

Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

ALGORITMI in PODATKOVNE STRUKTURE

zapiski predavanj 2013/2014

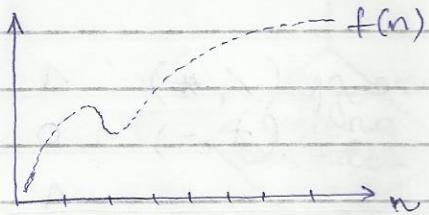
prof. dr. Borut Robič

Asimptotični rast funkcij (Big Oh notation, omega notacija)

Algoritem troši računarske vire (čas, prostor)

Poraba rač. vira narašča, če narašča velikost reševanega problema (n).

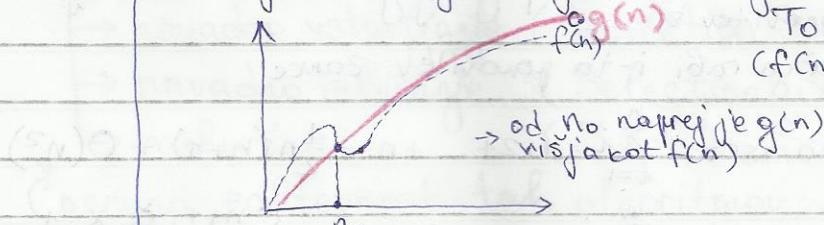
Obstaja zveza (funkcija) med velikostjo reševanega problema in časom potrebnih za rešitev reševanega problema.



Točna/natančna funkcija $f(n)$ je včasih težko, nemogoče najti.
Gotovo je pa nesmiselno.

Funkcija $f(n)$ pravimo, da je asimptotično:

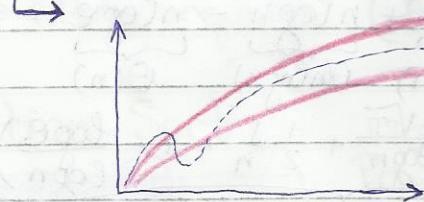
→ omejena navzgor s $g(n)$, če velja: $(\exists c, n_0 > 0), (\forall n \geq n_0) [f(n) \leq c \cdot g(n)]$
 To zapisemo: $f(n) = O(g(n))$,
 $(f(n))$ je kvečjemu reda $g(n)$



→ omejena navzdol s $h(n)$, če: $(\exists d, n_1 > 0), (\forall n \geq n_1) [f(n) \geq d \cdot h(n)]$
 To zapisemo: $f(n) = \Omega(h(n))$,
 $(f(n))$ je vsaj reda $h(n)$



$c_2 \cdot k(n)$ To zapisemo: $f(n) = \Theta(k(n))$,
 $\text{če } f(n) = O(k(n)) \wedge f(n) = \Omega(k(n))$
 $(f(n))$ je točno reda $k(n)$



Primer: $3n^3 = \Theta(n^3) = \Omega(n^3) = O(n^3)$

$$3n^3 = O(n^3 \log n) = \Omega(n^2) = \Omega(n)$$

NE $\rightarrow 2^n \geq O(n^k), k \in \mathbb{R}$ (2^n narašča hitreje kot n^k)

DA $2^n = \Omega(n^k)$

Ocenjevanje poraba časa

Poenostavitev: npr. množljivne op. ($/, *$) 1

aditivne op. ($+, -$) 0

primerjave 1

$\sum \dots$ poenostavitev bo odvisna od samega problema

Zanke so pogoste \Rightarrow v izrazih se pojavljujo razne vsote \sum_i^n

(*) \rightarrow st. ponovitev zanke

$\sum_{i=1}^n t_i \rightarrow$ čas ki ga rabi i-ta ponovitev zanke

Npr. aritmetična vsota $\sum_{i=1}^n i = 1+2+\dots+n = \frac{1}{2}n(n+1) = O(n^2)$

geometrijska vsota $\sum_{i=1}^n x^i = 1+x+x^2+\dots+x^n = \begin{cases} n+1, \text{ če } x=1 \\ \frac{x^{n+1}-1}{x-1}, \text{ sicer} \end{cases}$

harmonična vsota $\sum_{i=1}^n \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} = \ln(n) + \underbrace{O(1)}_{\text{konstanta}}$

$\sum_{i=1}^n \log i = \log 1 + \log 2 + \dots + \log n = \log(1 \cdot 2 \cdot \dots \cdot n) = \log(n!) =$
 / Stirlingova ocena / $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n = \log\left(\sqrt{2\pi n} \left(\frac{n}{e}\right)^n\right) =$

$= \underbrace{\log \sqrt{2\pi}}_{\text{const. } O(1)} + \underbrace{\frac{1}{2} \log n}_{\Theta(\log n)} + \underbrace{n \log n}_{\Theta(n \log n)} - \underbrace{n \log e}_{\Theta(n)} = \Theta(n \log n)$

Preverimo: $n \log n \left(\underbrace{\frac{\log \sqrt{2\pi}}{n \log n} + \frac{1}{2} \cdot \frac{1}{n} + 1 - \frac{\log e}{\log n}}_{\text{če to ne narašča hitreje kot konstante}} \right)$

če to ne narašča hitreje kot konstante \Rightarrow funkcijo pred okreplji; narašča hitreje

Recurzivne enačbe

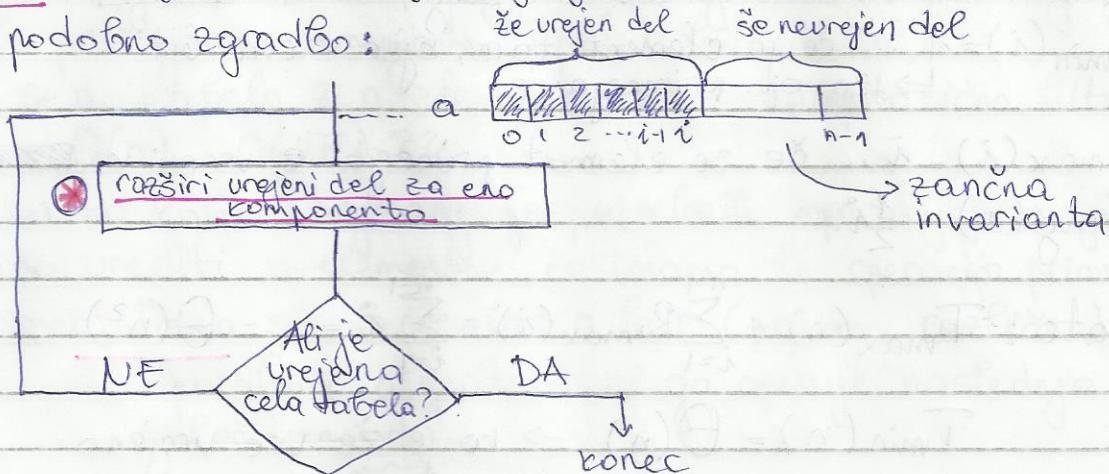
$$T(n) = \begin{cases} b \\ aT\left(\frac{n}{c}\right) + f(n) \end{cases}$$

uporabimo isto funkcijo za c-krat manjši element

Notranje urejanje (urejanje tabel velikosti n v glavnem pomnilniku)

Navadni → Alg. za notranje urejanje

Imajo podobno zgradbo:



Navadni alg. se razlikujejo po temu, kako izvedejo :

Npr.

→ navadno vstavljanje (Insertion Sort)

→ navadno izbiranje (Selection Sort)

→ mehurčno (Bubble Sort)

Časovna zahtevnost teh algoritmov:

→ $T(n)$... čas. zahtevnost za vreditev tabele z n -elémenti

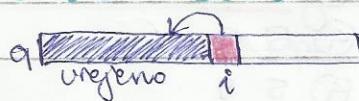
$$\rightarrow T(n) = \sum_{i=1}^n R(i) = \text{sez} / \text{izkazalo} / = \Theta(n^2) \text{ pri vsih treh!}$$

↳ čas. zahtevnost operacije

Izboljšani alg. za notranje urejanje:

- | | | | |
|---------------|--------------------------------|---|--|
| (Insertion) ⇒ | Shellovo urejanje (Shell Sort) | } | \rightarrow povprečju $T(n) = O(n \log n)$ |
| (Selection) ⇒ | urejanje s kopico (Heap Sort) | | |
| (Bubble) ⇒ | hitro urejanje (Quick Sort) | | |

Insertion Sort



Prvi element neurejenega dela vrini v urejeni del na ustrezeno mesto.

for $i := 1$ to $n - 1$ do

$j := i$

$x := a[j]$

 while $(j \geq 1) \wedge (x < a[j-1])$ do

$a[j] := a[j-1]$

$j := j - 1$

 end while

$a[j] := x$

end for

$R(i)$... čas za razširitev urejenega dela 

$R_{\min}(i) = 1$... če je element tam, kjer mora biti
 \hookrightarrow za op. primerjanje

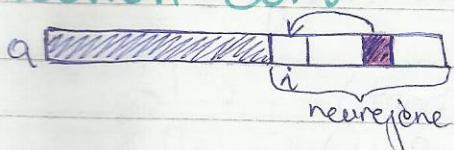
$R_{\max}(i) = i$... če se element primerja z vsemi v 

$$R_{\text{avg}}(i) = \frac{1}{2}i$$

$$\text{Zato: } T_{\max}(n) = \sum_{i=1}^{n-1} R_{\max}(i) = \sum_{i=1}^{n-1} i = \dots = \Theta(n^2)$$

$$T_{\min}(n) = \Theta(n) \rightarrow \text{ko je že vse urejeno}$$

Selection Sort



Med neurejenimi poišči najmanjšega in ga prestavi na i -to mesto

```

for i := 1 to n do
    k = i
    for j := i+1 to n do
        if a[j] < a[k]
            end for
            a[k] := a[j]
    end for
  
```

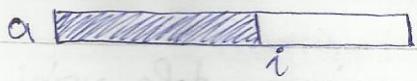
→ Not sure if right?

Časovna zahtevnost:

$$R_{\min}(i) = R_{\max}(i) = R_{\text{avg}}(i) = n-i-1 \text{ primerjanj}$$

$$T(n) = \Theta(n^2)$$

Bubble Sort



Pr. 8 6 7 ④ 9 5

8 6 7 ④ 5 9

8 6 ④ 7 5 9

8 ④ 6 7 5 9

④ 8 6 7 5 9

← Na prvem mestu dobimo najmanjši element neurejenega dela

$$R_{\min}(i) = R_{\max}(i) = R_{\text{avg}}(i) = n-i \text{ primerjanj}$$

$$T(n) = \sum_{i=1}^{n-1} R(i) = \sum_{i=1}^{n-1} (n-i) = \Theta(n^2)$$

Spodnja meja za čas urejanja tabel

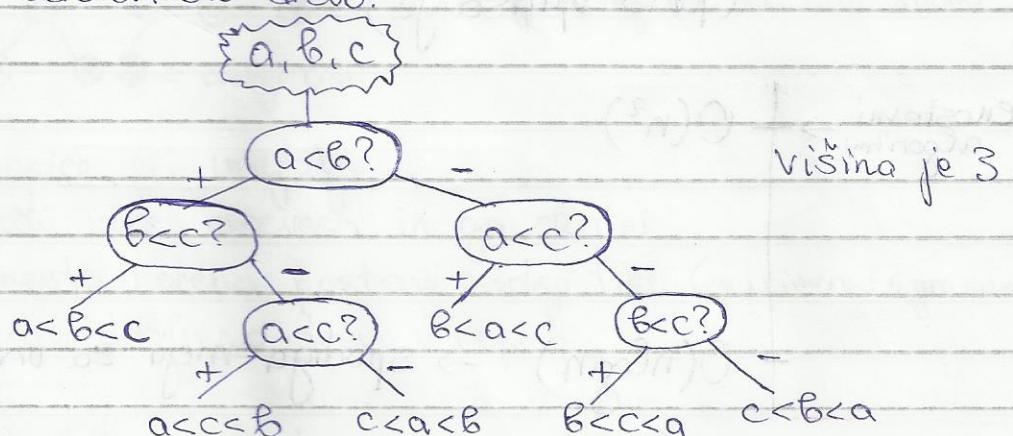
Naradni algoritmi za urejanje zahtevajo $\Theta(n^2)$ primerjanj (tudi če pristojemo zamjenjave, spet dobimo $\Theta(n^2)$).

? Ali se da tabelo z n -elementi urediti osimptotično hitreje?
Npr. $O(n)$, $O(\sqrt{n})$??

Ocenimo koliko najmanj je potrebnih operacij primerjanja.
? Kako urediti n -elementov, če imamo le operacijo primerjanja?

Primer: Dana so 3 števila: a, b, c (pačoma različna)

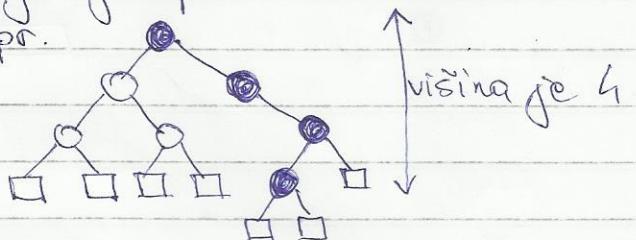
Uporabimo algoritmom, ki ga opisuje naslednje odločitveno drevo.



Spošno: Če so dana števila a_1, \dots, a_n lahko odnose med njimi ugotovimo z nekim podobnim drugiščnim drevesom, ki ima v notranjih vozliščih operacije primerjanja. Drevo ima v listih permutacije n -elementov. Torej je listov ($n!$).

Višina drugiščnega drevesa, ki ima N listov je vsaj $\geq \lceil \log_2 N \rceil$. (Višina je število notranjih vozlišč (ali število povezav na najdaljši veji) na najdaljši poti do listov.)

Npr.



Naše odločitveno drevo z $n!$ listi, bo imelo višino $C \geq \lceil \log n! \rceil$

$\lceil \log n! \rceil$ je število primerjaj, ki so potrebna za ureditev n števil (v tabeli)
 (Višina pove koliko najmanj primerjaj potrebuje.)

$$\lceil \log n! \rceil \geq \log n! = /n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n / \dots = \text{const. } n \log n = \underline{\underline{\Theta(n \log n)}}$$

\Rightarrow Ne išči alg. ki ima $O(n)$, $O(\sqrt{n})$ primerjaj
 (prvo vprašanje)

enostavni algoritmi $\rightarrow O(n^2)$

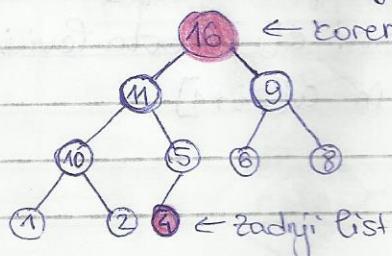
$\rightarrow O(n \log n)$ → spodnja meja za urejanje

Heap Sort (urejanje s kopico)

Kopica (Heap)

- dvojiško drevo (v vozlišči so števila, ki želimo urediti)
- urejeno - število v vozlišču je večje od števil v sinovih tega vozlišča
- leva poravnana
 - ↳ vse veje so dolge bodisi d ali $d-1$, $d \in \mathbb{R}$
 - ↳ vse daljše veje so na levri strani drevesa

Primer:



- dolžine vej so bodisi 2, bodisi 3
- daljše veje so na levri strani

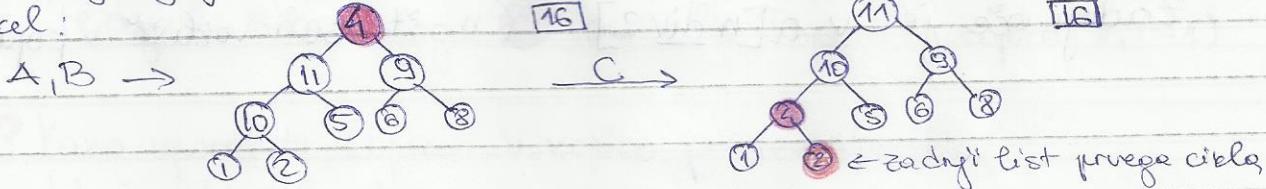
Uporaba kopice pri urejanju:

- A Izloči koren drevesa in ga shrani
- B Namesto korena postavi zadnji list (v primeru: 4 gre namesto 16)
- C Popravi dobijeno drevo v kopico

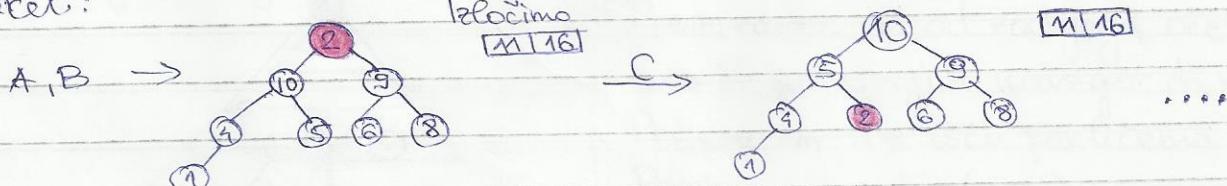
Na koncu izločeni koreni so urejena števila.

