19. ožujka 2014.

Infokup 2013

Državno natjecanje / Osnovna škola (5. i 6. i 7. i 8. razred)

Algoritmi (Basic/Python/Pascal/C/C++)

OPISI ALGORITAMA

Agencija za odgoj i obrazovanje
Education and Teacher Training Agency

MINISTARSTVO ZNANOSTI, OBRAZOVANJA I ŠPORTA REPUBLIKE HRVATSKE

5.1. Zadatak: Obrisana

Autor: Nikola Dmitrović[1]

Bez potrebe za kompliciranjem, jednostavno prebrojimo koliko se puta pojavila svaka od 9 znamenki. Ona koja se pojavila točno jednom je tražena obrisana znamenka.

Programski kod (pisan u Pythonu)

```
V0 = V1 = V2 = V3 = V4 = V5 = V6 = V7 = V8 = V9 = 0
for i in range(7):
    x = int(input())
    if x == 0: V0 += 1
    if x == 1: V1 += 1
    if x == 2: V2 += 1
    if x == 3: V3 += 1
    if x == 4: V4 += 1
    if x == 5: V5 += 1
    if x == 6: V6 += 1
    if x == 7: V7 += 1
    if x == 8: V8 += 1
    if x == 9: V9 += 1
if V0 == 1: print(0)
if V1 == 1: print(1)
if V2 == 1: print(2)
if V3 == 1: print(3)
if V4 == 1: print(4)
if V5 == 1: print(5)
if V6 == 1: print(6)
if V7 == 1: print(7)
if V8 == 1: print(8)
if V9 == 1: print(9)
```

Potrebno znanje: naredba učitavanja i ispisivanja, naredba pridruživanja, naredba odlučivanja

Kategorija: ad hoc

[1] član Državnog povjerenstva, voditelj natjecanja u kategoriji primjena algoritama (programski jezik Basic/Python/Pascal/C/C++), profesor informatike u XV. gimnaziji, Zagreb

5.2. Zadatak: Znamenke

Autor: Matija Milišić[2]

Svaki mogući odabir triju Ivičinih znamenaka spada pod jedan od ova dva slučaja:

1. Sve Ivičine znamenke su različite od nule.
2. Jedna Ivičina znamenka jednaka je nuli.

Opisanim rješenjem naći ćemo sve trojke koje spadaju pod prvi slučaj te isto tako sve trojke koje spadaju pod drugi slučaj. Time ćemo naći sve moguće trojke znamenki.

1. SLUČAJ

U prvom slučaju imat ćemo tri znamenke različite od nula: a, b i c. Od njih se može složiti sljedećih 15 brojeva:

Brojevi	a	b	c	ab	ac	ba	bc	ca	cb
Rastav	a	b	c	10a+b	10a+c	10b+a	10b+c	10c+a	10c+b

Brojevi	abc	acb	bac	bca	cab	cba
Rastav	100a+10b+c	100a+10c+b	100b+10a+c	100b+10c+a	100c+10a+b	100c+10b+a

1. način - informatički

Uz pomoć tri ugniježđene petlje prođemo po svim mogućim trojkama. Za svaku trojku izračunamo sve moguće brojeve (pogledati tablicu) i sve ih zbrojimo. Ako je ukupan zbroj jednak S, onda je to moguća Ivičina trojka. Pseudokod ovog načina dan je u nastavku.

```
za a := 1 do 9 činiti
     za b := a+1 do 9 činiti
          za c := b+1 do 9 činiti
               zbroj = 0;
               zbroj = zbroj+(a);
               ...
               zbroj = zbroj+(10*a+b);
               ...
               zbroj = zbroj+(100*c+10*b+a);
               ako je zbroj = S onda
                    izlaz(a, b, c);
```

[2] student Fakulteta elektrotehnike i računarstva, osvajač brončane medalje na IOI 2011., srebrne medalje na CEOI 2011. te dviju brončanih medalja na IMO 2011. i IMO 2012, dvostruki državni prvak iz informatike (2. i 4. razred)

2. način - matematički

Ukupan zbroj ovih 15 brojeva jednak je 245*(a+b+c). To znači da su moguće Ivičine trojke one kojima je zbroj znamenki Z jednak S/245. Sada je lako naći rješenja ovog slučaja.

Uz pomoć tri ugniježđene petlje prođemo po svim mogućim trojkama. Za svaku trojku provjerimo je li zbroj znamenki jednak Z. Ako je, onda je to moguća Ivičina trojka. Pseudokod ovog načina dan je u nastavku.

```
Z = S/245;
za a := 1 do 9 činiti
      za b := a+1 do 9 činiti
            za c := b+1 do 9 činiti
                  ako je a+b+c = Z onda
                        izlaz(a, b, c);
```

2. SLUČAJ

Drugi slučaj riješit ćemo na sličan način. Imat ćemo tri znamenke: a, b i 0. Od njih se može složiti sljedećih 10 brojeva:

Brojevi	a	b	ab	a0	ba	b0	ab0	a0b	ba0	b0a
Rastav	a	b	10a+b	10a	10b+a	10b	100a+10b	100a+b	100b+10a	100b+a

1. način - informatički

Uz pomoć dvije ugniježđene petlje prođemo po svim mogućim trojkama koje čine: dvije znamenke iz petlje i znamenka nula. Za svaku trojku izračunamo sve moguće brojeve (pogledati tablicu) i sve ih zbrojimo. Ako je ukupan zbroj jednak S, onda je to moguća Ivičina trojka. Pseudokod ovog načina dan je u nastavku.

```
za a := 1 do 9 činiti
      za b := a+1 do 9 činiti
            zbroj = 0;
            zbroj = zbroj+(a);
            ...
            zbroj = zbroj+(10*a+b);
            ...
            zbroj = zbroj+(100*b+a);
            ako je zbroj = S onda
                  izlaz(a, b, 0);
```

2. način - matematički

Ukupan zbroj ovih 10 brojeva jednak je 233*(a+b). To znači da su moguće Ivičine trojke one među kojima je znamenka 0 i kojima je zbroj znamenki Z jednak S/233. Sada je lako naći rješenja ovog slučaja.

Uz pomoć dvije ugniježđene petlje prođemo po svim mogućim trojkama, jedna znamenka mora biti 0. Za svaku trojku provjerimo je li zbroj znamenki jednak Z. Ako je, onda je to moguća Ivičina trojka. Pseudokod ovog slučaja dan je u nastavku.

```
Z = S/233;
za a := 1 do 9 činiti
      za b := a+1 do 9 činiti
            ako je a+b+0 = Z onda
                  izlaz(a, b, 0);
```

ZAKLJUČAK

Prođemo li kroz oba slučaja, dobit ćemo sva moguća rješenja. Primijetite također da nije moguće da u oba slučaja naiđemo na isto rješenje, stoga na to ne trebamo obraćati pažnju.

