Infokup 2012

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

Opis algoritama (Basic/Pascal/C/C++)

Sponzori

Medijski pokrovitelji

Agencija za odgoj i obrazovanje
Education and Teacher Training Agency

MINISTARSTVO ZNANOSTI, OBRAZOVANJA I ŠPORTA REPUBLIKE HRVATSKE

udruga mladih programera dump

Zadatak: Snijeg

Motivacijski zadatak za početnike koji se sastoji od dva dijela. Opis rješenja je vrlo jednostavan.

```
učitaj(granica);
učitaj(V1);
učitaj(V2);
učitaj(V3);
ispiši(V1-V2+V3);
ako je V1-V2+V3>granica tada
      ispiši('NE')
inače
      ispiši('DA');
```

Potrebno znanje: naredba učitavanja, naredba ispisa, osnovni oblik naredbe odlučivanja.

Kategorija: ad hoc

Zadatak: BMI

Prilikom rješavanja ovog zadatka treba paziti na pravilno osmišljavanje višestruke naredbe odlučivanja, na pravilno postavljanje granica pojedinih segmenata te točno prepisivanje zadane vrijednosti koja se spominje u zadatku.

```
učitaj(T); učitaj(V); učitaj(spol);

BMI:=(T/sqr(V))*10000;
ispiši(BMI); // dozvoljeno je 0.1 odstupanje od službene vrijednosti

ako je spol='M' tada
   ako je BMI<20.7 tada
      ispiši('BMI prenizak')
   inače
       ako je BMI<26.5 tada
          ispiši('BMI idealan')
       inače
           ako je BMI<27.9 tada
              ispiši('BMI malo iznad normale')
           inače
               ako je BMI<31.2 tada
                  ispiši('BMI visok')
               inače
                   ako je BMI<45.4 tada
                      ispiši('BMI previsok')
                   inače
                       ispiši('BMI izrazito visok')
inače
   ako je BMI<19.1 tada
      ispiši('BMI prenizak')
   inače
       ako je BMI<25.9 tada
```

```
        ispiši('BMI idealan')
    inače
        ako je BMI<27.4 tada
            ispiši('BMI malo iznad normale')
        inače
            ako je BMI<32.3 tada
                ispiši('BMI visok')
            inače
                ako je BMI<44.8 tada
                    ispiši('BMI previsok')
                inače
                    ispiši('BMI izrazito visok');
```

Potrebno znanje: naredba učitavanja i ispisivanja, realni tip podataka, višestruka naredba odlučivanja

Kategorija: ad hoc

Zadatak: Password

Password je jedan od onih zadataka u kome je odmah jasno što se traži te samo rješenje na papiru bez kodiranja je vrlo jednostavno. Prilikom kodiranja pojavljuje se par stvari na koje treba paziti. Kako zadatak ne traži da se šifra i kreira, tada je dovoljno pratiti samo zadnje dvije znamenke u šifri i čekati da one budu jednake početnim zadanim vrijednostima. Pri tome je potrebno posebno pamtiti te prve dvije znamenke te paziti da točno pridružimo vrijednost zadnjoj znamenki i da točno promijenimo vrijednosti zadnjim dvijema znamenkama.

```
učitaj(prva);
učitaj(druga);

predzadnja:=prva;
zadnja:=druga;
broj_zn_u_sifri:=2;

ponavljaj
      ako je predzadnja+zadnja<=9 tada
         tmp:=predzadnja+zadnja
      inače
          tmp:=(predzadnja+zadnja) mod 10;

      predzadnja:=zadnja;
      zadnja:=tmp;

      broj_zn_u_sifri:= broj_zn_u_sifri+1;

sve dok ne bude (predzadnja=prva) i (zadnja=druga);

ispiši(broj_zn_u_sifri);
```

Potrebno znanje: naredba ponavljanja

Kategorija: ad hoc

Zadatak: Picard

Zadanu tablicu sa slovima možemo promatrati kao dvodimenzionalni niz znakova. Za svaki redak te strukture trebamo provjeriti može li se u nju upisati zadana riječ. Kako bi to otkrili, simuliramo upisivanje te riječi u redak. Riječ počinjemo upisivati prvo od prve pozicije u retku, pa od druge i tako sve do broj_stupaca - duljina(rijeci)+1 pozicije. Slovo iz riječi možemo upisati u polje retka ako je to polje prazno (<>'*') ili na tom polju već piše to slovo (ovu situaciju moramo prepoznati i pamtiti koliko se takvih podudaranja dogodilo pri simulaciji upisa riječi u redak od te pozicije te istovremeno pratiti poziciju na kojoj se dogodio maksimalan broj podudaranja).

