2017 iz informatike Natjecanje

16. ožujka 2017.

Državna razina / Osnovna škola (5. i 6. i 7. i 8. razred)

Primjena algoritama OŠ

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: Rekord

Autor: Nikola Dmitrović

U tekstu zadatka je definirano da je Sarin osobni rekord **66.18** metara, olimpijski rekord **71.53** metara, a svjetski rekord **72.28** metara. Za N zadanih vrijednosti potrebno je odrediti u kakvom su odnosu sa zadanim vrijednostima i ovisno o tome ispisati odgovarajuću poruku ili vrijednost.

Programski kod (pisan u Python 3.x)

```
N = int(input())

PB = 66.18
OR = 71.53
WR = 72.28

for i in range(N):
    X = float(input()) # vrijednost x je realnog tipa podataka
    if X > WR:
        ispis = 'WR'
    elif X > OR:
        ispis = 'OR'
    elif X > PB:
        ispis = 'PB'
    else:
        ispis = X
    print(ispis)
```

Potrebno znanje: realni tip podataka, višestruka naredba odlučivanja, naredba ponavljanja

Kategorija: ad hoc

5.2. Zadatak: Kevin

Autor: Nikola Dmitrović

Uočimo da se u ovom zadatku mogao dobiti veći broj bodova rješavajući dijelove zadatka kako je to opisano u sekciji „Bodovanje“.

Riješimo prvo dio zadatka za pola bodova, tj. odredimo ukupno vrijeme u kojem nije bilo struje izraženo u minutama. Iz teksta zadatka u kojem piše da će na početku i na kraju dana biti struje kao i iz činjenice da je N paran broj zaključujemo da će struja N pola puta prvo nestati, a zatim ponovno doći. Zbog toga možemo učitavati ulazne podatke u parovima, pretvarati ih u minute i oduzeti vrijeme nestanka struje od vremena dolaska struje kako bi odredili koliko dugo struje nije bilo u trenutnom periodu bez struje.

Programski kod (pisan u Python 3.x)

```
N = int(input())
nema_struje = 0
for i in range(N // 2):
    sat_nestanka, minuta_nestanka = map(int, input().split())
    nestanak = sat_nestanka * 60 + minuta_nestanka

    sat_dolaska, minuta_dolaska = map(int, input().split())
```

```
    dolazak = sat_dolaska * 60 + minuta_dolaska

    nema_struje += dolazak - nestanak

print(nema_struje)
```

U sekciji „Bodovanje“ zapisano je da će u test podacima vrijednima 30 bodova vrijediti da je **N** = 2, tj. da će struja samo jednom nestati i vratiti se. Tih 30 bodova može se dobiti ispisivanjem razlike vremena dolaska i vremena nestanka struje te ulaznog podatka koji predstavlja vrijeme kada je struja nestala.

Programski kod (pisan u Python 3.x)

```
N = int(input())

sat_nestanka, minuta_nestanka = map(int, input().split())
nestanak = sat_nestanka * 60 + minuta_nestanka

sat_dolaska, minuta_dolaska = map(int, input().split())
dolazak = sat_dolaska * 60 + minuta_dolaska

print(dolazak - nestanak)
print(sat_nestanka, minuta_nestanka)
```

I za kraj, pokažimo jedno od rješenja za sve bodove. Znamo da u jednom danu ima 1440 minuta (24 sata po 60 minuta). U svakoj toj minuti struje je moglo ili biti ili ne biti. Zamislimo tu situaciju kao niz (listu) od 1440 komponenti koje mogu imati vrijednosti nula (nema struje u i-toj minuti) ili jedan (ima struje u i-toj minuti). U početku sve su vrijednosti postavljene na jedan. Kada u nekom trenutnu vremena, tj. u nekoj minuti nestane struje, sve komponente niza za sve minute nakon tog vremena postavimo na nula. Vrijedi o obrnuto. Na kraju je broj komponenti s vrijednošću nula odgovor na prvo pitanje, a zbroj vrijednosti svih komponenti do minute u kojoj je zadnji put došla struja odgovor na drugo pitanje.

