Tartalomjegyzék

Bevezetés

Erről a jegyzetről

Ez a jegyzet a 2009. tavaszi félév előadásainak anyagát tartalmazza.
A mindenkori legfrissebb változat elérhető a BME-VIK Hallgatói Tudásbázisa által hivatalosnak kikiáltott jegyzetoldalon (jelenleg ez https://wiki.sch.bme.hu/bin/view/Infoalap/AlgEl), más oldalakra való feltöltéséhez viszont az alkotók tudta és beleegyezése szükséges.

A tantárgyról

A tantárgy hivatalos adatlapjának elérhetősége:
http://cs.bme.hu/algel

Ajánlott irodalom

Tankönyv: Rónyai Lajos, Ivanyos Gábor, Szabó Réka: Algoritmusok
Feladatgyűjtemény: A tárgy honlapjáról letölthető.
Számos feladat és megoldásuknak javaslata a http://www.cs.bme.hu/~drotos/algel2009tavasz/ weboldalon is található, Drótos Márton gyakorlatvezető oldalán, illetve a már említett jegyzetoldalon is, használati útmutatóval együtt.

A jegyzet készítéséhez felhasznált anyagok

A jegyzet túlnyomórészt Dr. Friedl Katalin előadásainak anyagából készült, helyenként kibővítve az ott elhangzott magyarázatokat. Azokon a helyeken, ahol egy tétel, definíció vagy képlet helyességében nem voltunk biztosak, az ajánlott irodalom részét képező tankönyvet használtuk az esetleges hibák javításhoz.

Figyelmeztetés

A jegyzet tartalmazhat hibákat a többszöri ellenőrzés ellenére is, nem jelenthet hivatkozási alapot számonkérésekkel kapcsolatos reklamációkor. Elolvasása nem helyettesíti az előadásokon és gyakorlatokon való részvételt, vagy az ajánlott irodalom alapos áttanulmányozását.

A készítők elérhetőségei

A talált hibák javításával, vagy bármely, a jegyzettel kapcsolatos ügyben a készítők elérhetők e-mailben a következő címeken:
Elekes Csaba: elekescsabi@indamail.hu
Kiss Dávid: lordyhun@gmail.com

Kiss Dávid
[Hallgatói Tudásbázis]
Elekes Csaba
infojegyzet.sch.bme.hu

1. előadás

A tárgyról röviden

Témák

- algoritmusok
- adatszerkezetek
- nehéz problémák

Algoritmusok

A problémamegoldás menete

Valós problémák → absztrakt modell → algoritmus → program

Algoritmus

Az *algoritmus* egy hatékony eljárás egy feladat vagy probléma megoldására, melynek helyessége bizonyítható.

Hatékonyság

A *hatékonyság*ot a *futási idő* és a *memóriaigény* határozza meg.

Futási idő

A *tényleges idő* és a ***lépésszám*** összege a *bemenet* hosszának függvényében.

Lépésszám – illusztráció

A számítógép $10^{10}\ \frac{lépés}{perc}$sebességű.

A program $n$nagyságú bemenetekre $f(n)$darab műveletet hajt végre. A táblázat a futási időt tartalmazza.

Bemenet h.	$f(n)=n$	$f(n)=n^2$	$f(n)=2^n$	$f(n)=n!$
$n=10$	10^{-9}	10^{-8}	10^{-7}	$3{,}6\cdot 10^{-4}$
$n=20$	$2\cdot 10^{-9}$	$4\cdot 10^{-8}$	10^{-4}	$>2év$
$n=40$	$4\cdot 10^{-9}$	$1{,}6\cdot 10^{-7}$	$\sim 2perc$	
$n=100$	10^{-8}	10^{-6}	$\frac{2^{100}}{10^{10}}=\frac{(2^{10})^{10}}{10^{10}}\approx\frac{10^{}}{10^{}}$ $=10^{20}\approx 3\cdot 10^{13}év($	
$n=10^6$	10^{-4}	100		
$n=10^9$	10^{-1}	$10^8 s(3év)$		

Idő becslése

Egy program az $n=20$nagyságú bemenetere a választ 6 másodperc alatt számolja ki. Mennyi idő alatt számolja ki az $n=400$nagyságú bemenetre a választ, ha rendre $f(n)$=$n, n^2, \log n, 2^n$?

$f(n)=n \rightarrow 400 \overset{6s}{=} 20 \cdot 20 = 6 \cdot 20s = 2perc$

$f(n)=n^2 \rightarrow 400^2=(20 \cdot 20)^2 \overset{6s}{=} 20^2 \cdot 20^2 = 6 \cdot 20^2 = 40perc$

$f(n)=\log(n) \rightarrow \log 400 = \log 20^2 = 2 \cdot \overset{6s}{\log 20} = 2 \cdot 6s = 12s$

$f(n)=2^n \rightarrow 2^{400} \overset{6s}{=} 2^{20} \cdot 2^{380} = 6 \cdot 2^{380} = 6 \cdot (2^{10})^{38} > 6 \cdot 1000^{38} = 6 \cdot 10^{114}s =$

$= 6 \cdot \underset{\sim 2év}{10^7} \cdot \underset{univerzumbanlevőrészecskékszáma}{10^{80}} \cdot 10^{27}$

Nagyságrendek

$f, g: \mathbb{N} \rightarrow \mathbb{R}$

Ordó (nagy ordó)

$f(n) = O(g(n))$, ha $\exists c > 0, n_0 > 0, |f(n)| \leq c \cdot |g(n)|, \forall n > n_0$

Omega

$f(n) = \Omega(g(n))$, ha $\exists c > 0, n_0 > 0, |f(n)| \geq c \cdot |g(n)|, \forall n > n_0$

Theta

$f(n) = \Theta(g(n))$, ha $f(n) = O(g(n))$és $f(n) = \Omega(g(n))$, azaz $\exists c_{1,} c_2 > 0, n_0 > 0,$ $c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|, \forall n > n_0$

Példa

$f(n) = 3n^2 - 100n + 6$

$f(n) \overset{?}{=} O(n^2)$ $\quad |\, 3n^2 - 100n + 6 \,| \leq cn^2$ ha $n > n_0$

ha $n > 100$, $f(n) > 0$, tehát az abszolút érték elhagyható.

$3n^2 - 100n + 6 \leq cn^2$

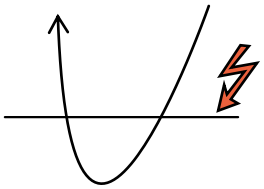
ha $c = 3$már biztos, hogy teljesül

Tehát $c = 3n_0 = 100$jó lesz, vagyis a kifejezés valóban $O(n^2)$

$f(n) \overset{?}{=} O(n^3)$ igen, mert $f(n) = O(n^2)$és $3n^2 < 3n^3, c = 3, n_0 = 100$

$f(n) \overset{?}{=} O(n)$ $\quad 3n^2 - 100\text{n} + 6 \leq c \cdot n (n \geq 100)$

$3n^2 - (100 + c)n + 6 \leq 0$Nem teljesül minden n>n0-ra, hiszen a parabola felül nyitott, valahol mindenképp átlépi az x tengelyt



$f(n) \overset{?}{=} \Omega(n^2)$ $3\text{n}^2 - 100n + 6 \geq cn^2$

$\geq 2n^2$

$c = 2, n_0 = 100$ jó lesz, tehát valóban $\Theta(n^2)$-es

$f(n) \overset{?}{=} \Omega(n)$ Ez is teljesül.

$f(n) = \Theta(n^2)$ Igaz, a legnagyobb nagyságrendű tag határozza meg.

$\log_a n = \Theta(\log_2 n)$ Ezért ezentúl mindig *kettes alapú logaritmust* használunk.

Szuperforrás keresése

Legyen $G = (V, E)$hurokélmentes irányított gráf

forrás: $s \in V$: egy csúcsból se megy bele él

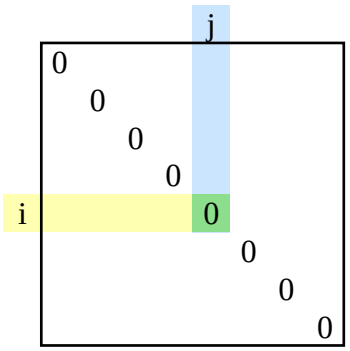
szuperforrás: $s \in V$: forrás és ∀csúcsba megy belőle él

A feladat tehát az, hogy találjuk meg a szuperforrást, ha létezik, vagy nem létezésének bizonyítása.

Megjegyzés

Egy gráfban legfeljebb egy szuperforrás lehet.

G szomszédossági mátrixa



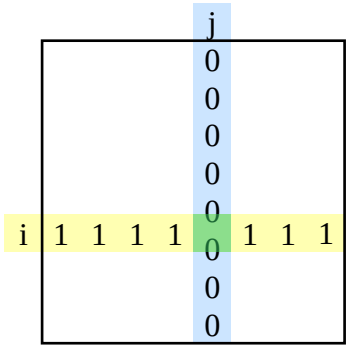
(i, j): i-ből j-be vezető élek száma

lépés: $A[i, j] =$?, vagyis van-e i-ből j-be él.

pontra ellenőrizzük, hogy szuperforrás-e

Egy pont ellenőrzése $2n - 2$lépés (az átló kihagyható)

∀ csúcsra $n(2n - 2) = 2n^2 - 2n = \Theta(n^2)$



b) $(i, j) \in E \Rightarrow j$nem jó

$(i, j) \notin E \Rightarrow i$ nem jó

Vagyis minden lépésben egy csúcsot ki tudunk zárni.

Az algoritmus:

$i = 1, j = u$ $V = \{1,2,3, \dots, n\}$

Amíg $i \neq j$

ha $A[i, j] \neq 0$ $j - -;$

ha $A[i, j] = 0$ $i + +;$

Ha van szuperforrás, akkor az csak $i$lehet ⇒ellenőrizzük, hogy valóban az-e.

Lépésszám: $\underset{\text{keresés}}{n - 1} + \underset{\text{ellenőrzés}}{2n - 2} = 3n - 3 = \Theta(n)$.

∀ algoritmus használ legalább $2n - 2$lépést (1 pont ellenőrzése).

Jelölje $T(n)$a legjobb algoritmus lépésszámát.

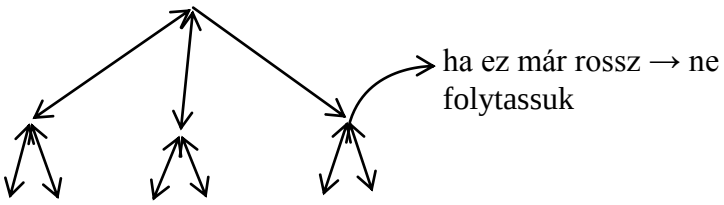
$T(n) \geq 2n - 2$

$T(n) \leq 3n - 3$

jobb alsó becslés: bármely algoritmus $n - 2$lépés után 2 jelöltet hagy: i-t és j-t. ∀-reannyi lépés kell, amit még nem tudtunk. Mindre maximum $\frac{n-2}{2}$kérdés vonatkozott. Ha a szuperforrás az, amelyikre a kevesebb kérdés vonatkozott, $2n - 2 - \frac{n-2}{2}$lépés kell még.

$$T(n) \geq n - 2 + 2n - 2 - \frac{n - 2}{2} = 2{,}5n - 3$$

2. előadás



Elágazás és korlátozás

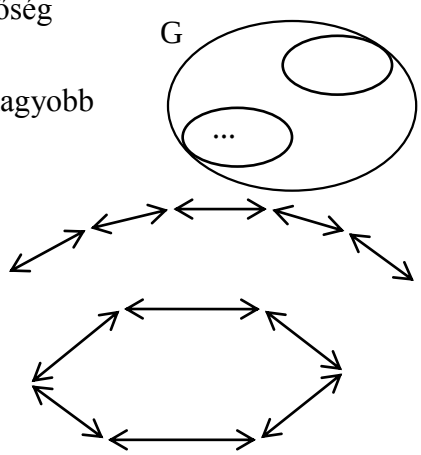
Példa

$G = (V, E)$Egyszerű és irányítatlan gráf

feladat: maximális független ponthalmaz meghatározása (független pontok között nem megy él).

1. algoritmus: ∀ponthalmazt kipróbálunk → 2^nlehetőség
2. algoritmus:

$MF(G)$: REKURZÍV algoritmus, meghatározza a legnagyobb független ponthalmazt a Ggráfban.



Az algoritmus inputja, a Ggráf kétféle lehet:

- minden pont foka ≤ 2: csak izolált pontok, körök és utak diszjunkt uniója. MF(G) meghatározása egyszerű: körnél és útnál minden második pont ugyanabba a halmazba kerül, hiszen nem fut közöttük él, függetlenek.
- Van olyan $v$csúcs, aminek foka ≥ 3: a gráfot két részre bontjuk, ezekre futtatjuk le az algoritmust (vagyis: ha az adott részben minden csúcs foka ≤ 2, akkor az első módszer alapján adjuk meg MF-et, ha nem, akkor ismét meghívjuk ezt a részt).

A 2 részre osztás:

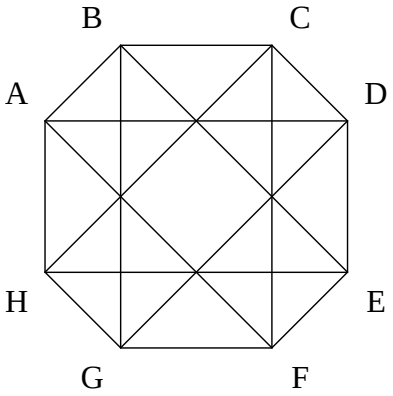
az egyik rész az, amelyikben v-n kívül az összes csúcs szerepel $S_1 = MF(G - v)$

a másik pedig a $v$csúcsnak és szomszédainak elhagyásával keletkezik $S_2 = \{v\} \cup MF(G - \{v\text{ésazösszesszomszédja}\})$

Legyen Sa nagyobb méretű S_1és S_2közül, vagy ha egyenlőek, akkor az egyik közülük, (pl. S_1)

Miután a rekurzió véget ért, $MF(G)$pont a kapott Shalmaz lesz.

Példa



$MF(\{ABCDEFGH\})$

— $deg(A) = 4 \geq 3$

$S_1 = MF(\{BCDEFGH\}) \rightarrow S_1 = \{BDFH\}$

$S_2 = \{A\} \cup MF(\{CEG\}) \Rightarrow S_2 = \{ACEG\}$

$S = \{BDFH\} = MF(G)$

— $deg(B) = 3 \geq 3$

$S_1' = MF(\{CDEFGH\}) \rightarrow S_1' = \{DFH\}$

$S_2' = \{B\} \cup MF(\{DFH\}) \Rightarrow S_2' = \{BDFH\}$

$S' = \{BDFH\} = S_1$

— $deg(C) = 3 \geq 3$

$S_1'' = MF(\{DEFGH\}) \rightarrow S_1'' = \{DFH\}$

$S_2'' = \{C\} \cup MF(\{EG\}) \Rightarrow S_2'' = \{CEG\}$

$S'' = \{DFH\} = S_1'$

— $deg(E) = 3 \geq 3$

$S''' = max\{S_1''', S_2'''\} = \{DFH\} = S_1''$

$S_1''' = \{DFH\}$

$S_2''' = \{E\} \cup MF(\{G\})$

$\Rightarrow S_2''' = \{EG\}$

$S_1''' = MF(\{DFGH\})$

$deg(G) = 3 \geq 3$

$S'''' = max\{S_1'''', S_2''''\} = \{DFH\} = S_1'''$

$S_1'''' = \{DFH\}$

$S_2'''' = \{G\} \cup \emptyset$

$\Rightarrow S_2'''' = \{G\}$

$S_1'''' = MF(\{DFH\})$

Lépésszám

$T(n)$: az $n$csúcsú gráf esetén legfeljebb hányszor hívódik meg az MF

$T(n) \leq 1 + T(n-1) + T(n-4)$

$G-v$ $G-vésszomszédai(min3)$

Bebizonyítható, hogy $T(n) \leq c^n (c > 0$ konstans), és az algoritmus lépésszáma $O(1{,}381^n)$.

Megjegyzés

Bár ez is exponenciális, lényegesen jobb, mint az előző:

$n = 30 \rightarrow \quad 2^{30} = (2^{10})^3 \approx (10^3)^3 = 10^9$

$1{,}381^{30} \approx 20000$

Példa

$G = (V, E)$egyszerű gráf; $| V | = n$

feladat: 3 szintű csúcsszínezés

1. algoritmus: ∀csúcs háromféle színt kaphat.

Próbáljuk végig → 3^nlehetőség.

Az algoritmus lépésszáma: $O(3^n \cdot n^2)$

2. algoritmus: az egyik színosztály pontjait kiválasztjuk, 2^nlehetőség (ennyi különböző részhalmaza van egy n elemű halmaznak, mindegyikre megnézzük, hogy jó-e). A maradékot két színnel színezzük → van gyors algoritmus.

Látni fogjuk, hogy a két színezés $O(n^2)$.

A második algoritmus lépésszáma: $O(2^n \cdot n^2)$.

$n = 30$-nál $2^n \approx 10^9$, $3^n \approx 2 \cdot 10^{14}$

Rekurzív eljárás

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

$\binom{n}{k}$

$n-1 \quad n-1$
$k-1 \quad k$

$n-2 \quad n-2 \quad n-2 \quad n-2$
$k-2 \quad k-1 \quad k-1 \quad k$
0
0

1 1
0 1 Itt pazarlunk, ha az $\binom{n-2}{k-1}$-et kétszer számoljuk ki, és emiatt végül exponenciálisan sok

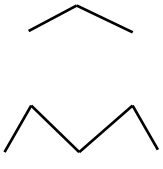
2 2 2
0 1 2
hívás lesz.

