Algoritmusok és gráfok

Csima Judit

http://www.cs.bme.hu/algraf/

2018. 09. 06. – előadás

Témák

- könnyű problémák
- létező algoritmusok
- általános technikák, algoritmusok
- adat szerkezetek

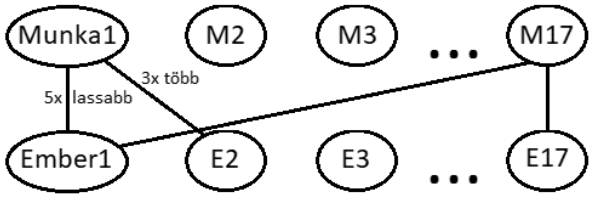
Algoritmus

- véges eljárás
- determinisztikus (x-szeres elindítással x-szer ugyanaz az eredmény kapható)
- jól definiált utasítássorozat
- mit kell megadni, leírni?
  - leírás:
    - szöveges, pontos
    - pszeudo-kód
  - helyesség ≠ jóság bizonyítása
  - futási idő = lépésszám (LSZ)
    - algoritmus hány elemi lépést tesz n méretű input esetén
    - feladattól függ
    - n-nek függvénye lesz
    - általában felső becslés (legrosszabb eset)
    - definíció: van olyan c konstans (n-től független), hogy az LSZ legfeljebb $c\ f(n)$ ha n elég nagy, (ha $n \geq n_0$) akkor $LSZ\ O(f(n)) \rightarrow$ ordó $f(n)$
- input: adat → algoritmus → elvárás: output

Példák

Waze

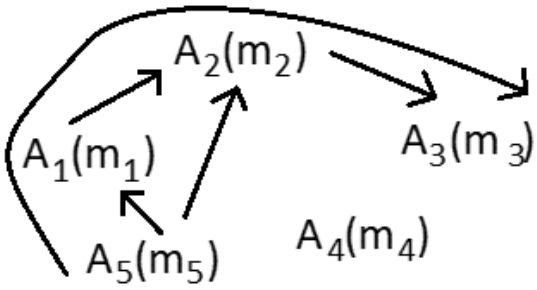
- input: honnan? hová? forgalmi adatok
- cél: leggyorsabb út
- brute-force: lassú
- feladatkiosztás:



  cél: melyik munkákat és ki végzi el leghamarabb?
- 17 ember, 17 munka → 17! lehetőség
- szuperszámítógép információsebessége: $3 \cdot 10^8\ \frac{m}{s^2}$
- node: magyar módszer – gyors

Akrobaták

- felül: alacsonyabb, könnyebb
- lent: magasabb, nehezebb
- input: súlyok, magasságok
- cél-output: legmagasabb oszlop
- brute-force: minden részhalmaz 2^n
- megoldás: leghosszabb út, Ø irányított kör
  → van rá algoritmus



Lépésszám (LSZ)

	piros algoritmus	sárga algoritmus	zöld algoritmus	fekete algoritmus
S input	3 S	1000 S	$\frac{1}{1000}$ S	2^S
17 S input	$17 \cdot 3$ S	$17 \cdot 1000$ S	$\frac{1}{1000}(17S)^2$	2^{17S}
mekkora változás?	x17	x17	$x17^2$	$x2^{16S}$
hatékonyság	jó	jó	rossz	nagyon rossz

- tanulság: LSZ-ban a konstans nem érdekes, csak az n szereplése $(n, n^2, n^3, 2^n)$

Minimumkeresés (Minimum search)

- kérdés: legkisebb szám a tárolók közül
- szövegesen:
  - minimumban tárolom az eddigi legkisebbet, eleinte: $T[0]$
  - végig megyek T-n és ha $T[i] < min \rightarrow$ új $min = T[i]$

	0	1	2		$n-1$
T	8	5	16	...	7
	$T_{[0]}$	$T_{[1]}$	$T_{[2]}$		$T_{[n-1]}$

- pszeudo-kód:

```
min=T[0]
   for i=1 to n-1:
   if T[i]<min:
      min=T[i]
      min-hely=i
return(min, min-hely)
```

- jóság: mert amikor a legkisebb elemhez érek, akkor az minimumba kerül be és ott is marad
- input mérete $= T$ hossza (n)
- elemi lépés = értékadás $(\leq n)$és összehasonlítás $(n-1)$ $(\leq 2n$ ha minimum hely)
- elvárt output = T tömb minimális eleme – $O(n)$
- $LSZ \leq 2n - 1$
- LSZ legfeljebb olyan n-szerű
- LSZ $O(n)$ (ordó)
- definíció: van olyan c konstans (n-től független), hogy az LSZ legfeljebb $c \cdot f(n)$ ha n elég nagy, (ha $n \geq n_0$ kisebb/küszöb?) akkor LSZ $O(f(n))$ → ordó $f(n)$
- ha $LSZ \leq 17n^2 + 1000n - 3 \rightarrow O(n^2)$, mert ha
  $LSZ \leq 17n^2 + 1000n - 3 \leq 17n^2 + 1000n \leq 17n^2 + n^2 = 18n^2$
  $\rightarrow$ igaz, ha $n \geq 1000$

2018. 09. 13. – előadás

Rendező algoritmusok

- $LSZ(n) =$ hány elemi lépést tesz az algoritmus egy n méretű inputon a legrosszabb esetben
- $LSZ(n) \leq \underbrace{\dots\dots\dots}_{\text{felső becslés}}$
- c konstans nem fontos – elhagyható
- definíció: $LSZ(n) O(f(n))$ ha van $\begin{cases} c \\ n_0 \end{cases}$ $LSZ(n) \leq c \cdot f(n)$ ha $n \geq n_0$
  - például: $LSZ(n) \leq 8n^2 + 3n - 7 \underset{n\geq 1}{\leq} 8n^2 + 3n = 8n^2 + 3n^2 = \underbrace{11}_{c} n^2$

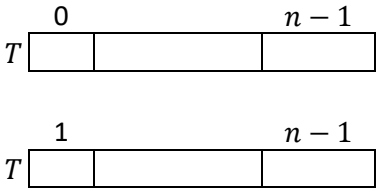
Példák

Függvény lépésszámok

- n $O(n)$
- $100n^2 + \log_{2n} + 7$ $O(n^2)$
- $\frac{n^3}{10000}$ $O(n^3)$
- $7n$ $O(n)$
- $38n + 17$ $O(n)$
- $10n - 3$ $O(n)$
- $n^2 - n + 5$ $O(n^2)$
- $3n^2 - 27n + 3$ $O(n^2)$

Kiválasztásos rendezés (Selection sort)

- $O(n^2)$ összehasonlítás
- $O(n)$ csere
- 0. kör:
  - minimum keresés 0-tól $n-1$-ig terjedő tömbben és minimum $T[0]$-ba mozgatása
- 1. kör:
  - minimum keresés 1-tól $n-1$-ig terjedő tömbben és minimum $T[1]$-be mozgatása



- pszeudo-kód:

```
for j=0 to n-2                          – ≤ n lefutás
   min=T[j]
   min-hely=j
   for i=j+1 to n-1                     } ≤ c · n lépés
      if T[i]<min
         min=T[i]
         min-hely=i
   csere T[j] és T[min-hely]
```

	0	1	2	3	4
T	8	5	−4	0	10

$j = 0$ után

T	−4	5	8	0	10

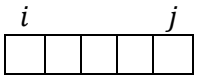
- jóság:
  - 0. körben a minimum bekerül előre, 1. körben a 2. legkisebb minimum kerül …
- lépésszám:
  - $LSZ(n) =$ összehasonlítás, értékadás – $O(n^2)$
  - $\leq \underbrace{n}_{\leq n \text{ db } j} \cdot \underbrace{cn}_{\text{egy } j \text{ esetén} \leq c \cdot n \text{ munka}}$ csere – $O(n)$

Lineáris keresés

- adott: T különböző számokkal
- kérdés: s helye T-ben
- lineáris keresés: végig a tömbön, ha $\begin{matrix}\text{van } s\rightarrow \text{megvan}\\ \text{nincs } s\rightarrow \text{None}\end{matrix}$
- $LSZ(n) \leq n$

Bináris keresés – oszd meg és uralkodj (Binary sort)

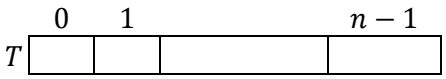
- ha T rendezett → helyek egymás után indexelve
- ötlet: s és $T[\text{közép}]$ összehasonlítása
  - $T[\text{közép}] > s \rightarrow$ keresés az 1. félben
  - $T[\text{közép}] = s \rightarrow$ ✓
  - $T[\text{közép}] < s \rightarrow$ keresés a 2. félben
- kérdés:
  - közép:
    - alsó egész rész: $\left\lfloor \frac{i+j}{2} \right\rfloor$
    - felső egész rész: $\left\lceil \frac{i+j}{2} \right\rceil$



