

Algoritmusok és adatszerkezetek 2

2015/16 tavaszi félév

Előadó: Dr. Ásványi Tibor

Készítettek:

Koruhely Gábor (Koru)

Szalay Richárd (Whisperity)

Frissítve: 2016. 06. 04.

1₂ EA

első blokk: 10:15 - 11:00

szünet: 11:00 - 11:15

második blokk: 11:15 - 12:00

Rendezés lineáris időben

Edényrendezés (bucket sort)

Tételezzük fel, hogy a kulcsok: [0,1) intervallumba esnek és egyenletesen oszlanak el.

Példa számok: 0,32; 0,81; 0,89; 0,17; 0,53 ($n = 5$) a „k” kulcsot $[k * n]$ edénybe tesszük

0.edény: 0,17

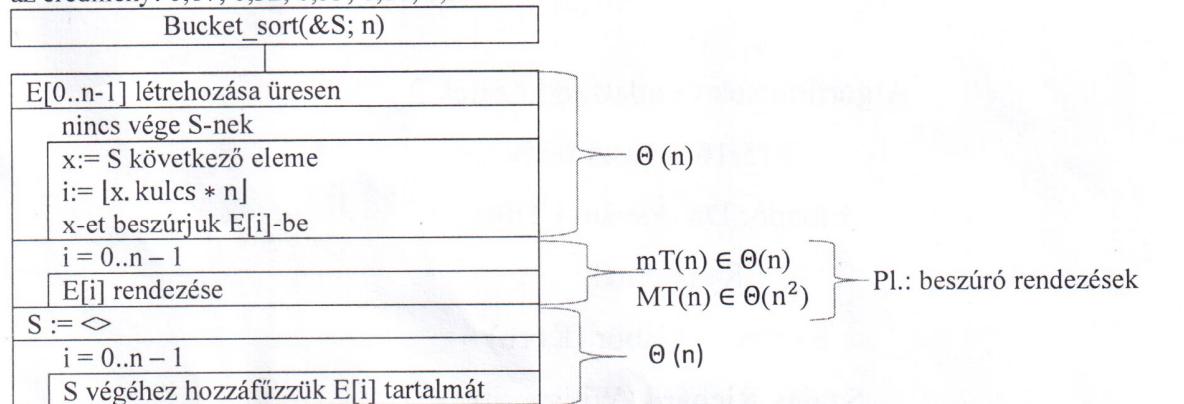
1.edény: 0,32

2.edény: 0,53

3.edény:

4.edény: 0,81; 0,89

az eredmény: 0,17; 0,32; 0,53; 0,81; 0,89



azonban hiába lehet n^2 -ig a műveletigény, a legtöbb esetben (a legtöbb inputra) marad az átlagos n-es műveletigény

Most tegyük fel, hogy a kulcsok [0..k]-ba esnek, $k \in O(n)$ (azaz k legfeljebb n-nel lineáris)

Leszámláló rendezés (Counting sort)

φ : kulcsfüggvény

A[1..n]: T

φ : T \rightarrow [0;k]

C[0..k]: 0..n

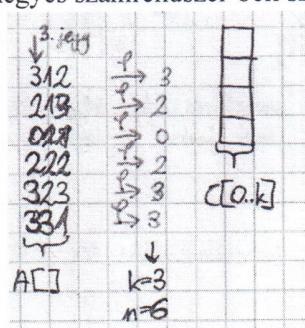
Példa: (k = 3)

(a 0. lépés az init)

C[] adottodik elemkor								
h	0.	1.	2.	3.	4.	5.	6.	végén
0	0		1			1		
1	0					0		
2	0		1	2		2		
3	0	1		2	3	3		

ennyi ilyen kulcsú
elem van φ szerint

négyes számrendszer-beli számok



Counting_sort(A[1..n], k, φ, B[1..n])

```

C[0..k] létrehozása
i := 0..k
C[i] := 0
i = 1 .. n
C[φ(A[i])] := C[φ(A[i])] + 1
i = 1..k
C[i] := C[i] + C[i - 1]
i = n .. 1 (-1)
j := φ(A[i])
B[C[j]] := A[i]
C[j] := C[j] - 1
    
```

ején →
jána →

most már a bal szélső számjegy szerint is rendezett a tömb; a többi számjegy szerint rendezettséget megtartotta..

i másodlagos
most már

C[]:	
vegreh.	várha.
1	1
0	1
2	3
3	6

legföljebb 2 kicsi elemek száma

C[]:	
• várha.	6. 5. 4. 3. 2. 1.
0	0
1	1
2	3
3	5 4 2 1 3

A[] adottadik elemekre

B[]	
0	021
1	213
2	222
3	322 312
	323
	331

Radix (számjegypozíciós) rendezés

Radix(n elem, d számjegy, (szjegyek 0..k))

```

i := 1..d
rendezzük az elemeket stabil rendezéssel
hátról i-edik számjegyük szerint
    
```

Stabil rendezés: amely az elemek rendezési feltételeken kívüli egymáshoz viszonyított sorrendjét nem változtatja (pl Counting_sort ilyen).

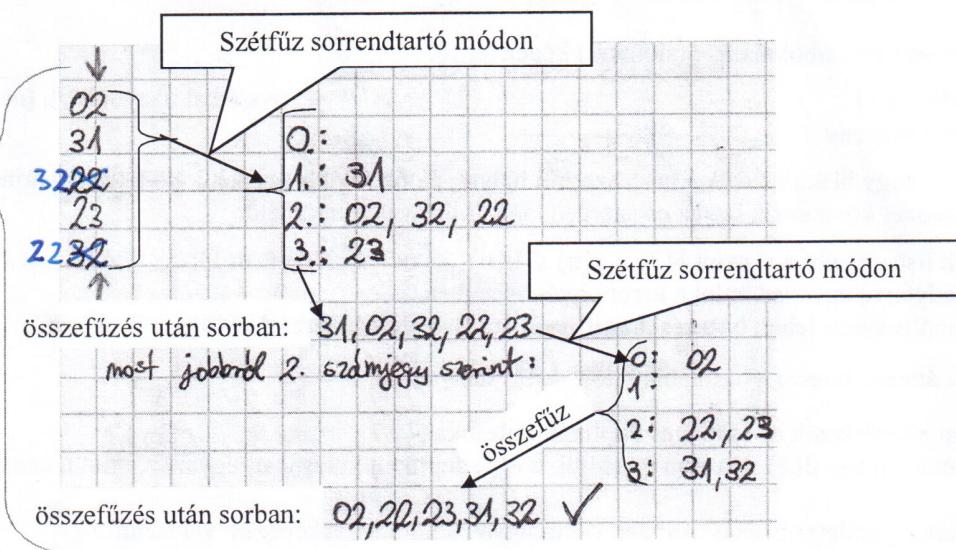
Tradic $\in \Theta(d * T_{\text{stabil}}(n))$
 $T_{\text{counting}}(n, k) \in \Theta(n + k)$

-nekk, egyenlő kulcsok esetén, az

$T_{\text{Radix}} \text{ rendezés tömbökre} \in \Theta(d * (n + k)) = \Theta(n)$

ha d konstans és $k \in O(n)$ volt

Radix rendezés
láncolt listákra,
stabil edény-
rendezéssel, jobbról
az i.-edik számjegy
szerint ($i := 1, \dots, d$)



$T_{\text{radix/edényrendezés}}(n, k) \in \Theta(d * (n + k)) = \Theta(n)$

ha d konstans és $k \in O(n)$.

10₂ EA

tárgy követelményei:

- Legalább elégséges gyakorlati jegy kell a vizsgázáshoz,
- vizsga
 - 3szor lehet maximum vizsgálni egy vizsgaidőszakban,
 - vizsga ugyanolyan felépítésű, mint előző félévben az algo 1

tematika:

- A tárgy honlapján megtalálható a tárgy tematikája: <http://aszt.inf.elte.hu/~asvanyi/ad/ad2programok.pdf>
- 8tételek
- nem összehasonlító rendezésekről lesz szó,
- hasítótáblák
- gráf algók
 - legrövidebb út
 - minimális feszítőfa
- veszteségmentes tömörítés
- mintaillesztés

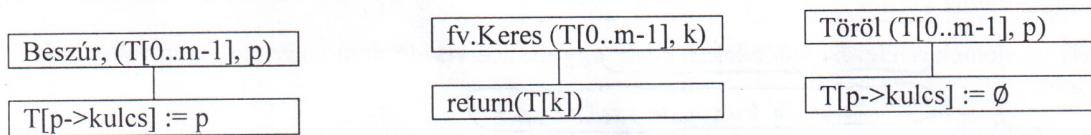
Hasítótáblák (Hash tables)

Hasonlóan a kiegyensúlyozott fákhoz cél lenne a 3 alapművelet (keresés, beszúrás, törlés) optimálisan fusson le.

- átlagos esetben $\Theta(1)$, legrosszabb esetben $\Theta(n)$.

Direkt – címzés (Direct addressing)

a lehetséges kulcsok az $U = \{0, \dots, m-1\}$ (m nem túl nagy pozitív egész) univerzumba esnek
műveletei: beszúr; keres; törlő



A tárígeny miatt problémás a módszer, pl. ha a személyi szám szerint így tárolnánk a magyarokat, akkor a <10 millió embernek 75 millió hely kellene, nagyon sok az üres hely.

h: kulcsuniverzumból részekre (slotokra) képez le.

$$h: U \rightarrow [0..m-1] \quad (|U| \gg m \text{ /* } \gg := \text{sokkal nagyobb*/}, (m \in O(n)) \quad \text{(in a tárolt adatok száma)}$$

h hasító függvény

A baj az, hogy hiába tesszük a hash szerinti helyre, előbb, utóbb lesz két külön kulcs, aminél $h(k) = h(k')$. Ekkor kulcsütközés következik be, ez célszerűen láncolt listával elkerülhető.

Láncolt listás esetben viszont $MT_{\text{keres}}(n) \in \Theta(n)$; $mT_{\text{keres}}(n) \in \Theta(1)$; $AT_{\text{keres}}(n) \in \Theta(1 + \alpha)$, a törlés műveletigénye ugyanaz, mint a keres egyes eseteiben.

a minimális akkor lehet, ha a réshez tartozó lista első elemét keressük, vagy a lista üres.

a listák átlagos hossza a hashtábla kitöltöttségi hányszáma, $\alpha = \frac{n}{m}$

Ha nem ellenőrizzük az esetleges duplikált kulcsokat, a beszúrás igénye $\Theta(1)$. Ha nem engedjük meg a duplikált kulcsokat, ugyanaz, mint a keresés-nél.

