



Opisi algoritama

Zadatke, testne primjere i rješenja pripremili: Vito Anić, Fran Babić, Toni Brajko i Petar Sruk.

Primjeri implementiranih rješenja dani su u priloženim izvornim kodovima.

Zadatak Vlak

Pripremio: Fran Babić

Potrebno znanje: cjelobrojno dijeljenje, zaokruživanje brojeva

Možemo primijetiti da ćemo biti najefikasniji kada u svakoj turi pošaljemo što je više moguće turista. Broj turista u jednoj turi je ograničen kapacitetima prvog i drugog vagona, što znači da u jednoj turi možemo poslati $a + b$ turista. Sada vidimo da će nam odgovor biti najmanji prirodni broj m za koji vrijedi $n \leq m(a + b)$.

Taj broj možemo pronaći simulacijom tj. iteriranjem `while` petljom po m , ali lako se vidi da je m jednak $\frac{n}{a + b}$ zaokružen na više ili $\frac{n + (a + b) - 1}{a + b}$ zaokružen na niže.

Programski kod (pisan u Python 3):

```
n = int(input())
a = int(input())
b = int(input())

print((n + a + b - 1) // (a + b))
```

Zadatak Kamen Škare

Pripremili: Fran Babić i Toni Brajko

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

Održavamo dva brojača, `magda` i `stjepan`, koji predstavljaju koliko je igara pojedina osoba pobijedila. Naredbom ponavljanja n puta učitavamo bačene znakove, a zatim naredbom odlučivanja određujemo tko je pobijedio pojedinu igru.

Postoji ukupno 9 različitih kombinacija ishoda (oba igrača neovisno biraju jedan od 3 znakova). Ako su bačeni znakovi isti, ne povećavamo nijedan brojač. U suprotnom, određujemo igre u kojima Magda pobjeđuje i povećavamo brojač `magda` za 1. Ako znakovi nisu isti, a Magda nije pobijedila, pobjednik je Stjepan pa povećavamo brojač `stjepan`.

Na kraju, ako je Magda pobijedila više ili jednako igara, ispisujemo "Magda", a inače ispisujemo "Stjepan". Ovo se lako provjerava usporedbom dvaju brojača pomoću naredbe odlučivanja.

Programski kod s komentarima (pisan u Python 3):

```
n = int(input()) # učitavanje broja igara

magda = 0 # inicijalizacija brojača za svakog igrača na 0
stjepan = 0

for i in range(n):
    m, s = input().split() # za svaku od n igara izvrši:
    # učitavanje bačenih znakova Magde i Stjepana, redom
    if m == s: # ako su bačeni znakovi jednaki,
        continue # igra je neriješena
```



```
elif m == 'K' and s == 'S':      # ako je igra jednaka jednoj od 3 igre u kojima Magda pobjeđuje,
magda += 1                      # povećaj brojač za 1
elif m == 'S' and s == 'P':
magda += 1
elif m == 'P' and s == 'K':
magda += 1
else:                           # inače, igra nije neriješena i Magda nije pobijedila
stjepan += 1                    # stoga je pobjednik ove igre Stjepan

if magda >= stjepan: # konačno, ako je Magda pobijedila više (ili jednako) igara,
print("Magda")      # ispiši "Magda"
else:               # inače,
print("Stjepan")    # ispiši "Stjepan"
```

Zadatak Šah

Pripremio: Fran Babić

Potrebno znanje: matrice (dvodimenzionalni nizovi)

Za ostvariti sve bodove na zadatku, dovoljno je bilo održavati skup napadnutih polja u matrici tj. dvodimenzionalnom nizu. Svaku vrstu figure ćemo rješavati na zaseban način.

Topove možemo rješavati tako što ćemo iterirati po svim ćelijama ploče koje dijele isti red kao što i taj top, zatim po svim ćelijama koje dijele isti stupac kao i taj top. To je bilo dovoljno za prvi podzadatak.

