Algoritmuselmélet

Függvények nagyságrendje, elágazás és korlátozás, dinamikus programozás

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

1. előadás

Algoritmus fogalma

- Egyelőre nem definiáljuk rendesen az algoritmus fogalmát.
- Eljárás, recept, módszer.
- Jól meghatározott lépések egymásutánja, amelyek már elég pontosan, egyértelműen megfogalmazottak ahhoz, hogy gépiesen végrehajthatók legyenek.

A szó eredete

Al Khvarizmi (Mohamed ibn Músza) bagdadi matematikus a IX. században könyvet írt az egészekkel való alapműveletek végzéséről.

$$\text{algoritmus} \leftrightarrow \text{számítógép program}$$

$$\text{valós probléma} \Longrightarrow \text{absztrakt modell} \Longrightarrow \text{algoritmus} \Longrightarrow \text{program}$$

Cél: feladatokra hatékony eljárás kidolgozása
Hatékony $\Longrightarrow$ gyors, kevés memória, kevés tárhely

Milyen hatékony egy algoritmus?

- Legtöbbször csak a lépésszám nagyságrendje érdekes. Hogyan függ a lépésszám az input méretétől?
- Az input méretét legtöbbször n-nel jelöljük.
- A lépésszám ennek egy f függvénye, azaz ha n méretű az input, akkor az algoritmus $f(n)$ lépést végez.
- Igazából az f függvény az érdekes.
- $100n$ vagy $101n$, általában mindegy
- n^2 vagy n^3 már sokszor nagy különbség, de néha mindegy
- n^2 vagy 2^n már mindig nagy különbség

Például

Kérdés

Egy 10^{10} művelet/mp sebességű számítógép mennyi ideig dolgozik, ha $f(n)$ műveletet kell végrehajtani n méretű bemenetre?

n	$f(n)$ n	n^2	$\log_{10} n$	2^n	$n!$
10	10^{-9}	10^{-8}	10^{-10}	$1{,}02 \cdot 10^{-7}$	$3{,}6 \cdot 10^{-4}$
10^2	10^{-8}	10^{-6}	$2 \cdot 10^{-10}$	$4 \cdot 10^{12}$ év	$2.9{,}\cdot 10^{140}$ év
10^6	10^{-4}	100	$6 \cdot 10^{-10}$	$3{,}1 \cdot 10^{301.012}$ év	$2{,}6 \cdot 10^{5.565.691}$ év
10^9	0,1	3,1 év	$9 \cdot 10^{-9}$	sok év	sok év

Függvények nagyságrendje

Definíció

Ha $f(n)$ és $g(n)$ az $\mathbb{R}^+$ egy részhalmazán értelmezett, valós értékeket felvevő függvények, akkor $f = O(g)$ jelöli azt a tényt, hogy vannak olyan $c, n_0 > 0$ állandók, hogy $|f(n)| \leq c|g(n)|$ teljesül, ha $n \geq n_0$.

Példák

- $100n + 300 = O(n)$
  Biz: $100n + 300 \leq 100n + n \leq 101n \leq cn$, ha $n \geq 300,\ c = 101$
- $5n^2 + 3n = O(n^2)$
  Biz: $5n^2 + 3n \leq 5n^2 + 3n^2 \leq 8n^2 \leq cn^2$, ha $n \geq 100,\ c = 8$
- $n^4 + 5n^3 = O(n^5)$
  Biz: $n^4 + 5n^3 \leq 6n^4 \leq n^5 \leq cn^5$, ha $n \geq 6,\ c = 1$

Példák

- $n^{1000} = O(2^n)$
  Biz: Teljes indukcióval, legyen $c = 1, n_0 = 10^6$.
  $n = 10^6$-re igaz, mert $10^{6000} \leq (2^4)^{6000} \leq 2^{10^6}$.
  Tegyük fel, hogy k-ra igaz.
  Felhasználjuk, hogy ha $k \geq 10^6$ akkor
  $\binom{1000}{i} \leq 1000^i = 1000 \cdot 1000^{i-1} \leq 1000^{i-1} \cdot 1000^{i-1} =$
  $= (10^6)^{i-1} \leq k^{i-1}$.
  $(k+1)^{1000} = k^{1000} + \cdots + \binom{1000}{i} k^{1000-i} + \cdots \leq k^{1000} + \cdots +$
  $k^{i-1}k^{1000-i} + \cdots \leq k^{1000} + 1000k^{999} \leq 2 \cdot k^{1000} \leq 2 \cdot 2^k = 2^{k+1}$,
  ha $k \geq 10^6$.
- $\log_2^{1000}(n) = O(n)$
  Biz: Mivel a logaritmus függvény monoton nő, vehetjük a fentiek logaritmusát.
- $2^n = O(n!)$
- $n! = O(n^n)$

Példák

Igaz-e, hogy $n^2 = O(n)$?

Nem.
Biz: Indirekt, tegyük fel, hogy létezik olyan c, n_0, hogy $n^2 \leq cn$ teljesül minden $n \geq n_0$ esetén.
Ekkor $n \leq c$ teljesül minden $n \geq n_0$ esetén, ami nyilván nem igaz, ha $n > c$.

Függvények nagyságrendje

Definíció

Ha $f(n)$ és $g(n)$ az $\mathbb{R}^+$ egy részhalmazán értelmezett, valós értékeket felvevő függvények, akkor $f = \Omega(g)$ jelöli azt a tényt, hogy vannak olyan $c, n_0 > 0$ állandók, hogy $|f(n)| \geq c|g(n)|$ teljesül, ha $n \geq n_0$.

Például:

- $100n - 300 = \Omega(n)$, hiszen $n > 300,\ c = 99$-re teljesülnek a feltételek
- $5n^2 - 3n = \Omega(n^2)$
- $n^4 - 5n^3 = \Omega(n^4)$
- $2^n = \Omega(n^{1000})$

Függvények nagyságrendje

Definíció

Ha $f = O(g)$ és $f = \Omega(g)$ is teljesül, akkor $f = \Theta(g)$.

Például:

- $100n - 300 = \Theta(n)$
- $5n^2 - 3n = \Theta(n^2)$
- $n^4 - 5n^3 = \Theta(n^4)$
- $1000 \cdot 2^n = \Theta(2^n)$

Függvények nagyságrendje

Definíció

Legyenek $f(n)$ és $g(n)$ a pozitív egészeken értelmezett, valós értékű függvények. Ekkor az $f = o(g)$ jelöléssel rövidítjük azt, hogy

$$\frac{f(n)}{g(n)} \to 0, \text{ ha } n \to \infty.$$

Például:

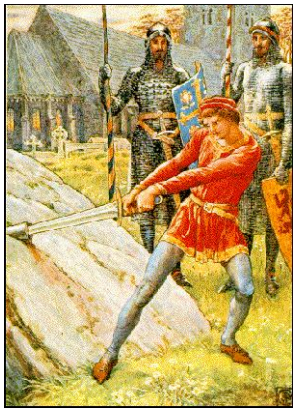
- $100n + 300 = o(n^2)$, hiszen $\frac{100n+300}{n^2} \to 0$ ha $n \to \infty$
- $5n^2 + 3n = o(n^3)$
- $n^4 + 5n^3 = o(n^4 \log_2 n)$
- $n^{1000} = o(2^n)$

Algoritmikus problémák megoldása

Algoritmikus probléma $\xrightarrow{\mathcal{A}}$ modell $\xrightarrow{\mathcal{B}}$ program

- $\mathcal{A}$: pontosítás, egyszerűsítés, absztrakció, lényegtelen elemek kiszűrése, a lényeg kihámozása
- Modell: sokféle lehet, elég tág, de elég egyszerű, formalizált, pontos
- $\mathcal{B}$: hatékony algoritmus, bemenő adatok $\rightarrow$ eredmény, érdemes foglalkozni a kapott algoritmus *elemzésével, értékelésével*, megvizsgálva, hogy a módszer mennyire hatékony

Arthur király civilizációs törekvései

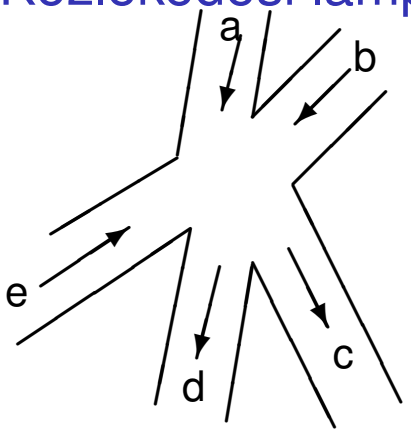


Arthur király fényes udvarában 150 lovag és 150 udvarhölgy él. A király, aki közismert civilizációs erőfeszítéseiről, elhatározza, hogy megházasítja jó lovagjait és szép udvarhölgyeit. Mindezt persze emberségesen szeretné tenni. Csak olyan párok egybekelését akarja, amelyek tagjai kölcsönösen vonzalmat éreznek egymás iránt. Hogyan fogjon hozzá?

Természetesen pártfogójához, a nagyhatalmú varázslóhoz, Merlinhez fordul. Merlin rögvest felismeri, hogy itt is bináris szimmetrikus viszonyok ábrázolásáról van szó.

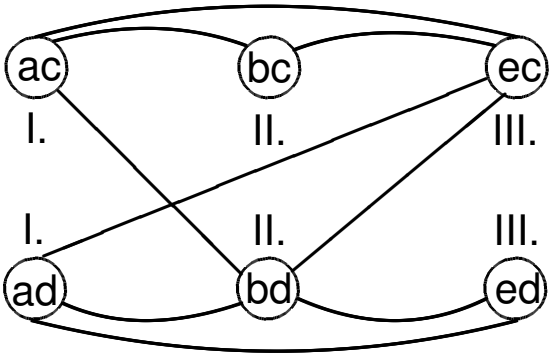
Nagy darab pergament vesz elő, és nekilát egy páros gráfot rajzolni. A királyi parancs teljesítéséhez Merlinnek élek egy olyan rendszerét kell kiválasztania a gráf éleiből, hogy a kiválasztott élek közül a gráf minden pontjához pontosan egy csatlakozzon. A kiválasztott élek felelnek meg a tervezett házasságoknak. A gráfelmélet nyelvén teljes párosítást kell keresnie.

Közlekedési lámpák ütemezése



lámpák: ac, ad, bc, bd, ec és ed
állapot: lámpák $\to \{P, Z\}$

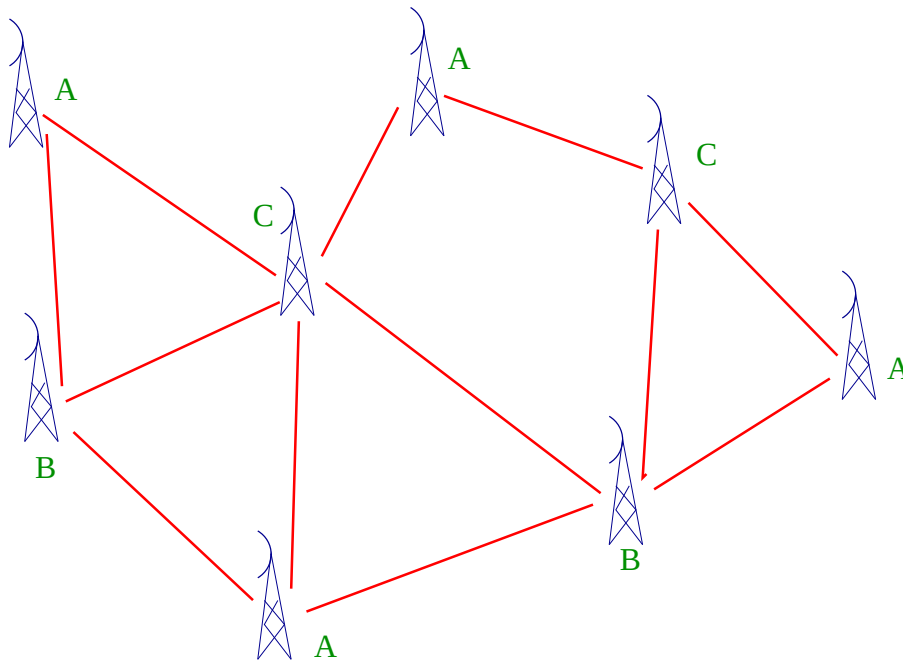
Feladat: Mennyi a minimális számú állapot, ami biztonságos és nem okoz örök dugót?



Gráfelméleti nyelven: Mennyi G kromatikus száma?

Mobiltelefon-átjátszók frekvencia kiosztása

Egy adott átjátszóhoz egy adott frenkvenciát rendelnek.
Egy telefon a közelben levő átjátszók közül választ.
„Közel levő” átjátszók frekvenciája különbözzön.



Elágazás és korlátozás

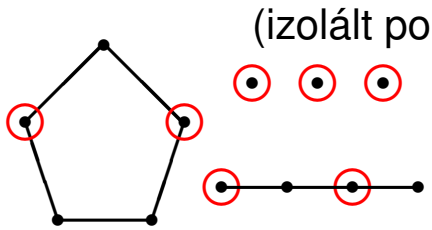
Legtöbbször van c^n-es algoritmus, de nem mindegy mekkora c.

Bontsunk esetekre, azokat alesetekre, ... $\Longrightarrow$ fa
Értékeljük az eseteket $\Longrightarrow$ bizonyos irányokba nem kell továbbmenni.
$\Longrightarrow$ (korlátozó heurisztika)
Pl. sakkállások

Feladat: Keressünk maximális méretű független ponthalmazt egy adott G gráfban.
Nyilvánvaló módszer:
Minden részhalmazt végignézünk $\Longrightarrow$ $O(2^n)$ lépés

Jobb algoritmus

Észrevétel: *Ha G-ben minden pont foka legfeljebb kettő, akkor a feladat lineáris időben megoldható: G izolált pontok, utak és körök diszjunkt uniója.* $\Longrightarrow$ *komponensenként minden „második" pontot bevesszük a halmazba.*



MF(G)
1. Ha G-ben minden pont foka ≤ 2, akkor MF(G) az előbbi eljárás által adott maximális független halmaz, és a munkát befejeztük.
2. Legyen $x \in G$, $fok(x) \geq 3$.
   $S_1 := \text{MF}(G \setminus \{x\})$
   $S_2 := \{x\} \cup \text{MF}(G \setminus \{x \text{ és szomszédai}\})$.
3. Legyen S az S_1 és S_2 közül a nagyobb méretű, illetve akármelyik, ha $|S_1| = |S_2|$.
4. MF(G) $:= S$.

Legyen $T(n)$ az MF(G)-n ($|\ V(G)\ | \leq n$) belüli MF hívások maximális száma, beleértve MF(G)-t magát is.

Tétel

Van olyan c állandó, hogy $T(n) \leq c\gamma^n$, tetszőleges n természetes számra, ahol γ a $\gamma^4 - \gamma^3 - 1 = 0$ egyenlet pozitív gyöke ($\gamma \approx 1,381$).

Bizonyítás.

Legyen $t(n) := T(n) + 1$.
$T(n) \leq T(n-1) + T(n-4) + 1$, ha $n > 4$. $\Longrightarrow$
$t(n) \leq t(n-1) + t(n-4)$, ha $n > 4$.
Indukcióval: $t(n) \leq c\gamma^n$ igaz$n < 5$-re elég nagy c-vel $\surd$
$\Longrightarrow$ Ezután, ha $n \geq 5$, indukciós feltevésből:

$$\begin{aligned} t(n) &\leq t(n-1) + t(n-4) \leq c\gamma^{n-1} + c\gamma^{n-4} = \\ &= c\gamma^{n-4}(\gamma^3 + 1) = c\gamma^{n-4}\gamma^4 = c\gamma^n. \end{aligned}$$

□

Összköltség: $O(n^d T(n)) = O(n^d \gamma^n) = O(1,381^n)$.

3-színezés keresése

Feladat: Adott G, keressünk egy 3-színezést.

Minden lehetséges színezést végignézünk $\Longrightarrow$ $O(3^n)$ lépés

Ötlet: Bizonyos csúcsokat kiszínezünk pirosra, a többiről polinom időben el tudjuk dönteni, hogy kiszínezhetők-e kékkel és sárgával.

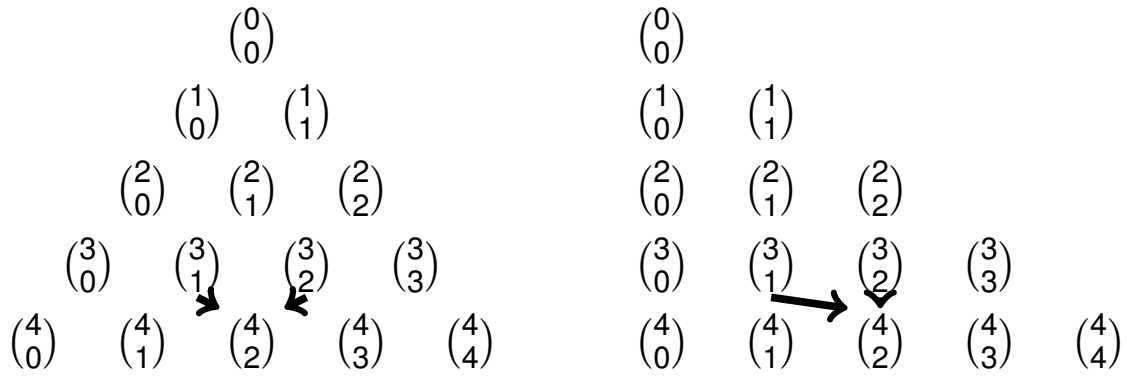
Összköltség: $O(2^n n^c)$.

Dinamikus programozás

Optimum meghatározása kisebb részfeladatok optimumainak felhasználásával.

Általában egy táblázat kitöltése, az új elemeket a korábban kitöltött elemekből számoljuk.

Binomiális együtthatók kiszámítása



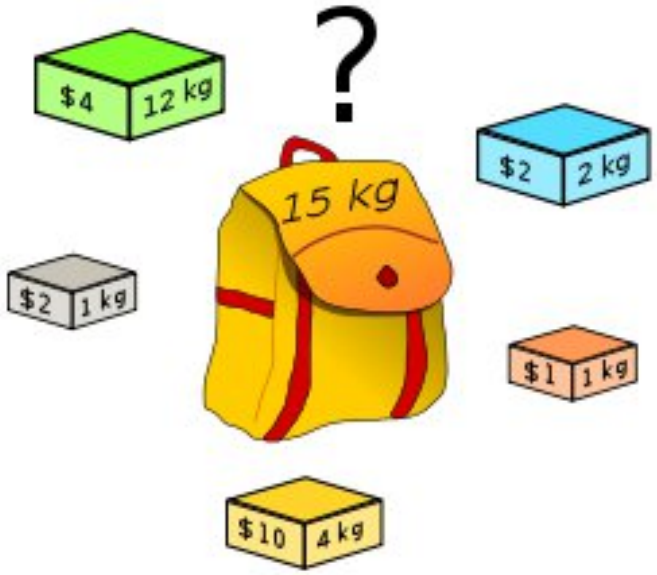
Tétel

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

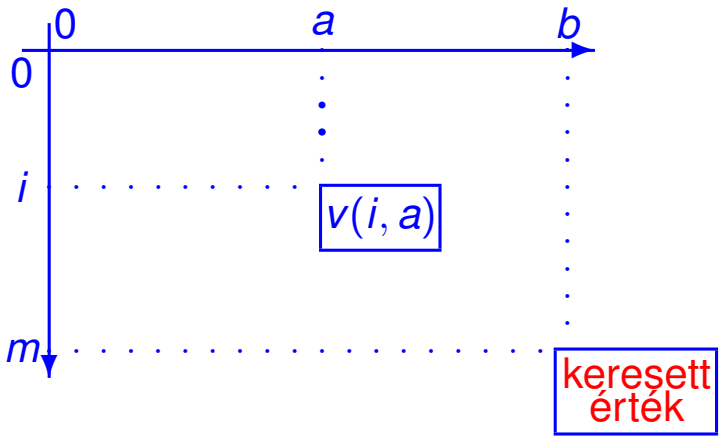
Hátizsák probléma

Probléma

Adottak az $s_1, \ldots, s_m$ súlyok, a b súlykorlát és a $v_1, \ldots, v_m$ értékek. (Minden érték pozitív egész.) Melyik az az $I \subseteq \{1, \ldots, m\}$ részhalmaz, melyre teljesül, hogy $\sum_{i \in I} s_i \leq b$ és $\sum_{i \in I} v_i$ maximális?



Először kisebb problémára oldjuk meg: $v(i, a)$ a maximális elérhető érték az $s_1, \ldots, s_i$ súlyokkal, $v_1, \ldots, v_i$ értékekkel és a súlykorláttal megadott feladatra.
Ekkor $v(0, a) = v(i, 0) = 0 \; \forall a, i$-re
cél $\rightarrow v(m, b)$



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

$\Longrightarrow$ Soronként kitölthető $\Longleftarrow$ minden érték két felette levőből számolható.
Összköltség: $O(bL)$
b-től függ (nem $\log b$-től!), ha b sokjegyű szám, ez sok idő

Algoritmuselmélet

Gráfok megadása, szélességi bejárás, összefüggőség, párosítás

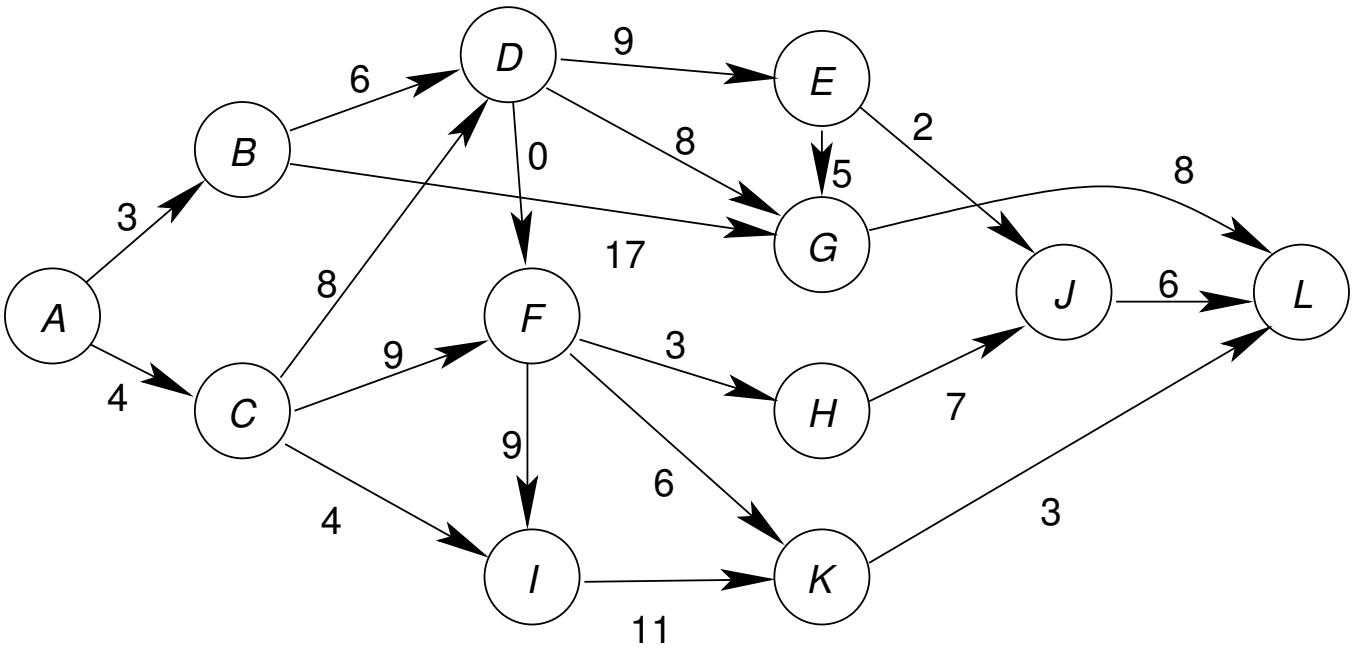
Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

2. előadás

Gráfalgoritmusok

- irányított gráfok: $G = (V, E)$
- irányított él, irányított út, irányított kör
- élsúlyok: $c(e)$ — lehetnek negatívak is



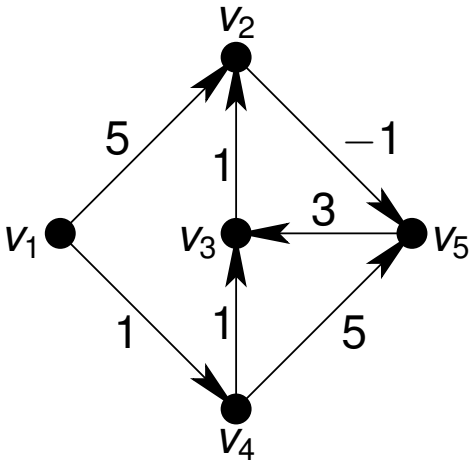
Adjacencia-mátrix

Definíció

A $G = (V, E)$ gráf adjacencia-mátrixa (vagy szomszédossági mátrixa) a következő – a V elemeivel indexelt – n-szer n-es mátrix:

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

Irányítatlan gráfok esetén a szomszédossági mátrix szimmetrikus lesz (azaz $A[i,j] = A[j,i]$ teljesül minden i, j csúcspárra).

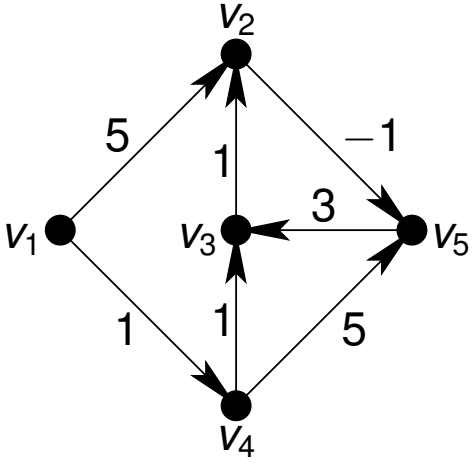


$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Súlyozott adjacencia-mátrix

Ha költségek is vannak $\Longrightarrow$

$$C[i,j] = \begin{cases} 0 & \text{ha } i = j, \\ c(i,j) & \text{ha } i \neq j \text{ és } (i,j) \text{ éle } G\text{-nek}, \\ * & \text{különben.} \end{cases}$$

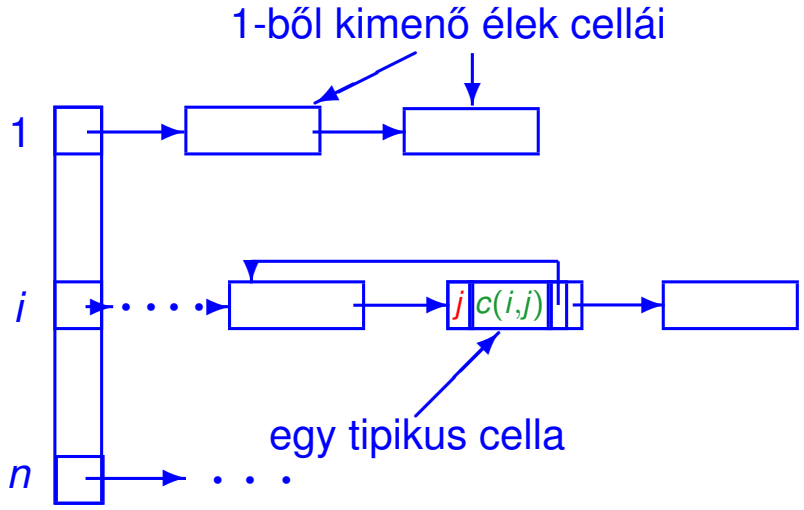


$$C = \begin{pmatrix} 0 & 5 & * & 1 & * \\ * & 0 & * & * & -1 \\ * & 1 & 0 & * & * \\ * & * & 1 & 0 & 5 \\ * & * & 3 & * & 0 \end{pmatrix}$$

Hátránya: a mérete (n^2 tömbelem) teljesen független az élek számától.

Éllistás megadás

$G = (V, E)$ gráf minden csúcsához egy lista tartozik.
Az $i \in V$ csúcs listájában tároljuk az i-ből kimenő éleket, és ha kell, ezek súlyát is.
Az i listáján egy élnek a lista egy eleme (cellája) felel meg.



Az (i, j) élnek megfelelő cella tartalmazza a j sorszámot, a $c(i, j)$ súlyt (ha van), egy mutatót a következő cellára, és esetleg még egyet az előzőre is.
Tárigény: $n + e$ cella,
Irányítatlan gráfoknál:
$n + 2e$ cella

Szélességi bejárás

BFS: Breadth First Search

Meglátogatjuk az első csúcsot, majd ennek a csúcsnak az összes szomszédját. Aztán ezen szomszédok összes olyan szomszédját, ahol még nem jártunk, és így tovább.
megvalósítás: *sor* (FIFO-lista)
Berakjuk az épp meglátogatott csúcsot, hogy majd a megfelelő időben a szomszédaira is sort keríthessünk.
Általános lépés: vesszük a sor elején levő x csúcsot, töröljük a sorból, meglátogatjuk azokat az y szomszédait, amelyeket eddig még nem láttunk, majd ezeket az y csúcsokat a sor végére tesszük.

