Ocenjevanje porabe računskih virov

S formulami zapišite definicije relacij:
$f(n) = O(g(n))$, $f(n) = \Omega(h(n))$, $f(n) = \Theta(k(n))$

$f(n) = O(g(n))$: $f(n)$ je omejena navzgor z $g(n)$

$(\exists c_1, n_0 > 0)(\forall n \geq n_0)\ [f(n) \leq c_1 g(n))]$

$f(n) = \Omega(h(n))$: $f(n)$ je omejena navzdol z $h(n)$

$(\exists c_2, n_1 > 0)(\forall n \geq n_1)\ [f(n) \geq c_2 h(n))]$

$f(n) = \Theta(k(n))$: $f(n)$ je enaka funkciji $k(n)$ (f je od zgoraj in spodaj omejena z k)

$(\exists c_1, c_2, n_2 > 0)(\forall n \geq n_2)\ [c_1 k(n) \leq f(n) \leq c_2 k(n)]$

Kakšni sta asimptotični zahtevnosti funkcij $f(n)$ in $g(n)$, kjer je
$f(n) = 2n^2 \log n + 2n^2 \log\log n$ in
$g(n) = n^2 \log n + 10n^2 \log\log n$
$\Theta(n^2 \log n)$

Izpeljite asimptotično (Θ) zahtevnost funkcije $f(n) = 8^{\log_2 n}$.

$f(n) = 8^{\log_2 n} = n^{\log_2 8} = n^3 \Rightarrow \Theta(n^3)$

Izpeljite asimptotično (Θ) zahtevnost funkcije $f(n) = 4^{\log_2 n}$.

$f(n) = 4^{\log_2 n} = n^{\log_2 4} = n^2 \Rightarrow \Theta(n^2)$

Kaj hitreje narašča?

$f(n) = e^n \cdot n!$
$g(n) = n^{(n+1)}$

$\underset{n \to \infty}{f(n) \approx n!}$ $\quad \underset{n \to \infty}{g(n) = n^n}$ $\quad n! < n^n \rightarrow g(n) > f(n)$

$g(n)$ hitreje narašča

Izpeljite asimptotično (Θ) zahtevnost za naslednji funkciji.

$f(n) = n^2 + 3n \qquad T(n) = n^2 + 3n \rightarrow n^2 \Rightarrow \Theta(n) = n^2$

$g(n) = n^2 - e^{(-\frac{n}{2})} \qquad T(n) = n^2 - e^{-\frac{n}{2}} \rightarrow n^2 \Rightarrow \Theta(n) = n^2$

Poiščite asimptotično zahtevnost naslednjih funkcij:

$$f(n) = \sum_{k=1}^{n} k$$

$$= \frac{1}{2} n(n+1) = \frac{1}{2} n^2 + \frac{1}{2} n = \frac{1}{2} n^2 \left(1 + \frac{1}{n}\right) \to \frac{1}{2} n^2$$

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

$$f(n) = n!$$

$$n! = /\text{STIRLING}/ \doteq \sqrt{2\pi n} \left(\frac{n}{e}\right)^n = \begin{cases} \Theta\left(n^{n+\frac{1}{2}}\right) \\ \Omega\left(\frac{n^{n+\frac{1}{2}}}{3^n}\right) \end{cases}$$

$$f(n) = \sum_{i=1}^{n} \log i = \log 1 + \log 2 + \dots + \log n = \log(1 + 2, \dots, n) = \log n!$$

$$/\text{stirling}/ = \log \sqrt{2\pi n} \left(\frac{n}{e}\right)^n = \underbrace{\log \sqrt{2\pi}}_{\text{KONSTANTA}} + \frac{1}{2}\log n + \underbrace{n \log n}_{\text{NAJHITREJŠI}} - n \log e$$

$$= n \log n \left[\frac{\log \sqrt{2\pi}}{n \log n} + \frac{1}{2n} + 1 - \frac{\log e}{\log n} \right] \doteq k \cdot n \log n = \Theta(n \log n)$$

$$f(n) = c \cdot a^{\log_b n}$$

$$= c \cdot \underbrace{\left(b^{\log_b a}\right)}_{a}{}^{\log_b n} = c\left(b^{\log_b n}\right)^{\log_b a} = c \cdot n^{\log_b a} = \theta(n^k)$$

za nek k

Urejanje

Definiraj urejanje zaporedij.

Dani so $a_1, a_2, \ldots, a_n$ in relacija popolne urejenosti $\preccurlyeq$.

Cilj je poiskati tako razporeditev, da velja $a_{i1} \preccurlyeq a_{i2} \preccurlyeq \ldots \preccurlyeq a_{in}$.

Naštejte 5 algoritmov za zunanje urejanje.

Navadno zlivanje, uravnoteženo zlivanje, večsmerno zlivanje, polifazno zlivanje, naravno zlivanje.
(Zunanje urejanje - Če je n preveliko število, so elementi na zunanjem pomnilniku v neki datoteki.)

Naštejte 5 algoritmov za notranje urejanje.

(Če je n takšno število, da so vsi elementi v glavnem pomnilniku v neki tabeli).

Razlikujejo se v načinu, kako razširijo urejeni del.

- **urejanje z izbiranjem** (*Selection sort*)
  Prvi element v neurejenem delu zamenjajamo z najmanjšim elementom v neurejenem delu (N).
- **urejanje z zamenjavami** (*Bubble sort*)
  Sprehodimo se po neurejenem delu od desne v levo, vsakokrat primerjajamo soseda in ju po potrebi zamenjamo.
- **urejanje z vstavljanjem** (*Insertion sort*)
  Prvi element v neurejenem delu vrinemo v ustrezno mesto v urejeni del
- **urejanje s kopico** (*Heap sort*)
- **hitro urejanje** (*Quick sort*)
- **zlivanje** (*Merge sort*)
  metoda: deli in vladaj
  deli: tabelo razdelimo na podtabele, dokler ne pridemo do trivialno urejene podtabele (1 element v tabeli)
  vladaj: urejene podtabele združujemo v urejeno tabelo

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

R(i) ... časovna zahtevnost razširjanja urejenega dela (U) v i-ti ponovitvi.