Iz zgornjega primera:

I cikel:



II cikel:



Podatkovna struktura (za kopico)

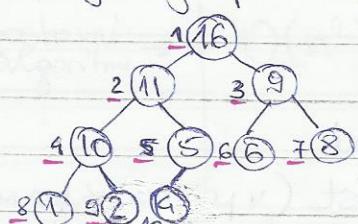
Kopico realiziramo s tabelo:



kjer so:

- koren kopice je v prvi komponenti $a[1]$
- če je vozlišče v komponenti $a[i]$ bosta levi in desni sin v komponentah $a[2i]$ in $a[2i+1]$

Iz prejšnjega primera:



zadnji oče

→ eden izmed in dotorov v tabeli

Lastnosti: če je $s > 1$, potem je v $a[s]$ sin nekoga.

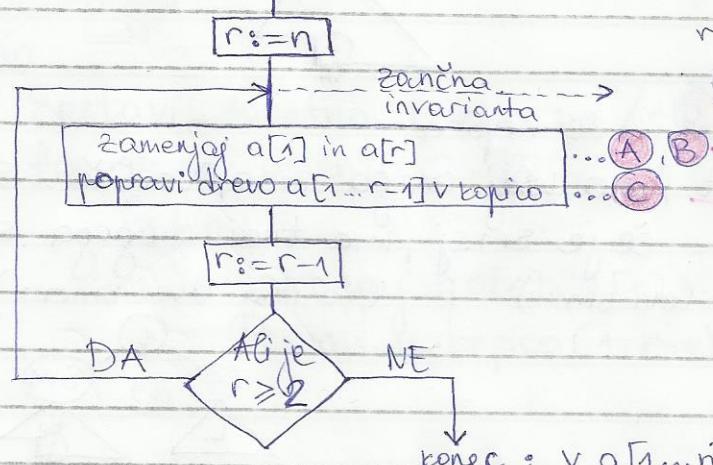
→ ta oče je v $a[s \div 2]$

→ če je s sod, potem je sin levi, sicer je desni

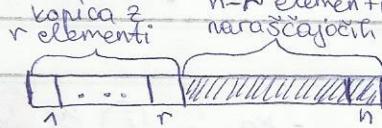
Zadnji oče je v $a[n \div 2]$ (n - št. vseh vozlišč v kopici)

Urejanje

sestavi začetno kopico v $a[1..n]$... D

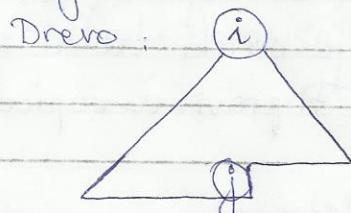


urejen del z
 $n-r$ elementi po
naraščajočih vrednosti



? Kako izvedemo C in D

Definimo, da bi imeli na voljo proceduro Popravi_v_kopico(i, j), ki zna naslednje:

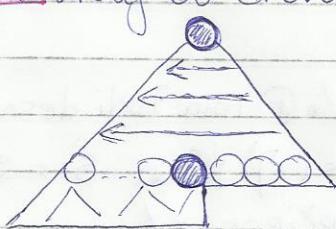


v katerem sta dve poddrnevesi vozlissa i ,
kopici, zna popraviti v kopico.

Tedaj bi bila naloga C kar klic Popravi_v_kopico(1, r-1)

? Kako uporabiti Popravi_v_kopico pri izvedbi D?

Ideja: Naj bo drevo:



Premikamo se od zadnjega očeta v levo in po nivojih navzgor do korena.
Pri vsakem vozlisušču ponovimo
Popravi_v_kopico(i, n)

Procedure Sestavi_zacetno_kopico;

var i: integer

begin

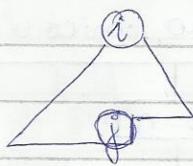
for i:=n div 2 do ento 1 do

Popravi_v_kopico(i, n)

end

? Kakšna je procedura Popravi_v_kopico?

Poglejmo poddružo:

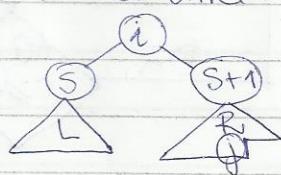
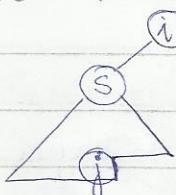


- če je i list, je že kopica
- če i ni list, ima vsaj enega sina, vozlišče $s = 2 \cdot i$

če je $s+1 \leq j$, ima tudi drugega sina

(če ima samo levi sin:)

(če ima dva sina:)



Predpostavimo, da sta L in R že kopici, pozorno bomo usmerili na večjega sina (Njega bomo po potrebi imenovali s). Če je ta (največji) sin večji od očeta, ju zamenjamo (oče se pogreze). Poddružo, ker mor se je pogreznil oče, je morda treba spet popraviti kopico. Zato spet ponovljemo (rekurzivno) Popravi_v_kopico(s, j).

Procedure Popravi_v_kopico (var i, j : integer)

var s : integer

begin

if $i \leq (j \text{ div } 2)$ then /* i ima sina */

begin

$s := 2 \cdot i$;

if $(s+1 \leq j)$ then /* i ima tudi desnega sina */

if $(a[s] \leq a[s+1])$ then $s := s+1$; /* sedaj

"s" označuje večjega */

if $(a[i] < a[s])$ then

begin

zamenjaj ($a[i]$, $a[s]$)

Popravi_v_kopico (s, j)

end

end

end

Celoten algoritem:

Procedure Heapsort (var a[1...n])

var r: integer;

begin

Sestavi-zacetno-kopico ; /* D */

for r:=n do vsto 2 do

begin

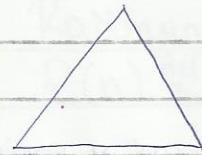
zamenjaj (a[1], a[r]) /* A, B */

Popravi-v-kopico(1, r-1) /* C */

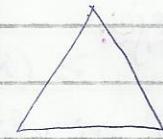
end

end

$\lceil \log_2 n \rceil$



$\lceil \log_2(n-1) \rceil$



$\lceil \log_2 i \rceil$



← Pogrezanje korena
traja sorazmerno z
 $\lceil \log_2 i \rceil$

Višine se bodo manjšale n-krat

Heapsort: Analiza časovne zahtevnosti

Priprava: Spomnimo se, vrsta $\sum_{j=1}^{\infty} \alpha_j$ ($\alpha_j > 0$) konvergira če je $\lim_{j \rightarrow \infty} \frac{\alpha_{j+1}}{\alpha_j} < q < 1$ (kvocientni kriterij)

Nas bo zanimala vrsta: $\sum_{j=1}^{\infty} \frac{1}{2^j}$. Tu je $\alpha_j = \frac{1}{2^j}$

$$\text{Velja } \lim_{j \rightarrow \infty} \frac{\alpha_{j+1}}{\alpha_j} = \lim_{j \rightarrow \infty} \left(\frac{1}{2^{j+1}} \cdot \frac{2^j}{1} \right) = \frac{1}{2} \lim_{j \rightarrow \infty} \left(1 + \frac{1}{j} \right) < \frac{1}{2} < 1$$

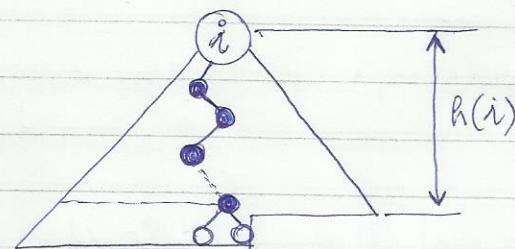
Slep: $\sum_{j=1}^{\infty} \frac{1}{2^j}$ konvergira.

Popravi-v-kopico(i, j) ... primerja in zamenjuje vozlišča s sinovi, vzdolž ene od vej, ki se začnejo v vozlišču i .



Ne nujno vzdolž cele veje.

DEF $T(i) :=$ število primerjav/zamenjav pri klicu Popravi-v-kopico(i, j)
Velja $T(i) \leq$ število notranjih vozlišč na najdaljši veji iz i -ja $\stackrel{\text{def.}}{=} h(i)$



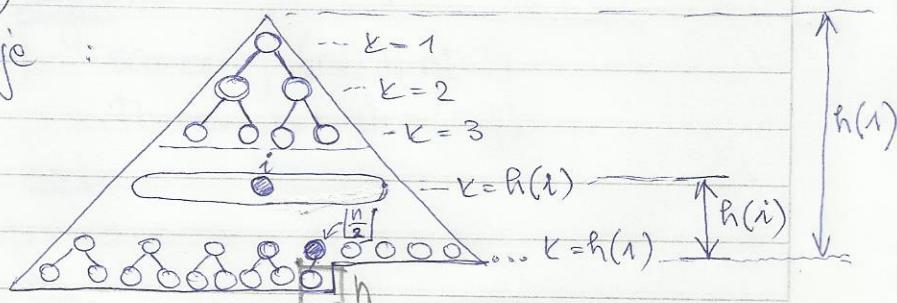
Procedura Sestavi-zacetno-kopico kliče Popravi-v-kopico(i, n)

v zanki pr. $i = \lfloor \frac{n}{2} \rfloor, \lfloor \frac{n}{2} \rfloor - 1, \dots, 2, 1$. Za to izvrši vsega skupaj
 $Z(n) = \sum_{i=\lfloor \frac{n}{2} \rfloor}^n T(i)$ primerjav/zamenjav.

? Kako hitro asimptotično narašča $Z(n)$?

Trditev: $Z(n) = \Theta(n)$

Dokaz: Uredimo nivoje :



Vsak nivo $1 \dots h(1)$ je poln in vsebuje 2^{k-1} vozlišč.

Nivo $h(1)$ vsebuje med 1 in $2^{h(1)-1}$ očetov.

Nivo $h(1)+1$ vsebuje med 1 in $2^{h(1)}$ listov.

Priprava:

$$\rightarrow \text{Opazimo: } \sum_{k=1}^{h(1)} 2^{k-1} = \underbrace{1+2+\dots+2^{h(1)-1}}_{\text{geometrijska}} = 2^{h(1)} - 1 < n$$

Zato je $2^{h(1)} \leq n$

\rightarrow Opazimo: če je i oče in je na nivoju k , je na višini $h(i) = h(1) + 1 - k$

Računamo:

$$Z(n) \stackrel{\text{def.}}{=} \sum_{i=[\frac{n}{2}]}^1 T(i) \leq \begin{cases} \text{seštevajmo po} \\ \text{vseh vozliščih} \\ \text{po nivojih} \\ h(1), \dots, 2, 1 \end{cases} \leq \sum_{k=h(1)}^1 2^{k-1} (h(1) + 1 - k) =$$

višina nivoja k
vseh vozlišč na nivoju k

ni samo enako, ker ne štejemo liste (nas ne zanimalo)

$$= \begin{cases} \text{vedemo} \\ j := h(1) + 1 - k \\ \text{in zato} \\ k-1 = h(1) - j \end{cases} = \sum_{j=1}^{h(1)} 2^{h(1)-j} \cdot j = 2^{h(1)} \sum_{j=1}^{h(1)} \frac{j}{2^j} \leq n \cdot \underset{\substack{\downarrow \\ \text{ker vsote}}}{\text{const.}} = O(n)$$

konstanta
sicer konverging

Skljep: Procedura Sestavi-zacetno-kopico ima časovno zahtevnost $O(n)$.

Celoten HeapSort ima časovno zahtevnost:

↳ sestavi začetno kopico (naloge $\Theta(n)$) ... $\Theta(n)$
 ↳ zanka:

for $r := n$ do while $r > 2$ do
 begin

zamenjaj ($a[1], a[r]$)
 Popravi-v-kopico ($1, r-1$)

end

$$\text{zahteva čas: } \sum_{r=n}^2 (1 + T(r-1)) =$$

$$= \sum_{r=1}^{n-1} (1 + T(r)) =$$

$$= n-1 + \sum_{r=1}^{n-1} T(r) \quad \rightarrow \text{visina drevesa}$$

$$< / T(r) \leq \lceil \log_2 r \rceil < \log_2 r + 1 / <$$

$$< n-1 + \sum_{r=1}^{n-1} (\log_2 r + 1) =$$

$$= 2(n-1) + \sum_{r=1}^{n-1} \log_2 r <$$

$$< 2(n-1) + \sum_{r=1}^n \log_2 r =$$

$$= 2(n-1) + \Theta(n \log n) = \underline{\Theta(n \log n)}$$

Heapsort: $\Theta(n)$

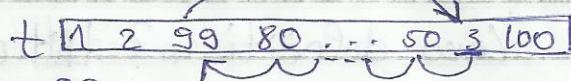
$\Theta(n \log n)$

Sklanjaj: $\Theta(n \log n)$

Quicksort

Motivacija:

Bubblesort

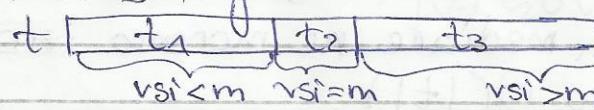


? Zaradi ne bi zamenjali 3 in 99 takoj?

Zamisel: dovoljno zamenjave oddaljenih elementov (pr. 3 in 99)

? Kako to zamisel kar se da izkoristiti?

