2018 iz informatike Natjecanje

15. ožujka 2018.

OPISI ALGORITAMA

Agencija za odgoj i obrazovanje
Education and Teacher Training Agency

HRVATSKI SAVEZ INFORMATIČARA

Ministarstvo znanosti, obrazovanja i sporta

HRVATSKA ZAJEDNICA TEHNIČKE KULTURE

5.1. Zadatak: Todor — Autor: Stjepan Požgaj

Zadatak možemo rješiti if-ovima. Ako su oba broja različita od 7, onda ispišemo manji broj od razlike većeg i manjeg broja iz ulaza, odnosno 6 - ta ista razlika. Ako su oba broja jednaka ispišemo nula. Inače ako je jedan broj 7, njega pretvorimo u 1 i ispišemo rješenje kao u prvom slučaju samo uvećano za 1 (jer smo skočili sa 7 na 1).

5.2. Zadatak: MSV — Autor: Nikola Dmitrović

Simulacijom uvjeta definiranih u tekstu zadatku možemo riješiti ovaj problem. Za prvu parcijalu, tj. točan ispis prvog retka dovoljno je zbrojiti N prirodnih brojeva s ulaza. Za točan ispis drugog retka trebamo nekako pratiti tko je na potezu te je li se pojavila šestica. Tko je na potezu možemo pratiti na razne načine. Jedan od njih je da imamo niz/listu u kojoj nam element na poziciji 0 odgovara Marinu, na poziciji 1 Stjepanu i na poziciji 2 Vedranu. Varijablom *potez* s vrijednostima 0, 1 ili 2 pratimo tko je na potezu tako što nakon svakog učitavanja ovu vrijednost povećamo za jedan. Problem je kako osigurati cikličnost i što napraviti kada se pojavi šestica. U dijelu zadatka, kako je opisano u sekciji bodovanje, neće biti potrebno paziti na šesticu. Općenito, cikličnost se postiže uz pomoć ostatka pri djeljenju, konkretno u ovom slučaju s 3.

Programski kod (pisan u Python 3.x)

```
N = int(input())
prijatelj = [0] * 3
potez = 0                          # Marin je prvi na potezu
for i in range(N):
    Bi = int(input())
    prijatelj[potez] += Bi
    if Bi != 6:                    # ako nije pala šestica
        potez = (potez + 1) % 3    # cikličnost
print(sum(prijatelj)) # prvi redak, zbroj elemenata u listi
print(prijatelj[0], prijatelj[1], prijatelj[2]) # drugi redak
```

Potrebno znanje: naredba ponavljanja, lista (izbor)

Kategorija: ad hoc

5.3. Zadatak: Polica — Autor: Marin Kišić

Najlakši način za riješiti ovaj zadatak jest simulirati svako preslagivanje kamenja. U niz A duljine 500 na mjesto A[i] ćemo zapisati broj T ako se na njemu nalazi kamen na kojem piše broj T, inače A[i] će biti jednako 0. Sada for petljom koja ide od 500 prema 0 nađemo zadnju poziciju u nizu na kojoj piše neki broj, neka je to pozicija P1. Nakon toga, krenimo od pozicije P1-X prema 0 i nađimo zadnju poziciju takvu da od te pozicije prema 0 u nizu A pišu samo nule, neka je to pozicija P2. Nakon što imamo i P1 i P2 da izvršimo preslagivanje trebamo samo zamijeniti brojeve A[P1] i A[P2], A[P1-1] i A[P2-1], …, A[P1-X+1] i A[P2-X+1].

Potrebno znanje: for petalja, nizovi

Kategorija: simulacija

Potrebno znanje:

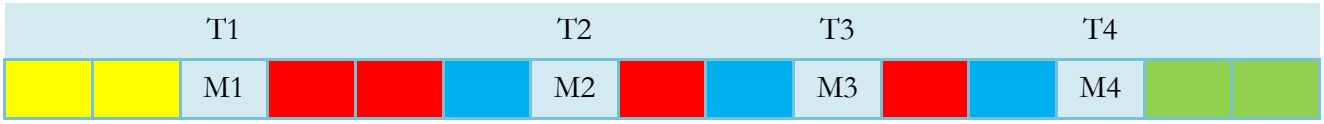
Kategorija: ad hoc

6.1. Zadatak: Senzor

Autor: Nikola Dmitrović

Sekcija Bodovanje daje mogućnost odabira različitih podzadataka i rješenja koja su mogla donijeti različit broj bodova.

