1. előadás

Algoritmuselmélet 1. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Február 11.

Források

- Rónyai Lajos–Ivanyos Gábor–Szabó Réka: **Algoritmusok**, TYPOTEX, 1999
- Feladatgyűjtemény letölthető: http://www.cs.bme.hu/˜kiskat/algel
- Egyéb információk, hirdetmények ugyanitt.

Követelmények

Zárthelyi dolgozat: 2002. április 8. — 6 db 10 pontos feladat 100 percre, mindet lehet használni.

Pontozás: 20-29 pont 2-es, 30-39 pont 3-es, 40-49 pont 4-es, 50-60 pont 5-ös.

Aláírás: Feltétele a ZH megírása 2-esre. Szükség esetén pótZH.

Vizsga: Írásbeli — 2 elméleti kérdés, 4 példa, nem lehet semmit használni. Pontozás: mint a ZH-n.

Vizsga jegy: Ha érdemes, akkor lehet a vizsga ZH eredményét átlagolni az évközi eredménnyel. Ha aláírás csak pótZH-val született, ez a lehetőség nem áll fenn.

Javítás: Ha az így kialakult jegy nem elég jó, akkor kizárólag a vizsga eredményhirdetésének időpontjában lehet szóban felelni, amivel ± 1 jegyet lehet változtatni.

Algoritmus fogalma

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

A szó eredete

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

algoritmus $\leftrightarrow$ számítógép program

Milyen hatékony egy algoritmus?

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

Függvények nagyságrendje

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

Például:

- $100n + 300 = O(n)$, hiszen $n = 300,\ c = 101$-re teljesülnek a feltételek, $100n + 300 \le 101n$, ha $n \ge 300$
- $5n^2 + 3n = O(n^2)$
- $n^4 + 5n^3 = O(n^5)$
- $n^{1000} = O(2^n)$

Függvények nagyságrendje

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

Például:

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

Függvények nagyságrendje

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

Például:

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

Függvények nagyságrendje

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

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

Például:

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

Algoritmikus problémák megoldása

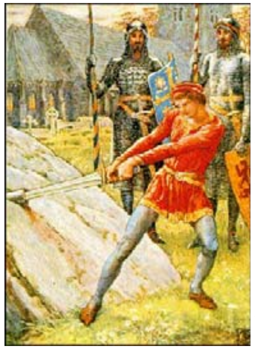
$$\boxed{\text{Algoritmikus probléma}} \xrightarrow{\mathcal{A}} \boxed{\text{modell}} \xrightarrow{\mathcal{B}} \boxed{\text{program}}$$

$\mathcal{A}$: pontosítás, egyszerűsítés, absztrakció, lényegtelen elemek kiszűrése, a lényeg kihámozása

Modell: sokféle lehet, elég tág, de elég egyszerű, formalizált, pontos

$\mathcal{B}$: hatékony algoritmus, bemenő adatok → eredmény, érdemes foglalkozni a kapott algoritmus *elemzésével, értékelésével*, megvizsgálva, hogy a módszer mennyire hatékony

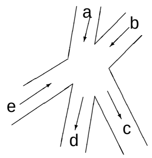
Arthur király civilizációs törekvései



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

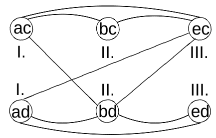
Természetesen pártfogójához, a nagyhatalmú varázslóhoz, Merlinhez fordul. Merlin rögvest felismeri, hogy itt is bináris szimmetrikus viszonyok ábrázolásáról van szó.
Nagy darab pergament vesz elő, és nekilát egy páros gráfot rajzolni.
A királyi parancs teljesítéséhez Merlinnek élek egy olyan rendszerét kell kiválasztania a gráf éleiből, hogy a kiválasztott élek közül a gráf minden pontjához pontosan egy csatlakozzon. A kiválasztott élek felelnek meg a tervezett házasságoknak. A gráfelmélet nyelvén teljes párosítást kell keresnie.

Közlekedési lámpák ütemezése



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

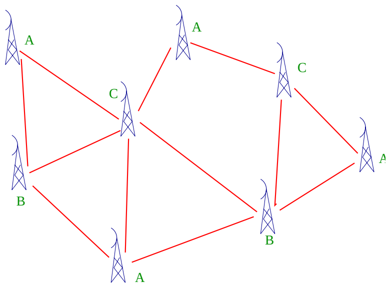
Feladat: Mennyi a minimális számú állapot, ami biztonságos és nem okoz örök dugót?



Gráfelméleti nyelven: Mennyi G kromatikus száma?

Mobil telefon átjátszók frekvencia kiosztása

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



Szuperforrás keresése

Definíció. *A G irányított gráf $s \in V$ csúcsa szuperforrás, ha minden s-től különböző $y \in V$ csúcs esetén teljesül, hogy $(s, y) \in E$ és $(y, s) \notin E$. A szuperforrás olyan s csúcs, amiből a gráf minden más csúcsába él vezet, az s-be pedig egyetlen más csúcsból sem megy él.*

A feladat a következő: tegyük fel, hogy az A adjacencia mátrixával adott a $G = (V, E)$ irányított gráf, aminek a csúcshalmaza $V = \{1, \ldots, n\}$. Döntsük el, hogy van-e G-ben szuperforrás. Ha igen, találjuk meg.

Első ötlet: Sorra vesszük az $i \in V$ csúcsokat, mindegyikről megnézve, hogy szuperforrás-e.

Ennek költsége: az A mátrix $2(n^2 - n)$ elemét vizsgáljuk meg.

Jobb módszer: Ha $(i, j) \in E$, akkor j nem lehet szuperforrás, ha $(i, j) \notin E$ akkor i nem lehet szuperforrás

Gyorsabb algoritmus

$i := 1, j := n$;
while $i \neq j$ **do**
    if $A[i, j] = 1$ **then** $j := j - 1$

    else $i := i + 1$;
(Amikor ideérünk, már csak i lehet szuperforrás,*
  *ezt ellenőrizzük a továbbiakban. *)*
for $k = 1$ **to** n **do**
    if $k \neq i$ és ($A[i, k] \neq 1$ vagy $A[k, i] \neq 0$) **then return**(nincs szuperforrás)
return(i szuperforrás).

Ennek költsége: összesen $3n - 3$ elemet vizsgálunk meg, $n - 1$-et a while ciklusban, $2n - 2$-t az ellenőrzésnél

Mennyire jó ez?

A költség elemzése

Jelölje $T(n)$ a legjobb (leggyorsabb) algoritmus által megvizsgált mátrix-elemek számának maximumát az összes n pontú gráfra.

Tudjuk, hogy $T(n) \le 3n - 3$.

Nyilvánvaló, hogy $2n - 2 \le T(n)$, mert le kell ellenőrizni, hogy szuperforrás-e.

1 él megkérdezése legfeljebb 1 csúcsot zár ki mint lehetséges szuperforrást.

Egy algoritmus első $n - 2$ kérdése után még legalább két csúcs – mondjuk i és j – lehet szuperforrás. Ahhoz, hogy belássuk, hogy i szuperforrás, meg kell vizsgálni az i-edik sor és i-edik oszlop minden elemét.

Vagy i-re vagy j-re igaz lesz, hogy az első $n - 2$ kérdés közül legfeljebb $(n-2)/2$ kérdés vonatkozott rá. Így még $2n - 2 - (n-2)/2$ legalább kérdés kell. Tehát

$$T(n) \ge 2n - 2 - \frac{n-2}{2} + n - 2 = 3n - 4 - \frac{n-2}{2}.$$

Azaz a fenti algoritmus közel optimális.

Rendezési reláció

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

1. $a \not< a$ minden $a \in U$ elemre ($<$ irreflexív);
2. Ha $a, b, c \in U$, $a < b$, és $b < c$, akkor $a < c$ ($<$ tranzitív);
3. Tetszőleges $a \ne b \in U$ elemekre vagy $a < b$, vagy $b < a$ fennáll ($<$ teljes).

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

Példák:

- $\mathbb{Z}$ az egész számok halmaza. A $<$ rendezés a nagyság szerinti rendezés.
- Az abc betűinek Σ halmaza; a $<$ rendezést az abc-*sorrend* adja. Az x betű kisebb, mint az y betű, ha x előbb szerepel az abc-sorrendben, mint y.

- A Σ betűiből alkotott szavak Σ^* halmaza a szótárszerű vagy lexikografikus rendezéssel. $\Rightarrow$ legyen $X = x_1x_2 \cdots x_k$ és $Y = y_1y_2 \cdots y_l$ két szó. Az X kisebb mint Y, ha vagy $l > k$ és $x_i = y_i$ ha $i = 1, 2, \ldots, k$; vagy pedig $x_j < y_j$ teljesül a legkisebb olyan j indexre, melyre $x_j \ne y_j$. Tehát például $kar < karika$ és $bor < bot$.

A rendezés feladata: *adott az $(U, <)$ rendezett halmaz elemeinek egy $u_1, u_2, \ldots, u_n$ sorozata; rendezzük ezt át egy nem csökkenő $v_1, v_2, \ldots, v_n$ sorrendbe.*

Input: tömb, láncolt lista, (vagy bármi)

Output: általában, mint az Input

Lépések: elemek mozgatása, cseréje, összehasonlítása

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

Keresés rendezett halmazban

Bar Kochba játék: gondolok egy számot $1 - 100$-ig, hány eldöntendő kérdésből lehet kitalálni?

Adott az $(U, <)$ rendezett halmaz véges $S = \{s_1 < s_2 < \ldots < s_{n-1} < s_n\}$ és $s \in U$. részhalmaza.

Összehasonlításokkal akarjuk eldönteni, hogy igaz-e $s \in S$. Hány összehasonlítás kell?

Lineáris keresés

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

Bináris keresés

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

És így tovább:

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

Pl. keressük meg, benne van-e $\mathbf{21}$ az alábbi sorozatban!

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

Bináris keresés

Addig kell csinálni, amig $|S_k| \ge 1$, vagyis $1 \le \frac{n}{2^k}$.
$\Longrightarrow 2^k \le n \Longrightarrow k \le \lfloor \log_2 n \rfloor$
Ez $k+1$ összehasonlítás volt. $\Longrightarrow k + 1 \le \lfloor \log_2 n \rfloor + 1 \le \lceil \log_2(n+1) \rceil$
Tétel. *Ez optimális, nincs olyan kereső algoritmus, ami minden esetben kevesebb mint $\lceil \log_2(n+1) \rceil$ kérdést használ.*

Bizonyítás: Az ellenség nem is gondol egy számra, csak mindig úgy válaszol, hogy minél többet kelljen kérdezni.
Ha egy kérdést felteszek, és az igen válasz után mondjuk szóba jön x lehetőség, akkor a nem esetén szóba jön még $n - x$ lehetőség.
Az ellenség úgy válaszol, hogy minél több lehetőség maradjon, így el tudja érni, hogy legalább $n/2$ marad. $\Longrightarrow k$ kérdés után is marad még $\frac{n}{2^k}$ lehetőség.
Ha tehát $\frac{n}{2^k} > 1$, akkor nem tudom, hogy az-e a gondolt szám, vagy nincs benne a sorozatban. Tehát még egy kérdésre szükség van.
$\Longrightarrow \frac{n}{2^k} > 1 \Longrightarrow n > 2^k \Longrightarrow \log_2 n > k$.

2. előadás

Algoritmuselmélet 2. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Február 12.

Buborék-rendezés

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

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

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

cserék száma: $\frac{n(n-1)}{2}$

Java animáció: Buborék rendezés

Beszúrásos rendezés

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

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

Bináris beszúrásos rendezés lépésszáma

$$K := \lceil \log_2 2 \rceil + \lceil \log_2 3 \rceil + \ldots + \lceil \log_2 n \rceil \le n \lceil \log_2 n \rceil$$

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

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

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

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

Ezért $K \le n(\log_2 n - 0,442)$ elég nagy n-re.
Java animáció: Beszúrásos rendezés

Alsó becslés összehasonlítás alapú rendezésre

Ugyanaz, mintha Bar Kochba-ban kellene kitalálni, hogy az elemek melyik sorrendje (permutációja) az igazi sorrend.
Kezdetben $n!$ lehetséges sorrend jön szóba.
Két elemet összehasonlítva, a válasz két részre osztja a sorrendeket. Ha pl. azt kapjuk, hogy $x < y$, akkor az olyan sorrendek, amikben x hátrébb van y-nál, már nem jönnek szóba.
Ha az ellenség megint úgy válaszol, hogy minél több sorrend maradjon meg, akkor k kérdés után még szóba jön $\frac{n!}{2^k}$ sorrend.
Ha $\frac{n!}{2^k} > 1$ nem tudjuk megadni a rendezést. $\Longrightarrow$

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

Összefésüléses rendezés

Összefésülés (MERGE):

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

Példa

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

[trans='Replace']

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

Összefésüléses rendezés

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

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

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

Összehasonlítások száma

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

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

$T(n) \le n - 1 + 2(n/2 - 1 + 2T(n/4)) = n - 1 + 2(n/2 - 1) + 4T(n/4).$
$T(n) \le n - 1 + 2(n/2 - 1) + 4(n/4 - 1) + \cdots + 2^{k-1}(n/2^{k-1} - 1) \le n \lceil \log_2 n \rceil.$
Felhasználva, hogy $T(1) = 0$.

Az összefésüléses rendezés konstans szorzó erejéig optimális.

Mozgatások száma: $2n\lceil \log_2 n \rceil$
Tárigény: $2n$ cella (bonyolultabban megcsinálva elég $n + konst.$)

Java animáció: Összefésüléses rendezés

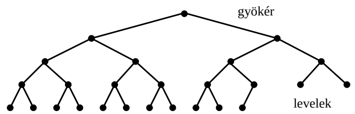
Kupac adatszerkezet

Egy $(U, <)$ rendezett halmaz egy S véges részhalmazát szeretnénk tárolni, hogy a *beszúrás* és a minimális elem törlése (*mintör*) hatékony legyen.

Alkalmazások:

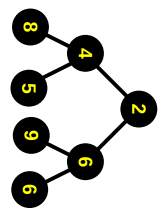
- Jobok indítása
- Több rendezett halmaz összefésülése
- Gyors rendezési algoritmus

[trans='Box,I']Teljes bináris fa:



Bináris fa ábrázolása tömbbel

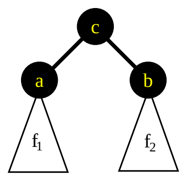
A fa csúcsai az $A[1:n]$ tömb elemei.
Az $A[i]$ csúcs bal fia $A[2i]$, a jobb fia pedig $A[2i+1]$. [trans='Replace']
$\Longrightarrow A[j]$ csúcs apja $A[\lfloor j/2 \rfloor]$



2	4	6	8	5	9	6

Kupac tulajdonság: apa < fia

Kupacépítés



f_1 és f_2 kupacok

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

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

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

Kupacépítés költsége

Bináris fában:

1. szint: 1 pont

2. szint: 2 pont

3. szint: 2^2 pont

$\vdots$

l-edik szint: > 1 és $\leq 2^{l-1}$ pont

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

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

$j = l - i$ (azaz $i = l - j$) helyettesítéssel

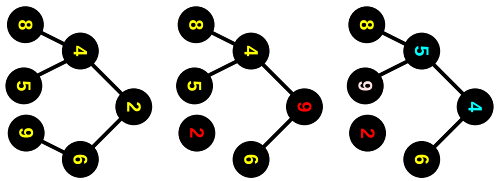
$$\sum_{j=0}^{l-1} j2^{l-j-1} = 2^{l-1}\sum_{j=0}^{l-1} j/2^j < 2^l \leq 2n.$$

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

Tétel. *Kupacépítés költsége:* $O(n)$

MINTÖR

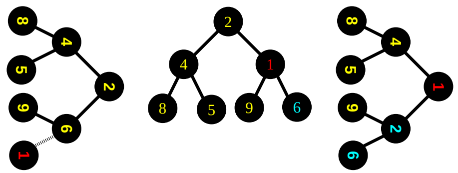
A minimális elem az f gyökérben van, ezt töröljük.
A f-be tesszük a fa utolsó szintjének jobb szélső elemét, majd *felszivárog*(f).



Költség: $O(l) = O(\log_2 n)$

BESZÚR

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



Költség: $O(l) = O(\log_2 n)$

A kupacos rendezés

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

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

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

Java animáció: Kupacos rendezés

3. előadás

Algoritmuselmélet 3. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Február 18.

Gyorsrendezés

[C. A. R. Hoare, 1960]

oszd meg és uralkodj: véletlen s elem a tömbből $\Longrightarrow$ PARTÍCIÓ(s) $\Longrightarrow$

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

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

A PARTÍCIÓ(s) működése

Legyen $i := 1, j := n$,
$\Longrightarrow$ i-t növeljük, amíg $A[i] < s$ teljesül
$\Longrightarrow$ j-t csökkentjük, amíg $A[j] \geq s$
$\Longrightarrow$

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

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

A k-adik elem kiválasztása

Első ötlet: válasszuk ki a legkisebbet, majd a maradékból a legkisebbet, stb.
$\Longrightarrow$ $O(nk)$ lépés
Második ötlet: Rendezzük az elemeket, vegyük ki a k-adikat
$\Longrightarrow$ $O(n \log_2 n)$ lépés
De van jobb: Tetszőleges k-ra meg lehet keresni $O(n)$ lépésben.

Kulcsmanipulációs rendezések

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

Ládarendezés (binsort)

Tudjuk, hogy $A[1:n]$ elemei egy m elemű U halmazból kerülnek ki, pl. $\in \{1, \ldots, m\}$
$\Longrightarrow$ Lefoglalunk egy U elemeivel indexelt B tömböt (m db ládát), először mind üres.
első fázis $\Longrightarrow$ végigolvassuk az A-t, és az $s = A[i]$ elemet a $B[s]$ lista végére fűzzük.
Példa: Tegyük fel, hogy a rendezendő $A[1:7]$ tömb elemei 0 és 9 közötti egészek:

$A:$	5	3	1	5	6	9	6

$B:$		1		3		5 5	6 6			9

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

$B:$		1		3		5 5	6 6			9

$A:$	1	3	5	5	6	6	9

Lépésszám: B létrehozása $O(m)$, első fázis $O(n)$, második fázis $O(n + m)$, összesen $O(n + m)$.

Ez gyorsabb, mint az általános alsó korlát, ha pl. $m \leq cn$.

Java animáció: Láda rendezés

Radix rendezés

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

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

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

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

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

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

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

Miért működik a radix jól?

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

	1969. jan. 18.	1969. jan. 1.	1955. dec. 18.	1955. jan. 18.	1918. dec. 18.
1.	1969. jan. 1.	1969. jan. 18.	1955. dec. 18.	1955. jan. 18.	1918. dec. 18.
2.	1969. jan. 1.	1969. jan. 18.	1955. jan. 18.	1955. dec. 18.	1918. dec. 18.
3.	1918. dec. 18.	1955. jan. 18.	1955. dec. 18.	1969. jan. 1.	1969. jan. 18.

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

Ez lehet gyorsabb az általános korlátnál

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

Java animáció: Radix rendezés

A Batcher-féle páros-páratlan összefésülés

Párhuzamos algoritmus, az összefésüléses rendezés egy változata.
Az összefésülés lépése különböző, a többi rész ugyanaz.
$\mathcal{A} = a_1 < \ldots < a_l$ és $\mathcal{B} = b_1 < \ldots < b_m$ összefésülése
$\Longrightarrow \mathcal{C} = c_1 < \ldots < c_{l+m}$
1. brigád: $a_1 < a_3 < a_5 < \ldots$ és $b_2 < b_4 < b_6 < \ldots$
$\Longrightarrow u_1 < u_2 < u_3 < \ldots$
2. brigád: $a_2 < a_4 < a_6 < \ldots$ és $b_1 < b_3 < b_5 < \ldots$
$\Longrightarrow v_1 < v_2 < v_3 < \ldots$
Tétel. $c_{2i-1} = \min\{u_i, v_i\}$ és $c_{2i} = \max\{u_i, v_i\}$ $(1 \leq i \leq (l + m)/2)$.

Tétel. $c_{2i-1} = \min\{u_i, v_i\}$ *és* $c_{2i} = \max\{u_i, v_i\}$ *$(1 \le i \le (l+m)/2)$.*

Bizonyítás: Belátjuk, hogy
$\mathcal{C}_{2k} := \{c_1, \dots, c_{2k}\} = \{u_1, \dots, u_k\} \cup \{v_1, \dots, v_k\}$.
Mivel $\mathcal{C}$ a növekvő $\mathcal{A}$ és $\mathcal{B}$ összefésülése
$\Longrightarrow \mathcal{C}_{2k} = \{a_1, \dots, a_s\} \cup \{b_1, \dots, b_{2k-s}\}$.
$|\{a_1, \dots, a_s\} \cap \mathcal{U}| = \lceil \frac{s}{2} \rceil$ és $|\{b_1, \dots, b_{2k-s}\} \cap \mathcal{U}| = \lfloor \frac{2k-s}{2} \rfloor$
$|\{a_1, \dots, a_s\} \cap \mathcal{V}| = \lfloor \frac{s}{2} \rfloor$ és $|\{b_1, \dots, b_{2k-s}\} \cap \mathcal{V}| = \lceil \frac{2k-s}{2} \rceil$
Mivel $\lceil s/2 \rceil + \lfloor (2k-s)/2 \rfloor = \lfloor s/2 \rfloor + \lceil (2k-s)/2 \rceil$ beláttuk az első állítást.

$\Longrightarrow \{c_{2i-1}, c_{2i}\} = \{u_i, v_i\}$ □

A tétel miatt $\mathcal{U}$ és $\mathcal{V}$ már egy párhuzamos lépésben összefésülhető.

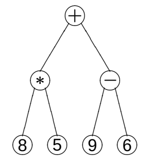
A rendezés:
MSORT$(A[1:n]) :=$
PM(MSORT$(A[1:\lceil n/2 \rceil])$, MSORT$(A[\lceil n/2 \rceil + 1 : n])$),
ahol PM a fenti párhuzamos összefésülés.
Párhuzamos lépésszám: $O(log^2 n)$

Keresőfák

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

Bináris fa bejárása

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

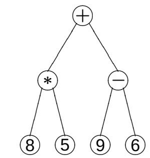


ha x a gyökér, y pedig a 9-es csúcs, akkor

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

PREORDER, INORDER, POSTORDER

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



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

PREORDER: $+ * 8 5 - 9 6$

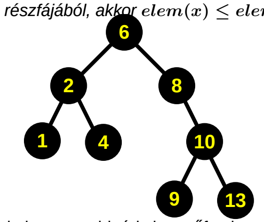
INORDER: $8 * 5 + 9 - 6$

POSTORDER: $8 5 * 9 6 - +$

Lépésszám: $O(n)$

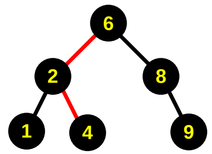
Bináris keresőfa

Definíció (Keresőfa-tulajdonság). *Tetszőleges x csúcsra és az x baloldali részfájában levő y csúcsra igaz, hogy $elem(y) \le elem(x)$. Hasonlóan, ha z egy csúcs az x jobb részfájából, akkor $elem(x) \le elem(z)$.*



Házi feladat: Igazoljuk, hogy egy bináris keresőfa elemeit a fa inorder bejárása *növekvő sorrendben* látogatja meg.
Egy kényelmes megállapodás: a továbbiakban feltesszük, hogy nincsenek ismétlődő elemek a keresőfában.

Naiv algoritmusok



KERES$(4, S)$

KERES(s,S): Összehasonlítjuk s-et S gyökerével s'-vel.

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

Ugyanezt az utat járjuk be a KERES$(5, S)$ kapcsán, de azt nem találjuk meg.

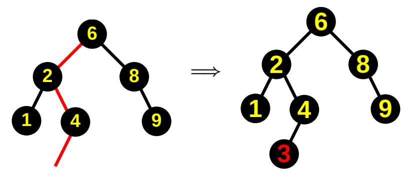
Lépésszám: $O(l)$, ahol l a fa mélysége

MIN: mindig balra lépünk, amig lehet
MAX: mindig jobbra lépünk, amig lehet Lépésszám: $O(l)$

TÓLIG(a, b, S): KERES(a, S) $\Longrightarrow$ INORDER a-tól b-ig Lépésszám: $O(n)$

Naiv BESZÚR

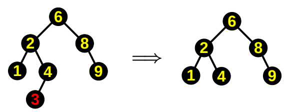
BESZÚR(s, S): KERES(s, S)-sel megkeressük hova kerülne és új levelet adunk hozzá, pl. BESZÚR$(3, S)$:



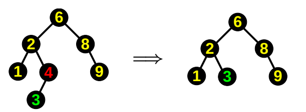
Lépésszám: $O(l)$

Naiv TÖRÖL

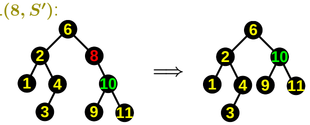
- TÖRÖL(s, S): Ha s levél, akkor trivi, pl. TÖRÖL$(3, S)$:



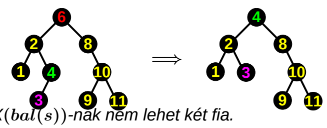
- TÖRÖL(s, S): Ha s-nek egy fia van, akkor: $s \leftarrow$ fiú(s), pl. TÖRÖL$(4, S)$:



- Vagy pl. TÖRÖL$(8, S')$:



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

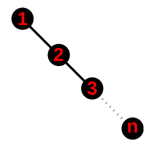


Állítás. *$y :=$ MAX$(bal(s))$-nak nem lehet két fia.*

Bizonyítás:
Ha lenne két fia, akkor lenne egy y' jobb fia is. De ekkor $y' > y$ ↯
Lépésszám: $O(l)$

Faépítés naiv beszúrásokkal

Ha pl. az $1, 2, \dots, n$ sorozatból építünk fát így, akkor ezt kapjuk:
Az építés költsége: $2 + 3 + \dots + (n-1) = O(n^2)$



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

4. előadás

Algoritmuselmélet 4. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Február 25.