Programski kod (pisan u Python 3.x)

```
ima_struje = 1
N = int(input())

L = 1440 * [1]

for i in range(N):
    ima_struje *= -1

    sat, minuta = map(int, input().split())
    promjena = sat * 60 + minuta

    if ima_struje == 1:
        for j in range(promjena, 1440):
            L[j] = 1
    else:
```

```
        for j in range(promjena, 1440):
            L[j] = 0

print(1440 - sum(L))
print(sum(L[0:promjena]) // 60, sum(L[0:promjena]) % 60 )
```

Potrebno znanje: naredba odlučivanja, naredba ponavljanja, pretvaranje sata i minute u minute, jednodimenzionalni niz (lista) - opcija

Kategorija: ad hoc

5.3. Zadatak: Bojan — Autor: Marin Tomić

Simulacijom koraka opisanih u tekstu zadatka možemo riješiti ovaj zadatak. Kako čista simulacija ne prolazi unutar vremenskog ograničenja za neke test podatke, tada trebamo pronaći način da odredimo debljinu sloja boje bez da simuliramo sve poteze kistom.

Programski kod (pisan u Python 3.x)

```
n = int(input())
boji = list(map(int, input().split()))
sol = [0 for i in range(n)]
k = int(input())
for it in range(k):
    a, b, x = map(int, input().split())
    a -= 1; b -= 1
    c = 0
    for i in range(a, b + 1):
        if boji[i]:
            c += 1
    for i in range(a, b + 1):
        if boji[i]:
            sol[i] += 2 * (x // (2 * c))
    x %= (2 * c)
    for i in range(a, b + 1):
        if x > 0 and boji[i]:
            sol[i] += 1
            x -= 1
    for i in range(b, a - 1, -1):
        if x > 0 and boji[i]:
            sol[i] += 1
            x -= 1
```

```
for i in range(n):
    print(sol[i], end=' ')
print()
```

Potrebno znanje: lista

Kategorija: simulacija

6.1. Zadatak: Set — Autor: Nikola Dmitrović

Uočimo da se opis rezultata jednog seta ne mijenja i da je uvijek oblika „znamenka:znamenka/“. Zbog toga možemo jednostavno parsirati string uzimajući četiri po četiri znaka.

Programski kod (pisan u Python 3.x)

```
mec = input()

ime1 = ime2 = 0
for i in range(0, len(mec), 4):
    if mec[i] > mec[i+2]:
        ime1 += 1
    else:
        ime2 += 1

if ime1 > ime2:
    print('ime1')
else:
    print('ime2')

print(ime1, ime2)
```

Potrebno znanje: string

Kategorija: ad hoc

6.2. Zadatak: Nogomet — Autor: Nikola Dmitrović

Ideja rješenja ovog zadatka je okupiti sve ulazne podatke u jedan niz u obliku koji će nam omogućiti najlakšu analizu, sortirati taj niz po vremenu postizanja gola i zatim ispisati članove niza pazeći da pravilno odredimo trenutni rezultat utakmice. Primjer jedne moguće implementacije rješenja (ne nužno optimalne) slijedi u nastavku.

Programski kod (pisan u Python 3.x)

```
H = int(input())
HG = []

for i in range(H):
```

```
    igrac, minuta = input().split()
    HG += [('H', igrac, int(minuta))]

B = int(input())
BG = []

for i in range(B):
    igrac, minuta = input().split()
    BG += [('B', igrac, int(minuta))]

tekma = HG + BG

for i in range((H + B)-1):
    for j in range(i+1, H + B):
        if tekma[i][2] > tekma[j][2]:
            tekma[i], tekma[j] = tekma[j], tekma[i]

H_golovi = B_golovi = 0
for i in tekma:
    if i[0] == 'H':
        H_golovi += 1
        print(H_golovi, ':', B_golovi,' ', i[1], sep = '')
    else:
        B_golovi += 1
        print(H_golovi, ':', B_golovi,' ', i[1], sep = '')
```