Zamisel: ① tabelo prerazporedimo tako da bo "grubo" urejena, glede na neko delitno vrednost m, tako da bo sestavljena iz treh podtabel t_1, t_2, t_3 , vjer:



(ni potrebno, da so lepo urejeni, samo da so strepo maysi/vejji kot m)

② na podoben način uredimo še t_1 in $t_3 \Rightarrow$ rekurzija (DELI IN VLADAJ)

Osnova zgradbo algoritma:

procedure Quicksort (t) // vrne urejeno tabelo t

begin

| if t ima kvečjemu sam element then
| return (t)

else

begin

| $m :=$ izberi med elementi tabele t ali
| pa ga kar drugače izračunaj
| prerazporedi elemente tabele t v podtabele
| (konkatenacijo) t_1, t_2, t_3 glede m
| return (Quicksort(t_1) \oplus $t_2 \oplus$ Quicksort(t_3))

end

end

Opazimo: Odvisno od določanja m-ja je t_2 lahko tudi prazna tabela (Npr. če je bil m izračunan in $m \neq t$)

Kako določiti m? Hitro!! \Rightarrow v času $O(1)$ (konstanten čas)

Več možnosti:

- $m :=$ prvi el. tabele t
- $m :=$ srednji el. tabele t
- $m :=$ naključno izbrani el. tabele t
- $m := \left\lfloor \frac{\text{prvi} + \text{srednji} + \text{zadnji}}{3} \right\rfloor$

idealno: $m :=$ mediana tabele t

Toda, mediane ne moremo izračunati v $O(1)$, lahko pa jio v času $\Theta(|t|)$

? Kako hitro prerazporediti elemente tabele t v tabele t_1, t_2, t_3 ?

Ideje:

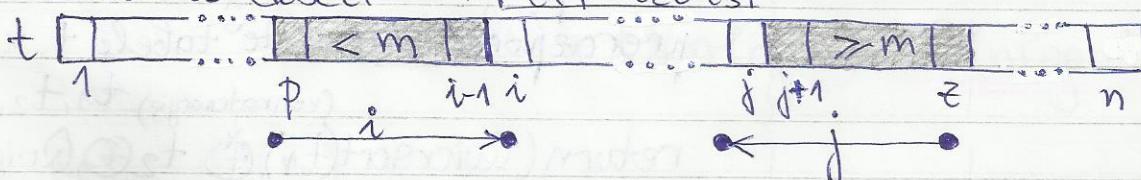
PRVA MOŽNOST: Najprej iz tabele t $\boxed{\text{?}}$

seskrivimo $\boxed{t_1 \ t_2 \ t_3}$

in nato iz tega $\boxed{t_1 \ t_2 \ t_3}$

DRUGA MOŽNOST: Ostanemo pri $\boxed{t_1 \ t_2 \ t_3}$ in si prikravimo iskanje meje med t_2 in t_3 in v glavnem algoritmu podlicemo $\text{return}(\text{Quicksort}(t_1) \oplus \text{Quicksort}(t_2 \cup t_3))$

? Kako to doseči? $\rightarrow \boxed{t_1 \ t_2 \ t_3}$



procedure Partition (t, p, z, m). /* prerazporedi elemente $t[p \dots z]$ glede na m pri čemer $1 \leq p \leq z \leq n$ */

begin $i := p; j := z;$

while $i < j$ do

begin

while ($t[i] < m \wedge i \leq z$) do $i++$;

while ($t[j] \geq m \wedge j \geq p$) do $j--$;

```

if (i < j) then
begin
    zamenjaj (t[i], t[j]);
    i++;
    j--;
end
end
return (t1, t2 ∪ t3) {① t1 = t[p...i-1]
                           ② t2 ∪ t3 = t[j+1...z]}

```

Partition(t₁, p, z, m) ima čas. zahtevnost $O(|t|)$

procedure Quicksort(t)

begin

if t ima kvečjeno en element then return(t)

else

begin

m := izberi ali izračunaj

porazdeli t v (t₁ in t₂ ∪ t₃) // Partition

return (Quicksort(t₁) + Quicksort(t₂ ∪ t₃))

end end

\downarrow
T(|t|)

\downarrow
T(|t₂ ∪ t₃|)

Naj bo $T(|t|)$ čas, potreben za ureditev tabele t s proceduro Quicksort.

Dolžimo enačbo:

$$T(|t|) = T(|t_1|) + T(|t_2 \cup t_3|) + \Theta(|t|) + \Theta(1)$$

To je rekurzivna enačba.

Resitev T je odvisna od tega kakšne so t₁ in t₂ ∪ t₃

$$T(|t|) = T(|t_1|) + T(|t_2 \cup t_3|) + \Theta(|t|) + \Theta(1)$$

urejanje t_1 urejanje $t_2 \cup t_3$ prerazporejanje izbiranje delitnega elementa

* Analiza za najslabši primer:

$T_{\max}(n)$... najslabši čas za urejanje, $n = |t|$

Trditev: $T_{\max}(n) = \Theta(n^2)$

Dokaz: Denimo, da vsako prerazporejanje (Partition) funkcija vrne najbolj nepravnoteženo tabelo:

t	$ t_1 $	$t_2 \cup t_3$
-----	---------	----------------

$$\text{Tedaj } T(n) = T_{\max}(|t_1|) + T_{\max}(|t_2 \cup t_3|) + \Theta(n) + \Theta(1)$$

$$\text{torej } T_{\max}(1) = T_{\max}(n-1) + \Theta(n)$$

Ocenjevanje reda velikosti $T_{\max}(n)$:

$$T_{\max}(n) = T_{\max}(n-1) + \Theta(n) =$$

$$= T_{\max}(n-1) + C_n \cdot n =$$

$$= T_{\max}(n-2) + C_n(n-1) + C_n \cdot n =$$

$$= \dots =$$

$$= C_1 + C_2 \cdot 2 + C_3 \cdot 3 + \dots + C_n \cdot n =$$

$$= \begin{cases} \leq C_{\max} (1+2+\dots+n) & \Leftrightarrow \frac{n(n-1)}{2} \\ \geq C_{\min} (1+2+\dots+n) \end{cases} = \Theta(n^2)$$

* Analiza za najboljši primer

$T_{\min}(n)$... najboljši čas za urejanje t ($|t| = n$)

Trditev: $T_{\min}(n) = \Theta(n \log n)$

Dokaz: Denimo, da je delitni element vedno mediana (trenutna tabela) t

t_1	$t_2 \cup t_3$
$\approx \frac{1}{2}n$	$\approx \frac{1}{2}n$

Tedaj glasi:

$$T_{\min}(n) = T_{\min}\left(\frac{n}{2}\right) + T_{\min}\left(\frac{n}{2}\right) + \Theta(n) + \Theta(1)$$

$$T_{\min}(n) = 2 \cdot T_{\min}\left(\frac{n}{2}\right) + \Theta(n)$$

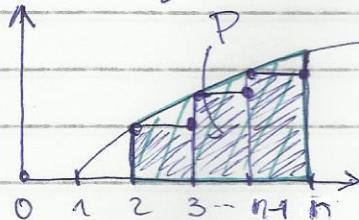
Rešitev je: $T_{\min}(n) = \underline{\Theta(n \log n)}$

GLEJ METODO
DELI IN VLADAJ

* Analiza za povprečni primer:

Priprava:

$$\sum_{i=2}^n i \ln i \leq \int_2^n x \ln x dx = \frac{x^2 \ln x}{2} - \frac{x^2}{4} - \text{konst.} \leq \frac{n^2 \ln n}{2} - \frac{n^2}{4}$$



Povprečni čas: $T_{ave}(n)$

Predpostavka 1: elementi tabele so paroma različni
(analiza ne bo preveč optimistična)

Velja: $T_{ave}(0) = T_{ave}(1) = b$

Definimo, da bi bil delilni element m i-ti po velikosti v tabeli $t \quad [t_1 \quad | \quad m; \quad t_2 \cup t_3]$

Eračba bi bila:

$$T(n) = T(i-1) + T(n-i) + \Theta(n)$$

Toda i je lahko $1, 2, 3, \dots, n$;

? Kako to upoštevati?

Naj bo $p(i)$ verjetnost da je m i-ti po velikosti.

$$\text{Te dan: } T_{ave}(n) = \sum_{i=1}^n p(i) \cdot [T_{ave}(i-1) + T_{ave}(n-i)] + \Theta(n)$$

Toda, koliko je $p(i)$?

Predpostavka 2: $p(i) = \frac{1}{n}, \forall i$

Sedaj, se da nadaljevati:

$$\begin{aligned} T_{ave}(n) &= \frac{1}{n} \sum_{i=1}^n [T_{ave}(i-1) + T_{ave}(n-i)] + \Theta(n) = \\ &= \frac{1}{n} [T_{ave}(0) + \dots + T_{ave}(n-1) + T_{ave}(n-1) + \dots + \\ &\quad + T_{ave}(0)] + \Theta(n) = \\ &= Cn + \frac{2}{n} \sum_{i=0}^{n-1} T_{ave}(i) \end{aligned}$$

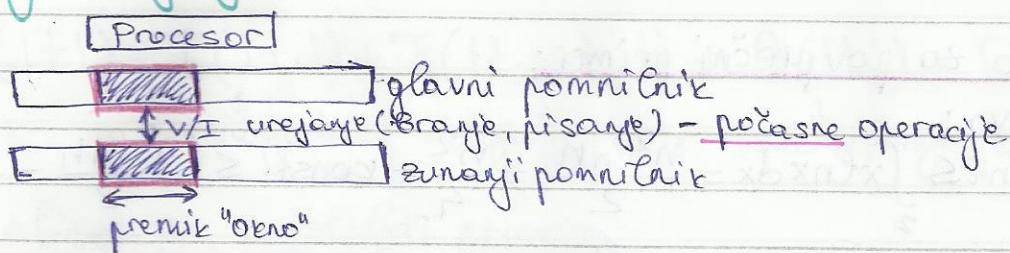
Trditev: $T_{ave}(n) \leq 2(B+c)n \ln(n)$

Doraz: indukcija po n

$$\begin{array}{l} \stackrel{n=2}{\leftarrow} \text{Trebimo pripraviti} \\ \stackrel{n=1}{\leftarrow} \text{Trebimo pripraviti} \\ \stackrel{n}{\sum} \text{Zbirni} \end{array}$$

} Torej $T_{ave}(n) = O(n \log n)$, ker
je $n \log n$ spodnja meja, je
torej $T_{ave}(n) = \underline{\Theta(n \log n)}$

Zunanje urejanje



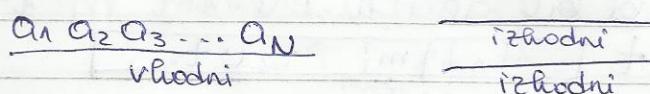
Algoritmi za znanje urejanje

navadno
zlivaje

izložjane: → ravnotežno
→ večsmerno
→ naravno
→ polifazno

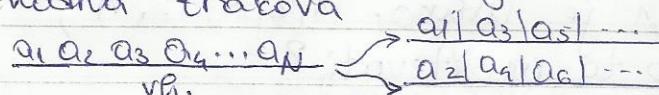
Navadno zlivanje (Merge Sort)

Dano: 3 trakovi (1 vhodni, 2 izhodna)



1. korak

→ Beri po vrsti podatke z vhodnega traku in ih porazdeli na izhodna trakova



→ izhodna trakova feri in vsaka prebrana elementa

zij v urejen par, tega pa izpiši na vi. trak

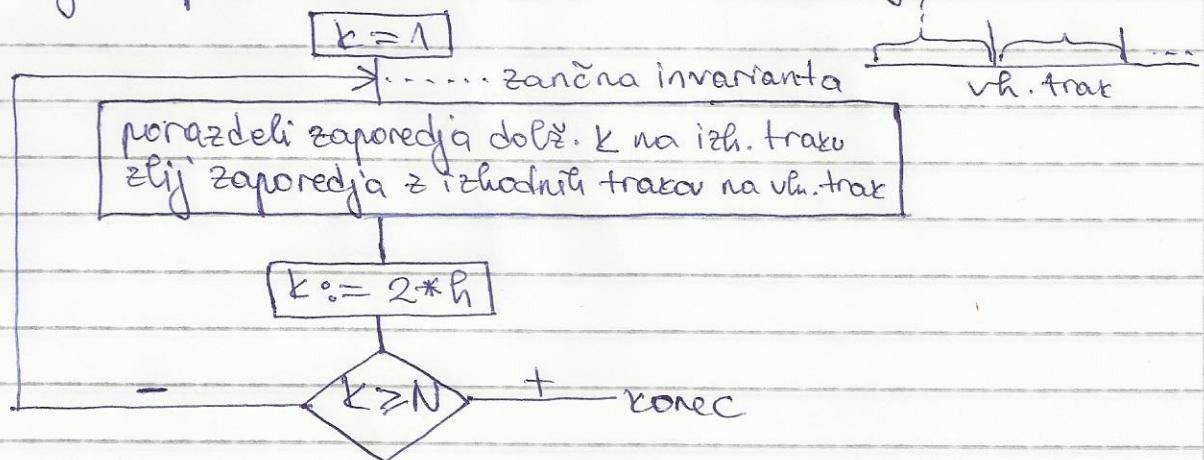


Splošni korak

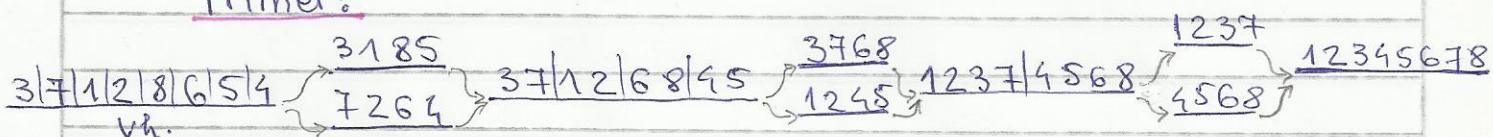
→ (Denimo, da so na vklj. traku že urejena zaporedja, dolžine k , $k \geq 1$)

- norazdeli ta zaporedja izmenično na dva izhodna trakova
- zlij po dve zaporedji (dolžine k) z obeli izhodnili trakovi v eno urejeno zaporedje dolžine $2k$ na vh. trak.

Diagram poteka:



Primer:



Zlivanje dveh k-teric: $|f_1, f_2, \dots, f_k|$
 $|c_1, c_2, \dots, c_k|$

Postopek:

- beri z izh. trakov prva elementa obeh zaporedij (a_1, b_1, c_1)
- (*) - izpiši manjšega od njiju na vln. trak (če sta enaka, je vseeno zadnega)

if izpisani ni zadnji v svoji k-terici
then

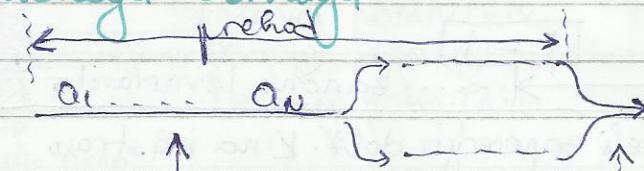
beri njegovega naslednika z izh. traku in se vrni na (*)

else

izpiši preostanek druge k-terice

Analiza navadnega zlivanja

Operacije:



Branje podatka
na zun. pomnilnik
(N branj)
št. podatkov

zapis podatka na
zun. pomnilnik
 $\frac{N}{2} + \frac{N}{2} = N$ branj

Zapisovanje na zun. pomnilnik je toliko kot branj.

Vsek prehod zahteva $2N$ zun. branj.

? Koliko prehodov je potrebnih?

Ker so po t. prehodu urejena zaporedja 2^x daljša.