Egyszerű egyenletes hasítás: minden slotra ugyanakkora a h leképezési valószínűsége.

Ilyen hash-ek megadhatók, de elég komoly matematika van mögötte.

Osztómódszer: $h(k) = k \bmod m$, ahol m olyan prím amely messze van a 2-hatványoktól

Ha a kulcsok a $[0, 1)$ intervallumon egyenletesen oszlanak el, akkor a $h(k) = \lfloor k * m \rfloor$ is jó hashfüggvény

Szorzómódszer: $0 < A < 1$ konstans

$$h(k) = \lfloor \{k * A\} * m \rfloor \quad (\{x\} \text{ a törtrész fgv.})$$

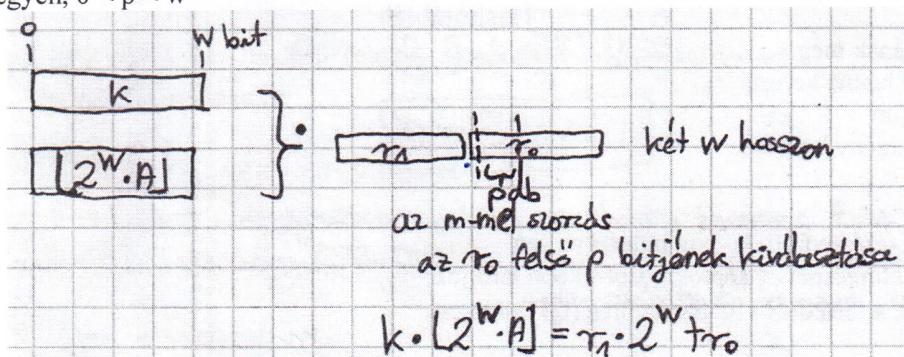
(különböző vizsgálatok szerint $A = \frac{\sqrt{5}-1}{2} \approx 0,618 \dots$ jól szórja szét a kulcsokat.)

Ezt a kiszámítást azonban nehéz elvégezni a lebegőpontos aritmetika miatt.

$m = 2^p$ a hashtábla *mérete*

$w = 32$

$k \in 0 \dots 2^w - 1$; legyen, $0 < p < w$



A kulcsütközések feloldására a láncolt lista helyett vannak más módszerek is.

Nyílt címzés:

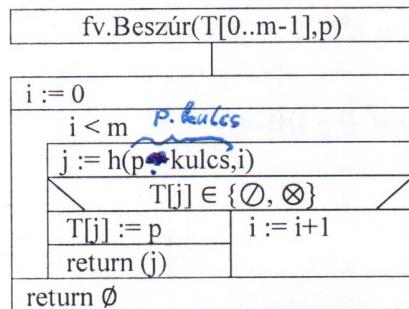
$h: U \times [0 \dots m-1] \rightarrow [0 \dots m-1]$

k kulcshoz m darab hash-érték fog tartozni: $\langle h(k,0); h(k,1); \dots h(k,m-1) \rangle \leftarrow$ próbasorozat

A műveletek a próbasorozaton mennek végig:

szabad

Pl: beszúrásnál ha $h(k,0)$ -hoz van foglalt elem akkor $h(k,1)$ -re szúr be, feltéve ha az már üres vagy törölt



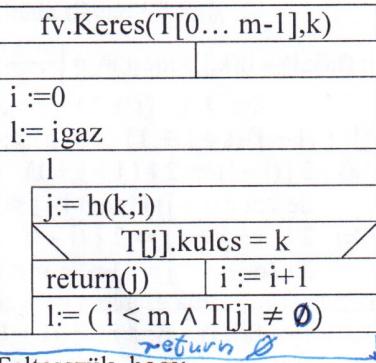
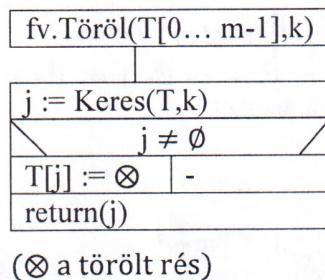
ahol: \emptyset : üres rés
 \otimes : törölt rés

A próbasorozatnak a $\langle 0, 1, \dots, m-1 \rangle$ permutáltjának kell lennie

A próbasorozatok egyik előállítási módja pl. a lineáris próbálás:

$$h(k,i) = (h'(k) + i) \bmod m$$

$$\emptyset \in \mathbb{Z} \setminus [0..m-1]$$



Feltessük, hogy

\emptyset .kulcs, \otimes .kulcs $\notin U$

$k \in U$

11₂ EA

Hasítótáblák

egy szótárt valósítanak meg
kulcs alapján lehet benne keresni
törölni is lehet benne
általános műveleti igény konstans
átlagos

a tábla (ism):

egy tömbnek lehet tekinteni; 0..m-1-ig indexelve van
speciálisan a nyílt címzésnél az adatokat slot-okban tároljuk;
megkülönböztetjük a törölt (\otimes) illetve az üres (\emptyset) slotokat

sikertelen a keresés, ha üres slothoz ér, vagy ha túl sokat próbálkozik (n db próba után megáll)

valójában nem egy hash függvény van, hanem n db hash függvény

feltehetjük, hogy van n db hash függvény

$$h: \{ \dots, i \} : U \rightarrow 0..m-1 \quad (i \in 0..m-1)$$

$$\langle h(k,0), \dots, h(k,m-1) \rangle \text{ perm } \langle 0, \dots, m-1 \rangle$$

lineáris próba: $h(k,i) = (h'(k) + i) \bmod m$

négyzetes próba: $h(k,i) = (h'(k) + c_1 * i + c_2 * i^2) \bmod m \quad /* \text{ahol } h'(k) = h(k,0) */$

ahol jól bevált ajánlás, hogy m 2-hatvány, c_1, c_2 pedig $\frac{1}{2}$ $(m = 2^p, 0 < p \in \mathbb{Z})$

$$i \neq j: (h(k,i) - h(k,j)) \bmod m = \left[\frac{i*(i+1)}{2} - \frac{j*(j+1)}{2} \right] \bmod m \neq 0 \Leftrightarrow$$

$$2m \nmid i * (i+1) - j * (j+1) = i^2 + i - j^2 - j = i^2 - j^2 + (i-j) = (i-j)(i+j+1)$$

$2^{p+1} \nmid (i-j)(i+j+1)$ ezek párossága különbözik

a) $2 \mid (i-j) \Rightarrow 2 \nmid (i+j+1)$

de $2m \nmid (i-j)$ hiszen $i-j < 2m-1$

b) $2 \mid (i+j+1) \Rightarrow 2 \nmid (i-j)$

de $2m \nmid (i+j+1)$ -nek, mivel $(i+j+1) < 2m-1$

azaz 2-hatvány tábla esetén a négyzetes próba szépen lefedi az egész táblát

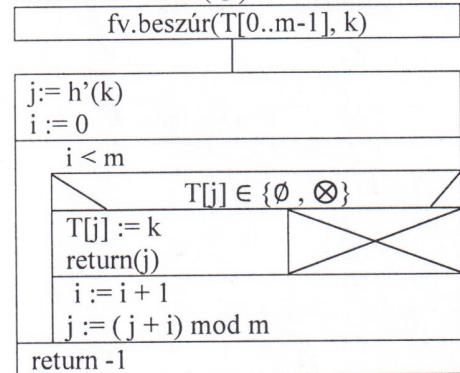
$$(h(k,i+1) - h(k,i)) \bmod m = ((i+1)*(i+2)/2 - i*(i+1)/2) \bmod m = (i+1) \bmod m$$

$$(h(k,i+1) - h(k,i)) \bmod m = \left[\frac{(i+1)*(i+2)}{2} - \frac{i*(i+1)}{2} \right] \bmod m = (i+1) \bmod m$$

azaz: $h(k,i+1) = (h(k,i) + (i+1)) \bmod m$

és $h(k,0) = h'(k)$

feltessük: törölt(\otimes) slot kulcsa és az üres(\emptyset) slot kulcsa nem eleme az U-nak

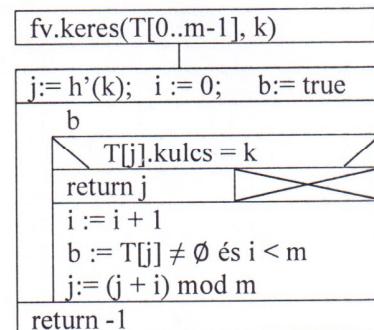


nyílt címzés, négyzetes próba és \emptyset .kulcs és \otimes .kulcs \notin kulcsuniverzum
az általános négyzetes próbánál (ahol c_1 és c_2 tetszőleges)

$$\text{ha } h(k_1, 0) = h(k_2, 0), \text{ akkor } \forall i \text{ re } h(k_1, i) \neq h(k_2, i)$$

Másodlagos csomósodás

Tehát csak m különböző próbásorozat van a lehetséges m! közül, így a (másodlagos) csomósodás fellép, de ez jobb, mint a lineáris próba elsődleges csomósodása, ahol a sorozatok menet közben gabalyodhatnak össze...



Megjegyzés: ha $T[j]$ NIL, akkor $T[j].kulcs$ olyan extremális érték, amely bármilyen

Kettős hasítás

$$h(k, i) = [h_1(k) + i * h_2(k)] \bmod m$$

a próbasorozatok száma itt már $\theta(n^2)$ szemben $\theta(n)$ -nel.

ez – tapasztalat alapján – már majdnem ideális tud lenni

$$\text{ha } \ln_{h_2}(h_2(k), m) = 1 \Rightarrow \langle h(k, 0), \dots, h(k, m-1) \rangle \text{ perm } \langle 0, \dots, m-1 \rangle \text{-nak}$$

hiszen $i \neq j \Rightarrow (h(k, i) - h(k, j)) \bmod m \neq 0$

$$0 \neq (h(k, i) - h(k, j)) \bmod m \Leftrightarrow (i - j) * h_2(k) \bmod m \neq 0$$

$\Leftrightarrow m \nmid (i - j) * h_2(k)$ mivel $|i - j| < m$, $h_2(k)$ pedig relatív prím (ez feltétel volt, hogy lehet biztosítani,

hogy a $h_2(k)$ és az m relatív prímek legyenek).

a) m legyen prím, akkor $h_1(k) := k \bmod m$

$$h_2(k) = \underbrace{(1+k) \bmod m}_{\text{ahol: } m' = m-1 \vee m' = m-2} \text{ például, azaz } m\text{-nél pici kisebb}$$

b) $m = 2^p$ és $2 \nmid h_2(k)$ $1+(k \bmod m')$

Hashelés Ideális esete: Egyenletes hasítás

ideális lenne, ha $\langle 0..m-1 \rangle$ összes permutáció azonos valószínűséggel fordulna elő

$0 < \alpha$ telítettségi együttható $\langle 1$ esetén

- sikertelen keresés várható hossza $\leq \frac{1}{1-\alpha}$
- beszúrás várható hossza $\leq \frac{1}{1-\alpha}$
- sikeres keresés várható hossza $\leq \frac{1}{\alpha} * \ln(\frac{1}{1-\alpha})$

Feltéve, hogy nincs a hasító táblában törölt rés (slot).