Potrebno znanje: ugniježđena naredba ponavljanja, naredba odlučivanja, osnovne matematičke operacije

Kategorija: ad hoc

5.3. Zadatak: Pogodi — Autor: Antun Razum[3]

Za ovaj zadatak postoji nekoliko mogućih rješenja. Ovdje ćemo razmotriti tri slična rješenja. U odjeljku *Provjera rješenja* i *Pronalazak rješenja* opisani su djelovi rješenja zajednički svim opisanim rješenjima. Odjeljak *Usporedba nizova* dio je koji razlikuje ova rješenja i objašnjen je u trima inačicama od kojih svaka odgovara jednom od mogućih rješenja.

Opis rješenja

Provjera rješenja

Krenimo razmišljati o rješenju s obrnute strane. Da imamo zadan broj X, bismo li znali odrediti je li upravo taj broj rješenje tj. broj koji je Ivica zamislio? Odgovor na ovo pitanje prilično je jednostavan. Učinit ćemo točno ono što piše u zadatku. Za svaki od članova niza A (niza brojeva koje su prijatelji zamislili) izračunamo udaljenost od zadanog broja X i te brojeve zapišemo u neki privremeni niz, nazovimo ga P. Sada usporedimo nizove P i D. Ako niz P sadrži sve brojeve koje sadrži i niz D (računajući broj ponavljanja brojeva koji se pojavljuju više puta), to znači da je upravo taj, zadani X rješenje.

Usporedba nizova

Inačica A

Za ovu inačicu bit će nam potreban još jedan pomoćni niz, nazovimo ga Z. To će biti niz zastavica koji će označavati koji su brojevi niza P već posjećeni. Naravno, sve zastavice na početku će biti spuštene (postavljene na neistinu). Postupak je sada jednostavan. Prolazimo redom po brojevima niza D. Za svaki broj u nizu D pokušamo ga pronaći u nizu P, pritom pazeći da ne gledamo brojeve koji već imaju podignutu zastavicu. Kada ga pronađemo, njegovu zastavicu podignemo (postavimo na istinu) kako ga

[3] student Fakulteta elektrotehnike i računarstva, osvajač srebrnih medalja na IOI 2012. i CEOI 2012., dvostruki državni prvak iz informatike (1. i 3. razred)

ne bismo drugi put ponovo brojili. Ako neki broj niza D ne pronađemo u nizu P koristeći opisan algoritam, to znači da se nizovi ne poklapaju tj. da trenutni X nije rješenje. Ako pak uspijemo proći po svim brojevima niza D i sve ih pronađemo u nizu P, trenutni X je moguće rješenje.

Inačica B

U ovoj inačici brojeve niza uspoređivat ćemo redom, tako što ćemo po brojevima niza P i niza D prolaziti u rastućem poretku. Kako bismo lakše ilustrirali rješenje, zamislimo brojeve niza A na brojevnom pravcu. Također, smjestimo i zadani broj X na taj isti brojevni pravac. Gdje god se broj X u odnosu na niz nalazio, najudaljeniji će mu uvijek biti jedan od krajeva niza, bilo lijevi, bilo desni. Kako bismo se uvjerili u ovo, razmotrimo moguće slučajeve:

1. ako je broj X lijevo od cijelog niza, najdesniji broj niza očito je najdalji,
2. ako je broj X desno od cijelog niza, najljeviji broj niza očito je najdalji,
3. konačno, ako je broj X usred niza, jedan od krajeva niza sigurno je najdalji.

Na ovaj način usporedbom udaljenosti krajeva niza možemo odrediti koji je broj niza najdalji od broja X. Nakon što smo to odredili, spremimo udaljenost tog broja i broja X u pomoćni niz P. Nakon toga maknemo kraj niza koji je udaljeniji iz niza i na isti način možemo odrediti koji je idući najdalji broj od broja X. Kada smo to utvrdili, također spremamo njegovu udaljenost u pomoćni niz P i uklanjamo ga iz niza. Ovaj postupak ponavljamo sve dok nismo prošli sve brojeve niza A. Budući da ovim postupkom prolazimo brojevima niza A od najudaljenijeg broju X do najbližeg, udaljenosti će u nizu P biti sortirane od najveće prema najmanjoj tj. u padajućem poretku. Ako ovaj niz prođemo od kraja prema početku, prošli smo niz udaljenosti u rastućem poretku i to je upravo ono što smo tražili.

Inačica C

Ako je netko upoznat s algoritmom sortiranja niza (ili ako jezik u kojem rješava zadatak posjeduje biblioteku s funkcijom koja to omogućava), problem objašnjen u inačici B može se jednostavno riješiti sortiranjem niza i uspoređivanjem brojeva niza u rastućem poretku.

Pronalazak rješenja

Vratimo se sada na početni problem. Znajući kako za neki X odrediti je li on rješenje, vrlo je lagano pronaći koji je X rješenje zadatka. Jednostavno prođemo po svim mogućim vrijednostima broja X i pronađemo najveću za koju vrijedi da je rješenje. Iz ulaznih podataka vidljivo je da je najveća moguća vrijednost broja X jednaka 200. Ovaj slučaj moguć je ako postoji jedan broj u nizu A čija vrijednost je 100 (što je maksimalna vrijednost) i on je tada udaljen od broja X za 100 (što je maksimalno koliko može biti udaljen).

Pseudokod (inačica B)

```
// najveći mogući broj X
MAKSIMUM := 200;
ulaz(n); ulaz(A); ulaz(D);
// kreće se od najvećeg broja X tako da je prvo pronađeno
// rješenje ujedno i najveće
x := MAKSIMUM;
dok je (istina) činiti {
    // broj pređenih brojeva niza A
```

```
    pozicija := 0;
    // indeks lijevog kraja niza A
    lijevi_indeks := 0;
    // indeks desnog kraja niza A
    desni_indeks := n - 1;
    dok je (pozicija < n) činiti {
        lijevi := x - A[lijevi_indeks]; // udaljenost lijevog kraja niza A
        desni = A[desni_indeks] - x; // udaljenost desnog kraja niza A

        ako je (lijevi > desni) onda {
            trenutni := lijevi; // uzima se lijevi kraj
            lijevi_indeks := lijevi_indeks + 1; // izbaci lijevi kraj iz niza
        }
        inače {
            trenutni := desni; // uzima se desni kraj
            desni_indeks := desni_indeks - 1; // izbaci desni kraj iz niza
        }
        // provjeri je li došlo je do razlike u izračunatoj
        // i zadanoj udaljenosti
        ako je (trenutni != D[n - pozicija - 1]) onda
            prekini petlju;
        pozicija := pozicija + 1;
    }
    // provjeri je li prijeđen cijeli niz A, jer ako jest,
    // trenutni X je rješenje
    ako je (pozicija = n) onda
        prekini petlju;
    x := x - 1;
}
izlaz(x);
```