Szélességi bejárás

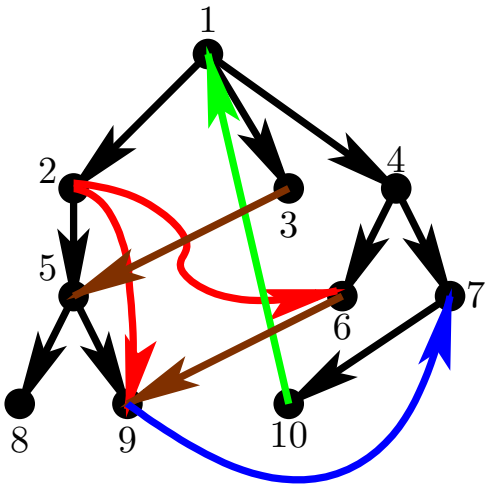
```
procedure szb (v: csúcs) (* szélességi bejárás egy komponensre *)
    var
        Q: csúcsokból álló sor;
        x, y: csúcsok;
    begin
        bejárva[v] := igaz;
        sorba(v, Q);
        while Q nem üres do begin
            x := első(Q); (ez egyben törli is x-et a sorból)
            for minden x → y ∈ E élre do
                if bejárva[y] = hamis then begin
                    bejárva[y] := igaz;
(*)                 sorba(y, Q)
                end
        end
    end
```


Szélességi bejárás

```
procedure bejár (* szélességi bejárás minden komponensre *)
    begin
        for v := 1 to n do
            bejárva[v] := hamis;
        for v := 1 to n do
            if bejárva[v] = hamis then
                szb(v)
    end
```

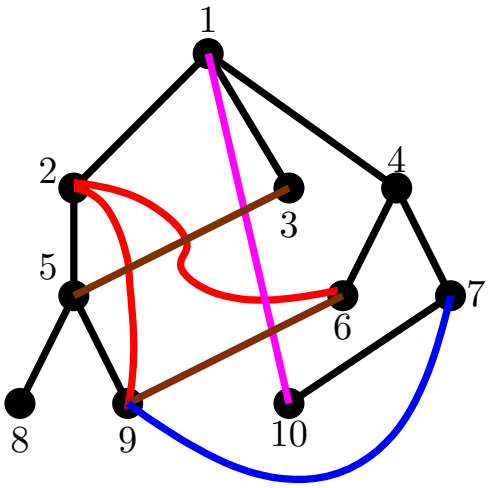
Összköltség: $O(n + e)$

Faél: megvizsgálásukkor még be nem járt pontba mutatnak



faél
ilyen nincs
visszaél
keresztél
keresztél

Irányítatlan esetben csak faél és keresztél lehet.



faél
ilyen nincs
ilyen nincs
keresztél
keresztél

Összefüggőség eldöntése

Kérdés

Adott G gráf összefüggő-e?

```
procedure Összefüggő
    begin
        öf:=igaz;
        tetszőleges v-re szb(v);
        for u := 1 to n do
            if bejárva[u] = hamis then öf:=hamis;
        return(öf);
    end
```

Összköltség: $O(n+e)$

Maximális párosítás keresése páros gráfban

BSZ-ből tudjuk, hogy javító utakat kell keresnünk.

Tétel

Egy párosítás akkor és csak akkor maximális, ha nincs hozzá tartozó javítóút.

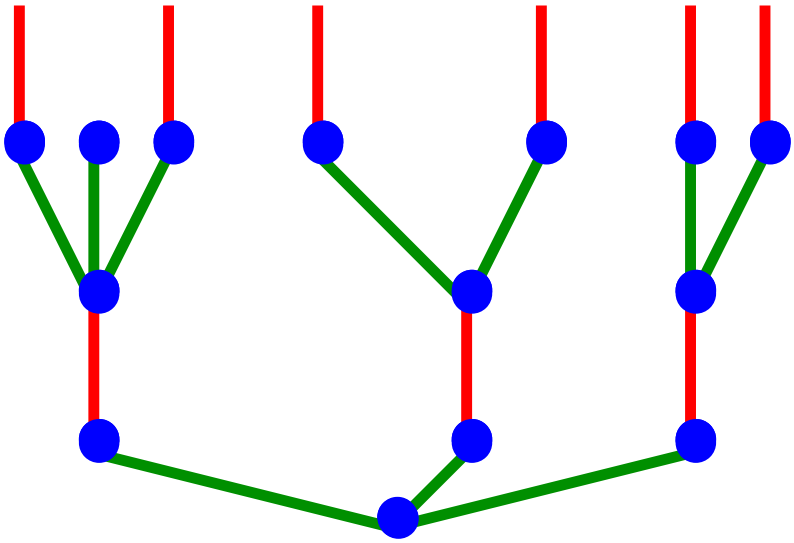
Kérdés

Hogyan döntjük el, hogy van-e javítóút?
Hogyan keresünk javítóutat?

Módosítjuk a szélességi keresést:

- Egy párosítatlan pontból indulunk
- Minden páratlan szinten a BFS lépéseit használjuk.
- Minden páros szinten a párosítás éleit használjuk.
- Ha olyan pontba érünk, aminek nincs párja, találtunk javítóutat

Maximális párosítás keresése páros gráfban



Költség:
Ha minden párosítatlan pontból építünk egy ilyen fát
$\Longrightarrow$ alternáló erdő $\Longrightarrow$ $O(e)$
Ezután eggyel növelhetjük a párok számát $\Longrightarrow$ $O(n)$-szer kell ismételni
Összköltség: $O(ne)$

Legrövidebb utak súlyozatlan gráfokban

Ha minden él hossza egy $\Longrightarrow$ út hossza = élek száma
Szélességi kereséssel: Jelentse $D[v]$ a v csúcsnak az s-től való távolságát az s-gyökerű szélességi fában.
Legyen kezdetben $D[s] := 0$
az *szb* eljárásba tegyük be a $D[y] := D[x] + 1$; utasítást, miután elértük y-t.
Lépésszám: $O(n + e)$

Tétel

Az előzőek szerint módosított szélességi bejárás végeztével teljesülnek a következők:
1. Legyen $s = x_1, x_2, \ldots, x_n$ a csúcsoknak a szélességi bejárás szerinti sorrendje. Ekkor $D[x_1] \leq D[x_2] \leq \ldots \leq D[x_n]$.
2. Ha $x \to y$ éle G-nek, akkor $D[y] \leq D[x] + 1$.
3. $D[v] = d(s, v)$ teljesül minden $v \in V$ csúcsra.

Tétel

1. $D[x_1] \leq D[x_2] \leq \ldots \leq D[x_n]$.

Bizonyítás.

A csúcsok az $s = x_1, x_2, \ldots, x_n$ sorrendben kerülnek bele a Q sorba $\Longrightarrow$ ebben a sorrendben is kerülnek ki a sorból.
Ha $x \neq s$ csúcs előbb van, mint y $\Longrightarrow$ $apa(x)$ nem lehet később, mint $apa(y)$, hiszen ha előbb lenne, y-hoz előbb eljutottunk volna.
Indukció $\Longrightarrow$ Gyökérre $D[s] = 0$, fiaira mind nagyobb. $\surd$
$D[x_i] = D[apa(x_i)] + 1$ és $D[x_{i+1}] = D[apa(x_{i+1})] + 1$ $\Longrightarrow$
Ha a két apa különböző
$\Longrightarrow D[apa(x_i)] \leq D[apa(x_{i+1})] \Longrightarrow D[x_i] \leq D[x_{i+1}]$.
Ha pedig az apák megegyeznek $\Longrightarrow D[x_i] = D[x_{i+1}]$.

□

Tétel

2. Ha $x \to y$ éle G-nek, akkor $D[y] \leq D[x] + 1$

Bizonyítás.

Nézzük, hogy mi történik, amikor x kikerül a Q sorból, és éppen az (x, y) élet vizsgáljuk.
Ha bejárva$[y] = \mathit{hamis} \implies y$ apja x, vagyis $D[y] = D[x] + 1$.
Különben y-t már korábban láttuk $\implies y$ apja előbb van, mint x
$\implies D[\mathit{apa}(y)] \leq D[x] \implies D[y] = D[\mathit{apa}(y)] + 1 \leq D[x] + 1$. □

Tétel

3. $D[v] = d(s, v)$ teljesül minden $v \in V$ csúcsra.

Bizonyítás.

$d(s, v) \leq D[v]$ $\surd$
Legyen $s = y_0, y_1, \ldots y_k = v$ egy minimális hosszúságú G-beli irányított út s-ből v-be.
$\implies$ az út éleire: $D[y_1] \leq D[s] + 1 = 1$, majd $D[y_2] \leq D[y_1] + 1 \leq 2 \ldots$
$D[v] = D[y_k] \leq k = d(s, v) \implies \surd$ □

Algoritmuselmélet

Legrövidebb utak, Bellmann-Ford, Dijkstra

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

3. előadás

A legrövidebb utak problémája

Legyen adott egy $G = (V, E)$ irányított gráf a $c(f)$, $f \in E$ élsúlyokkal.

Feladat

Mekkora a legrövidebb út egy adott pontból egy másik adott pontba?

Feladat

Mekkora a legrövidebb út egy adott pontból az összes többibe?

Feladat

Mekkora a legrövidebb út bármely két pont között?

A legrövidebb utak problémája

A G gráf egy u-t v-vel összekötő (nem feltétlenül egyszerű) $u \rightsquigarrow v$ irányított útjának a *hossza* az úton szereplő élek súlyainak összege.
Legrövidebb $u \rightsquigarrow v$ út: egy olyan $u \rightsquigarrow v$ út, melynek a hossza minimális a G-beli $u \rightsquigarrow v$ utak között.
u és v csúcsok (G-beli) $d(u, v)$ *távolsága*:
— 0, ha $u = v$;
— ∞, ha nincs $u \rightsquigarrow v$ út
— egyébként pedig a legrövidebb $u \rightsquigarrow v$ út hossza.

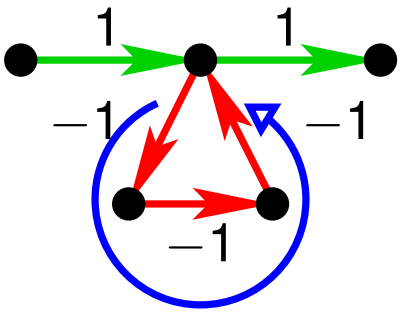
Vigyázat, itt u és v nem felcserélhető: ha az egyik csúcs valamilyen távol van a másiktól, akkor nem biztos, hogy a másik is ugyanolyan távol van az egyiktől!

A Bellman—Ford-módszer

Feladat

Egy pontból induló legrövidebb utak (hosszának) meghatározására, ha bizonyos élsúlyok negatívak. De feltesszük, hogy G nem tartalmaz negatív összhosszúságú irányított kört.

Ha van negatív összhosszúságú irányított kör $\Longrightarrow$ nincs legrövidebb út.



Legyen a $G = (V, E)$ súlyozott élű irányított gráf a C szomszédossági mátrixával adott (az i, j helyzetű elem a $c(i, j)$ élsúly, ha $i \to j$ éle G-nek, a többi elem pedig ∞).
Legyen $V = \{1, 2, \ldots, n\}$ és $s = 1 \iff s$-ből induló utakat keressük

Módszer: egy $T[1:n-1, 1:n]$ táblázat sorról sorra haladó kitöltése $\Longrightarrow$ Dinamikus programozás

$(*)$ $\quad T[i,j] =$ a legrövidebb olyan $1 \rightsquigarrow j$ irányított utak hossza, melyek legfeljebb i élből állnak.

$\Longrightarrow$ $T[n-1,j]$ a legrövidebb $1 \rightsquigarrow j$ utak hossza

A $T[1,j]$ sor kitöltése: $T[1,j] = C[1,j]$

Tegyük fel ezután, hogy az i-edik sort már kitöltöttük

$\Longrightarrow$ $T[i,1], T[i,2], \ldots, T[i,n]$ értékekre $(*)$ igaz. $\Longrightarrow$

$$(**) \qquad T[i+1,j] := \min\{T[i,j], \min_{k \neq j}\{T[i,k] + C[k,j]\}\}$$

$\Longleftarrow$ Ugyanis egy legfeljebb $i+1$ élből álló $\pi = 1 \rightsquigarrow j$ út kétféle lehet:

(a) Az útnak kevesebb, mint $i+1$ éle van. Ekkor ennek a hossza szerepel $T[i,j]$-ben.

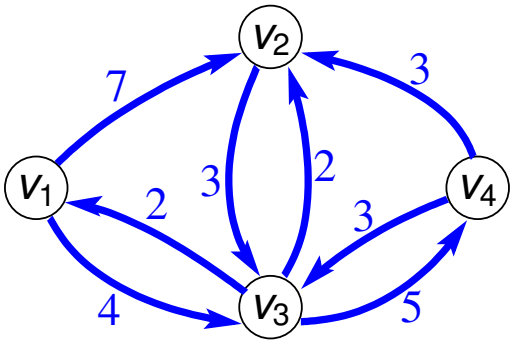
(b) Az út éppen $i+1$ élből áll. Legyen l a π út utolsó előtti pontja. Ekkor a π út $1 \rightsquigarrow l$ szakasza i élből áll, és minimális hosszúságú a legfeljebb i élű $1 \rightsquigarrow l$ utak között $\Longrightarrow$ π hossza $T[i,l] + C[l,j]$.

Lépésszám: Egy érték $(**)$ szerinti számítása $n-1$ összeadás és ugyanennyi összehasonlítás (minimumkeresés n elem közül)

$\Longrightarrow$ $O(n^3)$ művelet.

Példa

$$C[i,j] = \begin{pmatrix} 0 & 7 & 4 & \infty \\ \infty & 0 & 3 & \infty \\ 2 & 2 & 0 & 5 \\ \infty & 3 & 3 & 0 \end{pmatrix}$$



$T[i,j]$	1	2	3	4
1	0	7	4	∞
2	0	6	4	9
3	0	6	4	9

$T[2,2] = \min(7, 0+7, 4+2, \infty+3) = 6$

$T[2,4] = \min(\infty, 0+\infty, 7+\infty, 4+5) = 9$

Floyd módszere

Feladat

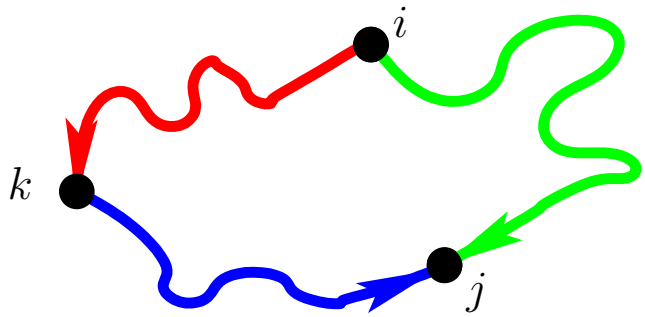
Miként lehet egy irányított gráfban az összes pontpár távolságát meghatározni?

≥ 0 élsúlyok $\Longrightarrow$ ha a Bellmann-Ford algoritmust minden csúcsra mint forrásra lefuttatjuk $\Longrightarrow$ $nO(n^3) = O(n^4)$
Van gyorsabb.

Feladat

Adott egy $G = (V, E)$ irányított gráf és egy $c : E \to \mathbb{R}$ súlyfüggvény úgy, hogy a gráfban nincs negatív összsúlyú irányított kör. Határozzuk meg az összes $v, w \in V$ rendezett pontpárra a $d(v, w)$ távolságot.

A G gráf a C szomszédossági mátrixával adott.
Egy szintén $n \times n$-es F mátrixot fogunk használni a számításhoz.
Kezdetben $F_0[i, j] := C[i, j]$.
ciklus $\Longrightarrow$ k-adik lefutása után $F_k[i, j]$ azon $i \rightsquigarrow j$ utak legrövidebbjeinek a hosszát tartalmazza, amelyek közbülső pontjai k-nál nem nagyobb sorszámúak.
Az új $F_k[i, j]$ értékeket kiszámíthatjuk, ha ismerjük $F_{k-1}[i, j]$-t $\forall i, j$-re.
Egy legrövidebb $i \rightsquigarrow j$ út, melyen a közbülső pontok sorszáma legfeljebb k, vagy tartalmazza a k csúcsot vagy nem.
— igen $\Longrightarrow$ $F_k[i, j] := F_{k-1}[i, k] + F_{k-1}[k, j]$



— nem $\Longrightarrow$ $F_k[i, j] = F_{k-1}[i, j]$

Floyd algoritmusa

```
(1) for i := 1 to n do
        for j := 1 to n do
            F[i,j] := C[i,j]
(2) for k := 1 to n do
        for i := 1 to n do
            for j := 1 to n do
                F[i,j] := min{F[i,j], F[i,k] + F[k,j]}
```

Tétel

$F[i,j]$ a (2)-beli iteráció k-adik menete után azon legrövidebb $i \rightsquigarrow j$ utak hosszát tartalmazza, amelyek belső csúcsai $1, 2, \ldots, k$ közül valók.

$k = n \implies F_n[i,j] = d(i,j)$

Lépésszám: n-szer megyünk végig a táblázaton, minden helyen $O(1)$ lépés $\implies O(n^3)$

A legrövidebb utak nyomon-követése

Menet közben karbantartunk egy $n \times n$-es P tömböt.
Kezdetben $P[i,j] := 0$.
Ha egy $F[i,j]$ értéket megváltoztatunk, mert találtunk egy k-n átmenő rövidebb utat, akkor $P[i,j] := k$.
$\implies$ Végül $P[i,j]$ egy legrövidebb $i \rightsquigarrow j$ út „középső" csúcsát fogja tartalmazni.
$i \rightsquigarrow j$ út összeállítása rekurzív $\implies$

```
procedure legrövidebb út(i, j:csúcs);
var k:csúcs;
begin
    k := P[i,j];
    if k = 0 then return;
    legrövidebb út(i, k);
    kiír(k);
    legrövidebb út(k, j)
end;
```

Tranzitív lezárás

Bemenet: $G = (V, E)$ irányított gráf.
Csak arra vagyunk kíváncsiak, hogy mely pontok között vezet irányított út.
Floyd $\Longrightarrow$ ha a végén $F[i, j] \neq \infty$, akkor van út, különben nincs.
Kicsit egyszerűbb, korábbi algoritmus: S. Warshall

Definíció (tranzitív lezárt)

Legyen $G = (V, E)$ egy irányított gráf, A a szomszédossági mátrixa. Legyen továbbá T a következő $n \times n$-es mátrix:

$$T[i, j] = \begin{cases} 1 & \text{ha } i\text{-ből } j \text{ elérhető irányított úttal;} \\ 0 & \text{különben.} \end{cases}$$

Ekkor a T mátrixot – illetve az általa meghatározott gráfot – az A mátrix – illetve az általa meghatározott G gráf – tranzitív lezártjának hívjuk.

Feladat

Adott a (Boole-mátrixként értelmezett) A szomszédossági mátrixával a $G = (V, E)$ irányított gráf. Adjuk meg a G tranzitív lezártját.

Warshall algoritmus

(1) ciklusban a kezdőértékek beállítása helyett

$$T_0[i, j] := \begin{cases} 1 & \text{ha } i = j \text{ vagy } A[i, j] = 1, \\ 0 & \text{különben.} \end{cases}$$

A (2) ciklusban F értékeinek változtatása helyett (ugyanazt megfogalmazva logikai műveletekkel)

$$(+) \qquad T_k[i, j] := T_{k-1}[i, j] \vee (T_{k-1}[i, k] \wedge T_{k-1}[k, j]).$$

Bizonyítás ugyanúgy.
Lépésszám: $O(n^3)$

Dijkstra módszere

Feladat

A legrövidebb utak problémája (egy forrásból):
Adott egy $G = (V, E)$ irányított gráf, a $c : E \to \mathbb{R}^+$ nemnegatív értékű súlyfüggvény és egy $s \in V$ csúcs (a forrás). Határozzuk meg minden $v \in V$-re a $d(s, v)$ távolságot.

$D[\,]$:

- Egy a G csúcsaival indexelt tömb
- az eljárás során addig megismert legrövidebb $s \rightsquigarrow v$ utak hossza
- Felső közelítése a keresett $d(s, v)$ távolságnak
- A közelítést lépésről lépésre finomítjuk, végül $d(s, v)$-t érjük el.

Dijkstra módszere

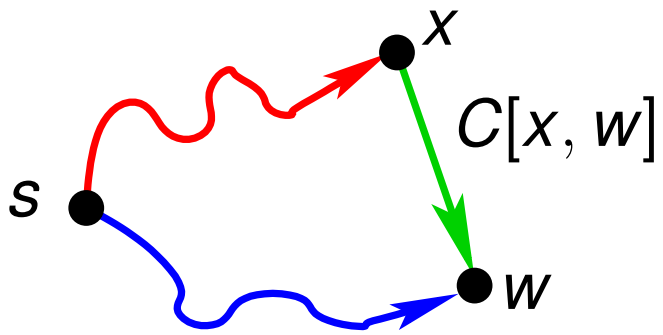
Tegyük fel, hogy a G gráf az alábbi alakú C szomszédossági mátrixával adott:

$$C[v, w] = \begin{cases} 0 & \text{ha } v = w, \\ c(v, w) & \text{ha } v \neq w \text{ és } (v, w) \text{ éle } G\text{-nek}, \\ \infty & \text{különben.} \end{cases}$$

Kezdetben $D[v] := C[s, v]$ minden $v \in V$ csúcsra.
Válasszuk ki ezután az s csúcs szomszédai közül a hozzá legközelebbit, vagyis egy olyan $x \in V \setminus \{s\}$ csúcsot, melyre $D[x]$ minimális.
Biztos, hogy az egyetlen (s, x) élből álló út egy legrövidebb $s \rightsquigarrow x$ út (az élsúlyok nemnegatívak!).
A KÉSZ halmaz azokat a csúcsokat tartalmazza, amelyeknek s-től való távolságát már tudjuk.
$\Longrightarrow$ x-et betehetjük (s mellé) a KÉSZ halmazba.

Dijkstra módszere

Ezek után módosítsuk a többi csúcs $D[w]$ értékét, ha az eddig ismertnél rövidebb úton el lehet érni oda x-en keresztül, azaz ha $D[x] + C[x, w] < D[w]$.



Újra válasszunk ki a $v \in V \setminus$ KÉSZ csúcsok közül egy olyat, amelyre $D[v]$ minimális.
Ezen csúcs $D[\,]$-értéke már az s-től való távolságát tartalmazza.
Majd megint a $D[\,]$-értékeket módosítjuk, és így tovább, míg minden csúcs be nem kerül a KÉSZ halmazba.

Dijkstra algoritmusa szomszédossági mátrixszal

(1) KÉSZ $:= \{s\}$
  for minden $v \in V$ csúcsra **do**
    $D[v] := C[s, v]$ ($*$ a $d(s, v)$ távolság első közelítése $*$)
(2) **for** $i := 1$ **to** $n - 1$ **do begin**
    Válasszunk olyan $x \in V \setminus$ KÉSZ csúcsot, melyre $D[x]$ minimális.
    Tegyük x-et a KÉSZ-be.
(3) **for** minden $w \in V \setminus$ KÉSZ csúcsra **do**
      $D[w] := \min\{D[w], D[x] + C[x, w]\}$ ($*$ $d(s, w)$ új közelítése $*$)
  end

Definíció

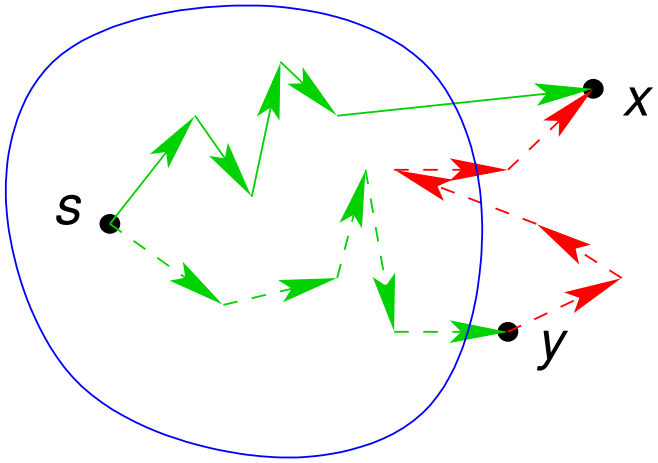
különleges út: egy $s \rightsquigarrow z$ irányított út különleges, ha a z végpontot kivéve minden pontja a KÉSZ halmazban van. A különleges úttal elérhető pontok éppen a KÉSZ-ből egyetlen éllel elérhető pontok.

Tétel

A (2) ciklus minden iterációs lépése után érvényesek a következők:

- *(a) KÉSZ pontjaira $D[v]$ a legrövidebb $s \rightsquigarrow v$ utak hossza.*
- *(b) Ha $v \in$ KÉSZ, akkor van olyan $d(s, v)$ hosszúságú (más szóval legrövidebb) $s \rightsquigarrow v$ út is, amelynek minden pontja a KÉSZ halmazban van.*
- *(c) Külső (vagyis $w \in V \setminus$ KÉSZ) pontokra $D[w]$ a legrövidebb különleges $s \rightsquigarrow w$ utak hossza.*

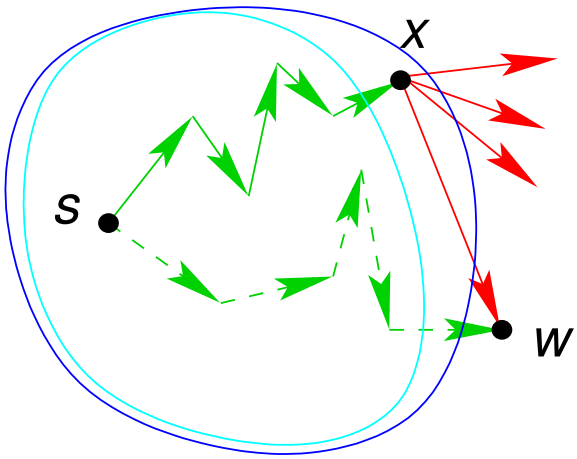
Bizonyítás.
Indukcióval
(2) előtt $\surd$
Tegyük fel, hogy igaz a j-edik iteráció után.
Belátjuk, hogy igaz a $j+1$-edik iteráció után is.
Tegyük fel, hogy az algoritmus a $j+1$. iterációs lépésben az x csúcsot választja a KÉSZ-be.



(a) Indirekt: mi van, ha $D[x]$ nem a $d(s, x)$ távolságot jelöli, azaz van ennél rövidebb $s \rightsquigarrow x$ út?
Ezen út „eleje" különleges, (c) miatt $\Longrightarrow$

$$D[y] \leq d(x, s) < D[x]$$

(b) Elég x-re $\Longleftarrow$ KÉSZ korábbi pontjaira az indukciós feltevésből
Láttuk, hogy $d(s, x) = D[x]$, ez egy különleges $s \rightsquigarrow x$ út hossza volt a $j+1$. iteráció előtt (itt a (c)-re vonatkozó indukciós feltevést használtuk) annak végeztével az út minden pontja KÉSZ-beli lesz.



(c) A $j+1$. iteráció előtt

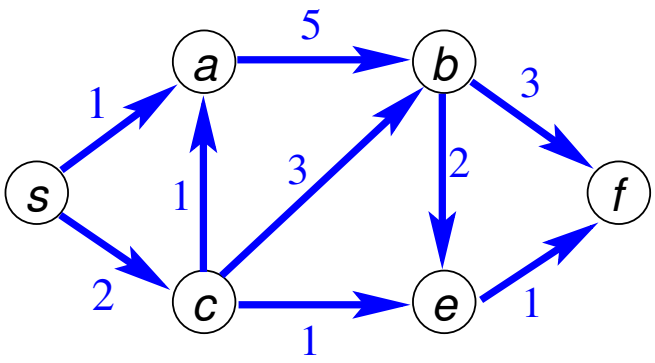
$$D[w] = \min_{v \in \text{KÉSZ} \setminus \{x\}} \{d(s,v) + C[v,w]\}.$$

Utána $$D[w] = \min_{v \in \text{KÉSZ}} \{d(s,v) + C[v,w]\}.$$

$\Longrightarrow$ Elég megnézni, hogy $D[w]$ vagy $d(s,x) + C[x,w]$ nagyobb

Lépészám: (2) ciklus $O(n)$, ez lefut $n-1$-szer $\Longrightarrow$ $O(n^2)$

Példa



D	a	b	c	e	f	KÉSZ
1.	1	∞	2	∞	∞	$\{s,a\}$
2.		6	2	∞	∞	$\{s,a,c\}$
3.		5		3	∞	$\{s,a,c,e\}$
4.		5			4	$\{s,a,c,e,f\}$
5.		5				$\{s,a,c,e,f,b\}$

Algoritmuselmélet

Kupac, Dijkstra kupaccal

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

4. előadás

Adatszerkezetek

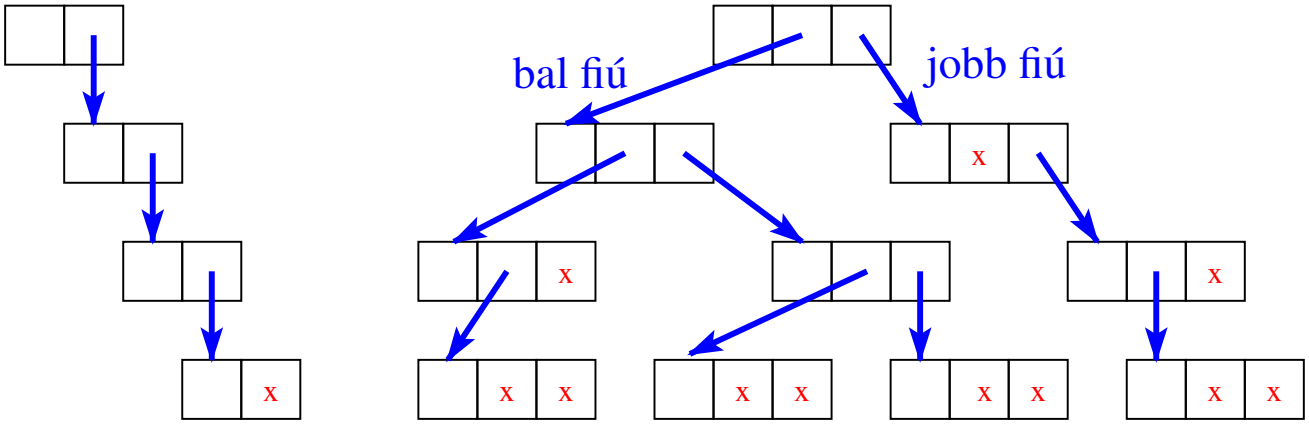
Definíció

Egy adatszerkezet elemek egy halmazának tárolása, az elemek közötti kapcsolat módja és az elemek kezeléséhez tartozó műveletek összessége.