Za kraljice smo radili isto što smo radili i za topove, barem što se tiče redova i stupaca, samo što smo trebali primijeniti drugačiji pristup za dijagonale. Primijetimo da za polje (x, y) , neko polje (x', y') je na istoj dijagonali ako i samo ako $x + y = x' + y'$ ili $x - y = x' - y'$. Prolaskom po svim takvim poljima smo označili i sva napadnuta polja na dijagonalama. Time smo osvojili prva dva podzadatka.

Za konje ćemo proći sve moguće pomake tj. pozicije na koje ta figura može skočiti. Najjednostavnija karakterizacija skoka jest da apsolutna vrijednost umnoška pomaka u redovima i stupcima mora biti jednaka 2. Jedino na što je ovdje potrebno paziti su granice ploče tj. matrice.

Detalje implementacije možete vidjeti u priloženom rješenju.

Zadatak Benzinska

Pripremio: Toni Brajko

Potrebno znanje: sortiranje, set

Ključna ideja zadatka jest da za neki restoran ne moramo odmah odlučiti hoće li Mr. Malnar objedovati u njemu kada dođemo do njega, već to možemo napraviti i naknadno.

Na početku se nalazimo u $x = 0$ i na raspolaganju imamo D energije. To znači da možemo dosegnuti sve restorane i za koje vrijedi $x_i \leq D$. Ako je $D < X$, barem jedan od tih restorana moramo iskoristiti (u suprotnom će se Mr. Malnar onesvijestiti nakon $x = D$). Lako je pokazati da ćemo uvijek htjeti izabrati onaj restoran s najvećim y_i . To nas vodi na sljedeći algoritam:

1. Neka je $R = D$ trenutni "doseg", tj. najdalja pozicija do koje Mr. Malnar može doći s trenutnim odabirom restorana, i S skup svih neiskorištenih restorana koji su unutar dosega ($x_i \leq R$).
2. Ako je $R \geq X$, Mr. Malnar može doći do Čakovca s trenutnim odabirom restorana pa prekidamo algoritam.
3. Inače, Mr. Malnar ne može doći do Čakovca. Iz skupa S dosežnih restorana biramo onaj s najvećim y_i i mičemo ga iz skupa. Novi doseg je $R' = R + y_k$, gdje je k odabrani restoran.



4. U skup S dodajemo sve restorane do kojih možemo doći s novim dosegom ($R < x_i \leq R'$).
5. Ponavljamo korak 2.

Naivna implementacija ovog algoritma dovoljna je za rješavanje drugog podzadatka. Sve bodove ostvarujemo ako algoritam implementiramo pomoću **set**-a (ili prioritetskog reda) za brzo ubacivanje, izbacivanje i dohvaćanje najvećeg elementa. Kako bismo riješili dodavanje novih elemenata u skup, najprije sortiramo sve restorane prema vrijednosti x_i . Za detalje implementacije pogledajte priloženi izvorni kod.

Zadatak Xor

Pripremili: Fran Babić i Toni Brajko

Potrebno znanje: bitovne operacije, tehnika dvaju pokazivača

Prvi podzadatak može se riješiti naivnom implementacijom zadatka u $O(n^2)$.

Drugi podzadatak rješavamo tako da iteriramo po svakom mogućem zbroju dvaju elemenata iz niza. Neka smo fiksirali zbroj s ; želimo znati koliko parova (i, j) postoji takvih da je $a_i + a_j = s$. Umjesto prebrojavanja parova indeksa, fiksirajmo sve x i y takve da je $x + y = s$ i prebrojimo koliko postoji načina da sumiramo $a_i = x$ i $a_j = y$ u sumu s . Ako su x i y različiti, broj načina je $f_x \cdot f_y$ gdje je f_i broj elemenata s vrijednosti i u nizu a . Ako su x i y jednaki, broj načina je $\frac{f_i \cdot (f_i + 1)}{2}$. Primijetimo da suma s doprinosi rješenju ako i samo ako je nju moguće dobiti na neparan broj načina. Složenost ovog pristupa je $O(m^2)$ gdje je $m = \max_{1 \leq i \leq n} a_i$.