Bináris keresőfa

Java animáció: Bináris keresőfa

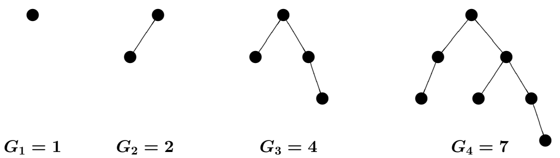
AVL-fák

G. M. Adelszon-Velszkij, E. M. Landisz, 1962 $\Longrightarrow$ kiegyensúlyozott bináris keresőfa

Jelölje $m(f)$ az f bináris fa magasságát (szintjeinek számát), ha x az f fa egy csúcsa; ekkor $m(x)$ jelöli az x-gyökerű részfa magasságát.
Definíció (*AVL*-**tulajdonság).** *Egy bináris keresőfa AVL-fa, ha minden x csúcsára teljesül, hogy*

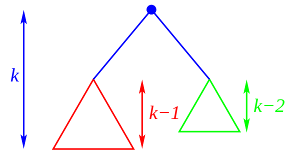
$$|m(bal(x)) - m(jobb(x))| \leq 1.$$

Mekkora a k szintű *AVL*-fa minimális csúcsszáma?



$G_1 = 1$ $G_2 = 2$ $G_3 = 4$ $G_4 = 7$

A k szintszámú minimális csúcsszámú *AVL*-fa gyökerének egyik részfája $k-1$, a másik $k-2$ szintű; az eredeti fa minimalitása miatt pedig mindkét részfa minimális csúcsszámú.



$\Longrightarrow$ rekurzió

$$G_k = 1 + G_{k-1} + G_{k-2}.$$

Tétel. $G_k = F_{k+2} - 1$ *ha* $k \geq 1$.

Bizonyítás: $k = 1,\ 2$ $\surd$ $\Longrightarrow$ indukció:

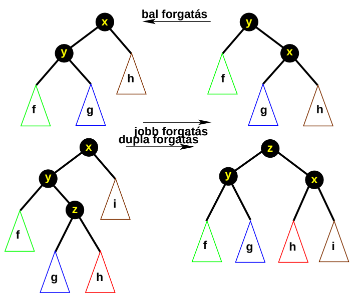
$$G_k = 1 + G_{k-1} + G_{k-2} = 1 + F_{k+1} - 1 + F_k - 1 = F_{k+2} - 1.$$

Tétel. *Egy n-pontú AVL-fa szintjeinek k száma nem több mint $O(\log n)$, pontosabban $k \leq 1.44\log_2(n+1)$.*

Bizonyítás: tétel $\Longrightarrow n \geq F_{k+2} - 1$
Fibonacci-számokra vonatkozó alsó becslésből $n + 1 \geq \phi^k$
$\Longrightarrow \log_\phi(n+1) \geq k \Longrightarrow k \leq 1.44\log_2(n+1)$

Az *AVL*-tulajdonság megőrzése

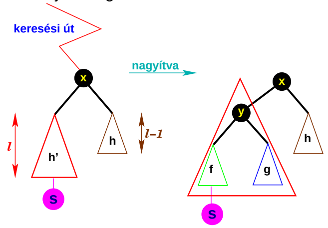
Hogyan lehet a BESZÚR és TÖRÖL eljárásokat úgy megvalósítani, hogy megtartsák az *AVL*-tulajdonságot? $\Longrightarrow$ forgatás



Tétel. *Legyen S egy n csúcsból álló AVL-fa.* BESZÚR(s, S) *után legfeljebb*

egy (esetleg dupla) forgatással helyreállítható az AVL-tulajdonság. A beszúrás költsége ezzel együtt is $O(\log n)$.

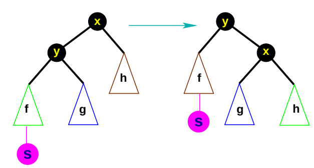
Bizonyítás: Minden csúcsban tartsuk számon $m(f)$-et, az f gyökerű részfa mélységét, ez könnyen karban tartható.
KERES(s, S) segítségével megkeressük s helyét.
A keresési utat végigjárva megkeressük a legalsó olyan csúcsot, ahol sérül az *AVL*-tulajdonság $\Longrightarrow x$



x definíciója $\Longrightarrow m(h') \neq m(h)$
feltehetjük, hogy $m(h') = l, m(h) = l-1$
Két eset van:

a) s az f részfába kerül

b) s a g részfába kerül

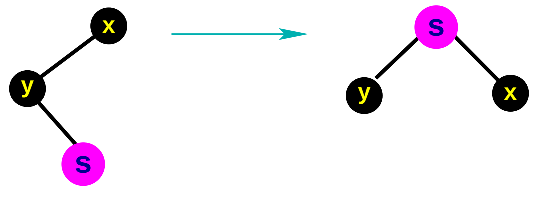


a) eset:
$m(f) < m(g) \Longrightarrow x$-ben nem sérül az *AVL*-tul.
$m(f) > m(g) \Longrightarrow y$-ben is sérül az *AVL*-tul.
$\Longrightarrow$
$m(f) = m(g) = m(h) =$
$= m(y) - 1 = l - 1$
$\Longrightarrow$ jobb forgatás helyre állítja az *AVL*-tul.

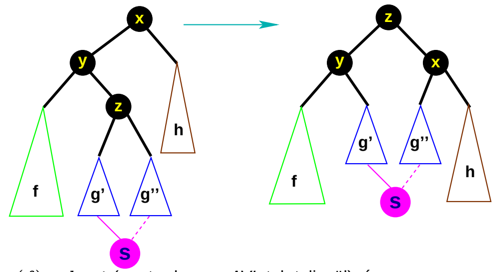
A forgatás után y mindkét részfájának a magassága l lesz, x új részfái g és h, mindkettő szintszáma $l-1$.

y feletti csúcsok magassága nem változik, így az *AVL*-feltétel feljebb is megmarad a kereső út mentén.

b1) eset: s a g részfába került és s az y csúcs fia ($l = 1$)
$\Longrightarrow$ dupla forgatás



b2) eset: s a g részfába került és s nem az y csúcs fia ($l > 1$)



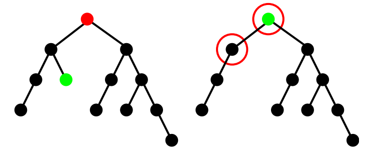
$\Longrightarrow m(f) = l - 1$ (mert y-ban az *AVL*-tul. teljesül), és
$m(g') = m(g'') = l - 2$ (mert z-ben sincs baj az *AVL*-tulajdonsággal)
$\Longrightarrow$ dupla forgatás elég

Költség: $1.44 \log_2(n + 1) = O(\log n)$

Törlés *AVL*-fából

Tétel. *Az n pontú AVL-fából való naiv törlés után legfeljebb $1.44 \log_2 n$ (szimpla vagy dupla) forgatás helyreállítja az AVL-tulajdonságot.*

Nem bizonyítjuk, a módszer hasonló, de nem mindig elég egy forgatás. Pl.:



Java animáció: *AVL*-fa

További kiegyensúlyozott fák

Az *AVL*-tulajdonság csak egy a lehetséges kiegyensúlyozottsági feltételek közül. Általánosítása:
Definíció. HB[k]-fák: (C. C. Foster, 1973)
Legyen $k \geq 1$ egy egész szám. Egy bináris keresőfa $HB[k]$-fa, ha minden x csúcsára teljesül, hogy

$$|m(bal(x)) - m(jobb(x))| \leq k.$$

$\Longrightarrow HB[1]$-fák $\rightarrow$ *AVL*-fák

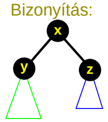
Súlyra kiegyensúlyozott fák

A részfák súlya legyen a csúcsszámuk: $s(f)$.
Definíció. *Egy bináris keresőfát súlyra kiegyensúlyozott fának (röviden SK-fának) nevezünk, ha minden x belső csúcsára teljesül, hogy*

$$\sqrt{2} - 1 < \frac{s(bal(x))}{s(jobb(x))} < \sqrt{2} + 1.$$

Feladat: Igazoljuk, hogy a leheletnyivel szigorúbb $1/2 < \frac{s(bal(x))}{s(jobb(x))} < 2$ korlátokat már csak az l szintből álló $2^l - 1$ pontú bináris fák a teljesítik.
Tétel. *Egy n csúcsú SK-fa mélysége $\leq 2 \log_2 n + 1$.*

Bizonyítás:



$s(x) > s(y) + s(z) > s(y) + (\sqrt{2} - 1)s(y) = \sqrt{2}s(y)$
Legyenek $x_1, x_2, \ldots, x_k$ egy k-hosszúságú gyökértől levélig menő út csúcsai.
$n = s(x_1) > \sqrt{2}s(x_2) > (\sqrt{2})^2 s(x_3) > \cdots > (\sqrt{2})^{k-1} s(x_k) = (\sqrt{2})^{k-1} = 2^{(k-1)/2}$.
$\Longrightarrow \quad \checkmark$

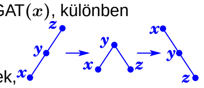
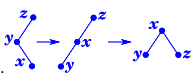
S-fák

Splay-tree: D. D. Sleator és R. E. Tarjan, 1983.
Olyan bináris fa, ami tanul: pl. gyakran keresett elemet feljebb teszi.
$\Longrightarrow$ Hosszú átlagos műveletsor alatt az egy lépésre eső költség optimális lesz.
Fő öltlet:

> KIFORDÍT(x, f) átszervezi az f S-fát úgy, hogy x lesz az új gyökér, ha $x \in f$; különben f gyökere x valamelyik szomszédja lesz:
> vagy $\max\{y \in f;\ y < x\}$ vagy $\min\{y \in f;\ y > x\}$.

KIFORDÍT(x, f) implementációja: KERES$(x, f) \Longrightarrow$ ha $x \in f$, akkor majd x-et visszük fel, ha $x \notin f$, akkor a keresés x egyik szomszédjánál ($\max\{y \in f;\ y < x\}$ vagy $\min\{y \in f;\ y > x\}$) ér véget, vegyük ezt.
$\Longrightarrow$ feltehetjük, hogy $x \in f$.

A következő eljárás x-et maximum két szinttel viszi feljebb, addig alkalmazzuk, amig x felér.

> (0) Ha x gyökér, akkor készen vagyunk.
> (* A továbbiakban jelölje y az x apját. *)
> (1) Ha x-nek nincs nagyapja, akkor FORGAT(x), különben
>
> 
>
>
> (2) ha x és y is baloldali (jobboldali) gyerek,
> akkor FORGAT(y), majd FORGAT(x), különben
> (3) ha x és y különböző oldali gyerekek,
>
> 
>
>
> akkor FORGAT(x), majd ismét FORGAT(x).

Műveletek az S-fákban

A keresőfákra jellemző KERES(x, f), BESZÚR(x, f) és TÖRÖL(x, f) műveletek a szokásosak.
A RAGASZT(f, f') művelet az f és f' S-fákból egyetlen S-fát szervez, feltéve, hogy $x < y$ teljesül minden $x \in f$ és $y \in f'$ kulcsra.
A VÁG(x, f) művelet szétvágja f-et az f' és f'' S-fákra úgy, hogy $y \leq x \leq z$ teljesül minden $y \in f'$ és $z \in f''$ csúcsra.
Tétel. *Az ismertetett műveletek mindegyike megvalósítható konstans számú KIFORDÍT és konstans számú elemi operáció (összehasonlítás, mutató beállítás, stb.) segítségével.*

Bizonyítás: Csak két művelet, a RAGASZT és a TÖRÖL esetét nézzük meg közelebbről, többi trivi.

RAGASZT(f, f') $\Longrightarrow$ először KIFORDÍT$(+\infty, f) =: f^* \Longrightarrow f^*$ gyökere x az új fa legnagyobb kulcsa
$\Longrightarrow$ csatoljuk f'-t x jobb fiának

TÖRÖL(x, f) $\Longrightarrow$ KIFORDÍT(x, f)ha a kapott fa gyökere nem x, akkor $x \notin f \Longrightarrow \checkmark$
Ha x az új fa gyökere, töröljük és a kapott két f_1 és f_2 részfára RAGASZT(f_1, f_2)
Tétel. *Egy üres S-fából induló olyan m műveletből álló sorozat költsége, melyben n beszúrás van, $O(m \log n)$.*

Java animáció: S-fa

5. előadás

Algoritmuselmélet 5. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Február 26.

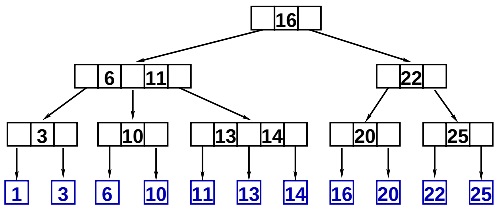
2-3-fák

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

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

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

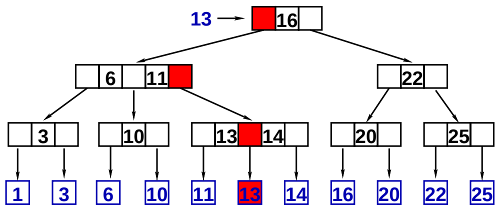
Példa 2-3-fára



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

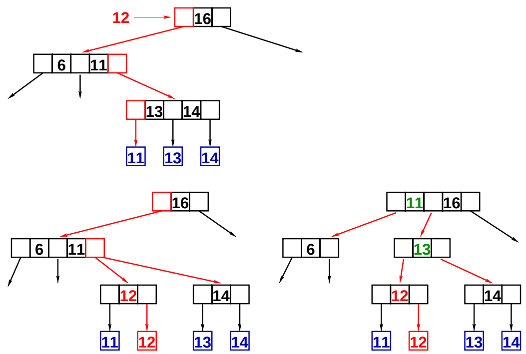
Bizonyítás: Minde belső csúcsnak legalább 2 fia van $\Longrightarrow$
az i-edik szinten legalább 2^{i-1} csúcs van $(1 \leq i \leq m)$. $\Longrightarrow$
$2^{m-1} \leq n, \Longrightarrow m - 1 \leq \log_2 n.$ $\surd$

Keresés 2-3-fában



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

BESZÚR 2-3-fába



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

TÖRÖL 2-3-fából

Legyen x a legalsó belső csúcs a kereső út mentén.

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

Ez is „felgyűrűzhet” $\Longrightarrow$

Lépésszám: $O(m)$

B-fák

R. Bayer, E. McCreight, 1972
A 2-3-fa általánosítása.
Nagy méretű adatbázisok, külső táron levő adatok feldolgozására használják.
Több szabvány tartalmazza valamilyen változatát.

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

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

B-fa definíciója

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

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

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

c) A levelek a gyökértől egyforma messze vannak.

d) Egy csúcsnak legfeljebb m fia lehet.

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

A belső csúcsok hasonlítanak a 2-3-fák belső csúcsaira. Egy belső csúcs így néz ki: $\boxed{m_0 \,|\, s_1 \,|\, m_1 \,|\, s_2 \,|\, m_2 \,|\, \dots \,|\, s_i \,|\, m_i}$

A B-fa szintszáma

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

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

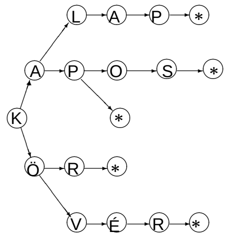
Minden művelet lépésszáma: $\sim \frac{\log_2 n - 1}{\log_2 \lceil \frac{m}{2} \rceil} = O(\log n)$, de a konstans kicsi, ha m nagy.

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

Szófák

Legyen Σ egy véges halmaz, Σ^* a Σ-beli elemekből alkotott véges hosszú sorozatok.
Σ^* egy részhalmazát (szavak egy halmazát) szeretnénk tárolni.

KÖR, KÖVÉR, KAPOS, KAP, KALAP $\Longrightarrow$



$*$ azt jelenti, hogy itt a szó vége
A szófa adatszerkezet érzéketlen a beszúrások sorrendjére, a fa alakja csak a tárolt szavak összességétől függ.

A tömbbel megvalósított szófában való keresés: a szóbeli betűk száma

Hash-elés

Hash-elés

Nem tételezzük fel a lehetséges kulcsok összességének (az U univerzumnak) a rendezettségét.
Olyan módszercsalád, amely a keresés, beszúrás, törlés és módosítás gyors és egyszerű megvalósítását teszi lehetővé.

Nincs rendezés $\Longrightarrow$ nincs MIN, MAX, ...

Cél $\Longrightarrow S \subseteq U$ kulcshalmazzal azonosított állomány megszervezése úgy, hogy a fenti műveletek átlagos értelemben hatékonyak legyenek.

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

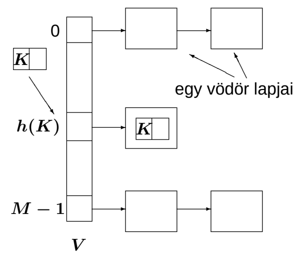
Hash-elés alapvető ötelete

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

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

Vödrös hash-elés

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



Kulcsok a vödörben

Hogyan helyezzük az új kulcsot a vödörbe?

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

Keresés a hash-táblában

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

Törlés ugyanígy.

Hash-elés költsége

Külső táras szerkezet $\Longrightarrow$ lapelérések száma.
M vödör van, és l-lapnyi rekordot tárolunk
$\Longrightarrow$ egy vödörbe átlagosan $\approx l/M$ lap kerül
$\Longrightarrow$ átlagos lánchossz

$\Longrightarrow$ Keresés átlagos lépéshossza: $1 + l/M$

Hogyan válasszuk meg M-et?:

l/M legyen kb. 1, de hagyjunk rá 20%-ot

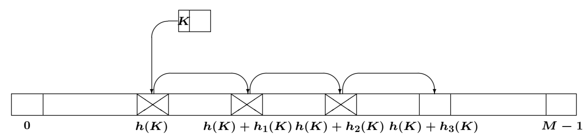
Példa: 1 000 000 rekordból álló állományt szeretnénk láncolásos módszerrel kezelni, egy lapon 5 rekord fér el.
Ekkor $l = 1\,000\,000/5 = 200\,000 \Longrightarrow M \approx 220\,000 - 240\,000$
$\Longrightarrow$ keresés átlagos költsége valamivel 2 lapelérés alatt marad.

Hashelés nyitott címzéssel

Csak belső memóriás módszerként hasznosak.

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

Legyen $0, h_1(K), h_2(K), \ldots, h_{M-1}(K)$ a $0, 1, \ldots, M - 1$ számok egy permutációja
$\Longrightarrow$ végigpróbálgatjuk a $h(K) + h_i(K) \pmod M$ sorszámú cellákat ($i = 0, 1, \ldots, M - 1$) az első üres helyig, ahol a rekordot elhelyezzük.
$\Longrightarrow$ Ha nincs üres, a tábla betelt.



Lineáris próbálás

$h_i(K) := -i$

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

Sikeres keresés átlagos költsége:

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

Sikertelen keresés átlagos költsége:

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

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

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

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

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

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

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

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

Java animáció: Hash-elés lineáris próbával

6. előadás

Algoritmuselmélet 6. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Március 4.

Hash-elés álvéletlen próbával

A $0, h_1(K), h_2(K), \ldots, h_{M-1}(K)$ próbasorozat a $0, 1, \ldots, M-1$ számoknak egy a K kulcstól független álvéletlen permutációja.

A sorozatnak gyorsan és hatékonyan reprodukálhatónak kell lennie $\Longleftrightarrow$ véletlen

Ha $h(K) = h(L)$, akkor a K és L kulcsok teljes próbasorozata is megegyezik $\Longrightarrow$ másodlagos csomósodásnak

Kvadratikus maradék próba

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

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

$\Longrightarrow$ Belátjuk, hogy ez tényleg permutáció.

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

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

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

Az utolsó lépésnél a kis Fermat-tételt használtuk. ↯

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

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

$i^2 \not\equiv -j^2 \pmod M \Longleftarrow (ij^{-1})^2 \not\equiv -1 \pmod M$ $\checkmark$

Sikeres keresés költsége:

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

Sikertelen keresés költsége:

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

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

Kettős hash-elés

G. de Balbine, J. R. Bell, C. H. Kaman, 1970 körül.

Lényeg: h mellett egy másik h' hash-függvényt is használunk
a $h'(K)$ értékek relatív prímek legyenek az M táblamérethez
A K kulcs próbasorozata: $h_i(K) := -ih'(K)$.
Ha M és $h'(K)$ relatív prímek
$\Longrightarrow 0, -h'(K), -2h'(K), \ldots, -(M-1)h'(K)$ sorozat elemei mind különbözők modulo M
Fontos sajátossága: különböző K és K' kulcsok próbasorozatai jó eséllyel akkor is különbözők lesznek, ha $h(K) = h(K')$.
A legjobb ismert implementációk időigénye (empirikus adatok alapján)

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

- A kettős hash-elés kiküszöböli mindkét-féle csomósodást
- Sikertelen keresés esetén minden érdekes α-ra gyorsabb, mint a lineáris próbálás
- Sikeres kereséskor csak az $\alpha \ge 0,8$ tartományban lesz gyorsabb a lineáris próbálásnál.

Hash-függvények

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

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

Osztómódszer

Legyen $h(K) := K \pmod M$,
ahol M a tábla vagy a vödörkatalógus mérete.
Feltesszük, hogy a kulcsok egész számok.
A $h(K)$ számítása gyors és egyszerű.

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

M prím $\Longrightarrow$ lényeges feltétel a kvadratikus maradék próbánál.
Könnyű hozzájuk relatív prím számot találni $\Longrightarrow$ kettős hash-elés

Szorzómódszer

β egy rögzített paraméter

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

$\{x\}$ jelöli az x valós szám törtrészét

Szemléletesen $\Longrightarrow$ $\{\beta K\}$ kiszámításával a K kulcsot „véletlenszerűen" belőjük a $[0, 1)$ intervallumba $\Longrightarrow$ az eredményt felskálázzuk a címtartományba.

Hatékonyan számítható speciális eset:
$M = 2^t$, $w = 2^{32}$, és legyen A egy a w-hez relatív prím egész.
Ekkor $\beta = \frac{A}{w}$ választás mellett $h(K)$ igen jól számolható.
A számok bináris ábrázolásával dolgozva lényegében egy szorzást és egy eltolást kell elvégezni.

A szorzómódszer jól viselkedik számtani sorozatokon
pl. termék1, termék2, termék3, ... esetében
Megmutatható, hogy a $h(K), h(K+d), h(K+2d)\ldots$ sorozat közelítőleg számtani sorozat lesz, azaz h jól „szétdobja" a kulcsok számtani sorozatait.
Tétel (T. Sós Vera, 1957). *Legyen β irracionális szám, és nézzük a 0, $\{\beta\}$, $\{2\beta\}$, ..., $\{n\beta\}$ pontok által meghatározott $n+1$ részintervallumot $[0, 1)$-ben. Ezek hosszai legfeljebb 3 különböző értéket vehetnek fel, és $\{(n+1)\beta\}$ a leghosszabbak egyikét fogja két részre vágni.*

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

A kettős hash-elés második függvénye

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

Ha M prím $\Longrightarrow$
$$h'(K) := K \pmod{M-1} + 1.$$
$\Longrightarrow$ $h'(K)$ és M relatív prímek
$\Longrightarrow$ elég sok különböző próbasorozatot ad

Java animáció: Hash-elés

Szekvenciális keresés

Ha egy állomány (tömb, lista, stb.) szegényes szerkezetű $\Longrightarrow$ nincs jobb , mint „elejétől a végéig" bejárni, vagy legalábbis addig, amíg a keresett adatot meg nem találjuk.

Ha egyenlő eséllyel kell keresni az elemeket $\Longrightarrow$
Sikeres keresés átlagos költsége:
$$\frac{1+2+\cdots+N}{N} = \frac{N+1}{2}.$$

Sikertelen keresés költsége: N

Ha az állományban az elemek nagyság szerint rendezettek $\Longrightarrow$

Sikeres keresés átlagos költsége: $\frac{N+1}{2}$.
Sikertelen keresés költsége:
$$\frac{1+2+\cdots+N-1+N+N}{N+1} = \frac{N}{2} + \frac{N}{N+1} < \frac{N}{2} + 1.$$

Tegyük fel, hogy csak sikeres keresésekkel van dolgunk, és legyen p_i annak a valószínűsége, hogy az R_i rekordot keressük. $\Longrightarrow$

Sikeres keresés átlagos költsége:
$$C_N = p_1 + 2p_2 + 3p_3 + \cdots + Np_N.$$

Hogy érdemes sorba rendezni az R_i rekordokat? $\Longrightarrow$
csökkenő sorrendben

Különböző eloszlások esetén:

Egyenletes: $p_i = \frac{1}{N}$ $\Longrightarrow$ $C_N = \frac{N+1}{2}$

Egy nagyon ferde eloszlás: $p_i = \frac{1}{2^i}$ $\Longrightarrow$ $C_N = 2 - \frac{1}{2^{N-1}}$

Zipf eloszlás: $p_i = \frac{1}{iH_N}$, ahol
$$H_N = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} = \log n + 0.55721\ldots + o(1)$$
$\Longrightarrow$
$$C_N = \sum_{i=1}^{N} ip_i = \sum_{i=1}^{N} i\frac{1}{iH_N} = \frac{N}{H_N} \approx \frac{N}{\log N}$$

80-20 szabály: Tapasztalat $\Longrightarrow$ Az adatelérési igények 80%-a a rekordoknak körülbelül csak 20%-át érinti.

$$p_i = \frac{c}{i^{1-\vartheta}}, \text{ ahol}$$
$$\vartheta = \frac{\log 0,8}{\log 0,2} \approx \frac{1}{7}, \quad c = \frac{1}{H_N^{(1-\vartheta)}} \quad \text{és} \quad H_N^{(1-\vartheta)} = 1 + \frac{1}{2^{(1-\vartheta)}} + \cdots + \frac{1}{N^{(1-\vartheta)}}$$
$\Longrightarrow$ $C_N \approx 0,122N$

Önszervező módszerek

Mit tehetünk abban az esetben, ha a p_i keresési valószínűségeket nem ismerjük, vagy esetleg azok idővel változnak?

- A keresett (és megtalált) R_i elemet a tábla elejére visszük. Az eredmény:

R_i	R_1	R_2	$\ldots$		R_N

- A keresett (és megtalált) R_i elemet felcseréljük a megelőzővel.

R_1	$\ldots$	R_i	R_{i-1}	$\ldots$		R_N

Ha az eloszlás időben változik, akkor az első megoldás ajánlatos. Ha a $\{p_i\}$ eloszlás stabil, azaz időben nem változik, akkor a második heurisztika eredményesebb.

