Opisi algoritama

IBT 2010., 2. kolo, juniori P1

1 Kutije

Optimalno umetanje kutija je ono u kojemu najmanju dimenziju unutarnje kutije pridružujemo najmanjoj dimenziji vanjske i analogno za ostale dimenzije. Dakle, sortirat ćemo dimenzije obiju kutija i provjeriti može li na opisani način prva stati u drugu ili druga u prvu.

2 Umag

Ključni dio zadatka je kako provjeriti odgovara li riječ zadanome uzorku.

Ako je duljina riječi manja ili jednaka duljini uzorka, tada sigurno riječ ne odgovara uzorku. Inače, moramo provjeriti je li dio uzorka prije zvjezdice jednak istom tom dijelu u zadanoj riječi. Ako nije, tada riječ ne odgovara uzorku. Zatim bismo isto to trebali napraviti gledajući od kraja, a to je najjednostavnije tako da obrnemo i uzorak i zadanu riječ, te ponovimo prošli korak. Ako su i sada jednaki početni dijelovi, ispisujemo DA.

3 Milan

Definirajmo *doprinos* neke karte kao broj za koji se ukupna suma (sretni broj) povećava nakon što izvučemo tu kartu. Očito, doprinos karte p jednak je broju p pomnoženom s brojem pojavljivanja broja p na lopticama.

Dovoljno je promatrati samo pozitivne doprinose (tj. one za koje uopće postoje brojevi na lopticama). Njih ćemo izračunati tako da sortiramo brojeve na lopticama - to će dovesti do toga da se svi jednaki brojevi na lopticama nalaze jedan pored drugoga - i onda u jednom prolasku kroz taj, sortirani niz, zbrajajući uzastopne brojeve koji su jednaki, nalazimo sve doprinose koji nas zanimaju i spremamo ih u novi niz.

Konačno, treba naći K najvećih doprinosa. Sortirat ćemo dobiveni niz doprinosa od najvećeg do najmanjeg i uzeti K prvih. Ako je K veći od broja doprinosa, uzimamo sve doprinose. Zbroj uzetih doprinosa rješenje je zadatka.

4 Jane

Za potrebe ovog rješenja, definirajmo proceduru *flood fill*(x) koja pronalazi sve vrhove u grafu do kojih možemo doći iz vrha x; nju je jednostavno implementirati običnim BFS algoritmom.

Promatrajmo graf čiji su vrhovi - parovi stabala. Ako u jednoj sekundi (jednom dvostrukom skoku) možemo doći iz vrha x u vrh y (koji predstavljaju parove stabala), tada između vrhova x i y postoji brid.

Par (J, T) je dobar ako iz njega možemo doći u (T, J) (ne nužno izravno). Dakle, (J, T) je dobar ako se nalazi u istoj komponenti grafa kao i (T, J). Preostaje nam sada za svaki vrh grafa odrediti kojoj komponenti pripada.

To rješavamo tako da, prolazeći redom po vrhovima grafa (parovima stabala), ako trenutni vrh x prije još nismo posjetili, napravimo *flood fill*(x), nalazeći tako sve vrhove koji se nalaze u istoj komponenti kao x, usput ih označivši istim brojem komponente. Nakon ove procedure, za svaki par stabala pogledamo nalazi li se u istoj komponenti kao i njegov obrnuti par.

Opisano rješenje radi u vremenskoj i memorijskoj složenosti $O(N^2)$ i boduje se sa 70% bodova.

Za 100% bodova potrebno je primijetiti da iz svakog dobrog para (J, T) možemo doći u neki par oblika (K, K) ili $(K, K+1)$, jer se Jane i Tarzan u nekom trenutku moraju sresti. Zato je dovoljno učiniti *flood fill* samo iz svih vrhova oblika (K, K) i $(K, K+1)$ - on će pronaći sve dobre parove.

Preostaje problem: gdje ćemo spremati parove koje nađemo, budući da moramo provjeravati jesmo li do nekog para već bili došli. Koristit ćemo u tu svrhu strukturu *set* koja se u C++u nalazi u istoimenoj biblioteci. Ona će nam podržati operacije ubacivanja para u set, provjeravanja nalazi li se neki par već u setu, kao i konačno ispisivanje svih parova koje smo ubacili.

Budući da *set* radi u logaritamskoj složenosti, ukupna složenost opisanog rješenja je $O(Mlog_2M)$, gdje je M broj dobrih parova.

5 Snake

Kretanje zmije možemo promatrati kao da u svakome koraku odlučujemo na koje ćemo se sljedeće polje pomaknuti (gore, dolje, lijevo ili desno). S tim da polje na koje se odlučimo pomaknuti ne smije biti jednako nekom od prethodna četiri polja na kojima smo bili.

Kada smo to uočili možemo napraviti dinamiku tako da je stanje opisano trenutačnom pozicijom i još sa prethodne četiri pozicije, te sa brojem K, koji označava koliko još koraka moramo napraviti. Ovakav algoritam dovodi do prevelikog broja stanja - $O(N^{10} \cdot K)$.

Broj stanja možemo značajno smanjiti tako da umjesto pamćenja prethodne četiri pozicije, pamtimo samo smjerove kojima smo se kretali da bi došli do trenutačne pozicije. Kako postoje četiri smjera, broj stanja bi se smanjio na $O(4^4 \cdot N^2 \cdot K)$. Kada su N i K jednaki 50, tada je broj mogućih stanja $32 \cdot 10^6$. To bi značilo da bi naš program koristio 128 Mb memorije, što je previše za ograničenja dana u zadatku.

Da bismo smanjili potrošnju memorije potrebno je proučiti prijelaz u dinamici.

Funkcija $DP(X, S, K)$ označava koliku najveću sumu zmija može pokupiti ako se trenutačno nalazi na poziciji X, te je do nje došla koristeći skup smjerova S i ako mora napraviti još K koraka. Broj $V(X)$ označava broj bodova na poziciji X. Pozicije X_1, X_2, X_3 i X_4 redom označavaju pozicije dobivene pomicanjem za jedno polje gore, dolje, lijevo ili desno, a na isti način i S_1, S_2, S_3 i S_4 označavaju skupove smjerova kojima smo došli do trenutačne pozicije. Vrijedi relacija:

$$DP(X, S, K) = max \begin{cases} DP(X_1, S_1, K-1) + V(X_1) \\ DP(X_2, S_2, K-1) + V(X_2) \\ DP(X_3, S_3, K-1) + V(X_3) \\ DP(X_4, S_4, K-1) + V(X_4) \end{cases}$$

Ključna stvar koju moramo uočiti je, da bismo izračunali vrijednost za preostalih K koraka, jedino je potrebno znati vrijednosti za preostalih $K-1$ koraka. Tu činjenicu možemo iskoristiti tako da tokom računanja u dinamici ne pamtimo u memoriji sva stanja, već samo prethodni redak (za prethodni K), odnosno, broj stanja u memoriji se smanji na $O(4^4 \cdot N^2)$, što je dovoljno malo memorije, oko 2.5 Mb.

Konačno memorijska složenost ovog rješenja je $O(4^4 \cdot N^2)$, a vremenska $O(4^4 \cdot K \cdot N^2)$.

Komentare i pitanja uputite na frane.kurtovic@gmail.com *ili na* askurdija@gmail.com.