Opisi algoritama

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

Zadatak: Parnost (P1)

autor: Adrian Satja Kurdija
pripremili: Vedran Kurdija i Adrian Satja Kurdija

U opisima složenosti pretpostavljamo $N = M$.

Za podzadatak 1 bilo je dovoljno na sve moguće načine odabrati dva kuta pravokutnika (4 for petlje) te još dvama for petljama zbrojiti sva polja u tom pravokutniku. Složenost ovog pristupa je $O(N^6)$.

Za podzadatak 2 bilo je potrebno ovaj pristup ubrzati. Dio koji možemo ubrzati zbrajanje je polja u odabranim pravokutnicima. Ubrzavamo ga stvaranjem pomoćne matrice s, koja za svako polje (i, j) ima pohranjenu sumu prvih j polja u retku i. Matricu s dovoljno nam je izračunati jednom, može već i pri unosu podataka, korištenjem izraza $s[i][j] = s[i][j-1] + p[i][j]$. Ovu matricu sad možemo iskoristiti za brže zbrajanje polja u pravokutniku na način da umjesto dvije for petlje koje smo imali originalno, sada koristimo samo jednu koja prolazi po zadnjem stupcu odabranog pravokutnika i pribraja sume redaka pravokutnika, pribrajajući cijeli redak odjednom korištenjem izraza $p[i][j] - p[i][j - sirina]$. Složenost ovog pristupa je $O(N^5)$.

Za podzadatak 3 trebalo je primjetiti da pristup iz prošlog podzadatka funkcionira i u dvije dimenzije. Naime, možemo u matrici s umjesto suma u retcima imati pohranjene sume u podmatricama. Drugim riječima, u polju $s[i][j]$ bit će pohranjena suma prvih j polja u prvih i redaka, za razliku od prošlog pristupa koji je imao pohranjenu sumu samo za i-ti redak. Matricu s sada računamo korištenjem izraza $s[i][j] = s[i-1][j] + s[i][j-1] - s[i-1][j-1] + p[i][j]$, a zbrajanje polja u pravokutniku ovoga puta možemo izvesti samo jednom operacijom. Ako je donji desni kut pravokutnika (i, j), suma je $s[i][j] - s[i][j - sirina] - s[i - visina][j] + s[i - visina][j - sirina]$. Uspjeli smo složenost smanjiti samo na odabir pravokutnika, to jest na $O(N^4)$.

Kako ovaj pristup dodatno ubrzati, tako da možemo riješiti podzadatak 4? Sada više ne možemo na sve načine odabrati pravokutnik, ali možemo na sve načine odabrati stupce L i D kojima je omeđen, to jest u kojima se nalaze njegovi kutevi. To su dvije for petlje. Možemo si priuštiti još jednu koja prolazi po retcima. Sada za svaki redak i moramo nekako odgovoriti na pitanje: koliko postoji traženih pravokutnika s donjim desnim kutem u (i, D), a lijevim u (i, L)? Primijetimo da su to svi pravokutnici s gornjim lijevim kutem u (j, L) za koje vrijedi da je $s[i][D] - s[i][L-1] - (s[j-1][D] - s[j-1][L-1])$ paran broj. Budući da smo već potrošili tri for petlje za odabir vrijednosti L, D i i, taj broj moramo izračunati u jednoj operaciji. Primijetimo da je u tom izrazu $s[i][D] - s[i][L-1]$ fiksan te možemo tu vrijednost nazvati vrijednošću retka $v[i]$. Sada vidimo da se i drugi dio izraza može opisati kao vrijednost retka $v[j-1]$, tj. da smo izraz sveli na to koliko ima parnih brojeva $v[i] - v[j-1]$. To je isto kao da se pitamo koliko postoji redaka s vrijednošću $v[j-1]$ iste parnosti kao vrijednost retka $v[i]$. Uvedemo li dvije nove varijable, po jednu za svaku parnost, u kojima ćemo pamtiti koliko smo redaka s tom parnošću vrijednosti prošli, na početno pitanje možemo odgovoriti jednom operacijom. Nakon odgovora samo povećamo za 1 varijablu odgovarajuće parnosti za redak i. Ukupna složenost je $O(N^3)$.

