



PRIJEVODI RJEŠENJA ZADATAKA CEOI 2005.

ZADATAK: Depot Rearrangement

Neka nam je n broj trgovina i m broj proizvoda.

Sad promatrajmo usmjeren graf $G=(V, E)$, gdje

$$V=\{p_1, p_2, \dots, p_n\} \cup \{q_1, \dots, q_m\}$$

Za svaki i i j skup veza E od G sadrži (p_i, q_j) k puta ako i samo ako je broj kontejnera označenih sa j na području $[m^*(i-1)+1, m^*i]$ jednak $k+1$ i $k > 0$.

Par (q_j, p_i) se nalazi u skupu veze ako i samo ako ne postoji kontejner označen sa j u intervalu $[m^*(i-1)+1, m^*i]$.
Sad je jasno da svaki čvor ima jednak broj čvorova koji ulaze i izlaze iz njega. Dakle takav problem nam rješava algoritam zvan Eulerian circuit. Nju ćemo primijeniti na na svaku komponentu grafa i tad na je ukupan broj premještanja jednak sljedećem izrazu:

$$c + \sum_{j=1}^m \text{BrojIzlaza}(q_j). \text{ Gdje nam je } c \text{ broj komponenata u grafu.}$$

Dakle radimo Eulerevu turu.

Složenost:

Vrijeme: $O(|E|+n^*m)$

Memorija: $O(|E|+n^*m)$

ZADATAK: Mobile Service

Ovaj zadatak se rješava metodom dinamičkog programiranja. Prvo uočimo da prije zahtjeva r_i jedan od trojice zaposlenika mora biti na mjestu zahtjeva r_{i-1} . Sada ćemo za svaki i promatrati sve moguće pozicije druge dvojice radnika(jedan se nalazi na trenutnom zahtjevu).

Rješenje podzadatka A je onda: $\min_{v_2, v_3} \text{Opt}(n, v_2, v_3)$.

Uočimo da je $\text{Opt}(i, v_2, v_3)$ simetrično, tj. $\text{Opt}(i, v_2, v_3) = \text{Opt}(i, v_3, v_2)$.

Te ako je $r_i = r_{i-1}$ tada je $\text{Opt}(i, v_2, v_3) = \text{Opt}(i-1, v_2, v_3)$.

Sada promatrajmo relaciju u rješenju. Tablicu popunjavamo unaprijed, tj. širimo se iz trenutnog u sljedeće stanje. Imamo tri slučaja(trenutno smo na $\text{Opt}(i, v_2, v_3)$), i neka nam je r_i trenutni zahtjev, a r_{i+1} sljedeći zahtjev):

- 1.) na sljedeći zahtjev idemo se pozicije r_i ; tada je moguća vrijednost za $\text{Opt}(i+1, v_2, v_3)$ jednaka:

$$\text{Opt}(i, v_2, v_3) + C(r_i, r_{i+1})$$

- 2.) na sljedeći zahtjev idemo sa pozicije v_2 ; tada je moguća vrijednost za $\text{Opt}(i+1, r_i, v_3)$ jednaka:

$$\text{Opt}(i, v_2, v_3) + C(v_2, r_{i+1})$$

3.) na sljedeći zahtjev idemo se pozicije v_3 ; tada je moguća vrijednost za $Opt(i+1, v_2, r_i)$ jednaka:

$$Opt(i, v_2, v_3) + C(v_3, r_{i+1})$$

Dakle za rješavanje stanja $Opt(i, v_2, v_3)$ trebamo stanja oblika $Opt(i-1, x_2, x_3)$. Pa to možemo pametno iskoristiti za štednju memorije(imamo polje Opt veličine $2*n*n$).

Za rekonstrukciju rješenja pamtimo još jedno polje gdje za svako stanje pamtimo od kud je netko došao riješiti i -ti zahtjev($Move[i, v_2, v_3]$).

Složenost:

Vrijeme: $O(n*m^2)$

Memorija: $O(n*m^2)$

ZADATAK: Multi-key Sorting

Ovo je lagan zadatak. Rješenje je trivijalno i postoje razni načini implementacije istog. Jedino što moramo zaključiti u ovom zadatku je da se redoslijed operacija može nekim algoritmom pojednostaviti. To pojednostavljenje je rješenje zadatka, a ono glasi:

Ako imamo niz operacija $Sort(a) XX Sort(a)$, gdje nam XX predstavlja bilo koji niz sort operacija, tada je to jednako $XX Sort(a)$. To jest pamtit ćemo samo zadnje pojavljivanje operacije $Sort$ nad nekim stupcom.

