



## Opisi algoritama

Zadatke, testne primjere i rješenja pripremili: Nikola Vujica, Fran Škarica, Sofija Velkovska i Jakov Celin.

Primjeri implementiranih rješenja dani su u priloženim izvornim kodovima.

### Zadatak Noge

Pripremila: Sofija Velkovska

Potrebno znanje: naredbe učitavanja i ispisivanja

Svaki od  $n$  pasa ima 4 noge, pa svi psi zajedno imaju  $4 \cdot n$  nogu. Fran ima dvije noge. Dakle, sva živa bića u dvorištu imaju ukupno  $4 \cdot n + 2$  nogu. Da bismo riješili zadatak, dovoljno je učitati broj  $n$  i ispisati  $4 \cdot n + 2$ .

*Programski kod (pisan u Python 3):*

```
n = int(input())  
print(4 * n + 2)
```

### Zadatak Put

Pripremio: Fran Škarica

Potrebno znanje: naredba odlučivanja (if)

Usporedimo zbroj cijene karte busa i večere s cijenom karte aviona.

Ako je zbroj manji ili jednak, ispisujemo "bus" i razliku između cijene karte aviona i zbroja cijene karte busa i večere. Inače ispisujemo "avion" i razliku između zbroja cijene karte busa i večere i cijene karte aviona.

*Programski kod (pisan u Python 3):*

```
karta = int(input())  
vecera = int(input())  
avion = int(input())  
  
bus = karta + vecera  
  
if bus > avion:  
    print("avion")  
    print(bus - avion)  
else:  
    print("bus")  
    print(avion - bus)
```



## Zadatak Matcha

Pripremila: Sofija Velkovska

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

Potrebno je ponoviti sljedeći postupak  $n$  puta: učitati tri broja  $x$ ,  $y$  i  $z$ , saznati tko je pobijedio, ispisati ime pobjednika i koliko je matche popio. Imamo 4 mogućnosti:

- $y = -1$  i  $z = -1$ : Lana i Ivor nisu ništa rekli, pa nisu popili više od Maše, pa je Maša popila najviše. Popila je  $x$  matcha čajeva.
- $y = -1$  i  $z \neq -1$ : Lana nije popila više od Maše, ali Ivor je rekao da je popio  $z$  više od djevojke koja je zadnja govorila, a to je bila Maša. Ivor je popio više od Maše, a Maša je popila više od Lane, pa je Ivor popio najviše - popio je  $x + z$  matcha čajeva.
- $y \neq -1$  i  $z = -1$ : Lana je popila  $y$  više od Maše, a Ivor nije popio više od Lane. Lana je popila najviše, ukupno  $x + y$ .
- $y \neq -1$  i  $z \neq -1$ : Lana je popila  $y$  više od Maše, a Ivor je popio  $z$  više od Lane. Dakle, Ivor je popio najviše, ukupno  $x + y + z$ .

*Programski kod (pisan u Pythonu 3):*

```
n = int(input())
for i in range(n):
    x, y, z = map(int, input().split())
    if y == -1 and z == -1:
        print("Masa", x)
    if y == -1 and z != -1:
        print("Ivor", x + z)
    if y != -1 and z == -1:
        print("Lana", x + y)
    if y != -1 and z != -1:
        print("Ivor", x + y + z)
```

## Zadatak Zombie Apocalypse

Pripremila: Sofija Velkovska

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

Postoji više načina za rješavanje ovog zadatka, no ovdje ćemo opisati jedan od njih. Za svakog zombija posebno određujemo hoće li uspjeti stići do grada te brojimo koliko je zombija u tome uspjelo. Za fiksno zombija, odnosno  $i$ -tog zombija, redom provjeravamo svaku bombu i utvrđujemo može li ga ta bomba uništiti.

