11. srednješolsko tekmovanje ACM v znanju računalništva

Šolsko tekmovanje

22. januarja 2016

NASVETI ZA TEKMOVALCE

Naloge na tem šolskem tekmovanju pokrivajo širok razpon težavnosti, tako da ni nič hudega, če ne znaš rešiti vseh.

Nekatere naloge so tipa **napiši program** (ali **napiši podprogram**), nekatere pa tipa **opiši postopek**. Pri slednjih ti ni treba pisati programa ali podprograma v kakšnem konkretnem programskem jeziku, ampak lahko postopek opišeš tudi kako drugače: z besedami (v naravnem jeziku), psevdokodo (glej spodaj), diagramom poteka itd. Glavno je, da je tvoj opis dovolj natančen, jasen in razumljiv, tako da je iz njega razvidno, da si dejansko našel in razumel pot do rešitve naloge.

Psevdokodi pravijo včasih tudi strukturirani naravni jezik. Postopek opišemo v naravnem jeziku, vendar opis strukturiramo na podoben način kot pri programskih jezikih, tako da se jasno vidi strukturo vejitev, zank in drugih programskih elementov.

Primer opisa postopka v psevdokodi: recimo, da imamo zaporedje besed in bi ga radi razbili na več vrstic tako, da ne bo nobena vrstica preširoka.

```
naj bo trenutna vrstica prazen niz;
pregleduj besede po vrsti od prve do zadnje:
    če bi trenutna vrstica z dodano trenutno besedo (in presledkom
    pred njo) postala predolga,
        izpiši trenutno vrstico in jo potem postavi na prazen niz;
    dodaj trenutno besedo na konec trenutne vrstice;
če trenutna vrstica ni prazen niz, jo izpiši;
```

(Opomba: samo zato, ker je tu primer psevdokode, to še ne pomeni, da moraš tudi ti pisati svoje odgovore v psevdokodi.)

Če pa v okviru neke rešitve pišeš izvorno kodo programa ali podprograma, obvezno poleg te izvorne kode v nekaj stavkih opiši, kako deluje (oz. naj bi delovala) tvoja rešitev in na kakšni ideji temelji.

Pri ocenjevanju so vse naloge vredne enako število točk. Svoje odgovore dobro utemelji. Prizadevaj si predvsem, da bi bile tvoje rešitve pravilne, ob tem pa je zaželeno, da so tudi čim bolj učinkovite (take dobijo več točk kot manj učinkovite). Za manjše sintaktične napake se načeloma ne odbije veliko točk. Priporočljivo in zaželeno je, da so tvoje rešitve napisane pregledno in čitljivo. Če je na listih, ki jih oddajaš, več različic rešitve za kakšno nalogo, jasno označi, katera je tista, ki naj jo ocenjevalci upoštevajo.

Če naloga zahteva branje ali obdelavo vhodnih podatkov, lahko tvoja rešitev (če v nalogi ni drugače napisano) predpostavi, da v vhodnih podatkih ni napak (torej da je njihova vsebina in oblika skladna s tem, kar piše v nalogi).

Nekatere naloge zahtevajo branje podatkov s standardnega vhoda in pisanje na standardni izhod. Za pomoč je tu nekaj primerov programov, ki delajo s standardnim vhodom in izhodom:

- Program, ki prebere s standardnega vhoda dve števili in izpiše na standardni izhod njuno vsoto:

```
program BranjeStevil;
var i, j: integer;
begin
  ReadLn(i, j);
  WriteLn(i, ' + ', j, ' = ', i + j);
end. {BranjeStevil}
```

```
#include <stdio.h>
int main() {
  int i, j; scanf("%d %d", &i, &j);
  printf("%d + %d = %d\n", i, j, i + j);
  return 0;
}
```

