Opisi algoritama

Primjeri implementiranih rješenja dani su u priloženim izvornim kodovima koji nužno ne odgovaraju u svim detaljima ovdje opisanim algoritmima.

Zadatak: Piramida

Ideja u ovom zadatku je uočiti da operacije zapravo samo *premještaju* prvo polje matrice. Na primjer, ako je operacija "uzmi prva 2 retka i stavi ih na kraju", ta operacija je zapravo prvo polje matrice premjestila dva retka ispod. Na isti način možemo operacije raditi unatrag, tj. samo oduzimati te pomake te tako doći do prvog polja početne matrice.

Zadatak: Raspored

Ovaj zadatak rješit ćemo binarnim pretraživanjem. Binarnim pretraživanjem tražit ćemo odgovor i ostaje pitanje kako napraviti provjeru može li Ivica rješiti sve zadatke ako mu najduža dozvoljena pauza traje x minuta. Primjetimo da možemo održavati vremenski interval u kojem Ivica može započeti sljedeći zadatak. Ako znamo kada je mogao započeti trenutni zadatak, onda za izračunati sljedeći interval potrebno je presjeći trenutni interval s intervalom kada on smije raditi sljedeći zadatak te zatim proširiti taj interval za vrijeme trajanja sljedećeg zadatka i dodati na to maksimalno dozvoljeno vrijeme pauze. Ako se ikada dogodi da interval nije pozitivne duljine, onda provjera vraća da nije moguće s tom duljinom pauze završiti sve zadatke. Za točne formule pogledajte u kodove autora.

Zadatak: Obratna kocka

Cilj ovog zadatka je bio da natjecatelj dobro razmisli o implementaciji rješenja prije nego krene kodirati jer je lako moguće doći do jako kompliciranog koda koji nije lagano ispraviti.

Ideja rješenja je zapravo identična kao i rješenje zadatka Kocka, treba nacrtati zadanu sliku. Primjetimo da se dimenzije matrice iz koje je nastao crtež lako odrede preko doljnjeg i desno ruba crteža.

Jednom kada znamo dimenzije matrice možemo krenuti određivati visine stupaca počevši od najdoljnjeg i najdesnijeg polja matrice. Jako je bitno početi od tog polja jer u tom slučaju uvijek određujemo prije one stupce koji zaklanjaju neke druge stupce.

Određivanje visine stupca može se na više načina. Općenita ideja je prvo probati nacrtati cijeli stupac visine x i ako smo to uspjeli zapamtimo najveći x za koji smo to uspjeli. Taj x je visina tog stupca. Nakon što smo oderdili x, nacrtamo taj stupac u neki naš privremeni crtež i idemo dalje.

Zadatak: Maratonci

Prva korak u rješavanju ovog zadatka je da svakom heksagonu pridružimo koordinate kako bi lakše pratili kretanje maratonaca kroz njih. Jedan od načina da to postignemo je da početni heksagon označimo koordinatama (0, 0) te stavimo da se x-os proteže od dolje-lijevo do gore-desno, a y-os od dolje prema gore. Time smo svakom heksagonu pridružili jedinstvenu (x, y) koordinatu te sada prepreke možemo jedinstveno definirati pomoću toga koja dva heksagona povezuju. To će nam olakšati provjeru koja govori smiju li neke dvije prepreke postojati zajedno ili ne.

Za prvi podzadatak potrebno je samo isprobati sve kombinacije na koji brid ćemo postaviti prepreku, za svaku kombinaciju provjeriti je li valjana te prebrojati koja valjana kombinacija postavlja najviše prepreka.

Za drugi podzadatak možemo riješiti tako da nam bridovi između dva heksagona predstavljaju bridove u bipartitnom grafu gdje su čvorovi točke gdje se sijeku tri heksagona. Na tom bipartitnom grafu zatim možemo pronaći takozvani "maximum bipartite matching" što nam daje rješenje. Ovaj pristup je relativno kompliciran te se nije očekivalo da su natjecatelji su upoznati s time, no navodimo ovo rješenje

iz edukacijskih razloga. Ovakav pristup je ponekad nužan u nekim drugim, kompliciranijim zadacima pa želimo skrenuti pozornost na to da takvo rješenje postoji za određena ograničenja.

Rješenje za sve bodove znatno je jednostavnije. Naime, u zadatku je navedeno da put neće prolaziti kroz niti jedan heksagon više od jednom. To ograničenje je važno zato što nam omogućuje da iskoristimo pohlepni algoritam. Pratimo put te na svaki brid preko kojega prođemo postavljamo prepreku ako ona nije u konfliktu s već postavljenim preprekama. U točnost tog rješenja lako se možete uvjeriti i sami stoga konkretan dokaz ostavljamo čitateljima za vježbu.

Zadatak: Kovanice

Rješenje ovog zadatka lakše je pogoditi intuitivno nego strogo dokazati. Logično je razmišljati da želimo da su što manje kovanice susjedne što većima jer tada veći dio te manje kovanice stane "ispod" veće kovanice. To se lako potvrdi promatranjem formule za širinu niza kovanica (formula se dobiva osnovnom geometrijom i primjenom Pitagorinog poučka) koja nam govori da se ukupna širina dobiva preko sume umnožaka korijena polumjera susjednih kovanica. Iz toga dobivamo sljedeću ideju: u sredinu stavimo najmanju kovanicu, okružimo ju s najvećim kovanicama, njih okružimo s najmanjim preostalim kovanicama, njih opet okružimo s najvećim preostalim kovanicama i tako dalje. Može se pokazati da upravo ova konstrukcija daje optimalno rješenje. Sam dokaz ostavljamo čitateljima za vježbu.

