

# **HRVATSKA INFORMATIČKA OLIMIJADA 2011.**

## **OPISI ALGORITAMA**

HIO 2011.	Zadatak SORT

Najvažnije je za uočiti kako je za sortiranje bilo kojeg niza dovoljno koristiti najviše dvije cikličke zamjene. Možemo podijeliti rješenje u nekoliko zasebnih slučajeva:

1. *Početni niz je sortiran* - U ovom slučaju nije potrebna nijedna zamjena.
2. *Dovoljna je jedna zamjena* - Ukoliko se na mjestu **P** u ulaznom nizu nalazi broj **a**, na mjestu **a** nalazi broj **b**, na mjestu **b** nalazi broj **c**, ..., na mjestu **z** nalazi broj **P**, tada takvu konfiguraciju na nazivamo **zanimljivom**. Primjerice, ulazni niz [5, 4, 1, 3, 2] je ustvari jedna velika zanimljiva konfiguracija. U takvim je slučajevima dovoljno napraviti samo jednu zamjenu.
3. *Dovoljne su dvije zamjene* - Lako se pokaže kako je ulazni niz uvijek moguće savršeno particionirati u konačan broj zanimljivih podnizova. Ukoliko napravimo cikličku zamjenu s jednim brojem iz svake zanimljive konfiguracije, one će se potom nužno spojiti u jednu zanimljivu konfiguraciju koju je moguće riješiti po gornjem pravilu pod brojem 2, ukupno koristeći točno dvije cikličke zamjene.

#### Potrebno znanje:

-

#### Kategorija

ad-hoc

HIO 2011.	Zadatak TELKA

Jedno moguće rješenje ovog problema bilo bi podijeliti niz koji označava popularnost svake sekunde u nekoliko blokova. Broj sekundi u danu je 86400. Neka je A odabrana veličina bloka. Intervale o gledanju obrađivat ćemo redom jedan po jedan te prema njima osvježavati niz popularnosti. Promatrajmo sada neki interval koji obrađujemo - postoje dva slučaja obrade:

- a) interval prelazi preko ponoći - update se zapravo razdvaja na dva, jedan prije ponoći, drugi nakon
- b) interval ne prelazi preko ponoći - ubrzavamo brute force update tako što ćemo, umjesto updateanja svakog elementa posebno zapravo updateati blok po blok. Prvi obrađujemo rubne blokove jedan po jedan element, a zatim obrađujemo sve blokove između ta dva rubna. Kada je cijeli blok obuhvaćen, njegove sve elemente možemo updateati jednom operacijom.

Upute radimo na način analogan updateu.

Primijećujemo da nam brzina algoritma ovisi o izboru veličine bloka A. Za potrebe ovog zadatka, A je bio podešen na 100 što je učinilo rješenje dovoljno brzim.

Napomena: Postoji i brže alternativno rješenje koje cijeli zadatak rješava u linearnom vremenu. U tom rješenju koriste se prefix sume za čuvanje podataka o popularnosti svake sekunde.

Potrebno znanje  
Bucketi, prefix sume

Kategorija  
Strukture podataka, ad hoc

**Potrebno znanje:**  
Bucketi, prefix sume

**Kategorija**  
Strukture podataka, ad hoc

<b>HIO 2011.</b>	<b>Zadatak RIJEKA</b>

Jasno je da treba naći put sa što manje „vraćanja“. Ako u svom putovanju čamcem dođemo do sela B, pa se vratimo natrag do sela A, pa ponovno putujemo do B i nastavimo dalje, mi smo interval  $[A, B]$  prošli tri puta, dok bismo ga bez vraćanja prošli samo jednom. Dakle, intervale po kojima se vraćamo prolazimo dva puta više, pa će konačno rješenje biti

$$M + 2 * \text{ukupna\_duljina\_intervala\_po\_kojima\_se\_vraćamo}.$$

Pronađimo što manje intervale vraćanja. Osobe koje putuju slijeva na desno možemo potpuno zanemariti: njih ćemo prevesti u svakom slučaju. Osobe koje putuju zdesna na lijevo bit će nam važne: svaka od njih definira jedan interval po kojem se moramo vraćati (ali možda samo kao dio nekog većeg intervala vraćanja).

