APS 2 - teorija

(npr. minimizacijski pri optimizacijskih problemih)

① Tehnike reševanja problemov (nahrbtnik, problem najcenejših poti; razveji in omeji (sestopanje))

požrešna, deli in vladaj, dinamično, linearno prog., branch and bound

Dijkstra; ↳ Quicksort, dvojiško iskanje; ↳ Simplex

② Najcenejše poti med vsemi vozlišči - posplošen BF in Floyd-Warshal - značilnosti - časovne zahtevnosti

<u>Floyd-Warshal</u> → graf mora biti brez negativnih ciklov → $\Theta(n^3)$

$$u_{ij}^{(m)} = \begin{cases} c_{ij}, & \text{če } m=0 \\ \min\{u_{ij}^{(m-1)}, u_{im}^{(m)} + u_{mj}^{(m)}\}, & \text{če } m = 1, 2, \ldots, n \end{cases}$$

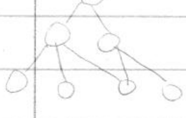
→ cena najcenejše poti iz i v j, ki imajo vmesna vozlišča oznako kvečjemu m

<u>posplošeni Bellman-Ford</u> → graf (brez) negativnih ciklov in graf je povezan → $\Theta(n^3)$

$$u_{ij}^{(p)} \begin{cases} 0; & i=j \\ c_{ij}; & p=1 \\ \min_k\{u_{ij}^{(p-1)}, \min_k\{u_{ik}^{(p-1)} + c_{kj}\}\}; & \text{za } i=1,2,3,\ldots,n \neq j \end{cases}$$

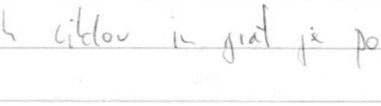
→ cena najcenejše poti iz i do j, ki vsebuje <u>kvečjemu p povezav</u>

③ Dinamično programiranje - splošno



↑ <u>smer reševanja</u>: od manjšega proti večjemu (obratno kot deli in vladaj)

<u>Načelo optimalnosti</u>: vsako podzaporedje optimalnega zaporedja je tudi zase optimalno:



Če so $u_1, u_2, \ldots, u_n$ rešitve problema najcenejših poti iz začetne do vseh drugih točk, potem zanje velja enačba

<u>Bellmanove enačbe - splošno</u>

Imamo graf in veljata Ⓐ in Ⓑ

$$u_i = \begin{cases} 0, & \text{za } i=1 \\ \min_{k \neq i}\{u_k + c_{ki}\}, & \text{za } i=2,3,\ldots,n \end{cases}$$

Ⓐ začetno vozlišče je povezano z vsemi ostalimi

Ⓑ graf nima negativnih ciklov → [illegible]

→obseg: $\mathbb{Z}$ in $\mathbb{C}$ števila

④ DFT splošno

→n-ti primitivni koren enote

- Naj bo V_K vektorski prostor V nad obsegom K
- Naj bo $n = \dim V_K$ (dimenzija V_K)

DEF:

Naj bo v obsegu K element ω z lastnostima:

(a) $\omega^n = 1$

(b) $\omega^j \neq 1$ za $j = 1, 2, \ldots, n-1$

→ ω se imenuje n-ti primitivni koren enote

DFT

↳Def. DFT prostora V_K je linearna transformacija tega prostora z matriko F ... F^{-1} je inverzna transformacija DFT

$$F = \begin{matrix} 0 \\ 1 \\ \vdots \\ n-1 \end{matrix} \begin{bmatrix} & & & \overset{0}{} & \overset{1}{} & \overset{j}{} & \overset{n-1}{} \\ & & & 1 & 1 & 1 & 1 \\ & & & 1 & \omega & \omega^j & \omega^{n-1} \\ & & & \vdots & & & \\ & & & 1 & \omega^i & \cdots & \omega^{(n-1)i} \end{bmatrix}$$

⑤ Kaj je kopica? Časovna zahtevnost za izgradnjo kopice. Zakaj je $O(n)$?

Kopica: → dvojiško drevo (vozlišča hranijo elemente ali le njihove ključe)

↳ je urejeno: za $\forall$ vozlišče: ključ v vozlišču je $\geq$ od ključa v (usaklenem) sinu

→ levo uravnoteženo

Časovna zahtevnost (gradnje) kopice $\neq \Theta(n)$

→ začnemo od zadnjega očeta (listi so trivialne kopice), po nivojih od desne proti levi, in spodaj navzgor, pri čemer bi pri $\forall$ vozlišču poklicali Popravi_v_kopico

Sestavi_v_kopico kliče proceduro Popravi_v_kopico v zanki $i = \lfloor \frac{n}{2} \rfloor, \ldots, 1$, zato se izvrši vsega skupaj $Z(n)$ zamenjav oz. primerjav

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

⑥ Sestopanje, ~~[illegible]~~ razveji in omeji

↳ sistematično pregledujemo <u>vse možnosti rešitve</u>. Sestopanje lahko ponazorimo z drevesom stanj, kjer rešitev najdemo v listih oli pa je rešitev pot od korena do lista.