Információtömörítés

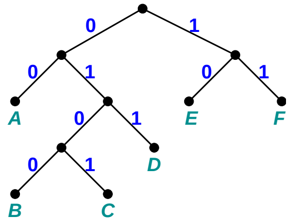
A $b_1, \ldots, b_n$ betűkből álló szöveget szeretnénk bitsorozatként kódolni.

uniform kódolás $\Longrightarrow$ $\lceil \log_2 n \rceil$ a legrövidebb lehetséges kódhosszúság egy betűre

eltérő hosszú kódszavak $\Longrightarrow$ dekódolás problémásabb

Definíció. *Egy bitsorozatokból álló kód prefix kód, ha egy betű kódja sem prefixe (kezdőszelete) egy másik kódjának. Formálisan: ha x és y két különböző betű kódja, akkor nincs olyan z bitsorozat, melyre $xz = y$.*

Egy prefix-tulajdonságú kóddal leírt üzenet egyértelműen visszafejthető.

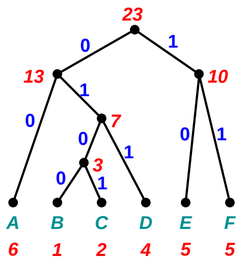


Probléma: Adott egy szöveg, melyben a b_i karakter q_i-szer fordul elő. Keressünk olyan fát, amelyre a $\sum q_i h(b_i)$ összeg minimális, ahol egy x csúcsra $h(x)$ a gyökértől x-ig vezető úton bejárt élek száma.

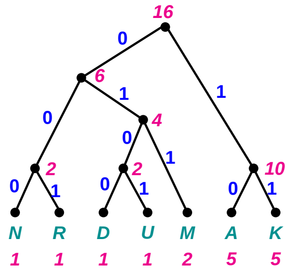
Huffman-kód

Optimális ilyen fa:

- Kezdetben n izolált csúcspontunk van. b_i címkéje legyen q_i.
- Tegyük fel, hogy már megépítettük az $S_1, \ldots, S_k$ fákat, ezek gyökérpontjai $x_1, \ldots, x_k$, utóbbiak címkéi az $r_1, \ldots, r_k$ számok. Ekkor vesszük a két minimális címkéjű gyökeret (legyenek ezek x_i és x_j).
- Ezek fölé egy új y gyökérpontot teszünk, melynek fiai x_i és x_j. Az y címkéje $r_i + r_j$.
- A fák száma eggyel csökken. Megállunk, ha már csak egy fa marad. Összesen $n - 1$ ilyen összevonó lépés szükséges.



KAKUKKMADARAMNAK $\Longrightarrow$ 7 betű $\Longrightarrow$ uniform kódolással betűnként 3 bit, összesen 48 bit.



A betűk kódjai: K: 11, A: 10, M: 011, U: 0101, D: 0100, N: 000, R: 001

kódszó: $1110110101111101110010010001100110001011$

KAKUKKMADARAMNAK

összesen 40 bit

Optimális kód

Tétel. *A Huffman-fa optimális. Pontosabban fogalmazva, a Huffman-fa esetén az $I = \sum q_i h(b_i)$ összeg minimális azon bináris fák között, amelyek levelei $b_1, \ldots, b_n$.*

Bizonyítás: A Huffman-fa által adott I érték legyen $H(q_1, q_2, \ldots, q_n)$. konstrukció $\Longrightarrow$

$$H(q_1, \ldots, q_n) = H(q_1 + q_2, q_3, \ldots, q_n) + q_1 + q_2$$

Jelölje $Opt(q_1, q_2, \ldots, q_n)$ a q_i gyakoriságok esetén elérhető optimális I-értéket.

Lemma. *Legyen továbbra is $q_1 \le q_2$ a két legkisebb gyakoriság. Ekkor*

$$Opt(q_1, q_2, \ldots, q_n) = Opt(q_1 + q_2, q_3, \ldots, q_n) + q_1 + q_2.$$

Bizonyítás: Minden belső csúcsnak két fia van.
Legyen most x egy levél az optimális fánkban, melyre $h(x)$ a lehető legnagyobb, x-nek van a fában egy y testvére, címkéik q_1 és q_2.
Töröljük az x és y csúcsokat, és írjuk az új levélbe a $q_1 + q_2$ címkét.
A fa I-értéke legyen J. $\Longrightarrow$
$Opt(q_1, q_2, \ldots, q_n) = J - (h(x) - 1)(q_1 + q_2) + q_1 h(x) + q_2 h(x) = J + q_1 + q_2,$
$\Longrightarrow$ az új fának is optimálisnak kell lennie a $q_1 + q_2, q_3, \ldots, q_n$ gyakoriságokra vonatkozóan, hiszen, ha J-n tudnánk javítani, akkor az eredeti fán is. $\surd$

Bizonyítás: (Tétel) Indukció, $n = 2$ $\surd$
Tegyük fel, hogy legfeljebb $n - 1$ betű esetén igaz $\Longrightarrow$
Az indukciós feltevés szerint

$$Opt(q_1 + q_2, q_3, \ldots, q_n) = H(q_1 + q_2, q_3, \ldots, q_n)$$

$\Longrightarrow$

$$Opt(q_1, q_2, q_3, \ldots, q_n) = H(q_1, q_2, q_3, \ldots, q_n)$$

Java animáció: Huffman-fa

7. előadás

Algoritmuselmélet 7. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Március 11.

Múltkori animációk

Java animáció: Hash-elés

Java animáció: Huffman-fa

A Lempel–Ziv–Welch-módszer

A. Lempel és J. Ziv, 1970; T. Welch 1984

Használja: GIF, v.42bis, compress; ZIP, ARJ, LHA

Nem betűnként kódól, hanem a szöveg bizonyos szavaiból *szótárat* épít. $\Longrightarrow S$

- az egybetűs szavak, azaz Σ elemei mind benne vannak S-ben;
- ha egy szó benne van a szótárban, akkor annak minden kezdődarabja is benne van;
- a szótárban tárolt szavaknak fix hosszúságú kódjuk van; az $x \in S$ szó kódját $c(x)$ jelöli.

Gyakorlatban a kódok hossza $\sim$12-15 bit.

LZW kódolás

- az összenyomni kívánt szöveget S-beli szavak egymásutánjára bontjuk
- a szavakat a szótárbeli kódokkal helyettesítjük
- Az eredeti szöveg olvasásakor egyidőben épül, bővül az S szótár
- nincs optimalizálás, de a gyakorlatban jól működik

A szótár egyik szokásos tárolási módja a szófa adatszerkezet.

Ha az olvasás során egy $x \in S$ szót találunk, aminek a következő Y betűvel való folytatása már nincs S-ben, akkor $c(x)$-et kiírjuk a kódolt szövegbe.
Az xY szót felvesszük az S szótárba. A szó $c(xY)$ kódja a legkisebb még eddig az S-ben nem szereplő kódérték lesz.
Ezután az Y betűvel kezdődően folytatjuk a bemeneti szöveg olvasását.

Legyen z egy szó típusú változó, K egy betű típusú változó. A z változó értéke kezdetben az összenyomni szánt állomány első betűje. Végig teljesül, hogy $z \in S$.

(0) Olvassuk a bemenő állomány következő betűjét K-ba.
(1) Ha az előző olvasási kísérlet sikertelen volt (vége a bemenetnek), akkor írjuk ki $c(z)$-t, és álljunk meg.
(2) Ha a zK szó is S-ben van, akkor $z \leftarrow zK$, és menjünk vissza (0)-ra.
(3) Különben (azaz ha $zK \not\in S$) írjuk ki $c(z)$-t, tegyük a zK szót S-be. Legyen $z \leftarrow K$, majd menjünk vissza (0)-ra.

A $c(x)$ kódok rögzített hosszúságúak. Ha például ez a hosszúság 12 bit, akkor az S szótárba összesen 4096 szó kerülhet. $\Longrightarrow$
Ha a szótár betelt, nem bővítünk tovább, úgy folytatjuk.

Példa LZW-re

Legyen $\Sigma = \{a, b, c\}$ és $c(a) = 1$, $c(b) = 2$, $c(c) = 3$.

Szótár

$a \rightarrow 1$
$b \rightarrow 2$
$c \rightarrow 3$

$ababcbababaaaaaaa$
$1\ 2\ 4\ 3\ 5\ 8\ 1\ 10\ 11\ 1$

$0000\ 0001\ 0011\ 0010\ 0100$
$1000\ 0000\ 1001\ 1010\ 0000$

$\Rightarrow\ ab \rightarrow 4$
$\Rightarrow\ ba \rightarrow 5$
$\Rightarrow\ abc \rightarrow 6$
$\Rightarrow\ cb \rightarrow 7$
$\Rightarrow\ bab \rightarrow 8$
$\Rightarrow\ baba \rightarrow 9$
$\Rightarrow\ aa \rightarrow 10$
$\Rightarrow\ aaa \rightarrow 11$
$\Rightarrow\ aaaa \rightarrow 12$

Dekódolás

Elvégezhető pusztán az egybetűs szavak kódjainak, valamint a kódok képzési szabályának ismeretében, nem kell tárolni S-et.

Pl. Ha a kódolt szöveg: $1\ 2\ 4\ 3\ 5\ 8\ 1\ 10\ 11\ 1$ és tudjuk, hogy $c(a) = 1$, $c(b) = 2$, $c(c) = 3 \Longrightarrow$
Ez első két jel betű kódja $\Longrightarrow ab$ $\surd$
ab nem volt a kezdeti S-ben, de az eleje igen; $\Longrightarrow$ a kódoló algoritmus ezt $c(ab) = 4$ értékkel S-be tette $\Longrightarrow abab \Longrightarrow$
ba volt a következő szó, ami S-be került $\Longrightarrow c(ba) = 5 \ldots$ Ha itt tartunk

$1\ 2\ 4\ 3\ 5\ 8\ 1\ 10\ 11\ 1$
$ababcba$

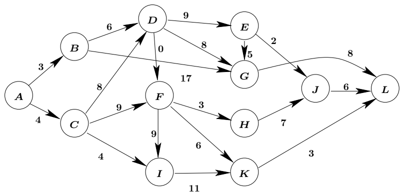
$a \rightarrow$ 1
$b \rightarrow$ 2
$c \rightarrow$ 3
$ab \rightarrow$ 4
$ba \rightarrow$ 5
$abc \rightarrow$ 6
$cb \rightarrow$ 7

$\Longrightarrow$ a 8-as kódú szó $ba*$ alakú, következő betű biztos b
$c(bab) = 8$
$ababcbabab$

Java animáció: LZW-kódolás
Laczay Bálint féle GIF animáció

Gráfalgoritmusok

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

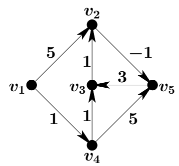


Adjacencia-mátrix

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

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

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

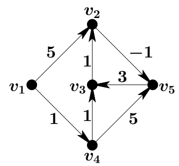


$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Súlyozott adjacencia-mátrix

Ha költségek is vannak $\Longrightarrow$

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

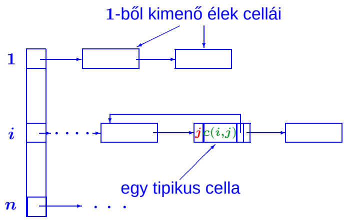


$$C = \begin{pmatrix} 0 & 5 & * & 1 & * \\ * & 0 & * & * & -1 \\ * & 1 & 0 & * & * \\ * & * & 1 & 0 & 5 \\ * & * & 3 & * & 0 \end{pmatrix}$$

Hátránya $\Longrightarrow$ a mérete (n^2 tömbelem) teljesen független az élek számától.

Éllistás megadás

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



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

A legrövidebb utak problémája

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

Feladat. *Mekkora a legrövidebb út egy adott pontból egy másik adott pontba?*
Feladat. *Mekkora a legrövidebb út egy adott pontból az összes többibe?*
Feladat. *Mekkora a legrövidebb út bármely két pont között?*

A G gráf egy u-t v-vel összekötő (nem feltétlenül egyszerű) $u \rightsquigarrow v$ irányított útjának a *hossza* az úton szereplő élek súlyainak összege.
Legrövidebb $u \rightsquigarrow v$ út $\Longrightarrow$ egy olyan $u \rightsquigarrow v$ út, melynek a hossza minimális a G-beli $u \rightsquigarrow v$ utak között.
u és v csúcsok (G-beli) $d(u, v)$ *távolsága*:
— 0, ha $u = v$;
— ∞, ha nincs $u \rightsquigarrow v$ út
— egyébként pedig a legrövidebb $u \rightsquigarrow v$ út hossza.
Vigyázat, itt u és v nem felcserélhető: ha az egyik csúcs valamilyen távol van a másiktól, akkor nem biztos, hogy a másik is ugyanolyan távol van az egyiktől!

Dijkstra módszere

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

$D[\,] \Longrightarrow$

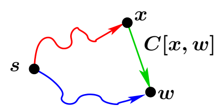
- Egy a G csúcsaival indexelt tömb
- az eljárás során addig megismert legrövidebb $s \rightsquigarrow v$ utak hossza
- Felső közelítése a keresett $d(s, v)$ távolságnak
- A közelítést lépésről lépésre finomítjuk, végül $d(s, v)$-t érjük el.

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

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

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

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



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

Dijkstra algoritmusa adjacencia-mátrixszal

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

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

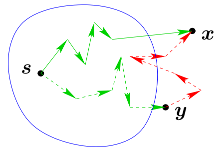
Tétel. *A (2) ciklus minden iterációs lépése után érvényesek a következők:*

(a) KÉSZ pontjaira $D[v]$ a legrövidebb $s \rightsquigarrow v$ utak hossza.

(b) Ha $v \in$ KÉSZ, akkor van olyan $d(s, v)$ hosszúságú (más szóval legrövidebb) $s \rightsquigarrow v$ út is, amelynek minden pontja a KÉSZ halmazban van.

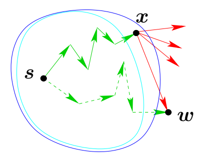
(c) Külső (vagyis $w \in V \setminus$ KÉSZ) pontokra $D[w]$ a legrövidebb különleges $s \rightsquigarrow w$ utak hossza.

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



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

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



(c) A $j + 1$. iteráció előtt

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

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

$\Longrightarrow$ Elég megnézni, hogy $D[w]$ vagy $d(s, x) + C[x, w]$ nagyobb
Lépészám: (2) ciklus $O(n)$, ez lefut $n - 1$-szer $\Longrightarrow O(n^2)$

8. előadás

Algoritmuselmélet 8. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Március 12.

Dijkstra animációk

Java animáció: Dijkstra-algoritmus, másik, harmadik

Dijkstra algoritmusa éllistával

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

Java animáció: Dijkstra-algoritmus

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

Dijkstra irányítatlan gráfokra is működik.

A legrövidebb utak nyomon-követése

Minden pontra tárolunk és karbantartunk egy $P[x]$ csúcsot is, ami megadja egy az eddig ismert hozzá vezető legrövidebb úton az utolsó előtti csúcsot.

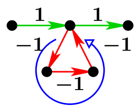
Kezdetben $P[v] := s$ minden $v \in V$-re.
$\Longrightarrow$ (3) ciklus belsejében, ha egy külső w csúcs $D[w]$ értékét megváltoztatjuk, akkor $P[w] := x$.

Lépésszám: $O(n^2)$

A Bellman—Ford-módszer

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

Ha van negatív összhosszúságú irányított kör, akkor a legrövidebb út ∞.



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

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

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

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

A $T[1, j]$ sor kitöltése $\Longrightarrow T[1, j] = C[1, j]$
Tegyük fel ezután, hogy az i-edik sort már kitöltöttük
$\Longrightarrow T[i, 1], T[i, 2], \ldots, T[i, n]$ értékekre $(*)$ igaz. $\Longrightarrow$

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

$\Longleftarrow$ Ugyanis egy legfeljebb $i+1$ élből álló $\pi = 1 \rightsquigarrow j$ út kétféle lehet:
(a) Az útnak kevesebb, mint $i+1$ éle van. Ekkor ennek a hossza szerepel $T[i, j]$-ben.
(b) Az út éppen $i+1$ élből áll. Legyen l a π út utolsó előtti pontja. Ekkor a π út $1 \rightsquigarrow l$ szakasza i élből áll, és minimális hosszúságú a legfeljebb i élű $1 \rightsquigarrow l$ utak között $\Longrightarrow \pi$ hossza $T[i, l] + C[l, j]$.

Lépésszám: Egy érték $(**)$ szerinti számítása $n-1$ összeadás és ugyanennyi összehasonlítás (minimumkeresés n elem közül)
$\Longrightarrow O(n^3)$ műveletet
Java animáció: Bellman-Ford algoritmus

Floyd módszere

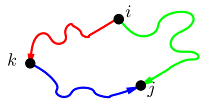
Feladat. *Miként lehet egy irányított gráfban az összes pontpár távolságát meghatározni?*

≥ 0 élsúlyok $\Longrightarrow$ ha a Dijkstra-algoritmust minden csúcsra mint forrásra lefuttatjuk $\Longrightarrow nO(n^2) = O(n^3)$

Van olyan, ami nem lassabb és működik negatív élsúlyokra is, ha nincs negatív összsúlyú kör.
Feladat. *Adott egy $G = (V, E)$ irányított gráf, és egy $c : E \to \mathbb{R}$ súlyfüggvény úgy, hogy a gráfban nincs negatív összsúlyú irányított kör. Határozzuk meg az összes $v, w \in V$ rendezett pontpárra a $d(v, w)$ távolságot.*

A G gráf a C adjacencia-mátrixával adott.
Egy szintén $n \times n$-es F mátrixot fogunk használni a számításhoz.
Kezdetben $F[i,j] := C[i,j]$.
$\Longrightarrow$ ciklus $\Longrightarrow$ k-adik lefutása után $F[i,j]$ azon $i \rightsquigarrow j$ utak legrövidebbjeinek a hosszát tartalmazza, amelyek közbülső pontjai k-nál nem nagyobb sorszámúak

Az új $F_k[i,j]$ értékeket kiszámíthatjuk ha ismerjük $F_{k-1}[i,j]$-t $\forall i,j$-re
Egy legrövidebb $i \rightsquigarrow j$ út, melyen a közbülső pontok sorszáma legfeljebb k, vagy tartalmazza a k csúcsot vagy nem.
— igen $\Longrightarrow F[i,j] := F[i,k] + F[k,j]$



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

FLOYD algoritmusa

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

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

$k = n \Longrightarrow F[i,j] = d(i,j)$

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

Java animáció: Floyd algoritmus

A legrövidebb utak nyomon-követése

Menet közben karbantartunk egy $n \times n$-es$P$ tömböt.
Kezdetben $P[i,j] := 0$.
Ha egy $F[i,j]$ értéket megváltoztatunk, mert találtunk egy k-n átmenő rövidebb utat, akkor $P[i,j] := k$.
$\Longrightarrow$ Végül $P[i,j]$ egy legrövidebb $i \rightsquigarrow j$ út „középső" csúcsát fogja tartalmazni.
$i \rightsquigarrow j$ út összeállítása rekurzív $\Longrightarrow$
procedure legrövidebb út(i, j:csúcs);
var k:csúcs;
begin
  $k := P[i,j]$;
  if $k = 0$ **then return**;
  legrövidebb út(i, k);
  kiír(k);
  legrövidebb út(k, j)
end;

Tranzitív lezárás

Bemenet: $G = (V, E)$ irányított gráf.
Csak arra vagyunk kíváncsiak, hogy mely pontok között vezet irányított út.

Floyd $\Longrightarrow$ ha a végén $F[i,j] \neq \infty$, akkor van út, különben nincs.

Kicsit egyszerűbb korábbi algoritmus: S. Warshall
Definíció. *[tranzitív lezárt] Legyen $G = (V, E)$ egy irányított gráf, A az adjacencia-mátrixa. Legyen továbbá T a következő $n \times n$-es mátrix:*

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

Ekkor a T mátrixot – illetve az általa meghatározott gráfot – az A mátrix – illetve az általa meghatározott G gráf – tranzitív lezártjának hívjuk.
Feladat. *Adott a (Boole-mátrixként értelmezett) A adjacencia-mátrixával a $G = (V, E)$ irányított gráf. Adjuk meg a G tranzitív lezártját.*

Warhall algoritmus

(1) ciklusban a kezdőértékek beállítása helyett

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

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

$$(+) \qquad T[i,j] := T[i,j] \vee (T[i,k] \wedge T[k,j]).$$

Bizonyítás ugyanúgy.

Lépésszám: $O(n^3)$

Alkalmazás Floyd módszerére

A súlyozott élű G irányított gráf v csúcsára legyen

$$e(v) = \max\{d(w,v) : \quad w \in V\}.$$

A v csúcs *centruma* G-nek, ha $e(v)$ minimális az összes $v \in V$ között.
Feladat. *Keressük meg a G gráf centrumát.*

Algoritmus centrum keresésére:
(1) Először Floyd módszerével kiszámítjuk a G-beli pontpárok közötti távolságokat. $\Longrightarrow O(n^3)$ művelet
(2) Az előző lépésben kapott F mátrix minden oszlopában meghatározzuk a maximális értéket ($\Longrightarrow e(v)$). $\Longrightarrow$ n-szer keresünk maximumot n elem közül $\Longrightarrow$ összesen $O(n^2)$.
(3) Végül megkeressük az n darab $e(v)$ érték minimumát. $\Longrightarrow O(n)$.

Mélységi bejárás

Gráf bejárás $\Longrightarrow$ minden pontot felsorolunk, bejárunk

Mélységi bejárás (depth-first-search, DFS), Szélességi bejárás (breadth-first-search, BFS)

Pl. lámpagyújtogató algoritmus

Mélységi keresés

Mohó menetelés, addig megyünk előre, amíg tudunk, csak aztán fordulunk vissza.

Java animáció: Mélységi keresés

$G = (V, E)$ egy irányított gráf, ahol $V = \{1, \ldots, n\}$.
$L[v]$ a v csúcs éllistája ($1 \leq v \leq n$).
bejárva$[1:n]$ logikai vektor $\Longrightarrow$ jártunk-e már ott

Mélységi keresés algoritmusa

procedure bejár (∗ elvégzi a G irányított gráf mélységi bejárását ∗)
 begin
  for $v := 1$ **to** n **do**
   bejárva$[v] := hamis$;
  for $v := 1$ **to** n **do**
   if bejárva$[v] = hamis$ **then**
    mb(v)
 end

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

Lépésszám: $O(n+e)$Mélységi számok és befejezési számok

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

Előző algoritmus kis módosítással:

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

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

Java animáció: Mélységi és befejezési számok

Mélységi feszítő erdő

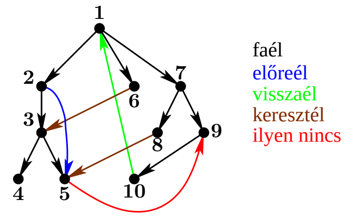
Definíció. *[faél] A $G = (V, E)$ irányított gráf $v \to w$ éle faél (az adott mélységi bejárásra vonatkozóan), ha megvizsgálásakor a (2) sorban a (bejárva$[w] = hamis$) feltétel teljesül.*

Jelölje T azt a gráfot, amelynek csúcshalmaza V , élei pedig a faélek.
$\Longrightarrow$ ez erdő

Definíció. *[mélységi feszítő erdő, feszítőfa] Az előbb meghatározott T gráfot a G gráf egy mélységi feszítő erdejének nevezzük. Ha T csak egy komponensből áll, akkor mélységi feszítőfáról beszélünk.*

Definíció. *[élek osztályozása] Tekintsük a G irányított gráf egy mélységi bejárását és a kapott T mélységi feszítő erdőt. (Ezen bejárás szerint) G egy $x \to y$ éle*

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



faél, előre él: kisebb mélységi számúból nagyobb mélységi számúba mutat
visszaél, keresztél: nagyobb mélységi számúból kisebb mélységi számúba mutat

visszaél: kisebb befejezési számúból nagyobb befejezési számúba mutat

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

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

9. előadás

Algoritmuselmélet 9. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Március 18.

Mélységi feszítőerdő

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

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

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

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

Bizonyítás: T_x éppen azokból a pontokból áll, amelyek x-ből faélek mentén elérhetők. $\Longrightarrow$ faélekre mindig nő a mélységi szám $\Longrightarrow T_x \subseteq S_x$

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

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

Bizonyítás: mszám$[x] = 1 \Longrightarrow S_x = V \Longrightarrow T_x = V$

Irányított körmentes gráfok

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

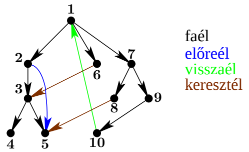
Directed Acyclic Graph

Alkalmazásai például:

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

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

DAG



Ha a gráf egy mélységi bejárása során találunk visszaélet akkor a gráf nyilván tartalmaz irányított kört, azaz nem DAG.
Tétel. *Legyen $G = (V, E)$ egy irányított gráf. Ha G egy DAG, akkor egyetlen mélységi bejárása során sincs visszaél. Fordítva: ha G-nek van olyan mélységi bejárása, amelyre nézve nincs visszaél, akkor G egy DAG.*

Bizonyítás: $\Longrightarrow$ $\surd$
$\Longleftarrow$ tegyük fel, hogy G nem DAG $\Longrightarrow$ van benne irányított kör $\Longrightarrow$ vegyük ennek a legkisebb mélységi számú v csúcsát, a kör előző pontja legyen u
$\Longrightarrow$ mszám$[v]$ < mszám$[u]$ $\Longrightarrow$ vissza- vagy keresztél
de u elérhető v-ből irányított úton ; (részfa lemma) $\Longrightarrow$ u a v leszármazottja
$\Longrightarrow$ visszaél $\surd$

DAG topologikus rendezése

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

Bizonyítás: $\Rightarrow$**:** Ha G nem DAG, akkor nem lehet topologikus rendezése, mert egy irányított kör csúcsainak nyilván nincs megfelelő sorrendje.

$\Leftarrow$**:** G-ben van olyan csúcs, amibe nem fut be él (forrás)
Indukció pontszámra $\Longrightarrow$ hagyjunk el egy forrást, ez legyen az első pont
$\Longrightarrow$ a többit az indukció miatt rendezhető $w_1, \dots, w_{n-1}$
$\Longrightarrow x, w_1, \dots, w_{n-1}$ $\surd$

Topologikus rendezés mélységi kereséssel

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

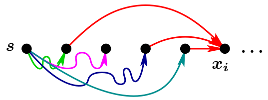
Bizonyítás: Azt kell belátnunk, hogy ha $w_i \to w_j$ éle G-nek, akkor $i > j$.
Ha volna olyan $w_i \to w_j$, amire j = bszám$[w_j]$ > bszám$[w_i]$ = i, akkor az csak visszaél lehetne. $\lightning$

Lépésszám: $O(n + e)$

Legrövidebb utak DAG-ban

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

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



Ezt sorban elvégezzük minden i-re.