Končamo pa ko so dolga $= N$, je potrebnih p prehodov, kjer je p najmanjše naravno število za katerega je $2^p \geq N \Rightarrow p = \lceil \log_2 N \rceil$

Časovna zahtevnost: $2N \lceil \log_2 N \rceil = \Theta(N \log N)$ zun. branj

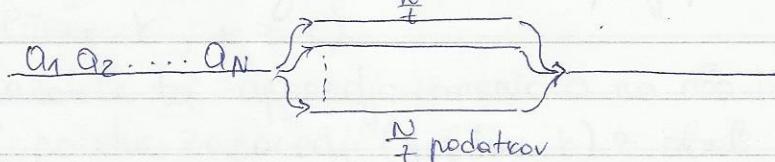
(fiba Mergesort-a: zahteva preveč dodatnega pomnilnika, ko je N zelo velik)

Izboljšave navadnega zlivanja

Večsmerno zlivanje (Multilevel Mergesort)

Za porazdeljevanje uporabimo več kot 2 trakova.

Naj bo t št. teh trakov.



Časovna zahtevnost:

$$\lceil N \left(\frac{\text{branj}}{\text{zun. pom.}} \right) + t \cdot \frac{N}{t} \rceil = 2N \text{ zun. branj na en prehod}$$

po vsekem prehodu bodo urejene r-terice t-krat daljše
 \Rightarrow potrebnih bo $\lceil \log_t N \rceil$ prehodov za ureditev vseh podatkov

Sleč: Časovna zahtevnost večsmernega urejanja je $2N \lceil \log_2 N \rceil$ zun. branj

Dohitrter glede na 2 trakova:

$$\frac{2N \lceil \log_2 N \rceil}{2N \lceil \log_2 N \rceil} \approx \frac{\log_2 N}{\log_2 N} = \frac{\log_2 N}{\frac{\log_2 N}{\log_2 t}} = \underline{\log_2 t}$$

Npr. če je $t=8$, je urejanje $\log_2 8 = 3$ krat hitrejše

Uравnoteženo zlivanje (Balanced Merging)

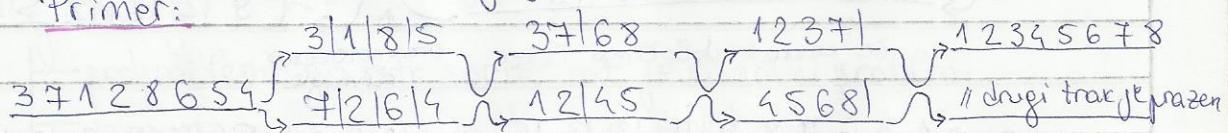
Opozimo pri navadnem zlivanju:

Porazdeljevajo ne permutira podatkov, zato ne prispeva direktno k končni urejenosti.

? Ali se lahko porazdeljevanju izognimo?

DA : Zamisel: namesto, da želite zaporedja izpisujemo na en sam trak, ih želimo izmenično izpisujemo na 2 trakova (porazdeljujemo karati ob zlivanju)

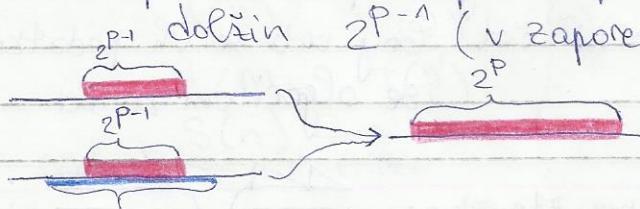
Primer:



Časovna zahtevnost: število zun. branj se prepolovi, ker ni več branj z izh. trakom (izjema je začetni del) $\Rightarrow N \lceil \log_2 N \rceil$

Naravno zlivanje

Opozimo: doslej zlivali zaporedja enakih dolžin (npr. pri $t=2$ smo v p-tem prehodu zlivali zaporedja dolžin 2^{P-1} (v zaporedja dolžine 2^P)



Toda, včasih je več kot le 2^{P-1} elementov že urejenih.

Zato je škoda, da tega ne izkoristimo!
Zamisel:

① DEF: Četa je urejeno podzaporedijo, ki se ga ne da razširiti, ne da bi pri tem izgubilo svojo urejenost.

$$a_{i-1} > a_i \leq a_{i+1} \leq \dots \leq a_j > a_{j+1}$$

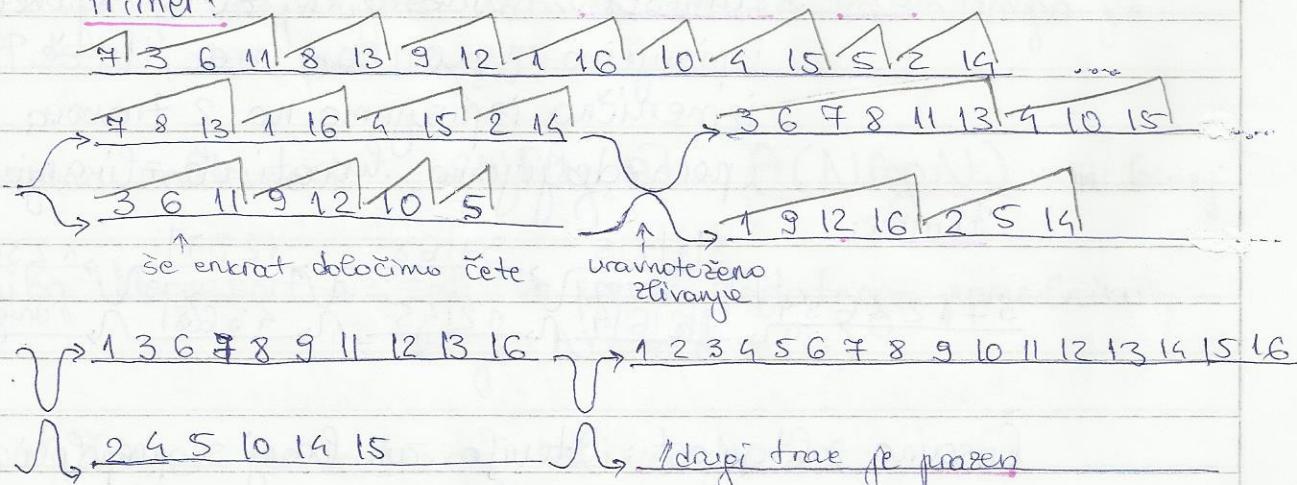
četa

Podzaporedje čete ni več četa

✓ → simbol čete

② Namesto z zaporedji predpisane velikosti se odslej "uvajamo" s četami: [jih porazdeljujemo
jih zlivamo

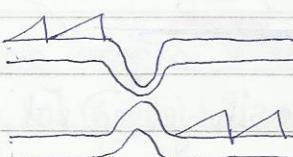
Primer:



Analiza:

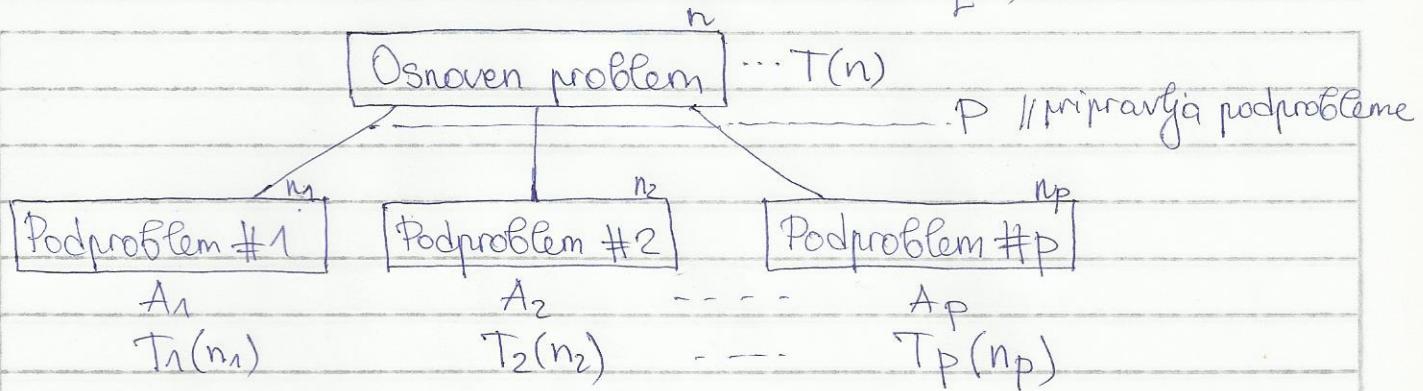
- ko zlijemo dve četi, nastane četa
- pri vsacem prehodu se št. čet prepelovi
- če je bilo na začetku c čet, bo potrebnih $\frac{c(c+1)}{2}$ prehodov da dobimo urejene podatke
- $1 \leq c \leq N$ čet, točka kot so vse podatki urejeni, hitreje se algoritmom konča

Vse izboljšave lahko združimo:



(polifazno
urejanje)

METODA DELI IN VLADAJ (Divide and conquer)



procedure A(n)

begin $P(n)$

$A_1(n_1) // \dots T_1(n_1)$

$A_2(n_2) // \dots T_2(n_2)$

\vdots

$f(n) // \text{samo za } P(n) \text{ in } S(n)$

$A_p(n_p) // T_p(n_p)$

$S(n) // \text{sestavi delne re\v{s}itve podproblemov v re\v{s}itev glavnega problema}$

end

$$T(n) = T_1(n_1) + T_2(n_2) + \dots + T_p(n_p) + f(n)$$

Toda: Pogosto je situacija manj splošna.

Pogosto je:

podproblemni so iste sorte kot je osnovni problem

$$n_1 \approx n_2 \approx \dots \approx n_p = \frac{n}{c}, c \in \mathbb{N}$$

re\v{s}iti je treba nek $a \in \mathbb{N}$ podproblemov ($1 \leq a \leq p$)

Primeri:

due polovici tabele

\leftarrow dvoji\v{s}ko iskanje: $p=2, a=1, c=2 \rightarrow$ iskanje po eni polovici \rightarrow razdelimo na 2

quicksort : $p=2, a=2, c=2$

Tedaj je algoritem (n):

procedure A(n) ... T(n)

begin $P(n)$

$$A\left(\frac{n}{c}\right) \cdots T\left(\frac{n}{c}\right)$$

$$A\left(\frac{n}{c}\right) \cdots T\left(\frac{n}{c}\right) f(n)$$

$$A\left(\frac{n}{c}\right) \cdots T\left(\frac{n}{c}\right)$$

$$S(n)$$

end \leftarrow if $n=1$ then

$$T(n) = \begin{cases} a \cdot T\left(\frac{n}{c}\right) + f(n), & \text{če } n > 1 \\ b (= \text{const.}), & \text{če } n = 1 \end{cases}$$

$a = \# \text{ podproblemov v jedi\v{s}ki resitvi iz vseh } p \text{ problemov}$
 $b \text{ je const.}, c \in \mathbb{N}, 1 \leq a \leq p$

$$T(n) = \begin{cases} b, & \text{če } n=1 \\ a \cdot T\left(\frac{n}{c}\right) + f(n), & n>1 \end{cases}$$

? Kako rešiti enačbo za $T(n)$ pri zanimivih $f(n)$?

Zanimive $f(n)$ so polinomi od n . Npr. $f(n)=n^r$, $r \in \mathbb{R}$

Reševanje pri $f(n)=n^r$

Naj bo $n=c^m$, za nek $m \in \mathbb{N}$

Enačba (*) se glasi: $T(c^m) = \begin{cases} b, & \text{če } m=0 \\ aT(c^{m-1}) + f(c^m), & m>0 \end{cases}$

Računamo za splošni $k>0$:

$$T(c^k) = a \cdot T(c^{k-1}) + f(c^k) / : a^k$$

$$\frac{T(c^k)}{a^k} = \frac{T(c^{k-1})}{a^{k-1}} + \frac{f(c^k)}{a^k} / S(k) := \frac{T(c^k)}{a^k}, g(k) = \frac{f(c^k)}{a^k}$$

$$S(k) = S(k-1) + g(k) / \sum_{k=1}^m \dots$$

$$S(m) = S(0) + \sum_{k=1}^m g(k) / m \text{ nam ostane, pomnožimo z } a^m$$

$$T(c^m) = a^m \left[\frac{T(c^0)}{a^0} + \sum_{k=1}^m \frac{f(c^k)}{a^k} \right] / c^m = n \Rightarrow m = \log_c n$$

$$T(n) = a^{\log_c n} \left[b + \sum_{k=1}^{\log_c n} \frac{f(c^k)}{a^k} \right] / a^{\log_c n} = \underbrace{(c^{\log_c a})^{\log_c n}}_a = \underbrace{(c^{\log_c n})^{\log_c a}}_n =$$

Dobimo rešitev (*):

$$T(n) = n^{\log_c a} \left[b + \sum_{k=1}^{\log_c n} \frac{f(c^k)}{a^k} \right], \text{ če } n>1$$

Če upoštevamo, da $f(n)=b n^r$ in $m=\log_c n$:

$$T(n) = b n^{\log_c a} \left[1 + \frac{c^r}{a} + \left(\frac{c^r}{a}\right)^2 + \dots + \left(\frac{c^r}{a}\right)^m \right]$$

? Kaj lahko povem o asimptotičnem vedenju tega izraza?

Možnosti:

$$\textcircled{A} \quad \frac{c^r}{a} = 1, \text{ Tedaj (D.N.) } \dots T(n) = \Theta(n^r \log n)$$

$$\textcircled{B} \quad \frac{c^r}{a} > 1, \text{ Tedaj (D.N.) } T(n) = \Theta(n^{r \cdot p})$$

$$\textcircled{C} \quad \frac{c^r}{a} < 1, \text{ Tedaj (D.N.) } T(n) = \Theta(n^{\log_c a})$$

Enačba (*) ima rešitev:

Master Theorem:

$$T(n) = \begin{cases} \Theta(n^r), & \text{če } a < c^r \\ \Theta(n^r \log n), & \text{če } a = c^r \\ \Theta(n^{\log_a r}), & \text{če } a > c^r \end{cases}$$

Primer 1: (Master Theorem) - Dvojiško iskanje

$$\left. \begin{array}{l} c=2 \\ a=1 \\ r=0 \end{array} \right\} \begin{array}{l} c^r - 2^0 = 1 \Rightarrow a = c^r \\ \rightarrow \text{ker je tabela sortirana} \end{array}$$

Čas dvojiškega iskanja: $\Theta(n^0 \log n) = \underline{\Theta(\log n)}$ Quicksort: $c=2$ $a=2$ $r=1$ (ker je prerazporejanje redke $\Theta(n)$)Čas zah: $\underline{\Theta(n \log n)}$

Množenje matrik

 A, B matriki reda $n \times n$ $C = A \times B$, je matrika s komponentami $C_{ij} = \sum_{k=1}^n A_{ik} B_{kj}$

$$i \left[\begin{array}{c} j \\ C_{ij} \end{array} \right] = i \left[\begin{array}{c} \parallel \\ k \end{array} \right] \cdot \left[\begin{array}{c} j \\ A_{ik} B_{kj} \end{array} \right]$$

Preprosti algoritem:

inicjalizacija $C := 0$ for $i := 1$ to n do for $j := 1$ to n do for $k := 1$ to n do //skalarni produkt za j -ti stolpec
 in i -ta vrstico

$$C_{ij} = C_{ij} + A_{ik} B_{kj}$$

Čas. zahitnost: $\left[\begin{array}{l} n^3 \text{ skalarnih množenj} \\ n^2(n-1) \text{ seštevanj} \end{array} \right] \underline{\Theta(n^3)}$? Ali bi se dalo izračunati C hitrej? (z manj množenji)

Shmuel Winograd (e. 1967) našel alg. ki zahiteva:

$$\left[\begin{array}{l} \frac{1}{2} n^3 + n^2 \text{ množenj} \\ (= \frac{1}{2} n^3 (1 + \frac{2}{n})) \approx 50\% \text{ manj pri velikih } n \end{array} \right]$$

$$\left[\begin{array}{l} n^3 + 3n^2 - 2n \text{ seštevanj} \end{array} \right]$$

Toda: asimptotično pa je Win.-alg. še vedno $\Theta(n^3)$.

