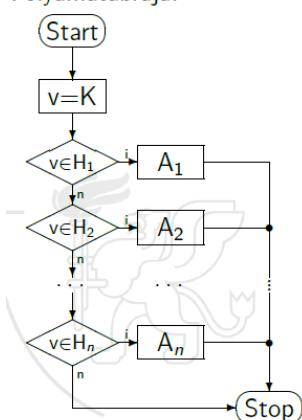
ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

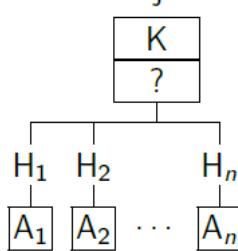
kiválasztó halmazokból és A_i műveletekből képzett esetkiválasztásos szelekciós vezérlés a következő vezérlési előírást jelenti:

1. Értékeljük ki a K kifejezést és folytassuk a 2.) lépéssel.
2. Adjunk választ a következő kérdésre: Van-e olyan i ($1 \leq i \leq n$), amelyre teljesül, hogy a $K \in H_i$, és $K \notin H_j$ ($1 \leq j < i$)?
3. Ha van ilyen i , akkor hajtsuk végre az A_i műveletet és fejezzük be az összetett művelet végrehajtását.
4. Egyébként, vagyis ha K nem eleme egyetlen H_i halmaznak sem, akkor fejezzük be az összetett művelet végrehajtását.

Folyamatábrája:



Szerkezeti ábrája:



• Megvalósítás:

```

switch(K) {
  case H1:
    A1;
    break;
  ...
  case Hn:
    An;
    break;
}
  
```

+ else ággal

A többszörös szelekció is bővíthető egyébként ággal úgy, hogy egy nemüres B műveletet hajtunk végre a 4. lépésben.

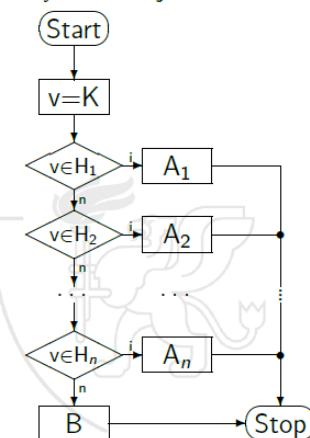
Legyen K egy adott típusú kifejezés, legyenek H_i ilyen típusú halmazok, A_i és B pedig tetszőleges műveletek $1 \leq i \leq n$ -re. A K szelektor kifejezésből, H_i kiválasztó halmazokból és A_i és B műveletekből képzett esetkiválasztásos szelekciós vezérlés a következő vezérlési előírást jelenti:

1. Értékeljük ki a K kifejezést és folytassuk a 2.) lépéssel.
2. Adjunk választ a következő kérdésre: Van-e olyan i ($1 \leq i \leq n$), amelyre teljesül, hogy a $K \in H_i$, és $K \notin H_j$ ($1 \leq j < i$)?
3. Ha van ilyen i , akkor hajtsuk végre az A_i műveletet és fejezzük be az összetett művelet végrehajtását.
4. Egyébként, vagyis ha K nem eleme egyetlen H_i halmaznak sem, akkor hajtsuk végre B -t és fejezzük be az összetett művelet végrehajtását.

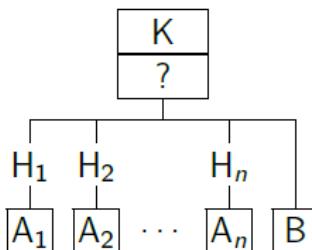
ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Folyamatábrája:



Szerkezeti ábrája:



• Megvalósítás:

```

switch(K) {
  case H1:
    A1;
    break;
  ...
  case Hn:
    An;
    break;
  default:
    B;
    break;
}
  
```

- A H_i halmazok elemszáma tetszőleges lehet, a `case`-ek után viszont csak egy-egy érték állhat.
- Kihasználhatjuk viszont a `switch` működési elvét, miszerint a címkek csak a belépési pontot határozzák meg. Ha $H_i = \{x_{i,1}, x_{i,2}, \dots, x_{i,n_i}\}$, akkor az adott ág lekódolható a

```

case x_{i,1}:
case x_{i,2}:
...
case x_{i,n_i}:
  A_i;
  break;
  
```

kódrészlettel.

Feltételes kifejezés

A feltételes operátor a C nyelv egyetlen háromoperandusú művelete.

`kif1 ? kif2 : kif3`

Először a `kif1` kerül kiértékelésre, ha

- igaz (nem 0) akkor a kifejezés értéke `kif2` lesz
- hamis (0) akkor a kifejezés értéke `kif3` lesz

Ismétléses/iterációs vezérlések

Ismétléses vezérlésben olyan vezérlési előírást értünk, amely adott műveletnek adott feltétel szerinti ismételt végrehajtását írja elő.

Az ismétlési feltétel szerint ötféle ismétléses vezérlést különböztetünk meg:

- Kezdőfeltételes
- Végfeltételes
- Számlálásos
- Hurok
- Diszkrét

Kezdőfeltételes

Kezdőfeltételes vezérlésről beszélünk, ha a ciklusmag (ismételt) végrehajtását egy belépési (ismétlési) feltételhez kötjük.

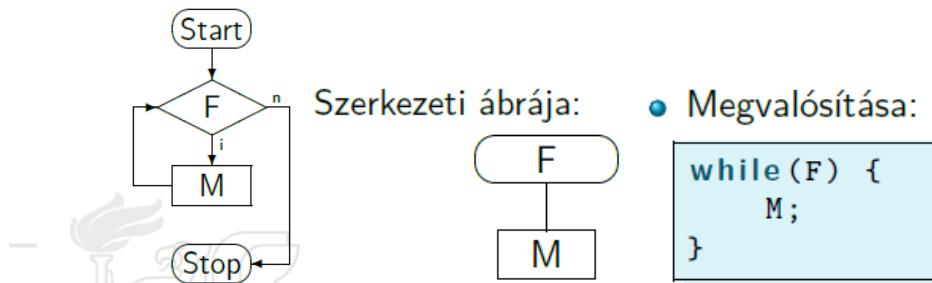
ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Legyen F logikai kifejezés, M pedig tetszőleges művelet. Az F ismétlési feltételből és az M műveletből (a ciklusmagból) képzett kezdőfeltéles ismétléses vezérlés a következő vezérlési előírást jelenti:

1. Értékeljük ki az F feltételt és folytassuk a 2.) lépéssel.
2. Ha F értéke hamis, akkor az ismétlés és ezzel együtt az összetett művelet végrehajtása befejeződött.
3. Egyébként, vagyis ha az F értéke igaz, akkor hajtsuk végre az M műveletet, majd folytassuk az 1.) lépéssel.

Folyamatábrája:



Ha az F értéke kezdetben hamis, M egyszer sem kerül végrehajtásra.

Ha az F értéke kezdetben igaz, és az M művelet nincs hatással F-re, akkor végtelen ciklust kapunk.

- **Ha valamilyen műveletet mindaddig végre kell hajtani, amíg egy feltétel igaz a `while` utasítás (is) használható.**
 - A művelet végrehajtása nem szükséges a feltétel kiértékeléséhez.
 - A feltétel ellenőrzése a művelet előtt történik, így ha a feltétel kezdetben hamis volt, a műveletet egyszer sem hajtjuk végre.
- **A `while` utasítás szintaxisa C-ben**



Végfeltételes

Végfeltételes vezérlésről akkor beszélünk, ha a ciklusmag elhagyását egy kilépési feltételhez kötjük.

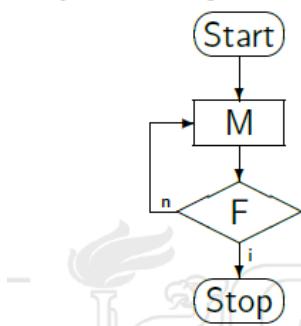
Legyen F logikai kifejezés, M pedig tetszőleges művelet. Az F kilépési feltételből és az M műveletből (a ciklusmagból) képzett végfeltételes ismétléses vezérlés a következő vezérlési előírást jelenti:

1. Hajtsuk végre az M műveletet majd folytassuk a 2.) lépéssel.
2. Értékeljük ki az F feltételt és folytassuk a 3.) lépéssel.
3. Ha F értéke igaz, akkor az ismétlés vezérlés és ezzel együtt az összetett művelet végrehajtása befejeződött.
4. Egyébként, vagyis ha az F értéke hamis, akkor folytassuk az 1.) lépéssel.

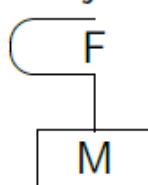
ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Folyamatábrája:



Szerkezeti ábrája:



• Megvalósítása:

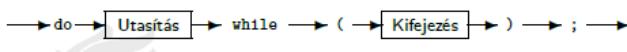
```

do {
  M;
} while (!F);
  
```

Ha az F értéke kezdetben hamis, és az M művelet nincs hatással F-re, akkor végtelen ciklust kapunk.

Ha az F értéke kezdetben igaz, M legalább egyszer akkor is végrehajtásra kerül.

- Ha valamelyen műveletet mindaddig végre kell hajtani, amíg egy feltétel igaz a do while utasítás (is) használható.
 - A művelet végrehajtása szükséges a feltétel kiértékeléséhez.
 - A feltétel ellenőrzése a művelet után történik, így ha a feltétel kezdetben hamis volt, a műveletet akkor is legalább egyszer végrehajtjuk.
- A do while utasítás szintaxisa C-ben



Számlálásos vezérlések

Számlálásos ismétléses vezérlésről akkor beszélünk, ha a ciklusmagot végre kell hajtani egy változó sorban minden olyan értékére (növekvő vagy csökkenő sorrendben), amely egy adott intervallumba esik.

Legyen a és b egész érték, i egész típusú változó, M pedig tetszőleges művelet, amelynek nincs hatása a, b és i értékére.

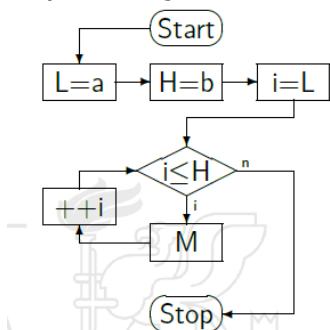
Növekvő

Az a és b határértékekből, i ciklusváltozóból és M műveletből (ciklusmagból) képzett növekvő számlálásos ismétlés vezérlés az alábbi vezérlési előírást jelenti:

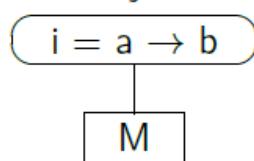
1. Legyen i = a és folytassuk a 2.) lépéssel.
2. Ha b < i, akkor az ismétlés és ezzel együtt az összetett művelet végrehajtása befejeződött.
3. Egyébként, vagyis ha i <= b, akkor hajtsuk végre az M műveletet, majd folytassuk a 4.) lépéssel.
4. Növeljük i értékét 1-gyel, és folytassuk a 2.) lépéssel.

Értelemszerűen hasonlóan működik a csökkenő számlálásos vezérlés is.

Folyamatábrája:



Szerkezeti ábrája:



ez is csak a növekvőre, csökkenőnél megfordulnak az irányok

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- C-ben a **for** utasítás általános alakja így néz ki:

```
for (kif1; kif2; kif3) utasítás;
```

ami egyenértékű ezzel:

```
kif1;
while (kif2) {
    utasítás;
    kif3;
}
```

A kif₁ és kif₃ többnyire értékkadás vagy függvényhívás, kif₂ pedig relációs kifejezés. Bármelyik kifejezés elhagyható, de a pontosvesszőknek meg kell maradniuk. kif₂ elhagyása esetén a feltételt konstans igaznak tekintjük, ekkor a break vagy return segítségével lehet kiugrani a ciklusból.

Hurok ismétléses

Amikor a ciklusmag ismétlését a ciklusmagon belül vezéreljük úgy, hogy a ciklus különböző pontjain adott feltételek teljesülése esetén a ciklus végrehajtását befejezzük, hurok ismétléses vezérlésről beszélünk.

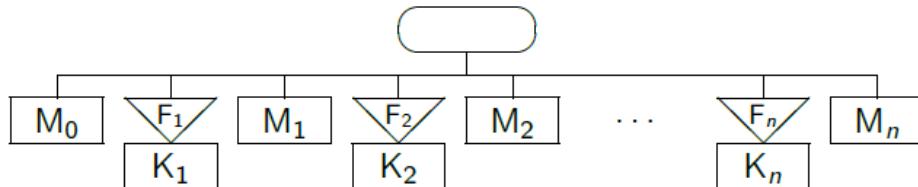
Legyenek Fi logikai kifejezések, Ki és Mj pedig tetszőleges (akár üres) műveletek $1 \leq i \leq n$ és $0 \leq j \leq n$ értékekre. Az Fi kijáratí feltételekből, Ki kijáratí műveletekből és az Mi műveletekből képzett hurok ismétléses vezérlés a következő előírást jelenti:

1. Az ismétléses vezérlés következő végrehajtandó egysége az M₀ művelet.
2. Ha a végrehajtandó egység az M_j művelet, akkor ez végrehajtódi. $j = n$ esetén folytassuk az 1.) lépéssel, különben pedig az F_{j+1} feltétel végrehajtásával a 3.) lépésbén.
3. Ha a végrehajtandó egység az F_i feltétel ($1 \leq i \leq n$), akkor értékeljük ki. Ha Fi igaz volt, akkor hajtsuk végre a Ki műveletet, és fejezzük be a vezérlést. Különben a végrehajtás az Mi művelettel folytatódik a 2.) lépésbén.

A C nyelvben nincs olyan vezérlési forma, amellyel közvetlenül megvalósíthatnánk a hurok ismétléses vezérlést, de a kezdőfeltételesismétléses vezérlés és egyszerű szelekció segítségével kifejezhetjük. Ez a megvalósítás nyelvfüggetlen.

A megvalósítás lényege, hogy választunk egy logikai változót (legyen az a tovább), ez lesz az ismétlés feltétele. Továbbá a ciklusmagban a feltételes kijáratokat úgy alakítjuk át, hogy ha a feltétel igaz, akkor a hozzá tartozó kijáratí művelet végrehajtásra kerül és a tovább hamis értéket kap, egyébként végrehajtódi a ciklusmag további része.

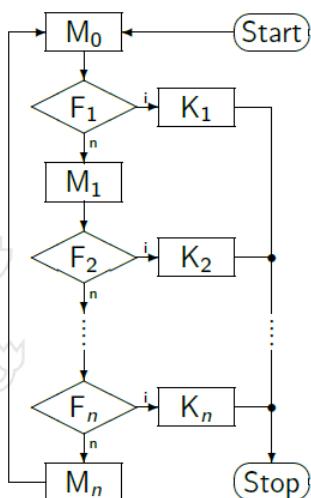
Szerkezeti ábrája:



ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Folyamatábrája:



Diszkrét ismétléses vezérlés

Diszkrét ismétléses vezérlésről akkor beszélünk, ha a ciklusmagot végre kell hajtani egy halmaz minden elemére tetszőleges sorrendben.

Legyen x egy T típusú változó, H a T értékkészletének részhalmaza, M pedig tetszőleges művelet, amelynek nincs hatása x és H értékére. A H halmazból, x ciklusváltozóból és M műveletből (ciklusmagból) képzett diszkrét ismétléses vezérlés az alábbi vezérlési előírást jelenti:

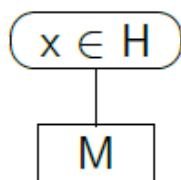
1. Ha a H halmaz minden elemére végrehajtottuk az M műveletet, akkor vége a vezérlésnek.
2. Egyébként vegyük a H halmaz egy olyan tetszőleges e elemét, amelyre még nem hajtottuk végre az M műveletet, és folytassuk a 3.) lépéssel.
3. Legyen $x = e$ és hajtsuk végre az M műveletet, majd folytassuk az 1.) lépéssel.

A diszkrét ismétléses vezérlésnek nincs közvetlen megvalósítása a C nyelvben. A megvalósítás elsősorban attól függ, hogy az ismétlési feltételben megadott halmazt hogyan reprezentáljuk. Algoritmustervezés során szabadon használhatjuk a diszkrét ismétléses vezérlést, ha erre van szükség a probléma megoldásához. A halmaz reprezentálásáról pedig döntünk, amikor elegendő információ áll rendelkezésünkre, hogy a legmegfelelőbbet kiválaszthassuk.

Folyamatábrája:



Szerkezeti ábrája:



Eljárásvezérlés

Eljárásvezérlésről akkor beszélünk, amikor egy műveletet adott argumentumokra alkalmazunk, aminek hatására az argumentumok értékei pontosan meghatározott módon változnak meg.

Az eljárásvezérlés fajtái:

- Eljárás művelet
- Függvény művelet

Az eljárás művelet specifikációja tartalmazza:

- A művelet elnevezését

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- A paraméterek felsorolását
- Mindegyik paraméter adattípusát
- A művelet hatásának leírását

A függvényművelet specifikációja a fentieken túl tartalmazza még:

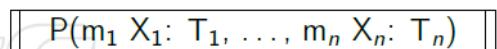
- A függvényművelet eredménytípusát

Eljárásművelet

Eljárásműveleten olyan tevékenységet értünk, amelynek alkalmazása adott argumentumokra az argumentumok értékének pontosan meghatározott megváltozását eredményezi.

Minden eljárásműveletnek rögzített számú paramétere van, és minden paraméter rögzített adattípusú.

Minden paraméter rögzített kezelési módú, ami lehet:

- Bemenő mód: Ha a művelet végrehajtása nem változtathatja meg az adott argumentum értékét.
- Kimenő mód: Ha a művelet eredménye nem függ az adott argumentum végrehajtás előtti értékétől, de az adott argumentum értéke a művelet hatására megváltozhat.
- Be- és kimenő (vegyes) mód: Ha a művelet felhasználhatja az adott argumentum végrehajtás előtti értékét és az argumentum értéke a művelet hatására meg is változhat.
- Eljárásműveletet $P(m_1 X_1 : T_1; \dots; m_n X_n : T_n)$ formában jelölhetünk, ahol
 - P az eljárásművelet neve
 - m_i az i . paraméter kezelési módja
 - X_i az i . paraméter azonosítója
 - T_i az i . paraméter adattípusa
- Az eljárás algoritmusá egy olyan szerkezeti ábrával adható meg, melynek a feje így néz ki:


A fent jelölt eljárásművelet adott A_1, \dots, An argumentumokra történő végrehajtását eljáráshívásnak nevezzük és a $P(A_1, \dots, An)$ jelölést használjuk.

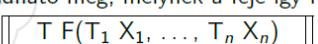
Ha az i . paraméter kezelési módja kimenő vagy be- és kimenő, akkor az A_i aktuális argumentum csak változó lehet.

Az eljáráshívás utasítás.

Ha eljárást szeretnénk készíteni C nyelven, akkor egy olyan függvényt kell deklarálni, melynek eredménytípusa void. Ebben az esetben a függvény definíciójában nem kötelező a return utasítás, illetve ha mégis van ilyen, akkor nem adható meg utána kifejezés.

Függvény művelet

A matematikai függvény fogalmának általánosítása. Ha egy részprobléma célja egy érték kiszámítása adott értékek függvényében, akkor a megoldást megfogalmazhatjuk függvényművelettel. A függvényművelet argumentumai ugyanúgy lehetnek kimenő és be és kimenő módúak is, mint az eljárásműveletek esetén, tehát a függvényművelet végrehajtása az aktuális argumentumok megváltozását is eredményezheti.

- A függvényművelet jelölésére a $T F(T_1 X_1, \dots, T_n X_n)$ formát használjuk, ahol
 - T a függvényművelet eredménytípusa
 - F a függvényművelet neve
 - T_i az i . paraméter adattípusa
 - X_i az i . paraméter azonosítója
- A zárójeleket üres paraméterlista esetén is ki kell tenni.
- A C jelölésmódhoz igazodva, a függvény algoritmusá egy olyan szerkezeti ábrával adható meg, melynek a feje így néz ki:

- Továbbá a szerkezeti ábrában lennie kell (legalább) egy olyan return utasításnak, amely visszaadja a függvény által kiszámított értéket.

A fent jelölt függvényműveletnek adott A_1, \dots, An aktuális argumentumokra történő végrehajtását függvényhívásnak nevezzük és az $F(A_1, \dots, An)$ jelölést használjuk.

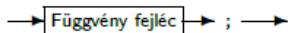
A függvényhívás kifejezés.

A zárójeleket paraméter nélküli függvény hívása esetén is ki kell tenni.

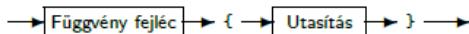
ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

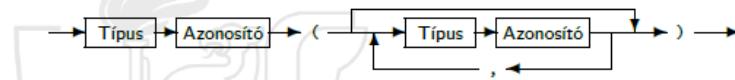
- Függvény deklaráció



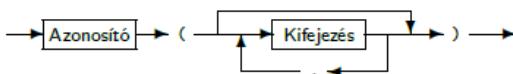
- Függvény definíció (egyben deklaráció is)



- Függvény fejléc



- Függvényhívás szintaxisa C-ben



Minden függvényben szerepelnie kell legalább egy return utasításnak.

Ha a függvényben egy ilyen utasítást hajtunk végre, akkor a függvény értékének kiszámítása befejeződik. A hívás helyén a függvény a return által kiszámított értéket veszi fel.

A return utasítás szintaxisa C-ben



2. Tétel

- Egyszerű adattípusok: egész, valós, logikai és karakter típusok és kifejezések. Az egyszerű típusok reprezentációja, számábrázolási tartományuk, pontosságuk, memória igényük és műveleteik

Adattípus

Az adattípus a programnak egy olyan komponense, amely két összetevője, az értékhalmaz és az értékhalmaz elemein végezhető műveletek által meghatározott. minden adattípus vagy elemi, vagy más adattípusokból képzett összetett adattípus.

| C típus | méret (bájt/bit) | alsó határ | felső határ |
|------------------------|------------------|--------------------------|-------------------------|
| char | 1 / 8 | ? | ? |
| signed char | 1 / 8 | -128 | 127 |
| unsigned char | 1 / 8 | 0 | 255 |
| short int | 2 / 16 | -32 768 | 32 767 |
| signed short int | 2 / 16 | -32 768 | 32 767 |
| unsigned short int | 2 / 16 | 0 | 65 535 |
| int | 4 / 32 | -2 147 483 648 | 2 147 483 647 |
| signed int | 4 / 32 | -2 147 483 648 | 2 147 483 647 |
| unsigned int | 4 / 32 | 0 | 4 294 967 295 |
| long int | 4 / 32 | -2 147 483 648 | 2 147 483 647 |
| signed long int | 4 / 32 | -2 147 483 648 | 2 147 483 647 |
| unsigned long int | 4 / 32 | 0 | 4 294 967 295 |
| long long int | 8 / 64 | -2 ⁶³ | 2 ⁶³ -1 |
| signed long long int | 8 / 64 | -2 ⁶³ | 2 ⁶³ -1 |
| unsigned long long int | 8 / 64 | 0 | 2 ⁶⁴ -1 |
| float | 4 / 32 | -3.4028234663852886E+38 | 3.4028234663852886E+38 |
| double | 8 / 64 | -1.7976931348623157E+308 | 1.7976931348623157E+308 |
| long double | 8 / 64 | -1.7976931348623157E+308 | 1.7976931348623157E+308 |

Egész típus

A C nyelvben az egész típus az int.

