FÜGGVÉNYEK NAGYSÁGRENDJE

<u>Definíció</u>: $f = O(g)$ jelöli azt a tényt, hogy $\exists$ olyan $c, n_0 > \emptyset$ állandók, hogy $|f(n)| \leq c \cdot |g(n)|$ teljesül, ha $n \geq n_0$.

<u>Definíció</u>: $f = \Omega(g)$ jelöli azt a tényt, hogy $\exists$ olyan $c, n_0 > \emptyset$ állandók, hogy $|f(n)| \geq c \cdot |g(n)|$ ha $n \geq n_0$.

<u>Definíció</u>: Ha $f = O(g)$ és $f = \Omega(g)$ akkor $f = \Theta(g)$.

→ tehát az $f = O(g)$ egy felső becslés → az algo. legrosszabb esetben ennyit fut

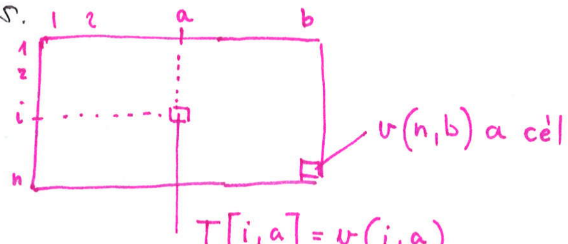
→ tehát az $f = \Omega(g)$ egy alsó becslés → az algo. legalább ennyit fut

DINAMIKUS PROGRAMOZÁS

→ optimum meghatározása kisebb feladatok optimumának felhasználásával.

→ <u>hátizsák probléma</u>: $s_1, s_2 \dots s_m$ súlyok, b súlykorlát és $v_1, v_2 \dots v_m$ értékek.

- feladat: $I \subseteq \{1 \dots m\}$ részhalmaz kiválasztása, melyre teljesül, hogy $\sum_{i \in I} s_i \leq b$ és $\sum_{i \in I} v_i$ maximális.
- költség: ~~O(b·L)~~ $O(n \cdot b)$



$v(i,a)$ = a maximális érték az $s_1 \dots s_i$ súlyokkal, $v_1 \dots v_i$ értékekkel és a súlykorláttal.

$$\boxed{v(i,a) = \max\{v(i-1,a);\ v_i + v(i-1, a - s_i)\}}$$

GRÁFALGORITMUSOK

<u>Szomszédossági mátrix</u>: $G = (V, E)$ gráf szomszédossági mátrixa (V elemeivel indexelve):
(adjacencia)

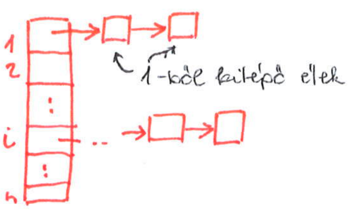
$$A[i,j] = \begin{cases} 1 & \text{ha } (i,j) \in E \\ 0 & \text{ha } (i,j) \notin E \end{cases}$$

→ irányítatlan gráfok esetén a szom. mátrix szimmetrikus lesz

→ ha élsúlyok is vannak:

$$A[i,j] = \begin{cases} c(i,j) & \text{ha } i \neq j \text{ és } (i,j) \in G \\ \emptyset & \text{ha } i = j \\ * & \text{egyébként} \end{cases}$$

<u>éllistás megadás</u>: a gráf $\forall$ csúcsához egy lista tartozik.



az $i \in V$ listában tároljuk az i-ből kilépő éleket (és azok súlyát ha vannak)

<u>tárigény</u>: $n + e$ cella

SZÉLESSÉGI BEJÁRÁS

→ algo: meglátogatjuk az első csúcsot, majd ennek a csúcsnak az összes szomszédját. Aztán ezen szomszédok összes olyan szomszédját amelyet még nem látogattunk meg.

- általánosan: vesszük a sor elején lévő x csúcsot → töröljük a sorból → meglátogatjuk azon y szomszédokat, amelyeket még nem → ezeket az y csúcsokat a sor végére tesszük.

- költség: $O(n+e)$

→ gráf szélességi feszítő erdejének felépítése lehetséges

- faél: azon él amelyek megvizsgálásukkor még be nem járt csúcsba mutatnak.
  - a faélek feszítő erdőt alkotnak

→ összefüggőség eldöntése: tetszőleges v csúcsból elindítjuk a szélességi bejárást → ha végzett ∄ van bejáratlan csúcs akkor G nem összefüggő

→ legrövidebb utak egy csúcsból: nagyon fontos hogy ez csak akkor működik, ha ∀ élsúly pontosan 1!
$O(n+e)$

- kiinduló csúcs: s
- $D[v]$ a v csúcs s-től való távolsága az s gyökerű szélességi fában.
- inicializáljuk $D[s]$-t $\emptyset$-ra
- az algo-t úgy módosítjuk, hogy amikor x csúcs még bejáratlan szomszédait a FIFO-ba rakjuk (= tehát az $x \to y$ él faél lesz) akkor végrehajtjuk a $D[y] = D[x]+1$ értékadást.

→ maximális párosítás keresése páros gráfokban: alternáló erdő építése + páros szinten a párosításbeli élek használata

- ~~∀ pontra lefuttatjuk~~
- mindig párosítatlan pontból építünk fát
- ha olyan pontba érünk ami párosítatlan → találtunk javítóutat
- összköltség: $O(n \cdot e)$

LEGRÖVIDEBB UTAK GRÁFOKBAN

I) EGY PONTBÓL AZ ÖSSZES TÖBBIBE → BELLMAN-FORD ALGORITMUS

- feltétel: a gráfban nem lehet negatív kör
- $T[1:n-1;\ 1:n]$ táblázat kitöltése
- $T[i,j]$ = a legrövidebb olyan $1 \rightsquigarrow j$ irányított utak hossza, melyek legfeljebb i élből állnak.
- ha az algo. lefut akkor $T[n-1, j]$-ből kiolvashatjuk s-ből j-be menő legrövidebb utak hosszát.
- első sor kitöltése: $T[1,j] = C(1,j)$

- feltesszük, hogy az i. sor már ki van töltve → a köv. sor kitöltése:

$$T[i+1,j] = \min\left\{T[i,j], \min_{k\neq j}\left\{T[i,k]+C[k,j]\right\}\right\} \qquad \boxed{O(n^3)}$$

→ Bellmann-Ford-nál tehát mindig a bevehető élek számát növeljük.

→ az algo. megáll, ha két ugyan olyan sort kapunk egymás után.

II) EGY PONTBÓL AZ ÖSSZES TÖBBIBE → DIJKSTRA ALGORITMUS

- feltétel: a gráfban nem lehet negatív él
- a probléma megoldásához, egy v csúcsaival indexelt D[] tömböt használunk.
  - → D[v] = az eddig megismert legrövidebb s ⇝ v utak hossza
- első lépés: D[v] = C[s,v]
- s szomszédai közül válasszuk a minimális távolságút
  D[x] = C(s,x) minimális ⟹ x-et betehetjük a KÉSZ halmazba (s mellé), hiszen x-be garantáltan nem tudunk rövidebb úton eljutni (nincsenek negatív élek).
  - → a KÉSZ halmaz azokat a csúcsokat tartalmazza, amelyek távolságát az s csúcstól ismerjük.
- ha x-en keresztül el lehet érni a többi csúcsot olcsóbban, akkor ezen csúcsok D[w] értékét frissítjük → tehát ha D[x] + c(x,w) < D[w] akkor frissítjük.
- újra válasszuk ki a D[] legkisebb elemét és folytassuk az algoritmust amíg ∀ csúcs a KÉSZ halmazba nem kerül.