  - ha T üres → none
- lépésszám:
  - $LSZ(n) =$ összehasonlítás $= ?$
  - $n = 2^k \rightarrow (1 \text{ öh.}) \rightarrow 2^{k-1} = \frac{n}{2} \rightarrow (2 \text{ öh.}) \rightarrow \frac{n}{4} = 2^{k-2}$ ... $2^0 = 1$
  - $\leq k + 1 = \log_2 n + 1 \leq 2 \cdot \log_2 n \rightarrow O(\log_2 n)$

Beszúrásos rendezés (Insertion sort)

- $O(n \cdot \log n)$ összehasonlítás
- $O(n^2)$ mozgatás
- $T[0]$ 1 db elem → rendezett
- $T[1]$-t beszúrja 0-ba
- $T[2]$-t beszúrja 1-be
- általánosan:
  - mikor $T[i]$ jön: $T[i]$ beszúrása $\begin{matrix}\nearrow \text{ helyének megkeresése}\\ \searrow \text{ elemek jobbra mozgatása}\end{matrix}$



- lépésszám:
  - $LSZ(n) - n$ db beszúrás
  - minden keresés: $O(\log n)$
  - összes mozgatás: $1 + 2 + .. + n - 1 \rightarrow \frac{n \cdot (n-1)}{2}$
  - $LSZ(n) \leq n \cdot O(\log n) + \frac{n \cdot (n-1)}{2} \leq c \cdot n^2 < c \cdot \log n \leq c \cdot n \rightarrow O(n^2)$

Buborékrendezés (Bubble sort)

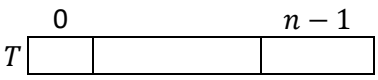
- $O(n^2)$ összehasonlítás
- $O(n^2)$ csere
- pszeudo-kód:

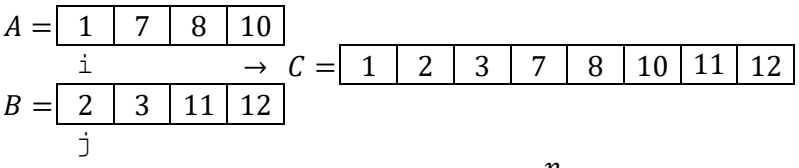
```
for  i = n to i > 0
   for j = 0 to i-1
      if tömb[j] > tömb[j+1]
         csere: tömb[j], tömb[j+1]
```

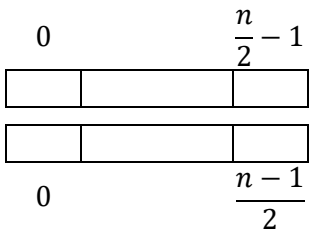
2018. 09. 27. – előadás

Összefésüléses rendezés (ÖFR) (Merge sort)

- input: T tömb
- output: rendezve T tömb
- ÖFR n hosszú T tömbre:
  - ha $n = 1$ → kész vagyunk, $output = input$
  - ha $n \geq 2$:
    - ÖFR a T 1. felére → A tömb
    - ÖFR a T 2. felére → B tömb
    - A és B összefésülése
      → $i = 0, j = 0, l = 0$
  - 1. kérdés – 1. fele
    - ha n = páros: $\frac{n}{2}$
    - ha n = páratlan: $\frac{n-1}{2}$
  - 2. kérdés – A és Bösszefésülése
    - input: rendezett A és B tömbök
    - output: 1 db $len(A) + len(B)$ rendezett tömb A és B elemeivel







- pszeudo-kód:

```
while i<len(A) and j<len(AB):
   if A[i]<B[j]:
      C[i]=A[i]
      i+=1
   else:
      C[l]=B[j]
      j+=1
   l+=1
while i<len(A):
   D[l]=A[i]
   i+=1
   l+=1
while j<len(B):
   D[l]=B[j]
   j+=1
   l+=1
```

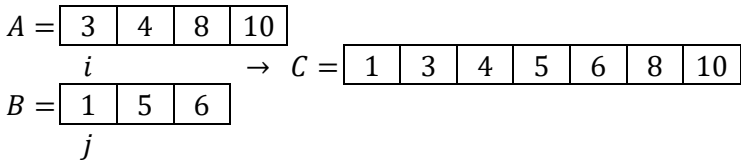
- `i`: itt tartok a A tömbben
- `j`: itt tartok a B tömbben
- `l`: itt tartok az outputban

- összefésüléses rendezés jósága:
  - C első eleme: A és B legkisebb eleme, mert a minimum elem $A[0]$ vagy $B[0]$
  - később: mindig a maradék legkisebb eleme kerül C-be
  - A és B összefésülésének lépésszáma:
    - összehasonlítás ≤ mozgatások
    - mozgatás = $len(A) + len(B)$
      → $LSZ = O(len(A) + len(B))$

- jóság teljes indukcióval:
  - n-re vonatkozó indukcióval
  - $n = 1$-re ✓
  - feltételezve, hogy $n = 1$-re igaz → ÖFR jó, $ln \leq n - 1$ méretű a tömb → A és B valóban rendezett A és B ÖFR-e is jó
  - ÖFR n méretűre:
    - $\log_2 n = \begin{cases} \text{ÖFR } n \text{ méretűekre} + O(n) \\ \text{ÖFR } \frac{n}{2} \text{ méretűekre} + O(n) \\ \text{ÖFR } \frac{n}{4} \text{ méretűekre} + O(n) \\ \text{ÖFR } \frac{n}{8} \text{ méretűekre} + O(n) \end{cases} \} \rightarrow O(n\, logn)$
- lépésszám:
  - $LSZ(n) = O(n \cdot \log n)$

Példa

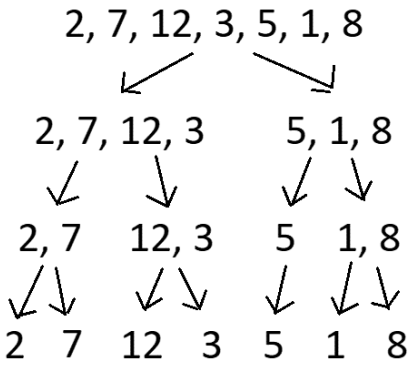
- $A = 3,4,8,10$
- $B = 1,5,6$
- pszeudo-kód:




```
amíg: i <len(A) és j<len(B)
   ha A[i]≤B[j]
      A[i] megy az output következő cellájába
   egyébként:
      B[i] megy az outputra
   amelyik nem fogyott el azt átmásolom az output végére
```

Példa – ÖFR 2,7,12,3,5,1,8-ra

- ÖFR a tömb 1. felére $(2,7,12,3) \rightarrow 2,3,7,12$
  - ÖFR 2,7-re
  - ÖFR 12,3-ra
- rendezés a 2. felére $(5,1,8) \rightarrow 1,5,8$
- 2 fél rendezése $(2,3,7,12)$ és $(1,5,8) \rightarrow 1,2,3,5,7,8,12$



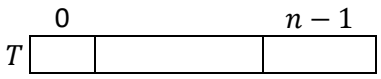
Rendezés

- van-e gyorsabb rendező algoritmus?
  - definíció: összehasonlításalapú rendező alap csak 2 módon nyúl a T tömb elemeihez
    - mozgatás
    - $T[i] < T[j]$ összehasonlítás
  - legjobb: összefésüléses: $O(n \cdot \log n)$
  - minden összehasonlítás alapú rendező algoritmus LSZ-a $\geq \frac{1}{4} \cdot n \cdot \log n$

2018. 10. 04. – előadás

Ládarendezés (Box sort)

- input: T tömb, minden eleme egész és $T \in \{0,1 \dots n-1\}$
- output: T rendezve = B tömb
- pszeudo-kód – menete:




```
m méretű (0-val teli) B tömb (bool tömb) létrehozása
for i=0 to n-1:
   B[T[i]]=1
for g=0 to n-1:
   if B[j]==i:
      print j
```

$T =$

1	8	3	7	2	6

$m = 9$

elején:

	0	1	2	3	4	5	6	7	8
$B =$	0	0	0	0	0	0	0	0	0

végén:

	0	1	2	3	4	5	6	7	8
$B =$	0	1	1	1	0	0	1	1	1

- kérdések: mért helyes? mennyi lépésszám?
  - LSZ:
    - $O(1)/O(m)$
    - $O(n)$
    - $O(m)$

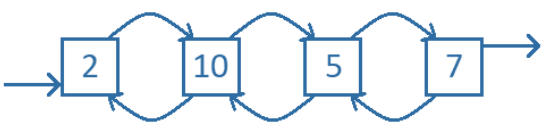
    } $O(n+m)$
  - egyik rész $\leq C_1 \cdot n$
  - másik rész $\leq C_2 \cdot m$ } $\rightarrow C_3$
  - egész $\leq C_1 \cdot n + C_2 \cdot m \leq \max\{C_1 C_2\}(n+m)$
- jóság: outputon a sorrend $= B$ indexeinek sorrendje és indexelés rendezett
- megjegyzések:
  - nem összehasonlítás alapú rendezés
  - ha $n = 0(n)$
    - pl.: $n = 1000n$
    - vagy $m =$ konstans $\rightarrow O(n+m) = O(n)$
  - csak akkor jó, ha m ismert és nem nagy
    - pl.: ha $m = 2^{128}$ $n \approx 20000$
- számrendezés:
  - valóságban nem csak számok rendezése (pl.: különböző adatok)
  - objektumok tárolása (pl.: több adatból álló sor; egyik komponens alapján könnyű keresni)