Ha sokat használjuk a hashtáblát, akkor feltöredezik, elfogynak az üres slotok, ekkor frissítést kell végrehajtani: beszúrásokkal új táblát létrehozni a mostani elemekből.

Gráf algoritmusok

Szomszédosági mátrix (C)

$$G = (V, E) \quad E \subseteq V \times V$$

$$V = 1..n$$

$$C[i, j] = \begin{cases} C[1..n, 1..n] \\ 1 \Leftrightarrow (i, j) \in E \\ 0 \Leftrightarrow (i, j) \notin E \end{cases}$$

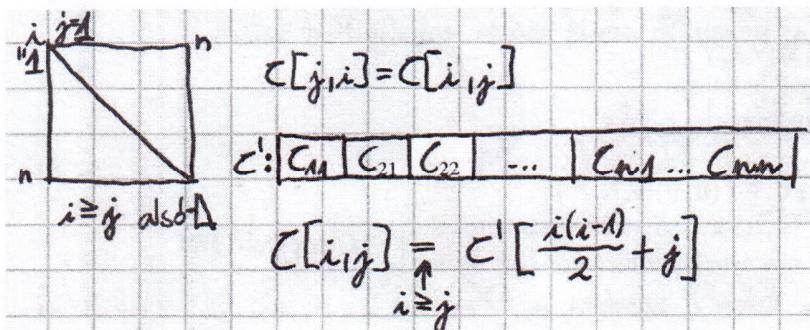
ha számít az él költsége

$$w: E \rightarrow \mathbb{R}$$

$$w(i, j) \Leftrightarrow (i, j) \in E \quad (i \neq j)$$

$$C[i, j] = \begin{cases} 0 \Leftrightarrow i = j \\ +\infty \text{ különben} \end{cases}$$

Azonban a mátrix tárolás miatt a műveletigény $\Theta(n^2)$, ami az irányítatlan esetben a szimmetria kihasználásával javítható (majdnem felezhető), de $\Theta(n^2)$ marad (csak alsó háromszög mátrixot tároljuk).

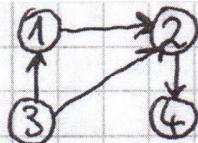


Szomszédosság

Élistás reprezentáció:

Memóriaigény n csúcs és e él esetén: $\theta(n + e)$

Például:



Adj:

	1	2	3	4
1	•	→ 2 ↗ ↘ 0		
2		•	→ 4 ↗ ↘ 0	
3			•	→ 2 ↗ ↘ 0
4	0			

100₂ EA

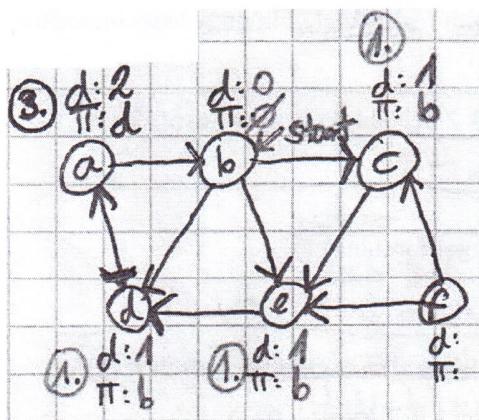
Elemi gráf algoritmusok

Szélességi keresés

Meghatározzuk a start csúcsból a további csúcsokba a legkevesebb élet tartalmazó utat.

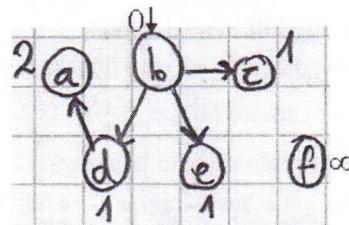
d – hány élen keresztül jutunk a csúcsba

π – melyik csúcsból jutunk a csúcsba



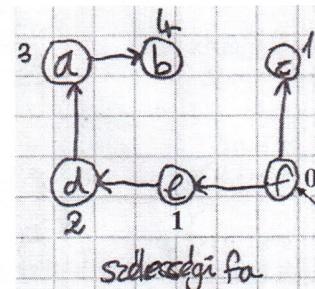
$$Q = \langle b \rangle$$

1. lépés: $Q = \langle c, d, e \rangle$ /* b szomszédai */
2. lépés: $Q = \langle d, e \rangle$ /* c rákövetkezője e, de ott már voltunk */
3. lépés: $Q = \langle e \rangle$ /* „a” d szomszédjai */
4. lépés: $Q = \langle \rangle$ /* e szomszédja csak a d, de ott már voltunk */
5. lépés: $Q = \langle \rangle$ /* a sor kiürült, a bejárás véget ért



f-ből indítva, másik szemléltetési móddal:

d	a	b	c	d	e	f	Q	π	a	b	c	d	e	f
∞	∞	∞	∞	∞	∞	0	$\langle f \rangle$	\emptyset						
f		1		1			$\langle c, e \rangle$			f		f		
c							$\langle e \rangle$							
e			2				$\langle d \rangle$				e			
d	3						$\langle a \rangle$		d					
a		4					$\langle b \rangle$			a				
b	3	4	1	2	1	0	$\langle \rangle$		d	a	f	e	f	\emptyset



Az irányított gráfon a „szomszédsági” kapcsolat nem szimmetrikus.

A fenti gráfban pl.: a szomszédja b, de b-nek nem szomszédja a.

Jelölés:

Ha $G = (V, E)$, $E \subseteq V \times V$ gráf,

$G.V$ a G csúcsainak halmaza

$G.E$ a G éleinek halmaza

$u \in G.V$ esetén

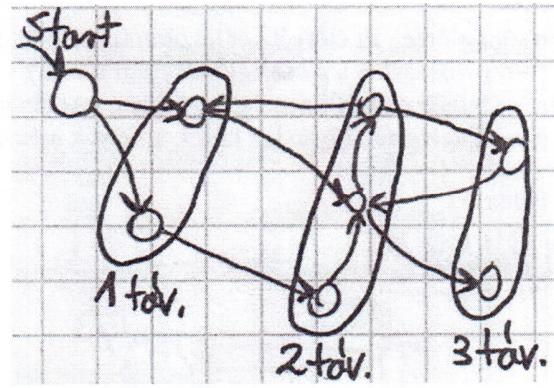
$G.Adj[u] \stackrel{\text{def}}{=} \{ v \in G.V \mid (u, v) \in G.E \}$

Feltesszük, hogy $(u, u) \notin G.E$, azaz nincs hurokél.

Feltesszük, hogy nincsenek párhuzamos élek sem.

G egyszerű gráf

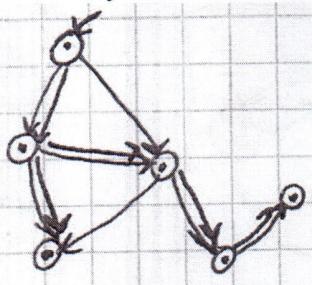
BFS(G,s)
$\forall u \in G.V$
$u.d := \infty$
$u.\pi := \emptyset$
$u.szín := \text{fehér}$
$s.d := 0$
$s.szín := \text{szürke}$
$Q := \langle s \rangle$
$Q \neq \emptyset$
$u := Q.\text{sorból}()$
$\forall v \in G.\text{Adj}[u]$
$v.szín = \text{fehér}$
$v.d := u.d + 1$
$v.\pi := u$
$v.szín := \text{szürke}$
$Q.\text{sorba}(v)$
$u.szín := \text{fekete}$



BFS műveletigénye: $T(n, e) \in O(n + e)$, ahol $n = |V|$ és $e = |E|$
minden legfeljebb egyszer dolgozunk fel, de előfordulhat, hogy néhány csúcs kimerül (a start csúcsból nem elérhető csúcsok és élek maradnak ki.).

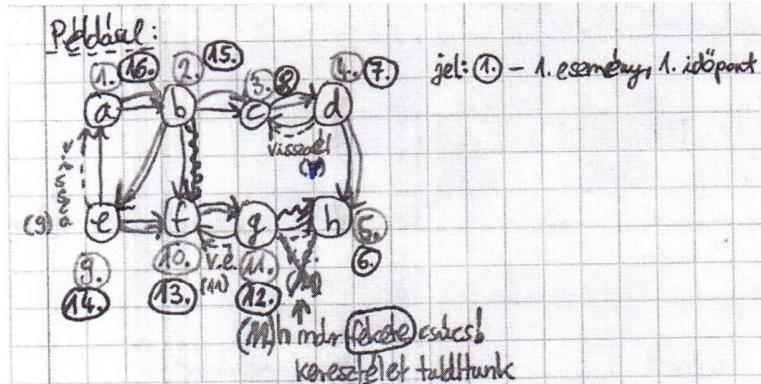
DFS (Mélységi keresés)

DFS(G)
$\forall u \in G.V$
$u.\pi := \emptyset$
$u.szín := \text{fehér}$
$ido := 0$
$\forall u \in G.V$
$u.szín = \text{fehér}$
$DFS_VISIT(u, G, ido)$
$u.d = \text{elérési idő} \quad (\text{discovery time})$
$u.f = \text{befejezési idő} \quad (\text{finishing time})$



DFS_VISIT(&u, G, &ido)
$ido := ido + 1$
$u.d := ido$
$u.szín := \text{szürke}$
$\forall v \in G.\text{Adj}[u]$
$v.szín = \text{fehér}$
$v.\pi := u$
$DFS_VISIT(v, G, ido)$
$v.szín = \text{szürke}$
$viszael(u, v)$
$u.szín := \text{fekete}$
$ido := ido + 1$
$u.f := ido$

Mélységi bejárás példa:



ha már fekete csúcs, keresztejtet találtunk

ÉLEK OSZTÁLYOZÁSA

Def.

$b \rightarrow f$ előreél, az előreél két leszármazottat köt össze, ahol b -ből kettő vagy több faélből álló út vezet f -be.