Vsako urejanje n števil, ki uporablja operacijo primerjanja, zahteva $\mathbf{h(n) \geq \lceil \log_2 n! \rceil \ldots k\, n\log n = \Omega(n \log n)}$ operacij primerjanja (minimalni čas za urejanje števil).

Najboljši, povprečen in najslabši case scenario časovne zahtevnosti za heapsort in quicksort.

- **quick sort (hitro urejanje)**
  najslabši primer: $\Theta(n^2)$, najboljši primer: $\Theta(n \log n)$, povprečni primer: $\Theta(n \log n)$
- **heap sort (urejanje s kopico)**
  najslabši primer: $\Theta(n \log n)$, najboljši primer: $\Theta(n \log n)$, povprečni primer: $\Theta(n \log n)$

Urejanje s kopico (*Heap sort*)

Kopica je dvojiško drevo, ki je urejeno, levo poravnano, podatki vzdolž padajo, koren ima največji podatek.

Algoritem

Izloči koren in ga shrani v vrsto, na mesto izločenega korena prestavi zadnji list, popravi dobljeno drevo v kopico, vrne se na 1. korak.

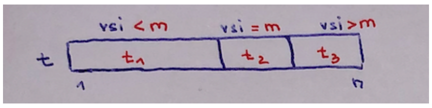
```
Heapsort
begin
      Sestavi_zacetno_kopico; // Θ(n)
      r := n;
      while r > 1
            begin zamenjaj(a[1], a[r]) // menjava korena z zadnjim listom; A in B
                  r := r - 1
                  Popravi_v_kopico(1, r)
            end
end
```

Časovna zahtevnost

Celotni heapsort: $\Theta(n \log n)$

Hitro urejanje (*Quick sort*)

Izberemo neko število m (pivot) in preuredimo tabelo t.



m: srednji element tabele [n div 2], naključno izbran, [1/3 (prvi + srednji + zadnji)]

Algoritem

```
begin
      if
            t ima kvecjemu en element then return (t)
      else
            begin
                  m := izberi med elementi tabele t ali izracunaj;
                  razdeli t v t1 in t2 U t3;
                  // naredimo stik med temi tremi tabelami (nastane 1 tabela)
                  return (QuickSort(t1); QuickSort(t2 U t3))
            end
end
```

Analiza časovne zahtevnosti

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

- najslabši primer (prva ima 1 element, druga pa vse ostale)

  $T(|t|) = T(1) + T(|t| - 1) + \Theta(|t|) + \Theta(1)$

  $|t| = n$

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

- najboljši primer (enako dolgi tabeli, število elementov je potenca števila 2, $|t| = n = 2^m$)

  $T(n) = T(n/2) + T(n/2) + \Theta(n) + \Theta(1) = 2T(n/2) + \Theta(n)$

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

- povprečni primer (vsi vhodni podatki enako verjetni, paroma različni elementi)

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

Metode

Naštejte 5 metod za razvoj algoritmov.
Dinamično programiranje, deli in vladaj, sestopanje, požrešni algoritmi, rekurzivni razcep.

Opiši razliko med dinamičnim programiranjem in deli in vladaj ter načelo optimalnosti.
Dinamično programiranje temelji na pravilu optimalnosti, kjer sistematično pregleduje delne rešitve in sestavlja optimalno rešitev. Če v danem koraku ugotovimo, da delna rešitev ne pripelje do optimalne, jo zavržemo. Vsaka naloga ima svojo Bellmanovo enačbo.
Primer: nahrbtnik
Deli in vladaj: problem najprej razdelimo na manjše podprobleme, le-te rešimo z uporabo rekurzije in iz njihovih rešitev sestavimo rešitev problema.
Primer: QuickSort (urejanje podtabele t_1 in urejanje podtabele $t_2 \cup t_3$)

Zaporedje odločitev $D_1, D_2, \dots, D_m$ (ki jih sprejmemo med računanjem rešitve naloge N) je optimalno, če nas privede do optimalne rešitve naloge N.
Načelo optimalnosti: Vsako podzaporedje optimalnega zaporedja odločitev je tudi optimalno.

Kdaj rečemo, da ima algoritem psevdo-polinomsko časovno zahtevnost?
Časovna kompleksnost algoritma odvisna od numerične vrednosti vhoda, in ne od velikosti vhoda.

Za problem P razvijamo algoritem po metodi deli in vladaj.
Izkaže se tole: P lahko razdelimo v 4 podprobleme (iste vrste kot P), kjer je vsak velikosti n/3; rešitev problema P se da sestaviti iz rešitev dveh izmed podproblemov; priprava podproblemov in seštevanje njihovih rešitev zahtevata skupaj 5n časa. Izračunaj asimptotično časovno zahtevnost tega algoritma.
$a = 4$

Zapišite glavni izrek (Master Theorem) metode deli & vladaj. Pojasnite pomen uporabljenih oznak.

$$T(n) = \begin{cases} b & \text{če } n = 1; \\ aT(\frac{n}{c}) + bn^d & \text{če } n > 0, \end{cases}$$

kjer so $a \geqslant 1$, $b > 0$, $c \geqslant 2$ in $d \geqslant 0$, velja naslednje:

$$T(n) = \begin{cases} \Theta(n^d) & \text{če } \frac{c^d}{a} > 1; \\ \Theta(n^d \log n) & \text{če } \frac{c^d}{a} = 1; \\ \Theta(n^{\log_c a}) & \text{če } \frac{c^d}{a} < 1. \end{cases}$$

b = konstanta brez bistvenega pomena

c = faktor delitve naloge (npr. število enako velikih podtabel, ki jih dobimo z delilnim elementom)

a = število podnalog

d = red velikosti zahtevnosti pri delitvi in združevanju

n = velikost naloge

Problem P rešujemo z metodo deli in vladaj takole:
P razdelimo v podprobleme (iste vrste kot P), od katerij je vsak velikosti n/2. Sedem podproblemov rešimo in iz njihovih rešitev sestavimo rešitev problema P. Priprava podproblemov in sestavljanje njihovih rešitev v končno skupaj terjata $5n^2$ časa.
Kolikšna je asimptotična zahtevnost tega algoritma?
Odgovor utemelji.