Potrebno znanje: nizovi, sortiranje

Kategorija: ad hoc

6.3. Zadatak: Znamenke Autor: Adrian Satja Kurdija

Naivno rješenje for petljom prolazi brojevima od A do B te za svaki od njih provjerava je li "dobar" (čine li mu znamenke rastući niz) dok ne pronađe 10 dobrih brojeva. No već u drugom primjeru iz teksta zadatka uviđamo da je takvo rješenje daleko presporo za naše potrebe jer mu treba jako dugo dok ne dođe do nekog dobrog broja, pogotovo ako zadani brojevi imaju do 100 znamenaka kao što piše u zadatku.

Zadatak efikasnije rješavamo tako da brojeve tretiramo kao stringove te izravno mijenjamo njihove znamenke. Najprije valja efikasno pronaći najmanji dobar broj veći od A. Ako malo razmislimo, uočavamo da je to broj koji dobivamo tako da pronađemo prvi par susjednih znamenaka broja A koji nije rastući: njega moramo "popraviti", a "najmanji" način da to učinimo jest da prvu od te dvije znamenke "produljimo" sve do kraja broja, npr. 1**60**320162017 → 166666666666.

Potom treba pronaći još 9 dobrih brojeva manjih od B. Napisat ćemo funkciju koja za neki dobar broj efikasno pronalazi **prvi sljedeći**. Nije teško zaključiti da ona funkcionira na sljedeći način:

- za jedan povećava posljednju znamenku ako je ona manja od 9;
- inače, za jedan povećava pretposljednju znamenku ako je ona manja od 9, a posljednju postavlja tako da je jednaka pretposljednjoj, npr. 1**3**9 → 1**4**4;
- inače, analogno, pronalazi najdesniju znamenku koja je manja od 9, povećava je za jedan te sve sljedeće znamenke postavlja tako da su jednake njoj, npr. 112333**4**9999999 → 112333**5**5555555;
- ako su sve znamenke jednake 9, prvi sljedeći dobar broj ima jednu znamenku više i sastoji se od samih jedinica.

Programski kod (pisan u Python 3.x)

```
A = input()
B = input()
a = list(map(int, A))
n = len(a)

def povecaj():
    global a
    global n
    i = n - 1
    while a[i] == 9:
        i -= 1
        if i == -1:
            a = [0] + a
            i = 0
            n += 1
    a[i] += 1
    for j in range(i + 1, n):
        a[j] = a[j-1]
```

```
for i in range(1, n):
   if a[i] < a[i-1]:
       for j in range(i, n):
           a[j] = a[j-1]
for k in range(10):
   x = int(''.join(map(str, a)))
   print(x)
   povecaj()
```

Potrebno znanje: stringovi

Kategorija: ad hoc

7.1. Zadatak: Dvojba — Autor: Nikola Dmitrović

Zadatak je moguće riješiti provjeravanjem svih mogućih slučajeva ili uspoređivanjem ponuđenih dana u odnosu na dan kada se trebao pisati test. Pokažimo jedan lakši način. Iz teksta zadatka možemo zaključiti da će se test sigurno pisati unutar trenutnog ili sljedećeg tjedna. Zbog toga ćemo kreirati niz od deset komponenti (po jednu za svaki radni dan u dva tjedna). U odgovarajuću komponentu za svaki od tri ponuđena dana upisat ćemo vrijednost jedan. Gledajući od kraja niza, prva komponenta koja ima vrijednost jedan je traženi dan u tjednu.