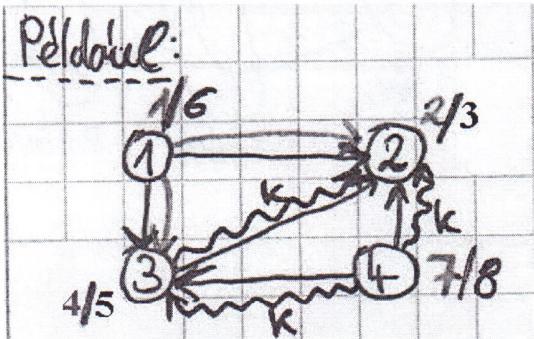
$f \leftarrow g$ visszaél (f a g öse egy mélységi fában.)

$a \rightarrow b$ faél: ez kerül a mélységi fába, e mentén járjuk be a gráfot $(a.d < b.d)$

$g \rightarrow h$ keresztél, olyan két csúcs, amelyek más ágon vannak vagy másik mélységi fába mutat át ($g.d > h.d$)

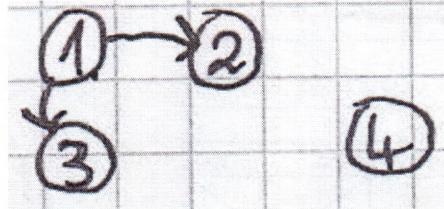
} *

Példa:



DFS műveletigénye: $T(n,e) = \theta(n + e)$ minden csúcsot néhányszor (3) és minden élet egyszer.

Azaz a keresés eredménye egy mélységi erdő



Tétel:

* Ha a

mélységi bejárás során egy (u,v) élet találunk:

(u,v) faél $\Leftrightarrow v.\text{szín} = \text{fehér}$

(u,v) visszaél $\Leftrightarrow v.\text{szín} = \text{sürke}$

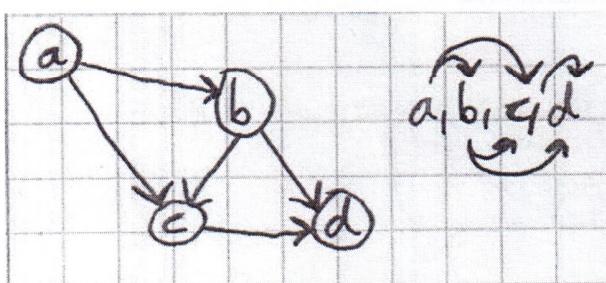
(u,v) előreél $\Leftrightarrow v.\text{szín} = \text{fekete} \wedge u.d < v.d$

(u,v) keresztél $\Leftrightarrow v.\text{szín} = \text{fekete} \wedge u.d > v.d$

CSÚCSAINAK IRÁNYÍTOTT GRAFOK TOPOLOGIKUS RENDEZÉSE

101₂ EA

Topologikus rendezés: A csúcsok olyan sorrendje, amelyben minden él egg-egy később jövő csúcsba (szemléletesen: balról jobbra) mutat.



④ Pontosan akkor \exists topologikus rendezés, ha nincs irányított kör a gráfban.

Ez igaz, hiszen ha veszünk egy megelőzővel nem rendelkező csúcsot és töröljük a gráfból, akkor a maradék gráfban nem keletkezik irányított kör, lesz megint legalább egy olyan, amelyiknek nincs megelőzője.

Sorban a törölt csúcsok adják a topologikus rendezést.

$$G = (V, \text{Adj}[V]) \quad \text{Adj}[u] = \{v \in V \mid (u, v) \in E\} \subseteq V$$

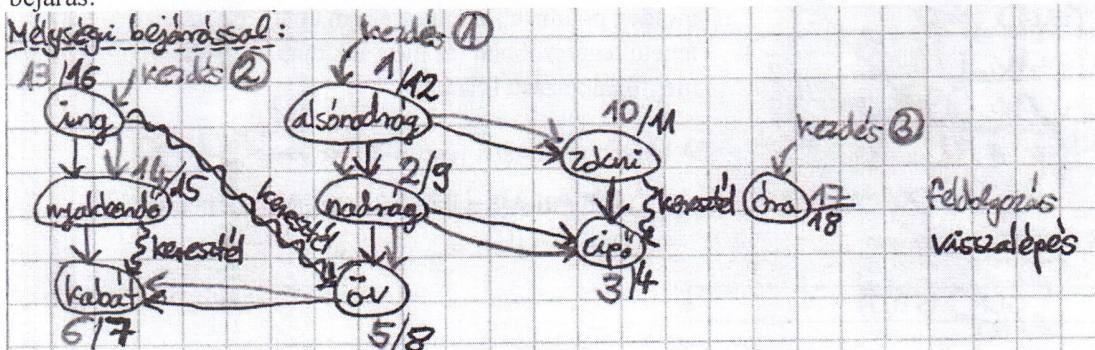
fv. TR(G, TR[1..n])
BEFOKOK(G, BE[1..n])
i := 0
H := {v ∈ V BE[v] = 0}
H ≠ {}
u 'from' H
i := i+1
TR[i] := u
$\forall v \in G.\text{Adj}[u]$
BE[v] := BE[v] - 1
BE[v] = 0
H := H ∪ {v}
return i

$\theta(n + e)$
 $\theta(n)$
 'from' ← kivesz H-ból egy elemet
 a kör csúcsai nem kerülnek H-ba, így az eredménybe (TR[1..n]) sem.
 $O(n+e)$

ha i=n, minden csúcsot feldolgoztunk

$$T_{\text{TR}}(n) \in O(n + e).$$

Mélységi bejárás:



A sorrend: <óra, ing, nyakkendő, alsónadrág, zokni, nadrag, óv, kabát, cipő>

Tetszőleges u csúcs, akkor kerül a TR-be, a háttérrel első szabad helyre, amikor befejeztük.

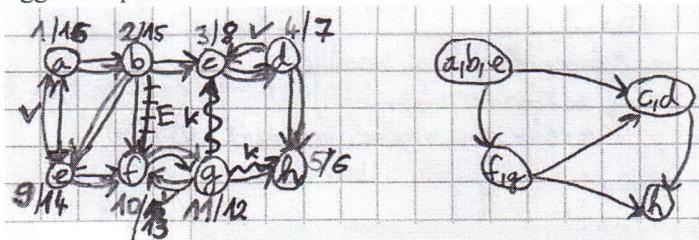
Ha a mélységi bejárás visszaélt talál \Leftrightarrow irányított kör van a gráfban \Leftrightarrow nincs topologikus rendezés.

- ⑤ a) Ha van irányított kör a gráfban, jelölje $u_1, u_2, \dots, u_k, u_1$!
 Ekkor egy topologikus rendezésben u_1 után jön valahol u_2 , u_2 után u3, és így tovább, végtelenül u_k után jön u_1 , és u_1 után u_2 , ami gy, tehát ekkor nincs topologikus rendezés (a gráf csúcsain).
 b) Ha nincs irányított kör a gráfban, akkor nyilván van olyan csúcs, aminek nincs megelőzö.

Erősen összefüggő komponensek

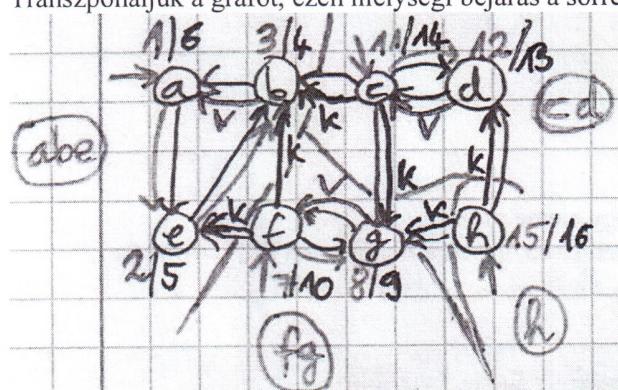
/*Minden csúcsból minden csúcsba vezet irányított út a komponensen belül*/
A gráf erősen összefüggő komponensei:

- a,b,e
- f,g
- c,d
- h

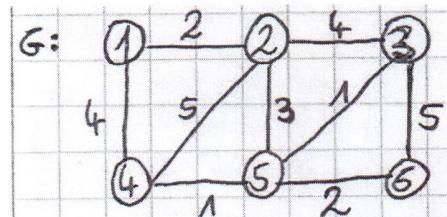


Előállítás: mélységi bejárással mintha topologikus rendezés lenne, de nem kezeljük a visszaéleket.
a sorozat: $\langle a, b, e, f, g, c, d, h \rangle$ a csúcsok befejezés szerinti sorrendben

Transzponáljuk a gráfot, ezen mélységi bejárás a sorrend szerint, az így talált mélységi fák a komponensek



Minimális feszítőfák



Kössünk össze minden várost, hogy minden pontba eljusson az áram, de a lehető legolcsóbban (és még sok más megfogalmazású feladat)

A lénveg minimális feszítőfa keresése.

$G = (V, E)$ $E \subseteq V \times V$ $(u, v) \in E \Leftrightarrow (v, u) \in E$ /* $u \neq v$, a hurokéleknek nincs értelme*/

w: $E \rightarrow \mathbb{R}$ élsúlyok

$w(u, v) = w(v, u)$

GEN_MST(G)
$A := \{\}$
$s := 0$
$s < n - 1$
(u, v) legyen olyan él a $G.E \setminus A$ -ból, ami A-hoz biztonságosan hozzávehető $A := A \cup \{(u, v)\}$ $s := s + 1$

azaz $A \cup \{(u, v)\}$ minimális feszítőfa része marad
 $\underline{\text{P}} \text{ invariáns} = \exists T = (V, T_E) \text{ MST, hogy } A \subseteq T_E$

Def1: $G = (V, E)$, $\emptyset \subsetneq S \subsetneq V$ esetben $(S, V \setminus S)$ vágás a G gráfban

Def2: $A \subseteq E$ és $(S, V \setminus S)$ vágás esetén a vágás elkerüli az A-t $\Leftrightarrow A$ egyetlen élé sem keresztezi a vágást

Def3: (u, v) él keresztezi az $(S, V \setminus S)$ vágást \Leftrightarrow az $u \in S$ és $v \in V \setminus S$ (vagy fordítva $u \in V \setminus S$ és $v \in S$)

Tétel: $G = (V, E)$ irányíthatlan $w: E \rightarrow \mathbb{R}$ -rel élsúlyozott gráf
 $(S, V \setminus S)$ vágás a gráfban elkerüli $A - t$, ahol $\exists T = (V, T_E) MST$, hogy $A \subseteq T_E$
 $(u, v) \in E$ egy könnyű él (legkisebb költségű) a vágásban
Ekkor (u, v) biztonságos A-ra nézve

Bizonyítás:

- a) $(u, v) \in T_E \checkmark$
- b) $(u, v) \notin T_E \Rightarrow (p, q) \in T_E$, ami keresztezi az $(S, V \setminus S)$ vágást, ui: T feszítőfa, tehát T-ben el lehet jutni u-ból v-be.