- Program, ki bere s standardnega vhoda po vrsticah, jih šteje in prepisuje na standardni izhod, na koncu pa izpiše še skupno dolžino:

```
program BranjeVrstic;
var s: string; i, d: integer;
begin
  i := 0; d := 0;
  while not Eof do begin
    ReadLn(s);
    i := i + 1; d := d + Length(s);
    WriteLn(i, '. vrstica: "', s, '"');
  end; {while}
  WriteLn(i, ' vrstic, ', d, ' znakov.');
end. {BranjeVrstic}
```

```
#include <stdio.h>
#include <string.h>
int main() {
  char s[201]; int i = 0, d = 0;
  while (gets(s)) {
    i++; d += strlen(s);
    printf("%d. vrstica: \"%s\"\n", i, s);
  }
  printf("%d vrstic, %d znakov.\n", i, d);
  return 0;
}
```

Opomba: C-jevska različica gornjega programa predpostavlja, da ni nobena vrstica vhodnega besedila daljša od dvesto znakov. Funkciji gets se je v praksi bolje izogibati, ker pri njej nimamo zaščite pred primeri, ko je vrstica daljša od naše tabele s. Namesto gets bi bilo bolje (in varneje) uporabiti fgets ali fscanf; vendar pa za rešitev naših tekmovalnih nalog zadošča tudi gets.

- Program, ki bere s standardnega vhoda po znakih, jih prepisuje na standardni izhod, na koncu pa izpiše še število prebranih znakov (ne vštevši znakov za konec vrstice):

```
program BranjeZnakov;
var i: integer; c: char;
begin
  i := 0;
  while not Eof do begin
    while not Eoln do
      begin Read(c); Write(c); i := i + 1 end;
    if not Eof then begin ReadLn; WriteLn end;
  end; {while}
  WriteLn('Skupaj ', i, ' znakov.');
end. {BranjeZnakov}
```

```
#include <stdio.h>

int main() {
  int i = 0, c;
  while ((c = getchar()) != EOF) {
    putchar(c); if (i != '\n') i++;
  }
  printf("Skupaj %d znakov.\n", i);
  return 0;
}
```

Še isti trije primeri v pythonu:

```
# Branje dveh števil in izpis vsote:
import sys

a, b = sys.stdin.readline().split()
a = int(a); b = int(b)
print "%d + %d = %d" % (a, b, a + b)
```

```
# Branje standardnega vhoda po vrsticah:
import sys

i = d = 0
for s in sys.stdin:
  s = s.rstrip('\n') # odrežemo znak za konec vrstice
  i += 1; d += len(s)
  print "%d. vrstica: \"%s\"" % (i, s)
print "%d vrstic, %d znakov." % (i, d)
```

```
# Branje standardnega vhoda znak po znak:
import sys

i = 0
while True:
  c = sys.stdin.read(1)
  if c == "": break # EOF
  sys.stdout.write(c)
  if c != '\n': i += 1
print "Skupaj %d znakov." % i
```

Še isti trije primeri v javi:

```
// Branje dveh števil in izpis vsote:
import java.io.*;
import java.util.Scanner;

public class Primer1
{
   public static void main(String[] args) throws IOException
   {
      Scanner fi = new Scanner(System.in);
      int i = fi.nextInt(); int j = fi.nextInt();
      System.out.println(i + " + " + j + " = " + (i + j));
   }
}
```

```
// Branje standardnega vhoda po vrsticah:
import java.io.*;

public class Primer2
{
   public static void main(String[] args) throws IOException
   {
      BufferedReader fi = new BufferedReader(new InputStreamReader(System.in));
      int i = 0, d = 0; String s;
      while ((s = fi.readLine()) != null) {
         i++; d += s.length();
         System.out.println(i + ". vrstica: \"" + s + "\""); }
      System.out.println(i + " vrstic, " + d + " znakov.");
   }
}
```

```
// Branje standardnega vhoda znak po znak:
import java.io.*;

public class Primer3
{
   public static void main(String[] args) throws IOException
   {
      InputStreamReader fi = new InputStreamReader(System.in);
      int i = 0, c;
      while ((c = fi.read()) >= 0) {
         System.out.print((char) c); if (c != '\n' && c != '\r') i++; }
      System.out.println("Skupaj " + i + " znakov.");
   }
}
```