Példák:

- Tömb – Műveletek: szelekciós függvény
- Láncolt lista – Műveletek: első, következő, keres, beszúr, beszúrMögé, töröl
- Sor – Műveletek: első, sorba, sorból

Bináris fa adatszerkezet



Láncolt lista Bináris fa

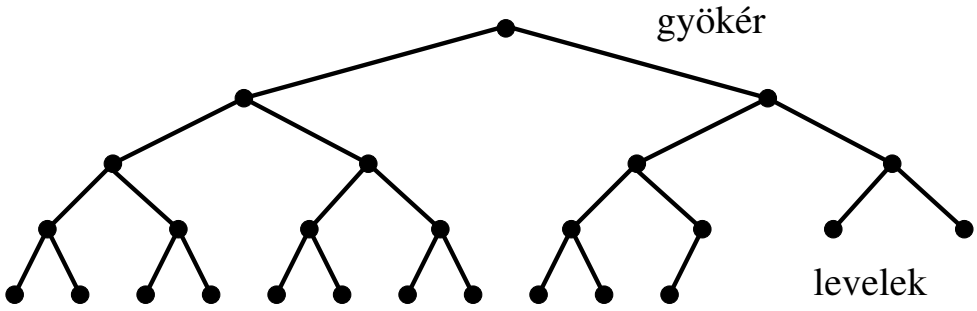
Műveletek: gyökér, bal-fiú, jobb-fiú, keres, beszúr, töröl

Kupac adatszerkezet

Egész számok egy S véges részhalmazát szeretnénk tárolni, hogy a *beszúrás* és a minimális elem törlése (*mintör*) hatékony legyen.
Alkalmazások:

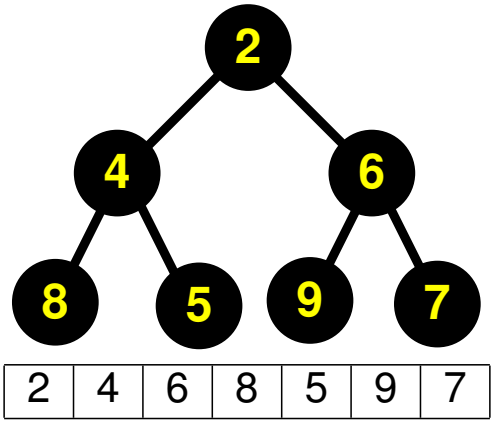
- Jobok indítása
- Több rendezett halmaz összefésülése
- Gyors rendezési algoritmus

Teljes bináris fa:



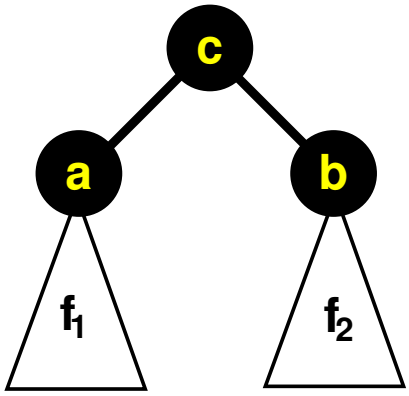
Bináris fa ábrázolása tömbbel

A fa csúcsai az $A[1:n]$ tömb elemei.
Az $A[i]$ csúcs bal fia $A[2i]$, a jobb fia pedig $A[2i+1]$.
$\Longrightarrow$ $A[j]$ csúcs apja $A[\lfloor j/2 \rfloor]$



Kupac tulajdonság: apa $<$ fia

Kupacépítés



f_1 és f_2 kupacok

kupacol(f)
{ Ha $\min\{a,b\} < c$, akkor $\min\{a,b\}$ és c helyet cserél
Ha a c elem a-val cserélt helyet, akkor *kupacol*(f_1), ha b-vel, akkor *kupacol*(f_2) }

c addig megy lefelé, amíg sérti a kupac tulajdonságot.
Lépésszám: Ha l a fa szintjeinek száma, akkor $\leq l-1$ csere és $\leq 2(l-1)$ összehasonlítás

kupacépítés(f)
{ Az f fa v csúcsaira lentről felfelé, jobbról balra *kupacol*(v). }

Kupacépítés költsége

n-csúcsú bináris fában:

1. szint: 1 csúcs
2. szint: 2 csúcs
3. szint: 2^2 csúcs
$\vdots$
$l-1$. szint: 2^{l-2} csúcs
l-edik szint: ≥ 1 és $\leq 2^{l-1}$ csúcs

$\Longrightarrow n \geq 1 + \sum_{i=0}^{l-2} 2^i = 2^{l-1} \Longrightarrow l \leq 1 + \log_2 n$

Az i-edik szinten levő v csúcsra *kupacol*(v) költsége legfeljebb $l-i$ csere és legfeljebb $2(l-i)$ összehasonlítás.
A cserék száma ezért összesen legfeljebb $\sum_{i=1}^{l}(l-i)2^{i-1}$.
$j = l - i$ (azaz $i = l - j$) helyettesítéssel

$$\sum_{j=0}^{l-1} j2^{l-j-1} = 2^{l-1} \sum_{j=1}^{l-1} j/2^j$$

Kupacépítés költsége

$$\begin{array}{llllll} 1/2 & & & & & \\ 1/4 & 1/4 & & & & \\ 1/8 & 1/8 & 1/8 & & & \\ \vdots & & & & & \\ \frac{1}{2^{l-1}} & \frac{1}{2^{l-1}} & \frac{1}{2^{l-1}} & \cdots & \frac{1}{2^{l-1}} \\ \hline < 1 & < 1/2 & < 1/4 & \cdots & \leq \frac{1}{2^{l-1}} \end{array}$$

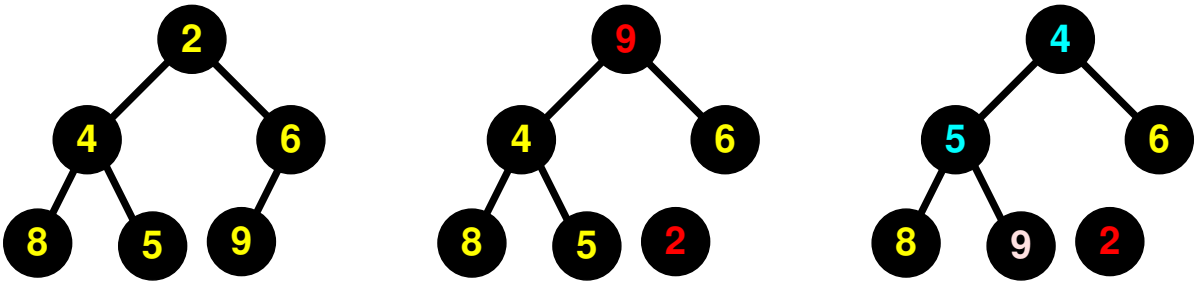
$$2^{l-1} \sum_{j=1}^{l-1} j/2^j < 2^l \leq 2n.$$

Tétel

Kupacépítés költsége: $O(n)$

MINTÖR

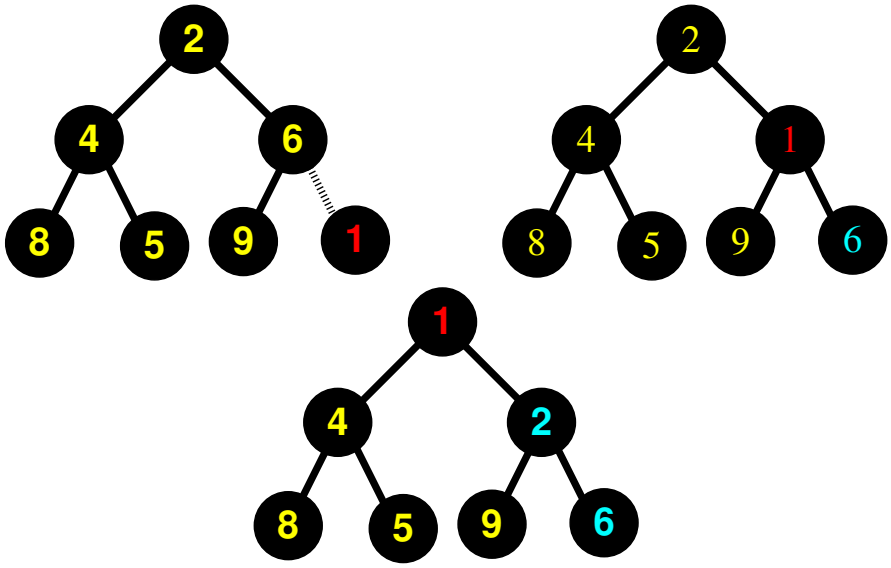
A minimális elem az f gyökérben van, ezt töröljük.
A f-be tesszük a fa utolsó szintjének jobb szélső elemét, majd *kupacol*(f).



Költség: $O(l) = O(\log_2 n)$

BESZÚR

Új levelet adunk a fához (ügyelve a teljességre), ide tesszük az s elemet. Ezután s-et mozgatjuk felfelé, mindig összehasonlítjuk az apjával.



Költség: $O(l) = O(\log_2 n)$

Dijkstra algoritmusa éllistával

Ha kevés él van $\Longrightarrow$ gráfot éllistával tároljuk.
$V \setminus$KÉSZ csúcsait *kupacba rendezve* tartjuk a $D[\,]$ érték szerint.
A kupacépítés $O(n)$ költség, a (2) ciklusban minimumkeresést $O(\log n)$ költségű MINTÖR végrehajtásával számoljuk.
A $D[\,]$ értékek újraszámolását és a kupac-tulajdonság helyreállítását csak a választott csúcs szomszédaira kell elvégezni.
Minden csúcsot pontosan egyszer választunk ki, és a szomszédok számának összege e. $\Longrightarrow$ Összidőigény $O((n+e)\log n)$.

Sűrű gráfokra: d-kupac.
$\Longrightarrow O(n + nd\log_d n + e\log_d n)$
Ha $n^{1,5} \leq e \leq n^2$ és legyen $d = \lceil e/n \rceil$ $\Longrightarrow d \geq \sqrt{n}$ $\Longrightarrow \log_d n \leq 2$.
$\Longrightarrow$

$$O(n+nd\log_d n+e\log_d n) = O(n+nd+e) = O(n+n\cdot e/n+e) = O(e).$$

Dijkstra irányítatlan gráfokra is működik.

A legrövidebb utak nyomon követése

Minden pontra tárolunk és karbantartunk egy $P[x]$ csúcsot is, ami megadja egy az eddig ismert hozzá vezető legrövidebb úton az utolsó előtti csúcsot.
Kezdetben $P[v] := s$ minden $v \in V$-re.
A (3) ciklus belsejében, ha egy külső w csúcs $D[w]$ értékét megváltoztatjuk, akkor $P[w] := x$.
Lépésszám: $O(n^2)$

Algoritmuselmélet

Keresés, minimumkeresés

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

5. előadás

Rendezési reláció

Legyen U egy halmaz, és $<$ egy kétváltozós reláció U-n. Ha $a, b \in U$ és $a < b$, akkor azt mondjuk, hogy a kisebb, mint b.

A $<$ reláció egy rendezés, ha teljesülnek a következők:
1. $a \not< a$ minden $a \in U$ elemre ($<$ irreflexív);
2. Ha $a, b, c \in U$, $a < b$, és $b < c$, akkor $a < c$ ($<$ tranzitív);
3. Tetszőleges $a \neq b \in U$ elemekre vagy $a < b$, vagy $b < a$ fennáll ($<$ teljes).

Ha $<$ egy rendezés U-n, akkor az $(U, <)$ párt rendezett halmaznak nevezzük.

Rendezési reláció

Példák:

- $\mathbb{Z}$ az egész számok halmaza. A $<$ rendezés a nagyság szerinti rendezés.
- Az *abc* betűinek Σ halmaza; a $<$ rendezést az *abc-sorrend* adja. Az x betű kisebb, mint az y betű, ha x előbb szerepel az *abc*-sorrendben, mint y.
- A Σ betűiből alkotott szavak Σ^* halmaza a szótárszerű vagy lexikografikus rendezéssel. $\Rightarrow$ legyen $X = x_1 x_2 \cdots x_k$ és $Y = y_1 y_2 \cdots y_l$ két szó.
  Az X kisebb mint Y, ha vagy $l > k$ és $x_i = y_i$ ha minden $i = 1, 2, \ldots, k$ esetén;
  vagy pedig $x_j < y_j$ teljesül a legkisebb olyan j indexre, melyre $x_j \neq y_j$. Tehát például *kar* $<$ *karika* és *bor* $<$ *bot*.

Keresés rendezetlen halmazban

Feladat

Adott az U halmaz véges $S = \{s_1, s_2, \ldots, s_{n-1}, s_n\}$ részhalmaza és $s \in U$.
El akarjuk eldönteni, hogy igaz-e $s \in S$, és ha igen, akkor melyik i-re teljesül $s_i = s$.
Hány összehasonlítás kell?

Itt összehasonlítás: Igaz-e, hogy $s_i = s$?
Válasz: Igen vagy nem.

Legrosszabb esetben minden elemet végig kell nézni $\Longrightarrow$
n összehasonlítás kell legrosszabb esetben.
$n/2$ összehasonlítás kell átlagosan.

Keresés rendezett halmazban

Barkochba játék: gondolok egy számot 1 és 100 között, hány eldöntendő kérdésből lehet kitalálni?

Feladat

Adott az $(U, <)$ rendezett halmaz véges
$S = \{s_1 < s_2 < \ldots < s_{n-1} < s_n\}$ részhalmaza és $s \in U$.
Összehasonlításokkal akarjuk eldönteni, hogy igaz-e $s \in S$, és ha igen, akkor melyik i-re teljesül $s_i = s$.
Hány összehasonlítás kell?

Itt összehasonlítás: Mi a viszonya s-nek és s_i-nek?
Válasz: $s_i = s$ vagy $s_i < s$ vagy $s_i > s$.

Lineáris keresés

Sorban mindegyik elemmel összehasonlítjuk.
Költség a legrosszabb esetben: n, mert lehet, hogy pont az utolsó volt.
Költség átlagos esetben esetben: $(n/2) + 1$.

Bináris keresés

Oszd meg és uralkodj: először a középső s_i-vel hasonlítunk.
Hasonló feladatot kapunk egy S_1 halmazra, amire viszont $|S_1| \leq |S|/2$.
És így tovább:

$$|S_2| \leq \frac{|S|}{4}, |S_3| \leq \frac{|S|}{2^3}, \ldots |S_k| \leq \frac{|S|}{2^k}$$

Pl. keressük meg, benne van-e 21 az alábbi sorozatban!

$$15, 22, 25, 37, \boxed{48}, 56, 70, 82 \quad (1)$$
$$15, 22, \boxed{25}, 37, 48, 56, 70, 82 \quad (2)$$
$$15, \boxed{22}, 25, 37, 48, 56, 70, 82 \quad (3)$$
$$\boxed{15}, 22, 25, 37, 48, 56, 70, 82 \quad (4)$$

Bináris keresés

Addig kell csinálni, amíg $|S_k| = 1$ lesz. Innen $1 = |S_k| \leq \frac{n}{2^k}$.
$\Longrightarrow 2^k \leq n \Longrightarrow k \leq \lfloor \log_2 n \rfloor$
Ez $k+1$ összehasonlítás volt. $\Longrightarrow k+1 \leq \lfloor \log_2 n \rfloor + 1 = \lceil \log_2(n+1) \rceil$

Tétel

Ez optimális, nincs olyan kereső algoritmus, ami minden esetben kevesebb mint $\lceil \log_2(n+1) \rceil$ kérdést használ.

Bináris keresés

Tétel

Ez optimális, nincs olyan kereső algoritmus, ami minden esetben kevesebb mint $\lceil \log_2(n+1) \rceil$ kérdést használ.

Bizonyítás.

Az ellenség nem is gondol egy számra, csak mindig úgy válaszol, hogy minél többet kelljen kérdezni. Ha egy kérdést felteszek, és az igen válasz után mondjuk szóba jön x lehetőség, akkor a nem esetén szóba jön még $n - x - 1$ lehetőség. (A „-1” az $s = s_i$ válasz miatt van).
Az ellenség úgy válaszol, hogy minél több lehetőség maradjon, így el tudja érni, hogy legalább $\frac{n-1}{2}$ marad.
$\Longrightarrow$ 2 kérdés után legalább $\frac{\frac{n-1}{2}-1}{2} = \frac{n}{2^2} - \frac{1}{2} - \frac{1}{2^2}$ marad.
$\Longrightarrow$ k kérdés után is marad még $\frac{n}{2^k} - \frac{1}{2} - \cdots - \frac{1}{2^k}$ lehetőség.
Tehát teljesülnie kell $\frac{n}{2^k} - \frac{1}{2} - \cdots - \frac{1}{2^k} \leq 1$-nek.
Vagyis $n \leq 2^k + 2^{k-1} + \ldots + 1 = 2^{k+1} - 1$. $\Longrightarrow \lceil \log_2(n+1) \rceil - 1 \leq k$.
Ha még van egy lehetséges elem, akkor még +1 egy kérdés. □

Minimumkeresés

Feladat

Adott az $(U, <)$ rendezett halmaz véges $S = \{s_1, s_2, \ldots, s_{n-1}, s_n\}$ részhalmaza.
Összehasonlításokkal keressük meg az S minimális elemét, azaz egy olyan s_i elemet, hogy minden $i \neq j$ esetén $s_i < s_j$.

- Hány összehasonlítás kell a legrosszabb esetben?

Minimumkeresés

Tétel

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

Bizonyítás.

$n - 1$ összehasonlítás mindig elég: Rendezzünk kiesés versenyt, mindig a kisebb elemet megtartva egy-egy összehasonlítás után. Mivel „mindenki pontosan egyszer kap ki a győztest kivéve”, ez $n - 1$ összehasonlítást igényel.
$n - 1$ összehasonlításnál kevesebb nem mindig elég: Legyenek az elemek egy gráf pontjai, ha kettőt összehasonlítottunk, húzzunk közöttük élet. Amíg a gráf nem összefüggő, bármely komponensében lehet a minimális elem.
Ha a gráf már összefüggő, akkor legalább $n - 1$ éle van, tehát kell ennyi összehasonlítás. □

Algoritmuselmélet

Rendezés, buborék, beszúrásos, összefésüléses, kupacos, láda, radix

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

6. előadás

Rendezés

Feladat

Adott az $(U, <)$ rendezett halmaz véges $S = \{s_1, s_2, \ldots, s_{n-1}, s_n\}$ részhalmaza.
Összehasonlításokkal rendezzük az S elemeit a rendezés szerint növekvő sorrendbe, azaz keressünk olyan σ permutációt, hogy $s_{\sigma(1)} < s_{\sigma(2)} < \cdots < s_{\sigma(n)}$.

Input: tömb, láncolt lista, (vagy bármi)
Output: általában, mint az input
Lépések: elemek mozgatása, cseréje, összehasonlítása

A rendezés önmagában is előforduló feladat, de előjön, mint hasznos adatstruktúra is. Rendezett halmazban könnyebb keresni (pl. telefonkönyv).

- Hány összehasonlítás kell a legrosszabb esetben?
- Hány összehasonlítás kell átlagos esetben?
- Hány csere kell a legrosszabb esetben?
- Mennyi plusz tárhely szükséges?

Buborék-rendezés

Input: $A[1:n]$ (rendezetlen) tömb
Ha valamely i-re $A[i] > A[i+1]$, akkor a két cella tartalmát kicseréljük.
A tömb elejéről indulva, közben cserélgetve eljutunk a tömb végéig.
Ekkor a legnagyobb elem $A[n]$-ben van. Ismételjük ezt az $A[1:n-1]$ tömbre, majd az $A[1:n-2]$ tömbre, stb.

procedure buborék
(az $A[1:n]$ tömböt növekvően (nem csökkenően) rendezi *)*
for $(j = n-1, j > 0, j := j-1)$ **do**
  for $(i = 1, i \le j, i := i+1)$ **do**
    { ha $A[i+1] < A[i]$, akkor cseréljük ki őket.}

összehasonlítások száma: $n-1+n-2+\ldots+1 = \frac{n(n-1)}{2}$

cserék száma: $\le \frac{n(n-1)}{2}$

Beszúrásos rendezés

Ha az $A[1:k]$ résztömb már rendezett, akkor szúrjuk be a következő elemet, $A[k+1]$-et, lineáris vagy bináris kereséssel, majd a következőt ebbe, stb.

	lineáris	bináris
összehasonlítás	$\frac{n(n-1)}{2}$	$\sum_{k=1}^{n-1} \lceil \log_2(k+1) \rceil$
mozgatás	$\frac{(n+2)(n-1)}{2}$	$\frac{(n+2)(n-1)}{2}$
átlagos összehasonlítás	$\frac{n(n-1)}{4}$	$\sum_{k=1}^{n-1} \lceil \log_2(n+1) \rceil$
átlagos mozgatás	$\frac{n^2}{4}$	$\frac{n^2}{4}$

Bináris beszúrásos rendezés lépésszáma

$$K := \lceil \log_2 2 \rceil + \lceil \log_2 3 \rceil + \cdots + \lceil \log_2 n \rceil \leq n \lceil \log_2 n \rceil$$

Jobb becslés: használjuk fel, hogy $\lceil \log_2 k \rceil \leq 1 + \log_2 k$

$$K < n - 1 + \log_2 2 + \cdots + \log_2 n = n - 1 + \log_2(n!)$$

Felhasználva a Stirling formulát: $n! \sim (n/e)^n \sqrt{2\pi n}$ kapjuk, hogy

$$\log_2 n! \sim n(\log_2 n - \log_2 e) + \frac{1}{2} \log_2 n + \log_2 \sqrt{2\pi} \sim n(\log_2 n - 1,442)$$

Ezért $K \leq n(\log_2 n - 0,442)$ elég nagy n-re.

Alsó becslés összehasonlítás alapú rendezésre

Ugyanaz, mintha barochba-ban kellene kitalálni, hogy az elemek melyik sorrendje (permutációja) az igazi sorrend.

Kezdetben $n!$ lehetséges sorrend jön szóba.
Két elemet összehasonlítva a válasz két részre osztja a sorrendeket.
Ha pl. azt kapjuk, hogy $x < y$, akkor az olyan sorrendek, amelyekben x hátrébb van y-nál, már nem jönnek szóba.
Ha az ellenség megint úgy válaszol, hogy minél több sorrend maradjon meg, akkor k kérdés után még szóba jön $\frac{n!}{2^k}$ sorrend.
Ha $\frac{n!}{2^k} > 1$, nem tudjuk megadni a rendezést. $\Longrightarrow$

Tétel

Minden összehasonlítás alapú rendező módszer n elem rendezésekor legalább $\log_2(n!)$ *összehasonlítást használ.*

Összefésüléses rendezés

Összefésülés (MERGE):

Két már rendezett sorozat (tömb, lista, stb.) tartalmának egy sorozatba való rendezése:
$A[1:k]$ és $B[1:l]$ rendezett tömbök $\longrightarrow$ $C[1:k+l]$ rendezett tömb
Nyilván $C[1] = \min\{A[1], B[1]\}$, pl. $A[1]$,
ezt rakjuk át C-be és töröljük A-ból.
$C[2] = \min\{A[2], B[1]\}$, stb.

Példa

A	B	C
12, 15, 20, 31	13, 16, 18	
15, 20, 31	13, 16, 18	12,
15, 20, 31	16, 18	12, 13
20, 31	16, 18	12, 13, 15
20, 31	18	12, 13, 15, 16
20, 31		12, 13, 15, 16, 18
31		12, 13, 15, 16, 18, 20
		12, 13, 15, 16, 18, 20, 31

összehasonlítások száma: $k + l - 1$, ahol k, l a két tömb hossza

Összefésüléses rendezés

Alapötlet: Rendezzük külön a tömb első felét, majd a második felét, végül fésüljük össze.
Ezt csináljuk rekurzívan.

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

Hogy elvarrjuk a rekurzió alját, legyen $\text{MSORT}(A[i,i])$ az üres utasítás.

Összehasonlítások száma

Jelöljük $T(n)$-el a lépésszámot n hosszú tömb rendezésekor. Az egyszerűség kedvéért tegyük fel, hogy $n = 2^k$.

$$T(n) \leq n - 1 + 2T(n/2),$$

$$T(n) \leq n - 1 + 2(n/2 - 1 + 2T(n/4)) = n - 1 + 2(n/2 - 1) + 4T(n/4).$$

$$T(n) \leq n-1+2(n/2-1)+4(n/4-1)+\cdots+2^{k-1}(n/2^{k-1}-1) \leq n\lceil \log_2 n\rceil.$$

Felhasználva, hogy $T(1) = 0$.
Az összefésüléses rendezés konstans szorzó erejéig optimális.
Mozgatások száma: $2n\lceil \log_2 n\rceil$
Tárigény: $2n$ cella (bonyolultabban megcsinálva elég $n + konst$.)

Példa összefésüléses rendezésre

2	$\vert_3$	8	$\vert_2$	7	$\vert_4$	5	$\vert_1$	6	$\vert_6$	4	$\vert_5$	1	$\vert_7$	3
2		8	$\vert_2$	5		7	$\vert_1$	4		6	$\vert_5$	1		3
2		5		7		8	$\vert_1$	1		3		4		6
1		2		3		4		5		6		7		8

A kupacos rendezés

Először kupacot építünk, utána n darab MINTÖR adja nem csökkenő sorrendben az elemeket.
[J. W. J. Williams és R. W. Floyd, 1964]

Költség: $O(n) + O(n \log_2 n) = O(n \log_2 n)$

Legjobb ismert rendező algoritmus.
Pontos implementációval:
$2n\lfloor \log_2 n \rfloor + 3n$ (összehasonlítások száma) és $n\lfloor \log_2 n \rfloor + 2,5n$ (cserék száma).

Gyorsrendezés

[C. A. R. Hoare, 1960]
oszd meg és uralkodj: véletlen s elem a tömbből $\longrightarrow$ PARTÍCIÓ(s) $\longrightarrow$

s-nél kisebb elemek	s	$\ldots$	s	*s-nél nagyobb elemek*

GYORSREND($A[1:n]$)
1. Válasszunk egy véletlen s elemet az A tömbből.
2. PARTÍCIÓ(s); az eredmény legyen az $A[1:k]$, $A[k+1:l]$, $A[l+1:n]$ felbontás.
3. GYORSREND($A[1:k]$); GYORSREND($A[l+1:n]$).

Véletlen elemnek választhatjuk mindig a tömb első helyén állót.

A PARTÍCIÓ(s) működése

Legyen $i := 1, j := n$,

- i-t növeljük, amíg $A[i] < s$ teljesül
- j-t csökkentjük, amíg $A[j] \geq s$

$\Longrightarrow$

	$i \rightarrow$		$\leftarrow j$	
s-nél kisebb elemek				*s-nél nem kisebb elemek*

Ha mindkettő megáll (nem lehet továbblépni), és $i < j$, akkor $A[i] \geq s$ és $A[j] < s \Longrightarrow$
Kicseréljük $A[i]$ és $A[j]$ tartalmát, majd $i := i + 1$ és $j := j - 1$. Ha a két mutató összeér (már nem teljesül $i < j$), akkor s előfordulásait a felső rész elejére mozgatjuk.
PARTÍCIÓ lépésszáma: $O(n)$
GYORSREND lépésszáma *legrosszabb esetben*: $O(n^2)$
GYORSREND lépésszáma *átlagos esetben*:
$1,39 n \log_2 n + O(n) = O(n \log n)$

Kulcsmanipulációs rendezések

Nem csak összehasonlításokat használ.
Pl. ismerjük az elemek számát, belső szerkezetét.

Ládarendezés (binsort)

Tudjuk, hogy $A[1:n]$ elemei egy m elemű U halmazból kerülnek ki, pl. $\in \{1,\ldots,m\}$
$\Longrightarrow$ Lefoglalunk egy U elemeivel indexelt B tömböt (m db ládát), először mind üres.
Első fázis: végigolvassuk az A-t, és az $s = A[i]$ elemet a $B[s]$ lista végére fűzzük.

$\Longrightarrow$ konzervatív rendezés, azaz az egyenlő elemek sorrendjét megtartja.

Ládarendezés

Példa: Tegyük fel, hogy a rendezendő $A[1:7]$ tömb elemei 0 és 9 közötti egészek:

A:	5	3	1	5	6	9	6

B:		1		3		5 5	6 6			9

Második fázis: elejétől a végéig növő sorrendben végigmegyünk B-n, és a $B[i]$ listák tartalmát visszaírjuk A-ba.

B:		1		3		5 5	6 6			9

A:	1	3	5	5	6	6	9

Lépésszám: B létrehozása $O(m)$, első fázis $O(n)$, második fázis $O(n+m)$, összesen $O(n+m)$.
Ez gyorsabb, mint az általános alsó korlát, ha pl. $m \leq cn$.

Radix rendezés

A kulcsok összetettek, több komponensből állnak, $t_1 \dots t_k$ alakú szavak, ahol a t_i komponens az L_i rendezett típusból való, legyen $|L_i| = s_i$, a rendezés lexikografikus.
Példa: Legyen $(U, <)$ a *huszadik századi dátumok* összessége az időrendnek megfelelő rendezéssel.

$$L_1 = \{1900, 1901, \dots, 1999\}, \quad s_1 = 100.$$

$$L_2 = \{\textit{január, február,..., december}\}, \quad s_2 = 12.$$

$$L_3 = \{1, 2, \dots, 31\}, \quad s_3 = 31.$$

A dátumok rendezése éppen az L_i típusokból származó lexikografikus rendezés lesz.

Radix rendezés

- Rendezzük a sorozatot az utolsó, a k-adik komponens szerint ládarendezéssel.
- A kapottat rendezzük a $k-1$-edik komponens szerint ládarendezéssel.
- stb.

Fontos, hogy a ládarendezésnél, az elemeket a ládában mindig a lista végére tettük. Így ha két azonos kulcsú elem közül az egyik megelőzi a másikat, akkor a rendezés után sem változik a sorrendjük.
$\longrightarrow$ Az ilyen rendezést konzervatív rendezésnek nevezzük.

Miért működik a radix jól?

Ha $X < Y$, az első $i-1$ tag megegyezik, de $x_i < y_i$, akkor az i-edik komponens rendezésekor X előre kerül.
A láderendezés konzervatív $\Longrightarrow$ később már nem változik a sorrendjük.
Példa:

1969.01.18.	1969.01.01.	1955.12.18.	1955.01.18.	1918.12.18.

Napok szerint rendezve:

1969.01.01.	1969.01.18.	1955.12.18.	1955.01.18.	1918.12.18.

Hónapok szerint rendezve:

1969.01.01.	1969.01.18.	1955.01.18.	1955.12.18.	1918.12.18.

Évek szerint rendezve:

1918.12.18.	1955.01.18.	1955.12.18.	1969.01.01.	1969.01.18.

Radix rendezés

Lépésszám: k ládarendezés összköltsége: $O(kn + \sum_{i=1}^{k} s_i)$

Ez lehet gyorsabb az általános korlátnál

- c, k állandók és $s_i \leq cn$
  $\Longrightarrow O(kn + \sum_{i=1}^{k} cn) = O(k(c+1)n) = O(n)$.
  pl. az $[1, n^{10} - 1]$ intervallumból való egészek rendezése
- $k = \log n$, $s_i = 2 \Longrightarrow O(n \log n + 2 \log n) = O(n \log n)$.