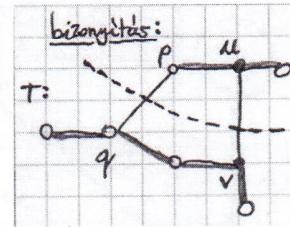
$$\Rightarrow w(p, q) \geq w(u, v)$$

(p, q) törlésével az MST szétesik, de ha ehhez (u, v) -t hozzávesszük, akkor újra feszítőfa lesz.

$$T' = T \setminus \{(p, q)\} \cup \{(u, v)\}$$

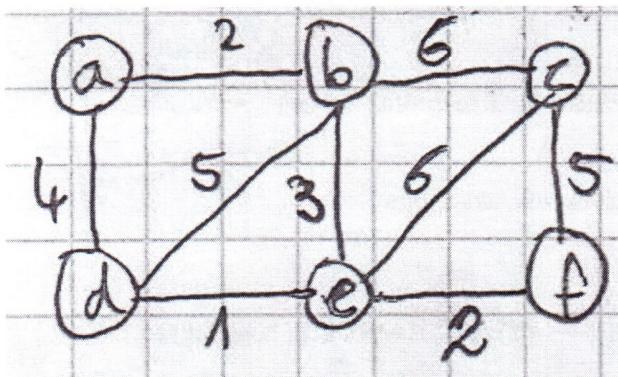
$$w(T') = w(T) - w(p, q) + w(u, v) \leq w(T)$$

viszont mivel T MST volt, $w(T') \geq w(T) \Rightarrow$ azaz $w(T') = w(T)$ és T' is MST kell, hogy legyen...



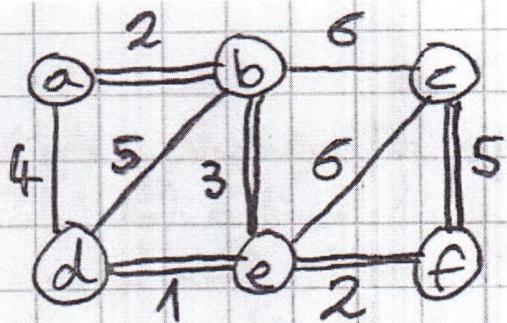
110₂ EA

Minimális feszítőfa ismétlése példán keresztül



A	S	V \ S
\emptyset	{a, b, d}	{c, e, f}
d-e	{a, b, d, e, f}	{c}
d-e c f	{a}	{b, c, d, e, f}
a-b	{a, b, d, e}	{c, f}
d-e c f		
a-b	{a, b}	{c, d, e, f}
d-e-f c		

a-b-e-f
d c' a végeredmény:



Kruskal algoritmus

- Elsőként súly szerint sorba rendezzük a csúcsokat! minden csúcs egy egyelemű fát képezzen!
- Majd minden lépésben hozzávesszük a minimális élt, ha külön komponenseket kötnek össze (ha nem, kihagyjuk)

0. lépés	1	2	3	4	5	6	7	8	9	végeredmény:
	d ¹ e	a ² b	e ² f	b 3 e	a 4 d	b 5 d	c 5 f	b ⁶ c	b ⁶ c 0	
a b c d e f	a b c d ¹ e f	a ² b c d ¹ e f	a ² b c d ¹ e ² f	a ² b c 3 d ¹ e ² f kihagyjuk	a ² b c 4 3 d ¹ e ² f kihagyjuk	a ² b c 3 5 d ¹ e ² f kihagyjuk	a ² b ⁶ c 3 5 d ¹ e ² f kihagyjuk	a ² b c 3 5 d ¹ e ² f kihagyjuk	a ² b c 3 5 d ¹ e ² f kihagyjuk	@ 2 6 1 2 3 5

A komponensek nyilvántartásához menetközben egy másik fát is kezelünk:

0	1 (d-e)	2 (a-b)	3 (f-d)	4 (b-e)	5	6	7 (c-f)	8	9
a b c	a b c	a d c	a d c	a ← d c				(kihagyjuk)	(kihagyjuk)
d e f	d f ↑ e	↑ b e f	↑ ↑ ↗ b e f	↑ ↑ ↗ b e f	(kihagyjuk)	(kihagyjuk)		(kihagyjuk)	(kihagyjuk)

MST_Kruskal(G,w)
G.E mon. növ. rend. w szerint
$\forall u \in G.V$
Make_Set(u)
$A := \{\}$
$\forall (u,v) \in G.E$ (w szerint mon. növ.)
x := HOL_VAN(u)
y := HOL_VAN(v)
$x \neq y$
A := A ∪ {(u,v)} Unió(x,y)
return(A)

$$e = |G.E| < n^2$$

$$\begin{aligned} O(e * \log e) &\leq O(e * \log n) \\ \Theta(n) \\ \Theta(1) \\ O(\log n) \text{ e-szer} \\ O(n) \text{ (n-1-szer néhány konstans)} \end{aligned}$$

$$O(e * \log n)$$

A HOL_VAN() megállapítja a csúcs melyik ilyen köztes fában van, ennek megfelelően az unió később a fákat köti össze. (Mindegyik fát a gyökércsúcsa azonosítja.)

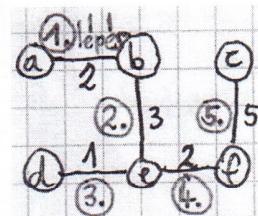
Az új élt úgy vesszük a köztes fához, hogy a fák magassága ne változzon, de a komponenshez tartozás jelezve maradjon. (Az alacsonyabb fa gyökerét a nagyobb fa gyökere alá kötjük be.)

A fák magassága legfeljebb $\log_2 n$ ($n = |G.V|$)

Közös magasság esetén az uniázott fa magassága $h+1$ lesz (h közös magasság volt)

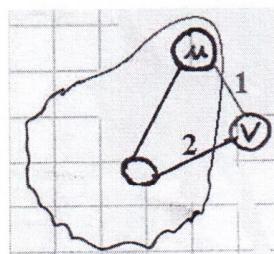
Prim algoritmus

- Kiindulva egy csúcsból és a (csúcs, maradék) vágás közé választja a minimális élt
- Ezt hozzáveszi az eredményhez és az (eddig eredmény, maradék) vágásban keresi a könnyű élt
- addig folytatva az előző két lépést, míg Minimális feszítőfát nem kapunk.



meg kell adni, hogy hol kezdi (s), de mindegy, hogy hol kezdi

MST_PRIM(G, s, w)
$\forall u \in G.V$
u.kulcs := ∞
u. π := \emptyset
s.kulcs := 0
Q : PrSor(G.V, kulcs alapján)
Q ≠ \emptyset
u := Q.MIN_KIVESZ()
$\forall v \in G.Adj[u]$
$v \in Q \wedge v.kulcs > w(u, v)$
v.kulcs := w(u, v)
v. π := u



Lehet, hogy az u hozzávétele után v közelebb kerül a részleges feszítőfához.

Az egész algoritmus műveletigénye: $O((n+e) * \lg(n))$, de a gráf összefüggő ($e \geq n-1$), azaz $O(e * \lg(n))$



kezdő csúcs: a

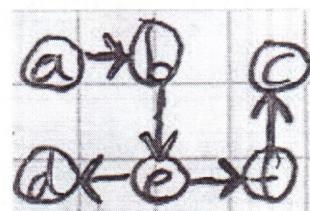
lépés	a	b	c	d	e	f	a	b	c	d	e	f
start	0	∞	∞	∞	∞	∞	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset
a		2		4				a		a		
b	-		6		3				b		b	
e		-		1		2				e		e
d	-	-			-							
f			5		-				f			
c	nincs már több lehetséges szomszéda											
vége:	0	2	5	1	3	2	\emptyset	a	f	e	b	e

-: a szomszéd már korábban belekerült a fába

1: új, jobb távolság

prim optimalizációja:

Fibonacci kupacokkal le lehet csökkenteni a műveleti igényt: $O(e+n^*lg(n))$



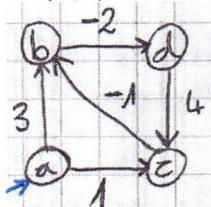
Az MST, amit kaptunk, irányított formáktól reprezentált IRÁNYÍTATLAN FA

111₂ EA

Legrövidebb út probléma

Adott egy tetszőleges hálózat, amiben az utaknak költségeik vannak, ezt a hálózatot egyszerű gráffal ábrázoljuk.

$$G = (V, E) \quad w: E \rightarrow \mathbb{R}$$



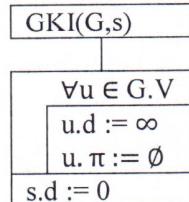
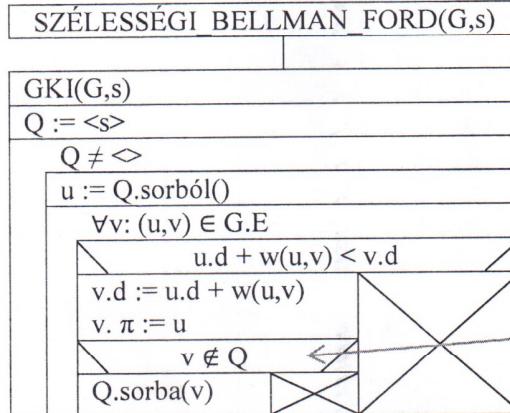
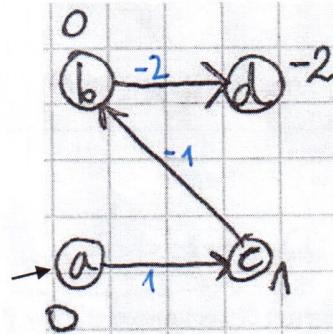
Negatív költség is lehetséges. Adott pontból az összes többi csúcsba meg szeretnénk határozni a legrövidebb utat a költségekkel, nem biztos, hogy a legkevesebb élből álló út a legrövidebb.

Mikor értelmes a kérdés:

- ha létezik a start csúcsból elérhető negatív kör, nincs legrövidebb út egyetlen a körből elérhető csúcsba sem.
- Ha a startcsúcsból csak nem negatív kör érhető el, az az útból elhagyható.
- Optimális útnak tetszőleges részüja is optimális út.

