



Opisi algoritama

Zadatke, testne primjere i rješenja pripremili: Fran Babić, Toni Brajko, Jakov Celin, Nikola Dmitrović i Petar Sruk

Primjeri implementiranih rješenja dani su u priloženim izvornim kodovima.

Zadatak Brojanje

Pripremio: Toni Brajko

Potrebno znanje: naredbe učitavanja i ispisivanja

Franov broj će biti n puta uvećan za y . Ukupno će se povećati za $n \cdot y$, a konačan broj bit će $x + n \cdot y$ (početni broj + povećanje).

Jakovljevi broj će biti n puta smanjen za y . Ukupno će se smanjiti za $n \cdot y$, a konačan broj bit će $x - n \cdot y$ (početni broj - smanjenje).

Programski kod (pisan u Python 3):

```
x = int(input())
y = int(input())
n = int(input())
```

```
print(x + n * y)
print(x - n * y)
```

Zadatak Clarkson

Pripremio: Nikola Dmitrović

Potrebno znanje: naredbe odlučivanja, traženje maksimuma/minimuma

Clarkson će, očito, prvoga dana prebaciti žito iz skladišta s najmanje žita u skladište s najviše žita. Količina žita u preostalom skladištu neće se promijeniti. Primijetimo da će drugoga dana skladište koje nije imalo ni najviše ni najmanje žita sigurno imati manje žita od onoga koje je na početku imalo najviše, budući da se količina žita u tome skladištu dodatno povećala. Stoga će količina žita koju prebacujemo drugoga dana biti jednaka količini u skladištu koje je na početku imalo sljedeću najmanju količinu žita.

Zaključujemo da će odgovor na prvo pitanje biti zbroj količina žita u dva skladišta s najmanje žita. Također, odmah dobivamo i odgovore na drugo i treće pitanje. Prvoga dana prebacit će se svo žito iz skladišta s najmanje žita u skladište s najviše žita, a zatim će se iz preostalog skladišta ponovno prebaciti žito u ono skladište koje je na početku imalo najviše žita.

Za određivanje zbroja količina žita u dva skladišta s najmanje žita možemo primijeniti *obrnut* pristup, tj. umjesto da tražimo dva najmanja broja i zbrajamo ih, možemo zbrojiti sva tri broja i oduzeti najveći, tj. maksimum među njima. Za traženje najmanjega, najvećega i preostalog skladišta možemo redom pronaći koji su najmanji i najveći, a zatim pronaći preostalo koristeći naredbe odlučivanja, kao što je prikazano u službenoj implementaciji. Čitatelju ostavljamo kao vježbu primjenu istog principa za traženje zbroja najmanja dva skladišta kako bi se implementacija pojednostavila.



Programski kod (pisan u Python 3):

```
X = int(input())
Y = int(input())
Z = int(input())

L = [X, Y, Z]

najmanji = L.index(min(L))
najveci = L.index(max(L))

print(X + Y + Z - max(L))

print(chr(65 + najmanji), chr(65 + najveci))

if najmanji == 0:
    if najveci == 1:
        drugi = 2
    else:
        drugi = 1
elif najmanji == 1:
    if najveci == 0:
        drugi = 2
    else:
        drugi = 0
elif najmanji == 2:
    if najveci == 0:
        drugi = 1
    else:
        drugi = 0

print(chr(65 + drugi), chr(65 + najveci))
```

Zadatak Autobus

Pripremio: Fran Babić

Potrebno znanje: naredbe ponavljanja, stringovi

Prije svega, potrebno je odvojiti sve linije *Zagreb-Graz* i *Graz-Wroclaw*. To se može jednostavnom usporedbom unesenog stringa sa stringom "Zagreb-Graz" ili "Graz-Wroclaw". Postoji i jednostavnije rješenje, dovoljno je samo usporediti prva slova tih stringova, zbog garancije u tekstu zadatka. Za izdvajanje potrebnih brojeva iz ulaza možemo pažljivo obraditi te podatke iz učitanih stringova tako što znamenke odvajamo od ostalih znakova. Također, moguće je koristiti posebne funkcije/metode poput `split` u Pythonu ili `scanf` u C++-u.

Nakon što smo podijelili linije, dovoljno je fiksirati jednu liniju iz Zagreba i jednu liniju iz Graza i izračunati vrijeme puta ako putujemo tim fiksiranim linijama. Za izračunati vrijeme puta, treba provjeriti stigne li gospodin Malnar u istom danu i time odrediti vrijeme provedeno u Grazu i na to dodati vrijeme trajanja obje linije. Za detalje, pogledajte službenu implementaciju.