Algoritmuselmélet

Keresőfák, piros-fekete fák

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

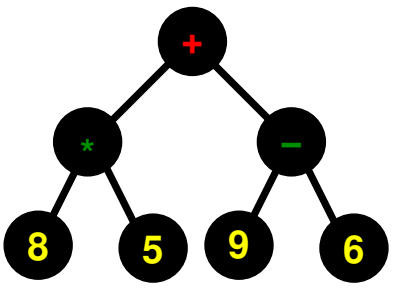
7. előadás

Keresőfák

Tároljuk az U rendezett halmaz elemeit, hogy BESZÚR, TÖRÖL, KERES, MIN, (MAX, TÓLIG) hatékonyak legyenek.

Bináris fa bejárása

teljes fa (új def.): az alsó szint is tele van $\Longrightarrow$ l szintű, teljes fának $2^l - 1$ csúcsa van.
Fa csúcsai $\rightarrow$ $elem(x), bal(x), jobb(x)$ esetleg $apa(x)$ és $reszfa(x)$

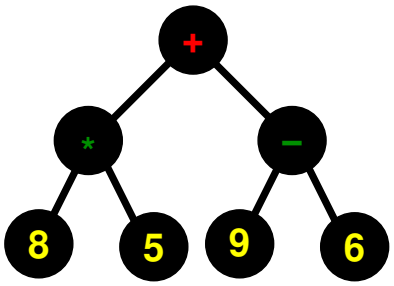


Ha x a gyökér, y pedig a 9-es csúcs, akkor

$$\begin{aligned} bal(jobb(x)) &= y, \\ apa(apa(y)) &= x, \\ elem(bal(x)) &= *, \\ reszfa(x) &= 7. \end{aligned}$$

PREORDER, INORDER, POSTORDER

```
pre(x)                  in(x)                  post(x)
begin                   begin                  begin
látogat(x);             in(bal(x));            post(bal(x));
pre(bal(x));            látogat(x);            post(jobb(x));
pre(jobb(x))            in(jobb(x))            látogat(x)
end                     end                    end
```



PREORDER: $+ * 8\ 5 - 9\ 6$

INORDER: $8 * 5 + 9 - 6$

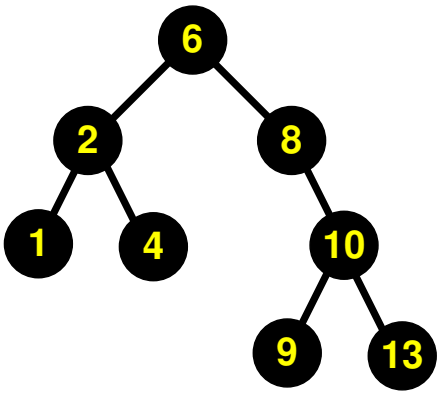
POSTORDER: $8\ 5 * 9\ 6 - +$

Lépésszám: $O(n)$

Bináris keresőfa

Definíció (Keresőfa-tulajdonság)

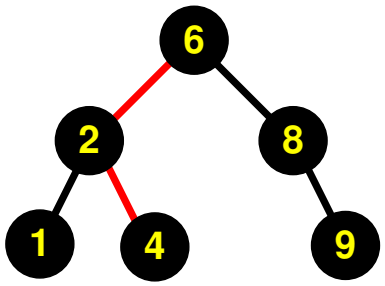
Tetszőleges x csúcsra és az x baloldali részfájában levő y csúcsra igaz, hogy elem(y) ≤ elem(x). Hasonlóan, ha z egy csúcs az x jobb részfájából, akkor elem(x) ≤ elem(z).



Házi feladat: Igazoljuk, hogy egy bináris keresőfa elemeit a fa inorder bejárása *nemcsökkenő sorrendben* látogatja meg.

Egy kényelmes megállapodás: a továbbiakban feltesszük, hogy nincsenek ismétlődő elemek a keresőfában.

Naiv algoritmusok



KERES(4, S)

KERES(s,S): Összehasonlítjuk s-et S gyökerében tárolt s' elemmel.

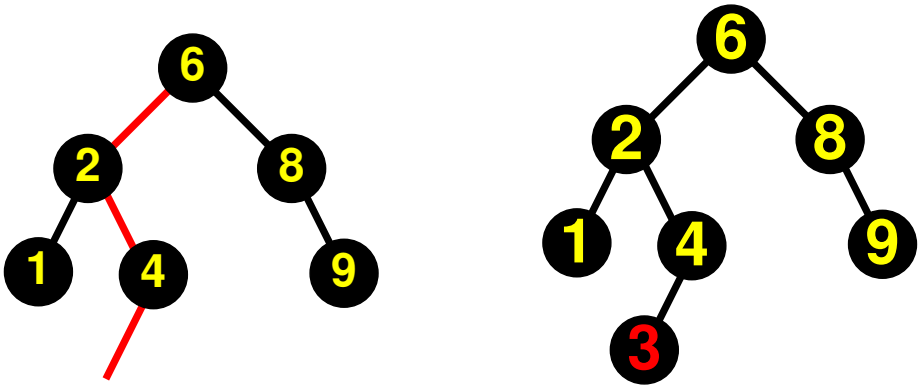
- Ha $s = s'$, akkor megtaláltuk.
- Ha $s < s'$, akkor balra megyünk tovább.
- Ha $s > s'$, akkor jobbra megyünk.

Ugyanezt az utat járjuk be a KERES(5, S) kapcsán, de azt nem találjuk meg.
Lépésszám: $O(l)$, ahol l a fa mélysége
MIN: mindig balra lépünk, amíg lehet
MAX: mindig jobbra lépünk, amíg lehet
Lépésszám: $O(l)$
TÓLIG(a, b, S): KERES(a, S) $\longrightarrow$ INORDER a-tól b-ig
Lépésszám: $O(l + k)$, ahol k az a és b között levő elemek száma

Naiv BESZÚR

BESZÚR(s, S): KERES(s, S)-sel megkeressük, hova kerülne, és új levelet adunk hozzá, pl. BESZÚR(3, S):

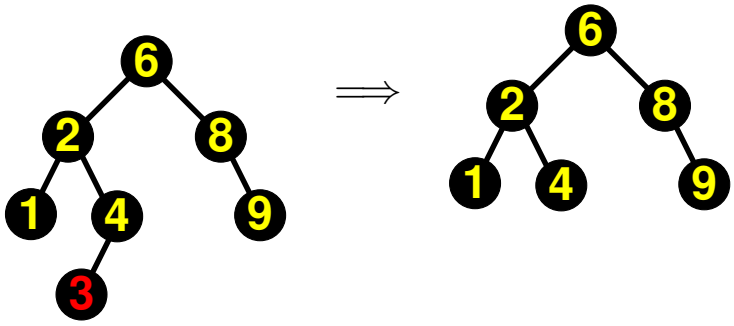
$\Longrightarrow$



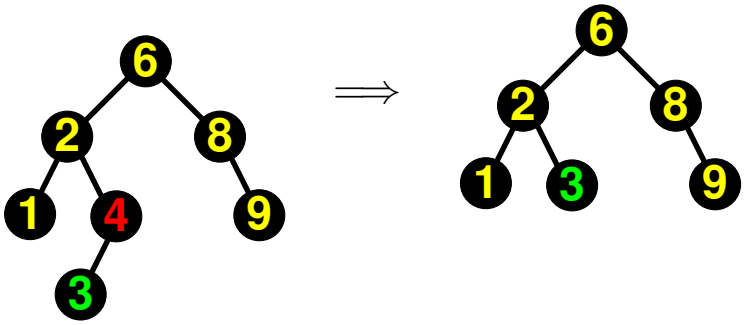
Lépésszám: $O(l)$

Naiv TÖRÖL

- TÖRÖL(s, S): Ha s levél, akkor triviális, pl. TÖRÖL(3, S):

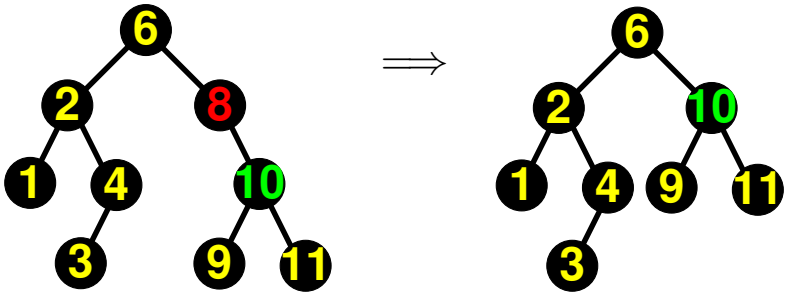


- TÖRÖL(s, S): Ha s-nek egy fia van, akkor: $s \leftarrow$ fiú(s), pl. TÖRÖL(4, S):

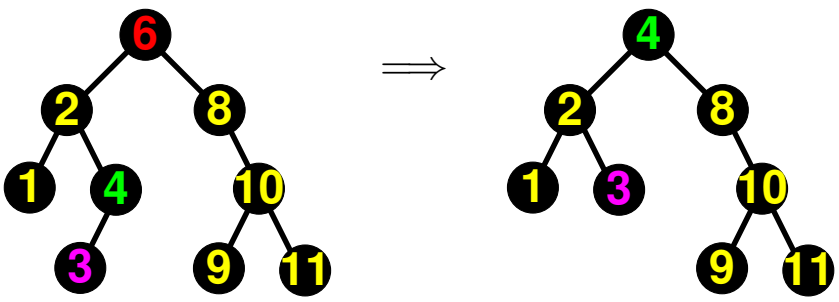


Naiv TÖRÖL

- Vagy pl. TÖRÖL(8, S'):



- TÖRÖL(s, S): Ha s-nek két fia van, akkor visszavezetjük az előző esetre. s helyére tegyük $y := \mathrm{MAX}(bal(s))$-t és töröljük y-t. Pl. TÖRÖL(6, S'):



Naiv TÖRÖL

Állítás

$y := MAX(bal(s))$ csúcsnak nem lehet két fia.

Bizonyítás.

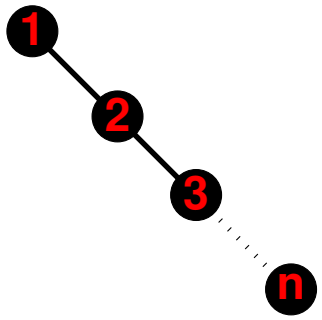
Ha lenne két fia, akkor lenne egy y' jobb fia is. De ekkor $y' > y$.

□

Lépésszám: $O(l)$

Faépítés naiv beszúrásokkal

Ha pl. az $1, 2, \ldots, n$ sorozatból építünk fát így, akkor ezt kapjuk:



Az építés költsége: $2 + 3 + \ldots + (n-1) = O(n^2)$

Tétel

Ha egy véletlen sorozatból építünk fát naiv beszúrásokkal, akkor az építés költsége átlagosan $O(n \log_2 n)$. A kapott fa mélysége átlagosan $O(\log_2 n)$.

Piros-fekete fák

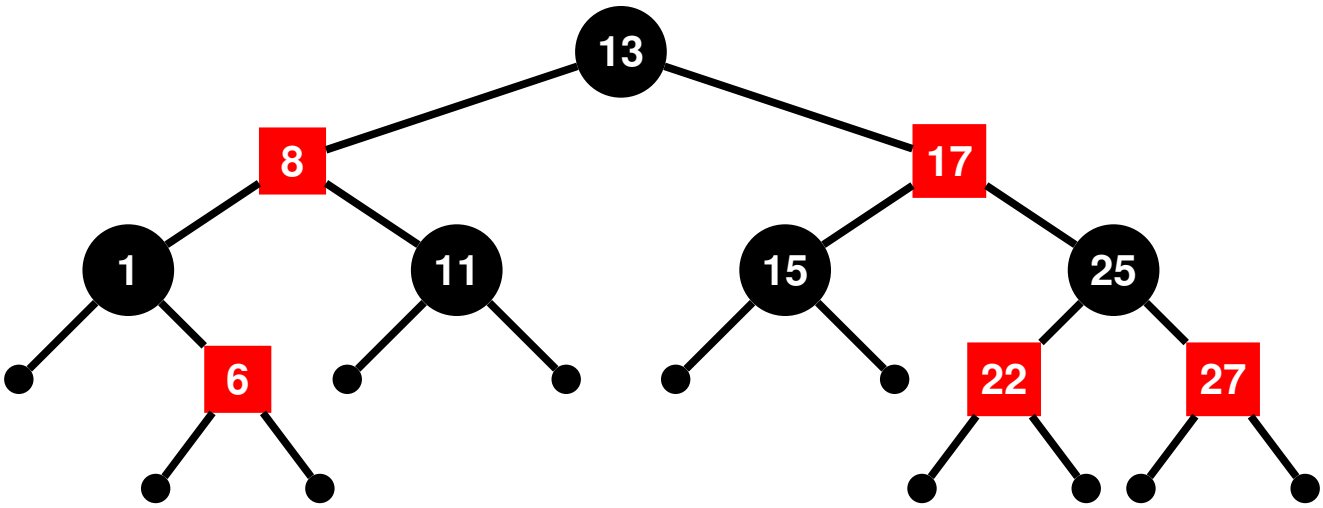
Olyan bináris keresőfa, melynek mélysége nem lehet nagy.
BESZÚR, TÖRÖL, KERES, MIN, (MAX, TÓLIG) hatékonyak.

Definíció

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

1. *Minden nem levél csúcsnak 2 fia van.*
2. *Elemeket belső csúcsokban tárolunk.*
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 az összes v-ből levélbe vezető úton ugyanannyi fekete csúcs van.*

Példa



Megj.: A szokásos bináris fát kiegészítjük üres levelekkel.

Piros-fekete fák

Jelölések

- F_v: v gyökerű részfa
- $m(v)$: v magassága, a leghosszabb v-ből levélbe vezető út éleinek száma
- $fm(v)$: v fekete-magassága, a v-ből levélbe vezető összes úton a fekete csúcsok száma, v-t nem számolva.
  (Ez minden úton egyforma a 8. tulajdonság miatt.)

Tulajdonságok

Állítás

Egy piros-fekete fa minden v csúcsára teljesül

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

Bizonyítás.

A leghosszab levélbe vezető úton a feketék száma nem lehet több az élek számánál $\Longrightarrow fm(v) \leq m(v)$. $\surd$

7. pont miatt a leghosszabb úton a pontoknak legalább a fele fekete $\Longrightarrow \frac{m(v)}{2} \leq fm(v)$. $\surd$ □

Tulajdonságok

Állítás

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

Bizonyítás.

Indukcióval $m(v)$-re: $m(v) = 0 \implies fm(v) = 0, b_v \geq 2^0 - 1$ $\surd$
Ha $m(v) > 0$, akkor legyen x, y a két fia.
$\implies m(x) < m(v)$ és $m(y) < m(v)$
$fm(v) - 1 \leq fm(x) \leq fm(v)$ és $fm(v) - 1 \leq fm(y) \leq fm(v)$
$b_v = b_x + b_y + 1 \implies$
$b_v \geq (2^{fm(x)} - 1) + (2^{fm(y)} - 1) + 1 \geq 2 \cdot (2^{fm(v)-1} - 1) + 1 = 2^{fm(v)} - 1$. $\square$

Tulajdonságok

Állítás

Ha egy piros-fekete fában n elemet tárolunk, akkor a fa magassága $\leq 2 \log(n + 1)$.

Bizonyítás.

Ha r a gyökér $\implies b_r = n$.
$n = b_r \geq 2^{fm(r)} - 1 \implies \log(n + 1) \geq fm(r) \geq \frac{m(r)}{2}$ $\surd$ $\square$

Tétel

KERES, MAX, MIN lépésszáma piros-fekete fában $O(\log n)$.

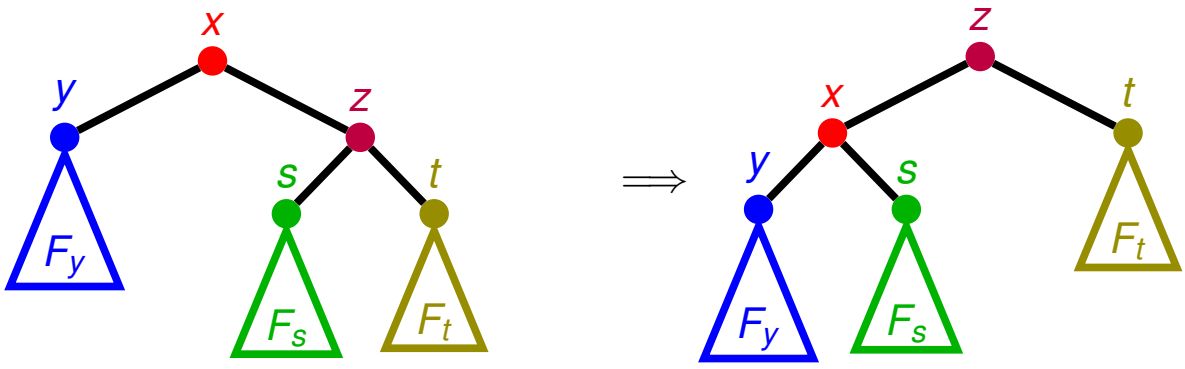
Bizonyítás.

Általában minden keresőfában a lépésszám a fa magasságával arányos $\implies O(l) = O(\log n)$. $\square$

BESZÚR lépésszáma

Ha a keresőfáknál használatos beszúrást használnánk, akkor megsérülhetne a piros-fekete tulajdonság.

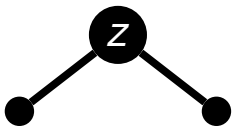
Forgatás

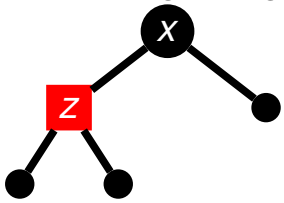


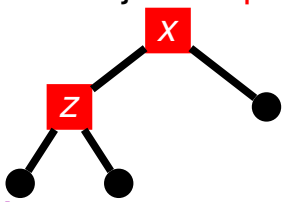
Megj.: Ez a művelet megtartja a keresőfa tulajdonságot.

BESZÚR

Szúrjuk be az új elemet a keresőfáknál megismert módon. ⟹
Új belső csúcs keletkezik (gyerekei csak üres fekete levelek): z

- Ha z a gyökér, akkor legyen fekete ⟹ 

- Ha z nem gyökér, akkor legyen az apja x, ⟹
  z legyen piros.
  (1) Ha x fekete ⟹ fekete-magasságok sehol nem változnak $\surd$ ⟹ 

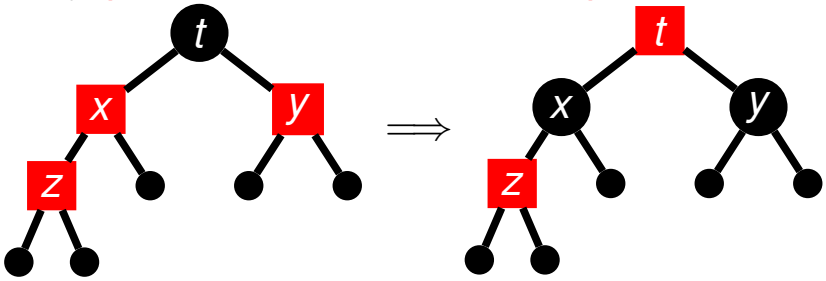
  (2) Ha x piros ⟹ nem teljesül a piros-fekete tulajdonság ⟹ 

  ⟹ további lépések kellenek.

BESZÚR

(2) Mivel x piros, nem gyökér $\Longrightarrow$
legyen x apja t (fekete), x testvére y.

(2.1) Ha y piros $\Longrightarrow$ átszínezzük t-t pirosra

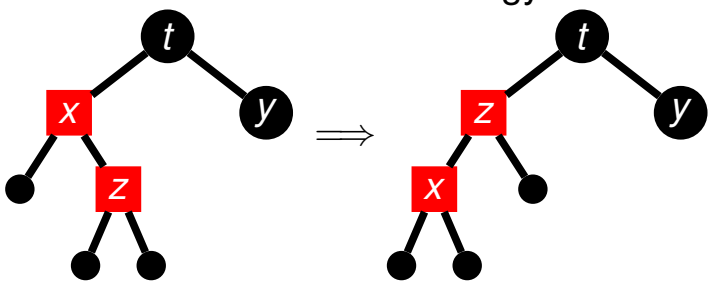


Evvel a problémát két szinttel feljebb toltuk, ott folytatjuk a fa rendbetételét.
Kivéve, ha t a gyökér $\Longrightarrow$ t marad fekete $\Longrightarrow$ $fm(t)$ eggyel nagyobb lesz.

BESZÚR

(2.2) Ha y fekete:

(2.2.1) Ha z és x nem azonos oldali gyerek $\Longrightarrow$ forgatunk x körül.

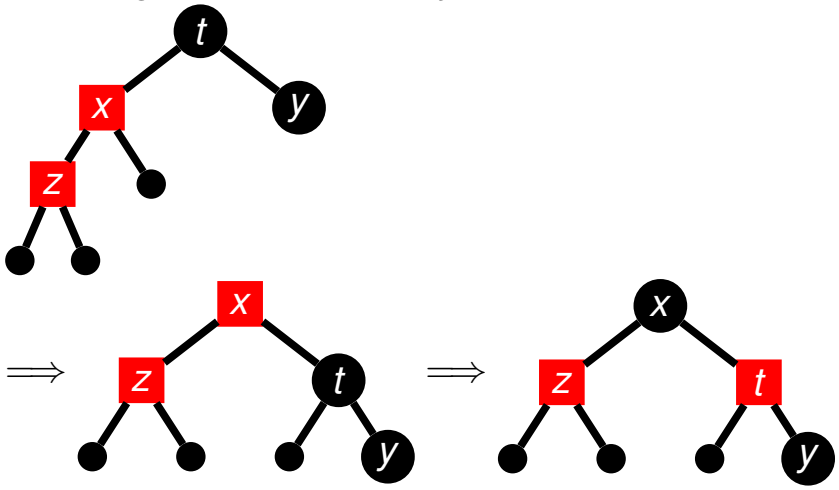


Evvel a következő esetre vezettük a problémát.

BESZÚR

(2.2) Ha y fekete:

(2.2.2) Ha z és x azonos oldali gyerek
$\Longrightarrow$ forgatunk t körül, majd átszínezünk.



Evvel a gyökér fekete-magassága nem változik, és teljesül a piros-fekete tulajdonság. $\surd$

BESZÚR

Tétel

A BESZÚR során

(a) *a lépésszám $O(\log n)$,*

(b) *legfeljebb 2 forgatás történik.*

Bizonyítás.

(a) y piros esetben a (2.1) pontban 2 szinttel feljebb kerül a baj $\Longrightarrow$ szintenként konstans lépés $\Longrightarrow O(\log n)$. $\surd$

(b) Forgatás csak a (2.2) esetben történik, de ekkor nincs felgyűrűzés, rögtön kijavítjuk a fát. $\surd$

$\square$

TÖRÖL

Hasonló módszerek, de bonyolultabb.

Tétel

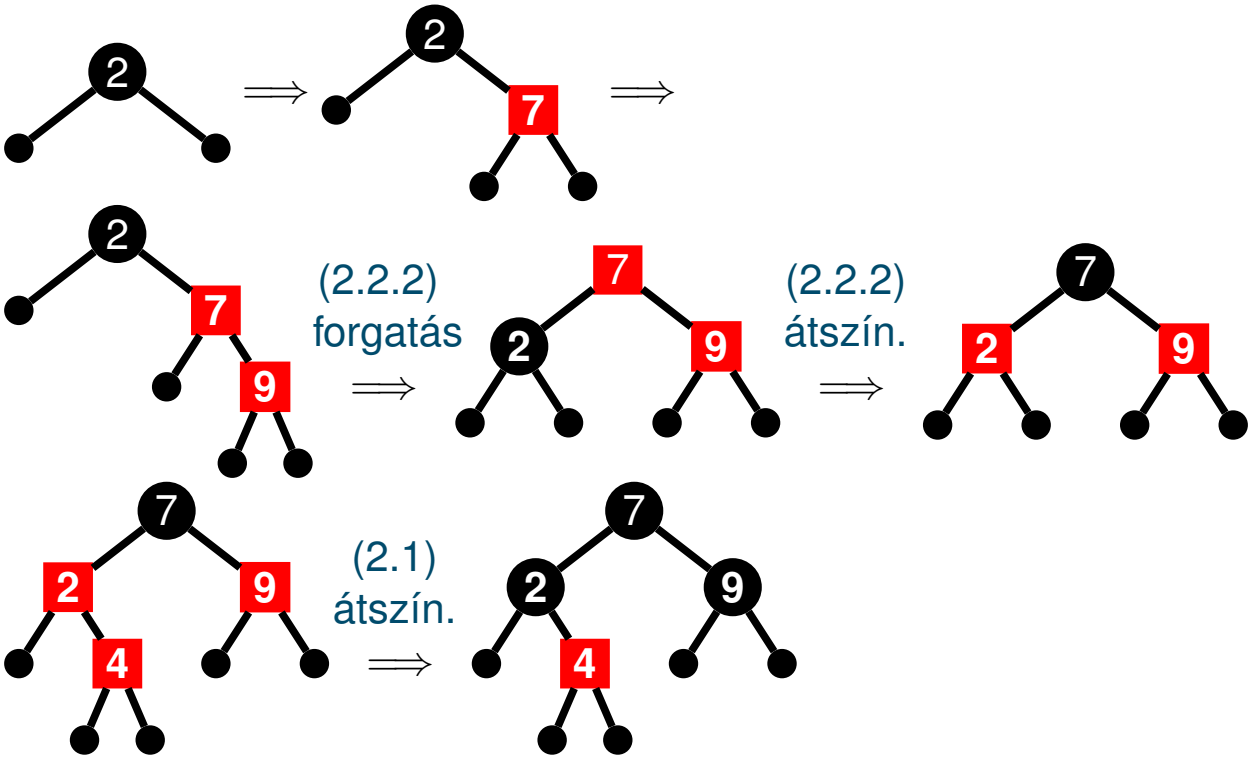
A TÖRÖL során

(a) *a lépésszám* $O(\log n)$,

(b) *legfeljebb 3 forgatás történik.*

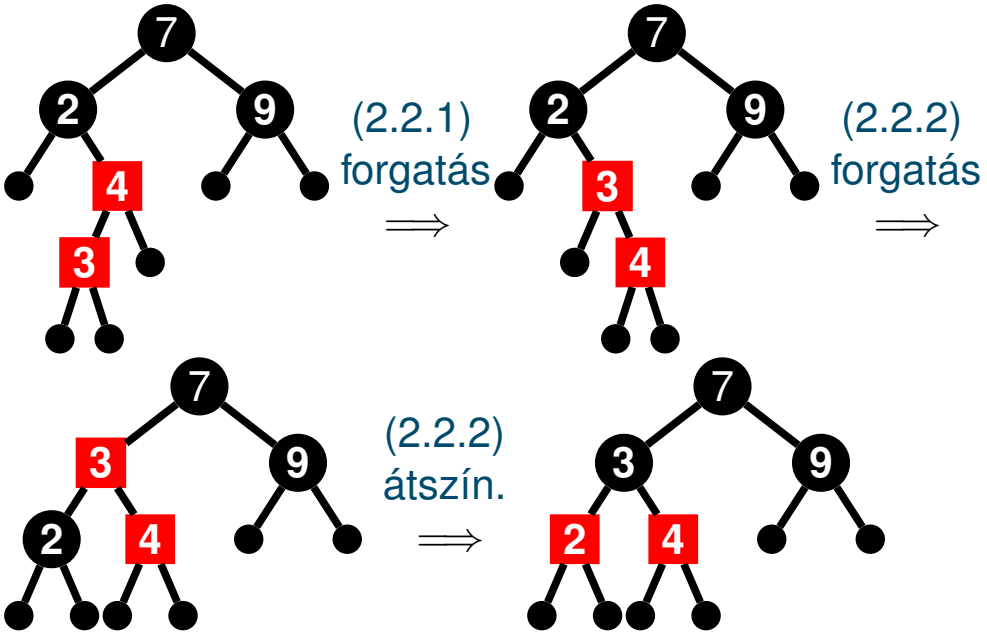
Példa BESZÚRásokra

Szúrjuk be egy üres fába sorban a 2, 7, 9, 4, 3, 1 elemeket.



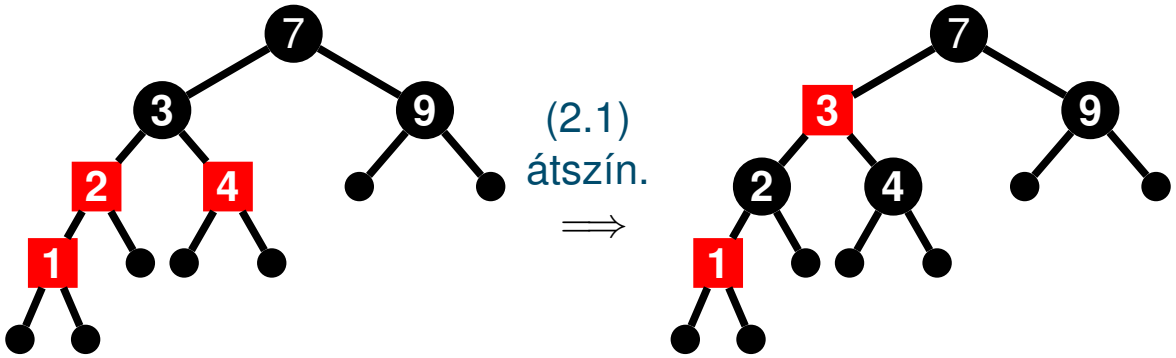
Példa BESZÚRásokra

Szúrjuk be egy üres fába sorban a $2, 7, 9, 4, 3, 1$ elemeket.



Példa BESZÚRásokra

Szúrjuk be egy üres fába sorban a $2, 7, 9, 4, 3, 1$ elemeket.