Általános algoritmus: (Robert Endre Tarjan: Data Structures and Network Algorithms)
„Queue-based BELLMAN-FORD”

d	a	b	c	d	Q	π	a	b	c	d	menet
init	0	∞	∞	∞	$\langle a \rangle$		\emptyset	\emptyset	\emptyset	\emptyset	\emptyset
a		3	1		$\langle b, c \rangle$		a	a			0.
b				1	$\langle c, d \rangle$				b		1.
c		0			$\langle d, b \rangle$		c				
d					$\langle b \rangle$						2.
b				-2	$\langle d \rangle$				b		3.
d					$\langle \rangle$						
	0	0	1	-2			\emptyset	c	a	b	



a sorban levés ellenőrizhető pl. egy adattaggal, mindenkor a sort nem lenne hatékony.

Tulajdonság:

$\forall u \in G.V$, hogy ha az u csúcsba vezet k élből álló $s \rightarrow u$ optimális út \Rightarrow a k. menet elején már $u.d = w(s \rightarrow u)$ és $(s, \dots, u.\pi.\pi, u.\pi, u)$ egy optimális út

Lemma: s -ból nem érhető el negatív kör (amely összköltsége negatív) \Rightarrow tetszőleges u elérhető csúcsra létezik $s \rightarrow u$ optimális út, és ez legfeljebb $n-1$ élből áll.

T(Lemma és Tulajdonság következménye):

s -ból nem érhető el negatív kör

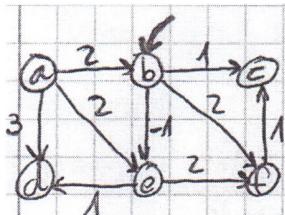
\Rightarrow tetszőleges u elérhető (s -ból) csúcsra létezik $s \rightarrow u$ optimális út, és egy optimális út az $n-1$. menet elején már rendelkezésünkre áll (ki van számolva).

\Rightarrow az $n-1$. menet végére kiürül a sor, az algoritmus $O(n^2e)$ időben megáll.

Következmény: Ha az $(n-1)$. menet végén van a Q sorban elem \Rightarrow van s -ból elérhető negatív kör. (\Leftarrow is igaz)

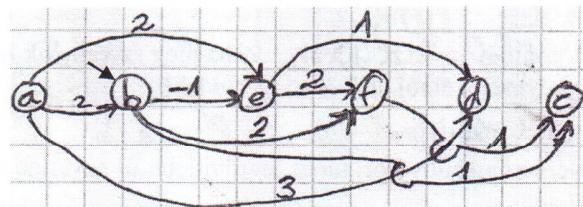
Megjegyzés: vegyük egy ilyen w elemet. Ebből a π pointereken visszafele haladva megtalálható egy ismétlődő v elem, amire $\exists k \in 1..n \quad \pi^k(v) = v$, és $(v, \pi^{k-1}(v), \pi^{k-2}(v), \dots, \pi(v), v)$ egy negatív kör.

Legrövidebb utak egy forrásból, körmentes irányított gráfokon



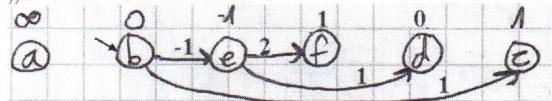
feltétel: nincs irányított kör,
akkor lehet topologikus rendezéssel optimális utakat keresni.

topologikus rendezés:



	a	b	c	d	e	f	a	b	c	d	e	f
init	∞	0	∞	∞	∞	∞	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset
a												
b				1		-1		b		b		b
e					0			e		e		e
f						1						
d												
c												
vége:	∞	0	1	0	-1	1	\emptyset	\emptyset	b	e	b	e

„a” nem volt elérhető



Műveletigény: $\theta(n+e)$ műveletigény (topologikus rendezés)

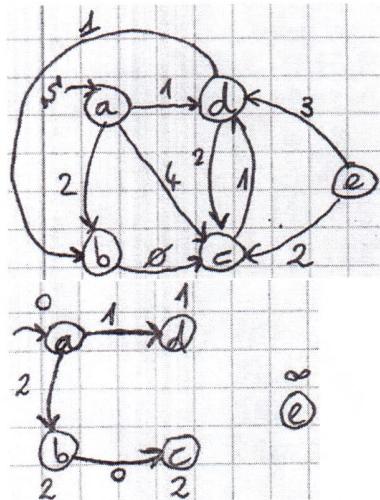
\forall csúcs kiterjesztése ($\theta(n+e)$)

Teljes műveletigénye: $\theta(n+e)$

nem történik semmi, el kell jutni a start csúcsba

Legyen $G = (V, E)$ akár irányított, akár irányítatlan
 $w: E \rightarrow \mathbb{R}_0$ nem negatív valós számok

DIJKSTRA algoritmus



Dijkstra(G, w, s)

```

 $\forall u \in G.V$ 
.d :=  $\infty$ 
. $\pi$  :=  $\emptyset$ 
s.d := 0
Q := G.V
Q  $\neq \emptyset$ 
u := Q.MINKIVESZd()
 $\forall v: (u, v) \in G.E$ 
  v.d > u.d + w(u, v)
  v.d := u.d + w(u, v)
  v. $\pi$  := u
  Q.HELÝREÁLLÍTd(v)

```

alsó index d azt jelenti, hogy a minkivesz a d értékei szerinti szomszédsági éllista + bináris kupac esetén $T(n, e) \in O((n+e)*\log n)$

$\Theta(n)$
 $\Theta(1)$
 $\Theta(n)$
d szerint n-szer $\Theta(n)$
 $e < n^2$
 $e * O(1)$

$T(n) \in \theta(n^2)$
ha a Q rendezetlen

Q.MINKIVESZ_d(): $n * O(\lg n)$
Q.HELÝREÁLLÍT_d(v): $O(e * \lg n)$

Áll: Amikor a csúcsot kivesszük a sorból, oda már optimális út vezet.

Bizonyítás:

start csúcsra ✓, a költség optimális. Tegyük fel, hogy igaz az Állítás! Legyen u az 1. ilyen csúcs!

Legyen az u csúcs t pedig az s \rightarrow u optimális út első csúcsa amely $\in Q$ sornak.

u-ra nem igaz, u-t válasszuk kiterjesztésre, ekkor nincs talált optimális út (de attól még létezik)

u.d > w(s \rightarrow u), mivel egyébként megtaláltuk volna az optimális utat.

t.d $\stackrel{\text{opt}}{=} w(s \rightarrow t) \leq w(s \rightarrow u) < u.d$

az s \rightarrow t optimális út, mivel optimális út részülya.

hiszen „t”-t megelőző csúcs kiterjesztésekor beállítottuk,

de mivel t.d < u.d, ezért a következő építés „t”-t választja kiterjesztésre ↳

Minden csúcsból minden csúcsba legrövidebb út minden csúcsra futtatott Dijkstrával:

- ritka gráfra $O(n(n+e)*\log n) = O(n^2*\log n)$
- sűrű gráfra $O(n^3*\log n)$, nem jó választás...
 - ehelyett a vektorpáros Dijkstra stabil $O(n^3)$ -öt ($n * O(n^2)$) fut.

Floyd – Warshall - algoritmus

Engedjük meg a negatív élsúlyt, de ne lehessen negatív kör.

$G = (V, E)$

$G: V = 1..n$

csúcsmátrixos ábrázolás

$D_{ij} := i \rightarrow j$ optimális út költsége (véglesen, ha nincs optimális út)

$\pi_{ij} := i \rightarrow j$ optimális úton j szülője

$D_{ij}^{(k)} =$ az $i \rightarrow j$ úton az $[1..k]$ indexű csúcsok lehetnek csak közbenső csúcsok.

$$\max_{w(i \xrightarrow{k} j)} \{w(i \xrightarrow{k} j)\}$$

$\pi_{ij}^{(k)}$ = egy ilyen $[1..k]$ közbensős úton a j csúcsok szülője

$$D_{ij}^{(0)} = \begin{cases} 0, & \text{ha } i = j \\ w(i,j), & \text{ha } (i,j) \in G.E \wedge i \neq j \\ \infty, & \text{ha } i \neq j \text{ és } (i,j) \notin G.E \end{cases} \quad (\text{közvetlen út szülője})$$

$$\pi_{ij}^{(0)} = \begin{cases} \emptyset, & \text{ha } i = j \\ i, & \text{ha } i \neq j \text{ és } (i,j) \in G.E \\ \emptyset, & \text{különben} \end{cases}$$

$$D_{ij}^{(n)} = D_{ij}$$

$$\pi_{ij}^{(n)} = \pi_{ij}$$



ha $D_{ij}^{(k-1)} > D_{ik}^{(k-1)} + D_{kj}^{(k-1)}$, akkor $D_{ij}^{(k)} = D_{ik}^{(k-1)} + D_{kj}^{(k-1)}$
különben: $D_{ij}^{(k)} = D_{ij}^{(k-1)}$

hasonlóan $\pi_{ij}^{(k)}$ is ha nem tudtuk az utat javítani = $\pi_{ij}^{(k-1)}$

ha tudtuk = $\pi_{kj}^{(k-1)}$

$$D_{ik}^{(k)} = D_{ik}^{(k-1)}$$

$$D_{kj}^{(k)} = D_{kj}^{(k-1)}$$

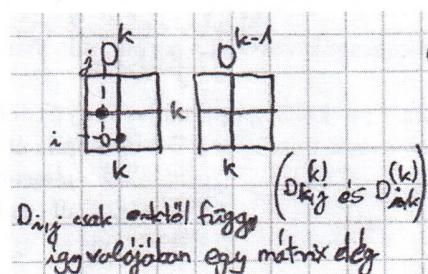
Az algoritmus lépésenként előállítja mátrixpárok sorozatát.

$$D^{(k)} = \left(D_{ij}^{(k)} \right)_{i,j=1..n} \quad k = 0, \dots, n\text{-ig}, n+1 \text{ db mátrix.}$$

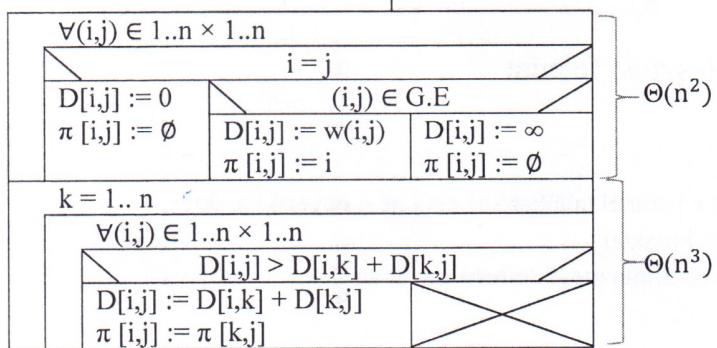
D és π k-adik sora és oszlopa a (k-1) és k. lépésben egyenlő:

D_i . sora: i. csúcsból optimális utak hossza.

ut D_i -edik sora



Floyd-Warshall(G,D[1..n, 1..n], w, π[1..n, 1..n])



$T_{FW} \in \Theta(n^3)$

asztimptotikus műveletigény

Súrú Ritka gráfokon, mint a Dijkstra, de a Dijkstra asztimptotikusan jobb (feltéve hogy nincs negatív él)

A negatív körök a D főátlójában negatív számként jelennek meg.

jelennek

részleggy a Dijkstra
is alkalmazható.

