

Zadatke, test primjere i rješenja pripremili: Goran Gašić, Frane Kurtović, Gustav Matula, Ivo Sluganović i Goran Žužić i Ante Đerek. Primjeri implementiranih rješenja su dani u priloženim izvornim kodovima koji nužno ne odgovaraju u svim detaljima ovdje opisanim algoritmima.

UNIX

Predložio: Ante Đerek

Potrebno znanje: dinamičke strukture podataka, polja, stringovi, osnovne petlje, simulacija

U ovom implementacijskom zadatku potrebno je samo točno simulirati postupak opisan u tekstu. Vjerojatno najlakši pristup se postiže korištenjem dinamičkih struktura podataka - hijerarhija direktorija je stablo u kojemu svaki čvor odgovara jednom direktoriju te sadrži pokazivače na direktorije koje direktno sadrži te na direktorij u kojemu je direktno sadržan. Međutim, ovdje ćemo opisati rješenje koje ne koristi dinamičke strukture podataka nego samo jednodimenzionalna polja.

Root direktorij ćemo označiti brojem 0, a ostale direktorije ćemo prilikom stvaranja označavati redom brojevima 1, 2, 3, itd. Za svaki direktorij pamtimo sljedeće vrijednosti: Brojevenu oznaku direktorija u kojemu je direktorij direktno sadržan, ime direktorija, te da li je direktorij izbrisan ili ne.

```
int parent[MAX];  
string name[MAX];  
int deleted[MAX];
```

Dodatno, u svakom trenutku pamtimo indeks trenutnog direktorija i ukupni broj direktorija napravljenih do sada. Sve zadane operacije se na jednostavan način daju implementirati kao operacije na ova tri polja. Na primjer, naredba 'rmdir X' se radi na sljedeći način:

1. Najprije prolaskom kroz polja tražimo indeks direktorija X kojeg treba izbrisati, njega ćemo prepoznati lako, to je jedinstveni direktorij koji nije izbrisan, ima ime X i roditelj mu je upravo trenutni direktorij (!deleted[i] && name[i] == X && parent[i] == current)
2. Ako nismo našli takav direktorij ispisujemo 'greska'.
3. Provjeravamo da li je direktorij prazan tako da probamo naći bilo koji neizbrisani direktorij koji je sadržan u X (opet još jednim prolaskom kroz polja).
4. Ako direktorij nije prazan ispisujemo 'greska'.
5. Brišemo direktorij tako da samo postavimo vrijednost polja deleted na istinu.
6. Putanju direktorija računamo tako da u while petlji spajamo imena direktorija dok ne dođemo do root direktorija.

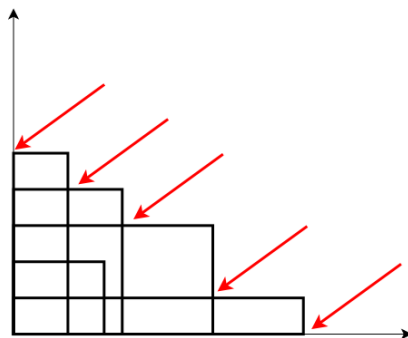
Ostale naredbe se implementiraju na analogan način.

DIREKTORICA

Predložio: Goran Žužić

Potrebno znanje: polja, brzi algoritmi sortiranja

Potrebno je pronaći nenegativne cijele brojeve x , y tako da minimiziramo njihovu sumu, a ujedno ispunimo N uvjeta oblika ($x \geq X_i$ ili $y \geq Y_i$). Ukoliko se svi projekti (X_i, Y_i) vizualiziraju kao točke u ravnini, dobivamo zadatak sljedećeg oblika: točka (x, y) se ne smije nalaziti unutar niti jednog pravokutnika određenog s ishodištem i i točkom (X_i, Y_i) . Drugim riječima, moramo pronaći točku (x, y) s minimalnom sumom $x+y$ takvu da se nalazi na rubu unije pravokutnika čija je donja lijeva točka ishodište. Nije teško uočiti kako će geometrijski ta unija imati oblik "stepenica" kao na slici niže.



