APS 2 2009/2010

Ustni faq

KAZALO

UVOD:

TEHNIKE REŠEVANJA PROBLEMOV

- **Požrešna** (Dijkstra) (išče lokalne optimizacije, upamo da pridemo do globalne optimizacije)
- **Deli in vladaj** (Quicksort) (razdelimo na manjše podprobleme in iz njihovih rešitev, dobimo rezultat za prvotni problem)
- **Dinamično programiranje** (Nahrbtnik, problem najcenejših poti)(način reševanja problemov, ki sestojijo iz podproblemov, ki se prekrivajo ali katerih rešitev vključuje iskanje optimalne sestave)
- **Linearno programiranje** (Simplex)(z danimi pogoji hočemo dobiti, max ali min linearne funkcije)
- **Razveji in omeji** (Sestopanje)(metoda za iskanje rešitve problema, pri kateri hranimo možne rešitve in ocenjujemo njihovo obetavnost)

PREVERJANJE PRAVILNOSTI ALGORITMOV

- s poskusi - izbereš vhode in preveriš, če je izhod OK
- z logično analizosklepanje – predikati, diagram poteka, aksiomi
    - pravilo za stekališče poti – če velja nekaj pred stekališčem, naj velja tudi po njem
    - if – če nekaj velja potem gremo na eno stran drugeče pa na drugo??
    - z diagramom poteka (dvojiško drevo – višina = log2listov) lahko tudi dobimo najmanjše število možnih operacij, ki je še dosegljivo
- preverjanje ustavljanja (tehnika monotono padajoče funkcije)

PORABA ČASA IN PROSTORA

- O() NOTACIJA
- O(n)-omejena navzgor
- Ω(n)-omejena navzdol
- Θ(n)-asimptotično enaka

UREJANJE:

NOTRANJE UREJANJE

Podatki so maloštevilčni zato lahko vse lahko hranimo v pomnilniku.

NAVADNO VSTAVLJANJE (INSERTION SORT)

Vzame vedno prvi element iz neurejenega dela in ga vstavi na ustrezno mesto v urejenem delu (vrinemo). Ustrezno mesto se splača poiskati z metodo bisekcije.
Časovna zahtevnost algoritma je O(n^2) oz. v najboljšem primeru O(n).

NAVADNO IZBIRANJE (SELECTION SORT)

Sprehodiš se od leve proti desni v neurejenem delu in vedno zamenjaš element z najmanjšim.
Včasih sicer pride bolj prav (pri npr. seznamu), da najdemo najmanjši element ter ga vrinemo na začetek (ta naj bi se delal na vajah). O(n^2).

NAVADNE ZAMENJAVE (BUBBLE SORT)

Se sprehodi od desne proti levi, po potrebi elemente zamenjuje (sosede), časovna zahtevnost O(n^2).

Zadevo lahko pohitrimo tako, da gledamo kje se je zgodila zadnja zamenjava. Do tam kjer se je zgodila je zadeva urejena tako, da naslednjič rabimo iti samo do tja.

IZMENIČNE ZAMENJAVE (SHAKER SORT)

Isto kot navadne zamenjave samo, da uvedemo še sprehod iz leve proti desni, ki po potrebi elemente zamenjuje (sosede). O(n^2)

Pohitritev na podoben način le, da za obe strani.

IZBOLJŠANO VSTAVLJANJE(SHELL SORT)

V bistvu gre za to, da delamo navadno vstavljanje na tabelah, kjer ena tabela predstavlja elemente, ki so odaljeni za h mest (npr. …, 15, 7, 3, 1).
Število tabel dobimo po enačbi t = log2n – 1.

Časovna zahtevnost O(n^1.2).

UREJANJE S KOPICO

Sprva je potrebno kopico sestaviti kar vzame O(n) časa. Potem pa dokler ne zmanjka elementov:

- Izloči koren in ga nekam shrani
- Namesto korena vstavi zadnji list
- Popravi kopico

Časovna zahtevnost je O(n log n).

KAJ JE KOPICA

Levo-uravnoteženo urejeno dvojiško drevo. Osnovne operacije:

- Brisanje najmanjšega – zbriše prvi element, da zadnji list na koren ter kopico uredi - O(log n)
- Dodajanje elementa – doda element na zadnji list, ter preuredi - O(log n)

UREJANJE Z PORAZDELITVAMI (QUICKSORT)

Princip: Razdelimo tabelo na 3 podtabele t1, t2 in t3, kjer t2 predstavlja en element (pivot), t1 predstavlja elemente manjše od pivota in t3 predstastavlja elemente večje od pivota. Zatem postopek rekurzivno ponovimo na tabelah t1 in t3.