$\boxed{\cdot \text{ lépésszám} : O(n^2)}$

- az algoritmus megvalósítása éllistás, kupacos módszerrel:
  - → akkor használjuk ha G-nek kevés éle van.
  - → a még nem KÉSZ halmaz-beli csúcsokat egy min. kupacban tároljuk, D[] értékek szerint.
    - a kezdeti kupacépítés O(n) lépésben megvan.
    - a minimum keresés pedig O(log n) lépésben megvalósul
    - a kupactulajdonság helyreállítása további O(log n) művelet (csak a választott csúcs szomszédaira kell elvégezni)
    - az összidő tehát $\boxed{O((n+e)\cdot \log n)}$
- legrövidebb utak nyomonkövetése: P[] tömb karbantartása, amely ∀ csúcshoz megadja az eddig megismert hozzá vezető legrövidebb út utolsó előtti csúcsát.

III) AZ ÖSSZES PONTPÁR TÁVOLSÁGÁNAK MEGHATÁROZÁSA – FLOYD ALGO.

- feltétel: a gráfban nem lehet negatív összsúlyú irányított kör
- a gráf adott a C[i,j] szomszédossági mátrixával.
- vezessük be a szintén n×n-es F_k mátrixot.
  - → $F_k[i,j]$ = a k-adik lefutás után F[i,j] azon i,j utak legrövidebbjeinek hossza amelyek közbülső pontjai k-nál nem nagyobb sorszámúak.

• $F_k[i,j]$-t mindig $F_{k-1}[i,j]$-ből számítjuk

• $F_k[i,j]$ jelentéséből nyilvánvaló, hogy egy $i \leadsto j$ út mely legfeljebb k sorszámú belső csúcsot tartalmaz vagy tartalmazza a k csúcsot vagy nem.

→ ha tartalmazza akkor $F_{k-1}[i,k]$-ból kiolvashatjuk az $i \leadsto k$ út minimális költségét

→ k-ból még ugye el kell jutnunk j-be, de ez is kiolvasható az $F_{k-1}[k,j]$ mátrixból.

→ a ~~k so~~ legfeljebb k sorszámú csúcsokat tartalmazó út költsége, ha k szerepel az ilyen $i \leadsto j$ utakban: $F_k[i,j] = \min_j \{ F_{k-1}[i,k] + F_{k-1}[i,j] \}$

→ ha nem tartalmazza k-t akkor $F_k[i,j] = F_{k-1}[i,j]$

• FLOYD - algoritmus:

```
for k=1 to n do
  for i=1 to n do
    for j=1 to n do
      F_k[i,j] = min{ F_{k-1}[i,j], F_{k-1}[i,k] + F_{k-1}[k,j] }
```

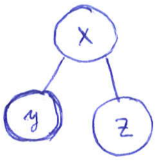
$\boxed{O(n^3)}$

ADATSZERKEZETEK

→ Kupac adatszerkezet

• teljes bináris fában tároljuk az elemeket.

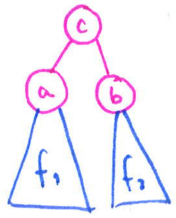
• kupac-tulajdonság: egy tetszőleges elem nem lehet nagyobb a fiaiban lévő elemeknél.



$y > x$ és $z > x$.

• műveletek kupacok esetén: jelölések: - f: részfa egy gyökere - itt tárolt elem: a - f fiai f_1, f_2 - a_1, a_2 a tárolt elemek

- felszivárog(f) / kupacol(f)



ha $\min\{a,b\} < c$ akkor $\min\{a,b\}$ helyet cserél c-vel majd a csere után (~~min{a,b} ba~~) a csere helyén lévő részfára kupacol()-t hívunk.

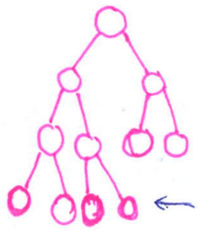
→ c addig megy lefelé amíg sérti a kupac tulajdonságot

f_1 & f_2 kupacok

- kupacépítés (f)

a fa v csúcsaira lentről felfelé, jobbról balra kupacol(v).

→ kupacépítés lépésszáma: $\boxed{O(n)}$



← csak innen hiányozhatnak levelek (teljes bin. fa miatt)

- mintör
a legkisebb elem a gyökérben van → ezt töröljük
a fa gyökerébe tesszük az utolsó szint legjobboldalibb elemét, majd kupacol(f)
→ a mintör lépésszáma: $\boxed{O(\log_2 n)}$

- beszúr(s)
új levelet adunk a fához, a levélbe tesszük az s elemet.
ezután addig mozgatjuk felfelé amíg nem teljesíti a kupac-tulajdonságot
vagyis amíg $x > s$, ahol x az s-et tartalmazó csúcs apja.
→ a beszúr lépésszáma: $\boxed{O(\log_2 n)}$

KERESÉSEK & RENDEZÉSEK

Keresés rendezett halmazban:

1) Lineáris keresés $\quad S = \{s_1 < s_2 \dots s_n\}$
- ha $x < s_1$ akkor biztosak lehetünk benne, hogy $x \notin S$
- ha $x = s \Rightarrow x \in S$
- ha $x > s_1$ akkor tovább haladunk $s_2 \dots s_n$ elemekre
  ⇒ ha $x \geq s_n$ akkor n db összehasonlítást végeztünk
- lépésszám: $O(n)$ / átlagosan: $\left(\frac{n}{2}\right) + 1$

2) Bináris keresés: könyv 28. oldal ⟶ lépésszám: $O(\log n)$

Minimumkeresés: n elem közül a minimális kiválasztásához legrosszabb esetben $n-1$ összehasonlítás kell.

Összehasonlítás alapú rendezők

1) Buborék-rendezés: az $A[1:n]$ tömböt növekvően rendezi
- ha valamely i-re $A[i] > A[i+1]$ akkor a két cella tartalmát kicseréli
- összehasonlítások száma: $\frac{n(n-1)}{2}$ $\quad$ cserék száma: $\leq \frac{n(n-1)}{2}$
- a buborék rendezés konzervatív: egyforma elemek egymás közti sorrendjén nem változtat.

2) Beszúrásos rendezés: tegyük fel, hogy $A[1:k]$ már rendezve van és a elemet $A[k+1]$-et be akarjuk szúrni. Az elem helyét lineáris vagy bináris kereséssel is megtalálhatjuk → ez befolyásolja a lépésszámot.