(Win.-alg. ni primeren za "deli in vladaj").

? Ali se da množiti matrice v času, ki je asimptotično manjši od $\Theta(n^3)$??? (npr. v času $\Theta(n^\alpha)$, kjer $2 < \alpha < 3$)

Poskus izboljšanja z metodo deli in vladaj

Predpostavka: n je potenca števila 2.

Matrici A in B razdelimo v 4 podmatrike.

$$A \times B = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = C$$

$$\left. \begin{array}{l} C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} \\ C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22} \\ C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21} \\ C_{22} = A_{21} \times B_{12} + A_{22} \times B_{22} \end{array} \right\} \text{deli in vladaj}$$

Naj bo $T(n)$ čas za izračun produkta $A \times B$ matrik A in B reda $n \times n$

Za izračun vseh 4: $C_{11}, C_{12}, C_{21}, C_{22}$, je potrebeni:

→ 8 matričnih množenj reda $\frac{n}{2} \times \frac{n}{2}$

→ 4 matričnih sestevanj matrik reda $\frac{n}{2} \times \frac{n}{2}$

$$\text{Torej: } T(n) = 8 \cdot T\left(\frac{n}{2}\right) + 4 \cdot \frac{n}{2} \cdot \frac{n}{2} = 8T\left(\frac{n}{2}\right) + n^2$$

To pa je enačba, ki jo znamo asimptotično rešiti:

$$a = 8 ; \text{ (8t. podproblemov)}$$

$$c = 2 ; \text{ (problem je 2-krat manjši)}$$

$$b = 1$$

$$r = 2$$

$$\text{ker je } a > c^r \Rightarrow T(n) = \Theta(n^{\log_2 a}) = \Theta(n^3)$$

2.81 (Strassen)
(deli in vladaj ni izboljšalo)

$$\log_2 a < 3 \rightarrow \text{če bi bil } a = 7$$

Polifarno uređenje 14.11. APS

- zaporedni dostop hitrejši kot naveljavčni dostop
(disk) (RAM)

sortBenchmark.com → najhitrejše alg. za zvrn. vr.

- tukom algoritma se naredi:

- ispraznit samo 1 trak v enem koraku
- na koncu vidimo do starejših

→ vse čete up
enem traku
or. sorted

Primer 2-smerno: $N+1 = 3$

1 0 0 (~~bottom~~ Bottom-up) konec → začetek
 0 1 1
 1 0 2 $\downarrow +1$
 3 2 0 $\downarrow +2$
 0 5 3 $\downarrow +3$
 5 0 8 $\downarrow +5$
 1 3 8 0 $\downarrow +8$

Porazdeljenje novidezničnih čet

$$N+1 = 6$$

$t_1 \ t_2 \ t_3 \ t_4 \ t_5 \ t_6$
100 000

1 1 1 1 0
2 2 2 8 1 0
4 4 4 3 2 0
8 8 7 6 4 0

$t_1 \ t_2 \ t_3 \ t_4 \ t_5 \ t_6$
1 2 3 6 5
6 7 8 9 17
10 11 12 16

(13 14 15)
(18 18)

→ st. čet

→ gledamo kje
majka najde
čet

Wirth, Algorithm and Data Structures

Knuth, TAOCP, vol 3: Sorting and Searching

Wittner, Algorithms and Data Structures for
External Memory

Strassenovo množenje matrik

$$\underbrace{\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}}_{A_{n \times n}} \times \underbrace{\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}}_{B_{n \times n}} = \underbrace{\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}}_{C_{n \times n}}$$

(n je poteca št. 2, da lahko vsakič razpolovimo)

Strassenovo odkritje:

$$\begin{aligned} J_{11} &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ J_{12} &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ J_{13} &= (A_{11} + A_{12}) \times B_{22} \\ J_{14} &= A_{22} \times (B_{11} - B_{21}) \\ J_{21} &= (A_{22} + A_{11}) \times (B_{22} + B_{11}) \\ J_{22} &= (A_{21} - A_{11}) \times (B_{12} + B_{11}) \\ J_{23} &= (A_{22} + A_{21}) \times B_{11} \\ J_{24} &= A_{11} \times (B_{22} - B_{12}) \end{aligned}$$

Ključna za Strassenov algoritmom:

$$\begin{aligned} J_{11} &= J_{21} \\ \Rightarrow \text{f matričnih množenj} \end{aligned}$$

$$\begin{aligned} C_{11} &= J_{11} + J_{12} - J_{13} - J_{14} \\ C_{12} &= J_{13} - J_{24} \\ C_{21} &= J_{23} - J_{14} \\ C_{22} &= J_{21} + J_{22} - J_{23} - J_{24} \end{aligned}$$

\sum : f matričnih množenj matric reda $\frac{n}{2} \times \frac{n}{2}$
18 matričnih seštevanj

DEF: Torej: $T_{\text{stra}}(n)$... čas za izračun $A_{n \times n} \times B_{n \times n}$

$$\text{Tedaj: } T_{\text{stra}}(n) = f \cdot T_{\text{stra}}\left(\frac{n}{2}\right) + 18 \cdot \frac{n}{2} \cdot \frac{n}{2} = f \cdot T_{\text{stra}}\left(\frac{n}{2}\right) + 4,5n^2$$

Rošitev te enačbe je: $T_{\text{stra}}(n) = \Theta(n^{\log_2 7}) = \Theta(n^{2,80735})$
(Doraz: glej splošno rošitev in vzemti: $a=7, c=2, b=4, s=2$)

Psevdo: 1. A in B dopolni, da bosta reda n, ter $n=2^k$, $k \in \mathbb{N}$

function Str(A, B: matrix, n: integer) := matrix;

if $n=1$ then return $A \cdot B \hookrightarrow$ stvarno

else /* gre res za matriki A, B */

begin

razdeli A in B na bloke A_{ij}, B_{ij} ;

$\left\{ \begin{array}{l} S_{11} := \text{Str}(A_{11} + A_{12}, B_{12} + B_{22}, \frac{n}{2}) \\ S_{12} := \dots \\ \vdots \\ S_{24} := \text{Str}(A_{11}, B_{22} - B_{12}, \frac{n}{2}) \end{array} \right.$

7 res.
klicev

$C_{11} := S_{11} + S_{12} - S_{13} - S_{14}$

$C_{12} := \dots$

$C_{13} := \dots$

$C_{14} := \dots$

return C

end

- prevelika poraba prostora
- velika poraba časa
- Strassenov alg. je uporaben, če je n zelo velik in če je matrika posta (nimata veliko ničel)

Pomen je predvsem v teoriji izračunljivosti oz. rač. zahternosti (computability complexity)

Trenutno najhitrejši alg. za matrično množenje (asimptotično):
Winograd - Coppersmith ($O(n^{2.376})$)

$O(n^2 \log n)$?

Iškanje k -tega el. po velikosti

Dana je neurejena tabela t z $|t| = n$ elementi in k , kjer $1 \leq k \leq n$

Ugotovite:

→ zgornja meja za časovno zahtevost algoritma je $O(n \log n)$
 → spodnja meja: $\Omega(n)$

Zamisel (po metodi deli in vladaj)

- 1. tabelo razporedi, da dobis $t \begin{array}{|c|c|c|} \hline & t_1 & t_2 & t_3 \\ \hline m & < m & = m & > m \\ \hline \end{array}$
 $(m$ je delilni element)
- 2. če je $k \leq |t_1|$, je iskanje el. v t_1
 če $|t_1| < k \leq |t_1| + |t_2|$, je el. v t_2
 če $k > |t_1| + |t_2|$, je v t_3

Algoritem:

```

procedure Izberi( $k, t$ )
begin
    if  $|t| < \min$  then
        begin
            UrediT(); // uporabimo eden od alg. za sortiranje
            return ( $k$ -ti v  $t$ )
        end
    else
        begin
             $m :=$  določi delilni element
            prerazporedi  $t$  v  $t_1, t_2, t_3$  glede na  $m$ 
            if  $k \leq |t_1|$  then Izberi( $k, t_1$ )
            elseif  $k \leq |t_1| + |t_2|$  then return ( $m$ )
            else Izberi ( $k - |t_1| - |t_2|, t_3$ )
        end
    end

```

Poenostavimo:

```

procedure Naivnolzberi (v, t)           ... T(|t|)
begin
    if |t|=1 then return (edini el. v t)   ... O(1)
    else
        begin
            m := izberi delilni element      ... O(1)
            porazdeli t v t1, t2, t3 glede na m ... O(|t|)
            if v ≤ |t1| then Naivnolzberi (v, t1) ... T(|t1|)
            elseif v ≤ |t1| + |t2| then return (m) ... O(1)
            else Naivnolzberi (v - |t1| - |t2|, t3) ... T(|t3|)
        end
    end

```

Analiza najslabše čas. zahtevnosti Naivnega izbiranja

$T_{\max}(n)$ najslabša čas. zahtevost algoritma

Izditev: $T_{\max}(n) = \Theta(n^2)$

Dokaz: Pesimistični scenarij

\rightarrow m je vsakokrat največji v tabeli
 \rightarrow zato se t porazdeli v t [t₁ | t₂]

$$T_{\max}(|t|) = T_{\max}(|t_1|) + \Theta(|t|) + O(1)$$

$$T_{\max}(n) = T_{\max}(n-1) + \Theta(n)$$

; //enkrat smo naredili

$$T_{\max}(n) = \Theta(n^2) \quad // \text{slabo, želimo boljši } \Theta \text{ kot } n \log n, \text{ problem je viziranji m}$$

Izboljšani algoritem Izberi (Blum, Floyd, Pratt, Rivest, Tarjan)

Opozimo: [dobro je, da nobena od tabel t₁ in t₃ ni "prevelika"
 [dobro bi bilo, da se t porazdeli v t₁, t₂, t₃,
 vjer sta t₁ in t₃ "primerno" veliki

? Kako to doseči? (vaj je "primerno")

Odg: z drugačnim izbiranjem delilnega elementa m!

Izboljšani rekurzivni algoritem

Opozimo:

- dobro je, da nobena od t_1 in t_3 ni "pretirano" velika - hocemo da je primerno velica
- $$\rightarrow \max \left\{ \frac{|t_1|}{|t|}, \frac{|t_3|}{|t|} \right\} < q < 1$$

Takrat se bo osnovni problem nadomestil s primerno manjšim podproblemom.

? Kako to doseči?

Odgovor: z drugačnim izbiranjem m-jä

Izbiraje m:

- ① t razdelimo na peterke
- ② v vsaki peterki poiščemo njeno mediano (s el. po velikosti)
(npr. tako, da peterko uredimos)
- ③ naj bo M tabela vseh teh median

$$m := \text{Izberi}\left(\frac{|M|}{2}, M\right) \quad (\text{m je mediana median peterk})$$

$$\textcircled{1} \quad t \square \boxed{x} | \boxed{x} | \boxed{x} | \dots | \boxed{x}$$

$$\textcircled{2} \quad M \boxed{x} \times \boxed{x} \dots \boxed{x} \leftarrow v M \text{ je } \lceil \frac{|t|}{5} \rceil \text{ elementov}$$

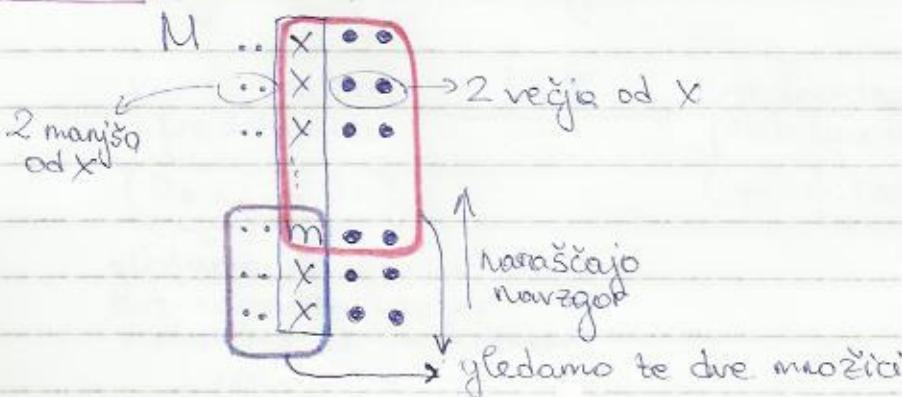
$$\textcircled{3} \quad m$$

? Kaj primaša dobrega tak m?

Trditev: Če je m mediana median, potem po prerazporeditvi tabele t (glede na m) velja:

$$|t_1| \leq \frac{3}{5}|t| \text{ in } |t_3| \leq \frac{3}{5}|t|$$

Dovaz: (nestaver)



Analiza čas. zahtevnosti algoritma Izberi

Oznaka: $T(n)$... čas za iskanje k -toga elementa v tabeli velikosti n z izboljšanim algoritmom

Trditev: $T_{\max}(n) = \Theta(n)$

Dokaz: $T_{\max}(n) = \Theta(n)$... za izračun $\lceil \frac{n}{5} \rceil$ median petek
 $+ T_{\max}\left(\frac{n}{5}\right)$

↳ št. elementov v tabeli M

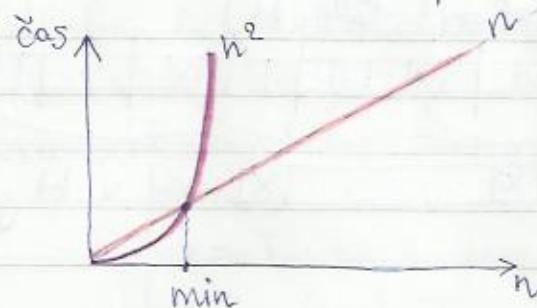
+ $\Theta(n)$... za prerazporejanje tabel ?

+ $T_{\max}\left(\frac{3}{4}n\right)$... za najslabši del: t_1 ali t_3

$$\Rightarrow T_{\max}(n) \leq T_{\max}\left(\frac{n}{5}\right) + cn + T_{\max}\left(\frac{3}{4}n\right)$$

Če je $n < \min$, rešujemo direktno

Za \min vzamemo vrednost, do katere je enostavno urejajče še vedno filtrirje od linearne asimptotične funkcije



$$T_{\max}(n) \leq \begin{cases} \text{const. } n, & n < \min \\ T_{\max}\left(\frac{n}{5}\right) + T_{\max}\left(\frac{3}{4}n\right), & \text{sicer} \end{cases}$$

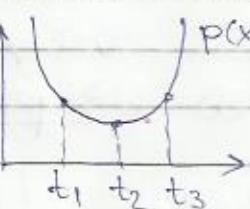
Trditev: $T_{\max}(n) \leq k \cdot n = \Theta(n)$

Motivacija

Dana polinoma: $p(x) = \sum_{i=0}^{77} a_i x^i$ in $q(x) = \sum_{i=0}^{34} b_i x^i \rightarrow$ koeficientna predstavitev polinoma
 Koliko je: $r(x) = p(x)q(x)$?
 $r(x) = \left(\sum_{i=0}^{77} a_i x^i\right) \cdot \left(\sum_{i=0}^{34} b_i x^i\right) = \sum_{i=0}^{94+77} c_i x^i$

Za izračun vseh c_0, c_1, \dots, c_{171} so potrebeni veliko množenj.
 Vemo: polinom stopnje n je povsem določen z vrednosti v $n+1$ različnih točkah (vrednostna predstavitev polinoma).