11. srednješolsko tekmovanje ACM v znanju računalništva

Šolsko tekmovanje

22. januarja 2016

NALOGE ZA ŠOLSKO TEKMOVANJE

Svoje odgovore dobro utemelji. Če pišeš izvorno kodo programa ali podprograma, **OBVEZNO** tudi v nekaj stavkih z besedami opiši idejo, na kateri temelji tvoja rešitev. Če ni v nalogi drugače napisano, lahko tvoje rešitve predpostavljajo, da so vhodni podatki brez napak (da ustrezajo formatu in omejitvam, kot jih podaja naloga). Zaželeno je, da so tvoje rešitve, poleg tega, da so pravilne, tudi učinkovite (bolj učinkovite rešitve dobijo več točk). Nalog je pet in pri vsaki nalogi lahko dobiš od 0 do 20 točk.

Rešitve bodo objavljene na `http://rtk.ijs.si/`.

1. Nadležne besede

Neko besedilo hočemo natipkati s telefonsko tipkovnico, kjer se več črk tipka z isto tipko (črke *abc* so na tipki 2, *def* na tipki 3, *ghi* na tipki 4, *jkl* na tipki 5, *mno* na tipki 6, *pqrs* na tipki 7, *tuv* na tipki 8 in *wxyz* na tipki 9). Zato je nadležno, če se v besedi pojavita dve zaporedni črki, ki se tipkata z isto tipko. V nekaterih besedah se to zgodi celo po večkrat — na primer, v besedi *praprababica* kar šestkrat (*pr*, še enkrat *pr*, *ab*, *ba*, še enkrat *ab* in na koncu še *ca*).

Napiši program, ki prebere seznam besed in izpiše tisto, v kateri se največkrat zgodi, da se dve zaporedni črki tipkata z isto tipko. (Če je več takih besed, je vseeno, katero od njih izpišeš.) Vsaka beseda je v svoji vrstici in je dolga največ 100 znakov; vsi znaki so male črke angleške abecede. Tvoj program lahko bere s standardnega vhoda ali pa iz datoteke `besede.txt` (kar ti je lažje).

2. Prepisovanje

Na šoli za moderno umetnost je izvirnost najbolj cenjena vrlina. Učenci redno pišejo teste iz izvirnosti, ki potekajo tako, da n učencev posedejo v vrsto drug zraven drugega, jih oštevilčijo od 1 do n in jim naročijo, naj na list napišejo število. Toda jojmene, učitelj je opazil, da učenci prepisujejo, saj njihova števila niso izvirna, ampak so si števila učencev, ki sedijo skupaj, pogosto zelo podobna.

Učenec lahko prepisuje le od sošolca, ki je neposredno levo ali desno od njega (če ga ima). Učitelj ima določeno toleranco izvirnosti t: če se števili dveh sosednjih učencev razlikujeta za t ali manj, sta premalo izvirni in se za oba tadva učenca sumi, da sta prepisovala.

Napiši program, ki prebere n, t in seznam števil, ki so jih na testu napisali učenci, in izpiše število učencev, ki so osumljeni prepisovanja. Vhodne podatke lahko bereš s standardnega vhoda ali pa iz datoteke `prepisovanje.txt` (kar ti je lažje). V prvi vrstici vhoda sta število učencev n in učiteljeva toleranca t, ločeni s presledkom (veljalo bo $n \leq 1\,000\,000$ in $0 < t \leq 1\,000\,000\,000$). V drugi vrstici je n celih števil, ločenih s presledki; vsako od teh števil je med $-1\,000\,000\,000$ in $+1\,000\,000\,000$).

Primer vhoda:

```
7 2
1 4 2 6 -3 -5 -4
```

Pripadajoči izhod:

```
5
```

Komentar: tu imamo $n = 7$ učencev in toleranco $t = 2$. Števili 1 in 4 sta za 3 narazen, tako da tu ni suma prepisovanja. Števili 4 in 2 sta za 2 narazen, tako da sta drugi in tretji učenec osumljena. Števila 2 in 6 ter 6 in -3 so dovolj narazen, ni suma. Števila -3 in -5 ter -5 in -4 so preveč skupaj, tako da so peti, šesti in sedmi učenec osumljeni. Tako je vsega skupaj osumljenih prepisovanja pet učencev.

