19. ožujak 2013.

Infokup 2013

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

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

# OBJAŠNJENJA ZADATAKA

Agencija za odgoj i obrazovanje
Education and Teacher Training Agency

MINISTARSTVO ZNANOSTI, OBRAZOVANJA
I ŠPORTA REPUBLIKE HRVATSKE

## 5.1. Zadatak: OP

Autor: Nikola Dmitrović[1]

Kako se u uvjetima zadatka izraz ograničava na izraz oblika **A op1 B op2 C**, tada rješenje možemo bazirati na provjeri svim slučajeva koji se mogu dogoditi. Postoji 16 različitih slučajeva, npr. A+B+C, A+B*C. Posebno treba obratiti pažnju na operator djeljenja koji u rješenju mora biti predstavljen operatorom cjelobrojnog djeljenja. Isto tako treba paziti na postavljanje zagrada prilikom izvršavanja izraza zbog drugačijih prioriteta operatora koji su definirani u zadatku.

Pseudokod rješenja:

```
 //a,b,c prirodni brojevi; o1, o2 znakovi
 učitaj(a); učitaj (o1); učitaj (b); učitaj (o2); učitaj (c);
ako je o1=o2 tada
{
   ako je o1='+' tada izraz:=a+b+c;
   ako je o1='-' tada izraz:=a-b-c;
   ako je o1='*' tada izraz:=a*b*c;
   ako je o1='/' tada izraz:=a div b div c;
}
inače
{
    ako je (o1='+') i (o2='-') tada izraz:=a+b-c;
    ako je (o1='-') i (o2='+') tada izraz:=a-b+c;
    ako je (o1='*') i (o2='/') tada izraz:=a * b div c;
    ako je (o1='/') i (o2='*') tada izraz:=a div b * c;

    ako je (o1='+') i (o2='*') tada izraz:=(a + b) * c;
    ako je (o1='*') i (o2='+') tada izraz:=a * (b + c);
    ako je (o1='+') i (o2='/') tada izraz:=(a + b) div c;
    ako je (o1='/') i (o2='+') tada izraz:=a div (b + c);
    ako je (o1='-') i (o2='*') tada izraz:=(a - b) * c;
    ako je (o1='*') i (o2='-') tada izraz:=a * (b - c);
    ako je (o1='-') i (o2='/') tada izraz:=(a - b) div c;
    ako je (o1='/') i (o2='-') tada izraz:=a div (b - c);
};
ispiši(izraz);
```

**Potrebno znanje:** naredba učitavanja i ispisivanja podataka, naredba odlučivanja

**Kategorija:** ad hoc

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

## 5.2. Zadatak: Slalom — Autor: Nikola Dmitrović

Ovaj zadatak je tipična jednostavna simulacija. Najbolja metoda koju možemo primjeniti prilikom rješavanja je da zamislimo sami sebe kako sjedimo pored TV-a ili još bolje da smo osobno prisutni navedenoj utrci u Lenzerheideu. U tom trenutku, tražimo rješenje navedenog problema kao da nismo na natjecanju već navijamo za našeg Ivicu i strepimo koje će mjesto na kraju zauzeti. Razmišljanje bi moglo ovako izgledati:

- znamo koji startni broj ima naš Ivica. Do njegovog startnog broja samo pratimo što se dešava;
- nakon Ivičine vožnje, zapamtimo koje je mjesto on zauzeo;
- za svakog sljedećeg skijaša koji starta poslije Ivice, pratimo koje je mjesto zauzeo. Ako je to mjesto bolje ili jednako od trenutnog mjesta našeg Ivice, tada trenutno Ivičino mjesto povećamo za jedan.