Lépésszám: $O(n + e)$

Leghosszabb utak DAG-ban

Leghosszabb út $\Longrightarrow$ egyszerű út
Általában nehéz, nem ismert rá gyors algoritmus.

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

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

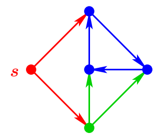
$$l(s, x_i) = \max_{(x_j, x_i) \in E} \{l(s, x_j) + c(x_j, x_i)\}.$$

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

Alkalmazás: PERT

Erősen összefüggő (erős) komponensek

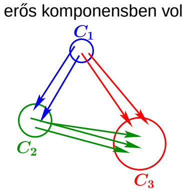
Definíció. *Egy $G = (V, E)$ irányított gráf erősen összefüggő, ha bármely $u, v \in V$ pontpárra létezik $u \rightsquigarrow v$ irányított út.*

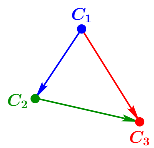


Definíció. *Legyen $G = (V, E)$ egy irányított gráf. Bevezetünk egy relációt V-n: $u, v \in V$-re legyen $u \approx v$, ha G-ben léteznek $u \rightsquigarrow v$ és $v \rightsquigarrow u$ irányított utak.*

Ez ekvivalenciareláció $\Longrightarrow$
Definíció. *A $\approx$ reláció ekvivalenciaosztályait a G erősen összefüggő (erős) komponenseinek nevezzük.*

Tétel. *Egy irányított gráf két erős komponense között az élek csak egy irányba mehetnek.*
Bizonyítás: Ha menne él a $C_1 \to C_2$ és $C_2 \to C_1$-be is, akkor C_1 és C_2 ugyanabban az erős komponensben volna.





Definíció. *Legyen $G = (V, E)$ irányított gráf. G redukált gráfja egy olyan gráf, melynek pontjai a G erős komponensei; a C_1, C_2 komponensek között akkor van $C_1 \to C_2$ él, ha G-ben a C_1 komponens valamely pontjából vezet él a C_2 komponensbe.*

A redukált gráf mindig DAG lesz. $\Longleftarrow$ $C_1 \to C_2 \to \dots \to C_k \to C_1$ irányított kör a redukált gráfban azt jelentené, hogy $C_1 \cup C_2 \cup \dots \cup C_k$ a G ugyanazon erős komponensében van.

Erősen összefüggő komponensek meghatározása

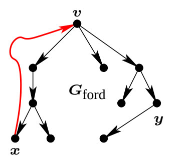
(1) Mélységi bejárással végigmegyünk G-n, közben minden pontnak sorszámot adunk: a befejezési számát

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

(3) Bejárjuk a G_{ford} gráfot mélységi bejárással, a legnagyobb sorszámú csúccsal kezdve (az (1)-beli befejezési számozás szerint). Új gyökérpont választásakor mindig a legnagyobb sorszámú csúcsot vesszük a maradékból.

Tétel. *A (3) pontban kapott fák lesznek G erős komponensei, azaz G-ben $x \approx y$ pontosan akkor igaz, ha x és y egy fában vannak.*

Bizonyítás: $\Rightarrow$: Azt kell belátni, hogy egy erős komponens pontjai egy fába kerülnek
Legyen K egy erős komponens, és legyen x a K legkisebb mélységi számú pontja.
$\Longrightarrow K \subseteq S_x$ $\Longleftarrow$ részfa-lemma $\surd$



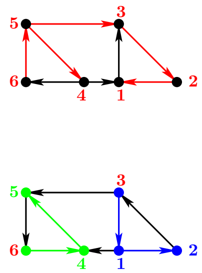
$\Leftarrow$**:** Tegyük fel, hogy x és y egy fában vannak a (3) pont szerinti mélységi bejárás után. Azt kell belátnunk, hogy ekkor $x \approx y$ a G gráfban, azaz x és y egymásból irányított úton elérhetők.
Legyen a v csúcs a gyökere annak a fának, mely x-et és y-t is tartalmazza.
$\Longrightarrow G_{\text{ford}}$ gráfban van $v \rightsquigarrow x$ irányított út, $\Longrightarrow G$ gráfban van egy L irányított út $x \rightsquigarrow v$-be.
Legyen x' az L-nek az a pontja, amelynek az első bejárás szerinti mélységi száma a legkisebb.
részfa-lemma $\Longrightarrow L$-nek az $x' \rightsquigarrow v$ darabjában levő csúcsok az (1) bejárásnál x' leszármazottjai lesznek.

Az x' gyökerű részfában x' befejezési száma a legnagyobb $\Longrightarrow v$ nem választhattuk v-t gyökérnek $\Longrightarrow x' = v$.
Az L pontjai közül tehát v-t látogattuk meg legelőször, és v-nek a befejezési száma volt a legnagyobb. $\Longrightarrow$ Így G-ben van $v \rightsquigarrow x$

$\Longrightarrow x \approx v$, hasonlóan $y \approx v$ $\Longrightarrow x \approx y$ $\surd$

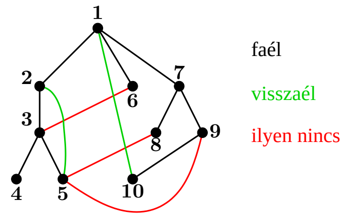
Lépésszám: $O(n+e) + O(e) + O(n+e) = O(n+e)$

Példa



Irányítatlan gráfok mélységi bejárása

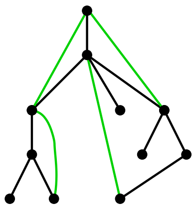
Mélységi keresés ugyanígy.
Mélységi feszítő erdő komponensei $\Longrightarrow$ összefüggő komponensek



faél $\Longleftrightarrow$ faél
előreél, visszaél $\Longleftrightarrow$ visszaél
keresztél $\Longrightarrow$ nem létezik

Artikulációs pont keresése

Definíció. *Legyen $G = (V, E)$ összefüggő irányítatlan gráf. A $v \in V$ csúcs artikulációs (elvágó) pontja G-nek, ha v és a rá illeszkedő élek elhagyásával a gráf több komponensre esik szét.*



A fa gyökere pontosan akkor artikulációs pontja a gráfnak, ha egynél több fia van
Ha elhagyunk egy v csúcsot $\Longrightarrow$ A visszaélek csak úgy tarthatják egybe a részfákat, ha a v alatti nem üres részfák mindegyikéből megy visszaél a v feletti feszítőfadarabba.
Kiszámítjuk a fel$[v]$ értéket. Ez megadja a v csúcshoz annak a „feszítőfában legmagasabban levő" w csúcsnak a mélységi számát, amelyhez el tudunk jutni v-ből úgy, hogy „lefelé" megyünk faélen, aztán egy visszaélen „felmegyünk" w-be.
A v csúcs tehát artikulációs pont $\Longleftrightarrow$ van olyan w fia, melyre fel$[w] \geq$ mszám$[v]$.

Algoritmus

1. Végezzük el a gráf mélységi bejárását, és határozzuk meg a csúcsok mélységi számát

2. Számítsuk ki minden v csúcsra a fel$[v]$ értéket $\Longrightarrow$ Járjuk be a feszítőfát a befejezési számok szerinti sorrendben, és ebben a sorrendben töltsük ki a fel[ ] tömböt.

$$\text{fel}[v] = \min \left\{ \begin{array}{l} \text{mszám}[v], \\ \min\{\text{mszám}[z],\ \text{ahol } v \to z \text{ visszaél}\}, \\ \min\{\text{fel}[y],\ \text{ahol } y \text{ fia } v\text{-nek}\} \end{array} \right\}$$

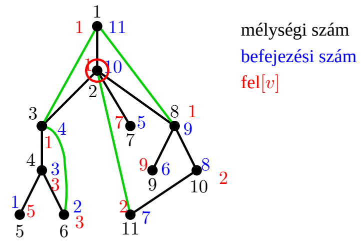
3. Artikulációs pontok megkeresése: a feszítőfát bejárva a csúcsokról ellenőrizzük, hogy elvágó pontok-e.

(a) a gyökér pontosan akkor artikulációs pont, ha legalább 2 fia van a fában.
(b) a gyökértől különböző v csúcs akkor és csak akkor artikulációs pont, ha van v-nek olyan y fia, hogy fel$[y] \geq$ mszám$[v]$.

Lépésszám: $O(n+e)$

Példa

$$\text{fel}[v] = \min \left\{ \begin{array}{l} \text{mszám}[v], \\ \min\{\text{mszám}[z],\ \text{ahol } v \to z \text{ visszaél}\}, \\ \min\{\text{fel}[y],\ \text{ahol } y \text{ fia } v\text{-nek}\} \end{array} \right\}$$



10. előadás

Algoritmuselmélet 10. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Március 25.

Szélességi bejárás

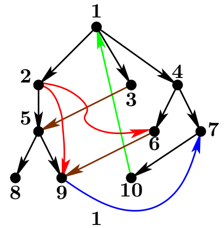
BFS: Breadth First Search
Meglátogatjuk az első csúcsot, majd ennek a csúcsnak az összes szomszédját. Aztán ezen szomszédok összes olyan szomszédját, hol még nem jártunk, és így tovább.
megvalósítás $\Longrightarrow$ *sor* (FIFO-lista)
Berakjuk be az épp meglátogatott csúcsot, hogy majd a megfelelő időben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

```
procedure bejár (* elvégzi a G irányított gráf szélességi bejárását *)
    begin
        for v := 1 to n do
            bejárva[v] := hamis;
        for v := 1 to n do
            if bejárva[v] = hamis then
                szb(v)
    end
```

```
procedure szb (v: csúcs)
    var
        Q: csúcsokból álló sor;
        x, y: csúcsok;
    begin
        bejárva[v] := igaz;
        sorba(v, Q);
        while Q nem üres do begin
            x := első(Q);
            for minden x → y ∈ E élre do
                if bejárva[y] = hamis then begin
                    bejárva[y] := igaz;
(*)                 sorba(y, Q)
                end
        end
    end
```

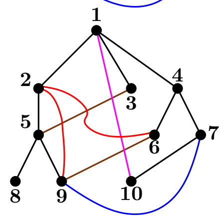
JAVA animáció: BFS

Faél:
megvizsgálásukkor még be nem járt pontba mutatnak



faél
ilyen nincs
visszaél
keresztél
keresztél

Irányítatlan esetben csak faél és keresztél lehet.
Lépésszám: $O(n+e)$



faél
ilyen nincs
ilyen nincs
keresztél
keresztél

Legrövidebb utak súlyozatlan gráfokban

Ha minden él hossza egy $\Longrightarrow$ út hossza = élek száma
Szélességi kereséssel $\Longrightarrow$ Jelentse $D[v]$ a v csúcsnak az s-től való távolságát az s-gyökerű szélességi fában.

Legyen kezdetben $D[s] := 0$
az *szb* eljárásba tegyük be a $D[y] := D[x] + 1$; utasítást, miután elértük y-t.

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

Tétel. *1. $D[x_1] \le D[x_2] \le \ldots \le D[x_n]$.*

Bizonyítás: A csúcsok az $s = x_1, x_2, \ldots, x_n$ sorrendben kerülnek bele a Q sorba $\Longrightarrow$ ebben a sorrendben is kerülnek ki a sorból
ha $x \ne s$ csúcs előbb van mint y $\Longrightarrow$ $apa(x)$ nem lehet később, mint $apa(y)$, hiszen ha előbb lenne, y-hoz előbb eljutottunk volna
Indukció $\Longrightarrow$ Gyökérre $D[s] = 0$, fiaira mind nagyobb $\checkmark$
$D[x_i] = D[apa(x_i)] + 1$ és $D[x_{i+1}] = D[apa(x_{i+1})] + 1$ $\Longrightarrow$
Ha a két apa különböző
$\Longrightarrow D[apa(x_i)] \le D[apa(x_{i+1})] \Longrightarrow D[x_i] \le D[x_{i+1}]$
Ha pedig az apák megegyeznek $\Longrightarrow D[x_i] = D[x_{i+1}]$
Tétel. *2. Ha $x \to y$ éle G-nek, akkor $D[y] \le D[x] + 1$*

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

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

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

JAVA animáció: Legrövidebb út

Minimális költségű feszítőfák

Most irányítatlan gráfokkal foglalkozunk
kör és út $\Longrightarrow$ valóban egyszerű

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

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

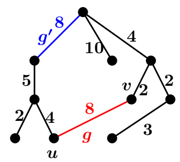
Például: Villamosvezetékek kiépítése

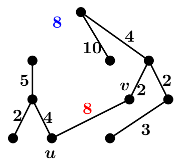
Fák tulajdonságai

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

Bizonyítás: 1–4 volt már BSZ-ben $\checkmark$

Bizonyítás: 5.





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

A piros-kék algoritmus

Sorra nézzük G éleit: bizonyosakat beveszünk a minimális feszítőfába, másokat pedig eldobunk.
$\Longrightarrow$ színezzük a G éleit:
a *kék* élek belekerülnek a végeredményt jelentő minimális feszítőfába,
a *pirosak* pedig nem
$\Longrightarrow$ Úgy színezünk, hogy az eddig kialakult (részleges) színezés mindig *takaros* legyen.
Definíció. (takaros színezés) *Tekintsük a súlyozott élű G gráf éleinek egy részleges színezését, melynél bármely él piros, kék vagy színtelen lehet. Ez a színezés takaros, ha van G-nek olyan minimális költségű feszítőfája, ami az összes kék élet tartalmazza, és egyetlen piros élet sem tartalmaz.*

KÉK SZABÁLY: Válasszunk ki egy olyan $\emptyset \neq X \subset V$ csúcshalmazt, amiből nem vezet ki kék él. Ezután egy legkisebb súlyú X-ből kimenő színezetlen élet fessünk kékre.

PIROS SZABÁLY: Válasszunk G-ben egy olyan egyszerű kört, amiben nincs piros él. A kör egyik legnagyobb súlyú színtelen élét fessük pirosra.

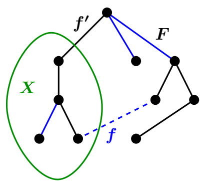
Kezdetben G-nek nincs színes éle. $\Longrightarrow$ a két szabályt tetszőleges sorrendben és helyeken alkalmazzuk, amíg csak lehetséges.
$\Longrightarrow$ piros-kék algoritmus

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

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

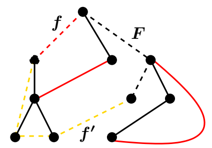
Két eset van aszerint, hogy melyik szabályt használjuk:

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



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

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



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

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

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

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

Prim, Kruskal és Borůvka módszerei

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

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

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

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

Prim módszere

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

```
procedure Prim (G: gráf; var F: élek halmaza);
   var
      U: csúcsok halmaza;
      u, v: csúcsok;
   begin
      F := ∅;
      U := {1};
      while U ≠ V do begin
(*)      legyen (u, v) egy legkisebb súlyú olyan él,
         melyre u ∈ U és v ∈ V \ U;
         F := F ∪ {(u, v)};
         U := U ∪ {v}
      end
   end
```

Jól működik, mert piros-kék algoritmus.
JAVA animáció: Prim módszere

Naiv implementáció

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

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

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

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

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

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

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

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

Lépésszám: Egy él színezés $O(n) \Longrightarrow O(n^2)$
JAVA animáció: Prim módszere

Kupacos-éllistás implementáció

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

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

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

11. előadás

Algoritmuselmélet 11. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Március 26.

Kruskal módszere

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

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

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

Bizonyítás: ez is piros-kék algoritmus

JAVA animáció: Kruskal

Implementáció:

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

Hogyan döntjük el, hogy a kiválasztott él kört alkot-e az eddigi kiválasztottakkal?

$\Longrightarrow$ Tartsuk nyilván az aktuálisan egy kék fába tartozó csúcsokat mint halmazokat.
Kell: UNIÓ, HOLVAN

Az UNIÓ-HOLVAN adatszerkezet

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

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

Kruskalban:

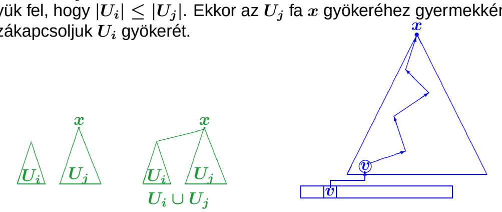
annak megvizsgálása, hogy milyen színű legyen az új (u, v) él
$\Longrightarrow$ Ha HOLVAN$(u) \neq$ HOLVAN(v), akkor kék, különben piros.

Új él színezése $\Longrightarrow$ UNIÓ(HOLVAN(u), HOLVAN(v))

Implementáció fákkal

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

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



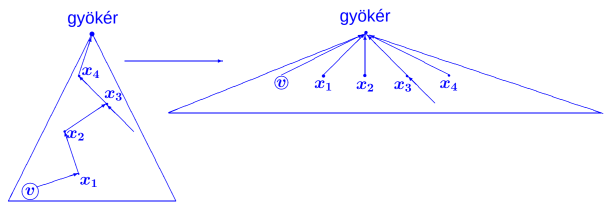
- **HOLVAN:** A $v \in S$ elemet tartalmazó részhalmaz nevét, azaz a megfelelő fa gyökerét a szülőkhöz menő mutatók végigkövetésével találhatjuk meg.

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

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

A HOLVAN gyorsítása: útösszenyomás

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



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

A korlátban szereplő $\alpha : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ függvény az *inverz Ackermann-függvény*.

$\alpha(m)$ a végtelenhez tart, ha $m \rightarrow \infty$, de nagyon lassan, lassabban mint a logaritmus k-szori önmagába helyettesítésével adódó $\log \log \cdots \log m$ függvény ($k \in \mathbb{Z}^+$ tetszőleges).

Pl. $\alpha(m) \leq 4$, ha $m < 2^{65536}$
A normális méretű feladatoknál tehát $\alpha(m)$ állandónak (≤ 4) tekinthető.
Tétel. *Ha az élek rendezésével kapcsolatos teendők $O(e)$ időben megoldhatók, akkor a Kruskal-algoritmus $O(e\alpha(e))$ időben megvalósítható.*

Manipuláció a súlyokkal $\Longrightarrow$ Yao (1975): $O(e \log \log n)$
Véletlen módszerek $\Longrightarrow$ D. R. Karger, P. N. Klein, R. E. Tarjan, (1994): várhatóan $O(e)$

On-line változatban is működik a piros-kék algoritmus: színezzük az új élet élet kékre; ha emiatt kialakul egy kék kör, akkor abból töröljünk egy maximális súlyú élet.

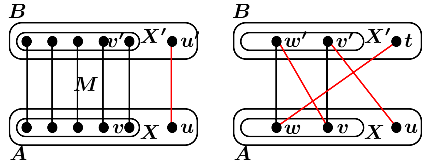
JAVA animáció: Kruskal

Maximális párosítás páros gráfokban

Definíció. *A $G = (V, E)$ gráfot* párosnak *nevezzük, ha V csúcshalmaza felosztható két diszjunkt részre – V_1-re és V_2-re – úgy, hogy minden él ezen két halmaz között fut, vagyis $(x, y) \in E$ esetén $x \in V_1$ és $y \in V_2$ vagy fordítva.*
Definíció. *Legyen $G = (V, E)$ egy tetszőleges gráf. Az E élhalmaz $E' \subseteq E$ részhalmaza G egy* párosítása, *ha a $G' = (V, E')$ gráfban minden pont foka legfeljebb egy.*
Definíció. *A G gráf egy E' párosítása* maximális, *ha G minden E'' párosítására $|E''| \leq |E'|$.*

A probléma: Adott egy $G = (V_1, V_2; E)$ páros gráf. Határozzuk meg G egy maximális párosítását.
Definíció. *Legyen G egy tetszőleges gráf, és E' a G egy párosítása. Egy G-beli utat* E'-alternáló útnak *hívunk, ha felváltva tartalmaz párosított és nem párosított éleket.*
Definíció. *Legyen E' a $G = (V, E)$ gráf egy párosítása. Ekkor egy olyan E'-alternáló út, melynek mindkét végpontja párosítatlan,* E'-re nézve javító út, *vagy röviden* E'-javító út.

Tétel. *Legyen $G = (V, E)$ egy tetszőleges gráf és E' egy párosítása. Ha E'-re nézve nincs javító út G-ben, akkor E' a G egy maximális párosítása.*



Hogyan keressünk javító utat?

Javító út keresése alternáló erdő építésével

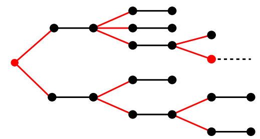
$0.$ szint: V_1 azon pontjai, melyeket E' nem fed le, vagyis a párosítatlan pontok.

$\vdots$

$2k-1.$ szint: V_2 azon még fel nem vett pontjai, melyek egy párosítatlan, azaz egy $E \setminus E'$-beli éllel elérhetők egy $2k-2.$ szintbeli pontból; ezen éllel együtt.

$2k.$ szint: V_1 azon még fel nem vett pontjai, melyek egy párosított, azaz egy E'-beli éllel elérhetők egy $2k-1.$ szintbeli pontból; ezen éllel együtt.

$\vdots$



Tétel. *A $G = (V_1, V_2; E)$ páros gráfban akkor és csak akkor van az E' párosításra nézve javító út, ha az E'-hez tartozó alternáló erdőben valamelyik páratlan szinten megjelenik egy párosítatlan pont.*

Bizonyítás: Ha találtunk páratlan szinten párosítatlan utat, akkor a gyökér felé vezető út javító út $\surd$

Fordítva:
Megmutatjuk, hogy a V_1 párosítatlan csúcsaiból (ezek vannak az erdőben a nulladik szinten) alternáló úton elérhető pontok mindegyikét beválasztjuk valamikor az alternáló erdőbe.

Tegyük fel, hogy $v_0, v_1, \dots, v_k$ egy alternáló út, és $v_0 \in V_1$ egy párosítatlan csúcs; i szerinti indukcióval megmutatjuk, hogy v_i bekerül az erdőbe.
Ha $i = 0$ $\surd$

Az út v_{2j} csúcsa V_1-ben van és a (v_{2j}, v_{2j+1}) él párosítatlan. $\Longrightarrow$ ha v_{2j}-t beválasztottuk, akkor v_{2j+1} is bekerül

Az út v_{2j-1} csúcsa V_2-ben van és $(v_{2j-1}, v_{2j}) \in E'$. Így v_{2j-1} után v_{2j} is sorra kerül, ha korábban ez még nem történt meg.

Tegyük fel, hogy G-ben a v és w csúcsok egy javító út végpontjai.
$v \in V_1$ párosítatlan $\Longrightarrow w \in V_2$
w elérhető alternáló úton v-ből $\Longrightarrow$ valamikor beválasztjuk
De V_2-beli csúcsokat csak páratlan szintekre veszünk fel $\Longrightarrow w$ is itt lesz

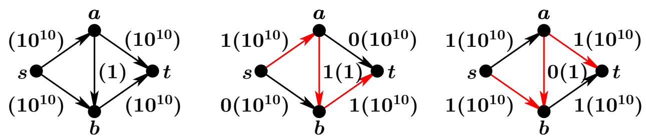
Lépésszám: Alternáló erdő építése: $O(e)$, össze lépésszám: $O(ne)$

Karp (1973): $O(e\sqrt{n})$

Maximális folyamok hálózatokban

JAVA animáció: Ford-Fulkerson algoritmus

Ha minden kapacitás egész és a maximális folyam értéke f, akkor legfeljebb f javítással megkapjuk a maximális folyamot.



Ha az élkapacitások racionális számok $\Longrightarrow$ véges lépés

Ha irracionális kapacitásokat is megengedünk $\Longrightarrow$ lehet, hogy nem ér véget véges sok lépésben, sőt lehet, hogy nem is jó értékhez konvergál

Edmonds–Karp algoritmus

A folyam növelésére mindig a legrövidebb – vagyis a legkevesebb élből álló – növelő utak egyikét válasszuk.

Tekintsük a G_f javító gráfot; legyen benne π egy legrövidebb növelő út.
Ennek hosszát (élszámát) jelölje l.
Szélességi kereséssel osszuk szintekre $\Longrightarrow D[v]$

(1) Egy él legfeljebb egy réteggel mehet előre.
A G_f egy $x \to y$ élét nevezzük *vastagnak*, ha $D[y] = D[x] + 1$.

(2) Az l hosszúságú $s \rightsquigarrow t$ utak csupa vastag élből állnak, és nincs l-nél rövidebb $s \rightsquigarrow t$ út.
$\Longleftarrow$ Legrövidebb útnak muszáj mindig feljebb menni

Mi történik, ha javítunk π mentén?
Legalább egy él telítődik és eltűnik a javító gráfból.
Legfeljebb l darab új él jelenik meg a $G_{f'}$-ben (π élei ellenkező irányítással, ha eddig még 0 folyt át rajtuk).
$\Longrightarrow$ *a következő növelő út sem lehet rövidebb l-nél.*

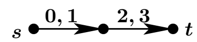
Hányszor adódhat egymás után l hosszú növelő út?
Minden javítás után eggyel kevesebb vastag él lesz (legalább egy kritikus él törlődik).
Addig lesz l élből álló növelő út, amíg marad vastag élekből álló $s \rightsquigarrow t$ út.

Tétel. *Az Edmonds–Karp-heurisztika szerinti növelésnél a növelő utak hosszai nem csökkenő sorozatot alkotnak. Ebben a sorozatban egy adott úthosszúság legfeljebb e-szer fordulhat elő. Következésképpen legfeljebb $e(n-1)$ növelés lehetséges. A heurisztika alkalmazásával $O(e^2 n)$ elemi lépésben kapunk maximális folyamot.*

Bonyolultabb algoritmusok
Dinic: $O(en^2)$
Goldberg, Tarjan: $O(en \log(n^2/e))$

Hálózatok alsó korlátokkal

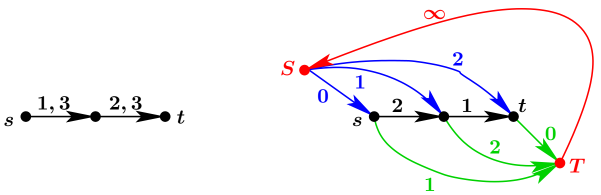
Tegyük fel, hogy a $c(u, v)$ kapacitások mellett (felső korlát) *alsó korlátok* is vannak az $f(u, v)$ mennyiségekre. $\Longrightarrow k(u, v) \leq f(u, v)$ is teljesüljön a G minden $u \to v$ éléře.
$\Longrightarrow (G, s, t, c, k)$
Van-e egyátalán ilyen folyam?



Belátjuk, hogy ez a hagyományos folyamproblémára visszavezethető.