Programski kod (pisan u Python 3.x)

```
D = input()
D1 = input()
D2 = input()

dani = ["ponedjeljak", "utorak", "srijeda", "cetvrtak", "petak",
        "ponedjeljak", "utorak", "srijeda", "cetvrtak", "petak"]
kopija = dani [:]

L = [0] * 10

L[dani.index(D)] = 1
dani[dani.index(D)] = ''

L[dani.index(D1)] = 1
dani[dani.index(D1)] = ''

L[dani.index(D2)] = 1
dani[dani.index(D2)] = ''

for i in range(9, -1, -1):
    if L[i] == 1:
```

```
        print(kopija[i % 5])
        break
```

Potrebno znanje: nizovi

Kategorija: ad hoc

7.2. Zadatak: Jasna — Autor: Adrian Satja Kurdija

Ovaj zadatak, inspiriran stvarnim događajem, rješavamo tretirajući učitani broj mobitela kao string i pažljivo primjenjujući pravila opisana u tekstu zadatka. Proći ćemo po drugom stringu ulaznih podataka, onom koji sadrži kriterije koje treba primijeniti, te za svaki od njegovih znakova ('2', '3' ili '4') pronaći sve stringove koji dolaze u obzir na temelju odgovarajućeg kriterija te ih ubaciti u niz rješenja. Na kraju iz niza rješenja treba izbaciti duplikate i početni string.

Sada opisujemo detalje primjene pojedinih kriterija.

1. Ovaj kriterij uvijek se primjenjuje na način da prva tri znaka stringa (pozivni broj) ostaju nepromijenjena. U službenom rješenju to je implementirano tako da smo ulaznom stringu "odrezali" pozivni broj i na kraju ga dodavali prilikom ispisivanja rješenja.
2. Za sve moguće pozicije K (koje biramo for petljom) zamijenit ćemo K-ti i (K+1)-i znak ulaznog stringa da bismo dobili moguće rješenje.
3. Za sve moguće pozicije K (koje biramo for petljom) i sve moguće znamenke Z od 0 do 9 (koje biramo unutarnjom for petljom) postavljamo K-ti znak ulaznog stringa na Z da bismo dobili moguće rješenje.
4. Ovisno o broju znakova ulaznoga broja, izbacujemo K-ti znak na sve moguće načine, ili na K-to mjesto ubacujemo znamenku Z na sve moguće načine, slično prethodnom kriteriju.

Programski kod (pisan u Python 3.x)

```
broj = input()
pozivni = broj[:3]
broj = broj[3:]
n = len(broj)
brojevi = set()
pravila = input()
for p in pravila:
    if p == '2':
        for i in range(n - 1):
            brojevi.add(broj[:i] + broj[i + 1] + broj[i] + broj[i + 2:])
    if p == '3':
        for i in range(n):
            for z in range(10):
                brojevi.add(broj[:i] + chr(ord('0') + z) + broj[i + 1:])
    if p == '4':
```

```
        if n == 7:
            for i in range(n):
                brojevi.add(broj[:i] + broj[i + 1:])
        else:
            for i in range(n + 1):
                for z in range(10):
                    brojevi.add(broj[:i] + chr(ord('0') + z) + broj[i:])
brojevi = brojevi - {broj}
for b in brojevi:
   print(pozivni + b)
```

Potrebno znanje: stringovi

Kategorija: simulacija

7.3. Zadatak: Banka Autor: Adrian Satja Kurdija

Pretpostavite da je poznat broj X koji je prvi izvučen iz aparata. Biste li tada znali riješiti zadatak? Najprije razmislite sami, a tek onda čitajte dalje.

Znajući broj X, slijedi da su brojevi izvučeni iz aparata sljedećim redoslijedom:

X, X+1, …, K-1, K, 1, 2, …, K-1, K, 1, 2, …

-- i tako dalje, ciklički se ponavljaju brojevi od 1 do K. Preostaje svakom izvučenom broju pridružiti osobu koja ga je izvukla, a to činimo na sljedeći način.