Napomene

1. Rješenja izvedena samo pomoću naredbi grananja koja ispravno rade za manje vrijednosti broja N pokrivena su parcijalnim bodovima opisanim u bodovanju zadatka.
2. Inačica C usporedbe niza zahtijeva predznanje (sortiranje niza) koje nije propisano za ovu razinu natjecanja. Ono predstavlja alternativno rješenje ako natjecatelj ima takva predznanja. Ako natjecatelj ne posjeduje ta predznanja, zadatak može riješiti koristeći inačicu A, inačicu B ili neko drugo, slično rješenje.

Potrebno znanje: uvjetna naredba ponavljanja, nizovi

Kategorija: ad hoc

6.1. Zadatak: Memory Autor: Antun Razum

Postoji više varijacija rješenja ovoga zadatka. Ovdje ćemo spomenuti nekoliko njih.

Zajedničko svakom od ovih rješenja je učitavanje podataka. Svaki red je trebalo razbiti u tri znamenke koje treba spremiti u tri odvojene cjelobrojne varijable. Ovo smo mogli napraviti na dva načina.

Prvi je učitati cijeli red kao jedan cjelobrojni broj i zatim svaku znamenku dobiti pomoću operacija dijeljenja i ostatka pri dijeljenju s brojevima 100 i 10.

Drugi način je učitati cijeli red kao niz znakova i zatim dobiti znamenke oduzimanjem ASCII vrijednosti znaka znamenke nula od svakog učitanog znaka kako bi se dobila stvarna cjelobrojna vrijednost.

U daljnjem tekstu objašnjavat ćemo različite načine rješavanja zadatka ne obazirući se na detalje vezane za učitavanje podataka.

Jedan od najjednostavnijih načina rješavanja ovog zadatka je sljedeći. Učitamo sve znamenke iz ulaza u jedan niz redom kako se pojavljuju. Uz niz u kojem držimo znamenke napravimo i niz istinitosnih vrijednosti koji na nekoj poziciji ima istinu ako je znamenka na istoj poziciji u prvom nizu sparena.

Ispravne vrijednosti ovog drugog niza dobijemo vrlo jednostavno. Prije svega postavimo sve vrijednosti drugog niza na neistinu. Zatim prođemo jednom for petljom po prvom nizu i zatim drugom ugniježđenom for petljom ponovo prođemo po nizu uspoređujući vrijednosti niza na pozicijama prve i druge petlje. Pritom pazimo da se druga petlja ne nalazi na istoj poziciji kao i prva. Ako drugom petljom pronađemo poziciju na kojoj se nalazi ista znamenka kao ona na poziciji prve petlje, možemo postaviti istinu na poziciju prve petlje u drugom nizu čime naznačujemo da je ta znamenka sparena.

Kada smo tako prošli cijeli niz, provjerimo koja je znamenka nesparena. Postoji samo jedna takva jer je tako zadano u ulazu pa jednostavno možemo for petljom proći po drugom nizu tražeći neistinu. Kada smo je našli, možemo odmah ispisati znamenku koja se nalazi na toj poziciji.

Redak i stupac ove znamenke dobit ćemo na sljedeći način. Primijetimo da znamenke na pozicijama (prvu poziciju označavamo s nula) u nizu 0, 1 i 2 pripadaju prvom retku, 3, 4 i 5 drugom, a 6, 7 i 8 trećem. Možemo vidjeti da je redak zapravo pozicija cjelobrojno podijeljena s brojem 3 i zatim uvećana za 1. Isto tako, vidljivo je da je stupac ostatak pri djeljenju pozicije s 3 uvećan za 1.

Prethodno opisani način također smo mogli riješiti tako da nismo koristili nizove nego matrice. Postupak bi bio vrlo sličan jedino na kraju ne bi trebali računati redak i stupac nesparene znamenke nego bi ga direktno mogli očitati.

Još jedan način na koji smo mogli riješiti ovaj zadatak je sljedeći. Ovaj način će možda nekom teže pasti na pamet, ali ga je lakše za napraviti jednom kada se smisli.

Kao i u prethodna dva načina, učitane znamenke možemo držati u nizu ili u matrici, kako god želimo. Osim niza u kojem držimo znamenke imamo još jedan pomoćni niz koji za svaku znamenku pamti koliko se puta pojavljuje.

Ovaj drugi niz popunimo vrlo jednostavno. Prvo sve vrijednosti postavimo na 0. Sada prođemo for petljom po prvom nizu i poziciju drugog niza koja je vrijednost koja se nalazi na trenutnoj poziciji petlje u prvom nizu uvećamo za jedan. Sada je na svakoj poziciji u drugom nizu jedna od tri moguće

vrijednosti: 0 - znamenka se ne pojavljuje u ulazu, 1 - znamenka se pojavljuje samo jednom tj. nesparena je, 2 - znamenka se pojavlju dva puta tj. sparena je.

Kako bi pronašli nesparenu znamenku, samo u drugom nizu for petljom pronađemo broj 1. Sada još trebamo odrediti njenu poziciju, a to napravimo samo tako što nađemo nesparenu znamenku u prvom nizu.

Redak i stupac znamenke odredimo isto kao i u prvom odnosno drugom načinu, ovisno o tome jesmo li koristili niz ili matricu za spremanje učitanih znamenaka.

Pseudokod je napisan za posljednje opisano rješenje uz korištenje niza za spremanje učitanih podataka. Nizovi će u pseudokodu, kako je opisano i u rješenjima, počinjati s pozicijom nula.

Programski kod (pisan u psudojeziku)

```
ulaz(znamenka); //učitavanje znamenaka jednim od načina opisanih na početku
za i := 0 do 8 činiti
    broj[i] := 0;
za i := 0 do 8 činiti
      broj[znamenka[i]] := broj[znamenka[i]] + 1;
za i := 0 do 9 činiti
    ako je broj[i] = 1 onda
        nesparena := i;
za i := 0 do 8 činiti
    ako je broj[i] = nesparena onda
        pozicija := i;
izlaz(nesparena);
izlaz(pozicija / 3 + 1); // znak '/' predstavlja cjelobrojno dijeljenje
izlaz(pozicija % 3 + 1); // znak '%' predstavlja ostatak pri dijeljenju
```

Potrebno znanje: naredba ponavljanja, naredba odlučivanja, osnovne matematičke operacije, jednostavan rad sa znamenkama ili stringovima

Kategorija: ad hoc

6.2. Zadatak: Autobus

Autor: Adrian Satja Kurdija[4]

Zadatak rješavamo u trima fazama.