$\mathcal{H} = (G, s, t, c, k) \to \mathcal{H}'$

- Új forrás: S, új nyelő: T
- Régi éleken új kapacitás: $c'(u, v) := c(u, v) - k(u, v)$
- 2 új él minden pontra: $S \to v$ és $v \to T$
  $c'(S, v) := \sum_{(u,v)\in E} k(u, v)$ és $c'(v, T) := \sum_{(v,w)\in E} k(v, w)$
- Új $T \to S$ él ∞ kapacitással



Tétel. *A $\mathcal{H} = (G, s, t, c, k)$ hálózatban akkor és csak akkor létezik folyam, ha a $\mathcal{H}'$ hálózat (S-ből T-be menő) maximális folyamának az értéke $\sum_{(u,v)\in E} k(u, v)$.*

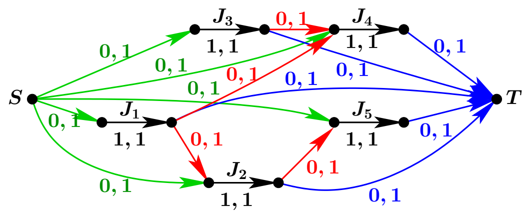
Ha a $T \to S$ él kapacitást d-nek választjuk $\Longrightarrow$ ugyanígy megkaphatjuk, hogy van-e legfeljebb d értékű folyam
$\Longrightarrow$ algoritmus alsó korlátos folyamokra

Egy ütemezési feladat

Tegyük fel, hogy egy légitársaság a $J_1, J_2, \ldots, J_m$ járatokat szeretné üzemeltetni, és d darab azonos típusú repülőgépe van erre a célra.
Minden J_i, J_j járatpárra ismert, hogy van-e elég idő arra, hogy a J_i teljesítése után egy gép felkészüljön a J_j repülésére.
Ha J_i, J_j-re a válasz igenlő, akkor azt mondjuk, hogy J_j *követheti* J_i-t.

Gráf:

Egy J_i légijáratnak $\Longrightarrow$ két csúcs, i és i' és egy $i \to i'$ él
Ha J_j követheti J_i-t, akkor vezessünk irányított élet i'-ből j-be.
Vegyünk még fel egy s forrást és egy t nyelőt, és adjuk a hálózathoz az $s \to i$ és $i' \to t$ éleket ($1 \le i \le m$).
Az összes él kapacitása legyen 1. Az $i \to i'$ alakú élek alsó korlátja legyen 1, a többi élé pedig 0.
Tétel. *A $J_1, J_2, \ldots, J_m$ járatok akkor és csak akkor teljesíthetők legfeljebb d géppel, ha a hálózathoz van olyan g folyam, amelyre $|g| \le d$.*



Hirdetmények

Április 1. $\Longrightarrow$ Húsvét hétfő $\Longrightarrow$ elmarad az előadás

Április 8. $\Longrightarrow$ Előadás helyett konzultáció

ZH Április 8. 16:15

A–He	CH. max.
Hi–Ka	I.B. 27
Ke–M	I.B. 28
N-Se	E.I.B.
Si-Z	St. nagy

12. előadás

Algoritmuselmélet 12. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Április 9.

Turing-gépek

Az algoritmus fogalmának pontosabb meghatározása.
Számítógép elméleti, egyszerűsített modellje.

Alan Turing 1912-1954
Definíció. *Többszalagos Turing-gép (TG), $k \ge 1$ egész szám*

- *k db szalag cellákra osztva, cellákba jelek, a szalag egyik (mindkét) irányban végtelen*
- *minden szalaghoz tartozik egy fej, ami jobbra és balra is lépegethet a szalagon cellánként*
- *véges vezérlő, ennek véges sok állapota van*
- *Lépés: függ a belső állapottól és a fej alatti jelektől*
  - ⋆ *a gép megáll*
  - ⋆ *átmegy új állapotba, ír valamit minden fej alatti cellába, minden fej lép vagy jobbra, vagy balra egyet vagy helyben marad*

Definíció. *Egy k-szalagos Turing-gép egy hetessel jellemezhető:*

$$M = (Q, T, ü, I, q_0, F, \delta),$$

ahol

Q : *egy véges halmaz, az M gép belső állapotainak halmaza.*

T : *egy véges halmaz, a szalagjelek halmaza.*

ü : *egy kitüntetett szalagjel, az üresjel. A bemenetként felírt jeleken és a gép számítása során kiírt jeleken kívül minden szalagcellában ü van.*

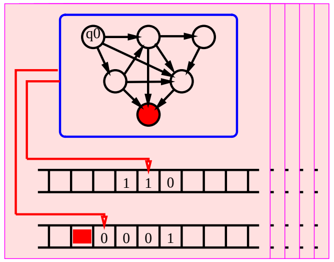
I : *$I \subseteq T \setminus \{ü\}$ az input jelek vagy bemenő jelek halmaza, más szóval az input abc. Az üresjel nem lehet input jel.*

q_0 : *$q_0 \in Q$ a kezdő állapot.*

F : $F \subseteq Q$ *az* elfogadó állapotok *halmaza.*
Elfogadó állapotban áll meg $\Longrightarrow$ *IGEN*
Nem elfogadó állapotban áll meg $\Longrightarrow$ *NEM*
$\Longrightarrow$ *a* $Q \setminus F$*-be tartozó állapotokat* elutasító állapotoknak *is nevezik.*

δ : *A* $\delta : Q \times T^k \longrightarrow Q \times (T \times \{jobb, bal, helyben\})^k$ *egy parciálisan értelmezett függvény, a gép* átmenetfüggvénye*.*
Az átmenetfüggvény tekinthető a gép programjának.
Ha a $\delta(q; a_1, a_2, \ldots, a_k)$ *nincs értelmezve,* $\Longrightarrow$ megállás

Példa



animáció: Turing gép

Turing-gép működése

- Kezdetben a q_0 állapotban van, az $s \in I^*$ bemenet az első szalag elejére van írva, az összes többi mezőn ü
- A δ függvénynek megfelelően lépeget $\Longrightarrow$ új állapot, írás, fej lépése
- Ha δ nincs értelmezve, akkor megáll (akár elfogadó állapotban van, akár nem)

Két felhasználás:

- Függvények kiszámolása
- Kérdések eldöntése ($0 - 1$ értékű függvény)

Definíció. *Az* M *Turing-gép által* felismert L_M nyelv *azokból az* $s \in I^*$ *szavakból áll, amelyekkel mint bemenetekkel elindítva az* M *megáll, mégpedig elfogadó (azaz* F*-beli) állapotban.*

Ha M az L_M nyelvet ismeri fel, akkor a nyelvbe nem tartozó szavakra vagy megáll elutasító állapotban, vagy végtelen ciklusba kerül.

Definíció. *Legyen* M *egy kitüntetett output szalaggal rendelkező Turing-gép. Az* M *által* kiszámított $f_M : I^* \rightarrow I^*$ parciális függvényt *így értelmezzük:* $f_M(s) = w$*, ha* M *az* $s \in I^*$ *inputtal indulva megáll, és megállás után az output szalagon a* $w \in I^*$ *szó szerepel az üresjelek óceánja előtt.*

Az f_M függvény tehát csak azokra az $s \in I^*$ szavakra értelmezett, amelyekkel mint bemenetekkel az M gép véges sok lépés megtétele után megáll. Mindegy, hogy a megállás elfogadó vagy elutasító állapotban történt-e.

Példa Turing-gépre

$$Q = \{q_0, q_1, q_2, q_v\},\ T = \{0, 1, ü\},\ I = \{0, 1\},\ F = \{q_v\}.$$

$$\begin{aligned} \delta(q_0, 1) &= (q_1, 1, jobb), \quad \delta(q_0, ü) = (q_v, ü, helyben), \\ \delta(q_1, 1) &= (q_2, 1, jobb), \quad \delta(q_2, 1) = (q_0, 1, jobb). \end{aligned}$$

Más párokra δ nincs értelmezve.

Ha 0-t olvas $\Longrightarrow$ elutasít
Ha ü üresjelet olvas és nem q_0-ban van $\Longrightarrow$ elutasít
Ha ü üresjelet olvas és q_0-ban van $\Longrightarrow$ elfogad
Ha 1-et olvas és q_i-ben van $\Longrightarrow$ jobbra lép, és átmegy a q_{i+1} állapotba;
$\Longrightarrow$ M pontosan azokat a szavakat fogadja el, amelyek csupa egyesekből állnak, és a hosszuk osztható hárommal.
$\Longrightarrow$ Az M által felismert L_M nyelv tehát

$$L_M = \{1^n :\ n \text{ hárommal osztható természetes szám}\}.$$

Példa Turing-gépre

$$Q = \{q_0, q_v\},\ T = \{0, 1, ü\},\ I = \{0, 1\},\ F = \{q_v\}$$
$$\delta(q_0, 0) = (q_0, 0, helyben),\ \delta(q_0, 1) = (q_0, 1, jobb),$$
$$\delta(q_0, ü) = (q_v, 1, helyben).$$

Másutt δ nem definiált.

Meghatározzuk az $L_{M'}$ nyelvet és az $f_{M'}$ függvényt is.
Az M' a q_0 belső állapotban maradva lépdel jobbra a szalag mentén, amíg 0 vagy ü jelet nem talál.
0 $\Longrightarrow$ végtelen ciklus
ü $\Longrightarrow$ még egy 1-est ír az input után, majd elfogad
$\Longrightarrow$ $L_{M'} = \{1^n; n \geq 0$ egész$\}$
$\Longrightarrow$ A gép az 1^n bemeneten az 1^{n+1} eredményt adja, vagyis $f_{M'}(1^n) = 1^{n+1}$.

Turing-gép $\Longleftrightarrow$ igazi számítógép
Turing-gép többet tud, mint a véges automata.

A kiszámíthatóság alapfogalmai

Algoritmus: ami Turing-géppel kiszámítható
Definíció. *Az* $L \subseteq I^*$ *nyelvet* rekurzíve felsorolhatónak *nevezzük, ha van olyan* M *Turing-gép, melyre* $L = L_M$*, azaz a gép által felismert nyelv éppen* L*.*
Definíció. *Az* $L \subseteq I^*$ *nyelv* rekurzív*, ha van olyan* M *Turing-gép, melyre* $L = L_M$*, és* M *minden* $s \in I^*$ *szóra megáll.*

$$\mathcal{RE} = \{L \subseteq I^* :\ L \text{ rekurzíve felsorolható}\}.$$
$$\mathcal{R} = \{L \subseteq I^* :\ L \text{ rekurzív}\}.$$

$\Longrightarrow$ $\mathcal{R} \subseteq \mathcal{RE}$
Definíció. *Az* $f : I^* \rightarrow I^*$ *parciális függvény* parciálisan rekurzív *függvény, ha létezik olyan* M *Turing-gép, hogy* $f = f_M$*. Ha ezen túl még* f *minden* $s \in I^*$ *inputra értelmezve van, akkor* f *egy* rekurzív *függvény.*

Tétel. *Van olyan* $L' \subseteq I^*$ *nyelv, amely nem rekurzíve felsorolható.*

Bizonyítás: Egy Turing-gép leírható véges jelsorozattal $\Longrightarrow$ az összes gép számossága megszámlálható $\Longrightarrow$ felsorolható: $M_0, M_1, M_2, \ldots$
$\Longrightarrow$ a rekurzíve felsorolható nyelvek is felsorolhatók: $L_{M_0}, L_{M_1}, L_{M_2}, \ldots$
$\Longrightarrow$ a rekurzíve felsorolható nyelvek halmaza megszámlálható

Belátjuk, hogy az összes nyelv halmaza nem megszámlálható (egyébként kontinuum).

Az I^* elemei, a véges hosszúságú I-beli jelekből képzett szavak is megszámlálhatóak $\Longrightarrow$ felsorolhatóak: $w_0, w_1, w_2, \ldots$
Tegyük fel, hogy az összes nyelvek halmaza megszámlálható, megmutatjuk, hogy van olyan $L' \subseteq I^*$ nyelv, amely nem lehet benne az összes nyelvek $L_{M_0}, L_{M_1}, L_{M_2}, \ldots$ sorozatában.
Az L' nyelvnek a w_i szó pontosan akkor legyen eleme, ha $w_i \notin L_{M_i}$, $i = 1, 2, \ldots$.
$L' \neq L_{M_i}$, hiszen a $w_i \notin L'$ és $w_i \in L_{M_i}$ $\checkmark$

Cantor-féle átlós módszer

	w_0	w_1	$\ldots$	w_i	w_{i+1}	$\ldots$
L_{M_0}	nem	nem	$\ldots$	nem	nem	$\ldots$
L_{M_1}	igen	nem	$\ldots$	nem	nem	$\ldots$
$\vdots$						
L_{M_i}	nem	igen	$\ldots$	igen	nem	$\ldots$
$L_{M_{i+1}}$	igen	nem	$\ldots$	nem	nem	$\ldots$
$\vdots$						
L'	igen	igen	$\ldots$	nem	igen	$\ldots$

Tétel. *Létezik olyan* $f :\ I^* \rightarrow I^*$ *parciális függvény, amely nem parciálisan rekurzív.*

Church–Turing-tézis

A rekurzív nyelveket és a rekurzív függvényeket fogjuk algoritmussal kezelhető nyelveknek és függvényeknek tekinteni.

Church–Turing-tézis: *Ami algoritmussal – azaz véges eljárással – kiszámítható (eldönthető), az Turing értelmében kiszámítható (eldönthető). Nevezetesen:*

- *Egy $f : I^* \to I^*$ parciális függvény kiszámítható $\Leftrightarrow$ f parciálisan rekurzív.*
- *Egy $f : I^* \to I^*$ (teljes) függvény kiszámítható $\Leftrightarrow$ f rekurzív.*
- *Egy $L \subseteq I^*$ nyelvre a nyelvbe tartozás problémája algoritmussal eldönthető $\Leftrightarrow$ L rekurzív.*

Idő- és tárigény

Az M Turing-gép *számolási ideje* az s inputon a megállásáig végrehajtott lépések száma *tárigénye* pedig a felhasznált (olvasott) szalagcellák száma.

A tárigénybe nem feltétlenül számítjuk bele az inputot és outputot.
Definíció. *Jelölje $T_M(n)$ az M gép maximális számolási idejét az n jelből álló bemeneteken. Az n hosszú szavakon a maximális tárigényt $S_M(n)$-nel jelöljük.*

Ha van olyan n jelből álló $s \in I^*$ szó, amellyel elindítva M nem áll meg véges sok lépés után $\Longrightarrow T_M(n) = \infty$
Pl. a hárommal oszthatóságot vizsgáló M TG-re: $T_M(n) = n + 1$, $S_M(n) = n + 1$, ha beleszámítjuk az inputot, $S_M(n) = 0$, ha nem.
Ha M és N két Turing-gép, melyekre $T_M(n) < T_N(n)$ teljesül minden elég nagy n-re, akkor az M algoritmust gyorsabbnak mondhatjuk az N algoritmusnál.

Van-e legjobb TG minden L nyelvhez?

Tétel. [gyorsítási tétel] *Van olyan L nyelv, amelyre igazak az alábbiak:*
1. Az L felismerhető egy olyan M Turing-géppel, melyre $T_M(n)$ véges minden n-re.
2. Tetszőleges, az L-et felismerő N Turing-géphez van olyan N' Turing-gép, amelyre $L = L_{N'}$ szintén teljesül, továbbá $T_{N'}(n) = O(\log T_N(n))$.

k-szalagos TG szimulációja 1-szalagossal

Tétel. *Legyen M egy k-szalagos Turing-gép. Van olyan egyszalagos M' Turing-gép, melyre*

$$L_M = L_{M'} \ \text{(vagy } f_M = f_{M'}\text{), továbbá}$$

$$T_{M'}(n) \le 2T_M^2(n),$$
$$S_{M'}(n) \le S_M(n) + n.$$

Bizonyítás: M' építésekor az M-hez képest alaposan felfújjuk a szalag abc-t, és megnöveljük a belső állapotok számát is. Az M' egyetlen szalagján $2k$ csík lesz. Egy cellája egy oszlop.

1. szalag	D	$\cdots$	A	$\cdots$	B	$\cdots$
1. fej	ü	$\cdots$	x	$\cdots$	ü	$\cdots$
.						
.						
.						
k. szalag	D	$\cdots$	D	$\cdots$	B	$\cdots$
k. fej	ü	$\cdots$	ü	$\cdots$	x	$\cdots$

$\Longrightarrow$ szalagjelek száma: $(2t)^k$

1. szalag	D	$\cdots$	A	$\cdots$	B	$\cdots$
1. fej	ü	$\cdots$	x	$\cdots$	ü	$\cdots$
.						
.						
.						
k. szalag	D	$\cdots$	D	$\cdots$	B	$\cdots$
k. fej	ü	$\cdots$	ü	$\cdots$	x	$\cdots$

M' az M gép egy lépését egy legfeljebb $2T_M(n)$ lépésből álló menetben utánozza.
Jobbra elmegy a legmesszebb levő x jelig, és közben leolvassa az x-ek felett található eredeti szalagjeleket. Ezeket a belső állapotaiban tárolja.
Közben egy hellyel jobbra mozdítja az x-eket, kivéve az utolsó oszlopban levő(ke)t.
Most a gép ismeri M állapotát és a fejei alatti jeleket, így meghatározhatja a M következő állapotát és a kiírandó jeleket.

M' visszamegy a szalag elejére, közben kiírja az M fejeinek régi helyére a k darab jelet, amiket M írt volna,
és az M fejmozgásainak megfelelően áthelyezi az x-eket.

Ha n jelből álló bemenettel kezdjük a munkát, akkor egy meneteben az M' feje legfeljebb $T_M(n)$ lépést tesz jobbra $\Longrightarrow$ legfeljebb ugyanennyit mehet balra.
Az M' pontosan akkor álljon meg (fogadja el a bemenetet), ha M ezt teszi.
$\Longrightarrow T_{M'}(n) \le 2T_M(n)T_M(n) = 2T_M^2(n)$

Ha függvényt számítunk, a végén a felesleget letöröljük.

Ha M-nek kitüntetett input szalagja volt, akkor a bemenet hosszát, ami n, nem számítottuk bele $S_M(n)$-be $\Longrightarrow S_{M'}(n) \le S_M(n) + n$.

Tétel. *Az M k-szalagos Turing-géphez megadható olyan 2-szalagos M' Turing-gép, amely az előbbi értelemben szimulálja M-et,*

$$T_{M'}(n) \le O(T_M(n) \log T_M(n)), \text{ és}$$

$$S_{M'}(n) \le S_M(n) + n.$$

Tétel. *Tegyük fel, hogy az M Turing-gép az L nyelvet ismeri fel, és $T_M(n) \le cn$ teljesül egy $c > 0$ állandóval. Ekkor tetszőleges $\epsilon > 0$-ra van olyan L-et felismerő M' Turing-gép is, hogy alkalmas $n_0 \in \mathbb{N}$ számmal*

$$T_{M'}(n) \le n(1 + \epsilon), \text{ ha } n \ge n_0.$$

Bizonyítás: Az M' gépet úgy tervezzük, hogy az M gép m lépését az új legfeljebb 7 lépésben elvégzi.
Osszuk fel az M szalagjait m egymás utáni mezőből álló blokkokra $\Longrightarrow$ egy új szalagjel $\Longrightarrow$ az új gép t^m betűt használ.
Az M'-nek eggyel több szalagja lesz, mint M-nek. Az első input szalag, ezt először átkódolja egy másik szalagra.

Mi történik a szalagokkal az M gép m egymást követő lépése során?
Ami ezalatt végbemegy, az csak a fejeket tartalmazó blokkoktól és azok közvetlen szomszédaitól függ.
M': szalagonként három szomszédos jel megvizsgálása után a „memóriájában" meglépi M következő m lépését.

$\Longrightarrow$ felülírja a szomszédos mezőhármasokat, helyükre teszi a fejeket.

$\Longrightarrow$ szimuláció elvégezhető egy bal–jobb–jobb–bal lépéssorozatban esetleg további 2 jobbralépés szükséges lehet $\Longrightarrow M'$ feleinek mozgatása $\Longrightarrow$ 7 lépés
A bemenet átkódolása, majd a kódolt szalagon a fejnek a szalag elejére mozgatása $\Longrightarrow \le n + \left\lceil \frac{n}{m} \right\rceil$ lépés

$$T_{M'}(n) \le n + \left\lceil \frac{n}{m} \right\rceil + 7\left\lceil \frac{T_M(n)}{m} \right\rceil \le n + \frac{n}{m} + \frac{7T(n)}{m} + 8 \le$$
$$\le n + \frac{n}{m} + \frac{7cn}{m} + \frac{8n}{n} \le n\left(1 + \frac{1}{m} + \frac{7c}{m} + \frac{8}{n}\right) \le n(1 + \epsilon),$$

ha m és n olyan nagyok, hogy $\frac{1}{m} + \frac{7c}{m} + \frac{8}{n} < \epsilon$. $\checkmark$

Komolyabb, „hardverrel" könnyebb.

A Turing-gépmodellben a feladatok időigénye függ a jelkészlet méretétől

13. előadás

Algoritmuselmélet 13. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Április 15.

Univerzális Turing gép

Turing-gép ↔ program
Univerzális Turing-gép ↔ fordító program (interperter)

M TG leírása és $s \in I^*$ bemenete $\Longrightarrow$ Univerzális TG $\Longrightarrow$ szimulálja M-et s bemenettel

Hogyan írjuk le az M TG-et?

Tegyük fel, hogy $k = 1$, $M = (Q, T, I, ü, \delta, q_0, F)$, $I = \{0, 1\}$ és $|F| = 1$.
Pl.

- $I = \{0, 1\}, \quad T = \{0, 1, \ldots, t\}, \quad ü = t,$
- $Q = \{0, 1, \ldots, q\}, \quad q_0 = 0, \quad F = \{q\},$
- balra $= 0$, jobbra $= 1$, helyben $= 2$

Ekkor az M Turing-gép leírása, kódja $\Longrightarrow$

$$q\#t\#q_1\#x_1\#q'_1\#x'_1\#m'_1\# \ldots \#q_r\#x_r\#q'_r\#x'_r\#m'_r\#\#,$$

ahol a megfelelő számokat binárisan írjuk le, továbbá δ értékeit a következőképpen soroljuk fel (ahol értelmezett):

a $\delta(q_i, x_i) = (q'_i, x'_i, m'_i)$ tény kódja $q_i\#x_i\#q'_i\#x'_i\#m'_i$.

Minden szóba jövő M gépet egy $w \in I^*$ szóval írunk le.
Tetszőleges $w \in I^*$ szóhoz legfeljebb egy gép van, amelynek a kódja $w \Longrightarrow M_w$
Egymásból kiszámolható M és M_w.

Tétel. *Van olyan 3-szalagos U Turing-gép, amelyre teljesül a következő: ha $w, s \in I^*$, és M_w létezik, akkor az U gép a $w\#s$ bemenetet pontosan akkor fogadja el (utasítja el, kerül vele végtelen ciklusba), ha M_w az s bemenetet elfogadja (elutasítja, végtelen ciklusba kerül vele).*

Bizonyítás: (vázlat)
Első szalag $\Longrightarrow$ $w\#s$ input, w értelmezgetése
Második szalag $\Longrightarrow$ M_w egyetlen szalagjának felel meg.
Harmadik szalag $\Longrightarrow$ az M_w belső állapota

Előkészítés $\Longrightarrow$ ellenőrzi, hogy M_w létezik-e.
$\Longrightarrow$ NEM $\longrightarrow$ megáll elutasító állapotban
$\Longrightarrow$ IGEN $\longrightarrow$ átmásolja az s inputot a második szalagjára, és a harmadik szalagra a kezdőállapot kódját jegyzi fel.