Ako tijekom simulacije upisa riječi po retcima ne pronađemo traženu poziciju, tj. otkrijemo da nije moguće upisati riječ u tablicu, tada možemo ponoviti postupak simulacije i brojiti polja u koja je već unaprijed upisano slovo koje je različito od onog kojeg želimo upisati. Međutim, taj postupak je moguće ukomponirati u algoritam za rješavanje prvog dijela zadatka.

```
učitaj (n); učitaj (m);
učitaj (T); // T je tablica znakova, upis ovisi o jeziku
učitaj (rijec);

pozicija_redak:=0; pozicija_stupac:=0;

max:=-1; min:=duljina(rijec);
za i:=1 do n radi
    za j:=1 do m-duljina(rijec)+1 radi
    {
         nasao:=1; koliko:=0; brisi:=0;
         za k:=0 do duljina(rijec)-1 do
         begin
              ako je (T[i,j+k]<>'*') i (T[i,j+k]<>rijec[k+1]) tada
              {
                 nasao:=0;
                 brisi:=brisi+1;
              };

              ako je T[i,j+k]=rijec[k+1] tada koliko:=koliko+1;
         };

         ako je (nasao=1) i (koliko>max) tada
         {
              max:=koliko;
              pozicija_redak:=i;
              pozicija_stupac:=j;
         };

         ako je brisi<min tada min:=brisi;
```

```
    };

ako je (pozicija_redak<>0) i (pozicija_stupac<>0) tada
   ispiši('DA ', pozicija_redak,' ',pozicija_stupac)
inače
    ispiši('NE ', min);
```

Potrebno znanje: dvodimenzionalno polje znakova(ili polje stringova), naredba ponavljanja, naredba odlučivanja, algoritam traženja minimuma i maksimuma

Kategorija: ad hoc, simulacija

Zadatak: Mark

Jedina zamka koju krije ovaj zadatak je trenutak kada je na semaforu zeleno svjetlo, a vrijeme do promjene boje na semaforu je strogo manje od vremena potrebnog da se prijeđe ulica. U tom slučaju Mark ne može odmah krenuti prijelaziti ulicu već mora čekati da se sljedeći put upali zeleno svjetlo.

```
učitaj(boja);
učitaj (X);
učitaj (Y);

presao:=Y;
ako je boja='C' tada
   presao:=presao+X
inače
    ako je X<Y tada
       presao:=presao+X+30;

ispiši(presao);
```

Potrebno znanje: naredba odlučivanja

Kategorija: ad hoc

Zadatak: Mancala

Mancala je jedan od onih zadataka u kojima je nužno precizno nakodirati uvjete koji su zadani. Ipak, u ovakvim zadacima ako točno nakodirate samo neke uvjete možete dobiti određeni broj bodova. Na samom natjecanju uvijek treba težiti općem rješenju koje će pokriti sve uvjete. Ako to nije moguće, uvijek treba pokušati točno nakodirati što je više moguće postavljenih uvjeta.

U Mancali se trebalo paziti na nekoliko stvari:

1. kako osmisliti strukturu koja će omogućiti najbolju simulaciju igre. U ovom slučaju je to niz;
2. kako osmisliti način simulacije prebacivanja kamenčića u druga udubljenja;
3. treba paziti je li odabrano udubljenje s prave strane i je li neprazno;
4. treba osmisliti kako omogućiti kretanje u krug po igraćoj ploči;
5. kako pratiti tko je na potezu i promjenu poteza;

6. kako provjeriti je li posljednji kamenčić došao u do tada neprazno polje s igračeve strane i kako u svoju banku preseliti sve kamenčiće iz tog i nasuprotnog udubljenja.