Tranzitív lezárt algoritmus (Warshall alg.)

T_{ij} : van-e út i -ból j -be (megállapítja, hogy van-e a két csúcs között út (csak az számít, hogy van-e, hogy milyen költségű az nem számít))

(legfeljebb az 1..k csúcsokat)

$T_{ij}^k \Leftrightarrow i \xrightarrow{k} j$ megint csak a k . indexet már nem érintő út.

, ha $i = j \vee (i,j) \in G.E$

$T_{ij}^0 = \begin{cases} \uparrow & , \text{ha } i = j \\ \downarrow & , \text{különben} \end{cases}$

$T_{ij}^k = T_{ij}^{k-1} \vee (T_{ik}^{k-1} \wedge T_{kj}^{k-1})$

$T_{ij}^n = T_{ij}$

$T_{ik}^k = T_{ik}^{k-1}$

$T_{kj}^k = T_{kj}^{k-1}$

Sűrű gráfokra sokkal jobb, mint n db szélességi bejárás. *

TRANZ LEZ(G, T[1..n, 1..n])

$\forall (i,j) \in 1..n \times 1..n$

$T[i,j] := (i = j \vee (i,j) \in G.E)$

$k = 1 .. n$

$\forall (i,j) \in 1..n \times 1..n$

$T[i,j] := T[i,j] \vee (T_{ik}^{k-1} \wedge T_{kj}^{k-1})$

$\Theta(n^2)$

$\Theta(n^3)$

$(T[i,k] \wedge T[k,j])$

Mj.: A "felső" indexet mindenütt zárójelekbe kell tenni;

pl. $T_{ij}^{(k)}, T_{ij}^{(k-1)}, \dots$ (nem hogy vki határozják, hanem nézze).

($e \in O(n)$)

* Ritka gráfokra viszont n db. szélességi bejárás műveletigénye $O(n * (n+e)) = O(n^2)$ asztimptotikusan

$e \in O(n)$

Kisebb, mint a $\Theta(n^3)$ Warshall alg.-nal.

Sűrű gráfokra $T_{BFS}(n) \in O(n * (n+e)) = O(n^3)$.

$e \in O(n^2)$

Ez hasonló a Warshall-hoz, de annak kisebb a konstans szorzója.

Mintaillesztési feladat

1	2	3	4	5	6	7	8	9	10	11
T[1..11] =	A	B	A	B	B	A	B	A	B	A
	B	A	B	A						
	B	A	B	A						
	B	A	B	A						
	B	A	B	A						
	B	A	B	A						
	B	A	B	A						
	B	A	B	A						
	B	A	B	A						
	B	A	B	A						
	B	A	B	A						

P[1..4] = BABA

piros betű: annál a betűnél található az első nem egyező karakter

zöld betű: egyező karakter

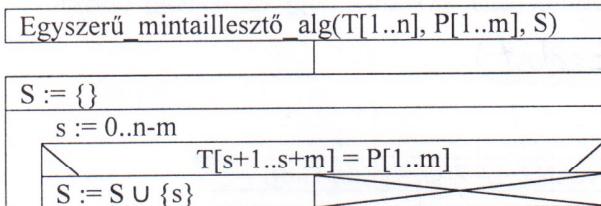
zöld szó: a keresett minta megtalálható a szóban

s = 4 eltolás

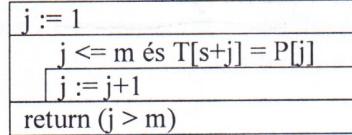
két találat

s = 6 eltolás

Fa: $T[1..n]: \Sigma; P[1..m]: \Sigma \quad 0 < m \leq n$
 $S := \{ s \in 0..n-m \mid T[s+1..s+m] = P[1..m] \}$ programozása



T[s+1..s+m] = P[1..m]



O(m)
 $MT(m) \in \Theta(m)$
 $mT(m) \in \Theta(n)$ $\Theta(1)$

Az egész eljárás műveletigényei:

 $MT(n,m) \in \Theta((n-m+1)*m) = \Theta((n-m)*m)$ $mT(n,m) \in \Theta(n-m+1) = \Theta(n-m)$ *n > m esetén*

Megjegyzés:

Gyakran feltehető: $n \gg m$ Ekkor: $MT(n,m) \in \Theta(n*m)$ $mT(n) \in \Theta(n)$

Elég jónak tűnhet, de nagy inputra hamar elszáll, kell ennél jobb algoritmus.

$$T_{s+1} = \sum_{j=1}^m \varphi(T[s+1+j])d^{m-j} = (T_s - \varphi(T[s+1])d^{m-1})d + \varphi(T[s+m+1]) : \theta(1) \text{ időben}$$

$$\begin{array}{|c|c|c|c|} \hline T_s & 1 & 2 & 8 & 25 \\ \hline \end{array}$$

$$T_{s+1} = \begin{array}{|c|c|c|c|} \hline & 2 & 8 & 2514 \\ \hline \end{array}$$

ha d^{m-1} -et elölne kizártuk.

T_0 és P_0 számítása $\theta(m)$ és $\theta(m)$ idejű a rekurzió $\theta(1)$ idejű.

A teljes műveletigény így $\theta(m) + (n-m) * \theta(1)$, ami $\theta(n)$ igényű.

ha a rekurzió tényleg $\theta(1)$ és a T_i, P_0 számokat a gép tudja hatékonyan ábrázolni.

Mi lesz a nagy mintákkal?

$$p := P_0 \bmod q \quad (q \text{ „nagy” prím})$$

$$t_s := T_s \bmod q$$

$$h := d^{m-1} \bmod q$$

$$d^{m-1} \quad t_{s+1} = (t_s - \varphi(T[s+1])h)d + \varphi(T[s+m+1]) \bmod q$$

a különbség átcaphat negatívba...

$$\textcircled{\times} \quad t_{s+1} = \left(\left(\frac{(t_s + dq - \varphi(T[s+1])h)}{dq+1} \bmod q \right) * d + \varphi(T[s+m+1]) \right) \bmod q \quad \textcircled{\Delta}$$

hogy jól működjön $(d+1)q \leq$ legnagyobb ábrázolható szám kell legyen (size_t, Integer'Last'...)

Lehet hamis találat (fals pozitív), ekkor ellenőrizni kell, hogy valódi találat-e, eredeti karakterekkel.

$\textcircled{\times}$ -bol:

$$\textcircled{\Delta} \quad t_s = \left((t_{s-1} + dq - \varphi(T[s])h) \bmod q \right) * d + \varphi(T[s+m]) \bmod q$$

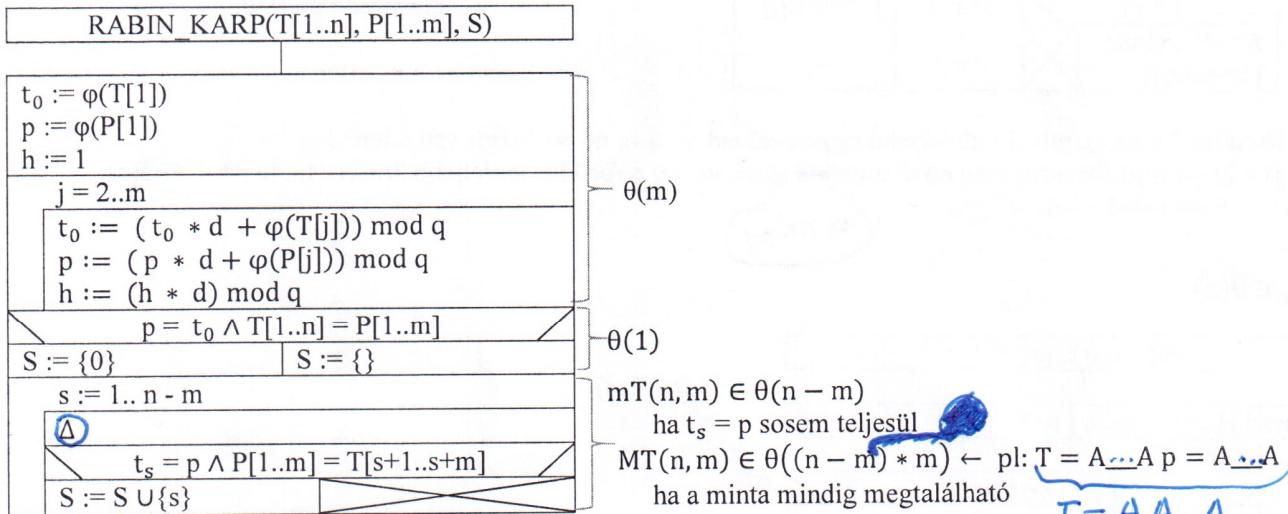
RABIN-KARP folytatása

$q = \frac{\max N}{d+1}$ legnagyobb ábrázolható N szám, prímszám használatával a mod q számítások nem fognak túlcordulni

$$p \neq t_s \Rightarrow P_0 \neq T_s$$

$p = t_s \Rightarrow \{ P_0 \neq T_s$ így potenciális találat esetén a valódi találatot ellenőrizni kell.
 $P_0 = T_s$

a modulo számítás miatt
 elveszik az információ



teljes algoritmus:

$$mT_{RK} \in \theta(n)$$

$$MT_{RK} \in \theta((n - m + 1) * m)$$

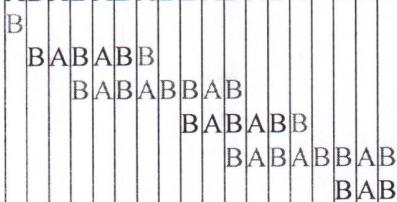
A gyakorlatban nincs túl sok valódi találat, nagy q esetén a hamis találatok száma se túl sok van.
 Így $AT_{RK} \sim mT_{RK}$ (nem biz)

Mintaillesztés lineáris időben

Példa:

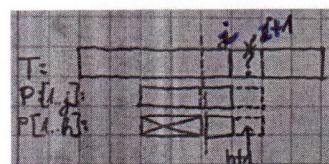
$$P[1..8] = BABABBAB$$

$$T[1..18] = |AB|AB|AB|AB|B|AB|AB|AB|B|AB$$



eltoljuk a mintát annyival, hogy a minta illeszkedő része az eltolás után illeszkedjen részben a szövegre

B|AB... (vége a szövegnek)



Keressük a legnagyobb olyan h -t, hogy $P[1..h]$ suffixe $T[1..i]$ és $h < j$

$T[1..i]_j$ hosszú suffixe $= P[1..j] \Rightarrow T[1..i]_h$ hosszú suffixe $= P[1..j]_h$ hosszú suffixe $= P[1..h]$
 ilyen feltételt kielégítő h -t keresünk.