Dinamikus programozás

A megoldás részfeladatok megoldásaiból áll elő

⇒ rekurzió helyett részfeladatok fokozatos megoldása, eredmények eltárolása

„táblázat kitöltés” (pl. Pascal-háromszög a binomiális együtthatókra)



n-edik sor k-adik elem a keresett $\binom{n}{k}$.

Hátizsák probléma

$n$db tárgy, $s_1 \dots s_n$súlyok, $v_1 \dots v_m$értékek (mindkét halmaz elemei pozitív egészek).
Egy hátizsáknak b a súlykorlátja (és összesen egy hátizsákunk van).
$s_1 \dots s_n, v_1 \dots v_m, b > 0$ egész
Feladat: Kiválasztani a tárgyak egy $I = \{1 \dots m\}$részhalmazát úgy, hogy $\sum_{j \in I} \quad s_j \leq b$;
$\sum_{j \in I} \quad v_j$maximális

Példa

$m = 3$súlyok: 2,2,3, $b = 4$; értékek: 4,4,7.
Mohó pakolás: $I = \{3\}; v = 7$
Gondolkodás: $I = \{1,2\} v = 8$

1. algoritmus

$\forall I$-t kipróbálunk, a lehetőségek száma 2^n.

2. algoritmus (dinamikus programozás)

Táblázat:

	1	2	...		***a***		b
1	0	0	...	0	v_1	...	v_1
2							
3							
4							
...							
i							
m							

$T[i, a]$: $\{1,2, \dots, i\}$ tárgyak közül max. érték; súlykorlát: a
Kell: $T[m, b]$

$$T[1, a] = \begin{cases} 0, \text{ha} s_1 > a \\ v_1, \text{ha} s_1 \leq a \end{cases}$$

A táblázat inicializálása: az első sor kitöltése. Ha az első súlynál kisebb a súlykorlát, nem tehetjük bele, az érték 0, ha nagyobb vagy egyenlő, beletesszük a hátizsákba, az érték pedig v_1.

$$i > 1: T[i, a] = \begin{cases} T[i-1, a], s_i > a \\ max(T[i-1, a], \quad v_i + T[i-1, a - s_i] \quad) \end{cases}$$

i.nincs — benne van→súly csökken érték nő

$T[i, a]$mező kitöltése: az i.súlyt akkor tesszük bele, ha nő az érték, különben az előző érték marad (lemásoljuk a fölötte levő mezőt). A képlet azért jó, mert azt tudjuk, hogy az előző sorban az $a - s_i$súlyú elem a lehető legnagyobb összértékű súlyokat tartalmazta, a kitöltés ott optimális volt. Ha ehhez hozzáadjuk v_i-t, és nagyobb értéket kapunk, mint $s_i$nélkül, akkor ezzel növeltük a hátizsák értékét (a lehető legnagyobb mér-

tékben), kitöltés továbbra is optimális.

A fenti képlet alapján fentről lefelé, balról jobbra sorban kitöltjük a táblázat celláit.

Lépésszám: Egy elem kitöltése $O(1)$ →konstans lépés, az egész táblázat kitöltése pedig $m \cdot b \cdot O(1) = O(m \cdot b)$.

Bemenetek: $s_1, \dots, s_n, v_1, \dots v_n, b$

hossza: $\lceil \log(s_1 + 1) \rceil + \dots + \lceil \log(s_n + 1) \rceil + \lceil \log(v_1 + 1) \rceil + \dots + \lceil \log(v_n + 1) \rceil + \lceil \log(+1) \rceil$

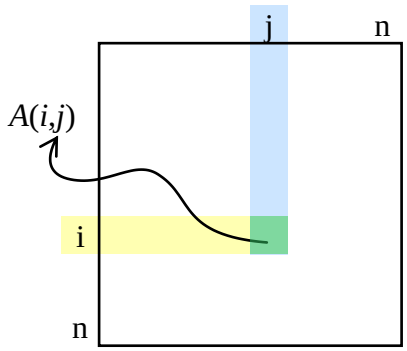
$O(m \cdot b) \lceil \log(b + 1) \rceil$-ben exponenciális (bemenetek hossza > m) → nem polinomiális → nem szeretjük, de kicsi b-nél gyors az algoritmus.

Megoldás

	1	2	3	4
1	0	4	4	4
2	0	4	4	8
3	0	4	7	8

Gráfok megadása

$G = (V, E); V = \{1 \dots n\}$



Szomszédossági mátrix

$A(i, j)$: Az *i*-ből *j*-be menő élek száma. (egyszerű gráfban *0* vagy *1*).

Irányítatlan gráfban a főátlóra szimmetrikus.

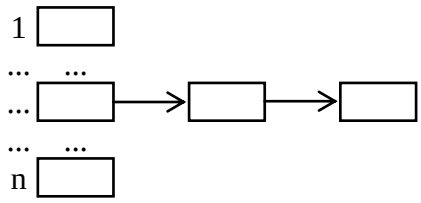
Súlyozott szomszédossági mátrix

Súlyozott gráf: $c: E \Rightarrow \mathbb{R}$súlyfüggvény.

Egyszerű gráf: $A[i,j] = \begin{cases} c(ij) \text{ha} (i,j) \in E \\ 0, \text{ha} (i,j) \notin E \end{cases}$, vagy $A[i,j] = \begin{cases} c(ij), \text{ha} (i,j) \in E \\ 0 \text{ha} i = j \\ *\text{vagy} \infty, \text{ha} (i,j) \notin E \end{cases}$

A gyakorlatban a *értéke egy jó nagy szám, nagyobb, mint az előforduló legnagyobb élsúly.

Éllistás megadás



i-ből kimenő élek láncolt listában
távolság: él másik vége + súly

Mérete

szomszédossági mátrix: $n^2 \frac{\text{bit}}{\text{szám}}$

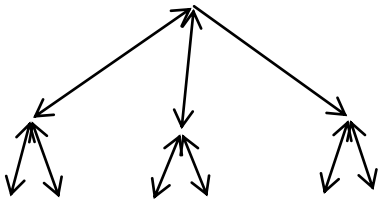
éllista: $O(n+e)$vagy irányítatlan esetben $O(n+2e)$, de mivel $n + 2e < 2\text{n} + 2e = 2(n+e) \Rightarrow O(n+e)$

Példák

	mátrix	éllista
$(i,j) \overset{?}{\in} E$	1 lépés	*i* fokszáma db. lépés (*i* listáján végig)
1 db. i-ből kimenő él	*n* lépés *i* során megy végig	1
G éleinek száma	$\Theta(n^2)$lépés *G*-n végig	$\Theta(n+e)$ egész struktúrán
izolált pont keresés	$\Theta(n^2)$	*n*

3. előadás

Szélességi bejárás



Leírása

$G = (V, E)$

angol neve: **Breadth First Search** = **BFS**

kiindulópontja: $v \in V$.

bejárva$[v] =$ igaz

Lista $L = (v)$ **FIFO** lista

amíg L nem üres

$x = L[1]$ (L első eleme); L-ből töröljük.

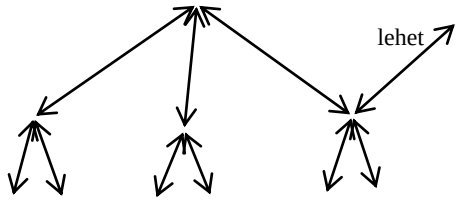
$\forall (x,y) \in E$-re ha bejárva$[y]$hamis, akkor bejárva$[y]$-t igazzá tesszük.

y-t L végére rakjuk.

Ha nem ∀csúcsot járhatunk be, választhatunk egy tetszőleges nem bejárt csúcsot és onnan folytathatjuk.

Az algoritmus tulajdonképpen bejárja a $v$csúcsot, ezután sorra az összes bejáratlan szomszédját, és közben fel is jegyzi őket az Llistában. Ha $v$szomszédait már mind bejártuk, akkor ugyanezt ismételjük $v$először, majd másodszor, ... bejárt szomszédjaira.

ha irányított:



irányított út:

ha jó helyről kezdjük, bejárja, ha nem, nem hatékony.
(természetesen fordított sorrendben is megpróbálhatjuk bejárni ugyanezt gráfot, és ekkor a feszítőerdő izolált pontok halmaza lesz)

A bejárás során keletkezik a szélességi feszítőerdő.

Lépésszám

Éllistás megadás

∀csúcs egyszer kerül a listába, minden élet egyszer nézünk meg (irányított esetben), tehát a lépésszám $O(n+e)$, ahol $n = |V|$, $e = |E|$. Irányítatlan esetben minden élet kétszer nézünk meg, ekkor a lépésszám $O(n+2e) = O(n+e)$

Mátrixos

x-ből kiinduló élek megnézése $n \Rightarrow O(n^2)$

Alkalmazása (példák)

1. G irányítatlan gráf összefüggő-e?

tetszőleges $v \in V$-ből ha a bejárás során ∀csúcsot elérünk, akkor összefüggő, ellenkező esetben nem.

$O(n+e)$ éllistás megadásnál

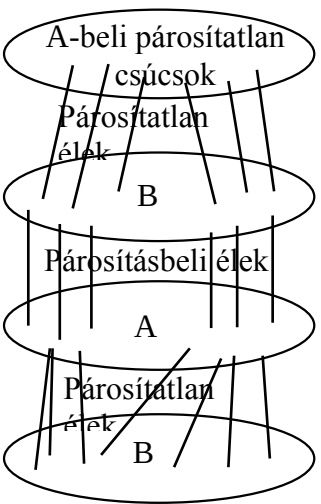
2. G irányítatlan, páros gráf összefüggő komponensei?

BFS → szélességi feszítőerdő fái az összefüggő komponensek. $O(n+e)$

3. Girányítatlan gráf páros gráf-e?

BFS, kiindulunk Acsúcshalmazból, bejárjuk az összes szomszédos Bcsúcsot, és így tovább BFS mentén. Az algoritmus egy hangyányit módosítani kell, ugyanis a nem bejárt csúcsokat is vizsgálni kell. Ha ellentmondásba kerülünk, vagyis egy már bejárt csúcsba ismét eljutunk, de most a másik halmazba kerülne, mint az előbb, akkor a gráf nem páros.

4. Girányítatlan páros gráfban maximális párosítás keresése (a maximális párosítás a legnagyobb számosságú független élhalmaz)



Algoritmus (magyar módszer)

- Keresünk egy egyszerű, könnyen megtalálható párosítást (akár üres párosítást).

- Szélességi feszítőerdőt indítunk az összes A-beli *párosítatlan csúcsból*. Itt A-ból B-be csak a *párosítatlan éleken* haladhatunk.
- Az így elért B-beli csúcsokból a *párosításbeli éleken* visszamegyünk A-ba
- A *párosítatlan éleken* ismét elmegyünk B-be...

Ezt addig folytatjuk, míg találunk olyan B-beli pontot találunk, amiből nem tudunk párosított élen továbbmenni. Ekkor az útban több a párosítatlan, mint a párosított él, ezeket felcserélve kapjuk meg a javítóutat.
Az algoritmus addig fut, míg van olyan B-beli pont, ahol elakadunk. Ha nem B-ben akadunk el, hanem A-ban, akkor nincs több javítóút.

Lépésszám:

bejárás: $O(n+e)$
párosítás: $\frac{n}{2}$ csúcs $\Rightarrow O(n(n+e))$

Ha G összefüggő, akkor $e \geq n-1 \Rightarrow O(n+e) = O(e)$, és így a párosítás lépésszáma $O(n \cdot e)$.

Az algoritmuson lehet még javítani, lehet $O(\sqrt{n} \cdot e)$is.

Tetszőleges nem páros gráfra is $\exists O(\sqrt{n} \cdot e)$algoritmus a maximális párosítás megtalálásához.

Maximális súlyú párosítás is létezik, $O(n \cdot e + n^2 \log n)$lépésszámmal.

1. Ggráf $v$csúcsából minimum hány élen át érjük el a többit?

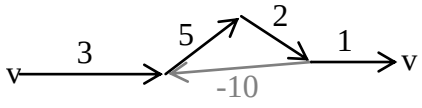
BFS – ∀elért csúcsnál letároljuk, hogy hányadik lépésben értük el (a bejáráson belül).

Legrövidebb (legkisebb súlyú) út keresése

Példa

$G = (V, E)$, $c: E \to \mathbb{R}$súlyfüggvény

út súlya (hossza): élsúlyok összege



$n \to 11$
1 kör a háromszögön $\to 8$
2 kör a háromszögön $\to 5$

Konklúzió

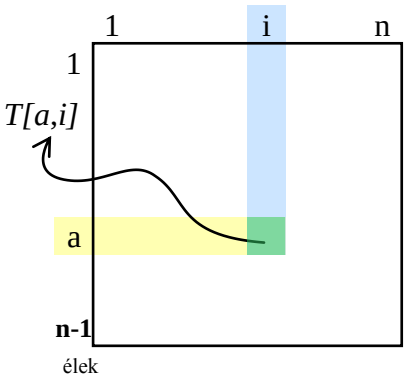
Nincs legkisebb súlyú út! Ennek oka a negatív súlyú él, pontosabban a negatív súlyú kör.

Ha nincs negatív súlyú kör, elég utakat keresni, célunk a legrövidebb (legkisebb súlyú) út keresése.

A minimális élszámú út nem feltétlenül a legjobb → a mohó algoritmus se jó.

Bellman-Ford algoritmus

Ez az algoritmus egy rögzített csúcsból az összes többibe meghatározza a legkisebb súlyú/legrövidebb utat.



Megjegyzés:

a legrövidebb út ezentúl a legkisebb súlyú utat jelenti.

$v = (1 \ldots n)$, a rögzített csúcs az 1.

$T[a, i]$: 1-ből i-be menő max. a darab élt tartalmazó utak közül a legrövidebb hossza.

$T[1, i] = c(1, i) \leftarrow (1, i)$él súlya

$$c(i,j) = \begin{cases} 0, i = j \\ (i,j), (i,j) \in E \\ \infty, (i,j) \notin E \end{cases}$$

$\forall j = 1, \ldots, nj \neq i$

$T[a, i] = min\{T[a-1, i], T[a-1, j] + c[j, i]\}$

Tehát minden $T[a, i]$mező kitöltésénél megvizsgáljuk, hogy melyiknek kisebb a súlya: az eggyel rövidebb, már meghatározott súlyú útnak ($T[a-1, i]$), vagy pedig valamelyik $a$hosszú útnak, ami két részből tevődik össze: egy $a-1$hosszú, 1. és j.csúcs közötti útból és a j.és i.csúcs közötti élből.

1-ből i-be menő legkisebb súlyú út súlya: $T[n-1, i]$

Lépésszám

$$\underbrace{n(n-1)}_{a\,táblázat\,mezőinek\,száma} \cdot \underbrace{(n-1)}_{\text{min. keresés}} = O(n^3)$$

Megjegyzések

A táblázatban csak az előző és az aktuális sort kell tárolni (elég csak 2nhely)

A legrövidebb út megtalálásához elég elrakni azt a j-t, amire a minimumot kapjuk (utolsó előtti pontja az útnak)

Bármely két pont közti legrövidebb út megkeresése

Ezt megtehetjük úgy, hogy minden pontra meghívjuk a Bellman-Ford algoritmust. Ekkor $t = n \cdot O(n^3) = O(n^4)$. Azonban a feladatot ennél hatékonyabban is meg tudjuk oldani, a Floyd algoritmus alkalmazásával.

Floyd algoritmus

```
F[i,j] = C[i,j]; i,j ∈ V
for (k = 1 to n) {
   for (i = 1 to n) {
      for(j = 1 to n) {
            F[i,j] = min(F[i,j], F[i,k] + F[k,j]);
      }
   }
};
```

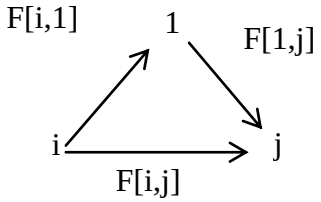
A végén $F[i,j]$az i-ből j-be menő legrövidebb út hossza lesz.
Az algoritmus úgy működik, hogy minden lépésben (k iterációja) újra kitölti a táblázatot úgy. Ekkor $F[i,j]$egy olyan $i \rightarrow j$út hossza, aminek belső pontjai az $1 \ldots k$csúcsok közül kerülnek ki.

Lemma

$\forall k$-ra: $F[i,j]$az i-ből j-be menő legrövidebb olyan út hossza, amelynek belső pontjai$\in \{1 \ldots k\}$.
Ebből az állítás $k = n$-re teljesül (minden pont lehet belső).

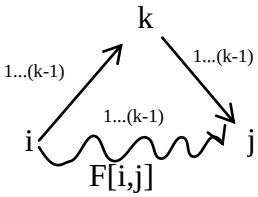
Bizonyítás

teljes indukcióval:
$k = 1$-re:



általános k-ra:

legjobb - van nem



$1 \ldots k$belső pontú élsorozat.
ilyen súlyú út is (nincs negatív kör)
lesz végtelen ciklus.

Lépésszám

$O(n^3)$

Megjegyzés

Ez a két algoritmus használható éllistában is, ekkor azonban a lépésszámok a fentiektől eltérőek.

Tranzitív lezárt keresése

$G(V,E)$tranzitív lezártja $G'(V,E')$
E': $(i,j) \in E'$ha G-ben van i-ből j-be irányított út.
A tranzitív lezárt keresésére egy módosított Floyd-algoritmust használunk, a Warshall-algoritmust.