Parsiranje ulaznih podataka. Da bismo pojednostavnili računanje, nužno je svako vrijeme iz ulaza pretvoriti u minute (npr. 05:30 → 5 * 60 + 30 = 330). Kako to učiniti? Najprije, vrijeme oblika AB:CD učitamo kao string, iz njega izdvojimo četiri znamenke: A, B, C, D i potom računamo:

broj_sati = A * 10 + B,

broj_minuta = C * 10 + D,

ukupan_broj_minuta = **broj_sati** * 60 + **broj_minuta**.

Posljednji je broj ono što nam zapravo treba i čime ćemo dalje računati.

Pronalazak traženog autobusa. Najlakše je generirati sva vremena kada autobus dolazi i od tih vremena pronaći ono najbliže ciljanom vremenu **S**. Kako to učiniti?

Ako su **T1** i **T2** vremena dolazaka prvog i drugog autobusa u minutama, onda razmak između uzastopnih dolazaka autobusa iznosi **R** = **T2** - **T1** minuta. Sada autobusi dolaze u terminima: **T1**, **T1** + **R**, **T1** + 2 * **R**, **T1** + 3 * **R**, … i možemo ih sve proći uz pomoć petlje. Petlju prekidamo kada vrijeme **T1** + **k** * **R** postane veće od 20 * 60 (tj. 20:00).

Unutar te petlje, za promatrano vrijeme dolaska **T** zanima nas koliko je ono udaljeno od **S**. To je zapravo apsolutna vrijednost razlike **T** - **S**. Tu udaljenost uspoređujemo s dosad najmanjom pronađenom udaljenosti koju pamtimo u pomoćnoj varijabli, kao i odgovarajuće vrijeme **T_best**.

Ispis rezultata. Na koncu broj **T_best** u minutama valja ispisati u obliku **HH:MM**. Nije teško vidjeti da vrijedi:

HH = **T_best** div 60 (količnik cjelobrojnog dijeljenja),

MM = **T_best** mod 60 (ostatak cjelobrojnog dijeljenja).

Brojeve **HH** i **MM** valja ispisati na dvije znamenke čak i ako su jednoznamenkasti (npr. 5 → 05). To možemo riješiti naredbom odlučivanja: ako je broj manji od 10, ispiši najprije nulu.

Programski kod (pisan u psudojeziku)

```
ulaz(A1); ulaz(A2); ulaz(Q);
T1 = pretvori_u_minute(A1);
T2 = pretvori_u_minute(A2);
S = pretvori_u_minute(Q);
dosad_najmanja_udaljenost := 24 * 60;  // na početku neki velik broj
T_best := -1;  // na početku neodređen
T := T1;
dok je T <= 20 * 60 činiti
```

[4] niv. bacc. math., student PMF-MO, višestruki državni prvak iz matematike i informatike, osvajač srebrne i brončane medalje na IOI-ju te dviju brončanih medalja na IMO olimpijadi znanja

```
{
    udaljenost := apsolutna_vrijednost(T - S);
    ako je udaljenost < dosad_najmanja_udaljenost onda
      {
        dosad_najmanja_udaljenost := udaljenost;
        T_best := T;
      }
    T := T + (T2 - T1);  // dodajemo razmak
}
H := T_best div 60;
M := T_best mod 60;
ako je H < 10 onda
    izlaz('0');
izlaz(H);
izlaz(':');
ako je M < 10 onda
    izlaz('0');
izlaz(M);
```

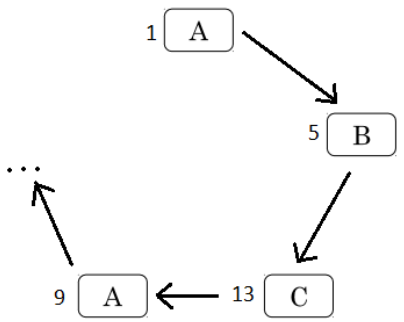
Potrebno znanje: naredba ponavljanja, stringovi, traženje najmanjeg elementa

Kategorija: ad hoc

6.3. Zadatak: Zmije — Autor: Adrian Satja Kurdija

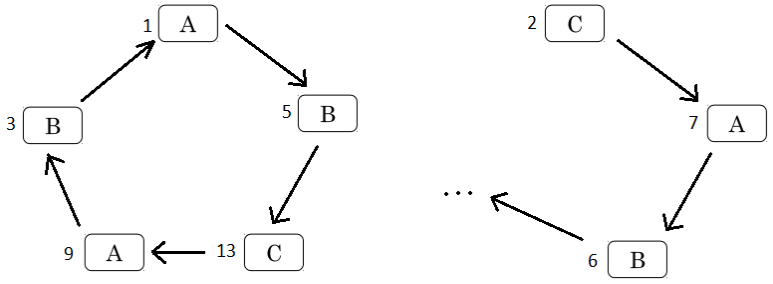
Radi jednostavnosti, osvjetljenja A, B, C zvat ćemo bojama. Boje kaveza spremat ćemo u string koji ćemo na koncu ispisati. Znak na njegovoj **K**-toj poziciji bit će boja kaveza **K**. Na početku se taj string sastoji npr. od **N** znakova 'X', koji označava da kavez još nije obojen.

Obojimo kavez 1 bojom A. Pogledajmo u koji kavez prelazi zmija iz kaveza 1. Neka je to npr. kavez 5. Njega sada ne smijemo obojiti bojom A: obojimo ga stoga bojom B. Idemo dalje: pogledajmo gdje prelazi zmija iz kaveza 5. Neka je to npr. kavez 13. Njega ne smijemo obojiti bojom B: možemo bojom A ili bojom C, ali radi uvjeta 2 obojimo ga bojom C. Ovaj postupak nastavljamo dalje bojeći kaveze redom bojama A, B, C, A, B, C, i tako dalje. Time osiguravamo jednaku zastupljenost boja A, B, C.