Pseudokod rješenja:

```
 učitaj(N); // N je broj skijaša u utrci
učitaj (X); // X je startni broj našeg Ivice
za i:=1 do N radi
{
     učitaj (trenutno); // mjesto koje zauzima i-ti skijaš
     ako je i=X tada    // ako je ovaj skijaš Ivica
        ivica:=trenutno;
     ako je i>X tada    // starta skijaš poslije Ivice
        ako je trenutno<=ivica tada // ako je bolji od Ivice
           ivica:=ivica+1; // pomakni Ivicu za jedno mjesto
};
ispiši(ivica);
```

**Potrebno znanje:** naredba učitavanja i ispisivanja, naredba ponavljanja i naredba odlučivanja

**Kategorija:** ad hoc, jednostavna simulacija

## 5.3. Zadatak: Bomboni — Autor: Antun Razum[2]

Prvo što je potrebno primijetiti u ovom zadatku je da pojedine vrste bombona možemo promatrati odvojeno. Kada to znamo, potrebno je samo za svaku pojedinu vrstu bombona odrediti koliko ih je najviše sa sigurnošću moglo biti te onda dobivene brojeve zbrojiti kako bi dobili traženi najveći mogući ukupan broj bombona. Pojedine vrste bombona je sigurno bilo barem onoliko koliko je ima u vrećici u kojoj je najbrojnija, tako da najviše za taj broj bombona te vrste možemo sa sigurnošću tvrditi da je bio u vrećici prije nego se na njoj pojavila rupa.

Rješenje zadatka implementirat ćemo na sljedeći način. Imat ćemo po jednu pomoćnu varijablu za svaku vrstu bombona kako bi pamtili koliko je najviše bombona te vrste do sad bilo. Proći ćemo po

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

svim vrećicama te ćemo za trenutnu vrećicu zapamtiti koliko ima bombona pojedine vrste u njoj. Nakon što smo prebrojali bombone u trenutnoj vrećici, za svaku vrstu bombona, uspoređujući njegov broj s vrijednošću pomoćne varijable, provjerit ćemo je li ovo vrećica u kojoj se nalazi najviše bombona te vrste do sada. Ako je, onda ćemo zapamtiti taj najveći trenutni broj za tu vrstu bombona u pomoćnoj varijabli. Na posljetku, vrijednosti pomoćnih varijabli u kojima pamtimo najveće brojeve bombona koje smo našli jednostavno zbrojimo čime dobivamo traženi broj iz zadatka.

**Potrebno znanje:** naredba odlučivanja, naredba ponavljanja

**Kategorija:** ad hoc

## 6.1. Zadatak: Golum

Autor: Nikola Dmitrović

Golum je slojevit zadatak za čije rješenje je bolje potrošiti više vremena na razmišljanje nego na kodiranje. Samo tako se može uočiti da zadatak treba podijeliti na slučajeve i na taj način pojednostaviti rješenje, povećati mogućnost dobijanja parcijalnih bodova i smanjiti mogućnost pogreške.

Naravno, postoje i rješenja koja mogu općenito riješiti ovaj problem (za svaki prirodan broj), ali zbog dogovorenog opsega znanja njih ovdje nećemo promatrati.

Za dobiveni broj treba napraviti rekonstrukciju svih riječi od kojih je Golum mogao krenuti tj. moramo rastaviti broj na znamenke te od njih kreirati svaku kombinaciju jednoznamenkastih i dvoznamenkastih brojeva. Podijelimo brojeve u 4 grupe, kada je broj jednoznamenkast (A, jedna kombinacija), dvoznamenkast (AB, dvije kombinacije), troznamenkast (ABC, tri kombinacije) i četveroznamenkast (ABCD, pet kombinacija).

| **BROJ** | **A** | **AB** | **ABC** | **ABCD** |
|---|---|---|---|---|
| **Rekonstrukcija** | A | A-B | A-B-C | A-B-C-D |
| | | AB | AB-C | AB-C-D |
| | | | A-BC | A-BC-D |
| | | | | A-B-CD |
| | | | | AB-CD |

Neka kombinacija rastava broja ne predstavlja riječ ako se u njoj pojavljuje dvoznamenkast broj strogo veći od 25. Očito je da je najduža riječ koju je Golum zamislio upravo ona koja se sastoji samo od jednoznamenkastih brojeva. Slovo koje odgovara nekom broju možemo dobiti provjeravanjem svih slučajeva ili korištenjem znanja o ASCII tablici i naredbama pretvaranja broja u znak.