- Prilikom unošenja podataka, u pomoćnu tablicu za svaki izvučeni broj Y (od 0 do K) spremamo sve osobe koje su izvukle broj Y.
- Na kraju, prolazeći izvučenim brojevima od X nadalje, za trenutačni broj ispisujemo osobu koja ga je izvukla (iz tablice) i nju brišemo iz tablice. Ako u tablici nema takve osobe, ispisujemo osobu koja je "izvukla" broj 0 (iz tablice) i nju također brišemo iz tablice. Ako nema ni takve osobe, znači da redoslijed izvučenih brojeva nije ispravan, tj. pogriješili smo u odabiru početnog broja X.

Dakle, znamo riješiti zadatak ako znamo X. Međutim, mi ga ne znamo. Što ćemo sada? Najprije razmislite sami, proučite primjere iz teksta zadatka, a tek onda čitajte dalje.

Jedno moguće rješenje isprobava sve moguće početne brojeve X i za svaki od njih provodi gore opisani postupak sve dok taj postupak ne uspije provesti do kraja (što znači da imamo rješenje), ili dok postupak ne "zapne" (došli smo do broja kojemu više ne možemo pridružiti nijednu osobu), što znači da trenutačni X nije ispravan, pa se cijeli postupak "resetira" te se isprobava sljedeći X. Takvo rješenje je točno, ali ne osvaja sve bodove jer je presporo za velike test podatke.

Za sve bodove valja se zapitati koliko je puta izvučen koji broj. Budući da se brojevi izvlače redom ("ravnomjerno"), svaki broj izvučen je:

- N/K puta ako je N djeljiv s K,
- a inače je svaki broj izvučen N/K ili N/K + 1 puta (cjelobrojno dijeljenje).

U prvom slučaju možemo krenuti od bilo kojeg početnog X jer je svaki izvučen jednako mnogo puta pa postupak pridruživanja osoba izvučenim brojevima ne ovisi o redoslijedu izvlačenja.

U drugom slučaju, početni X je prvi u skupini brojeva koji su izvučeni jedan put više, dakle N/K + 1 puta, što znači da su brojevi koje dolaze neposredno prije X (npr. broj X-1, ili broj K ako je X=1) izvučeni jedan put manje, dakle N/K puta.

Takav X lako je pronaći ako u ulaznim podatcima nema nula (nepoznatih brojeva), tj. ako znamo koliko je puta koji broj izvučen, što je u dijelu test podataka i bio slučaj kao što stoji u sekciji Bodovanje.

A što ako nije tako? U općem slučaju potražit ćemo X koji je (s obzirom na poznate podatke) izvučen najviše puta, a ako je više takvih kandidata, tražimo onaj sa što više prethodnika manjih od sebe. Drugim riječima, tražimo "maksimum" koji je, gledajući ciklično (niz ide "u krug" -- nakon K dolazi 1), najudaljeniji od prethodnog "maksimuma". Taj X sigurno je ispravan ako rješenje postoji[1], no ako ne uspijemo ni za njega provesti postupak dodjeljivanja osoba, nema rješenja pa ispisujemo 0.

Programski kod (pisan u Python 3.x)

```
n, k = map(int, input().split())

a = list(map(int, input().split()))

osobe = [[] for i in range(k + 1)]

puta_izvucen = [0 for i in range(k + 1)]

for i, y in enumerate(a):
    osobe[y].append(i + 1)
    puta_izvucen[y] += 1


m = max(puta_izvucen[1:])

x = puta_izvucen[1:].index(m) + 1

i = j = x


najveca_udaljenost_do_prethodnog_maksa = 0

trenutna_udaljenost_do_prethodnog_maksa = 0

while True:
    j += 1
    if j > k:
        j = 1
    trenutna_udaljenost_do_prethodnog_maksa += 1
    if puta_izvucen[j] == m:
        if trenutna_udaljenost_do_prethodnog_maksa >\
                najveca_udaljenost_do_prethodnog_maksa:
            x = j
```