$w\#s$
s
q_0

Az U az M_w gép egy lépését több lépésben szimulálja $\Longrightarrow$

$w\#s$
M_w szalagja az i-edik lépés után
M_w belső állapota az i-edik lépés után

U akkor áll meg, ha M_w megáll, pontosan akkor fogadja el a $w\#s$ bemenetét, ha a megállás után az M_w elfogadó állapotának kódja van az utolsó szalagon. $\surd$

Alapvető kiszámíthatatlansági tételek

Be fogjuk látni, hogy

$$\mathcal{R} \subsetneq \mathcal{RE} \subsetneq 2^{I^*}.$$

Definíció. *Azon gépek kódjainak L_d nyelve, amik nem fogadják el saját kódjukat a diagonális nyelv:*

$$L_d = \{w \in I^*; \text{ az } M_w \text{ gép létezik, és } w \notin L_{M_w}\}.$$

Tétel. *L_d nem rekurzíve felsorolható.*

Bizonyítás: Indirekt, tegyük fel hogy rekurzíve felsorolható $\Longrightarrow \exists\, M$ TG amire $L_d = L_M$, ennek kódja legyen w

- $w \in L_d \Longrightarrow L_d$ definíciója szerint $w \notin L_{M_w} = L_d$ $\lightning$
- $w \notin L_d \Longrightarrow L_d$ definíciója szerint $w \in L_d = L_{M_w}$ $\lightning$

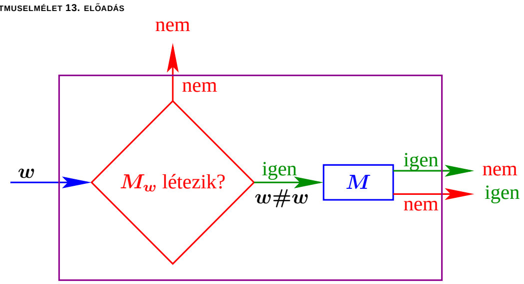
Az univerzális nyelv

Definíció. *Az olyan* (TG kód, input szó) *párok L_u nyelve, amelyekre a gép elfogadja az inputot az univerzális nyelv:*

$$L_u = \{w\#s \in I^*; \text{ az } M_w \text{ gép létezik, és } s \in L_{M_w}\}.$$

Tétel. [A. Turing, 1936] *L_u egy rekurzíve felsorolható, de nem rekurzív nyelv.*

Bizonyítás: L_u-t éppen az univerzális Turing-gépek ismerik fel $\Longrightarrow L_u$ rekurzíve felsorolható.
Tegyük fel indirekt, hogy L_u rekurzív; legyen M egy mindig megálló TG, ami felismeri L_u-t.
Konstruáljuk meg az M' TG-et:



Az M' gép mindig megáll, mert M mindig megáll.
M' pontosan akkor fogadja el w-t, ha M_w létezik és $w \notin L_{M_w}$.
$\Longrightarrow M'$ éppen az L_d nyelvet fogadja el $\lightning$
hiszen L_d nem rekurzíve felsorolható $\surd$

Összefüggések a kiszámíthatósági fogalmak között

Ha van egy mindig megálló M algoritmusunk L felismerésére, akkor van algoritmus az $I^* \setminus L$ nyelv felismerésére is.
Ha csak olyan „fél" algoritmusunk van, ami nem mindig áll meg, ezt nem lehet megtenni.
Ha van fél algoritmus L-re és $I^* \setminus L$-re is, akkor ebből összejön egy egész algoritmus
Definíció. *A nyelvekből álló halmazokat (2^{I^*} részhalmazait) nyelvosztályoknak nevezzük. (Pl. $\mathcal{R}$, $\mathcal{RE}$, 2^{I^*}.)*
Legyen $X \subseteq 2^{I^*}$ egy nyelvosztály. Ekkor a komplementer nyelvosztály, co$X$ az X-beli nyelvek komplementereiből áll:

$$\text{co}X = \{L \subseteq I^* : \ I^* \setminus L \in X\}.$$

$\Longrightarrow$

$$X \subseteq Y \subseteq 2^{I^*} \Longrightarrow \text{co}X \subseteq \text{co}Y.$$
$$\text{co}(\text{co}X) = X$$

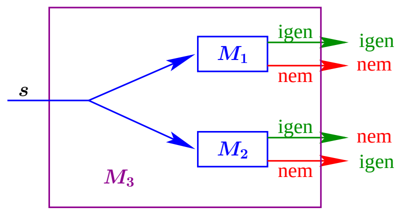
Tétel. $\mathcal{R} = co\mathcal{R}$

Bizonyítás: Ha $L \in \mathcal{R} \implies \exists\, M$ TG, mely minden inputon megáll és az L nyelvet fogadja el.
Cseréljük fel M elfogadó és elutasítva megálló állapotait $\implies$ co$\mathcal{R} \subseteq \mathcal{R}$. $\surd$
Másik irány:

$$\mathcal{R} = \mathrm{co}(\mathrm{co}\mathcal{R}) \subseteq \mathrm{co}\mathcal{R}. \quad \surd$$

Tétel. $\mathcal{R} = \mathcal{RE} \cap co\mathcal{RE}$

Bizonyítás: $\mathcal{R} \subseteq \mathcal{RE} \implies \mathcal{R} = \mathrm{co}\mathcal{R} \subseteq \mathrm{co}\mathcal{RE} \implies \mathcal{R} \subseteq \mathcal{RE} \cap \mathrm{co}\mathcal{RE}$
Másik irány:
Tegyük fel, hogy $L \in \mathcal{RE} \cap \mathrm{co}\mathcal{RE} \implies$ legyen M_1, illetve M_2 két TG, melyek az L, illetve az $I^* \setminus L$ nyelvet ismerik fel.
Egy mindig megálló M_3 TG-et szerkesztünk, melyre $L = L_{M_3}$:



Az M_3 pontosan akkor álljon meg, ha M_1 és M_2 valamelyike megáll.
M_3 akkor fogad el, ha a megállás M_1 elfogadó, vagy pedig M_2 elutasító állapotában történt.
$\implies M_3$ az L nyelvet ismeri fel és mindig megáll

M_3 megvalósítható párhuzamosság nélkül is $\implies M_3$ felváltva lépteti M_1-et és M_2-t. $\surd$

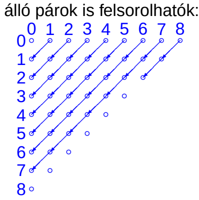
Függvények és halmazok

Tétel. *Az $L \subseteq I^*$ nyelv akkor és csak akkor rekurzíve felsorolható, ha van olyan $f: I^* \to I^*$ parciálisan rekurzív függvény, melynek értékkészlete éppen az L nyelv (szokásos jelöléssel Im$(f) = L$).*

Bizonyítás: $\Rightarrow$: Tegyük fel, hogy L rekurzíve felsorolható $\implies \exists M$ TG, melyre $L = L_M$.
$\implies$ Legyen M' olyan TG, mely kezdetben az $s \in I^*$ bemenő szót felmásolja az output szalagjára, azután szimulálja az M gépet.
Kivétel: $\implies$ ha M elutasító állapotban áll meg, akkor M' végtelen ciklusba esik.
$\implies M'$ gép csak az $s \in L$ szavakra áll meg; megálláskor s lesz az output szalagon. $\implies$
Az M' által kiszámított $f_{M'}$ függvény értékkészlete L. $\surd$

$\Leftarrow$: Tegyük fel, hogy f egy parciálisan rekurzív függvény, $f = f_M$, ahol M egy TG.
Tekintsük az I^* szavainak $w_0, \ldots, w_n \ldots$ kanonikus felsorolását.
Ki kellene próbálni minden w_i inputtal, hogy hátha pont s-et a keresett szót számolja ki M.

Baj van, ha valamelyik szóra végtelen ciklusba kerül.
A természetes számokból álló párok is felsorolhatók:



Az M' TG az (i,j) párok sorozata szerint megy sorba. Tegyük fel, hogy $s \in I^*$ az M' bemenete.
$\implies M'$ szimulálja az M első $\leq i$ lépését a w_j szón.
Ha M megáll és o outputot produkál, akkor ellenőrzi, hogy $s = o$ teljesül-e.
IGEN $\to M'$ megáll és elfogadja s-et.
NEM (nem áll meg i lépésen belül, vagy az eredmény nem s) $\implies$ következő pár

Mi lesz az M' gép $L_{M'}$ nyelve?
$L_{M'} \subseteq$ Im(f) $\surd$
Ha $s \in$ Im$(f) \implies \exists j$, hogy $s = f_M(w_j)$
$\implies$ ha M a w_j bemeneten i lépésben kapja meg az s eredményt $\implies M'$ az (i,j) pár feldolgozásakor elfogadja s-et
$\implies L_{M'} \supseteq$ Im(f) $\surd$
Definíció. *Egy $L \subseteq I^*$ nyelv* karakterisztikus függvénye, *χ_L a következő:*

$$\chi_L(s) = \begin{cases} 1 \in I^*, & \text{ha } s \in L \\ 0 \in I^*, & \text{ha } s \notin L \end{cases}$$

Tétel. *Az $L \subseteq I^*$ nyelv pontosan akkor rekurzív, ha χ_L egy rekurzív függvény.*

Bizonyítás: $\Rightarrow$: M' írojon ki a végén 0-t vagy 1-et $\surd$
$\Leftarrow$: Ha 1-et ír ki $\to$ menjen elfogadóba, ha 0-t $\to$ elutasítóba $\surd$

Eldönthetetlen problémák

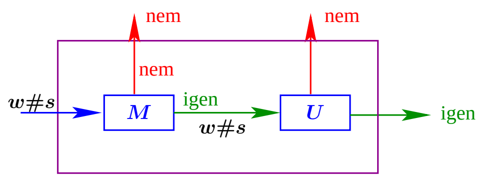
Definíció. *Az $L \subseteq I^*$ nyelvet* eldönthetetlen nyelvnek *nevezzük, ha L nem rekurzív.*
Definíció. Megállási probléma: *Megáll-e egy TG egy adott inputon?*

$$L_h = \left\{ w\#s \in I^* \;\middle|\; \begin{array}{l} \text{az } M_w \text{ gép létezik, és az } s \text{ bemenettel} \\ \text{elindítva véges sok lépésben megáll} \end{array} \right\}.$$

Tétel. $L_h \in \mathcal{RE} \setminus \mathcal{R}$.

Bizonyítás: $L_h \in \mathcal{RE}$: Vegyünk egy univerzális Turing-gépet, amit kicsit módosítunk: ha megáll menjen át elfogadóba

$L_h \notin \mathcal{R}$: Indikekt, tegyük fel, hogy rekurzív $\implies \exists M$ TG, ami felismeri és mindig megáll.



Ez gép L_u-t felismeri és mindig megáll $\lightning$
Tétel. [A. Church, 1936] *Legyen*

$$L_\epsilon = \{w \in I^*: \; M_w \text{ létezik és az } \epsilon \text{ (üres) inputon megáll}\}.$$

$$L_\epsilon \in \mathcal{RE} \setminus \mathcal{R}.$$

Bizonyítás: $L_\epsilon \in \mathcal{RE}$: Az üres inputtal futassuk az L_h-t felismerő gépet $\surd$
$L_\epsilon \notin \mathcal{R}$: Belátjuk, hogy ha L_ϵ rekurzív ($\exists M$, ami felismeri és mindig megáll), akkor L_h is.
Ha L_h bemenete $w\#s$, akkor olyan M'-t konstruálunk, aminek belső állapotaiba kódoljuk az s inputot. $\surd$

Hilbert 10. problémája

Legyen $f(x_1, \ldots, x_m)$ egész együtthatós m változós polinom:

$$f(x_1, \ldots, x_m) = \sum_{i_1=0}^{n_1} \cdots \sum_{i_m=0}^{n_m} a_{i_1 \ldots i_m} {x_1}^{i_1} \cdots {x_m}^{i_m}$$

Az f polinom *foka* az előző felírásban előforduló legnagyobb kitevőösszeg:

$$\deg f = \max\{i_1 + \ldots + i_m \mid a_{i_1 \ldots i_m} \neq 0\}.$$

Az

$$(*) \qquad f(x_1, \ldots, x_m) = 0$$

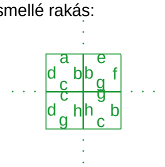
alakú egyenleteket *diofantikus egyenleteknek* nevezzük.
A $(*)$ diofantikus egyenlet *megoldásán* egy olyan $(u_1, \ldots, u_m) \in \mathbb{Z}^m$ egészekből álló m-est értünk, melyre $f(u_1, \ldots, u_m) = 0$.

Van-e megoldása egy adott diofantoszi egyenletnek?
Tétel. [Matijaszevics, 1970] *Ez eldönthetetlen probléma.*

A Dominóprobléma

Dominó:

Egymásmellé rakás:



Forgatni nem szabad!

Dominóprobléma: *Adott dominó-típusok egy véges $\mathcal{F}$ halmaza; eldöntendő, hogy a sík lefedhető-e hézagtalanul szabályosan illeszkedő $\mathcal{F}$-beli típusú dominókkal.* $\Longrightarrow$ D nyelv

Tétel. $D \notin \mathcal{R}$ és $D \in co\mathcal{RE}$.

Post megfeleltetési problémája

Emil Post

Legyen Σ egy véges abc. *Post megfeleltetési problémájának* egy bemenete egy (s,t) $(s,t \in \Sigma^*)$ alakú rendezett párokból álló véges $\mathcal{P}$ halmaz.
A megfeleltetési feladat $\mathcal{P}$ bemenetét *megoldhatónak* nevezzük, ha vannak olyan (nem feltétlenül különböző) $\mathcal{P}$-beli $(s_1,t_1),(s_2,t_2),\ldots,(s_n,t_n)$ párok úgy, hogy

$$s_1s_2\cdots s_n = t_1t_2\cdots t_n.$$

Ilyenkor az $s_1s_2\cdots s_n$, vagy ami ugyanaz, a $t_1t_2\cdots t_n$ szót a $\mathcal{P}$ *megoldásának* nevezzük.
Például a $\mathcal{P} = \{(iz, riz), (kar, ka), (ma, ma)\}$ rendszer megoldható. Egy lehetséges megoldás a *karizma* szó.

Tétel. *Ez a probléma eldönthetetlen.*

14. előadás

Algoritmuselmélet 14. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Április 22.

Idő- és tárkorlátok

Definíció. *Legyen $t : \mathbb{Z}^+ \to \mathbb{Z}^+$ egy függvény, melyre minden $n \in \mathbb{Z}^+$ esetén $t(n) \geq n$ teljesül. Az M Turing-gép $t(n)$ időkorlátos, ha n hosszú inputokon legfeljebb $t(n)$ lépést tesz (más szóval $T_M(n) \leq t(n)$).*

$t(n) \geq n$ $\Longrightarrow$ a gép legalább végigolvashatja az inputot
M gyors $\Longrightarrow$ $t(n)$ lassan növekedő függvény

Példa: hárommal való oszthatóságot ellenőrző Turing-gép $\Longrightarrow n+1$ időkorlátos

Definíció.

$$TIME(t(n)) := \left\{ L \subseteq I^* \;\middle|\; \begin{array}{l} L \text{ felismerhető egy } O(t(n)) \text{ időkorlátos} \\ M \text{ Turing-géppel} \end{array} \right\}.$$

$TIME(t(n))$ $\Longrightarrow$ létezik $ct(n)$ időkorlátos TG
n hosszú x inputokon a számítás *mindig befejeződik* legfeljebb $ct(n)$ lépésben, tekintet nélkül arra, hogy $x \in L$ igaz-e
$\Longrightarrow TIME(t(n)) \subset \mathcal{R}$

Példa: $TIME(n) = \{$az $O(n),$ azaz lineáris időben felismerhető nyelvek$\}$.

A P nyelvosztály

Definíció. $\mathrm{P} = \cup_{k \geq 1} TIME(n^k)$*, a polinom időben felismerhető nyelvek osztálya.*

Tétel. *Ha az L nyelv $n^{\log n}$-nél rövidebb időben nem ismerhető fel, akkor $L \notin \mathrm{P}$.*

Bizonyítás: Indirekt tegyük fel, hogy $L \in \mathrm{P}$. $\Longrightarrow$ van olyan $k > 0$, melyre $L \in TIME(n^k)$ $\Longrightarrow n^{\log n} \leq cn^k$ teljesülne végtelen sok n-re $\lightning$

Tárkorlát

Definíció. *Legyen $s : \mathbb{Z}^+ \to \mathbb{Z}^+$ olyan függvény, melyre minden $n \in \mathbb{Z}^+$ számmal igaz, hogy $s(n) \geq \log_2 n$. Az M Turing-gép $s(n)$ tárkorlátos, ha n hosszú inputokon legfeljebb $s(n)$ tárcellát használ a munkaszalagokon (azaz $S_M(n) \leq s(n)$).*

$s(n) \geq \log_2 n$ $\Longrightarrow$ Ennyi hely kell ahhoz, hogy egy n cellából álló szalagrészt címezni tudjunk.

Definíció.

$$SPACE(s(n)) := \left\{ L \subseteq I^* \;\middle|\; \begin{array}{l} \text{az } L \text{ felismerhető egy } O(s(n)) \\ \text{tárkorlátos } M \text{ Turing-géppel} \end{array} \right\}.$$

Ez nem biztos, hogy egyátalán megáll!

Példa: $SPACE(\log n)$ a *logaritmikus tárban felismerhető nyelvek* osztálya.

Függvényosztályok

Definíció. $FTIME(t(n)) :=$ *az $O(t(n))$ időkorlátos TG-k által kiszámítható $f : I^* \to I^*$ függvények osztálya.*

Definíció. $FSPACE(s(n)) :=$ *az $O(s(n))$ tárkorlátos TG-k által kiszámítható $f : I^* \to I^*$ (parciális) függvények osztálya.*

Definíció. $\mathrm{FP} := \cup_{k \geq 1} FTIME(n^k)$.

Tár–idő-tétel

Tétel. [tár–idő-tétel] *Ha $L \in SPACE(s(n))$, akkor van olyan L-től függő c konstans, mellyel $L \in TIME(c^{s(n)})$ teljesül.*

Bizonyítás: Legyen M egy $S(n) = c_1 s(n)$ tárkorlátos ($c_1 \geq 1$) k-szalagos TG, mely felismeri L-et $\Longrightarrow$ konstruálunk olyan $O(c^{S(n)})$-időkorlátos N TG-t, melynek a nyelve szintén L.

a gép egy pillanatnyi helyzete $\Longrightarrow$ az input, a munkaszalagok tartalma, a gép aktuális belső állapota, valamint a fejek helyzete $\Longrightarrow$ PHL
kétszer ugyanaz a PHL $\Longrightarrow$ utána ugyanaz fog történni $\Longrightarrow$ végtelen ciklus
Belátjuk, hogy ha M tárkortátos $\Longrightarrow$ véges sok PHL lehet $\Longrightarrow$ biztos végtelen ciklusba fog kerülni (ha nem áll meg)
Ezt kellene felismerni $O(c^{S(n)})$ idő alatt.

Hány darab PHL lehetséges összesen, ha a gépet n hosszú inputtal indítjuk?

$$\#PHL \leq |Q||T|^{S(n)}(n+1)S(n)^k \leq \text{konstans} \cdot c_2^{S(n)} =: t,$$

(az $(n+1)$ tényező az input fej lehetséges helyzeteinek a száma, az $S(n)^k$ tényező pedig a többi fej lehetséges helyzeteinek a száma)

Ha t lépés után lelőjük ✓ de lehet, hogy t nem rekurzív ↯

N konstrukciója: megduplázzuk M-et $\to M_1$ és M_2
M_1-et elindítjuk az x inputtal.
Minden egyes lépése után ideiglenesen megállítjuk
$\to$ ekkor M_2-t elindítjuk x inputtal a kezdő állapotból, és működtetjük legfeljebb addig a lépésig, ahol M_1 tart ($\to$ ennek sorszáma l, amit $O(S(n))$ extra cellán tárolunk és léptetünk.)

Ha valamely $j < l$-re az M_2 gép j-edik lépés utáni PHL-je megegyezik M_1-ével $\implies$ végtelen ciklus $\implies x \notin L \implies N$ megáll elutasítva x-et
Ha ilyen ismétlődés nem fordult elő, akkor meglépjük M_1 következő, $l+1$-edik lépését, stb.
ha M_1 megáll elfogadva (elutasítva) x-et $\implies N$ is megáll elfogadva (elutasítva) x-et.

$\implies$ felismerjük a végtelen ciklusokat
mindig teljesül $l \le t \implies$ a maximális futási idő legfeljebb
$O(t^2) = O((c_2^{S(n)})^2) = O((c_2^2)^{c_1 s(n)}) \implies c = c_2^{2c_1}$ ✓
a tárfelhasználás közben nem nőtt lényegesen

Tétel. *Ha $f \in FSPACE(s(n))$, akkor van olyan f-től függő c konstans, mellyel $f \in FTIME(c^{s(n)})$.*

Tétel. $TIME(t(n)) = coTIME(t(n))$.

Bizonyítás: Megcseréljük M elfogadó és elutasító (azaz nem elfogadó) állapotait ✓
Tétel. $SPACE(s(n)) = coSPACE(s(n))$.

Bizonyítás: Alkalmazzuk a tár–idő-tétel szimulációját. Az adódó N TG szintén $O(s(n))$ tárkorlátos, és *minden inputra megáll*.
Most cseréljük fel az elfogadó és az elutasító állapotokat.

Definíció. $\mathrm{EXPTIME} := \cup_{k \ge 1} TIME(2^{n^k})$.
Definíció. $\mathrm{PSPACE} := \cup_{k \ge 1} SPACE(n^k)$.

Tétel. $\mathrm{P} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXPTIME}$

Bizonyítás: Ha M egy $t(n)$ időkorlátos TG $\implies ct(n)$ tárkorlátos $\impliedby$
$\implies TIME(n^k) \subseteq SPACE(n^k) \implies \mathrm{P} \subseteq \mathrm{PSPACE}$ ✓
Legyen $L \in \mathrm{PSPACE} \implies L \in SPACE(n^k)$, valamely k-ra
tár–idő-tétel $\implies$ van olyan $c > 0$, hogy
$L \in TIME(c^{n^k}) \subseteq TIME(2^{\lceil c \rceil n^k}) \subseteq TIME(2^{n^{k+1}}) \subseteq \mathrm{EXPTIME}$. ✓

Tétel. $TIME(t(n)) \subset \mathcal{R}$, $SPACE(s(n)) \subset \mathcal{R}$ *és* $\mathrm{EXPTIME} \subset \mathcal{R}$.

Bizonyítás: Csak azt látjuk be, hogy $\exists L$ rekurzív nyelv, hogy $L \notin \mathrm{EXPTIME}$, a többi állítás hasonlóan kijön

$L = \{w \in I^*;$ az M_w TG létezik, és legfeljebb $2^{2^{|w|}}$ lépésben elutasítja w-t$\}$.

Ez rekurzív, mert mindig megálló univerzális TG felismeri.

Belátjuk, hogy $L \notin TIME(2^{2^{n-1}})$
Indirekt, tegyük fel, hogy L felismerhető egy $c2^{2^{n-1}}$ időkorlátos M TG-vel.
Legyen n_0 olyan nagy, hogy $c2^{2^{n-1}} < 2^{2^n}$ teljesüljön, ha $n > n_0$.

Legyen w egy n_0-nál hosszabb szó, melyre M_w létezik, és ugyanúgy viselkedik mint M (ilyen van).
$\implies$ ha $w \in L$, akkor M_w elfogadja w-t $c2^{2^{|w|-1}} < 2^{2^{|w|}}$ lépésben
$\implies w \notin L$
fordítva is ↯

Nemdeterminisztikus Turing-gépek

Olyan TG, ahol az átmenetfüggvény nem igazi függvény, több lehetőség van:

$$\delta(q, a) \subseteq Q \times T \times \{\text{jobb}, \text{bal}, \text{helyben}\}.$$

Gép futása, számítási út: Mint a TG, csak ha több lehetőség van, választ egyet, ha nincs értelmezve, akkor megáll.

Definíció. *Az M NTG elfogadja az $x \in I^*$ inputot, ha az M-et x bemenettel a kiinduló helyzetből indítva van legalább egy elfogadó (egy elfogadó állapotban véget érő) számítási út.*

Tétel. *Az $x \in I^*$ input szó pontosan akkor nincs L_M-ben, ha az M gépet x inputtal indítva nincs elfogadó számítási út.*

NTG számítási $\implies$ gyökeres fa $\implies$ csúcsok $\leftrightarrow$ PHL

Időkorlátos NTG

Definíció. *Egy M nemdeterminisztikus Turing-gép $t(n)$ időkorlátos, ha n hosszúságú inputokon M minden számítási út mentén legfeljebb $t(n)$ lépést téve megáll.*

Senki nem tud egy $t(n)$ időkorlátos NTG-t $O(t(n))$ időben szimulálni.

Definíció. $NTIME(t(n)) :=$
$\{$*az $O(t(n))$ időkorlátos NTG-k által elfogadott nyelvek*$\}$.

Definíció. $\mathrm{NP} := \cup_{k \ge 1} NTIME(n^k)$.

Tétel. $\mathrm{P} \subseteq \mathrm{NP}$.

Bizonyítás: A NTG egyben TG is ugyanolyan időkorláttal
$\implies TIME(n^k) \subseteq NTIME(n^k) \implies$ ✓

A legfontosabb megoldatlan probléma

$$\mathrm{P} \stackrel{?}{=} \mathrm{NP}$$

Tétel. $\mathrm{P} \subseteq \mathrm{NP} \cap \mathrm{coNP}$.

Bizonyítás: $\mathrm{P} \subseteq \mathrm{NP} \implies \mathrm{coP} \subseteq \mathrm{coNP}$.
$\mathrm{P} = \mathrm{coP} \implies$ ✓

Szintén megoldatlan problémák:

$$\mathrm{P} \stackrel{?}{=} \mathrm{NP} \cap \mathrm{coNP}.$$

$$\mathrm{NP} \stackrel{?}{=} \mathrm{coNP}.$$

Nemdeterminisztikus felismerés

Hogyan tudjuk belátni, hogy $L \in \mathrm{NP}$?

Legyen M kétszalagos determinisztikus TG, inputja két részből áll, egyik része $x \in I^*$ az első szalagon van, a másik része $y \in I^*$ a másikon $\implies$ ez csak olvasható
$\implies$ az M *súgásszalagja*

Az M által felismert L_1 nyelv $\implies$ azon (x, y) szópárok halmaza ($x, y \in I^*$), melyeket M elfogad
Definíció. *Az M által nemdeterminisztikusan felismert L nyelv a következő:*

$x \in L$ akkor, és csak akkor, ha van olyan y súgás, hogy $(x, y) \in L_1$.

Nem tudunk semmit arról, hogyan lehet jó súgást találni.

Tanú–tétel

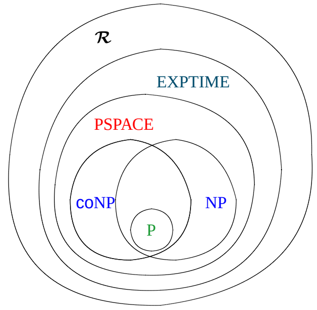
Tétel. *[tanú-tétel] Egy $L \subseteq I^*$ nyelvre a következő két állítás egyenértékű:*
(a) $L \in \mathrm{NP}$.
(b) Van olyan $c > 0$ állandó, továbbá egy $L_1 \in \mathrm{P}$ nyelv, mely olyan $(x, y) \in (I^*)^2$ párokból áll, hogy $|y| \le |x|^c$ és $x \in I^*$ esetén $x \in L$ pontosan akkor, ha van $y \in I^*$ úgy, hogy $(x, y) \in L_1$.

Bizonyítás: $(a) \Rightarrow (b)$ **:**
$L \in \mathrm{NP} \implies \exists \quad n^{c_1}$ időkorlátos N NTG, mely felismeri L-et
Tegyük fel, hogy N-nek egy lépésnél legfeljebb d elágazási lehetősége van
Adunk egy M-et és egy megfelelő súgást

$x \in L$, $|x| = n \implies y = y_1 y_2 \cdots y_m$ legyen az x elfogadását leíró számítási útja
$\implies |y| \le n^{c_1} \lceil \log_2(d+1) \rceil \le n^c$
Az M TG szimulálja N-et, úgy hogy mindig a kijelölt úton megy tovább
N egy lépését konstans időben szimulálja.
N futási ideje éppen $c_2|y| \implies M$ futása az inputhoz képest lineáris
Ha viszont $x \notin L \implies (x, y) \notin L_1 = L_M$ tetszőleges $y \in I^*$-ra $\surd$

$(b) \Rightarrow (a)$ **:** Egy $x \in L$ inputhoz a feltétel szerint van legfeljebb n^{c_1} hosszúságú y, hogy $(x, y) \in L_1$.
Legyen N olyan mint M, de amikor M y-ból olvasna 0-t vagy 1-et
$\implies N$-ben δ-ra két lehetőség
Az N NTG n^{c_2} időkorlátos $\surd$
Tétel. $\mathrm{NP} \subseteq \mathrm{PSPACE}$

Bizonyítás: Ha $L \in \mathrm{NP} \implies \exists$ súgás
Adott $x \in L$-re végigpróbáljuk az összes n^c szóbajövő súgást
Ez nagyon sokáig tart, de csak $\log_2 2^{n^c} = n^c$ tár kell hozzá



Sejtés: $\mathrm{P} \ne \mathrm{PSPACE}$

$\mathrm{NP} \ne \mathrm{coNP} \Rightarrow \mathrm{P} \ne \mathrm{NP} \Rightarrow \mathrm{PSPACE} \ne \mathrm{P}$

15. előadás

Algoritmuselmélet 15. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Április 23.