Az int típus értékkészlete az alábbi kulcsszavakkal módosítható:

- signed A típus előjeles értékeket fog tartalmazni (int, char)
- unsigned A típus csak előjeltelen, nemnegatív értékeket fog tartalmazni (int, char).

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

- short Rövidebb helyen tárolódik, így kisebb lesz az értékkészlet (int).
- long Hosszabb helyen tárolódik, így bővebb lesz az értékkészlet (int). Duplán is alkalmazható (long long).

Az egész típusok az értékkészlet határain belüli minden egész értéket pontosan ábrázolnak. Az egyes gépeken az egyes típusok mérete más-más lehet, de minden C megvalósításban teljesülne kell a `sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)` relációnak.

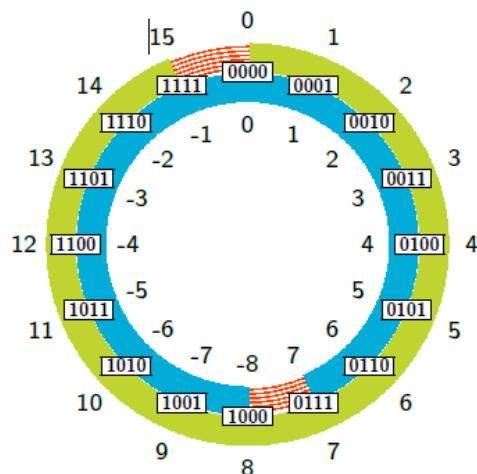
A C nyelv különféle egész adattípusai az értékhalmazukban különböznek egymástól, az értelmezett műveletükben megegyeznek. Az egész adattípusokon általában az 5 matematikai alapműveletet és az értékadás műveletét értelmezzük, de C nyelven ennél jóval többet.

Egy n bites tárterületnek $2n$ állapota van, vagyis egy n biten tárolt adattípusnak legfeljebb ennyi különböző értéke lehet. Egész típusoknál az értékhalmaz szokásos leképezése a következő:

Ha negatív számok nem szerepelnek az értékhalmazban, akkor az értékhalmaz a $[0 \dots 2n - 1]$ zárt intervallum. Az n egymás utáni bitet bináris számként értelmezve kapjuk meg a reprezentált értéket, illetve az értéket bináris számként felírva és n számjegyig balról 0 számjegyekkel kiegészítve kapjuk a számot leíró bitsorozatot.

Ha az értékhalmazban negatív számok is szerepelnek, akkor az értékhalmaz a $[-2n-1 \dots 2n-1]$ zárt intervallum.

- Ha például $n = 4$ bites egész típussal dolgoznánk, akkor összesen $2^4 = 16$ értéket tudnánk megkülönböztetni:
 - Előjeltelen esetben az értékhalmaz a $[0 \dots 15]$ zárt intervallum lenne.
 - Előjeles esetben a $[-8 \dots 7]$ értékeket tudnánk ábrázolni.
 - Az előjeltelen $[8 \dots 15]$ értékek fizikailag pontosan ugyanúgy tárolódnak, mint az előjeles $[-8 \dots -1]$ értékek.



A negatív számoknak ezt a tárolási módját kettes komplemensnek hívjuk.

Egy x érték kettes komplemense a $\sim x + 1$ érték, vagyis x minden bitjét negáljuk, majd hozzáadunk egyet.

- **int → int**
egy operandusú műveletek
 - ~ bitenkénti negáció

```

~a
a & 1
a | 0xf0
a ^ b
1 << a
a >> 3
  
```

- **int × int → int**
két operandusú műveletek
 - & bitenkénti és
 - | bitenkénti vagy
 - ^ bitenkénti kizáráó
 - vagy
 - << balra léptetés
 - >> jobbra léptetés

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Karakter típus

A char adattípus a C nyelv eleve definiált elemi adattípusa, értékkészlete 256 elemet tartalmaz. A char adattípus egészként is használható, de alapvetően karakterek (betűk, számjegyek, írásjelek) tárolására való. Hogy melyik értékhez melyik karakter tartozik, az az alkalmazott kódtáblázattól függ. Bizonyos karakterek (általában a rendezés szerint első néhány) vezérlő karakterek számítanak, és nem megjeleníthetők.

Egy C programban karakter értékeket megadhatunk

- karakterkóddal számértékként, vagy
- aposztrófok (') közé írt karakterrel.

A speciális karaktereket, illetve magát az aposztrófot (és végső soron tetszőleges karaktert is) escape-szekvenciákkal lehet megadni.

- Az escape-szekvenciákat a \ (backslash) karakterrel kell kezdeni.
- Néhány példa

| | | | | | |
|-------------------|-----------|------|---------------|-------------|----|
| újsor | lf | \n | kocsi-vissza | cr | \r |
| vízszintes tab | ht | \t | backspace | bs | \b |
| backslash | \ | \\ | lapdobás | ff | \f |
| aposztróf | ' | \' | csengő | bell | \a |
| Odd kódú karakter | | \ddd | null karakter | null | \0 |

Az, hogy egy adott karakterkódhoz milyen karakter tartozik, a használt kódtáblázat határozza meg.

A kódtábláról csak annyit tételezhetünk fel, hogy

- 'a' < 'b' < ... < 'z'
- 'A' < 'B' < ... < 'Z'
- '0' + 1 == '1', ..., '8' + 1 == '9'

A C nyelv alapvetően az ASCII kódtáblát használja (ez a 0 és 127 közötti kódú karaktereket rögzíti), de a megjelenített karakterkép nagyon sok mindenről függhet.

- Mint említettük a **char** adattípus egészként is használható.
 - A konverzió a kétfajta megadott érték között automatikus, így például '\ddd' == Odd, vagyis ASCII kódtáblázat esetén például '\060' == '0' == 48 == 060, '\101' == 'A' == 65 == 0101, '\172' == 'z' == 122 == 0172.
- Konvertálunk egy tetszőleges számjegy karaktert (ch) a neki megfelelő egész számmá és egy egyjegyű egészet (i) karakterré.

```
i = ch - '0';
ch = i + '0';
```

- Konvertálunk kisbetűt (ch) nagybetűvé (CH) és nagybetűt kisbetűvé.

```
CH = ch - 'a' + 'A';
ch = CH - 'A' + 'a';
```

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Logikai adattípus

A C nyelvnek nem része a logikai (bool) típus, de azért logikai értékek persze keletkeznek.

- A logikai hamis érték a 0 egész értéknek felel meg, és a 0 egész érték logikai értékként értelmezve hamisat jelent.

- A logikai igaz értéket egy tetszőlegesen választott 0-tól különböző egész érték reprezentálhatja, és egy 0-tól különböző egész érték logikai értékként értelmezve igazat jelent. Logikai értékeket egész típusú változókban tudunk tárolni a fenti konverziók figyelembevételével.

A C99 szabvány bevezette az stdbool.h header-t, ami definiálja a bool típust, valamint a false és true literálokat.

Valós típusok

A C nyelvben a valós adattípusok a float és double.

A double adattípus az alábbi kulcsszóval módosítható:

long Implementáció függő módon 64, 80, 96 vagy 128 bites pontosságot megvalósító adattípus.

A tárolási mérete különféle megfontolásokból a valós pontosság által igényeltnél nagyobb lehet (pl. egy 80 bites pontosságú érték 96 biten tárolódhat).

A valós adattípusok az értékkészlet határain belül sem képesek minden valós értéket pontosan ábrázolni. Viszont az értékkészlet határain belüli minden a valós értéket képesek egy típusfüggő e relatív pontossággal ábrázolni, az a-hoz legközelebbi a típus által pontosan ábrázolható x valós értékkel. ($|(x - a)/a| \leq e$)

Az egyes típusok méretére az alábbiak biztosan teljesülnek:

`sizeof(float) < sizeof(double) <= sizeof(long double)`

Műveletek: same mint az intnél

Tárolás:

- Egy valós értéket tároló memóriaterület három részre osztható: az előjelbit, a törtet és az exponenciális kitevőt kódoló részre.
- Az értékhalmaz szokásos leképezése a következő:
 - Az előjelbit 0 értéke a pozitív, 1 értéke a negatív számokat jelöli.
 - A számot kettes számrendszerben $1.m \times 2^k$ alakra hozzuk, majd az m számjegyeit eltároljuk a törtnek, a k-nak egy típusfüggő b konstanssal növelt értékét pedig a kitevőnek fenntartott részen.
 - Így a tört rész hossza az ábrázolás pontosságát (az értékes számjegyek számát), a kitevő pedig az értéktartomány méretét határozza meg.
 - Nagyon kicsi számokat speciálisan $0.m \times 2^{1-b}$ alakban tárolhatunk, ekkor a kitevő összes bitje 0.
 - Ha a kitevő összes bitje 1, az csupa 0 bitből álló tört esetén a ∞ , minden más esetben NaN értéket jelenti.
 - A 32/64 bites float/double az 1 előjelbit mögött 8/11 biten a kitevő b = 127-tel/1023-mal növelt értékét, majd 23/52 biten a törtet tárolja.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Típus definíció

- A C nyelvben lehetőségünk van típusok tetszésünk szerinti elnevezésére, azaz típusdefinícióra melyet a **typedef** kulcsszó vezet be, alakja:

```
typedef típus új_típusnév;
```

- A definíciótól kezdve a **típus**-ra az **új_típusnév** azonosítóval (is) hivatkozhatunk.
- A típusdefinícióval élhetünk például akkor, mikor több helyen kell ugyanolyan típusú változót deklarálni, de ez a típus
 - a jövőbeni fejlesztések során esetleg változhatat, vagy
 - egy bonyolult módon megadható típus, amit nehézkes lenne többször leírni (és a többszöri leírás mellesleg hibaforrás is)!

```
typedef unsigned short int u16;
```

- Felsorolás adattípus értékhalmaza a típusképzésben felsorolt azonosítók, mint konstans azonosítók által meghatározott értékek.
- Felsorolás adattípus esetén az értékadás műveletét és a relációs műveleteket szokás értelmezni, ahol a < rendezési relációt a típusképzésben az elemek felsorolásával definiáljuk.
- Felsorolás adattípust az **enum** kulcsszóval definiálhatunk:

```
enum {elem1, ..., elemn}
```

A felsorolás típus C nyelven nagyon szorosan kötődik az **int** típushoz.

Alapesetben a felsorolásban szereplő első azonosító értéke 0, a többié 1-gyel több az öt közvetlenül megelőzőnél. Ezzel egyben a rendezési relációt is definiáltuk.

A C nyelv lehetőséget ad viszont a felsorolt azonosítók értékének közvetlen megadására:

```
enum {hetfo=1, kedd, szerda, csutortok, pentek, szombat, vasarnap} nap;
```

- Az összetett adattípusok és a típusokképzések, valamint megvalósításuk C nyelven. A pointer, a tömb, a rekord és az unió típus. Az egyes típusok szerepe, használata

Összetett adattípusok

Az összetett adattípusok értékei tovább bonthatóak, további értelmezésük lehetséges.

A C nyelv összetett adattípusai:

- Pointer típus
 - o Függvény típus
- Tömb típus
 - o Sztringek
- Rekord típus
 - o Szorzat-rekord
 - o Egyesítési-rekord

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Pointer

Az olyan változókat, amelyek a blokkok aktivizálásától függetlenül létesíthetők és megszüntethetők, dinamikus változóknak nevezzük.

Dinamikus változók megvalósításának általános eszköze a pointer típus.

Egy pointer típusú változó értéke (első megközelítésben) egy meghatározott típusú dinamikus változó.

- Pointer típusú változót a * segítségével deklarálhatunk:

```
tipus * változónév;
```

A változóhivatkozás szintaktikus egység, meghatározott formai szabályok szerint képzett jelsorozat egy adott programnyelven, tehát egy kód részlet.

A változó a program futása során a program által lefoglalt memóriaterület egy része, amelyen egy adott (elemi vagy összetett) típusú érték tárolódik.

Különböző változóhivatkozások hivatkozhatnak ugyanarra a változóra, illetve ugyanaz a változóhivatkozás a végrehajtás különböző időpontjaiban különböző változókra hivatkozhat.

Egy változóhivatkozáshoz nem biztos, hogy egy adott időben tartozik hivatkozott változó.

Műveletek:

- NULL

```
NULL
```

- Létesít($\leftarrow X:PE$)

```
X = malloc(sizeof(E));
```

- Értékkadás($\leftarrow X:PE; \rightarrow Y:PE$)

```
X = Y;
```

- Törlés($\leftrightarrow X:PE$)

```
free(X);
```

- Dereferencia($\rightarrow X:PE$)

```
*X
```

- Egyenlő($\rightarrow p:PE; \rightarrow q:PE$): bool

```
p == q
```

- NemEgyenlő($\rightarrow p:PE; \rightarrow q:PE$): bool

```
p != q
```

- Az & és a * jobbasszociatív műveletek és magas precedenciájúak.

```
int i, j, *p;
p = &i;           /* p i-re mutat */
j = *p;           /* hatása ua., mint j = i; */
*p = 2;           /* hatása ua., mint i = 2; */
j = *p + 1;       /* hatása ua., mint j = i + 1; */
*p += 1;          /* hatása ua., mint i += 1; */
++*p;            /* hatása ua., mint ++i; */
(*p)++;          /* hatása ua., mint i++; */
```

Pointer típusú változó 32 bites rendszereken 4 bájt, 64 bites rendszereken 8 bájt hosszban a hozzá tartozó dinamikus változóhoz foglalt memóriamező kezdőcímét (sorszámát) tartalmazza.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Tömb

- Algoritmusok tervezésekor gyakran előfordul, hogy adatok sorozatával kell dolgozni, vagy mert az input adatok sorozatot alkotnak, vagy mert a feladat megoldásához kell.
- Tegyük fel, hogy a sorozat rögzített elemszámú (n) és mindegyik komponensük egy megadott (elemi vagy összetett) típusból (E) való érték.
- Ekkor tehát egy olyan összetett adathalmazzal van dolgunk, amelynek egy eleme $A = (a_0, \dots, a_{n-1})$, ahol $a_i \in E$, $i \in (0, \dots, n-1)$ -re.
- Ha az ilyen sorozatokon a következő műveleteket értelmezzük, akkor egy (absztrakt) adattípushoz jutunk, amit Tömb típusnak nevezünk.
- Jelöljük ezt a Tömb típust T -vel, a $0, \dots, n-1$ intervallumot pedig I -vel.

- **Kiolvas** ($\rightarrow A:T$; $\rightarrow i:I$; $\leftarrow x:E$)

- Adott $i \in I$ -re az A sorozat i . komponensének kiolvasása adott x , E típusú változóba.

- **Módosít** ($\leftrightarrow A:T$; $\rightarrow i:I$; $\rightarrow x:E$)

- Adott $i \in I$ -re az A sorozat i . komponensének módosítása adott x , E típusú értékre.

- **Értékkadás** ($\leftarrow A:T$; $\rightarrow X:T$)

- Értékkadó művelet. Az A változó felveszi az X , T típusú kifejezés értékét.

- **Tömb** típusú változót az alábbi módon deklarálhatunk:

típus **változónév**[elemszám];

- A Kiolvas és a Módosít műveletek megvalósítása a tömbelem-hivatkozással történik.
- A tömbelem-hivatkozásra a [] operátort használjuk. Ez egy olyan tömbökön értelmezett művelet C-ben, ami nagyon magas precedenciával rendelkezik és balasszociatív.
- Egy tömbre a tömbindexelés operátort (megfelelő index használatával) alkalmazva a tömb adott elemét változóként kapjuk vissza.

Rekord típus

- A $T = \text{Rekord}(T_1, \dots, T_k)$ típust C-ben a **struct** kulcsszóval definiáljuk:

```
typedef struct T {  
    T1 M1;  
    ...  
    Tk Mk;  
} T;
```

- A fenti típusképzésben az $M1, \dots, Mk$ azonosítókat mezőazonosítóknak (tagnak, member-nek) hívjuk és lokálisak a típusképzésre nézve.

Az absztrakt típus műveletei mezőhivatkozások segítségével valósíthatóak meg.

- A mezőhivatkozásra a . operátort használjuk. Ez egy olyan rekordokon értelmezett művelet C-ben, ami nagyon magas precedenciával rendelkezik és balasszociatív.
- Egy rekordra a mezőkiválasztás operátort (megfelelő mezőnévvel) alkalmazva a rekord mezőjét változóként kapjuk vissza.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Műveletek:

- **Kiolvas_i** ($\rightarrow A:T; \leftarrow x:T_i$)

```
x = A.Mi
```

- **Módosít_i** ($\leftrightarrow A:T; \rightarrow x:T_i$)

```
A.Mi = x
```

- **Értékkadás** ($\leftarrow A:T; \rightarrow X:T$)

```
A = X
```

A struct által lefoglalt memóriamező mérete lekérhető a sizeof() függvényel.

Union

- Az egyesített-rekord típust C-ben az union kulcsszó segítségével valósíthatjuk meg.
- Az union kulcsszó nagyon hasonlít a struct-hoz:

```
typedef union T {  
    T1 M1;  
    ...  
    Tk Mk;  
} T;
```

- A fenti típusképzésben az M1, ..., Mk azonosítókat mezőazonosítóknak (tagnak, member-nek) hívjuk és lokálisak a típusképzésre nézve.

Az absztrakt típus műveletei itt is mezőhivatkozások (. operátor) segítségével valósíthatóak meg, ami ugyanúgy működik mint a struct típusnál.

- Megjegyzendő, hogy a C megvalósításában (megfelelő környezetben) minden hivatkozhatunk bármelyik mezőre, függetlenül attól, hogy az union aktuálisan melyik mező értékét tárolja.
- A C union típusképzésében nem adhatunk meg változati mezőazonosítót (T0), így nincs lehetőségünk az aktuális változatról információ tárolására.

The main difference between union and struct is that union members overlay the memory of each other so that the sizeof of a union is the one , while struct members are laid out one after each other (with optional padding in between).

Egyesített-rekord:

- Mint látható, az union konstrukcióban nincs lehetőség a jelzőmező (T_0) megadására, ezért ha szükségünk van a jelzőmezőre is, akkor a C megvalósításhoz kombinálnunk kell a struct és union típusképzéseket.

```
typedef struct T {  
    T0 Milyen;  
    union {  
        T1 M1;  
        ...  
        Tk Mk;  
    };  
} T;
```

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Programozás I és II

1. Tétel

- Objektum orientált paradigmá és annak megvalósítása a JAVA és C++ nyelvekben
- Az absztrakt adattípus, az osztály
- Az egysége zárás, az információ elrejtés, az öröklődés, az újrafelhasználás és a polimorfizmus
- A polimorfizmus feloldásának módszere

2. Tétel

- Objektumok életciklusa, létrehozás, inicializálás, másolás, megszüntetés
- Dinamikus, lokális és statikus objektumok létrehozása
- A statikus adattagok és metódusok, valamint szerepük a programozásban
- Operáció és operátor overloading a JAVA és C++ nyelvekben
- Kivételkezelés

3. Tétel

- Java és C++ programok fordítása és futtatása
- Parancssori paraméterek, fordítási opciók, nagyobb projektek fordítása
- Absztrakt-, interfész- és generikus osztályok, virtuális eljárások
- A virtuális eljárások megvalósítása, szerepe, használata

prog1-2_1 és prog1-2_2 → pdf-ek

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Programozási nyelvek

1. Tétel

- A programozási nyelvek csoportosítása (paradigmák), az egyes csoportokba tartozó nyelvek legfontosabb tulajdonságai

Nyelvcsoportok (paradigmák):

-**Imperatív procedurális** pl.: C, C++, Pascal

Az ilyen nyelven írt program utasítások és eljáráshívások sorozata, amelyek sorrendben hajtódnak végre. Az állítások végrehajtása egy vagy több érték megváltozását jelenti a memóriában. Az imperatív nyelveken megírt programok szerkezete a következő általános formában írható le:

utasítás₁;
utasítás₂;

Az utasítások paramétereik segítségével hatnak egymásra, a változóban való érték tárolásával a benne tárolt előző érték elvész. A programozás így megfelelő memória-állapotok létrehozását jelenti, amely végül a megoldáshoz vezet. A legtöbb modern procedurális nyelv tartalmaz továbbá számos funkcionális elemet is.

Utasítások

- Imperatív programozás fő jellemzője
- Utasítások végrehajtása állapotátmenetet okoz
- Egyszerű utasítások
 - Üres utasítás, értékadás, alprogramhívás
- Összetett utasítások (vezérlési szerkezetek)
 - Szekvencia, (párhuzamosság)
 - Feltételes utasítás, ismétlés

Mellékhatás

- Kifejezés kiértékelése közben külső változó értéke megváltozhat
- A mellékhatások nehezen követhetővé teszik a programot
- A mellékhatásokat lehetőleg kerülni, előfordulásukat megfelelően dokumentálni kell

-**Objektum orientált** pl.: C++, Java, Smalltalk

Az objektumorientált nyelvekben a program egymással kölcsönhatásban álló objektumok összesége. Az objektumok típusosak, és magukban foglalják a típusazonosítókat. minden függvénynek többféle definíciója lehetséges, melyeket ebben a terminológiában metódusnak nevezünk. Az objektumorientált megközelítés használata a programozási nyelvekkel szemben újabb és újabb követelményeket támasztott.

Objektum alapú nyelvek: Az objektum alapú nyelvek támogatják az objektumot, mint nyelv eszközt. Az osztály, mint fogalom még nincs kialakítva. Az objektumokat modulokkal (csomagokkal) lehet ábrázolni. (pl. Ada, Modula)

Osztály alapú nyelvek: Megjelennek az osztályok, és ezek lesznek a modularizáció alapjai is, nincs azonban osztályhierarchia.

Objektumorientált nyelvek: Napjainkban a legfontosabbak az objektumorientált nyelvek, ahol az objektumok osztályokhoz tartoznak és az osztályok közötti hierarchiát az öröklődés.

Objektum orientált programozás

- Osztályok, objektumok, metódusok (kizárolagos) használata -> szimuláció, modellezés
- Szemléletmód (analízis, tervezés, megvalósítás), ábrázolásmód
- Adatelrejtés, láthatóság
- Az objektumok elsőrendű értékek
- Öröklődés (osztályok közötti hasonlóság kifejezésére)
- Dinamikus típushozzárendelés

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

-Applikatív, funkcionális pl.: Haskell, ML

A funkcionális programozás alapelveit követve a funkcionális nyelvet tisztán függvényekre alapozzuk, műveletenként csupán a függvény összehasonlítást, a rekurziót és a feltételes kifejezést használva. A függvény értékeit kizárolag az argumentumok értékei határozzák meg, így a függvény hívása meghatározott argumentumok hazsnálatával minden ugyanazt az értéket hozza létre. Ezt a tulajdonságot hivatkozási átláthatóságnak nevezzük. A programok formája általában a következő:

`függvényn(... függvény2(függvény1()))`

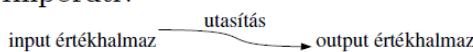
-Szabály alapú, logikai pl.: Prolog, HASL

A logikai nyelvek interaktív nyelvek, amelyek számításait a formális logika és a halmazelmélet szabályait követve végzik. A logikai program alapszerkezete a kifejezés. A kifejezés lehet állandó, változó vagy összetett kifejezés. A szabály egy függvénynevet és egy vagy több kifejezést, mint paramétert tartalmaz. A tények 0 változós szabályoknak tekinthetők. A logikai program szabályok véges készlete. A program feladata egy olyan eset keresése, amely az adott kérdésre a programból logikailag következik. A logikai nyelven írt programok formája általában a következő:

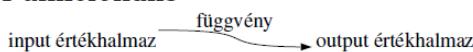
`megengedő feltétel1 -> akció1
megengedő feltétel2 -> akció2
...
megengedő feltételn -> akción`

Logikai programozás

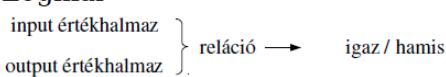
- Imperatív



- Funkcionális



- Logikai



-Párhuzamos pl.: Occam, PVM, MPI

A programok végrehajtása több végrehajtási szál (folyamat, angolul: process) mentén történik. Előnyei:

- o Természetes kifejezésmód,
 - o Sebességnövekedés megfelelő hardver esetén.
- Hátrányai:
- o bonyolultabb a szekvenciális programozásnál.