Dodatni problem u zadatku je, npr. test primjer oblika 1001. U ovom slučaju, možemo složiti sljedeće kombinacije: 1-0-0-1, 10-0-1, 1-00-1, 1-0-01, 10-01. Treba uočiti da od treće, četvrte i pete kombinacije nije moguće složiti riječ jer broj 00 i 01 nemaju svoje pripadno slovo. Na ovu situaciju je dodatno trebalo paziti prilikom kodiranja.

Pseudokod rješenja:

```
učitaj(n);
ako je n<=9 tada // jednoznamenkast broj
{
    ispiši(1);
    ispiši (chr(n+65));//chr vraća odgovarajući znak u ASCII tablici
}
inače
   ako je n<=99 tada // dvoznamenkast broj
   {
        ako je n<=25 tada
           ispiši (2)
        inače
```

```
            ispiši (1);
         ispiši (chr(n div 10+65),chr(n mod 10+65));
    }
    inače
        ako je n<=999 tada // troznamenkast broj
        {
             // sdj=s d j, sd j, s dj
             j:=n mod 10; d:=(n div 10) mod 10; s:=n div 100;
             koliko:=3;
             ako je (s*10+d>25) tada
                  koliko:=koliko-1;
             ako je (d*10+j>25) or (d=0) tada // d=0 je slučaj 00, 0X
                  koliko:=koliko-1;
             ispiši (koliko);
             ispiši (chr(s+65),chr(d+65),chr(j+65));
        }
        inače
        ako je n<=9999 tada // četveroznamenkast broj
        {
             // tsdj=t s d j, ts d j, t sd j, t s dj, ts dj
             j:=n mod 10; d:=(n div 10) mod 10;
            s:=(n div 100) mod 10; t:=n div 1000;
             koliko:=5;
             ako je (t*10+s>25) tada
                  koliko:=koliko-1;
             ako je (s*10+d>25) or (s=0) tada
                  koliko:=koliko-1;
             ako je (d*10+j>25) or (d=0) tada
                  koliko:=koliko-1;
             ako je (t*10+s>25) or (d*10+j>25) or (d=0) tada
                  koliko:=koliko-1;
             ispiši (koliko);
             ispiši (chr(t+65),chr(s+65),chr(d+65),chr(j+65));
        };
```

**Potrebno znanje:** rad sa znamenkama, naredba odlučivanja

**Kategorija:** ad hoc

## 6.2. Zadatak: ZET — Autor: Nikola Dmitrović

Zadatak možemo riješiti simulacijom navedenih uvjeta. Znamo da postoji Z2 putnika (najviše 100) koji mogu ulaziti i izlaziti iz tramvaja na svakoj od N stanica (najviše 10) kroz bilo koja od šest vrata na tramvaju. Zadatak garantira nekoliko stvari koje nam mogu olakšati rješavanje. Tramvaj će prilikom dolaska na prvu stanicu biti prazan kao i prilikom odlaska sa zadnje stanice. Putnici u tramvaj ulaze i izlaze samo na stanicama te da će putnik **sigurno prije ući** u tramvaj nego što će iz njega izaći.

Kreiramo niz maksimalne duljine 100 te svaku komponentu u nizu inicijaliziramo na vrijednost nula. Na taj način označavamo da su svi putnici trenutno van tramvaja. Kada učitamo vrijednost jednog zapisa (zapis kaže da je na stanici Z1 u tramvaj ušao/izašao putnik Z2 kroz vrata Z3) trebamo provjeriti je li taj putnik već u tramvaju ili nije. Ako je vrijednost komponente u nizu na poziciji Z2 jednaka nula, to znači da putnik ulazi u tramvaj. Tada na tu poziciju upišemo oznaku stanice i vrata u obliku Z1*10+Z3. Ako je vrijednost različita od nule, to znači da je putnik već u tramvaju i da trenutno izlazi iz njega. Tada na osnovu podatka koji već piše u nizu i novih podataka odredimo cijenu prijevoza i prijeđeni broj stanica te na taj način lako dobijemo ukupnu vrijednost prijevoza. Nakon toga je nužno vratiti vrijednost u nizu na nulu jer je putnik izašao iz tramvaja.