Kezdetben $W[i,j]$= igaz, ha $(i,j) \in E$, ellenkező esetben pedig hamis.
Itt a ciklusok belsejében a $W[i,j] = W[i,j] || (W[i,k] \&\& W[k,j])$képlet áll.
Az algoritmus lépésszáma $O(n^3)$.

4. előadás

Legrövidebb út keresése

Dijkstra-algoritmus

Algoritmus leírása

$G(v,e)\ c: E \to \mathbb{R}$

c minden élre nemnegatív: $\forall e$-re $c(e) \geq 0$.

s a kezdőpont;

s-ből a többibe vivő legrövidebb út megtalálására szolgál.

D tömb, ∀csúcsra tárolja az aktuális legrövidebb utat.

Kezdetben $D(v) = c(s,v)$ $\quad (s,v)$ él súlya

$KÉSZ$ halmaz;

Kezdetben $KÉSZ = \{s\}$,

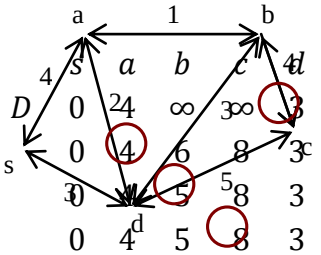
amíg $KÉSZ \neq V$

Legyen xa Dtömb legkisebb eleme, ami még nincs benne a $KÉSZ$tömbben.

$KÉSZ = KÉSZ \cup \{x\}$

$D[w] := min(D[w], D[x] + c(x,w)), \forall w \notin KÉSZ$

Példa



s	a	b	c	d	
0	4	∞	∞	3	$KÉSZ = \{s\}$
0	4	6	8	3	$KÉSZ = \{s,d\}$
0	4	5	8	3	$KÉSZ = \{s,d,a\}$
0	4	5	8	3	$KÉSZ = \{s,d,a,b\}$
0	4	5	8	3	$KÉSZ = \{s,d,a,b,c\} = V$

Állítás

Az algoritmus végén $D[x]$az s-ből x-be vezető legrövidebb út hossza, minden x-re (azaz az algoritmus jó).

Bizonyítás

$d(x)$legyen az s-ből x-be vezető legrövidebb út hossza.

Tegyük fel, hogy $\exists x$, hogy $d(x) \neq D[x]$.

Legyen $x$az elsőnek kézbekerülő ilyen csúcs.

Vegyünk s-ből x-be egy legrövidebb utat. Amikor először elhagyja a $KÉSZ$-t, $y \in KÉSZ, z \notin KÉSZ$.

Ekkor $D[x] > d(x) \underset{c \geq 0}{\geq} d(y) + c(y,z) \underset{x\text{az1.rossz}}{=} D[y] + c(y,z) \underset{\text{algoritmus}}{\geq} D[z] \Rightarrow$

indirekt feltevés

$D[x] > D[z]$. Ekkor az algoritmus nem választhatja x-et a $KÉSZ$-be (mert mindenképpen a kisebbet választja), ezzel valóban igazoltuk állításunkat.

Lépésszám

$$O(n) + (n-1)(\underbrace{n-1}_{\text{min. szám.}} + \underbrace{\text{konstans}}_{\text{frissítés}}(n-1)) = O(n^2).$$
kezdeti érték

Adatszerkezetek (adatstruktúrák)

adattípus, műveletek valamint a megvalósításuk

Példa

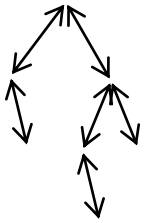
Lista: azonos típusú adatok;

műveletek: első, előző, következő, utolsó

megvalósítás: a használat szempontjából lényegtelen, hogy tömbbel, vagy (kétirányú) láncolt listával tesszük-e.

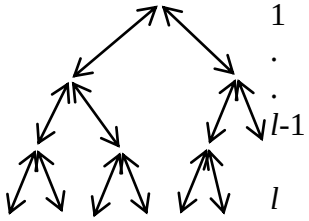
Bináris fa

Minden csúcsnak maximum két szomszédja lehet a következő szinten (a bal és a jobb **fia**) és a **gyökér** kivételével egy szomszédja van az előző szinten, a szülője. Amelyik csúcsnak nincs fia, azt **levél**nek nevezzük.



Teljes bináris fa

$0,1,\dots,l-1$.szinten minden csúcs megvan, az l. szinten az utolsó (jobb szélső) csúcsok közül hiányozhatnak.



Megvalósítása

Lehet mutatókkal (*pointer*ekkel), vagy tömbbel:

$A[i]$ bal fia $A[2\text{i}]$, jobb fia $A[2\text{i}+1]$, apja pedig$A\left[\frac{i}{2}\right]$.

Kupac

Műveletek

beszúr

mintör: minimális elemet visszaadja, és törli.

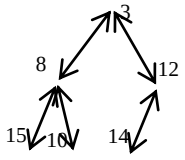
kupacépítés: adott elemeket kupacba rendezi.

Feltevés

Csupa különböző elemet tárolunk teljes bináris fában, melyben az elemek a csúcsok.

Kupactulajdonság

Minden $x$csúcsra kisebb az x-beli elem, mint a fiaiban levő elem.



⇒ min. elem a gyökérben.

A szintek száma: l.

Az i. szinten $(i < l)2^i$csúcs található.

l szintű teljes bináris fában $1 + 2 + 2^2 + \cdots + 2^{l-1} + 1 \leq n \leq 1 + 2 + 2^2 + \cdots + 2^l$

$2^l \leq n \leq 2^{l+1} - 1$

$\Rightarrow l = \lfloor \log n \rfloor$

⇒ ha egy kupacban $n$db elemet tárolunk, akkor ennek $\Theta(\log n)$szintje van.

Beszúr

Az új elem létrehozása (ha még van hely a legalsó sorban, akkor oda szúrjuk be, az első üres helyre, ha nincs, akkor az új sor bal szélére.).
Ha az apja nagyobb, mint az új elem, akkor kicseréljük vele. Addig ismételjük ezt a műveletet, míg az új elem az aktuális helyén kisebb lesz az apjánál, vagy a gyökérbe jut.
Lépésszám: $O(\log n)$, ahol na csúcsok száma.

Mintör

A legkisebb elem törlése.
Töröljük a gyökeret. ezután a gyökérbe betesszük a legutolsó elemet. A gyökeret cseréljük a *kisebbik fiával*, és ha ezután is sérül a kupactulajdonság, akkor addig cserélgetjük a kisebbik fiával, amíg helyre nem áll.
Lépésszám: $O(\log n)$

Kupacépítés

A legegyszerűbb módszer a megvalósítására a sorozatos $BESZÚR$. Ez $O(n \cdot \log n)$(van olyan eset, hogy $\geq c \cdot n \log n$).
Egy ennél lényegesen gyorsabb algoritmust szoktunk alkalmazni:
Alulról fel, jobbról balra haladva a részfát kupaccá tesszük, úgy, mint a $MINTÖR$-ben (a szülőt kicseréljük a kisebbik fiával). Szintenként fölfele haladunk, amikor elérjük a gyökeret, az algoritmus véget ér.

Lépésszám

2 összehasonlítás és legfeljebb 1 csere.
Cserék száma: i.szinten 2^icsúcs található, mindenre maximum $l - i$csere fordul elő (a szülőbeli elemet le kell vinni a legaljára, ez $l - i$darab szint).
Összesen tehát:

$$\sum_{i=0}^{l-1} 2^i(l-i) = \underbrace{1+1+\cdots+1}_{l} + \underbrace{2+2+\cdots+2}_{l-1} + \underbrace{4+4+\cdots+4}_{l-2} + \cdots + 2^{l-1} =$$

$$\underbrace{(1+2+4+\cdots+2^{(l-1)})}_{geometriaisor,q=2} + (1+2+4+\cdots+2^{(l-2)}) + \cdots + (1+2) + 1 =$$

$$(2^l - 1) + (2^{l-1} - 1) + \cdots + (4-1) + (2-1) =$$

$$2^{l+1} - 1 - (l+1) = 2^{l+1} - l - 2 < 2^{l+1} = 2 \cdot 2^l \leq 2n$$

$$\Rightarrow O(n)$$

Fogyaszt

További művelet a $FOGYASZT(x, a)$, mely a kupacban az $x$helyen lévő elemet a-ra cseréli ($a < x$), és utána helyreállítja a kupactulajdonságot. (a műveletet bővebben az 5. előadás anyaga tartalmazza).

5. előadás

Adatszerkezetek (adatstruktúrák)

Kupac

Fogyaszt

Egy $x$elemet csökkent, utána visszaállítja a kupactulajdonságokat. Problémát okozhat, ha $x$kisebb, mint az apja, ekkor megcseréljük az apjával, és ezt folytatjuk, amíg a kupactulajdonság helyre nem áll. A lépésszám≤szintszám, az algoritmus $O(\log n)$.

Max-kupac

Az eddig tárgyalt kupacok bináris min-kupacok voltak. Azonban léteznek egyéb kupacfajták is, mint a max-kupac, ahol az apa > fia reláció érvényesül az elemek között (a $MINTÖR$művelet helyett pedig $MAXTÖR$van).

D-kupac

A d-kupac egy teljes fa, ahol minden csúcsnak $d$darab fia van az $l-1$.szintig, kivéve az utolsó nem levél csúcsot.

Kupactulajdonság: apa < fia

Szintszám: $\Theta(\log_d n)$

$BESZÚR$: $O(\log_d n)$

$MINTÖR$: $O(d \cdot \log_d n)$

$KUPACÉPÍTÉS$: $O(n)$

$FOGYASZT$: $O(\log_d n)$

Példa: $d = \sqrt{n}$, $\log_d n = 2 \Rightarrow BESZÚR: O(1)$, $MINTÖR: O(\sqrt{n})$

(d értéke aszerint választandó, hogy melyik műveletet csináljuk gyakran)

Dijkstra-algoritmus

$D[]$értékeket kupacban tartjuk (bináris, minimum kupac).

A kupacon végrehajtjuk a $KUPACÉPÍTÉS$műveletet.

x elemet kiválasztjuk, végrehajtjuk a $MINTÖR$műveletet.

$D[w]$ frissítése: $FOGYASZT$

$w \notin KÉSZ$ és $(x, w) \in E$.

Lépésszám (ha G éllistával adott):

$$\underbrace{O(n)}_{\substack{kezd.ért \\ KUPACÉPÍTÉS}} + (n-1) \cdot \underbrace{O(\log n)}_{MINTÖR} + \sum_{x} ((x\text{-ből kimenő élek száma}) \underbrace{O(\log n)}_{FOGYASZT}) =$$

$$O(n) + O(n \cdot \log n) + O(\underbrace{e}_{élszám} \cdot \log n) = O((n+e) \cdot \log n)$$

Lépésszám (d-kupaccal):

$O(n) + (n-1) \cdot O(d \cdot \log_d n) + e \cdot O(\log_d n) = O((n \cdot d + e) \cdot \log_d n)$

$\Rightarrow d$ legyen $\frac{e}{n}$(az irányított gráf átlagos fokszáma) $\Rightarrow nd \approx O(e \cdot \log_d n)$

Ha például $e \geq n\sqrt{n} \Rightarrow d \geq \sqrt{n} \Rightarrow \log_d n \leq 2$
$\Rightarrow O(e)$lesz az algoritmus, tehát sok él esetén jó.
Megjegyzés: ∃olyan algoritmus, amelyre ∀gráfra a lépésszám $O(e + n \cdot \log n)$, de ehhez más adatszerkezet kell.

Keresési algoritmusok

Példa

$a_1, a_2, \ldots a_n$elemek adottak, valamint a $b$elem.

Kérdés

Van-e olyan i, hogy $b = a_i$?

Megoldás

A fő lekérdezésünk a $b \stackrel{?}{=} a_i$lesz, és ha $i = 1$-től $i = n$-ig $\forall a_i$-re végrehajtjuk a lekérdezést, akkor $O(n)$lépésben megoldjuk a feladatot.

Keresés rendezett listákban

Rendezett lista

$a_1 < a_2 < \cdots < a_n$
Kérdés: $b$szerepel-e a listában?
Lépés: b? a_i, ebből megtudjuk, hogy b kisebb-e, vagy nagyobb-e a-nál, vagy azt, hogy egyenlő-e vele.

Lineáris keresés

```
for i = 1 to n
   b?a[i];
```

Az algoritmus akkor ér véget, ha $b = a_i$, vagy ha $b > a_i$(mert ekkor b nincs a listán).
A lépésszám ezért $O(n)$.

Bináris (felezéses) keresés

b? $a_{\lceil \frac{n}{2} \rceil}$ ha egyenlő, akkor megvan a keresett a,

ha kisebb, akkor $a_1 \ldots a_{\lceil \frac{n}{2} \rceil}$között folytatjuk a keresést (ugyanígy)

ha nagyobb, akkor $a_{\lceil \frac{n}{2} \rceil + 1} \ldots a_n$között folytatjuk.

$O(\log n)$ lépésben végrehajtható a keresés, mert minden lépésben felezzük a halmaz méretét, melyben vizsgálódunk → A lépésszám $\lfloor \log n \rfloor + 1$(A +1a megmaradt egy darab elem összehasonlítása b-vel).

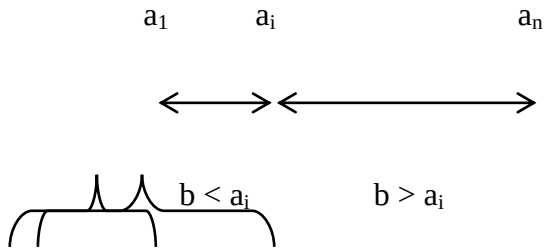
Tétel

A bináris keresés az adott feltételekkel optimális.

Bizonyítás

Tetszőleges algoritmusra

A tételt úgy bizonyítjuk be, hogy megmutatjuk, tetszőleges más algoritmusra van olyan eset, amikor a keresés tovább tart, mint a bináris keresés leghosszabb esete.



Barchobázzunk, vagyis az ellenfél kitalál egy $b$számot, és mi a lehető legkevesebb kérdésből rá szeretnénk jönni, hogy mi a b. Egy kérdés: $b?\, a_i$(a kérdőjel művelet eredménye a <, = vagy > relációk közül az egyik).
Ellenfelünk trükkos, és azt szeretné, hogy a játék minél tovább tartson. Ezért nem talál ki előre egy $b$számot, hanem mindig úgy válaszol, hogy a lehetséges a_j-k közül a $b$mindig a nagyobb halmazban legyen. Így bármilyen algoritmussal egy kérdésben legfeljebb felére csökkenthetjük a megmaradó lehetőségek számát, vagyis $n$elem esetén legalább log$n$kérdésre van szükségünk, hogy csak egy elem maradjon. Tehát a bináris keresés optimális.

Megjegyzés

Ez a keresés tömbökben jó, láncolt listákhoz lineáris keresést kell alkalmaznunk.

Interpolációs keresés

Akkor, ha tudjuk, hogy hol van a $b\rightarrow$ akörül keresünk.

Rendező algoritmusok

$a_1 \ldots a_n$különböző → növekvő sort akarunk készíteni belőle.

Beszúrásos rendezés

Minden$k = 1,2, \ldots, n-1$-re: ha az első $k$elem már rendezett, a $k+1$-ediket beszúrjuk közéjük a helyére.
A $k+1$.elem helyének keresése:

Lineáris kereséssel

Az összehasonlítások száma:

Bináris kereséssel

Az összehasonlítások száma: $\sum_{k=1}^{n-1} \log k+1 \leq (n-1) \cdot \sum_{k=1}^{n-1} \log k =$
$= n-1+\log(n-1)! < n+n(\log n-1{,}442) = n(\log n-0{,}442) = O(n\log n)$

Mozgatások száma

$$1+2+\cdots+n-1 = \frac{n(n-1)}{2}$$

Tárolók összehasonlítása

Tömbben tárolva az adatokat az összehasonlításokat $O(n \cdot \log n)$lépésben tudjuk elvégezni, a mozgatásokat pedig $O(n^2)$lépésben.
Láncolt listás tárolás esetén az összehasonlításokat $O(n^2)$lépésben tudjuk elvégezni, míg a mozgatásokat $O(n)$lépésben.

Összefésülés

Leírás

$b_0 < b_1 < \cdots < b_{r-1}$
$c_0 < c_1 < \cdots < c_{s-1}$csupa különböző → $d_0 < d_1 < \cdots < d_{s+r-1}$

$d_0 = min(b_0, c_0)$

$i = 0, j = 0,$

ha $b_i < c_j$akkor $d_{i+j} = b_j$, és i++

ha $b_i > c_j$akkor $d_{i+j} = c_j$és j++

Az algoritmus véget ér, ha $i = r$és $j = s$.

Összehasonlítások száma

$s + r - 1$darab összehasonlítást végzünk.

Összefésüléses rendezés

Leírás

A két tartomány: $a_1 \dots a_{\frac{n}{2}}$, $a_{\frac{n}{2}+1} \dots a_n$Ezeket külön-külön rendezzük, majd a kettőt összefésüljük.

Összehasonlítások száma

Szintenként$< n$ darab összehasonlítást végzünk el, log$n$szint található → $O(n \cdot \log n)$.

Mozgatások száma

Szintenként n, összesen $O(n \cdot \log n)$darab.

Lépésszám

Az algoritmus lépésszáma tehát $O(n \cdot \log n)$.

