



Opisi algoritama

Zadatke, testne primjere i rješenja pripremili: Fran Babić, Toni Brajko, Jakov Celin, Nikola Dmitrović, Ivan Janjić, Josip Klepec i Martina Licul.

Primjeri implementiranih rješenja dani su u priloženim izvornim kodovima.

Zadatak Program

Pripremio: Nikola Dmitrović

Potrebno znanje: naredbe učitavanja i ispisivanja, naredba odlučivanja (if)

Ivica smije nastaviti gledati film samo ako ima jednako ili više godina od onog broja godina koji je čuo u upozorenju. Točnije, ako je X broj godina u upozorenju, a Y broj godina koje ima Ivica, tada Ivica smije nastaviti gledati film ako vrijedi da je Y jednak ili veći od X , tj. $Y \geq X$.

Programski kod (pisan u Python 3):

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

if Y >= X:
    print("DA")
else:
    print("NE")
```

Zadatak Slatkiši

Pripremio: Toni Brajko

Potrebno znanje: ad-hoc, naredbe ponavljanja

Učiteljica Ana želi da svi učenici imaju jednak broj bombona. Zato mora nekim učenicima (ali ne nužno svima) dodati određeni broj bombona. Imajte na umu da Ana smije samo dodavati bombone, ne može ih oduzimati.

Kako bismo to postigli, trebamo prvo pogledati koliko bombona ima učenik s najviše bombona. Označimo taj broj s M . Svaki učenik na kraju mora imati jednak broj bombona, a taj broj će biti upravo M . To znači da će Ana morati dodati bombone učenicima koji imaju manje od M .

Da bismo saznali koliko bombona Ana mora dodati, trebamo za svakog učenika izračunati koliko bombona manje ima od učenika s najviše bombona. To ćemo izračunati kao:

$$(M - s_1) + (M - s_2) + \dots + (M - s_n),$$

gdje su s_1, s_2, \dots, s_n brojevi bombona koje svaki učenik trenutno ima. Drugačije rečeno, ukupni broj bombona koje Ana mora dodati je jednak razlici između $n \cdot M$ i ukupnog broja bombona S koje učenici trenutno imaju.

$$\text{Ukupni broj dodanih bombona} = n \cdot M - S.$$

Zadatak Osobna

Pripremila: Martina Licul

Potrebno znanje: znakovna polja (string), naredbe ponavljanja

U zadatku je potrebno odrediti ime, prezime, datum rođenja i OIB osobe iz dana tri znakovna niza.



Ime i prezime možemo odrediti pomoću *while* petlji. Na početku postavimo prazan string `ime` i dodajemo mu znak po znak iz trećeg retka dok ne nađemo na znak `'<'`. Tada završavamo prvu *while* petlju, preskačemo dva znaka (jer se između imena i prezimena nalaze dva znaka `'<'`) i ponavljamo sličan postupak za odrediti prezime. Nakon što smo odredili ime i prezime, preostaje nam postaviti da je samo prvo slovo veliko: u Pythonu za to možemo iskoristiti metodu `capitalize`.

```
ime = ""
i = 0
while treci_redak[i] != '<':
    ime += treci_redak[i]
    i += 1

prezime = ""
i += 2 # preskačemo dva znaka '<'
while i < len(treci_redak) and treci_redak[i] != '<':
    prezime += treci_redak[i]
    i += 1

ime = ime.capitalize()
prezime = prezime.capitalize()
```

Datum rođenja možemo odrediti iz drugog retka, gdje se nalazi zapisan na prvih 6 pozicija (dvije za godinu, dvije za mjesec i dvije za dan). Kako bismo ispravno odredili godinu, potrebno je provjeriti čine li prve dvije znamenke retka broj manji ili jednak 24, jer u tom slučaju dodajemo 20 ispred broja, a inače 19.

```
godina = drugi_redak[0:2]
mjesec = drugi_redak[2:4]
dan = drugi_redak[4:6]

if int(godina) <= 24:
    godina = "20" + godina
else:
    godina = "19" + godina

datum = dan + "-" + mjesec + "-" + godina
```

OIB se nalazi u prvom retku i proteže se od 16. do 26. pozicije. Dovoljno je uzeti taj podniz i ispisati ga kao OIB. Pripazimo da je prva pozicija u nizu pozicija 0 (a ne 1), i da je prva granica uključena, dok je druga isključena kada u Pythonu uzimamo podniz pomoću pozicija.