Zašto je to dobro rješenje:

1.) to je najkraće jer svaku sort operaciju zapisujemo samo jednom.

2.) Dokaz da je $Sort(a) XX Sort(a)$ jednako $XX Sort(a)$:

Promatrajmo neka dva retka a i b .

Ako $a[i] < b[i]$, tada će zadnji $Sort(i)$ staviti a prije b neovisno o prijašnjim operacijama. Isto vrijedi za $a[i] > b[i]$.

Ako $a[i] = b[i]$, tada zadnji $Sort(i)$ nema efekta, ali nema ni prvi $Sort(i)$ efekta.

Dakle prvi $Sort(i)$ ne utječe na rezultat, tj. $Sort(i)XXSort(i) = XXSort(i)$.

Sad nam preostaje implementacija koja i nije neki problem. Možemo jednostavno za svaki broj pamtit kad se zadnji put pojavila $Sort$ operacija sa tim brojem i na kraju sortirati brojeve prema zadnjem pojavljivanju $Sort$ funkcije.

Složenost:

Vrijeme: $O(\text{duljina inputa})$

Memorija: $O(C)$

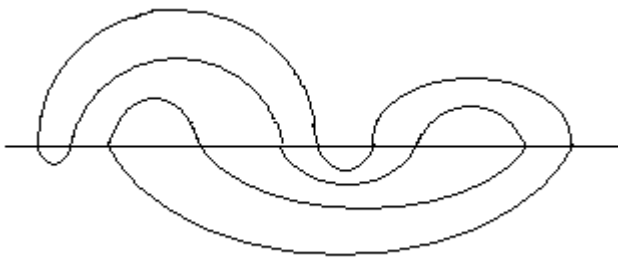
ZADATAK: Electric Fence

Neka je m broj sjecište ograde sa cestom. Indukcijom po m se može pokazati da je ukupan broj različitih regija jednak $m/2+1$. Dakle, potrebno je izračunati broj sjecišta i broj regija koje se nalaze sa lijeve strane ceste. Neka su u i v dva sjecišta takva da na putu od u do v ne postoji ne jedno drugo sjecište. Takav dio ograde se naziva odsječak i u i v su krajevi tog odsječka. Ako se segment ograde koji povezuje stup i i $(i \bmod n + 1)$ sječe sa cestom tada je sjecište prezentirano sa oznakom i . Uoči da su odsječci sa obje strane ugnježdjeni. Pridodajmo oznaku ugnježđenosti svakom odsječku tako da vanjski odsječak(poslije kojeg nema više ni jednog) ima vrijednost 1

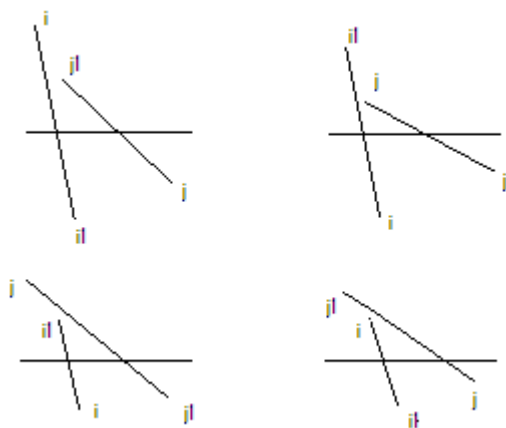
Može se lagano dokazati da je broj regija na lijevoj strani jednak broju odsječak sa lijeve strane koji imaju neparnu vrijednost ugnježđenosti. Jasno je, da ako su p i q dva susjedna odsječka, tada se oni nalaze na suprotnim stranama ceste.

Najprije izračunajmo najljevije sjecište(najbliže kraju ceste a), označimo ga sa leftmost. Krećući se od leftmost, posjetimo stupove ograde u rastućem poretku ako je leftmost na desnoj strani ceste, a inače u padajućem poretku. Odsječak na lijevoj strani je neparan ako i samo ako je njegovo početno sjecište bliže kraju a ceste nego drugom kraju ceste.

Slika 1.



Slika 2.



Situacija kad je sjecište segmenta $(i, i1)$ i ceste bliže kraju ceste a, nego sjecište segmenta $(j, j1)$ i ceste. Ta relacija se može riješiti koristeći samo operaciju Drift.