6. előadás

Rendezések

Kupacos rendezés

$KUPACÉPÍTÉS, MINTÖR, \underset{n}{MINTÖR}, \dots$

$O(n) + n \cdot O(\log n) = O(n \cdot \log n)$

Buborék rendezés

$j = n - 1, \dots, 1$

$i = 1, \dots, j$

Ha $a_{i+1} < a_i$akkor $a_{i+1} \Leftrightarrow a_i$

Összehasonlítások száma

$$(n-1)+(n-2)+\cdots+1=\frac{n\cdot(n-1)}{2}=\Theta(n^2)$$

A cserék száma≤ az összehasonlítások száma

Állítás

A buborékrendezés rendez.

Bizonyítás

$j=n-1$végén a legnagyobb elem lesz az a_n→ ehhez többet nem nyúlunk

$j=n-2$ végén a legnagyobb elem lesz az a_{n-1}→ szintén a helyére került

...

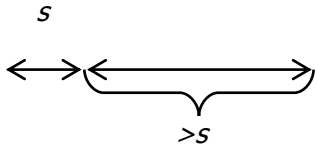
$j=1$ végén az $n-1$.legnagyobb elem az $a_2$lesz, tehát $a_1$csak a legkisebb elem lehet.

Gyors rendezés (quicksort)

Leírása

Választunk egy $s=a_i$véletlen elemet. Ezután s-től jobbra és balra rendezzük az elemeket. ($O(n)$ összehasonlítás)

Az algoritmus előnye, hogy nem kell összefésülni a végén a két tartományt.



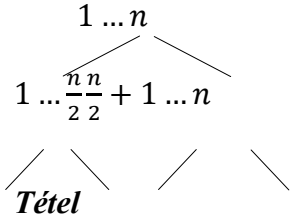
Példa

Mindig a legkisebb elemet választjuk: $n-1$összehasonlítás

Az összehasonlítások száma: $n-1+n-2+\cdots+1=\frac{n(n-1)}{2}=\Theta(n^2)$

Példa

Mindig a középső elemet választjuk: $n-1$összehasonlítás → fele akkora lista.



szintenként $<n$összehasonlítás
log$n$szint van
$n\log n$

Tétel

A gyorsrendezésnél az összehasonlítások átlagos száma: $O(n\log n)\neg B$.

Ha biztosan gyors futás kell, akkor nem ez az optimális algoritmus, de ha sokszor használjuk, akkor jó.

Tétel

$a_1 \dots a_n$csupa különböző elemek. Bármely algoritmus egy lépésben két elemet tud összehasonlítani, kimenetük tehát kétféle lehet ($a_i ? a_j$ $<$ vagy $>$)és minden lehetséges bemenetet rendez legfeljebb k összehasonlítással ⇒ $k \geq \log(n!)$

$n! = (\frac{n}{e})^n \cdot \sqrt{2\pi n} \cdot$ kicsi → $\log(n!) = \Omega(n\log n)$

$\log(n!) = O(n\log n)$

$\log(n!) = \Theta(n\log n)$

(vagyis az alsó becslés is $(n\log n)$)

Bizonyítás

Az alapötlet ugyanaz, mint a bináris keresésnél. Bebizonyítjuk, hogy minden algoritmusnál van olyan eset, ami $\log(n!)$ideig fut.
Kezdetben $n!$lehetséges sorrend van.
Minden összehasonlítás két részre vágja a még lehetséges sorrendeket.
A végén egyetlen lehetőség marad.
Mivel minden lépésben megmaradhat a lehetőségeknek a fele, ezért kell, hogy legyen legalább $\log(n!)$lépés.

Kulcs-manipulációs rendezés

Láda rendezés (bin sort)

A:alaphalmaz $=$ lehetséges elemek(kulcsok) halmaza
Algoritmus: $\forall x \in A$-hoz létrehoz egy $B[x]$ládát (lista)
$i = 1 \dots n$ a_i-t a $B[ai]$lista végére rakja
a listák tartalmát sorban kiírja

Lépésszám

$$\underset{\text{lista létrehozása}}{O(|A|)} + \underset{\text{bepakoljuk}}{O(n)} + \underset{\forall\text{elem és láda kiolvasása}}{O(n+|A|)} = O(n+|A|)$$

Ha $|A| \leq c \cdot n$, vagyis minden elem legfeljebb c-szer fordulhat elő $\rightarrow O(n)$
Szükséges, hogy az alaphalmaz értelmes méretű legyen.

Megjegyzés

Ládaelrendezésnél az egyforma elemek sorrendje nem változik.

Példák

$A \subseteq A_1 \times A_2 \times \dots \times A_k$(azaz k koordinátájuk van)
Pl. dátum: év, hónap, nap
Szavak: $A_i = abc$
Ezekre a ládarendezés $O(n+|A|) = O(n+|A_1|\cdot|A_2|\cdot \dots \cdot|A_k|)$

Lexikografikus rendezés

Leírás

$A_n$rendezés = lexikografikus rendezés: $(s_1, \dots, s_k) < (t_1, \dots, t_k)$, ha van olyan j: $s_i = t_i$ $i < j$, és $s_j < t_j$(az első olyan számít, amiben különböznek)
Ez a rendezés lassú, az utolsót sokszor használjuk.

Radix rendezés

Leírás

Ládarendezést hajtunk végre a k. koordináta szerint az összes elemen, majd a $k-1$.koordináta szerint, és így tovább, egészen az első koordinátáig.

Példa

a_1: CCAB, a_2: DCBA, a_3: ABAC, a_4: ACAC.

A rendezés menete:

$kiind.$	1.	2.	3.	4.
$CCAB$	$DCBA$	$CCAB$	$ABAC$	$ABAC$
$DCBA$	$CCAB$	$ABAC$	$CCAB$	$ACAC$
$ABAC$	$ABAC$	$ACAC$	$ACAC$	$CCAB$
$ACAC$	$ACAC$	$DCBA$	$DCBA$	$DCBA$

Tétel

A radix rendezés rendez.

Bizonyítás

Elég megmutatni, hogy ha $(s_1, \dots, s_k) < (t_1, \dots, t_k)$, akkor az algoritmus végén az $s$megelőzi t-t.

$\exists j: s_j < t_j, s_i = t_i$,ha$i < j$.

Radix: k. koord., ..., $(j+1)$.koord., nem tudjuk a sorrendjüket.

j. koord., $\rightarrow$ $s$előbb lesz, mint t.

$j-1., \dots 1.$ koord $\rightarrow$nem változik a sorrend, mert a ládarendezés az egyforma elemek sorrendjét nem változtatja. Ezzel az állítást bebizonyítottuk.

Lépésszám

$O(n+|A_k|) + O(n+|A_{k-1}|) + \dots + O(n+|A_1|) =$

$O(k \cdot n + |A_1| + |A_2| + \dots + |A_k|)$.

A ládarendezésben szorzat, itt összeg

Például: $|A_i| \leq c \cdot n \quad i = 1, \dots k \rightarrow O(k \cdot n)$

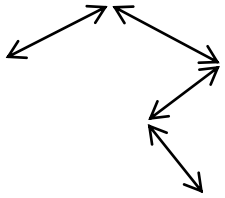
$|A_i| \leq c \qquad i = 1, \dots k \; k = \log n \rightarrow O(n \log n)$

számok rendezésekor az n szám jegyeinek száma log n

Adatszerkezet

Bináris fa

(nem feltétlen teljes)



Bejárás

preorder – csúcs, bal részfa, jobb részfa

inorder – bal részfa, csúcs, jobb részfa

posztorder – bal részfa, jobb részfa, csúcs

Megjegyzés

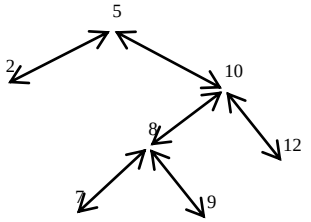
Ezt úgy egyszerű megjegyezni, hogy a pre-, in- és poszt- előtagok a *gyökér* kiolvasására vonatkoznak.

Példa

preorder: $5, \underset{\square}{2}, 10, 8, \underset{\square}{7}, 9, 12$

inorder: $\underset{\square}{2}, \underset{\square}{5}, 7, 8, 9, \underset{\square}{10}, 12$

posztorder: $\underset{\square}{2}, 7, 9, 8, \underset{\square}{12}, 10, 5$



Bináris keresőfa

Leírás

$KERES$, $BESZÚR$, $TÖRÖL$, MIN, MAX, $TÓLIG$

bináris fa

Az elemeket csúcsokban tároljuk (elem ↔ csúcs kölcsönösen egyértelműen megf.)

(feltevés: csupa különböző elem van)

Keresőfa tulajdonság

bal részfa elemei < csúcs eleme < jobb részfa elemei – teljesül minden csúcsra.

Példa

$7 < x < 10$

Keresés

$KERES(x)$: $gyökér?$ x =megtaláltuk
<keresést a gyökér$jobb$részfájában folytatjuk
>keresést a gyökérbalrészfájában folytatjuk

Ha nem tudunk lépni, a keresett elem nincs a fában.

Lépésszám

$O(l)$, ahol l a szintszám.

Beszúrás

$BESZÚR(x)$: $KERES(x)$

- ha talál, akkor nem rakjuk be
- ha nem talál, ahova lépni akart (de nem tudott), ott kell létrehozni egy új csúcsot x értékkel.

Lépésszám

$O(l)$

Elgondolkodtató kérdés

Milyen sorrendtől lesz cikcakkos a fa?

7. előadás

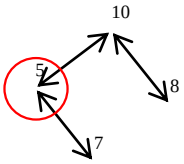
Bináris keresőfa (folytatás)

Műveletek

Legkisebb elem keresése

$MIN()$- visszaadja a legkisebb elemet.

A gyökértől minden lépésben balra kell menni, és ahol elakadunk, ott van a legkisebb. Fontos, hogy ez nem feltétlenül levél lesz (lásd ábra).



Lépésszám

$O(l)$, ahol l a szintek számát jelöli.

Legnagyobb elem keresése

$MAX()$- visszaadja a legnagyobb tárolt elemet.

A gyökértől minden lépésben jobbra kell menni, és ahol elakadunk, ott található a keresett elem.

Lépésszám

Szintén $O(l)$.

Állítás

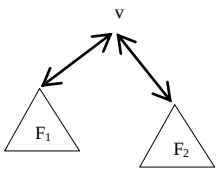
Bináris keresőfa inorder bejárása a tárolt elemeket növekvő sorrendben adja vissza.

Bizonyítás

$F_1 v F_2$

→ végig a megfelelő helyén van a gyökér (v)

→ Minden csúcs a megfelelő helyén lesz, mert mindegyik pont „annyiadik” helyen van, ahányadik a sorban.



Elem törlése

$TÖRÖL(x)$- x-et törli a fából.

- keresés → $KERES(x)$
- törlés
- a fa újrarendezése

$KERES(x)$: ha nem talál, akkor az algoritmus kész.

Ha x levél, akkor töröljük a csúcsot, és nem szükséges rendezni.

Ha x-nek egy fia van, akkor megszüntetjük x-et, és a fiához tartozó részfát betoljuk a helyére.

Ha x-nek két fia van, akkor vegyük a jobb oldali részfájának a minimális elemét (jelöljük y-nal), és azt tegyük x helyére. y-nak biztosan csak egy fia volt, ezért törlése egyszerű. Legyen ez a csúcs y. y-nak biztosan csak egy fia van, tehát törlése egyszerű.

Lépésszám

$O(l)$ennél a műveletnél is.

Több elem keresése

$TÓLIG(x, y)$- azon a csúcsokat adja vissza, melyekre igaz, hogy $x \leq a \leq y$.
Először keressük meg *x* helyét. Ehhez legfeljebb $l = \log n$lépés szükséges.
$KERES(x)$csúcsból az inorder bejárást követve megyünk *y*-ig.
Ez a művelet gyors, ha a bináris keresőfa minden csúcsából létezik egy mutató az inorder bejárás szerinti következő elemre (inorder fonál).

Lépésszám

Ha létezik inorder fonál, akkor a $TÓLIG$lépésszáma $O(l + t)$, ahol *t* a megfelelő elemek száma.

Tétel

Ha egy kezdetben üres bináris fába *n* elemet beszúrunk, akkor az átlagos összlépésszám $O(n\log n)$(különböző sorrendekre nézve).
A fa átlagos magassága $O(\log n)$.

Cél

A cél a fa konkrét meghatározása, vagyis ne átlagos értékekről beszéljünk → kiegyensúlyozás.

AVL-fa (leírása a tankönyvben)
piros-fekete fa

Piros-fekete fák

Leírás

- bináris keresőfa
- minden csúcsnak 0 vagy 2 gyermeke van
- csak belső csúcsban tárolunk elemeket
- minden csúcs PIROS vagy FEKETE
- a gyökér FEKETE
- a levelek FEKETÉK
- PIROS csúcsnak nincs PIROS gyermeke
- minden $v$csúcsra igaz, hogy minden v-ből levélhez vivő úton ugyanannyi FEKETE csúcs van.
  $fm(v)$: fekete magasság, *v* nem számít bele!
  $m(v)$: a leghosszabb út, amellyel *v*-ből levélig érünk.

Állítás

$m(v) \geq fm(v) \geq \frac{m(v)}{2}$minden *v* csúcsra.

Bizonyítás

$m(v) \geq fm(v)$: fm-be csak a feketéket számoljuk. Egy úton két piros nem jöhet egymás után, tehát az úton a csúcsoknak legalább a fele fekete kell, hogy legyen.

Állítás

A $v$gyökerű fában tárolt elemek száma $\geq 2^{fm(v)} - 1$.

Bizonyítás

$m(v)$szerinti indukció

$m(v) = 0$ (Ha $v$levél $\Rightarrow fm(v) = 0, 2^{fm(v)} - 1 = 2^0 - 1 = 0$(levélben nem tárolunk)

$m(v) > 0 \Rightarrow$ van két fia, $x$és y.

$\frac{m(x)}{m(y)} \leq m(v) - 1 \Rightarrow$ alkalmazható az indukciós feltevés, a tárolt elemek száma a $v$gyökerű fában $\geq \underset{v}{1} + 2^{fm(x)} - 1 + \underset{\text{fiai}}{2^{fm(y)} - 1} = 2^{fm(x)} + 2^{fm(y)} - 1 \geq 2 \cdot 2^{fm(v)-1} - 1 = 2^{fm(v)} - 1$

$fm(x) \leq fm(v)$

$fm(x) \geq fm(v) - 1$

$fm(y) \geq fm(v) - 1$

Ha $n$elemet tárolunk, mit tudunk mondani a fa magasságáról?

Állítás

Ha $n$elemet tárolunk, akkor a gyökér magassága $\leq 2 \cdot \log(n+1)$.

Bizonyítás

v- gyökér

$n \geq 2^{fm(v)} - 1 \geq 2^{\frac{m(v)}{2}} - 1$

$\log(n+1) \geq \frac{m(v)}{2}$

$m(v) \leq 2 \cdot \log(n+1)$

Következmény

$KERES, MIN, MAX$: $O(\log n)$

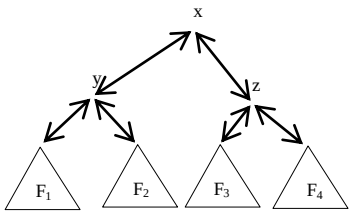
Itt majd az eljárás a levelekben akadhat el (bináris keresőfa – nem tudsz lépni, piros-fekete fa – lehetséges lépni, de levél van).

Műveletek

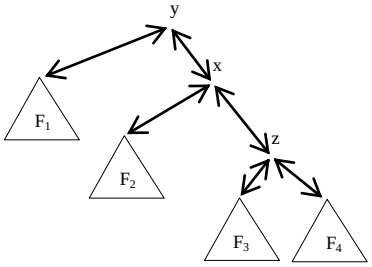
Az alábbi műveletek kis mértékben változnak, mert el lehet a fát rontani velük (pl. két piros kerülne egymás alá, vagy a fekete magasság elromlik).

$BESZÚR()$ és $TÖRÖL()$. A helyreállítás eszköze a forgatás.

Forgatás jobbra:



Forgatás balra:



Fontos: keresőfából keresőfát csinál

Beszúrás

Rakjuk be az új elemet az eredeti keresőfába. Az új belső csúcs legyen piros: *z* csúcs.
Ha *z* gyökér → ő az első elem, könnyű beszúrni.
Ha *z* nem gyökér, akkor létezik apja: *x*.
Ha *x* fekete, akkor készen vagyunk.
Ha *x* piros, akkor biztosan van apja, ő pedig biztosan fekete (*t* csúcs). *x*-nek van testvére is, *y* (hiszen minden elemnek két fia van).
Baj akkor van, ha *t* apja piros. Ekkor, ha *y* fekete, $FORGATÁS$-t végzünk, és így *z t* és *x* közé kerül.

Tétel

$BESZÚR()$lépésszáma $O(\log n)$, és legfeljebb két forgatást használ.
$TÖRÖL()$ lépésszáma $O(\log n)$, és legfeljebb három forgatást használ.

Gyakorlati megvalósítása

- keresőfa
- levelek: a gyakorlatban egy darab csúcs helyettesíti az összes levelet, mert azokban nem tárolunk elemeket, és így nem foglal fölösleges erőforrásokat (így már nem fa, de ez nem baj a használat szempontjából).

$KERES$, MIN, MAX, $TÓLIG$, $BESZÚR$, $TÖRÖL$- ugyanaz, mint a keresőfánál, csak más adatszerkezettel.