Rješenje se može opisati na sljedeći način:

```
inicijaliziraj(udubljenje); // udubljenje je niz od 14 komponenti

učitaj(n); potez:=1; // 1:igrač B na potezu; 0:igrač A na potezu
za i:=1 do n radi
{
     učitaj(x); // pozicija udubljenja
     x:=x+1*potez; //povećaj za jedan ako je igrač B na potezu (zbog polja 7)

     // simulacija igre
     ako je (x>=1+7*potez) i (x<=6+7*potez) i (udubljenje[x]<>0) tada
     {
          kamencici:=udubljenje[x]; // uzmi kamenčiće u ruku
          udubljenje[x]:=0;         // udubljenje x je sada prazno

          dok je kamencici>0 radi  // dok ima kamenčića u ruci
          {
               povećaj(x,1); x:=x mod 14; ako je x=0 tada x:=14;
               ako je x<>14-7*potez tada
               {
                    povećaj(udubljenje[x],1);
                    smanji(kamencici,1);
               };
          };
        // provjera slučaja broj 6.
          ako je (x>=1+7*potez) i (x<=6+7*potez) i (udubljenje[x]=1) i
(udubljenje[abs(x-14)]<>0) tada
          {
               povećaj(udubljenje[7+7*potez],udubljenje[abs(x-14)]+1);
               udubljenje[x]:=0;
               udubljenje[abs(x-14)]:=0;
          };
// provjera i promjena poteza
          ako je (x<>7+7*potez) tada potez:=(potez+1) mod 2;
     };
};

ispiši(udubljenje);
```

Potrebno znanje: nizovi, naredbe ponavljanja, dobra ideja

Kategorija: simulacija

Zadatak: Avioni

Ovaj zadatak pripada u kategoriju „simulacija“ – zadana su pravila po kojima se neki sustav ponaša, a mi trebamo napisati program u kojem ćemo pažljivo slijediti ta pravila iz koraka u korak. Na primjer, možemo ovako pristupiti rješavanju zadatka:

for t = 1, 2, 3, ...

- Obradi sve događaje koji se događaju u trenutku t: provjeri koji su sve avioni kandidati za polijetanje u trenutku t, među njima odredi one koji smiju poletjeti (linija im se ne siječe s nekim tko je već u zraku), provjeri treba li neki avion sletjeti u trenutku t, itd.

Za svaki avion trebamo pratiti njegovu trenutnu lokaciju: je li došao predviđeni trenutak polijetanja, je li avion kandidat za polijetanje, je li upravo u zraku ili je već sletio.

Zadatak je moguće riješiti i na drugačije načine, pa ćemo sada opisati i jedno takvo rješenje.

Neka je A polje u kojemu čuvamo informacije o avionima: A[i].start i A[i].cilj predstavljaju početni i završni grad, A[i].polazak je predviđeni trenutak polaska aviona, a A[i].trajanje trajanje leta.

Za svaki avion, čuvat ćemo još jednu informaciju: stigao[i]. Ova varijabla je jednaka nuli ako i-ti avion još nije poletio, a u protivnom (dakle, ako avion leti ili je već sletio) je jednaka trenutku u kojem će i-ti avion sletjeti u grad na sjevernoj obali.

Lako je vidjeti da sljedeći algoritam rješava naš problem:

Broj stiglih aviona = 0;
Sve dok je broj stiglih aviona < L radi

- Pronađi avion (označimo ga s X) za kojeg vrijedi:
  1. stigao[X]=0 (to jest, avion X još nije poletio);
  2. A[X].polazak je najmanji od svih za kojeg vrijedi 1;
  3. X je najzapadniji avion od svih za koje vrijedi 1 i 2;
- Označi da će avion X poletjeti: stavimo da je
  stigao[X] = A[X].polazak + A[X].trajanje;
- Povećaj broj stiglih aviona za 1.
- Svim avionima koji još nisu poletjeli (za njih je stigao[i]=0) i čija bi se linija sjekla s linijom aviona X, pomakni trenutak mogućeg polaska na trenutak kada avion X sleti: A[i].polazak = stigao[X].
  Za svaki avion, ispiši njegov trenutak slijetanja (to piše u stigao[i]).

Uočite da će zadnji korak u petlji, promjenom predviđenog trenutka polaska aviona koji bi se sjekli s avionom X, automatski onemogućiti polijetanje takvim avionima sve dok avion X ne sleti. Zbog toga u prvom koraku u petlji uopće ne treba provjeravati da li se linija aviona koji je idući na redu za polijetanje siječe s linijom nekog drugog aviona.

Potrebno znanje: nizovi, naredbe ponavljanja

Kategorija: simulacija