3. Riziko

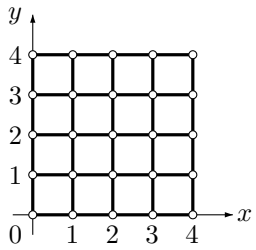
Dva igralca se igrata naslednjo igro. Najprej prvi igralec vrže tri kocke in jih uredi padajoče po številu pik; nato enako naredi še drugi igralec, le da ima ta samo dve kocki namesto treh. Nato primerjata prvo kocko prvega igralca in prvo kocko drugega igralca; tisti igralec, čigar kocka ima več pik, dobi dve točki (če imata oba enako število pik, dobi vsak po eno točko). Nato na enak način primerjata še drugo kocko prvega igralca in drugo kocko drugega igralca; spet dobi dve točki tisti, čigar kocka ima več pik (če pa imata oba enako število pik, dobi vsak po eno točko).

Napiši program, ki prebere pet celih števil od 1 od 6 — najprej tri kocke prvega igralca, nato dve kocki drugega igralca, vendar ne ene ne druge še niso nujno urejene padajoče — in izpiše, koliko točk dobi prvi in koliko drugi igralec.

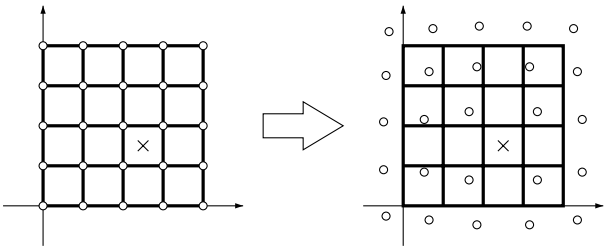
Primer: recimo, da prvi igralec vrže 1, 6, 2, drugi pa 4, 5. Po urejanju padajoče ima prvi igralec 6, 2, 1, drugi pa 5, 4. Najprej torej primerjata 6 in 5, pri čemer dobi dve točki prvi igralec; nato pa primerjata še 2 in 4, pri čemer dobi dve točki drugi igralec. Izid te igre je torej ta, da je vsak igralec dobil po dve točki.

4. Eksplozija

V koordinatni ravnini imamo kvadrat velikosti 4×4; njegov spodnji levi kot je v koordinatnem izhodišču. Ta kvadrat lahko v mislih razdelimo na 16 enotskih kvadratov velikosti 1×1. V vsako točko, ki je oglišče kakšnega od teh enotskih kvadratov, postavimo nek točkast predmet (teh predmetov je torej skupaj 25), kot kaže naslednja slika:



V središču enega od 4×4 enotskih kvadratov pride do eksplozije. Ta povzroči, da se vsak od naših 25 točkastih predmetov premakne za $1/2$ enote (z drugimi besedami, za polovico dolžine stranice enotskega kvadrata) stran od središča eksplozije (premakne se torej po poltraku, ki povezuje središče eksplozije s prvotnim položajem tistega predmeta). Primer kaže naslednja slika; križec $\times$ označuje središče eksplozije:



Opiši postopek, ki kot vhodne podatke dobi koordinate vseh 25 točkastih predmetov po eksploziji (vendar ne v kakšnem posebnem vrstnem redu — točke so lahko poljubno premešane) in izračuna, v središču katerega kvadrata je prišlo do eksplozije. **Dobro utemelji**, zakaj je tvoja rešitev pravilna.

5. Barvanje plošče

Imamo okroglo ploščo, razdeljeno na n enako širokih izsekov. Izseki so oštevilčeni v smeri urinega kazalca od 0 do $n-1$. Na začetku so vsi izseki bele barve, mi pa bi radi nekatere pobarvali črno; dana je tabela pobarvaj, ki nam pove, katere izseke hočemo pobarvati črno (vrednost pobarvaj[k] je **true**, če je treba izsek k pobarvati črno, sicer pa je **false**). Plošča je vpeta v napravo, ki zna ploščo vrteti in ima na eni strani barvno glavo, s katero lahko pobarva po en izsek naenkrat. Plošča podpira tri ukaze:

- Levo: če je bil prej pod barvno glavo izsek številka k, je po tem premiku pod glavo izsek številka $k+1$ (razen če je bil $k = n-1$, takrat pa je po premiku pod glavo izsek številka 0);
- Desno: če je bil prej pod barvno glavo izsek številka k, je po tem premiku pod glavo izsek številka $k-1$ (razen če je bil $k = 0$, takrat pa je po premiku pod glavo izsek številka $n-1$);
- Pobarvaj: pobarva s črno barvo izsek, ki je trenutno pod barvno glavo. Nič ni narobe, če isti izsek pobarvaš večkrat, vendar od tega tudi ni nobene koristi. Izseka, ki je bil nekoč že pobarvan črno, kasneje ne moremo pobarvati nazaj na belo.

Ploščo bi radi pobarvali v skladu s zahtevami iz tabele pobarvaj in pri tem izvedli čim manj ukazov. Znan je tudi začetni položaj plošče (torej številka izseka, ki je na začetku pod barvno glavo — recimo ji z). Vseeno nam je, kako bo plošča zasukana na koncu našega postopka. **Opiši postopek** ali pa napiši program ali podprogram (kar ti je lažje), ki iz teh podatkov izračuna najmanjše število ukazov, ki jih potrebujemo.

11. srednješolsko tekmovanje ACM v znanju računalništva

Šolsko tekmovanje

22. januarja 2016

REŠITVE NALOG ŠOLSKEGA TEKMOVANJA

1. Nadležne besede

Naša rešitev v zanki bere besede eno po eno, pri vsaki prebrani besedi pa gre v še eni gnezdeni zanki po znakih te besede in šteje, kolikokrat se zgodi, da se dve zaporedni črki tipkata z isto tipko (spremenljivka stNadleznih). Za ugotavljanje, s katero tipko se tipka posamezna črka, si pomagamo s tabelo tipke, v kateri kot indekse uporabimo kar položaje črk v angleški abecedi (tipke[0] se nanaša na črko a, tipke[1] na črko b in tako naprej). Najbolj nadležno besedo doslej si zapomnimo v spremenljivki najBeseda, število nadležnih parov črk v njej pa v maxNadleznih. Ko za trenutno besedo ugotovimo, koliko nadležnih parov vsebuje, moramo preveriti, če je to več kot pri najbolj nadležni besedi doslej, in če je, si trenutno besedo zapomnimo (v najBeseda in maxNadleznih).

```
#include <stdio.h>
#include <string.h>

const char *tipke = "22233344455566677778889999";

int main()
{
  char beseda[101], najBeseda[101];
  int i, stNadleznih, maxNadleznih = 0;
  najBeseda[0] = 0;
  while (gets(beseda))
  {
    for (i = 0, stNadleznih = 0; beseda[i]; i++)
      if (i > 0 && tipke[beseda[i - 1] - 'a'] == tipke[beseda[i] - 'a'])
        stNadleznih++;
    if (stNadleznih > maxNadleznih)
      maxNadleznih = stNadleznih, strcpy(najBeseda, beseda);
  }
  printf("%s\n", najBeseda); return 0;
}
```

2. Prepisovanje

Števila, ki so jih napisali naši učenci, berimo po vrsti v zanki; pri tem si v spremenljivki prejsnji zapomnimo število prejšnjega učenca, da ga bomo lahko primerjali s številom trenutnega učenca in tako videli, če ju moramo osumiti prepisovanja. Pri tem pazimo na to, da ko smo pri prvem učencu (i == 0 v spodnjem programu), prejšnjega učenca še ni.

Če vidimo, da sta trenutni in prejšnji učenec oddala preveč podobni števili, ju osumimo prepisovanja; to pomeni, da moramo ustrezno povečati števec osumljenih (spremenljivka stOsumljenih). Povečati ga moramo vsaj za 1, da zajamemo dejstvo, da je trenutni učenec zdaj osumljen (doslej pa ni bil); preveriti pa moramo še, ali je bil prejšnji učenec osumljen že prej (npr. ker je bilo njegovo število preblizu števila predprejšnjega učenca) ali ne. Če prej še ni bil osumljen, moramo števec osumljenih zdaj povečati tudi zaradi njega, drugače pa ne (da ne bi istega učenca šteli dvojno). Torej potrebujemo podatek o tem, ali je bil prejšnji učenec že osumljen ali ne; v ta namen ima spodnji program spremenljivko prejOsumljen.