```
oib = prvi_redak[15:26]
```

Na kraju je potrebno ispisati izvučene podatke u obliku opisanom u tekstu zadatka.

Zadatak Skokovi

Pripremio: Toni Brajko

Potrebno znanje: for petlja

Održavamo interval $[l, r]$ koji predstavlja najmanju i najveću vrijednost do koje možemo doći s trenutnih pozicija. Inicijalno postavljamo $l = r = a_0$, jer krećemo s pozicije a_0 .

Za svaku poziciju a_i (gdje je $i > 0$), provjeravamo je li a_i unutar intervala proširenog za K , tj. provjeravamo vrijedi li:

$$l - K \leq a_i \leq r + K.$$



Ako je taj uvjet ispunjen, znači da možemo doći do pozicije i . U tom slučaju, ažuriramo interval $[l, r]$ tako da uključuje vrijednost a_i , odnosno:

$$l = \min(l, a_i),$$

$$r = \max(r, a_i).$$

Ako a_i nije unutar tog intervala, ne možemo doći do pozicije i i interval ostaje nepromijenjen.

Rješenje prolazi kroz niz jedanput, pa je vremenska složenost $O(N)$.

Primjer (iz zadatka):

Za niz $a = [5, 4, 8, 7, 2]$ i $K = 2$, intervali će se ažurirati na sljedeći način:

- početni interval: $[5, 5]$
- pozicija $a_1 = 4$: unutar intervala $[5 - 2, 5 + 2] = [3, 7]$, pa ažuriramo interval na $[4, 5]$
- pozicija $a_2 = 8$: izvan intervala $[4 - 2, 5 + 2] = [2, 7]$, ne možemo doći do ove pozicije, interval ostaje $[4, 5]$
- pozicija $a_3 = 7$: unutar intervala $[4 - 2, 5 + 2] = [2, 7]$, pa ažuriramo interval na $[4, 7]$
- pozicija $a_4 = 2$: unutar intervala $[4 - 2, 7 + 2] = [2, 9]$, pa ažuriramo interval na $[2, 7]$

Na kraju ispisujemo koji su elementi dostižni: pozicije 0, 1, 3 i 4 su dostižne, dok pozicija 2 nije.

Zadatak Hijerarhija

Pripremio: Fran Babić

Potrebno znanje: teorija grafova, set

Primijetimo da, zbog uvjeta u ulaznim podacima, strukturu odnosa između zaposlenika možemo zamisliti kao stablo, kada bi veze bile neusmjereni bridovi između čvorova (zaposlenika). Međutim, zbog prirode ovog zadatka, ima više smisla promatrati dane veze kao usmjerene bridove. U tom slučaju, pitanje je li dana struktura odnosa valjana *hijerarhija* postaje ekvivalentno pitanju je li zamišljeni graf valjano usmjereno stablo. Usmjereno stablo mora imati svoj *korijen* i mora biti moguće doći iz korijena do svakog drugog čvora koristeći dane usmjerene veze.

Za prva dva podzadatka, dovoljno je bilo na početku i nakon svake promjene iterirati po svim čvorovima i provjeriti je li ijedan od njih korijen stabla. Složenost takvog algoritma je $O(N^2Q)$. Za treći i četvrti podzadatak dovoljno je primijetiti da u usmjerenom stablu samo korijen može (i mora) imati ulazni stupanj jednak 0, tj. mora postojati jedinstveni čvor koji nema nijednog sebi nadređenog. To nam smanjuje broj čvorova koje ćemo provjeravati i možemo poboljšati složenost algoritma na $O(NQ)$ uz spretnu implementaciju.

Konačno, za sve bodove potrebno je primijetiti da je graf pod uvjetima zadatka usmjereno stablo ako i samo ako vrijedi da postoji jedinstveni zaposlenik bez direktno nadređenih (čvor s ulaznim stupnjem jednakim nuli) i svi ostali zaposlenici imaju točno jednog nadređenog. Očito je da to zaista jest nužan uvjet. Dokaz da je to dovoljan uvjet ostavljamo čitatelju za vježbu. Algoritam koji rješava ovaj zadatak sada možemo implementirati u složenosti $O((N + Q) \log N)$. Detalje možete vidjeti u službenoj implementaciji.

Zadatak Učiteljica

Pripremio: Jakov Celin

Potrebno znanje: sweep line, segmentno stablo, formula uključivanja i isključivanja

Nazovimo interval *dobrim* ako vrijedi da postoji broj koji se pojavljuje točno jednom, broj koji se pojavljuje točno dvaput, ..., broj koji se pojavljuje točno K puta.



Za rješenje prvog podzadatka bilo je dovoljno fiksirati svaki interval, dodavati element jedan po jedan te zatim provjeriti je li fiksirani interval dobar. Složenost opisanog algoritma je $O(N^2K)$.

U drugome podzadatku elementi niza su brojevi između 1 i K . Primjećujemo da je interval dobar ako se svaki broj od 1 do K pojavljuje različito puta te ako je interval duljine $\frac{K(K+1)}{2}$. Preostaje provjeriti $O(N)$ intervala postupkom iz prvog podzadatka.

U trećem podzadatku želimo prebrojati intervale u kojima postoji element koji se pojavljuje točno jednom. Za svaki indeks i između 1 i N želimo nejednadžbama opisati kada se element na mjestu i pojavljuje samo jednom.

- Pronađimo najveći indeks L_i manji od i takav da je $a[L_i] = a[i]$ i najmanji indeks R_i veći od i takav da je $a[R_i] = a[i]$.