Npr.



$$(p(t_1), p(t_2), p(t_3))$$

Zanisel: Če bi bila $p(x)$ in $q(x)$ vrednostno predstavljena

v točkah t_1, t_2, \dots, t_{172}

$$\text{t.j. } p(x) = (p(t_1), p(t_2), \dots, p(t_{172}))$$

$$q(x) = (q(t_1), q(t_2), \dots, q(t_{172}))$$

potem bo vrednostno predstavitev njunega produkta izračunalni enostavno:

$$p(x)q(x) = (p(t_1)q(t_1), p(t_2)q(t_2), \dots, p(t_{172})q(t_{172}))$$

Toda: kaj če $p(x)$ in $q(x)$ nista podana v vrednostni obliki?

? Ali in kar lahko (hitro) sestavimo vrednostno predstavitev iz koeficiente?

? Ali in kar lahko sestavimo koeficientno predstavitev iz vrednostne?

? Karšna naj toda števila t_1, t_2, \dots, t_{172} ?

Odgovor na to daje: DFT

koeficientna
predstavitev

$$(a_0, \dots, a_n)$$

glejamo na temu

pot navaden vektor
sestavljen iz koeficiente
polinoma

DFT

vrednostna
predstavitev

$$(p(t_1), p(t_2), \dots, p(t_n))$$

Discretna Fourierova transformacija (DFT) - splošno

28. 11. 2013

Naj bo dan vektorški prostor V_K nad obsegom K



... Abelova grupa $(V, +)$



... obseg $(K, +, *)$
↳ sestevanje vektorjev
↳ neutralni el. so $*: 1$ (enota)
↳ sestevanje skalarjev
↳ neutralni el. so $+ : 0$

Naj bo n dimenzija V_K .

DEF: Naj bo $v \in V$ elem. w z lastnostmi:

$$a) w^n = 1$$

$$b) w^1, w^2, \dots, w^{n-1} \text{ vsi } \neq 1$$

Tedaj rečemo, da je w n -ti primitivni koren enote.

Lastnosti w :

$$a) 1$$

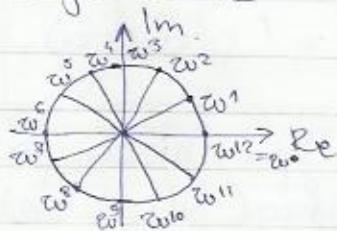
$$b) \Rightarrow \sum_{j=0}^{n-1} w^{pj} = \begin{cases} n, & \text{če } p=n \\ 0, & \text{sicer} \end{cases}$$

Dovaz: (D.N.)

Primer: $K \subseteq \mathbb{C}$ (mn. kompleksnih števil)

$$\text{Tedaj je } w = e^{i \cdot \frac{2\pi j}{n}} \quad (j = 0, 1, \dots, n-1)$$

Npr., če je $n=12$



DEF: DFT stopnje n prostora V_K je linearna transformacija tega prostora. Predstavljena je z matiko F , katere komponente F_{ij} so: $F_{ij} = w^{ij}$ ($0 \leq i, j \leq n-1$) in je w n -ti primitivni koren enote.

Opomba: $w^{ij} = w^{ij \bmod n}$



Matrika F predstavlja preslikavo enega elementa v drugega

? Ali obstaja tudi F^{-1} (inverzna matriča)?

Trditev: Če k vsebuje element n^{-1} (glede na Vič) potem F^{-1} obstaja in velja:

$$F_{ij}^{-1} = \frac{1}{n} \cdot w^{-ij} \quad (F - \text{Fourierjeva matriča})$$

Primer: $n=2, r \in \mathbb{C}$

$$\text{Tedaj: } w = -1 \quad F = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, F^{-1} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Primer: $n=4, r \in \mathbb{C}$

$$\text{Tedaj: } w = e^{i \frac{2\pi}{4}} = e^{i \frac{\pi}{2}}$$

$$F = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 \\ 1 & w^2 & 1 & w^2 \\ 1 & w^3 & w^2 & w \end{bmatrix} \quad F^{-1} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w^{-1} & w^{-2} & w^{-3} \\ 1 & w^{-2} & 1 & w^{-2} \\ 1 & w^{-3} & w^{-2} & w^{-1} \end{bmatrix}$$

$$\downarrow \quad \quad \quad \downarrow$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w & -1 & -w \\ 1 & -1 & 1 & -1 \\ 1 & -w & -1 & w \end{bmatrix} \quad \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -w & 1 & w \\ 1 & -1 & 1 & -1 \\ 1 & w & 1 & -w \end{bmatrix}$$

Npr. če $a = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \end{bmatrix}$, potem je $F_a = \begin{bmatrix} 6 \\ -1 \\ -w \\ -1+w \end{bmatrix}$

DFT in predstavitev polinomov

V k... vektorski prostor polinomov sprem. X (n -dimensionalni)

Baza: $x^0, x^1, x^2, \dots, x^{n-1}$

S to bazo lahko izrazimo vsak polinom stopnje $\leq n-1$

$$\sum_{i=0}^{n-1} a_i x^i$$

Trditev: DFT stopnje n preslikava koeficientno predstavitev polinoma v vrednostno predstavitev polinoma na točkah $t_i = \omega^n i$, $i = 0, 1, \dots, n-1$

Dokaz: Naj bo dan poljuben polinom $p(x) = \sum_{i=0}^{n-1} a_i x^i$
koeficientna pred: $p(x) = (a_0, a_1, \dots, a_{n-1}) = a$

Računajmo:

$$\begin{aligned} f_a &= \begin{bmatrix} 1 & \omega^n & \omega^{2n} & \dots & \omega^{(n-1)n} \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & \omega^n & \omega^{2n} & \dots & \omega^{(n-1)n} \\ \sum_{j=0}^{n-1} a_j \omega^{ij} & \sum_{j=0}^{n-1} a_j \omega^{(i+1)j} & \dots & \sum_{j=0}^{n-1} a_j \omega^{(i+n-1)j} \end{bmatrix} = \begin{bmatrix} p(\omega^n) \\ p(\omega) \\ \vdots \\ p(\omega^{n-1}) \end{bmatrix} \rightarrow O(n^2) \end{aligned}$$

↑
vrednostna pred. $p(x)$
v točkah $\omega^n i$, $i = 0, 1, \dots, n-1$

Hitra Fourierjeva transformacija - algoritem FFT

1 DEF: Naj bo $n = 2 \cdot r$ in ω n -ti primitivni koren enote
Problem DFT(a, n)... Iz dane koeficientne
predstavitev $a = (a_0, \dots, a_{n-1})$ polinoma $p(x) = \sum_{i=0}^{n-1} a_i x^i$
izračunaj vrednosti $p(\cdot)$ v točkah $\omega^0, \omega^1, \dots, \omega^{n-1}$
(Splošno: izračunaj DFT stopnje n vektorja a)

1 Pripravljamo "teren" za metodo deli-in-vladaj

2 Koeficienti ob "sodih" potencah: a_0, a_2, \dots, a_{n-2} (vseli je $r = \frac{n}{2}$)
koef. ob "lihih" potencah: a_1, a_3, \dots, a_{n-1} (vseli je $r = \frac{n}{2}$)

Sestavimo nova polinoma:

$$P_0(x) \stackrel{\text{def}}{=} a_0 + a_2 x + a_4 x^2 + \dots + a_{n-2} x^{r-1} \stackrel{\text{koef. oblika}}{=} (a_0, a_2, a_4, \dots, a_{n-2}) \equiv f_0$$

$$P_1(x) \stackrel{\text{def}}{=} a_1 + a_3 x + a_5 x^2 + \dots + a_{n-1} x^{r-1} \stackrel{\text{koef. oblika}}{=} (a_1, a_3, a_5, \dots, a_{n-1}) \equiv f_1$$

3 $\phi(x)$ lahko izrazimo $\Rightarrow P_0(x)$ in $P_1(x)$ formole:

$$\phi(x) = P_0(x^2) + x \cdot P_1(x^2)$$

4 Problem DFT(a, n) t.j. izračunaj $p(\cdot)$ v n točkah w^0, w^1, \dots, w^{n-1} se razdeli na dva podproblema:

4.1 izračunaj vrednosti dveh polinomov $p_0(\cdot)$ in $p_1(\cdot)$ v n točkah $(w^0)^2, (w^1)^2, \dots, (w^{n-1})^2$

4.2 Iz dobavljenih vrednosti sestavi na osnovi 3) vrednosti $p(\cdot)$ v točkah w^0, w^1, \dots, w^{n-1}

Toda, 4.1 lahko pospešimo. Kako?

$$5 \text{ Velja } (w^{k+r})^2 = w^{2k+2r} \stackrel{k=\frac{n}{2}}{=} w^{2k+n} = w^{2k} \cdot w^n = w^{2k} \text{ polovica}$$

6 To pomeni, da med n točkami $(w^0)^2, (w^1)^2, (w^2)^2, \dots, (w^{r-1})^2, (w^r)^2, \dots, (w^{n-1})^2$, po dve sta enaki (pr. $(w^r)^2 = (w^{r+r})^2$)

Zato je le $\frac{n}{2} = r$ različnih točk v zaporedju pri 4.1. Zato bo treba $p_0(\cdot)$ in $p_1(\cdot)$ računati v $\frac{n}{2}$ točkah:

$$(w^0)^2, (w^1)^2, \dots, (w^{r-1})^2 \\ (w^r)^2, (w^{r+1})^2, \dots, (w^{n-1})^2 \quad \rightarrow \text{(obrnemo eksponente)}$$

7 Ker je w n -ti primitivni koren enote, je $w^2 \frac{n}{2} = r$ -ti koren enote.

$$\text{Doraz: (za C)} \cdot w = e^{i \frac{2\pi}{n}}, \text{ zato je } w^2 = e^{i \frac{2\pi}{\frac{n}{2}}} = e^{i \frac{4\pi}{n}} \text{ Qed}$$

Pišimo: $\Psi := w^2$. (Torej Ψ je r -ti primitivni koren enote)

8 Če upoštevamo 6, 7, se 4.1 glasi tako:

8.1 Izračunaj $p_0(\cdot)$ in $p_1(\cdot)$ v r točkah $\Psi^0, \Psi^1, \dots, \Psi^{r-1}$

Isti kot karac 4, samo z drugimi parametri. Torej 8.1 je sestavljena iz dveh problemov:

DFT(J_0, r)

DFT(J_1, r)

9 Sedaj je odprta možnost za rekurzivno reševanje DFT(a, n) (Deli in vladaj)

II Učinkovitost tega računanja bo odvisna od učinkovitosti točke 4.2 t.j. od sestavljanja delnih rešitev v končno.

10 V 4.2 je treba iz vrednosti $p_0(\cdot)$ in $p_1(\cdot)$ sestaviti vse vrednosti $p(w^0), p(w^1), \dots, p(w^{r-1}), p(w^r), \dots, p(w^{n-1})$.

11 Prva polovica so $p(w^k)$, druga pa $p(w^{k+r})$, kjer $k=0, 1, \dots, r-1$.

12) Računanje teh $p(\omega^k)$ teče po ③:

$$p(\omega^k) \stackrel{③}{=} p_0(\omega^{2k}) + \omega^k p_1(\omega^{2k}) \stackrel{⑧.1}{=} p_0(\psi^k) + \omega^k p_1(\psi^k)$$

sta izračunane po ver. bližih

Za izračun vseh r vrednosti $p(\omega^k)$ bo potrebenih r množenj in r seštevanj

13) Pri računanju druge polovice: tj. vrednosti $p(\omega^{k+r})$
 $k=0, \dots, r-1$ sploh ni treba množiti

$$\begin{aligned} \text{Dokaz: } p(\omega^{k+r}) &\stackrel{③}{=} p_0(\omega^{2(r+k)}) + \omega^{r+k} \cdot p_1(\omega^{2(r+k)}) = \\ &\stackrel{⑤}{=} p_0(\omega^{2k}) + \omega^{r+k} \cdot p_1(\omega^{2k}) \stackrel{⑦}{=} \\ &= p_0(\psi^k) + \omega^{r+k} \cdot p_1(\psi^k) = \\ &= / \text{velja: } \omega^{r+k} = -\omega^k / = \\ &= p_0(\psi^k) - \omega^k p_1(\psi^k) \quad \text{zato je izračunati} \end{aligned}$$

Za vse to je potrebnih r odštevanj.

14) Tačka ④.2 lahko izvedemo z $r=\frac{n}{2}$ množenji (in $2r=n$ odštevanj)

Časovna zahtevnost in pseudokoda algoritma FFT(a,n)

procedure FFT(a,n) T(n)

begin

if $n=1$ then return a

else

begin

Prpravi $\omega, \delta_0, \delta_1$; //glej ② trivialno

FFT($\delta_0, \frac{n}{2}$); T($\frac{n}{2}$)

FFT($\delta_1, \frac{n}{2}$); //glej ⑧.1 T($\frac{n}{2}$)

Sestavi $p(\cdot)$ v vseh točkah; // ⑩-⑯ Θ(n) ... $\frac{1}{2}n$

end

end

časovna zahtevnost: $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n)$

Rešitev je funkcija $T(n) = \Theta(n \log n)$ (glej Master Theorem)

// FFT rešuje DFT

// Prikaz s FFT: $\frac{n^2}{n \log n} = \frac{n}{\log n} = / n=1000 / = \frac{1000}{10} = 100$

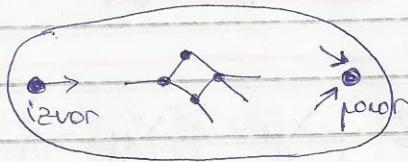
PRETOKI

Optimizacijski problem:

Problem max przetoka skozi omrežje (algoritem Ford-Fulkerson)

Opis:

- omrežje
 - izvor, ponor
 - tekočina se v omrežju ne izgublja
 - bar priteče iz izvora, odteče v ponor
 - celoten (maksimalni pretok odvisen od omrežja)
 - ? kolikšen je max možni pretok skozi dano



Natancneje:

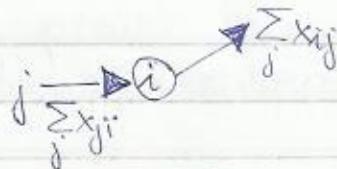
L Dan je označen usmerjen graf $G(V, E)$

povezava $(i,j) \in E$ ima kapaciteto $c_{ij} \in Q$

$\rightarrow E \subseteq V \times V$
 (usmerjene povezave)
 vozlišča $\{1, 2, \dots, n\}$
 izvor \downarrow ponor \downarrow

Lastnosti:

$$\text{① } 0 \leq x_{ij} \leq c_{ij}$$



$$\text{② } \sum_j x_{ji} - \sum_j x_{ij} = \begin{cases} 0, & \text{če } i \neq 1, n \\ -v, & \text{če } i=1 \\ v, & \text{če } i=n \end{cases}$$

Množica vrednosti x_{ij} , $i, j \in V$ (pretor)

? Količen je V_{\max} ?