Példa

- cél: gyors, komponens alapú keresés
- 1. megoldás:
  - rendezett tömb a komponens alapján és utána bináris keresés $O(\log n)$
  - kitérő: T tömb
    - adatszerkezet = tárolási mód, ahol 1 lépésben i cellára lehet menni
    - beszúrás nehéz: $O(n)$ $[\leq n$ lépés$]$
    - törlés: $O(n)$
- 2. megoldás: láncolt lista

Láncolt lista (Linked list)

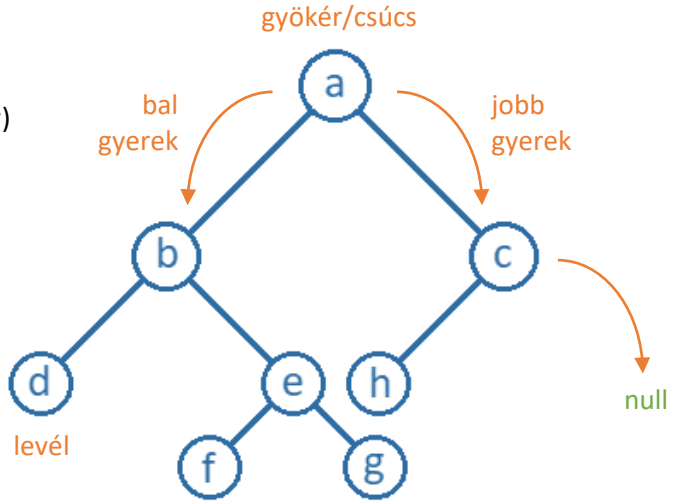
- beszúrás: $O(1)$
- keresés: $O(n)$ → törlés: $O(n)$
- különbség: tömbben vannak indexek
- $\underbrace{\text{keresés,}}_{\text{nehéz listában}}$ $\underbrace{\text{bekérés, törlés}}_{\text{nehéz tömbben}}$
- cél: adatszerkezet, ahol keresés, beszúrás, törlés is gyors → bináris fa



Bináris-fa (Binary tree)

Részei

- gyökér (pl.: a)
- szint (pl.: a, $b - c$)
- csúcs: ≤ 2 gyereke (pl.: $a \rightarrow b$ & c)
- levél: 0 gyerek (pl.: d, f, g, h)



Tárolás - mutatókkal

- gyökérmutató
- minden csúcsnál:
  - szülő mutató
  - bal gyerekmutató
  - jobb gyerekmutató
- Ø gyerek: nullpointer
- például:
  - $szülő(b) = a$
  - $bal - gyerek(f) = null$
  - $jobb - gyerek(e) = g$
- kérdés: eljárás a fa összes csúcsának egyszeri végig nézésére (teljes lista)

Preorder fabejárás

- rekurzív
- input: lineáris fa egy x csúcsa
- cél: x-nél gyökerező farész bejárása
- működés:
  - $pre(x)$:
    - $x\ látogatása$
    - $pre(bal\text{-}gyerek(x)), ha\ van$
    - $pre\big(jobb\text{-}gyerek(x)\big), ha\ van$

$$a\ \underbrace{b\ \overbrace{d}^{pre(d)}\ \overbrace{e\ f\ g}^{pre(e)}}_{pre(b)}\ \underbrace{c\ h}_{pre(c)}$$

- példa:
  - $pre(a)$:
    - $a\ látogatása$
    - $pre(b)$:
      - $b\ látogatása$
      - $pre(d)$
      - $pre(e)$:
        - $e\ látogatása$
        - $pre(f)$
        - $pre(g)$
    - $pre(c)$:
      - $c\ látogatása$
      - $pre(h)$

Inorder bejárás

- rekurzív
- input: x csúcs
- cél: x-nél gyökerező farész bejárása
- működés:
  - $in(x)$:
    - $in(bal\text{-}gyerek(x))$
    - x *látogatása*
    - $in(jobb\text{-}gyerek(x))$

$$\underbrace{\overbrace{d}^{in(d)}\ b\ \overbrace{f\ e\ g}^{in(e)}}_{in(b)}\ a\ \underbrace{h\ c}_{in(c)}$$

- példa:
  - $in(a)$:
    - $in(b)$:
      - $in(d)$
      - b *látogatása*
      - $in(e)$:
        - $in(f)$
        - e *látogatása*
        - $in(g)$
    - a *látogatása*
    - $in(c)$:
      - $in(h)$
      - c *látogatása*

Postorder bejárás

- rekurzív
- input: x csúcs
- cél: x-nél gyökerező farész bejárása
- működése:
  - $post(x)$:
    - $post(bal\text{-}gyerek(x))$
    - $post(jobb\text{-}gyerek(x))$
    - x *látogatása*

$$\underbrace{\overbrace{d}^{post(d)}\ \overbrace{f\ g\ e}^{post(e)}\ b}_{post(b)}\ \underbrace{h\ c}_{post(a)}\ a$$

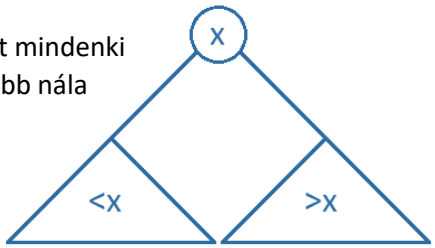
- példa:
  - $post(a)$:
    - $post(b)$:
      - $post(d)$
      - $post(e)$:
        - $post(f)$
        - $post(g)$
        - e *látogatása*
      - b *látogatása*
    - $post(c)$:
      - $post(h)$
      - c *látogatása*
    - a *látogatása*

Lépésszám

- $pre(x)$:
  - 3 lépés/x csúcs:
    - x *látogatása*
    - $bal\text{-}gyerek\ van\text{-}e$
    - $jobb\text{-}\ gyerek\ van\text{-}e$
  - $\rightarrow\ O(1) =$ konstans lépés/csúcs egy x-re
  - összesen: $O(n)$ $\quad n =$ csúcsok száma
- inorder és postorder esetén is hasonló

Bináris keresőfa (Binary search tree)

- bináris fa, csúcsaiban számok vannak
- minden x értékre telesül, hogy a bal gyerekei között mindenki kisebb nála, a jobb gyerekei között mindenki nagyobb nála



Műveletek

- o = szintszám
- s = érték

Keres(s) (Search)

$$s \sim gyökér$$

$balra \leftarrow gyökér > s$	$=$	$s > gyökér \rightarrow jobbra$
megyek és ua. úgy tovább	✓	megyek és ua. úgy tovább

Beszúrás(s) (Insert)

- $keres(s)$, aminek a végén észrevesszük, hogy s nincs a fában
- pont ott vesszük észre, ahol lennie kéne → oda beszúrás

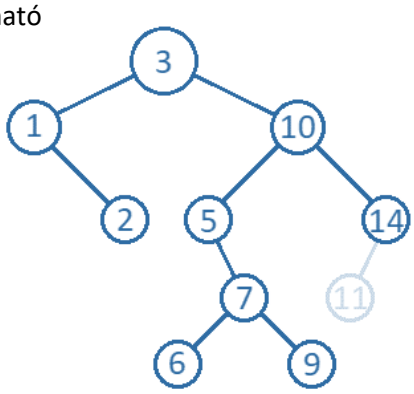
Törlés(s) (Delete)

- $keres(s)$
  - megtalálom → törlés → nincs ✓
  - megtalálom → 3 eset:
    - s-nek nincs gyereke → s törlése (pl.: 1-es jobb gyerek mutatója $null$ lesz)
    - s-nek 1 gyerek van → s helyére kerül az 1 gyerek
    - s-nek 2 gyereke van – mi legyen s helyén:
      - bal oldal legnagyobb elemét
      - jobb oldal legkisebb elemét
- lépésszám:
  - $keres(s)$: $O(h)$, utána $O(1)$
  - (konstans) állítás a könnyű esetben és a nehézben minimumkeresés $O(h)/O(h)$ szintet fel és $O(1)$ esetleg a 2 törlésre → $O(h)$
  - $\log_2 n \leq h \leq n$
  - legkisebb szint, ha
    - 1. szint 1
    - 2. szint 2
    - …
    - k. szint 2^k

      → tele van mind
  - $n = 1 + 2 + 2^2 + \cdots + 2^{k-1} = 2^k - 1$
  - $2^k - 1 \rightarrow 2^k = n + 1 \rightarrow k = \log_2(n+1)$
  - ha teljes bináris fa van → minden művelet $O(\log_2 n)$