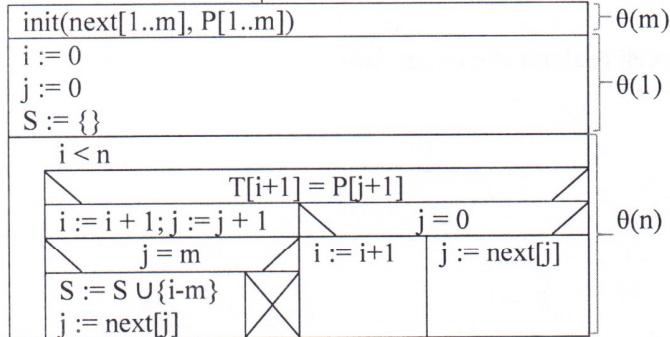
Ez az információ csak a minta ismeretében megkapható.

Def: $\text{next}(j) = \max\{h \in [0..j-1] \mid P[1..h] \sqsupseteq P[1..j]\}$

$\text{next}[1..m] := \text{next}(1..m)$

Ezt kiszámolva a $\text{next}(j)$ -vel toljuk el a mintát, mivel korábban biztos nem lehet illeszkedés.

úgy KMP($T[1..n]$, $P[1..m]$, S)



$$0 \leq i \leq n \wedge 0 \leq j < m \wedge i \geq j \wedge P[1..j] \sqsupseteq T[1..i] \wedge S = \{s \in 0..i-m \mid P[1..m] = T[s+1..s+m]\}$$

$\text{next}^k(j) = P[1..j]$ k-adik leghosszabb illeszkedő prefixe

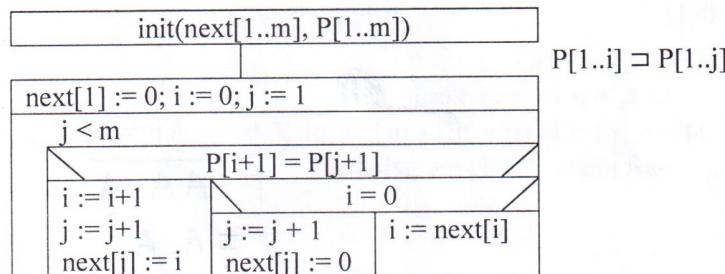
A fő ciklus $T \in \Omega(n)$, mivel i növekedni egyesével tud, és n-ig nő \Rightarrow biztos van n lefutás

$t(i,j) = 2i - j \in [0..2n]$ szig mon növő minden ágon, de így a főciklus legfeljebb 2n-szer fut le $\Rightarrow T \in O(n)$

$0 \leq j \leq i \leq n$

a négy

\Downarrow
 $T_{fö} \in \Theta(n)$



most is a $2j-i \in [0..2n]$, az ágakon szigorúan monoton növő minden így legfeljebb $2m$, legalább m -szer fut le $\Rightarrow \Theta(m)$

1011₂ EA

Folytatás init stuki:

Def: next: [1..m] → [0..m-1] ($m \in \mathbb{N}_+$)

$\text{next}(j) = \max\{h \in [0..j-1] \mid P[1..h] \supseteq P[1..j]\}$

Tulajdonságok:

$$0 \leq h < j$$

1. $P[1..h]$ ps-párosa $P[1..j]$ -nek $\Leftrightarrow 0 \leq h < j \wedge P[1..h] \supseteq P[1..j]$

2. $P[1..i+1] \supseteq P[1..j+1] \Leftrightarrow P[1..i] \supseteq P[1..j] \wedge P[i+1] = P[j+1]$

3. $0 \leq \text{next}(j) < j$ Köv: $j > \text{next}(j) > \text{next}^2(j) > \dots > \text{next}^k(j) = 0$

4. $0 \leq \text{next}(j+1) \leq \text{next}(j) + 1$

5.

- a. $\text{next}^k(j)$ értelmezve van $\Leftrightarrow \exists a P[1..j]$ -nek a k. leghosszabb ps-párosa
- b. $\text{next}^k(j)$ értelmezve van $\Rightarrow P[1.. \text{next}^k(j)]$ a $P[1..j]$ k. leghosszabb ps-párosa

Q: $m > 0$ /*előfeltétel*/

R: $\text{next}[1..m] = \text{next}(1..m)$ /*utófeltétel*/

Inv: $0 \leq i \leq j \leq m \wedge \text{next}[1..j] = \text{next}(1..j) \wedge P[1..i] \supseteq P[1..j] \wedge (\text{a } P[1..j] \text{ tetszőleges } i\text{-nél hosszabb } P[1..i] \text{ ps-párosára: } P[i+1] \neq P[j+1] \text{ } (0 \leq i < j))$

$2j-i \in 2..2m$] a ciklus legfeljebb (2n-2)-szer hajtódik végre
2-ről indul, szig. mon. nő] a ciklus legalább (n-1)-szer végrehajtódik

Példa:

j	1	2	3	4	5	6	7	8
P[j]	A	B	A	B	B	A	B	A
next[j]	0	0	1	2	0	1	2	3

Az algoritmus működése lépésről-lépére:

i	j	next[j]	1	2	3	4	5	6	7	8
0	1	0	A							
0	2	0		A						
1	3	1			AB					
2	4	2				AB	A			
0	4	OK					A			
0	5	0					A			
1	6	1					AB			
2	7	2						ABA		
3	8	3								

⊗ üres mező

Tömörítés

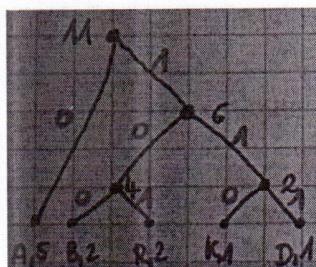
Naív módszer:

$T[1..n]: \Sigma = \{\sigma_1, \dots, \sigma_d\}$
 $n * |\lg d|$ bittel kódolható a szöveg

Huffman kód: ^{egyszerűsített} prefix kód

a szöveg: ABRAKADABRA

jel	előfordulás	Kód
A	5	0
B	2	100
R	2	101
K	1	110
D	1	111



$|\lg d|$ bittel kódolható a karakter

Mi: it karakterenként kódolva tömörítések között a Huffman kód ~~minimális~~ hossza minimális.

Tömörített kód: 01001010110011101001010

kitömörítés a kódta]: A B R A K A D A B R A
alapjain]

IN:	a	b	a	b	c	b	a	b	a	a	a	a
OUT	1	2	4	3	5	8	1	10	11	1		

szó	kód
a	1
b	2
c	3
ab	4
ba	5
abc	6
cb	7
bab	8
baba	9
aa	10
aaa	11
aaaa	12

Rekonstruálás:

IN'	1	2	4	3	5	8	1	10	11	1
OUT'	a	b	ab	c	ba	bab	a	aa	aaa	a

8-as kód nincs a szótárban!
generáláskor a 8-as kód a ba... szövegből jött létre

$\underbrace{ba}_{5} \underbrace{bab}_{8} \rightarrow \underbrace{baba}_{8} \dots ?$

(hasonlóan járunk el a 10-es, 11-es esetben is)

szó	kód
a	1
b	2
c	3
ab	4
ba	5
abc	6
cb	7
bab	8
baba	9
aa	10
aaa	11
aaaa	12

az utolsó sor dekódoláskor
legtöbbször felesleges márLZW_COMPRESS (In, Out, Σ)

D := { $\underline{\sigma_1}: 1, \underline{\sigma_2}: 2, \dots, \underline{\sigma_d}: d$ }
kód := d + 1
s := get_char(In)
~eof(In)
c := get_char(In)
($\widehat{s:c}: \underline{}$) $\in D$
s := $\widehat{s:c}$ write(Out, kód(D,s))
D := D $\cup \{\widehat{s:c}: \underline{\text{kód}}\}$ ⊕
s := c
kód := kód + 1
write(Out, kód(D, s))

 $\Sigma = \{\sigma_1, \dots, \sigma_d\}$ ábécé

s: string

c: char

D: szótár, „string: kód” alakú rendezett párosok halmaza

 $\widehat{s:c}$: az s string és a c char konkatenálta $\underline{\sigma_i}$: a σ_i betűből álló string (egybetűs string)

kód(D,s) = a D szótárban az s string kódja

eof(In) \Leftrightarrow az In inputról V karaktert beolvastunk

„s: _” olyan rendezett páros, aminek az első komponense s

⊕ Az így jelzett utasítások csak akkor hajtandók végre,
ha még kód \leq MAXKÓD, ahol MAXKÓD a kódok előre rögzített hosszától függ.Pl: ha ez 12bit, akkor MAXKÓD = $2^{12}-1 = 4095$.LZW_DECOMPRESS (In, Out, Σ)

D := { $\underline{\sigma_1}: 1, \underline{\sigma_2}: 2, \dots, \underline{\sigma_d}: d$ }
kód := d + 1
k := get_code(In)
s := string(D, k)
write(Out, s)
~eof(In)
k := get_code(In)
($\underline{}: \underline{k}$) $\in D$
t := string(D, k) t := $\widehat{s:c}$
write(Out, t)
D := D $\cup \{\widehat{s:c}: \underline{\text{kód}}\}$ ⊖
s := t
kód := kód + 1

s,t: string

k, kód: kódok

t₁: a t string első betűje

string(D,k)= a D szótárban k kódnak megfelelő string

eof(In) \Leftrightarrow az In inputról V karaktert beolvastunk„ $\underline{}: \underline{k}$ ” olyan rendezett páros, aminek a 2. komponense k

⊗: Itt k = kód teljesül

Balra, a ⊖

Megjegyzés: A fenti elágazásban az „ $\underline{}: \underline{k}$ $\in D$ ” feltétel helyén „k < kód” is állhatna (így egy kicsit gyorsabb lenne az algoritmus).

Felhasznált segédanyagok:

- Az előadáson készített jegyzetem (Koru),
- Whisperity előadás jegyzete (a jegyzetben található összes kép és néhány szöveges kiegészítés).

A jegyzetet átolvasták és a talált hibákat jelezték, amiért köszönnet:

- Bán Róbert
- Fekete Anett
- László Tamás
- Nagy Vendel
- Szabó Gergő
- Szécsi Péter
- Tőkés Anna