Funkcionális programozás

- Értékek, (matematikai) kifejezések és függvények
- A program egy függvény
- Változók értéke nem változik
- Ciklus helyett rekurzió
- Interakció (input/output) nehezen fejezhető ki
- Hivatkozási átlátszóság (referential transparency)
- Helyesség ellenőrzés könnyebb

Prolog - PROgramming in LOGic

- Egy Prolog program csak az adatokat és az összefüggéseket tartalmazza.

Kérdések hatására a "programvégrehajtást" beépített következtető-rendszer végzi

- Programozás Prologban:

- Objektumok és azokon értelmezett relációk megadása
- Kérdések megfogalmazása a relációkkal kapcsolatban

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Párhuzamos programozás

- Sokféle párhuzamos programozási modell van
- Közös problémák:
 - Adathozzáférés folyamatokból
 - Közös memória (shared memory)
 - Osztott memória (distributed memory) + kommunikáció
 - Folyamatok létrehozása, megszüntetése, kezelése
 - Folyamatok együttműködése (interakciója)
 - Független
 - Erőforrásokért versenáció

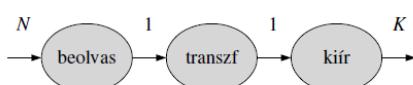
Occam

- Imperatív, folyamatok saját memóriával rendelkeznek, üzenetküldéssel kommunikálnak
- Occam program részei:
 - Változók
 - Folyamatok
 - Csatornák
- Szigorú formai követelmények

- Folyamat életciklusa:
 - Elindul
 - Csinál valamit
 - Befejeződik (terminál)
- Befejeződés helyett holtpontba is kerülhet, erre különös figyelmet kell fordítani
- Elemi és összetett folyamatok

Struktúraütközés

- N méretű rekordok beolvasása, elemek transzformációja, K méretű rekordok kiírása
- Legtermészetesebb kifejezésmód a párhuzamosság, 3 elemű csővezeték

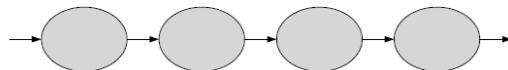


Párhuzamos program

- Sebességfüggő: a folyamatok relatív sebessége minden futáskor más lehet
- Nemdeterminisztikus: ugyanarra az inputra különböző output
- Holtpont (deadlock): kölcsönös egymásra várakozás
- Éhezés (starvation): Nincs holtpont, egy folyamat mégsem jut hozzá az erőforrásokhoz

- A csatorna két folyamat közötti adatátvitelre szolgál
 - Egyirányú
 - Küldő és fogadó is legfeljebb egy lehet
 - Biztonságos
 - Szinkron: A küldő és a fogadó bevárják egymást, megtörténik az adatátvitel, majd a küldő és a fogadó folytatódik
- Csatorna típusa: protokoll
 - CHAN OF INT c:

Párhuzamosság csővezetékkel



- minden adatelemet minden folyamat feldolgoz
- Különböző adatelemek a feldolgozás különböző fázisaiban lehetnek
- Sebesség szempontjából a kiegyszűlyozott fázisok a legjobbak

Aszinkron kommunikáció

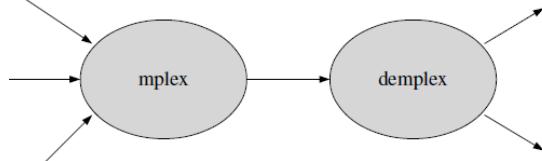
- Aszinkron kommunikáció esetén a küldő nem vár a fogadóra
- Lehetnek már elküldött, de még nem fogadott üzenetek (pufferelt csatorna)
- A csatornakapacitás határozza meg, mennyi úton lévő üzenet lehet
 - Korlátlan
 - Korlátos, ezt occamban is meg lehet valósítani

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Multiplexer

- Multiplexer: több forrásból érkező adatok átirányítása egy kimenő csatornára
- Demultiplexer: Bejövő csatornán érkező adatok szétválogatása kimenő csatornákra

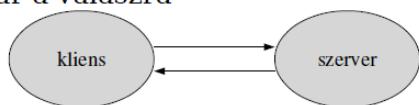


Terheléselosztás - munkafelosztás

- Funkcionális párhuzamosság: különböző folyamatok különböző feladatot végeznek, a szükséges adatokat átadják egymásnak. Lásd csővezeték.
- Adatpárhuzamosság: a folyamatok ugyanazt a feladatot végzik el, az adatokat fel kell osztani közöttük (particionálás)

Kliens - szerver rendszer

- Osztott rendszerek gyakori felépítési módja
- A szerver passzív. Kérésre vár, majd feldolgozás után válaszol
- A kliens aktív. Kezdeményezi a kapcsolat-felvételt a szerverrel, majd vár a válaszra



Folyamatok közös memóriában

- Osztott modellben a csatornák szinkronizálják is a folyamatokat
- Közös memória esetén is szükség van szinkronizációra az erőforrások kezelésekor

$y := y + 3$

$y := y + 5$

$r \leftarrow y$
 $r += 3$
 $y \leftarrow r$

$r \leftarrow y$
 $r += 5$
 $y \leftarrow r$

Terheléselosztás

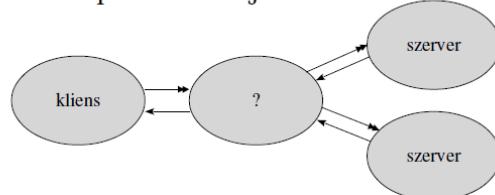
- Több processzor esetén akkor érhetjük el a legnagyobb sebességnövekedést, ha a processzorok folyamatosan dolgoznak és egyszerre végeznek
- A processzorok teljesítményét figyelembe véve a munkát fel kell osztani ugyanakkora terhelést jelentő részekre
 - Statikus terheléselosztás: futás előtt
 - Dinamikus terheléselosztás: futás közben

Processzor farm

- Az adatpárhuzamosság leggyakoribb megvalósítási formája, 1 farmer és több worker (dolgozó) folyamatból áll.
- A farmer feladata a feldolgozandó adatok particionálása és az eredmények begyűjtése
- A dolgozók feladata az adatok fogadása, feldolgozása és visszaküldése
- Egy nagyságrenddel több részfeladat kell, mint amennyi dolgozó van

Kliens - szerver rendszer

- Egyes folyamatok kliensként is és szerverként is viselkedhetnek
- Nem a folyamatokat, hanem a közöttük lévő kapcsolatokat jellemizzük



Kölcsönös kizáráás

- Kritikus szekció: közös erőforrást kezelő kódrészlet egy folyamatban
- Kölcsönös kizáráás: egyszerre csak egy folyamat lehet kritikus szekciójában
- Szemafor: kölcsönös kizáráás megvalósítására szolgáló eszköz
 - Egész értékű változó, várakozó sor
 - Műveletek: init, wait, signal

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Rendszerfejlesztés I.

1. Tétel

- Szoftverfejlesztési folyamat és elemei; a folyamat különböző modelljei

A szoftverfolyamat

- Tevékenységek és kapcsolódó eredmények, amely során elkészítjük a szoftvert
- Fázisok
 - Specifikáció (mit)
 - Fejlesztés (hogyan)
 - Validáció (ellenőrzés)
 - Evolúció (változás)

2018.12.05. 21

Folyamat szerepe

- Szoftverfejlesztés = folyamat + menedzsment + technikai módszerek + eszközök használata
- Minőségi szoftver biztosítéka
- Folyamat meggátolja hogy elveszítsük az uralmat a projekt felett
- Adaptálás adott projekthez és környezethez

2018.12.05. 22

Folyamat elemek

- Fő elemek
 - Feladatok, termékek
 - Határidők, átadandók
- Kiegészítő elemek
 - Projektmenedzsment
 - Konfiguráciomenedzsment
 - Dokumentáció
 - Minőségbiztosítás, kockázatmenedzsment
 - Mérés

2018.12.05. 23

Folyamat fejlettsége

- Egy szerveztnél alkalmazott folyamat minősítése meghatározhatja a megrendelők bizalmát
- SEI CMM(I)
 - Kezdeti
 - Reprodukálható
 - Definiált
 - Ellenőrzött
 - Optimalizált

2018.12.05. 24

Folyamat modellek

- Triviális modell: lineáris
- Iteratív modellek: prototípus, RAD
- Majdnem minden modell inkrementális (pl. XP)
- Evolúciós
- Komponens alapú
- Formális módszerek

2018.12.05. 25

A szoftverfejlesztés költsége

- Nehéz kérdés a teljes költség és folyamatban belüli megoszlás is
- Költségbecsílási modellek
- Egyedi szoftver minden költségesebb
- Evolúciós költségek gyakran a legnagyobbak
- Szoftverváltozásnak nagy költsége van
 - Újrafelhasználhatóság fontos!

2018.12.05. 26

CASE eszközök

- Szoftver rendszer, mellyel a folyamat tevékenységeit lehet automatizálni
- Különféle eszközök
 - Követelményelemzés
 - Modellezés
 - Implementáció, kódgenerálás
 - Nyomkövetés, tesztelés
 - Dokumentációkészítés
 - Folyamatirányítás

2018.12.05. 27

A jó szoftver tulajdonságai

- Nem csak az elvárt funkcionális és teljesítmény!
- Üzembiztonság
- Jó használhatóság
- Karbantartható, evolúcióra felkészített
 - Jó folyamatot kell használni
 - Komponens alapú fejlesztés
 - Fejlesztési dokumentáció

2018.12.05. 28

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Folyamat elemek

- ▶ Fő elemek
 - Feladatok, termékek
 - Határidők, átadandók
- ▶ Kiegészítő elemek
 - Projektmenedzsment
 - Konfigurációmenedzsment
 - Dokumentáció
 - Minőségbiztosítás, rizikómenedzsment
 - Mérés

A szoftverfolyamat fázisai

- ▶ minden folyamatnak elemei:
 - Specifikáció: szoftver funkcionalitása, megszorítások („mit”)
 - Fejlesztés: tervezés és implementáció specifikáció alapján („hogyan”)
 - Validáció: tesztelés, fejlesztés megfelel-e a specifikációknak
 - Evolúció: változás kezelése, szoftver „utóélete”

Specifikáció

- ▶ Szoftver definíálása
 - Milyen funkciókat, szolgáltatásokat követelünk meg a rendszertől
 - Követelménytervezés
- ▶ Kritikus szakasz
 - Ilt a legkisebb a változás költsége
- ▶ Eredmény: követelményspecifikáció dokumentum, esetleg prototípusok
 - Végfelhasználónak (magas szintű)
 - Fejlesztőknek (részletes, technikai)

Követelménytervezés fázisai

- ▶ Megvalósíthatósági tanulmány (feasibility study)
 - Költséghatékonyosság ellenőrzése
- ▶ Követelmények feltárása és elemzése
 - Rendszermodellek, prototípusok
- ▶ Követelményspecifikáció
 - Egységes dokumentum
- ▶ Követelmény validáció

Fejlesztés

- ▶ Specifikációból futtatható rendszer
 - Tervezés
 - Implementáció (programozás)
- ▶ Tervezés
 - Szoftver struktúrája, adatok, interfészek
 - Rendszermodellek különböző absztrakciós szinteken

Tervezés tevékenységei

- ▶ Architektúra tervezés
 - Alrendszer meghatározása
- ▶ Absztrakt specifikáció
 - Alrendszer szolgáltatásai
- ▶ Interfész tervezés
 - Alrendszer között
- ▶ Komponens tervezése
- ▶ Részletek: adatszerkezetek, algoritmusok
- ▶ Gyakorlati folyamatok speciálisan definiálják ezeket

Tervezési módszerek

- ▶ Ad hoc
- ▶ Strukturált
 - Structured Design (SD)
 - SSADM
 - Jackson
- ▶ Objektumorientált
 - OMT, UML, RUP
- ▶ Közös: grafikus rendszermodellek, szabványos jelölérendszer, CASE támogatás

Tipikus rendszermodellek

- ▶ Adatfolyam modell (data-flow)
- ▶ Egyed-kapcsolat modell (ER)
 - Adatbázis-szerkezetekhez
- ▶ Strukturált modell
 - Komponensekhez
- ▶ Objektumorientált módszerek
 - Használati eset-, szerkezeti- és viselkedési modellezés
- ▶ Részletek: állapotátmenet, egyed-élettartam, stb.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

| | | | |
|--|--|------------------------|-----------|
| <p>Evolúció</p> <ul style="list-style-type: none">▶ Szoftver flexibilitás miatt nagy, összetett rendszerek születnek▶ Változás bár költséges,<ul style="list-style-type: none">■ Eredményesebb meglévő rendszerekből kialakítani az újat■ Kevés a teljesen új szoftver■ Egy szoftver sosincs készén | <p>Evolúció (folyt.)</p> <ul style="list-style-type: none">▶ Kötetelményspecifikáció után a meglévő rendszereket kiértékeljük<ul style="list-style-type: none">■ Manuális vizsgálat■ Automatikus elemzés (<i>reverse engineering</i>)■ Kimenet: dokumentáció, magasabb szintű rendszermodell▶ Rendszermódosítások után jön létre az „új” rendszer<ul style="list-style-type: none">■ Újratervezés (<i>re-engineering</i>)■ Folyamatos evolúció (<i>roundtrip engineering</i>) | | |
| <p>2007. június 6.</p> | <p>57</p> | <p>2007. június 6.</p> | <p>58</p> |
| <p>Automatizált folyamatok</p> <ul style="list-style-type: none">▶ CASE = Computer Aided Software Engineering▶ Folyamat minden fázisához léteznek eszközök, amelyekkel a rutinfeladatok automatizálhatók▶ Nem forradalmasítják a szoftverfejlesztést:<ul style="list-style-type: none">■ Kreatív gondolkodást nem lehet helyettesíteni■ Sok csapatmunka van, közvetlen eszmecserekkel▶ CASE rendszerek méretei<ul style="list-style-type: none">■ Eszközök, eszközökészletek, környezetek | <p>CASE eszközök</p> <ul style="list-style-type: none">▶ Tervező-, újratervező eszközök▶ Szerkesztő-, nyomkövető eszközök▶ Változáskezelő eszközök▶ Konfigurációkezelő eszközök▶ Prototípuskészítő eszközök▶ Módszertámogató eszközök▶ Nyelvi feldolgozó eszközök▶ Programelemző eszközök▶ Tesztelő eszközök▶ Dokumentációs eszközök | | |
| <p>2007. június 6.</p> | <p>59</p> | <p>2007. június 6.</p> | <p>60</p> |
| <p>Folyamat modell kategóriák</p> <ul style="list-style-type: none">▶ Triviális: lineáris, vizesés<ul style="list-style-type: none">■ Tevékenységek különálló fázisok■ Iteratív modellek: prototípus, vizesés, RAD▶ Evolúciós<ul style="list-style-type: none">■ Prototípusok gyors gyártása, finomítása▶ Formális módszerek<ul style="list-style-type: none">■ Matematikai rendszer, transzformációk▶ Újrafelhasználható, komponens alapú | <p>Vizesés modell</p> <ul style="list-style-type: none">▶ Első publikált, „klasszikus” modell (<i>életciklus modell</i>)▶ Lényegében egy szekvenciális modell<ul style="list-style-type: none">■ Fázisok lépcsősen kapcsolódnak■ Visszacsatolás is van▶ Fázisok kimenetei teljesen el kell, hogy készüljenek, mielőtt továbbmegyünk<ul style="list-style-type: none">■ A hibákat összegyűjtik a fázisok végén■ Javításra a folyamat végén van lehetőség■ Iteráció közvetve van jelen▶ Ha jó a specifikáció, akkor működőképes | | |
| <p>2007. június 6.</p> | <p>61</p> | <p>2007. június 6.</p> | <p>62</p> |
| <p>Vizesés modell (folyt.)</p> <pre>graph TD; A[Követelmények] --> B[Tervezés]; B --> C[Implementáció]; C --> D[Tesztelés]; D --> E[Karbantartás]; D -. dashed line .-> B;</pre> | <p>Vizesés modell problémái</p> <ul style="list-style-type: none">▶ Ritkán van egyszerű lineáris fejlesztés▶ Követelményeket nehéz pontosan specifikálni a legelején▶ A megrendelő csak a legvégén látja meg először a terméket<ul style="list-style-type: none">■ Sok hiba ekkor derül ki, melyek javítási költsége nagy▶ Előnye: megrendelő könnyebben tud megállapodni, mert a specifikáció pontos<ul style="list-style-type: none">■ Viszont nehezen módosítható szoftver alakul ki | | |
| <p>2007. június 6.</p> | <p>63</p> | <p>2007. június 6.</p> | <p>64</p> |

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Iteráció, inkrementalitás

- ▶ Folyamat iterációja elkerülhetetlen
 - Ha a követelmények változnak, akkor a folyamat bizonyos részeit is változtatni kell
- ▶ Iteráció szélsőséges esetei:
 - Vízesés modellnél minimális lehetőség
 - Prototípus (vagy evolúciós) modellnél minimális a specifikáció, fejlesztésben sok iteráció van, és menet közben alakul ki a végleges specifikáció
- ▶ Inkrementalitás: részfunkciókkal már működő rendszert fejlesztünk, amit minden iterációban (inkrementálisan) javítunk

UNIVERSITAS SCIENTIARUM ET STUDIORUM BUDAPESTENSIS
SZEGEDI TUDOMÁNYEGYETEM
Szentegyházi Sándor Egyetem

2007. június 6.

65

Evolúciós fejlesztés

- ▶ Prototípus modell: az evolúciós fejlesztés egy szélsőséges iteratív+inkrementális példája
 - Durva specifikáció megrendelő részéről
 - Ezután gyors fejlesztés, eredménye prototípus
 - Prototípus kiértékelése után követelményspecifikáció újraíródik
 - Sok-sok iteráció a végtermékig
- ▶ Nagy dilemma: a prototípusból lesz-e a végtermék, vagy az csak eldobható
- ▶ Intenzív kapcsolat kell a megrendelővel
- ▶ Kész komponensek alkalmazása előnyös

UNIVERSITAS SCIENTIARUM ET STUDIORUM BUDAPESTENSIS
SZEGEDI TUDOMÁNYEGYETEM
Szentegyházi Sándor Egyetem

2007. június 6.

66

Prototípus modell problémái

- ▶ A megrendelő azt gondolja, hogy a prototípus kész rendszer, nehéz ellenállni, hogy ne használja
- ▶ Gyors fejlesztés miatt minőség romolhat, kevésbé hatékony megoldások alkalmazása miatt
 - Ezek beépülhetnek a végső rendszerbe
- ▶ Megrendelő sokszor vállalja a rizikókat, mert:
 - Szeret „belélni” a fejlesztésbe
 - Kezdetben pontosan tudja, hogy mit szeretne, de a részletekről fogalma sincs
- ▶ Hibrid megoldások kellenek, vízesés modellel

UNIVERSITAS SCIENTIARUM ET STUDIORUM BUDAPESTENSIS
SZEGEDI TUDOMÁNYEGYETEM
Szentegyházi Sándor Egyetem

2007. június 6.

67

Inkrementális modell

- ▶ Vízesés: véglegesíteni kell minden fázis eredményét a következő fázis előtt
 - Változtatás nehéz, robosztus rendszerek
 - Szigorú folyamat
- ▶ Evolúciós: elhalasztathatók a döntések
 - Gyengén strukturált rendszerek
 - Flexibilis folyamat
- ▶ Inkrementális fejlesztés: kettőt kombinálja

UNIVERSITAS SCIENTIARUM ET STUDIORUM BUDAPESTENSIS
SZEGEDI TUDOMÁNYEGYETEM
Szentegyházi Sándor Egyetem

2007. június 6.

68

Inkrementális modell (folyt.)

- ▶ Nagy körvonalakban specifikáljuk a rendszert
 - „Inkremensek” meghatározása
 - Funkcionalitásokhoz prioritásokat rendelünk
 - Magasabbakat előbb kell biztosítani
- ▶ Architektúrát meg kell határozni
- ▶ További inkremensek pontos specifikálása menet közben történik

UNIVERSITAS SCIENTIARUM ET STUDIORUM BUDAPESTENSIS
SZEGEDI TUDOMÁNYEGYETEM
Szentegyházi Sándor Egyetem

2007. június 6.

69

Inkrementális modell (folyt.)

- ▶ Egyes inkremensek kifejlesztése történhet akár különböző folyamatokkal is
 - Vízesés vagy evolúciós, amelyik jobb
- ▶ Az elkészült inkremenseket akár szolgálatba is lehet állítani
 - Tapasztalatok alapján lehet meghatározni a következő inkremenseket
- ▶ Az új inkremenseket integrálni kell a már meglévőkkel

UNIVERSITAS SCIENTIARUM ET STUDIORUM BUDAPESTENSIS
SZEGEDI TUDOMÁNYEGYETEM
Szentegyházi Sándor Egyetem

2007. június 6.

70

Inkrementális modell előnyei

- ▶ A szoftver már menet közben használható
- ▶ Korábbi inkremensek prototípusként használhatók, a későbbi követelmények pontossítása érdekében
- ▶ Ha határidő csúszás van kilátásban, inkrementális modell bevethető
 - Teljes projekt nem lesz kudarcról ítélezve, esetleg csak egyes inkremensek
- ▶ A legfontosabb inkremensek lesznek többször tesztelve
 - Mivel azokkal kezdtük a megvalósítást

UNIVERSITAS SCIENTIARUM ET STUDIORUM BUDAPESTENSIS
SZEGEDI TUDOMÁNYEGYETEM
Szentegyházi Sándor Egyetem

2007. június 6.

71

Inkrementális modell hátrányai

- ▶ Megfelelő méretű inkremensek meghatározása nem triviális feladat
 - Ha túl kicsi: nem működőképes
 - Ha túl nagy: elveszítjük a modell lényegét
- ▶ Bizonyos esetekben számos alapvető funkcionálitást kell megvalósítani
 - Egész addig nincs működő inkremens
 - Csak akkor pörög be a rendszer, ha minden összeállt

UNIVERSITAS SCIENTIARUM ET STUDIORUM BUDAPESTENSIS
SZEGEDI TUDOMÁNYEGYETEM
Szentegyházi Sándor Egyetem

2007. június 6.

