Opisi algoritama

IBT 2011. MASTERS LIGA – 4. kolo

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

Monitor (P1-1)

Da bismo provjerili odgovaraju li stranice nekom omjeru *a:b*, potrebno je ili tretirati stranice kao razlomak pa ga skratiti do kraja, ili možemo for petljom proći po svim mogućim koeficijentima koji množe omjer *a:b* te zaustaviti petlju ako *k*a* ili *k*b* postanu veći ili jednaki zadanim stranicama. Ako su jednaki učitanim stranicama tada taj omjer odgovara omjeru stranica, te ga ispišemo.

Tele (P1-2, P2-1)

Primijetimo da se uvijek isplati skočiti sa prvog teleporta lijevo od zida, te kada nas taj teleport odvede do sljedećeg, tada pronađemo prvi zid desno, i opet skočimo sa prvog teleporta lijevo. Te ispišemo broj skakanja teleportom.

Kugle (P2-2)

Algoritam je simulacija opisanog načina sudaranja kugli u zadatku. Učitamo podatke o masama kugli, te pamtimo dvije varijable koje označavaju sljedeću Gandalfovu i Sarumanovu kuglu. U još jednoj varijabli pamtimo masu najnovije kugle (koja je možda nastala sudaranjem dvaju kugli), i čija je, odnosno s koje strane dolazi. Ako je masa preostale kugle veća od nule tada je pobijedio onaj čija je ta kugla, i ispišemo sumu svih preostalih kugli.

Zamjena (P1-3)

Ovo je klasičan primjer dinamičkog programiranja, tj. rekurzije s memoizacijom.
Primijetite da kad bi zadatak bio pretvoriti niz u 1-alternirajući da bi tada postojala samo dva moguća konačna niza, onaj koji počinje slovom 'A', odnosno slovom 'B' (prvim elementom su definirani svi ostali).
Rekurzija redom gradi niz, odnosno proba staviti na svako mjesto slovo 'A' ili slovo 'B'. U nekom trenutku rekurziji nije bitan izgled cijelog niza koji je ona do sada napravila, već samo pozicija elementa kojeg pokušava odrediti i broj alternacija koje mora ostvariti do kraja niza. Prijelaz na sljedeće stanje je znači isprobavanje stavljanja svakog slova na tu poziciju, stoga je složenost algoritma **O (N*K)**.

Skoro (P1-4, P2-3)

Pretpostavimo da smo uspjeli generirati niz za neki **K,** tj. imamo niz od 2^K domina, nazovimo taj niz ***Q***. Sada želimo od tog niza stvoriti niz od domina veličine *K+1*.
Ako na početak svake domine niza *Q* stavimo 0, tada ćemo taj niz zvati *0Q*, a ako stavimo 1 tada ćemo taj niz zvati *1Q*. Primijetite da je svaki od ta dva niza sastavljen od domina veličine *K+1*, i da je svaki od nizova „pravilan“, tj. svake dvije susjedne domine se razlikuju na točno jednoj poziciji.
To znači da bismo mogli ta dva niza domina spojiti, te bi oni također bili pravilni, jedini problem bi mogao stvoriti prijelaz između niza *0Q* i *1Q*, jer se te dvije domine mogu razlikovati u više od jedne pozicije, stoga niz 1Q napišemo obrnutim redoslijedom. To znači da će zadnjih K znakova u te dvije prijelazne domine biti jednaki, odnosno samo će se razlikovati u *K+1* znaku.

Simuliranjem ovog postupka možemo generirati niz domina bilo koje duljine, pažljivo generiranje niza dovodi do složenosti ***O (K*2^K)***.

Jagodica (P1-5, P2-4)

Zadatak se rješava dinamičkim programiranjem. Zanima nas koliko je moguće jagoda skupiti ako se nalazimo na nekoj jagodi, i krećemo samo gore ili desno. Za neku jagodu možemo izračunati taj podatak tako da pregledamo sve jagode u pravokutniku kojemu je naša pozicija donji lijevi rub, a gornji desni mu je kraj vrta. Rješenje je maksimalna od tih vrijednosti + 1.