NP-beli nyelvek

A tanú tétel segítségével könnyű belátni, hogy egy nyelv NP-beli.

Csak azt kell belátni, hogy létezik tanú, megkeresni nem kell tudni.

A mi szerepünk a bíró szerepe, nem a nyomozóé.

3 színnel színezhető gráfok

$G \to$ pl. adjacencia mátrix sorai egymásután fűzve
$x \in$ 3-SZÍN $\implies$ ha az x szónak megfelelő gráf 3 színnel színezhető.

Tétel. *3-SZÍN* $\in \mathrm{NP}$.

Bizonyítás: Alkalmas tanú G egy jó színezése
Ez leírható $2n$ bittel $\implies$ (pl.legyen 01=piros, 10=sárga, 11=zöld)

G, színezés $\to$ bíró $\implies$ jó színezés-e
Ez polinom időben megtehető TG-vel.
Ha G nem 3-színezhető, akkor nem lehet tanúja.

Hamilton-körrel rendelkező gráfok

$H \implies$ Azon gráfok szavai, amik tartalmaznak Hamilton-kört.

Tétel. *H* $\in \mathrm{NP}$.

Bizonyítás: A $G \in$ H állításnak rövid tanúja egy Hamilton-kör.
csúcsok sorrendje $\implies O(n \log n)$ bit
a bíró ellenőrzi, hogy van-e él a következő csúcsba G-ben.

Hasonlóan Hamilton-útra, irányított Hamilton-körre, -útra

Legyen NH a Hamilton-kört nem tartalmazó gráfok nyelve.

Tétel. *NH* $\in \mathrm{coNP}$.

Síkba rajzolható gráfok

Legyen L a síkba rajzolható gráfok nyelve

Tétel. $L \in \mathrm{NP}$

Bizonyítás: G tanúja egy síkbarajzolása.
Fáry-Wágner $\implies$ van olyan is, ami egyenes szakaszokat használ. Sőt olyan is van, hogy a koordináták nem túl nagyok.
$\implies$ Tanú a csúcsok koordinátái.
A bíró ellenőrzi, hogy az élek nem metszik egymást.

Tétel. $L \in \mathrm{coNP}$ *(* $\implies L \in \mathrm{NP} \cap \mathrm{coNP}$: *jól karakterizált)*

Bizonyítás: Van tanú a $G \notin L$ állításra is.
Vagy nem gráf vagy Kuratowski $\implies$ van benne vagy K_5-tel vagy $K_{3,3}$-mal topologikusan izomorf részgráf.
Tanú egy ilyen leírása, ezt a bíró könnyen ellenőrizheti.
Tétel. $L \in \mathrm{P}$

Sejtés: $H \notin \mathrm{coNP}$ és 3-SZÍN $\notin \mathrm{coNP}$

A prímszámok nyelve

Jelölje Π a (binárisan ábrázolt) prímszámok nyelvét.

Tétel. [V. R. Pratt, 1975]

$$\Pi \in \mathrm{NP} \cap \mathrm{coNP}.$$

Bizonyítás: $\Pi \in \mathrm{coNP}$: Ha egy szám nem prím, arra tanú egy osztója, pl. 6 nem prím, mert $2|6$.

$\Pi \in \mathrm{NP}$:
Lemma. *Legyen $p \ge 2$ egy egész szám. A p pontosan akkor prímszám, ha van olyan $1 \le g < p$ egész, melyre teljesülnek az alábbiak:*

1. $g^{p-1} \equiv 1 \pmod p$*,*

2. $g^{\frac{p-1}{r}} \not\equiv 1 \pmod p$ *minden r prímszámra, melyre $r|p-1$.*

Gyors hatványozás: a^m alakú hatvány legfeljebb $2\log_2 m$ szorzással kiszámítható.
$m = e_0 + e_1 2^1 + e_2 2^2 + \cdots + e_k 2^k$, $k \le \log_2 m$ és $e_j \in \{0,1\}$.
Ismételt négyzetre emelésekkel $\Longrightarrow a^{2^j} \Longrightarrow$ ez k szorzás
szorozzuk össze az a^{2^j} hatványokat azokra a j értékekre, melyekre $e_j = 1$
$\Longrightarrow a^m = a^{e_0+e_1 2^1+e_2 2^2+\cdots+e_k 2^k} = a^{e_0} a^{e_1 2^1} a^{e_2 2^2}\cdots a^{e_k 2^k}$
$\Longrightarrow k$ szorzás

a^m mérete $m+a$ méretében exponenciális $\Longrightarrow$ exponenciális alg.
$a^m \pmod n$ mérete legfeljebb $\log_2 n$ $\Longrightarrow$ ha mindig a maradékot vesszük
$\Longrightarrow$ polinomiális alg.

A p *prím állításra* tanú: g és $p-1$ egész $r_1,\dots,r_k$ prímosztói
a bíró gyors hatványozással ellenőrizheti, hogy a Lemma feltételei teljesülnek.
Azt is tanúsítani kell, hogy $r_1,\dots,r_k$ éppen a $p-1$ prímosztói (más nincs)
$\Longrightarrow$ prímtényezős felbontás $\surd$
és azt is, hogy $r_1,\dots,r_k$ prímek $\Longrightarrow$ rekurzívan $\surd$
Belátható, hogy a tanú össz mérete $O(n^2)$.

Tétel. [M. Agrawal, N. Kayal, N. Saxena, 2002] $\Pi \in \mathrm{P}$

Felismerés és keresés

Prím-e ↔ legkisebb prím osztója

$$F = \left\{ (a,c) \;\middle|\; \begin{array}{l} 1<c\le a \text{ egészek és van olyan } 1<b\le c \text{ egész,} \\ \text{melyre } b \text{ osztója } a\text{-nak} \end{array} \right\}.$$

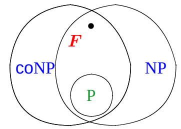
Tétel. $F \in \mathrm{NP}\cap\mathrm{coNP}$.

Bizonyítás: $(a,c)\in F$ $\Longrightarrow$ tanú egy jó b érték
$(a,c)\notin F$ $\Longrightarrow$ tanú → az $a = p_1^{\alpha_1}p_2^{\alpha_2}\cdots p_k^{\alpha_k}$ és a p_i számok prím-tulajdonságának tanúi

Legjobb ismert algoritmus a prím-felbontásra: D. Shanks $\Longrightarrow$ n bites inputon $c2^{n/4}$.

Tétel. *Ha $F\in\mathrm{P}$ igaz lenne, akkor {prímtényezős felbontás} $\in \mathrm{FP}$ is igaz lenne.*

Bizonyítás: Legyen E egy gyors alg. F felismerésére
$(a, a-1) \in F$?
ha nem $\rightarrow$ a prím $\surd$
ha igen $\Longrightarrow$ bináris kereséssel megkeressük a legkisebb olyan c-t amire $(a,c)\in F$ $\Longrightarrow$ $\le \log_2 a$ hívás
Utána a/c-re folytatjuk ...
$\Longrightarrow$ egy prím-osztó megtalálása: $O((\log a)^d)$
prím-osztók száma $\le \log a$ $\Longrightarrow$ összköltség $O((\log a)^{d+1})$



Karp-redukció

Mikor nem lényegesen nehezebb egy L_1 probléma egy L_2 problémánál?
$\Longrightarrow$ Ha L_2 felhasználásával meg lehet oldani L_1-et is.
$\Longrightarrow$ L_1 visszavezethető a L_2 problémára.
Definíció. *Az $f: I^*\to I^*$ leképezés az $L_1\subseteq I^*$ nyelv Karp-redukciója az $L_2\subseteq I^*$ nyelvre, ha*

1. Tetszőleges $x\in I^*$ szóra $x\in L_1$ pontosan akkor teljesül, ha $f(x)\in L_2$;

2. $f\in FP$, azaz f polinom időben számítható.

Jelölés: $L_1 \prec L_2$ ha L_1-nek van Karp-redukciója L_2-re.

Ha tehát van algoritmusunk L_2 eldöntésére $\Longrightarrow$ $x\in L_1$-re kiszámítjuk $f(x)$-et eldöntjük $f(x)\in L_2$? $\Longrightarrow$ tudjuk, hogy $x\in L_1$ igaz-e $\surd$

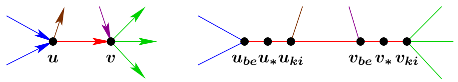
Ha tudnánk, hogy L nehéz és tudjuk, hogy $L\prec L'$ $\Longrightarrow$ L' is nehéz lenne
Ha L' könnyű lenne, és L nem lényegesen nehezebb nála, akkor L is könnyű.

Irányított Hamilton-kör probléma

Tétel. *IH ≺ H.*

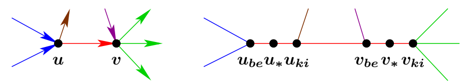
Bizonyítás: $G=(V,E)$ egy irányított gráf → $G'=(V',E')$ irányítatlan gráf hogy G' gyorsan megépíthető és
G-ben ∃ irányított Hamilton-kör ↔ G'-ben ∃ irányítatlan Hamilton-kör.

$$\begin{aligned} V' &= \{v_{be}, v_*, v_{ki} \mid v\in V\},\\ E' &= \{(v_{be},v_*),(v_*,v_{ki}) \mid v\in V\}\cup\{(u_{ki},v_{be}) \mid u\to v\in E\}.\end{aligned}$$



$v(G)=n, e(G)=e$ $\Longrightarrow$ $v(G')=3n, e(G')=2n+e$ $\Longrightarrow$ $(n+e)^c$ lépésben megkapható.

G-beli F irányított Hamilton-körének $\Longrightarrow$ G' egy F' Hamilton-köre



Az F egy $u\to v$ éle $\Longrightarrow$ az F'-ben az $u_* - u_{ki} - v_{be} - v_*$ út
$\Longrightarrow$ $G\in IH$ $\Longrightarrow$ $G'\in H$

Ha G'-ben van egy $F'\subseteq E'$ Hamilton-kör $\Longrightarrow$ egy u_*-ból indulva egy u_{ki} felé lépjünk először
$\Longrightarrow$ csak $u_* - u_{ki} - v_{be} - v_*$ alakú lehet utána $\Longrightarrow$ stb. $\Longrightarrow$ Ha $G'\in H$ akkor $G\in IH$.

A Karp-redukció felhasználása

Tétel.
1. Ha $L_1\prec L_2$ és $L_2\in\mathrm{P}$, akkor $L_1\in\mathrm{P}$.
2. Ha $L_1\prec L_2$ és $L_2\in\mathrm{NP}$ akkor $L_1\in\mathrm{NP}$.
3. Ha $L_1\prec L_2$, akkor $\overline{L}_1\prec\overline{L}_2$, ahol $\overline{L}_i = I^*\setminus L_i$.
4. Ha $L_1\prec L_2$ és $L_2\in\mathrm{coNP}$, akkor $L_1\in\mathrm{coNP}$.
5. Ha $L_1\prec L_2$ és $L_2\in\mathrm{NP}\cap\mathrm{coNP}$, akkor $L_1\in\mathrm{NP}\cap\mathrm{coNP}$.
6. Ha $L_1\prec L_2$ és $L_2\prec L_3$, akkor $L_1\prec L_3$.

Bizonyítás:

Legyen $f: I^*\to I^*$ az L_1 Karp-redukciója L_2-re, $f\in FTIME(n^k)$.
$x\in I^*$ egy input szót, melyre szeretnénk eldönteni, hogy $x\in L_1$ teljesül-e, n az x hossza.

1. Kiszámítjuk $f(x)$-et $\Longrightarrow$ időigénye $\le c_1 n^k$ $\Longrightarrow$ $|f(x)|\le c_1 n^k$
L_2 felismerő algoritmusával eldöntjük, hogy $f(x)\in L_2$ igaz-e
$\Longrightarrow$ időigénye $\le c_2(c_1 n^k)^l$
$x\in L_1 \leftrightarrow f(x)\in L_2$ $\Longrightarrow$ összidő $O(n^{kl})$ $\surd$

2.: Ha $L_1\prec L_2$ és $L_2\in\mathrm{NP}$ akkor $L_1\in\mathrm{NP}$: Az $f(x)\in L_2$ tény egy y tanúja jó $x\in L_1$ tanújának is, és az L_2-höz tartozó bíró egy kis módosítással jó lesz az L_1 bírójának is
$|y|\le|f(x)|^c$ $\Longrightarrow$ $|y|\le c_1^c|x|^{kc}$
Az L_1 bírója az $(x,y)\to f(x)$ $\Longrightarrow$ $(f(x),y)\to L_2$ bírájának
L_1 bírója pontosan akkor fogadja el az (x,y) párt ↔ L_2 bírója elfogadja az $(f(x),y)$ párt

3.: Ha $L_1\prec L_2$, akkor $\overline{L}_1\prec\overline{L}_2$: Mint 1. hiszen $x\in I^*$ szóra $x\notin L_1$ ↔ $f(x)\notin L_2$.

4.: Ha $L_1\prec L_2$ és $L_2\in\mathrm{coNP}$, akkor $L_1\in\mathrm{coNP}$: $\Longleftarrow$ 2.,3.

5.: Ha $L_1\prec L_2$ és $L_2\in\mathrm{NP}\cap\mathrm{coNP}$, akkor $L_1\in\mathrm{NP}\cap\mathrm{coNP}$.: $\Longleftarrow$ 2.,4.

6.: Ha $L_1\prec L_2$ és $L_2\prec L_3$, akkor $L_1\prec L_3$: Legyen f az $L_1\prec L_2$ függvénye, ami $O(x^k)$ időben számolható
és g az $L_2\prec L_3$ függvénye, ami $O(x^l)$ időben számolható
Az $L_1\prec L_3$ függvénye $g(f(x))$ lesz, ami $O((x^k)^l)=O(x^{kl})$ időben számolható

NP-teljes nyelvek

Definíció. *Az $L\subseteq I^*$ nyelv NP-teljes, ha*

1. $L\in\mathrm{NP}$,

2. tetszőleges (azaz minden) $L'\in\mathrm{NP}$ nyelv esetén létezik $L'\prec L$ Karp-redukció.

Egy NP-teljes nyelv tehát legalább olyan nehéz, mint bármely más NP-beli nyelv.

Ha egy ilyen nyelvről kiderülne, hogy P-beli (coNP-beli), akkor ugyanez igaz lenne minden NP-beli nyelvre.

Van-e NP-teljes nyelv?

Cook–Levin-tétel

Boole-formula:
pl. $(x_1 \vee \overline{x_2} \vee x_5) \wedge (\overline{x}_3 \vee x_2 \vee x_6 \vee x_1) \wedge \overline{(x_5 \vee x_6)}$

Definíció. *SAT nyelv: a kielégíthető Boole-formulák nyelve.*
Tétel. *[S. A. Cook, L. Levin, 1971]*
A SAT nyelv NP*-teljes.*

Bizonyítás: SAT $\in NP$, mert egy kielégítés (értékadás a változóknak) megfelelő tanú $\surd$

Be kell látni, hogy $\forall L \in \mathrm{NP}$ nyelvre létezik egy $L \prec$ SAT Karp-redukció
$\Longrightarrow$ $(x \in L?)$ kérdés tetszőleges $x \in I^*$ inputjához meg kell adnunk egy ϕ Boole-formulát, mely pontosan akkor kielégíthető, ha $x \in L$.
$\Longrightarrow$ tanú-tétel miatt elég leírni Boole-formulával az (x, y)-t felismerő TG-t

$$0x[i,j] = \begin{cases} 1 & \text{ha az } i\text{-edik lépés után a } j\text{-edik cella tartalma 0} \\ 0 & \text{különben} \end{cases}$$

$$1x[i,j] = \begin{cases} 1 & \text{ha az } i\text{-edik lépés után a } j\text{-edik cella tartalma 1} \\ 0 & \text{különben} \end{cases}$$

$$\ddot{u}x[i,j] = \begin{cases} 1 & \text{ha az } i\text{-edik lépés után a } j\text{-edik cella tartalma ü} \\ 0 & \text{különben} \end{cases}$$

$$f[i,j] = \begin{cases} 1 & \text{ha az } i\text{-edik lépés után a fej } j\text{-edik cellán áll} \\ 0 & \text{különben} \end{cases}$$

$$q[i,s] = \begin{cases} 1 & \text{ha az } i\text{-edik lépés után } M \text{ belső állapota } q_s \\ 0 & \text{különben.} \end{cases}$$

Le kell írni $\Longrightarrow$ a szalag egy mezőjén minden időpontban éppen egy szalagjel van; a fej minden időpontban a szalag egyetlen celláján van; a gép minden időpontban egyetlen állapotban van.
pl. az első: $(0 \le i \le n^c, 1 \le j \le n^c)$ párra

$$(0x[i,j] \vee 1x[i,j] \vee \ddot{u}x[i,j]) \wedge$$
$$\wedge (\overline{0x[i,j]} \vee \overline{1x[i,j]}) \wedge (\overline{0x[i,j]} \vee \overline{\ddot{u}x[i,j]}) \wedge (\overline{1x[i,j]} \vee \overline{\ddot{u}x[i,j]}).$$

Összesen $(n^c + 1)n^c$ ilyen formula

Átmenet függvény leírása: pl. $\delta(q_s, 1) = (q_k, 0, bal) \Longrightarrow$

$$\Big((q[i,s] \wedge 1x[i,l] \wedge f[i,l]) \longrightarrow (q[i+1,k] \wedge 0x[i+1,l] \wedge f[i+1,l-1])\Big)$$

$O(n^{2c})$ ilyen formula

Ahol nincs fej, nincs változás:

$$\overline{f[i,l]} \longrightarrow (0x[i,l] \leftrightarrow 0x[i+1,l])$$

$O(n^{2c})$ ilyen formula (1-re és ü-re is)

Kezdő helyzet:
$$q[0,0] \wedge f[0,1]$$

Input leírása:
$$0x[0,1] \wedge 1x[0,2] \wedge 0x[0,3] \ldots$$

Elfogadó állapot:
$$1x[n^c, 1]$$

Minden ilyen $i = 0, 1, 2, \ldots, n^c$-re ÉS-sel
Szabadsági fok: a súgás helyén nincs megkötve semmi

Ez a Boole formula akkor és csak akkor kielégíthető, ha van megfelelő súgás x-hez. $\surd$

További NP-teljes feladatok

Tétel. *Ha az L_1 nyelv* NP*-teljes, $L_2 \in$ NP és $L_1 \prec L_2$, akkor L_2 is* NP*-teljes.*

Bizonyítás: Láttuk, hogy a Karp-redukció tranzitív
Ha $L_1 \prec L_2$ és $L' \prec L_1$ $\forall L' \in$ NP-teljes
$\Longrightarrow L' \prec L_2$ $\forall L' \in$ NP-teljes $\surd$

Nem kell már *minden* NP-beli nyelvet az L_2-re redukálni; elég ezt megtenni *egyetlen* NP-teljes L_1 nyelvvel.

Definíció. *Ha az L_2 nyelvről csak azt tudjuk, hogy van olyan* NP*-teljes L_1 nyelv, melyre $L_1 \prec L_2$, akkor L_2-t* NP-nehéz *nyelvnek nevezzük. Az előző állítás szerint L_2 pontosan akkor* NP*-teljes, ha* NP*-beli és ugyanakkor* NP*-nehéz is.*

16. előadás

Algoritmuselmélet 16. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Április 29.

Konjunktív normálforma

Konjunktív normál forma:
$(\overline{x}_1 \vee x_2 \vee x_3) \wedge (x_3 \vee x_4 \vee \overline{x}_7 \vee \overline{x}_{18}) \wedge (x_9 \vee \overline{x}_{10} \vee \overline{x}_{13}) \wedge \ldots$

Tétel. *Minden Boole-formulának létezik konjuktív normál formája.*

k-konjunktív normál forma: ha $\phi = \phi_1 \wedge \cdots \wedge \phi_n$, akkor minden ϕ_i-ben legfeljebb k literál szerepel. Pl.: 4-CNF:
$(\overline{x}_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_4 \vee \overline{x}_7 \vee \overline{x}_{18}) \wedge (x_9 \vee \overline{x}_{10} \vee \overline{x}_{13})$.

Definíció. *Jelölje k-SAT a kielégíthető k-CNF-ekből álló nyelvet.*

k-SAT $\in$ NP $\Longleftarrow$ tanú egy kielégítés

Tétel. *2-SAT $\in$ P.*

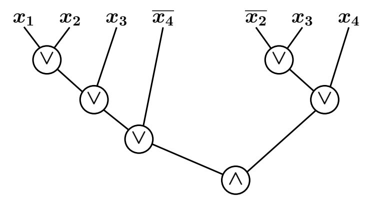
Tétel. *3-SAT* NP*-teljes.*

Tétel. *3-SAT* NP*-teljes.*

Bizonyítás: Belátjuk, hogy SAT $\prec$ 3-SAT.
Tetszőleges ϕ formulához kell adni egy ψ-t ami 3-CNF, ekvivalens ϕ-vel és polinom időben számolható.

Kifejezésfa:

$$(x_1 \vee x_2 \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_2} \vee x_3 \vee x_4)$$



A kifejezésfa minden csúcsához rendeljünk egy új változót: z_i
és egy ϕ_i Boole-formulát, ami megadja z_i értékét:
$u \leftrightarrow v = (u \vee \overline{v}) \wedge (\overline{u} \vee v)$

$$\begin{aligned} z_i = z_j \wedge z_k &= z_i \leftrightarrow (z_j \wedge z_k) = \\ &= (z_i \vee \overline{(z_j \wedge z_k)}) \wedge (\overline{z_i} \vee (z_j \wedge z_k)) = \\ &= (z_i \vee \overline{z_j} \vee \overline{z_k}) \wedge (\overline{z_i} \vee z_j) \wedge (\overline{z_i} \vee z_k) \end{aligned}$$

$$\begin{aligned} z_i = z_j \vee z_k &= z_i \leftrightarrow (z_j \vee z_k) = \\ &= (z_i \vee \overline{(z_j \vee z_k)}) \wedge (\overline{z_i} \vee (z_j \vee z_k)) = \\ &= (z_i \vee \overline{z_j}) \wedge (z_i \vee \overline{z_k}) \wedge (\overline{z_i} \vee z_j \vee z_k) \end{aligned}$$

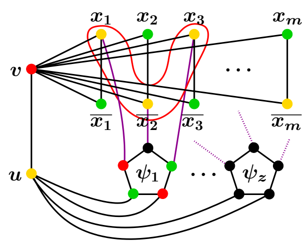
$$\psi = (\wedge \phi_i) \wedge z_m$$

Ez ekvivalens ϕ-vel, és polinom időben számolható. $\surd$

3-SZÍN

Tétel. *A 3-SZÍN nyelv* NP*-teljes.*

Bizonyítás: Már láttuk, hogy $\in$ NP, beláttuk, hogy 3-SAT $\prec$ 3-SZÍN.
Tetszőleges ψ 3-CNF-hez építenünk kell egy G gráfot úgy, hogy $\psi \in$ 3-SAT pont akkor igaz, ha $G \in$ 3-SZÍN.



Zöld igen $\Longrightarrow$ minden ψ_i-ben van zöld.

Maximális méretű független pontrendszer gráfokban

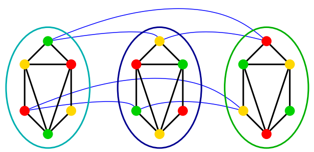
$$\text{MAXFTLEN} = \left\{ (G,k) \;\middle|\; \begin{array}{l} G \text{ egy gráf, } k \in \mathbb{Z}^+, \text{ és} \\ G\text{-nek van } k \text{ elemű független csúcshalmaza} \end{array} \right\}.$$

Tétel. *A MAXFTLEN nyelv* NP*-teljes.*

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

Megadunk egy 3-SZÍN $\prec$ MAXFTLEN Karp-redukciót, $G \to G_1$

$$G \in \text{3-SZÍN pontosan akkor, ha } (G_1, k) \in \text{MAXFTLEN}. \qquad (*)$$



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

$|V(G_1)| = 3|V(G)|$ és $|E(G_1)| = 3|V(G)| + 3|E(G)|$,
legyen $k = |V(G)|$.

Ha G színezhető 3 színnel $\Longrightarrow$ legyen H a piros pontok halmaza G_1-ben. $\surd$

Ha G_1-ben van $|V(G)|$ független $\Longrightarrow$ legyen egy pont színe olyan, hogy melyik $G^{(i)}$-ben van. $\surd$

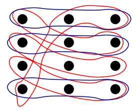
MAXKLIKK $= \{(G,k) |\ G$-ben van k pontú teljes részgráf$\}$.

Tétel. *A MAXKLIKK nyelv* NP*-teljes.*

Bizonyítás: $G \leftrightarrow \overline{G}$ $\surd$

A 3 dimenziós házasítás

Párosítási feladat általánosítása: Legyenek U_1, U_2, U_3 azonos méretű véges halmazok $\Longrightarrow |U_i| = t$.
Adott még $U_1 \times U_2 \times U_3$ valamely S részhalmaza $\Longrightarrow (u_1, u_2, u_3)$ alakú hármasok
Kiválasztható-e S-ből egy *3 dimenziós házasítás*?
$\Longrightarrow$ olyan t-elemű $S' \subseteq S$ részhalmaz, mely minden U_i-beli pontot lefed.



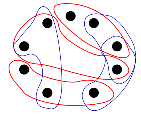
U_1 U_2 U_3

3-DH: olyan $U_1, U_2, U_3;\ S \subseteq U_1 \times U_2 \times U_3$ rendszerek, melyeknél S-ből kiválasztható egy 3 dimenziós házasítás.
Tétel. *A 3-DH feladat* NP*-teljes.*

Bizonyítás: 3-DH $\in$ NP $\surd$ $\exists$ 3-SAT $\prec$ 3-DH

X3C

Pontos fedés hármasokkal: adott egy U véges halmaz, és U háromelemű részhalmazainak egy $\mathcal{F} = \{X_1, X_2, \ldots, X_k\}$ családja.
Eldöntendő, hogy az $\mathcal{F}$-ből kiválaszthatók-e páronként diszjunkt halmazok, melyek együttesen lefedik U-t.
Jelölje X3C azokat az $(U, \mathcal{F})$ párokat, melyekre igen.