Puni domet  $j$ -te bombe opisan je intervalom  $[x_j - r_j, x_j + r_j]$ . Ipak, treba uzeti u obzir da bomba može djelovati samo unutar granica puta. Drugim riječima, bomba uništava samo zombije koji se nalaze na pozicijama između 1 i  $n$ , uključivo. Zbog toga promatramo interval  $[\max\{x_j - r_j, 1\}, \min\{x_j + r_j, n\}]$ . Na taj način izbjegnemo razmatranje slučajeva u kojima bi bomba djelovala izvan puta, odnosno na zombije koji su već stigli do grada ili se još nalaze u špilji.

Budući da bomba  $j$  eksplodira u trenutku  $t_j$ , poziciju na kojoj se  $i$ -ti zombi nalazi u tom trenutku možemo izračunati pomoću formule  $t_j - i + 1$ . Zatim provjeravamo nalazi li se zombi u dosegu bombe u tom trenutku. Ako se to dogodi za barem jednu od  $k$  bombi, zombi neće stići do grada.



## Zadatak Tomahawk

Pripremila: Jakov Ceiln

Potrebno znanje: ad-hoc

Običnom simulacijom pečenja pomoću for petlji rješavamo prvi podzadatak. Drugi podzadatak rješavamo koristeći tvrdnju da svako pečenje povećava cijele retke ili cijele stupce te pamteći za svaki redak ili stupac za koliko ih moramo povećati. Na kraju matricu konstruiramo pomoću ta 2 niza i odredimo najveću i najmanju temperaturu u matrici. Ovisno o implementaciji, pomoću prethodno opisanog rješenja možemo riješiti i treći podzadatak.

Za riješiti posljednja dva podzadatka, potrebno je primijetiti da se najveća temperatura u matrici mora nalaziti u donjem lijevom ili donjem desnom kutu matrice. Slično zaključujemo za najmanju temperaturu matrice ovisno o parnosti dimenzije  $n$  matrice. Ako je  $n$  paran, tada se najmanja temperatura u matrici nalazi se među srednja dva stupca prvog retka. Ako je  $n$  neparan, najmanja temperatura u matrici nalazi se među srednja 3 stupca prvog retka matrice.

Prethodno napisane tvrdnje lagane su za dokazati te njihove dokaze ostavljamo čitatelju za vježbu.

Nakon što znamo sve kandidate za najveću i najmanju temperaturu u matrici, potrebno je za svakog od kandidata odrediti njegovu temperaturu. Prolaskom po popisu pečenja te dodavajući temperaturu ovisno o udaljenosti polja od nekog ruba matrice možemo pronaći točnu temperaturu nekog kandidata. Složenost ovog pristupa je  $O(q)$  jer za  $O(1)$  kandidata prolazimo po popisu pečenja.

## Zadatak Magija

Pripremio: Jakov Celin

Potrebno znanje: union find, binary search, hashiranje, tournament stablo

Primjetio da ako je Ivor naučio zamijeniti 2 disjunktna intervala jednakih duljina, da je on zapravo naučio zamijeniti poziciju  $l$  sa  $r$ ,  $l + 1$  sa  $r + 1$ ... Umjesto mijenjanja, spojiti ćemo sve takve parove pozicija. Sada je odgovor na pitanje o osobi  $x$  zapravo minimum i maksimum komponente u kojoj se  $x$  nalazi. Obična implementacija ovog algoritma pomoću DFS-a ili BFS-a rješenja su prva 2 podzadatka u složenosti  $O(nq^2)$ .

Ako parove pozicija spajamo samo kada se one nalaze u različitim komponentama, zadatak je moguće riješiti pomoću union find strukture i održavanja minimuma i maksimuma komponente u složenosti  $O(nq)$  te ovaj algoritam postiže 73 boda na zadaku (ili 56 ovisno o implementaciji). Jedini nedostatak ovog algoritma jest prolazak po  $O(nq)$  parova pozicija kako bi sveukupno napravili  $O(n)$  spajanja u najgorem slučaju izvršavanja.

Ideja za postizanje svih bodova na zadatku jest pronaći samo "bitne" parove pozicija tj. one koje se nalaze u različitim komponentama grafa kako bi samo njih spajali. Provjerimo postoji li "bitan" par ako promatramo interval  $[l, l + x]$  te  $[r, r + x]$ . Neka  $d_i$  predstavlja reprezentanta komponente osobe  $i$  u grafu. Ako su vrijednosti u intervalima u nizu  $d$  identične, tada ne postoji bitan par među ta 2 intervala. Uspoređivanje jesu li vrijednosti u intervalima niza  $d$  jednake radimo pomoću hashiranja u  $O(1)$ .