Zadevo izvedemo tako, da gremo kot prvo iz leve in iščemo element, ki je večji oz enak pivotu. Zatem gremo iz desne in iščemo element, ki je manjši oz. enak od pivota ter elementa zamenjamo. Če se iskanje prekriža zadeve ne menjamo, ampak le razdelimo na 2 oz. 3 dele.

Časovna zahtevnost je v najslabšem primeru O(n^2) v povprečju oz. najboljšem pa O(nlogn).

ISKANJE K-TEGA ELEMENTA PO VELIKOSTI

S štetjem elementov v tabelah lahko ugotovimo v kateri podtabeli se nahaja element ter nato nadaljujemo z iskanjem samo v tisti podtabeli. O(n) in O(n^2)

COUNTING SORT

Za vsak element izračunamo mesto kam spada zato vemo kam ga moramo dat, zato ni nobenih primerjav. O(n)

ZUNANJE UREJANJE

Velika količina podatkov, zato jih hranimo jih na disku. V določenem trenutku vidimo le tiste trakove(datoteke), ki so v ramu.

NAVADNO ZLIVANJE

- na vhodnem traku je zaporedje podatkov, ki jih želimo urediti
- na voljo sta tudi dva izhodna trakova

Potem ponavljamo dokler ni vse urejeno na enem traku:

- na vhodnem traku so urejena zaporedja dolžine k
- porazdeli jih (izmenično jih zapiši) na dva izhodna trakova
- zlij po dve zaporedji (dolžine k) z izhodnih trakov v eno zaporedje dolžine 2k na vhodni trak

Čas. zahtevnost: O(N*log_2(N))

URAVNOTEŽENO ZLIVANJE

Zaporednji namesto, da zlijemo na en trak že kar izmenično porazdelimo na dva trakova.

VEČSMERNO ZLIVANJE

Navadno zlivanje s tem, da uporabimo n trakov.

VEČSMERNO URAVNOTEŽENO

Uporabimo torej 2n trakov, ter iz prvih n-trakov prestavljamo na drugih n-trakov.

O(N log N).

NARAVNO ZLIVANJE

- ne delamo s četami fiksne dolžine ampak jih naravno jemljemo kot so na traku. Vse naraščajoče skupaj vedno vzamemo kot četo
- na primer če imamo na začetku zaporedje 1 5 8 3 2 6 7 9 5 6 2 3, potem dobimo iz tega čete: 1 5 8; 3; 2 6 7 9; 5 6; 2 3

O(N*log C)
C-št čet

POLIFAZNO ZLIVANJE

Pri uravnoteženem, naravnem večsmernem zlivanju opazimo da je izkoriščenost izhodnih trakov nizka. Od n izhodnih trakov je aktiven le eden, ostali mirujejo in čakajo da pride na njih vrsta.

Zamisel
Imejmo le n trakov, vsak trak bo lahko zamenjal svojo vlogo – včasih bo vhodni, drugič pa izhodni. V vsakem trenutku bo n-1 vhodnih in le en izhodni trak.

Kako to zagotoviti
Vhodne trakove zlivajmo na izhodnega tako, da bo vedno eden med vhodnimi usahnil oz. da se bo izpraznil, na koncu zlivanja na njem ne bo nobenih podatkov več. Ko nek vhodni trak usahne, takoj postane naslednji izhodni. Prejšnji izhodni pa postane vhodni.

Začetna razporeditev čet:

1. Naredimo fibonačo/tribonači/xbonaci tabelo
2. Čete razporedimo po trakovih glede na tabelo
3. Pazimo, če pride do zlitja čet

Potem iz tabele izberemo "najmanjšo" tabelo, da naše razporejene čete še pašejo vanjo. V primeru, ko imamo premalo elementov se doda navidezne čete na začetek ter nato začne zlivati.

ALGORITMI Z REKURZIVNIM RAZCEPOM:

MNOŽENJE MATRIK

Najpreprostejši pristop: zračunaj po vrsti vseh n^2 elementov (n^3 množenj, n^3 seštevanj = O(n^3))

WINOGRADOVO MNOŽENJE

Zmanjša št. množenj za 50% (več seštevanja, ampak seštevanje se izvede hitreje), toda asimptotično je še vedno reda O(n^3). n mora biti sodo, drugače dopolnimo z praznim stolpcem in vrstico.