Tétel. *Az X3C nyelv* NP*-teljes.*

Bizonyítás: X3C $\in$ NP teljesül $\Longrightarrow$ tanú egy pontos fedés.
Megmutatjuk, hogy 3-DH $\prec$ X3C.
X3C általánosabb probléma, mint 3-DH. Ha van algoritmus az általánosra, akkor avval a speciális is megoldható. $\surd$

Ha L NP-teljes és L' általánosítása L-nek, akkor L' is NP-teljes.

Utazó ügynök probléma

Tétel. *Az IH nyelv* NP*-teljes.*
Tétel. *Az H nyelv* NP*-teljes.*

Bizonyítás: Már láttuk, hogy H $\in$ NP $\surd$
és láttuk, hogy IH $\prec$ H $\surd$

Utazó ügynök feladat:
Adott egy G irányítatlan gráf pozitív egészekkel súlyozott élekkel.
A cél minél rövidebb összsúlyú Hamilton-kört találni G-ben.
$\Longrightarrow Ut$ nyelv $\Longrightarrow$ olyan (G,k) párokból áll, melyekben G egy súlyozott élű irányítatlan gráf, k egy nemnegatív egész, és G-ben van k-nál nem nagyobb súlyú Hamilton-kör.

Tétel. *Az Ut nyelv* NP*-teljes.*

Bizonyítás: H általánosítása $\Longleftarrow$ minden él súlya 1 és $k = |V(G)|$ $\surd$

A Hátizsák feladat

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

$\Longrightarrow$ Input: $s_1, \ldots, s_m; v_1 \ldots, v_m; b; k$

$$\textit{Hát} = \{(s_1, s_2, \ldots, s_m; v_1, v_2, \ldots, v_m; b; k) | s_i, b, k > 0 \text{ egészek, és}$$
$$\text{van olyan } I \subseteq \{1, \ldots, m\} \text{ melyre } \sum_{i \in I} s_i \leq b \text{ és } \sum_{i \in I} v_i \geq k\}.$$

Vegyük azt a speciális esetet, amikor $s_i = v_i$ és $b = k$:
Részhalmazösszeg probléma:

$$\text{RH} = \left\{ (s_1, \ldots, s_m; b) \;\middle|\; \begin{array}{l} s_i, b > 0 \text{ egészek,} \\ \text{és van olyan } I \subseteq \{1, \ldots, m\} \text{ hogy } \sum_{i \in I} s_i = b \end{array} \right\}.$$

Tétel. *Az RH nyelv* NP*-teljes.*

Bizonyítás: $RH \in$ NP $\surd$
X3C $\prec$ RH

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

$$\textit{Partíció} = \left\{ (s_1, \ldots, s_m) \;\middle|\; \begin{array}{l} s_i > 0 \text{ egészek, és van olyan} \\ I \subseteq \{1, \ldots, m\}, \text{ hogy } \sum_{i \in I} s_i = \frac{1}{2} \sum_{i=1}^{m} s_i \end{array} \right\}.$$

Tétel. *A Partíció nyelv* NP*-teljes.*

Bizonyítás: *Partíció* $\in$ NP $\surd$
Belátjuk, hogy RH $\prec$*Partíció* RH általánosabb!
Vegyük az RH egy $x = (s_1, \ldots, s_m; b)$ inputját.
$\Longrightarrow$ Feltehető, hogy $b \leq s = \sum_{i=1}^{m} s_i$.
$x \to$ *Partíció* inputja: $(s_1, \ldots, s_m, s + 1 - b, b + 1)$.
A számok összege $2s + 2$, az utolsó két szám nem lehet egy partíció ugyanazon osztályában, mert az összegük túl nagy: $s + 2 > \frac{1}{2}(2s + 2)$.

17. előadás

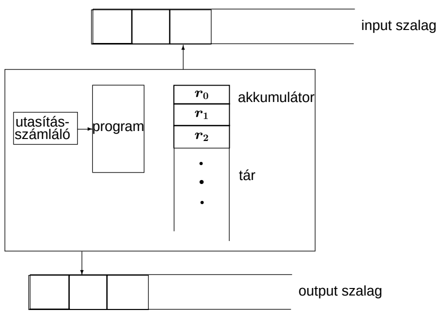
Algoritmuselmélet 17. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Május 6.

A közvetlen elérésű gép (RAM)

Random Access Machine
$\Longrightarrow$ Memória minden cellája egy lépésben elérhető.



Utasítások $\Longrightarrow$

Aritmetika		*Adatmozgatás*		*Vezérlés*	
ADD	*op*	LOAD	*op*	JUMP	*címke*
SUB	*op*	STORE	*op*	JGTZ	*címke*
MULT	*op*	READ	*op*	JZERO	*címke*
DIV	*op*	WRITE	*op*	HALT	

op
$\Longrightarrow$ $=i$: az i számot jelenti
$\Longrightarrow$ i: i sorszámú cella tartalma
$\Longrightarrow$ $*i$: az i sorszámú cellában levő sorszámú cella tartalma, pl. $r[0]=-4$ és $r[1]=0$ $\Longrightarrow$ a $*1$ operandus jelentése -4

Az aritmetikai műveleteknél az első argumentum az akkumulátor, a második az op
$\Longrightarrow$ eredménye $\longrightarrow$ akkumulátor

Például ha $r[0]=17$ és $r[1]=2$, akkor DIV 1 hatására a $8=\lfloor 17/2\rfloor$ érték kerül az akkumulátorba.

Aritmetika		*Adatmozgatás*		*Vezérlés*	
ADD	*op*	LOAD	*op*	JUMP	*címke*
SUB	*op*	STORE	*op*	JGTZ	*címke*
MULT	*op*	READ	*op*	JZERO	*címke*
DIV	*op*	WRITE	*op*	HALT	

A READ beolvassa az input cella tartalmát op-ba és lép
A WRITE kiírja az op-ot az output cellába és lép

A STORE op $\Longrightarrow$ akkumulátor tartalma $\longrightarrow$ op
LOAD fordítva.

HALT $\Longrightarrow$ megáll.
JZERO: Ha $r[0]=0$, akkor ugrik
JGTZ: Ha $r[0]\geq 0$, akkor ugrik.

Költségszámítás

Uniform költség: végrehajtott utasítások száma
Pl. az $f(n)=3^{2^n}$ függvény kiszámítható $O(n)$ uniform költséggel:
$\Longrightarrow$ legyen $x:=3$, majd n-szer iterálva $x:=x^2$.
De a k-adik iterációs lépésben két 2^k-jegyű számot szorzunk össze.
$\Longrightarrow$ Az eredménynek tehát 2^{n+1} jegye lesz. $\lightning$

Logaritmikus költség: egy utasítás költsége a benne szereplő adatok összhossza. Egy program logaritmikus költsége a végrehajtott utasítások költségeinek az összege.

Példa: ADD $*1$ uniform költsége 1. A logaritmikus költsége viszont

$$\text{hossz}(r[0])+\text{hossz}(r[1])+\text{hossz}(r[r[1]])+\text{hossz}(1),$$

ahol $r[i]$ a belső tár i-edik cellájának a tartalmát jelöli.

Ha nincs MULT és DIV akkor a logaritmikus költség valós költség
Legjobb ismert algoritmus $O(n\log n\log\log n)$

Ha tudjuk, hogy a RAM-program végig $\leq l$ hosszú szavakkal dolgozik $\Longrightarrow$ m uniform költség $\longrightarrow$ $O(lm)$ logaritmikus

Szimulációk

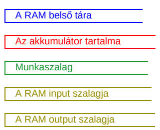
Tétel. [Turing-gép$\leftrightarrow$RAM szimulációk]
(1) Tetszőleges M Turing-gép szimulálható $O(T_M(n)\log T_M(n))$ logaritmikus költségű RAM programmal. A szimuláció uniform költsége $O(T_M(n))$.
(2) Egy $t(n)$ logaritmikus költségű, MULT és DIV utasításokat nem tartalmazó RAM-program szimulálható olyan N Turing-géppel, melynek az időigényére $T_N(n)=O(t^2(n))$ teljesül.

Bizonyítás: (1) M egy k-szalagos Turing-gép.
M bemenete $\Longrightarrow$ RAM input szalag
A RAM belső tárának első c celláját munkaterületként használjuk $\Longrightarrow$ M belső állapota, és itt kap helyet az a k cella is, amelyekben a M fejeinek helyzete van
$\Longrightarrow$ *összefésülve* az M szalagjainak tartalmát

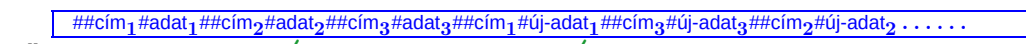
A RAM programja $\Longrightarrow$ M átmeneteinek leírása $\Longrightarrow$ megvalósítható **if...then...** utasításokkal (indirekt címzés)

uniform költség: $O(T_M(n))$
logaritmikus költség: az M szalagjelei és belső állapotai (M-től függő) konstans hosszúságúak
a fejek helyét leíró mutatók pedig $O(\log T_M(n))$ bittel ábrázolhatók
$\Longrightarrow$ $O(T_M(n)\log T_M(n))$

(2) A RAM-programot szimuláló N gép szalagjelei $\Longrightarrow$ $\{0,1,+,-,\#,\ddot{u}\}$.



Első szalag:



Belső tár olvasása $\surd$ írása (végére) $\surd$
RAM alaputasítások $\Longrightarrow$ N állapotcsoportjai $\Longrightarrow$ ezek között átmenet

Lépésszám: az alaputasítások – a MULT és DIV kivételével – megvalósíthatók úgy, hogy N lépésszáma az utasításban szereplő adatok hosszával plusz az első szalag érdemi részének hosszával arányos legyen.
$\Longrightarrow$ egy lépés szimulációjának a költsége $O(t(n))$
összköltség: $t(n)O(t(n))=O(t^2(n))$

Ha MULT és DIV is van benne: $O(t^3(n))$

Elágazás és korlátozás

Mit tegyünk ha kiderül, hogy a megoldandó probléma NP-teljes?

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

Bontsunk esetekre, azt alesetekre, ... $\Longrightarrow$ fa
Értékeljük az eseteket $\Longrightarrow$ bizonyos irányokban nem kell továbbmenni.
$\Longrightarrow$ (korlátozó heurisztika)

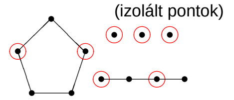
Pl. sakkállások

Feladat: Keressünk maximális méretű független ponthalmazt egy adott G gráfban.

NP-teljes
Minden részhalmazt végignézünk $\Longrightarrow$ $O(2^n)$ lépés

Jobb algoritmus

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



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

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

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

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

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

$\checkmark$

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

3-színezés keresése

Feladat: Adott G, keressünk egy 3-színezést.

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

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

Összköltség: $O(2^n n^c)$.

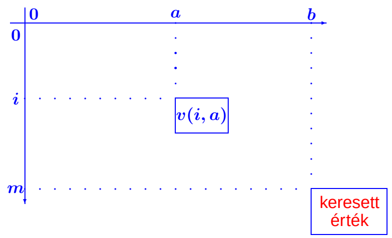
Dinamikus programozás

Táblázat kitöltése soronként, rekurziót használva.
Pascal háromszög, Bellman-Ford, Floyd

A Hátizsák probléma: Adottak az $s_1, \ldots, s_m$ súlyok, a b súlykorlát, a $v_1, \ldots, v_m$ értékek és a k értékkorlát. A kérdés, hogy van-e olyan $I \subseteq \{1, \ldots, m\}$ részhalmaz, melyre teljesül, hogy $\sum_{i \in I} s_i \leq b$ és $\sum_{i \in I} v_i \geq k$.

NP-teljes

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



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

$\Longrightarrow$ Soronként kitölthető $\Longleftarrow$ minden érték két felette levőből számolható.

Összköltség: $O(bL)$ nem polinomiális, mert b-től függ, nem $\log b$-től!

Definíció. *A b egész unáris ábrázolása: $0^b := 0 \cdots 0$ (összesen b darab 0 egymás után).*

Definíció. *Egy feladat egy egész input paramétere apró, ha unárisan számítjuk bele az input hosszába.*

Tétel. *A Hátizsák probléma apró súlykorlát esetén megoldható polinom időben.*

18. előadás

Algoritmuselmélet 18. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

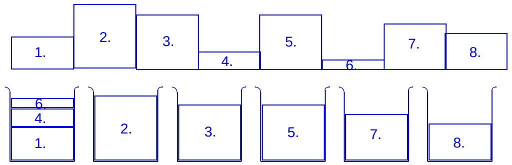
2002 Május 7.

Közelítő algoritmusok

Hátha nem szükséges pontos megoldás, elég az optimumtól nem túl messze levő is.

Ládapakolás: Adottak az $s_1, \dots, s_m$ (racionális) súlyok, $0 \le s_i \le 1$. A cél a súlyok elhelyezése minél kevesebb 1 súlykapacitású ládába.
NP-teljes

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



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

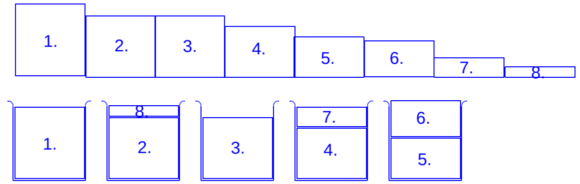
Bizonyítás: $\lceil \sum_{i=1}^{m} s_i \rceil \le OPT(I)$
$FF(I) \le \lceil 2 \sum_{i=1}^{m} s_i \rceil \Longleftarrow$ nincs két olyan láda, ami nincs félig kitöltve.
Felhasználjuk, hogy $\lceil 2x \rceil \le 2\lceil x \rceil$

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

$\surd$

Tétel. [D. S. Johnson és munkatársai, 1976]
A probléma tetszőleges I inputjára teljesül, hogy $FF(I) \le \lceil 1.7OPT(I) \rceil$. Továbbá vannak tetszőlegesen nagy méretű I inputok, melyekre $FF(I) \ge 1.7(OPT(I) - 1)$.

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



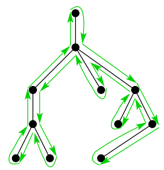
Tétel. [D. S. Johnson, 1973]
Tetszőleges I inputra teljesül, hogy $FFD(I) \le \frac{11}{9} OPT(I) + 4$, és tetszőlegesen nagy méretű I inputok vannak, melyekre $FFD(I) \ge \frac{11}{9} OPT(I)$. ($\frac{11}{9} = 1.222\dots$)

Tétel. [F. de la Vega, G. S. Lueker]
Tetszőleges $\varepsilon > 0$-hoz van olyan P lineáris algoritmus, amire $P(I) \le (1 + \varepsilon)OPT(I)$.

Euklideszi utazó ügynök probléma

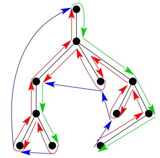
Az n pontú K_n teljes gráf élein adott a nemnegatív értékű d súlyfüggvény. Erre teljesül a háromszög-egyenlőtlenség: tetszőleges különböző u, v, w csúcsokra érvényes a $d(u, w) \le d(u, v) + d(v, w)$ egyenlőtlenség (az euklideszi feltétel).
A cél egy minimális összsúlyú Hamilton-kör keresése.

Keresünk egy minimális összsúlyú feszítőfát (pl. Kruskal). $\Longrightarrow$



Ennek összsúlya legyen s $\Longrightarrow$ Euler séta hossza $2s$.

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



Ha az optimális Hamilton-körből elhagyunk egy élet $\Longrightarrow$ egy legalább s súlyú feszítőfa
A módszer legfeljebb 2-szer akkora utat ad, mint az optimális.

JAVA animáció: TSP

Véletlent használó módszerek

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

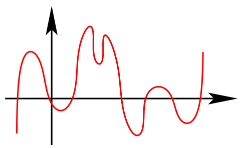
Probléma: Adott behelyettesítéssel (**fekete dobozzal**) egy n-változós $f \in \mathbb{Z}[x_1, \dots, x_n]$ egész együtthatós polinom. Tudjuk, hogy $\deg f \le d$. El akarjuk dönteni, hogy f azonosan nulla-e.

Példa: $f(x_1, x_2, \dots, x_{2n}) = (x_1 + x_2)(x_3 + x_4) \cdots (x_{2n-1} + x_{2n})$

$$D = \det \begin{pmatrix} x_{11} & \dots & x_{1n} \\ \vdots & & \vdots \\ x_{n1} & \dots & x_{nn} \end{pmatrix}$$

Mminél hamarabb szeretnénk találni egy $\alpha = (\alpha_1, \dots, \alpha_n)$-t, amire $f(\alpha) \ne 0$.

Pl. egy változóra jól látszik



Tétel. **[J. Schwartz lemmája]** *Ha $\deg f \le d$, és $\alpha_1, \dots, \alpha_n$ egyenletes eloszlású, egymástól független véletlen elemei az $\{1, \dots, N\}$ számhalmaznak, akkor $f \not\equiv 0$ esetén $Prob(f(\alpha) = 0) \le \frac{d}{N}$.*

Tétel. *Az $\{1, 2, \dots, 2d\}$ halmazból vett véletlen n-komponensű α vektor esetén $Prob(f(\alpha) \ne 0) \ge 1/2$, ha $f \not\equiv 0$. Ekkora halmazból választva tehát legalább $1/2$ valószínűséggel adódik tanú. Ha t-szer függetlenül választunk ilyen helyettesítést, akkor legalább $1 - \frac{1}{2^t}$ valószínűséggel kapunk tanút.*

Alkalmazás

Randomizált módszer teljes párosítás keresésére páros gráfban.

Legyen $G = (L, U; E)$ páros gráf, $L = \{l_1, \dots, l_n\}$ és $U = \{u_1, \dots, u_n\}$
$M = (m_{ij})$ n-szer n-es mátrix $\Longrightarrow$

$$m_{ij} = \begin{cases} x_{ij} & \text{ha } (l_i, u_j) \in E, \\ 0 & \text{különben.} \end{cases}$$

Tétel. *G-ben akkor és csak akkor van teljes párosítás ha $\det M \ne 0$.*

Bizonyítás: A determináns egy tagja $\Longrightarrow \pm m_{1\pi(1)} m_{2\pi(2)} \cdots m_{n\pi(n)}$
Ha nem 0 $\Longrightarrow (l_i, u_{\pi(i)}) \in E, i = 1, \dots, n,$ $\Longrightarrow$ teljes párosítás
Ha tehát G-ben nincs teljes párosítás, $\Longrightarrow \det M = 0$.
Ha viszont van G-ben teljes párosítás $\Longrightarrow \exists$ nem 0 kifejtési tag
nem ejthetik ki egymást, mert bármely kettőben van két különböző változó.

Az előző módszerrel eldönthetjük, hogy $\det M = 0$ igaz-e.

Hasonlóan megy nem páros gráfokra is.

Prímtesztelés

Bemenő adatként adott (binárisan) egy m páratlan egész; szeretnénk eldönteni, hogy m prímszám-e.

Fermat-teszt (m)
1. Válasszunk egy véletlen a egészet az $[1, m)$ intervallumból.
2. Ha $a^{m-1} \equiv 1 \pmod m$, akkor a válasz „m valószínűleg prím”, különben a válasz „m összetett”.

gyors hatványozással ez gyorsan végrehajtható

Ha azt kapjuk, hogy „m összetett” $\Longrightarrow$ ez biztos igaz

Pl.: $m = 21 = 7 \cdot 3$ és $a = 2$ $\Longrightarrow a$ az m Fermat-tanúja, hiszen $2^{20} \equiv 4 \pmod{21}$.

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

Bizonyítás: Legyen a tanú $\Longrightarrow a^{m-1} \not\equiv 1 \pmod m$ és $c_1, c_2, \ldots, c_s$ nem tanúk $\Longrightarrow c_i^{m-1} \equiv 1 \pmod m$
Feltehetjük, hogy a, c_i relatív prímek m-hez.
$\Longrightarrow (ac_i)^{m-1} \equiv a^{m-1}c_i^{m-1} \equiv a^{m-1} \not\equiv 1 \pmod m \Longrightarrow ac_i$ tanú
ac_i mind különbözőek lesznek $\Longrightarrow$ legalább annyi tanú, mint nem tanú $\surd$

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

Rabin-Miller teszt

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

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

számok egyike sem osztható m-mel.

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

Bizonyítás:

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

m prím $\Longrightarrow$ a kis Fermat-tétel szerint m osztja a bal oldalt.
$\Longrightarrow m$ osztja a jobb oldal valamelyik tényezőjét $\Longrightarrow a$ nem Rabin–Miller-tanú.
Tétel. *Ha m összetett, akkor az $1 \le a < m$ feltételt teljesítő a egészeknek legalább a fele Rabin–Miller-tanú.*

Rabin-Miller teszt

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

Kolmogorov-bonyolultság

Milyen röviden lehet leírni valamit?
Tekintsük azokat a természetes számokat, amelyeket magyar nyelven legfeljebb 100 billentyű-leütéssel definiálni lehet.
A billentyűk száma véges $\Longrightarrow$ ezen számok halmaza is véges $\Longrightarrow$ Van tehát egy legkisebb természetes szám, amit nem lehet definiálni a fenti módon.
$\Longrightarrow$ *az a legkisebb szám, amely nem definiálható magyar nyelven legfeljebb száz billentyű-leütéssel* $\Longleftarrow$ egy száznál rövidebb jelsorozat $\lightning$
írógép-paradoxon

$n = 2^k - 10$ alakú számok bináris kódja $k = \log_2 n$ hosszú
Viszont a $2^k - 10$ kifejezés hossza csak $\log_2 \log_2 n +$ konstans.

Rögzítsünk egy U univerzális Turing-gépet, és értelmezzük az $x \in I^*$ szó bonyolultságát mint a legrövidebb $y\#z$ input szó hosszát, melyre U az x szót számítja ki.
Az U gép választásától nagy mértékben független, és aszimptotikus értelemben jó közelítését adja az „optimumnak”.

Definíció. *Legyen M egy TG ami az $f_M : I^* \to I^*$ parciális függvényt számolja ki. Jelöljük $C_M(x)$-szel annak a legrövidebb bemenő szónak a hosszát, mellyel elindítva M az x szót adja eredményül:*

$$C_M(x) = \begin{cases} \min\{|y| :\ y \in I^*,\ f_M(y) = x\} & \text{ha ilyen } y \text{ létezik,} \\ \infty & \text{különben.} \end{cases}$$

A $C_M(x)$ szám méri, hogy x mennyire nyomható össze akkor, ha a kibontást, vagyis az összenyomott szó visszafejtését az M algoritmus végzi.
konkrét x-re $\Longrightarrow \exists\, M_1$ gép, hogy $C_{M_1}(x) = 0$
és $\exists\, M_2$ gép, hogy $C_{M_2}(x) = \infty$.

Tétel. [invariancia-tétel] *Legyen U egy univerzális Turing-gép. Ekkor tetszőleges M Turing-gépre létezik egy (csak M-től függő) $c_M \in \mathbb{Z}^+$ állandó, mellyel minden $x \in I^*$ szóra teljesül a következő egyenlőtlenség:*

$$C_U(x) \le C_M(x) + c_M.$$

Bizonyítás: M gép leírása $\Longrightarrow w \in I^*$
legyen y egy legrövidebb szó, amiből M az x-et bontja ki:
$\Longrightarrow y \in I^*$, $f_M(y) = x$, és $|y| = C_M(x)$
$\Longrightarrow U$ a $w\#y$ bemeneten x-et adja eredményül $\Longrightarrow$

$$C_U(x) \le |w\#y| = |w\#| + |y| = |w\#| + C_M(x)$$

$\Longrightarrow c_M = |w\#|$ $\surd$

Tétel. *Legyenek U_1 és U_2 univerzális Turing-gépek, melyek input abc-je $I = \{0, 1\}$. Ekkor van olyan $c = c_{U_1,U_2}$ állandó, hogy minden $x \in I^*$ szóra*

$$|C_{U_1}(x) - C_{U_2}(x)| \le c.$$

Bizonyítás: $C_{U_1}(x) \le C_{U_2}(x) + c_{U_2}$ és $C_{U_2}(x) \le C_{U_1}(x) + c_{U_1}$ $\surd$

Definíció. *Rögzítsünk egy U univerzális Turing gépet. Legyen $x \in I^*$. A $C(x) := C_U(x)$ mennyiség az x szó* Kolmogorov-bonyolultsága.

$C(0010) = ?$ $\Longrightarrow C$ függvény nem rekurzív
Vizsgálhatjuk a $C(x_n)$ alakú sorozatok növekedési rendjét, ahol $x_1, x_2, \ldots$ növekvő hosszúságú I^*-beli szavak sorozata.
Pl. az $n = 2^k - 10$ alakú számokra $C(n) \le \log_2 \log_2 n + c'$ teljesül alkalmas c' állandóval.

Tétel. *Legyen $x \in I^*$. Ekkor $C(x) \le |x| + k$, ahol k egy x-től független állandó.*

Bizonyítás: x-hez hozzá kell írni egy semmit nem csináló TG kódját.

Definíció. *Az $x \in I^*$ szó* összenyomhatatlan*, ha $C(x) \ge |x|$.*

Tétel. *Legyen $k \in \mathbb{Z}^+$. Legfeljebb $2^{k+1} - 1$ $x \in I^*$ szó van, melyre $C(x) \le k$. Következésképpen minden $n \ge 1$ egészre létezik n hosszúságú összenyomhatatlan szó. Ha $n > 8$, akkor az n hosszú I^*-beli szavak több, mint 99 százalékának a Kolmogorov-bonyolultsága nagyobb, mint $n - 8$.*

Bizonyítás: Egyforma y-okra egyforma lesz $f_U(y) = x$ is.
$\Longrightarrow$ legfeljebb annyi k-nál nem hosszabb kódú x lehet, amennyi k-nál nem hosszabb szó van: $1 + 2 + \cdots + 2^{k-1} + 2^k = 2^{k+1} - 1$

$$H_k = \{x \in I^* :\ C(x) \le k\}$$

$k = n - 1 \Longrightarrow n$ hosszú szavak száma $2^n \Longrightarrow H_{n-1}$-ben legfeljebb $2^n - 1$ szó van
$\Longrightarrow$ Van legalább n hosszú kódú szó. $\surd$

A H_{n-8} halmaznak legfeljebb $2^{n-7} - 1 < 2^{n-7}$ eleme van.
$\Longrightarrow$ A kedvezőtlen esetek aránya az n hosszú szavak között legfeljebb $2^{n-7}/2^n = 1/128 < 1/100$. $\surd$