



Opisi algoritama

Zadatke, testne primjere i rješenja pripremili: Dominik Fistrić, Marin Kišić, Josip Klepec, Pavel Kliska, Vedran Kurdija, Ivan Paljak, Stjepan Požgaj i Paula Vidas. Primjeri implementiranih rješenja su dani u priloženim izvornim kodovima.

Zadatak: Autobahn

Pripremili: Marin Kišić i Vedran Kurdija

Potrebno znanje: sweep-line algoritam

Prvi podzadatak mogao se riješiti naivnom implementacijom traženih stvari iz teksta zadatka pa ga ostavljamo čitateljima za vježbu.

U drugom podzadatku može se izračunati pomoćni niz u kojem na poziciji i piše koliko ljudi plaća prekovremeno u i -toj minuti. Jednom kada imamo taj niz, možemo metodom dva pokazivača (ili prefiks sumama) pronaći interval duljine X s najvećom sumom. Taj pomoćni niz možemo izračunati tako da radimo sweep-line po minutama i na odgovarajući način obrađujemo događaje koji mogu biti tri različita tipa:

- Osoba ulazi na autobahn.
- Osobi je isteklo plaćeno vrijeme.
- Osoba izlazi s autobahna.

Za sve bodove ne možemo direktno konstruirati taj niz, međutim možemo primjetiti da je taj niz zapravo “napravljan” od $\mathcal{O}(N)$ disjunktnih intervala koje možemo dobiti iz prethodno spomenutog sweepa. Također, bitno je primijetiti da naš interval duljine X s najvećom sumom će ili počinjati ili završavati na rubu (početku ili kraju) nekog od prije spomenutih intervala. To nas dovodi do zaključka da možemo metodom dva pokazivača pronaći naš interval duljine X s najvećom sumom.



Zadatak: Cigle

Pripremio: Josip Klepec

Potrebno znanje: dinamičko programiranje, sweep-line algoritam

Prvi podzadatak mogao se riješiti generiranjem svih mogućih zidova tako da nakon svake cigle odlučimo hoćemo li prelomiti zid ili ne.

Ostatak zadatka rješavamo dinamičkim programiranjem.

$dp[l][r]$ - predstavlja najveću moguću ljepotu zida koji se može dobiti pomoću prvih r cigli tako da je zadnji redak od l -te do r -te cigle

$$dp[l][r] = \max_{1 \leq x \leq l-1} (dp[x][l-1] + broj_novonastalih_točaka) \quad (*)$$

Naivnom implementacijom postizemo složenost $\mathcal{O}(N^4)$. Za ubrzanje možemo primjetiti kako broj novonastalih točaka ne moramo svaki put iznova računati.

Odnosno, računanje danog maksimuma u formuli (*) radimo tako da prolazimo x -evima od $l-1$ do 1. Odnosno proširivanjem prethodnog retka.

Vrijednost $broj_novonastalih_točaka$ dodavanjem cigle u prethodni redak može ili ostati ista ili povećati se za 1.

Ovime postizemo složenost $\mathcal{O}(N^3)$

Fiksirajmo l i odjednom pokušajmo izračunati sve vrijednosti $dp[l][j], l \leq j \leq N$. Primjetimo da isto kao što smo proširivali prethodni redak možemo proširivati i trenutni redak.

Računamo $dp[l][j], l \leq j \leq n$ (nazovimo ta stanja trenutni retci) pomoću prethodnih redaka $dp[i][l-1], 1 \leq i \leq l-1$.

Cilj je paralelno prolaziti svim "trenutnim retcima" i "prethodnim retcima" **po veličini**.

Kada dodemo do trenutnog retka njegova vrijednost dinamike bit će najbolji prethodni redak koji smo do sada obradili ili neki prethodni redak koji još nismo odradili + broj trenutnih poklapanja.

Odnosno zanima nas maksimalan prethodni redak koji još nije obrađen i na njega ćemo dodati broj trenutnih preklapanja i najbolji prethodni redak koji je već obrađen jer se za njih više ne nastaju nove točke.