~~2a) Beszúrásos rendezés lineáris kereséssel~~ —— KÖNYV 31. oldal

3) Összefésüléses rendezés
- összefésülés: $A[1:i]$ és $B[1:j]$ már feldolgozásra került, tartalmuk már a $C[1:i+j]$-ben van növekvően, akkor

$$C[i+j+1] = \min\{A[i+1], B[j+1]\}$$ ← az eljárás neve összefésülés (MERGE)

- a rendezés megvalósítása: az $A[1:n]$ rendezetben tömb első felét majd második felét is rendezzük majd összefésüljük őket.

$$\text{MSORT}(A[1:N]) = \text{MERGE}(\text{MSORT}(A[1:\lceil N/2\rceil]), \text{MSORT}(A[\lceil N/2\rceil+1:N]))$$

- mozgatások száma: $2n\lceil \log_2 n \rceil$ $\quad$ tárigény: $2n$ cella

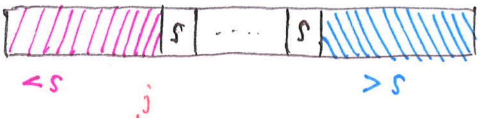
4) Kupacos rendezés

kupacot építünk, majd n db MINTÖR() műveletet végzünk

lépésszám: $O(n \cdot \log n)$ → a kupacos rendezés a leggyorsabb rendező algoritmus

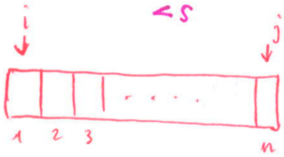
5) Gyorsrendezés

a rendezendő $A[1:n]$ elem egy véletlen módon választott s elemét kiválasztjuk → ezután a tömb elejébe mozgatjuk az s-nél kisebb elemeket, a végébe pedig s-nél nagyobbakat (kettő között pedig a kiválasztott s elem(ek) helyezkednek el.



az eljárás neve PARTÍCIÓ(s)

→ $A[1:n]$ $i=1$ $j=n$



- i-t növeljük amíg $A[i] < s$ teljesül
- j-t csökkentjük amíg $A[j] \geq s$ teljesül
- csere $A[j]$ és $A[i]$ tartalmát ha $i < j$
- ezután $i += 1$ & $j -= 1$
- ha a két mutató összeér akkor az s előfordulásait a „nagyobb rész" elejére mozgatjuk

→ példa a PARTÍCIÓ működésére: $s = 19$

24	5	3	35	14	23	19	43	2

i ugye rögtön megáll mert $24 > 19$, így i nem léphet tovább

j sem indul el, mert $2 \leq 19$

2	5	3	35	14	23	19	43	24

2	5	3	19	14	23	35	43	24

2	5	3	14	$\boxed{19}$	23	35	43	24

$O(n)$

19 a végleges helyén van

az algo. vázlata:

GYORSREND(A[1:n]) → átlagos esetben: $O(n \cdot \log n)$ → legrosszabb esetben: $O(n^2)$

1) válasszunk egy $s \in A$ véletlen elemet
2) PARTÍCIÓ(s) → eredménye: $A[1:k]$, $A[k+1:l]$, $A[l+1:n]$
3) GYORSREND(A[1:k]); GYORSREND(A[l+1:n])

Kulcsmanipulációs rendezések: többet tudunk a rendezendő adatokról

1) Ládarendezés (binsort)

tudjuk, hogy $A[1:n]$ elemei egy U halmazból kerülnek ki melyek mérete m
lefoglalunk egy B tömböt melyet U elemeivel indexelünk meg.

az algo működése:

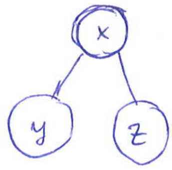
1) végigolvassuk az A tömböt és az $s = A[i]$ elemet a $B[s]$ lista végére fűzzük (konzervatív rendezés)

2) az elejétől a végéig végigolvassuk a B tömböt és az elemeket visszaírjuk az A tömbbe az olvasás sorrendjében.

- lépésszám: $O(n+m)$

2) Radix rendezés: lexikografikus rendezés → könyv 48. oldal

KERESŐFÁK



bal(x) = y
jobb(x) = z
apa(y|z) = x

- bináris fák: ∀ csúcsnak legfeljebb 2 gyereke lehet
- teljes bináris fa: ∀ csúcsnak pontosan 2 gyereke van

→ csúcsok száma: $2^{\ell}-1$ ahol ℓ a fa magassága

- bináris $\underbrace{\text{fák bejárása}}_{O(n)}$ lehet: preorder, inorder, postorder

- keresőfa tulajdonság: tetszőleges x csúcsra és az x bal oldali részfájában lévő tetszőleges y csúcsra igaz, hogy $elem(y) \leq elem(x)$. Hasonlóan ha egy z csúcs az x jobb fájában található, akkor $elem(z) \geq elem(x)$.

- tehát x bal oldalán a kisebb, x jobb oldalán a nagyobb értékek állnak.

- T.: bináris keresőfa elemeit a fa inorder bejárása növekvő sorrendben látogatja meg.

- más algoritmusok keresőfákban:

1) KERES(s, S): s elemet keressük az S fában ⟹ lépésszám: $O(\ell)$

ℓ - szintszám

→ ha $s = s'$ ⟹ találat

→ ha $s < s'$ ⟹ a bal részfában keresünk tovább

→ ha $s > s'$ ⟹ a jobb részfában keresünk tovább

- MIN: balra lépünk amíg lehet
- MAX: jobbra lépünk amíg lehet

TÓLIG(a, b, S): KERES(a, S) majd INORDER(a) (amíg b-t nem olvasunk)

2) BESZÚR(s, S): s elemet beszúrja a fába, ha nincs benne

→ KERES(s, S), ha nincs a fában akkor KERES utolsó csúcsához s értékének megfelelően új levelet veszünk fel, melyben s-et eltároljuk.

⟹ lépésszám: $O(\ell)$

3) TÖRÖL(r, s): 3 esetre bontható

3a) Ha s levél → simán töröljük a levelet

3b) Ha s-nek egy gyereke van → a gyerek a törölt szülő helyére lép

3c) Ha s-nek két gyereke van → jelölje x a törlendő s csúcs bal részfájának legnagyobb elemét. s törlése után x-et a helyére tesszük.

T.: egy véletlen sorozatból épülő fa építési költsége átlagosan $O(n \cdot \log_2 n)$

→ a fa mélysége átlagosan $O(\log_2 n)$

PIROS-FEKETE FÁK

Definíció: A piros-fekete fa egy bináris keresőfa, melyre teljesülnek a következők:

1) minden nem levél csúcsnak 2 fia van.
2) elemeket belső csúcsokban tároljuk.
3) teljesül a keresőfa tulajdonság
4) a fa minden csúcsa piros vagy fekete
5) a gyökér fekete
6) a levelek feketék
7) minden piros csúcs mindkét gyereke fekete
8) minden v csúcsra igaz, hogy a v-ből a levélbe vezető összes úton ugyanannyi fekete csúcs van.

→ további jelölések: F_v - v gyökerű részfa

$m(v)$ - v csúcs magassága, vagyis a leghosszabb v-ből levélbe vezető út hossza

$fm(v)$ - v fekete-magassága, v-ből levélbe vezető összes úton a fekete csúcsok száma.

T.: $\frac{m(v)}{2} \leq fm(v) \leq m(v)$

T.: F_v belső csúcsainak száma $b_v \geq 2^{fm(v)} - 1$

T.: p-f fában n elem tárolása esetén a fa magassága $\leq 2 \cdot \log(n+1)$

→ műveletek piros-fekete fákon:

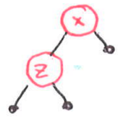
- KERES, MIN, MAX — $O(\log n)$
- BESZÚR — a beszúrás nem triviális, bevezetjük a forgatást, mint szükséges eszközt a pf-tulajdonság megtartásához.