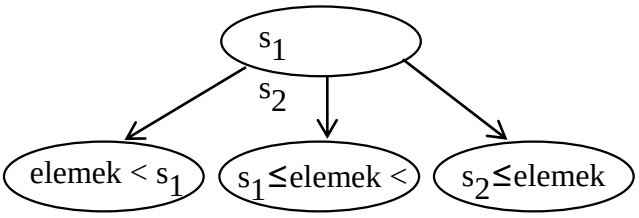
8. előadás

2-3 fa

Leírása

Gyökeres, szintezett fa. A levelek egy szinten vannak.
A nem leveleknek (belső csúcsoknak) kettő vagy három fia van (ha az elemek száma nagyobb, mint egy).
Elemeket csak a levelekben tárolunk.

Belső csúcs



Az ábrán az látható, mikor a csúcsnak három fia van. A két gyerekű csúcs annyiban különbözik ettől, hogy nem $s_1$és s_2, hanem csak $s$értéket határozza meg, a két fiú közül az egyik az s-nél kisebb elemeket,a másik meg az s-t és annál nagyobbakat tartalmazza.

Állítás

Az $n$elemet tároló 2-3 fa magassága $\Theta(\log n)$.

Bizonyítás

A k.szinten levő csúcsok száma $\geq 2^k$, és $\leq 3^k \rightarrow \log_3 n \leq k \leq \log n$

Műveletek

$KERES$: az útjelzők szerint lép a következő szintre $\Theta(\log n)$

Az elemek a levelekben növekvő sorrendben vannak.

MIN: balra megyünk, $\Theta(\log n)$.

MAX: jobbra megyünk $\Theta(\log n)$.

$TÓLIG(a, b)$: $KERES(a)$, ha a levelek sorba vannak láncolva, akkor ezen a láncon végigmegyünk. $\Theta((\log n) + t)$, ahol ta találatok száma.

$BESZÚR$: $KERES$ új levelet kell berakni. Két gyerekes csúcsnál egyszerűen beillesztjük, és három gyereke lesz, három gyerekes csúcsnál csúcsvágást kell alkalmaznunk, vagyis egy szinttel feljebb szúrunk be egy új csúcsot. Ha ez a művelet felmegy a gyökérig, akkor új gyökeret kell készítenünk, és nő a fa magassága. A művelet $\Theta(\log n)$lépésszámú.

$TÖRÖL$: $KERES$. Ha a megtalált levél három gyerekes családban van, akkor töröljük a levelet, és frissítjük az útjelzőket a gyökér felé vezető úton.

Ha a család két gyerekes, akkor töröljük a levelet, majd igazságosan elosztjuk a gyerekeket. Ha van három gyerekes szomszédos testvér, akkor vele, ha a szomszédos testvérnek is csak két gyereke van, akkor egy szinttel feljebb törlünk, és összevonjuk a 2 családot egy darab 3 gyerekesre. Végezetül átállítjuk az útjelzőket. Ez az algoritmus is $\Theta(\log n)$lépéses.

B-fa

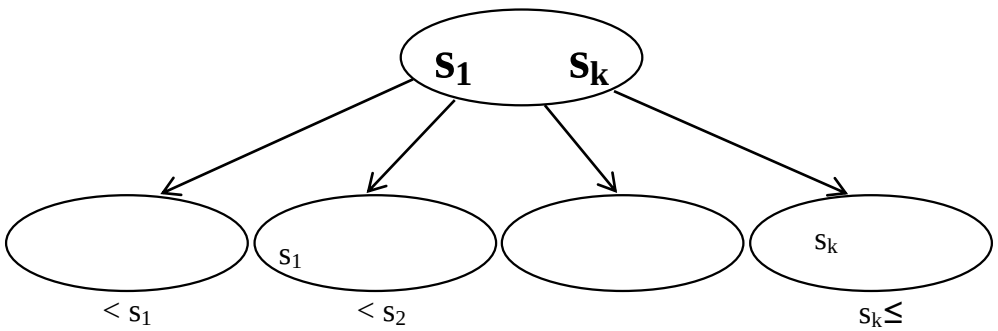
$B_m$fa

Leírása

Gyökeres, szintezett fa. A leveleket egy szinten tároljuk, és csak a levelekben tárolunk értékeket. Minden belső csúcsnak legfeljebb $m$fia van, és minden belső csúcsnak, ami nem gyökér legalább $\left\lfloor \frac{m+1}{2} \right\rfloor$fia van. A gyökérnek legalább két fia van, ha egynél több elemet tárolunk.

$m = 3$ esetén megegyezik a 2-3 fával.

Belső csúcs



Műveletek

$KERES, MIN, MAX$: mint a 2-3 fánál, $\Theta(l)$lépésben végrehajtható műveletek, ahol la szintek száma.

$BESZÚR$: m-nél kevesebb gyerek mellé létre tudjuk hozni az új gyereket.

m gyerek mellé való beszúrásnál csúcsvágást kell alkalmaznunk. Az egyiknek , a másiknak $\left\lceil\frac{m-1}{2}\right\rceil$db fia van.

$TÖRÖL$: A 2-3 fa algoritmusa megfelelő.

Állítás

$n$elemet tároló $B_m$fa magassága $\Theta\frac{\log n}{\log m}$.

Bizonyítás

k.szinten levő csúcsok száma $\leq m^k$

$$\geq 2 \cdot \left\lfloor\frac{m+1}{2}\right\rfloor^{k-1}$$

$$m^k \geq n \geq 2 \cdot \left\lfloor\frac{m+1}{2}\right\rfloor^{k-1}$$

$$m^k \geq n \Rightarrow k \geq \frac{\log n}{\log m}$$

$$n \geq 2 \cdot \left\lfloor\frac{m+1}{2}\right\rfloor^{k-1} \Rightarrow \frac{n}{2} \geq \left\lfloor\frac{m+1}{2}\right\rfloor^{k-1} \Leftrightarrow \frac{\log\frac{n}{2}}{\log\left\lfloor\frac{m+1}{2}\right\rfloor} + 1 \geq k$$

$$\log\frac{n}{2} \leq \log n$$

$$\log\left\lfloor\frac{m+1}{2}\right\rfloor \geq \log\frac{m}{2} \underset{m\geq 4}{\geq} \frac{\log m}{2}$$

$$\Rightarrow k \leq \frac{\log\frac{n}{2}}{\log\left\lfloor\frac{m+1}{2}\right\rfloor} + 1 \leq \frac{\log n}{\frac{\log m}{2}} + 1 = \frac{2 \cdot \log n}{\log m} + 1 \underset{n\geq m}{\leq} \frac{3 \cdot \log n}{\log m}$$

Tehát $\frac{\log n}{\log m} \leq k \leq \frac{3 \cdot \log n}{\log m}$

Megjegyzés

A külső táraknak lassú a beolvasása, ezért némi optimalizáció szükséges: a levelekben nem egyetlen elemet tárolunk, hanem annyit, amennyi a fizikai lapra ráfér (a fizikai lap a beolvasás egysége).

m megválasztása: egy belső csúcs = egy fizikai lap.

Szintszám: a lapbeolvasások száma.

Nem feltétlenül csak a levelekben tárolunk elemeket, hanem a belső csúcsokban is.

Külső tárak

Hash

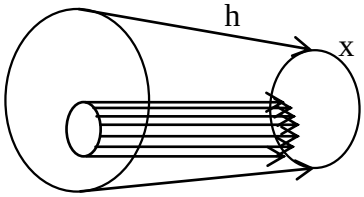
(olyan, mint a ládarendezés)

$KERES, BESZÚR, TÖRÖL$

A lehetséges elemek száma nagy.

Elemhalmaz: K =kulcsok halmaza

A ténylegesen használt elemek kulcsainak száma lényegesen kisebb (nagy luxus minden kulcsnak egy láda, mert a nagy részük üres).



$|x| \ll |K|$

: hash függvény

Példa

K:emberek, x:az év napjai, h:születésnap.

Probléma: már 23 ember között ½-nél nagyobb valószínűséggel van kettő, akinek ugyanarra a napra esik a születésnapja. A várható egybeeső párok száma ötven ember között három, 200 ember között 54 – ez elég magas, tehát a függvény nem túl jó a célra.

K: marslakók, $|x| = 669$(ilyen hosszú egy év a Marson).

Már 31 marslakó között van ½-nél nagyobb valószínűséggel azonos születésnap, tehát az ütközéseket nem tudjuk elkerülni az x halmaz növelésével → meg kell oldani kezelésüket.

Vödrös hash

$h: \rightarrow X = \{0,1,2, \dots, M-1\}$

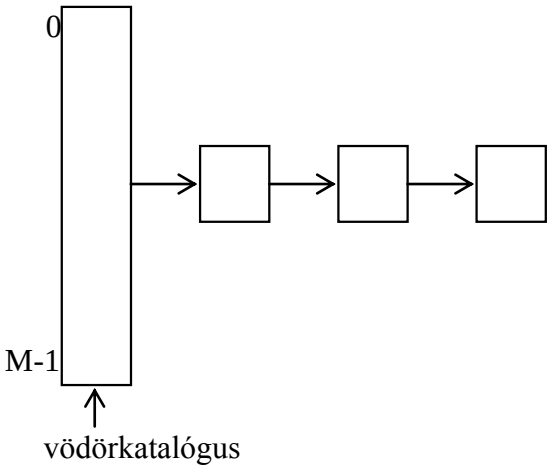
s elem helye: $h(s)$indexű lista.

$KERES(s)$: $h(s)$listában lineárisan keres.

$BESZÚR(s)$: $h(s)$listába berak

$TÖRÖL(s)$: $h(s)$listából töröl

$O(n)$ lépés.



Ha $h$közel egyenletesen szórja szét az elemeket, a listák várható hossza $\sim \frac{n}{M}$.

Megjegyzés

Külső táron szokásos használni (vödörkatalógus lehetőség szerint a memóriában), a listák elemei lapok, egy lapon több rekordot is tárolunk.

Ha Llap kell az összes elem tárolásához, akkor az átlagos lapelérés-szám $\frac{L}{M}+1$, vagyis átlagosan ennyi lapot kell megvizsgálnunk ahhoz, hogy megtaláljuk a keresett elemet. A +1 lépés a vödörkatalógus elérése. Például ha $M \simeq 1{,}2L$, az átlagos lapelérés kettőnél kisebb.

Nyitott címzésű hash

(memóriában)

Egy tömbben tároljuk az elemeket (T). $s$helye: $T[h(s)]$.

Ugrások: $h_0(s)=0, h_1(s), h_2(s), \dots h_{M-1}(s)$a $0{,}1, \dots M-1$egy permutációja.

Próbasorozat

$h(s)+h_0(s)=h(s)$
$h(s)+h_1(s)$
$h(s)+h_2(s)$ $(modM)$minden helyet kipróbál.
...
$h(s)+h_{M-1}(s)$

9. előadás

Külső tárak (folytatás)

Nyitott címzésű hash (folytatás)

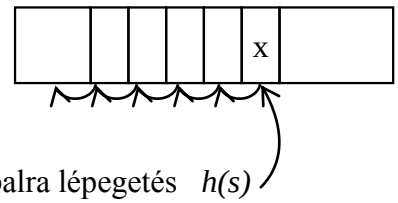
$KERES$: addig megy a próbasorozaton végig, amíg nem talál egy *üres*et (ahol soha nem volt senki), vagy megtalálja a keresett elemet.

$BESZÚR$: a próbasorozat első *szabad* helyére berakja (ahol most épp nincs senki)

$TÖRÖL$: $KERES$, ha talált, törli és beállít „törölt” bitet (azért, hogy a keresés ne álljon meg rajta, de be lehessen szúrni).

Lineáris próbálás

$h_i(s)=--i(modM)$



Ha két próbasorozat találkozik, onnan együtt mennek tovább → elsődleges csomósodás

Kvadratikus próbálás

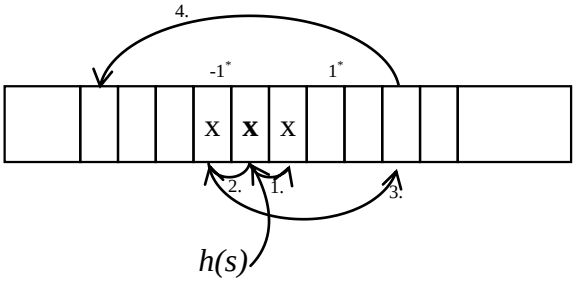
(álvéletlen próba)

Ugrás a kezdőponthoz képest:

$h_{2j-1}(s) = j^2 (mod M)$ ptlan.

$h_{2j}(s) = -j^2 (mod M)$ ps.

$1 \le j \le \frac{m}{2}$ között



⇒kusza lesz...

egy foglalt blokkon (csomón) belül különböznek a próbasorozatok, de ha $h(s) = h(t)$, akkor a próbasorozatok is azonosak lesznek → másodlagos csomósodás

Helytelen példa

Ha $M = 8$, $h_0 = 0$, $h_1 = 1$, $h_2 = 7$, $h_3 = 2^2 = 4$, $h_4 = -4 = 4(8)$, $h_5 = 9 \equiv 1$(tehát $h_3 = h_4$; $h_1 = h_5$, ami nem jó).

$h_6 = 7 = h_2$, $h_7 = 16 \equiv 0(8) = h_0$→ ez nem permutáció, tehát nem jó.

Helyes példa

$M = 7$, $h_0 = 0$; $h_1 = 1$; $h_2 = -1 \equiv 6(7)$; $h_3 = 2^2 = 4$; $h_4 = -4 \equiv 3(7)$; $h_5 = 3^2 = 9 \equiv 2(7)$; $h_6 = -2 \equiv 5(7)$

Ez jó, mert a 0, … ,6elemek egy permutációja.

Állítás

Ha $M = 4k + 3$és prím, akkor jó $0, k_1 \dots k_{n-1}$permutációja a$0, \dots, n-1$-nek

Bizonyítás

1)

$$h_{2j-1} = h_{2i-1} \Leftrightarrow i^2 = j^2 (M) \Leftrightarrow M \mid (j^2 - i^2) = (j-i) \cdot \underset{j,i \le \frac{M}{2}}{(j+i)}$$

$$\Leftrightarrow M \mid \underset{\text{csak, ha } j=i}{(j-i)} \text{vagy} M \mid \underset{j+i<M}{(j+i)}_{\text{(nem lehet)}}$$

Következtetés: a páratlan indexűek (négyzetszámok) különbözőek.

2)

A -1-szereseik is (a páros indexűek is).

3)

Lehet-e páros és páratlan indexű egyenlő?

$h_{2j-1} = h_{2i} \Leftrightarrow j^2 \equiv -i^2 (M)$ → van-e ilyen$i$ésj?

Feltehetjük, hogy $i, j \neq 0$

j, i relatív prímek M-hez, ezért létezik olyan t $\quad i \cdot t \equiv 1(M)$

$$i^2 \cdot t^2 \equiv 1(M)$$
$$j^2 \cdot t^2 \equiv -i^2 t^2 = -1(M)$$
$$(j^2 t^2)^{2k+1} \equiv (-1)^{2k+1} \equiv -1(modM)$$
$$(jt)^{2(2k+1)} = (jt)^{4k+2} = (jt)^{M-1} \equiv -1(M)$$

Ez ellentmond az Euler-Fermat tételnek → $(jt)^{M-1} \equiv 1(M)$(M prím!)

Ellentmondásra jutottunk, tehát a módszer jó.

(az egész módszer azon múlik, hogy nincs olyan x:

$x^2 = -1(modM)$ → a -1nem négyzetszám)

Kvadratikus próbálás (folytatás)

→ másodlagos csomósodás

($h(s) = h(t)$ → próbasorozathoz)

Kettős hash

$h_i(s)$függjön s-től

Tipikusan: $h'(s)$ második hashfüggvény

$h_i(s) = -i \cdot h'(s)(modM)$ feltétel: $h'(s), M = 1$

Hash függvények

Elvárások

A fő elvárásaink a hash függvényekkel szemben, hogy legyenek gyorsan számolhatóak és kevés ütközés merüljön fel (kenje szét az elemeket véletlenszerűen)

Megoldások

Osztómódszer

$h(s) = s(modM)$

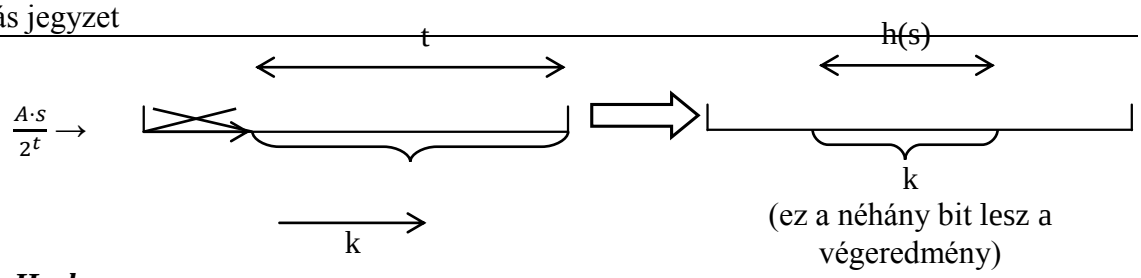
Ebből következik, hogy nem jó, ha Mkettő hatványa, mert ilyenkor csak az utolsó bitek levágása történik.

Szorzómódszer

$$h(s) = \underbrace{\lfloor \{\beta \cdot s\} \cdot M \rfloor}_{\substack{\text{törtrész} \\ \text{alsó egészrész}}}$$

$\Rightarrow h(s) \in \{0, \dots M-1\} \qquad M = 2^k$

β eredetileg irracionális szám, gyakorlatban: $\beta = \frac{A}{2^t}$, ahol Apáratlan egész.