Složenost:

Vrijeme: $O(N)$

Memorija: $O(N)$

ZADATAK: Critical Network Lines

Promatrajmo graf G čiji su čvorovi mrežni čvorovi, a veze direktne komunikacijske linije. Kritične mrežne linije moraju biti oni mostovi u grafu. Most u grafu je ona veza koju kad maknemo podijelimo graf u dva nepovezana grafa, tj u dvije komponente, recimo da su ti novi grafovi G_u i G_v . Sada je taj most kritična linija ako i samo ako:

$$nA = 0 \text{ ili } nB = 0 \text{ ili } nA = K \text{ ili } nB = L$$

pri čemu je nA broj čvorova u G_v koji nude uslugu A , nB broj čvorova u G_v koji nude uslugu B , K ukupni broj čvorova koji nude uslugu A i L ukupni broj čvorova koji nude uslugu B .

Mostovi se mogu odrediti pretraživanjem u dubinu(DFS). Krenemo iz nekog čvora, i proširimo se na sve njegove susjede. Kad dođemo na neki vrh tada za njega zapišemo dubinu na kojoj se on nalazi. Kad se proširimo na susjeda tada to stanje promatramo kao jedan podgraf i za njega pamtimo najmanju dubinu do koje nam neki čvor iz tog podgraf može doći direktnom linijom.

To rješenje se može implementirati jednim DFS obilaskom kroz graf.

Složenost:

Vrijeme: $O(n+m)$

Memorija: $O(n+m)$

ZADATAK: Ticket Office

Ovaj zadatak možemo riješiti na više načina.

U prvom načinu koristimo dinamičko, a u drugom greedy(pohlepan) algoritam.

Oba algoritma imaju istu vremensku i memorijsku složenost.

1.) Način

Zadatak ćemo riješiti u dva koraka.

Prvi korak:

Pretpostavimo da imamo beskonacno mnogo zahtjeva kojima nije određeno mjesto i cijena im je $C/2$, i drugu vrstu, a to su stvarni zahtjevi sa poznatim mjesto želje i cijena im je C . Sada dinamičkim programiranje popunimo tablicu $1*n$, te na svakom mjestu pamtimo najveću zaradu i poziciju prethodnog ispunjenog zahtjeva.

Relacija je sljedeća $Sol[i] = \text{MAX}(Sol[i-1], Sol[i-k]+ kol[i])$. Gdje nam je k broj karatau svakom zahtjevu, a $kol[i]$ je 2 ako postoji stvarni zahtjev za tu poziciju, a 1 ako je to jedan od onih beskonačno mnogo koji nemaju određenu poziciju.

Time završava prvi korak.

Drugi korak:

Kod prvog koraka nam se može dogoditi jedan problem. Neka nam je Z ukupni broj zahtjeva, a mi smo za najbolje rješenja potrošili više od Z zahtjeva. Da bi našli točno rješenje, radimo sljedeći algoritam: Preko drugog parametra(pozicija prethodnog zahtjeva, ispunjenog) možemo rekonstruirat trenutno rješenje. U tom rješenju znamo točno koji su zahtjevi ispunjeni, tj. dali je ispunjen zahtjev za cijenu C (njih ima $Z1$) ili cijenu $C/2$ (njih ima $Z2$). Da bi dobili pravo rješenje, uzimamo sve zahtjeve sa cijenom C , tada uzimamo još $\min(Z-Z1, Z2)$ zahtjeva sa cijenom $C/2$. Dakle uzimamo sve sa cijenom C u najboljem rješenju i koliko možemo sa polovičnom cijenom.

2.) Način

Prvo postavimo sve zahtjeve koje možemo(Poredak $S1$). To radimo tako da krenemo od kraja i ako trenutni možemo ispuniti, a da se ne preklapa ni s jednim do sada, tada to učinimo. Sada modificiramo taj poredak $S1$ tako da imamo što manje glupo izgubljenog prostora. To radimo tako da zamijenimo neki zahtjev $p1$, sa novim p za koje vrijedi $p < p1$ i $p \bmod K$ je minimalno(gdje je K broj mjesta u zahtjevu). Treći i zdanji korak je da na slobodna mjesta stavimo zahtjeve sa upola manjom cijenom. I to je krajnje i najbolje rješenje.

Složenost:

Vrijeme: $O(n + m)$

Memorija: $O(n + m)$