Zadatak Karte

Pripremio: Jakov Celin

Potrebno znanje: pohlepni algoritmi, bitmaske

Drugu parcijalu rješavamo provjeravanjem svih mogućih kombinacija odabira crvenih i plavih karata te izravno računajući jačinu svakog špila. Složenost algoritma je $O(2^n \cdot 2^m \cdot n \cdot m)$.

Opservacija 1. Ako fiksiramo skup plavih karata koje će biti u špil, uzet ćemo samo one crvene karte koje grade više od X COMBO poteza s odabranim plavim kartama.

Dokaz Opservacije 1. je izrazito jednostavan pa ga ostavljamo čitatelju za vježbu.

Riješimo sada prvu parcijalu $Y = 0$. Zadatak možemo pojednostaviti: imamo N crvenih karata i u špil ćemo uzeti sve plave karte. Pronađite špil s najvećom jačinom. Prolaskom po crvenim kartama i korištenjem Opservacije 1. možemo lako odrediti jačinu najjačeg špila. Složenost algoritma je $O(n \cdot m)$ zbog upisa tablice karata.

Za preostale parcijale bilo je potrebno fiksirati svaku kombinaciju uzimanja plavih karata te proći po svakoj od N crvenih karata uz korištenje Opservacije 1.

Izravnim brojanjem COMBO poteza između svake crvene karte i izabranih plavih dobivali su se bodovi za treću parcijalu. Složenost algoritma je $O(n \cdot m \cdot 2^m)$.

Za rješenje svih parcijala, bilo je potrebno brojati COMBO poteze pomoću binarnih operacija. Za svaku crvenu kartu, skup svih plavih karata s kojima ona gradi COMBO predstavimo kao binarni broj C . Ako fiksiramo skup plavih karata te njega predstavimo kao binaran broj D , tada je za neku crvenu kartu broj COMBO poteza koje ona gradi s izabranim plavim kartama: broj jedinica u binarnom zapisu od $(C \& D)$.

Korištenjem standardnih funkcija za binarne brojeve (npr. `__builtin_popcount` u C++ ili `count` u Pythonu) složenost algoritma postaje $O(n \cdot 2^m)$ zbog sposobnosti računala da binarne operacije izvodi efikasno i brzo. Implementacijske detalje pogledati u priloženom rješenju.

Zadatak Bojanje

Pripremili: Toni Brajko i Fran Babić

Potrebno znanje: ad-hoc

Za prvi podzadatak dovoljno je primijetiti da se neka ćelija koja treba biti crvena može obojati ako se nalazi u retku ili stupcu koji treba biti u crveni u potpunosti, ako to nije istina, nije moguće obojati tu ćeliju. Stoga, dolazimo do nužnog i dovoljnog uvjeta: svaka ćelija mora biti u retku ili stupcu koji je u potpunosti treba biti obojan u crveno. Za konstrukciju bojanja, dovoljno je izabrati sve retke i stupce koje možemo obojati u crno.

Za preostale podzadatke potrebno je primijeniti *obrnuti* način razmišljanja tj. tražit ćemo poteze od zadnjeg prema prvom. Kada pronađemo neki potez, sve ćelije u tom retku/stupcu mogu biti proizvoljne boje. Za idući potez, biramo neki redak/stupac koji može biti u cijelosti obojan u neku od dvije boje, a da već nije izabran u ranijim koracima. Naivna implementacije tog algoritma se može napisati u složenosti $O(n^4)$, što može ostvariti bodove za drugi podzadatak. Za ostvariti sve bodove na zadatku, potrebno je ubrzati algoritam. Jedan od mogućih načina je korištenje reda (*engl. queue*) i držanja informacija o redovima i stupcima koji nam pomažu u određivanju možemo li obojati promatrani redak ili stupac. Složenost takvog algoritma je $O(n^2)$. Implementaciju ovakvog algoritma možete vidjeti u službenom rješenju.



Zadatak Stablo

Pripremio: Jakov Celin

Potrebno znanje: DFS, binarno podizanje

Funkciju $f(y)$ po definiciji gledamo kao:

Pomnoži vrijednost svakog čvora s pripadajućim koeficijentom (udaljenost od y) te ju pribroji u $f(y)$.

Prvu parcijalu riješavamo tako da sagradimo stablo za svaki upit te izravno izračunamo $f(y)$. Složenost algoritma je $O(n \cdot q)$.

Svaku operaciju na stablu možemo promatrati kao promjenu koeficijenata umjesto spajanja i prespajanja čvorova. Ova informacija nam je izrazito korisna jer su manipulacije koeficijenata puno jednostavnije od prespajanja čvorova i podstabala.

Tvrdnja 1. Da odgovorimo na bilo koji upit, moramo moći:

- Izračunati $f(y)$ za svaki čvor y .
- Povećati ili smanjiti koeficijente u podstablu nekog čvora.
- Pronaći dijete od y koje je predak od x .