VELIKOST PODPROBLEMA : $n/2$
↳ $c=2$ faktor delitve

ŠT. PODPROBLEMOV: $a=7$

PRIPRAVA PODPROBLEMOV IN SESTAVLJANJE REŠITVE: $5n^2$ ↗ d

$T(n) = aT(\frac{n}{c}) + bn^d$ ← priprava + sestavljanje

$T(n) = 7 \cdot T(\frac{n}{2}) + O(5n^2)$ → konstanta gre stran

$T(n) = 7 \cdot T(\frac{n}{2}) + O(n^2)$

$a=7 \quad c=2 \quad d=2 \quad \rightarrow \frac{2^2}{7} = \frac{4}{7} \rightarrow$ MASTER THEOREM

$\Theta(n^{\log_c a})$ če $\frac{c^d}{a} < 1$

$cd < a$

$\Downarrow$

$\underline{\underline{\Theta(n^{\log_2 7})}}$

$n^{\log_n x} = x$

$\log_n n = 1$

$\log_n 1 = 0$

$\log_n x^r = r \cdot \log_n x$

Matrično množenje

Navadno množenje

$\Theta(n^3)$

Deli in vladaj

$\Theta(n^3)$

Strassenovo matrično množenje

```
procedure Strassen(A,B) return C;
      begin
      if n = 1 then C := AB else
      begin
            P11 := Strassen(A11 + A22, B11 + B22) ...
            C11 := P11 + P12 - P13 - P14
            ...
            C := (C11,C12,C21,C22)
      end
end
```

Asimptotično hitrejši od $\Theta(n^3)$ -> $\Theta(n^{2.373})$.

Iskanje k-tega največjega elementa

Kakšna je asimptotična časovna zahtevnost hitrega algoritma za iskanje k-tega elementa po velikosti med n urejenimi števili. Opišite, kako pri tem algoritmu uporabimo metodo deli in vladaj.

Heapsort: $\Theta(n\log n)$.

Izboljšani (Blum-Floyd-Pratt-Rivest-Tarjanov) algoritem

Zahteva vsaj $\Omega(n)$ časa ter največ $O(n)$ časa. Ker sta spodnja in zgornja meja časovne zahtevnosti tega problema enaki, ima ta problem časovno zahtevnost $\Theta(n)$ --> asimpotično optimalen.

Problem razdelimo na 3 podnaloge.

Izberemo število m (delilni element) in razdelimo tabelo t na 3 podtabele, kjer bodo v

- v t_1 elementi $< m$
- v t_2 elementi $= m$
- v t_3 elementi $> m$.

Kje je k?

Če je

- $k \leq |t_1|$ v t_1
- $|t_1| \leq k \leq |t_1| + |t_2|$ v t_2
- $k > |t1| + |t2|$ v t_3.

Ko izvemo, v kateri od t_1, t_2, t_3 je iskani element, nadaljujemo iskanje v njej.

Diskretna Fourierova transformacija

Polinom $p(x) = \sum_{i=0}^{n} a_i x^i$ lahko predstavimo z

- koeficientno predstavitvijo: $p(x) = (a_0, \ldots, a_n)$
- vrednostjo predstavitvijo (vrednosti polinoma na izbranih točkah):
  $p(x) = (p_0, \ldots, p_n)$, kjer so p_i njegove vrednosti v $n + 1$ paroma različnih točkah $t_0, \ldots, t_n$

Naj bo n dimenzija vektorskega prostora.

a) Zapišite definicijo n-tega primitivnega korena.

b) Zapišite definicijo Fourierove matrike F.

c) Kakšna je inverzna matrika F^{-1}?

d) Napiši matriki F in inverzni F 4 X 4 za splošni ω pri DFT.

e) Kakšna je časovna zahtevnost algoritma za Hitro Fourierovo Transformacijo (FFT)?

Primitivni koren enote je skalar ω, ki ga definirata dve lastnosti:

1. $\omega^d = 1$ in // ω je *koren* enote
2. $\omega^k \neq 1$ za $k = 1, \ldots, d-1$. // ω je *primitiven* koren enote

V obsegu $\mathbb{C}$ je

$$\omega = e^{\iota \frac{2\pi}{d}}. \qquad (\iota \text{ je imaginarna enota})$$

$$\omega = \cos\frac{2\pi}{n} + i\sin\frac{2\pi}{n}$$

S pomočjo Fourierove matrike predstavimo diskretno Fourierovo transformacijo (DFT), ki je linearna transformacija prostora V.

Matriko F je reda d × d, njene komponente pa so

$$F_{ij} = \omega^{i*j} \quad , \quad 0 \leqslant i, j \leqslant d-1.$$

$$F = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ 1 & \omega & \ldots & \omega^{n-1} \\ 1 & \omega^2 & \ldots & \omega^{2(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \ldots & \omega^{(n-1)^2} \end{bmatrix}$$

Inverzna DFT je predstavljena z matriko F^{-1} reda d × d, njene komponente so

$$F_{ij}^{-1} = \frac{1}{d}\omega^{-i*j} \quad , \quad 0 \leqslant i, j \leqslant d-1.$$