Kiegyensúlyozás

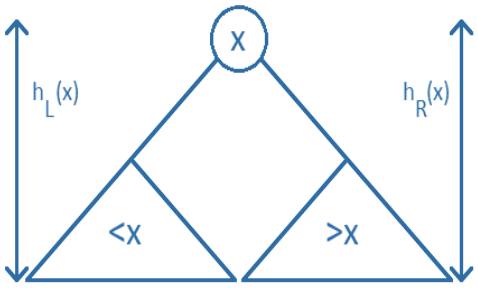
- baj: beszúrás, törlés elszúrja a teljességet → h nem lesz $O(\log_2 n)$
- megoldás: kiegyensúlyozott bináris keresőfa:
  - bináris keresőfa + tulajdonság a fa alakjára
    - $h = O(\log_2 n)$
    - e tulajdonság törlés és beszúrás után gyorsan helyreállítható



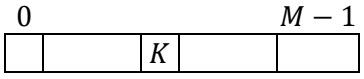
Példa

- $keres(3)$: 1 lépés
- $keres(1)$: gyökérben 3 → ha van 1, akkor biztos balra
  → $bal\text{-}gyerek(3) = 1$
- $keres(9)$: keresési út: $3, 10, 5, 7, 9$ ✓
- $keres(11)$: $3, 10, 14$ → vége → nincs, tehát: $beszúrás(11)$

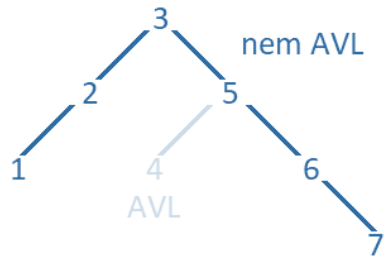
AVL-fa (AVL-tree)



- bináris keresőfa, ahol $\forall x$
- $h_L(x) =$ bal fa magassága
- $|h_L(x) - h_R(x)| \leq 1$
- beszúrás után: ≤ 2 forgatás elég az AVL visszaállítására
- törlés után: $\leq h \leq 1{,}44 \log_2 n$ forgatás elég
- másik megoldás számok tárolására: keres, beszúr, törlés
  - például: Neptun kód alapú ládarendezés szerinti tárolás
  - minden művelet 1 lépés
  - hány cella? $36^6 = 2 \cdot 10^9 - 25000$ diáknak
  - módosított ötlet: hash
  - alaphelyzet: számok tárolása

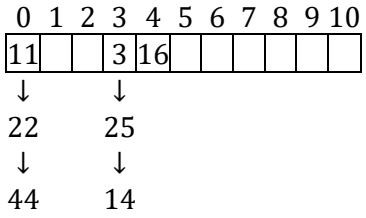


  - számok $\in \{0,1,\dots,m-1\}$
  - van egy h hash függvény: $a: k \to h \in \{0,1,\dots,M-1\}$ ahol $M \ll n$
  - ha K-ra $h(K)$-t kiszámolom és K-t a $k(K)$ cellába rakom
  - pl.: $h(K) = K$ M-mel vett osztási maradéka – $h(K) = K \bmod M$
- probléma: ütközés



Példa

- $h(K) = K \bmod M$
- $h(3) = 3$ $h(11) = 0$ $h(15) = 4$



2018. 10. 18. – előadás

Hash

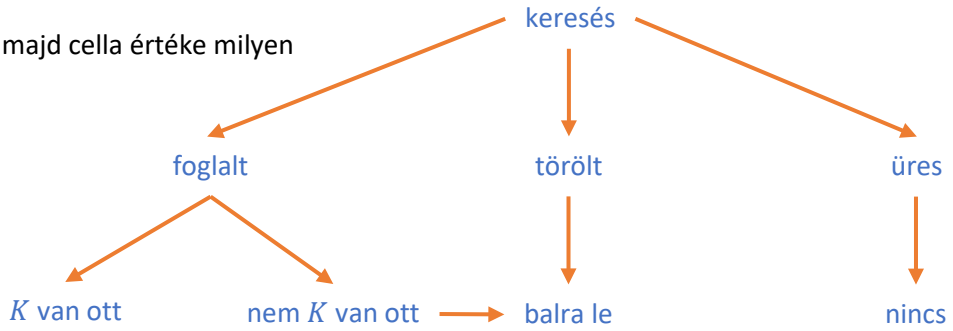
- hash: vödrös, lineáris próba

Cellák állapota

- foglalt: van értéke
- üres: nincs és nem is volt értéke
- törölt (*): nincs, de volt értéke

Keres(k)

- kiszámítás, majd cella értéke milyen

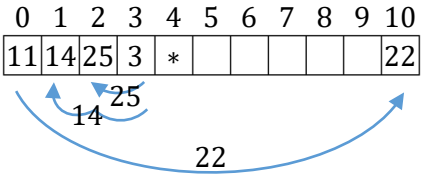


Beszúrás(k)

- $keres(k)$ + annak megfelelően beszúrás, ahova nem foglalt (ha foglalt → balra menni)

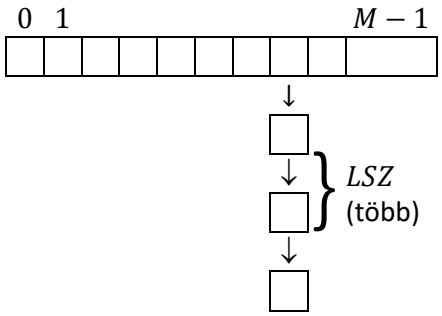
Törlés(k)

- $keres(k)$ + töröltre állítás



Hatékonyság

- legrosszabb eset LSZ-a:
  - keresésnél: n vödrösnél/lineáris próba
- átlagosan jó (nem legrosszabb eset)
- vödrös hash:



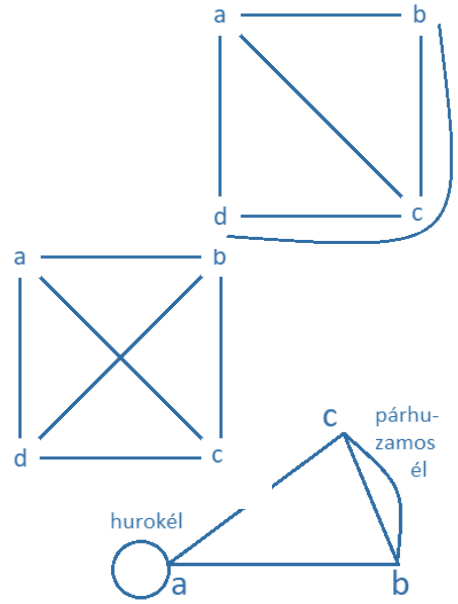
  - ha szép a hash (= egyenletes szétszórás) függvény és M helyre rakok n elemet
  - $\frac{n}{M}$ hosszú listák ≤ 1
  - tipikusan $M = 1.2$ (elemek várható száma) → minden lista rövid → lépésszám átlagosan: $O(1)$
  - nyílt címzésnél is:
    - $\leq 80\%$-a van tele a táblának
    - várhatóan ≤ 4 lépés minden művelet

	hash	AVL-tábla
legrosszabb eset LSZ	n	$O(\log_2 n)$
átlagos LSZ	konstans	$O(\log_2 n)$
kell-e rendezés	nem	igen
min, max	nem tudok	tudok
nagyon sok elem jön	baj	nem baj

Gráfok

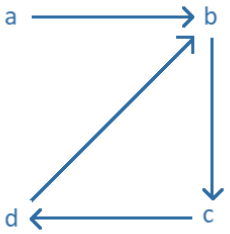
Irányítatlan gráf

- $G = (\underbrace{V}_{\substack{\text{vortex}\\ \text{csúcsok}}}, \underbrace{E}_{\substack{\text{edge}\\ \text{élek}}})$
- $n =$ csúcsok száma
- $e =$ élek száma – ki kivel van összekötve
- $\{1, 2, \dots, n\}$
- $\{v_1, v_2, \dots, v_n\}$
- $G = (\{a, b, c, d\}, \{ab, bc, cd, da\})$
- $G = (\{a, b, c, d\}, \{ab, ac, ad, bc, bd, cd\})$ ↗
- teljes gráf: minden él minden éllel összekötve (itt: K_4) →
- egyszerű gráf: nincs hurokél, nincs párhuzamos/összetett él →
  - $0 \leq e \leq \binom{n}{2} = \frac{n(n-1)}{2} = O(n^2)$
- összetett gráf: van hurokél és/vagy párhuzamos él →