Pocas algoritma:

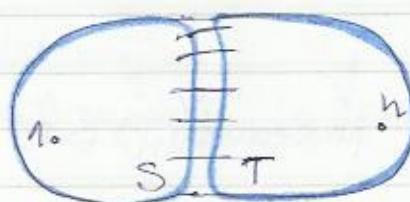
DEF: (S, T) je $(1, n)$ - prevez grafa $G(V, E)$,

če $S \cap T = V$

$S \cap T = \emptyset$

$1 \in S$

$n \in T$



graf prenežemo na 2

DEF: Kapaciteta preresa (S, T) je $c(S, T) \stackrel{\text{def}}{=} \sum_{i \in S, j \in T} c_{ij}$

največ koliko lahko teče promet iz $1 \sim n$ preko preresa (več ne more)

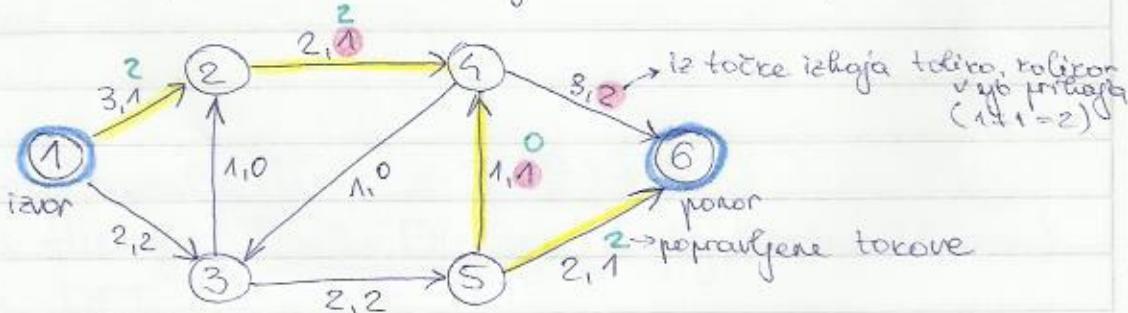
Izrek: Naj bo v celoten pretor pri neki prerazporeditvi (x_{ij}) in naj bo (S, T) ner prevez grafa G .

Tedaj $v \leq c(S, T)$.

Posledica: $V_{\max} \leq V_{\min} c(S, T)$

Težava: vseh prevez (S, T) je veličina (eksponentna)

Primer:



12. 12. 2013

Pojmi: Naj bo P zaporedje povezav iz 1 do n :



Izrek: Vse "poti" iz izvora do ponora so zasičene \Leftrightarrow pretok maksimalen

Zamisel: za samo metodo: po vrsti išči poti ki niso zasičene, vsako nato zasiti.

Algoritem: osnovna orodja (pojni)

Ideja: - začnimo v izvoru in postopoma označuj vozlišča kar ne označimo ponora

- označi naj bodo tare, da bo iz njih možno razkriti (rekonstruirati) nezasičeno pot
- odkrito nezasičeno pot zasiti

Izvedba:

vozlišče

neoznačeno

označeno [nepregledano (ima še neoznačenega sosedja)]

[pregledano (vsi sosedje so označeni)]

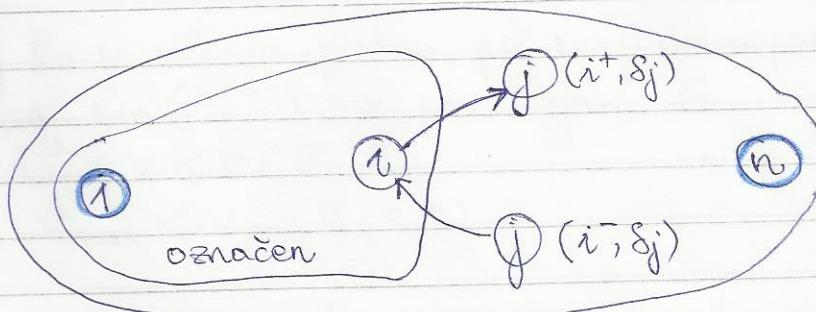
Začetek: izvor naj bo označen - nepregledan $(-, \infty)$

ostala vozlišča neoznačena

predhodnik ob potico latinsko iz bog. slamo

Označevanje: Naj bo i označeno - nepregledano in naj bo j sosed (od i-ja), ki je neoznačen.

Vozlišču j bomo podolili oznako, ki pa je odvisna od smeri povezave med i in j:



Prva komponenta oznake bo omogočila, da izsledimo predhodnika vozlišča j:

s_j pove, da bi vzdolž "poti" $\dots - i - j$ lahko povečali za s_j .

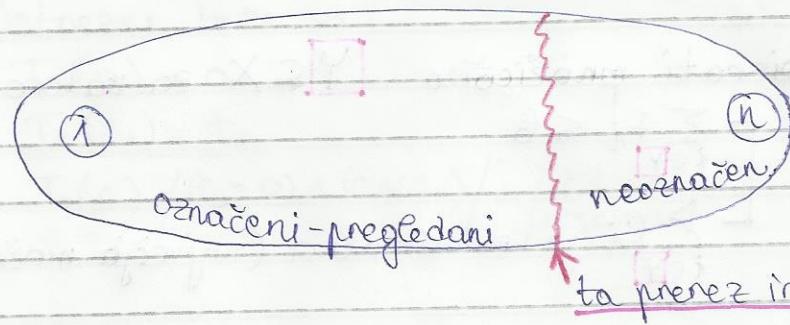
s_j izračunamo sproti pri j tacole:

$$s_j = \begin{cases} \min \{ s_i, c_{ij} - x_{ij} \}, & \text{če } i \rightarrow j \\ \min \{ s_i, x_{ji} \}, & \text{če } i \leftarrow j \end{cases}$$

Vidimo: Ko dosežemo $\langle n \rangle$ in je v njegovi oznaki (\leftarrow, S_n)

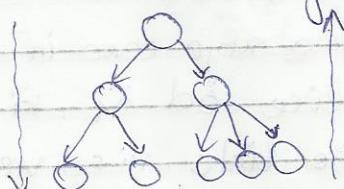
Če je $S_n > 0$, pomeni da pretok po poti ki jo rekonstruiramo preko \leftarrow , lahko povečamo za S_n .
Pot torej zasitim.

Konec izvajanja: Če ne moremo označiti ponora $\langle n \rangle$.
Kdaj je to?

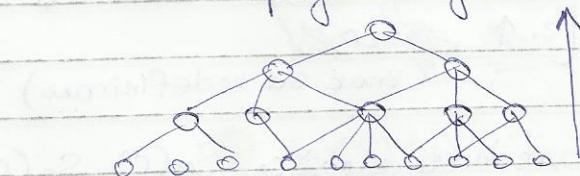


Dinamično programiranje

Deli in vladaj:



Dinamično programiranje:



- za reševanje optimizacijskih problemov
↳ načelo optimalnosti

Naj bo D_1, D_2, \dots, D_n zaporedje odločitvi pri iskanju optimalne rešitve. Pravimo da je zaporedje optimalno, če vrne optimalni rezultat.

Vsako podzaporedje optimalnega zaporedja je optimalno.

Problem Nahrbtnika

? Kako nahrbtnik z omejeno nosilnostjo napolniti s čim bolj vrednim plenom?

DEF:

- $X = \{1, 2, \dots, n\}$ --- množica predmetov
- vsak predmet i ima svojo ceno, $c_i \in \mathbb{N}$
- vsak predmet ima svojo težo, $t_i \in \mathbb{N}$
- dana je "nosilnost" $b \in \mathbb{N}$

Naloga: Poiscati podmnožico $Y \subseteq X$ za kateno je:

$$\sum_{i \in Y} t_i \leq b$$

$\sum_{i \in Y} c_i$ je maksimalna (največja močna)

Zamisel: - Naj bo $C = \sum_{i \in X} c_i$ (severna vrednost vseh predmetov)

- Naj bo $c \in [0, C]$
- Naj bo $X_i = \{1, 2, \dots, i\}$
- Vsaka podmnožica od X_i ima svojo ceno in težo.

Opozijemo le tiste podmnožice od X_i , ki so težke $\leq b$. Med temi pa še tiste, ki so vredne raven c

Def: $S_i(c) \stackrel{\text{def.}}{=} \begin{cases} \text{najlažja med podmnožicami množice } X_i, \\ \text{ki so vredne } = c \text{ in težke } \leq b \\ 2. \uparrow, \text{sicer} \\ \hookrightarrow (\text{znak za ne definirano}) \end{cases}$

- Algoritem: računaj vrednosti/množice $S_n(C)$, $S_{n-1}(C-1)$, $S_{n-2}(C-2)$... in končaj pri prvi množici ki je definirana.

ta množica je rešitev našega problema $S_n(C^*)$

? Kako izračunat $S_i(c)$ in $T_i(c)$?

Odg: Po naraščajočih $i: i=1, 2, 3, \dots, n$

$i=1$ $S_1(0) = \emptyset$ (ker \emptyset je edina podmnožica mn. $X_1 = \{1\} \neq$ vrednostjo 0)

$$\underline{S_1(c_1) = \{1\}}$$

$S_1(c) \uparrow$, če $(c > 0) \wedge (c \neq c_1)$

Ustrezne teže so:

$$T_1(0) = 0$$

$$\underline{T_1(c_1) = t_1}$$

$$\underline{T_1(c) / (c > 0) \wedge (c \neq c_1) / \vdash / \text{po dogovoru} / = 1 + \sum_{i=1}^n t_i}$$

ker je $S_1(c) \uparrow$ (nedef.)
(da lahko izrazimo računanje)

$i \geq 2$ Izrazimo S_i s S_{i-1} in T_i s T_{i-1} . Kako?

Recimo, da je $S_i(c)$ že izračunan. Ta množica bodisi vsebuje element i ali pa ga ne.

Ⓐ $i \in S_i(c)$. Tedaj $S_i(c)$ sestavlja i ter najlažja podmnožica od $X_{i-1} = \{1, 2, \dots, i-1\}$, ki je vedno $c - c_i$

$$\text{Torej: } S_i(c) = \{i\} \cup S_{i-1}(c - c_i)$$

$$\text{Tedaj: } T_i(c) = t_i + T_{i-1}(c - c_i)$$

$$\text{Vse velja če: } c - c_i \geq 0, S_{i-1}(c - c_i) \downarrow \text{definirana množica}$$

$$t_i + T_{i-1}(c - c_i) \leq b$$

Ⓑ $i \notin S_i(c)$ Tedaj je očitno $S_i(c) = S_{i-1}(c)$
in $T_i(c) = T_{i-1}(c)$

Toda za $S_i(c)$ mora biti izbrana lažja od obet možnosti A,B

To bo povedala teža sestavljenje $S_i(c)$:

$$\underline{T_i(c) = \min \{ T_{i-1}(c), t_i + T_{i-1}(c - c_i) \}}$$

Dinamična implementacija računa množice $S_i(c)$ od "spodaj navzgor":

najprej $S_{i-1}(\cdot)$ $T_{i-1}(\cdot)$

in nato $S_i(\cdot)$ $T_i(\cdot)$ na podlagi pravil

To napreduje po naraščajočih i.

Algoritem:vhod X, n, t_i, c_i, b , za vše $i=1, \dots, n$ izhod Y, C^*

metoda (dinamično programiranje)

begin

$$C := \sum_{i=1}^n c_i$$

for $c := 0$ to C do

$$S_i(c) := \uparrow$$

$$T_i(c) := 1 + \sum_{i=1}^n t_i$$

end

$$S_1(0) := \emptyset; T_1(0) := 0;$$

$$S_1(c_1) = \{i\}; T_1(c_1) = t_1; // \boxed{i=1}$$

for $i := 2$ to n dofor $c := 0$ to C doif $(c - c_i) \geq 0 \wedge S_{i-1}(c - c_i) \downarrow \wedge$ $t_i + T_{i-1}(c - c_i) \leq b \wedge t_i + T_{i-1}(c - c_i) \leq T_i(c)$ then

$$S_i(c) := \{i\} \cup S_{i-1}(c - c_i); // \boxed{\square}$$

$$T_i(c) := t_i + T_{i-1}(c - c_i);$$

else

$$S_i(c) := S_{i-1}(c);$$

$$T_i(c) := T_{i-1}(c)$$

endendend $C^* = \text{največji } c \text{ pri katerem je } S_n(c) \downarrow$ $Y = S_n(C^*)$ end

(iz drugine for zanke)

Časovna zahtevnost $\rightarrow nC$. Je reda $\boxed{O(nC)}$

$$c_i \quad \boxed{1} \quad c_i = 1$$

$$c_i = 2^d - 1$$

 \rightarrow odvisen od določene podatkov c_i exponenten algoritmom
(pseudo polinomska)

Ta algoritmom je v eksponentni časovni odvisnosti od dolžine podatkov. Vendar izgleda, kot da je problem rešljiv v polinomskem času. Temu rečemo "pseudopolinomska" časovna zahtevnost.

$$T(n) = O(nC)$$

$\sum c_i$

$$1 \leq c_i \leq 2^d - 1 = \Theta(2^d)$$

d bitov

$n^2 \leq T(n) \leq n^2 2^d$ zato pseudopolinomski, včasih žrtvujemo natančnost na račun hitrosti

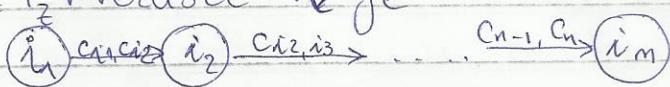
Problem najcenejših poti

- optimizacijski problem
- metodai: dinamično programiranje

DEF: Dan je utežen usmerjen graf $G(V, E, c)$

$$c: E \rightarrow \mathbb{R} \text{ (generacija funkcija)}$$

Pot iz i_1 v vozlišče i_k je

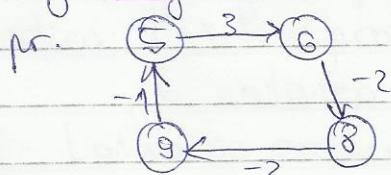


$$\begin{aligned} i_1 &= i_2 \\ i_n &= i_k \end{aligned}$$

Cena (dolžina) poti iz i_1 do i_n je vsota cen na povezavah na tej poti ($\sum_{j=0}^{n-1} c_{ij}; i_{j+1}$)

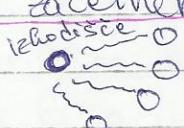
Cikel je pot, ki se konča v začetni točki.

Cikel je negativen, če ima negativno ceno.

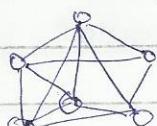


Problem najcenejših poti

A) med začetnem in vsemi ostalimi vozlišči



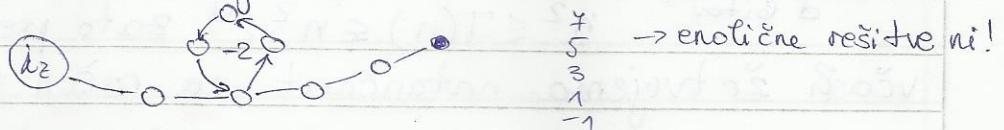
B) med vsemi pari vozlišč



A) Problem najcenejših poti med začetnim in vsemi ostalimi vozlišči

Odslej zahtevamo dvoje:

- začetno vozlišče je povezano z vsemi ostalimi (tj. Če vsmerjena pot iz i_2 do vsakega drugega)
- graf G nima negativnih ciklov



Stalno bomo (implicitno) rabili načelo optimalnosti:

Naj bo i_p, \dots, i_q del najcenejše poti iz vozlišča i_2 do i_k .



Tedaj je i_p, \dots, i_q najcenejša pot iz i_p do i_q .