72

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

eXtreme Programming (XP)

- ▶ Szélsőséges inkrementális modell
 - Nagyon kis funkcionális inkremensek
 - Megrendelő intenzív részvételle
- ▶ Programozás csoportos tevékenység
 - Többen ülnek a képernyő előtt
 - ▶ Az utóbbi időben sok kiegészítés készül
 - Sajnos kezdi kinöni az eredeti elképzélést
 - ▶ Sok támadója van

XP – Dilbert szerint

FIRST, PICK A PARTNER. THE TWO OF YOU WILL WORK AT ONE COMPUTER FOR FORTY HOURS A WEEK.
THE NEW SYSTEM IS A MINUTE OLD AND I ALREADY HATE EVERYONE.

XP – Felhasználó rémálma

EXTREME PROGRAMMING
I CAN'T GIVE YOU ALL THE FEATURES IN THE FIRST VERSION.
AND EACH FEATURE NEEDS TO HAVE WHAT WE CALL A "USER STORY."
OKAY, HERE'S A STORY: YOU GIVE ME HALF OF MY FEATURES OR I'LL RUIN YOUR LIFE.

XP – Csoportmunka

EXTREME PROGRAMMING
STUDIES PROVE THAT TWO PROGRAMMERS ON ONE COMPUTER IS THE MOST PRODUCTIVE ARRANGEMENT.
THE TWO OF YOU WILL BE A CODE-WRITING TEAM.
SOMETIMES I CAN WHISTLE THROUGH BOTH NOSTRILS. I'VE SAVED A FORTUNE IN HARMONICAS.

RAD

- ▶ Rapid Application Development
- ▶ Extrém rövid életciklus
 - Működő rendszer 60-90 nap alatt
- ▶ Vizesés modell „hagysebességű” adaptálása
 - Párhuzamos fejlesztés
 - Komponens alapú fejlesztés

RAD fázisok

- ▶ Üzleti modellezés
 - Milyen információk áramlanak funkciók között
- ▶ Adatmodellezés
 - Finomítás adatszerkezetekre
- ▶ Adatfolyam processzus
 - Adatmodell megvalósítása
- ▶ Alkalmazás generálás
 - 4GT alkalmazása, automatikus generálás, komponensek
- ▶ Tesztelés
 - Csak komponens tesztelés

RAD problémái

- ▶ Nagy emberi erőforrásigény
- ▶ Fejlesztők és megrendelők intenzív együttműködése
- ▶ Nem minden típusú fejlesztésnél alkalmazható
 - Modularizálhatóság hiánya problémát jelenthet

RAD

UNIVERSITAS SCIENTIARUM SIEGEL DILBERT TUDOMANYEGYETEM
Dilbert © 2003 United Feature Syndicate, Inc.

60-90 days

147

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Spirális modell

- ▶ Olyan evolúciós modell, amely kombinálja a prototípus modellt a vizesés modellel
- ▶ Inkrementális modellhez hasonló, csak általánosabb megfogalmazásban
- ▶ Nincsenek rögzített fázisok, minden egyedi modellek
- ▶ Más modelleket ölelhet fel, pl.:
 - Prototípuskészítés pontatlan követelmények esetén
 - Vizesés modell egy későbbi körben
 - Kritikus részek esetén formális módszerek

UNIVERSITAS SCIENTIARUM ET ARTium
SZEGEDI TUDOMÁNYEGYETEM
Szegeci Tudományegyetem

2007. június 6.

81

Spirális modell

- ▶ A spirál körei a folyamat egy-egy fázisát reprezentálják
- ▶ minden körben a kimenet egy „release” (modell vagy szoftver),
- ▶ Körök céljai pl.:
 - Megvalósíthatóság (elvi prototípusok)
 - Követelmények meghatározása (prototípusok)
 - Tervezés (modellek és inkremensek)
 - Stb. (javítás, karbantartás, stb.)
- ▶ A körök szektorokra oszthatók
 - 3-6 darab

UNIVERSITAS SCIENTIARUM ET ARTium
SZEGEDI TUDOMÁNYEGYETEM
Szegeci Tudományegyetem

2007. június 6.

82

Spirális modell 4 szektorral

- ▶ Célok kijelölése
- ▶ Kockázat becslése és csökkentése
- ▶ Fejlesztés és validálás
- ▶ Tervezés
 - Következő spirálkör megtervezése

UNIVERSITAS SCIENTIARUM ET ARTium
SZEGEDI TUDOMÁNYEGYETEM
Szegeci Tudományegyetem

2007. június 6.

83

Spirális modell 6 szektorral

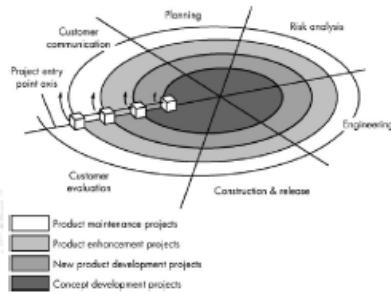
- ▶ Kommunikáció megrendelővel
- ▶ Tervezés
- ▶ Kockázatelemzés
- ▶ Fejlesztés
- ▶ Megvalósítás és telepítés
- ▶ Kiértékelés megrendelő részéről

UNIVERSITAS SCIENTIARUM ET ARTium
SZEGEDI TUDOMÁNYEGYETEM
Szegeci Tudományegyetem

2007. június 6.

84

Egy tipikus spirális modell



UNIVERSITAS SCIENTIARUM ET ARTium
SZEGEDI TUDOMÁNYEGYETEM
Szegeci Tudományegyetem

2007. június 6.

85

WINWIN spirális modell

- ▶ WINWIN = mindenki nyer
 - Megrendelő és fejlesztő is
- ▶ Sok tárgyalás kell a két fél között
- ▶ WINWIN modell számos tárgyalási szempontot visz bele a spirális modellbe
 - Egyes (al)rendszerök kulcsszereplői, érdekeltek
 - Az érdekeltek nyerő feltételei
 - Tárgyalás, kompromisszumok

UNIVERSITAS SCIENTIARUM ET ARTium
SZEGEDI TUDOMÁNYEGYETEM
Szegeci Tudományegyetem

2007. június 6.

86

Újrafelhasználás-orientált

- ▶ Komponens alapú fejlesztés
 - Elérhető, újrafelhasználható komponensek
 - Ezek integrációja
- ▶ Hagyományos modellekkel megegyezik
 - Követelményspecifikáció és validáció
- ▶ Közte levő fázisok eltérnek
 - Komponens elemzés
 - Követelménymódosítás
 - Rendszertervezés újrafelhasználással
 - Fejlesztés és integráció

UNIVERSITAS SCIENTIARUM ET ARTium
SZEGEDI TUDOMÁNYEGYETEM
Szegeci Tudományegyetem

2007. június 6.

87

Komponens alapú fejlesztés

- ▶ Előnyök:
 - Kevesebb fejlesztendő komponens, csökken a költség
 - Gyorsabb leszállítás
- ▶ Hátrányok:
 - Kompromisszumok követelményekkel szemben
 - Evolúció során a felhasznált komponensek új verziói már nem integrálhatók
- ▶ Objektumorientált paradigmája jó alap
 - UML használata
 - Rational Unified Process (RUP) egy iteratív, inkrementális és komponens alapú folyamat

UNIVERSITAS SCIENTIARUM ET ARTium
SZEGEDI TUDOMÁNYEGYETEM
Szegeci Tudományegyetem

2007. június 6.

88

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Formális módszerek

- ▶ Vízesés modellre hasonlít
 - Specifikáció: formális, matematikai apparátus
 - Kidolgozás: ekvivalens transzformációk
 - Venifikáció: hagyományos értelemben nem szükséges
- ▶ Kisebb lépésekkel áll, amelyek finomítják az egyes formális modelleket
 - Könnyebb a formális bizonyítás
- ▶ Speciális területeken alkalmazható
 - Pl. kritikus (al)rendszerknél, ahol elvárt a bizonyítottság
- ▶ Kölcsönhatások nem minden formalizálhatók

2007. június 6.

89

Cleanroom módszer

- ▶ „Fejlesszünk (bizonyítottan) hibátlanul és akkor nem kell tesztelni”
 - Csak rendszertesztes kell, modulhelyesség bizonyított
- ▶ Az egyik legismertebb formális módszer
- ▶ Inkrementális fejlesztésen alapul
- ▶ Fejlesztőszökök egyszerűbbek, szigorúbbak
 - Pl. csak strukturált programnyelvek
- ▶ Dobozokkal reprezentálják a rendszert
- ▶ Képzett, elkötelezettszerű tervezők

2007. június 6.

90

4GT

- ▶ 4GT = negyedik generációs technikák
- ▶ Magas szintű reprezentáció (absztraktió)
 - 4G (vizuális) nyelvek, grafikus jelölés
 - Automatikus kódgenerálás
- ▶ Vizuális eszközök: adatbázis lekérés, riportgyártás, adatmanipuláció, GUI, táblázatkezelés, HTML-oldalak, web, stb.

2007. június 6.

91

4GT (folyt.)

- ▶ Pro:
 - Rövidebb fejlesztési idő
 - Jobb produktivitás
 - Kis és közepes alkalmazásoknál jó
- ▶ Kontra:
 - Vizuális nyelvet nem könnyebb használni
 - Generált kód nem hatékony
 - Karbantarthatóság rosszabb
 - Nagy alkalmazásoknál nem előnyös
- ▶ Komponens alapú technikával alkalmazva még jobb

2007. június 6.

92

Egyéb (aktuális) modellek

- ▶ Kliens/szerver modell
 - Sok területre értelmezhető
 - Kliens adatokat/szolgáltatást kér, szerver szolgáltatja
 - Objektumorientált és komponens alapú paradigmánál definiálják
- ▶ Web fejlesztés
 - Széleskörű felhasználó közönség
 - Hagyományos módszerek + kliens/szerver + 4GT + OO + komponensek
 - Web tartalom és design tervezés is ide tartozik

2007. június 6.

93

Egyéb (aktuális) modellek

- ▶ Nyílt forráskódú fejlesztés
 - Ad hoc fejlesztés
 - Fejlesztési ütemezés, költségvetés nem definiált
 - Nem strukturált folyamat
 - Web kommunikáció
 - Közösségi ellenőrzés
 - Bizalmatlanság megbízhatóság terén
 - Nyílt forráskódú, bárki mérheti a minőséget és javíthat
 - Nem feltétlenül jobb a kereskedelmi termék
 - Viszont (sokszor) ingyenes licensz

2007. június 6.

94

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

2. Tétel

- Projektmenedzsment, Költségbecslés

Tényezők (4P)

- Munkatársak (people)** – a sikeres projekt legfontosabb tényezői
- Termék (product)** – a létrehozandó termék
- Folyamat (process)** – a feladatok, tevékenységek halmaza a munka elvégzése során
- Projekt** – minden olyan tevékenység, ami kell ahhoz, hogy a termék létrejöjjön

2018.12.05.

258

Projekt sikertelenségének okai

- Nem reális a határidők megválasztása
- A felhasználói követelmények változnak
- A szükséges ráfordítások alulbecslése
- Kockázati tényezők
- Technikai nehézségek
- A projekt csapatban nem megfelelő a kommunikáció
- A projekt menedzsment hibái

2018.12.05.

260

Emberek menedzseltése

- Szoftverfejlesztő szervezet legnagyobb vagyona az emberek
 - Szellemi töke
 - Lehető legjobban kamatozzon!
- Sok projekt bukásának legfőbb oka a rossz humánmenedzsment
- Egyik legfontosabb feladat az emberek motivációja
 - Szociális szükséletek, megbecsülés, önmegvalósítás igénye

2018.12.05.

261

Csoportmunka

- Valódi szoftvereket 2-1000 fős csapatok készítik (team)
- Hatókony együttműködés fontos
 - Csoportszellemet kell kialakítani (csoport sikere fontosabb mint az egyéné)
 - Csoportépítés (team building)
 - sociális tevékenységek
- Munkakörnyezet fontos
 - Nyitott és privát tér kombinálása
 - Közös területek

2018.12.05.

262

Csoportmunka (folyt.)

- Befolyásoló tényezők:
 - Csoport összetétele
 - egymást kiegészítő személyiségek
 - nemkívánatos vezető végzetes lehet
 - Csoportösszetartás
 - pl. csoportos programozás
 - Csoportkommunikáció
 - Csoport szerkezete
 - informális szervezés
 - vezető programozó-csoport: kell tartalék programozó és adminisztrátor is

2018.12.05.

263

Csapatfelépítés szempontjai

- A megoldandó probléma nehézsége
- A programok mérete (LOC vagy funkciópont)
- A team működésének időtartama
- A feladat modularizálhatósága
- A létrehozandó rendszer minőségi és megbízhatósági követelményei
- Az átadási határidők szigorúsága
- A projekt kommunikációs igénye

2018.12.05.

264

Csapatfelépítés (folyt.)

- Zárt forma – hagyományos strukturális felépítés
- Véletlenszerű forma – laza szerkezet, egyedi kezdeményezések a döntők
- Nyitott forma – a zárt és a véletlenszerű paradigmák előnyeinek kombinálása
- Szinkronizált forma – az adott probléma felosztása szerint történik a team szervezése. Az egyes csoportok között kevés kommunikáció van

2018.12.05.

265

Emberek kiválasztása

- Különböző tesztekkel történhet
 - Programozási képesség
 - Psichometrikus tesztek
- Sok tényező: alkalmazási terület, platform, programozási nyelv, kommunikációs készség, személyisésg, stb.
- Szakmai karrier megállhat egy szinten, ha vezetői szerepkört kap
 - Azonos értékű kell hogy legyen a szakember és a vezető!

2018.12.05.

266

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Szoftver projekt tervezés

- ▶ Mennyi pénz?
- ▶ Mennyi ráfordítás?
- ▶ Mennyi idő?

...ezeket kell megbecsülni

Munkaköltség

- ▶ Legjelentősebb
 - Általában a fizetés kétszerese körül
- ▶ Fejlesztők fizetése
- ▶ Kisegítő személyzet fizetése
- ▶ Bérleti díj, rezsi
- ▶ Infrastruktúra használat (pl. hálózat)
- ▶ Szórakozási lehetőségek
- ▶ Járulékok, adók
- ▶ Stb.

Termelékenység

- ▶ Ipari rendszerben a legyártott egységek száma / emberórák
- ▶ Szoftvernél nehézkes
 - Egyik kód hatékony, a másik karbantartató, stb.
- ▶ Ezért mérik a szoftver valamely jellemzőjét (metrika)
- ▶ Két típus:
 - Mérő-alapú (pl. programsorok száma)
 - Funkció-alapú (funkciópont, objektumpont)

Méret alapú mérés

- ▶ LOC = Lines Of Code
- ▶ Több technika
 - Csak nem üres sorok
 - Csak végrehajtható sorok
 - Dokumentáció mérete
- ▶ Félrevezető lehet
 - Nem összehasonlítható programozási nyelvek
 - assembly: 700 sor / hónap
 - magas szintű nyelv: 300 sor / hónap

Termék (product)

- ▶ Szoftver hatásköre
 - Környezet
 - Input-output objektumok
- ▶ Probléma dekompozíció

Folyamat (process)

- ▶ A megfelelő folyamat kiválasztása
- ▶ Előzetes projekt terv
- ▶ CPF (common process framework)
 - Felhasználói kommunikáció
 - Tervezés (planning)
 - Kockázat analízis
 - Fejlesztés
 - Release
 - Felhasználói kiértékelés

Szoftverköltség becslése

- ▶ Projekt tevékenységeinek kapcsolódása a munka-, idő- és pénzköltségekhez
- ▶ Becsléseket lehet és kell adni
 - Folyamatosan frissíteni
- ▶ Projekt összköltsége:
 - Hardver és szoftver költség karbantartással
 - Utazási és képzési költség
 - Munkaköltség

Projekt

- ▶ W5HH módszer
 - Miért fejlesztük a rendszert? (why?)
 - Mit fog csinálni? (what?)
 - Mikorra? (when?)
 - Ki a felelős egy funkcióért? (who?)
 - Hol helyezkednek el a felelősök? (where?)
 - Hogyan meg a technikai és menedzsment munka? (how?)
 - Mennyi erőforrás szükséges? (how much?)

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Funkciópont számítás

- ▶ Jobb, de nehezebben határozható meg
- ▶ Nyelv független
- ▶ Rendszer funkcionalitásának „mennyisége”
- ▶ Több programjellemző súlyozott kombinációja
 - Külső bemenetek és kimenetek
 - Felhasználói interaktivitás
 - Külső interfések
 - Használt állományok

Funkciópont számítás (folyt.)

- ▶ Vannak további módosító tényezők
 - Projekt összetettsége
 - Teljesítmény
 - Ezek nagyon szubjektívek
- ▶ Sorok átlagos száma
 - Assembly: 200-300 LOC/FP
 - 4GL: 22-40 LOC/FP

 UNIVERSITAS SCIENTIARUM ET ARTIUM SZEGEDIENSIS

Objektumpontok

- ▶ Nem az osztályok vagy objektumok száma!
- ▶ 4GL nyelvekhez
- ▶ Súlyozott becslés:
 - Megjelenítendő képernyők száma
 - 1-3 pont
 - Elkészített jelentések száma
 - 2-8 pont
 - 3GL modulok száma
 - modulonként 10 pont

 UNIVERSITAS SCIENTIARUM ET ARTIUM SZEGEDIENSIS

Funkciópont és méret

- ▶ Egyéni termelékenység
 - 30-900 sor / programozóhónap
 - 4-50 objektumpont / programozóhónap
- ▶ Nem funkcionális jellemzőket nem veszik figyelembe
 - Pl. újrafelhasználás, teljesítmény
- ▶ Akkor van probléma, ha a vezető a termelékenységi mérésből ítéli meg a fejlesztő képességét

Becslési módszerek

- ▶ Hasonló projektek
- ▶ Egyszerű dekompozíciós technika alkalmazása
- ▶ Algoritmikus módszerek
- ▶ Tapasztalati modellek költség és ráfordítás becslése
- ▶ Szakértői vélemények
- ▶ Parkinson törvénye (munka kitölti az időt)
- ▶ Nyerő ár (amennyit a vevő képes kifizetni)

UNIVERSITAS SCIENTIARUM ET ARTIUM SZEGEDIENSIS
SZEGEDI EGYETEM

Dekompozíciós technikák

- ▶ Szoftver méret (ennek meghatározása a legfontosabb)
 - Fuzzy-logic: approximációs döntési lépések
 - FP méret
 - Szabványos komponens méretek használata
 - Változás alapú méret
 - Létező komponenseket módosítunk

Dekompozíciós technikák (folyt.)

- ▶ Probléma alapú becslés
 - Funkciókra való bontás a lényeges
 - „Baseline” metrikák (alkalmazás specifikus)
 - LOC becslés - dekompozíció a funkciókra
 - FP becslés - dekompozíció az alkalmazás jellemzőire koncentrál
- ▶ Folyamat alapú becslés
 - Meghatározzuk a funkciókat
 - minden funkcióhoz megadjuk a végrehajtandó feladatokat
 - a feladatokra becsüljük a feladatok költségeit

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

| <div style="background-color: #e0e0e0; padding: 10px;"> <p>Tapasztalati becslés modellek</p> <ul style="list-style-type: none"> ▶ Erősen alkalmazás-függő ▶ Általános alak <ul style="list-style-type: none"> ■ $E = A + B * (x)^C$ ■ ahol $x = \text{LOC}$ vagy FP ■ E: emberhónap ■ A, B, C: konstansok </div> | <div style="background-color: #e0e0e0; padding: 10px;"> <p>Tapasztalati becslés (folyt.)</p> <ul style="list-style-type: none"> ▶ LOC modellek <div style="margin-top: 20px;"> <p style="text-align: center;"><small>UNIVERSITAS SCIENTIARUM ET TUDORUM ANVENGETERUM Szegedi Tudományegyetem</small></p> </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div style="width: 45%;"> $E = 5.2 \times (\text{KLOC})^{0.91}$ $E = 5.6 + 0.73 \times (\text{RLOC})^{1.16}$ $E = 3.2 \times (\text{KLOC})^{1.05}$ $E = 6.288 \times (\text{KLOC})^{1.647}$ </div> <div style="width: 45%;"> <p>Walston-Felix model Bailey-Basil model Boehm simple model Dwy model for KLOC > 9</p> </div> </div> </div> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|---|----------------------------|--------------------|------|-----------|--------|-----------|--------|---|---|---|--------|---|---|---|---------------|--|--|----|---|-----------------------------------|----------|-----|-------|------|-----------|---------------------------------|----------|-----|-------|------|-----------|------|---|---|----|----|----|
| <small>2018.12.05.</small> | <small>283</small> | <small>2018.12.05.</small> | <small>284</small> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <div style="background-color: #e0e0e0; padding: 10px;"> <p>Tapasztalati becslés (folyt.)</p> <ul style="list-style-type: none"> ▶ FP modellek <div style="margin-top: 20px;"> <p style="text-align: center;"><small>UNIVERSITAS SCIENTIARUM ET TUDORUM ANVENGETERUM Szegedi Tudományegyetem</small></p> </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div style="width: 45%;"> <ul style="list-style-type: none"> ■ $E = -91.4 + 0.355 \times \text{FP}$ Albrecht és Gaffney ■ $E = -37 + 0.96 \times \text{FP}$ Kemerer regressziós ■ $E = -12.88 + 0.405 \times \text{FP}$ Kis projektek regressziós </div> <div style="width: 45%;"></div> </div> </div> | <div style="background-color: #e0e0e0; padding: 10px;"> <p>Tapasztalati becslés (folyt.)</p> <ul style="list-style-type: none"> ▶ COCOMO modell (Constructive Cost Model) ▶ Iparban a COCOMO 2 használt <ul style="list-style-type: none"> ■ Alap az objektumpontok ■ $PM = (\text{obj.pont})^*[(100\%-\text{újrahaszn})/100] / \text{PROD}$ <ul style="list-style-type: none"> – PM: munkamennyiség (emberhónap) – PROD: termelékenység (obj.pont/hónap) – Táblázatból, 4-50 </div> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <small>2018.12.05.</small> | <small>285</small> | <small>2018.12.05.</small> | <small>286</small> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <div style="background-color: #e0e0e0; padding: 10px;"> <p>COCOMO (folyt.)</p> <div style="margin-top: 20px;"> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th rowspan="2">Object type</th> <th colspan="3">Complexity weight</th> </tr> <tr> <th>Simple</th> <th>Medium</th> <th>Difficult</th> </tr> </thead> <tbody> <tr> <td>Screen</td> <td>1</td> <td>2</td> <td>3</td> </tr> <tr> <td>Report</td> <td>2</td> <td>5</td> <td>8</td> </tr> <tr> <td>SQL component</td> <td></td> <td></td> <td>10</td> </tr> </tbody> </table> </div> <div style="margin-top: 20px;"> <p style="text-align: center;"><small>UNIVERSITAS SCIENTIARUM ET TUDORUM ANVENGETERUM Szegedi Tudományegyetem</small></p> </div> <div style="margin-top: 10px;"> <p>Table 05.01 Complexity weighting for object types [BOE96]</p> </div> </div> | Object type | Complexity weight | | | Simple | Medium | Difficult | Screen | 1 | 2 | 3 | Report | 2 | 5 | 8 | SQL component | | | 10 | <div style="background-color: #e0e0e0; padding: 10px;"> <p>COCOMO (folyt.)</p> <div style="margin-top: 20px;"> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Developer's experience/capability</th> <th>Very low</th> <th>Low</th> <th>Mixed</th> <th>High</th> <th>Very high</th> </tr> </thead> <tbody> <tr> <td>Environment maturity/capability</td> <td>Very low</td> <td>Low</td> <td>Mixed</td> <td>High</td> <td>Very high</td> </tr> <tr> <td>PROD</td> <td>4</td> <td>7</td> <td>13</td> <td>23</td> <td>38</td> </tr> </tbody> </table> </div> <div style="margin-top: 20px;"> <p style="text-align: center;"><small>UNIVERSITAS SCIENTIARUM ET TUDORUM ANVENGETERUM Szegedi Tudományegyetem</small></p> </div> <div style="margin-top: 10px;"> <p>Table 05.02 Productivity rates for object points [BOE96]</p> </div> </div> | Developer's experience/capability | Very low | Low | Mixed | High | Very high | Environment maturity/capability | Very low | Low | Mixed | High | Very high | PROD | 4 | 7 | 13 | 23 | 38 |
| Object type | | Complexity weight | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Simple | Medium | Difficult | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Screen | 1 | 2 | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Report | 2 | 5 | 8 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SQL component | | | 10 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Developer's experience/capability | Very low | Low | Mixed | High | Very high | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Environment maturity/capability | Very low | Low | Mixed | High | Very high | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PROD | 4 | 7 | 13 | 23 | 38 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <small>2018.12.05.</small> | <small>287</small> | <small>2018.12.05.</small> | <small>288</small> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <div style="background-color: #e0e0e0; padding: 10px;"> <p>Tapasztalati becslés (folyt.)</p> <ul style="list-style-type: none"> ▶ Dinamikus modell <ul style="list-style-type: none"> ■ $E = [\text{LOC} * B^{0.333}/P]^3 * (1/t^4)$ ■ E: pm ráfordítás ■ B: speciális képzettségi igény ■ P: produktivitási paraméterek ■ t: projekt időtartam </div> | <div style="background-color: #e0e0e0; padding: 10px;"> <p>A szoftverminőség</p> <ul style="list-style-type: none"> ▶ mindenki célja: termék vagy szolgáltatás minőségének magas szinten tartása ▶ Minőség klasszikus definíciója: <ul style="list-style-type: none"> ■ Termék feleljen meg a specifikációknak ■ Szoftverminőség esetében nem ilyen egyszerű <ul style="list-style-type: none"> ■ Fejlesztőnek is lehetnek belső igényei, pl. karbantarthatóság ■ Egyes jellemzőket nem könnyű specifikálni, pl. karbantarthatóság ■ Ritka a részletes és pontos követelményspecifikáció </div> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <small>2018.12.05.</small> | <small>289</small> | <small>2018.12.05.</small> | <small>290</small> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