- Primjećujemo da će se element i pojavljivati u intervalu indeksa $[x, y]$ jednom samo ako je $L_i < x \leq i \leq y < R_i$.

Podzadatak se sada sveo na prebrojavanje parova indeksa x i y takvih da postoji barem 1 indeks i za koji je $L_i < x \leq i \leq y < R_i$. Rješenje ovog problema nalazimo promatrajući uvjete za svaki indeks kao pravokutnike u ravnini. Tada primećujemo da je broj valjanih intervala broj cjelobrojnih točaka koje su pokrivene barem jednim pravokutnikom tj. broj cjelobrojnih točaka koje su prekrivene unijom svih pravokutnika. Problem površine unije svih pravokutnika je poznat te se jedno od objašnjenja algoritma za pronalazak površine unije nalazi na [linku](#). Složenost algoritma je $O(N \log N)$.

U četvrtom podzadatku koristiti ćemo svojstva iz trećeg podzadatka te ćemo za svaki broj x između 1 i K pronaći pravokutnike takve da za svaki indeks i od 1 do N vrijedi da se element a_i pojavljuje točno x puta. Primjećujemo da je rješenje zadatka površina točaka prekrivenih barem jednim pravokutnikom svakog x -a. Prikažemo li sve točke prekrivene pravokutnicima za fiksirani x kao skup S_x , zapravo tražimo broj elemenata skupa $S_1 \cap S_2 \cap \dots \cap S_K$. Izravno računanje presjeka skupova izrazito je teško, no korištenjem formule uključivanja i isključivanja rješenje možemo izračunati pomoću unija skupova. Neka je S skup koji sadrži S_1, S_2, \dots, S_K tj. $S = \{S_1, S_2, \dots, S_K\}$. Prema formuli uključivanja i isključivanja, $S_1 \cap S_2 \cap \dots \cap S_K$ je

$$\sum_{Q \subseteq S} f(Q) \cdot (-1)^{|Q|}$$

gdje je $f(Q)$ površina unije svih pravokutnika u skupu Q . Složenost algoritma je $O(2^{K-1}KN \log N)$. Za implementacijske detalje pogledati priložen kod u rješenjima.

Zadatak Zbunjenost

Pripremio: Ivan Janjić

Potrebno znanje: dinamičko programiranje, teorija grafova

Osnovna opservacija je da je jednostavna zatvorena šetnja (nadalje ciklus) mnogokut čiji vrhovi odgovaraju podskupu vrhova čiji se relativan poredak nije promijenio. Ovo je bilo dovoljno za riješiti prvi podzadatak direktnom provjerom svakog podskupa. Složenost je $O(n2^n)$ iako su ograničenja dopuštala nešto sporije algoritme.

Promatrajući jedan ciklus može se primijetiti da je taj mnogokut također trianguliran te da trokuti na koje je podijeljen podskup trokuta na koje je podijeljen cijeli mnogokut. To motivira uvođenje sljedećeg grafa: vrhovi predstavljaju trokute na koje je podijeljen početni mnogokut a između dva vrha postoji brid ako trokuti koje ti vrhovi predstavljaju imaju zajedničku stranicu. Dvije ključne opservacije: uveden je graf stablo te svaki ciklus odgovara povezanoj komponenti u stablu. Skica dokaza: u triangulaciji uvijek postoji „uho“, to jest trokut kojem su vrhovi uzastopni na mnogokutu, on je povezan s točno jednim trokutom, maknemo ga iz triangulacije i induktivno nastavimo na ostatku. Povezanost je očito nužan uvjet te se može pokazati da svaka komponenta odgovara nekom ciklusu.



Zadatak se sada dijeli na dva dijela. Prvi je izgradnja stabla, a drugi je računanje broja povezanih komponentata. Stablo se može naivno izgraditi u $O(n^3)$ ili $O(n^2)$ tako da se pronadu svi trokut i za svaki par provjeri jesu li povezani. To je bilo dovoljno za drugi, odnosno treći podzadatak. Za puno rješenje trebalo je osmisliti pametniji algoritam u složenosti $O(n)$ ili $O(n \log n)$. Slijedi opis jednog takvog od mnogih: sortiramo dijagonale po duljini. Može se pokazati da će u svakom trenutku trenutna dijagonala biti dio jednog „uha“ ako ih tim redom mičemo. Vrhove redom čuvamo u dvostruko povezanoj listi te kod micanja svakog uha maknemo onaj vrh koji nije kraj dijagonale ažuriranjem liste. Za svaku dijagonalu zapamtimo koja dva trokuta ju dijele te ih spojimo. Za detalje implementacije pogledajte službeno rješenje.

Za računanje broja povezanih komponentata ukorijenimo stablo u proizvoljnom vrhu te računamo $dp(x)$ – broj povezanih komponenti koje sadrže x te eventualno neke čvorove u njegovom podstablu. Prijelaz izgleda ovako:

$$dp(x) = \prod_{y \text{ dijete od } x} (dp(y) + 1)$$

Iz podstabla nekog djeteta možemo ne uzeti ništa ili neku povezanu komponentu koja sadrži to dijete. Djeca su neovisna jedna o drugom pa sve te mogućnosti pomnožimo.

Konačan odgovor je $\sum_{x=1}^n dp(x)$ zato što svaka povezana komponenta ima jedinstveni vrh takav da su svi ostali vrhovi u njegovom podstablu.