→ szúrjuk be egy új csúcsot → új belső csúcs keletkezik

 ⇐ Ha z a gyökér, z legyen fekete

→ ha z nem gyökér, legyen az apja x

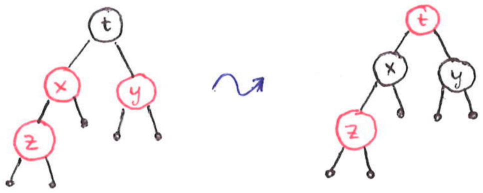
- ha x fekete ⇒ z legyen piros (fekete magasságok nem változnak)
- ha x piros ⇒ sérül a pf tulajdonság



→ mivel x piros ⇒ nem lehet a gyökér

- legyen x apja t → t fekete
  x testvére y

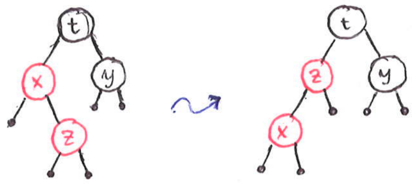
→ ha y piros ⇒ átszínezzük t-t pirosra



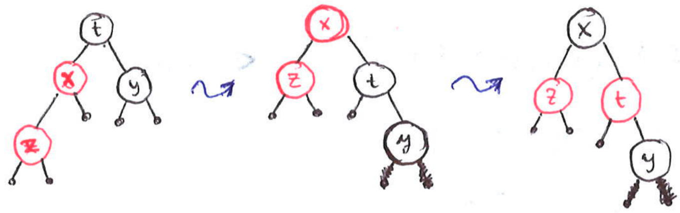
ha t a gyökér
⇒ t marad fekete és a fa fekete magassága eggyel nő

→ ha y fekete

- ha z és x nem azonos oldali gyerekek ⇒ forgatás x körül



- ha z és x azonos oldali gyerekek ⇒ forgatás t körül, majd átszínezünk



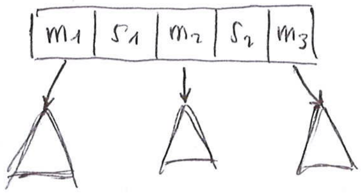
T.: BESZÚR során a lépésszám $O(\log n)$ és legfeljebb 2 forgatás történik.

2-3 FÁK

→ egy nem levél csúcsnak 2 vagy 3 gyereke van

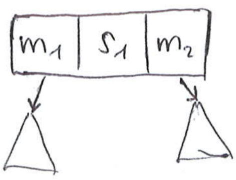
→ tulajdonságok:

1) a tárolt rekordok a fa leveleiben vannak, balról jobbra növekvő sorrendben.
2) ∀ belső csúcsból 2 vagy 3 él megy lefelé → a belső csúcsok 1 illetve 2 kulcsot tartalmaznak



teljesül, hogy $s_1 < s_2$

- m_1 által mutatott részfa ∀ eleme kisebb mint s_1
- m_2 részfájában s_1 a legkisebb kulcs és ∀ kulcs < mint s_2
- m_3 részfájában az s_2 a legkisebb kulcs



- m_1 részfájában ∀ elem < s_1
- m_2 ~~a jobb oldali~~ részfájában s_1 a legkisebb kulcs

3) A fa levelei egyforma távolságra vannak a gyökértől.

T.: ha egy 2-3 fának m szintje van, akkor a levelek száma $\geq 2^{m-1}$

$\Rightarrow m \leq \log_2 n + 1$ ahol n a tárolt rekordok száma

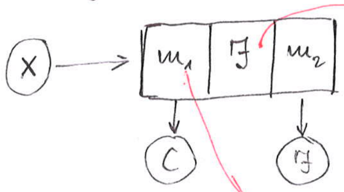
→ műveletek a 2-3 fákban:

1) KERES(s, S): a mutatók mentén haladunk, összehasonlításokat használva (szintenként legfeljebb kettőt)

- a mutatók nem csak megmondják, hogy a keresett elem benne van-e a fában, hanem a pontos helyét is megmondja
- szintenként max. 2 összehasonlítás $\Longrightarrow 2m \leq 2\cdot(\log_2 n + 1) = O(\log n)$
- a korlát ∀ esetben érvényes

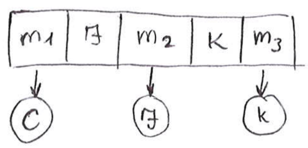
2) BESZÚRÁS és TÖRLÉS

- a BESZÚR(s, S) egy kereséssel indul → ha nincs találat, akkor van igazándiból dolgunk.
- jelölje x a legalsó még nem levél csúcsot, x így néz ki:

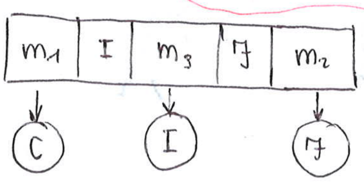


3 eset lehetséges:

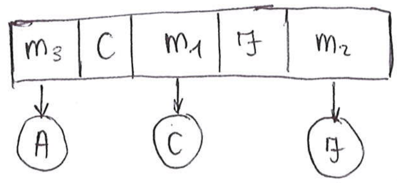
I) ha s = K (vagyis s > x kulcsa)



II) ha pl.: s = I (vagyis s > m_1 és s < x kulcsa)



III) ha pl.: s = A (vagyis s < m_1)



új rekord beillesztése viszonylag egyszerű ha csak 2 fia van x-nek.

más a szituáció ha egy $\boxed{m_1 | s_1 | m_2 | s_2 | m_3}$ szerkezetű csúcsba $[m_1; m_3]$ intervallumba eső (vagyis $s_1 < z < s_2$) értéket akarunk beszúrni.

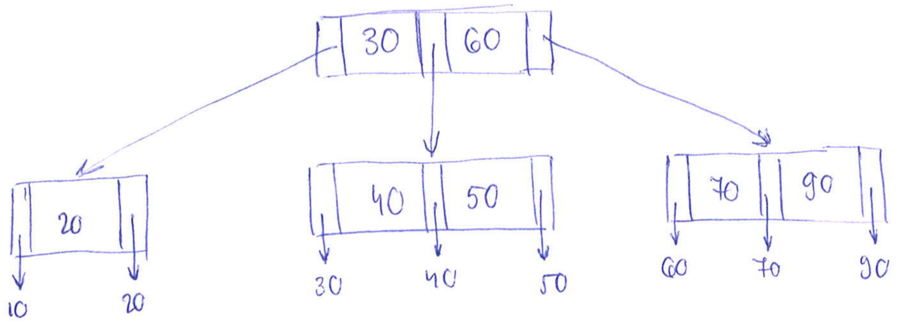
ekkor alkalmazzuk a csúcsvágás módszert. A 3 gyereket birtokló csúcsot felbontjuk 2 db 2 leszármazottal rendelkező csúcsra.

Ekkor a csúcsok ősét bővíteni kell egy mutatóval → a mutató értéke legyen a 2 régi és új mutató középső eleme.