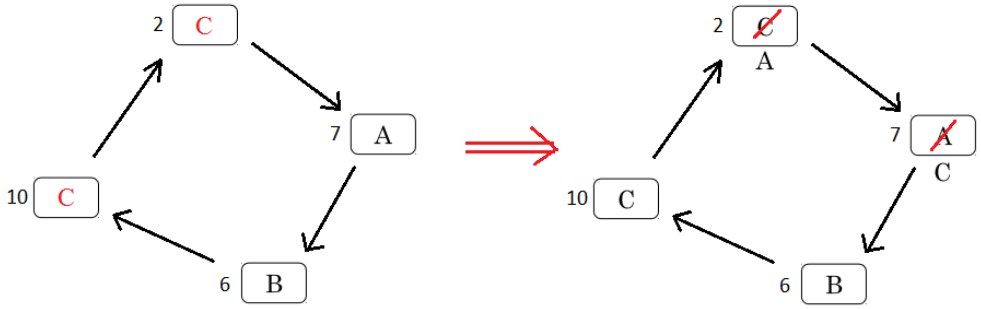


Do kada to činimo? Kad-tad ćemo doći do kaveza iz kojeg zmija prelazi u kavez koji smo već obojili. Taj može biti samo kavez 1, jer inače bi dvije zmije prešle u isti kavez. Time smo zatvorili "krug" kaveza 1, 5, 13, …, 1 koje smo ravnomjerno obojili bojama A, B, C. Taj krug vjerojatno ne sadrži sve kaveze, pa moramo isti postupak provesti za kavez 2 i njegov "krug", pa za kavez 3 i njegov "krug" i tako dalje.

Naravno, ako je kavez 2 već obojen, obojen je i cijeli njegov krug pa ga nema smisla ponovno bojiti. Ako nije, bojimo krug kao prije, ali ne započinjemo nužno od A, nego od one boje na kojoj smo stali u prethodnome krugu. Na primjer, ako je posljednja iskorištena boja B, onda ćemo novi krug započeti bojom C.



Još valja riješiti problem koji se može dogoditi prilikom zatvaranja kruga: što ako se boje prvog i posljednjeg kaveza u krugu podudaraju (npr. A, B, C, A, B, C, A)? To ne smijemo dozvoliti radi uvjeta 1. U tom slučaju zamjenjujemo boje prvih dvaju kaveza u krugu, čuvajući tako ravnopravnost boja.



Programski kod (pisan u psudojeziku)

```
ulaz(N);
za i := 1 do N činiti
ulaz(kamo[i]);


za i := 1 do N činiti
    boja[i] := 'X';


trenutno_slovo := 'A';


// Pomocni niz, sluzi za povecavanje slova
sljedece_slovo['A'] := 'B';
sljedece_slovo['B'] := 'C';
```

```
sljedece_slovo['C'] := 'A';


za i := 1 do N činiti
    ako je boja[i] = 'X' onda
    {
        boja[i] := trenutno_slovo;
        trenutno_slovo = sljedece_slovo[trenutno_slovo];
        j := kamo[i];
        dok je j <> i činiti
        {
            boja[j] := trenutno_slovo;
        trenutno_slovo = sljedece_slovo[trenutno_slovo];
            ako je (kamo[j] = i) i (boja[j] = boja[i]) onda
                zamijeni(boja[i], boja[kamo[i]]);
            j := kamo[j];
        }
}
izlaz(boja);
```

Potrebno znanje: ugniježđena naredba ponavljanja, niz, string

Kategorija: ad hoc

7.1. Zadatak: Brazil Autor: Nikola Dmitrović

U zadatku se definira da će domaćin i gost moći postići najviše 6 golova. Zbog toga postoji konačan broj rezultata kojim utakmica može završiti (počevši od 0:0 do 6:6). Ideja je prebrojiti koliko puta se koji rezultat pojavio te zatim odrediti koji se pojavio najviše puta. Za to ćemo definirati tablicu sa sedam redaka i sedam stupaca te u i-ti redak i j-ti stupac upisati broj pojavljivanja rezultata i:j.

Situacija se komplicira kada trebamo iskoristiti dodatno zadane uvjete prilikom traženja maksimalne vrijednosti.

Programski kod (pisan u Pythonu)

```
N = int(input())
rezultati = []
for i in range(7):
    rezultati += [[0] * 7]
for i in range(N):
    a, b = map(int, input().split())
    rezultati[a][b] += 1
max = x = y = 0
for i in range(7):
    for j in range(7):
        if rezultati[i][j] > max:
            max = rezultati[i][j]
            x = i
            y = j
        else:
            if rezultati[i][j] == max:
                if i + j > x + y:
                    max = rezultati[i][j]
                    x = i
                    y = j
                elif i + j == x + y:
                    if j > y:
                        max = rezultati[i][j]
                        x = i
                        y = j
print(x,y)
```

Potrebno znanje: naredba odlučivanja, naredba ponavljanja, dvodimenzionalna tablica, traženje maksimalne vrijednosti

Kategorija: ad hoc

7.2. Zadatak: ANK

Autor: Adrian Satja Kurdija

Neka su A i B neka dva uzastopna broja u prvome ANK-u. Ako su oni uzastopni i u drugome ANK-u, onda oni mogu pripadati istome komadu: između njih ne trebamo rezati. Ako pak u drugome ANK-u nisu uzastopni, očito moramo napraviti rez između njih u prvome ANK-u pa povećavamo traženi broj rezova i označavamo rez u pomoćnome nizu.

Ponavljanjem ovakvog zaključivanja za svaka dva uzastopna broja prvoga ANK-a nalazimo sve tražene rezove i njihova mjesta. Za pronalazak mjesta spojeva u drugome ANK razmišljamo potpuno analogno (vidi pseudokod).

Jednu smo stvar prešutjeli: kako pronaći gdje se broj A iz prvog ANK-a nalazi u drugom ANK-u? Tražimo li ga for-petljom, broj operacija raste na oko **N** * **N** jer provjeravamo čitav drugi ANK (**N** operacija) i to za svaki A iz prvog ANK (kojih također ima **N**). Budući da **N** može biti jako velik, prekoračujemo tako vremensko ograničenje. Rješenje je koristiti pomoćni niz u koji ćemo na poziciju A spremiti lokaciju broja A u drugome ANK-u. Taj pomoćni niz napunit ćemo samo jednom, na početku.