Hash lépés-száma

Lehet lineáris: $O(n+M)$

Átlagos lehet: $O(\log n)$.

Példa (átlagos lépésszámok)

1. Ha $\frac{n}{M} = \frac{2}{3}$(telítettség)

	sikeres	sikertelen
$lineáris$	2	3
$kvadr.$	1,8	~3
$kettősh.$	1,5	~3

2. Ha $\frac{n}{M} = \frac{4}{5}$

	sikeres	sikertelen
$lineáris$	3	13
$kvadr.$	2,2	4,8
$kettősh.$	2,9	5

Keresés lépésszáma

Lineáris keresés rendezett listában

$$\underbrace{\frac{n+1}{2}}_{\text{sikeres}} = \frac{1+2+\cdots+n}{n} \quad \underbrace{\frac{\frac{n\cdot(n+1)}{2}+n}{n+1}}_{\text{sikertelen}} \simeq \frac{n}{2}+1$$

$\frac{n}{2}$ mindkettő

Bináris keresés

Sikertelen: $\log n$

Sikeres:

Keresőfa

$O(\log n)$

Ehhez képest a hash-elés konstans lehet (jó esetben). Hirtelen lineáris is lehet. Akkor jó a *hash*, ha elhúzódhat a keresés, de általában, *átlagban gyors* kell.

Nem egyenletes eloszlás

Lineáris keresés

Zipf-eloszlás

A szavak eloszlása egy természetes szövegben.

Az i.leggyakoribb valószínűsége: $\frac{c}{i}$.

Sikeres keresés: $\frac{n}{\log n}$(átlagos).

80-20 eloszlás

Az esetek 80%-át megoldjuk az idő 20%-a alatt. A többi 20%-ot nem az idő 80%-a alatt oldjuk meg, hanem a maradék 20% 80%-át oldjuk meg az idő 20%-ában, és így tovább, rekurzívan.

Az i.legnagyobb valószínűség: $\frac{c}{i^{(1-\vartheta)}}$ ϑkonstans

Sikeres keresés: $0{,}12 \cdot n$(átl.)

Gráfok

Mélységi bejárás

(Depth First Search – DFS)

„bátor felfedező”

Addig megyünk előre, amíg van felfedezetlen út, ha elértünk a végére, visszalépünk az előző csúcshoz, és másik irányban próbálkozunk újra …

$mb(v)$ algoritmus, a gráf $v$csúcsból induló mélységi bejárás

$bejárva[v] := igaz;$

$\forall w: (v,w) \in E:$

Ha $bejárva[v] := hamis$, akkor $mb(w)$;

Lépésszám

Éllistás megadással $O(n+e)$az algoritmus lépésszáma, minden csúcsot és élet egyszer járunk be.

Mélységi szám

Jelzése: $msz[v]$

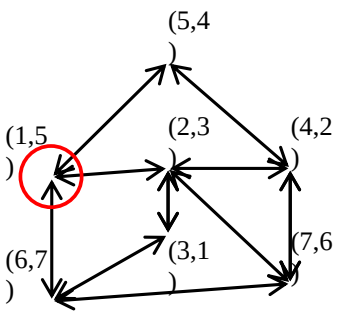
Csúcs jellemzője, voltaképpen azt jelöli, hogy „hányadiknak érünk bele” a csúcsba. Akkor kap értéket, amikor $bejarva[v] = igaz$lesz.

Befejezési szám

Jelölése: $bsz[v]$

Csúcs jellemzője, azt jelöli, hogy „hányadiknak fejezzük be” az adott csúcsot. Akkor kap értéket, amikor az $mb(v)$algoritmusban a $v$csúcsnak már csak bejárt szomszédai vannak.

Példa



Járjuk be a gráfot!

A bekarikázott csúcstól indulunk. a jelölés a következő:

$(\underset{msz}{2}, \underset{bsz}{3})$

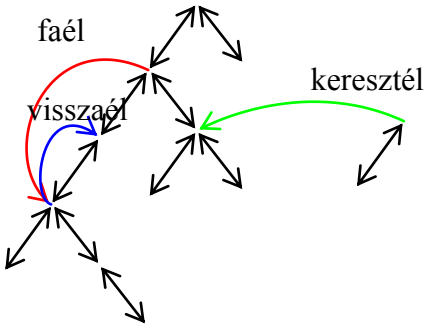
Ha még nem voltunk mindenhol, folytassuk az algoritmust! (lásd: bal alsó sarok).

10. előadás

Mélységi bejárás (folytatás)

Mélységi feszítőerdő

Elemei azon élek, amelyek bejárásban pontba vittek (faélek).



Élek osztályozása

A G irányított gráf élei egy bejárás szerint négyfélék lehetnek. Már a bejárás közben meg tudjuk mondani az élek típusát.

Egy $(x, y) \in E$

faél, ha még nincs $msz[y]$-nak értéke.
előreél, ha $msz[x] < msz[y]$
visszaél, ha $msz[x] > msz[y]$, és nincs $bsz[y]$-nak értéke
keresztél, ha $msz[x] > msz[y]$, és $bsz[y]$-nak van értéke.

Probléma csak azzal van, hogy a mélységi szám nem elégséges a vissza- és a keresztélek megkülönböztetéséhez. Ebben a megkülönböztetésben van segítségünkre a befejezési szám, a visszaéleknél még nincs $bsz[y]$-nak értéke, míg a keresztéleknél van.

Ezt a négy típust rögtön oda is írhatjuk az élekhez, ha úgy gondoljuk, hogy hasznunkra lehet a későbbiekben (és számos esetben így is van).

Irányítatlan gráf esetén kevesebb éltípus van, csak a faél és a visszaél marad meg.

Irányított körmentes gráfok

Girányított gráf, nincs irányított kör benne, azaz $Gdag$(**directed acyclic graph**) gráf.

Tétel

Gpontosan akkor dag, ha egy mélységi bejárásban nincs visszaél.

(Vagy másképpen $Gdag$ ⇔egy mélységi bejárásban nincs visszaél)

Bizonyítás

„⇒”

A megfelelő faélek plusz a visszaél egy irányított kört alkot.

A dag-ban nincs irányított kör, következésképpen a mélységi bejárásban nem létezhet visszaél.

„⇐”

Indirekt módon bizonyítunk. Tegyük fel, hogy van irányított kör G-ben. Vegyük ennek a körnek azt a pontját, aminek legkisebb a mélységi száma (x).

Ha a kör többi pontja az $x$gyökerű részfában van, és $y$az x-et a körben megelőző pont,

akkor az (y, x)visszaél. Ez egy ellentmondás, hiszen nincs a gráfban visszaél. Tehát Gvalóbandag

A másik lehetőség az az, hogy van a körnek a részfán kívül is pontja. Ez azt jelenti, hogy van egy olyan él, mely külső, nem részfabeli ponthoz vezet. Ez az él nem lehet faél, mert akkor nnek a részfának lenne az éle. Nem lehet előreél sem, ugyanezen okból. Feltettük, hogy visszaél nincs. Kizárásos alapon csakis keresztél lehet. Ez a keresztél csak úgy keletkezhetett, hogy a bejárás során már korábban érintettük, és ezért nem része a fának. Viszont ezért a kezdőpontjának mélységi száma kisebb, mint az x-é. Ez ellentmondás. Tehát G-ben nincs irányított kör, vagyis dag.

Minden lehetőségnél ellentmondásra jutottunk, tehát a tételt mindkét irányból bebizonyítottuk.

Következmény

Ha a Gegy mélységi bejárásában nincs visszaél, akkor semelyik mélységi bejárásában nem lesz.

Bizonyítás

Ha nincs visszaél, akkor a gráf dagnincs visszaél benne.

Megjegyzés

A keresztélekre ez nem igaz. Például $u$irányított út (lásd az ábrán, minden él faél benne). Visszafele menve minden él keresztél lenne.

Következmény

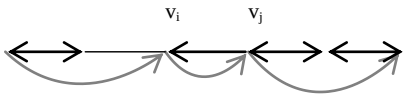
Azt, hogy egy gráf dag-e, könnyen el tudjuk dönteni algoritmikusan. Éllistás megadás esetén megadásnál $O(n + e)$lépésben eldönthető. A felhasznált algoritmus DFS. $Gdag$ ⇔nem létezik visszaél.

Topologikus rendezés

Definíció

A Gcsúcsainak egy olyan $v_1, v_2, \ldots, v_n$sorrendje, amelyre igaz, hogy ha $(v_i, v_j) \in E$, akkor $i < j$.

Szemléltetés



Tétel

G-nek akkor és csak akkor van topologikus rendezése, ha dag.

Bizonyítás

Odafele („⇒”):

$(v_i, v_j) \in E \rightarrow$ nem lehet benne kör.

Visszafele(„⇐”):

Ha dag, akkor van benne nyelő, azaz olyan pont, amiből nem megy ki él. dag-ban

nincs kör, ezért tetszőleges pontból egy kimenő él mentén elindulva nem juthatunk vissza, tehát egyszer vége lesz az útnak → ez a pont a nyelő.

Legyen $v_n$nyelő. v_n-et elhagyjuk a gráfból, dagmarad, így van nyelő: v_{n-1} Ezt is elhagyjuk, és így tovább. Ez egy jó topologikus sorrend lesz. Az állítást ezzel bebizonyítottuk visszafelé is.

Lépésszám

Annak eldöntése, hogy van-e topologikus elrendezés, és ha van, akkor ennek megadása $O(n+e)$lépésből lehetséges.

Másik megoldás

DFS. Topologikus rendezés: a bszszerinti csökkenő sorrend. Ez jó, mert kisebb bsz-ből nagyobba csak a visszaél vinne, de ilyen nincs, hiszen daggráfról beszélünk. Ez az algoritmus $O(n+e)$-s.

Legrövidebb út keresése

dagban s-ből a többi pontba vivő legrövidebb utak megtalálhatók $O(n+e)$lépésben éllistás megadásnál.

Algoritmus

$DFS\rightarrow$ topologikus rendezés ezzel $O(n+e)$lépést használtunk el.

A topologikus elrendezésben $v_1, v_2, \dots, v_n, \quad (v_i, v_j) \in E \Rightarrow i < j$ (csak jobbra mennek élek).

Legyen $s = v_1$, kezdetben $s$minden $v_j$szomszédjára $d(s, v_j) = c(s, v_j)$, az összes többi csúcsra $d(s, v_j) = \infty$.

$d(s, s) = 0$

$i = 1, \dots, n-1$

Ha $(v_i, v_j) \in E$akkor $d(s, v_j) = min\{d(s, v_j), d(s, v_i) + c(v_i, v_j)\}$.

Végül $d(s, v) = k$a keresett értékek

Lépések: $v_i$kimenő fokszáma

Összesen $O(n+e)$.

Leghosszabb út keresése

dagban s-ből a többibe vivő leghosszabb utak $O(n+e)$megtalálhatóak, az előző algoritmusban minhelyettmax kell.

Megjegyzés

Általánosságban egy gráfban a leghosszabb út keresése nem ismert, polinom idejű algoritmus.

PERT

Egy összetett munkánál, mely több részfeladatból áll, és közöttük lehetnek olyanféle megkötések, mint hogy a háznak előbb kell alapot csinálni, mint tetőt, (matematikai nyelven leírva Aelőbb kell, hogy bekövetkezzen, mint B) és hogy mennyivel előbb

(életközeli példaként: meg kell száradnia a betonnak).
Ebből kapunk egy súlyozott gráfot. Van egy kezdőpont, mikor nekifogunk a feladathoz, és van egy végpont, mikor befejezzük.
Ilyenkor a kezdő és a végpont között a leghosszabb útra van szükségünk, mert az lesz egyenlő a legrövidebb végzési idővel. Tehát a leghosszabb út keresésének van gyakorlati haszna!
A gráf (ha jól csináltuk) daglesz, és a keresett érték a leghosszabb út forrás és nyelő között). Ezt az utat szokás nevezni **kritikus út**nak, éleit pedig **kritikus tevékenység**eknek (természetesen lehet több kritikus út is a megfelelő gráfban).
A szóban forgó gráfot $PERT$diagrammnak szokták nevezni, mely a **Program Evaluation and Review Technique** kifejezés rövidítése.

PERT módszer

$d(s, v_j) = max\{d(s, v_i) + c(v_i, v_j)\}$
$(v_i, v_j) \in E$

$O(n + e)$, ha fordított éllista van (bemenő élekből)
Ez alkalmazható minimumra is,.ekkor maxhelyett minkell.

Gráfok erős összefüggősége

Erősen összefüggő a Ggráf, ha bármely két pontja között van irányított út.

Erősen összefüggőség eldöntése

1. algoritmus

csinálunk minden $v$pontból DFSbejárást.
Erősen összefüggő a gráf akkor és csak akkor lesz, ha mindegyik mélységi feszítőerdő fa. (ha v-nél fát kapunk, akkor minden csúcs elérhető irányított úton v-ből)
$n \cdot O(n + e) = O(n^2 + ne)$

2. algoritmus

Tetszőleges $v$pontból DFSbejárás, ha nem jut végig, akkor már most befejezzük az algoritmust.
Ezután újra a $v$pontból kezdünk egy DFSbejárást, csak nem a G-n, hanem a megfordítottján.
G erősen összefüggő, ha mindkét mélységi feszítőerdő fa.

Helyesség bizonyítása

Az első bejárásnál fát kapunk → v-ből minden pont elérhető irányított úton G-ben.
A második bejárásnál fát kapunk → v-ből minden pontból elérhető irányított úton Gfordítottjában, tehát $v$minden pontból elérhető irányított úton G-ben.
Ebből az következik, hogy van egy olyan pontunk, amelyből minden más pontot el tudunk érni, és őt is minden más pontból, tehát ha van egy $a$és egy $b$csúcsunk, akkor a-ból elmegyünk v-be, és onnan b-be, és viszont.

Erősen összefüggő komponensek nem erősen összefüggő gráfban

Két komponens között legfeljebb csak egy irányban mehetnek az élek, különben egy komponenst alkotnának. Tehát a komponensek gráfja (ahol a csúcsok a komponenseket jelképezik) daglesz.

Erősen összefüggő komponensek megtalálása

G-ben DFS, majd Gfordítottjában is DFSbejárás, minden fát a még bejáratlan pontok közül a legnagyobb befejezési számúval kezdünk (az előző bejárásból maradt bsz).

Állítás

Az erősen összefüggő komponensek a második bejárás fái → $O(n+e)$

Bizonyítás

Nem bizonyítjuk, nem lesz számon kérve.

Minimális költségű feszítőfa

$G=(V,E)$irányítatlan egyszerű, összefüggő gráf, súlyozott, $c: E \to \mathbb{R}$

Cél

$F \subseteq E$feszítőfa, melynek minimális a súlya.

11. előadás

Piros-kék algoritmus

Kék szabály

$X \in V, X \neq \emptyset, X \neq V$, X-ből nem megy kék él $V-X$-be, akkor egy legkisebb súlyú színtelen élet kékre színezünk.

Piros szabály

Ha egy Ckörben még nincs piros él, akkor egy legnagyobb súlyú színtelen élét pirosra színezzük.

Algoritmus

Kezdetben Gminden éle színtelen.

Kék és piros szabályt alkalmazzuk tetszőleges sorrendre, és tetszőleges ponthalmazra, körre.

Vége: ha egyik sem használható.

Tétel

Az algoritmus végén a kék élek egy minimális súlyú feszítőfát alkotnak.

Bizonyítás:

S: színtelen élek

K: kék élek

P: piros élek

Kezdetben $E=S$, $K=P=\emptyset$

$E = S \cup K \cup P$ felbontás $TAKAROSSZÍNEZÉS$, ha van olyan F_{min} feszítőfa G-ben, hogy $F \supseteq K$és $F \cap P = \emptyset$.

Lemma1

Az algoritmus minden lépése után takaros a színezés.

Bizonyítás: indukcióval

Kezdetben E = S takaros.

Vegyünk egy takaros színezést:

$E = S \cup K \cup P$ takaros, $\exists F$min. feszítőfa, $F \supseteq K$, $F \cap P = \emptyset$.

Kék szabályt használunk az $X \subset V$halmazra, $g \in E$élet színezzük (kékre).

Ha $g \in F$, akkor Ftovábbra is jó, a színezés takaros.

Ha $g \notin F$, a fában út van $g$végpontjai között, ennek van olyan éle, ami Xés $V - X$között megy. Legyen g'ilyen $\Rightarrow F' = F - \{g'\} \cup \{g\}$is feszítőfa.

g'színe: nem piros, mert $g' \in F$, és piros él nem lehetne a fában; nem kék a kék szabály miatt, g'kilép X-ből $\Rightarrow g' \in S \Rightarrow c(g') \geq c(g) \Rightarrow c(F) \geq c(F')$, de $c(F)$minimális $\Rightarrow c(F) = c(F')$ $\Rightarrow F'$minimális feszítőfa, ami mutatja, hogy az új színezés is takaros.

Piros szabály: F'minimális feszítőfa, ami mutatja, hogy az új színezés is takaros. A piros szabályt alkalmazzuk a Cponthalmazra (körre). A $g$élet színezzük (pirosra).

Ha $g \in F$, akkor Ftovábbra is jó, a színezés takaros.