Zadatak: Izvanzemaljci

Pripremili: Ivan Paljak i Pavel Kliska

Potrebno znanje: binarno pretraživanje, sweep-line algoritam, rastav na slučajeve, implementacijske vještine

Za prvi podzadatak u kojemu je $K = 1$, bilo je dovoljno pronaći AABB (*Axis-Aligned Bounding Box*) svih točaka, te ga po potrebi proširiti tako da bude kvadrat.

Za drugi podzadatak u kojemu je $K = 2$, bilo je potrebno uočiti da će u svakom ispravnom rasporedu kvadrata, točke koje su unutar jednoga kvadrata biti ili prefiks točaka sortiranih po x koordinati ili prefiks točaka sortiranih po y koordinati. Sve preostale točke nužno će se nalaziti u drugom kvadratu. Za slučaj u kojemu su točke sortirane po x koordinati dovoljno je pronaći AABB svih prefiksa i sufiksa točaka i AABB-ove odgovarajućih prefiksa i sufiksa proširiti do kvadrata tako da AABB-ove prefiksa širimo ulijevo i dolje/gore, a AABB-ove sufiksa udesno i dolje/gore. Analogno riješavamo slučaj u kojemu su točke sortirane po y koordinati

Za rješenje u slučaju $K = 3$, potrebno je uočiti da za sve pravilne rasporede kvadrata vrijedi slično svojstvo kao i za slučaj $K = 2$, samo što u ovome slučaju moramo promatrati točke sortirane uzlazno i silazno po x i po y koordinati. Kako bismo pojednostavnili kod, umjesto različitih sortova točaka tri puta ćemo rotirati točke oko središta koordinatnog sustava za 90° i promatrati rasporede kvadrata u kojima je prvi kvadrat uvijek prefiks točaka sortiranih uzlazno po x koordinati. Za neki takav prefiks pronaći ćemo AABB i proširiti ga do kvadrata ulijevo ili gore/dolje.

Za svaki tako dobiveni kvadrat potrebno je pronaći što bolju podijelu preostalih točaka u dva kvadrata tako da oni ne sijeku prvi. Može se pokazati da je dovoljno riješiti drugi podzadatak s dodatnim ograničenjem da niti jedan kvadrat ne smije sadržavati točke čija je koordinata manja ili jednaka od x_0 , x koordinate desne strane prvoga kvadrata. Slično kao u slučaju za $K = 2$ promatramo AABB-ove prefiksa i sufiksa točaka sortiranih po x i po y koordinati. Ako je u optimalnom rasporedu preostalih točaka jedan AABB prefiks točaka sortiranih po y osi, taj AABB širimo dolje desno, a drugi AABB širimo gore desno. Ako je u optimalnom rasporedu preostalih točaka jedan AABB prefix točaka sortiranih po x osi, taj AABB pokušavamo širiti lijevo i desno između x_0 i AABB-a preostalih točaka. Ako ga se ne može proširiti, izbacujemo taj raspored točaka po AABB-ovima iz razmatranja. Ako ga se može proširiti, po potrebi ga dodatno širimo dolje, a AABB sufiksa širimo do kvadrata desno i gore. Cijeli ovaj postupak moguće je napraviti u vremenskoj složenosti $O(n^2 \log n)$ što je dovoljno za 3. i 4. podzadatak. Za sve bodove bilo je potrebno uočiti da ako povećamo dimenzije prvoga pravokutnika, maksimalna dimenzija druga dva se smanjuje pa je bilo moguće napraviti binary search po maksimalnoj dimenziji kvadrata. Usto za dodatno ubrzanje su u službenom kodu memoizirane maksimalne dimenzije zadnja dva kvadrata za svaki sufiks točaka što daje ukupnu složenost $O(n \log n)$



Zadatak: MalnaRISC

Pripremio: Ivan Paljak

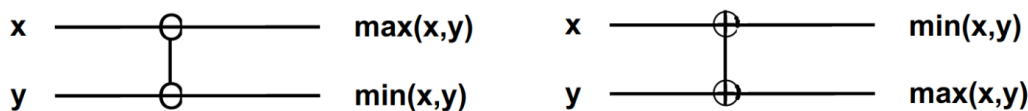
Potrebno znanje: algoritmi za sortiranje, podijeli pa vladaj, sortirajuće mreže, bitonic sort

