



Opisi algoritama

Zadatke, testne primjere i rješenja pripremili: Fran Babić, Toni Brajko, Jakov Celin, Dorijan Lendvaj i Patrik Pavić.

Primjeri implementiranih rješenja dani su u priloženim izvornim kodovima.

Zadatak Osijek

Pripremio: Toni Brajko

Potrebno znanje: naredba odlučivanja

U zadatku je potrebno izračunati ukupan broj izrađenih zadataka kao zbroj $A + B + C$. Ako je taj zbroj veći ili jednak 12, ispisujemo OCPC. Inače, potrebno je napraviti još $12 - (A + B + C)$ zadataka.

Programski kod (pisan u Python 3):

```
A = int(input())
B = int(input())
C = int(input())

if A + B + C >= 12:
    print("OCPC")
else:
    print(12 - (A + B + C))
```

Zadatak Svjetlo

Pripremio: Fran Babić

Potrebno znanje: naredbe ponavljanja i odlučivanja

Na samom početku znamo da je svijetlo ugašeno, zatim promatranjem stanja svijetla nakon svakog okretanja prekidača zaključujemo da će svijetlo ostati upaljeno ako je N neparan i obratno ako je N paran. Kada je n paran, potrebno je zbrojiti razlike svaka druga dva uzastopna unesena broja. Jednostavnije, možemo svaki neparni broj oduzeti i svaki parni broj zbrojiti na rješenje. Detalje možete vidjeti u službenoj implementaciji.

Programski kod (pisan u Python 3):

```
n = int(input())

s = 0
for i in range(n):
    x = int(input())
    if i % 2 == 0:
        s -= x
    else:
        s += x

if n % 2 == 0:
    print(s)
else:
    print("UPALJENO")
```



Zadatak Wow

Pripremio: Dorijan Lendvaj

Potrebno znanje: stringovi

Radi jednostavnije implementacije, moguće je primijetiti da je dovoljno promatrati samo prvi red ulaza za određivanje svakog slova. Poznato je da će prvi znak gornjeg reda nekog slova sigurno biti jednak '\'. Ako je riječ o slovu 'w', onda će 5. slovo gornjeg reda biti jednako '\ ' (ako ono postoji). U suprotnome, odgovarajuće slovo je 'v'. Shodno s odlukom koje je trenutno odgovarajuće slovo, pomičemo *pokazivač* za odgovarajući pomak. Detalje možete pogledati u službenoj implementaciji rješenja.

Zadatak Zid

Pripremio: Patrik Pavić

Potrebno znanje: prefiks sume, tehnika dvaju pokazivača

Fiksirajmo gornji i donji redak pravokutnika, l i r . Neka je x_j broj zabijenih čavla u j -tom stupcu ako promatramo samo retke od l do r . Vrijednost x_j možemo izračunati pomoću prefiks suma. Konkretno, ako označimo s p_{ij} broj čavala j -tom stupcu i svim retcima $\leq i$, tada vrijedi

$$x_j = p_{rj} - p_{(l-1)j}$$

Ako su lijevi i desni stupac slike a i b , tada se u pravokutniku $[l, r] \times [a, b]$ može nalaziti najviše jedan čavao. Ovaj uvjet ekvivalentan je izrazu

$$\sum_{j=a}^b x_j \leq 1$$

Dakle, trebamo prebrojati koliko postoji podnizova niza x_j takvih da im je suma najviše 1. Ovo možemo izračunati tehnikom dvaju pokazivača (*engl. two pointers*) u složenosti $O(m)$.

Ukupna složenost algoritma je $\mathcal{O}(n^2 \cdot m)$.

Zadatak Tornjevi

Pripremio: Fran Babić

Potrebno znanje: svojstva funkcije najvećeg zajedničkog djelitelja, ad-hoc

Za neki skup prirodnih brojeva S možemo primijetiti da vrijedi $\gcd(S) \leq \min(S)$ tj. $\gcd(S) \leq m$ za svaki $m \in S$. Stoga, za svaki $l \leq i \leq r$ vrijedi $\gcd(h_l, h_{l+1}, \dots, h_r) \leq h_i$ i time zaključujemo da je h_i dobar u nekom intervalu ako i samo ako dijeli sve brojeve u tom intervalu. Možemo onda primijetiti da je i -ti toranj dobar u intervalu $[l, r]$, gdje $l \leq i \leq r$, ako i samo ako je dobar u intervalima $[l, i]$ i $[i, r]$.

Za neki član niza s indeksom i , dovoljno je nezavisno odrediti najmanju lijevu granicu l_i koja zadovoljava uvjet da je toranj i dobar u intervalu $[l_i, i]$ i najveću desnu granicu r_i koja zadovoljava sličan uvjet. S obzirom na to da su ta dva problema simetrična, opisujemo algoritam za određivanje granice l_i .