Za svaki bit neovisno računamo hoće li u rješenju on biti 0 ili 1. U nastavku slijedi opis algoritma za fiksni bit b .

Za svaki element niza a_i , svi bitovi na pozicijama većima od b su nebitni za računanje tog bita pa ih možemo maknuti. Suma dvaju elemenata x i y će imati 1 na bitu b ako:

- x i y na bitu b imaju istu vrijednost i $x + y \geq 2^b$,
- x i y na bitu b nemaju istu vrijednost i $x + y < 2^b$.

Elemente niza možemo podijeliti u dva skupa; elementi koji na bitu b imaju 0 i elementi koji imaju 1. Neka su to skupovi A i B , redom. Sada za pojedini skup možemo u $O(\text{veličina skupa})$ izračunati broj parova (x, y) iz tog skupa takvih da je $x + y \geq 2^b$. Ovo radimo metodom dvaju pokazivača na sortiranom skupu. Slično, u $O(n)$ možemo izračunati broj parova (x, y) iz različitih skupova takvih da je $x + y < 2^b$. Sada znamo koliko suma ima 1 na bitu b pa, ovisno o parnosti, znamo i vrijednost bita b u konačnom rješenju.

Složenost ovog pristupa je $O(n \cdot \log n \cdot \log m)$. Svaki bit računamo zasebno, i za svaki je potrebno sortirati skupove A i B .

Za sve bodove možemo izbjeći sporo sortiranje skupova A i B . Naime, ako imamo niz sortiran za bit b (ne promatrajući bitove veće od b), tada u $O(n)$ možemo dobiti sortirani niz za $b + 1$. Potrebno je samo elemente koji imaju 0 na bitu $b + 1$ staviti prije elemenata koji imaju 1. Skup A tada će biti neki prefiks, a skup B sufiks niza. Konačna složenost je $O(n \cdot \log m)$. Za detalje implementacije pogledajte priloženi izvorni kod.

Prvi podzadatak može se riješiti naivnom implementacijom zadatka u $O(n^2)$.

Drugi podzadatak rješavamo tako da iteriramo po svakom mogućem zbroju dvaju elemenata iz niza. Neka smo fiksirali zbroj s ; želimo znati koliko parova (i, j) postoji takvih da je $a_i + a_j = s$. Umjesto prebrojavanja parova indeksa, fiksirajmo sve x i y takve da je $x + y = s$ i prebrojimo koliko postoji načina da sumiramo $a_i = x$ i $a_j = y$ u sumu s . Ako su x i y različiti, broj načina je $f_x \cdot f_y$ gdje je f_i broj elemenata s vrijednosti i u nizu a . Ako su x i y jednaki, broj načina je $\frac{f_i \cdot (f_i + 1)}{2}$. Primijetimo da suma s doprinosi rješenju ako i samo ako je nju moguće dobiti na neparan broj načina. Složenost ovog pristupa je $O(m^2)$ gdje je $m = \max_{1 \leq i \leq n} a_i$.



Za svaki bit neovisno računamo hoće li u rješenju on biti 0 ili 1. U nastavku slijedi opis algoritma za fiksni bit b .

Za svaki element niza a_i , svi bitovi na pozicijama većima od b su nebitni za računanje tog bita pa ih možemo maknuti. Suma dvaju elemenata x i y će imati 1 na bitu b ako:

- x i y na bitu b imaju istu vrijednost i $x + y \geq 2^b$,
- x i y na bitu b nemaju istu vrijednost i $x + y < 2^b$.

Elemente niza možemo podijeliti u dva skupa; elementi koji na bitu b imaju 0 i elementi koji imaju 1. Neka su to skupovi A i B , redom. Sada za pojedini skup možemo u $O(\text{veličina skupa})$ izračunati broj parova (x, y) iz tog skupa takvih da je $x + y \geq 2^b$. Ovo radimo metodom dvaju pokazivača na sortiranom skupu. Slično, u $O(n)$ možemo izračunati broj parova (x, y) iz različitih skupova takvih da je $x + y < 2^b$. Sada znamo koliko suma ima 1 na bitu b pa, ovisno o parnosti, znamo i vrijednost bita b u konačnom rješenju.