Za sve bodove trebalo se sjetiti dodatne lukavštine koja može ubrzati čak i rješenje za podzadatak 4. Primijetimo da radimo s parnostima, to jest s bitovima. Zašto ne iskoristiti brzinu bitovnih operacija? Bitovne se operacije na jednom podatku izvršavaju u složenosti $O(1)$. U pristupu iz prethodnog podzadatka, treća for petlja ne može ici po svim retcima, ali može ići po svakom B-tom retku. B mora biti dovoljno velik da dovoljno smanji složenost, a dovoljno malen da B bitova stane u odabrani tip podataka, primjerice *long long* u programskom jeziku *C++*, s kojime onda možemo izvoditi bitovne operacije u $O(1)$. U službenom rješenju $B = 63$. Ukupna će složenost tada biti $O(N^2 * \lceil \frac{N}{63} \rceil)$, što je dovoljno brzo za $N \leq 1000$.

Postavlja se pitanje kako točno implementirati ovu optimizaciju. Možemo uvesti pomoćnu matricu *bits*, u koju će na svakom B-tom polju svakog stupca biti pohranjen broj čiji bitovi predstavljaju parnosti prethodnih B vrijednosti matrice s. Ovu matricu izračunamo unaprijed. Za svaki B-ti redak kojeg posjetimo, rješenju moramo pridodati one tražene pravokutnike koji se nalaze isključivo u posljednjih B redaka, kao i one čiji se donji kutevi nalaze u posljednjih B redaka, a gornji kutevi negdje prije. Tražene pravokutnike koji se nalaze isključivo u posljednjih B redaka lako izračunamo tako da prebrojimo jedinice i nule u $bits[i][D] \oplus bits[i][L-1]$ te izračunamo broj parova jedinica te broj parova nula. Primijetimo da korištenjem operacije $\oplus$ nad matricom *bits* koja je izgrađena nad matricom s, efektivno računamo parnosti vrijednosti redaka v za prethodnih B redaka. Preostaje nam prebrojiti tražene pravokutnike koji imaju gornje kuteve prije prethodnih B redaka. Njih računamo tako da u pomoćnim varijablama za svaku parnost pamtimo broj vrijednosti redaka s tom parnošću koje smo prošli prije trenutnih B redaka, a te vrijednosti množimo odgovarajućim brojevima parnosti vrijednosti redaka u trenutnih B redaka koje razmatramo, to jest brojevima nula i jedinica u $bits[i][D] \oplus bits[i][L-1]$. Nakon izračuna, povećamo pomoćne vrijednosti za odgovarajuće brojeve parnosti vrijednosti trenutnih B redaka.

Napomenimo da se ovo moglo jednostavnije implementirati korištenjem strukture podataka *bitset*, ali je službeno rješenje ne koristi, budući da cilj službenog rješenja nije bio osloniti se na određenu strukturu podataka. Rješenje koje koristi *bitset* također je dano u prilogu.

Zadatak: Trening (P1)

autor: Adrian Beker
pripremili: Pavel Kliska i Adrian Satja Kurdija

Za prvi podzadatak (svi znakovi su upitnici), trebamo "što ravnomjernije" rasporediti k znakova S i $n-k$ znakova V. Pretpostavimo da je manje znakova S nego znakova V (obrnuti slučaj rješava se analogno). Onda $n-k$ znakova V raspoređujemo u $k+1$ "pretinaca" koji su određeni dijelovima niza između k znakova S, brojeći i dijelove sasvim lijevo i desno. Broj znakova V u svakom od tih pretinaca određujemo cjelobrojnim dijeljenjem.

Za drugi podzadatak možemo isprobati svih 2^n mogućih rasporeda i pronaći najbolji od njih koji zadovoljava tražene uvjete. Iteraciju po svim mogućih rasporedima ostvarujemo rekurzijom ili bitmaskama.

Rješenje za sve bodove služi se dinamičkim programiranjem. Zamislimo da slijeva nadesno stvaramo traženi niz. Stanje (i, k, z) jest trenutna pozicija i, broj k preostalih znakova S koje treba staviti, te idući znak $z \in \{S, V\}$ koji stavljamo. U nekom stanju, na sve načine biramo koliko ćemo uzastopnih znakova z staviti (primjerice njih j), provjeravamo smijemo li staviti te znakove, te prelazimo u stanje gdje je nova pozicija $i+j$, idući znak onaj koji se razlikuje od z, te je k smanjen za j u slučaju $z = S$. Za pojedino stanje zanima nas koja je minimalna monotonija koju možemo postići od tog trenutka do kraja. Monotoniju za odabrani j računamo kao maksimum od samog broja j (uzastopnih znakova z) te rješenja za iduće stanje u koje smo prešli za taj j. Najbolji j, onaj koji daje najmanju monotoniju, pamtimo radi kasnije rekonstrukcije.

