



## Opisi algoritama

Zadatke, testne primjere i rješenja pripremili: Fran Babić, Toni Brajko, Nikola Dmitrović, Sofija Velkovska, Marin Kupanovac, Ivan Jambrešić i Jakov Celin.

Primjeri implementiranih rješenja dani su u priloženim izvornim kodovima.

### Zadatak Zvono

Pripremio: Nikola Dmitrović

Potrebno znanje: naredbe učitavanja i ispisivanja

Nakon što je upisan broj  $X$ , želimo ispisati broj zvona za kraj prvog sata, to jest odrediti koliko je  $X$  udaljen od 45. Dakle, dovoljno je učitati broj  $X$  s ulaza, a zatim ispisati  $45 - X$ .

*Programski kod (pisan u Python 3):*

```
X = int(input())  
  
print(45 - X)
```

### Zadatak Korekcija

Pripremio: Nikola Dmitrović

Potrebno znanje: naredba odlučivanja (if), određivanje znamenki dvoznamenkastog broja

Za zadane dvoznamenkaste brojeve  $X$  i  $Y$  želimo odrediti koliko pripadajućih znamenki im se razlikuje. Trebamo im usporediti znamenke jedinica i desetica. Znamenke jedinica pronalazimo uzimajući ostatak broja pri dijeljenju s 10, a znamenke desetica tako da broj podijelimo s 10. Nakon određivanja znamenki jedinica i desetica, uspoređujemo znamenke jedinica i znamenke desetica. Na kraju ispisujemo koliko znamenki se razlikovalo.

*Programski kod (pisan u Python 3):*

```
X = int(input())  
Y = int(input())  
  
jx = X % 10  
dx = X // 10  
  
jy = Y % 10  
dy = Y // 10  
  
koliko = 0  
  
if jx != jy: koliko += 1  
if dx != dy: koliko += 1  
  
print(koliko)
```



## Zadatak Trening

Pripremila: Sofija Velkovska

Potrebno znanje: naredba odlučivanja (if), naredba ponavljanja (for)

Ako se Marin nalazi na zgradi na poziciji  $i$ , on **neće** moći skočiti na sljedeću zgradu (na poziciji  $i + 1$ ) ako je sljedeća zgrada viša za **više** od jednog metra od trenutne zgrade. Dakle, ne smije vrijediti  $h_{i+1} > h_i + 1$ . U svim ostalim slučajevima (sljedeća zgrada je niža od trenutne, zgrade su iste visine, sljedeća zgrada je za jedan metar viša od trenutne) Marin će moći skočiti na sljedeću zgradu i nastaviti svoj trening.

Ako za sve susjedne zgrade tj. za sve  $i$  i  $i + 1$ , Marin može skočiti s prethodne na sljedeću, on će bez problema doći do kraja ulice, tj. završit će svoj trening na  $n$ -toj zgradi. Inače, trebamo pronaći prvi indeks  $i$  za koji vrijedi  $h_{i+1} > h_i + 1$ . Indeks  $i$  je broj prve zgrade s koje se Marin neće moći skočiti dalje pa na  $i$ -toj zgradi završava svoj trening.

*Programski kod (pisan u Python 3):*

```
n = int(input())
h = list(map(int, input().split()))

mini = n
for i in range(n - 1):
    if h[i + 1] > h[i] + 1:
        if i < mini:
            mini = i + 1

print(mini)
```

## Zadatak Minesweeper

Pripremio: Marin Kupanovac

Potrebno znanje: matrice, naredba ponavljanja (for), naredba odlučivanja (if)

Zadatak se mogao riješiti na više načina, no opisati ćemo službeno rješenje. Zadatak rješavamo upisujući pozicije bombi (označimo ih sa slovom  $B$  ili  $-1$ , rješenje je identično) te susjedna polja na kojima nisu bombe povećati za 1. Pronalazak pozicija susjednih polja je zahtjevniji dio zadatka, no njih možemo odrediti ručno i zapisati u pomoćnu listu. Nakon određivanja svih brojeva matrice, ispisujemo ju kao što je traženo u zadatku. Za detalje, pogledajte službenu implementaciju.

## Zadatak Dodatna

Pripremio: Jakov Celin

Potrebno znanje: sweep line, multiset

Želimo prvo pojednostaviti uvjet održavanja dodatne nastave iz teksta zadatka. Dodatna nastava će se moći održati u nekom intervalu  $[L, R]$  ako za barem  $k$  učenika vrijedi da se nalaze u školi u  $L$ -toj sekundi te će oni ostati u školi barem do  $R$ -te milisekunde.

Ovaj pristup nas dovodi do ideje za rješenje zadatka. Za svaku lijevu granicu  $L$  odrediti ćemo koliko Jakov **najduže** može držati dodatnu nastavu ako ju odluči započeti u  $L$ -toj sekundi te zatim zapamtiti maksimalnu duljinu održavanja. Primjetimo da su nam jedine lijeve granice bitne za dobivanje rješenja samo one kada je neki učenik dolazio u školu jer Jakov može započeti dodatnu u trenutku prije trenutnoga ako nijedan učenik tada došao u školu.