Algoritmuselmélet

2-3 fák

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

8. előadás

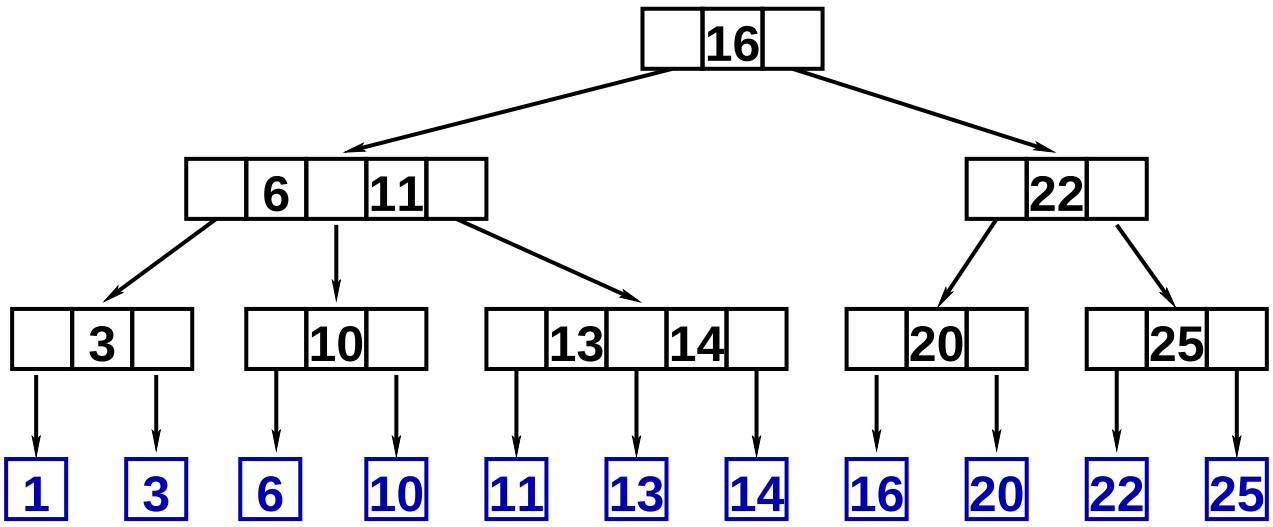
2-3-fák

Ez is fa, de a binárisnál bonyolultabb: egy nem-levél csúcsnak 2 vagy 3 fia lehet.

A **2-3-fa** egy (lefelé) irányított gyökeres fa, melyre:

- A rekordok a fa leveleiben helyezkednek el, a kulcs értéke szerint balról jobbra növekvő sorrendben. Egy levél egy rekordot tartalmaz.
- Minden belső (azaz nem levél) csúcsból 2 vagy 3 él megy lefelé; ennek megfelelően a belső csúcsok egy, illetve két $s \in U$ kulcsot tartalmaznak. A belső csúcsok szerkezete tehát kétféle lehet. Az egyik típus így ábrázolható: | m_1 | s_1 | m_2 | s_2 | m_3 |
  Itt m_1, m_2, m_3 mutatók a csúcs részfáira, s_1, s_2 pedig U-beli kulcsok, melyekre $s_1 < s_2$. Az m_1 által mutatott részfa minden kulcsa kisebb, mint s_1, az m_2 részfájában s_1 a legkisebb kulcs, és minden kulcs kisebb, mint s_2. Végül m_3 részfájában s_2 a legkisebb kulcs. A másik típusú csúcsoknál az az utolsó két mező hiányzik:
  | m_1 | s_1 | m_2 |
- A fa levelei a gyökértől egyforma távolságra vannak.

Példa 2-3-fára



2-3-fa tulajdonságai

Tétel

Ha a fának m szintje van, akkor a levelek száma legalább 2^{m-1}*. Megfordítva, ha* $|S| = n$ *(itt* $S \subseteq U$ *a fában tárolt kulcsok halmaza;* $|S|$ *megegyezik a tárolt rekordok számával), akkor* $m \leq \log_2 n + 1$*.*

Bizonyítás.

Minden belső csúcsnak legalább 2 fia van $\Longrightarrow$
az i-edik szinten legalább 2^{i-1} csúcs van $(1 \leq i \leq m)$. $\Longrightarrow$
$2^{m-1} \leq n \Longrightarrow m - 1 \leq \log_2 n$. $\surd$ □

2-3-fa tulajdonságai

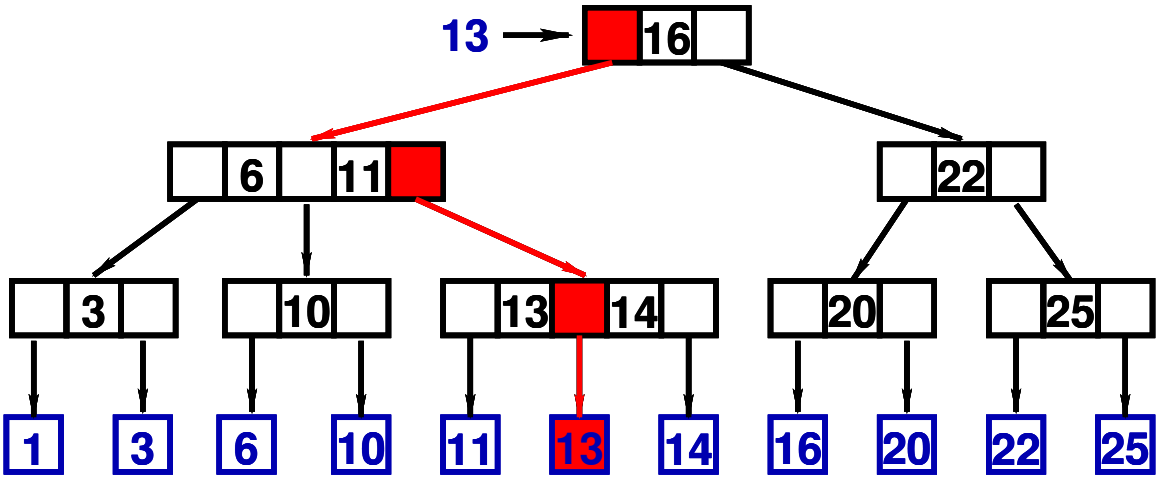
Tétel

Ha a fának m szintje van, akkor a levelek száma legfeljebb 3^{m-1}*.*
Megfordítva, $m \geq \log_3 n + 1$*.*

Bizonyítás.

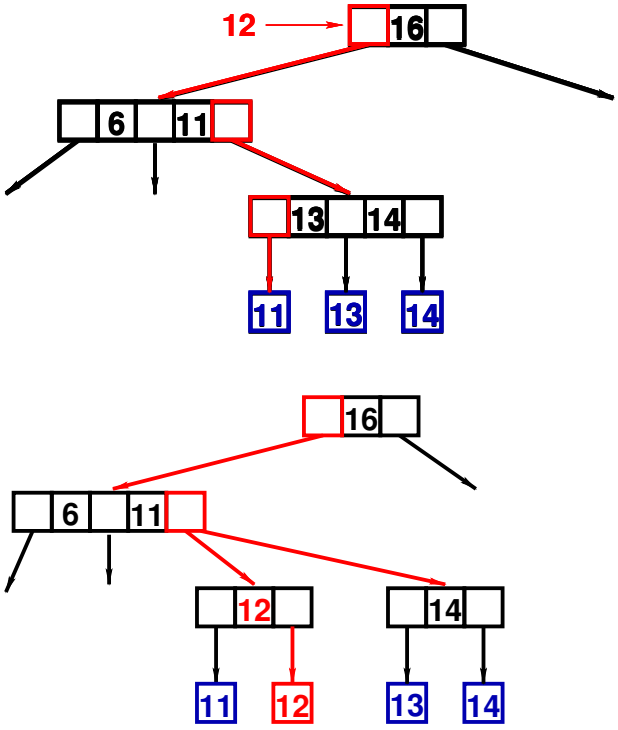
Minden belső csúcsnak legfeljebb 3 fia van $\Longrightarrow$
az i-edik szinten legfeljebb 3^{i-1} csúcs van $(1 \leq i \leq m)$. $\Longrightarrow$
$n \leq 3^{m-1} \Longrightarrow m - 1 \geq \log_3 n$. $\surd$ □

Keresés 2-3-fában



Hasonló, mint a bináris keresőfában.
Lépésszám: $O(m)$, ahol $\log_3(n) + 1 \leq m \leq \log_2(n) + 1$

BESZÚR 2-3-fába



Ha a gyökeret is „vágni” kell $\Longrightarrow$ új gyökér, nő a fa magassága.
Lépésszám: $O(m)$, minden szinten legfeljebb 1 „vágás”.

TÖRÖL 2-3-fából

Legyen x a legalsó belső csúcs a kereső út mentén.

- Ha x-nek három fia van $\Longrightarrow$ $\surd$
- Ha x-nek csak két fia van:
  - ha x (valamelyik) szomszédos testvérének 3 fia van $\Longrightarrow$ egyet átteszünk x alá;
  - ha x egyik szomszédos testvérének sincs három fia $\Longrightarrow$ „összevonunk” két kettes csúcsot.

Ez is „felgyűrűzhet”. $\Longrightarrow$

Lépésszám: $O(m)$

B-fák

R. Bayer, E. McCreight, 1972

A 2-3-fa általánosítása.

Nagy méretű adatbázisok, külső táron levő adatok feldolgozására használják. Több szabvány tartalmazza valamilyen változatát.

Probléma

Nem az összhasonlítás időigényes, hanem az adatok kiolvasása, de sokszor egy adat kiolvasásához amúgy is kiolvasunk több más adatot, egy lapot.

$\Longrightarrow$ A fa csúcsai legyenek lapok, a költség a lapelérések száma.

B-fa definíciója

Egy *m-edrendű B-fa*, röviden B_m-fa egy gyökeres, (lefelé) irányított fa, melyre érvényesek az alábbiaknak:

a) A gyökér foka legalább 2, kivéve esetleg, ha a fa legfeljebb kétszintes.

b) Minden más belső csúcsnak legalább $\lceil \frac{m}{2} \rceil$ fia van.

c) A levelek a gyökértől egyforma messze vannak.

d) Egy csúcsnak legfeljebb m fia lehet.

d) A tárolni kívánt rekordok itt is a fa leveleiben vannak; egy levélben a lapmérettől és a rekordhossztól függően több rekord is lehet, növekvően rendezett láncolt listában.

A belső csúcsok hasonlítanak a 2-3-fák belső csúcsaira. Egy belső csúcs így néz ki:

m_0	s_1	m_1	s_2	m_2	$\ldots$	s_i	m_i

A B-fa szintszáma

Tegyük fel, hogy egy B-fának n levele és k szintje van, és keressünk összefüggést e két paraméter között.
A kicsi fáktól eltekintve a gyökérnek legalább két fia van, a többi belső csúcsnak pedig legalább $\lceil \frac{m}{2} \rceil$.
$\Longrightarrow n \geq 2\lceil \frac{m}{2} \rceil^{k-2}$, $\Longrightarrow \log_{\lceil \frac{m}{2} \rceil} \frac{n}{2} + 2 \geq k$

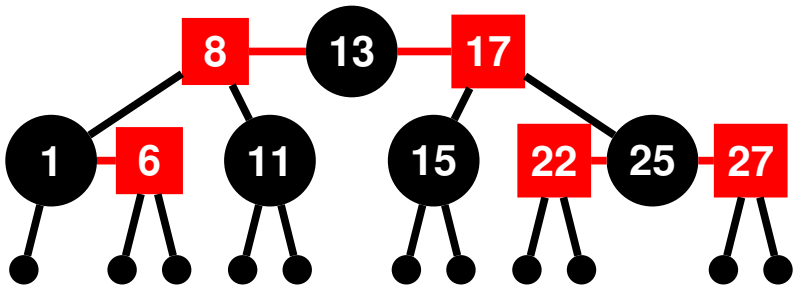
$$k \leq \frac{\log_2 n - 1}{\log_2 \lceil \frac{m}{2} \rceil} + 2.$$

Minden művelet lépésszáma: $\sim \frac{\log_2 n - 1}{\log_2 \lceil \frac{m}{2} \rceil} = \Theta\left(\frac{\log n}{\log m}\right)$, azaz a konstans szorzó kicsi, ha m nagy.
m viszont nem lehet túl nagy, hiszen a belső csúcsoknak egy lapon el kell férniük.

Például: Például, ha $m = 32$, $n = 2^{20}$ (itt n az alsó szint *lapjainak* száma), akkor $k \leq \frac{19}{4} + 2 < 7$. Egy rekord keresése tehát legfeljebb 6 lap elérését igényli.

A piros-fekete fa és a B-fa kapcsolata

A piros-fekete fa olyan B_4-fa, aminek a belső csúcsaiban tároljuk az elemeket.



A piros csúcsokat összevonjuk apjukkal, az így összevont csúcsoknak 2, 3 vagy 4 gyereke van.
Ezért a mélység csak a fekete csúcsokkal nő. $\Longrightarrow$ Mivel a fekete magasság állandó, minden levél azonos szinten lesz.

Algoritmuselmélet

Hashelés

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

9. előadás

Hashelés

Nem tételezzük fel a lehetséges kulcsok összességének (az U univerzumnak) a rendezettségét.
Olyan módszercsalád, amely a keresés, beszúrás, törlés és módosítás gyors és egyszerű megvalósítását teszi lehetővé.
Nincs rendezés $\Longrightarrow$ nincs MIN, MAX, ...
Cél: $S \subseteq U$ kulcshalmazzal azonosított állomány megszervezése úgy, hogy a fenti műveletek átlagos értelemben hatékonyak legyenek.

Példa: Magyar állampolgárok személyi nyilvántartása
$\Longrightarrow$ kulcs= 11 jegyű személyi szám
Lehetséges személyi számok: $4 \cdot 10^2 \cdot 12 \cdot 31 \cdot 10^3 \approx 148$ millió darab.
Elég lefoglalni 11 millió rekordnak helyet.
Olyan h függvény kell, ami minden személyi számhoz rendel egy egészet a $[0, 12 \cdot 10^6 - 1]$ intervallumból.
Jó lenne ha, $K \neq K'$ esetén $h(K) \neq h(K')$ teljesülne, de ez nem lehetséges. $\Longrightarrow$ ütközések elkerülhetetlenek

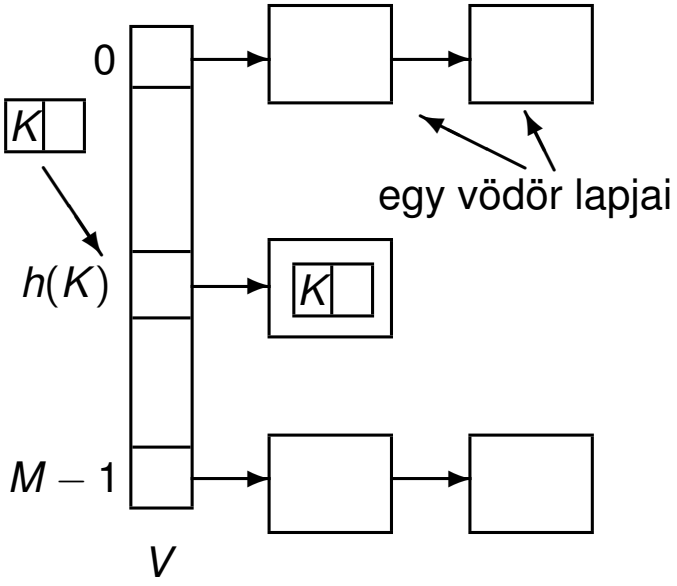
Hashelés alapvető ötelete

Veszünk egy alkalmas h hash-függvényt, elsőnek a K kulcsú elemet a $h(K)$ cellába próbáljuk illeszteni.
Ha később érkezik egy K' elem, amire $h(K) = h(K')$, akkor ütközés van.
Az ütközések feloldására több módszer is van, próbálunk más helyet találni K'-nek.

Fontos kérdés a megfelelő hash függvény kiválasztása is, pl. $h(K) = konst.$ nyilván nem praktikus.

Vödrös hashelés

Főleg külső táron tárolt, nagy állományok kezelésére.
Minden elemet, amelyre $h(K) = i$ beteszünk $V(i)$-be, ha több ilyen is van, láncolt listaként.
$V[0 : M-1]$ vödörkatalógus, $V[i]$ mutató egy vödörbe, amiben az elemek listái (lapláncai) vannak. A vödrök mérete általában kicsi.



Kulcsok a vödörben

Hogyan helyezzük az új kulcsot a vödörbe?

- Az első szabad helyre tesszük, ha kell, új lappal bővítünk (az elején).
- Kulcs szerint rendezve vannak, beszúráskor a helyére tesszük.

Keresés a hash-táblában

- Kiszámítjuk $h(K)$-t.
- A $V[h(K)]$ vödörben keresünk szekvenciálisan, addig megyünk, amíg megtaláljuk, vagy véget ér.

Törlés ugyanígy.

Hashelés költsége

Külső táras szerkezet $\Longrightarrow$ lapelérések száma.
M vödör van, és l-lapnyi rekordot tárolunk
$\Longrightarrow$ egy vödörbe átlagosan $\approx l/M$ lap kerül
$\Longrightarrow$ átlagos lánchossz: l/M
$\Longrightarrow$ Sikeres keresés átlagos lépésszáma: $1 + l/2M$
$\Longrightarrow$ Sikertelen keresés átlagos lépésszáma: $1 + l/M$

Hogyan válasszuk meg M-et?
l/M legyen kb. 1, de hagyjunk rá 20%-ot.

Példa: 1 000 000 rekordból álló állományt szeretnénk láncolásos módszerrel kezelni, egy lapon 5 rekord fér el.
Ekkor $l = 1\,000\,000/5 = 200\,000 \Longrightarrow M \approx 220\,000 - 240\,000$
$\Longrightarrow$ keresés átlagos költsége valamivel 2 lapelérés alatt marad.

Hashelés nyitott címzéssel

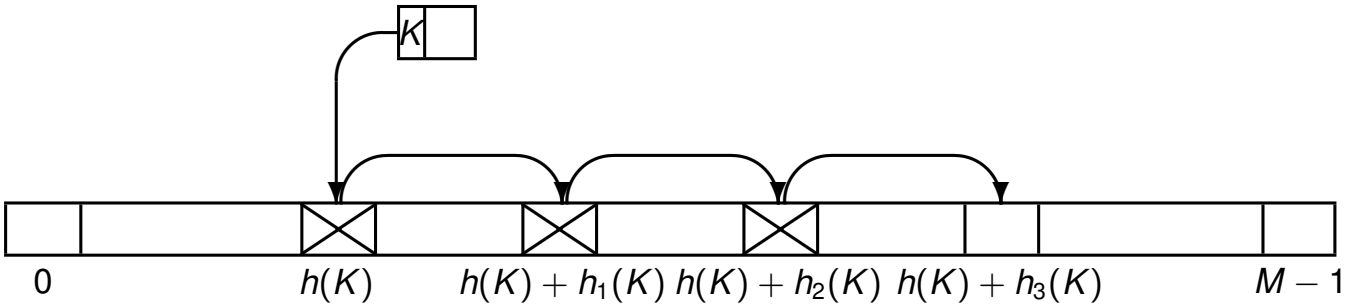
Csak belső memóriás módszerként hasznosak.

Fő ötlet: ha $h(K)$ már foglalt, keresünk egy üreset valamilyen módszerrel.

Legyen $0, h_1(K), h_2(K), \ldots, h_{M-1}(K)$ a $0, 1, \ldots, M-1$ számok egy permutációja.

$\Longrightarrow$ Végigpróbálgatjuk a $h(K) + h_i(K) \pmod M$ sorszámú cellákat ($i = 0, 1, \ldots, M-1$) az első üres helyig, ahol a rekordot elhelyezzük.

$\Longrightarrow$ Ha nincs üres, a tábla betelt.



Lineáris próbálás

$h_i(K) := -i$

Visszafelé lépkedünk egyesével $h(K)$-tól indulva az első üres helyig.

Sikeres keresés átlagos költsége:

$$C_N = \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$$

Sikertelen keresés átlagos költsége:

$$C'_N = \frac{1}{2}\left(1 + \left(\frac{1}{1-\alpha}\right)^2\right)$$

ahol $\alpha = N/M$ – a telítettségi (betöltöttségi) tényező, N – a táblában levő rekordok száma, M – a tábla celláinak száma.

α	$2/3$	$0,8$	$0,9$
C_N	2	3	$5,5$
C'_N	5	13	$50,5$

Lineáris próbálás

Példa: $M = 7, h(K) := K \pmod 7$, lineáris próba,
beillesztendő: $3, 11, 9, 4, 10$.

0	1	2	3	4	5	6
10	4	9	3	11		

Ha most töröljük a 9-et, akkor később nem találnánk meg a 4-et.
$\Longrightarrow$ 9 helyére egy speciális TÖRÖLT jelet pl. $*$-ot teszünk. $\Longrightarrow$

0	1	2	3	4	5	6
10	4	$*$	3	11		

Lineáris próba hátránya: Ha már sok cella tele van, kialakulnak egybefüggő csomók, megnő a keresési, beillesztési út.
$\Longrightarrow$ *elsődleges csomósodás*

Hashelés álvéletlen próbával

A $0, h_1(K), h_2(K), \ldots, h_{M-1}(K)$ próbasorozat a $0, 1, \ldots, M-1$ számoknak egy a K kulcstól független álvéletlen permutációja.
A sorozatnak gyorsan és hatékonyan reprodukálhatónak kell lennie, ezért nem lehet „valódi” véletlent használni.
Ha $h(K) = h(L)$, akkor a K és L kulcsok teljes próbasorozata is megegyezik. $\Longrightarrow$ másodlagos csomósodás

Kvadratikus maradék próba

Legyen M egy $4k+3$ alakú prímszám, ahol k egy egész.
Ekkor a próbasorozat legyen

$$0, 1^2, -(1^2), 2^2, -(2^2), \ldots, \left(\frac{M-1}{2}\right)^2, -\left(\frac{M-1}{2}\right)^2.$$

Belátjuk, hogy ez tényleg permutáció.

Hashelés kvadratikus próbával

Lemma

Ha M egy $4k+3$ alakú prímszám, akkor nincs olyan n egész, melyre $n^2 \equiv -1 \pmod{M}$.

Bizonyítás.

Indirekt tegyük fel, hogy n egy egész szám és $n^2 \equiv -1 \pmod{M}$. $\Longrightarrow$

$$-1 = (-1)^{\frac{M-1}{2}} \equiv n^{2\frac{M-1}{2}} = n^{M-1} = n^{\varphi(M)} \equiv 1 \pmod{M}.$$

Az utolsó lépésnél az Euler-Fermat-tételt használtuk.  □

Hashelés kvadratikus próbával

Ha $0 \leq i < j \leq \frac{M-1}{2}$, akkor $i^2 \not\equiv j^2 \pmod{M}$. $\Longleftarrow j^2 - i^2 = (j-i)(j+i)$ felbontás egyik tényezője sem lehet osztható M-mel, tehát a szorzatuk sem. $\surd$

Ugyanígy $\Longrightarrow -i^2 \not\equiv -j^2 \pmod{M}$. $\surd$

A lemma miatt $(ij^{-1})^2 \not\equiv -1 \pmod{M}$, ahol j^{-1} a j elem inverze a $\pmod{M}$ maradékosztályok csoportjában a szorzásra nézve.

$\Longrightarrow i^2 \not\equiv -j^2 \pmod{M}$ $\surd$

Hashelés kvadratikus próbával

Sikeres keresés költsége:

$$C_N \approx 1 - \log(1-\alpha) - \frac{\alpha}{2}$$

Sikertelen keresés költsége:

$$C'_N \approx \frac{1}{(1-\alpha)} - \alpha - \log(1-\alpha)$$

Ezek az összefüggések valamivel általánosabban érvényesek az olyan módszerekre, amelyekre $h_i(K) = f_i(h(K))$, vagyis ahol a $h(K)$ érték már az egész próbasorozatot meghatározza.

Kettős hashelés

G. de Balbine, J. R. Bell, C. H. Kaman, 1970 körül.
Lényeg: h mellett egy másik h' hash-függvényt is használunk de azt csak a próbasorozat előállításához.
A $h'(K)$ értékek relatív prímek legyenek az M táblaméréthez.
A K kulcs próbasorozata: $h_i(K) := -i \cdot h'(K)$.
Ha M és $h'(K)$ relatív prímek
$\Longrightarrow 0, -h'(K), -2h'(K), \ldots, -(M-1)h'(K)$ sorozat elemei mind különbözők modulo M.
Fontos sajátossága: különböző K és K' kulcsok próbasorozatai jó eséllyel akkor is különbözők lesznek, ha $h(K) = h(K')$.

Kettős hashelés

A legjobb ismert implementációk időigénye (empirikus adatok alapján)

$$C_N \approx \frac{1}{\alpha} \log \frac{1}{(1-\alpha)} \quad \text{és} \quad C'_N \approx \frac{1}{1-\alpha}.$$

- A kettős hashelés kiküszöböli mindkétféle csomósodást.
- Sikertelen keresés esetén minden érdekes α-ra gyorsabb, mint a lineáris próbálás.
- Sikeres kereséskor csak az $\alpha \geq 0,8$ tartományban lesz gyorsabb a lineáris próbálásnál.

Hash-függvények

- Legyen könnyen (gyorsan) számítható,
- és minél kevesebb ütközést okozzon.

A második követelmény elég nehezen megfogható, mert a gyakorlatban előforduló kulcshalmazok egyáltalán nem véletlenszerűek.
Hasznos tanácsok: $h(K)$ értéke lehetőleg a K kulcs minden bitjétől függjön és a h értékkészlete a teljes $[0, M-1]$ címtartomány legyen.

Két gyakran használt módszer hash-függvény előállítására az osztó- és a szorzómódszer.

Osztómódszer

Legyen $h(K) := K \pmod M$,
ahol M a tábla vagy a vödörkatalógus mérete.
Feltesszük, hogy a kulcsok egész számok.
A $h(K)$ számítása gyors és egyszerű.
A tábla mérete sem teljesen közömbös.
Például ha M a 2 egy hatványa, akkor $h(K)$ csak a kulcs utolsó néhány bitjétől függ.
A jó M értékeket illetően van egy széles körben elfogadott recept:
D. E. Knuth javaslata: M-et prímnek választjuk, úgy, hogy M nem osztja $r^k \pm a$-t, ahol r a karakterkészlet elemszáma (pl. 128, vagy 256) és $a,\ k$ „kicsi" egészek.
M prím:

- Lényeges feltétel a kvadratikus maradék próbánál.
- Kettős hashelésnél könnyű hozzájuk relatív prím számot találni .

Szorzómódszer

β egy rögzített paraméter.

$$h(K) := \lfloor M \cdot \{\beta K\} \rfloor.$$

$\{x\}$ jelöli az x valós szám törtrészét.
Szemléletesen: $\{\beta K\}$ kiszámításával a K kulcsot „véletlenszerűen" belőjük a $[0, 1)$ intervallumba, majd az eredményt felskálázzuk a címtartományba.
Hatékonyan számítható speciális eset:
$M = 2^t$, $w = 2^{32}$, és legyen A egy a w-hez relatív prím egész.
Ekkor $\beta = \frac{A}{w}$ választás mellett $h(K)$ igen jól számolható.
A számok bináris ábrázolásával dolgozva lényegében egy szorzást és egy eltolást kell elvégezni.
A szorzómódszer jól viselkedik számtani sorozatokon.
pl. termék1, termék2, termék3, . . . esetében.
Megmutatható, hogy a $h(K), h(K + d), h(K + 2d) \ldots$ sorozat közelítőleg számtani sorozat lesz, azaz h jól „szétdobja" a kulcsok számtani sorozatait.

Szorzómódszer

Tétel (T. Sós Vera, 1957)

Legyen β irracionális szám, és nézzük a $0, \{\beta\}, \{2\beta\}, \ldots, \{n\beta\}$ pontok által meghatározott $n+1$ részintervallumot $[0,1)$-ben. Ezek hosszai legfeljebb 3 különböző értéket vehetnek fel, és $\{(n+1)\beta\}$ a leghosszabbak egyikét fogja két részre vágni.

Következmény: A szorzómódszer esetén mindig elég egyenletesen lesznek szétszórva a hashértékek (ha a bemenet egy számtani sorozat).

A $[0,1)$-beli számok közül a legegyenletesebb eloszlást a $\beta = \phi^{-1} = \frac{\sqrt{5}-1}{2} = 0.618033988\ldots$ és a $\beta = \phi^{-2} = 1 - \phi^{-1}$ értékek adják.
$\Longrightarrow$ Érdemes a szorzómódszernél az A-t úgy választani, hogy $\frac{A}{w}$ közel legyen ϕ^{-1}-hez. $\Longrightarrow$ *Fibonacci-hashelés*

A kettős hashelés második függvénye

Olyan h' függvény kell, melynek értékei a $[0, M-1]$ intervallumba esnek, és relatív prímek az M-hez.
Ha M prím $\Longrightarrow$

$$h'(K) := K \pmod{M-1} + 1.$$

$\Longrightarrow$ $h'(K)$ és M relatív prímek.
Mivel $h'(K)$ minden értéket felvesz 1 és $M-1$ között, ezért elég sok különböző próbasorozatot ad.

Algoritmuselmélet

Mélységi keresés és alkalmazásai

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

10. előadás

Mélységi bejárás

Gráf bejárás $\Longrightarrow$ minden pontot felsorolunk, bejárunk.
Szélességi bejárás (breadth-first-search, BFS) már volt, ott „széltében” haladtunk.
Most mélységben haladunk:
Mélységi bejárás (depth-first-search, DFS, backtrack)
Pl. a lámpagyújtogató algoritmusa

Mélységi keresés

Mohó menetelés, addig megyünk előre, amíg tudunk, csak aztán fordulunk vissza.
$G = (V, E)$ egy irányított gráf, ahol $V = \{1, \ldots, n\}$.
$L[v]$ a v csúcs éllistája ($1 \leq v \leq n$).
bejárva$[1 : n]$ logikai vektor $\Longrightarrow$ jártunk-e már ott.

Mélységi keresés algoritmusa

```
procedure mb (v: csúcs) (* egy komponens bejárása *)
    var
        w: csúcs;
    begin
(1)     bejárva[v] := igaz; (* meggyújtjuk a lámpát *)
        for minden L[v]-beli w csúcsra do
(2)         if bejárva[w] = hamis then
              mb(w) (* megyünk a következő még sötét lámpához *)
    end
```


Mélységi keresés algoritmusa

```
procedure bejár (* minden komponens bejárása *)
    begin
        for v := 1 to n do
            bejárva[v] := hamis;
        for v := 1 to n do
            if bejárva[v] = hamis then
              mb(v)
    end
```

Lépésszám: $O(n+e)$, (ahol n a gráf pontszáma, e pedig az élszáma.)

Mélységi számok és befejezési számok

Definíció (mélységi számozás)

A G irányított gráf csúcsainak egy mélységi számozása a gráf v csúcsához azt a sorszámot rendeli, mely megadja, hogy az mb eljárás (1) sorában a csúcsok közül hányadikként állítottuk bejárva[v] értékét igaz-ra. A v csúcs mélységi számát mszám[v] jelöli.

Definíció (befejezési számozás)

A G irányított gráf csúcsainak egy befejezési számozása a gráf v csúcsához azt a sorszámot rendeli, mely megadja, hogy a csúcsok közül hányadikként ért véget az mb(v) hívás. A v csúcs befejezési számát bszám[v] jelöli.