Irányított gráf

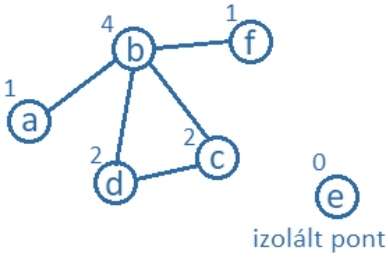
- $G = (\underbrace{V}_{\substack{\text{vortex}\\ \text{csúcsok}}}, \underbrace{E}_{\substack{\text{edge}\\ \text{élek}}})$
- E-beli éleknél fontos a sorrend
- $G = (\{a, b, c, d\}\{\overrightarrow{ab}, \overrightarrow{bc}, \overrightarrow{cd}, \overrightarrow{db}\})$ →
- egyszerű gráf:
  - $0 \leq e \leq n(n-1) = O(n^2)$



Fokszám

Irányítatlan gráfban

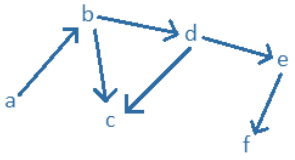
- v csúcs foka: $\underbrace{d}_{\substack{degree\\ fok}}(v) =$ ráilleszkedő élek száma
- $\sum_{v \in V} d(v) = 2e$
- egy él 2 csúcs fokszámát növeli



Irányított gráfban

- v csúcs foka: $d_{be}(v) = v$-be befutó él
- v csúcs foka: $d_{ki}(v) = v$-ből kifutó él
- $\sum_{v \in V} d_{be}(v) = \sum_{v \in V} d_{ki}(v) = e$

Út



Irányítatlan gráfban

- irányítatlan út $u_0, u_1, \dots, u_n$ úgy, hogy u_0u_1 él, u_1u_2 é,, $\dots, u_{n-1}u_n$ él
- egyszerű út: 1 csúcs maximum 1 alkalommal kerül bejárásra

Irányított gráfban

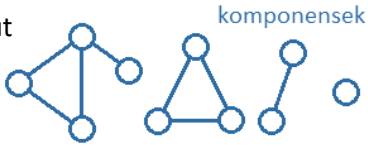
- irányított út: $u_0, u_1, \dots, u_n$, ahol $\overrightarrow{u_0u_1}, \overrightarrow{u_1u_2}, \dots, \overrightarrow{u_{n-1}u_n}$
- egyszerű irányított út: út, ahol 1 csúcs csak 1 alkalommal kerül bejárásra
- kör: $u_0, u_1, \dots, u_n$ rögzített, egyszerű, ahol $\overrightarrow{u_nu_0}$ is él

Kör

- irányított kör: $u_0, u_1, \dots, u_n$ rögzített, egyszerű út $\overrightarrow{u_n u_0}$

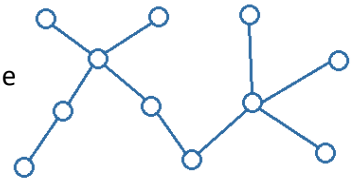
Összefüggőség

- irányítatlan gráf összefüggő: bármely csúcsból bármely csúcsba van út
- komponens: önmagukban összefüggő gráf



Fa

- összefüggő, körmentes, (általában) irányítatlan gráf
- tétel: ha F egy ≥ 2 pontú fa, $\rightarrow F$-ben van ≥ 2 elsőfokú csúcs
- bizonyítás: vegyük a leghosszabb utat F-ben, 2 vége: x, y
  $\rightarrow d(x) = d(y) = 1$,
  - mert ha x-nek lenne még 1 éle, ahol:
    - z nincs a leghosszabb úton: z-be menve még hosszabb út lenne
    - z rajta van a leghosszabb úton: kör lenne, de ez fa



- tétel: n csúcsú F fában $n-1$ él van
- bizonyítás: indukcióval n-re
  - $n = 1 \rightarrow 0$ él ✓
  - feltételezzük, hogy tudjuk, hogy n csúcsú fában $n-1$ él van, kéne: $n+1$ csúcsban n él van
  - $n+1$ csúcsú F' fában van elsőfokú csúcs → hagyjuk el ezt a csúcsot és a rá illő élet
  - F'' n csúcsú fa → $n-1$ éle van F''-nek → F'-nek n van

Gráfok reprezentációja

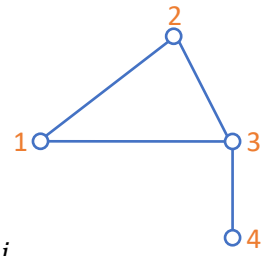
Szomszédossági mátrix

- n csúcsú gráf → nxn-es mátrix
- irányítatlan: $A\left[\underbrace{i}_{i.\text{ sor}}, \underbrace{j}_{j.\text{ oszlop}}\right] = \begin{cases} 1 & i-j \\ 0 & \text{különben} \end{cases}$ – szimmetrikus

A	1	2	3	…	n
1					
2					
3					
…					
n					

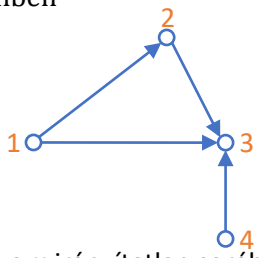
- például:

	1	2	3	4
1	0	1	1	0
2	1	0	1	0
3	1	1	0	1
4	0	0	1	0



- irányított: $A[i,j] = \begin{cases} 1 & i \rightarrow j \\ 0 & \text{különben} \end{cases}$ – nem szimmetrikus
- például:

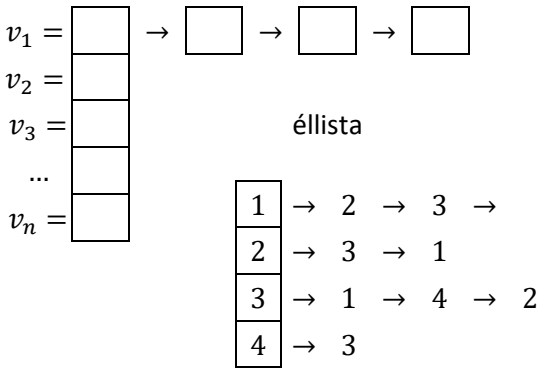
	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0



- v csúcs fokszám: hány 1 van a v irányítatlan sorában
- v csúcs kifoka: hány 1 van v irányított sorában
  v csúcs befoka: hány 1 van v oszlopában
- mátrix mérete: n^2

Szomszédossági lista / éllista

- lista, a v_1 szomszédai esetén v_1-ből megy él
- v csúcs fokszáma: $L[v]$ listájának hossza
- v csúcs kifoka: $L[v]$ listájának hossza
- v csúcs befoka: ahányszor v szerepel összesen a listában
- éllista mérete:
  - irányítatlan: $n + 2$
  - irányított: $n + e$



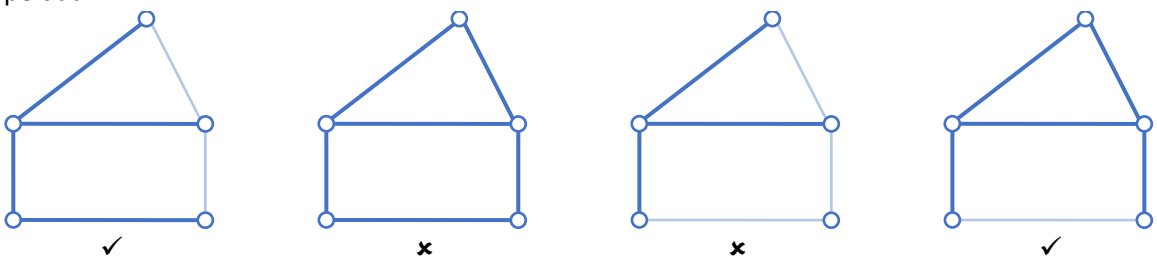
Hatékonyság

	mátrix	éllista
LSZ	$O(n^2)$	$O(n+e)$
van-e él (***i-j***)	1	$\frac{d(i)}{d_{ki}(i)}$
$\boldsymbol{d(i)}$ kiszámolása	n	$d(i)$
$\boldsymbol{d_{ki}(i)}$ kiszámolása	n	$d_{ki}(i)$
$\boldsymbol{d_{be}(i)}$ kiszámolása	n	$O(n+e)$
$\boldsymbol{i}$ szomszédban végigmenni	n	$d(i)$

2018. 10. 25. – előadás

Feszítőfa

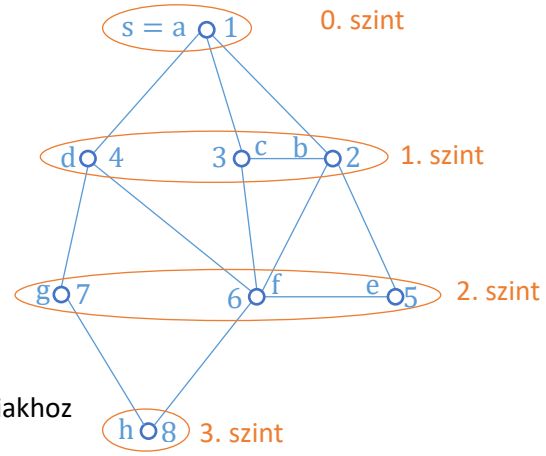
- G irányítatlan gráfban:
  - F fa
  - F élei G-nek is élei (F részgráfja G-nek)
  - G minden csúcsát eléri F
- például:



kört tartalmaz nem elérhető ∀ csúcs

- mikor van G-ben feszítőfa:
  - G összefüggő
    - bizonyítás: G-ben van F feszítőfa → F élein át minden csúcsból minden másik csúcs elérhető ← G összefüggő, ha G körmentes, akkor $F = G$