Dokaz tvrdnje 1. Tvrdimo da su napisani uvjeti dovoljni da riješimo cijeli zadatak. Računanje funkcije uz promjenu u podstablu možemo prikazati kao niz postupaka:

1. Izračunaj $f(y)$.
2. Smanji koeficijente u podstablu od x za 1.
3. Pronađi čvor od y koji je predak od x te povećaj koeficijente u njegovom podstablu za 1.
4. Koeficijent od x smanji za 1.

Svaki od postupaka možemo izravno prikazati pomoću nekog od uvjeta pa je tvrdnja dokazana.

Ostaje još samo pokazati kako efikasno i brzo riješiti svaki uvjet:

- Uvjet povećanja/smanjenja koeficijenata u podstablu nekog čvora jednak je dodavanju/oduzimanju sume vrijednosti čvorova u podstablu. Neka je funkcija $g(y)$ jednaka sumi vrijednosti u podstablu od y . Vidimo da je:

$$g(y) = \text{vl}[y] + \sum g(x), \quad \text{par}[x] = y$$

- Računanje funkcije $f(y)$ se može izračunati na mnoge načine, no jedan od najjednostavnijih je:

$$f(y) = g(y) - \text{vl}[y] + \sum f(x), \quad \text{par}[x] = y$$

- Pronaći dijete od y koje je predak od x pronalazimo metodom binarnog podizanja. Opis navedenog algoritma nalazi se na [linku](#).

Računanje funkcija f i g može se lako napraviti DFS obilaskom stabla (ili prolaskom čvorova u obrnutom poretku) te korištenjem gore opisanih formula. Složenost algoritma je $O((n + q) \log n)$. Zadatak je bilo moguće riješiti u složenosti $O(n + q)$, no to ostavljamo čitatelju za vježbu.



Zadatak Procesor

Pripremili: Toni Brajko i Marin Kišić

Potrebno znanje: Tournament stablo

Nakon svakog dodavanja naivno možemo proći cijelim nizom iznova i u najviše $\sum x_i$ upita dobiti najmanji element. Ukupno ćemo postaviti najviše $8 \cdot 10^4$ upita i ostvariti 15 bodova.

Ako u jednom upitu dodamo $x_i > n$ elemenata, dovoljno nam je promatrati samo n najmanjih. Štoviše, u bilo kojem trenutku nam je dovoljno promatrati najmanjih n elemenata trenutnog niza. Zato možemo pamtit i samo njih i održavati poredak. Za svaki novi element treba nam n upita (ili $\log n$ s binarnim pretraživanjem) kako bismo održali njihov poredak. Ovime ćemo postaviti najviše $2 \cdot 10^4$ upita i ostvariti 35 bodova.

Promotrimo sljedeći problem: imamo $x = 2000$ elemenata u nizu i želimo u što manje upita dobiti $n = 40$ najmanjih. Struktura rješenja ovog problema slična je ciljanom rješenju zadatka. Niz možemo podijeliti u blokove veličine k i za svaki blok izračunati najmanji element u njemu. Za ovo nam inicijalno treba $x - k$ upita. Za dobiti najmanji element u cijelome nizu, treba nam $k - 1$ usporedba najmanjih elemenata za svaki blok. Nakon što smo pronašli najmanji element, znamo u kojem se bloku on nalazi i sve što trebamo je pronaći novi najmanji element u tom bloku. Za to nam je potrebno oko $\frac{x}{k}$ upita (veličina pojedinog bloka). Postupak ponavljamo n puta. Za $k = \sqrt{x}$ postićemo najviše 7000 upita. Primijetimo da u zadatku možemo primijeniti isto rješenje, samo nećemo iterirati po dosad nepostojećim blokovima (posljednji blok može biti nedovršen). Ovime ostvarujemo 75 bodova.

Konačno, primijetimo li da blokove možemo dodatno dijeliti na podbloke i ponavljati proces, vidimo da nad početnim nizom možemo izgraditi tournament stablo u $x - 1$ upita. U svaki čvor spremićemo najmanji element iz njegovog podniza. Analogno prošlom rješenju, neki čvor u stablu ćemo stvoriti tek kada stvorimo cijeli podniz koji on obuhvaća. U bilo kojem trenutku ćemo imati najviše $\log x$ čvorova za usporediti (to će biti čvorovi koji čine prefiks niza koji je stvoren do tog trenutka) i trebat će nam još $\log x$ usporedbi nakon što izbacimo najmanji element (njegovu vrijednost možemo postaviti na beskonačno). Ukupno nam treba najviše $x + n \cdot 2 \log x$ upita. Ovo je gornja granica, no izbacivanjem elemenata iz niza broj upita će se smanjivati, zbog čega je ukupan broj upita nešto manji.