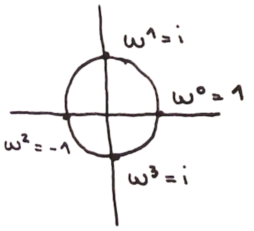
$F * F^{-1} = I$

4 x 4

primitivni koren enote za d = 4:

$$\omega = e^{\iota \frac{2\pi}{d}} = e^{\iota \frac{2\pi}{4}} = e^{\iota \frac{\pi}{2}} = \iota.$$

n = 4



$$F_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}$$

$$F_4^{-1} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}$$

$(n^{-1} = 4^{-1} = 4)$

$\omega = i$

$\omega^{-1} = -i$

Matrika

1. vrstica in stolpec: 1
2. vrstica: krog enote
3. vrstica: prejšna vrstica * 2. vrstica
4. vrstica: prejšna vrstica * 2. vrstica

Inverzna matrika

1. vrstica in stolpec: 1
2. inverz: |1| ostane |1|, |i| - ju se spremeni predznak

Algoritem FFT v času Θ(n log n) pretvori koeficientno predstavitev polinoma stopnje n v njegovo vrednostno predstavitev na točkah $\omega_0, \ldots, \omega_n$, kjer je ω (n+1)-ti primitivni koren enote.
Polinoma stopnje n lahko zmnožimo v času Θ(n log n).

Algoritem FFT za DFT(a, d) razvijemo v 2 delih

- razdelimo DFT (a, d) na dva podproblema (sodi in lihi koeficienti) - deli in vladaj

$$p_S(x) \stackrel{\text{def}}{=} a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-1} x^{r-1} \qquad a_S \stackrel{\text{def}}{=} (a_0, a_2, a_4, \ldots, a_{n-1})$$
$$p_L(x) \stackrel{\text{def}}{=} a_1 + a_3 x + a_5 x^2 + \cdots + a_n x^{r-1} \qquad a_L \stackrel{\text{def}}{=} (a_1, a_3, a_5, \ldots, a_n)$$

$$p(x) = p_S(x^2) + x\, p_L(x^2)$$

a) Izračunamo $p_S(x^2)$ in $p_L(x^2)$ v d točkah $x^2 = (\omega^0)^2, (\omega^1)^2, \ldots, (\omega^n)^2$.
b) S temi vrednostmi izračunamo $p(x)$ v d točkah $x = \omega^0, \omega^1, \ldots, \omega^n$.

- sestavimo končno rešitev iz delnih rešitev

```
procedure FFT(a,d);
        begin
           if d=1 then return a else
                begin
                     Pripravi \omega; // Θ(d)
                     Razdeli(a,a_S,a_L); // Θ(d)
                     p_S := FFT(a_S,d/2); // T(d/2)
                     p_L := FFT(a_L,d/2); // T(d/2)
                     p := Sestavi(p_S,p_L); // Θ(d)
                end
         end
```

Nahrbtnik

Nahrbtnik je optimizacijski problem, ker sprašuje po najboljši rešitvi glede na dani kriterij. Je NP-težek (ni rešljiv v polinomskem času).

Algoritem zmanjšuje želeno vrednosti plena, dokler ne najde plena, ki se da odnesti.

Časovna zahtevnost celega algoritma O(nV), V velikost vhodnih podatkov, n pa število elementov.

Zapišite strogo (formalno) definicijo Problema nahrbtnika

Dana je množica R = {1, 2, . . . , n} z elementi, ki predstavljajo neke reči, ter funkciji

v : R → N in t : R → N, ki vsaki reči i priredita vrednost v(i) in težo t(i).

Dano je tudi število b ∈ N, ki predstavlja nosilnost nahrbtnika.

Poiskati moramo podmnožico P ⊆ R, imenovano plen, za katero bo veljalo $\sum_{i \in P} t(i) \leqslant b \ \ (plen\ ni\ pretežek)$

in ki bo maksimirala vsoto (V): $\sum_{i \in P} v(i). \ \ (plen\ je\ najvrednejši)$

Največji pretok

Definicija problema

Omrežje je označen usmerjen graf G(V, A, c).

- V = {1, 2, . . . , n} množica vozlišč (1 - izvor, n - ponor)
- $A \subseteq V \times V$ je množica usmerjenih povezav med vozlišči
- $c : A \to R^{+}_{0}$ funkcija, ki vsaki povezavi $(i, j) \in A$ priredi njeno kapaciteto $c_{i,j} \geq 0$.

$v_{i,j}$ je pretok dobrine po povezavi *(i, j)*, veljati mora

(1) $0 \leqslant v_{i,j} \leqslant c_{i,j}$ ... pretok čez povezavo je med 0 in kapaciteto povezave;

(2) $\sum_i v_{i,k} - \sum_j v_{k,j} = \begin{cases} -v & \text{če } k = 1; \\ 0 & \text{če } k \neq 1, n; \\ v & \text{če } k = n; \end{cases}$ iz vozlišča mora odtekati toliko dobrine kolikor je vanj doteka

Količina *v* je trenutni pretok skozi omrežje, množica $\{v_{i,j}\}$ pa razporeditev trenutnega pretoka *v* po povezavah omrežja.

Ford-Fulkersonov algoritem

Temeljna pot P je pot iz izvora v ponor, ki nima ciklov in na kateri zanemarimo smeri povezav.

Povezava (i, j)

- pozitivna (P^+), če kaže v smeri od izvora proti ponoru; sicer je negativna (P^-)
- P^+ je zasičena, če je $v_{i,j} = c_{i,j}$ (pretok čez njo doseže njeno kapaciteto)
- P^- je zasičena, če je $v_{i,j} = 0$ (pretok čez njo padel na 0)

P je zasičena, če vsebuje vsaj eno zasičeno povezavo (pretoka čez njo ne moremo več povečati).

P ni zasičena, če za vsako $(i, j) \in P^+$ velja $v_{i,j} < c_{i,j}$ in če za vsako $(i, j) \in P^-$ velja $v_{i,j} > 0$.

Cilj: Po vrsti zasitimo vse nezasičene temeljne poti. Algoritem se konča, ko pri nekem pretoku v∗ čez G(V, A, c) v omrežju ni več nezasičenih temeljnih poti.