Uzimajući u obzir gornju intuiciju, potrebno je samo odrediti koji pravokutnici (odnosno projekti) čine rub tih stepenica te potom isprobati sve kandidatne rubne točke kao što je prikazano na slici crvenim strelicama. Ukupna složenost samog algoritma je $O(N \log N)$ jer je potrebno sortirati početne projekte kako bismo odredili geometriju stepenica.

MEDIJAN

Predložili: Ivan Mandura i Ante Đerek

Potrebno znanje: Fenwick tree (logaritamska struktura)

Za određivanje je li medijan niza veći ili jednak K nisu bitne točne vrijednosti elemenata, već samo jesu li veći ili jednaki K . Zato možemo niz zapisati tako da brojeve manje od K zamijenimo s -1 , a veće ili jednake K s 1 . Primijetite da ako takav niz ima pozitivnu sumu da to upravo znači da mu je i medijan barem K . Preostali dio problema je efikasno prebrojati koliko ima takvih uzastopnih podnizova.

Proći ćemo po nizu od početka do kraja i pritom pratiti vrijednosti suma svih intervala koji završavaju na trenutnoj poziciji. Preciznije, ako smo sad na elementu i , tada znamo sume intervala $[i, i]$, $[i - 1, i]$, ..., $[1, i]$, $[0, i]$, i želimo moći efikasno provjeriti koliko je takvih suma pozitivno, za što ćemo iskoristiti odgovarajuću strukturu podataka. Primijetite da proširivanje s i na $i + 1$ odgovara dodavanju nule u strukturu te povećavanju svih elemenata u strukturi za $A[i + 1]$.

Kako bismo ostvarili navedeno potrebna nam je struktura podataka koja podržava sljedeće operacije:

- `ubaci_nulu()` -> ubacuje broj 0 u strukturu
- `povečaj_sve(x)` -> povećava sve brojeve u strukturi za x
- `prebroji_pozitivne()` -> vraća broj pozitivnih brojeva u strukturi

Operacija `povečaj_sve` sugerira korištenje pomoćnog brojača s kojeg povećavamo umjesto povećavanja svih brojeva u strukturi. Kako bi to funkcioniralo, u strukturu umjesto 0 ubacujemo $-s$ (jer je $-s + s = 0$).

Uz ovaj brojač potrebna nam je struktura s nešto klasičnijim operacijama:

- `ubaci_broj(x)` -> ubacuje broj x u strukturu
- `prebroji_veće(x)` -> vraća broj brojeva u strukturi većih od x

Kao što smo rekli,

- `ubaci_nulu()` se sada implementira kao `ubaci_broj(-s)`.
- `povečaj_sve(x)` se implementira kao `s += x`.
- `prebroji_pozitivne()` se implementira kao `prebroji_veće(-s)`.

Za niz duljine n složenost ovakve obrade je $O(n \log n)$ pod uvjetom da operacije nad strukturom imaju složenost $O(\log n)$.

Operacije `ubaci_broj(x)` i `prebroji_veće(x)` u složenosti $O(\log n)$ podržava svako balansirano binarno stablo, ali i implementacijski puno jednostavnija logaritamska struktura (koja zapravo po upitu ima složenost $O(\log V)$, gdje je V maksimalna vrijednost broja u strukturi, što je još uvijek dovoljno efikasno). Za detalje pogledajte izvorne kodove službenih rješenja.

LINUX

Predložio: Ante Đerek

Potrebno znanje: dinamičke strukture podataka, polja, stringovi, simulacija, obilazak stabala

Ovaj zadatak se razlikuje od zadatka 'Unix' samo po tome što sadrži naredbu find pa se stoga u ovom opisu oslanjamo na opis rješenja zadatka 'Unix'. Naredba 'find X' treba na neki načini obići stablo svih poddirektorija sadržanih direktno ili indirektno u X te ispisati sve njihove putanje. Stablo se može obići bilo kojim standardnim algoritmom obilaženja grafova, ovdje opisujemo pretraživanje u dubinu (BFS).