Složenost ovog pristupa je $O(n \cdot \log n \cdot \log m)$. Svaki bit računamo zasebno, i za svaki je potrebno sortirati skupove A i B .

Za sve bodove možemo izbjeći sporo sortiranje skupova A i B . Naime, ako imamo niz sortiran za bit b (ne promatrajući bitove veće od b), tada u $O(n)$ možemo dobiti sortirani niz za $b + 1$. Potrebno je samo elemente koji imaju 0 na bitu $b + 1$ staviti prije elemenata koji imaju 1. Skup A tada će biti neki prefiks, a skup B sufiks niza. Konačna složenost je $O(n \cdot \log m)$. Za detalje implementacije pogledajte priloženi izvorni kod.

Zadatak Cipele

Pripremio: Vito Anić

Potrebno znanje: Pohlepni algoritmi (*greedy*), Logaritamska struktura (*Fenwick tree*) ili alternativno tournament (*Segment tree*)

Nažalost, službeno rješenje ovog zadatka nije ispravno, nakon natjecanja pronađen je protuprimjer za koji naše rješenje ne ispisuje najmanje moguće vrijeme.

U ovom slučaju jako je teško donijeti odluku što napraviti s zadatkom. Odlučili smo se ostaviti bodove za dvije parcijale za koje imaju ispravno rješenje. Rješenja tih parcijala ćemo opisati u ovom opisu.

Rješenje parcijale $n = m$ je sljedeće: Primijetimo da imamo dovoljno mjesta u hodniku da nam stanu sve cipele. Dakle, ako Lana treba neke cipele više od jedanput, drugi puta neće trošiti vrijeme na njihovo vađenje. Kad izvadi neke cipele iz ormara, vrijeme vađenja svih cipele koje se nalaze *dublje* od trenutačnih će se smanjiti za jedan. Dakle potrebno je vrijeme vađenja svih cipela na poziciji većoj od trenutačno izvađenih cipela smanjiti za jedan. Vremena vađenja cipela ćemo održavati u logaritamskoj strukturi. Kada neke cipele želimo izvaditi iz ormara prvo provjerimo koliko vremena nam treba za to, te onda na toj poziciji smanjimo logaritamsku za 1. Više o logaritamskoj možete saznati na ovoj [poveznici](#).

Rješenje parcijale $m = 0$ je sljedeće: Budući da nema mjesta u hodniku, Lana će cipele na kraju svakog dana morati vraćati u ormar. Pogledajmo što se desi s pozicijama cipela u ormaru. Pretpostavimo da smo izvadili neke cipele x iz ormara. Na kraju dana ih vraćamo na vrh ormara. Primijetimo da se sve cipele koje se nalaze između početka ormara i početne pozicije cipela x na početku dana će biti *dublje* u ormaru za 1 mjesto. Sve cipele koje su u početku bile *dublje*, će ostati na istoj poziciji. Slično kao i u prvoj ideji, koristit ćemo logaritamsku strukturu, no s još jednim malim trikom. Opet ćemo pamtiti koliko nam vremena treba da cipele izvučemo iz ormara, no umjesto da imamo samo n mjesta u logaritamskoj imat ćemo $n + q$ mjesta. Cipele će se na početku nalaziti na zadnjih n mjesta. Kad neke cipele izvučemo iz ormara, poziciju gdje se nalaze te cipele ćemo smanjiti za 1. Tim ćemo cipelama sada dodijeliti novu poziciju koju nisu imale dosad niti jedne druge cipele na početku ormara. Budući da imamo $n + q$ mjesta u ormaru ta će pozicija uvijek postojati. Na toj poziciji povećamo vrijednost za 1 i time smo dobili novo stane ormara nakon tog dana.



U moje ime te u ime cijelog znanstvenog povjerenstva ispričavam se na ovoj greški.