Programski kod (pisan u psudojeziku)

```
ulaz(N);
za i := 1 do N činiti
{
      ulaz(prvi[i]);
      lokacija_u_prvome[ prvi[i] ] := i;  // pomocni niz
}
za i := 1 do N činiti
{
      ulaz(drugi[i]);
      lokacija_u_drugome[ drugi[i] ] := i;  // pomocni niz
}
broj_rezova := 0;
za i := 1 do N - 1 činiti
{
    A := prvi[i];
    B := prvi[i + 1];
    ako je lokacija_u_drugome[A] + 1 <> lokacija_u_drugome[B] onda
    {
        broj_rezova := broj_rezova + 1;
        rez[i] := istina;
    }
}
za i := 1 do N - 1 činiti
```

```
{
    A := drugi[i];
    B := drugi[i + 1];
    ako je lokacija_u_prvome[A] + 1 <> lokacija_u_prvome[B] onda
        spoj[i] := istina;
}
izlaz(broj_rezova + 1);
za i := 1 do N činiti
    ako je rez[i] = istina onda
        izlaz(i);
za i := 1 do N činiti
    ako je spoj[i] = istina onda
        izlaz(i);
```

Potrebno znanje: rad s nizovima

Kategorija: ad hoc

7.3. Zadatak: Paula — Autor: Matija Milišić

Rješenje ovog zadatka zahtijeva nekoliko zaključaka, stoga ćemo do njega dolaziti postupno.

Primijetimo prvo da su izjave koje se odnose na različite elemente niza nezavisne, odnosno da neka izjava za i-ti element nikako ne utječe na j-ti element niza (gdje je i≠j). To znači da na samom početku možemo rasporediti izjave na elemente niza koje one pogađaju. To radimo simulirajući tri pravila opisana u tekstu zadatka.

Sada možemo zamisliti kao da niz ima samo jedan element i da imamo izjave za taj element na temelju kojih moramo naći minimalnu i maksimalnu moguću vrijednost. To smijemo napraviti jer se zadatak u kojem niz ima više elemenata rješava na isti način ponavljajući istu stvar za svaki član niza.

Rješenje koje je lako primijetiti je isprobati sve brojeve između 1 i **K** i za svaki broj provjeriti odgovara li svim izjavama. Zatim, među svim brojevima koji odgovaraju izjavama naći minimalni i maksimalni i njih ispisati. Međutim, takvo rješenje je presporo jer brojeva za koje treba isprobati sve izjave ima jako puno. Stoga za postizanje maksimalnog broj bodova moramo zaključivati dalje.

Riješimo prvo najjednostavniji slučaj, kada nema nijedne izjave. Tada je minimalna moguća vrijednost elementa 1, a maksimalna K jer je Marin zamislio brojeve između 1 i **K**.

Riješimo sada isti zadatak, ali u kojem ne postoji izjava oblika „= X NE“. Kasnije ćemo posebno razmotriti taj oblik izjave.

Primijetimo da sve vrste operatora možemo svesti na samo dvije vrste: „>=“ i „<=“. Pogledajmo to na sljedećim primjerima:

Izjava	Pretvorba	Primjer izjave	Primjer pretvorbe
< X DA	<= X-1 DA	< 7 DA	<= 6 DA
> X DA	>= X+1 DA	> 7 DA	>= 8 DA
= X DA	>= X DA i <= X DA	= 7 DA	>= 7 DA i <= 7 DA
< X NE	<= X-1 NE	< 7 NE	<= 6 NE
> X NE	>= X+1 NE	> 7 NE	>= 8 NE

Primijetimo nadalje da se sve izjave s odgovorom „NE“ mogu svesti na izjave s odgovorom „DA“. Pogledajmo to na sljedećim primjerima:

Izjava	Pretvorba	Primjer izjave	Primjer pretvorbe
<= X NE	> X DA	<= 7 NE	> 7 DA
>= X NE	< X DA	>= 7 NE	< 7 DA

Kombinirajući gornja dva zaključka sve izjave možemo svesti na samo dva oblika: „>= X DA“ i „<= X DA“. Kada obavimo pretvorbe, nađimo **najveći** broj uz operator „>=” (nazovimo ga DONJI) i **najmanji** broj uz operator „<=” (nazovimo ga GORNJI).

Broj koji odgovara svim izjavama mora se nalaziti između granica DONJI i GORNJI. Dodatno, vrijedi i: svaki broj koji se nalazi između granica DONJI i GORNJI odgovara svim izjavama upravo zbog načina na koji su te granice odabrane. Napokon zaključujemo da je minimalna moguća vrijednost broj DONJI, a maksimalna moguća vrijednost broj GORNJI.

Sada još moramo proširiti rješenje tako da prihvaća izjavu oblika „= X NE“. Kod te izjave je problem što je ne možemo svesti na oblike „>= X DA“ i „<= X DA“, jer je moguće da dođe do grananja (ili - ili) što komplicira stvari. Pogledajmo to na sljedećem primjeru:

Izjava	Pretvorba	Primjer izjave	Primjer pretvorbe
= X NE	ili < X DA ili > X DA	= 7 NE	ili < 7 DA ili > 7 DA

Da bismo mogli prihvatiti taj oblik izjave, morat ćemo uvesti niz JEDNAKO_NE[K] veličine **K** u koji ćemo zapisivati koju vrijednost element niza ne može poprimiti. Broj 1 na nekoj poziciji označavat će da element niza ne može poprimiti tu vrijednost. Na početku je na svim pozicijama broj 0. Pogledajmo sljedeći niz izjava i stanje našeg niza nakon pojedine izjave (**K** = 4):

Izjava	Stanje niza `JEDNAKO_NE`
	0, 0, 0, 0
„= 4 NE“	0, 0, 0, 1
„= 2 NE“	0, 1, 0, 1
„= 4 NE“	0, 1, 0, 1

Cjelokupno rješenje radimo na sljedeći način. Prolazimo kroz sve izjave, ako je izjava oblika „= X NE“ zapisujemo to u niz JEDNAKO_NE[X], a za sve ostale oblike izjava postupamo na način opisan u prvom dijelu (računamo granice DONJI i GORNJI).

Minimalnu moguću vrijednost nalazimo tako da prolazimo od DONJE do GORNJE granice i tražimo prvu vrijednost X za koju je JEDNAKO_NE[X] = 0.

Maksimalnu moguću vrijednost nalazimo na sličan način. Prolazimo od GORNJE do DONJE granice i tražimo prvu vrijednost X za koju je JEDNAKO_NE[X] = 0.

Vidljivo je da je ovo rješenje puno brže od onog danog na samom početku jer nećemo morati za svaki element X od 1 do K provjeriti svaku izjavu već samo vrijednost niza JEDNAKO_NE na poziciji X.