| | |
|--|--|
| <p>CMM(I): a szoftver folyamat mérése</p> <ul style="list-style-type: none">▶ Capability Maturity Model (Integration)▶ Cél: a szoftverfejlesztési folyamat hatékonyságának mérése▶ Egy szervezet megkaphatja valamely szintű minősítését<ul style="list-style-type: none">■ Speciálisabb az ISO minőség-szabványnál▶ 5 besorolási szint<ul style="list-style-type: none">■ Fölsőbb szintek magukban foglalják az alsóbb szinteket | <p>CMM besorolási szintek</p> <ul style="list-style-type: none">▶ 1. Kezdeti: csak néhány folyamat definiált, a többségük esetleges (Alapszint)▶ 2. Reprodukálható: az alapvető projekt menedzsment folyamatok definiáltak. Költség, ütemezés, funkcionális kezelése megoldott és megismételhető. (Bevezési szint)▶ 3. Definiált: a menedzsment és a fejlesztés folyamatai is dokumentáltak és szabványosítottak az egész szervezetre. (Végelegesítési szint) |
| <p>2018.12.05.</p> | <p>201</p> |
| <p>CMM (folyt.)</p> <ul style="list-style-type: none">▶ 4. Ellenőrzött: a szoftver folyamat és termék minőségének részletes mérése, ellenőrzése. (Bevezetési szint)▶ 5. Optimalizált: a folyamatok folytonos javítása az új technológiák ellenőrzött bevezetésével (Optimalizálási szint) <p>A nagyobb szintknél teljesülni kell a korábbi szintek követelményeinek is</p> | <p>CMM (folyt.)</p> <ul style="list-style-type: none">▶ Bizonyos tevékenységeket végre kell hajtani az egyes szinteknél:<ul style="list-style-type: none">■ 2. Reprodukálható:<ul style="list-style-type: none">- Konfigurációs menedzsment- Szoftver minőség biztosítás- Alvvállalkozói menedzsment- Projekt tervezés(planning)- Kovetelmény menedzsment- Projekt követés és felügyelet |
| <p>2018.12.05.</p> | <p>202</p> |
| <p>CMM (folyt.)</p> <ul style="list-style-type: none">■ 3. Definiált:<ul style="list-style-type: none">- Alapos átnézések (review)- Csoportok közötti koordináció- Termék menedzsment- Integrált szoftver menedzsment- Tréning programok- Szervezeti folyamat definíció- Szervezet folyamat összpontosítás | <p>CMM (folyt.)</p> <ul style="list-style-type: none">■ 4. Ellenőrzött:<ul style="list-style-type: none">- Szoftver minőség menedzsment- Mennyiségi folyamat menedzsment■ 5. Optimalizált:<ul style="list-style-type: none">- Folyamat változás menedzsment- Technológiai változás menedzsment- Hiba megelőzés |
| <p>2018.12.05.</p> | <p>203</p> |
| <p>Konfigurációkezelés</p> <ul style="list-style-type: none">▶ A rendszer változásainak kezelése<ul style="list-style-type: none">■ Változások felügyelt módon történjenek■ Eljárások és szabványok fejlesztése és alkalmazása▶ Fejlesztés, evolúció, karbantartás miatt van rá szükség▶ Sokszor hiba-követéssel egybekötött▶ Verziók kezelése<ul style="list-style-type: none">■ Változtatási javaslatok■ Hibajavítások■ Adaptáció (pl. platform)■ Egy időben több „élő” verzió | <p>Változások forrásai</p> <ul style="list-style-type: none">▶ Új piaci feltételek▶ Vásárló, megrendelő új követelménye▶ Szervezet újraszervezése (pl. felvásárlás)■ Új belső szabalyozások, integráció■ Megváltozott költségvetés▶ Új platform támogatása▶ Stb. |
| <p>2018.12.05.</p> | <p>204</p> |

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

| | |
|--|--|
| <p>Konfigurációkezelés (folyt.)</p> <ul style="list-style-type: none">▶ Minőségskezelés része<ul style="list-style-type: none">■ CMM elvárt tevékenysége▶ Szoftver változatok<ul style="list-style-type: none">■ Verziók (<i>version, revision</i>)■ Kiadások (<i>release</i>)■ Alapvonal (<i>baseline, mainline, trunk</i>)■ Fejlesztési ágak (<i>branch</i>)▶ Különböző megoldások kellenek<ul style="list-style-type: none">■ Vízesés modell■ Inkrementális modell (napi verzió kiadás) <p>2018.12.05. 309</p> | <p>Konfigurációkezelési tevékenységek</p> <ul style="list-style-type: none">▶ Konfigurációkezelés megtervezése▶ Változtatások kezelése (<i>change management</i>)<ul style="list-style-type: none">■ Hibák nyomkövetése is■ Eszközök: ürlapszerkesztő, folyamat-szervező, adatbázis▶ Verzió- és kiadáskezelés▶ Rendszerépítés (build)▶ Érvényesítés és felülvizsgálat (audit, review) <p>2018.12.05. 310</p> |
| <p>Konfigurációs adatbázis</p> <ul style="list-style-type: none">▶ Mindent tárol:<ul style="list-style-type: none">■ Forráskód, (bináris kód), dokumentumok■ Építési folyamat, szkriptek■ Hiba adatbázis■ Változtatások története■ Verziók▶ Fentiek összekötése▶ Lekérdezhetőség <p>2018.12.05. 301</p> | <p>Verziókezelés</p> <ul style="list-style-type: none">▶ Verziók és kiadások nyomon követése▶ minden verzió bármikor kinyerhető▶ Általában különálló fejlesztő csoport foglalkozik vele▶ Verzió:<ul style="list-style-type: none">■ Rendszer egy példánya: új funkciók, kijavított hibák, más platform (sokszor nem új számú)■ Al-verziók: variánsok▶ Kiadás (<i>Release</i>):<ul style="list-style-type: none">■ Nagyobb változat, amely eljut a felhasználóhoz is <p>2018.12.05. 302</p> |
| <p>Verzióazonosítás</p> <ul style="list-style-type: none">▶ Számozás (legelterjedtebb)<ul style="list-style-type: none">■ Alapvonal: v1.0, v1.1, v1.2, ...■ Karbantartási ág: v1.1.1 (párhuzamosan élhet v1.2-vel)■ Fejlesztési ág<ul style="list-style-type: none">— párhuzamosan él, egy idő után összefűsül az alapvonallal■ v2.0 már új kiadás■ Hozzá lehet még rakni a build sorszámát, dátumát■ Problémája, hogy nehezen rendelhetjük össze magukkal a változtatásokkal▶ Attribútum alapú▶ Változtatásorientált <p>2018.12.05. 303</p> | <p>Eszközök verziókezeléshez</p> <ul style="list-style-type: none">▶ minden változtatás figyelése és rögzítése<ul style="list-style-type: none">■ Csak a követett tételeken, mint pl. forrásfájlok▶ Van központi adatbázis (<i>repository</i>)▶ Fejlesztés munkaváltozaton történik<ul style="list-style-type: none">■ Új létrehozása (<i>checkout</i>)■ Saját változatunk frissítése (<i>update</i>)▶ Módosítások közzététele, adatbázis frissítése<ul style="list-style-type: none">■ <i>commit</i>■ Automatikusan új verzió jön létre <p>2018.12.05. 304</p> |
| <p>Eszközök verziókezeléshez (folyt.)</p> <ul style="list-style-type: none">▶ Szolgáltatások:<ul style="list-style-type: none">■ Verzióazonosítás: revision number■ Tárolás: csak változtatások tárolása (delta)<ul style="list-style-type: none">— csak szöveges fájlok!■ Változtatástörténet rögzítése■ Független fejlesztés: commit után összeveti más fejlesztők módosításával<ul style="list-style-type: none">— ütközésekkel kell összefűsülni■ Lekérdezések: változások lekérése, „blame”▶ Kereskedelmi: SourceSafe, ClearCase, CCC, LIFESPAN, DSEE, Rational▶ Szabad: RCS, CVS, Subversion <p>2018.12.05. 305</p> | <p>Kiadáskezelés</p> <ul style="list-style-type: none">▶ Kiadásmenedzserek elődöntik, mikor lesz verzióból kiadás<ul style="list-style-type: none">■ Fontos rögzíteni az adott verziót (forrásokat is!)▶ Költséges folyamat a pusztta fejlesztésen felül<ul style="list-style-type: none">■ Pl. marketing▶ Egy kiadás tartalmazza:<ul style="list-style-type: none">■ Konfigurációs állományok■ Adatállományok■ Telepítő program (CD, Web)■ Eletronikus és papír dokumentáció■ Licencszerződés■ Csomagolás és propagandaanyagok <p>2018.12.05. 306</p> |

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Rendszerépítés

- Software build process
- Komponensek fordítása és szerkesztése
- Komponensek (és fájlok) között építési függőségek vannak
- Nagy rendszernél bonyolult a folyamat
 - Hosszadalmas is, ezért inkrementálisan kell végezni
- Automatizálni kell: építési szkriptek
 - configure, make

Eszköözök építéshez

- Inkrementális építés:
 - make (UNIX) és leszármazottai
 - Konfigurációkezelő része
 - Intégrált fejlesztőszokoz részeként
- Szolgáltatók:
 - Függőségspecifikáló nyelv (pl. makefile)
 - Eszközkiválasztás (fordítóprogram), beállítások
 - Osztoz forrás
 - Szarmaztatott objektumok inkrementális létrehozása
 - utolsó módosítás dátuma alapján, csak akkor, ha a forrás frissebb mint a cél

Hibamenedzsment

- Hiba-követés
 - Fontos, mert sok hiba van/lesz: kategorizálás, prioritások felállítása, követés elengedhetetlen
- Általáosabb: változtatás-menedzsment
 - CR (change request) adatbázis nyilvántartása
 - Ki kezdeményezi a változtatást
 - Milyen jellegű (hibabejelentés, új feature, ...)
 - CR életútja: analízis, hatásanalízis, költségbecsles, kiadás kijelölése, megvalósítás, teszt környezet bővítése, utólagos követés

Hibák követése

- Hibaadatbázis
- Minden hibának egyedi azonosítója van
- További adatok:
 - Bejelentő
 - Kijelölt felelős személy, megfigyelők listája
 - Dátum
 - Rövid összegzés
 - Súlyosság: pl. triviális, kicsi, nagy, kritikus
 - Prioritás: skála szerint

Hibaadatbázis (folyt.)

- További adatok:
 - Státsz: bejelentett, ellenőrzött, kiosztott, megoldott, lezárt, újra megnyitott
 - Megoldási mód: javítva, érvénytelen, nem lesz javítva, duplikánum, nem reprodukálható
 - Platform, operációs rendszer
 - Termék, komponens, verziósázm
 - Függőségek más hibákkal
 - Részletes leírás (reprodukálás lépései)
 - Csatolmányok (pl. példa inputok)
- Fontos a hiba életútjának rögzítése

Eszköözök, szolgáltatásai

- Automatikus e-mail küldése bármilyen változás esetén
- Összekötés a verziókövető rendszerrel
 - Ha egy commit hibajavítás, meg lehet jelölni, melyik hibát javítja, pl. \$Bug=51
- Web alapú, önálló program, környezetbe integrált
- Projektmenedzsment rendszerek része, pl. Microsoft, IBM Rational
- Nyílt forráskódú: Bugzilla, IssueZilla, TRAC, ...

Szoftvermérés

Szoftvermérés, metrikák

- Szoftvermérés: termék vagy folyamat valamely jellemzőjét numerikusan kifejezni (metrika)
- Ezen értékekből következtetések vonhatók le a minőségre vonatkozóan
 - Pl. szoftvertesztelő eszköz bevezetése előtti és utáni hibagyakoriság
- „Amit nem mérünk, azt vezérelni sem tudjuk.” (Tom DeMarco)

Metrikák (folyt.)

- Két csoport:
 - Vezérlési metrikák. Folyamattal kapcsolatosak, pl. egy hiba javításához szükséges átlagos idő (folyamat és projekt metrikák)
 - Prediktor metrikák. Termékkel kapcsolatosak, pl. LOC, ciklomatikus komplexitás, osztály metódusainak száma (termék metrikák)
- Mindkettő befolyásolja a vezetői döntéshozatalt
- Frappáns analógia:
 - Leletkészítés → Diagnózis → Kezelés (vagy Megelőzés)

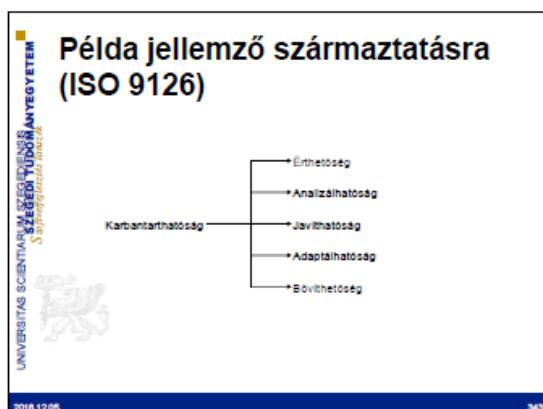
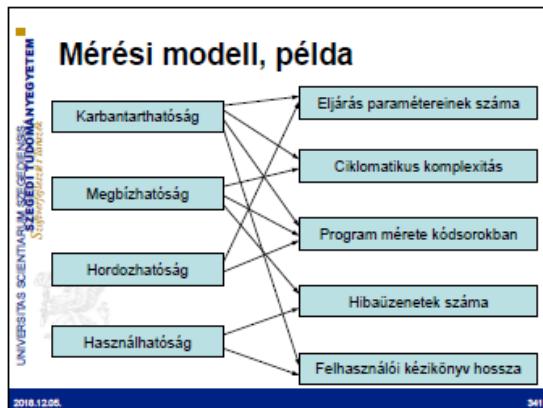
Minőségi jellemzők mérése

- Jellemzőket lehetetlen közvetlenül mérni
 - Magasabb szintű absztraktiók, sok mindenről függnek
 - Hierarchikus összetétel (jellemzők származtatása)
 - Sokszor szervezet- vagy termékfüggő
 - Több metrika együttes vizsgálata
 - Metrikák változása az idő függvényében
 - Statisztikai technikák alkalmazása
- Metrikák (belsı jellemzö) és (külsö) jellemzök közötti kapcsolatokra fel kell állítani egy modellt
- Sok projekt esettanulmányának vizsgálata

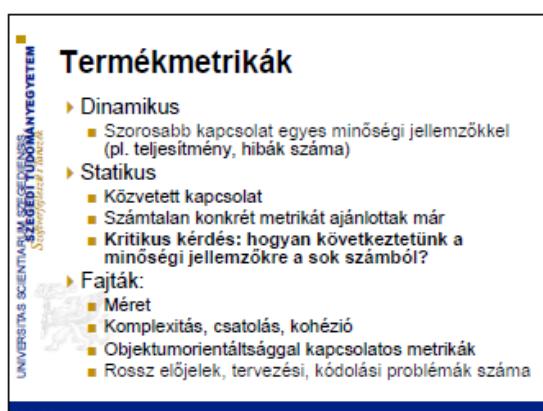
156

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

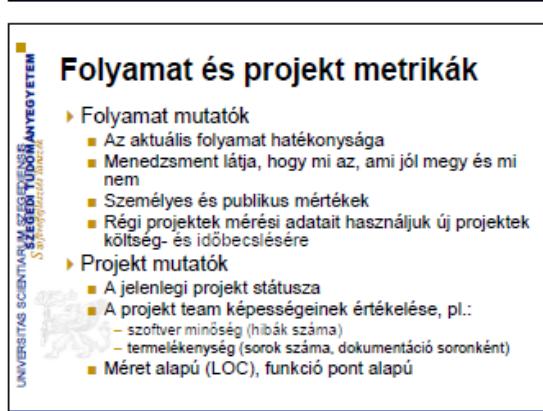
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK



- Mérési folyamat**
- ▶ Alkalmazandó mérések kiválasztása
 - ▶ Mérni kívánt komponensek kiválasztása
 - ▶ Mérés (metrika számítása)
 - ▶ Magasabb szintű jellemző meghatározása modell alapján
 - ▶ Rendellenes értékek összehasonlítása
 - Korábbi mérésekkel szemben
 - Ugyanaz a projekt
 - Más projektek
 - Ezen történeti adatbázis nagyon értékes szellemi tulajdon
 - ▶ Rendellenes komponensek részletes vizsgálata
- 2018.12.05. 344



- Tesztelés „mérése” és fejlesztése**
- ▶ Tesztelési környezet is minőségvizsgálatra szorul
 - Az is a szoftvernek egy fontos entitása!
 - ▶ Metrikák, amiket definiálhatunk:
 - Lefedettség: adott változatnál hány %-át érintik a tesztek
 - Regressziós teszt hatékonysága
 - Teszesetek redundanciája
 - ▶ Tesztelési/fejlesztési költségek becsülhetővé válnak!
 - ▶ Tesztelés továbbfejlesztése
 - Tesztek prioritizálása
 - Tesztek szűrése
 - Regressziós teszt támogatása
 - csak az érintett részeket teszteljük újra
 - csak a releváns teszesetek futtatása, nem az összes!
 - ▶ Súlyos összegek takaríthatók meg!
- 2018.12.05. 346



- Folyamat és projekt metrikák (folyt.)**
- ▶ A minőségi és hatékonysági adatok gyűjtése
 - ▶ Összevetés a korábbi mérési adatokkal
 - ▶ Trendek figyelése
 - ▶ Az adatokat normalizálni kell
 - ▶ Megalapozottabb becslések készíthetők
 - ▶ „Software metrics let you know when to laugh and when to cry” (Tom Gilb)
- 2018.12.05. 348

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Folyamat és projekt metrikák (folyt.)

- A fő okok folyamat és projekt metrikák számítására:
 - Jellemzők a projekteket és „baseline”-okat készítünk a további kiértékelésekhez
 - Az aktuális állapotot összevetjük a tervel
 - A mérések alapján becsléseket készíthetünk (költség, ütemezés, minőség)
 - Javíthatjuk a folyamat, projektek hatékonyságát
- Nagyon alaposan meg kell választani, hogy milyen mérést tekintünk metrikának
- A metrika olyan mutató, ami bepillantást nyújt a szoftver folyamatba (projektbe)

2018.12.05. 349

Folyamat és projekt metrikák (folyt.)

- „Not everything that can be counted counts, and not everything that counts can be counted” (Albert Einstein)
- Folyamat mutatók
 - Megmutatják az aktuális folyamat hatékonyságát
 - Menedzsment látja, hogy mi az, ami jól megy és mi nem
 - Ezeket a mutatókat hosszú távon minden projektre kell gyűjteni

2018.12.05. 350

Folyamat és projekt metrikák (folyt.)

- Projekt mutatók
 - A jelenlegi projekt státusa
 - Kockázatok és kritikus problémák feltárása
 - A projekt team képességeinek értékelése (pl. a szoftver minőség szempontjából)
- Sok metrika használható mind folyamat mind pedig projekt mutatónak is

2018.12.05. 351

A szoftver folyamat javítása

The diagram consists of a large circle divided into four quadrants by a horizontal and vertical axis. The top-left quadrant is labeled 'Customer characteristics'. The top-right quadrant is labeled 'Business conditions'. The bottom-left quadrant is labeled 'People'. The bottom-right quadrant is labeled 'Technology'. The center of the circle is labeled 'Process'. The outer boundary of the circle is labeled 'Development environment'.

Figure 04.01
Determinants for software quality and organizational effectiveness (adapted from IPA94).

2018.12.05. 352

Szoftverfolyamat javítása (folyt.)

- Az alapvető cél a minőség és a hatékonyság növelése
 - Ennek a szoftver folyamat csak az egyik eleme
- A technológia és a termékek bonyolultsága is befolyásoló tényező
- A minőség és hatékonyság szempontjából a legfontosabb tényező a munkatársak képzettsége és motiváltsága

2018.12.05. 353

Szoftverfolyamat javítása (folyt.)

- Személyes metrikák
 - Hiba riportok
 - Sorok száma modulonként
 - PSP (Personal Software Process) – személyre szabott folyamat
- Publikus metrikák
 - Projekt szinten összegzett metrikák

2018.12.05. 354

Szoftverfolyamat javítása (folyt.)