Kada u zadatku nije navedeno da se nešto ne može dogoditi, tada se obavezno treba pretpostaviti da se ta mogućnost može dogoditi. U ovom zadatku trebalo je paziti na dvije dodatne situacije. Naime, putnik je mogao na istoj stanici ući i izaći iz tramvaja te je isto tako mogao više puta u njega ući i izaći (naravno, poštujući uvjet zadatka da će prvo ući pa zatim izaći).

Naravno, u rješenju ovog zadatka, umjesto jednog možete koristiti više nizova (pa i dvodimenzionalni niz) te u svaki posebno upisivati vrijednost stanice, vrata i putnika.

Pseudokod rješenja:

```
 učitaj(N); // broj stanica, nepotreban u ovom rješenju
učitaj (X); // cijena prijevoza
učitaj (Z); // broj zapisa

za i:=1 do 100 radi // prazan tramvaj
    putnik[i]:=0;

ukupno:=0; //ukupna ZET-ova zarada
za i:=1 do Z radi
{
     učitaj (Z1,Z2,Z3); // Z1 stanica, Z2 putnik, Z3 senzor

     ako je putnik[Z2]=0 tada
          putnik[Z2]:=Z1*10+Z3 // zabilježimo stanicu i vrata
     inače
     {
         ulaz:=putnik[Z2] mod 10; izlaz:=Z3;
         ako je (ulaz mod 2=0 ) and (izlaz mod 2=1 ) then
             cijena:=X;
         ako je (ulaz mod 2=0 ) and (izlaz mod 2=0 ) then
             cijena:=2*X;
```

```
            ako je (ulaz mod 2=1 ) and (izlaz mod 2=1 ) then
                cijena:=3*X;
            ako je (ulaz mod 2=1 ) and (izlaz mod 2=0 ) then
                cijena:=4*X;
            ukupno:=ukupno+(Z1-putnik[Z2] div 10)*cijena;
            putnik[Z2]:=0;
        };
    };
    ispiši(ukupno);
```

**Potrebno znanje:** naredba učitavanja i ispisivanja, naredba odlučivanja, naredba ponavljanja, niz

**Kategorija:** ad hoc

## 6.3. Zadatak: CD — Autor: Antun Razum

Za početak, radi jednostavnosti, promotrimo zadatak uz uvjet **X = 0**. Cilj ovog zadatka tada se svodi na to da uz minimalni broj korištenih stalaka pronađemo način na koji ćemo složiti CD-e na te stalke tako da budu složeni uzlazno po polumjeru od vrha prema dnu. Pri dodavanju novog CD-a na jedan od stalaka ponekad postoje stalci na koje ga ne možemo staviti jer se na njihovim vrhovima nalaze CD-i manjeg polumjera. Kada je broj stalaka na koji ne možemo staviti taj CD jednak ukupnom trenutnom broju stalaka koji se koriste (tj. CD ne možemo staviti ni na jedan neprazan stalak), morat ćemo iskoristiti novi, prazan stalak. Nama je u ovom zadatku cilj što je više moguće smanjiti broj ovakvih situacija. To ćemo postići tako da svaki CD stavimo na stalak na čijem se vrhu nalazi CD najmanjeg polumjera koji je veći ili jedan polumjeru trenutnog CD-a. Da je ovo najbolji mogući način slaganja CD-a dokazat ćemo ovako.