PI[i,k,j] = (A[i,2k-1] + B[2k,j])*(B[2k-1,j] + A[i,2k]);
a[i,k] = A[i,2k-i]*A[1,2k];
b[k,j] = B[2k,j]*B[2k-1,j];
C[i,j] = sum(PI[i,k,j]-a[i,k]-b[k,j]) od x=1 do n/2;

REKURZIVNO MNOŽENJE

n je potenca števila dva. razstavimo na podmatrike in množimo podmatrike (rekurzija).
(Isto ko winograd?) – je verjetno bolj mišleno kot osnova za strassovo množenje.

STRASSOVO MNOŽENJE MATRIK

Je postopek tipa deli in vladaj za množenje matrik, ki zahteva 7 množenj (O(n^~2.81)) namesto običajnih 8 (O(n^3)). Matriki ki jih množimo morata biti kvadratni, če nista vstavljamo ničle dokler ne postaneta kvadratni.

Bistvo je to, da imamo manj bločnih množenj.

DFT

SPLOŠNO

Večinoma smo jo mi omenjali za množenje polinomov. Se dosti bolj splača kot "bruteforce" metoda, kjer bi pri polinomu velikosti n in drugem polinomu velikosti m morali med seboj koeficiente množiti m*n-krat, pri DFTju (brez da upoštevamo časovno zahte1vnost DFTja...) pa izvedemo samo m+n+1 množenj.

REKURZIVNI ALGORITEM

//pomojem ni treba znat dejansko zračunat, če je pa huda želja se pa v zapiskih najde nekih slavnih 14 točk kako to delat ki so baje zelo nerazumljive :p

O(n*log(n))

KONVOLUCIJA POLINOMOV

Množenje polinomov z DFT, kjer samo zmnožimo istoležne elemente vektorja.

POŽREŠNI ALGORITMI:

OSNOVNI ALGORITEM ZA ISKANJE NAJBOLŠEGA ELEMENTA

Na silo preišči celo množico – Bruteforce. Časovna zahtevnost torej verjetno O(n).

POŽREŠNI ALGORITEM

Med dopustnimi elementi dodaj v S (najboljša podmnožica) tistega ki najbolj izbolša vrednost S. Včasih je težko dokazati, da je S res najboljša.

LINEARNO PROGRAMIRANJE:

Problem linearnega programiranja lahko opišemo kot matematično metodo za doseganje najboljšega možnega rezultata s podano funkcijo in pogoji. Splošna shema je Ax=b, iščemo tak vektor x, ki maksimizira produkt x*c; (c, x sta vektorja)

MAKSIMALEN PRETOK

Kolikšen maksimalen pretok lahko v danem grafu dosežemo iz dane začetne točke v končno točko. V bistvu gremo iz začetne točke v končno dokler to lahko delamo (je še možno povečati pretok) ter jo pri tem zasičimo. Pri tem je možno uporabiti tudi obrnjeno smer, če je v njej že kaj pretoka ("preusmerimo" pretok).

MAKSIMALEN PREREZ

Ko izvedeš(izračunaš) maksimalen pretok, razdeliš graf na dva dela. V prvem mora biti izvor v drugem pa ponor. Črto (mejo) vmes lahko potegneš samo čez polno zasičene povezave v grafu.

Če potem seštejemo vrednosti vseh povezav ki grejo iz naše prve množice v drugo, dobimo ravno maksimalni pretok tega grafa.

FORD-FULKERSONOV

Časovna zahtevnost O(m*v), kjer je m število povezav. V primeru irracionalnih števil obstaja možnost, da vseh povezav ne moremo zasititi.

SIMPLEX

1. Postavi se v ekstremno točko KPM (konveksna poliedrska množica)
2. Ali obstaja sosednja točka z boljšo vrednostjo f?
3. NE – konec, našli smo globalni maksimum
   DA – Postavi se v tega soseda in gre na korak 2.

O(2^n)

DINAMIČNO PROGRAMIRANJE:

Je pristop k reševanju problemov ki so podobni tistim ki se jih rešuje z rekurzivnim razcepom, vendar so pri dinamičnem programiranju tej podproblemi med seboj odvisni, zato moramo hraniti vse rešitve podproblemov. Takih problemov je precej, omenili smo 0-1 nahrbtnik ter iskanje najcenejših poti.

NAČELO OPTIMALNOSTI

Zaporedje odločitev O1,O2,....On, je optimalno če nas privede do optimalne rešitve, kar pomeni da je vsako podzaporedje optimalnega zaporedja tudi optimalno.

0-1 NAHRBTNIK

worst case O(2^n), best pa O(n).