- Metrikák használata
 - Ne egyén, hanem szervezet szinten interpretáljuk
 - A metrikák céljáról egyeztetni kell a fejlesztői csapattal
 - Figyelni kell az egymással kapcsolatban álló metrikák értelmezésére
 - Ne használunk metrikát arra, hogy elmarasztalunk valakit

2018.12.05. 355

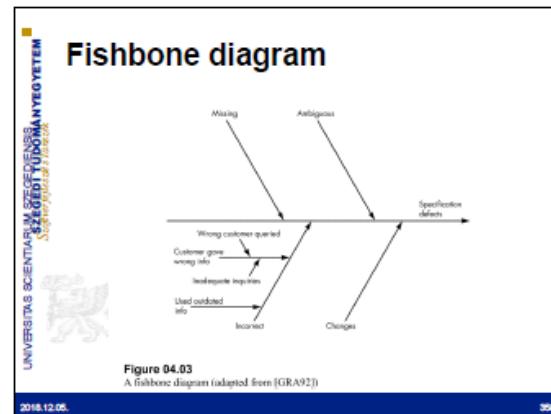
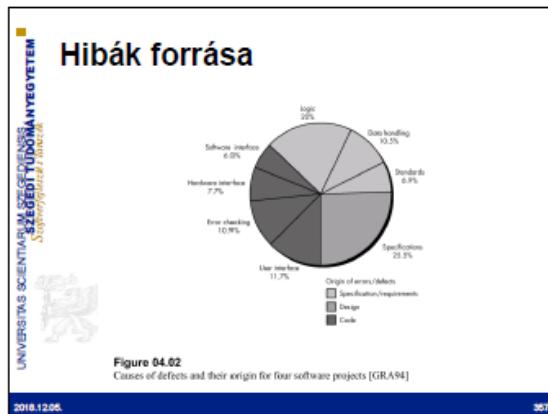
Szoftverfolyamat javítása (folyt.)

- Hiba analízis
 - Hibák eredetének kategorizálása
 - Hibák javítási költségei
 - Kategóriák szerinti összesítések
 - Hibaforrások és költségek elemzése
 - Kritikus költségű hibák azonosítása
 - A szoftver folyamat javítása
- (SSPI) Statistical Software Process Improvement

2018.12.05. 356

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK



Projekt metrikák

- Régi projektek mérési adatait használjuk új projektek költség- és időbecslésére
- Hatékonyúság (funkció pontok, dokumentációs oldalak, LOC)
- Minőség (hibák szoftverfejlesztési feladatonként)
- Egy másik modell
 - Input (az erőforrások mérése)
 - Output (a létrejött termék mérése)
 - Eredmény (a létrejött termék 'átadandó' használhatósága)
- Projekt menedzser használja ezeket a metrikákat

2018.12.05. 359

Méret alapú metrikák

| Project | LOC | Effort | \$/000 | Pp. doc. | Errors | Defects | People |
|---------|--------|--------|--------|----------|--------|---------|--------|
| alpha | 12,100 | 34 | 108 | 368 | 136 | 28 | 3 |
| beta | 27,200 | 42 | 440 | 1224 | 88 | 64 | 5 |
| gamma | 20,200 | 43 | 314 | 1050 | 231 | 6 | |
| * | * | * | | | * | | |
| • | • | • | | | • | | |
| • | • | • | | | • | | |

Figure 04.04
Size-oriented metrics

2018.12.05. 360

Méret alapú metrikák (folyt.)

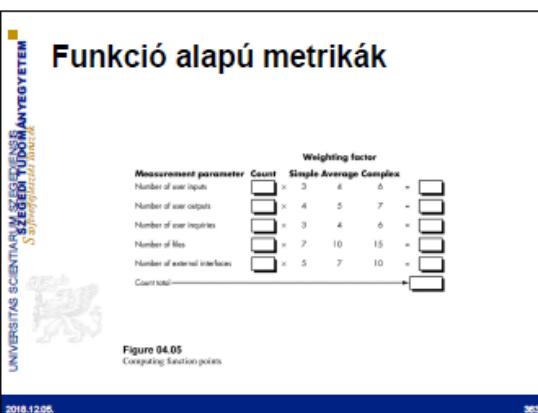
- Hibák / KLOC
- Defekt / KLOC
- Költség / LOC
- Dokumentációs oldalak / KLOC
- Hibák / emberhónap
- LOC / emberhónap
- Költség / dokumentációs oldal

2018.12.05. 361

Méret alapú metrikák (folyt.)

- Széleskörűen használják ezeket a metrikákat, de nagyon sok vita van alkalmazásokról
- Könnyű számítani
- Programozási nyelv és kódolási stílus függő

2018.12.05. 362



Funkció alapú metrikák (folyt.)

- Felhasználói inputok száma - alkalmazáshoz szükséges adatok
- Felhasználói outputok száma - riportok, képernyők, hibaüzenetek
- Felhasználói kérdések száma - on-line input és output
- Fájlok száma - adatok logikai csoportja
- Külső interfések száma - az összes gépi interfész (pl. adatfájlok), ami adatokat továbbít

2018.12.05. 364

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Funkció alapú metrikák (folyt.)

- ▶ Az aktuális szoftver bonyolultsági kategorizálása szubjektív
- ▶ Funkció pont számítása:
 - $FP = \text{Count total} \times [0.65 + 0.01 \times \sum (F_j)]$
- ▶ Az F_j ($j=1 \dots 13$) a bonyolultságot befolyásoló tényezők
- ▶ A tényezők számításához kérdések:
 - A rendszer megköveteli-e a biztonsági mentéseket és helyreállításokat?
 - Adatkommunikáció szükséges-e?

2018.12.05.

365

Funkció alapú metrikák (folyt.)

- A forráskód újrafelhasználhatóra lett tervezve?
- A konverzió és az installáció a tervezés része?
- A rendszer többszöri installációra lett tervezve különböző szervezeteknél?
- Változások támogatása lett tervezve?
- ▶ minden kérdést 0-5 skálán pontozunk

2018.12.05.

367

Funkció alapú metrikák (folyt.)

- Kritikus-e a hatékonyság?
- A rendszer intenzíven használt környezetben működik?
- Van on-line adatbevitel?
- Az on-line adatbevitelhez szükség van összetett képernyő kezelésre?
- A fájlok aktualizálása on-line módon történik?
- Bonyolultak az inputok, outputok, fájlok vagy lekérdezések?
- Bonyolult a belső feldolgozás?

2018.12.05.

366

Kiterjesztett FP metrikák

- ▶ Az eredeti FP mérték információs rendszerekre lett tervezve
- ▶ Egyéb rendszereknek kiegészítésekre van szükség
- ▶ 3D Funkció pont mérték
 - Adat dimenzió (korábbiak szerint)
 - Funkcionális dimenzió – a belső műveletek (transzformációk) száma
 - Vezérlési dimenzió – átmenetek állapotok között. Pl. telefonnál automatikus hívás állapotba pihenő állapotból

2018.12.05.

368

3D mérték (folyt.)

- ▶ Számítás:
 - $\text{Index} = I+O+Q+F+E+T+R$
 - I=input
 - O=output
 - Q=lekérdezés
 - F=fájlok
 - E=külső interfész
 - T=transzformáció
 - R=átmenetek
- ▶ Súlyozott bonyolultsági értékek

2018.12.05.

369

3D mérték (folyt.)

| Processing steps | Semantic statements | | |
|------------------|---------------------|---------|---------|
| | 1-5 | 6-10 | 11+ |
| I-10 | Low | Low | Average |
| 11-20 | Low | Average | High |
| 21+ | Average | High | High |

Figure 54.06

Determining the complexity of a transformation for 3D function points [WIBH].

2018.12.05.

370

Funkció pont mérték (folyt.)

- ▶ FP programozási nyelv független
- ▶ Hátránya, hogy sok szubjektív elemet tartalmaz, nincs konkrét fizikai jelentése
- ▶ FP és LOC összehasonlítása (csak nagyon durva becslések léteznek – LOC alapján ne számítsunk FP-t)

2018.12.05.

371

FP és LOC összehasonlítása

| Programozási nyelv | LOC/FP (átlag) |
|--------------------|----------------|
| Assembly | 320 |
| C | 128 |
| COBOL | 106 |
| Pascal | 90 |
| C++ | 64 |
| Visual Basic | 32 |
| Smalltalk | 22 |
| SQL | 12 |

2018.12.05.

372

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Mérték használata

- LOC/emberhónap, FP/emberhónap: ne használjuk személyek teljesítményének mérésére (sok egyéb szempont is közrejátszik a tényleges hatékonyságban)
- Költség és ráfordítás becslésre megfelelők, de kell egy hosszabb idejű, több projekten alapuló összehasonlítási alap (*baseline*)

Metriák alkalmazása szoftverminőség mérésére

- Elsődleges a hibák és hiányosságok mérése
- A minőség mérése
 - Helyesség (hiányosság/KLOC)
 - Hiányosság: az átadás után feltárt probléma
 - Karbantarthatóság – mennyire könnyen javítható a rendszer. Nincs direkt mérőszám. MTTC – (mean time to change)

A minőség mérése (folyt.)

- Integritás: külső támadások elleni védelem
 - Fenyegetettség: annak valószínűsége, hogy egy adott típusú támadás bekövetkezik egy adott időszakban
 - Biztonság: annak valószínűsége, hogy egy adott típusú támadást visszaver a rendszer
 - Integritás = $\Sigma [1-(fenyegetettség \times (1-biztonság))]$ (Osszegzés a különböző támadás típusokra történik)

A minőség mérése (folyt.)

- Használhatóság – a felhasználó barátság mérése
 - Milyen képzettség kell a rendszer használatához?
 - Mennyi idő alatt lehet hatékony felhasználóvá válni?
 - Mennyire növeli a produktivitást?
 - Szubjektív kiértékelés (kérdőív)

A minőség mérése (folyt.)

- DRE (defect removal efficiency)
 - DRE = E/(E+D), ahol E olyan hibák száma, amelyeket még az átadás előtt felfedezünk, D pedig az átadás után a felhasználó által észlelt hiányosságok száma
- Cél a DRE növelése(minél több hiba megtalálása az átadás előtt)
- Fontos, hogy a hibákat a fejlesztés minél korábbi fázisában találjuk meg (analízis, tervezés)

Metrikák integrálása a folyamatba

- Néhány tipikus kérdés, amikre metrikákkal tudunk válaszolni:
 - Milyen felhasználói igények változnak a leggyakrabban?
 - A rendszer melyik komponensében várható a legtöbb hiba?
 - Mennyi tesztelést tervezünk a komponensekre?
 - Mennyi és milyen típusú hibát várhatunk el a tesztelés kezdetekor?

Metrikus baseline

- Korábbi projektek adatai alapján
 - Pontosság
 - Nagy számú projekt
 - Konzisztems mérések (LOC értelmezés)
 - Alkalmazások közötti hasonlóság

Metrikák gyűjtésének folyamata

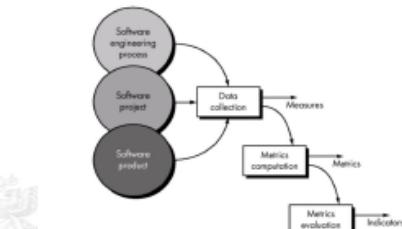


Figure 04.07
Software metrics collection process

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Statisztikai folyamat vezérlés

- ▶ Statisztikailag érvényes trendek megállapítása
- ▶ Vezérlési diagramm(metrikák stabilitásának mérése)
 - E_R =hibák száma/ellenőrzésre fordított idő

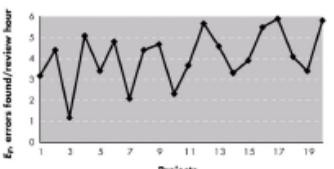


UNIVERSITAS SCIENTIARUM ET ARTIUM SZEGEDI TUDOMANYEGYETEM
Szegedi Tudományegyetem

2018.12.06.

381

Statisztikai folyamat vezérlés (folyt.)



| Projects | E_R errors found/review hour |
|----------|--------------------------------|
| 1 | 4 |
| 2 | 1 |
| 3 | 5 |
| 4 | 3 |
| 5 | 4 |
| 6 | 2 |
| 7 | 1 |
| 8 | 3 |
| 9 | 4 |
| 10 | 2 |
| 11 | 3 |
| 12 | 4 |
| 13 | 2 |
| 14 | 3 |
| 15 | 4 |
| 16 | 5 |
| 17 | 3 |
| 18 | 4 |
| 19 | 2 |

Figure 04.08
Metrics data for errors uncovered per review hour

2018.12.06.

382

Statisztikai folyamat vezérlés (folyt.)

- ▶ A változási tartomány mérése
 - mR bar – középvonal,UCL – felső vezérlési korlát

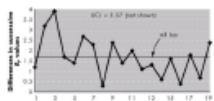


Figure 04.09
Moving range control chart

2018.12.06.

383

Metrikák kis szervezetekre

- ▶ Konkrét célkitűzéseket kell tenni pl. „csökkentsük a változások implementációs idejét és költségét”
- ▶ Metrikák
 - Mennyi idő alatt valósul meg egy változtatás kérés kiértékelése
 - Mennyi a ráfordítás igénye a kiértékelésnek
 - Változtatás implementációs ideje
 - Változtatás implementációs ráfordítás
 - A változtatás ideje alatt felfedezett hibák száma
 - A változtatás után felfedezett hibák száma

2018.12.06.

384

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Számítógép-hálózatok

1. Tétel
 - Számítógép-hálózati architektúrák, szabványosítók (ISO/OSI, Internet, ITU, IEEE)
2. Tétel
 - Kiemelt fontosságú kommunikációs protokollok (PPP, Ethernet, IP, TCP, http, RSA)

számháló_full → PDF

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Számítógép architektúra

1. Tétel

- Neumann-elvű gép egységei

Neumann-elvű gép

Neumann-nak volt egy olyan észrevétele, hogy a korabeli (20. század közepe) számítógépek programozása nagyon lassú és fáradtságos, mivel a folyamat nagyszámú kapcsolókkal és kábelekkel történt. Azt is felismerte, hogy a program digitális formában ugyanúgy tárolható a számítógép memoriájában, mint az adatok, továbbá „szerencsétlennek” tartotta a soros decimális aritmetikát. Ezt a tervezetet ismerjük Neumann-gépként.

A gépnak öt építőköve volt:

-memória

- o 4096 szóból, szavanként 40 bitból állt

- o minden szó vagy két 20 bites utasítást, vagy egy 40 bites előjeles számot tárolt

o az utasításokban 8 bit az utasítás típusát, 12 bit pedig a 4096 memóriaszó egyikét azonosította

- aritmetikai-logikai egység

- o volt egy speciális 40 bites belső regisztere, az akkumulátor

o egy utasítás az akkumulátor tartalmához hozzáadta egy memóriaszó tartalmát, vagy az akkumulátor tartalmát a memóriába tárolta

- o a gépben nem volt lebegőpontos aritmetika

- vezérlőegység

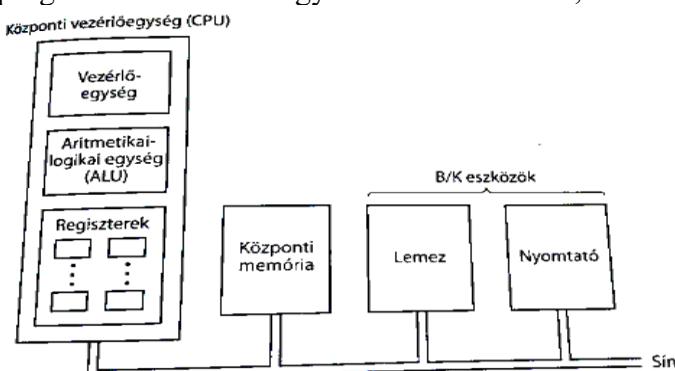
- o az aritmetikai-logikai egység és a vezérlőegység alkotta a számítógép agyát

- bemeneti és kimeneti eszközök

- CPU, adatút, utasítás-végrehajtás, utasítás- és processzorszintű párhuzamosság

CPU (Central Processing Unit)

Feladata az, hogy a központi memóriában tárolt programokat végrehajtsa úgy, hogy a programok utasításait egymás után beolvassa, értelmezi és végrehajtja.



12. ábra - Egy egyszerű, egy processzorból és két I/O egységből álló számítógép felépítése

A vezérlőegység feladata az utasítások beolvasása a központi memóriából és az utasítások típusának megállapítása. Az aritmetikai-logikai egység a program utasításainak végrehajtásához szükséges műveleteket végez.

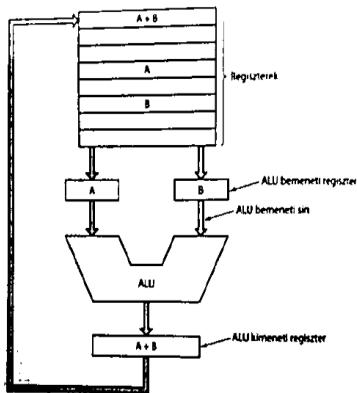
A CPU egy kisméretű, gyors memóriát is tartalmaz, amelyben részeredményeket és bizonyos vezérlőinformációkat tárol. Ez a memória több regiszterből áll, mindegyiknek meghatározott mérete és funkciója van. minden regiszter képes tárolni egy számot, amelynek az értéke kisebb a regiszter mérete által meghatározott maximummal.

Adatút

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Egy tipikus Neumann-elvű számítógép adatútja:



Az adatút részei a regiszterek, azt ALU és az ezeket összekötő néhány sín, A regiszterek két ALU bemeneti regiszterbe csatlakoznak, ezeket az ábra A-val és B-val jelöli. Ezek a regiszterek tárolják a bemeneti adatokat, amíg az ALU más számításokon dolgozik.

Maga az ALU a bemenő adatokon összeadást, kivonást és egyéb egyszerű műveleteket végez, és az eredményt a kimeneti regiszterbe teszi. Ennek a kimeneti regiszternek a tartalma visszaírható egy regiszterbe. Később, ha szükséges, a regiszter tartalma beírható a memóriába.

A legtöbb utasítás a következő két kategória egyikébe sorolható:

-regiszter – memória

o szavakat tölthetünk át a memóriából regiszterekbe, ahol a soron következő utasítások pl. az ALU bemeneteként használhatják

o visszaírhatjuk a regiszterek tartalmát a memóriába

- regiszter – regiszter

o egy tipikus utasítás vesz két operandust a regiszterekből. elhelyezi őket az ALU bemeneti regisztereibe, az ALU elvégez rajtuk valamilyen műveletet (összeadás, logikai ÉS), majd az eredményt tárolja az egyik regiszterbe

o a két operandusnak az ALU-n történő átfuttatásából és az eredmény regiszterbe tárolásából álló folyamatot adatútciklusnak nevezük (minél gyorsabb az adatútciklus, annál gyorsabban dolgozik a gép)

Utasítás-végrehajtás

A CPU minden utasítást apró lépések sorozataként hajt végre. Ezek a lépések durván a következők:

1. A soron következő utasítás beolvasása a memóriából az utasításregiszterbe.
2. Az utasításszámláló beállítása a következő utasítás címére.
3. A beolvasott utasítás típusának meghatározása.
4. Ha az utasítás memóriabeli szót használ, a szó helyének megállapítása.
5. Ha szükséges, a szó beolvasása a CPU regiszterébe.
6. Az utasítás végrehajtása.

7. Vissza az 1. pontra, a következő utasítás végrehajtásának megkezdése.

Ezt a lépéssorozatot gyakran nevezik betöltő-dekódoló-végrehajtó ciklusnak, és központi szerepet tölt be minden számítógép működésében.

Utasítás- és processzorszintű párhuzamosság

Utasításszintű párhuzamosság

Az egyes utasításokban rejlö párhuzamosságot használjuk ki, hogy több utasítást tudunk másodpercenként kiadni.

Csővezeték

Évek óta ismert, hogy az utasítások végrehajtásának egyik legszűkebb keresztmetszete az utasítások olvasása a memóriából. A csővezeték erre próbál megoldást nyújtani, úgy hogy az utasítás végrehajtását több részre osztja, minden részt külön hardverelem kezel, amelyek minden egyszerre működhetnek. Így a csővezeték lehetővé teszi, hogy kompromisszumot kössünk késleltetés és áteresztőképesség között.

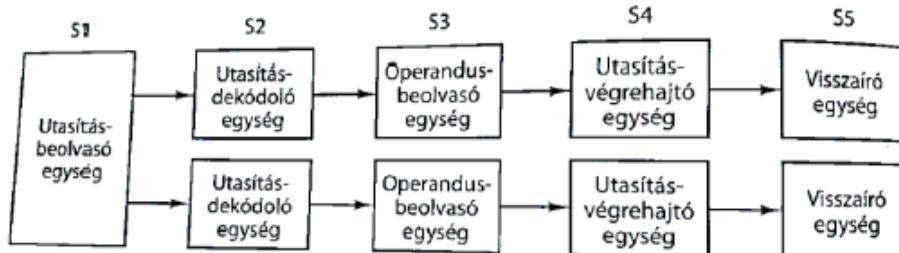
Működése

Szuperskaláris architektúrák

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

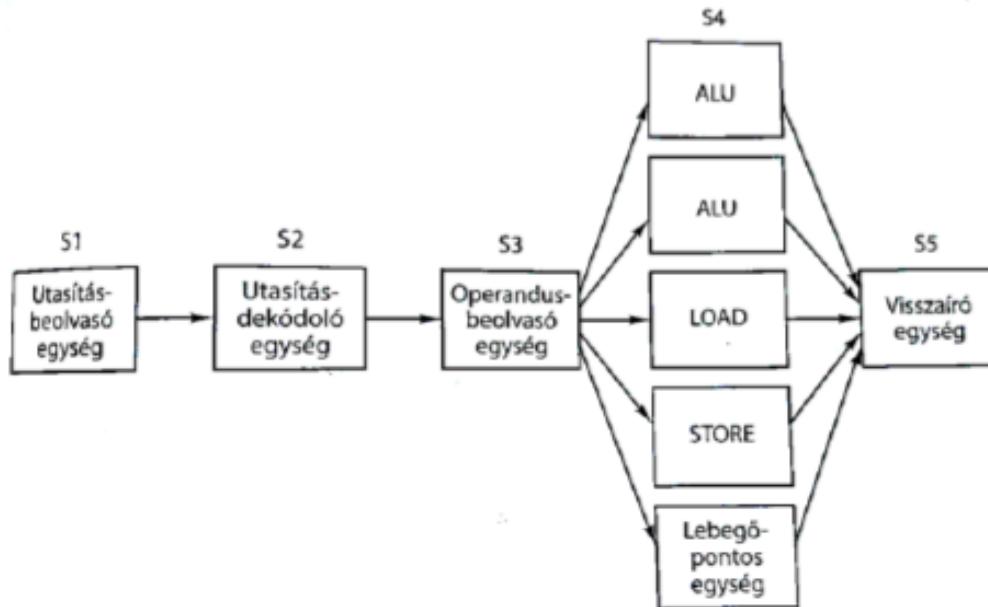
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Azon az elven működik, hogy ha egy csővezeték jó, akkor kettő még hatékonyabb. Van egy előolvasó egység, amely egyszerre két utasítást olvas be, majd azokat az egy-egy csővezetékre teszi. A csővezetékeknek saját ALU-juk van, így a két csővezeték párhuzamosan tud futni, ha nem használnak közös erőforrást, és egyik sem használja fel a másik eredményét. A feltételek betartását a fordítóprogramnak kell garantálnia.



Kettős csővezeték közös utasítás-beolvasó egységgel

Ez a felépítés azért nem ennyire egyszerű. A két csővezeték más-más utasítások végrehajtására alkalmas. A fő csővezeték az u pipeline tetszőleges Pentium-utasítást végre tud hajtani. A második csővezeték a v pipeline viszont csak egyszerű egész műveleteket tud elvégezni. Két utasítás kompatibilitását bonyolult szabályok határozzák meg. A csővezetékek számának növelésénél nagyon sok hardverelement kellene megduplázni, így az már nem lenne hatékony. Erre egy ötlet, hogy csak egy csővezetéket használnak, de több funkcionális egységgel. Ma a szuperskaláris jelzöt az olyan processzorokra használják, amelyek több utasítás végrehajtását kezdi el egyetlen órajel alatt.