Pretoki: pot, ki jo določajo vozlišča $v_1, v_2, ..., v_n$ usmerjenega grafa, ima povezave $e_{i,i+1} = (v_i, v_{i+1})$ s kapacitetami $c_{i,i+1}$ in trenutnimi tokovi $x_{i,i+1}$, kjer je i = 1, ..., n - 1.

Naj bosta P množica pozitivnih povezav na tej poti (usmerjenih k v_n) in N množica negativnih povezav na tej poti (usmerjenih k v_1). Zapišite (formalno) izraz, ki pove, za koliko lahko povečamo pretok iz v_1 do v_n po tej poti, da bo pot zasičena.

Temeljno pot zasitimo natanko tedaj, ko povečamo pretok čez njo za:

$$\min\{\min_{(i,j)\in P^+}\{c_{i,j} - v_{i,j}\}, \min_{(i,j)\in P^-}\{v_{i,j}\}\}.$$

Najcenejše poti iz izbranega izhodišča

Definicija problema

Dan je utežen usmerjen graf G (V, A, c), kjer je

- $V = \{1, 2, \ldots, n\}$ množica vozlišč
- $A \subseteq V \times V$ množica usmerjenih povezav med vozlišči
- $c : V \times V \to R$ funkcija, ki vsakemu paru $(i, j) \in V \times V$ priredi njegovo ceno $c_{i,j}$

Usmerjena pot iz vozlišča $i_z \in V$ v vozlišče $i_k \in V$ je zaporedje vozlišč $i_1, i_2, \ldots, i_l$, $l \geq 2$, kjer je $i_1 = i_z$ in $i_l = i_k$ ter $(i_j, i_{j+1}) \in A$ za $j = 1, \ldots l - 1$.

Cena $u_{iz,\, ik}$ poti je vsota cen na njenih povezaval $u_{i_z,i_k} = \sum_{j=1}^{j=\ell-1} c_{i_j,i_{j+1}}$

Cikel je usmerjena pot i_z iz v i_k, ki se konča v začetnem vozlišču.

- negativen, če je njegova cena negativno število

Problem

Za vsako vozlišče $i \in V$ poišči najcenejšo pot iz 1 (izhodišče) v i in njeno ceno $u_{1,i}$.

($u_i := c_{1,\, i}$ pomeni ceno najcenejše poti iz 1 v i v G)

Predpostavke

- (1) v G obstaja usmerjena pot do vsakega vozlišča (povezani)
- (2) G nima negativnih ciklov

Najcenejša pot iz 1 v i je oblike $1 \to k \to i$, kjer je $k \mathrel{!}= i$

(u_k: $1 \to k$ najcenejša pot iz 1 v k)

Če temu prištejemo še ceno povezave $k \to i$, dobimo $u_k + c_{k,i}$

(cena najcenejše poti iz 1 v i, ki prečka i-jevega soseda k)

Naj bo G splošen graf z n vozlišči in cenami povezav $c_{i,j}$. Zapišite splošne Bellmanove enačbe za računanje vrednosti u_i, kjer je u_i cena najcenejše poti iz vozlišča 1 v vozlišče i.

Za $u_1, u_2, u_3, \ldots u_n$ velja sistem Bellmannovih enačb (BE):

$$\boxed{u_i = \begin{cases} 0 & \text{če } i = 1; \\ \min\limits_{\substack{k \\ (k,i)\in A}} \{u_k + c_{k,i}\} & \text{če } i \geqslant 2. \end{cases}}$$

Če so $u_1, u_2, \ldots, u_n$ rešitev Problema najcenejših poti iz izhodišča za dani graf G(V, A, c), potem so tudi rešitev grafu pripadajočega sistema BE. Velja tudi obratno.

Problem Najcenejše poti iz izhodišča prevedli na problem reševanja sistema BE.

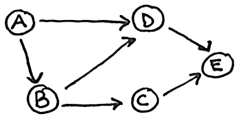
Topološko urejanje grafov

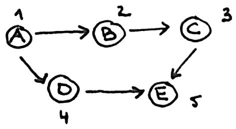
Kdaj pravimo, da je usmerjeni graf G({1, ..., m}, A) topološko urejen?

Naj bo G(V, A) graf z množico vozlišč $V = \{1, 2, \dots, n\}$.

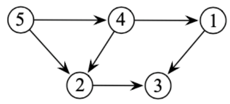
Če za graf G(V, A) obstaja bijektivna funkcija $\tau : V \to V$, ki vsakemu $i \in V$ priredi novo ime $\tau(i) \in V$ tako, da velja $(i, j) \in A \Rightarrow \tau(i) < \tau(j)$, rečemo, da τ **topološko ureja** G(V, A) oz. da je G(V, A) z njo topološko urejen.

Topološko uredite graf G(V, E), kjer je V = {a, b, c, d, e} in E = {a->b, a->d, b->c, b->d, c->e, d->e).

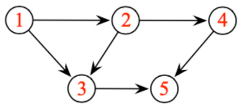




Topološko uredi dani graf.



i	$\tau(i)$
1	4
2	3
3	5
4	2
5	1



Graf je topološko urejen s $\tau(i)$.

Katere grafe lahko topološko uredimo?

Graf G(V, A) se da topološko urediti natanko tedaj ko je acikličen.

V psevdokodi napišite algoritem za topološko urejanje grafa G(V, E).

V psevdokodi zapišite algoritem za odločanje, ali je dani graf G(V, A) acikličen.

Časovna zahtevnost naj bo polinomska glede na |V|.