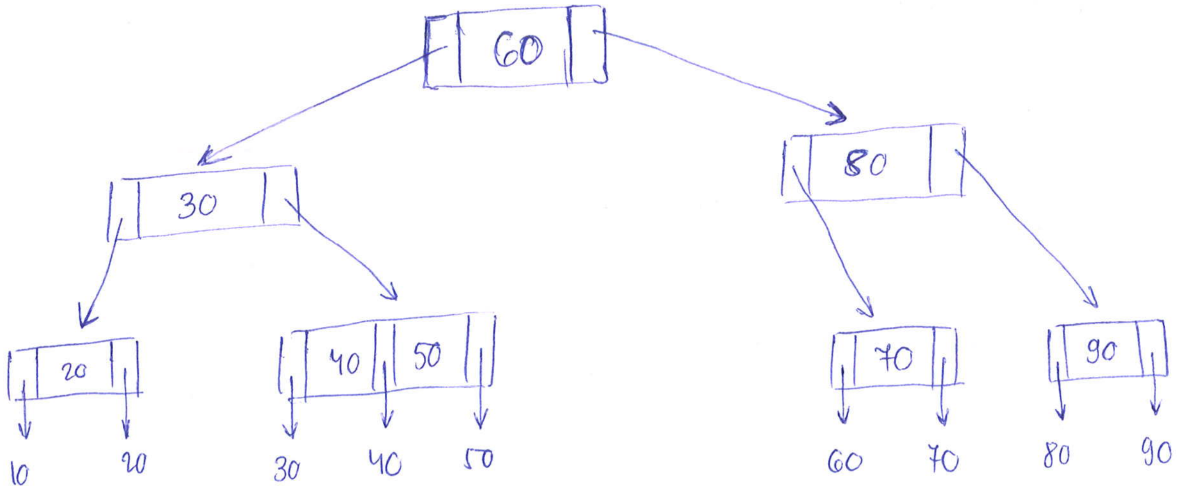
A hatás egészen a ~~csúcs~~ fa tetejéig eljuthat $\Longrightarrow$ a fa szintjeinek száma eggyel nő.

A növekedés a fa tetején következik be $\Longrightarrow$ még mindig igaz az, hogy ∀ gyökérből levélbe vezető út ugyanolyan hosszú.

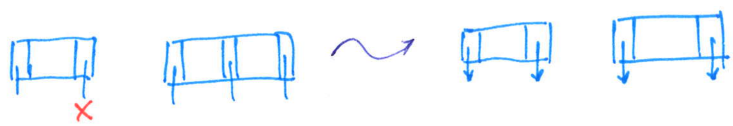
→ Példa beszúrásra:



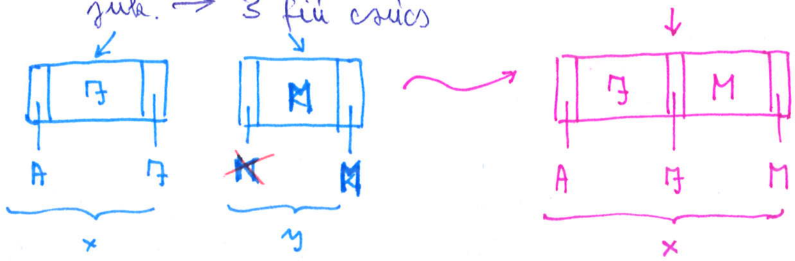
BESZÚR(80)



- a TÖRÖL(r, s) megvalósítása hasonlít a BESZÚR(r, s) eljárás analógiájához
  - → legyen x a legalsó még nem levél csúcs
    - ha x-nek 3 fia van → értelemszerűen töröljük s-et és módosítjuk a mutatókat.
    - ha x-nek 2 fia van → több ~~(művelet)~~ lehetőség
      - (1) ha x valamely szomszédos testvérének van 3 fia
        => a 3 fiú közül áttehetünk egyet x-be, módosítani kell x-et, a testvérét és az apjukat



      - (2) ha x-nek nincs 3 leszármazottat tartalmazó testvére
        => bevezetünk egy új műveletet: csúcsösszevonás
        - x-et és egyik testvérét egyetlen csúccsal helyettesítjük. → 3 fiú csúcs



        - az y csúcsot töröljük
        - a k kulcsértéket ~~(törölni)~~ és a hozzá tartozó mutatót törölni kell x apjából

→ a vágáshoz hasonlóan a törlések is eljuthatnak a fa csúcsához.

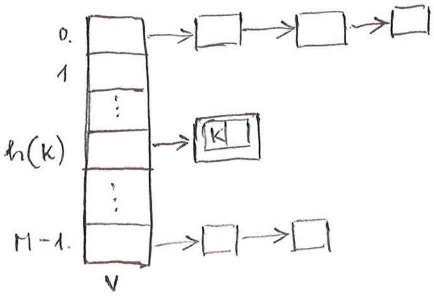
A BESZÚR, KERES, TÖRÖL műveletek mind $O(\log n)$ költségű műveletek.

HASHELÉS

→ h hash függvény, mely a K kulcshoz a $h(K)$ címet/értéket rendeli
→ a $h[K]$ értéke egy egész szám a $[0, M-1]$ tartományon.

→ Vödrös - hashelés

- főként nagy méretű állományok tárolására
- $V[0; M-1]$ vödörkatalógus: a $h()$ függvény értékkészletével indexelt tömb



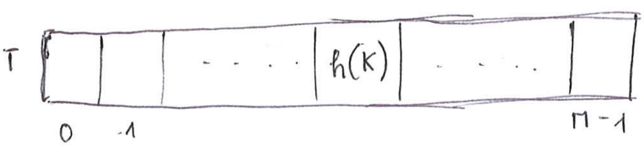
- V elemei mutatók, melyek vödrökre mutatnak.
  ⇒ a vödrökben található lapokon tárolódnak a rekordok, a lapok implementáció szerint lehetnek előre/hátra láncolva is
- $V[i]$ mutatóval kezdődő vödörbe kerülnek a K kulcsú $h(K) = i$ rekordok.

- BESZÚR-ás a vödrös hashbe: 1) kiszámítjuk $h(K)$ értékét
  2) a $V[h(K)]$ vödrön végigiterálva (lineárisan) ellenőrizzük, hogy a rekord nincs-e a vödörben
     2a) ha benne van → nincs teendőnk
     2b) ha nincs benne → beszúrjuk az elemet egy törölt/új lapra

- KERES, TÖRÖL műveletek triviálisak.
- költség: lapelérések száma a lényeges — M db vödör, ℓ db lap
  → 1 vödörbe $\sim \frac{\ell}{M}$ lap kerül — átlagos lánchossz
  → ha ugyan olyan eséllyel fordulunk minden vödörhöz & a vödörkatalógus is a háttértáron van ⟶ átlagosan $1 + \frac{\ell}{M}$ lapművelet a keresés
- fontos tervezési szempont: a vödörszám meghatározása
  → ℓ-t általában ismerjük - vagyis, hogy a rekordjaink hány lapon helyezkednek el
  → célszerű, ha $\frac{\ell}{M} \sim 1$ ⟶ általában 20%-al többet foglalnak (vödörből)

→ Nyitott címzéses hash

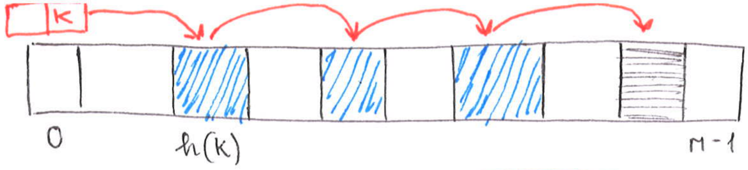
- csak belső memóriás tárolásra használják



→ T tömb, M mérettel
→ rekordok $T[h(K)]$ értéken tárolódnak
→ ha $T[h(K)]$ már foglalt ⇒ fel kell oldanunk az ütközést

- ütközésfeloldás: végigpróbáljuk a $h(K) + h_i(K)$ sorszámú cellákat addig amíg üres cellát nem találunk. $i = \{0, 1 \ldots M-1\}$

⇒ fontos, hogy az összegzést $h(K) + h_i(K) \pmod M$ értendő
$h_i(K)$ – próbasorozat



→ a pirossal jelölt rekord végül a feketével satírozott helyre kerül.