Raznim strategijama bilo je moguće implementirati manje ili više efikasne algoritme za sortiranje. Neke od najčešćih su bile:

- **cca. 10 bodova** – Direktna bubble sort implementacija u trajanju od $N * (N - 1)/2$ nanosekundi.
- **cca. 30 bodova** – Paraleliziranje usporedbi iz unutarnje petlje direktne bubble sort implementacije. Primjerice, u jednoj naredbi uspoređivanje registara R_1 i R_2 , R_3 i R_4 , ..., a u drugoj naredbi uspoređivanje registara R_2 i R_3 , R_4 i R_5 , ... Ponavljanjem ovih dviju linija N puta dobivamo popularan bubble sort algoritam u trajanju od $2N$ nanosekundi.
- **cca. 70 bodova** – Algoritmi temeljeni na tehnici podijeli pa vladaj inspirirani standardnim merge sort algoritmom. Također, ove je bodove bilo moguće osvojiti zamjedbom da je dvije naredbe iz opisanog algoritma za 30 bodova bilo dovoljno pokrenuti izvesti samo N puta.

Sve bodove na ovom zadatku bilo je moguće osvojiti koristeći teoretsku podlogu o sortirajućim mrežama. Odnosno, zaključiti da se problem svodi na traženje sortirajuće mreže dubine $\mathcal{O}(\log^2 N)$. Primjer jednog takvog algoritma jest tzv. *bitonic sort*.

Sortirajuća mreža apstraktni je uređaj izgrađen od niza žica. U svaku žicu na lijevom kraju ulazi neka vrijednost, dok na desnom kraju ta vrijednost izlazi. Svojstvo sortirajuće mreže jest da će skup izlaznih vrijednosti promatran odgozgo nadolje, biti sortiran. Dvije je žice moguće povezati posebnom žicom za usporedbu koja će po potrebi zamijeniti vrijednosti koje teku kroz te dvije žice ako se one nalaze u *pogrešnom* poretku. Na krajeve žice za usporedbu ucertat ćemo znak \oplus ako želimo da se na gornjoj žici nalazi manji broj, odnosno znak \ominus ako želimo da se na donjoj žici nalazi manji broj.



Dubina sortirajuće mreže neformalno se definira kao najveći broj žica za usporedbu kroz koje može proći neka ulazna vrijednost. Dakako, zamislit ćemo da se na lijevom kraju sortirajuće mreže nalaze vrijednosti u registrima R_1, R_2, \dots, R_N , žice za usporedbu odgovarat će asemblerskoj instrukciji $\text{CMPSWP } R_Y \ R_X$, odnosno $\text{CMPSWP } R_X \ R_Y$, a dubina vremenskom trajanju algoritma.

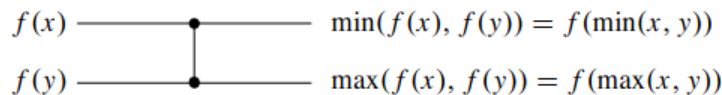
Prije nego što predstavimo sortirajuću mrežu temeljenu na *bitonic* sort algoritmu, iznijet ćemo par korisnih tvrdnji o samoj prirodi sortirajućih mreža.

Tvrdnja 1: Ako sortirajuća mreža pretvori ulaz $a = (a_1, a_2, \dots, a_n)$ u izlaz $b = (b_1, b_2, \dots, b_n)$, tada za svaku monotono rastuću funkciju f , ista mreža pretvara ulaz $f(a) = (f(a_1), f(a_2), \dots, f(a_n))$ u izlaz $f(b) = (f(b_1), f(b_2), \dots, f(b_n))$.

Dokaz. Promatrajmo što se događa nakon što ulazne vrijednosti naiđu na prvi paralelni skup žica za usporedbu, odnosno što će se dogoditi nakon prve nanosekunde u *MalnaRISC*-u. Ako pokažemo da će nailaskom na usporednu žicu dvije vrijednosti $f(x)$ i $f(y)$ zamijeniti mjesto ako bi i vrijednosti x i y zamijenile mjesto, tada je jasno da se skupovi ulaznih podataka nakon prve nanosekunde ponašati jednako neovisno o primjeni funkcije f . Induktivno tada slijedi da tvrdnja vrijedi.