Ha $g \notin F$, akkor van olyan $g' \in C$, hogy $F - \{g\} \cup \{g'\}$is feszítőfa: F'

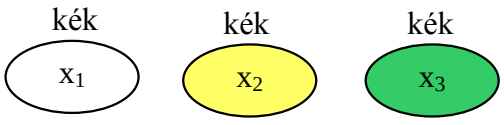
g'színe: nem piros (piros szabály miatt), nem kék, mert $g' \in F$, tehát színtelen ($g' \in S$) $\Rightarrow c(g') \leq c(g)$(mindig a legnagyobb súlyú élet színezzük pirosra) $\Rightarrow c(F') \leq c(F)$, de $c(F)$minimális $\Rightarrow c(F') = c(F)$, tehát az új színezés is takaros.

Lemma2

Az algoritmus végén az Shalmaz üres, azaz minden él $\in K$vagy $\in P$.

A bizonyítás alapja, hogy ha van színtelen él, akkor nincs vége, mert valamelyik szabály alkalmazható.

Legyen$g \in S$. A kék élek sosem alkotnak kört ⇒feszítőfát adnak.



A $g$él vagy a kettő között van, vagy két komponens között. Ha egy kék fán belül van akkor $g$és néhány fabéli él kört alkotnak. Ha két kék fát köt össze, akkor alkalmazzuk a piros szabályt, X-egyik kék fa pontjai, ezen a kék szabály.

A tétel bizonyítása

A lemmák segítségével: Lemma2 miatt $S = \emptyset$, tehát az élek kékek vagy pirosak. Lemma1 szerint ez a színezés takaros, azaz van olyan Fminimális súlyú feszítőfa, amire $F \geq K$, $F \cap P = \emptyset \Rightarrow F \equiv K$

Megjegyzés

Ha már van$|V|-1$ kék élünk, akkor abbahagyhatjuk, mert biztosan feszítőfa.

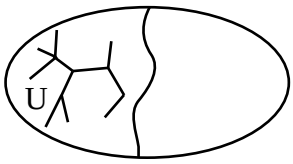
Alkalmazás: (JARNIK-)PRIM algoritmus

Vegyünk egy s in V csúcsot (kék) → egypontú kék fa

X = $\{s\}$-re kék szabály

X := kék élek által elért pontok → ezekre kék szabály...

A tételből következik, hogy ez jó.



Hogyan válasszunk élet?

U – kék fa pontjai.

$$KÖZEL[i] = \begin{cases} *, \text{ha} i \in U \\ j, j \in U, c(i,j) \text{minimális, ha} i \notin U \end{cases}$$

A kék szabály alkalmazásakor az $(i, KÖZEL[i])$élek közül a minimálisat tesszük bele az Uhalmazba $(i \notin U)$. Legyen ez a csúcs k.

A $KÖZEL[]$tömb frissítése:$\forall v \notin U$,ha$c(k,v) < c(v, KÖZEL[v]) \Rightarrow KÖZEL[v] = k, KÖZEL[k] = *$

Lépésszám: $(n-1) \cdot (n-1) + (n-1) \cdot O(n) = O(n^2)$

$\underset{minimumkeresése}{}$ $\underset{frissítés}{}$

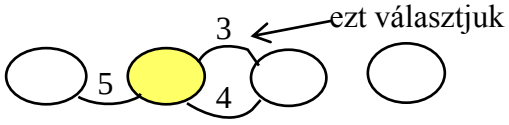
Kupaccal: Az éleket kupacban tároljuk. A legkisebb súlyút egy $MINTÖR$-rel választjuk ki az Ués $V-U$közötti élekből(+ a korábbi maradékokból, amik az aktuális Ués $V-U$között mentek, de már bekerültek U-ba). Vagyis nem akkor törlünk egy élet, amikor bekerült U-ba, mert az macerás lenne, hanem inkább akkor vizsgáljuk meg, hogy már U-ban van-e, amikor a kezünkbe kerül. Ha már U-ban volt, akkor eldobjuk, és újabb $MINTÖR$, …

Lépésszám éllistával: $O(e) + 2e \cdot (O(\log e)) = o(e \log e) = O(e \log n)$

$\underset{kupacépítés}{}$ $\underset{BESZÚR,MINTÖR}{}$

Ha sűrű a gráf, nem bináris kupacot használunk, jó paraméterekkel még kevesebb lépésben meg lehet valósítani.

Borůvka-algoritmus



Párhuzamos algoritmus.

Kiindulunk az összes csúcsból, kezdetben ∀csúcs egy kék fa, egy lépésben ∀kék fából kivezető legkisebb élet kékre színezzük(⇒ egyszerre több komponens egy nagyobbá olvad össze).

Problémát az okoz, ha több azonos súlyú él van, és kör alakulna ki. Ezt például a komponensek számozásával lehet megoldani, ha ilyenkor a legkisebb sorszámú komponenst választjuk.

Lépésszám: ∀menetben legalább feleződik a kék fák száma ⇒max. $\log n$lépés ⇒ $O(e \cdot \log n)$

Kruskal-algoritmus

1. Vegyük az éleket növekvő sorrendben.

Rendezés kupaccal $O(e \log e) = O(e \log n)$lépés.

2. Kékre színezzük, ha lehet (nem hoz létre kék kört)

Hogyan lehet eldönteni, hogy keletkezik-e kör?

Adatszerkezet: $UNIÓ - HOLVAN$

Xalaphalmaz

$A_1, A_2, \dots, A_k \subseteq X$, $A_1 \cup A_2 \cup \dots \cup A_k = X$, $A_i \cap A_j = \emptyset$, ha$i \neq j$(éldiszjunktak)

Műveletek: $UNIÓ(i,j)$: $A_i$és $A_j$helyett $A_i \cup A_j$

$HOLVAN(x) = i$, ha $x \in A_i$ $(x \in X)$

Kruskal-algoritmus: rendezés + élenként 2 db $HOLVAN$+ $n-1$db $UNIÓ$.

UNIÓ-HOLVAN megvalósítása

Tömbbel: $T[x] = i$, ha $x \in A_i$

$HOLVAN: O(1)$

$UNIÓ: O(n), n = |x|$

Kruskal: $O(e\log n) + eO(1) + (n-1) \cdot O(n) = O($ $e\log n$ $+$ n^2 $)$

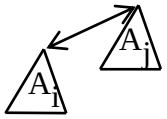
$e\log n$: sokélnéldominál; n^2: kevésél

Fákkal:

$A_i \rightarrow$ gyökeres fa

x elem → csúcs

$i \rightarrow$ gyökér (név, ezzel azonosítom a halmazt)



$HOLVAN$: csúcsból gyökérbe megyünk

lépésszám: $O(mmagasság)$

$UNIÓ$: a kisebbik fát beleépítjük a nagyobbikba: $|A_j| \geq |A_i|$

lépésszám: O(1), ha az elemszámot nyilvántartjuk.

Állítás

Ha kezdetben $\forall A_i$egyelemű ⇒magasságuk mindig $O(\log n)$

Bizonyítás: Amikor egy $A_i$beleolvad egy legalább ugyanakkora halmazba, mindig 1-gyel távolodik. Ilyenkor a halmaz mérete duplázódik ⇒ log$n$ilyen lehet.

Kruskal: $O(e\log n) + e \cdot O(\log n) + (n-1) \cdot O(1) = O(e\log n)$

12. előadás

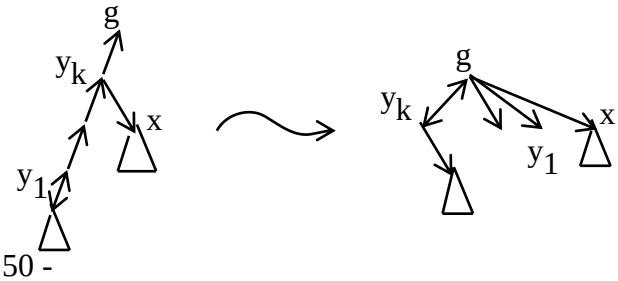
Minimális feszítőfa

Kruskal algoritmus, $UNIÓ - HOLVAN$adatszerkezet, fákkal: halmaz → fa, ennek mélysége lényeges.

$HOLVAN(x)$során $HOLVAN(y)$-ra is megkapjuk a választ.

UNIÓ-HOLVAN összevonással

$HOLVAN(x)$futtatásakor előny, hogy $y_1, y_2, \dots, y_k$-ra vonatkozó $HOLVAN$gyors lesz.



Tétel

Ha egyelemű halmazokkal kezdünk, $n-1 UNIÓ$és $m \geq n-1 HOLVAN$kérdés lépésszáma $O(m \cdot \alpha(m))$, ahol n az elemek, m pedig az élek száma. $(\neg B)$

$\alpha(m)$: Ackerman függvény inverze

$\alpha(m)$monoton nő

$\alpha(m) \to \infty$, ha $m \to \infty$

$\alpha(m) < 4$, ha$m < 2^{2^{2^2}} = 2^{2^{16}} = 2^{65536} (\neg B)$

Ebből a Kruskal lépésszáma: $\underset{rendezés}{O(e\log n)} + \underset{HOLVAN}{O(2e \cdot \alpha(2e))} = O(e\log n) + O(e \cdot \alpha(2e))$,

és itt már a második tag a kicsi, vagyis az algoritmus lépésszámát a rendezés határozza meg.

Megjegyzések

- Ha az élsúlyokat tudjuk lineáris időben rendezni (pl. a súlyok 1és $n$közötti egészek), akkor a lépésszám $O(e) + O(e \cdot \alpha(2e)) = O(e \cdot \alpha(2e))$
- Nyitott kérdés, hogy van-e $O(e)$lépésszámú algoritmus általános súlyok esetén.
- Ismert: azt ellenőrizni, hogy minimális feszítőfa-e $O(e)$lépés (de megtalálni még nem tudjuk).
- Véletlen mintavétel használatával elérhető, hogy a minimális feszítőfa találásának várható lépésszáma $O(e)$legyen.

Bonyolultság

Egy algoritmusról akkor mondjuk, hogy hatékony, ha polinom idejű, vagyis a lépésszáma a bemenet hosszának polinom-függvénye.

Példa

- n elemű halmaz összes részhalmazának felsorolásakor már az eredmény is exponenciálisan hosszú (tehát ha csak írunk, és nem számolunk, akkor is sokáig tart) ⇒nem lehet rá polinomiális algoritmust találni.
- Bemenet $n \in \mathbb{N}$, keressük 3^nértékét.
  - a bemenet hossza: $\log(n+1)$
  - a kimenet: $\log(3^n)$nem polinom hosszú
- $n \in \mathbb{N}, m \in \mathbb{N}$, cél $3^n (mod m)$
  - $\underset{n}{3 \cdot 3 \cdot \ldots \cdot 3} \to n-1$szorzás, nem polinom log$n$-ben
  - négyzetre emelések $mod m$
    - $3^2 mod m$
    - $3^4 mod m$ezek közül a megfelelőket összeszorozzuk
    - $3^8 mod m$

$\log n$darab ilyen lesz $\Rightarrow O(\log n)$szorzás, modulo osztás, ez $\log n$-ben polinom.

Döntési problémák

Kimenetük: *igen* vagy *nem*

Példa

Megállási probléma: bemenet ($\overset{program}{P}$, $\overset{bemenet}{Q}$)

Eredmény: Pa Qbemenettel megáll-e.

Erre a döntési problémára nincs algoritmus.

Definíció

P: polinom időben megoldható döntési problémák

Példa

Ggráf összefüggő: P-ben van

Definíció

NP: ***nem****determinisztikus* polinom időben megoldható döntési problémák

Ha az $x$bemenetre a válasz $IGEN$, akkor van olyan polinom hosszú y(bizonyíték, tanú), hogy az (x, y)pár együtt polinom időben leellenőrzi, hogy $y$bizonyítja az $IGEN$t (vagyis röviden bizonyítható).

Ha az $x$bemenetre a válasz NEM, akkor *nincs* ilyen y. (Azaz $A \in NP$ha van olyan $B \in P$, amikor x-re A-nál $IGEN$a válasz, akkor *van* olyan polinom hosszú y, hogy (x, y)-ra B-nél is $IGEN$a válasz. Ha x-re A-nál NEMa válasz, akkor *nincs* ilyen y)

Példa

H – Ggráfban van Hamilton-kör.

$H \in NP$, mert $x = G$-hez yHamilton-kör mentén a pontok felsorolása jó tanú.

y hossza = $|y|$polinomiális x-ben (vagyis $x$csúcs- és élhalmaz méretének polinom függvénye).

Ellenőrzés: y-ban minden pont egyszer szerepel, egymás utáni pontok, az első és az utolsó között van él. Ez jó, ha x-ben van Hamilton-kör, és nincs jó y, ha x-ben nincs Hamilton-kör (y hordozza azt a pici információt, aminek segítségével el tudjuk dönteni, hogy $x$lehet-e jó).

Példa

Ö$SSZETETT$: $n$szám összetett.

$\in NP$, mert 1<y<n egész ellenőrzés: $y \mid n$.

Tulajdonság

$P \leq NP$

Nyitott: $P \overset{?}{=} NP$

Aeldöntési probléma: $IGEN/NEM$

Akomplementer problémája ($\bar{A}$):

x bemenetre a válasz $IGEN \Leftrightarrow A$-nál a válasz NEM

Pl: Ö$SSZ\bar{E}TETT$= $PR\acute{I}M$

Definíció

coNP: NP-beliek komplementerei

A NEMválaszra van polinomiális bizonyíték, azaz $A \in coNP \Leftrightarrow \bar{A} \in NP$

$P \in coNP \Rightarrow P \in (NP \cup coNP)$

Nyitott:$P \stackrel{?}{=} NP \cap coNP$

Példa

$S\acute{I}K$: Ggráf síkba rajzolható-e

$\in NP$ y: pontok koordinátái a síkban (racionális, az élek szakaszok)

$\in coNP$ y: Kuratowszki gráf

$\Rightarrow S\acute{I}K \in (NP \cap coNP)$, sőt$S\acute{I}K \in P$ $(\neg B)$

Karp-redukció (polinomiális visszavezetés)

$A \prec B$: Aredukciója B-re

$f$polinomiális időben számolható függvény

f: $Abemenetei \rightarrow Bbemenetei$

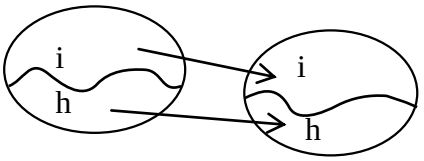
$A(x) = IGAZ \Leftrightarrow B(f(x)) = IGAZ$

$P \subseteq NP$

Nyitott kérdés $P =? NP$

$P \subseteq coNP$

Tulajdonság



Ha $A \prec B$akkor $\bar{A} \prec \bar{B}$

Bizonyítás: $f A$-ból B-re Karp redukció

Ugyanez az $f$jó $\bar{A}$-ból $\bar{B}$-ra Karp-redukcióval

Állítás

Ha $A \prec B, B \in NP \Rightarrow A \in NP$

Bizonyítás: $\forall x$-re $B(x) = IGAZ$van polinom hosszú $y$tanú, amit polinom időben ellenőrizni is tudunk.

z: bemenete Aproblémának

$f(z)$tanúja egy jó tanú A-ra

ellenőrzés: $(f(z), y)$-ra B-hez tartozó ellenőrzés

kell: $y$hossza, ellenőrzés lépésszáma polinomja $z$hosszának

tudjuk: y hossza, ellenőrzés lépésszáma polinomja $f(z)$hosszának

Mivel $f$polinomiális időben számolható, ezért $f(z)$hossza $z$hosszának polinom függvénye.

⇒valóban $|z|$-nek is polinomja

Állítás

Ha $A \prec B$és $B \in coNP$akkor $A \in coNP$

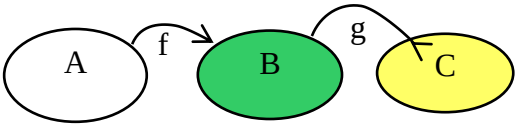
Bizonyítás: $B \in coNP \Leftrightarrow \bar{B} \in NP$

$A \prec B \Rightarrow \bar{A} \prec \bar{B} \Rightarrow \bar{A} \in NP \Rightarrow A \in coNP$

Állítás

Ha $A \prec B$és$B \prec C$ akkor $A \prec C$(tranzitivitás)

Itt a B-re vonatkozó programot nem használhatom, csak a végén, és az eredményét el kell fogadnom.



Bizonyítás:

$f: A \prec B$

$g: B \prec C$

$g(f(x))$Karp-redukció $A \prec C$

$A(x) = IGAZ \Leftrightarrow B(f(x)) = IGAZ \Leftrightarrow C(g(f(x))) = IGAZ$

és polinom időben számolható $x \underset{\substack{\text{polidő}|x|-\text{ben}}}{\rightarrow} f(x) \underset{\substack{\text{polidő}|f|(x)-\text{ben} \\ \text{polinom}x-\text{benis}}}{\rightarrow} g(f(x)) \Leftrightarrow$

$g(f(x))$polinom időben számolható$|x|$-ben

NP-teljesség definíciója

NP-teljes az Aprobléma, ha $A \in NP$, és $\forall B \in NP$-re $B \prec A$

Tétel

$3 - SZÍN$NP-teljes

Bizonyítás

$3 - SZÍN \in NP$

tanú: $a_1, \dots, a_n$(egy feltételezett színezés)

ellenőrzés:$\forall a_i$az1,2,3 valamelyike (tényleg max. 3 színnel van kiszínezve)

a gráfnak $n$csúcsa van

$\forall$ él végpontjai különböző színűek

(Annak ellenőrzésekor, hogy az algoritmus polinom időben fut mindegy, hogy a gráf éllistával vagy mátrixszal van megadva, mert egyik megadás a másikból poli nom időben előáll.)