Programski kod (pisan u psudojeziku)

```
za svaku izjavu
      odredi koji element niza pogađa;
za svaki element i niza
      DONJI = 1;
      GORNJI = K;
      postavi sve elemente niza JEDNAKO_NE na 0
      za svaku izjavu koja pogađa taj element
            ako je izjava = „= X NE“
                  JEDNAKO_NE[X] = 1;
            inače
                  pretvori izjavu u oblik „>= X DA“ i „<= X DA“
                  ako je izjava = „>= X DA“
                        DONJI = max(DONJI, X);
```

```
                    ako je izjava = „<= X DA“
                          GORNJI = min(GORNJI, X);
        za svaki X od DONJI do GORNJI
              ako je JEDNAKO_NE[X] = 0
                    MIN[i] = X;
                    izađi iz petlje;
        za svaki X od GORNJI do DONJI
              ako je JEDNAKO_NE[X] = 0
                    MAX[i] = X;
                    izađi iz petlje;
za svaki element i niza
        izlaz(MIN[i]);
za svaki element i niza
        izlaz(MAX[i]);
```

Potrebno znanje: rad s nizovima

Kategorija: ad hoc

8.1. Zadatak: Sokol Autor: Nikola Dmitrović

Ukažimo na jedno jednostavno rješenje. Od danih osam točaka, sortiramo posebno x-koordinate i posebno y-koordinate. Sada su srednje dvije x-koordinate (u oznaci x_1 i x_2) u tako sortiranom slijedu naši x_lijevi i x_desni, a srednje dvije y-koordinate (u oznaci y_1 i y_2) su y_donji i y_gornji pravokutnika koji nas zanima.

Nakon što se odrede te četiri točke, lako se odredi je li loptica pala u teren ili ne.

Programski kod (pisan u psudojeziku)

```
učitaj(niz_x_koordinata);
učitaj(niz_y_koordinata);
sortiraj(niz_x_koordinata);
sortiraj(niz_y_koordinata);
x1 = niz_x_koordinata[4];
x2 = niz_x_koordinata[5];
y1 = niz_y_koordinata[4];
y2 = niz_y_koordinata[5];
učitaj(N);
za i := 1 do N činiti
{
      učitaj (x, y);
      ako je (x1 <= x i x <= x2 i y1 <= y i y <= y2) tada
            ispiši('DA')
      inače
            ispiši('NE')
}
```

Potrebno znanje: algoritam sortiranja, naredba odlučivanja i ponavljanja

Kategorija: ad hoc

8.2. Zadatak: Hanžek Autor: Nikola Dmitrović

Prvo ćemo u dva niza (*U100*, *U110*) učitati startne brojeve sprintera kako je definirano u ulaznim podacima. Prvih K-a članova niza *U100* i *U110* treba prepisati u novi niz *pozvani*, pri čemu treba paziti da se neki sprinter ne prepiše dva puta. Uporedo s prepisivanjem treba brojati koliko puta je došlo do ponavljanja.

```
učitaj(N); učitaj(K); učitaj(U100); učitaj(U110)

pozvani := []; preklapanja := 0

za i := 1 do K činiti
    ako je U100[i]_nije_u_pozvani tada
            dodaj_U100[i]_u_pozvani
    inače
            broj_preklapanja += 1

    ako je U110[i]_nije_u_pozvani tada
            dodaj_U110[i]_u_pozvani
    inače
            broj_preklapanja += 1
```

Ako je broj preklapanja veći od nule, u niz *pozvani* prepisujemo prvog sprintera iz niza U100 na kojeg naiđemo desno od K-te pozicije, a koji već nije zapisan u niz *pozvani*. Pri tome trebamo paziti jesmo li došli do kraja. Isto to napravimo i s nizom U110. Postupak ponavljamo sve dok je broj preklapanja veći od nule.

```
trenutni_sprinter_100 := K + 1;
trenutni_sprinter_110 := K + 1;
dok je broj_preklapanja > 0 radi
     dok je trenutni_sprinter_100<N i U100[trenutni_sprinter_100]_u_pozvani radi
            trenutni_sprinter_100 += 1;
     ako je U100[trenutni_sprinter_100])_nije_u_pozvani tada{
            dodaj_U100[trenutni_sprinter_100]_u_pozvani
            broj_preklapanja -= 1
     }
     ako je broj_preklapanja = 0 tada izadji_iz_petlje;
     dok je trenutni_sprinter_110<N i U110[trenutni_sprinter_110]_u_pozvani radi
            trenutni_sprinter_110 += 1
     ako je U110[trenutni_sprinter_110])_nije_u_pozvani tada {
            dodaj_U110[trenutni_sprinter_110]_u_pozvani
            preklapanja -= 1
     }
sortiraj_pozvani;
ispiši(pozvani);
```

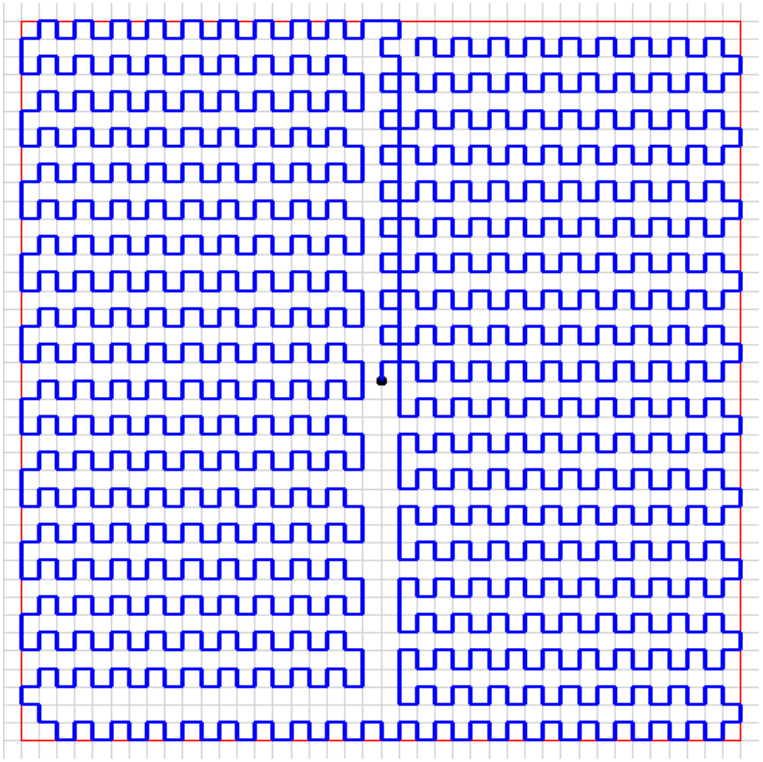
Potrebno znanje: uvjetna naredba ponavljanja, naredba odlučivanja, nizovi