Sada možemo pronaći prvi "bitan" par među intervalima pomoću binary searcha, no spajanje čvorova grafa malo je zahtjevnije zbog potrebe za održavanjem niza  $d$ . Primjetimo da pri spajanju manje komponente na veću u union findu možemo proći po svim čvorovima manje komponente te promijeniti njihove reprezentante u nizu  $d$ . Zbog potrebe za upitima hash vrijednosti niza  $d$  na intervalima te promjenama u nizu  $d$ , struktura koju trebamo koristiti je tournament stablo ili fenwick stablo. Nakon spajanja prvog bitnog para, postupak nastavljamo sve dok se među intervalima ne nalazi nijedan bitan par.

Ukupna vremenska složenost algoritma je  $O(n \log^2 n)$ , a memorijska  $O(n)$ .



## Zadatak Sladoled

Pripremio: Jakov Celin

Potrebno znanje: dinamičko programiranje, bitset, optimizacije koda

Neka je  $dp(i, j)$  jednak 1 ako je moguće napraviti kombinaciju s vrijednosti  $j$  na štandu  $i$ , u suprotnom je jednak 0. Tada dodavanje sladoleda s okusom  $k$  na štandu  $i$  možemo opisati prolaskom vrijednosti kombinacije  $tren$  koji prolazi od  $k$  do maksimalne vrijednosti  $M$  (u zadatku je  $M = 50000$ ) te prijelazom  $dp(i, tren) = dp(i, tren - k)$ . Za ispis broja različitih kombinacija samo trebamo prebrojati koliko 1 se nalazi u nizu  $dp(i)$ . Ovaj pristup rješava prva 2 podzadatka te je vremenska složenost algoritma  $O(MQ)$ .

Za potpuno rješenje zadatka potrebno je zamisliti niz  $dp(i)$  kao binaran broj. Tada prijelaz možemo zapisati kao  $dp(i) = (dp(i) \ll k)$  te navedeni prijelaz raditi sve dok se niz ne prestane mijenjati (ovaj prijelaz ekvivalentan je jednom dodavanju kuglice, a kako ih dodajemo beskonačno, prijelaz radimo sve dok se niz ne prestane mijenjati). Struktura podataka koja nam omogućava ovakve prijelaze te promatranje niza kao binarnog broja naziva se bitset.

Bez obzira na postojanje ove strukture, naš kod još uvijek nije dovoljno brz za postizanje svih bodova na zadatku te ga je potrebno ubrzati raznim optimizacijama. Umjesto uzastopnog ponavljanja dodavanje jedne kuglice dok se niz ne prestane mijenjati, dodati ćemo jednu kuglicu, dvije, četiri, osam... Ovu optimizaciju možemo iskoristiti jer se svaki broj kuglica  $k$  može zapisati pomoću nekih potencija broja 2 pomnoženih s  $k$  tj. pokazali smo da su prethodni načini dodavanja kuglica ekvivalentni. Ovo rješenje puno je brže od prethodnoga, no još uvijek nije dovoljno za postići sve bodove na zadatku.

Završna optimizacija za postizanje svih bodova bila je prestati s postupkom dodavanja kuglica ako želimo dodati kuglicu  $x$  u štand  $i$ , a na njemu se već mogla konstruirati kombinacija s vrijednosti  $x$ . Primjetimo da ako smo mogli konstruirati kombinaciju s vrijednosti  $x$ , tada možemo i konstruirati sve kombinacije s vrijednostima višekratnika od  $x$  tj. dodavanjem kuglice  $x$  se niz ne bi promijenio.

Ukupna vremenska složenost opisanog algoritma je  $O\left(\frac{nM \log M}{64}\right)$ , a memorijska  $O\left(\frac{nM}{64}\right)$ . Za implementacijske detalje pogledajte priloženo službeno rješenje.