BFS (szélességi bejárás) (Breadth first search)



- megtalálja a legrövidebb utat s-ből minden v-be
- cél: irányítatlan G-ben feszítőfa
- kezdőcsúcs: s
- 1. körben: s szomszédjaiba megyek
- 2. körben: s szomszédok új szomszédjaiba, ha még nem voltam ott
- majd minden körben az előző körben bekerültek új szomszédjait járom be
- G összefüggő → felfedező élek (amikkel új csúcsba jutok) feszítőfát alkotnak
- F összefüggő, mivel új élek mindig kapcsolódnak korábbiakhoz
- F körmentes, mivel élek mindig új csúcsba vezetnek

2018. 11. 08. – előadás

Példa

- $bejárva[v] = \begin{cases} 1 \text{ ha } v = s \\ 0 \text{ különben} \end{cases}$
- $Q = \{s\}$ bejárt csúcsok, de a szomszédai még nem kerültek vizsgálatra
  - sor: lista, mely hátuljába kerül be az elem és az elejéből kerül kivételre
- $T = \emptyset$ (felfedező élek)
- pszeudo-kód:

```
while Q≠∅
   v=Q első csúcsa
   for all w szomszédjára v-nek
      if bejárva[w]==0
         bejárwa[w]=1
         (v,w) T-be
         w-t Q végére
```

- megjegyzések:
  - G lehet irányított/irányítatlan
  - ha G mátrixosan adott:
    - $O(n)$ →BFS $O(n^2)$

```
for w=1 to n
   if A[v,w]==1
```

  - ha G éllistás
    - $O(d(v))$ →BFS $O(n+e)$

```
for w in L[v]
```

  - minden csúcsot bejár, ahova vezet s-ből út

BFS(G)

- pszeudo-kód:

```
bejárva[v]=0 minden v∈V-re
T=∅
for v=1 to n:
   if bejárva[v]==0:
      BFS(G,v)
```

BFS(G,u)

- pszeudo-kód:

```
bejárva[u]=1
Q={u}
```

BFS(G,s)

- pszeudo-kód:

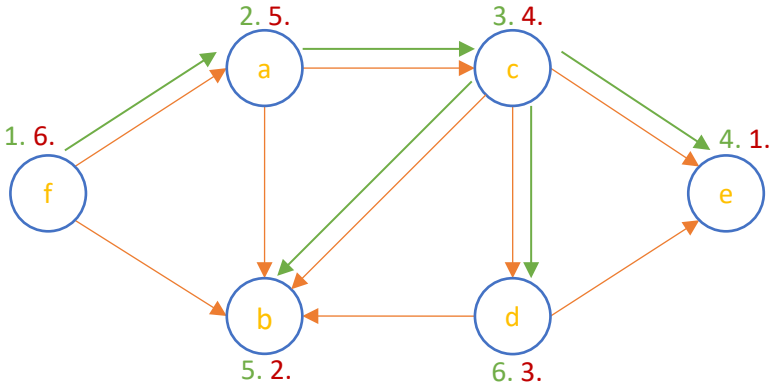
```
távolság[v]={0, ha v=s; * különben
Q={s}
T=∅
while Q≠∅
   v=Q első csúcsa
   kiveszem v-t Q-ból
   for all w szomszédjára v-nek
      if távolság[w]==*
         bejárwa[w]=távolság[v]+1
```

Lépésszám

- $BFS(G)$: $O(n)$
- $BFS(G,u)$: $O(n_i^2)$ vagy $O(n_i + e_i)$
- $n = \sum n_i \quad e = \sum e_i$

DFS (mélységi bejárás) (Depth first search)

- elve: megyek előre, amíg új csúcsba tudok menni és ha elakadok (nincs él új csúcsba), akkor visszalépek 1 élnyit (ahonnan idejöttem) és új élet próbálok



- elérési szám: ahányadikként kerül bejárásra
- befejezési szám: ahányadikként fordulok vissza

DFS(G)

- $bejárva[v] = 0$ minden $v \in V$-re
- $T = \emptyset$
- pszeudo-kód:

```
for v=1 to n:
   if bejárva[v]==0:
      DFS(G,v)
```

DFS(G, u)

- $bejárva[u] = 1$
- pszeudo-kód:

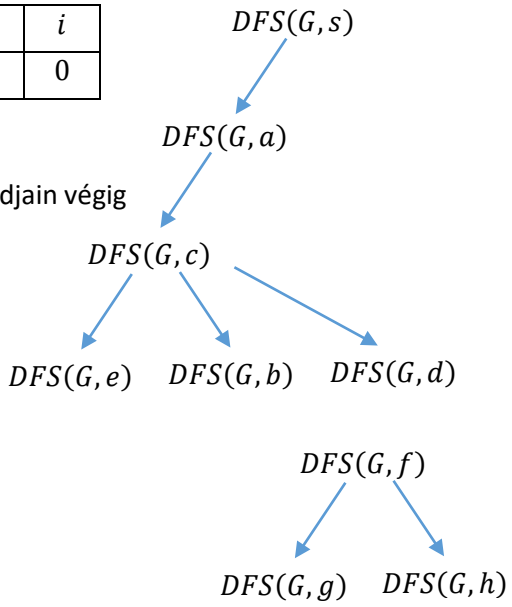
```
for minden w szomszédjára u-nak:
   if bejárva[w]==0:
      (u,w) T-be
      DFS(G,w)
```

Példa

- eleje:

$1 = s$	$2 = a$	$3 = b$	c	d	e	f	g	h	i
0	0	0	0	0	0	0	0	0	0

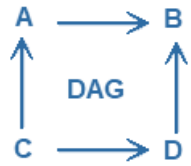
- $bejárva[s] == 0 \rightarrow$ igen $\rightarrow DFS(G, s)$
- végig haladás s szomszédjain:
  - $s \rightarrow a$: $bejárva[a] == 0 \rightarrow DFS(G, a)$: a szomszédjain végig
  - $s \rightarrow b$ (ha a-s ág kész):
    - $a \rightarrow c$: $bejárva[c] == 0 \rightarrow DFS(G, c)$
    - $a \rightarrow b$ …
  - ezután bejárva $[s, a, b, c, d, e] = 1$ különben 0 de $bejárva[f] = 0$
- megjegyzés:
  - G lehet irányított/irányítatlan
  - ciklikus → erdő (sok fa)
  - G irányítatlan összefüggő-e? – feszítőfa, ha igen
  - befejezési számok által eldönthető, hogy irányított gráfban van-e irányított kör

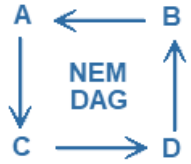


2018. 11. 15. – előadás

DAG (irányított körmentes gráf) (Directed acyclic graph)

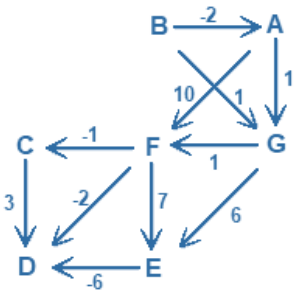
- irányított gráf, melyben nincs irányított kör
- topológikus sorrend: ha minden él kisebb sorszámból nagyobb sorszámúba megy
- minden DAG rendelkezik topológikus sorrenddel
- DFS-sel bejárás, befejezési számai fordított sorrendben = DAG topológiai sorrendje (ha nem alkot topológikus sorrendet, akkor a gráf biztos nem DAG)
- gráfban van topológikus sorrend → biztos nincs benne kör és biztos DAG





Példa

- topológikus sorrend: $B \to A \to G \to F \to C \to E \to D$
- élsúlyozott DAG-ban legrövidebb, -hosszabb út keresése