Če je graf acikličen, ga lahko topološko uredimo.

```
procedure Topolosko_Urejanje(G);
begin
      G' := G; // G ohrani, G' krči
      s := 0;
      while G'_ima_vozlisce_z_vhodno_stopnjo_0 do
            i := izberi_vozlisce_z_vhodno_stopnjo_0_v_G';
            s++; \tau(i) := s; // preimenuj i
            G' := G'-i // izloči i in povezave
      endwhile;
      if G' = prazen_graf
            then return(G_je_aciklicen; topolosko urejen s \tau)
            else return(G_je_ciklicen)
end
```

Časovna zahtevnost: $O(|V|^2)$

Najcenejše poti iz izhodišča v acikličnem grafu

Naj bo dan acikličen graf. Ko ga topološko uredimo, dobimo graf G(V, A, c), ki ga ima lastnost $(i, j) \in A \Rightarrow i < j$. Torej, če v G(V, A, c) obstaja povezava iz i v j, potem je $i < j$. To pa vpliva tudi na Bellmannove enačbe, v katerih pogoj $(k, i) \in A$ lahko nadomestimo s $k < i$.

Dan je označen graf G(V, E, c), kjer je $c_{i,j}$ cena povezave (i, j), |V| = n in |E| = m. Graf je topološko urejen. Napišite sistem Bellmanovih enačb za G.

$$\boxed{u_i = \begin{cases} 0 & \text{če } i = 1; \\ \min\limits_{\substack{k \\ k<i}} \{u_k + c_{k,i}\} & \text{če } i \geqslant 2. \end{cases}}$$

<u>Časovna zahtevnost</u>

$O(|V|^2)$

Najcenejše poti iz izhodišča v grafu s pozitivnimi cenami (Dijkstra)

Naj bo G(V, A, c) usmerjen graf, v katerem so cene vseh povezav pozitivne; torej $(i, j) \in A \Rightarrow c_{i,j} > 0$.

Na začetku računanja je že znana in dokončna cena $u_1 = 0$, cene $u_2, u_3, \ldots, u_n$ pa bo treba še izračunati, zato so vse še začasne.

$$u_1 := 0; \quad \forall i > 1 : u_i := c_{1,i}; \quad D := \{1\}; \quad Z := \{2, 3, \ldots, n\};$$

Želimo, da se množica D veča, tako, da bi vozlišča prestopala iz Z v D.

Kako začasna cena vozlišča postane dokončna? Če je u_k najmanjša začasna cena, torej $u_k = \min_{j \in Z}\{u_j\}$, potem se u_k ne bo več zmanjšal.

$$u_k := \min_{j \in Z}\{u_j\}; \quad D := D \cup \{k\}; \quad Z := Z - \{k\}; \quad \text{Če je } Z = \emptyset, \text{ končaj.}$$

Ko cena u_k postane dokončna, to lahko vpliva na začasne cene u_j vozlišč $j \in Z$ (ko so bile izračunane ne podlagi vozlišč v D, ko med njimi še ni bilo vozlišča k)

$$\forall j \in Z : u_j := \min\{u_j, u_k + c_{k,j}\}; \quad \text{Skoči na korak 2.}$$

Algoritem

```
procedure Dijkstrov_Algoritem(G(V, A, c));
begin
     u_1 := 1; for i := 2 to n do u_i := c_1,i;
     D := {1}; Z := {2,3,...,n};
     while NiPrazna(Z) then
          u_k := min_{j \in Z}{u_j};
          D := D + {k}; Z := Z - {k};
          IF NiPrazna(Z) then
               forall j \in Z do u_j := min{u_j, u_k + c_k,j};
     endwhile
end.
```

Časovna zahtevnost

$O(|V|^2)$

Najcenejše poti iz izhodišča v splošnem grafu (Bellmann-Ford)

Splošni graf: cikličen, ima tudi povezave z negativnimi cenami, nima negativnih ciklov

Naj bo G (V, A, c) utežen graf, $A \subseteq V \times V$ množica usmerjenih povezav med vozlišči in $c : V \times V \rightarrow R \cup \{\infty\}$ funkcija, ki vsakemu paru $(i, j) \in V \times V$ priredi njegovo ceno $c_{i,j}$.

Zahtevamo tudi, da za vse $1 \leq i, j \leq n$ velja $c_{i,i} = 0$ in (i, j) ne pripada $A \Rightarrow c_{i,j} = \infty$.

Naj bo G splošen graf z n vozlišči in cenami povezav c_{ij}. Naj bo $u_i^{(p)} \equiv$ cena najcenejše poti iz 1 v i, ki ima kvečjemu p povezav.

Zapišite Bellmanove enačbe, po katerih deluje Bellman-Fordov algoritem (za računanje najmanjših cen poti iz 1 do vseh vozlišč grafa G).

Obrazloži oznake. Kdaj ne deluje? Kdaj pot ni enolično določena?

Ne deluje, če iz izhodišča do nekega vozlišča ni usmerjene poti (zaradi nepovezanosti grafa ali pa zaradi smeri povezav) ter pa, če obstajajo negativni cikli v grafu.

i ... vozlišče, c ... cena, k ... vozlišče, ki je sosed i-ja, A ... množica usmerjenih povezav med vozlišči,

p ... povezava, $u_i^{(p)}$... cena najcenejše poti iz 1 v i, ki ima kvečjemu p povezav

$c_{k,i}$ .. cena najcenejše poti iz 1 v i, ki prečka i-jevega soseda k

$c_{1,i}$ ... cena najcenejše poti iz 1 v i v grafu

$$\boxed{u_i^{(p)} = \begin{cases} 0 & \text{če } i = 1, \text{ vsi } p; \\ c_{1,i} & \text{če } i > 1, p = 1; \\ \min\{u_i^{(p-1)}, \min\limits_{\substack{k \\ (k,i) \in A}} \{u_k^{(p-1)} + c_{k,i}\}\} & \text{če } i > 1, p > 1. \end{cases}}$$

```
procedure Bellman_Fordov_Algoritem(G(V,A,c));
begin
     for p := 1 to n-1 do u_1^(p) := 0 endfor;
     for i := 2 to n do u_i^(1) := c_{1,i} endfor;
     for p := 2 to n-1 do
          for i := 2 to n do
               u_i^(p) := min{u_i^(p-1),min_{k,k->i \in A}
               {u_k^(p-1)+c_{k,i}}
    endfor
  endfor
```

<u>Časovna zahtevnost</u>

$O(|V|^3)$

<u>Prostorska zahtevnost</u>

$O(|V|)$

Najcenejše poti med vsemi pari

Definicija problema

Dan je utežen usmerjen graf $G(V, A, c)$, kjer je $V = \{1, 2, \ldots, n\}$ množica vozlišč,

$A \subseteq V \times V$ množica usmerjenih povezav in

$c : V \times V \to R \cup \{\infty\}$ funkcija, ki vsakemu paru $(i, j) \in V \times V$ priredi njegovo ceno $c_{i,j}$.