SESTOPANJE

Splošno:
Metoda uporabna pri problemih, kjer rešitev sestavljamo postopoma. Rešitev zapišemo kot vektor, ki mora zadoščati nekemu pogoju. Elemente vektorja pobiramo iz neke končne množice kandidatov (recimo K). Podobno kot za 3. seminar.

Metode:

- Naivni algoritem z obhodom po drevesu se pregleda vse poti od korena do listov (zahtevnost 2^n)
- Izboljšan algoritem rabimo neko funkcijo ki nam pove če so delne rešitve obetavne.

NAJCENEJŠE POTI

Gre za tipičen optimizacijski problem, ki ga rešujemo s pomočjo dinamičnega programiranja.
Kateri so: Dijkstra, Bellman-Ford, Floyd-Warshal

MED ZAČETNIM IN VSEMI OSTALIMI

TOPOLOŠKO UREJANJE GRAFA + BELLMAN(UNNAMED METODA... >_<)

Graf najprej topološko uredimo da preverimo ali je cikličen. Če ni cikličen, nadaljujemo z našim (očitno neimenovanim?) algoritmom, ki je v bistvu dokaj enak Dijkstri, le da točke menjamo namesto po kriteriju katera je najmanjša, tako kot smo jih pri topološkem urejanju.
O(v^2) v št. vozlišč

DIJKSTRA

Uporablja se za iskanje najkrajše poti v grafu, a le na takšnih grafih, kjer ni negativnih povezav.
Začnemo v enem vozlišču od kjer so vse povezave do drugih točk "neskončno" oddaljene (sama do sebe ima 0).
Zatem pa ponavljamo:

- pogledamo do katere točke(X) je najkrajša pot in jo izberemo
- za vse točke(ai) na katere se točka X povezuje seštejemo Y = razdalja do X + "razdalja od ai do X". V primeru, če je ta razdalja krajša od trenutne najkrajše do ai je nova dolžina do ai enaka Y.

Po wiki je časovna zahtevnost O(E + V * log V), kjer E predstavlja št. povezav in V št. Točk – verjetno gre za to, da je še kopica zraven. Po zapiskih pa je O(n^2).

BELLMAN-FORDOV ALGORITEM

Podobno kot Dijkstra, le da omogoča negativne povezave. Med postopkom preverja koliko povezav je že naredil, da jih ne dela nikoli več kot n.

O(n^3)

FLOYD-WARSHALLOV ALGORITEM

Časovna zahtevnost O(n^3).

NAJCENEJŠE MED VSEMI PARI

POSPLOŠEN BF IN FLOYD-WARSHAL

Oba se uporabljata za iskanje najcenejših poti med vsemi točkami. Pri posplošenem BF je ideja da za osnovo uporabimo BF, vendar ga postavimo v matriko. S tem še nič ne pridobimo in je časovna zahtevnost še vedno O(n^4), vendar če za množenje matrike uporabime iterativno kvadriranje pade časovna zahtevnost na O(n^3*log2(n)). Floyd-Warshal je podoben, vendar si podnaloge algoritma razdeli glede na največji indeks v dani poti, to pa na koncu pripelje do časovne zahtevnosti O(n^3). <- *no idea why, če kdo ve naj dopiše*

BELLMANOVE ENAČBE

Definicija:

Ui – cena najcenejše poti iz 1 do i.

ODGOVORI Z FORUMA-DODATNE RAZLAGE

DFT

Ce jo uporabis za resevanje konvolucije, to je mnozenje polinomov, moras nad produktom narest se inverzno transformacijo.
Se pravi za primer konvolucije naredis dft nad obema polinomoma, zmnozis istolezne in potem inverznmo.
Torej mors DFT naredit 3x za konvolucijo.

Naivni dft, je reda n^2, hitri, to je rekurzivni pa nlogn, ga delas po deli in vladaj in racunas samo polovico polinoma, ostala se ponovi ...
Konvolucija brez dft je reda n^2, z hitrim dft pa 3*nlog+n za mnozenje istoleznih=O(nlog), glede prakse ne vem, ampak uporablja se pri vseh postopkih, kjer je treba polinome velikih stopenj mnozit, kar je verjetno tudi pri obdelavi šuma...(spreminjanje frekvence, amplitude se dela z mnozenjem funkcij)

DFT se v audio tehnologiji uporablja za to, da prevedeš zapis zvoka iz časovnega v frekvenčni prostor. Zvok je namreč posnet tako, da zgleda kot valovanje. In če hočeš ti karkoli studijsko zmontirat, rabiš imet zapis zvoka v frekvenčnem prostoru, ker frekvenčne pasove ojačuješ in oslabuješ, potem pa spet pretvoriš nazaj v časovni prostor, ki je uporaben za predvajanje.