Előző algoritmus kis módosítással:


```
procedure (* minden komponens bejárása *)
    begin  msz := 0;  bsz := 0;
        for v := 1 to n do
            bejárva[v] := hamis;  mszám[v] := 0;  bszám[v] := 0;
        for v := 1 to n do
            if bejárva[v] = hamis then
              mb(v)
    end

procedure mb (v: csúcs) (* egy komponens bejárása *)
    var
        w: csúcs;
    begin
(1)     bejárva[v] := igaz;  msz := msz + 1;  mszám[v] := msz;
        for minden L[v]-beli w csúcsra do
(2)         if bejárva[w] = hamis then
              mb(w);
          bsz := bsz + 1;  bszám[v] := bsz;
    end
```

Mélységi feszítőerdő

Definíció (faél)

A $G = (V, E)$ irányított gráf $v \to w$ éle faél az adott mélységi bejárásra vonatkozóan, ha megvizsgálásakor a (2) sorban a (bejárva[w] = hamis) feltétel teljesül.

Jelölje T azt a gráfot, amelynek csúcshalmaza V, élei pedig a faélek. Könnyen látható, hogy ez erdő.

Definíció (mélységi feszítőerdő, feszítőfa)

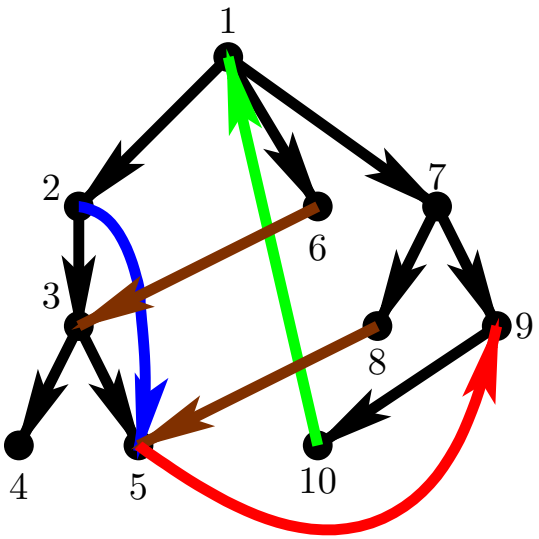
Az előbb meghatározott T gráfot a G gráf egy mélységi feszítőerdejének nevezzük. Ha T csak egy komponensből áll, akkor mélységi feszítőfáról beszélünk.

Élek osztályozása

Definíció (élek osztályozása)

Tekintsük a G irányított gráf egy mélységi bejárását és a kapott T mélységi feszítőerdőt. Ezen bejárás szerint G egy $x \to y$ éle

faél,	*ha $x \to y$ éle T-nek;*
előreél,	*ha $x \to y$ nem faél, y leszármazottja x-nek T-ben és $x \neq y$;*
visszaél,	*ha x leszármazottja y-nak T-ben (a hurokél is ilyen);*
keresztél,	*ha x és y nem leszármazottai egymásnak.*



faél
előreél
visszaél
keresztl
ilyen nincs

faél, előre él:
Kisebb mélységi számúból nagyobb mélységi számúba mutat.
visszaél, keresztél:
Nagyobb mélységi számúból kisebb mélységi számúba mutat.
visszaél:
Kisebb befejezési számúból nagyobb befejezési számúba mutat.

Élek osztályozása

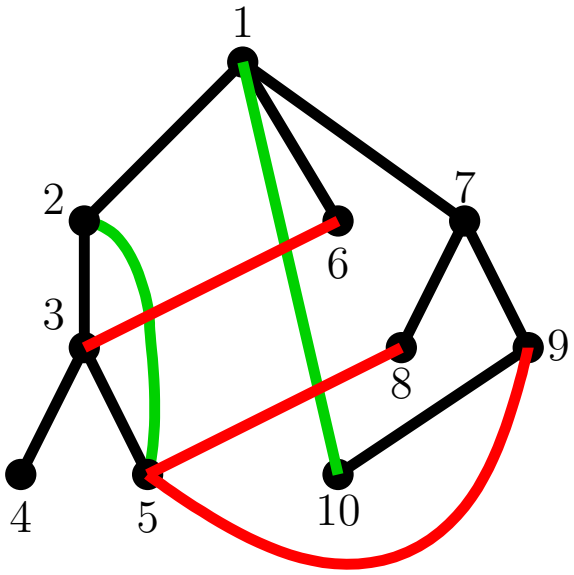
$x \to y$ egy	ha az él vizsgálatakor
faél	mszám$[y] = 0$
visszaél	mszám$[y] \leq$ mszám$[x]$ és bszám$[y] = 0$
előreél	mszám$[y] >$ mszám$[x]$
keresztél	mszám$[y] <$ mszám$[x]$ és bszám$[y] > 0$.

Tétel

A G irányított gráf mélységi bejárása – beleértve a mélységi, a befejezési számozást és az élek osztályozását is – $O(n + e)$ lépésben megtehető.

Irányítatlan gráfok mélységi bejárása

Mélységi keresés ugyanígy.
Mélységi feszítőerdő komponensei: összefüggő komponensek.



faél

visszaél

ilyen nincs

faél $\iff$ faél
előreél, visszaél $\iff$ visszaél
keresztél: nem létezik

Mélységi feszítőerdő

Legyen T a $G = (V, E)$ irányított gráf egy feszítőerdeje. Legyen $x \in V$ egy tetszőleges csúcs, és jelölje T_x a feszítőerdő x gyökerű részfájának a csúcshalmazát. Legyen

$$S_x = \left\{ y \in V \;\middle|\; \begin{array}{c} \text{van olyan } G\text{-beli } x \rightsquigarrow y \text{ irányított út, amelyen} \\ \text{a csúcsok mélységi száma legalább mszám}[x] \end{array} \right\}.$$

Lemma (részfa lemma)

Tetszőleges $x \in V$ csúcs esetén érvényes a $T_x = S_x$ egyenlőség.

Mélységi feszítőerdő

Lemma (részfa lemma)

Tetszőleges $x \in V$ csúcs esetén érvényes a $T_x = S_x$ egyenlőség.

Bizonyítás.

T_x éppen azokból a pontokból áll, amelyek x-ből faélek mentén elérhetők. Faélekre mindig nő a mélységi szám $\implies T_x \subseteq S_x$.

Fordított irány indirekt:
tegyük fel, hogy létezik egy $y \in S_x \setminus T_x$
Legyen $x \rightsquigarrow y$ egy az S_x meghatározásában szereplő irányított út, feltehetjük, hogy az út utolsó előtti v pontja T_x-ben van.
Az $y \in S_x$ feltétel szerint mszám$[y]$ >mszám$[x]$. Ez $y \notin T_x$ miatt azt jelenti, hogy y-t valamikor a T_x pontjai után látogatjuk meg.
$\implies (v, y)$ faél vagy előre él $\implies y \in T_x \implies S_x \subseteq T_x$. $\surd$ □

Mélységi feszítőerdő

Következmény

Tegyük fel, hogy a $G = (V, E)$ gráf x csúcsából minden pont elérhető irányított úton. Tegyük fel továbbá, hogy a G mélységi bejárását x-szel kezdjük. Ekkor a mélységi feszítőerdő egyetlen fából áll.

Bizonyítás.

mszám$[x] = 1 \implies S_x = V \implies T_x = V$ □

Irányított körmentes gráfok

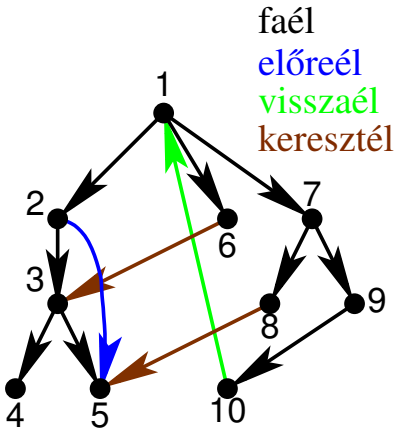
Definíció

Egy G irányított gráf DAG, ha nem tartalmaz irányított kört.

Directed Acyclic Graph
Alkalmazásai például:

- Teendők ütemezése $\Longrightarrow$ PERT
- Várakozási gráfok $\Longrightarrow$ adatbázisok

Fontos, hogy egy irányított gráfról el tudjuk dönteni, tartalmaz-e irányított kört.



Ha a gráf egy mélységi bejárása során találunk visszaélet akkor a gráf nyilván tartalmaz irányított kört, azaz nem DAG.

Tétel

Legyen $G = (V, E)$ egy irányított gráf. Ha G egy DAG, akkor egyetlen mélységi bejárása során sincs visszaél. Fordítva erősebb igaz: ha G-nek van olyan mélységi bejárása, amelyre nézve nincs visszaél, akkor G egy DAG.

Bizonyítás.

$\Longrightarrow$ $\surd$

$\Longleftarrow$ Indirekt bizonyítunk: Tegyük fel, hogy nincs visszaél, de G nem DAG $\Longrightarrow$ van benne irányított kör: vegyük ennek a legkisebb mélységi számú v csúcsát, a kör előző pontja legyen u.
$\Longrightarrow$ mszám$[v]$ < mszám$[u]$ $\Longrightarrow$ uv vissza- vagy keresztél, de u elérhető v-ből irányított úton, ahol a mélységi számok mindegyike $\geq mszm[v]$; (részfa lemma) $\Longrightarrow$ u a v leszármazottja $\Longrightarrow$ uv visszaél.

DAG topologikus rendezése

Definíció

Legyen $G = (V, E)$ ($|V| = n$) egy irányított gráf. G egy topologikus rendezése a csúcsoknak egy olyan $v_1, \ldots, v_n$ sorrendje, melyben $x \to y \in E$ esetén x előbb van, mint y (azaz ha $x = v_i, y = v_j$, akkor $i < j$).

Tétel

Egy irányított gráfnak akkor és csak akkor van topologikus rendezése, ha DAG.

Bizonyítás.

$\Rightarrow$: Ha G nem DAG, akkor nem lehet topologikus rendezése, mert egy irányított kör csúcsainak nyilván nincs megfelelő sorrendje.
$\Leftarrow$: G-ben van olyan csúcs, amibe nem fut be él (forrás).
Indukció pontszámra: $\Longrightarrow$ hagyjunk el egy x forrást, ez legyen az első pont $\Longrightarrow$ a többi az indukció miatt rendezhető $w_1, \ldots, w_{n-1}$
$\Longrightarrow$ $x, w_1, \ldots, w_{n-1}$ topologikus rendezése G-nek. $\surd$ □

Topologikus rendezés mélységi kereséssel

Tétel

Végezzük el a G DAG egy mélységi bejárását, és írjuk ki G csúcsait a befejezési számaik szerint növekvő $w_1, \ldots, w_n$ sorrendben. A $w_n, w_{n-1}, \ldots, w_1$ sorrend a G DAG egy topologikus rendezése.

Bizonyítás.

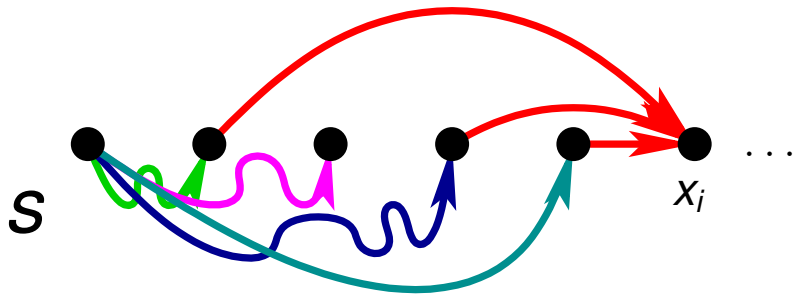
Azt kell belátnunk, hogy ha $w_i \to w_j$ éle G-nek, akkor $i > j$.
Ha volna olyan $w_i \to w_j$, amire $j = \text{bszám}[w_j] > \text{bszám}[w_i] = i$, akkor az csak visszaél lehetne.  □

Lépésszám: $O(n + e)$

Legrövidebb utak DAG-ban

Legrövidebb utak egy forrásból: Bellman-Ford $\Longrightarrow O(n^3)$
Ha nincs negatív élsúly: Dijkstra $\Longrightarrow O(n^2)$
Vegyünk egy topologikus rendezést: $x_1, x_2, \ldots, x_n$
Feltehetjük, hogy $s = x_1$ $\Longrightarrow$

$$d(s, x_i) = \min_{(x_j, x_i) \in E} \{d(s, x_j) + c(x_j, x_i)\}.$$



Ezt sorban elvégezzük minden i-re (dinamikus programozás).

Lépésszám: DFS+$\left(\sum_{i=1}^{n}$ min. keresés$\{d_{be}(x_i)\}\right) = O(n + e)$

Leghosszabb utak DAG-ban

Leghosszabb út $\rightarrow$ egyszerű út
Általában nehéz, nem ismert rá gyors algoritmus.
DAG-ban van:

Tétel

Ha G egy éllistával adott súlyozott élű DAG, akkor az egy forrásból induló legrövidebb és leghosszabb utak meghatározásának feladatai $O(n + e)$ lépésben megoldhatók.

Bizonyítás.

DAG-ban minden út csak előre megy $\Longrightarrow$

$$l(s, x_i) = \max_{(x_j, x_i) \in E} \{l(s, x_j) + c(x_j, x_i)\}.$$

ahol $l(s, x_i)$ a leghosszabb $s \rightsquigarrow x_i$ út hossza □

Alkalmazás: PERT

PERT

Program Evaluation and Review Technique
Egy nagy, összetett feladat részfeladatai legyenek egy gráf csúcsai.
Egy $x \to y$ él súlya, $c(x, y)$ azt mondja meg, hogy mennyi időnek kell eltelnie x elkezdése után y elkezdéséig. (Alapozás után 1 napot kell várni a festésig).
Ha van > 0 összsúlyú irányított kör, akkor nincs megoldás
$\Longrightarrow$ feltehetjük, hogy DAG.

Kérdés

Mikor lehet elkezdeni egy adott x részfeladatot?
Mikor lehet elkezdeni az utolsót?

Megkeressük a leghosszabb utat x-be a forrásból, illetve a legtávolabbi pontot a forrástól.

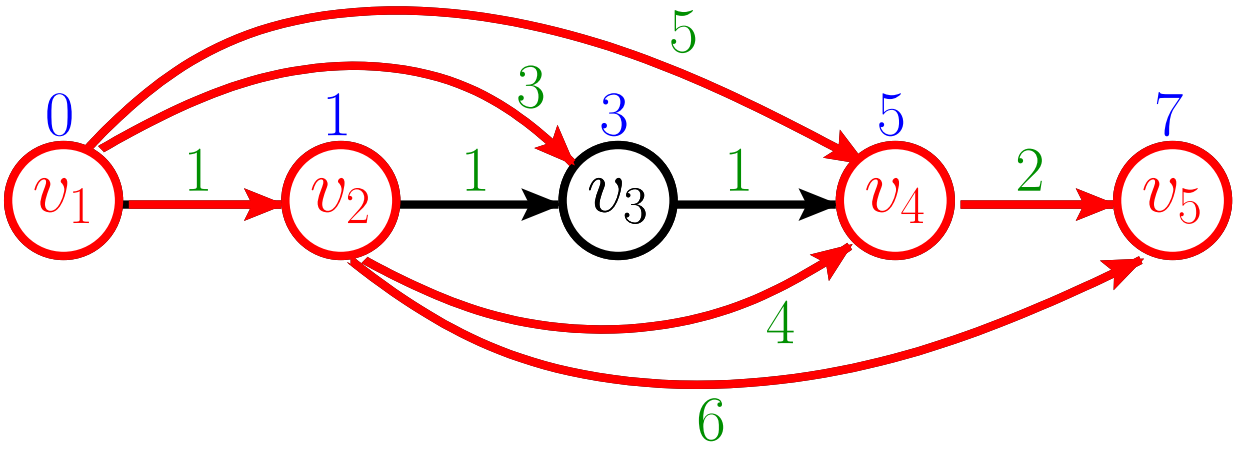
PERT

Definíció

Kritikus út: A forrásból induló valamely leghosszabb út.
Kritikus tevékenység: Ami rajta van kritikus úton.

Ha egy kritikus tevékenység késik, akkor késik az egész befejezése.
Az algoritmus közben színezzük pirosra az(oka)t az x-be befutó éleket, ahol a maximumot felveszi az összeg.
A kritikus utak a forrásból az utolsó feladatba menő csupa piros utak.
A kritikus tevékenységek azok, amik valamely ilyen úton rajta vannak.

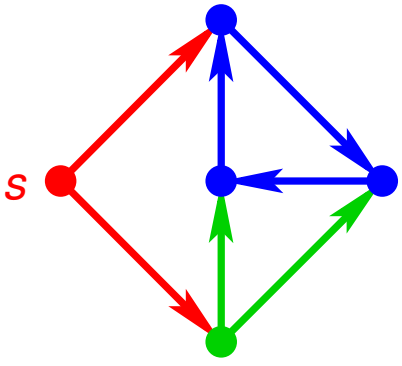
PERT példa



Erősen összefüggő (erős) komponensek

Definíció

Egy $G = (V, E)$ irányított gráf erősen összefüggő, ha bármely $u, v \in V$ pontpárra létezik $u \rightsquigarrow v$ irányított út.



Definíció

Legyen $G = (V, E)$ egy irányított gráf. Bevezetünk egy relációt V-n: $u, v \in V$-re legyen $u \approx v$, ha G-ben léteznek $u \rightsquigarrow v$ és $v \rightsquigarrow u$ irányított utak.

Ez ekvivalenciareláció $\Longrightarrow$

Definíció

$A \approx$ reláció ekvivalenciaosztályait a G erősen összefüggő (erős) komponenseinek nevezzük.

Erős összefüggőség eldöntése

(1) Mélységi bejárással végigmegyünk G-n egy v pontból indulva.

(2) Elkészítjük a G_{ford} gráfot, melyet úgy kapunk G-ből, hogy minden él irányítását megfordítjuk. Pontosabban: $G_{\text{ford}} := (V, E')$, ahol $u \to v \in E'$ akkor és csak akkor, ha $v \to u \in E$.

(3) Bejárjuk a G_{ford} gráfot mélységi bejárással v pontból indulva.

(4) Ha mindkét bejárás során minden pontot bejártunk, akkor a gráf erősen összefüggő.

Lépésszám: $O(n + e)$

Van $O(n + e)$ lépésszámú algoritmus, ami megadja az erősen összefüggő komponenseket is.

Megjegyzés: Az (1) és (3) pontokban valójában elég a mélységi bejárás mb(v) eljárását meghívni, hiszen ha a kiinduló pontból nem értünk el egy pontot, akkor a gráf nem erősen összefüggő.

Algoritmuselmélet

Minimális feszítőfák

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

11. előadás

Minimális költségű feszítőfák

Most irányítatlan gráfokkal foglalkozunk.
A kör és út most valóban egyszerű.

Definíció (minimális költségű feszítőfa)

Legyen $G = (V, E)$ egy összefüggő gráf. A G gráf egy körmentes összefüggő $F = (V, E')$ részgráfja a gráf egy feszítőfája. Legyen továbbá az éleken értelmezve egy $c : E \to \mathbb{R}$ súlyfüggvény. Ekkor a G gráf egy F feszítőfája minimális költségű, ha költsége (a benne szereplő élek súlyainak összege) minimális G összes feszítőfáját tekintve.

Probléma

Adott egy $G = (V, E)$ összefüggő irányítatlan gráf, és az élein értelmezett $c : E \to \mathbb{R}$ súlyfüggvény. Határozzuk meg a G egy minimális költségű feszítőfáját.

Például: villamosvezetékek kiépítése.

Fák tulajdonságai

Tétel

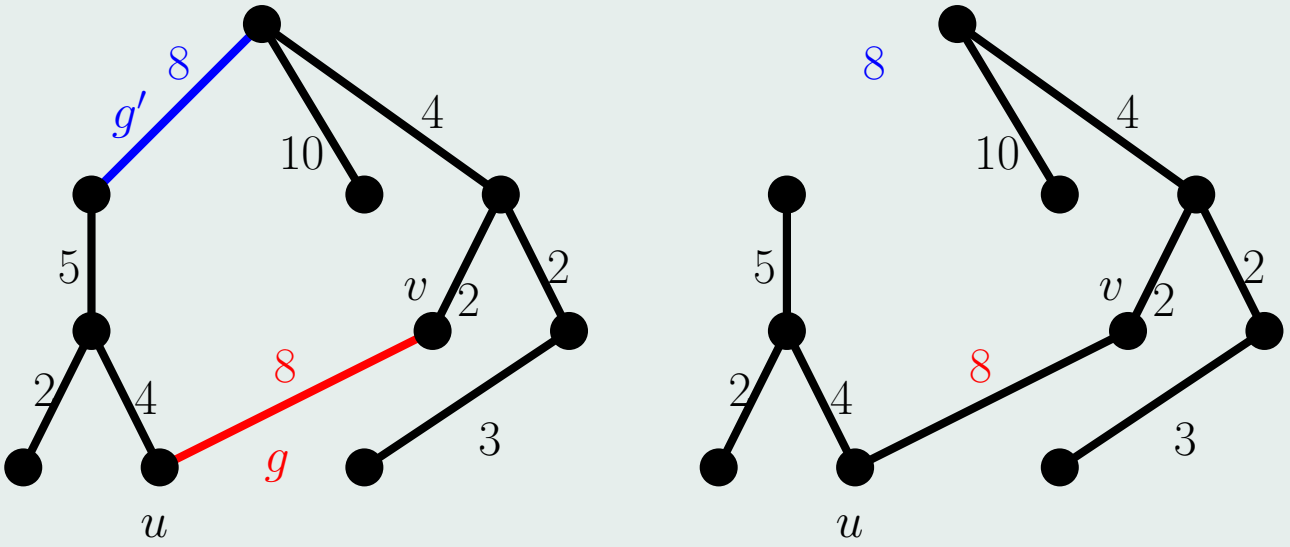
1. Minden legalább kétpontú fában van olyan csúcs, amiből csak egy él megy ki (elsőfokú csúcs).
2. Bármely összefüggő gráf tartalmaz feszítőfát.
3. Egy n-pontú összefüggő gráf akkor és csak akkor fa, ha $n-1$ éle van.
4. Egy fa bármely két pontja között pontosan egy út vezet.
5. Legyen G egy súlyozott élű összefüggő gráf, F egy minimális költségű feszítőfája. Legyen $g=(u,v)$ a G-nek egy olyan éle, ami nem éle F-nek, és tegyük fel, hogy az F-beli u-ból v-be vezető úton van olyan g' él, amelyre $c(g) \leq c(g')$. Ekkor az F-ből a g hozzávételével és a g' elhagyásával adódó F' gráf is egy minimális költségű feszítőfa G-ben.

Bizonyítás.

1–4 volt már BSZ-en. $\surd$

Bizonyítás.

5.



$F \cup \{g\}$ gráfban van olyan kör, amelynek g' éle. $\Longrightarrow$ A g' törlésével kapott F' gráf összefüggő marad.
F' költsége nem nagyobb F költségénél. □

A piros-kék algoritmus

Sorra nézzük G éleit: bizonyosakat beveszünk a minimális feszítőfába, másokat pedig eldobunk.
$\Longrightarrow$ Színezzük a G éleit:
a *kék* élek belekerülnek a végeredményt jelentő minimális feszítőfába, a *pirosak* pedig nem.
$\Longrightarrow$ Úgy színezünk, hogy az eddig kialakult (részleges) színezés mindig *takaros* legyen.

Definíció (takaros színezés)

Tekintsük a súlyozott élű G gráf éleinek egy részleges színezését, melynél bármely él piros, kék vagy színtelen lehet. Ez a színezés takaros, ha van G-nek olyan minimális költségű feszítőfája, ami az összes kék élet tartalmazza, és egyetlen piros élet sem tartalmaz.

A piros-kék algoritmus

KÉK SZABÁLY:	Válasszunk ki egy olyan $\emptyset \neq X \subset V$ csúcshalmazt, amelyből nem vezet ki kék él. Ezután egy legkisebb súlyú X-ből kimenő színezetlen élet fessünk kékre.
PIROS SZABÁLY:	Válasszunk G-ben egy olyan egyszerű kört, amelyben nincs piros él. A kör egyik legnagyobb súlyú színtelen élét fessük pirosra.

Kezdetben G-nek nincs színes éle. A két szabályt tetszőleges sorrendben és helyeken alkalmazzuk, amíg csak lehetséges.
$\Longrightarrow$ piros-kék algoritmus

A piros-kék algoritmus

Tétel

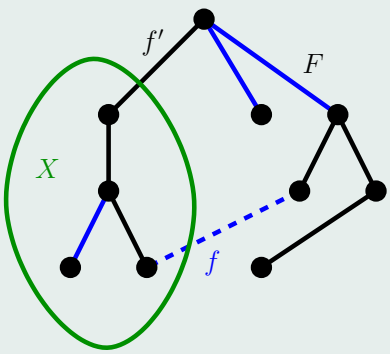
A piros-kék eljárás működése során mindig takaros színezésünk van. Ezen felül a színezéssel sosem akadunk el: végül G minden éle színes lesz.

Bizonyítás.

Belátjuk, hogy a színezés mindig takaros. Kezdetben $\surd$.
Tegyük fel, hogy egy takaros színezésünk van. Legyen F a G egy olyan minimális költségű feszítőfája, amely minden kék élet tartalmaz, és egyetlen pirosat sem. Tegyük fel továbbá, hogy ebben a helyzetben a gráf f élét festjük be.

Bizonyítás.

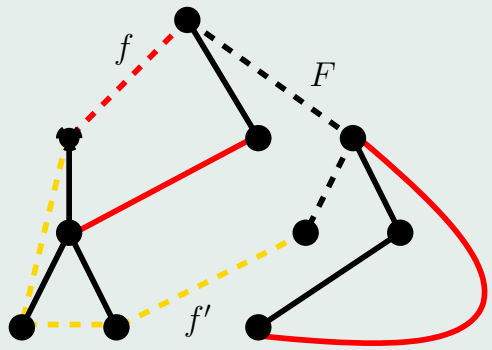
Két eset van aszerint, hogy melyik szabályt használjuk:
A kék szabályt használjuk: $\Longrightarrow$ *f kék lesz.*
Ha f éle F-nek $\surd$



Ha f nem éle F-nek $\Longrightarrow$ *nézzük azt az* $X \subset V$ *csúcshalmazt, amelyre a kék szabályt alkalmaztuk.*
Az F-ben van olyan út (mert feszítőfa), ami az f két végpontját összeköti. $\Longrightarrow$ *Ezen az úton pedig van olyan* f' *él, ami kimegy X-ből, ugyanis f kilép X-ből.*
Az F választása miatt f' *nem lehet piros.*
A kék szabály szerint kék sem lehet, továbbá $c(f') \geq c(f)$ *is teljesül.*
Legyen F' *az F-ből az* f' *törlésével és az f hozzáadásával kapott gráf.*
$\Longrightarrow$ *Eszerint* F' *egy minimális feszítőfa, tartalmaz minden kék élet és nem tartalmaz piros élet.*
$\surd$

Bizonyítás.

A piros szabályt használjuk: Ekkor f piros lesz. $\Longrightarrow$
Ha f nem éle F-nek $\surd$



Ha $f \in F$*, akkor az f törlésével az F két komponensre esik.*
$\Longrightarrow$ *A körnek, amelyre a piros szabályt alkalmaztuk, van olyan* $f' \neq f$ *éle, ami a két komponens között fut.*
A régi színezés takarossága és a piros szabály miatt az f' *színtelen és* $c(f') \leq c(f)$*.*
Az F-be f helyett f'*-t véve a kapott* F' *egy minimális költségű feszítőfa lesz.* $\surd$

Bizonyítás.

Miért nem akadunk el soha?
Tegyük fel, hogy van még egy *f* színtelen él.
A színezés takaros $\Longrightarrow$ a kék élek egy erdőt alkotnak.
$\Longrightarrow$ Ha *f* végpontjai ugyanabban a kék fában vannak, akkor a piros szabály alkalmazható arra körre, aminek az élei *f* és az *f* végpontjait összekötő (egyetlen) kék út élei.
$\Longrightarrow$ Ha *f* különböző kék fákat köt össze, akkor pedig a kék szabály működik; *X* legyen az egyik olyan fa csúcshalmaza, amihez *f* csatlakozik. (Ez utóbbi esetben nem biztos, hogy *f* fog színt kapni a következő lépésben.) □

A piros-kék algoritmus

Tétel

Ha a piros-kék algoritmussal befestjük az összefüggő $G = (V, E)$ gráf minden élét, akkor a kék élek egy minimális költségű feszítőfa élei. Sőt, ez már akkor is igaz, amikor van $|V| - 1$ kék élünk (és esetleg van még színezetlen él).

Bizonyítás.

Az első állítás $\Longleftarrow$ a végső színezés is takaros.
A második: végül összesen $|V| - 1$ kék él lesz. Ha már van ennyi, akkor több nem keletkezhet. □

Prim, Kruskal és Borůvka módszerei

A recept *helyessége* szempontjából tehát közömbös a sorrend, *hatékonyság* szempontjából viszont nem.

PRIM MÓDSZERE: Legyen s a G egy rögzített csúcsa. Minden egyes színező lépéssel az s-et tartalmazó F kék fát bővítjük. Kezdetben az F csúcshalmaza $\{s\}$, végül pedig az egész V. A következő kék élnek az egyik legkisebb súlyú élet választjuk azok közül, amelyek F-beli pontból F-en kívüli pontba mennek.

KRUSKAL MÓDSZERE: A következő befestendő f él legyen mindig a legkisebb súlyú színtelen él. Ha az f két végpontja ugyanazon kék fában van, akkor az él legyen piros, különben pedig kék.

BORŮVKA MÓDSZERE: Minden egyes kék fához válasszuk ki a legkisebb súlyú belőle kimenő (színtelen) élet. Színezzük kékre a kiválasztott éleket.

Prim módszere

Mindig a kék szabályt alkalmazzuk: Válasszuk X-nek a meglévő fa ponthalmazát. A kék élek végig fát alkotnak.

procedure Prim (G: gráf; **var** F: élek halmaza);
  var
    U: csúcsok halmaza;
    u, v: csúcsok;
  begin
    $F := \emptyset$; $U := \{1\}$;
    while $U \neq V$ **do begin**
(∗) legyen (u, v) egy legkisebb súlyú olyan él,
      melyre $u \in U$ és $v \in V \setminus U$;
      $F := F \cup \{(u, v)\}$;
      $U := U \cup \{v\}$
    end
  end

Jól működik, mert piros-kék algoritmus.