Zadatak: Vandal

Promotrimo korijen stabla. Znamo da u njegovom podstablu nalaze brojevi [1, N] te da on ima dva čvora ispod sebe. Od ta dva čvora, u jednom se sigurno nalaze brojevi [1, N/2], a u drugom [N/2 + 1, N]. Sada možemo pomoću N/2 upita provjeriti za svaki broj od 1 do N/2 nalazi li se taj broj u najljevijem listu lijevog djeteta. Time smo odredili da li se u lijevom podstablu nalazi prva ili druga polovica intervala [1, N] te smo za to iskoristili N/2 upita. Kada smo to odredili možemo se rekurzivno pozvati u lijevo i desno dijete s njima pripadajućim intervalima i ponavljati postupak. Kada dođemo do listova intervali će biti veličine jedan odnosno bit ćemo sigurni koji broj piše u tom listu. Analizirajmo koliko upita će ovaj algoritam napraviti.

Na prvoj razini imamo jedan čvor koji ima interval duljine N te koristimo N/2 upita kako bi saznali koja polovica intervala ide kojem djetetu. Na drugoj razini imamo dva čvora s intervalima duljine N/2 te za svaki od njih koristimo N/4 upita kako bi podijelili te intervale na intervale duljine N/4 što nam u sumi daje N/2 upita za drugu razinu. Analogno se ponavlja za sve preostale razine, odnosno na svakoj razini u sumi trošimo najviše N/2 upita. Razina na kojima obavljamo provjeru ima najviše 7 (logaritam u bazi 2 od 128) te je N najviše 128 dobivamo 7 * 64 = 448 upita što je dovoljno dobro za sve bodove.

Zadatak: Ograda

Zadatak ćemo rješiti dinamičkim programiranjem. Stanje će nam biti bitmaska u kojoj će 1 biti zapano na mjestima gdje se najdesnije pojavljuje neka boja u prvih i metara koje smo obojali. Na primjer ako smo ogradu obojali 121322, trenutna bitmaska nam je 001101. Primjetimo da sada znamo napraviti prijelaz u iduće stanje bojanjem sljedećeg metra ograde. Također, iz bitmaske se može odgovoriti na sve uvjete (jesu li zadovoljeni) kojima je desna granica metar ograde koji smo upravo obojali.

Zadatak: Žemlja

Nazovimo polje unutar ploče aktivnim ako se kroz njega može proći. Uočimo da će svi putevi unutar neke konfiguracije aktivnih polja koristiti disjunktne retke. Odnosno niti jedna dva puta neće se nalaziti u istom retku niti se na bilo koji način *ispreplitati*. Također, ako neko polje u srednjem stupcu nije aktivno možemo reći da problem možemo podijeliti na dva nezavisna problema *iznad* i *ispod* jer niti jedan put u optimalnom rješenju neće prolaziti kroz taj redak (u suprotnom taj put možemo skratiti).

Opišimo sad *Greedy* algoritam koji rješava problem u linearnoj složenosti za fiksan podskup aktivnih polja.

Počnemo od prvog retka i prolazeći kroz retke radimo sljedeće:

- ako naiđemo na redak u kojem srednji stupac nije aktivan, preskačemo taj stupac i sve polovične puteve brišemo.
- Inače, i ako je samo jedan od krajnih stupaca aktivan kreiramo polovični put ili zatvaramo polovični put ako je postajao od prije i suprotno krajnje polje u njemu je bilo aktivno.
- Inače, i ako su oba krajnja stupca aktivna, samo kreiramo novi put i nastavljamo dalje.

Inspirirani gornjim algoritmom, pokušajmo se fokusirati kako za određen podskup aktivnih polja držati rješenje pomoću neke strukture podataka koja će lako podržavati izmjene aktivnih polja. Struktura u pitanju koju predlažemo u ovom rješenju jest inačica turnir stabla (eng. *Segment Tree*).

Stablo gradimo nad retcima, gdje krajnji čvor u stablu predstavlja jedan redak unutar ploče. Unutar čvora turnirskog stabla koji predstavlja neki interval pamtit ćemo koliko je maksimalno puteva moguće napraviti promatraći samo taj interval redaka (još dodatno pod pretpostavkom da su sva polja u srednjem stupcu aktivna). Dodatno, pamtit ćemo još i maksimalan broj puteva ako smo imali neki polovični put iznad tog intervala koji je sadržavao prvi stupac te maksimalan broj puteva ako smo imali polovični put iznad intervala s zadnjim stupcem. Jedini problem koji preostaje je spajanje dva intervala. Da bismo to mogli za svaki od ova tri opisana broja iznad pamtimo još dodatno ostaje li na kraju intervala redaka neki polovični put ako rješenje nad njim gradimo gore opisanim algoritmom.

Dakle sumirano, za interval pamtimo maksimalan broj izgrađenih puteva te je li nam ostao polovični put na kraju (-1, 0, +1 u kodu). Također, pamtimo još dva puta isto s pretpostavkama da smo iznad intervala redaka imali neki polovični put od prije.

Za detalje kako navedenu strukturu održavati i iz nje izvući rješenje pogledajte u službenom kodu. Također, niz aktivnih srednjih stupaca možemo riješiti pomoću union find strukture ili na neki drugi način.