1. U prvom koraku pronađemo brojnu oznaku direktorija X postupkom opisanim kod zadatka 'Unix' te ispisujemo 'greska' ako u trenutnom direktoriju ne postoji direktorij 'X'.
2. U strukturu podataka red (queue) stavljamo oznaku direktorija X, string za ispis postavljamo na vrijednost putanje od X
3. Dok naš red nije prazan ponavljamo sljedeće
 - a. Izvadimo jedan element iz reda, označimo ga sa Y.
 - b. Pronađemo sve neizbrisane direktorije koji su direktno sadržani u Y, dodamo njihove putanje u string za ispis i sve njih ubacimo u red.

Obzirom da u hijerarhiji direktorija nema ciklusa nije potrebno paziti da neki direktorij ne posjetimo više od jednom.

DELETE

Predložio: Ante Đerek

Potrebno znanje: stringovi, rekurzivno pretraživanje (backtracking), jednostavno dinamičko programiranje

Očiti pristup je na temelju prve riječi A generirati sve regularne izraze takve da ih prva riječ zadovoljava. Za svaki stvoreni izraz je potrebno ispitati zadovoljava li ga i ostalih N riječi. Ako niti jedna od preostalih N riječi ne zadovoljava stvoreni izraz tada je on jedno od mogućih rješenja, te je samo potrebno ispisati najkraće.

Generiranje regularnih izraza se provodi rekurzivno tako da su parametri rekurzije indeks u riječi A, te trenutno izgrađeni regularni izraz koji zadovoljava prefiks riječi A do trenutnog indeksa (isključivo). Uvijek postoje dvije mogućnosti za odabir novog znaka u regularnom izrazu, slovo na trenutnom indeksu u riječi A ili zvjezdica. Nepotrebno je generirati regularne izraze koji zamjenjuju zvjezdicu samo s jednim slovom jer je taj izraz nepotrebno općenitiji, a ima jednak broj znakova kao i da je umjesto zvjezdice napisano slovo na tom indeksu. Također je nepotrebno stvarati izraze koji imaju dvije zvjezdice za redom jer je to ekvivalentno s izrazom koji ima samo jednu zvjezdicu, a kraći je.

Provjeru odgovara li neka riječ regularnom izrazu je moguće napraviti dinamičkim programiranjem, ali i pohlepnim algoritmom. Pretpostavimo da regularni izraz započinje i završava sa zvjezdicom, npr. '*abc*d*ab*'. Dovoljno je izraz pretvoriti u listu riječi nastalu odvajanjem po zvjezdicama. Za zadani izraz se dobiva 'abc', 'd' i 'ab'. Dovoljno je na riječi koju provjeravamo pokušati postaviti riječi iz niza što ranije moguće, npr. 'ef abc efabc d ghb ab aa'. Ako algoritam ne uspije postaviti sve riječi to znači da riječ ne odgovara zadanom regularnom izrazu. Postoje i mogućnosti da regularni izraz ne započinje ili ne završava sa zvjezdicom, ali to se vrlo lako riješi tako da se slova prije prve zvjezdice i nakon zadnje zvjezdice odmah probaju postaviti na početak i kraj riječi, te se time opet dobiva regularni izraz koji započinje i završava zvjezdicom. Složenost ovog postupka je $O(N \cdot M)$, gdje su N i M duljine regularnog izraza i riječi.

Za one koji žele više

Rekurzija generira regularne izraze, te provjerava je li taj izraz rješenje. Kada rekurzija pronađe neko rješenje, može ga spremiti te u budućnosti prekinuti generiranje trenutnog izraza nakon što izraz dosegne duljinu trenutno najboljeg rješenja jer u tom trenutku više ne može stvoriti bolje rješenje. Ova optimizacija znatno smanjuje broj generiranih regularnih izraza jer reže grane rekurzije, tj. svako novo rješenje smanjuje maksimalnu dubinu rekurzije. Na primjerima iz zadatka se pokazalo da ubrzava pronalaženje rješenja za nekoliko desetaka puta.

Za one koji žele još više

Koliko ima regularnih izraza koji odgovaraju zadanoj riječi a svaka zvjezdica zamjenjuje barem dva slova? Nađi rekurzivnu i zatvorenu formulu.