Sada se prirodno nameće sljedeće rješenje: naći ćemo **uniju** svih intervala po kojima se moramo vraćati (definiranih od osoba koje putuju zdesna na lijevo), i ta unija bit će tražena ukupna duljina intervala po kojima se vraćamo.

Preostaje nam dovoljno efikasno pronaći uniju zadanih intervala. To možemo učiniti koristeći način razmišljanja poznat kao „sweep line“: idući slijeva na desno registramo događaje tipa „početak intervala“ i „kraj intervala“ (te događaje prethodno sortiramo po vremenu). Složenost algoritma je  $O(N \lg N)$ .

### Potrebno znanje:

Matematička analiza problema, sweep-line

### Kategorija

Sweep

<b>HIO 2011.</b>	<b>Zadatak KAMION</b>

Rješenje ovog teškog problema ćemo prezentirati na pojednostavljenoj verziji samog zadatka. U ovog pojednostavljenoj verziji se neće nikada pojaviti ceste tipa 3, te kamion mora u zadnjem koraku doći u grad **N** upravo **prazan**. Drugim riječima, on smije više puta proći kroz grad **N** na svome putu bez ikakvog ograničenja, ali u svom posljednjem koraku kada ulazi u grad **N**, on mora u taj grad ući prazan.

Ostatak ovog teksta ćemo posvetiti upravo tom problemu s dva pojednostavljenja, dok ćemo rješenje originalnog problema ostaviti čitatelju za vježbu. Također, moguće je iskoristiti službeno rješenje kao referencu. adatka).

Dakle, rješavamo problem kako doći iz grada 1 u grad **N** koristeći najviše **K** koraka tako da je na kraju puta kamion prazan. Rješenje dobivamo korištenjem metode dinamičkog programiranja. Označimo s  $dp[a][b][k]$  broj načina na koje možemo iz grada **a** doći do grada **b** koristeći **točno k** koraka ukoliko smo počeli sa praznim kamionom te ukoliko u gradu **b** moramo završiti s praznim kamionom. Prva cesta koju ćemo morati proći prilikom svoga putovanja će morati biti cesta tipa 1 u kojoj ukrcavamo neki teret. Označimo tu cestu sa **a -> x**. Potom odabiramo cestu na kojoj iskrcavamo **točno taj komad** tereta (koja mora biti tipa 2). Označimo tu cestu oznakom **y -> z**. Potom u rješenje možemo pribrojiti sve puteve koje podrazumijeva  $dp[x][y][k1] * dp[z][b][k2]$  gdje su  $k1+k2+2 = k$ . Nažalost, rješenje implementirano po tim crtama će imati složenost  $O(E \cdot K^2)$  što nije dovoljno za dobivanje svih bodova na zadatku.

Efikasnije rješenje ćemo dobiti uvodom pomoćne dinamike  $dp2[a][b][k]$  u kojoj ćemo zapisati broj puteva između gradova **a** i **b** koristeći točno **k** koraka tako da nam je kamion na početku prazan, na kraju prazan, te **nikada** između tih trenutaka **nije prazan**.

Dinamičke tablice izračunavamo rastuće s brojem koraka **k**. Prilikom izračunavanja tablice  $dp$  tražimo prvi grad i korak u kojem će nam se kamion na putu isprazniti. Neka se to dogodi u gradu **c** i u koraku **k1**. Tada je:

$$dp[a][b][k] += dp2[a][c][k1] * dp[c][b][k-k1]$$

Paralelno s time, izračunavamo i dinamiku  $dp2[a][b][k]$ . Potražimo ceste **a -> x** i **y -> z** gdje redom ukrcavamo i iskrcavamo istu vrstu tereta. Tada je:

$$dp2[a][b][k] += dp[x][y][k-2]$$

Ukupna složenost ovog rješenja se može pokazati da je  $O(\mathbf{N} \cdot \mathbf{K}^2) + O(\mathbf{E}^2)$ , što je dovoljno za dobiti sve bodove na zadatku.