→ Ko neka pot od korena do lista ni več obetavna, se vrnemo nazaj in poskusimo po drugi poti.

Časovna zahtevnost:

(v drevesu)

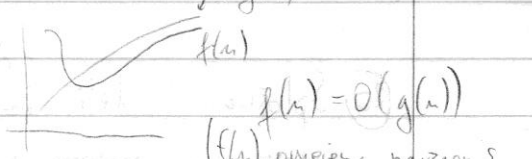
Lahkost pregledovanja odvisno od tega, koliko poti ≠ zavrženo na raznih pogojev. V splošnem rešitve ne dobimo v polinomskem času. Ko zavržemo kakšno rešitev, zavržemo celotno poddrevo.

<u>Razveji in omeji</u> - ... obetavnost gledamo

↳ izboljšano sestopanje: ne iščemo po vejah, kjer ni boljših od že pridobljene

↳ za veliko št. stanj, pri maksimalnem številu lahko preveč potratno zaradi <u>ocene vrednosti</u>

⑦ $O(.)$ notacija (asimptotična ...)



$f(n) = O(g(n))$

($f(n)$ omejena navzgor s funkcijo $g(n)$)

maksimum: $O(n)$ → omejitev navzgor

minimum: $\Omega(n)$ → omejitev navzdol

točna vrednost: $\Theta(n)$ → isti velikostni red

funkcija omejena navzdol s f. $h(n)$

⑧ Navadno vstavljanje (insertion sort), časovna zahtevnost

↳ izboljšanje je shell sort; da mesto za a[0] poiščemo z dvojiškim iskanjem ni izboljšava

↳ vzamči 1. element v neurejenem delu in ga dāš na pravo mesto v urejenem

→ Čas. zahtevnost: $O(n^2)$

$C_{min} = O(n)$

$C_{max} = O(n^2)$

$C_{average} = O(n^2)$

⇓ upoštevati moramo še premetitve

(8.1) Navadno izbiranje (Selection Sort) → $O(n^2)$
↳ v neurejenem delu iščemo najmanjšega in ga damo na konec urejenega dela

(8.2) Navadno zamenjave (Bubble Sort) → $O(n^2)$ → izboljšanje: Quicksort
↳ primerja od zadaj proti urejenemu delu in zamenja elementa, če je potrebno

(9) Topološko urejanje grafa - Na kakšnih grafih; kako izgledajo Bellmanove enačbe?
↳ na usmerjenih grafih, + acikličnih
topološko: ∀ povezava poteka iz vozlišča z nižjo oznako v vozlišče z višjo oznako.

Bellmanove enačbe: $u_i \begin{cases} 0, & i=1 \\ \min\limits_{k} \{u_k + c_{ki}\} & k \not< i \end{cases}$

(10) Splošno o deli in vladaj + v zvezi z njim Master Theorem (je metoda)
↳ osnovno nalogo razdelimo na podnaloge (ko jih rešimo dobimo delne rešitve)

Časovna zahtevnost $T(n) = \begin{cases} \Theta(n^r) & ; \text{če } a < c^r \to \text{polinomski} \\ \Theta(n^r \cdot \log n) & ; \text{če } a = c^r \to \text{logaritemsko-polinomski} \\ \Theta(n^{\log_c a}) & ; \text{če } a > c^r \to \text{eksponentni} \end{cases}$

c - število podnalog
a - št. podnalog, ki jih rešimo

+ dobimo: razbitje + združevanje
$O(\dots) + O(\dots) = O(b \cdot n^r)$

$a=2 \to$ Quicksort
$a=8 \to$ Vinograd $(c=2, r=2, b=1)$
$a=7 \to$ Strassen $(c=2, r=2, b=1)$

$a=1 \to$ binarno iskanje
$a=1 \to$ iskanje k-tega po velikosti

(11) Bellmanove enačbe

↳koliko enačb imamo pri grafu z n vozlišči in m povezavami
↳toliko kot je vozlišč

Bellmanova enačba + kaj predstavlja

za splošne grafe

$$u_i^{(p)} = \begin{cases} 0, & i=1 \\ c_{1i}, & p=1 \\ \min\left\{u_i^{(p-1)}, \min\left\{u_k^{(p-1)} + c_{ki}\right\}\right\}, & i=2,3,\dots,n \end{cases}$$