Za zadana ograničenja za N i M koliko najviše može biti zvjezdica u optimalnom rješenju?

Napravi rješenje koje uvijek radi ispod jedne sekunde ako su sve riječi duljine najviše 40 znakova, a ima najviše 100 riječi koje se ne smiju izbrisati.

SREDINA

Predložili: Ivan Mandura i Ante Đerek

Potrebno znanje: Fenwick tree (logaritamska struktura)

Primijetimo da je medijan nekog skupa barem K ako i samo ako je više od pola elemenata tog skupa veće ili jednako K.

Konstruirajmo tablicu $P[i, j] = +1$ ako je pripadni element originalne tablice veći ili jednak K, te -1 ako je manji od K. Problem se svodi na prebrojavanje pravokutnika s pozitivnom sumom, jer upravo ti pravokutnici imaju medijan barem K.

Kako je ograničenje na broj stupaca manje nego ograničenje na broj redaka, fiksirat ćemo na sve načine po dva stupca (c_1 i c_2) i napraviti niz $A[i] = \sum_{c=c_1}^{c_2} P[i, c]$.

Problem je sada smanjen na jednu dimenziju, tj. potrebno je prebrojati uzastopne podnizove s pozitivnom sumom. Rješenje tog problema je detaljno opisano u zadatku Medijan. Kako je n u našem slučaju jednako R, i imamo C^2 nizova, ukupna složenost algoritma je $O(C^2 R \log R)$.

SLAVICA

Predložio: Goran Žužić

Potrebno znanje: stringovi, pohlepni algoritam

Primjera radi, fiksirajmo $A = \text{"oskar"}$ i $B = \text{"rosa"}$. Originalni zadatak je lako svodljiv na sljedeći: u beskonačnom stringu $AAA\dots = \text{"oskaroskarosk\dots"}$ pronađi (ne nužno uzastopni) podniz "rosa" tako da je zadnji iskorišteni znak minimalan. Lako je uočiti kako je uvijek optimalno uzeti prvo pojavljivanje slova "r", potom prvo sljedeće pojavljivanje slova "o", itd. kako bismo došli do optimalnog rješenja.

Takav pristup (gdje ciklički tražimo prvo pojavljivanje slova B_1 , potom B_2 , ..., $B_{|B|}$) u svom naivnom obliku ima najgoru složenost $O(|A| |B|)$ i donosi 70% bodova. Kako bismo dobili sve bodove potrebno je ubrzati potragu za sljedećim slovom niza B u cikličkom nizu A. To možemo napraviti tako da unaprijed za svaku poziciju unutar A i svako slovo c izračunamo prvo sljedeće pojavljivanje i spremimo ih u tablicu veličine $26 * |A|$. S gornjom optimizacijom tu složenost svodimo na $O(|A| + |B|)$.

ŽABA

Predložio: Gustav Matula i Ante Đerek

Potrebno znanje: binarno pretraživanje, brzi algoritmi za sortiranje

Gornja i donja traka se rješavaju zasebno. Da bi žaba prešla cestu, ona mora prijeći donju traku krećući u trenutku (x, t) , ali i gornju traku krećući u trenutku $(x, t+3)$. Za početak prelaska gornje trake smatramo da se žaba nalazi između dviju traka.

Opisat ćemo samo postupak rješavanja donje trake jer je za gornju traku postupak analogan. Ideja je kamione držati statičnima, tj. promatrati ih u trenutku 0, ali zato x koordinatu žabe smanjiti za t . To je moguće napraviti jer svi kamioni u istoj traci imaju isti smjer i brzinu, zbog čega se njihove relativne pozicije ne mijenjaju u vremenu. Kamione sada promatramo kao intervale opisane lijevom i desnom granicom. Odgovor na pitanje može li žaba preći donju traku se svodi na ispitivanje postoji li neki interval koji sadrži točku $x - t$. Primijetite da je potrebno desnu granicu intervala povećati za tri jer nije dovoljno da u trenutku 0 žabu kamion ne pogazi, već je potrebno da ju ne pokazi niti nakon 3 sekunde koliko joj treba da pređe traku.