Praktičen primer tega so razni equalizerji, veliko avtoradiev, recieverjev, in softwara za predvajanje glasbe ima to že vgrajeno noter. Pri winampu recimo ti prikazuje tiste stolpce, ki hodijo gor in dol, kar predstavlja jakost na določenih frekvenčnih pasovih. Pri winampu imaš tudi več možnosti prikaza in lahko prikaz v frekvenčnem prostoru zamenjaš za prikaz v časovnem prostoru, in potem namesto stolpcev vidiš eno črto, ki se upogiba sem ter tja...

DELI IN VLADAJ

tehnika reševanja problemov, kjer glavni problem razbijes na manjše podprobleme, ki jih rešuješ neodvisno od

glavnega, iz njihovih rešitev pa sestaviš rešitev glavnega problema,
V praksi so ti podproblemi ponavadi enaki glavnemu, resujemo jih lahko rekurzivno z enakim algoritmom, ki kliče samega sebe.

Formula za izračun časovne zahtevnosti glavnega problema pa je:
T(n)=b pri n=1
sicer
a* T(n/c) + f(n),
kjer je funcija f(n) čas razbitja/sestavljanja delnih rešitev skupaj ter je v praksi polinomska oblike b*n^r, kjer je b pač neka konstanta...

Ker je natačna rešitev rekurivne enačbe T(n) "preveč" natančna, jo lahko opišemo glede na asimptotično vedenje in sicer a, c in r:
a= število podproblemov ki jih moramo izračunat, c pa število vseh podproblemov... r pa je tista potenca polinoma za časovni zahtevnost razbitja/sestavljanja.

Master theorem:
a<c^r sledi T(n)=O(n^r)
a=c^r T(n)=O(n^r *logn)
a>c^r T(n)=O(n^(logc(a)))

a=število podproblemov, ki jih je potrebno rešit
c=število podproblemov
b=konstanta pred polinomom časovne zahtevnosti za sestavljanje rešitve
n=prvotna velikost problema
r=stopnja polinoma časovne zahtevnosti za sestavljanje rešitve

RAZVEJI IN OMEJI

- za optimizacijske probleme, npr. minimizacijske
- iščemo neko rešitev x=(x1,x2,...,xn), ki ima najmanjšo ceno: c(x) = min c(y); y je dopustna rešitev

Denimo, da smo v drevesu prišli do nekega xi-1. Torej imamo delno rešitev x1,x2,...,xi-1, ki ima vrednost c(x1,x2,...,xi-1).
Recimo, da vemo, da je vrednost (še neznane) optimalne rešitve manjše ali enako M.

<u>Kako bi to lahko sploh vedeli?</u>
- Če smo že prej našli neko rešitev y1,y2,...,yn lahko tedaj rečemo da M=c(y1,y2,...,yn). Tako vemo, da minimalna rešitev ne bo večja od M.
- Sedaj pogledamo, koliko bi vsak od kandidatov za xi prispeval k ceni c(x1,x2,...,xi-1).

Če se za nekega kandidata ki izkaže, da je M<c(x1,x2,...,xi-1,ki), potem ki gotovo NE vodi k optimalni rešitvi, zato ki opustimo.
Vrednost M potem lahko izboljšamo vsakokrat ko pridemo do nove rešitve!

Zakaj razveji (branch)?

Sestopanje (čisto) je pregledovanje iskalnega drevesa v GLOBINO.
Pri Razveji in omeji (BB) pa lahko pregledujemo tudi v ŠIRINO.

Ko BB doseže neko vozlišče, njegove naslednike (kandidate za nadaljevanje) postavi na seznam čakajočih kandidatov S.
S pa lahko implementiramo kot sklad, vrsto, ...

BELLMANOVE ENAČBE

0-1 nahrbtnik (kniga 132)

k_i(W) = max(k_(i-1)(W), k_(i-1)(W-vi) + ci)

dijkstra (kniga 147)

u_j = min(u_j, u_k + c_kj)

topološko urejanje (kniga 144) - tuki nism 100%

u_i = min_k<i(u_k+c_ki) kjer je < un tak mal zaviti pač k je predhodnik i-ja

splošni bf (kniga 149)

(u_i)^h = { 0 če je i = 1;
c_ij če je h = 1;
min{(u_i)^(h-1), min{(u_k)^(h-1) + c_ki}}}