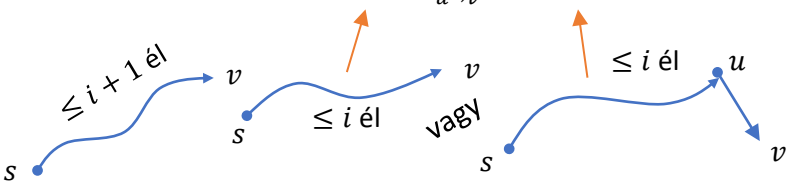


B	A	G	F	C	E	D
0	-2(B)	1(B-G) -2+2(B-A-G)	-2+10(B-A-F) 1+1(B-G-F) -2+1+1(B-A-G-F)	-2+10-1(B-A-F-C) 1+1-1(B-G-F-C) -2+1+1-1 (B-A-G-F-C)	1+6(B-G-E) -2+10+7(B-A-F-E) 1+1+7 (B-G-F-E) -2+1+1+7 (B-A-G-F-E)	1+6-6(B-G-E-D) -2+10+7-6 (B-A-F-E-D) 1+1+7-6 (B-G-F-E-D) -2+1+1+7-6 (B-A-G-F-E-D) -2+10-1+3(B-A-F-C-D) 1+1-1+3 (B-G-F-C-D) -2+1+1-1+3 (B-A-G-F-C-D) -2+10-2(B-A-F-D) 1+1-2 (B-G-F-D) -2+1+1-2 (B-A-G-F-D)
-	-2(B)	min: 0(B-A-G) max: 1(B-G)	min: 0 (B-A-G-F) max: 8 (B-A-F)	min: -1 (B-A-G-F-C) max: 7 (B-A-F-C)	min: 7 (B-G-E) max: 15 (B-A-F-E)	min: 0 (B-G-F-D) max: 10 (B-A-F-C-D)

2018. 11. 29. – előadás

Bellman-Ford algoritmus

- cél: legrövidebb út keresés s csúcsból
- input: G élsúlyozott gráf (nincs negatív kör), s csúcs
- feladat: s-ből minden v csúcsba legrövidebb út és ennek hossza
- algoritmus:
  - elve: $T[i,v] =$ minimális hosszú út hossza s-ből v-be, ahol az úton legfeljebb i él van
  - $i = 1,2,\dots,n-1$
- mivel nincs negatív kör → legrövidebb úton nincs kör
- legrövidebb út $\leq n-1$ élből áll
- $T[i,v] = f(x) = \begin{cases} 0 & \text{ha } v = s \\ c(s,v) & \text{ha van } s \to v \\ \infty & \text{különben} \end{cases}$
- $P[i,v] =$ minimális út $\leq i$ éllel honnan érkezik v-be
- $P[i,v] = s \ \ \forall v \in V$-re
- $i \to i+1$
- $T[i+1,v] = min\{T[i,v],\ \min\limits_{u \to v}\{T[i,u] + c(u,v)\}\}$

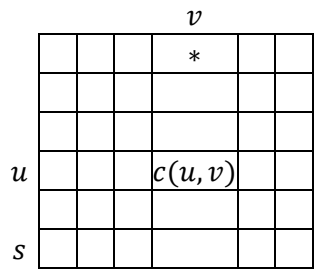


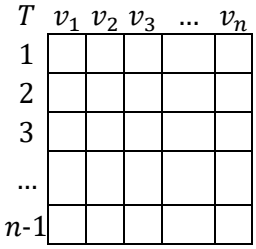
  - egyik: $\leq i$ él
  - másik: $\leq i+1$ él
- $P[i+1,v] =$

- $P[i, v]$ ha 1.<2.
- vagy u ha 2.<1. és a 2. min u-nál van

❖ lépésszám:

- G mátrixos esetén

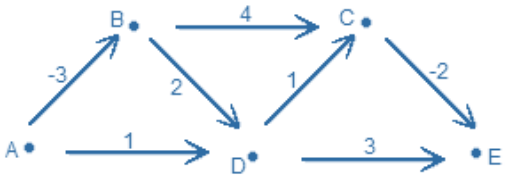




  - eleje: $O(n)$
  - i+1. sor kiszámolása= n db $T[i+1, v]$-t kiszámolni
  - egy darab ilyen: $O(n) < -O(n^2)$ és van $O(n)\ sor -> O(n^3)$
- G éllistás:
  - $i+1$ sávban $T[i+1, v]$ kiszámolása $O(d_{be}(v))$
  - ha van fordított éllista: $O(n+e)$-ben
  - teljes $i+1$ sor: $\sum O\left(d_{be}(v)\right) = O(e)$
  - összes sor: $O(n \cdot e)$
- megjegyzés:
  - ha $T[i+1, v] = T[i, v]$ → leállhat
  - ha kiszámolok még egy sort:
    - $T[n, v] = T[n-1, v]$ minden v-re?
    - 1. eset:
      - ha minden $T[n, v] = T[n-1, v]$ → nincs negatív kör
      - mert ekkor minden $T[i, v]$ ugyanaz, akármilyen nagy az i
      - ha van negatív kör → nincs így
    - 2. eset:
      - ha $T[n, v] \neq [n-1, v]$ legalább egy v-nél
      - → $\leq n$ élű út rövidebb, mint az $\leq n-1$ élű út
      - → biztosan van negatív kör

Példa

A	B	C	D	E
$0A$	$-3A$	$*$	$1A$	$*$
$0A$	$-3A$	$1B$	$-1B$	$4D$
$0A$	$-3A$	$1B$	$-1B$	$-1C$
$0A$	$-3A$	$1B$	$-1B$	$-1C$

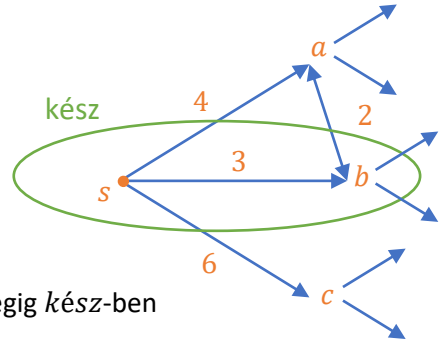


Hatékonyság

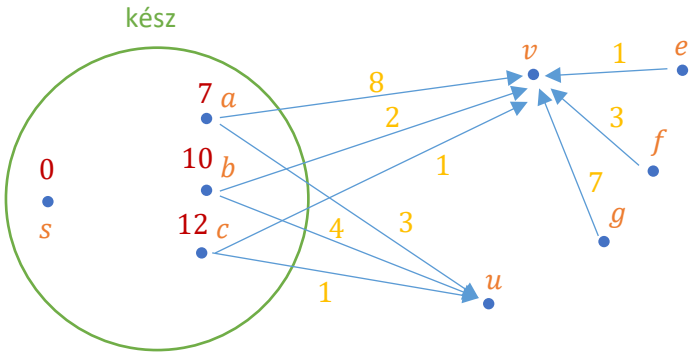
név	**mikor**	**lépésszám**	
		mátrixos	éllistás
BFS	nincs súlyozás = minden súly 1	$O(n^2)$	$O(n+e)$
legrövidebb út DAG-ban	G DAG	$O(n^2)$	$O(n+e)$
Dijkstra	minden élsúly≥ 0	$O(n^2)$	$O(e \cdot \log n)$
Bellman-Ford	nincs negatív kör	$O(n^3)$	$O(n \cdot e)$

Dijkstra algoritmus

- tegyük fel, hogy $c(f)$ élsúly ≥ 0
- ba él mentén út $\underbrace{távolság(b)}_{=3} + \underbrace{c(b,a)}_{=2} = 5$



- ötlet:
  - $kész$ halmaz: azon csúcsok, amikre ismerem a minimum utat
  - tömb: $távolság[v] = \begin{cases} \text{min út } v\text{-be, ha } v \in kész \\ * \quad \text{ha } v \in kész \end{cases}$
  - tömb: $d[v] =$ legrövidebb olyan út hossza s-ből v-be, ami végig $kész$-ben megy, kivéve az utolsó élet
    - tömb akkor nem $*$ ha $v \notin kész$



  - $d[v] = 12 \qquad d[u] = 10$

- algoritmus:
  - $kész = \{s\}$
  - $távolság[v] = \begin{cases} 0 & \text{ha } v = s \\ * & \text{különben} \end{cases}$
  - $d[v] = \begin{cases} * & \text{ha } v = s \\ c(s,v) & \text{ha } s \to v \\ \infty & \text{különben} \end{cases}$
  - $honnan[v] = s \, \forall v \in V$ ← honnan jött az ismert legrövidebb út s-ből v-be
  - pszeudo-kód:

```
while van olyan csúcs, amire d[v]≠x és d[v]≠∞:
   v* az a csúcs, amire d[v] minimális:
      v* kész-be megy
      távolság[v*]=d[v*]
      d[v*]= *
      for minden w szomszédjára v*-nak
         ha távolság[v*]+c(v*,w)<d[w]:
            d[w]:=távolság[v*]+c(v*,w)
            honnan[w]:=v*
```

Példa

távolság

A	B	C	D	E
0	$*$	$*$	$*$	$*$
0	3	$*$	$*$	$*$
0	3	$*$	5	$*$
0	3	$*$	5	6
0	3	7	5	6

kész

	A	B	C	D	E
$s = A$	$*$	3	∞	5	∞
A, B	$*$	$*$	8	5	∞
A, B, D	$*$	$*$	7	$*$	6
A, B, D, E	$*$	$*$	7	$*$	$*$
A, B, D, E, C	$*$	$*$	$*$	$*$	$*$