(12) Iskanje k-tega elementa po velikosti + še 1 algoritem, kjer je a manjši od c

→za delilni element vzameš tistega ~~[illegible]~~, katerega po vrsti izčeš.
→Laredis Quicksort
→vsakič vzameš za delilni element

a manjši od c ⇒ binarno iskanje

(13) Zunanje urejanje - če je podatkov veliko
- kakšne poznamo in kakšne so izboljšave
- navadno zlivanje (porazdeljevanje + zlivanje se ponavlja) → $\Theta(N \cdot \log N)$
- večsmerno
- uravnoteženo zlivanje (porazdeljevanje, zlivanje se ponavlja)
- uravnoteženo kosmerno zlivanje
- naravno zlivanje (= uravnoteženo s cetami) → $\Theta(N \cdot \log_{\lceil c \rceil})$ → število cet
- polifazno (cete, navidezne cete, fibonaccijeve št. cet na trakovih)
↳ predurejanje skupico (podaljšamo cete)

(14) Quicksort → $\Theta(n \cdot \log n)$, najslabše $\Theta(n^2)$

n = delilni element
↳lahko je to: - sredji element
- zaključni el.
- $\frac{\text{prvi} + \text{sredji} + \text{zadnji}}{3}$ (povprečje)

Quicksort

najslabši primer:

→ tabela vedno razpade na enega + vse ostale

↳ za delilni el. vedno vzamemo največji element

↳ (izrodi se navadno izbiranje)???

(15) Rekurzivni FFT → $\Theta(n \cdot \log n)$

→ polinom razbijemo na liki in sodi del → $p(x) = p_1(x^2) + x \cdot p_2(x^2)$

p_1 p_2 (stopnje r, oba)

$r = \frac{n}{2}$ → toliko sodih ik, toliko lihih potenc

→ polinom se (vedno) v vsakem koraku razdeli na 2 dela

→ velja $(w^{k+r})^2 = w^{2k}$, $k = 0, 1, \ldots, (r-1)$

levi so (prva polovica): $p(w^k)$, druga polovica pa so $p(w^{r+k})$, pri čemer

$$p(w^k) = w^k \cdot p_1(w^{2k}) + p_2(w^{2k})$$
$$p(w^{r+k}) = w^{r+k} \cdot p_1(w^{2k}) + p_2(w^{2k})$$
$$= -w^k$$

(16) Linearno programiranje → simplex

matrika

$A \cdot \vec{x} \leq \vec{b}$ ⇒ iščemo $\vec{x}$, ki izpolnjuje pogoje in ki maksimizira funkcijo $\vec{c} \cdot \vec{x}$

A reda $m \times n$

$\vec{b}$ reda m

$\vec{c}$ reda n

↳ funkcije narišemo (dolimo konveksno poliedrsko množico) → v ekstremnih točkah je rešitev (moramo pa prenehati v izhodišče)

Simplexni algoritem

Za računanje maksimuma: za (ekstremno) eno točko izračunaš vrednost in potem gledaš vse sosednje točke: če so vrednosti v sosedah manjše ali enake od naše vrednosti, je to globalni max, sicer se premakni v točko z večjo vrednostjo

→ hitri algoritem → $\Theta(n^2)$ n = velikost koeficientne predstavitve produkta

(17) DFT

koeficienti koeficientna

vrednosti v točkah t_n vrednostna predstavi

→ vrednostna predstavitev

$p(x) \equiv [a_0, a_1, \dots, a_{77}]$? ω $(p(t_1), p(t_2), \dots, p(t_{172}))$

$q(x) \equiv [b_0, b_1, \dots, b_{94}]$? $(q(t_1), q(t_2), \dots, q(t_{172}))$

$p(x) \cdot q(x) \equiv [c_0, c_1, \dots, c_{171}]$? ω^{-1} $(p(t_1)q(t_1), \dots, p(t_{172})q(t_{172}))$

Vrednostno predstavitev dobimo iz koeficientne z DFT.

→ konvolucija (polinomov)

↳ je koef. predstavitev produkta $p(x) \cdot q(x)$.

(18) Strassovo množenje matrik

↳ zmanjša število množenj (matrik) iz 8 na 7

$\Theta(n^{\log_2 8}) \Rightarrow \Theta(n^{\log_2 7}) = \Theta(n^{2,8})$

↳ splača se pri $n \geq 45$ in gostih matrikah

(19) Nahrbtnik 1-0 (problem lahko rešimo z linearnim (simplex) programiranjem, vendar ga rešujemo z dinamičnim programiranjem)

→ V nahrbtnik dodajaš predmete po vrsti

→ Po vsakem dodanem predmetu izločimo kombinacije, ki presegajo volumen nahrbtnika. Če sta 2 kombinaciji z enako ceno, vzamemo tisto z manjšim volumnom, drugo pa zlamo. Če sta 2 kombinaciji z istim volumnom, vzamemo tistega z večjo ceno.