• <u>Lineáris próbálás</u> $\boxed{h_i(K) = -i}$

→ a módszer lényegében annyi, hogy a $h(K)$ cellától visszafelé lépkedünk amíg üres cellát nem találunk.
⇒ $T[0]$ után $T[M-1]$-el folytatjuk

→ pl.: $M = 7$
$h(K) = K \pmod 7$

		9	3	11		
0	1	2	3	4	5	6

⇐ Beszúr 3, 11, 9

	4	9	3	11		
0	1	2	3	4	5	6

⇐ Beszúr 4

→ törlés esetén nem $\underbrace{\text{töröljük a cellát}}_{\text{NULL érték}}$, hanem helyette TÖRÖLT jelzést állítunk be (pl.: *)

→ lineáris próba hátránya: elsődleges csomósodás
R' rekord beszúrása során elérjük az R rekord keresési útját, akkor azt rendszerint végig is kell járnunk.

• <u>Hashelés álvéletlen próbával</u>

→ $0, h_1(K), \dots h_{M-1}(K)$ próbasorozat a $0 \dots M-1$ számok K-tól fgtlen álvéletlen permutációja

→ kvadratikus maradék próba: $M = 4k+3$ alakú prímszám
- a próbasorozat ekkor: $0, 1^2, (-1)^2, 2^2, (-2)^2 \dots \left(\frac{M-1}{2}\right)^2, -\left(\frac{M-1}{2}\right)^2$

→ T.: ha M egy $4k+3$ alakú prímszám, akkor ∄ olyan u egész, amelyre $u^2 \equiv -1 \pmod M$.

→ hátrány: ha $h(K) = h(L)$ akkor K és L próbasorozata is ugyan az
⇒ másodlagos csomósodás.

→ <u>Kettős hashelés</u>

→ h mellett egy másik h' függvényt is használunk

→ $h'(K)$ értékeitől elvárjuk, hogy relatív prímek legyenek az M tábla-mérethez.

→ a próbasorozat: $h_i(K) = -i \cdot h'(K)$

→ a módszer sajátossága, hogy $h(K) = h(K')$ esetén a próbasorozat különbözni fog.

→ nincs sem elsődleges, sem másodlagos csomósodás

→ ha M prím ⇒ $h'(K) = K \pmod{M-1} + 1$

→ <u>Hash függvények</u>

→ gyorsan számítható & kevés ütközés

• <u>osztómódszer</u> $h(K) = K \pmod M$

→ általában M-et prímnek választjuk, úgy hogy M nem osztja $r^k \pm a$-t ahol r a karakterkészlet elemszáma
a, k → „kicsi" egészek

• szorzómódszer

→ β egy rögzített valós szám $h(k) = \lfloor M \cdot \{\beta k\} \rfloor$

→ $\{\beta k\}$ jelöli a $\beta \cdot k$ szám valós törtrészét

• lényegében $\{\beta \cdot k\}$ számítása során a k kulcsot belégyük a $[0,1[$ intervallumba majd az eredményt felskálázzuk a címtartományba

• β meghatározása: $M = 2^t$ $w = 2^{32}$ (a gépünk szókapacitása)

A = egy w-hoz relatív prím egész

$\beta = \frac{A}{w}$ esetén $h(k)$ jól számolható.

GRÁFALGORITMUSOK

MÉLYSÉGI BEJÁRÁS

→ addig megyünk előre amíg tudunk, ha megakadunk akkor az eddig megtett út utolsó előtti pontjához lépünk vissza és új továbblépési lehetőséget keresünk.

→ éllistás megoldás esetén a lépésszám $O(n+e)$

→ mélységi számozás: a gráf v csúcsához azt a számot rendeli amely megadja, hogy a mélységi bejárás során az algoritmus hányadikként látogatja meg a csúcsot először.

→ befejezési számozás: a gráf v csúcsához azt az értéket rendeli, amely megadja, hogy a v csúcs esetében hányadikként fejeztük be (vagyis a csúcsból kiinduló összes élet bejártuk már) a mélységi bejárást.

→ élek osztályozása mélységi bejárás során:

• faél: a $v \rightarrow w$ él faél, ha az eljárás során w-t először látogatjuk meg.

• mélységi feszítő erdő, feszítőfa: legyen T a kiinduló G gráf csúcshalmazából épített olyan gráf, amely csak faéleket tartalmaz. Ekkor T a G gráf egy feszítőerdője, ha T egy komponensből áll akkor T mélységi feszítőfa.

→ mélységi bejárás után G élei a következő típusokra oszthatóak: G $x \rightsquigarrow y$ éle

- faél, ha $x \rightsquigarrow y$ éle T-nek
- előreél, ha $x \rightsquigarrow$ nem faél, de y leszármazottja x-nek és $x \neq y$
- visszaél ha x leszármazottja y-nak (pl.: hurokél)
- keresztél ha x és y nem leszármazottai egymásnak.

→ az egyes éltípusok hatékony felismerése:

$x \to y$ egy	ha az él vizsgálatakor
faél	~~ha~~ mszám[y] = 0
visszaél	mszám[x] ≥ mszám[y] és bszám[y] = 0
előreél	mszám[x] < mszám[y]
keresztél	mszám[y] < mszám[x] és bszám[y] > 0

→ az élek osztályozása nem módosítja az $O(n+e)$ lépésszámot.

IRÁNYÍTOTT KÖRMENTES GRÁFOK

→ DAG: G irányított gráf DAG, ha nem tartalmaz irányított kört.

→ irányított gráf körmentességének eldöntése: ha G-nek van olyan mélységi bejárása amely tartalmaz visszaélt, akkor G-ben van irányított kör ⟹ G nem DAG.

⟹ ha G-nek van olyan mélységi bejárása, amelyben nincs visszaél, akkor G egy DAG.

- tehát egy tetszőleges irányított gráfról $O(n+e)$ időben eldönthető hogy DAG-e.

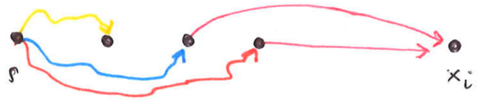
→ topologikus rendezés: G egy topologikus rendezése a csúcsoknak egy olyan $v_1 \cdots v_n$ sorrendje, amelyben $x \to y \in E$ esetben x előbb van mint y, vagyis $x = v_i$, $y = v_j$ esetén $i < j$.

→ T.: egy gráfnak ⟺ ∃ topol. rendezése, ha G DAG.

→ topologikus rendezés mélységi bejárással: végezzük el a gráf mélységi bejárását és írjuk ki a csúcsokat befejezési szám szerint ~~[illegible]~~ csökkenő sorrendben.