Jedan od načina za efikasno provjeriti nalazi li se točka unutar nekog intervala je sortirati intervale (nije bitno po kojem rubu jer se ne sijeku) te binarnim pretraživanjem po intervalima pronaći prvi interval koji je desno od tražene točke. Na taj način je potrebno samo provjeriti sadrži li prvi interval lijevo od njega traženu točku. Moguće rješenje je učitati sve kamione i sve žabe, zajedno ih sortirati te u jednom prolasku odrediti koje točke se nalaze u kojim intervalima. To rješenje ne zahtijeva korištenje binarnog pretraživanja. Oba rješenja imaju vremensku složenost $O((N + Z) \log N)$, gdje je N broj kamiona u traci, a Z broj žaba.

ZMIJA

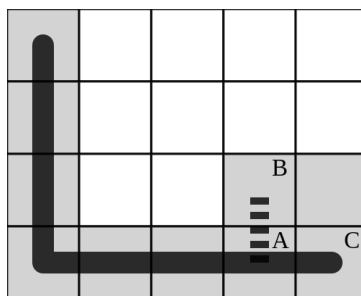
Predložio: Ante Đerek

Potrebno znanje: pohlepni algoritam

Opis algoritma je dosta dug i razmatra puno slučajeva, međutim za uspješno rješavanje zadatka nisu bile potrebne ovako detaljne analize već samo sljedeća intuicija: Zmija se može sastojati od najviše dvije spirale, gdje je spirala put koji skreće uvijek u istom smjeru te se sastoji od sve kraćih i kraćih dijelova. Jedna će spirala počinjati repom a druga glavom zmijske. Svaka spirala se pojedinačno može pronaći pohlepnim algoritmom, a te dvije spirale se mogu odvojiti jedna od druge horizontalnom ili vertikalnom linijom. Stoga je rješenje zadatka sljedeće: Na sve moguće načine rastavi zmijsku horizontalnom ili vertikalnom linijom na dva dijela te probaj svaki dio riješiti pohlepnim algoritmom.

Promatrajmo najprije slučaj kada se rep zmijske nalazi u gornjem lijevom kutu kvadratne mreže. Tvrdimo da se u ovom slučaju tijelo zmijske može naći sljedećim pohlepnim algoritmom (drugim riječima tijelo se sastoji od samo jedne spirale).

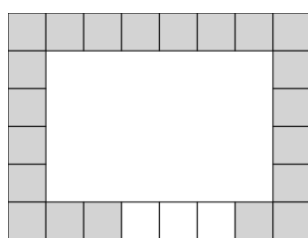
1. Krećemo iz repa (koji se nalazi u gornjem lijevom kutu) okrenuti prema dolje.
2. Idemo ravno koliko god možemo.
3. Ako ne možemo ići ravno skrenemo lijevo i vraćamo se na korak 2.



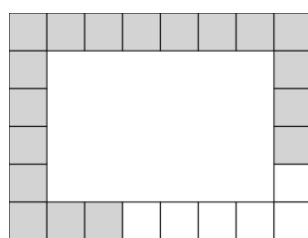
Dokažimo da ovaj pohlepni algoritam radi ispravno u ovom slučaju. Pretpostavimo da pohlepni algoritam daje put koji ćemo označiti s X, a da tijelo zmiје zapravo leži na putu Y koji je različit od puta X. Naravno, X i Y se moraju podudarati u prvoj točki - repu zmiје. Neka je A prvo polje nakon kojega se X i Y razlikuju - drugim riječima X i Y se podudaraju sve do točke A, a u polju neposredno nakon toga se razlikuju. Obzirom da je X konstruiran pohlepnim algoritmom mora biti da X nakon A ide ravno do sljedeće točke C dok Y u točki A skreće lijevo do točke B kao na slici gore. Međutim, lako se vidi da put Y nikako više ne može doći do točke C. Obzirom da je samo dozvoljeno skretanje u lijevo i nije dozvoljeno presijecanje, put Y nikako ne može doći do točke C bez da izađe iz zadane kvadratne mreže. Došli smo do kontradikcije sa činjenicom da postoji ispravan put različit od puta X što dokazuje da X mora biti ispravan put jer je garantirano da za svaki test podatak postoji rješenje.