**Potrebno znanje:**

Napredno dinamičko programiranje

**Kategorija**

Grafovi, dinamičko programiranje

HIO 2011.	Zadatak LOVCI

Promatrajmo ploču:

```

xxoxxx
xxxxxx
xxxxxx
xxxxxx
xxxxxx
xxxxxx
xxxxxx

```

Odvojimo bijela i crna polja (jedan lovac skače po bijelima, drugi po crnima):

```

x o x      x o x
x x x      x x x
  x x x    x x x
x x x      x x x
  x x x    x x x
x x x      x x x

```

Zarotirajmo za 45:

```

  xx      x
  xxxo    xxo
xxxxxxx  xxxxx
  xxxx    xxxxx
  xx      xxx
          x

```

Retke možemo sortirati po duljini, i dalje će isti skup polja biti vidljiv sa svakog polja:

```

xxxxxxx  xxxxx
  xxxx    xxxxx
  xxxo    xxx
  xx      xxo
  xx      x
          x

```

Isto vrijedi i za stupce:

```

xxxxxxx  xxxxx
xxxxx    xxxxx
xxxo     xxx
xx       xxo
xx       x
        x

```

Dakle treba riješiti gornju krnju sortiranu ploču. Za svaki K, koliko je rješenje ako lovac napravi u K optimalnih koraka na toj ploči?

Promatrajmo niz redaka R i niz stupaca S u kojima će se lovac bar jednom nalaziti. Vrijedi da se prvi redak iz R mora sjeći sa zadnjim stupcem iz S (inače se taj stupac ne sječe ni s kim, pa se u njega ne može doći).

Vrijedi da se zadnji redak iz R mora sjeći sa prvim stupcem iz S (inače se taj redak ne sjeće ni s kim, pa se u njega ne može doći).

Ako gornje vrijedi onda se prvi redak iz R sjeće sa svim stupcima iz S (jer je zadnji najmanji).

Onda također vrijedi da možemo napraviti ovakav niz skokova (koji će skočiti u svaki redak i stupac točno jednom):

- 1.) Skoči iz početnog retka u prvi redak iz R.
- 2.) Skoči u svaki stupac iz S bilo kojim redom, ali zadnje skoči u prvi stupac iz S. (prvi redak iz R vidi sve stupce iz S)
- 3.) Skoči u svaki preostali redak iz R. (prvi stupac iz S vidi sve retke iz R)

Primjetite da je u drugom koraku moguće da lovac mora 2 puta skočiti u isti stupac (ako je prvi stupac iz S početni stupac). Stoga, ako je prvi stupac iz S jednak početnom stupcu lovca, tada lovac neće skočiti ni u jedan drugi stupac (jer ne može skočiti u manje stupce), pa se drugi korak može izbaciti.

Nadalje:

Za  $k == 0$  treba oduzeti početno polje lovca jer se ono ne broji.

Za  $k \geq 2$  moguće je odlučiti skakati samo po već viđenim recima/stupcima, tako da treba napraviti  $dp[k] = \max(dp[k-1], dp[k])$ .

Implementacija:

1. Odaberi podskup stupaca koji sadrži početni stupac lovca na sve načine. Za svaki redak izračunaj koliko je para ostalo u njemu.
2. Odaberi najbolji redak iz kojeg se može skočiti u sve stupce u podskupu.
3. Odaberi početni redak.
4. Dalje greedy dodajemo jedan po jedan najbolji redak u koji se može skočiti iz prvog stupca iz podskupa.

Za 100% bodova još treba zaključiti kako se ne isplati nikad skočiti ni u jedan kut, pa se korak gdje se biraju svi podskupi može 4 puta brže napraviti. Bez ove optimizacije se dobiva 90% bodova.

### **Potrebno znanje:**

Dinamičko programiranje

### **Kategorija**

Ad hoc