## Zadatak Tjelesni

Pripremio: Nikola Vujica

Potrebno znanje: tournament stablo

Za prvi podzadatak bilo je dovoljno odsimulirati proces opisan u zadatku uzimajući sve brojeve iz intervala  $[l, r]$  te postavljajući ih na odgovarajuće pozicije nakon sortiranja. Složenost simulacije je  $O(nq \log n)$

U drugom podzadatku bilo je potrebno primijetiti da nakon operacije s najvećom desnom granicom  $r_{max}$  nijedna od ostalih operacija neće promijeniti izgled niza. Isto tako, promjene izvršene prije operacije s  $r_{max}$  neće biti vidljive u konačnom nizu. Stoga, možemo samo odsimulirati operaciju s  $r_{max}$  i očitati na kojoj se poziciji nalazi visina  $m$ . Složenost simulacije jedne operacije je  $O(n \log n)$

Od trećeg podzadatka nadalje, nećemo više pratiti poziciju svake visine, već samo poziciju visine  $m$ . Razmotrimo što će se dogoditi s visinom  $m$  tijekom  $i$ -te operacije u slučaju  $m = 1$ . Ako je trenutna pozicija visine 1 izvan intervala  $[l_i, r_i]$ , pozicija se neće promijeniti, a ako je visina 1 unutar intervala  $[l_i, r_i]$  nakon operacije naći će se na poziciji  $l_i$  jer je sigurno najmanja u intervalu. U slučaju  $m = n$  postupak je sličan, samo što će se visina  $n$  nakon operacije naći na poziciji  $l_i + 1$  jer je sigurno najveća u intervalu. Iznimka je slučaj  $l_i = r_i$ . Složenost ovog rješenja je  $O(n + q)$ .

Za preostale podzadatke primijetimo da nam je za svaki broj u nizu važno samo je li on manji ili veći od  $m$ . Označimo s 0 sve brojeve manje od  $m$ , a s 1 sve brojeve veće od  $m$ . Informacija o trenutnom razmještanju nula i jedinica u nizu dovoljna nam je da bismo odredili kakav će biti njihov razmještaj nakon operacije: na prvu poziciju u intervalu doći će 0 (uz uvjet da prije operacije postoji barem jedna nula u intervalu), na drugu poziciju doći će 1, na treću opet 0 itd. Nakon neke pozicije sve će vrijednosti biti iste,



ovisno o broju nula i jedinica u intervalu prije operacije. Ostaje nam odlučiti kako ćemo označiti broj  $m$ . Jedna opcija je da ga označimo brojem 2. No, kako bismo izbjegli previše slučajeva, elegantno rješenje je označiti brojeve **veće ili jednake**  $m + 1$ , odsimulirati postupak na opisani način, promijeniti oznake tako da sada s 1 budu označeni brojevi veći ili jednaki  $m + 1$ , zatim napraviti isti postupak još jedanput i pronaći poziciju na kojoj se konačni nizovi razlikuju. Takva pozicija bit će samo jedna i to će biti pozicija broja  $m$ . Složenost ovog postupka je  $O(nq)$ .

Za cijelo rješenje potrebno je optimizirati ovaj postupak korištenjem tournament stabla. Operacije koje struktura treba podržavati su računanje sume intervala te postavljanje brojeva u intervalu na određene vrijednosti. Postoji više različitih implementacija, no opisat ćemo jednu u kojoj koristimo odvojena tournament stabla za parne i neparne pozicije. Na početku ćemo postaviti nule i jedinice u listove stabala. Zatim ćemo za svaku operaciju prvo izračunati sumu u intervalu  $[l_i, r_i]$  (suma u intervalu odgovara broju jedinica u intervalu), a zatim ažurirati brojeve u intervalu u oba stabla. Imat ćemo nekoliko slučajeva u ovisnosti o parnosti pozicije  $l_i$  te broju jedinica u odnosu na duljinu intervala. Za implementacijske detalje pogledajte priloženo službeno rješenje. Složenost rješenja je  $O(n + q \log n)$ .