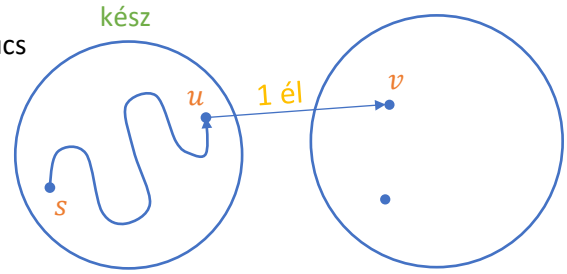
honnan

A	B	C	D	E
A	A	A	A	A
A	A	B	A	A
A	A	D	A	D
A	A	D	A	D

$A \to D \to E$

2018. 12. 06. – előadás

Dijsktra algoritmus (folytatás)



- input: G irányított gráf, élsúlyozott, ahol $c(f) \geq 0$ és s csúcs
- cél: s-ből ∀ más csúcsba minimum út és hossza
- elve →
- $távolság[u] = s$-ből u-ba menő minimum út hossza
- $d[v] =$ minimum olyan út hossza $s \rightsquigarrow v$-be, ahol $s \rightarrow$ valami u-ig $kész$-ben és utána 1 éllel $u \rightsquigarrow v$
- $honnan[v] =$ melyik $kész$-beli csúcsból jön a legjobb út
- $kész = \{s\}$
- $távolság[u] = \begin{cases} 0 \text{ ha } u = s \\ * \text{ különben} \end{cases}$
- $d[v] = \begin{cases} * \text{ ha } v = s \\ c(s,v) \text{ ha } s \rightarrow v \\ \infty \text{ különben} \end{cases}$
- $honnan[v] = \begin{cases} s \text{ ha } v = s \\ s \text{ ha } s \rightarrow v \\ * \text{ különben} \end{cases}$
- algoritmus:

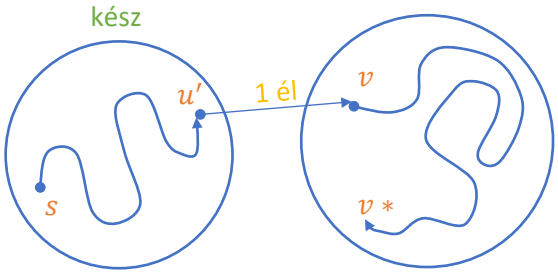
```
while van olyan v csúcs, amire d[]=*,∞:
   v* = min d[]-jű csúcs
   v* kész-be
   távolság[v*]:=d[v*]
   d[v*]:=*
   for ∀ w szomszédjára v*-nak:
      if d[w]=*:    //w∉kész
         if távolság[v*]+c(v*,w)<d[w]:
            d[w]:=távolság[v*]+c(v*,w)
            honnan[w]:=v*
```

Lépésszám

- G mátrixosan adott:
  - eleje: $O(n)$
  - $while$ ciklus: $\leq n-1$ loop van (mert minden körben 1 csúcs → $kész$)
  - 1 loopban:
    - 1 min keresés: $O(n)$
    - távolság, d: $O(1)$
    - for ciklus: $\leq 5 \cdot n$ mert $v*$ teljes sorának végigjárása a mátrixban → $O(n)$
  - → $LSZ = O(n^2)$
- G éllistás:
  - eleje: $O(n)$
  - $while$ loopok (összesen):
    - $(n-1)$ minimumkeresés $O(n^2)$
    - de for ciklus: $v*$ esetén $O(d(v*))$ lépés → összesen: $O(e)$
  - → $LSZ = O(n^2 + e)$ $\quad e \leq n^2 \rightarrow O(n^2)$
  - megjegyzés:
    - ha $d[v]$-ket egy kupac (heap) adatszerkezetben tároljuk → min keresés $O(\log n)$ és for ciklus lefut $O(d(v*)\log n)$-ben → $O(e \cdot \log n)$-ben megy

Helyesség

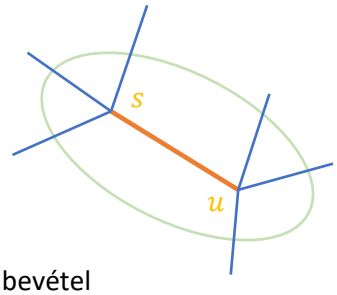
- kéne: $távolság[v]$ minden v-re jó
- $távolság[v] = \min s \rightsquigarrow v$ út hossza
- indukcióval: $kész$-be kerülés sorrendje szerint
  - $távolság[s] = 0$ helyes (nincs negatív él)
  - tegyük fel, hogy ∀ $távolság[u]$ helyes adott pillanatban → következő csúcsra, ami $kész$-be kerül $(v*)$ -ra is igaz:
    $távolság[v*] = \min s \rightsquigarrow v*$ út hossza – ezt kell belátni
    ↳ $d[v*] = távolság[u*] + c(u*, v*)$
  - kell:
    - A: van $távolság[u*] + c(u*, v*)$ hosszú út $s \rightarrow v* \rightsquigarrow ba$
    - B: ∀ út $s \rightsquigarrow v*$-ba $\geq d[v*] = távolság[u*] + c(u*, v*)$ hosszú
  - A:
    - $v*$-ba azaz út, ami min út
    - $\underbrace{s \rightsquigarrow u* \rightsquigarrow v*}_{távolság[u*]+c(u*,v*)}$ ← jó út
  - B:
    - tegyük fel hogy van egy út ∪ $s \rightsquigarrow v*$-ba megmutatom
    - u hossza $\geq d[v*] \rightarrow$ ✓
    - u hossza $= s \rightsquigarrow u'$ rész hossza $+c(u', v') + u' \rightsquigarrow v*$ rész hossza $\geq$ $\underbrace{távolság[u'] + c(u', v')}_{1} + v' \rightsquigarrow v*$ hossza $\geq \underbrace{d[v']}_{2} + v' \rightsquigarrow v*$ hossza $\geq d[v''] \geq \underbrace{d[v*]}_{3}$
    - 1: mert $távolság[u']$ helyes → ∀ út u'-be $\geq távolság[u']$ hosszú
    - 2: mert $d[v'] = min$ olyan út hossza, ami $kész$-ből 1 éllel megy v'-be
    - 3: $d[v*] \leq d[v']$ mert $v*$-ot választom most éppen be a $kész$-be



Feladat

- adott n város köztük közvetlen utak
- útfelújítás tervezése:
  - minden közvetlen útra ismert a költség
  - cél: bárhonnan bárhova el lehessen jutni (nem feltétlenül közvetlen út)
  - minimális költség
- gráfosan:
  - input: élsúlyozott n csúcsú irányítatlan, összefüggő G gráf (nem negatív élek)
  - cél: minimális összsúlyú rész (néhány él)
    - összefüggő részt alkotnak
    - körmentes – ha van kör: egy éle eldobható
    - minden csúcs lefednek
    - ↳ feszítőfa

Algoritmus



- ötlet:
  - indulás s csúcsból – innen fa építése
  - ∀ lépésben +1 él választása a fába
  - amíg ∀ csúcsot elérek
- pontosabban:
  - $lefedve$ halmaz: akiket elért a fa
  - ∀ lépéesben a minimális élt, ami kimegy $lefedve$-ből – épülő fába bevétel
- pszeudo-kód:

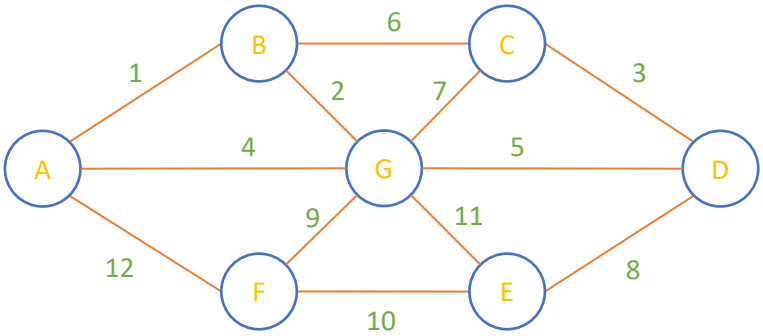
```
lefedve={s}
F=∅      //épülő fa
while van lefedve-ből kilépő él:
   min ⟦ilyen él⟧ (u,v) [(u∈) lefedve]
   (u,v)  F-be
   v lefedve-be
```

Prím algoritmus

- összefüggő súlyozott gráf minimális feszítőfájának meghatározása
- kiinduló csúcs kiválasztása – s
- fa csúcsainak kiválasztása, majd a gráf többi csúcsa közt futó élek közül mindig a legkisebbet
- kiválasztott él nem fabéli csúcsa: áttétel a fába az éllel együtt
- tömb: $lefedve$: már felfedezett csúcsok
- F: épülő feszítőfa

Példa

F	$lefedve$
AB	A
BG	B
GD	G
DC	D
DC	C
DE	E
GF	F



- be lehetne látni, hogy LSZ-a: $O(n^2)/O(e \cdot \log n)$
- helyesség: belátható