Szuperskaláris processzor 5 funkcionális egységgel

Processzorszintű párhuzamosság

Több processzor dolgozik egyszerre ugyanazon a feladaton.

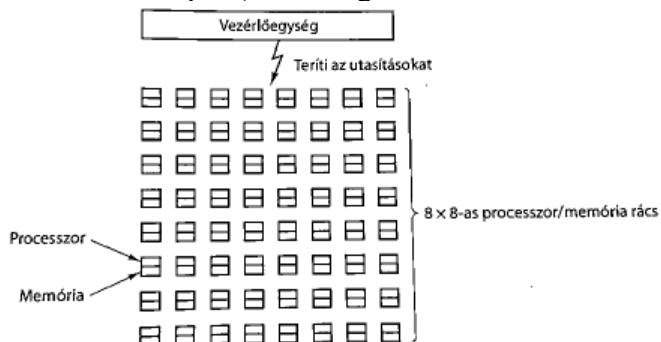
Tömbszámítógépek

Egy tömbprocesszor nagyszámú egyforma processzorból áll. ezek ugyanazt a műveletsorozatot végzik el különböző adathalmazokon. A világ első tömbprocesszora az ILLIAC IV számítógépe volt. Az eredeti terv szerint egy 4 negyedből álló gépet építettek volna, minden negyedben egy 8×8 -as négyzethálóban processzor/memória párokkal. Negyedenként egy vezérlőegység adta volna ki az utasításokat, melyeket a hozzá tartozó processzorok szinkronizálva hajtottak végre,

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

az adatokat minden egyik a saját memóriájából vette. Mivel a valódi költségek nagyon megnőttek, így csak egy negyedet építettek meg, de ez is elérte az 50 megaflop (millió lebegőpontos utasítás/másodperc) sebességet.



Egy ILLIAC IV típusú tömbszámítógép

A vektorprocesszor a tömbprocesszorhoz több szempontból hasonlít, viszont vektorprocesszor esetében minden összeadás egyetlen csővezeték elven működő összeadóegységen zajlik.

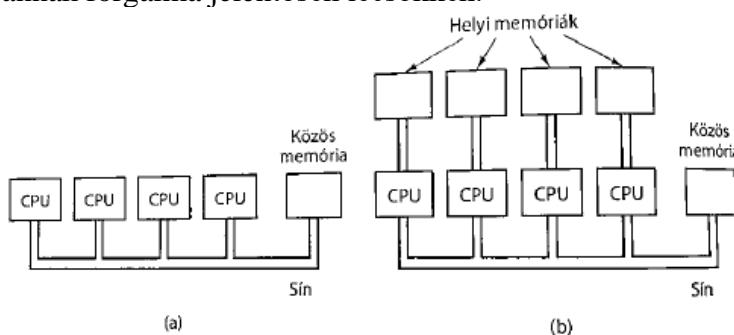
A tömb- és a vektorprocesszorok is adattömbökkel toldoznak. Mindkettő olyan egyedi utasításokat hajt végre, mint amilyen pl. két vektor elemeinek páronkénti összeadása, viszont eltérő módon valósítják meg. A tömbprocesszor úgy végzi el, hogy a vektor elemszámával megegyező számú összeadóegységet tartalmaz, a vektorprocesszorok pedig vektorregisztereket használnak hozzá.

Multiprocesszorok

A multiprocesszor az első olyan párhuzamos rendszerünk, amelyben több teljes CPU van, és ezek közös memóriát használnak. A CPU-knak együtt kell működniük, mivel minden egyikük írhatja és olvashatja a teljes memóriát.

Több implementációs séma lehet. Az egyszerűbb az, amikor a CPU-k és a memória egyetlen sínnel vannak összekapcsolva. Ezzel az lehet a gond, hogy ha sok gyors CPU próbálja elérni a közös memóriát, akkor az konfliktushoz vezethet.

Erre lehet megoldás az a kivitelezés, ahol minden CPU-nak van egy kis lokális memóriája, amihez csak ő fér hozzá. A lokális memóriák elérése nem a közös sínen keresztül történik, így annak forgalma jelentősen lecsökken.



(a) Egysínes multiprocesszor. (b) Multiprocesszor lokális memóriákkal

Multiszámítógépek

Azokat a rendszereket nevezzük multiszámítógépeknek, ahol az összekapcsolt számítógépeknek nincs közös memóriájuk. (A sok processzorból álló multiprocesszorok megépítése nehéz feladat a közös memória miatt.) A multiszámítógépek CPU-it időnként lazán kapcsolnak nevezik, megkülönböztetve őket a multiprocesszorokban található szorosan kapcsolt CPU-któl.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

A multiszámítógép processzorai üzenetküldéssel kommunikálnak egymással. Nagy rendszerekben nem kötik össze az összes számítógépet egymással, helyette 2 és 3 dimenziós rácsozás, fákát és gyűrűket használnak.

Mivel a multiprocesszorokat könnyebb programozni, de multiszámítógépeket könnyebb építeni, így sok kutató foglalkozik azzal, hogy valamelyen hibrid rendszerben a kettő előnyös tulajdonságát ötvözze.

- Korszerű számítógépek tervezési elvei. Példák RISC (UltraSPARC) és CISC (Pentium 4) architektúrákra, jellemzőik.

Korszerű számítógépek tervezési elvei.

Létezik tervezési elvek gyűjteménye, amit RISC tervezési elveknek is hívnak, és ezeket ma (jó kérdés hogy mikor volt ez a „ma”) minden általános célú processzor tervezője igyekszik legjobb képességei szerint követni.

A legfontosabbak:

- Minden utasítást közvetlenül a hardver hajtson végre
 - o ezek a gyakori utasítások nem bonthatók fel interpretált mikroutasításokra, így az utasítások gyorsabbak lesznek
- Maximalizálni kell az utasítások kiadásának ütemét
 - o egy ilyen trükk pl. hogy megpróbálják egy másodperc alatt a lehető legtöbb utasítás végrehajtását elkezdeni.
- Az utasítások könnyen dekódolhatók legyenek
 - o ide példa: az utasítások szabályosak, egyforma hosszúak és kevés mezőből állnak. Minél kevesebb utasításformátum van, annál jobb.
- Csak a betöltő és tároló utasítások hivatkozzanak a memóriára
 - o az operandusok mozgatása a regiszterek és a memória között külön utasításokkal történhet, és ez a művelet lassú, ezért legjobb őket más utasításokkal (amik csak adatot mozgatnak) átfedve végrehajtani → csak a LOAD és STORE utasításoknak szabad a memóriaára hivatkozni.
- Sok regiszter kell
 - o a memóriaművelet lassú, így jó ha egy beolvastott szó addig a regiszterben tud maradni, amíg szükség van rá, ezért van szükség a sok regiszterre

Példák RISC (UltraSPARC) és CISC (Pentium 4) architektúrákra, jellemzőik.

RISC (UltraSPARC)7

-Reduced Instruction Set Computer

-Kevesebb utasításból állt, viszont azok gyorsak voltak.

Az 1970-es években nagyon népszerű volt a UNIX, viszont ez személyi számítógépekben nem futott, ezért egy egyetemista elhatározta, hogy épít magának egy saját UNIX rendszert (SUN-1). Ez mások figyelmét is felkeltette, és három másik emberrel elhatároztak, hogy alapítanak egy céget (Sun Microsystems) Sun munkaállomások építésére és forgalmazására. Pár év alatt óriási bevételle lett a cégeknek, és elhatározták, hogy saját processzort terveznek a RISC II mintájára. Ez volt a SPARC (nyílt architektúra), és rövid időn belül minden Sun-géphez SPARC CPU került

A SPARC fejlődésében 1995-ben következett be komoly változás, amikor kifejlesztették az architektúra 9-es verzióját, ami egy valódi 64 bites architektúra, 64 címbittel és 64 bites regiszterkészlettel. Az UltraSPARC volt az első Sun-munkaállomás, amely a V9 architektúrára épült.

Az UltraSPARC nyitott a multimédia felé, így kezdetektől fogva képek, hang és videó kezelésére is szánták a szövegszerkesztés és táblázatkezelés mellett. A multimédia tartalom kezeléséhez új utasítások is megjelentek.

Az UltraSPARC I –et a II., III., és IV követte, amelyek elsősorban az órajelbességükben tértek el, de minden egyes fejlesztésbe néhány új tulajdonság is bekerült.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

CISC (Pentium4)8

- Complex Instruction Set Computer

- Az utasításkészlet több összetett utasításból állt, de végrehajtásukhoz több idő kellett.

Az architektúra az Intel Corporation nevéhez fűződik. A 60-as évek végén volt egy megrendelésük, amivel valami újat fedeztek fel, de még ők se tudták pontosan, hogy mit is. (Ez volt a 4004-es lapka, amit eredetileg ugye egy speciális céllal terveztek, de rájöttek, hogy általános célokra is alkalmazható).

A 4004-es lapka mintájára kifejlesztették a 8008-as 8 bites lapkát, aminek óriási sikere lett, ezért nekifogtak ennek a továbbfejlesztésére (8080, 8086, 8088). Sem a 8080-as, sem a 8086-os nem tudott 1 MB-nál nagyobb memóriát megcímezni, ezért még tovább dolgoztak rajta és így jött a 80286, 80386, 80486-os lapka. Idő közben a cég rájött, hogy a számozás nem épp a legjobb elnevezés, így a 5-ös verziónak nevet adtak: Pentium. A Pentium a 80486-os lapkára épült és itt került be a gyártásba a speciális MMX (multimédiás kiegészítések) utasítások, melyeknek célja hang és videoadatok feldolgozásához szükséges számítások felgyorsítása, és így nem kellett multimédiás társprocesszorokat alkalmazni.

A Pentium név annyira ismertté vált, hogy a további architektúrák is megkapták a nevet.

A Pentium4 belső architektúrája alapvetően más az elődjeihez képest. A 3,06 GHz-es változatban vezetettek egy új tulajdonságot, a hyperthreadingget, ami lehetővé tette a programot számára, hogy a munkát két vezérlési szálra bontsák, amit a Pentium 4 párhuzamosan futtat. Kiegészítésként további utasítások kerültek a processzorba, ezzel tovább gyorsítva a hang- és videoadatok feldolgozását.

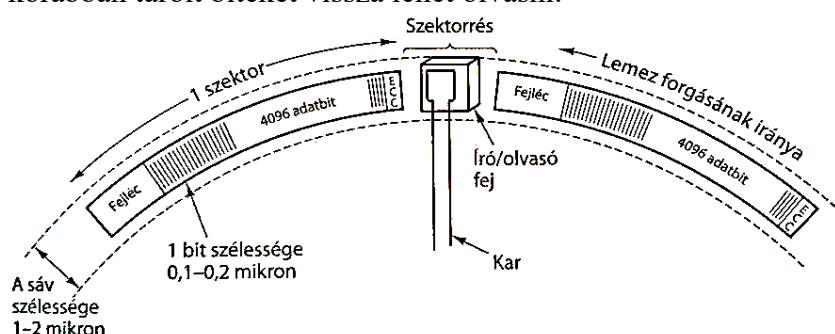
2. Tétel

- Számítógép perifériák: Mágneses és optikai adattárolás alapelvei, működésük (merevlemez, Audio CD, CD-ROM, CD-R, CD-RW, DVD, Bluray)

Mágneses és optikai adattárolás alapelvei, működésük

Mágneslemezek

Egy mágneslemez egy vagy több mágnesezhető bevonattal ellátott alumíniumkorongból áll. Egy indukciós tekercset tartalmazó fej lebeg a lemez felszíne felett egy vékony légpárnán. Ha pozitív vagy negatív áram folyik az indukciós tekercsben, a fej alatt a lemez magnetizálódik, és az áram polaritásától függően a mágneses részecskék balra vagy jobbra állnak be. Amikor a fej egy mágnesezett terület felett halad át, akkor pozitív vagy negatív áram indukálódik benne, így a korábban tárolt biteket vissza lehet olvasni.



Egy sáv részlete. Két szektor látható a képen.

Egy teljes körülfordulás alatt felírt bitsorozat a sáv. minden sáv rögzített méretű, tipikusan 512 adatbájtot tartalmazó szektorokra van osztva, melyeket egy fejléc előz meg, ami a fej szinkronizálását teszi lehetővé írás és olvasás előtt. Az adatok után hibajavító kód található, ez vagy a Hannimg-kód, vagy egyre gyakrabban a többszörös hibákat is javítani képes Reed-Solomon-kód. Az egymást követő szektorok között keskeny szektorrész van.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

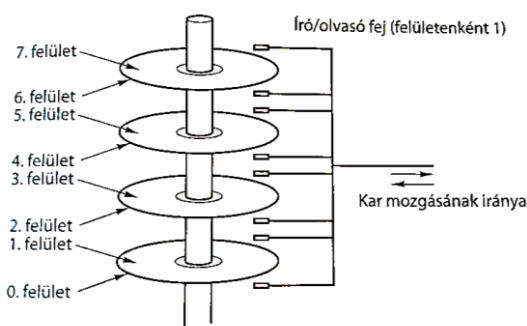
Minden lemeznek vannak mozgatható karjai, melyet a forgástengelytől sugárirányban ki-be tudnak mozogni. minden sugárirányú pozíciót egy-egy sáv szélessége attól függ, hogy milyen széles a fej, illetve hogy milyen pontosan lehet sugárirányban pozicionálni.

A bitek lineáris sűrűsége egy sáv kerülete mentén különbözik a sugárirányú sűrűségtől. A nagyobb írássűrűség érdekében a gyártók olyan eljárásokat fejlesztenek ki, amelyeknél a biteket nem a diszk kerületén hosszirányban, hanem függőlegesen a vas-oxid belseje felé rögzítik- Ezt merőleges rögzítésnek nevezik.

Merevlemez

A legtöbb lemezt gyárilag lémementesen lezárják, hogy por ne kerülhessen bele. Az ilyen lemezeket winchesternek hívják.

Lemezegység négy koronggal:

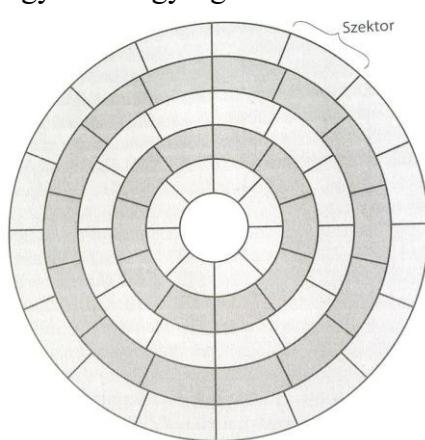


A legtöbb lemezegység egymás felett elhelyezett korongból áll. minden felülethez tartozik egy fej és egy mozgatókar. A karok rögzítve vannak egymáshoz, így a fejek minden ugyanarra a sugárirányú pozícióra állnak be. Egy adott sugárirányú pozícióhoz tartozó sávok összességét cilindernek nevezzük.

A lemezegység teljesítménye sok tényezőtől függ. Egy szektor beolvasásához vagy kiírásához elősör a fejet a megfelelő sugárirányú pozícióba kell állítani (seek). A fej kívánt sugárirányú pozícióba való beállása után van egy kis szünet, a forgási késleltetés, amíg a keresett szektor a fej alá fordul. Ebből következik, hogy a keresési idő és a forgási késleltetés teszi ki az adatátviteli idő nagy részét.

A következő probléma, hogy a külső sávok hosszabbak, mint a belsők. Mivel a lemezek a fejek pozíciójától függetlenül állandó szögsebességgel forognak, ez problémát vet fel. Erre megoldás, hogy a cilindereket zónákba osztják, és a külső zónákban több szektort tesznek egy sávba, ami növeli a kapacitást. Minden szektor mérete egyforma.

Egy lemezegység öt zónával:



Minden zónában sok sáv található

Minden lemezhez tartozik egy lemezvezérlő. egy lapka, amely vezéri a meghajtót. A vezérlő feladatai közé tartozik a szoftverből érkező parancsok fogadása, a kar mozgatása, hibák felismerése és javítása, valamint a 8 bites memóriabájtok oda- és visszaalakítása bitek sorozatává.

Optikai adattárolás

Audio CD

1980, Sony – Philips

A CD pontos technikai részleteit nemzetközi szabványban rögzítették, Red Book. minden CD 12 cm átmérőjű volt, 1.2 mm vastag, a közepén egy 15 mm-es lyukkal. Élettartamukat 100 évre becsülük. Ez volt az első sikeres tömeggyártású digitális adathordozó.

Egy CD úgy készül, hogy nagy energiájú infravörös lézerrel 0.8 mikron átmérőjű lyukakat égetnek egy bevonattal ellátott, üveg mesterlemezbe. Erről a lemezről negatív öntőforma készül, amelyen kiugrások vannak az eredeti lyukak helyén. Az öntőformába olvadt

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

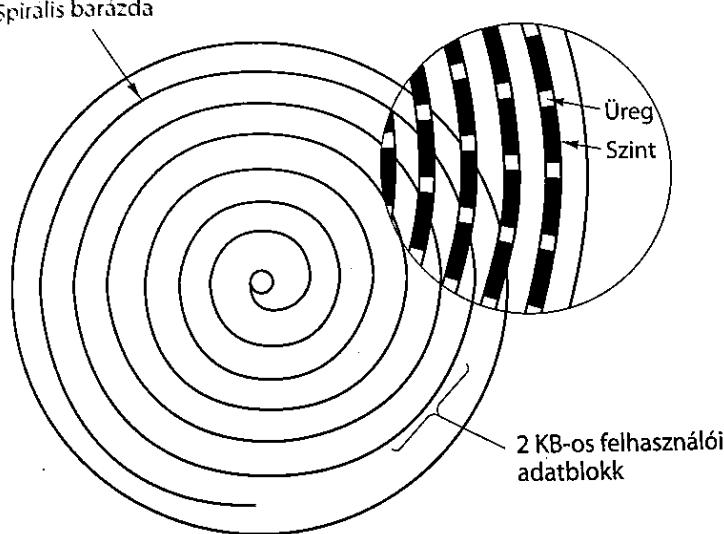
polikarbonát gyantát töltenek, így egy olyan CD-t kapnak, amelyen a lyukak mintázata azonos a mesterlemezével. Ezután egy nagyon vékony fényvisszaverő alumíniumréteg, majd egy lakk védőréteg és egy címke kerül a lemezre. A polikarbonát rétegen elhelyezkedő mélyedéseket üregnek (pit), az üregek közötti érintetlen területeket pedig szintnek (land) nevezzük.

Visszajátszáskor ez kis energiájú lézerdióda infravörös fénnel világítja meg az üregeket és a szinteket, ahogy elhaladnak alatta. A lézer a polikarbonátoldalon van, ezért az öregek a lézer felé kidudorodnak az amúgy is sima felületből. Mivel az üregek magassága negyede a lézerfény hullámhosszának, az üregekről visszaverődő fény fél hullámhossznyi fáziseltolódásban van a környező területről visszaverődő fényhez képest. Ennek eredménye, hogy a két rész interferenciája gyengíti egymást, emiatt a lejátszó fényérzékelőjébe kevesebb kény jut annál, mint amikor a fény a szintről verődik vissza. Így különbözteti meg a lejátszó az üreget a szinttől.

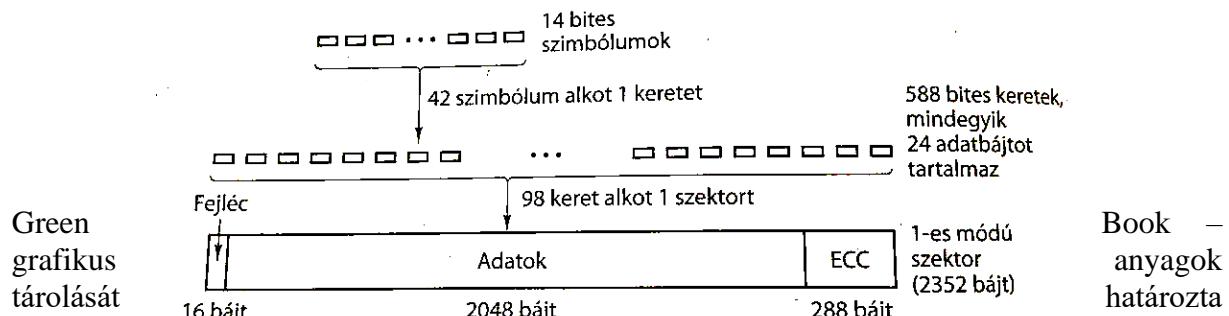
Az üreg/szint vagy a szint/üreg átmenet jelenti az 1 értéket, az átmenet hiánya pedig a 0-t.

Az üregek és a szintek egyetlen folytonos spirálban kerülnek felírásra, amely a lyuk közelében kezdődik és a lemez széle felé 35 mm széles sávot foglal el.

Adattárolás a kompaktlemezen vagy CD-ROM-on:



Növelték a hibajavító képességet. A Red Bookhoz képest azt adja még hozzá, hogy 98 keretet egy CD-ROM-szektorba csoportosít. minden CD-ROM-szektor egy 16 bájtos bevezetővel indul. A következő 3 bájt a szektor számát tartalmazza. A bevezető utolsó vájtja a mód. Az egyik az ábrán is látható mód, amelyik 16 bájtos bevezetővel, 2048 adatbájttal és 288 bájt hibajavító kóddal rendelkezik. A másik mód az adat és az ECC mezőket egyetlen 2336 bájtos adatmezőbe foglalja azoknak az alkalmazásoknak a kedvéért, amelyek nem igénylik a hibajavítást (pl zenék). Az adatok logikai elhelyezkedése CD-ROM-on:



ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

meg, valamint lehetővé tette egy szektoron belül audio-, video- és egyéb adatok egyidejű elhelyezését, ami a multimédiás CD-ROM-okhoz elengedhetetlen tulajdonság.

Ahhoz, hogy különböző számítógépekben lehessen használni a CD-ROM-okat, meg kellett egyezni a CD-ROM-fájlrendszer formátumában.