Općenito, pohlepni algoritam ne radi ispravno u slučajevima kada se rep zmiје ne nalazi u gornjem lijevom kutu mreže ali se zadatak može riješiti tako da se zmiја na pametni način podjeli na dva djela i da se svaki pojedini dio riješi pohlepnim algoritmom.

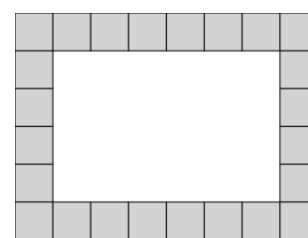
Promatramo takozvani rubni pravokutnik (*bounding box*) - minimalni pravokutnik koji sadrži tijelo zmiје i koji je ograničen redcima koji sadrže najgornji i najdonji kvadratić tijela te stupcima koji sadrže najlijeviji i najdesniji kvadratić tijela. Ako samo promatramo rub tog pravokutnika jasno je da svaka od četiri strane ruba mora sadržavati barem jedan kvadratić tijela. Također, može se dokazati sličnom metodom kao za gornji pohlepni algoritam da svi zauzeti kvadratići na rubu moraju biti povezani - nije moguće da zmiја napusti rub pravokutnika pa se na njega opet vrati. Dakle rubni pravokutnik može biti ili potpuno popunjen (slučaj 3 dolje) ili se svi prazni kvadratići nalaze unutar samo jedne strane (slučaj 1 dolje), ili se svi prazni kvadratići nalaze unutar dvije susjedne strane i sadrže jedan kut (slučaj 2 dolje). Sva tri slučaja ilustrirana su na slici niže.



1



2

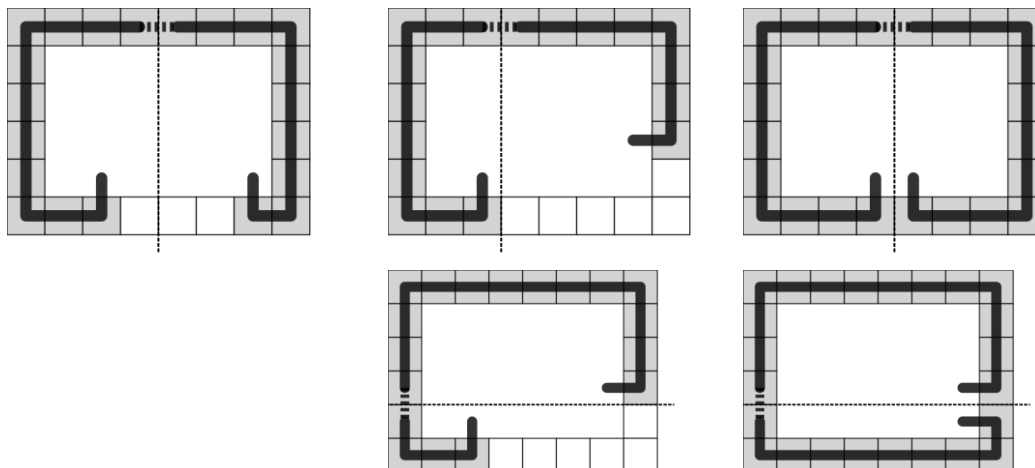


3

U prvom slučaju lako se vidi da se tijelo zmiје može pravcem razdvojiti na dva tijela - dio zmiје kada se napusti rub se nikako ne može poklopiti s dijelom zmiје prije nego što se dođe na rub. Dakle možemo povući pravac i time razbiti zmiју na dva tijela, na rubu možemo napraviti novu glavu odnosno novi rep zmiје te dobivamo dva slučaja u kojima se glava odnosno rep nalazi u uglu polja. Oba dva slučaja rješavamo pohlepnim algoritmom s tim da u jednom od njih uvijek idemo udesno umjesto ulijevo.