Za vsak $i \in V$ je $c_{i,i} = 0$, in če (i, j) ne pripada A, je $c_{i,j} = \infty$.

Predpostavka: $G(V, A, c)$ nima negativnih ciklov.

Problem

Za vsak par vozlišč $i, j \in V$ poišči ceno $u_{i,j}$ najcenejše poti iz i v j.

Floyd-Warshallov Algoritem

- $u_{i,j}^{(m)} \equiv$ cena najcenejše poti iz i v j, na kateri imajo vsa vmesna vozlišča oznake kvečjemu m
- velja $u_{i,j} = u_{i,j}^{(n)}$
- Najcenejša pot P iz i v j, na kateri imajo vsa vmesna vozlišča oznake m,
  - (a) *bodisi* ne gre čez vozlišče m
  - (b) *bodisi* gre čez vozlišče m(torej je $P = i \to \cdots \to m \to \cdots \to j$)

$$\boxed{u_{i,j}^{(m)} = \begin{cases} c_{i,j} & \text{če } m = 0; \\ \min\{u_{i,j}^{(m-1)}, u_{i,m}^{(m-1)} + u_{m,j}^{(m-1)}\} & \text{če } 1 \leqslant m \leqslant n. \end{cases}}$$

Zapišite Floyd-Warshallov algoritem (psevdokoda) za računanje dolžin najkrajših poti med vsemi pari vozlišč grafa G(V, E). Graf opisuje matrika sosednosti reda n x n, kjer je C[i, j] cena povezave (v_i, v_j).

```
procedure Floyd_Warshallov_Algoritem(G(V,A,c));
begin
  for i := 1 to n do
    for j := 1 to n do
           u_{i,j}^(0) := c_{i,j}
     endfor
  endfor;
  for m := 1 to n do
    for i := 1 to n do
      for j := 1 to n do
           u_{i,j}^(m) := min{u_{i,j}^(m-1),u_{i,m}^(m-1)+u_{m,j}^(m-1)}
     endfor
    endfor
  endfor
end
```

<u>Izboljšava Floyd-Warshallovega Algoritma - računanje na mestu</u>

```
procedure Floyd_Warshallov_Algoritem(G(V,A,c));
begin
     for i := 1 to n do
          for j := 1 to n do
               U[i,j] := C[i,j]
          endfor
     endfor;
     for m := 1 to n do
          for i := 1 to n do
               for j := 1 to n do
                    U[i,j] := min{U[i,j],U[i,m]+U[m,j}]}
               endfor
          endfor
     endfor
end
```

Bellmanove enačbe. Kje se uporabljajo, katere probleme rešujemo z njimi.

Dinamično programiranje - iskanje najcenejših poti v grafu. S predpostavkami: da v G obstaja usmerjena pot do vsakega vozlišča (povezani) ter da G nima negativnih ciklov.

- najcenejše poti iz izbranega izhodišča

$$u_i = \begin{cases} 0 & \text{če } i = 1; \\ \min\limits_{\substack{k \\ (k,i)\in A}} \{u_k + c_{k,i}\} & \text{če } i \geqslant 2. \end{cases}$$

- najcenejše poti iz izhodišča v acikličnem grafu

$$u_i = \begin{cases} 0 & \text{če } i = 1; \\ \min\limits_{\substack{k \\ k<i}} \{u_k + c_{k,i}\} & \text{če } i \geqslant 2. \end{cases}$$

- najcenejše poti iz izhodišča v splošnem grafu (Bellmann-Ford)

$$u_i^{(p)} = \begin{cases} 0 & \text{če } i = 1, \text{ vsi } p; \\ c_{1,i} & \text{če } i > 1, p = 1; \\ \min\{u_i^{(p-1)}, \min\limits_{\substack{k \\ (k,i)\in A}} \{u_k^{(p-1)} + c_{k,i}\}\} & \text{če } i > 1, p > 1. \end{cases}$$

- najcenejše poti med vsemi pari

$$u_{i,j}^{(m)} = \begin{cases} c_{i,j} & \text{če } m = 0; \\ \min\{u_{i,j}^{(m-1)}, u_{i,m}^{(m-1)} + u_{m,j}^{(m-1)}\} & \text{če } 1 \leqslant m \leqslant n. \end{cases}$$

Iskanje znakovnih podnizov

Definicija problema

Naj bosta T in P znakovna niza dolžin |T| = n in |P| = m, kjer je $1 \leq m \leq n$.

T imenujemo **besedilo**, P pa **vzorec**. Torej T in P nista prazna, P pa ni daljši od T.

Znak na k-tem mestu v T označimo s T[k], znak na k-tem mestu v P pa s P[k].

Vprašanje

Ali obstaja zaporedje naravnih števil $s_1, s_2, \ldots, s_z$ z lastnostima $z \geq 1$in $0 \leq s_1 < s_2 < \ldots < s_z$, tako da za vsako število s_i velja

$T[s_i + j] = P[j]$, za vse j = 1, 2 ,..., m

Števila $s_1, s_2, \ldots, s_z$ so torej odmiki od začetka T, kjer se pojavi P kot podniz v T.

Problem torej sprašuje, pri katerih odmikih od začetka besedila T se pojavi vzorec P.