Ideja algoritma je održavanja *indeksa promjene* tj. skup indeksa t takvih da $\gcd(h_t, h_{t+1}, \dots, h_i) \neq \gcd(h_{t-1}, h_t, \dots, h_i)$. Zato što vrijedi da $\gcd(h_l, \dots, h_r)$ dijeli $\gcd(h_{l+1}, \dots, h_r)$ (analogno, $\gcd(h_l, \dots, h_r)$ dijeli $\gcd(h_l, \dots, h_{r-1})$), veličina tog skupa može najviše biti $\log_2 h_i$ jer nakon svakog indeksa promjene najveći zajednički djelitelj će biti barem dvostruko manji. Također, lako vidimo da će skup indeksa promjene za indeks $i + 1$ biti podskup skupa indeksa promjene za i (do na indeks $i + 1$). Stoga, dovoljno je redom održavati taj skup promjena za svaki i i najveći indeks u tom skupu će biti tražena granica l_i . Složenost ovako opisanog algoritma je $O(n \log^2 n)$. Detalje implementacije može pogledati u priloženom izvornom kodu.



Zadatak Crtež

Pripremio: Jakov Celin

Potrebno znanje: tournament, brzo potenciranje, sažimanje

Autoru se potkrala greška pri pisanju teksta zadatka Crtež. Naime, autor je htio postaviti zadatak gdje se u drugom slučaju s bojanjem prestaje kada se naiđe na vrijednost **različitu** od 0. Testni primjeri su rađeni u skladu s postavkom zadatka bez greške u tekstu pa su zbog navedenih razloga svi natjecatelji dobili 0 bodova na ovom zadatku. Svim natjecateljima se ispričavamo na propustu.

U nastavku je opis rješenja zadatka gdje pri drugom slučaju bojanja prestajemo kada se naiđe na vrijednost različitu od 0.

Promatrajmo proizvoljan niz nakon kojega smo završili s operacijama bojanja.

Opservacija 1. Svaka boja $X > 0$ će prekrivati neki interval pozicija u završnom nizu.

Dokaz. Primjetimo da svako bojanje u boju $X > 0$ djeluje na intervalu pozicija niza čije su vrijednosti 0. Bojanje u boju -1 djeluje na susjednim pozicijama u nizu čije su vrijednosti 0 pa direktno slijedi opservacija 1.

Opservacija 2. Svaka igra je ekvivalentna s igrom koju dobijemo kada sortiramo intervale po lijevoj granici te im vrijednosti zamijenimo s njihovim mjestom u nizu nakon sortiranja.

Dokaz. Sortirajmo intervale po njihovim denim/lijevim granicama. Svaka boja $X > 0$ koja prekriva neki interval niza preslikati će se u boju $Y > 0$ koja predstavlja na kojem je mjestu u sortiranom poretku interval obojan u boju X . Primjećujemo da će svaki Y biti sparen s jedinstvenim X -om tj. preslikavanje je bijektivno pa slijedi da su svi uvjeti ekvivalentnosti igara zadovoljeni.

Opservacija 3. Fiksirajmo završni niz nakon proizvoljnog broja operacija te ga zamijenimo ekvivalentnim nizom u opservaciji 2., do njega uvijek možemo doći prvo radeći sve operacije bojanja polja u -1 pa zatim radeći operacije na intervalima s lijeva nadesno.

Dokaz. Ako smo neko polje pobojali u -1 , iz uvjeta bojanja da je to polje prije operacije bilo 0 slijedi da nijedan interval boja većih od 0 ne sadržava to polje. Slijedi da operaciju bojanja polja u -1 možemo raditi u bilo kojem trenutku prije trenutnoga tj. odmah na početku izvođenja operacija bojanja. Bojanje intervala u boju $X > 0$ prestaje kada izađemo iz niza, naiđemo na vrijednost -1 ili na desnu granicu intervala kojeg smo već prije obojali. Bojamo li niz poretkom iz opservacije 3. slijedi da su sva navedena svojstva očuvana tj. da ćemo izvođenjem operacija doći do fiksiranog niza.

Opservacija 4. Ako je N broj slobodnih polja(s vrijednošću 0) u nizu prije bilo kojeg bojanja, tada je broj različitih igri koje je moguće odigrati uz proizvoljan broj operacija tako da među njima ne postoji par ekvivalentnih igri jednak 3^N .