U drugom slučaju situacija je slična ali je potrebno razmatrati dvije mogućnosti, ili postoji horizontalni pravac koji razdvaja dva dijela zmiје ili postoji vertikalni pravac s tim svojstvom.

U trećem slučaju, kada je rubni pravokutnik potpuno popunjen, ne znamo gdje je točno zmiја došla na rub pravokutnika i gdje ga je točno napustila ali možemo zaključiti da je na nekom kvadratiću došla te ga je napustila na susjednom kvadratiću nakon što je opisala puni krug po rubu. U ovom slučaju moramo isprobati sve moguće načine da se tijelo zmiје podijeli na dvije spirala horizontalnim ili vertikalnim pravcem.



Iako je opis rješenja dugačak, vidimo da treći slučaj pokriva u potpunosti i prethodna dva pa se cjelokupni algoritam može sažeti na sljedeći način:

1. Za svaki mogući horizontalni i vertikalni pravac
2. Podijeli tijelo zmiје na dva dijela s tim pravcem
 - a. Probaj svaki od dva dijela riješiti pohlepnim algoritmom tako da se u jednom skreće ulijevo u drugom udesno i na kraju se glava iz jednog dijela nađe negdje na rubnom pravokutniku susjedna repu iz drugog dijela.
 - b. Ako su oba dva dijela uspješno riješena, unija dva puta daje rješenje za cijelu zmiју.

Za one koji žele više

Matematički dokaži sve tvrdnje iz teksta.

Koje je najbolje rješenje ako zmiја može skretati i ulijevo i udesno (ali se još uvijek ne smije presijecati)?

IGRA

Predložio: Ivo Sluganović

Potrebno znanje: simulacija, matrice

Zadatak je nastao prema igri 2048 koja se nedavno pojavila na Internetu (<http://gabrielecirulli.github.io/2048/>) i vrlo brzo postala iznimno popularna zbog svojih jednostavnih pravila i velike zanimljivosti.

U zadatku se od igrača traži da implementira igru koju je zatim moguće simuliranjem utipkavanjem znakova koji označavaju sva četiri poteza.

Kako vremenska ograničenja ne predstavljaju problem, važno je fokusirati se na što jednostavniju i čistiju implementaciju u zadatku opisanih koraka.

Kao primjer takvog pristupa, u službenom rješenju implementiran je samo pomak ulijevo, dok se za ostala tri poteza prvo ploča zarotira za potreban broj okreta, a zatim nakon izvođenja pomaka vrati u početnu orijentaciju.

MNOŽENJE

Predložio: Goran Žužić

Potrebno znanje: binarno pretraživanje

Potrebno je izračunati K-ti broj po veličini u opisanoj tablici. Tablica je naravno prevelikih dimenzija da bismo ju mogli zapisati u memoriju računala, stoga je potreban drugi pristup. Puno je lakše odgovoriti na pitanje koliko je brojeva u tablici manjih ili jednakih X. Dovoljan je jedan prolazak po redcima tablice i računanje koliko je brojeva u i-tom retku manje ili jednako X, to je X / i (cjelobrojno dijeljenje).

Rješenje, tj. K-ti broj u tablici možemo pronaći binarnim pretraživanjem. Ako brojeva manjih ili jednakih X u tablici ima manje od K tada donju granicu postavljamo na $X+1$, inače gornju na X.

Složenost ovog rješenja je $O(N \log N)$.

Za one koji žele više

Zadatak je moguće riješiti u vremenskoj složenosti $O(\log^2 N)$ ukoliko ubrzamo unutarnju petlju binarnog pretraživanja. Upućujemo zainteresirane da pogledaju rješenje zadatka Aladin koji se pojavio u 1. kolu HONI-ja 2009/2010.

FIRMA

Predložio: Gustav Matula

Potrebno znanje: sweep-line metoda

Na dan $d + 1$ radnik i je na poslu u intervalu $[T_i \pm d, T_i \pm d + M]$, gdje predznak $+$ uzimamo ako je kasnioc, a $-$ ako je ranioc. Problem je dakle pronaći maksimalni presjek intervala koji se diskretno pomiču lijevo ili desno (maksimalnim presjekom ovdje smatramo maksimalni broj intervala koji obuhvaćaju neku točku).