-Példa

o max 8 karakteres fájlnevek

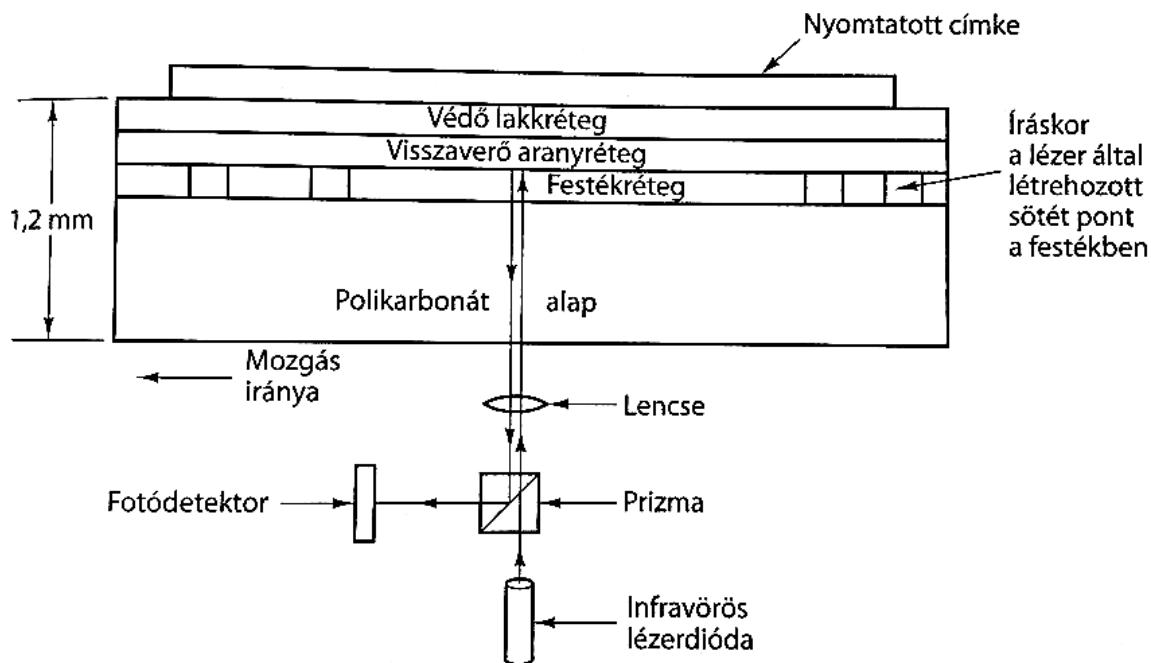
o max 3 karakter kiterjesztés

o alkönyvtárak max 8 mélysegig lehetnek egymásba ágyazva

CD-R – Orange Book

Fizikailag az írható CD-khez ugyanolyan üres 120 mm-es polikarbonát lemezeket használnak, amelyek hasonlítanak a CD-ROM-okra, kivéve, hogy egy 0.6 µm szélességű barázda van rajtuk az író lézernyaláb irányítására, illetve a felső oldaluk aranyszínű, nem ezüst. A polikarbonát és az aranyréteg között egy festékréteg van.

Kezdeti állapotban a festékréteg átlátszó, a lézerfényt átengedi, és visszaverődik az aranybevonatról. Amikor a lézersugár egy festékfoltot talál el, az felmelegszik és megváltozik a molekuláris szerkezete, aminek hatására egy sötét folt alakul ki. Visszaolvasáskor a fényérzékelő egysége különbséget tud tenni a lézerrel elroncsolt festék sötét foltjai és az épen hagyott átlátszó területek között. A színkülönbségeket úgy kezelik, mintha üregekés szintek közötti különbségek lennének. Egy CD-R keresztmetszete a lézerrel:



CD-RW

Maga lemez szinte ugyanolyan, mint elődei, azzal a különbséggel, hogy az adattároló réteg egy ötvözetet tartalmaz. Ennek az ötvözetnek két stabil állapota van: kristályos és amorf, különböző fényvisszaverő tulajdonságokkal.

A CD-RW-meghajtók három eltérő energiájú lézert alkalmaznak.

- A legmagasabb energián az ötvözet megolvad, és a nagy visszaverő képességű kristályos állapotból a kis visszaverő képességű amorf állapotba kerül. ezzel egy üreget reprezentálva.

- Közepes energián az ötvözet megolvad, és visszatérít természetes kristályos állapotába, ezzel újból szintállapotba kerül.

- Kisenergián az anyag állapotát lehet érzékelni, de nem történik átalakulás.

DVD

Amiben más, mint a CD:

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

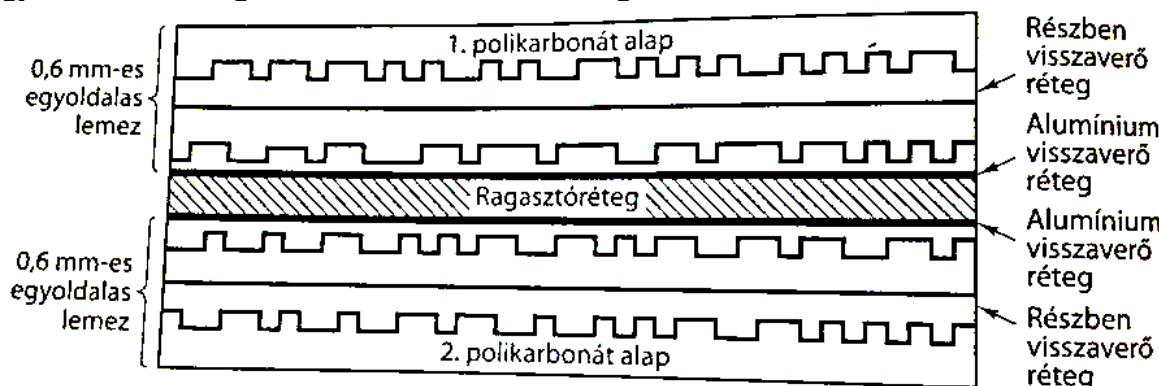
- kisebb üregek
- szorosabb spirál (kisebb rések vannak a sávok között)
- vörös lézer

Ezekkel a javításokkal a DVD kapacitása a CD-éhez képest hétszeresére nőtt, 4.7 GB. Egy $1 \times$ DVD-meghajtónak 1.4 MB/s az adatátviteli sebessége.

Kérdéses volt, hogy a tárolókapacitása elég lesz-e. Különböző cégek nem tudtak megállapodni növeléshez alkalmazott technológiáról, így több formátumot is definiáltak:

- Egyoldalas, egyrétegű (4.7 GB)
- Egyoldalas, kétrétegű (8.5 GB)
- Kétoldalas, egyrétegű (9.4 GB)
- Kétoldalas, kétrétegű (17 GB)

A kétréteges technológia úgy működik, hogy legalul egy visszaverő réteget helyeznek el, fölötté pedig egy részben visszaverő réteget. Attól függően, hogy a lézert hova fókusztálják, az egyik vagy másik rétegről verődik vissza. Az alsó réteg egy kicsit nagyobb üregeket és szinteket igényel a biztonságos visszaolvasáshoz, ezért a kapacitása egy kicsit kisebb, mint a felső rétegé. A kétoldalas lemezeket úgy készítik, hogy két 0.6 mm vastagságú egyoldalas lemez háttal egymásnak összeragasztanak. Kétoldalas, kétrétegű DVD lemez:



Blu-Ray

Onnan kapta a nevét, hogy kék lézert használ piros helyett. A kék fénynek rövidebb a hullámhossza, mint a pirosnak, ezért pontosabban fókuszálható, és így kisebb mélyedéseket tesz lehetővé. Az egyoldalas Blu-Ray lemez 25 GB, a kétoldalas 50 GB adatot tárol. Az átviteli sebessége 4.5 MB.

- SCSI, RAID

SCSI (Small Computer System Interface)

Ezek is cilinderekre, sávokra és szektorokra vannak osztva, de más az interfészük és sokkal nagyobb az adatátviteli sebességük, mint az IDE- lemezeknek. A nagyobb adatátviteli sebességük miatt a Sun és a HP szabványfelszereléséhez tartoznak.

A SCSI több egy merevlemez-interfésznel. Ez egy sín, amelyre egy SCSI-vezérlő s legfeljebb hét eszköz csatlakoztatható, aminek van egy egyértelmű azonosítója 0 és 7 között. Az eszközök sorban vannak kapcsolva (az egyik kimenete a következő bemenetével). Az utolsó eszközt le kell zárni, nehogy zavar keletkezzen az adatforgalomban.

A SCSI-vezérlők és -perifériák kezdeményező és fogadó üzemmódban működhetnek. Általában a kezdeményezőként működő vezérlő adja ki a parancsokat a fogadóként viselkedő lemezegységeknek és egyéb perifériáknak.

RAID (Redundant Array of Inexpensive/Independent Disks)

Az alapötlet az, hogy a számítógép mellé telepítünk egy dobozt tele lemezegységekkel, a lemezvezérlőt kicseréljük egy RAID-vezérlőre, átmásoljuk az adatokat a RAID-re, aztán folytatjuk a munkát. Vagyis a RAID-nek az operációs rendszer felé úgy kell viselkednie, mint

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

egy SLED-nek (Single Large Expensive Disk), amelynek azonban nagyobba a teljesítménye és a megbízhatósága

Mivel a SCSI-lemezeknek jó a teljesítménye, alacsony az ára és egyetlen vezérlő akár 7 meghajtót is képes kezelni, elég természetes, hogy a legtöbb RAID egy RAID SCSI-vezérlőből és sok SCSI-lemezről áll, ami az operációs lemez felé egyetlen lemezként látszik. Attól függetlenül, hogy egy lemezként látszik, az adatok szét vannak osztva a meghajtók között, lehetővé éve a párhuzamos működést. Ennek több lehetséges módját is definiálták RAID 0-tól RAID 8-ig.

- Nyomtatók, egér, billentyűzet

Nyomtatók

Mátrixnyomtatók

7-24 elektromágnesesen aktivizálható tűt tartalmazó nyomtatófej halad el minden nyomtatandó sorban. Az egyszerűbb fajtákból 7 tű van, és pl. soronként 80 karakter tud 5×7 -es pontmátrixokból előállítani.

A nyomtatási minőség kétféleképpen javítható:

-több tű

-egymást átfedő körök alkalmazása

A mátrixnyomtatók olcsók és nagyon megbízhatók, de lassúak, hangosak és gyengék a grafikus képességeik.

Alkalmazási területek:

- Nagy előre nyomtatott formanyomtatványoknál

- Pénztárgépben, ATM-ben

- Leporellókra való nyomtatáshoz

Tintasugaras nyomtató

Othoni nyomtatáshoz ez a típus a legkedveltebb.

Működése: A mozgatható, tintapatront tartalmazó nyomtatófej vízszintesen halad végig a papír előtt, mialatt tintát permetez a fűvökából.

Két típusa van a piezoelektromos és a hővezérlésű.

Piezoelektromos esetén egy speciális kristály van a tintapatron mellett, és ezzel a kristállyal szabályozza a festékcsap méretét.

A hővezérlésűeket gyakran festékbuborékosnak is. Ebben az esetben egy kis ellenállás van minden fűvökában. Erre, amikor feszültséget kapcsolnak, akkor a vele érintkező festéket felhevíti, ami elpárolog és gázbuborékot képez, és nyomás keletkezik a fűvökában, ami hatására a tinta a papírra kerül.

Lézernyomtató

A lézernyomtatók közel ugyanazt a technológiát használják, mint a fénymásolók. A nyomtató szíve egy fényérzékeny anyaggal bevont forgó precíziós henger. Ezt egy-egy lap nyomtatása előtt kb 1000 V-ra feltöltik. Ezt követően a lézer fénye pástázza végig a hengert hosszában. A fényt modulálják, hogy világos és sötét pontokat kapjanak. Azok a pontok, ahol fény éri a hengert, elvesztik elektromos töltésüket.

Amikor egy kész sor elér a tonerkazettához, amely elektrosztatikusan érzékeny fekete port tartalmaz, akkor a por hozzápad a feltöltött pontokhoz, így láthatóvá válik a sor. Amikor a henger tovább fordul, akkor hozzáér a papírhoz és átkerül rá a festék. Ezután a papír felmelegített görgők között halad el, ahol végelesen hozzápad a festék a papírhoz.

Színes nyomtatók

A színes nyomtatók a CMYK színmodell alapján dolgoznak. Alapszínei a Cián, Magenta, Sárga és a Fekete. Az RGB additív színmodellel ellentétben, ez egy szubtraktív modell, ami a színelnyelésen alapszik.

Mivel a nyomtató és a monitor nem egy színmodellt használ, így a megjelenített és kinyomtatott kép nem egyezik meg, köztük különböző konverziót kell végrehajtani.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Fajtái:

-festékalapú tinták

- o folyékony hordozóanyagban oldott színes festék
- o élénk színt adnak
- o könnyen folynak
- o napsütés hatására fakulnak

- pigmentalapú tinták

- o szilárd pigmentrészecskéket tartalmaznak folyékony hordozóanyagban
- o nem fakulnak
- o nincs olyan élénk színük, mint a festékalapúaknak

- szilárd tintás nyomtatók

- o négy speciális, szilárd tintatömböt kell elhelyezni a nyomtatóban
- o ezeket felolvasztják és tartályokban tárolnak
- o akár 10 percet is kell várni, mire felmelegszik

- színes lézernyomtató

- o pont úgy működik, mint a monokróm csak több a tóner
- o nagy memória kell hozzá
- o drága

- viasznyomtató

- o egy széles, négyszínű viasz tartalmazó szalag található benne
- o több fűtőelem olvasztja a viaszat, ahogy mozog alatta a papír

- festékszublimációs nyomtató

- o több ezer fűtőelemet tartalmazó nyomtatófej felett halad el a festék
- o a feltékek ekkor azonnal elpárolognak, és a speciális papírra tapatnak.
- o minél nagyobb a hő, annál több festék kerül a papírra és annál élénkebb lesz a szín

Egér

- Mechanikai

- o két gumikereke van, amik mozgatáskor ellenállásokat vezérelnek
- o az ellenállás változás mértékéből számítható a pozíció

- Optikai

- o LED és egy fénydetektor van az aljában

- Optomechanikus

- o golyó és LED is van benne

Leggyakoribb esetben az egér egy 3 bájtból álló üzenetet küld a számítógépnek, amikor megtesz egy bizonyos távolságot. Az első bajt egy előjeles egész szám, ami megadja, hogy mennyit mozdult x irányban, a második ugyanez y irányban, a harmadik pedig a gombok helyzetét jelenti.

Billentyűzet

Számos fajtája létezik

- membrános

- ollós

- mikrokapszolós

Egy billentyű lenyomásakor megszakítás generálódik, és a billentűzet-megszakításkezelő elindul. A megszakításkezelő kiolvassa a billentyűzet vezérlő regisztereiből a leütött billentyű kódját. Amikor egy billentyűt felengedünk, akkor egy második megszakítás generálódik.

- Telekommunikációs berendezések (modem, ADSL, Kábel TV-s internet)

Modem

A telefonvonalon nem alkalmas számítógépes jelek továbbítására. Egy 1000 és 2000 Hz közötti, vivőhullámnak nevezett tiszta szinuszos hullám azonban aránylag kis torzulással átvihető, így a legtöbb telekommunikációs rendszer ezt használja fel a működéséhez.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

Ahhoz, hogy a 0-k és 1-esek sorozatát át tudjuk vinni, változtatni kell az amplitúdót, frekvenciát, vagy a fázist. Ezt a folyamatot modulációnak nevezzük.

Amplitúdómoduláció: két feszültségszintet használ, egyiket a 0-khoz, a másikat az 1-esekhez.
Frekvenciamoduláció: a feszültségszint állandó, de a vivőhullám frekvenciája eltérő 1-esek és 0-k esetén.
Fázismoduláció: az amplitúdó és a frekvencia nem változik, de a vivőhullám fázis 180 fokkal eltolódik minden 0-1 vagy 1-0 váltásnál. Ennél lehet kifinomultabb is a moduláció, és akkor több bitet lehet intervallumonként átvinni. (Pl.: dabit fáziskódolás – 2 bit/intervallum)
Általában 8 bites egységeket akarunk átküldeni. Ennek az elejére egy ún. startbitet, végére pedig stopbitet fűzünk. Így összesen 10 bitet viszünk át.

A küldő modem az egy karakterhez tartozó biteket egyenlő időközönként küldi. A fogadó oldalon egy másik modem konvertálja a modulált jelet bináris számmá. Mivel a bitek szabályos időközönként érkeznek, ha a fogadó meghatározta a karakter elejét, már csak az órajelet kell figyelnie, hogy mikor vegyen mintát a következő bitek meghatározásához.

Full-duplex: egyszerre tudnak minden irányban kommunikálni

Half-duplex: olyan modem, vagy átviteli vonal, amely egyszerre csak egy irányban tud adatokat átvinni.

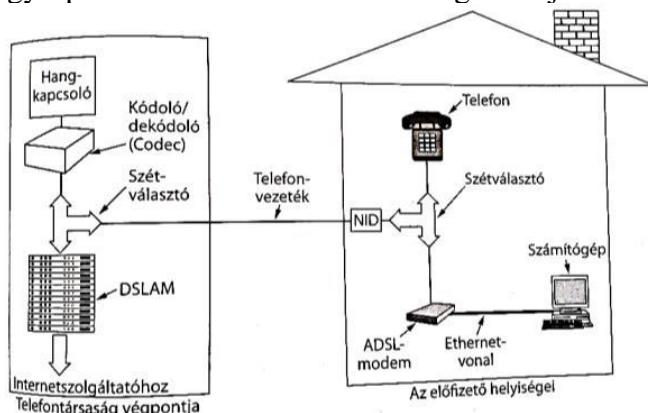
Szimplex: csak egy irányban működő vonal.

ADSL

A modenem lassúságának az az oka, hogy a telefont nem adatok továbbítására találták ki. Az előfizetőktől a telefontársaság központjába futó vezetéket egy szűrővel korlátozzák, ami behatárolja az adattovábbítás sebességét.

Az ADSL lényegéten azt valósítja meg, hogy a szűrőt eltávolítják, és az elérhető spektrumot 256 független csatornára osztja fel. A 0-s csatornát az egyszerű régi telefonszolgáltatás céljára használják. 1-5-ig pedig nem használják a csatornákat, hogy megelőzzék a telefonhang és az adatok interferenciáját. A fennmaradó 250 csatornából egyet a felmenő, egyet pedig a feljövő adatfolyam vezérlésére használnak. A többi csatorna a felhasználók adatait továbbítja. Az ADSL olyan, mintha 250 modemünk lenne. Mivel több adatot töltünk le, mint fel, így gyakori megosztás, hogy 32 csatorna van feltöltésre, a többi pedig letöltésre.

Egy tipikus ADSL-berendezés konfigurációja:



Az előfizető telephelyén egy hálózati interfészkelni kell felszerelni, amihez közel található a szétválasztó, egy analóg szűrő, amely lemosztja az adathálózatról a 0-4000 Hz-es sávot a POTS számára. A POTS a jelet egy meglévő telefonra vagy telefaxra, az adatjeleket pedig egy ADSL-modemre irányítják. Az ADSL-modemet és a számítógépet Ethernet hálózati kártya segítségével köti össze.

A telefontársaság végpontján egy megfelelő szétválasztót kell felszerelni. A jel hang részét itt is ki kell szűrni, és a normál telefonközpontba küldeni. A 26 kHz feletti jeleket egy újfajta készülékbe, a DSLAM-ba irányítják. (Ez ugyanolyan digitális jelfeldolgozó egység, mint az ADSL-modem.) Amint a digitális jelekből a bitsorozatokat előállították, csomagokat alakítanak ki belőlük, és az internetszolgáltatóhoz küldik tovább.

ZÁRÓVIZSGA TÉTELEK - 2020. JÚNIUS

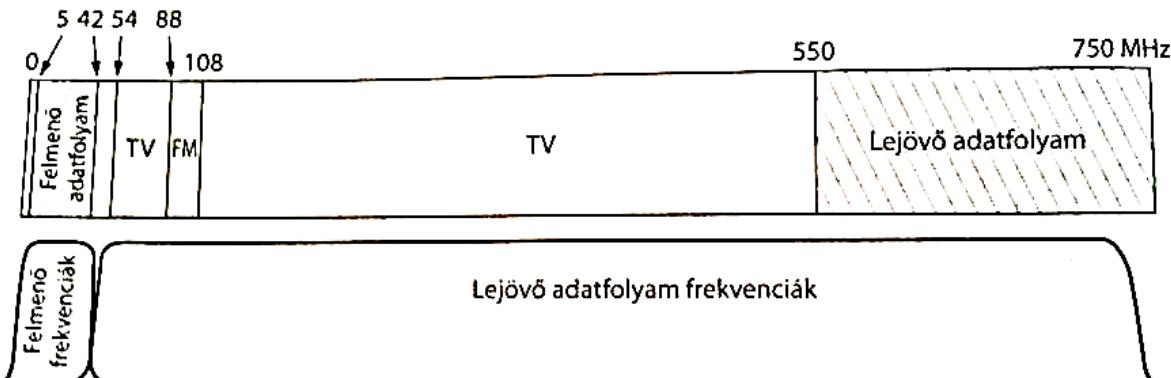
PROGRAMTERVEZŐ INFORMATIKUS BSC SZAK

KábelTV-s internet

A kábelszolgáltatónak vannak fő telephelyei és ezekhez több fejállomás tartozik, amiket nagy sávszélességű kábelek vagy optikai kábelek kapcsolnak össze. Az előfizetők a fejállomásokhoz kapcsolódnak.

A modern kábelek jóval 550 MHz felett is működnek. A feltöltésre szolgáló csatornák az 5-42 MHz sávban vannak, a letöltésre szolgáló csatornák forgalma a frekvenciatartomány felső részére esik.

Egy tipikus kábeltévés internetszolgáltatás frekvenciakiosztási diagramja:



Az internet-hozzáféréshez előfizetői oldalon egy kábelmodemre van szükség, amelyben két interfész van: az egyik a számítógéphez, a másik a kábeltévé-hálózathoz.

Amikor bekapcsoljuk a modemet, akkor az végigellenőrzi a lejövő csatornákat, és olyan speciális csomagokat keres, amelyeket a fejállomás periodikusan küld, és a rendszerparamétereket tartalmazzák az újonnan bekapcsolódó modemek számára. Amint a modem megtalálta ezt a csomagot, jelzi a jelenlété az egyik felmenő csatornán. A fejállomás azzal válaszol, hogy kijelöli a modem felmenő és lejövő csatornáit.

Ezután a modem meghatározza a fejállomástól mért távolságát egy speciális csomag küldésével (távolságbehatárolás). Az adatfogalmat rövid idejű szeletekre, ún., minislotokra osztják. minden felfelé haladó csomagnak egy vagy több minislotba kell beleférnie.

Az inicializálási fázisban a fejállomás minden egyes modem számára biztosít egy minislotot, amellyel felmenő sávszélességet kérhetnek. Itt versenyhelyzet is kialakulhat, mert több modemet is rendelhetnek egy minislothoz.

A lejövő csatornákat ettől eltérően irányítják. Itt nincs versenyhelyzet, mert csak a fejállomás küld adatokat. Rögzített méretű (204 bájt) csomagokat használnak, amelyek egy része Reed-Solomon-féle hibajavító kód, és más járulékos dolgok, így a felhasználónak 184 bájt hasznos adatmennyiséget jut. (A számokat az MPEG-2 szabványú digitális televízió szabvánnyal való kompatibilitás miatt választották.)

Visszatérve a modem inicializáláshoz, amint a modem behatárolta a távolságát, megkapta a felmenő és lejövő csatornáit, valamint a minislotkiosztását, elkezdheti a csomagok küldését. A fejállomások a csomagokat a fő telephelyre, majd az internetszolgáltató felé továbbítják. Az internetszolgáltató felé menő első csomag a hálózaticím-kérés, amelyet dinamikusan osztanak ki. Az inicializálási folyamat egyik része a titkosítási kulcsok beállítása.

Végül a modemnek be kell jelentkeznie, és meg kell adnia saját egyedi azonosítóját a titkos csatornán. Ekkor az inicializálás véget ér. A felhasználó bejelentkezhet az internetszolgáltatóhoz, és elkezdhet dolgozni.