Naiv implementáció

A gráf az élsúlyokat tartalmazó C adjacencia-mátrixával adott.
Az épp aktuális U és $V \setminus U$ halmazok közt futó legkisebb súlyú élek kiválasztása a legegyszerűbb implementációval: $O(n^2)$ lépés
$\Longrightarrow$ Minden $V \setminus U$-beli csúcshoz tároljuk, hogy milyen messze van az U halmaztól:

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

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

A következő kék él az $(i, \text{KÖZEL}[i])$ élek közül kerül majd ki $\Longrightarrow$ *kékes* élek.

$$\text{KÖZEL}[i] := \begin{cases} * & \text{ha } i = 1 \\ 1 & \text{ha } i \neq 1 \end{cases} \qquad \text{MINSÚLY}[i] := \begin{cases} * & \text{ha } i = 1 \\ C[i,1] & \text{ha } i \neq 1 \end{cases}$$

- *A következő kék él kiválasztása:* megkeressük a MINSÚLY[ ] tömb minimumát, $\Longrightarrow$ legrövidebb kékes él legyen a k-ba mutató.
  A minimumkeresés költsége: $O(n)$ lépés.
  A (KÖZEL$[k], k$) élet fogjuk F-be tenni, k-t pedig U-ba.
  $\Longrightarrow$ MINSÚLY$[k]$:= KÖZEL$[k] := *$.
- *A két tömb felfrissítése:* A $C[k,i]$ és a MINSÚLY$[i]$ értékeket ($i \in V \setminus U$) kell összevetni. $\Longrightarrow$

```
if KÖZEL[i] ≠ * and C[k,i] < MINSÚLY[i] then begin
    KÖZEL[i] := k;
    MINSÚLY[i] := C[k,i]
end
```

Lépésszám: Egy él színezés $O(n)$ $\Longrightarrow O(n^2)$

Kupacos-éllistás implementáció

Építsünk kupacot az aktuális U és $V \setminus U$ közötti élekből.
(Néhány) MINTÖR-rel lépéssel kiválaszthatjuk a minimálisat, amit kékre színezünk.
Megváltozott U $\Longrightarrow$ BESZÚR-ral beszúrjuk az új éleket.
Nem törődünk azokkal az élekkel, amik így U-n belül mennek
$\Longrightarrow$ ezért lehet, hogy MINTÖR-nél ilyet kapunk elsőre.

Lépésszám: A kupac mérete sosem haladja meg e-t.
A kezdeti kupacépítés $O(e)$, az egyes műveletek végrehajtása pedig $O(\log e)$ időt vesz igénybe.
Összesen kevesebb, mint e darab BESZÚR és legfeljebb e darab MINTÖR műveletet végzünk $\Longrightarrow O(e \log e)$.

Johnson: Kombináljuk a két ötletet, nyilvántartjuk a közeli csúcsokat, és d-kupacban tároljuk a kékes éleket.
$\Longrightarrow$ ha $n^{1,5} \leq e$ $\Longrightarrow O(e)$

Kruskal módszere

Mindig a legkisebb súlyú olyan élet színezzük kékre, amely még nem alkot kört az eddigi kék élekkel.
$\Longrightarrow$ A kék élek végig egy erdőt határoznak meg, akkor van kész, ha feszítőfa lesz.
$\Longrightarrow$ Az új kék él az eddigi erdő két komponensét fogja összekötni. Kezdetben n komponens, egy él színezésével eggyel csökken a komponensek száma.

Kruskal módszere

procedure Kruskal (G: gráf; **var** F, H: élek halmaza);
begin
  $F := \emptyset$; $H := E$;
  while $H \neq \emptyset$ **do begin**
    Töröljük a H minimális súlyú (v, w) élét.
    Ha $F \cup \{(v, w)\}$-ben nincs kör
    (azaz a v, w pontok különböző kék fákban vannak), akkor
      $F := F \cup \{(v, w)\}$
  end
end

$\Longrightarrow$ Ha a (v, w) él kört eredményez, akkor piros él, ha pedig nem, akkor kék él.

Kruskal módszere

Tétel

A Kruskal-algoritmus eredményeként végül F a G gráf egy minimális költségű feszítőfájának éleit tartalmazza.

Bizonyítás.

Ez is piros-kék algoritmus. □

Implementáció:

- élekből kupacot építünk $\rightarrow O(e \log e)$ lépés
- éleket előre rendezzük $\rightarrow O(e \log e)$ lépés

Hogyan döntjük el, hogy a kiválasztott él kört alkot-e az eddigi kiválasztottakkal?
$\Longrightarrow$ Tartsuk nyilván az aktuálisan egy kék fába tartozó csúcsokat mint halmazokat.
Kell: UNIÓ, HOLVAN

Az UNIÓ-HOLVAN adatszerkezet

Legyen adott egy véges S halmaz.
Ennek egy partícióját szeretnénk tárolni $\rightarrow U_1, \ldots, U_k \subseteq S$.

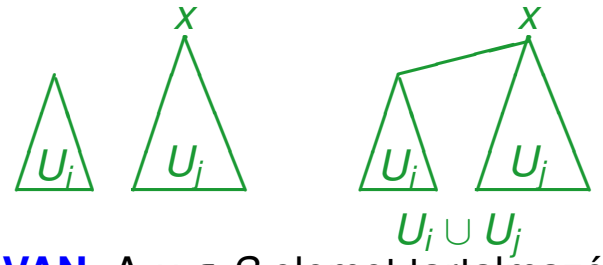
- Adott egy n elemű S halmaz, és ennek bizonyos $U_1, \ldots, U_m$ részhalmazai, melyekre $U_i \cap U_j = \emptyset$ $(i \neq j)$ és $U_1 \cup \ldots \cup U_m = S$ (vagyis az U_j részhalmazok S egy partícióját adják).
- *Műveletek:*
  UNIÓ$(U_i, U_j) = (\{U_1, \ldots, U_m\} \cup \{U_i \cup U_j\}) \setminus \{U_i, U_j\}$
  (az U_i, U_j halmazokat $U_i \cup U_j$ helyettesíti).
  HOLVAN(v) eredménye (itt $v \in S$) annak az U_i halmaznak a neve, amelynek v eleme.

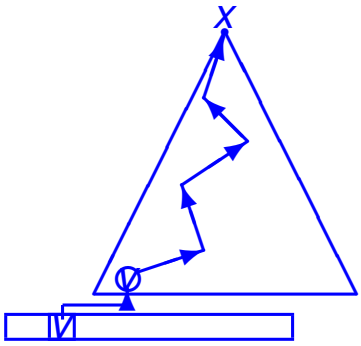
Kruskalban:
Annak megvizsgálása, hogy milyen színű legyen az új (u, v) él:
Ha HOLVAN$(u) \neq$ HOLVAN(v), akkor kék, különben piros.
Új él kékre színezése esetén $\Longrightarrow$ UNIÓ(HOLVAN(u), HOLVAN(v))

Implementáció fákkal

$U_j \to$ gyökeres, felfelé irányított fa
U_j elemeit a fa csúcsaiban tároljuk, egy szülőmutatóval.
Egy részhalmaz *neve* legyen az őt ábrázoló fa gyökere. A gyökérben nyilvántartjuk még a fa méretét (azaz a csúcsok számát) is.

- **UNIÓ:** $U_i \cup U_j$ fáját a következőképpen készítjük el: Tegyük fel, hogy $|U_i| \leq |U_j|$. Ekkor az U_j fa x gyökeréhez gyermekként hozzákapcsoljuk U_i gyökerét.





- **HOLVAN:** A $v \in S$ elemet tartalmazó részhalmaz nevét, azaz a megfelelő fa gyökerét a szülőkhöz menő mutatók végigkövetésével találhatjuk meg.

Az UNIÓ hívásakor az U_i és U_j halmazok a gyökerükkel adottak
$\Longrightarrow$ költség: $O(1)$
Ha egy v csúcs új gyökér alá kerül, akkor egy szinttel lesz távolabb a gyökértől, míg az új fájának a mérete legalább az eredeti duplájára változik.
$\Longrightarrow$ Egy csúcs legfeljebb $\log_2 n$-szer kerülhet új gyökér alá.
$\Longrightarrow$ szintszám $\leq \log_2 n$.
$\Longrightarrow$ HOLVAN költsége $O(\log n)$.

Tétel

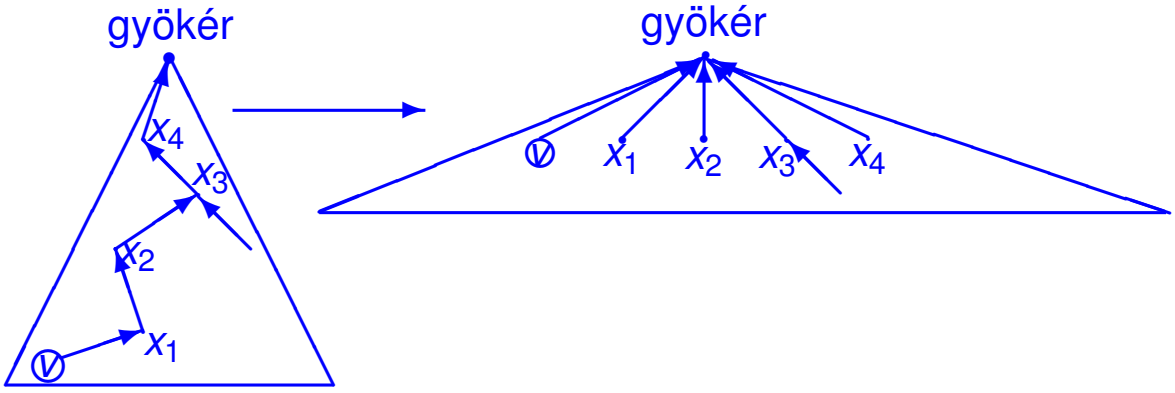
A Kruskal-algoritmus költsége $O(e \log e)$. □

Bizonyítás.

$2e$ HOLVAN, és $n-1$ UNIÓ műveletet jelent. Ezek időigénye $O(e \log n + n) = O(e \log n)$, vagy ami ugyanaz: $O(e \log e)$. □

A HOLVAN gyorsítása: útösszenyomás

Egy pontról többször is megnézzük HOLVAN, mindig log n lépés.
$\Longrightarrow$ Az első alkalommal minden ősét kössük közvetlenül a gyökérbe.



Tétel

Legyen $|S| = n$, és tegyük fel, hogy kezdetben mindegyik U_j egyelemű. Ha egy olyan utasítássorozatot hajtunk végre útösszenyomással, melyben $n-1$ UNIÓ és $m \geq n-1$ HOLVAN szerepel, akkor ennek az időigénye $O(m\alpha(m))$.

A korlátban szereplő $\alpha : \mathbb{Z}^+ \to \mathbb{Z}^+$ függvény az *inverz Ackermann-függvény*.
$\alpha(m)$ a végtelenhez tart, ha $m \to \infty$, de nagyon lassan, lassabban mint a logaritmus k-szori önmagába helyettesítésével adódó $\log \log \cdots \log m$ függvény ($k \in \mathbb{Z}^+$ tetszőleges).
Pl. $\alpha(m) \leq 4$, ha $m < 2^{65536}$
A normális méretű feladatoknál tehát $\alpha(m)$ állandónak (≤ 4) tekinthető.

Tétel

Ha az élek rendezésével kapcsolatos teendők $O(e)$ időben megoldhatók, akkor a Kruskal-algoritmus $O(e\alpha(e))$ időben megvalósítható.

Manipuláció a súlyokkal $\Longrightarrow$ Yao (1975): $O(e \log\log n)$
Véletlen módszerek $\Longrightarrow$ D. R. Karger, P. N. Klein, R. E. Tarjan, (1994): várhatóan $O(e)$
Online változatban is működik a piros-kék algoritmus: színezzük az új élet élet kékre; ha emiatt kialakul egy kék kör, akkor abból töröljünk egy maximális súlyú élet.

Algoritmuselmélet

Bonyolultságelmélet

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

12. előadás

Algoritmusok hatékonysága

Algoritmus lépésszáma:

- $\log n \implies$ remek
- $n \implies$ nagyon jó
- $n \log n \implies$ egész jó
- $n^2 \implies$ nem túl nagy n-re jó
- $n^3 \implies$ nem túl nagy n-re lehet jó
- $n^4 \implies$ nem túl nagy n-re néha jó
- $n^{10} \implies$ 50 év múlva, nem túl nagy n-re lehet jó
- $n^{100} \implies$ 1 millió év múlva lehet jó
- $2^n \implies$ nagyon kis n-re lehet jó, de nagy n-re sohasem

Algoritmusok hatékonysága

Az eddig tanult algoritmusok általában hatékonyak voltak:

- $a + b$ kiszámítása: Input mérete: $n = \log a + \log b$,
  Lépésszám: $O(\log a + \log b) = O(n)$
- ab kiszámítása: Input mérete: $n = \log a + \log b$,
  Lépésszám: $O((\log a + \log b) \log b) = O(n^2)$
- maradékos osztás, legnagyobb közös osztó:
  Input mérete: $n = \log a + \log b$, Lépésszám: $O(n^2)$
- Gráfalgoritmusok (összefüggőség, legrövidebb út, párosítás, ...)
  Input mérete: $n = e(G)$ vagy $n = v^2(G)$,
  Lépésszám: $O(n), O(n^2), O(n^3)$

Van olyan, amit nem lehet hatékonyan kiszámolni:

- 2^a kiszámítása: Input mérete: $n = \log a$,
  Lépésszám$\geq$ output mérete: $\log(2^a) = a = \Omega(2^n)$

Eldöntési problémák

Mostantól csak olyan típusú feladatokkal foglalkozunk, melyekre a válasz 1 bit: IGEN vagy NEM, ezek az úgynevezett *eldöntési problémák*.

- PRÍM
  Bemenet: $m > 0$ egész.
  Kérdés: m prímszám?
- ÚT
  Bemenet: $G = (V, E)$ irányítatlan gráf és két kitüntetett csúcs, $s, t \in V$.
  Kérdés: Van-e s és t között út a gráfban?
- MINÚT
  Bemenet: $G = (V, E)$ irányítatlan gráf és két kitüntetett csúcs, $s, t \in V$, k egész.
  Kérdés: Van-e s és t között legfeljebb k hosszú út a gráfban?

Eldöntési problémák

Legtöbbször igaz, hogy egy optimalizálási feladat megoldható egy megfelelő eldöntési feladat algoritmusának néhány futtatásával.

Például:
Ha MINÚT-ra van egy gyors algoritmusunk, akkor bináris kereséssel $O(\log e)$ db futtatással meghatározhatjuk a legrövidebb út hosszát.

Elég az eldöntési problémákat vizsgálni.

Definíció

Egy eldöntési problémához tartozó L nyelv azoknak a bemeneteknek a halmaza, amelyekre a válasz IGEN. A lehetséges bemeneteket (amelyek tehát vagy beletartoznak L-be vagy nem), szavaknak hívjuk. Egy X eldöntési probléma és x bemenet esetén $x \in X$ jelöli, hogy az x bemenetre a válasz IGEN.

Polinom időben eldönthető problémák

Definíció

Jelölje P azoknak az eldöntési problémáknak a halmazát, amelyekhez van olyan $\mathcal{A}$ algoritmus, amely minden x bemenetre helyesen megválaszolja a kérdést, és az algoritmus lépésszáma polinomiális, azaz $O(|x|^k)$ valamely k pozitív konstansra.
(Itt $|x|$ az x bemenet hosszát jelöli, k független x-től.)

Az eddig tanult algoritmusok eldöntési változata P-ben van.

Probléma

Milyen tulajdonsága lehet olyan problémáknak, amikről nem tudjuk, hogy P-ben van-e?

Hatékony tanúsítvány

Definíció

Azt mondjuk, hogy az X eldöntési problémához van hatékony tanúsítvány, ha van olyan $\mathcal{T}$ algoritmus, melynek a bemenete (x, t) párokból áll, ahol x az X probléma egy lehetséges bemenete és a következő feltételek teljesülnek: léteznek olyan c és k pozitív konstansok, hogy

- *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.*
- *A $\mathcal{T}$ algoritmus polinomiális, azaz a lépésszáma $O((|x| + |t|)^k)$.*

Röviden: Ha x bemenet esetén az eldöntési problémára a válasz IGEN, akkor erre van olyan polinom hosszú tanú (t), amiről $\mathcal{T}$-vel polinom időben ellenőrizhető, hogy valóban IGEN. Ha a válasz NEM, akkor viszont nincs olyan rövid érvelés, amivel be lehet minket csapni.

NP-beli problémák

Definíció

Jelölje NP azoknak az eldöntési problémáknak a halmazát, amelyekre van hatékony tanúsítvány.

Megjegyzés

Az NP a nemdeterminisztikus polinom idő rövidítése, ami arra utal, hogy t „megtalálása" nem feltétlenül determinisztikusan történik, egy $x \in X$ esetén elég, ha megsejtjük, mi lesz jó. Viszont utána az ellenőrzés, hogy valóban jó a t, amit választottunk (kaptunk valakitől), már polinom időben megy.

coNP-beli problémák

Definíció

Egy X eldöntési probléma komplementere *az az $\overline{X}$-sal jelölt probléma, melynek bemenete ugyanolyan mint az X esetén, de a válasz ellentétes. Azaz minden lehetséges x bemenetre*

$$x \in \overline{X} \Leftrightarrow x \notin X.$$

Például: ÖSSZETETT $= \overline{\text{PRÍM}}$

Definíció

Jelölje coNP *az* NP*-beli problémák* komplementereiből *álló halmazt, azaz $X \in \text{coNP} \Leftrightarrow \overline{X} \in \text{NP}$.*

Vagyis: Az NP-beli problémák esetében a definíció szerint az IGEN válaszra van polinom hosszú, polinom időben ellenőrizhető bizonyíték, a coNP-beli problémák azok, ahol a NEM válaszra van polinom hosszú, polinom időben ellenőrizhető bizonyíték.

Példák

- ÖSSZEFÜGGŐSÉG

  Bemenet: $G = (V, E)$ irányítatlan gráf.
  Kérdés: G összefüggő?

  $\in$NP: $t \rightarrow$ G egy feszítőfája: F;
  $\mathcal{T} \rightarrow$ ellenőrzi, hogy F tényleg feszítőfa-e G-ben.

  $\in$coNP: $t \rightarrow$ $U \subset V$
  $\mathcal{T} \rightarrow$ ellenőrzi, hogy nincs él U és $V - U$-beli pontok között

Példák

- PÁROS-GRÁF-PÁROSÍTÁS

  Bemenet: $G = (A, B; E)$ páros gráf

  Kérdés: Van-e G-ben teljes párosítás?

  $\in$NP: $t\rightarrow$ élek E' részhalmaza;
  $\mathcal{T}\rightarrow$ ellenőrzi, hogy E' tényleg teljes párosítás-e G-ben.

  $\in$coNP: $t\rightarrow X \subseteq A$
  $\mathcal{T}\rightarrow$ ellenőrzi, hogy teljesül-e $|X| > |N(X)|$.

Példák

- SÍKGRÁF

  Bemenet: $G = (V, E)$ gráf.

  Kérdés: Igaz-e, hogy G síkbarajzolható?

  $\in$NP: $t\rightarrow$ egy síkbarajzolásának leírása egy alkalmas számsorozattal; (Nem nyilvánvaló, hogy van ilyen polinom méretű t, de a Fáry-Wagner tételből következik, hogy igaz.)
  $\mathcal{T}\rightarrow$ ellenőrzi, hogy t tényleg G síkbarajzolása.

  $\in$coNP: $t\rightarrow$ egy K_5-tel vagy $K_{3,3}$-mal topologikusan izomorf részgráf G-ben;
  $\mathcal{T}\rightarrow$ ellenőrzi, hogy t tényleg ilyen részgráf.

Példák

- PRÍM

  Bemenet: $m > 0$ egész.
  Kérdés: m prímszám?

  $\in$NP: $t \rightarrow$ Bonyolult, de van ilyen;
  $\mathcal{T} \rightarrow$ ellenőrzi, hogy t tényleg tanúsítvány.

  $\in$coNP: $t \rightarrow$ $1 < k < m$ szám, ami m osztója;
  $\mathcal{T} \rightarrow$ ellenőrzi, hogy k osztja-e m-et.

Példák

- H

  Bemenet: G gráf.
  Kérdés: Van-e G-ben Hamilton-kör?

  $\in$NP: $t \rightarrow$ egy C Hamilton-kör G-ben;
  $\mathcal{T} \rightarrow$ ellenőrzi, hogy C Hamilton-kör-e.

  $\in$coNP: Nem tudjuk, hogy van-e hatékony tanúsítvány.

Példák

- 3SZÍN

  Bemenet: G gráf.

  Kérdés: Igaz-e, hogy G színezhető 3 színnel?

  $\in$NP: $t\rightarrow$ egy színezés megadása G-ben; (azaz egy $f: V(G) \rightarrow \{p, k, s\}$ függvény)
  $\mathcal{T}\rightarrow$ ellenőrzi, hogy f tényleg egy 3-színezése G-nek.

  $\in$coNP: Nem tudjuk, hogy van-e hatékony tanúsítvány.

Példák

- $\overline{\text{3SZÍN}}$

  Bemenet: G gráf.

  Kérdés: Igaz-e, hogy G nem színezhető 3 színnel?

  $\in$NP: Nem tudjuk, hogy van-e hatékony tanúsítvány.

  $\in$coNP: $t\rightarrow$ egy színezés megadása G-ben; (azaz egy $f: V(G) \rightarrow \{p, k, s\}$ függvény)
  $\mathcal{T}\rightarrow$ ellenőrzi, hogy f tényleg egy 3-színezése G-nek.

Hatékony tanúsítvány P-beli problémákra

A ÖSSZEFÜGGŐSÉG, PÁROS-GRÁF-PÁROSÍTÁS, SÍKGRÁF, PRÍM problémákra ismert polinomiális algoritmus is. $\Longrightarrow$
Az algoritmus lefutásának leírása egy hatékony tanúsítvány arra is, hogy a probléma NP-ben van, és arra is, hogy coNP-ben van, hiszen a végén a válasz vagy IGEN vagy NEM, a leírás pedig polinom hosszú. $\Longrightarrow$

Tétel

$$\mathrm{P} \subseteq \mathrm{NP} \textit{ és } \mathrm{P} \subseteq \mathrm{coNP}$$

Azaz: $\mathrm{P} \subseteq \mathrm{NP} \cap \mathrm{coNP}$

A legfontosabb nyitott kérdések

Sejtés

$$\mathrm{P} \neq \mathrm{NP}$$

Ha $\mathrm{P} = \mathrm{NP}$ teljesülne, akkor minden olyan problémára, amelyre van hatékony tanúsítvány (azaz NP-beli), lenne polinomiális algoritmus is.
Fogunk mutatni olyan problámákat, amik NP-beliek, de senki nem tud rájuk polinomiális algoritmust.

Sejtés

$$\mathrm{P} = \mathrm{NP} \cap \mathrm{coNP}$$

Eddig minden fontos $\mathrm{NP} \cap \mathrm{coNP}$-beli problémáról kiderült, hogy van rá polinomiális algoritmus, azaz P-beli.

Algoritmuselmélet

NP-teljes problémák

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

13. előadás

Karp-redukció

Mikor nem lényegesen nehezebb egy X probléma egy Y problémánál?
Ha Y felhasználásával meg lehet oldani X-et is.
$\Longrightarrow$ X visszavezethető a Y problémára.

Definíció

Legyen X és Y két eldöntési probléma. Az X Karp-redukciója (polinomiális visszavezetése) az Y problémára egy olyan polinom időben számolható f függvény, amely X minden lehetséges bemenetéhez hozzárendeli Y egy lehetséges bemenetét úgy, hogy

$$x \in X \Leftrightarrow f(x) \in Y.$$

Jelölés: $X \prec Y$, ha X-nek van Karp-redukciója Y-re.

Ha tehát van algoritmusunk Y eldöntésére $\Longrightarrow$ $x \in X$-re kiszámítjuk $f(x)$-et, eldöntjük $f(x) \in Y$? $\Longrightarrow$ tudjuk, hogy $x \in X$ igaz-e. $\surd$
Ha tudnánk, hogy X nehéz, és tudjuk, hogy $X \prec Y$
$\Longrightarrow$ Y is nehéz lenne.
Ha Y könnyű, és X nem lényegesen nehezebb nála, akkor X is könnyű.

Irányított Hamilton-kör probléma (IH)

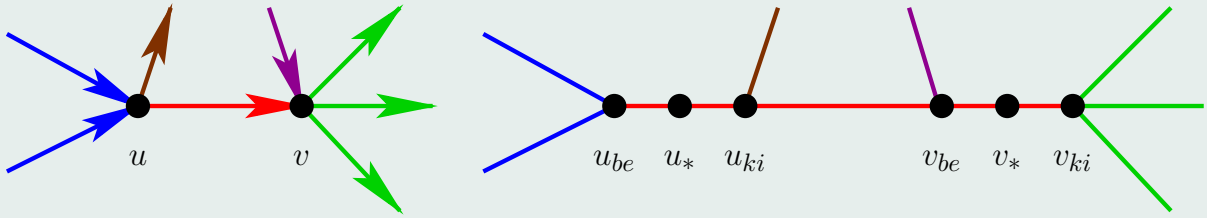
Tétel

$IH \prec H$.

Bizonyítás.

$G = (V, E)$ egy irányított gráf $\rightarrow$ $G' = (V', E')$ irányítatlan gráf hogy G' gyorsan megépíthető és
G-ben $\exists$ irányított Hamilton-kör $\leftrightarrow$ G'-ben $\exists$ irányítatlan Hamilton-kör.

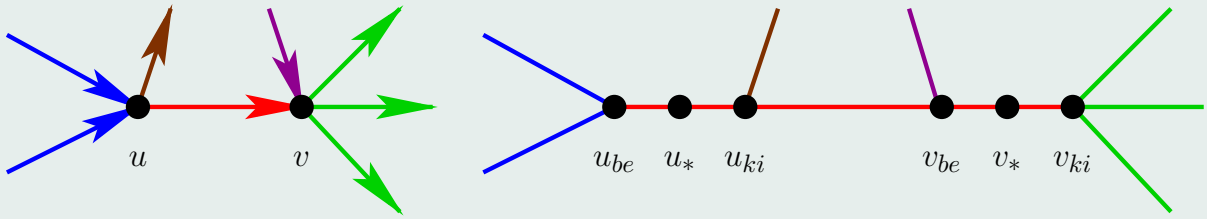
$$V' = \{v_{be}, v_*, v_{ki} \mid v \in V\},$$
$$E' = \{(v_{be}, v_*), (v_*, v_{ki}) \mid v \in V\} \cup \{(u_{ki}, v_{be}) \mid u \rightarrow v \in E\}.$$



$v(G) = n, e(G) = e \implies v(G') = 3n, e(G') = 2n + e \implies O(n + e)$ lépésben megkapható.

Bizonyítás.

G-beli F irányított Hamilton-körének megfelel G' egy F' Hamilton-köre.



Az F egy $u \rightarrow v$ éle $\longrightarrow$ az F'-ben az $u_* - u_{ki} - v_{be} - v_*$ út.
Ezért $G \in IH \implies G' \in H$
Ha G'-ben van egy $F' \subseteq E'$ Hamilton-kör $\implies$ egy u_*-ból indulva egy u_{ki} felé lépjünk először
$\implies$ csak $u_* - u_{ki} - v_{be} - v_*$ alakú lehet utána $\implies$ ez az út megfelel G-ben egy u -> v élnek. Ezt tovább folytatva Hamilton-kört kapunk G-ben.
Ezért $G' \in H \implies G \in IH$. □

A Karp-redukció tulajdonságai

Tétel

1. *Ha* $X \prec Y$ *és* $Y \in \mathrm{P}$, *akkor* $X \in \mathrm{P}$.
2. *Ha* $X \prec Y$ *és* $Y \in \mathrm{NP}$ *akkor* $X \in \mathrm{NP}$.
3. *Ha* $X \prec Y$, *akkor* $\overline{X} \prec \overline{Y}$
4. *Ha* $X \prec Y$ *és* $Y \in \mathrm{coNP}$, *akkor* $X \in \mathrm{coNP}$.
5. *Ha* $X \prec Y$ *és* $Y \in \mathrm{NP} \cap \mathrm{coNP}$, *akkor* $X \in \mathrm{NP} \cap \mathrm{coNP}$.
6. *Ha* $X \prec Y$ *és* $Y \prec Z$, *akkor* $X \prec Z$.

Bizonyítás.

Legyen f az X Karp-redukciója Y-re, ahol f $c_1 n^k$ időben számolható. x egy bemenet, melyről szeretnénk eldönteni, hogy $x \in X$ teljesül-e, n az x hossza.

Bizonyítás.

1.: Kiszámítjuk $f(x)$-et $\rightarrow$ időigénye $\leq c_1 n^k \implies |f(x)| \leq c_1 n^k$.
Y felismerő algoritmusával $c_2|f(x)|^l$ időben eldöntjük, hogy $f(x) \in Y$ igaz-e.
$\rightarrow$ időigénye $\leq c_2(c_1 n^k)^l$
$x \in X \Leftrightarrow f(x) \in Y \implies$ összidő $O(n^{kl})$ $\surd$

2.: Az $f(x) \in Y$ tény egy t tanúja jó $x \in X$ tanújának is, és az Y-hoz tartozó $\mathcal{T}$ tanúsító algoritmus egy kis módosítással jó lesz az X tanúsító algoritmusának is.
$\mathcal{T}'$ az (x, t) bemenetre először kiszámítja $f(x)$-et, majd az $(f(x), t)$ párra alkalmazza $\mathcal{T}$-t.
Ha az eredmény IGEN*, akkor legyen $\mathcal{T}'$ eredménye is* IGEN*, különben pedig* NEM*.*
$|t| = O(|f(x)|^c) \implies |t| = O(n^{kc})$
$\mathcal{T}'$ lépésszáma, ha $\mathcal{T}$ lépésszáma $O((|y| + |t|)^l)$:
$O(n^k) + O((|f(x)| + |t|)^l) = O(n^k) + O(|f(x)|^{cl}) = O(n^{kcl})$.

Bizonyítás.