[1] Strogi dokaz ove tvrdnje ostavljamo za vježbu ambicioznim čitateljima. Dokaze možete slati na askurdija@gmail.com.

```
            najveca_udaljenost_do_prethodnog_maksa =\
                    trenutna_udaljenost_do_prethodnog_maksa
        trenutna_udaljenost_do_prethodnog_maksa = 0
    if j == i:
        break


i = x
rjesenje = []
for it in range(n):
    if osobe[i]:
        rjesenje.append(osobe[i].pop())
    elif osobe[0]:
        rjesenje.append(osobe[0].pop())
    else:
        print(0)
        exit(0)
    i += 1
    if i > k:
        i = 1
print(' '.join(map(str, rjesenje)))
```

Potrebno znanje: analiza problema

Kategorija: ad hoc

8.1. Zadatak: Pronički — Autor: Nikola Dmitrović

Zbog ograničenja da je $N \leq 10^{13}$ rješenje koje će za svaki broj manji od N provjeravati je on pronički sve dok ne pronađe takav broj neće biti dovoljno brzo i neće zadovoljiti vremensko ograničenje na zadnja dva test podatka. Zato trebamo osmisliti neko pametnije rješenje. Po definiciji pronički broj je prirodan broj koji je jednak **umnošku dvaju uzastopnih** prirodnih brojeva. Zbog toga je dovoljno redom množiti uzastopne brojeve (počevši od 1 i 2) i tražiti najveći takav umnožak manji ili jednak N.

Programski kod (pisan u Python 3.x)

```
from math import *
N = int(input())

for i in range(1, int(sqrt(N)) + 1):
    if i * (i + 1) <= N:
        odgovor = i
```

```
if odgovor * (odgovor + 1) == N:
    print(odgovor, odgovor + 1)
else:
    print(odgovor * (odgovor + 1))
```

Potrebno znanje: naredba ponavljanja

Kategorija: ad hoc

8.2. Zadatak: Slagalica Autor: Mislav Bradač

U test primjerima vrijednim 10% bodova postoji samo jedno crno polje te se takvi primjeri mogu riješiti pomoću jedne ili dvije rotacije. Ako se crno polje nalazi u prvom retku potrebno je rotirati samo prvi redak za određeni broj polja tako da crno polje "dovedemo" na prvo polje matrice. Ako to nije slučaj u prvoj rotaciji rotiramo stupac s crnim poljem tako da crno polje dovedemo do prvog retka te ga u idućoj rotaciji dovodimo do prvog polja.

U idućoj skupini primjera matrica je dimenzija 2x2. S obzirom da postoji 4 polja matrice te svako polje može biti ili crno ili bijelo, slijedi da postoji samo 2^4 = 16 različitih matrica. Te primjere se moglo riješiti na papiru te *hardkodirati* rješenja u program.

Za 50% bit će najviše **N** crnih polja, tj. u riješenoj slagalici se sva crna polja nalaze u prvom retku slagalice. Takvu slagalicu možemo riješiti tako da prvi redak po redu punimo crnimo poljima. Neka smo za sad popunili **C** prvih polja crnima, tada pronalazimo crno polje koje se nalazi na poziciji većoj od **C**. Razmatramo tri slučaja:

1. To polje se nalazi u (**C** + 1). stupcu. Pomoću jedne rotacije stupca dovodimo polje iz retka u kojem se nalazi u prvi redak (u slučaju da je polje već u prvom retku ne činimo ništa).
2. Polje se ne nalazi u prvom retku. Jednom rotacijom retka dovodimo polje u (**C** + 1). stupac. Sada smo ovaj problem sveli na prvi slučaj.
3. Polje se nalazi u prvom retku. Jednom rotacijom stupca ga dovodimo u drugi stupac te smo sveli problem na drugi slučaj.