Recimo da trenutno koristimo dva stalka na čijim se vrhovima nalaze CD-i polumjera C i C + R, gdje je R prirodan broj, a polumjer CD-a kojeg želimo dodati je A, i vrijedi $A \leq C$. Očito je iz prethodnog uvjeta da trenutni CD možemo dodati na bilo koji od ova dva stalka. Recimo sada da je idući CD kojeg trebamo dodati polumjera B i vrijedi $C + R \geq B > C$. Kada bi prvi CD stavili na stalak s većim CD-om na vrhu drugi CD bi morali dodati na novi stalak, a u suprotnom bi ga mogli dodati upravo na taj stalak s većim CD-om na vrhu. Ovaj dokaz intuitivno vrijedi i u općem slučaju s proizvoljnim brojem stalaka na izboru ako promotrimo neke veće primjere.

Sada kada znamo riješiti zadatak uz postavljeni uvjet X = 0, rješenje nije teško proširiti na puni zadatak. Jedino što se zapravo mijenja je upravo ono što piše u zadatku, a to je da CD sada možemo staviti na sve CD-e koji su polumjera A – X ili veći, gdje je A polumjer trenutnog CD-a.

**Potrebno znanje:** naredba odlučivanja, naredba ponavljanja, nizovi

**Kategorija:** ad hoc

## 7.1. Zadatak: Bang — Autor: Nikola Dmitrović

Na početku definiramo jedno konstantno dvodimenzionalno polje s pet redaka i pet stupaca koje će na poziciji [i,j] imati upisanu vrijednost jedan ako je i-ti odabir jači od j-tog odabira ili vrijednost nula ako nije. Npr., na poziciji [1,3] treba upisati jedinicu jer je kamen jači od škara, a na poziciju [1,5] treba upisati nulu jer je kamen slabiji od Spocka. Uočimo da su vrijednosti na glavnoj dijagonali uvijek jednake nuli.

Za svaku odigranu rundu, na osnovu kreirane tablice, treba za svakog igrača odrediti kako je prošao u dvoboju sa svakim od preostala 4 igrača.

Moguće je i rješenje koje ne zahtjeva navedenu tablicu već se odnos između dva odabira može provjeriti naredbom odlučivanja.

Pseudokod rješenja:

```
stanje[5][5] = {{0, 0, 1, 1, 0}, {1, 0, 0, 0, 1}, {0, 1, 0, 1, 0},
                {0, 1, 0, 0, 1}, {1, 0, 1, 0, 0}};
učitaj(n);
za f:=1 do n radi
{
      za i:=1 do 5 radi // učitamo odabir za svakog od pet igrača
            učitaj(odabir[i]);
      za i:=1 do 5 radi // inicijalizacija stanja svakog igrača
            igrac[i]:=0;
      za i:=1 do 5 radi
        za j:=1 do 5 radi
            igrac[i]:=igrac[i]+stanje[odabir[i],odabir[j]];
       za i:=1 do 5 radi
            ispiši(igrač[i]);
};
```

**Potrebno znanje:** naredba ponavljanja, niz, dvodimenzionalni niz

**Kategorija:** ad hoc

## 7.2. Zadatak: Bowling — Autor: Nikola Dmitrović

Bowling je jedan od onih zadataka koji imaju puno teksta, a jednostavno rješenje. Nakon trećeg čitanja, sve postaje jasno i samo je potrebno uvjete zadatka točno nakodirati. Posebno treba samo paziti da odgovarajuće bacanje kugle promatramo u odgovarajućem okviru.

Razlikujemo tri slučaja. Prvi slučaj nastaje kada je zbroj srušenih čunjeva u trenutnom i sljedećem bacanju manji od deset. Tada je ukupan rezultat u tom okviru jednak zbroju srušenih čunjeva u ta dva bacanja. Nakon ovoga prelazimo na sljedeći okvir i preskačemo dva navedena bacanja.