3.: X-nek egy Karp-redukciója Y-ra egyben egy Karp-redukció $\overline{X}$-ről $\overline{Y}$-re, hiszen $x \in X \iff f(x) \in Y$ ugyanaz, mint $x \notin X \iff f(x) \notin Y$
4.: $\Longleftarrow$ 2.,3.
5.: $\Longleftarrow$ 2.,4.
6.: Legyen f az $X \prec Y$ függvénye, ami $O(n^k)$ időben számolható és g az $Y \prec Z$ függvénye, ami $O(n^l)$ időben számolható.
Az $X \prec Z$ függvénye $g(f(x))$ lesz, ami $O((n^k)^l) = O(n^{kl})$ időben számolható. □

NP-teljes problémák

Definíció

Az X eldöntési probléma NP-nehéz, ha tetszőleges (azaz minden) $X' \in$ NP probléma esetén létezik $X' \prec X$ 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.
Ha egy ilyen problémáról kiderülne, hogy P-beli (coNP-beli), akkor ugyanez igaz lenne minden NP-beli problémára.

Van-e NP-teljes probléma?

Boole-formulák

Definíció

Az $f : \{0,1\}^n \to \{0,1\}$ függvényeket n-változós Boole-függvényeknek *vagy* Boole-formuláknak *hívjuk.*

Tétel

Minden Boole-függvény felírható az $x_1, \ldots, x_n$ Boole-változók, az $\wedge, \vee, \neg$ logikai műveletek és zárójelek segítségével.

Pl. Boole-formula:
$\Phi = (x_1 \vee \neg x_2 \vee x_5) \wedge ((\neg x_3 \vee x_2 \vee (x_6 \wedge x_1)) \wedge \neg(x_5 \vee x_6))$

Boole-formulák

Definíció

Egy Boole-formula kielégíthető, ha lehet úgy értékeket adni a változóinak, hogy a függvény értéke 1 legyen.

Pl. $\Phi(x_1, x_2) = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ kielégíthető, mert ha $x_1 = 1$ és $x_2 = 0$, akkor $\Phi(x_1, x_2) = 1$
De pl. $(x_1 \wedge \neg x_1)$ nyilván nem kielégíthető.

Van-e NP-teljes probléma?

Definíció

SAT *probléma:*

Bemenet: Φ *Boole-fomula*

Kérdés: Kielégíthető-e Φ*?*

Tétel (S. A. Cook, L. Levin, 1971)

A SAT *probléma* NP*-teljes.*

Bizonyítás elég bonyolult.

Tétel

Ha az X *probléma* NP*-teljes,* $Y \in \mathrm{NP}$ *és* $X \prec Y$*, akkor* Y *is* NP*-teljes.*

Bizonyítás.

Láttuk, hogy a Karp-redukció tranzitív.

$\Longrightarrow$ Ha $X \prec Y$ és $Z \prec X$ teljesül $\forall Z \in \mathrm{NP}$ problémára.

$\Longrightarrow$ $Z \prec Y$ teljesül $\forall Z \in \mathrm{NP}$ problémára.

$\Longrightarrow$ $Y \in$ NP-nehéz.

Mivel $Y \in \mathrm{NP}$ is $\Longrightarrow$ $Y \in$ NP-teljes. □

Nem kell már minden NP-beli problémát az Y-ra redukálni; elég ezt megtenni egyetlen NP-teljes X problémával.

A 3-SZÍN probléma

Tétel

A 3SZÍN *probléma* NP*-teljes.*

Bizonyítás.

Már láttuk, hogy $\in$ NP*, belátható, hogy* SAT $\prec$ 3SZÍN*.*

Maximális méretű független pontrendszer gráfokban

MAXFTLEN

Bemenet: G gráf, $k \in \mathbb{Z}^+$.

Kérdés: Van-e G-nek k elemű független csúcshalmaza?

Tétel

A MAXFTLN *nyelv* NP*-teljes.*

Bizonyítás.

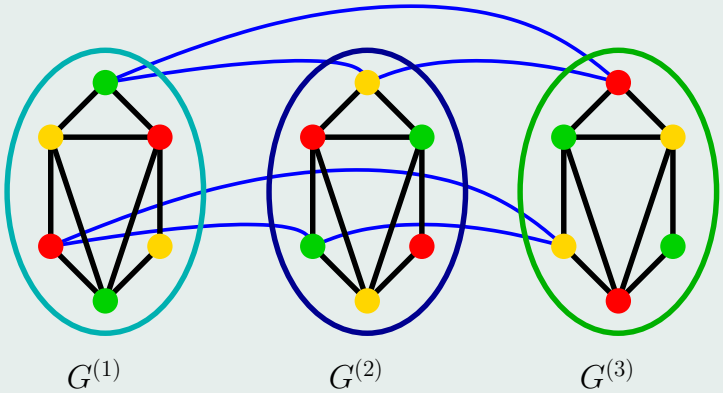
MAXFTLEN $\in$ NP*: tanú egy* k*-elemű* $S \subseteq V(G)$ *független csúcshalmaz.* $\surd$

Megadunk egy 3SZÍN $\prec$ MAXFTLEN *Karp-redukciót:* $G \to (G', k')$

$$G \in \text{3SZÍN} \Leftrightarrow (G', k') \in \text{MAXFTLEN}$$

Bizonyítás.

G' megadása: Vegyük G három másolatát ($G^{(1)}$, $G^{(2)}$, $G^{(3)}$), minden csúcs három példányát összekötjük.



$G^{(1)}$ $G^{(2)}$ $G^{(3)}$

$|V(G')| = 3|V(G)|$ és
$|E(G')| = 3|V(G)| + 3|E(G)|$,
legyen $k' = |V(G)|$.

Ha G színezhető 3 színnel $\Longrightarrow$ G' is $\Longrightarrow$
a piros pontok halmaza G'-ben független és $|V(G)|$ van belőlük. $\surd$
Ha G'-ben van $|V(G)|$ független, akkor legyen S egy ilyen ponthalmaz G'-ben.
$\Longrightarrow$ Minden G-beli x pontnak pontosan 1 példányát tartalmazza S.
$\Longrightarrow$ Az x pont legyen sárga / piros / zöld, ha ez a példány $G^{(1)}$-ben / $G^{(2)}$-ben / $G^{(3)}$-ban van. $\Longrightarrow$ ez jó színezés G-ben. $\surd$ $\square$

Maximális méretű klikk

MAXKLIKK
Bemenet: G gráf, $k \in \mathbb{Z}^+$.
Kérdés: Van-e G-ben k pontú teljes részgráf (k-klikk)?

Tétel

A MAXKLIKK *nyelv* NP*-teljes.*

Bizonyítás.

MAXKLIKK $\in$ NP: tanú egy k-elemű $S \subseteq V(G)$ teljes részgráf. $\surd$
Megadunk egy MAXFTLEN $\prec$ MAXKLIKK Karp-redukciót:
$f(G,k) = (\overline{G},k)$ (független ponthalmaz a komplementerben teljes gráf). $\surd$ $\square$

Részgráf izomorfia probléma

RÉSZGÁFIZO
Bemenet: G, H gráfok.
Kérdés: Van-e G-ben H-val izomorf részgráf?

Tétel

A RÉSZGRÁFIZO *nyelv* NP*-teljes.*

Bizonyítás.

RÉSZGRÁFIZO $\in$ NP: tanú egy részgráf és annak izomorfiája H-val. $\surd$
Megadunk egy MAXKLIKK $\prec$ RÉSZGRÁFIZO Karp-redukciót:
$f(G,k) = (G, K_k)$. $\surd$ □

Ha X NP-nehéz és Y általánosítása X-nek, akkor Y is NP-nehéz.
$\Longrightarrow$ RÉSZGRÁFIZO speciális esete a MAXKLIKK-nek $\Longrightarrow$ NP-nehéz.

Gráf izomorfia probléma

GRÁFIZO
Bemenet: G, H gráfok.
Kérdés: Igaz-e, hogy G és H izomorfak?

Tétel

A GRÁFIZO *nyelv* NP*-beli.*

Bizonyítás.

Tanú egy izomorfia a két gráf között. $\surd$ □

Nem ismert, hogy a GRÁFIZO NP-nehéz-e és az sem, hogy P-beli-e.

Hamilton-kör probléma

Tétel

A H *nyelv* NP*-teljes.*

Bizonyítás.

Már láttuk, hogy $\mathrm{H} \in \mathrm{NP}$. $\surd$
Belátható, hogy $\mathrm{SAT} \prec \mathrm{H}$. *(bonyolult)*

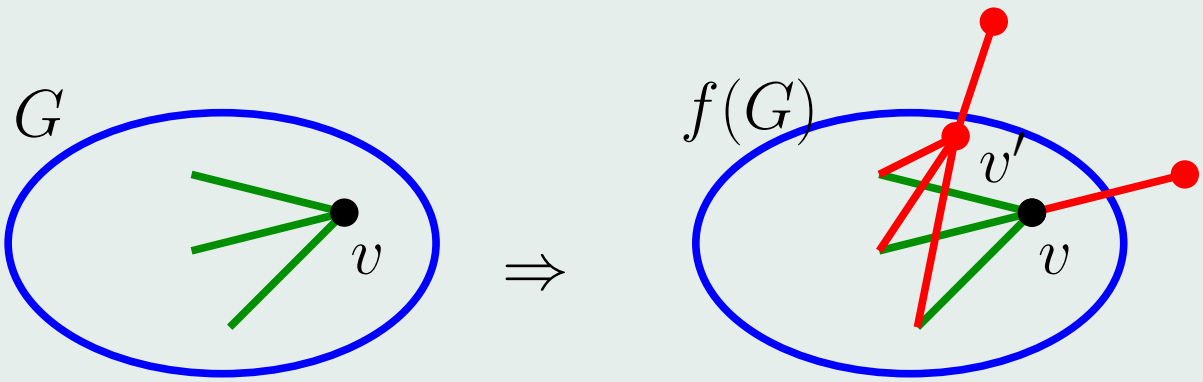
Hamilton-út probléma

Tétel

Az H-ÚT *nyelv* NP*-teljes.*

Bizonyítás.

$\mathrm{H\text{-}ÚT} \in \mathrm{NP}$, mert egy Hamilton-út tanú. $\surd$
Belátjuk, hogy $\mathrm{H} \prec \mathrm{H\text{-}ÚT}$.



G-ben akkor és csak akkor van Hamilton-kör, ha $f(G)$-ben van Hamilton-út. □

A Hátizsák feladat

Hátizsák feladat:

Adottak tárgyak $s_1, \ldots, s_m > 0$ súlyai, ezek $v_1, \ldots, v_m > 0$ értékei, valamint a b megengedett maximális összsúly.
Tegyük fel, hogy az s_i, v_i, b számok egészek.
A feladat az, hogy találjunk egy olyan $I \subseteq \{1, .., m\}$ részhalmazt, melyre $\sum_{i \in I} s_i \leq b$, és ugyanakkor $\sum_{i \in I} v_i$ a lehető legnagyobb.

$\Longrightarrow$

HÁT

Bemenet: $s_1, \ldots, s_m$; $v_1 \ldots, v_m$; b; k.
Kérdés: Van-e olyan $I \subseteq \{1, \ldots, m\}$ melyre $\sum_{i \in I} s_i \leq b$ és $\sum_{i \in I} v_i \geq k$?

Lemma

HÁT $\in$ NP

Vegyük azt a speciális esetet, amikor $s_i = v_i$ és $b = k$. $\Longrightarrow$

A Részhalmaz összeg probléma

RH

Bemenet: $(s_1, \ldots, s_m; b)$.
Kérdés: Van-e olyan $I \subseteq \{1, \ldots, m\}$ melyre $\sum_{i \in I} s_i = b$?

Tétel

Az RH *nyelv* NP*-teljes.*

Bizonyítás.

RH $\in$ NP. $\surd$
Belátható, hogy SAT $\prec$ RH*.*

Speciális eset: Partíció feladat: ahol $b = \frac{1}{2} \sum s_i$.

PARTÍCIÓ

Bemenet: $(s_1, \ldots, s_m)$.
Kérdés: Van-e olyan $I \subseteq \{1, \ldots, m\}$ melyre $\sum_{i \in I} s_i = \frac{1}{2} \sum_{i=1}^{m} s_i$?

A Partíció probléma

Tétel

A PARTÍCIÓ *probléma* NP*-teljes.*

Bizonyítás.

Partíció $\in$ NP. $\surd$

Belátjuk, hogy RH $\prec$*Partíció*, pedig RH általánosabb!

Vegyük az RH egy $x = (s_1, \ldots, s_m; b)$ inputját.

$\Longrightarrow$ Feltehető, hogy $b \leq s = \sum_{i=1}^{m} s_i$.

$f(x) = (s_1, \ldots, s_m, s + 1 - b, b + 1)$.

A számok összege $2s + 2$, az utolsó két szám nem lehet egy partíció ugyanazon osztályában, mert az összegük túl nagy: $s + 2 > \frac{1}{2}(2s + 2)$.

RH-nak megoldása az $R \subset \{s_1, ..., s_m\}$ számhalmaz$\Leftrightarrow$ a megoldáshoz vegyük hozzá $(s + 1 - b)$-t $\Leftrightarrow$ PARTÍCIÓ-nak megoldása az $R \cup \{s + 1 - b\}$ számhalmaz. $\square$

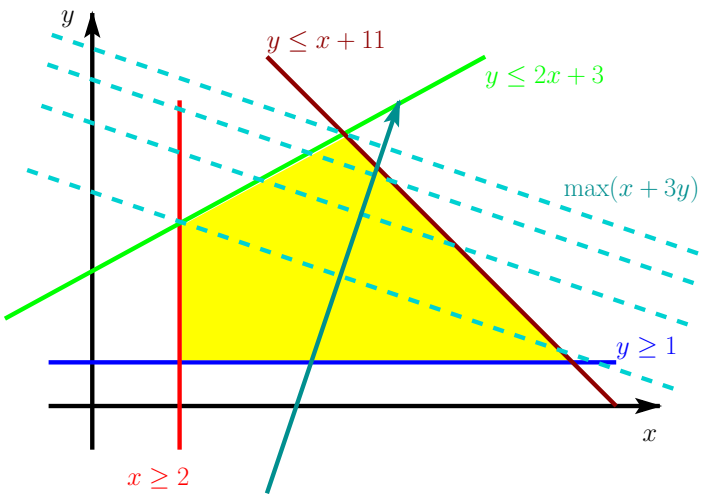
A Lineáris Programozás probléma

LP

Bemenet: Az $x_1, x_2, \ldots, x_m$ változókat tartalmazó lineáris egyenlőtlenségek.

Kérdés: Vannak-e olyan $x_1, x_2, \ldots, x_m$ számok, amelyek kielégítik az összes egyenlőtlenséget?

Optimalizációs változat: Mekkora $\max(c_1 x_1 + \ldots + c_m x_m)$, ha $x_1, x_2, \ldots, x_m$ kielégíti az egyenlőtlenségeket?



A Lineáris Programozás probléma

Tétel

A Lineáris Programozás probléma P*-ben van.*

Legjobb algoritmus (Karmarkar): $v^{3,5}e$, ahol v a változók száma, e az egyenletek „összmérete”.

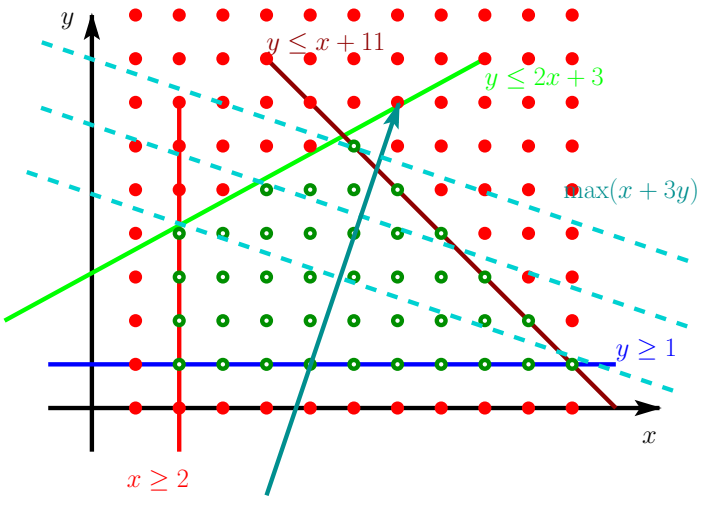
Az Egészértékű Lineáris Programozás probléma

IP

Bemenet: Az $x_1, x_2, \ldots, x_m$ változókat tartalmazó lineáris egyenlőtlenségek.

Kérdés: Vannak-e olyan $x_1, x_2, \ldots, x_m$ egészek, amelyek kielégítik az összes egyenlőtlenséget?

Optimalizációs változat: Mekkora $\max(c_1x_1 + \ldots + c_mx_m)$, ha $x_1, x_2, \ldots x_m$ kielégíti az egyenlőtlenségeket és mindegyik egész?



Az Egészértékű Lineáris Programozás probléma

Tétel

Az IP *probléma* NP*-teljes.*

Bizonyítás.

IP $\in$ NP: tanú egy megoldás, (bár nehéz belátni, hogy a megoldás polinom méretű!) $\surd$

Belátjuk, hogy SAT $\prec$ IP

$(x_1 \vee \neg x_2 \vee x_5) \wedge (x_2 \vee \neg x_3 \vee x_6) \wedge (\neg x_2 \vee x_3 \vee x_5 \vee \neg x_6) \Longrightarrow$

$0 \leq x_1 \leq 1; 0 \leq x_2 \leq 1; \ldots; 0 \leq x_6 \leq 1$

$x_1 + (1 - x_2) + x_5 \geq 1$

$x_2 + (1 - x_3) + x_6 \geq 1$

$(1 - x_2) + x_3 + x_5 + (1 - x_6) \geq 1$

□

Algoritmuselmélet

Közelítő algoritmusok

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

14. előadás

Közelítő algoritmusok

Hátha nem szükséges pontos megoldás, elég az optimumtól nem túl messze levő is, ha az polinom időben kiszámolható.

Közelítés additív konstanssal: $OPT - c \leq APPR \leq OPT + c$

Ilyen ritkán van.

Közelítő algoritmusok

C-KÖZÚT
Bemenet: G
Kérdés: Van-e legalább $v(G) - c$ hosszú út G-ben?

Tétel

C-KÖZÚT $\in$ NP-*teljes.*

Bizonyítás.

C-KÖZÚT $\in$ NP, tanú egy út. $\surd$
H-ÚT $\prec$ C-KÖZÚT: $G \Longrightarrow G' = G + c$ db izolált pont.
Ha G-ben van Hamilton-út, ez G'-ben egy $v(G') - c$ hosszú út. $\surd$
Ha G'-ben van egy $v(G') - c$ hosszú út, akkor az G minden pontját tartalmazza, tehát Hamilton-út. $\surd$ $\square$

Közelítő algoritmusok

Közelítés multiplikatív konstanssal: $\frac{1}{c} OPT \leq APPR \leq c \cdot OPT$

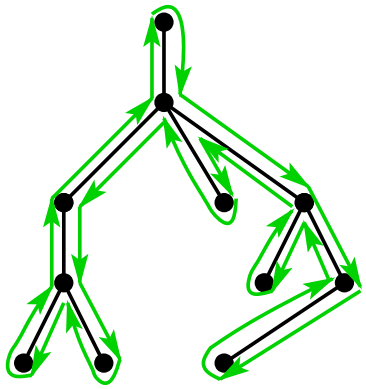
Ilyen sokszor van. Keressünk 1-hez minél közelebbi konstanst.

Euklideszi utazó ügynök probléma

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

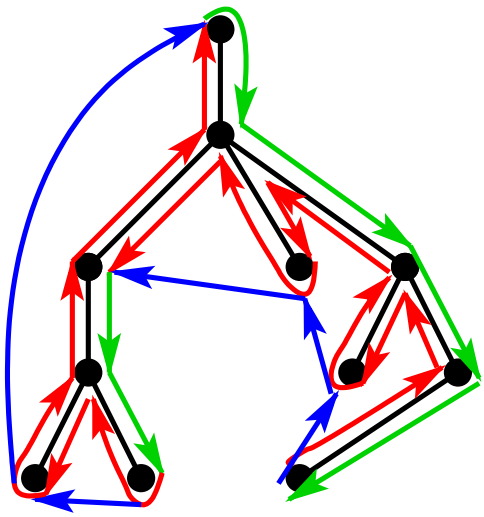
A cél egy minimális összsúlyú Hamilton-kör keresése.

Keresünk egy minimális összsúlyú feszítőfát (pl. Kruskal), megkettőzzük az éleit és „körbejárjuk” egy Euler-körsétával.



A minimális feszítőfa összsúlya legyen $s \implies$ Euler-séta hossza $2s$.

Ez nem Hamilton-kör $\implies$ levágjuk a fölösleges részeket, közben rövidítünk is.



Ha az optimális Hamilton-körből elhagyunk egy élet $\implies$ egy legalább s súlyú feszítőfát kapunk.

A módszer legfeljebb 2-szer akkora utat ad, mint az optimális.

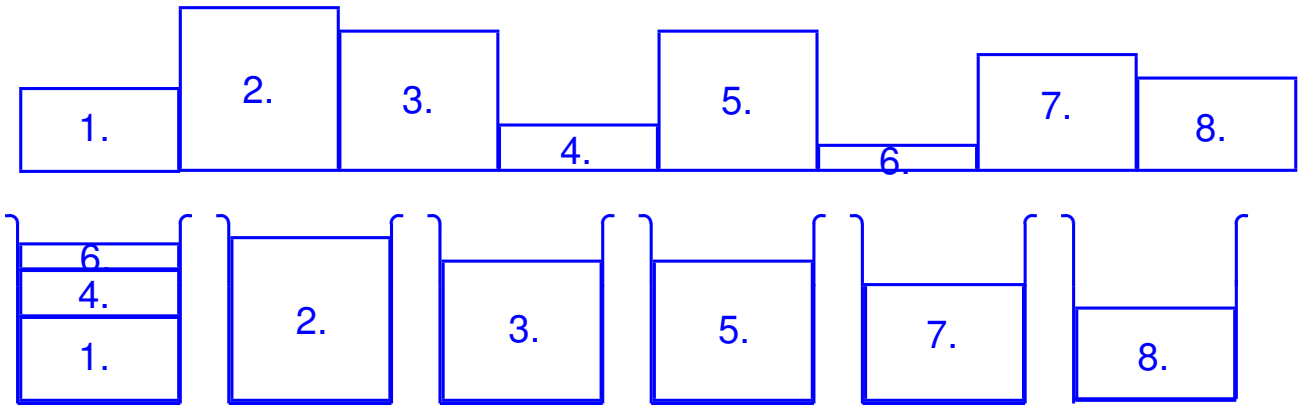
Ládapakolás

Ládapakolás: Adottak az $s_1, \ldots, s_m$ (racionális) súlyok, $0 \leq s_i \leq 1$. A cél a súlyok elhelyezése minél kevesebb 1 súlykapacitású ládába.

NP-nehéz, a PARTÍCIÓ probléma visszavezethető rá.

Ládapakolás

FF-módszer (*first fit*): Vegyünk először üres ládákat, és számozzuk meg őket az $1, 2, \ldots, m$ egészekkel.
Tegyük fel, hogy az $s_1, \ldots, s_{i-1}$ súlyokat már elhelyeztük. Ekkor s_i kerüljön az első (legkisebb sorszámú) olyan ládába, amelybe még befér.



First Fit

Tétel

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

Bizonyítás.

$\lceil \sum_{i=1}^{m} s_i \rceil \leq OPT(I)$
$FF(I) \leq \lceil 2 \sum_{i=1}^{m} s_i \rceil \iff$ nincs két olyan láda, amely nincs félig kitöltve.
Felhasználjuk, hogy $\lceil 2x \rceil \leq 2\lceil x \rceil$:

$$FF(I) \leq \lceil 2 \sum_{i=1}^{m} s_i \rceil \leq 2 \lceil \sum_{i=1}^{m} s_i \rceil \leq 2OPT(I).$$

√ □

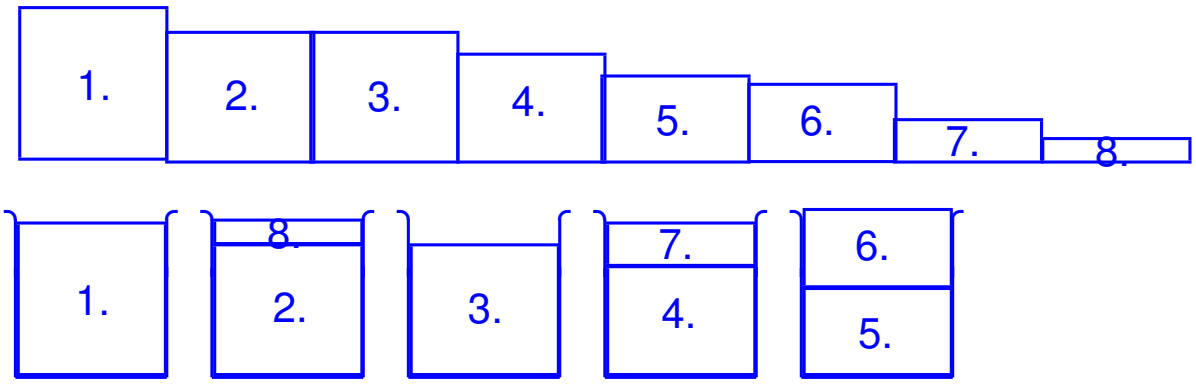
First Fit

Tétel (D. S. Johnson és munkatársai, 1976)

A probléma tetszőleges I inputjára teljesül, hogy $FF(I) \leq \lceil 1.7OPT(I) \rceil$. Továbbá vannak tetszőlegesen nagy méretű I inputok, melyekre $FF(I) \geq 1.7(OPT(I) - 1)$.

First Fit Decreasing

FFD-módszer (first fit decreasing): először rendezzük a súlyokat nem növő sorrendbe, utána alkalmazzuk az *FF*-módszert.



Tétel (D. S. Johnson, 1973)

Tetszőleges I inputra teljesül, hogy $FFD(I) \leq \frac{11}{9}OPT(I) + 4$, és tetszőlegesen nagy méretű I inputok vannak, melyekre $FFD(I) \geq \frac{11}{9}OPT(I)$. ($\frac{11}{9} = 1.222\ldots$)

Jobb approximáció

Tétel (W. Fernandez de la Vega, G. S. Lueker)

Tetszőleges $\varepsilon > 0$-hoz van olyan P lineáris algoritmus, amire $P(I) \leq (1+\varepsilon)OPT(I) + 1$.

Futásideje: $O(n) + 2^{2^{O((1/\varepsilon)\log(1/\varepsilon))}}$

Véletlent használó módszerek

Előny: Gyorsabb lehet.
Hátrány: Kis valószínőséggel hibás választ kapunk.

Prímtesztelés

Bemenő adatként adott (binárisan) egy m páratlan egész; szeretnénk eldönteni, hogy m prímszám-e.
Fermat-teszt (m)
1. Válasszunk egy véletlen a egészet az $[1, m)$ intervallumból.
2. Ha lnko$(a, m) \neq 1$, akkor a válasz „m összetett”.
3. Ha $a^{m-1} \equiv 1 \pmod{m}$, akkor a válasz „$m$ valószínűleg prím”, különben a válasz „m összetett”.

2. Euklideszi algoritmussal gyorsan végrehajtható.
3. Gyors hatványozással gyorsan végrehajtható.

Ha azt kapjuk, hogy „m összetett” $\implies$ ez biztos igaz.
Pl.: $m = 21 = 7 \cdot 3$ és $a = 2$ $\implies$ a az m Fermat-tanúja, hiszen $2^{20} \equiv 4 \pmod{21}$.

Prímtesztelés

Tétel

Ha m-nek van olyan a Fermat-tanúja ($1 \leq a < m$ és $a^{m-1} \not\equiv 1 \pmod m$), melyre $\text{lnko}(a, m) = 1$, akkor az $[1, m)$ intervallum egészeinek legalább a fele Fermat-tanú.

Bizonyítás.

Legyen a tanú $\Longrightarrow a^{m-1} \not\equiv 1 \pmod m$ és $c_1, c_2, \ldots, c_s$ nem tanúk $\Longrightarrow c_i^{m-1} \equiv 1 \pmod m$
Feltehetjük, hogy a, c_i relatív prímek m-hez.
$\Longrightarrow (ac_i)^{m-1} \equiv a^{m-1} c_i^{m-1} \equiv a^{m-1} \not\equiv 1 \pmod m \Longrightarrow ac_i$ tanú.
Ha $ac_i \equiv ac_j \pmod m \Longrightarrow m | ac_i - ac_j = a(c_i - c_j) \Longrightarrow m | c_i - c_j$, hiszen $\text{lnko}(a, m) = 1$. $\Longrightarrow ac_1, ac_2, \ldots, ac_s$ mind különbözőek lesznek $\Longrightarrow$ legalább annyi tanú, mint nem tanú. $\surd$ $\square$

Vannak olyan számok, amelyeknek nincs tanújuk: Carmichael-számok
Pl. $561 = 3 \cdot 11 \cdot 17$
Alford, Granville, Pomerance, 1992 $\Longrightarrow$ végtelen sok ilyen szám van

Rabin-Miller teszt

Definíció

Legyen m egy páratlan természetes szám. Írjuk fel $m - 1$-et $m - 1 = 2^k n$ alakban, ahol n páratlan. Az $1 \leq a < m$ egész Rabin–Miller-tanú *(m összetettségére), ha az*

$$a^n - 1,\ a^n + 1,\ a^{2n} + 1, \ldots,\ a^{2^{k-1} n} + 1$$

számok egyike sem osztható m-mel.

Rabin-Miller teszt

Tétel

Ha m prím, akkor m-hez nincs Rabin–Miller-tanú.

Bizonyítás.

$$a^{m-1} - 1 = (a^n - 1)(a^n + 1)(a^{2n} + 1) \cdots (a^{2^{k-1}n} + 1)$$

m prím $\implies$ a kis Fermat-tétel szerint m osztja a bal oldalt.
$\implies$ m osztja a jobb oldal valamelyik tényezőjét $\implies$ a nem Rabin–Miller-tanú. □

Tétel

Ha m összetett, akkor az $1 \leq a < m$ feltételt teljesítő a egészeknek legalább a fele Rabin–Miller-tanú.

Rabin-Miller teszt

RM(m)
1. Írjuk fel $m-1$-et $m-1 = 2^k n$ alakban, ahol n páratlan.
2. Válasszunk egy véletlen a egészet az $[1, m)$ intervallumból.
3. Ha az $a^n - 1,\ a^n + 1,\ a^{2n} + 1, \ldots,\ a^{2^{k-1}n} + 1$ számok egyike sem osztható m-mel, akkor megállunk azzal a válasszal, hogy „m összetett”, különben megállunk azzal a válasszal, hogy „m valószínűleg prím”.