Naivni algoritem

Zamisel: za vsak s = 0, 1, . . . preveri, ali je P podniz v T pri odmiku s.

```
procedure Naivno_Iskanje(T,P);
begin
      for s := 0 to n-m do
            j:= 1;
            while j <= m and P[j] = T[s+j] do j := j+1 endwhile;
            if j > m then Izpisi(s) endif
      endfor
end.
```

Časovna zahtevnost v najslabšem primeru

V najslabšem primeru se pri vsakem s = 0, 1, . . . , n − m cel vzorec primerja z delom besedila.

Ko je besedilo T sestavljeno iz n enakih znakov, denimo znakov a, vzorec P pa iz m znakov a.

V najslabšem primeru ima naivni algoritem časovno zahtevnost **$O(n^2)$**.

Da se izboljšati z algoritmov **KMP**, ki ima **linearno** časovno zahtevnost tudi v najslabšem primeru.

Za vsakega od spodnjih algoritmov povejte:

a) Kakšen problem rešuje?

b) Kakšne časovne zahtevnosti O(f(n)) ima?

Pri vsakem odgovoru povejte, kaj pomeni n.

1. **Dvojiško iskanje**

    a) iskanje elementa v urejeni tabeli, deli in vladaj

    b) best: O(1), average: O(log n), worst: O(log n)

    n ... dolžina tabele

2. **Quicksort**

    a) hitro urejanje, deli in vladaj

    b) best: O(n log n), average: O(n log n), worst: $O(n^2)$

    n ... dolžina tabele

3. **Heapsort**

    a) urejanje s kopico

    b) best: O(n log n), average: O(n log n), worst: O(n log n)

    n ... dolžina tabele

4. **Bubblesort**

    a) urejanje z zamenjavami/mehurčki

    b) best: O(n), average: $O(n^2)$, worst: $O(n^2)$

    n ... dolžina tabele

5. **Topološko urejanje grafa**

    a) topološko uredimo aciklični graf

    b) $O(|n|^2)$

    n ... vozlišča

6. **FFT**

    a) pretvorimo koeficientno predstavitev polinoma stopnje n v njegovo vrednostno predstavitev

    b) O(n log n)

    n ... stopnja polinoma

7. **Strassenovo množenje**

    a) množenje matrik

    b) $O(n^3)$

    n ... velikost matrike

8. Ford-Fulkersonov algoritem

a) iskanje največjega pretoka v grafu, dinamično programiranje

b) $O(v*|A|)$

Naj bo $v*$ največji pretok čez $G(V, A, c)$.

$v*$ ... temeljne poti (da izračuna $v*$ mora zasititi kvečjemu $v*$ temeljnih poti)

A ... povezave

9. Bellman-Fordov algoritem

a) iskanje najcenejše poti iz izhodišča v splošnem grafu, dinamično programiranje

b) $O(|V|^3)$

V ... število vozlišč v grafu

10. Floyd-Warshallov algoritem

a) iskanje najcenejše poti med vsemi pari v grafu, dinamično programiranje

b) $O(|V|^3)$

Časovne zahtevnosti

Notranje urejanje

Ω (n logn)

- **selection sort (izbiranje)**: $\Theta(n^2)$
- **bubble sort (zamenjave)**: $\Theta(n^2)$
- **insertion sort (vstavljanje)**: $\Theta(n^2)$
- **quick sort (hitro urejanje)**
  - najslabši primer: $\Theta(n^2)$
  - najboljši primer: $\Theta(n \log n)$
  - povprečni primer: $\Theta(n \log n)$
- **heap sort (urejanje s kopico)**
  - najslabši primer: $\Theta(n \log n)$
  - najboljši primer: $\Theta(n \log n)$
  - povprečni primer: $\Theta(n \log n)$
- **merge sort (zlivanje):** $\Theta(n \log n)$
- **counting sort (urejanje s štetjem)**: $\Theta(n)$
- **radix sort (korensko urejanje)**: $\Theta(n)$
- **binary search:** $\Theta(\log n)$

Matrično množenje

- **navadno**: $\Theta(n^3)$
- **deli in vladaj**: $\Theta(n^3)$
- **Strassen**: $\Theta(n^3)$ -> $\Theta(n^{2.373})$

Množenje števil

- **navadno množenje**: $\Theta(n^2)$
- **deli in vladaj:** $\Theta(n^2)$
- **Karatsubov algoritem**: $\Theta(n^{1.59})$

Iskanje k-tega največjega elementa

- **najslabši primer:** $T(n) = \Theta(n^2)$
- **z Heapsortom:** $\Theta(n \log n)$
- **izboljšani (Blum-Floyd-Pratt-Rivest-Tarjanov) algoritem**: $\Omega(n)$, $O(n)$, $\Theta(n)$, asimptotično optimalen

Izračun produkta polinomov

- **navadni izračun:** $O((n + m)^2)$
- **uporaba vrednostne predstavitve**: $\Theta(n + m)$
- **DFT**: $\Theta(n^2)$
- **FFT (množenje polinoma stopnje n)**: $\Theta(n \log n)$

Največji pretok

- **prerez grafa:** $\Theta(2^n)$
- **Ford-Fulkersonov algoritem**

    Naj bo v∗ največji pretok čez G(V, A, c).

    v* ... temeljne poti (da izračuna v* mora zasititi kvečjemu v∗ temeljnih poti)

    A ... število povezav

    $O(v*|A|)$

Problem nahrbtnika

$O(nV)$, $O(n2^d)$

Najcenejše poti iz izhodišča

- **v acikličnem grafu (topološko, $O(|V|^2)$)**: $O(|V|^2)$
- **v grafu s pozitivnimi cenami (Dijkstra)**: $O(|V|^2)$
- **v splošnem grafu (Bellmann-Ford)**
    - čas: $O(|V|^3)$
    - prostor: $O(|V|)$

Najcenejše poti med vsemi pari

- **Floyd-Warshallov Algoritem**
    - čas: $O(|V|^3)$
    - prostor: $O(|V|^2)$

Iskanje znakovnih podnizov

- **najslabši primer:** $O(n^2)$
- **KMP**: $O(n)$