LEGRÖVIDEBB UTAK DAG-BAN

- legyen G topologikus rendezése $x_1, x_2 \cdots x_n$
- $s = x_1$ mert a kisebb-től a nagyobb sorszámú csúcsok felé mehet él.
- $d(s, x_i) = \min\limits_{(x_j, x_i) \in E} \{ d(s, x_j) + c(x_j, x_i) \}$



- ezt sorra elvégezzük ∀ i-re.
- lépésszám: $\text{DFS} + \left(\sum_{i=1}^{n} \text{min. keresés} \{ d_{be}(x_i) \} \right) = O(n+e)$
- fordított éllista megkapható $O(n+e)$ időben!
- a maximumkeresés is ugyan ilyen költséggel megoldható DAG-ban! (min helyett max)

MINIMÁLIS KÖLTSÉGŰ FESZÍTŐFÁK

→ fontos, hogy mostantól irányítatlan gráfokról fogunk beszélni

→ minimális súlyú feszítőfa: G gráf összefüggő és élei súlyozottak. A gráf egy $F = (V, E')$ részgráfja a gráf egy feszítőfája. F minimális költségű ha a benne szereplő élek súlyainak összege minimális, G összes feszítőfája közül.

A PIROS - KÉK ALGORITMUS

→ a gráf éleinek 3 színe lehet: piros, kék, színtelen

→ takaros színezés: ha van G-nek olyan minimális költségű feszítőfája, ami az összes kék élet tartalmazza, de egyetlen pirosat sem.

→ a színezés során 2 szabályt használunk:
- kék szabály: válasszunk ki egy olyan $\emptyset \neq X \subset V$ halmazt, amiből nem vezet ki kék él. Ezután egy legkisebb súlyú színezetlen élt fessünk kékre, amely kilép X-ből.
- piros szabály: válasszunk G-ben egy kört, amelyben nincs piros él. A kör legnagyobb súlyú színtelen élét fessük pirosra.

→ algo: ~~a~~ G ∀ éle színtelen ⟶ olyan sorrendben és ott használjuk a két szabályt ahol akarjuk, amíg csak lehetséges.

→ Prim - algoritmus: egy s csúcsból indulva a kék szabály alkalmazásával bővítjük az s-et tartalmazó kék fát.
- U = az aktuális kék fa csúcsait tartalmazó ~~va~~ halmaz
- kiválasztjuk azon élek közül a minimálisat amelyik egyik végpontja U-beli, a másik pedig V\U-beli
  → a „külső" csúcsot U-ba tesszük, az élet pedig kékre színezzük (tehát a fába tesszük)
- naiv implementáció:
  → minden V\U-beli csúcshoz tároljuk, hogy milyen távol van az U halmaztól
    - pontosabban a csúcsból az U-ba futó él minimális súlyát.

  → tartsuk nyilván még ezen élek U-beli ~~(nagy)~~ végpontját

$$\text{KÖZEL}[i] = \begin{cases} * & i \in U \\ \text{az } i\text{-hez legközelebb levő } U\text{-beli csúcs} & i \in V \setminus U \end{cases}$$

$$\text{MINSÚLY}[i] = \begin{cases} * & \text{ha } i \in U \\ C[i,j] & \text{ha KÖZEL}[i] = j \neq * \end{cases}$$

→ a következő kék él az (KÖZEL[i], i) élek közül kerül ki
⟹ kékes élek

→ a következő kék él kiválasztása: megkeressük MINSÚLY[] minimumát – ez legyen a k csúcsnál.

=> a (KÖZEL[k], k) élet kékre színezzük (bevesszük F-be) és a k csúcsot bevesszük U-ba

=> KÖZEL[k] = MINSÚLY[k] = *

→ ezután frissítenünk kell a tömböket

if KÖZEL[i] ≠ * and C[k,i] < MINSÚLY[i]

KÖZEL[i] = k

MINSÚLY[i] = C[k,i]

=> az algoritmus lépésszáma: $O(n^2)$

- kupacos - éllistás implementáció

→ ha élek száma < n^2 => használjunk éllistát

→ építsünk és tartsunk fenn kupacot U és V\U közti élekből (élsúly szerint rendezve)

→ néhány MINTÖR után találunk olyan (u, v) élet melynek egyik végpontja U-ban másik V\U-ban van.

→ ezen él V\U csúcsából kilépő éleket hozzávesszük a kupachoz BESZÚR műveletekkel. de csak azokat amelyek V\U-ba lépnek

=> az algoritmus költsége: $O(e \cdot \log e)$

→ KRUSKAL ALGORITMUS

→ több helyen kezdjük el növeszteni a kék fát → összekapcsoljuk a még diszjunkt darabokat.

→ kezdetben n db 1 csúcsból álló fánk van.

=> minden lépésben a legkisebb súlyú, kört nem okozó él hozzávételével a kék komponensek száma eggyel csökken.

→ az összes él = H

töröljük H-ból a minimális súlyú (v, w) élet

Ha $F \cup \{(v,w)\}$ nem alkot kört akkor ← v, w pontok különböző kék fákban vannak.

$F = F \cup \{(v,w)\}$

→ UNIÓ - HOLVAN adatszerkezet

Adott egy n elemű S halmaz, és ennek ~~egy~~ bizonyos $U_1, U_2 \dots U_m$ részhalmazai, melyekre

1) $U_i \cap U_j = \emptyset \quad (i \neq j)$

2) $U_1 \cup U_2 \dots U_M = S$ (vagyis U_i halmazok S egy partícióját adják)

Műveletek:

- $\text{UNIÓ}(U_i, U_j) = (\{U_1, \dots U_m\} \cup \{U_i \cup U_j\}) \setminus \{U_i, U_j\}$

- HOLVAN (v) eredménye annak az U_i részhalmaznak a neve, amely tartalmazza v-t ($v \in U_i$)

UNIÓ - HOLVAN használata Kruskal - algoban:

- $S = V(G)$, U_i-k pedig a kék fa komponenseit ~~tartalmazó~~ jelentő csúcsok
- (v, w) nem okoz kört, ha végpontjaik külön komponensben vannak, vagyis HOLVAN $(v) \neq$ HOLVAN (w) ⟸ körmentesség eldöntése 2 HOLVAN kérdéssel
- miután a (v, w) élet bevettük F-be, egyesítenünk kell a v-t és a w-t tartalmazó két fát → UNIÓ(U_i, U_j) ahol $v \in U_i$ és $w \in U_j$

⟹ a KRUSKAL - algoritmus költsége: $O(e \cdot \log e)$

ELDÖNTÉSI PROBLÉMÁK, BONYOLULTSÁGELMÉLET

Egy eldöntési problémához tartozó $\mathcal{L}$ nyelv azoknak a bemeneteknek a halmaza, amelyekre a válasz IGEN.
Egy X eldöntési problémára és x bemenet esetén $x \in X$ jelöli, hogy az x bemenetre a válasz IGEN.