→ po zadnjem dodanem predmetu vzamemo tistega z največjo ceno

→ pogledaš, kateri predmeti so v nahrbtniku

$\Theta(n \cdot C)$ $C = \sum_1^n c_i$

(20) Shell sort - izboljšano ustavljanje

- imamo $(7, 3, 1)$ (h_1)

1. razdelimo na h_1 podtabel
2. razdelimo na h_2 podtabel
3. -11- h_3 podtabel

Elemente z istimi indeksi uredimo z navadnim ustavljanjem.

(ZAS) najslabši = povprečen

optimalen za algoritme, ki uporabljajo primerjave.

(21) Heapsort (urejanje s kopico) $\Theta(n \cdot \log n)$

→ Daj vsa števila v kopico in jo uredi

→ izloči koren kopice (in ga shrani)

→ namesto korena postavi zadnji list v drevesu

→ popravi dobljeno drevo v kopico (koren se pogrezne na pravo mesto)

max. kopica ⇒ naraščujoče

(22) Dijkstra → $O(n^2)$

↳ pozitivne povezave, brez negativnih oz. obiskanih

→ imamo 2 množici: - množica pregledanih točk → P

- -11- nepregledanih točk → T

↳ postavimo se v točko z najmanjšo potjo (na začetku je to izhodišče) ter si shranimo poti do sosedov, ki so krajše od poti v tabeli skupaj s prejšnjo točko. Ponavljamo, dokler ne obiščemo vseh točk.

(23) Ocenjevanje porabe časa rekurzivnih programov (razlaga + formula)
↳ glej Deli in vladaj

(24) Spodnja meja za urejanje tabel / notranje urejanje
→imamo obliko dvojiškega drevesa in operacijo primerjanja <
↳v listih so vse možne permutacije števil
↳v notranjih vozliščih so operacije primerjanja
↳višina takega drevesa je $\geq \lceil \log_2 n! \rceil$ →enakost velja, ko je drevo p
↳število je listov

$n! \overset{Stirling}{\doteq} \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \Rightarrow n\log n - \frac{1}{2}\log n + 1{,}44n \Rightarrow$

$O(n \cdot \log n)$

Število primerjanj je vsaj $\Omega(n \cdot \log n)$

(25) Pretoki →graf je označen in usmerjen
↳linearno programiranje
↳Ford-Fulkersonov algoritem

→vrednost maksimalnega pretoka je enaka kapaciteti minimalnega prereza
lastnosti: ① $0 \leq x_{ij} \leq c_{ij}$

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

→pozitivna povezava je zasičena, kadar je $x_{ij} = c_{ij}$
negativna —||— $x_{ij} = 0$

→Pot P je zasičena, če vsebuje vsaj eno zasičeno povezavo (pozitivno ali negativno

algoritem:

→začni v izviru, označuj ostala vozlišča dokler ne označiš ponora

→označuj tako, da lahko rekonstruiraš nezasičeno pot

→pot zasiči

točka i je lahko { neoznačena; označena { nepregledana ($\Leftrightarrow$ ima neoznačenega soseda); pregledana ($\Leftrightarrow$ vsi njeni sosedje označeni) }

Začetek: izvor (η): označena - nepregledana, njena oznaka: $(-, \infty)$

ostale točke: neoznačene

ko označujemo damo oznake: (i^+, δ_j) (i^-, δ_j) $\quad \delta_j = \begin{cases} \min\{\delta_i, c_{ij} - x_{ij}\} \\ \min\{\delta_i, x_{ij}\}, \text{če} \end{cases}$

Zahtevnost: $O(m \cdot v_{max})$ — čas. zahtevnost odvisna od velikosti rezultata (ki nam vseč)

m: toliko časa za izgradnjo ene poti

v_{max}: največ toliko je nezasičenih $\Rightarrow$ treba zasičiti

$O(m^2 \cdot n)$ - hitrejše

$\downarrow |V|$

V primeru, ko "vozlišča postanejo pregledana v istem vrstnem redu kot postanejo označena" postane dg. s časovno zahtevnost

(26) Požrešna metoda

Z njo rešujemo optimizacijske probleme. Problem rešujemo kot končno zaporedje podproblemov. V vsakem koraku izberemo med delnimi rešitvami tisto, ki daje trenutno največji profit.

- razvrščanje zapisov na magnetni trak
  - uredi zapise po dolžini
  - zapiši jih po naraščajočih dolžinah