Zadatak: Skijalište (P1 i P2)

autor: Adrian Beker
pripremili: Adrian Beker, Josip Klepec

Zbog korištenja modularnih operacija promatrat ćemo težine staza od 0 do $K-1$ umjesto od 1 do K. Recimo da je plan kretanja *dobar* ukoliko prolazi sve staze točno jednom te počinje i završava u korijenu stabla. Recimo da je plan *izvrstan* ukoliko je dobar te prolazi staze težina redom $0, 1, \ldots, K-1$ i tako dalje periodički. Dakle, tražimo izvrstan plan kretanja s najmanjom mogućom monotonijom.

Primijetimo najprije da je dobar plan kretanja jedinstveno određen poretkom djece za svaki čvor. Zaista, jednom kad uđemo u podstablo nekog djeteta, moramo ga rekurzivno obići te se vratiti u trenutni čvor. Za svaki čvor v označimo sa s_v veličinu njegovog podstabla te s t_v težinu staze koja spaja v s njegovim

roditeljem, pri čemu posebno definiramo $t_v = K - 1$ ukoliko je v korijen. Fiksirajmo sada proizvoljan čvor v te izgradimo usmjeren graf G_v sa skupom vrhova $\{0, 1, \ldots, K-1\}$ te bridovima $e_u = (t_u, t_u + s_u)$ za svako dijete u čvora v (pri čemu zbrajamo modulo K). Plan kretanja je izvrstan ako i samo ako vrijedi sljedeće: kada prolazimo bridove e_u u grafu G_v prema poretku djece u čvora v, dobivamo Eulerovu stazu koja počinje u vrhu $t_v + 1$ te završava u vrhu $t_v + s_v$.

Sada nije teško riješiti verziju zadatka za prvu podskupinu: za svaki čvor v provjerimo postoji li u grafu G_v Eulerova staza od $t_v + 1$ do $t_v + s_v$. Ako za neki v ona ne postoji, tada je odgovor "NE". U suprotnom, odgovor je "DA": za svaki v nađemo pogodnu Eulerovu stazu te iz njih konstruiramo željeni plan kretanja na gore opisani način.

Za težu verziju zadatka potrebno je primijetiti sljedeće. Fiksirajmo poredak djece za svaki čvor v u nekom dobrom planu kretanja. Neka z_v označava posljednje dijete u poretku djece čvora v. Definirajmo funkciju f koja svakom čvoru v pridružuju duljinu niza vožnji žičarama kojim izlazimo iz podstabla čvora v. Tada f možemo računati rekurzivno na sljedeći način: ako je v korijen, tada je $f(v) = f(z_v)$, u suprotnom ako je v list, tada je $f(v) = 1$, a inače je $f(v) = f(z_v) + 1$. Nije teško vidjeti da je monotonija promatranog plana kretanja jednaka $\max_v f(v)$.

Sada se nameće sljedeći algoritam: za svaki čvor v odredimo skup D_v koji se sastoji od njegove djece u sa svojstvom da brid e_u može biti posljednji na Eulerovoj stazi od $t_v + 1$ do $t_v + s_v$ u G_v. Ukoliko takva staza uopće ne postoji, tada je D_v naravno prazan te je odgovor "-1". U suprotnom, D_v je neprazan te se sastoji od one djece u takve da brid e_u ulazi u vrh $t_v + s_v$ te nije most u neusmjerenoj verziji grafa G_v, osim u slučaju kada $t_v + s_v$ ima točno jedan ulazni brid (tada D_v sadrži upravo dijete koje odgovara tom bridu). Na kraju, zadatak rješavamo dinamičkim programiranjem: imamo dinamiku dp_v za svaki čvor v koju računamo kao $dp_v = \min_{u \in D_v} dp_u + 1$ (osim u posebnim slučajevim kada je v list ili korijen) te je tražena najmanja monotonija dana s $\max_v dp_v$. Konstrukciju plana koji ostvaruje najmanju monotoniju možemo izvesti slično kao u lakšoj verziji zadatka.

Ukupna je složenost rješenja u obje verzije $O(N + K)$.

Zadatak: Devet (P2)

autor: Adrian Satja Kurdija
pripremili: Adrian Satja Kurdija i Vedran Kurdija

Zadatak možemo riješiti tako da smjestimo točke na konkretne koordinate u koordinatnoj ravnini. Točku s brojem b možemo smjestiti na koordinate $\left(\lfloor \frac{b-1}{3} \rfloor, (b-1) \mod 3\right)$. Sada za svake dvije dužine izračunamo njihovo sjecište, ako ono postoji, to jest ako se sijeku i nisu paralelne. Za paralelne dužine smatramo da nemaju sjecišta te smo se time pobrinuli da ne ubrajamo preklapanja u sjecišta. Nakon toga iz sjecišta izbacimo duplikate i koordinate osnovih devet točaka. Konačno je rješenje broj preostalih sjecišta.