Fiksirajmo lijevu granicu održavanja dodatne nastave  $L$ . Ako znamo sve **desne** granice intervala učenika koji se već nalaze u školi u  $L$ -toj sekundi, Jakov tada može održati dodatnu do njihove  $k$ -te najveće desne granice (u slučaju da je učenika nedovoljno puno tj. manje od  $k$ , onda ju ne može uopće održati).



Kako bi za svaku lijevu granicu  $L$  imali popis svih učenika koji se u toj sekundi nalaze u školi, koristiti ćemo sweep line algoritam i strukturu koja će održavati  $k$  najvećih desnih granica učenika te ubacivanje i izbacivanje desnih granica iz strukture. Jedna od takvih struktura je multiset.

Sve događaje (dolaski i odlasci učenika) sortiramo te ih redom prolazimo. Ako smo naišli na lijevu granicu nekog intervala, tada učenikovu desnu granicu ubacujemo u multiset (u slučaju da multiset sadrži više od  $k$  desnih granica, izbacimo najmanju), pronademo  $k$ -tu najveću desnu granicu te zapamtimo rješenje ako je desnih granica u multisetu barem  $k$ . Ako smo naišli na desnu granicu nekog intervala, izbacujemo je iz multisetu ako se ona nalazi u njemu i nastavljamo dalje s algoritmom.

Ukupna složenost algoritma je  $O(n \log k)$ , a detalje pogledajte u službenoj implementaciji.

## Zadatak Natjecanje

Pripremili: Jakov Celin, Fran Babić, Fran Škarica

Potrebno znanje: BFS, blossom algoritam, najveće težinsko sparivanje u grafu

Primjetimo da jedine informacije potrebne za dobivanje proizvoljnog rješenja su međusobne udaljenosti paketa i udaljenosti paketa od početnog polja. Sve udaljenosti pronalazimo BFS algoritmom pokrećući ga iz svakog paketa zasebno. Sada zadatak možemo promatrati na način da odabiremo neke pakete koje ćemo nositi zajedno u paru (dostavimo jedan, pa drugi i vratimo se na početno polje), a neke odvojeno. Prvi podzadatak rješavamo isprobavanjem svih takvih kombinacija te uzimajući najbolju.

Ostale podzadatke rješavamo dinamičkim programiranjem s bitmaskama. Primjetimo da za neku fiksnu masku  $mask$  želimo upariti 2 paketa ( $k^2$  prijelaza) ili dostaviti 1 paket i vratiti se ( $k$  prijelaza). Ovaj pristup rješava drugi podzadatak u složenosti  $O(2^k \cdot k^2)$ . Treći podzadatak rješavamo koristeći tvrdnju da će svaki paket u završnoj maski biti uparen s nekim drugim paketom ili samim sobom. Iz toga slijedi da pri uparivanju 2 paketa možemo fiksirati prvi neupareni paket u maski te ga probati upariti sa svim preostalim neuparenim paketima ( $k$  prijelaza po maski). Ovaj pristup rješava treći podzadatak u složenosti  $O(2^k \cdot k)$ .

Rješenje zadnjeg podzadatka zahtjeva znanje blossom algoritma i njegove modifikacije pronalaska maksimalnog težinskog sparivanja u grafu. Ovaj podzadatak je namijenjen za pripremu za najzahtjevnija natjecanja te se ostali sudionici natjecanja ne potiču na naknadno rješavanje ovog podzadatka. Rješenje ćemo svedeno opisati jer je jedini zahtjevan dio zadatka razumijevanje i implementacija težinskog blossom algoritma.

Umjesto pronalaska najmanjeg broja sekundi da se svi paketi dostave, rješenje zadatka trebamo "okrenuti naopako" te na početku sve pakete dostaviti neuparene. Zatim konstruiramo graf gdje su paketi  $i$  i  $j$  spojeni bridom težine broja sekundi koje će Dino uštedjeti ako ih upari i zajedno dostavi. Tada se zadatak svede na pronalazak maksimalnog težinskog sparivanja u potpunom grafu zato što je Dinin cilj uštedjeti što više vremena tj. u što manje sekundi dostaviti sve pakete. Za detalje, pogledajte službenu implementaciju i prethodno priložene linkove.

## Zadatak Rastući

Pripremio: Ivan Jambrešić

Potrebno znanje: dinamičko programiranje, metoda 2 pokazivača

Zadatak rješavamo dinamičkim programiranjem te konstrukcijom rješenja iz izračunatih stanja. Ovdje ćemo opisati službeno rješenje zbog njegove jednostavne implementacije.