Naivno rješenje za svaki mogući d iz intervala $[0, D - 1]$ računa maksimalni presjek, što je klasičan problem rješiv u $O(N \log N)$ (gdje je N broj intervala) sweep-line algoritmom u kojem su događaji nailasci na rubove intervala, a struktura podataka obični brojač koji prati koliko intervala linija trenutno siječe. Ukupna složenost rješenja je $O(DN \log N)$, gdje je D broj dana, a N broj radnika.

Korisno je primijetiti kako se intervali koji idu u istom smjeru međusobno relativno ne miču, pa ih možemo promatrati kao dvije zasebne grupe (nazovimo grupu koja ide lijevo *lijevom*, a grupu koja ide desno *desnom*). Također vidimo da se grupe jedna u odnosu na drugu kreću korakom 2, što nam govori da se točke koje su u početku (na dan 1) na koordinatama različitih parnosti nikada neće susresti.

Zbog toga točke koje razmatramo možemo podijeliti na parne i neparne, te problem riješiti za svaku skupinu posebno, i na kraju uzeti maksimum dva rješenja.

Nadalje, ako imamo točku T_i iz grupe koja se kreće desno, i točku T_j iz grupe koja se kreće lijevo. Pretpostaviti ćemo da se prvog dana T_i nalazi na koordinati x_i , a T_j na koordinati x_j , da vrijedi $x_i \leq x_j$, te da su x_i i x_j iste parnosti. Kako će nakon D dana T_i biti na $x_i + D - 1$, a T_j na $x_j - D + 1$, da bi se susrele mora vrijediti $x_i + D - 1 \geq x_j - D + 1$, odnosno $x_j - x_i \leq 2(D - 1)$. Zbog toga će nas zanimati samo točke koje su inicijalno na udaljenosti manjoj ili jednakoj $2(D - 1)$. Dakle tražimo par točaka na dozvoljenoj udaljenosti koji ima maksimalni ukupni presjek (ukupni presjek je suma presjeka *desnih* intervala u T_i i presjeka *lijevih* intervala u T_j intervala).

Navedene ideje možemo iskoristiti za oblikovanje *sweep-line* algoritma koji rješava problem. Zamisliti ćemo vertikalni pravac koji se kreće s lijeva na desno i siječe (horizontalne) intervale. Pravac se u nekom trenutku nalazi na nekoj koordinati x . U tom trenutku znamo koliko *lijevih* intervala pravac siječe (*trenutni_lijevi_presjek*), te koliko je maksimalno *desnih* intervala pravac sjekao za od $x - 2(D-1)$ do x (*max_desni_presjek*). Zanima nas maksimalna vrijednost *trenutni_lijevi_presjek* + *max_desni_presjek* preko svih bitnih vrijednosti x . Bitne vrijednosti su kao i obično rubovi intervala.

Računanje *trenutni_lijevi_presjek* je prilično jednostavno, pa preostaje opisati kako tražimo *max_desni_presjek*, odnosno maksimalni presjek koji smo imali od $x - 2(D - 1)$ do x . Za tu potrebu možemo koristiti strukturu podataka poznatiju pod imenom *monotoni red*, koja omogućuje sljedeće operacije:

- *push*(vrijednost) -> ubaci vrijednost na kraj reda
- *pop*() -> izbaci vrijednost s početka reda
- *max*() -> nađi maksimalnu vrijednost koja je trenutno u redu

U trenutku kada dođemo na koordinatu x , u red ubacimo trenutni presjek i njegovu koordinatu, te pozivamo *pop*() sve dok presjek na početku reda ima koordinatu manju od $x - 2^{(D - 1)}$.

Umjesto monotonog reda moguće je koristiti balansirano stablo ili jednostavno STL set.

Operacije nad redom imaju amortizirano konstantnu složenost (N operacija nad redom ima složenost $O(N)$), pa na ukupnu složenost utječe samo sortiranje događaja i ona iznosi $O(N \log N)$.