Drugi slučaj nastaje kada je broj srušenih čunjeva u trenutnom bacanju jednak deset. Tada je ukupan rezultat u tom okviru jednak zbroju srušenih čunjeva u tom i naredna dva bacanja. Nakon ovoga prelazimo na sljedeći okvir i preskačemo samo to navedeno bacanje.

Treći slučaj nastaje kada je zbroj srušenih čunjeva u trenutnom i sljedećem bacanju jednak deset. Tada je ukupan rezultat u tom okviru jednak zbroju srušenih čunjeva u ta dva bacanja i prvom sljedećem. Nakon ovoga prelazimo na sljedeći okvir i preskačemo dva navedena bacanja.

Pseudokod rješenja:

```
 //u niz „bacanje“ spremimo stanje za svako od n bacanja
 //u niz „okvir“ spremamo stanje za svaki od 10 odigranih okvira
 koje:=1; // trenutno bacanje
za i:=1 do 10 radi // odredimo stanje za svaki okvir
{
        ako je bacanje[koje]+bacanje[koje+1]<10 tada
        {
           okvir[i]:=bacanje[koje]+bacanje[koje+1];
           koje:=koje+2; // preskoči dva bacanja
        }
        inače
        ako je bacanje[koje]=10 tada
        {
           okvir[i]:=10+bacanje[koje+1]+bacanje[koje+2];  // strike
           koje:=koje+1; //preskoči jedno bacanje
        }
        inače
        ako je bacanje[koje]+bacanje[koje+1]=10 tada
        {
           okvir[i]:=10+bacanje[koje+2];  // spare
           koje:=koje+2; // preskoči dva bacanja
        }
};
za i:=1 do 10 radi
    ispiši(okvir[i]);
```

**Potrebno znanje:** naredba odlučivanja, naredba ponavljanja, niz

**Kategorija:** ad hoc, simulacija

## 7.3. Zadatak: Izraz Autor: Antun Razum

Kako bi olakšali rješavanje zadatka niz znakova iz ulaza učitat ćemo u dva odvojena niza od kojih će jedan sadržavati brojeve, a drugi operatore koji se nalaze u ulaznom nizu znakova. To ćemo učiniti tako što ćemo naizmjence učitavati jedan broj i jedan znak sve dok možemo učitavati tj. dok učitani znak ne bude jednak znaku novog reda što označava kraj našeg ulaznog niza znakova.

Sada ćemo za svaki red u kojem se nalaze operatori izvršavati operatore koji se nalaze u tom redu kako bi poštivali zadani poredak prioriteta operatora. Operatore ćemo izvršavati na način da ćemo s lijeva na desno proći po nizu u kojem se nalaze operatori i ako pronađemo operator koji se nalazi u trenutnom retku operatora izvršit ćemo ga tako da ćemo uzeti dva operanda tj. broja iz niza brojeva od kojih se prvi nalazi na istoj lokaciji na kojoj se nalazi trenutni operator koji izvršavamo, a drugi se nalazi na lokaciji za jedno mjesto desno od nje. Rezultat ćemo opet spremiti u niz s brojevima na lokaciju na kojoj se nalazi operator koji izvršavamo te ćemo nakon toga dio niza brojeva koji se nalazi desno od drugog operanda pomaknuti jedno mjesto ulijevo kako bi izbrisali desni operand iz njega. Isto ćemo učiniti i s nizom s operatorima – dio niza koji se nalazi desno od trenutnog operatora pomaknut ćemo također jedno mjesto ulijevo kako bi izbrisali trenutni operator iz niza nakon što smo ga izvršili. Sada smo dobili nizove koji predstavljaju novi izraz nakon izvršavanja tog operatora. Nakon što dođemo do kraja niza s operatorima, prelazimo na idući redak s operatorima i ponavljamo postupak dok ih ne prođemo sve. Na posljetku, krajnja vrijednost izraza nalazi se na početku niza brojeva.