U rješenju za sve bodove, htjeli to ili ne, moramo za svaku minutu odrediti vrijednost temperature u njoj. U listu *dan* upisat ćemo vrijednosti temperatura koje su već izmjerene, na poziciju *Mi-1* upisat ćemo vrijednost *Ti*. U posebnu listu *minute* upisat ćemo minute u kojima su izmjerene temperature kako bi mogli lakše odrediti vrijednosti temperatura u preostalim minutama. Posebno ćemo odrediti temperature u minutama koje prethode prvoj minuti u kojoj je izmjerena temperatura, posebno u onima koje se nalaze nakon posljednje minute u kojoj je izmjerena temperatura i posebno sve ostale. Primjer:



Programski kod (pisan u Python 3.x)

```
N = int(input())
dan = [0] * 1440 # dan ima 1400 minuta
minute = []
for i in range(N):
    Mi, Ti = map(int, input().split())
    minute += [Mi]; dan[Mi - 1] = Ti
X = int(input())
# minute u danu koje prethode prvoj minuti u kojoj je izmjerena temperatura
for i in range(0, minute[0]):  # žuto
    dan[i] = dan[minute[0]-1]
# sve ostale minute
for i in range(N - 1):
    l = minute[i]
    d = minute[i + 1]
    m = (l + d) // 2
    for i in range(l,m):   # minute koje su u lijevom dijelu segmenta [l,d]
        dan[i] = dan[l-1]  # crveno
    for i in range(m, d):  # minute koje su u desnom dijelu segmenta [l,d]
```

```
        dan[i] = dan[d-1]  # plavo
# minute u danu koje su nakon posljednje minute u kojoj je izmjerena temperatura
for i in range(minute[-1],1440):  # zeleno
    dan[i] = dan[minute[-1]-1]
print(dan[X-1])
print(sum(dan))
```

Potrebno znanje: lista/niz

Kategorija: ad hoc

6.2. Zadatak: Startup — Autor: Nikola Dmitrović

Za svaki od N gradova trebamo odrediti vremensku razliku u tom gradu u odnosu na Rijeku. Nakon toga, za +svaki sat *i* u danu (gledano iz Rijeke) trebamo prebrojiti u koliko je gradova trenutni sat unutar radnog vremena. Ako je broj gradova jednak N, tada smo našli termin u danu kada svi mogu. Ako nije jednako N, tada trebamo provjeriti je li to najveći broj gradova koji smo našli do tada.

Programski kod (pisan u Python 3.x)

```
N = int(input())
Rijeka = int(input())

vrijeme = []
for i in range(N):
    vrijeme += [int(input())]

razlika = [0] * N
for i in range(N):
    r = Rijeka - vrijeme[i]
    if r >= 0:
        if r <= 12:
            razlika[i] = -r
        else:
            razlika[i] = 24 - r
    else:
        if abs(r) < 12:
            razlika[i] = abs(r)
        else:
            razlika[i] = abs(r) - 24
print(sum(razlika))
```

```
termini = 0
maks = parcijala = 0
for i in range(24):
    koliko = 0
    for j in razlika:
        if 8 <= (i + j) % 24 <= 16:
            koliko += 1
    if koliko == N:
        termini += 1
    if koliko > maks:
        maks = koliko
        parcijala = i

print(termini)
print(parcijala)
```

Potrebno znanje: lista, traženje maksimalnog elementa

Kategorija: ad hoc

6.3. Zadatak: Rječnik — Autor: Vedran Kurdija

Zadatak ćemo riješiti sljedećim algoritmom:

1. Odaberi prvu riječ for petljom i počni graditi traženu riječ tako da staviš tražena=prva odabrana riječ.
2. Odaberi for petljom drugu riječ.
3. Provjeri nalazi li se druga riječ u traženoj riječi, ako da, možeš dalje, ako ne, proširi traženu riječ tako da se nalazi druga riječ u njoj. PAZI - ponekad je dovoljno na kraj tražene riječi dodati samo par slova drudge riječi.
4. Ponovi korake 2. i 3. za treću i četvrtu riječ.
5. Zapamti koju si riječ dobio i u neku varijablu rješenje zapamti najkraću traženu riječ koju si dobio.