NP-teljes: $\neg B$

NP-teljesség bizonyításának váza

$A \in NP$

Vegyünk egy tetszőleges NP-teljes problémát: B

Elég megmutatni, hogy$B \prec A$, mert ekkor $\forall C \in NP: C \prec B \prec A$,tehát$C \prec A$.

Ha egy NP-teljes problémára találunk egy polinom idejű algoritmust, akkor $P = NP$.

Mert ANP-teljes, $A \in P$.

Tetszőleges $B \in NP, B \prec A \Rightarrow B \in P \Rightarrow NP \subseteq P$,de$P \subseteq NP \Rightarrow P = NP$.

13. előadás

Néhány NP-teljes probléma

Független pontok maximális száma (MAXFTL)

Bemenet: Ggráf,$k \in \mathbb{N}$ pozitív egész.

Van-e $k$független pont (azaz $\alpha(G) \geq k$)?

Tétel

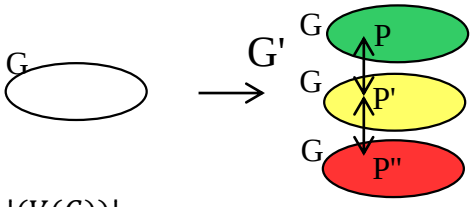
A$MAXFTL$ NP-teljes

Bizonyítás

$\in NP$

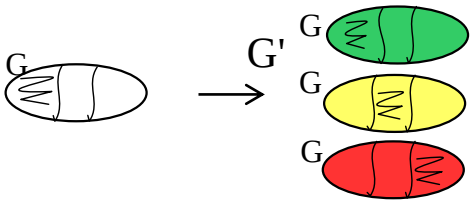
$3 - SZÍN \prec MAXFTL$

$f: G \to G', k$ függvény legyen olyan, hogy:



3 színnel színezhető ⇔ G'-ben van $k$független pont

G'-ben létrehozunk G-ből 3 példányt, egy pont mindhárom példányát összekötjük. $K = |(V(G))|$



„⇒”

Tudjuk, hogy $\chi(G) \leq 3$, G'komponenseit színezzük úgy, mint G-t, és az i.színű pont i.példánya kerüljön bele a független ponthalmazba. Így G'-ben valóban találunk $k$db független pontot.

„⇐”

Az előbb megkonstruált G'gráfban van $k$db független pont, és ezek között minden G-beli pontosan egyszer fordul elő. Az i.komponensbeli pont az i.színt kapja. Tehát Gvalóban 3 színnel színezhető.

Maximális klikk mérete (MAXKLIKK)

Bemenet: Ggráf, $k$pozitív egész szám.

Van-e k pontú teljes részgráf G-ben ($\omega(G) \geq k$)?

Tétel

A $MAXKLIKK$NP-teljes

Bizonyítás

NP-beliség: tanú adott $k$pont, ezekről kell ellenőrizni, hogy teljes részgráfot határoznak-e meg.

Kell: $MAXFTL \prec MAXKLIKK$

$(G,k) \to (G',k')$hozzárendelés úgy, hogy$\alpha(G) \geq k \Leftrightarrow \omega(G') \leq k'$

A keresett függvény: $f: G' = \bar{G}, k' = k$, és ez tényleg polinom időben számolható.

Hamilton-kör (H)

Bemenet: Ggráf

Van-e Hamilton-kör G-ben?

Hamilton-út (HÚT)

Bemenet: Ggráf

Van-e Hamilton-út G-ben?

Adott végpontú Hamilton-út (H-s-t-ÚT)

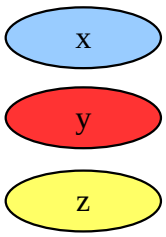
Bemenete: Ggráf, s, tcsúcsok.

Van-e olyan Hamilton-út, aminek végpontjai $s$és t?

Tétel

A H, $HÚT$és $H-s-t-ÚT$irányított és irányítatlan esetben is NP-teljesek. $\neg B$

3 dimenziós házasítási probléma (3DH)



→„ez a fiú ezzel a lánnyal és ezzel a kutyával igen :)”

Bemenet: x, y, zazonos elemszámú halmazok.

Van-e olyan házasítás (x, yész egy-egy elemének összerendelése), ami minden pontot pontosan egyszer fed le?

Tétel

A 3DHNP-teljes. (nem bizonyítjuk)

Pontos fedés hármasokkal (X3C)

(exactly 3 cover)

Bemenet: Xalaphalmaz, és ennek néhány háromelemű részhalmaza: $\{F_1, F_2, \dots, F_k\} \subseteq X$

Ki tudunk-e választani ezek közül néhány olyan páronként diszjunkt F_i-t, hogy ezeknek az uniója pontosan Xlegyen?

Matematikai nyelven megfogalmazva: van-e olyan $I \subseteq \{1,2, \dots, k\}$, hogy $\bigcup_{\forall i \in I} F_i = X$, és $F_i \cap F_j = \emptyset$,ha$i \neq j, i, j \in I$?

Tétel

$X3C$NP-teljes. (nem bizonyítjuk az órán)

Megjegyzés

$X2C$általános gráfban teljes párosítás $\in P$.

Részhalmazösszeg (RH)

Bemenet: $s_1, s_2, \dots, s_n \geq 0$egészek, $b \geq 0$egész

Van-e $I \subseteq \{1,2, \dots, n\}$, hogy $\sum_{i \in I} \quad s_i = b$?

Partíció (PARTÍCIÓ)

Bemenet: $s_1, \dots, s_n \geq 0$egészek

Van-e $I \subseteq \{1,2, \dots, n\}$, hogy $\sum_{i \in I} \quad s_i = \sum_{j \notin I} \quad s_j$?

Hátizsák (HÁTIZSÁK)

Bemenet: $s_1, \dots, s_n \geq 0$, $v_1, \dots, v_n \geq 0$, $b \geq 0$és $k \geq 0$egészek

Van-e $I \subseteq \{1,2, \dots, n\}$, hogy $\sum_{i \in I} \quad s_i \leq b$és $\sum_{i \in I} \quad v_i \geq k$

Tétel

RH, $PARTICIÓ$, $HÁTIZSÁK$NP-teljes. (ezeket nem bizonyítjuk)

Részgráf izomorfiája (RÉSZGRÁFIZO)

Bemenet: G_1, G_2gráf

Van-e G_1-nek G_2-vel izomorf részgráfja?

Ez is NP-teljes, visszavezethető rá a H, a $MAXKLIKK$és a $MAXFTL$is.

Gráf izomorfiája (GRÁFIZO)

Bemenet: G_1, G_2gráf

Izomorfak-e?

In NP, tanú az izomorfizmus

Nyitott kérdés, hogy $\overset{?}{\in} P$, vagy NP-teljes, vagy egyik sem.

Nyelvek

döntési problémák →L nyelv: az $IGAZ$válaszokhoz tartozó bemenetek halmaza $x \in L$

P, NP tekinthető nyelvek halmazaként (nyelvosztály).

14. előadás

Lineáris programozás

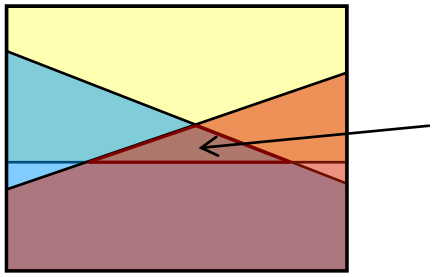
$\sum_{j=1}^{n} \quad a_{ij} x_j \leq b_j$adott egészek, $x_1, \dots, x_n$ismeretlen, $x_j \geq 0$, minden j-re.

A cél olyan c_j-k megtalálása, amikre $\sum c_j x_j$ maximális, $c_1, \dots, c_n$adott egészek.

célfv.: irány

Példa

Mondjuk két háromszögben geometriai tartalom: egyenlőtlenség→ félsík



Algoritmusok

Szimplex módszer (Dantzig, 1947)

Maximum csak a csúcsok valamelyikében lehet, ezért ezeket kell mind megvizsgálni. Ez az algoritmus exponenciális, de a gyakorlatban jól működik.

Ellipszoid módszer (Haczina, 1979)

Polinomiális algoritmus, melyet azonban csak nagyon nagy értékeknél szokás használni, így a gyakorlati felhasználhatósága korlátozott.

Belső pont módszer (Karnavkor, 1984)

Az ellipszoid módszer továbbfejlesztése, a gyakorlatban is jól használható.

Egész értékű programozás

Rövidítése: EP.

Akkor használjuk, ha egész megoldások szükségesek, azaz $x_{1,}x_{2,} \dots, x_n \in \mathbb{Z}$.

Tétel

Az EP döntési változata NP-teljes.

Már maga az NP-beliség se triviális, ehhez is az kell, hogy létezzen rövid megoldás.

Példák

MaxFTL átfogalmazása EP feladatra

A gráf csúcsait jelöljük x_i-vel, $0 \leq x_i \leq 1$, vagyis a csúcsok értéke nulla, vagy egy.

$\{i,j\}$ él $\rightarrow x_i + x_j \leq 1$

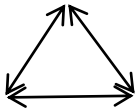
A maximális független halmaz tehát: $max\sum x_i$

Független halmaz $= \{i: x_i = 1\}$,

Azaz a szomszédos csúcsok közül legfeljebb az egyik lehet benne a kiválasztott csúcsok halmazában, és a lehető legtöbb csúcs beválasztására törekszünk, tehát a független pontok halmaza az a halmaz lesz, ahol a csúcsok értéke egy.

(könnyű belátni, hogy ez a probléma is NP-teljes)

A K_3 gráf



$0 \leq x_{1,}x_{2,}x_3 \leq 1$

$$2x_1 + 2x_2 + 2x_3 \leq \begin{cases} x_1 + x_2 \leq 1 \\ 3x_1 + x_3 \leq 1 \\ x_2 + x_3 \leq 1 \end{cases}$$

$x_1 = x_2 = x_3 = \frac{1}{2}$ maximalizálja $\sum x_i$-t.

maxkell

itt most nem csak egész

megoldást keresünk

Tanácsok nehéz problémák esetére

- Próbáljunk először egy speciális esetet megoldani, például a leghosszabb út keresése dag-ban könnyű, ellenben általánosságban nehéz feladat.
- Elágazás és korlátozás, dinamikus programozás – előfordulhat, hogy a talált algoritmus lépésszáma exponenciális lesz, azonban segítségünkre lehet ezen módszerek alkalmazása.

Közelítő algoritmusok

Maximalizálási feladat

Az optimális érték: OPT

c-közelítő algoritmus: egy olyan megoldást talál, amelynek értéke $\geq c \cdot OPT$(a jó algoritmus 1-hez közelítő)

Minimalizálási feladat

c-közelítő $\leq c \cdot OPT$ $\quad\quad$ $c \geq 1$

Megjegyzés

Az 1-közelítő algoritmus az optimumot találja meg.

Példa

Ggráf, a független élek maximális számát keressük (ez megegyezik a maximális párosítás problémájával, amire van polinomiális algoritmus)

Mohó algoritmus

Az algoritmus független éleket vesz, amíg tud.

Lineáris a futása, $O(n + e)$a lépésszáma.

A megtalált független élek száma $\geq \frac{\tau(G)}{2} \geq \frac{\nu(G)}{2} = \frac{OPT}{2}$, azaz az algoritmus $\frac{1}{2}$-közelítő.

Utazó ügynök

Adott egy Girányítatlan gráf, az élein súlyok, $s: \varepsilon \to \mathbb{R}$

A feladat minimális súlyú Hamilton-kör találása.

Döntési változat

Adott G, s, k, létezik-e legfeljebb $k$súlyú Hamilton-kör G-ben?

Tétel

A probléma döntési változata NP-teljes.

Bizonyítás

$\in NP$: tanú a csúcsok felsorolása – polinom hosszú.

Ellenőrzés: Minden csúcs pontosan egyszer szerepel, a szomszédok között és 2 vége között van él, így az összsúly legfeljebb $k$lehet.

A Karp-relációról

A karp-reláció voltaképpen egy nehézségi reláció, mely munkánkat úgy segíti, hogy könnyebb problémákra vissza tudjuk vezetni az éppen aktuálisat. Tehát, ha felírjuk azt, hogy $H \prec$, akkor H probléma nem lehet nehezebb a másiknál, melyet (a másik problémát megfelelően kiválasztva) esetleg könnyebben tudunk megoldani, vagy megoldása már adott.

Jelen esetben vegyük a következő halmazt: $G \to G^k$, $s \equiv 1$, $k = |V(G)|$. A megoldás ezután a következő lesz: sokszor $G = K_n$, a G-beli nem-élek nagy súlyt kapnak.

Tétel

Ha valamilyen $c \geq 1$konstansra van polinomidejű c-közelítő algoritmus az utazóügynök problémára, akkor $P = NP$

Bizonyítás

Ilyen algoritmussal eldönthető lenne a H-probléma.

2. probléma utazóügynökre

$n = |V(G)|$, K_n $\qquad D(i,j) = \begin{cases} 1, \text{ ha} \{i,j\} \in E(G) \\ c \cdot n + 1 \text{ha} \notin \end{cases}$

Erre a problémára utazó ügynök:

Ha G-ben van Hamilton-kör, akkor van $n$értékű megoldás (minden él súlya egy)

Ha G-ben nincs Hamilton-kör, akkor minden megoldás értéke nagyobb, mint $c \cdot n + n$(mert belekerül néhány nagy súlyú él is)

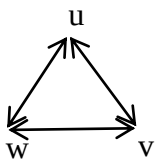
Ha a c-közelítő algoritmus legfeljebb $c \cdot n$értékű megoldást ad, akkor G-ben van Hamilton-kör, ha pedig nagyobbat, akkor nincs $n$értékű kör, így Hamilton-kör sincs G-ben.

Euklideszi utazóügynök

Definíció

Az utazóügynök egy olyan változata, melynél a súlyokra teljesül a háromszög-egyenlőtlenség: $G = K_n$,

$s(u,v) \leq s(u,w) + s(w,v)$, minden u, $v$és $w$csúcsra.



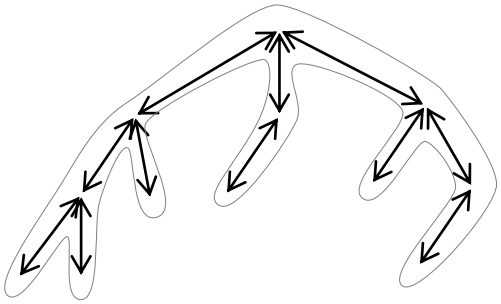
Tétel

Ennek a döntési változata is NP-teljes.

(nem bizonyítjuk)

2-közelítő algoritmus

A minimális súlyú feszítőfa Fkeresésére van polinomidejű algoritmusunk, érdemes tehát ezt használni, „menjünk körbe a fán” (kívülről, a szürke vonal mentén), és vágjuk le a visszatéréseket. Ekkor preorder sorrendben járjuk be a gráfot, tehát végeredményképp egy Hamilton-körét fogjuk kapni.



Ennek súlya legfeljebb $2s(F)$lesz.

A Hamilton-kör megegyezik egy Hamilton-úttal, valamint egy hozzávett megfelelő éllel, előbbi súlya megegyezik a feszítőfáéval, tehát legalább $s(F)$lesz, míg utóbbi súlya legalább nulla.

Ládapakolás

Leírás

1 méretű ládák vannak, ezekbe kell elhelyeznünk $n$darab tárgyat, melyek mérete $s_1, \ldots, s_n$, $0 < s_i < 1$, $s_i \in \mathbb{Q}$. A célunk minél kevesebb ládában elhelyezni minden tárgyat.

Tétel

A probléma döntési változata NP-teljes.
(nem bizonyítjuk)

Közelítő algoritmusok

First Fit

Sorban minden tárgyat az első olyan ládába helyezzük, melybe belefér. Könnyű belátni, hogy ez az algoritmus polinomidejű.

Példaképp vegyünk hat tárgyat, a súlyuk legyen $\frac{1}{3}; \frac{1}{3}; \frac{1}{3}; \frac{2}{3}; \frac{2}{3}; \frac{2}{3}$. Az optimális elrendezést akár fejben is elvégezhetjük könnyen, $OPT = 3$láda, míg az algoritmussal $FF = 4$, látható tehát, hogy az algoritmus nem optimális. De vajon akkor milyen hatásfokú?
Be tudjuk bizonyítani, hogy ez egy 2-közelítő algoritmus, mégpedig a következőképpen:
A ládák, legfeljebb egy darab kivételével több, mint a felükig tele vannak, különben két ilyen (kevesebb, mint a feléig megtelt) láda tartalma egymásba került volna. Ez azt eredményezi, hogy két ilyen láda együtt, több, mint 1 méretű tárgyat tartalmaz.
Ha Lláda van, $\underset{\text{kivétel valamelyik}}{1} + L - 2 < \sum_{i=1}^{n} s_i \leq OPT$.

Tétel

Minden bemenetre $FF \leq [1{,}7 \cdot OPT]$
Létezik olyan bemenet, hogy $FF \geq 1{,}7 \cdot OPT - 1$

First Fit decrementary algoritmus

Ez az algoritmus először csökkenő sorrendbe rendezi a tárgyakat, majd alkalmazza rájuk az előbb tanult First Fit algoritmust.

Tétel

Minden bemenetre $FFD \leq \frac{11}{9} OPT + 4$

Létezik olyan bemenet, hogy $FFD \geq \frac{11}{9} OPT$.

Tétel

Minden ε-ra van $1 + \varepsilon$-közelítő, polinom idejű algoritmus.