Zadatak Tura Mačkica

Pripremio: Petar Sruk

Potrebno znanje: Eulerov ciklus

Zadatak traži da odaberemo neke neusmjerene bridove i usmjerimo ih na takav način da postoji Eulerov ciklus zajedno s već usmjerenim bridovima. Eulerov ciklus na usmjerenom grafu postoji ako i samo ako svaki čvor ima jednak ulazni i izlazni stupanj i graf je povezan.

Primjetimo prvo da ako postoji cesta bez mačkice koja povezuje park samim sa sobom tu cestu nikad nećemo koristiti pa je neusmjeren dio grafa zapravo stablo. Nazovimo razliku ulaznog i izlaznog čvora balansom tog čvora. Prvo želimo osigurati da je balans svakog čvora jednak nula. Promotrite najdublje čvorove (s obzirom na neusmjereno stablo) s balansom nula. Kada bi dodali i usmjerili njihov jedini neusmjeren brid oni više nebi imali balans nula, pa te bridove ne smijemo dodati u Turu Mačkica. Najdubljim čvorovima koji nemaju balans nula moramo usmjeriti njihov jedini brid na način da nakon usmjerenja imaju balans nula. To je jedino moguće kada im je balans jednak 1 ili -1 . Sada kad svi jedinstveno odredili koje bridove i kako moramo usmjeriti isti postupak možemo ponoviti i na razini više te nastaviti sve dok ne dođemo do korijena.

Ako smo uspješno proveli cijeli postupak jednoznačno smo odredili koje bridove dodajemo i kako ih usmjerujemo, pa da saznamo postoji li Eulerov ciklus dovoljno je provjeriti da je graf s obzirom na usmjerene bridove, uključujući i dodane, povezan.

U slučaju da ne postoji cesta bez mačkice koja povezuje park samim sa sobom to je povezan graf s jednim ciklusom. Primjetite da takav graf možemo rastaviti na taj ciklus i podstabla koje sadrži točno jedan čvor iz ciklusa. Budući da nam u gornjem algoritmu balansiranja stabla trebaju samo balansi čvoreva, a ne i točne veze, taj algoritam možemo provesti i na svakom podstablu ovakvog grafa. Primjetite da u ovom slučaju nije nužno da nakon uspješno provedenog algoritma korijeni, odnosno čvorevi na ciklusu, imaju balans 0.

U slučaju da čvorevi na ciklusu imaju balans 0 možemo provjeriti jeli graf s obzirom na usmjerene bridove povezan. Ako nije možemo dodati sve bridove iz ciklusa u Eulerovu turu te ih proizvoljno cirkularno usmjeriti. Nakon ovog dodavanja balans svih čvoreva je i dalje 0, ali možda smo povezali graf s obzirom na usmjerene bridove, pa još jednom provjeravamo povezanost. Primjetite da održimo balans svih čvoreva 0 dodavanje novih bridova mora tvoriti ciklus tako da je ovo bila jedina mogućnost.

Ako ipak nakon provođenja algoritma balansiranja nisu svi čvorovi na ciklusu balansa 0 i njih moramo izbalansirati pomoću bridova na ciklusu. To radimo tako da izaberemo neki čvor na ciklusu koji nema balans 0 te dodamo jedan od njegova dva susjedna brida i usmjerimo ga na način da vrijednost balansa tog čvora približimo nuli. Nakon toga prebacujemo se na čvor koji je povezan s novododanim bridom te i njemu balans približimo nuli ili ako je već jednak nula odlučimo da nećemo dodati taj brid u Eulerov ciklus. Nakon što za svaki brid iz ciklusa odlučimo kako ćemo ga dodati i usmjeriti provjeravamo je li ciklus zaista izbalansiran te je li usmjeren dio grafa povezan. Primjetite da različit izbor među dva brida susjedna početnom čvoru mogu davati različite rezultate, ali da oba izbora jedinstveno određuju sve ostala dodavanja i usmjerenja, tako da moramo provjeriti obje mogućnosti i izabrati manju.

Ukupna složenost algoritma $O(n + m)$.