Dokaz. Fiksirajmo broj pojavljivanja -1 među slobodnim poljima nakon proizvoljnog broja operacija gdje završavamo s igrom. Dvije igre s različitim brojem pojavljivanja -1 u završnim nizovima nisu nikada ekvivalentne pa možemo broj različitih igri promatrati odvojeno za svaki broj pojavljivanja -1 u nizu te ih posumirati. Neka je X broj pojavljivanja -1 u nizu, tada je broj načina da ih rasporedimo na N pozicija jednak $\binom{N}{X}$. Promatrajmo preostala polja (njih $N - X$). Iz opservacije 3. slijedi da možemo fiksirati koje su pozicije desne granice intervala te da je tada niz jedinstveno određen (bojanjem s lijeva nadesno). Tada je za fiksni broj pojavljivanja -1 u nizu broj različitih igri jednak $\binom{N}{X} \cdot 2^{N-X}$. Sumiranjem po broju pojavljivanja -1 direktno dobivamo formulu za binomni poučak

$$\sum_{X=0}^N \binom{N}{X} \cdot 2^{N-X} = (2+1)^N = 3^N$$

Iz prve četiri opservacije slijedi da nas u svakom trenutku zanima broj pojavljivanja 0 u nizu te da je potrebno ispisati $3^{(\text{broj pojavljivanja } 0 \text{ u nizu})}$.



Prvi podzadatak rješavamo izravnim prolaskom po pozicijama u nizu te mijenjajući 0 u -1 i obrnuto te brojeći koliko ima 0 u nizu. Vremenska složenost algoritma je $O(NQ)$, a memorijska $O(N)$.

Drugi podzadatak rješavamo koristeći tournament stablo s propagacijama te brzo potenciranje. U svakom čvoru tournament stabla pamtimo veličinu intervala koji taj čvor prekriva, broj 0 u njemu te trebamo li ga preokrenuti ili ne. Koristeći propagacije lako možemo preokrenuti sve 0 u -1 i sve -1 u 0 pomoću zapamćenih vrijednosti u svakom čvoru. Nakon obavljanja svake promjene u nizu, pomoću tournament stabla također možemo saznati broj 0 u cijelom nizu. Računanje vrijednosti $3^{(\text{broj } 0 \text{ u nizu})}$ ne smijemo obavljati naivno jer bi složenost koda ostala ista pa za to koristimo brzo potenciranje. Vremenska složenost algoritma je $O(Q \log N)$, a memorijska $O(N)$.

Ostvariti sve bodove na zadatku bilo je moguće koristeći napredniju strukturu podataka zvanu sparse tournament, no njeno znanje nije bilo potrebno. Procesirajmo prvo svih Q intervala pa ih sažmimo tj. podijelimo niz duljine N na $O(Q)$ intervala za koje će vrijediti da su vrijednosti svih pozicija sadržanih u nekom od $O(Q)$ intervala jednake. Sada možemo izgraditi tournament stablo te koristiti rješenje drugog podzadatka. Vremenska složenost algoritma je $O(Q \log Q + Q \log N)$, a memorijska $O(Q)$.

Zadatak Stablo II

Pripremili: Toni Brajko i Patrik Pavić

Potrebno znanje: union-find

Rješenja za ovaj zadatak postoji više, a ovdje ćemo opisati jedno od jednostavnijih.

Zadatak ćemo riješiti *unazad*. Bojat ćemo bridove bojama od k -te do prve, a nakon što neki brid obojimo, nećemo ga prebojavati. Ovim postupkom dobit ćemo konačno bojanje stabla.

Ključna ideja jest da na pojedinom putu ne obilazimo bridove koje smo već obojali, već samo one koje nismo. Želimo moći efikasno podržati dvije vrste operacija:

- obojiti brid
- za dani vrh na nekom putu pronaći sljedeći neobojeni brid

Ukorijenimo stablo u vrhu 1. Umjesto pronalaženja sljedećeg brida na nekom putu, dovoljno nam je moći odrediti koji je sljedeći neobojeni brid na putu od danog vrha do korijena. Za vrhove u i v iz upita pronađimo takve bridove e_u i e_v . Ako su oni različiti, obojimo onaj koji je na većoj udaljenosti od korijena. Postupak ponavljamo dok ne dobijemo isti brid za oba vrha. U tom slučaju, taj se brid ne nalazi na putu od u do v , a sve željene bridove smo već obojali.

Ovo možemo realizirati pomoću *union-find* strukture. Kada bojimo neki brid, spajamo vrhove koje taj brid povezuje u istu komponentu (stablo ne mijenjamo, već za svaki vrh održavamo informaciju o njegovoj komponenti). Ako za neki vrh u želimo pronaći sljedeći neobojeni brid, pronaći ćemo vrh v s najmanjom dubinom u istoj komponenti kao u . Traženi brid bit će onaj koji spaja v s njegovim pretkom. Znamo da v i njegov predak nisu u istoj komponenti, što znači da brid između njih nije obojan. Bojanjem tog brida, vrh s najmanjom dubinom u toj komponenti više neće biti v .

Svaki brid obojit ćemo najviše jednom, a kako je ukupan broj bridova $n - 1$, imat ćemo $O(n)$ spajanja i pretraživanja komponenti. Ukupna složenost algoritma je $O(n \cdot \alpha(n))$, gdje je $\alpha(n)$ konstanta složenosti *union-find* strukture.