Neka je  $l \leq r$ . Definirajmo  $suma(l, r)$  kao sumu vrijednosti u nizu između  $l$ -te i  $r$ -te pozicije, a  $dp(l, r)$  kao maksimalan broj intervala na koji možemo podijeliti sufix niza od  $l$ -te pozicije takav da za 1. interval  $[l, R]$  tog dijeljenja vrijedi  $R \geq r$ .

Tada je  $dp(l, n) = 1$ , a inače je  $dp(l, r) = \max(dp(l, r + 1), 1 + dp(r + 1, k))$  gdje  $k$  predstavlja najmanji



indeks takav da je  $\text{suma}(l, r) \leq \text{suma}(r + 1, k)$ . Takav  $k$  za fiksne  $l$  i  $r$  pisati ćemo kao  $k(l, r)$ . Naivan pronalazak  $k(l, r)$  u složenosti  $O(n)$  za fiksne  $l$  i  $r$  donosio je 70 bodova uz pravilnu konstrukciju zbog prevelike vremenske složenosti algoritma  $O(n^3)$ .

Preostaje nam samo efikasno pronaći  $k(l, r)$  za svaki  $l \leq r$ .

Primjetimo da za  $l \geq 2$  vrijedi  $k(l - 1, r) \geq k(l, r)$ . Ovo svojstvo nam omogućava da fiksiramo  $r$  i obilazimo lijeve granice  $l$  u padajućem poretku, povećavajući  $k$  po potrebi. Za fiksnu desnu granicu  $r$  ćemo napraviti  $O(n)$  operacija pa je ukupna složenost algoritma  $O(n^2)$ . Za detalje, pogledajte službenu implementaciju.

## Zadatak Tornjevi

Pripremio: Toni Brajko

Potrebno znanje: tournament stablo

Za svaki upit možemo izravno simulirati slaganje tornjeva. Prolazimo kroz kockice redom od  $i = l$  do  $i = r$  i svaku pokušavamo staviti na vrh nekog postojećeg tornja koji završava kockicom suprotne boje. Ako takav toranj postoji, nadovezujemo se na njega. Ako ne postoji, započinjemo novi toranj čija je početna boja upravo boja trenutne kockice. Nakon što obradimo sve kockice iz intervala, broj stvorenih tornjeva predstavlja traženi odgovor. Složenost ovog pristupa iznosi  $O(nq)$ , što je dovoljno za rješavanje drugog podzadatka.

Ako u cijelom nizu postoji najviše 20 plavih kockica, svaka promjena boje može se dogoditi samo na jednoj od tih pozicija. U upitu  $(l, r)$  relevantne su samo plave kockice u tom intervalu, a između njih se nalaze blokovi crvenih. Ideja slaganja ostaje ista kao u gore navedenom rješenju, samo što preskačemo velike crvene blokove. Složenost rješenja za treći podzadatak je  $O(n + 20q)$ .

Uvedimo vrijednosti tako da je crvena kockica  $+1$ , a plava  $-1$ , i neka je  $p_i$  prefiksna suma tih vrijednosti do pozicije  $i$ .

Za zadani upit  $[l, r]$  razlika  $p_j - p_{i-1}$  za  $l \leq i \leq j \leq r$  predstavlja koliko je u kockicama od  $i$  do  $j$  više crvenih nego plavih, odnosno obrnuto ako je vrijednost negativna. Ključna tvrdnja je da minimalan broj tornjeva za interval  $[l, r]$  iznosi

$$\max_{l \leq i \leq j \leq r} |p_j - p_{i-1}|.$$

Intuitivno, dok prolazimo interval, svaki mogući “višak” crvenih nad plavima ili plavih nad crvenima opisuje trenutak kada se započinje novi toranj. Najgori takav višak odgovara najvećoj pozitivnoj sumi nekog podniza (ako je višak crvenih) ili najvećoj negativnoj sumi podniza (ako je višak plavih). Minimalan broj tornjeva upravo je najveća apsolutna vrijednost koju neka od tih suma može postići. Pronalaženje podniza s najvećom apsolutnom sumom možemo riješiti tournament stablom u složenosti  $O(n + q \log n)$ .

Dodatno, gornji izraz možemo pojednostaviti na

$$\max_{l-1 \leq i \leq r} p_i - \min_{l-1 \leq i \leq r} p_i,$$

što izravno daje odgovor na upit. Ova formula trivijalno vrijedi ako je indeks maksimuma veći od indeksa minimuma. Primijetimo da, u suprotnom slučaju, maksimalnu apsolutnu vrijednost poprima niz s minimalnom (negativnom) sumom, koju također dobivamo istom formulom. Zbog toga nije potrebno izravno tražiti podniz s najvećom apsolutnom sumom, već je dovoljno odvojeno pronaći najveću i najmanju prefiksnu sumu.

Za učinkovito dobivanje minimuma i maksimuma možemo koristiti tournament stablo ili RMQ tablicu (eng. *range minimum query*). Ukupna složenost je  $O(n + q \log n)$  ili  $O(n \log n + q)$ , ovisno o odabranoj implementaciji.