Preostaje opisati kako provjeriti jesu li dužine paralelne i sijeku li se. Neka prvu dužinu čine točke A i B, a drugu točke C i D. Predstavimo dužine kao pravce s jednadžbama $a_1x + b_1y = c_1$ i $a_2x + b_2y = c_2$. Koeficijent a_1 računamo kao $B.y - A.y$, a koeficijent b_1 kao $A.x - B.x$. Koeficijente a_2 i b_2 računamo analogno. Nakon toga računamo $d = a_1 \cdot b_2 - a_2 \cdot b_1$. Ako $d = 0$, dužine su paralelne te ih odbacujemo. Inače dobivamo sjecište kao $x = \frac{b2 \cdot c1 - b1 \cdot c2}{d}, y = \frac{a1 \cdot c2 - a2 \cdot c1}{d}$. No, valja imati na umu da je ovo sjecište pravaca na kojima se dužine nalaze. Još moramo provjeriti nalazi li se sjecište na dužinama. Za prvu dužinu postavljamo uvjete $min(A.x, B.x) \leq x \leq max(A.x, B.x)$ te $min(A.y, B.y) \leq y \leq max(A.y, B.y)$ te analogno činimo i za drugu dužinu. Ako su zadovoljeni, dužine se sijeku.

Zadatak: Bitstring (P2)

autor: Adrian Beker
pripremili: Adrian Beker, Pavel Kliska i Adrian Satja Kurdija

Dinamičko programiranje koje rješava zadatak u složenosti $O(N^3)$ dano je u gornjem opisu zadatka Trening.

Prvi korak k punom rješenju zadatka jest zamjedba da rješenje zadatka možemo tražiti binarnim pretraživanjem. Time se problem svodi na sljedeće: možemo li zamijeniti upitnike nulama i jedinicama tako da imamo ukupno K jedinica te nemamo blok od više od L uzastopnih jednakih znakova? Dakle, pitamo se koje sve brojeve jedinica možemo postići ukoliko nemamo blok od više od L uzastopnih jednakih znakova. Na to pitanje možemo odgovoriti dinamičkim programiranjem: za sve $i = 0, 1, \ldots, N$ i $k = 0, 1$ imamo skupove $D_{i,k}$ i $S_{i,k}$ koji su definirani na sljedeći način. $D_{i,k}$ sadrži sve moguće brojeve znakova $1 - k$, a $S_{i,k}$ sve moguće brojeve znakova k ukoliko rješavamo problem za prefiks danog stringa duljine i te i-ti znak postavljamo na k. Tada je odgovor na polazno pitanje "DA" ako je $K \in D_{N,0}$ ili $K \in D_{1,N-K}$, a inače "NE". Primijetimo da se skup $S_{i,k}$ sastoji upravo od onih j takvih da je $i - j \in D_{i,k}$, stoga preostaje samo odrediti skup $D_{i,k}$. To možemo učiniti rekurzivno: $D_{i,k}$ je unija skupova $S_{i',1-k}$ po svim $i' \in [\max(i - L, 0), i)$ takvim da se na pozicijama u intervalu $(i', i]$ u ulaznom stringu ne pojavljuje znak $1 - k$. Opisano rješenje nije teško implementirati u složenosti $O(N^2 \log N)$ pomoću prefiks suma ili $O(N^3/64 \cdot \log N)$ korištenjem strukture *bitset*.

Do punog rješenja dolazimo sljedećom zamjedbom: skupovi $D_{i,k}$, $S_{i,k}$ uvijek će biti intervali (moguće prazni). Zbog toga umjesto čitavih skupova možemo držati samo donje i gornje granice intervala te ih u prijelazu dinamike računati traženjem minimuma/maksimum na intervalu. To pak možemo učiniti u ukupnoj složenosti $O(N \log^2 N)$ korištenjem *tournament*. Za sve bodove bilo je potrebno još primijetiti da će se granice intervala na kojima tražimo minimume/maksimum monotono pomicati. Zato u prijelazu dinamike možemo umjesto tournament stabla koristiri monotoni red. Na taj način dobivamo rješenje složenosti $O(N \log N)$.