U primjerima za 80% bodova vrijedi da je broj crnih polja manji jednak od bijelih polja slagalice. Ove primjere rješavamo na sličan način kao i prošlu skupinu. Pozicije ćemo po redu puniti crnim poljima. Poziciju **C** definiramo na isti način kao u prošlom poglavlju. Neka je **R** redak u kojem se nalazi pozicija **C**, a **S** pripadni stupac (numerirano od 0). Pronalazimo prvo crno polje na poziciji većoj od **C** te ćemo upravo to polje dovesti do pozicije **C**. Pretpostavimo da se to polje nalazi u zadnjem retku u stupcu različitom od **S**. Tada željeno polje možemo dovesti na poziciju **C** u tri rotacije:

1. U prvoj rotaciji stupac **S** rotiramo za (**N** - **R** + 1).
2. Zatim zadnji redak rotiramo tako da crno polje dovedemo u stupac **S**.
3. U posljednjoj rotaciji stupac **S** rotiramo za -(**N** - **R** + 1).

Primijetimo da smo ovim koracima crno polje iz zadnjeg retka matrice doveli do pozicije **C** te da je ostatak matrice ostao nepromijenjen (osim zadnjeg retka). Na početku smo uveli određene pretpostavke o poziciji crnog polja. Pokažimo sada da crno polje lako dovodimo do takve pozicije. U slučaju da se crno polje nalazi u zadnjem retku u stupcu **S** tada moramo prije prve rotacije rotirati zadnji redak za jedno polje. Ako se crno polje ne nalazi u zadnjem retku pomoću slične tri rotacije onima gore opisanim dovodimo crno polje u zadnji redak (možemo zamisliti kao da jedno bijelo polje iz zadnjeg retka dovodimo do pozicije gdje se nalazi naše crno polje). Primijetimo da ovim algoritmom ne možemo riješiti jedino matrice u kojima se na kraju neko crno polje mora nalaziti i u zadnjem retku.

Konačni algoritam za 100% bodova. Promotrimo 2 slučaja:

1. U prvom slučaju broj crnih polja je manji jednak **N** * (**N** - 1). Tada rješavamo matricu algoritmom za 80% bodova.
2. Ako je broj crnih polja veći od **N** * (**N** - 1) tada možemo algoritmom za 80% (ili onim za 50%) dovesti bijela polja na početne pozicije. Tada rotacijom svakog stupca za -1 dovodimo sva bijela polja u zadnji redak matrice te završnom rotacijom pomičemo sva bijela polja na posljednje pozicije u matrici. S obzirom da se sada sva bijela polja nalaze na posljednjim pozicijama matrice tada slijeda da se sva crna polja nalaze na početnim pozicijama.

Kod u C++-u pomoću sličnog algoritma onom gore opisanom rješava zadatak za 100% bodova.