**Potrebno znanje:** naredba odlučivanja, naredba ponavljanja, nizovi, stringovi

**Kategorija:** ad hoc

## 8.1. Zadatak: Spust Autor: Nikola Dmitrović

Slično kao kod Slaloma, do rješenja ovog problema doći ćemo simulacijom stvarne životne situacije. Koristeći trenutnu poziciju svakog skijaša nakon njegove vožnje odredit ćemo završnu poziciju za svakog od njih. Detaljnije, trenutnu poziciju svakog skijaša usporedimo s pozicijom svih onih koji su startali prije njega. Ako je pozicija skijaša s manjim startnih brojem veća ili jednaka od pozicije promatranog skijaša, tada je povećamo za jedan, tj. ažuriramo trenutni ukupni poredak skijaša.

Na kraju, prema poziciji koju je zauzimao na kraju utrke lako ćemo odrediti statni broj našeg Ivice. Isto tako, poredak prva tri skijaša biti će lako dohvatljiv.

Pseudokod rješenja:

```
učitaj(N); //broj skijaša u utrci
za i:=1 do N radi
      učitaj(skijas[i]); // u niz „skijas“ upišemo trenutne pozicije
za i:=1 do N radi // za svakog skijaša
      za j:=1 do i-1 radi // pogledamo one koji su startali prije njega
            ako je skijas[i]<=skijas[j] tada //ako je bolje plasiran
                  skijas[j]:=skijas[j]+1; // pomaknimo j-tog skijaša u
                                          // poretku za jedno mjesto prema gore
učitaj(koje); // koji je mjesto na kraju zauzimao naš Ivica
za i:=1 do N radi
      ako je skijas[i]=koje tada // pronađi onog koji zauzima to mjesto
            ispiši(i);
za i:=1 do N radi
{
      ako je skijas[i]=1 tada
            prvi:=i; // prvi u poretku
      ako je skijas[i]=2 tada
            drugi:=i; // drugi u poretku
      ako je skijas[i]=3 tada
            treci:=i; // treći u poretku
};
ispiši(prvi,' ', drugi,' ',treci);
```

**Potrebno znanje:** naredba odlučivanja, naredba ponavljanja, niz

**Kategorija:** ad hoc, simulacija

## 8.2. Zadatak: Zagrade Autor: Matija Milišić[3]

U ovom zadatku treba za svaki upitnik odrediti koju **otvorenu zagradu zatvara**. Promatrajmo učitani niz s lijeva na desno. Neka se prva zatvorena zagrada (ili upitnik) nalazi na mjestu *i*. Primijetimo da ta zagrada zatvara zadnju otvorenu zagradu koja se nalazi na mjestu *i-1*. U suprotnom, niz ne bi bio pravilan. Kada nađemo takav par zagrada, možemo ga izbaciti iz niza i ponoviti postupak. Niz s izbačenim parom zagrada i dalje će biti pravilan te neće doći do promjene rješenja. Postupak ponavljamo sve dok ne odredimo svaki upitnik.

Budući da ovakvo rješenje nije najkraće i najbrže, možemo ga implementirati na nešto drugačiji način. Sve upitnike odredit ćemo u jednom prolasku kroz niz s lijeva na desno. Za to ćemo trebati dodatni niz u kojem ćemo držati otvorene zagrade. Zadnji znak u dodatnom nizu predstavljat će zadnju otvorenu zagradu (najdesniju) koja još nije dobila svoj par, tj. koja još nije zatvorena. Kada naiđemo na otvorenu zagradu, nju moramo ubaciti na kraj dodatnog niza. Ta zadnja ubačena zagrada sigurno će se zatvoriti prije svih zagrada koje se u dodatnom nizu nalaze ispred nje. Dakle, kada naiđemo na zatvorenu zagradu (ili upitnik), ta zagrada odnosit će se na zadnju zagradu iz dodatnog niza. Nakon što zagrade sparimo, zatvorenu zagradu moramo izbaciti iz dodatnog niza. Ovakva implementacija je brža i lakša.