Ako obilazimo jagode tako da prvo prođemo po jagodama s većom y-koordinatom, a ako ih nekoliko dijeli mjesto, tada obradimo prvo one s većom x-koordinatom, da će nam to garantirati da možemo izračunati vrijednost za trenutačnu jagodu, pomoću već izračunatih vrijednosti.

Ovakav algoritam bi nas doveo do složenosti ***O (N^2)***.

Od već izračunatih vrijednosti nas samo zanima koja je najveća, a da joj je x-koordinata veća ili jednaka trenutačnoj x-koordinati. To možemo postići tako da u logaritamsku strukturu ili tourament stablo, ubacujemo vrijednosti na poziciji x-koordinate jagode, te nam upit u strukturu podataka bude koji je najveći broj u prvih ili zadnjih *x* elemenata.

Složenost ovog algoritma je ***O (N log N)***. Bitno je još i sažeti koordinate po apscisi, tako da ih možemo stavljati u tournament stablo ili logaritamsku strukturu.

Inače logaritmska struktura ne podržava upit koji je najveći od x-tog elementa na dalje, ali zato možemo obrnutim redoslijedom obrađivati podatke, tako da pitamo koji je najveći element u prvih x. Još je bitno da nikada ne smanjujemo vrijednost na nekoj poziciji u logaritamskoj strukturi, jer inače neće raditi, a iz ovog algoritma uvijek proizlazi da će nova vrijednost biti veća ili jednaka sadašnjoj.

Spust (P2-5)

Učitane krajnje točke intervala spremimo u niz i sortiramo. Svaka dva susjedna člana u tome nizu čine interval, unutar kojeg nema nikojih drugih točaka intervala, što znači da učitani intervali ili u potpunosti sadrže taj interval ili uopće nemaju presjek s njim.

Označimo prvi interval s **D**.

Od svih tih intervala su nam samo zanimljivi oni koji su podskup intervala **D**, nazovimo neki takav interval sa **S**. Vjerojatnost da je prvi igrač imao vrijeme unutar intervala **S** je S/D (ovo su duljine svakog intervala). Još nam je i bitna vjerojatnost da je prvi igrač pobijedio sve ostale ako se našao u **S**. Ta vjerojatnost pomnožena sa već izračunatom vjerojatnošću da se prvi igrač našao unutar **S**, je vjerojatnost da su se oba dva događaja dogodila. Ako sumiramo sve te vjerojatnosti za svaki **S**, dobit ćemo traženu vrijednost da je prvi igrač pobijedio.

Računanje vjerojatnosti da je prvi igrač pobijedio sve ostale ako se našao u S

Za svaki učitani interval **T** vrijedi da je **S** u potpunosti unutar njega ili u potpunosti izvan. Ako je **S** u potpunosti izvan i ispred njega, tada taj interval ne utječe na ukupnu vjerojatnost, a ako je iza njega tada je vjerojatnost **0**.

Teži slučaj je ako se **S** nalazi unutar njega. Tada promatramo koja je vjerojatnost da je igrač stigao u intervalu [S.a, T.b], to je (T.b-S.a)/(T.b-T.a). Od tog preostalog dijela nam je bitno je li igrač stigao unutar **S** ili nakon **S**, tj. možemo odabrati sa određenom vjerojatnošću u koji dio je stigao. Kada smo odabrali za svakog igrača u koji je dio stigao, i ako je **K** igrača stiglo unutar **S**, tada je vjerojatnost da je prvi igrač pobijedio jednaka *1/K*, naravno taj **K** mora uključivati i našeg igrača.

Ovo možemo riješiti pomoću rekurzije s memoizacijom tako da nam je stanje na kojem se intervalu trenutačno nalazimo, te za koliko smo do sada igrača odredili da su unutar **S**. Prijelaz je određivanje u koji dio intervala je određeni igrač stigao na cilj.

Detalje o implementaciji pogledajte u službenom rješenju.