7.1. Zadatak: Braille — Autor: Nikola Dmitrović

Matematički izraz je oblika *X op1 Y op2 Z*. Iz dobivenih ulaznih podataka trebamo odrediti X, Y i Z te op1 i op2. Kada to odredimo neće biti problem odrediti vrijednost izraza jer su moguća samo dva operatora (+ i *), a posebno će to biti lako napraviti u Pythonu koji ima funkciju *eval()*.

Programski kod (pisan u Python 3.x)

```
# X op1 Y op2 Z
prvi = input()
drugi = input()
treci = input()
```

```
X = [prvi[0], prvi[1], drugi[0], drugi[1], treci[0], treci[1]]
Y = [prvi[3], prvi[4], drugi[3], drugi[4], treci[3], treci[4]]
Z = [prvi[6], prvi[7], drugi[6], drugi[7], treci[6], treci[7]]

op1 = drugi[2]
op2 = drugi[5]

# generiranje brojeva

braille = [['o','x','x','x','o','o'],
           ['x','o','o','o','o','o'],
           ['x','o','x','o','o','o'],
           ['x','x','o','o','o','o'],
           ['x','x','o','x','o','o'],
           ['x','o','o','x','o','o'],
           ['x','x','x','o','o','o'],
           ['x','x','x','x','o','o'],
           ['x','o','x','x','o','o'],
           ['o','x','x','o','o','o']]

for i in range(10):
    if braille[i] == X: X = i
    if braille[i] == Y: Y = i
    if braille[i] == Z: Z = i

izraz = str(X)+op1+str(Y)+op2+str(Z)
print(eval(izraz))
```

Potrebno znanje: stringovi, dvodimenzionalna lista

Kategorija: ad hoc

7.2. Zadatak: Tablica — Autor: Marin Kišić

Za 20% bodova u ovom zadatku dovoljno je bilo samo simulirati točno što zadatak kaže.

Za 50% bodova trebalo je primjetiti da možemo sortirati samo stupac iz upita, pronaći koji redak koji nas zanima te onda u njemu ispisti broj koji se traži od nas iz upita. Detalje oba parcijalna rješenja ostavljam čitatelju za vježbu.

Za 100% bodova prije nego što počnemo odgovarati na upite, napravit ćemo tablicu tko. Na polju tko[a][b] pisat će nam koji redak će biti b-ti redak ako tablicu sortiramo po a-tom stupcu. Primjetimo

sada da kad smo napravili tablicu tko i dobijemo upit, odgovor na upit je p[tko[a][x]][y] gdje je p početna tablica. Za implementacijske detalje pogledajte kod.

Složenost rješenja: O(N*M*log(N))

Potrebno znanje: Izračunavanje složenosti, preprocesurianje podataka, sortiranje

Kategorija: Ad hoc

7.3. Zadatak: Riječ Autor: Vedran Kurdija

Pogledaj 6.3.

8.1. Zadatak: OPG — Autor: Nikola Dmitrović

Koristeći zadane podatke trebamo za svaki dan u tjednu odrediti ukupnu cijenu zadanih K proizvoda na taj dan. U niz *cijena_po_danima* zapisivat ćemo ukupnu vrijednost K proizvoda na zadani dan u tjednu. Treba paziti na činjenicu da se dani periodički ponavljaju. Kada to odredimo neće biti teško odrediti dan za koji je ta ukupna cijena najmanja.

Programski kod (pisan u Python 3.x)

```
N = int(input())
K = int(input())
T = int(input())

cjenik = []
for i in range(N):
    redak = list(map(int, input().split()))
    cjenik.append(redak)
proizvodi = list(map(int, input().split()))

cijena_po_danima = [0] * 7
for i in range(7*T):
    zbroj = 0
    for j in proizvodi:
        zbroj += cjenik[j-1][i]
    cijena_po_danima[i % 7] += zbroj

dan_u_tjednu = ['PONEDJELJAK', 'UTORAK', 'SRIJEDA',
                'CETVRTAK', 'PETAK', 'SUBOTA', 'NEDJELJA']
print(dan_u_tjednu[cijena_po_danima.index(min(cijena_po_danima))])
```

Potrebno znanje: lista lista (dvodimenzionalni niz)

Kategorija: ad hoc

8.2. Zadatak: Tenis — Autor: Stjepan Požgaj

Za odgovor na prvo pitanje možemo izračunati sve umnoške, sortirati ih i onda metodom dva pokazivača održavati interval u kojem je razlika onog broja na koji pokazuje manji pokazivač i razlika broja na koji pokazuje veći pokazivač < M. U svakom koraku rješenje povećamo za veličinu intervala.