**Potrebno znanje:** naredba odlučivanja, naredba ponavljanja, stringovi, nizovi

**Kategorija:** ad hoc

## 8.3. Zadatak: Igra Autor: Matija Milišić

Za početak promotrimo zadatak uz uvjet K = 1. Kako postoji samo jedan način kretanja, do rješenja dolazimo simulacijom. Jedino što trebamo pamtiti je na kojoj se poziciji figurica nalazi i koliko smo koraka napravili. Simulacija završava kada figurica dođe na polje sa znakom 'X' ili izađe izvan tablice. To napravimo za svaku od P početnih pozicija. Takav pristup nosi 30% bodova.

Za potpuno rješenje ovoga zadatka potrebno je poznavati bfs (breadth-first search) algoritam. Budući da sada vrijedi K > 1, figurica nakon određenog broja koraka može završiti na više različitih pozicija. Dodatno, figurica se može na istoj poziciji pojaviti više puta i to nakon različitog broja poteza.

Ideja je bfs algoritma na svaku poziciju zapisati minimalan broj koraka u kojem figurica može doći do nje. Zbog toga na početnu poziciju možemo zapisati broj nula. Nakon toga se proširimo na sljedećih K pozicija i na njih zapišemo broj jedan. Zatim se iz svake od tih pozicija proširimo na sljedećih K pozicija. Ako smo na nekoj od tih pozicija već bili, tu poziciju ne diramo. Na sve ostale pozicije upišemo broj dva. Nakon toga se ponovno proširimo iz pozicija s dva koraka udaljenosti. To ponavljamo sve dok ne dođemo na polje sa znakom 'X' ili izađemo izvan tablice.

Sada se javlja problem kako brzo znati iz kojih se pozicija moramo proširiti. To možemo riješiti na više načina, npr. korištenjem strukture red (queue). Trenutna pozicija iz koje se širimo nalazit će se uvijek na početku reda, a pozicije na koje se proširujemo ubacivat ćemo naredbom push na kraj reda. Kada završimo sa širenjem iz neke pozicije, tu poziciju izbacujemo iz reda naredbom pop. Prilikom širenja

[3] 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)

moramo paziti da ne ubacimo poziciju na kojoj smo već bili. Jedan od načina na koji to možemo riješiti je da na početku na sva polja postavimo zastavicu nisam bio, a kada se proširimo na neko polje na njemu promijenimo zastavicu u bio.

Promotrimo vremensku složenost ovog rješenja. U bfs algoritmu u najgorem slučaju možemo proći kroz sva polja tablice. Isto tako, iz svakog polja u koje dođemo moramo se proširiti na sljedećih K polja. To nas dovodi do vremenske složenosti bfs algoritma O(N*M*K). Budući da isti algoritam moramo pokrenuti iz svake od P početnih pozicija ukupna vremenska složenost ovakvog rješenja iznosi O(N*M*K*P). To je dovoljno brzo za 70% test primjera, u kojima vrijedi N, M, P ≤ 100.

Za dobivanje svih bodova potrebno je primijetiti da algoritam ne trebamo pokrenuti iz svake od P početnih pozicija. Možemo ga pokrenuti samo jednom, ali na način da promijenimo početne uvjete. To napravimo tako da se krenemo širiti iz polja sa znakom 'X' i nekih polja izvan tablice. Algoritam prekidamo kada obiđemo sve pozicije iz tablice. Nakon toga je lako naći rješenje za svaku od P početnih pozicija, piše upravo na toj poziciji. Potrebno je još odrediti iz kojih se polja izvan tablice krećemo širiti. Dovoljno je napraviti devet puta veću matricu, originalnu matricu smjestiti u sredinu, a ostatak popuniti sa znakom 'X' te potom napraviti bfs iz svih polja s tim znakom. Vremenska složenost ovakvog rješenja je jedan bfs, odnosno O(N*M*K) što je dovoljno brzo za ograničenja dana u zadatku.

**Potrebno znanje:** bfs algoritam

**Kategorija:** ad hoc