Programski kod (pisan u C++)

```
#include <algorithm>
#include <cstdio>
const int MAXN = 20;

int n;
char a[MAXN][MAXN];

void rotate_row(int y, int t) {
  printf("R %d %d\n", y + 1, t);
  if (t < 0) {
    t = -t;
  } else {
    t = n - t;
  }
  for (int j = 0; j < t; ++j) {
    int tnp = a[y][0];
    for (int i = 0; i < n - 1; ++i) {
      a[y][i] = a[y][i + 1];
    }
    a[y][n - 1] = tnp;
  }
}
void rotate_col(int x, int t) {
  printf("S %d %d\n", x + 1, t);
  if (t < 0) {
    t = -t;
  } else {
    t = n - t;
  }
  for (int j = 0; j < t; ++j) {
    int tnp = a[0][x];
    for (int i = 0; i < n - 1; ++i) {
      a[i][x] = a[i + 1][x];
    }
    a[n - 1][x] = tnp;
  }
}

void solve(char c) {
  for (int i = 0; i < n * n; ++i) {
    int y = i / n;
```

```
    int x = i % n;
    if (a[y][x] == c)
      continue;
    for (int j = i + 1; j < n * n; ++j) {
      int y2 = j / n;
      int x2 = j % n;
      if (a[y2][x2] != c)
        continue;
      if (y == y2) {
        rotate_col(x2, 1);
        y2 += 1;
        y2 %= n;
        rotate_row(y2, 1);
        x2 += 1;
        x2 %= n;
        rotate_col((x2 - 1 + n) % n, -1);
      }
      if (x == x2) {
        rotate_row(y2, 1);
        x2 += 1;
        x2 %= n;
      }
      rotate_col(x, y2 - y);
      rotate_row(y2, x - x2);
      rotate_col(x, y - y2);
      rotate_row(y2, x2 - x);
      break;
    }
  }
}

int main() {
  scanf("%d", &n);
  for (int i = 0; i < n; ++i) {
    scanf("%s", a[i]);
  }

  int black = 0;
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
      if (a[i][j] == 'x') {
        black += 1;
      }
    }
  }

  if (n * n - black < n) {
    solve('o');
    for (int i = 0; i < n; ++i) {
      rotate_col(i, -1);
    }
    while (a[n - 1][0] == 'o') {
      rotate_row(n - 1, -1);
```

```
    }
  } else {
    solve('x');
  }
  printf("K");
  return 0;
}
```

Potrebno znanje: matrice

Kategorija: ad hoc

8.3. Zadatak: Kampovi Autor: Dominik Gleich

U ovom je zadatku bilo potrebno podijeliti N točaka u K grupa tako da smanjimo ukupnu sumu udaljenosti između svih parova točaka unutar svake od K grupa. S obzirom da se nije tražilo najbolje, već neko, dovoljno dobro rješenje i bodovanje je prilično tolerantno, možemo razmišljati o ne-optimalnim rješenjima.

Za početak primjećujemo kako se gotovo uvijek isplati napraviti grupe podjednake veličine zato što time smanjujemo broj parova točaka za koje računamo udaljenost (za vježbu probaj ovo matematički dokazati). To nas navodi na jednostvnu dodjelu slučajnog broja između 1 i K svakoj od N točaka. Takva rješenja su nosila 30-tak bodova, ovisno o implementaciji.

Daljnjim razmišljanjem možemo vidjeti kako možemo popraviti ovo rješenje još više. Možemo odabrati K točaka iz N točaka (koje će nam činiti 'središte' svaka svoje skupine) i zatim za svaku od preostalih N-K točaka pronaći najbližu od tih slučajno odabranih K. Takvo rješenje nosilo je 50-tak bodova.

S obzirom da imamo 2 sekunde vremena, a jednu iteraciju gore navedenog algoritma možemo napraviti prilično brzo za N i K, možemo ovaj postupak ponavljati i odabirati slučajnih K točaka sve dok imamo vremena! Takvo rješenje nosilo je 70-tak bodova.

Daljnim pokušajem optimizacije ovog rješenja možemo vidjeti kako odabir K središta točaka možemo popraviti tako da za svaku od K grupa točaka izračunamo koordinate težišta, i te nove koordinate koristimo kao novo središte svake od grupa i ovaj put pridjeljujemo svaku od N točaka njoj najbližem centru. Ponavljajući računanje težišta i raspodjelu točaka sve dok se težišta mijenjaju dobivamo rješenje koje nosi 80-tak bodova.

Ukoliko ponavljamo cijeli algoritam odabira slučajnih točaka i konvergiranja težišta tada dobivamo rješenje koje nosi 85 bodova.

Za maksimalnih 90 bodova potrebno je popraviti odabir slučajnih centra na početku, što implementiramo u službenom rješenju i ostavljamo čitatelju za vježbu, uz referencu na https://en.wikipedia.org/wiki/K-means++.

Potrebno znanje: matematička analiza problema

Kategorija: ad-hoc, heuristike