Budući da funkcija f monotonno raste, očigledno vrijedi $\min(f(x), f(y)) = f(\min(x, y))$. Analogno, vrijedi $\max(f(x), f(y)) = f(\max(x, y))$. Stoga, brojevi $f(x)$ i $f(y)$ će zamijeniti mjesto ako u toj situaciji i brojevi x i y mijenjaju mjesto.



□

Tvrđnja 2: Ako sortirajuća mreža ispravno sortira svih 2^n različitih nizova nula i jedinica, tada će ta sortirajuća mreža ispravno sortirati proizvoljan niz od n brojeva.

Dokaz. Pretpostavimo suprotno. Odnosno, pretpostavimo da neka mreža ispravno sortira sve nizove nula i jedinica, ali neispravno sortira neki ulazni niz $a = (a_1, a_2, \dots, a_n)$.

U tom nizu tada postoje neka dva elementa $a_i < a_j$ takvi da mreža na izlazu postavi broj a_j prije broja a_i .

Definirajmo monotonno rastuću funkciju f kao:

$$f(x) = \begin{cases} 0, & x \leq a_i \\ 1, & x > a_i \end{cases}$$

Prema prethodnoj tvrdnji, budući da mreža na izlaz postavlja a_j ispred a_i , tada ona mora na izlaz postaviti $f(a_j)$ ispred $f(a_i)$. S obzirom na to da je $f(a_j) = 1$, a $f(a_i) = 0$, to je u kontradikciji s pretpostavkom. □

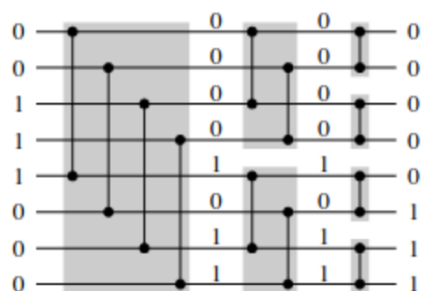
Niz brojeva nazivamo *bitoničkim* ako najprije monotonno raste, a zatim monotonno pada ili se pak može cirkularno posmaknuti tako da najprije monotonno raste, a zatim monotonno pada. Izgradit ćemo najprije sortirajuću mrežu dubine $\log N$ koja sortira bitoničke nizove brojeva. Radi lakše analize, u ostatku teksta pretpostavit ćemo da je N potencija broja 2.

Promatrajmo paralelan niz žica za usporedbu tipa \oplus koje spajaju žicu i sa žicom $i + \frac{N}{2}$. Propustimo li kroz tako postavljene žice za usporedbu neki bitonički niz nula i jedinica, na izlazu ćemo dobiti sljedeća svojstva:

- obje polovice bit će bitonički nizovi
- niti jedan element u prvoj polovici neće biti veći od nekog elementa iz druge polovice
- barem jedna polovica sastojat će se samo od nula ili samo od jedinica

Dokaz ovih svojstava ostavit ćemo čitatelju za vježbu (*smjernica*: pretpostavite da je ulazni niz oblika $0^a 1^b 0^c$ i raspišite sve slučajeve).

Pretpostavljate, isti postupak ćemo ponoviti na svakoj polovici i u konačnici na izlazu dobiti sortiran niz nula i jedinica. Primijetite da zbog tvrdnje 2, ovakva mreža dubine $\log N$ ispravno će sortirati bilo kakav bitonički niz.



Primjer sortirajuće mreže za bitonički niz duljine 8.

Zamislamo sada da radimo standardni *merge* sort algoritam. Dio slagalice koji fali jest postupak kojim ćemo spojiti dva sortirana niza u jedan. Prepostavimo ponovno da su ulazne vrijednosti samo nule i jedinice. Neka su dva sortirana niza oblika $0^a 1^b$ i $0^x 1^y$. Preokrenemo li drugi niz i najlijepimo li ga na kraj prvog, dobit ćemo niz $0^a 1^{b+y} 0^x$. To je bitonički niz!

Primjenom ove dosjetke dobivamo sortirajuću mrežu dubine $\mathcal{O}(\log^2 N)$ što je dovoljno za osvajanje svih bodova na ovom zadatku.