Polinom időben eldönthető problémák

Jelölje P azoknak az eldöntési problémáknak a halmazát, amelyeknek van olyan $\mathcal{A}$ algoritmusuk, amely $\forall x$ bemenetre helyesen megválaszolja a kérdést és az algo. lépésszáma polinomiális → $O(|x|^k)$

Hatékony tanúsítvány: X eldöntési problémához $\exists$ hatékony tanúsítvány, ha van olyan $\mathcal{T}$ algo, amely bemenete (x, t) párokból áll, ahol x az X el. prob. egy lehetséges bemenete és

→ ha $x \in X$, akkor van olyan t aminek hossza $|t| = O(|x|^c)$ és $\mathcal{T}(x, t) =$ IGEN

→ ha $x \notin X$, akkor nincs olyan t aminek hossza $|t| = O(|x|^c)$ és $\mathcal{T}(x, t) =$ IGEN

→ $\mathcal{T}$ algo. polinomiális → $O((|x| + |t|)^k)$

kicsit másképp: x bemenet esetén az eldöntési problémára a válasz IGEN, akkor erre van olyan polinom hosszú tanú (t), amiből $\mathcal{T}$-vel polinom időben ellenőrizhető, hogy valóban IGEN.

Ha a válasz NEM ⟹ nincs ilyen rövid válasz!

NP-beli problémák: Jelölje NP azoknak az eldöntési problémák halmazát, amelyekre van hatékony tanú.

coNP-beli problémák: Jelölje coNP az NP-beli problémák komplementereiből álló halmazt ~~[illegible]~~ azaz $X \in coNP \iff \overline{X} \in NP$

Pl.: $\overline{\text{ÖSSZETETT}}$ = PRÍM

⟹ a coNP-beli problémák esetén a NEM válaszra van polinom hosszú, polinom időben ellenőrizhető bizonyíték

T.: $P \subseteq NP$ és $P \subseteq coNP$ vagyis $P \subseteq NP \cap coNP$

Karp-redukció

→ egy X probléma nem lényegesen nehezebb Y-nál, ha Y felhasználásával meg lehet oldani X-et

→ Definíció: Az X Karp-redukciója az Y problémára, egy olyan polinom időben számolható f függvény, amely X minden lehetséges bemenetére hozzárendeli Y egy lehetséges bemenetét úgy, hogy

$$x \in X \iff f(x) \in Y$$

→ Jelölés: $X < Y$ [X probléma visszavezethető Y problémára]

=> Y egy általánosabb, míg X annak egy specializált ~~(ba)~~ problémája

→ tehát van algoritmusunk Y eldöntésére

$x \in X$ esetén kiszámítjuk $f(x)$-et, és eldöntjük $f(x) \in Y$?

→ ha Y könnyű és X nem lényegesebben nehezebb nála => X is könnyű

→ Tulajdonságok:

1) Ha $X < Y$ és $Y \in P$ akkor $X \in P$.
2) Ha $X < Y$ és $Y \in NP$ akkor $X \in NP$.
3) Ha $X < Y$ és $Y \in coNP$ akkor $X \in coNP$.
4) Ha $X < Y$ ~~és~~ akkor $\overline{X} < \overline{Y}$.
5) Ha $X < Y$ és $Y \in NP \cap coNP$ akkor $X \in NP \cap coNP$.
6) Ha $X < Y$ és $Y < Z$ akkor $X < Z$.

NP-teljes problémák

NP-teljes problémák: Az X eldöntési probléma NP-nehéz ha minden $X' \in NP$ probléma esetén $\exists\, X' < X$ Karp-redukció. Az X eldöntési probléma NP-teljes, ha $X \in NP$ és X NP nehéz.

=> egy NP-teljes probléma tehát legalább olyan nehéz, mint bármely más NP-beli probléma.

T.: Ha az X probléma NP-teljes, $Y \in NP$ és $X < Y$ => Y is NP-teljes.

1) 3-SZÍN probléma

- $\in NP$: t → egy színezés megadása G-ben (azaz egy $f: V(G) \to \{p, k, z\}$ függvény)

  T → ellenőrzi, hogy f tényleg egy 3-színezése G-nek.

2) MAXFTLEN

Bemenet: G gráf, $k \in \mathbb{Z}^+$

Kérdés: Van-e G-ben k elemű független csúcshalmaz?

a csúcshalmaz pontjai között nem fut él

- $\in NP$: t egy k-elemű $S \subseteq V(G)$ független csúcshalmaz.
- megadunk egy 3-SZÍN < MAXFTLEN karp-redukciót

  => lásd diasor

3) MAXKLIKK

Bemenet: G gráf, $k \in \mathbb{Z}^+$

Kérdés: Van-e G-ben k méretű teljes gráf?

- $\in NP$: t egy k-elemű $S \subseteq V(G)$ teljes részgráf.
- megadunk egy MAXFTLEN < MAXKLIKK Karp-redukciót:

$f(G, k) = (\bar{G}, k)$ → független ponthalmaz komplementere teljes gráf.

4) RÉSZGRÁFIZO

5) GRÁFIZO

6) HAMILTON KÖR

7) HAMILTON-ÚT

8) RÉSZHALMAZ ÖSSZEG

9) PARTÍCIÓ

Bemenet: $(s_1, s_2 \dots s_m)$

Kérdés: Van-e olyan $I \subseteq \{1, \dots, m\}$ melyre $\sum_{i \in I} s_i = \frac{1}{2} \sum_{i=1}^{m} s_i$

EUKLIDESZI UTAZÓ ÜGYNÖK PROBLÉMA

Az n pontú K_n teljes gráf élein adott a d nemnegatív értékű súlyfüggvény. Erre teljesül a háromszög-egyenlőtlenség: tetszőleges u, v, w csúcsokra teljesül, hogy $d(u, w) \leq d(u, v) + d(v, w)$.

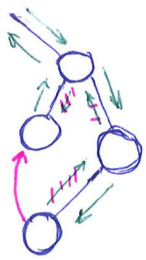
A feladat egy minimális összsúlyú Hamilton-kör keresése.

Megoldás: keresünk egy minimális súlyú feszítőfát G-ben (pl.: Prim vagy Kruskal algoritmussal)

⇒ megkettőzzük az éleit és körbejárjuk egy Euler körsétával

A minimális fesz. fa súlya legyen s ⇒ Euler-séta hossza $2s$.

A bejárás során rövidíthetünk is - levágjuk az utakat. Pl.:



A módszer legrosszabb esetben is legfeljebb kétszer akkora utat ad, mint az optimális Hamilton-kör.

LÁDAPAKOLÁS

Adottak az $s_1 \dots s_m$ súlyok, $0 \leq s_i \leq 1$.

Feladat: a súlyok elhelyezése minél kevesebb 1 összkapacitású ládába.

First-fit módszer: üres ládák, megszámozva $1 \dots m$ egészekkel. $s_1 \dots s_{i-1}$ súlyokat már elhelyeztük → s_i kerüljön az első olyan ládába amibe még belefér.

T.: jelölje a Ládapakolás probléma egy I inputjára OPT(I) az optimális, FF(I) pedig az FF-módszer által eredményezett ládaszámot. A probléma tetszőleges I inputjára teljesül, hogy $FF(I) \leq 2 \cdot OPT(I)$.

First-fit Decreasing módszer (FFD): rendezzük a súlyokat csökkenő sorrendbe, utána alkalmazzuk az FF módszert.

T.: tetszőleges I inputra $FFD(I) \leq \frac{11}{9} OPT(I) + 4$ és tetsz. nagy I inputok vannak melyre $FFD(I) \geq \frac{11}{9} OPT(I)$.