AGOGE i-training
e-mail: agogeit@gmail.com
datum: 26. maj 2010. godine

AGOGEit: SIO Training

Rešenja problema

Zadatak 01 [TheGraf] Dekompozicija usmerenog grafa na jako povezane komponete je poznat problem i ovde ga nećemo objašnjavati. Lepa osobina ovog tipa dekompozicije jeste da od dobijenih komponenata možemo konstruisati usmeren acikličan graf gde će nam čvorovi biti komponente a dva čvora povezana ukoliko postoji čvorovi iz dati komponenti koji su povezani. Dobije graf je svakako acikličan jer bi u suprotonom (a to je pri postojanju ciklusa) neke komponente ne bi bile maksimalne (mogle bi da se spoje u jednu). Postavlja se pitanje, koliko minimalno ivica treba dodati acikličnom grafu tako da je on jako povezan.

Minimalni uslova je svakako da svaka komponenta mora imati neku ivicu koja ulazi u nju i neku koja izlazi iz nje. Usposatavlja se da je ovo upravo je dovoljan uslova. Drugim rečima minimalni broj potrebnih ivica predstavlja maksimum od broja komponenti čiji je ulazni stepen 0 i broja komponenti čiji je izlazni stepen 0. Naravno, treba obratiti specijalan slučaj kada je sam graf jako povezan - gde je rešenje 0 a ne 1.

Zadatak 02 [ThePut] Nije teško primetiti da prilikom svakog ubacivanja nove tačke u skup P imamo samo dve mogućnosti s kojim tačkama je spojiti. Zaista, da bi na kraju dobili put, u svakom koraku skup P mora biti put, pa kada ubacujemo novu tačku, jedini kandidati za spajanje su krajnje tačke trenutnog puta.

Zadatak rešavamo pomoću dinamičkog programiranja. Označimo početnu tačku sa X, ostale tačke sa $A_0, A_1, \ldots, A_{n-1}$ u redosledu kojim su date i neka je specijalno $A_0 = Y$. Jasno je da X i Y moraju biti spojene. U svakom trenutku (pa i na kraju) put će biti oblika

$$A_{i_1}, A_{i_2}, \ldots A_{i_k}, X, Y, A_{j_1}, A_{j_2}, \ldots, A_{j_l}$$

gde je niz i opadajući, a niz j rastući (jer moramo u datom redosledu ubacivati tačke). Nazovimo deo puta čije su tačke dobijenje spajanjem preko X (niz i) *levim* a preostali deo (niz j) *desnim*. Sada možemo efektivno opisati stanje:

$$\begin{aligned} d[i][j] \quad = \quad & \text{minimalna dužina puta kome je krajnja leva tačka } A_i \text{ a krajnja desna } A_j, \\ & \text{pri čemu je do tada ubačeno prvih } \max(i,j) \text{ tačaka.} \end{aligned}$$

Pre svega, nije teško videti da za svako $i \neq j$ moguće napraviti put kome su ovo krajnje tačke. Neka je $i < j$. Ukoliko je $j \neq i+1$, pretposlednja tačka na desnom delu puta mora biti A_{j-1} jer je u svakom trenutku tačka sa najvećim rednim brojem na nekom od krajeva puta a u ovom slučaju na levom kraju je A_i. Međutim, ukoliko je $j = i+1$, onda nije jasno koja tačka je pretposlednja na desnom delu, pa je potrebno proveriti sve moguće kandidate i uzeti optimalnu vrednost. Odatle sledi

$$d[i][j] = \begin{cases} d[i][j-1] + dist(A_{j-1}, A_j), & \text{ako je } j \neq i+1 \\ \min_{0 \leq k < i}\{d[i][k] + dist(A_k, A_j)\}, & \text{ako je } j = i+1 \end{cases}$$

Analogno važi i za $i > j$. Krajnje resenje je najmanji od brojeva oblika $d[i][n-1]$ ili $d[n-1][i]$. Što se rekurentne veze tiče, za računanje $d[i][j]$ nam je potrebno $O(1)$ vremena, kad god se i i j ne razlikuju za 1, inače nam treba $O(n)$ vremena. Međutim situacije kada se to dešava su samo $(1,2),(2,3),\ldots,(n-2,n-1)$ i analogno za $i > j$ pa njih ima svega $O(n)$, što znči da je ukupna složenost algoritma $O(n^2)$.

Zadatak 03 [TheBroj] U svakom koraku mi moramo pronaći element na datoj poziciji u trenutnom nizu cifara i evenutalno ga obrisati. Ukoliko za to koristimo niz ili listu, jedna od operacija mora će raditi linearno po dužini niza što nije dovoljno brzo. Struktura koja nam to omogućava da ove operacije radimo brzo je *Segment Tree*.

Konstruišemo nad početnim nizom kompletno binarno stablo u kome svaki unutrašnji čvor pamti podatak c - broj neobrisanih cifara na segmentu za koji je on zadužen. Na ovaj način lako dolazimo do k-te pozicije u složenosti $O(\log n)$, spuštanjem niz stablo počevši od korena. Da bi smo u svakom trenutku znali trenutnu vrednost broja, svaki čvor stabla pamti dodatni podatak m - vrednost trenutnog broja u njegovom podsegmentu (kada mu spojimo cifre).

Sada su formule za *update*-ovanje čvorova jasne (V je otac, L i R su levi i desni sin, redom):

$$\begin{aligned} V.c &= L.c + R.c \\ V.m &= (L.m \cdot 10^{R.c} + R.m) \ mod \ 10.007 \end{aligned}$$

Vrednosti 10^i za $i \leq n$ možemo izračunati linearno na početku. Prilikom operacije 1, nalazimo poziciju, postavljamo vrednosti c i m za dati list (cifru) na 0 i *update*-ujemo na gore. Slično radimo i za operaciju 2, s tim što sada zamenjujemo vrednosti i ne diramo podatak c. Obe operacije rade u složenosti $O(\log n)$. Operacija 3 radi u $O(1)$ jer vraćamo samo *koren.m*. Ukupna složenost algoritma je $O(n + m \log n)$.