Kako bismo riješili drugi dio zadatka potrebno je za svaki tim zapamtiti kvalitete protiv svih drugih ekipa i sortirati ih (vidjet ćemo zašto je to bitno kasnije). Sada fiksiramo neke dvije ekipe i želimo prebrojati na koliko načina možemo odabrati drugi meč. Uz pomoć binarne pretrage u nizu svih umnožaka možemo odrediti koliko postoji umnožaka koji su udaljeni za najviše M od našeg umnoška. Međutim sada još

želimo oduzeti sve one kojima je jedan od timova neki od dva naša fiksirana tima. To možemo tako da opet binarnom pretragom, ali ovaj put po umnoščima naših fiksiranih timova pronađemu tu informaciju (zato je bilo bitno da kvalitete sortiramo za svaki tim kako bismo mogli raditi binarnu pretragu po njima).

Potrebno znanje: binarna pretraga

Kategorija: ad hoc

8.3. Zadatak: Višeslav Autor: Vedran Kurdija

Odmah se vidi da je ključni dio zadatka obrađivanje zahtjeva SALJI, jer na prvi pogled nije jasno kako na njega odgovoriti. Kako naći minimalan broj dodavanja od jednog do drugog igrača, uzimajući u obzir da možemo dodavati u bilo kojem smjeru za A, B i C te čak i mijenjati smjer -- npr. poslati loptu za A igrača desno i potom za B igrača lijevo? Nikakva jednostavna formula ovdje nam neće pomoći -- previše je mogućnosti u ovisnosti o veličini kruga, brojevima A, B, C i zadanim dvama igračima.

Ovdje je ključna dosjetka promatrati igrače kao vrhove grafa, a parove igrača koji jedan drugome mogu izravno poslati loptu (tj. koji su na udaljenosti A, B ili C) kao bridove grafa. Tada se zahtjev SALJI svodi na traženje najkraćeg puta (u smislu broja bridova) između dvaju igrača, što možemo riješiti BFS algoritmom. Susjedstva u grafu ne moramo eksplicitno konstruirati, dovoljno je pri širenju iz igrača s rednim brojem x razmotriti njegove susjede iz niza [x + a, x - a, x + b, x - b, x + c, x - c], uzimajući ostatak dobivenog broja pri dijeljenju s N kako bismo ostali unutar kruga.

Ovo još nije efikasno rješenje: izvođenje BFS-a po grafu od 100 000 vrhova za svaki zahtjev SALJI, kojih također može biti 100 000, presporo je. Potrebna nam je još jedna dosjetka, a to je sljedeća: za fiksnu veličinu kruga, odgovor na zahtjev SALJI ovisi samo o udaljenosti u krugu između zadanih igrača, a ne i o konkretnim igračima koji su zadani. Na primjer, najkraći put između igrača 6 i 15 isti je kao najkraći put između igrača 0 i 9. Zato je za fiksnu veličinu kruga dovoljno provesti jedan BFS, računajući najkraće putove od igrača s rednim brojem 0, a poslije za svaki zahtjev SALJI izračunamo udaljenost u krugu zadanih igrača i očitamo najkraći put za tu istu udaljenost od igrača 0.

Dakle, dovoljno je provesti jedan BFS za svaku veličinu kruga, tj. moramo ga ponovno provesti svaki put kad se promijeni veličina kruga. Budući da različitih zahtjeva IZBACI može biti samo 26 jer je toliko mogućih slova, veličina kruga mijenja se najviše 26 puta i toliko BFS-ova može se izvršiti unutar nekoliko sekundi. Moramo pritom paziti da zanemarujemo zahtjeve IZBACI koji "izbacuju" već izbačeno slovo, za što nam je potreban pomoćni niz koji za svako slovo pamti je li izbačeno. Izbacivanje igrača sa zadanim početnim slovom možemo napraviti običnim prolaskom po nizu igrača, spremajući neizbačene igrače u novi niz. Nakon toga slijedi već opisani BFS za novu veličinu kruga.

I napokon, implementacija rješenja u Pythonu neće biti dovoljno efikasna za vremensko ograničenje od jedne sekunde zbog inherentne sporosti Pythona. Ovakvi zadatci motivacija su za učenje "bržih" programskih jezika kao što je C++, koji se standardno koristi na "jačim" natjecanjima.

Programski kod (pisan u C++)

#

Potrebno znanje: BFS

Kategorija: Grafovi