Kategorija: ad hoc simulacija

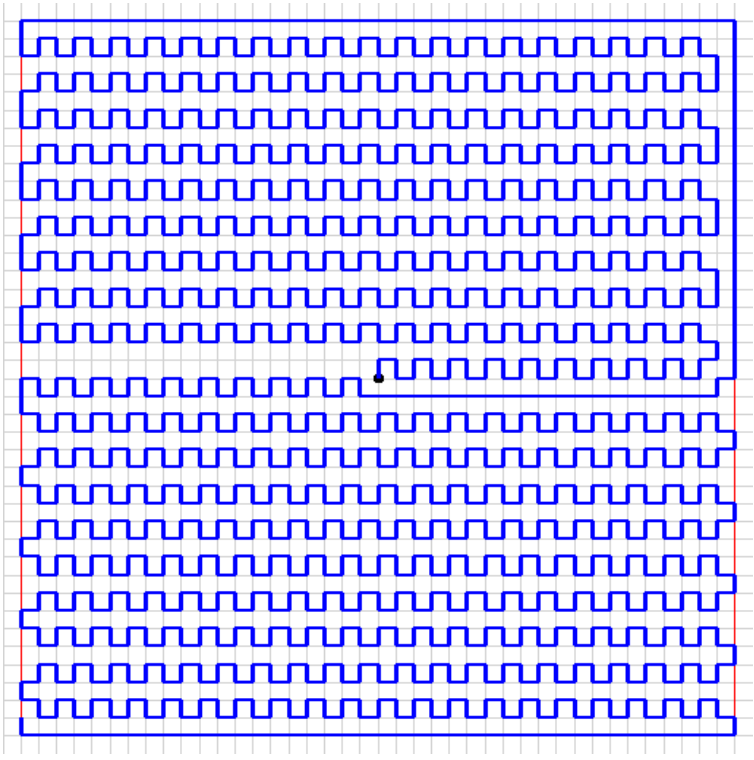
8.3. Zadatak: Šetnja

Autor: Adrian Satja Kurdija

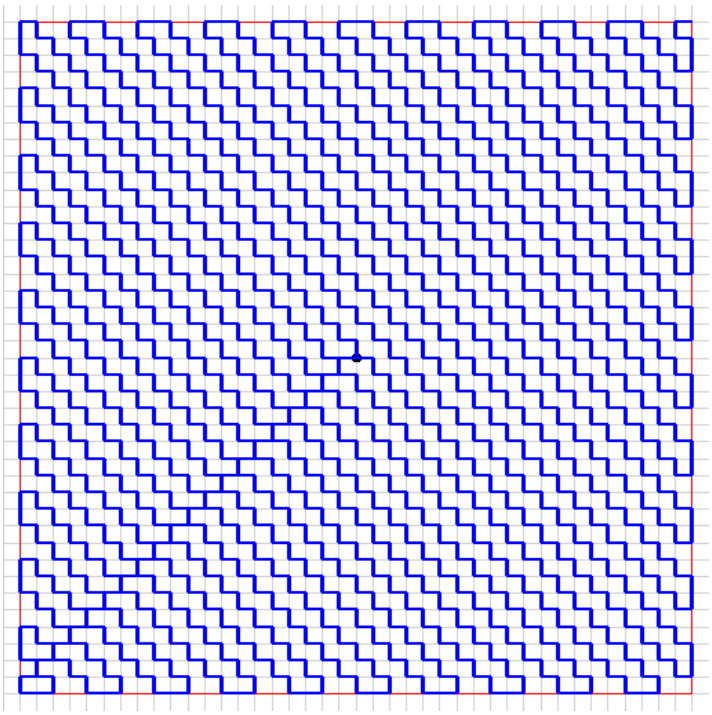
Postoji mnogo različitih rješenja koja nose visok broj bodova. Slikama ćemo prikazati četiri takva.



Rješenje natjecatelja Vilima Lendvaja (71/90 bodova)

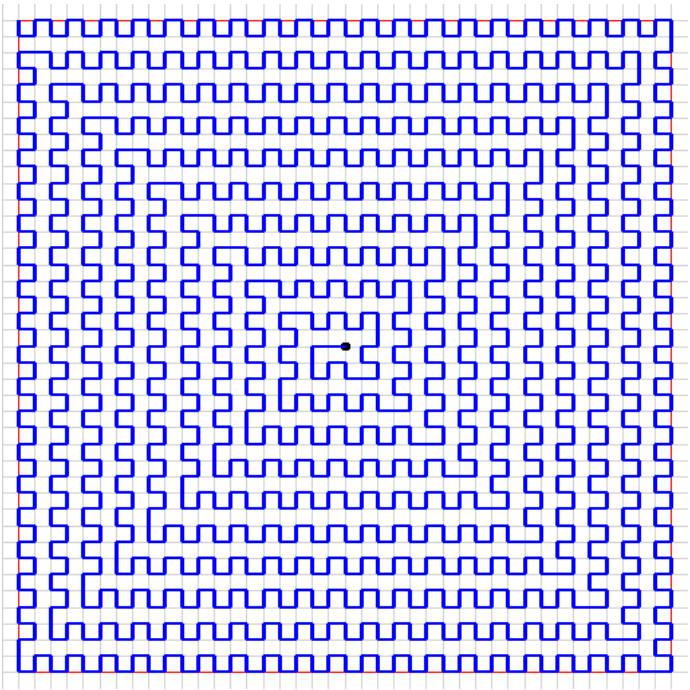


Rješenje Antuna Razuma (80/90 bodova)



Rješenje Adriana Satje Kurdije (85/90 bodova)

Ideja je Adrianovog rješenja ići po dijagonalama, budući da se one međusobno ne sijeku, a donose velik broj skretanja. Da bi se sve dijagonale mogle obići, prvo se spuštamo u donji lijevi kut kvadrata. Tako gubimo neke bodove zbog dvostrukoga posjećivanja istih polja, ali takvih je polja malo pa je postupak isplativ.



Rješenje Matije Milišića (90/90 bodova)

Ideja je ovog rješenja nacrtati "spiralu" punu skretanja. Da bi se olakšala implementacija, krećemo iz gornjeg lijevog kuta i završavamo u ishodištu, a na kraju "izvrnemo" put (tako da počinje u ishodištu). Zašto tako činimo? Nakon što obiđemo vanjski rub kvadrata, svodimo problem na manji (**N** - 1) pa

možemo ponavljati taj postupak dok ne dođemo do ishodišta. To je mnogo jednostavnije nego da spiralu gradimo iz središta prema van.

Za bilo koje od ovih rješenja potrebna je vrlo pažljiva implementacija, ali skripta za testiranje koju su natjecatelji imali na raspolaganju uvelike olakšava posao: svaka pogreška u implementaciji lako se uočava na crtežu.

Razne Python kodove vezane uz ovaj zadatak možete pronaći na http://hsin.hr/~satja/setnja.html.

Potrebno znanje: dizajn algoritma

Kategorija: ad hoc