Bellmanove enačbe (že splošni primer problema)

Dan je $G(V, E, c)$ in veljata pogoja A) in B). Recimo, da smo že izračunali najcenejše poti iz 1 do H vozlišča.

Njihove cene so: v_1, v_2, \dots, v_n .

Če je $i=1$ (izhodiščno vozlišče), je $v_1=0$

Če je $i \neq 1$, pride najcenejša pot iz 1 do i

$$\textcircled{1} \quad \min_{k \neq i} \{ v_k + c_{ki} \}$$

Ugotovitev: za vrednosti v_1, \dots, v_n veljajo Bellmanove enačbe (BE)

$$i=1,2,\dots,n : v_i = \begin{cases} 0, & i=1 \\ \min_{k \neq i} \{ v_k + c_{ki} \}, & \text{ostalo} \end{cases}$$

Izrek: Naj bo dan $G(V, E, c)$ in pogoja A), B) veljata.

Tedaj: če so v_1, \dots, v_n rešitev problema najcenejših poti (iz 1 v vse ostale) za graf $G(V, E, c)$, potem so v_1, \dots, v_n rešitev pravljajočega sistema Bellmanovih enačb.

Velja tudi obratno:

Izrek: Naj bo $G(V, E, c)$ in A), B) kot zgoraj.

Tedaj velja če so v_1, \dots, v_n rešitve ustreznega sistema BE, potem so v_1, \dots, v_n tudi rešitve problema najcenejših poti (od 1 do H vozlišča) v $G(E, V, c)$.

Posledica: Rešitev problema najcenejših poti za graf $G(V, E, c)$ je natanko rešitev ustreznega sistema Bellmanovih enačb.

Torej: Lahko se začнемo ukvarjati z vprašanjem kako reševati sisteme BE.

Opazimo težavo: za v_i rabimo v_k , $k=1,2,\dots,i-1, i+1, \dots, n$, ki pa jih tudi ravnokar računamo.

V dveh primerih se BE lahko reši "enostavno":

→ f če je G acikličen

če ima G samo pozitivne cene $c_{ik} (\geq 0)$

(Tedaj lahko uporabimo tudi Dijstrrov algoritem)

Topološko urejanje grafov

- rabili bomo za reševanje BE, ki pripadajo acikličnemu grafu
- za ugotavljanje ali je G acikličen

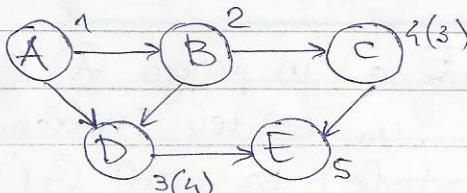
Dan graf $G(V, E)$

? Ali lahko G preoznačimo njegova vozlišča tako, da bo vsaka povezava terla iz vozlišča z nižjo (novo) oznako v vozlišče z višjo (novo) oznako?

Če je tako preimenovanje vozlišč možno in ga izvedemo, pravimo, da je novi graf topološko urejen.

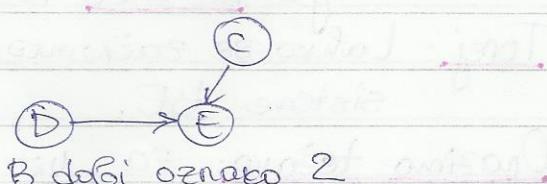
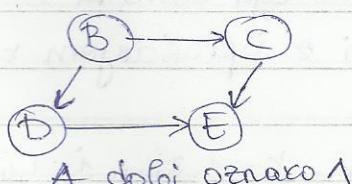
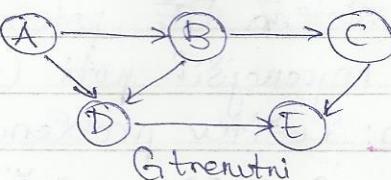
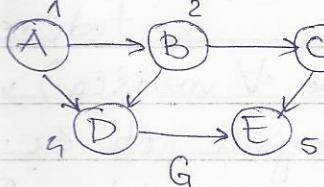
DEF: Usmerjen graf $G(V, E)$ je topološko urejen, če obstaja funkcija $\text{top}: V \rightarrow \{1, 2, \dots, |V|\}$, tako, da velja $(i, j) \in E \Rightarrow \text{top}(i) < \text{top}(j)$.

Primer:



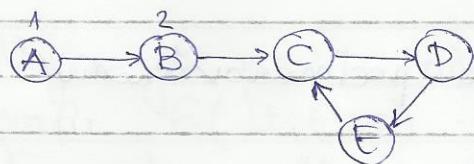
Primer: Ni možno vsak graf topološko urediti

Izrek: Usmerjeni graf G se da topološko urediti, natanko takrat (\Leftrightarrow) ko je G acikličen



A dobi oznako 1
B dobi oznako 2
C dobi oznako 3

D dobi oznako 4
E dobi oznako 5



Ostat nam je graf, ki ga ne moremo topološko urejiti, saj noben izmed C, D, E nima izvorne stopnje 0.

Algoritem: topološko urejanje grafa

vhod: usmerjen graf $G(V, E)$

izhod: DA (G je acikličen) in preimenovanje top: $V \rightarrow \{1 \dots |V|\}$
NE

begin

$G_t = G$; $s := 0$; // števec

while (G_t vsebuje vsaj eno vozlišče z vhodno stopnjo 0) do

$v :=$ izberi vozlišče z vhodno stopnjo 0;

$s := s + 1$;

 // večjemu v-krat

$\text{top}[v] := s$;

$G_t = G_{t-v}$ // izločitev vozlišča in vseh incidentnih povezav
 // brečjemu v-krat

end

if ($G_t = \emptyset$) // če ima G še kateno vozlišče (if $s=n$)

then

 return DA in top

else

 return NE

end

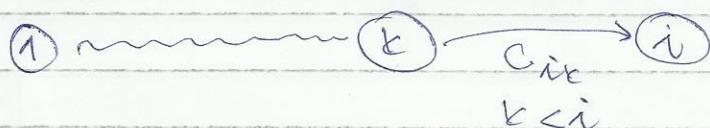
$O(|V|^2)$ → časovna zahtevnost algoritma
 // št. točk

Reševanje BE za aciklične grafe $G(V, E, c)$

G topološko uredimo in preimenujemo. ($i = \text{top}[i]$) .

Po tem preimenovanju zagotavo velja: $(i, j) \in E \Rightarrow i < j$

Torej v vozlišče i pridemo iz izhodiščnega vozlišča le iz vozlišča k , vjer $k < i$.



Bellmanove enačbe, ki ustrezajo preimenovanemu grafu bo sedaj:

$$\text{za } i=1 \dots n \quad u_i = \begin{cases} 0, & i=1 \\ \min_{k < i} \{ u_k + c_{ki} \} & \end{cases}$$

!

Računanje rešitve BE poteka po naraščajočih i :

$$u_1 = 0$$

$$u_2 = u_1 + c_{12}$$

$$u_3 = \min \{ u_1 + c_{13}, u_2 + c_{23} \}$$

- včasih rake povezave ne bo! → dano je zelo velika cena

$$u_4 = \min \{ u_1 + c_{14}, u_2 + c_{24}, u_3 + c_{34} \}$$

⋮

$$u_i = \min \{ u_1 + c_{1i}, u_2 + c_{2i}, \dots, u_{i-1} + c_{i-1,i} \}$$

$$u_n = \min \{ u_1 + c_{1n}, u_2 + c_{2n}, \dots, u_{n-1} + c_{n-1,n} \}$$

Časovna zahtevnost:

pri računanju u_i opravimo:

\sqsubseteq_{i-1} seštevanj

\sqsubseteq_{i-2} primerjanj

skupno za izračun u_1, \dots, u_n :

$$\text{seštevanj: } \sum_{i=2}^n (\sqsubseteq_{i-1}) = \Theta(n^2)$$

$$\text{primerjanj: } \sum_{i=2}^n (\sqsubseteq_{i-2}) = \Theta(n^2)$$

$\Theta(n^2)$ za aciklične grafe

Najcenejše poti iz začetnega do vseh ostalih vozlišč v splašnem grafu $G(V, E, c) \rightarrow$ Bellman - Ford

Se vedno zahteva: A, B

Razmislki: Naj bo $G(V, E, c)$ brez negativnih ciklov.

Najcenejša pot iz 1 do i gre čez največ $n-1$ povezav. (sicer bi se neko vozlišče na njej ponovilo, dobavljeni cikel bi bil pozitiven in zato je najkrajša pot ne najcenejša)

Naj pomeni $U_i^{(p)}$ = cena najcenejše poti iz 1 do i, ki vsebuje $\leq p$ povezav

Torej: najcenejša pot iz 1 do i, ki gre čez $\leq p$ povezav gre:

$\left[\begin{array}{l} \text{bodisi čez } p \text{ povezav} \rightarrow \text{tedaj ima cena } \min_{x \neq i} \{ U_x^{(p-1)} + c_{xi} \} \\ \text{ali pa gre čez večjemu } p-1 (\leq p-1) \text{ povezav} \rightarrow \\ \text{tedaj ima cena } U_i^{(p-1)} \end{array} \right]$

$$(x \leq p \Leftrightarrow x=p \vee x \leq p-1)$$

njena cena bo manjša izmed obeh možnosti:

$$U_i^{(p)} = \min \{ U_i^{(p-1)}, \min_{x \neq i} \{ U_x^{(p-1)} + c_{xi} \} \}$$

Začetne vrednosti: $U_i^{(0)} = 0$, za $i \geq 1$: $U_i^{(1)} = c_{ii}$

Od tod izboljšane BE:

$$U_i^{(p)} = \left\{ \begin{array}{ll} 0, & i=1 \\ c_{ii}, & p=1 \\ \min \{ U_i^{(p-1)}, \min_{x \neq i} \{ U_x^{(p-1)} + c_{xi} \} \} & \text{za } i=2, 3, \dots, n \end{array} \right.$$

? Kako reševati ta sistem?

Računanje:

p	
1	inicIALIZACIJA
2	
\vdots	
m	izračunati moramo m vrednosti $U_1^{(m)}, U_2^{(m)}, \dots, U_j^{(m)}, \dots, U_n^{(m)}$
\vdots	
$n-1$	

Pri tem uporabimo prejšnjo generacijo

Algoritem:

$$U_1^{(0)} := 0;$$

$$U_i^{(1)} := c_{1i}, \text{ za } i \geq 1$$

for $p := 2$ to n do

for $i := 1$ to n do

$$\text{izračunaj } U_i^{(p)} := \min \left\{ U_i^{(p-1)}, \min_{k \neq i} \{ U_k^{(p-1)} + c_{ki} \} \right\}$$

Časovna zahtevnost:

L Telo zanke zahteva: $\rightarrow \text{pri } \{ U_k^{(p-1)} + c_{ki} \}$

$\left[\begin{array}{l} n-1 \text{ seštevanj} \\ n-1 \text{ primerjanj} \end{array} \right] \Rightarrow \Theta(n)$

Telo se izvede kvečjemu

$$(n-1) \cdot n - \text{krat} = \underline{\Theta(n^2)}$$

$$\Theta(n^3)$$

// če sta v tabeli dve vrstici (eno za drugo) popolnoma enaki, se ne bo plačalo računati naprej – torej pri implementaciji je dobro beležiti če je karšen U spremenjen

Spometa ali itovščen val?

Floyd - Marshallov algoritem

→ za izračun dolžine najkrajše poti (med vsemi pari)

Naj bo $G(V, E, c)$ poljuben graf (brez negativnih ciklov).

Naj pomeni v_{ij} = cena najkrajše poti iz $i \rightarrow j$

Uvedimo še označo $v_{ij}^{(m)}$ = cena najkrajše poti iz $i \rightarrow j$,
kjer imajo vmesna vozlišča označa $\leq m$

Oditno: $v_{ij} = v_{ij}^{(n)}$

Velja: $v_{ij}^{(0)} = c_{ij}$ (inicIALIZACIJA)

Velja: najkrajša pot iz $i \rightarrow j$, kjer so vmesna vozlišča $\leq m$

└ bodisi gre čez vozlišče m $i \xrightarrow{v_{im}^{(m)}} m \xrightarrow{v_{mj}^{(m)}}$

└ ali pa ne gre čez vozlišče m $i \xrightarrow{v_{ij}^{(m-1)}}$

└ zato je cena manjša od obeh cen:

$$v_{ij}^{(m)} = \min \{ v_{ij}^{(m-1)}, v_{im}^{(m-1)} + v_{mj}^{(m-1)} \}$$

(BE za vsako vozlišče z vsakim)

Če želimo izračunati $v_{ij}^{(n)}$, moram rešiti sistem enačb:

$$v_{ij} \in \{1, n\}, \quad m \in \{0, \dots, n\}: \quad v_{ij}^{(m)} = \begin{cases} c_{ij}, & \text{če } m=0 \\ \min \{ v_{ij}^{(m-1)}, v_{im}^{(m-1)} + v_{mj}^{(m-1)} \}, & m=1, 2, \dots, n \end{cases}$$

Računanje poteka po naraščajočih m :

m	
0	inicIALIZACIJA
1	
\vdots	
$k-1$	
k	izračunaj n^2 parov $v_{ij}^{(k)}$. Pri tem rabimo te prejšnje generacije.
\vdots	
n	

Algoritem (shema)

inicIALIZACIJA;

for $m:=1$ to n do

 for $i:=1$ to n do

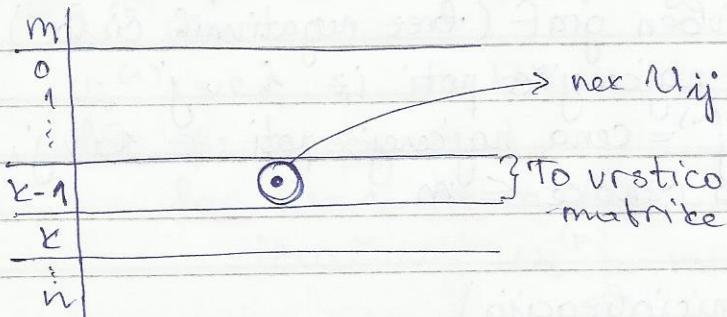
 for $j:=1$ to n do

$$v_{ij}^{(m)} = \min \{ v_{ij}^{(m-1)}, v_{im}^{(m-1)} + v_{mj}^{(m-1)} \}$$

Casovna zahtevnost: $\Theta(n^3)$

Prostorska zahtevnost: videti, da je $2n^2 = \underline{\Theta(n^2)}$

Se enkrat tabelo računamo:



To vrstico si predstavljamo v obliki matrice: $U = [\dots \quad \text{circle} \quad \dots]_{n \times n}$

Definimo, da novo matrica $U = (\text{za } k\text{-to vrstico})$ izračunamo po pravila:

$$U_{ij} := \min \{ U_{ij}, U_{ik} + U_{kj} \}$$

↓
 nova vrednost ↑
 stare vrednosti

$$U = [\dots \quad \text{circle} \quad \dots]_k$$

- na diagonali je najkrajša pot (same vrste)

Algoritam (Floyd-Marshall)

vhod: $C = (c_{ij})$ /matrica cen povezav grafe $G(V, E, c)$

izhod: $U = (u_{ij})$ /matrica cen najcenejših poti med vsakim parom

begin

```

 $U := C;$ 
  for  $k := 1$  to  $n$  do
    for  $i := 1$  to  $n$  do
      for  $j := 1$  to  $n$  do
         $U_{ij} := \min \{ U_{ij}, U_{ik} + U_{kj} \}$ 
end
  
```