Ko končamo z obdelavo trenutnega učenca, si podatke o njem zapomnimo v spremenljivkah prejsnji in prejOsumljen, da bomo pripravljeni na obdelavo naslednjega učenca (tedaj bo učenec, ki je zdaj še trenutni, postal prejšnji).

Na koncu imamo v spremenljivki stOsumljenih želeni rezultat, torej skupno število osumljenih učencev, in ga moramo le še izpisati.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

int main()
{
  int n, t, stOsumljenih = 0, i, prejsnji, trenutni;
  bool prejOsumljen;
  scanf("%d %d", &n, &t);
  /* Pojdimo v zanki po vseh učencih. */
  for (i = 0; i < n; i++)
  {
    scanf("%d", &trenutni);
    /* Ali se številka trenutnega in prejšnjega učenca premalo razlikujeta? */
    if (i > 0 && abs(prejsnji - trenutni) <= t)
    {
      /* Trenutni učenec je zdaj osumljen. */
      stOsumljenih++;
      /* Če prejšnji še ni bil osumljen, je zdaj osumljen tudi on
         in moramo števec osumljenih povečati tudi zaradi njega. */
      if (! prejOsumljen) stOsumljenih++;
      /* Ko bo trenutni učenec v naslednji iteraciji postal prejšnji,
         si zapomnimo, da je bil osumljen. */
      prejOsumljen = true;
    }
    /* Sicer pa trenutni učenec zaenkrat ni osumljen;
       zapomnimo si to za naslednjo iteracijo. */
    else prejOsumljen = false;
    /* Zapomnimo si številko trenutnega učenca za v naslednjo iteracijo. */
    prejsnji = trenutni;
  }
  /* Izpišimo rezultat. */
  printf("%d\n", stOsumljenih);
  return 0;
}
```

3. Riziko

Podatke o metih posameznih kock lahko shranimo v dve tabeli — tabela a (s tremi elementi) za prvega igralca in tabela b (z dvema elementoma) za drugega igralca. Zdaj moramo obe tabeli urediti padajoče; pri tem bi načeloma lahko uporabili katerega koli od številnih znanih postopkov urejanja (ali pa kakšno funkcijo za urejanje iz standardne knjižnice našega programskega jezika — v C-ju je to na primer qsort); ker pa sta obe tabeli tako zelo majhni in je njuna velikost znana vnaprej, ju lahko uredimo tudi s pomočjo čisto preprostega razmisleka. Pri tabeli b, ki ima le dva elementa, moramo le preveriti, če je prvi element manjši od drugega, in če je, ju zamenjamo.

Pri tabeli a s tremi elementi je stvar podobna, le malo bolj zapletena: če je $a[0] < a[1]$, ju zamenjamo; po tem vemo, da je manjši od prvih dveh elementov zdaj v $a[1]$; zdaj lahko primerjamo $a[1]$ z $a[2]$ in vemo, da bo manjši od teh dveh tudi najmanjši element cele tabele, ta pa mora biti na koncu tabele. Če se torej izkaže, da je $a[1] < a[2]$, ju moramo zamenjati, sicer pa lahko $a[2]$ ostane tam, kjer je bil. Zdaj vemo, da je v $a[2]$ najmanjši element tabele, v $a[0]$ in $a[1]$ pa sta ostala dva elementa, ki pa še nista nujno v pravem vrstnem redu, zato ju moramo še enkrat primerjati in po potrebi zamenjati med sabo.

Ko sta tako obe tabeli urejeni, moramo le še primerjati istoležne elemente — $a[0]$ z $b[0]$ ter $a[1]$ z $b[1]$ — in ustrezno dodeliti točke obema igralcema. Spodnji program šteje točke prvega igralca v spremenljivki ta, točke drugega pa v tb. Na koncu moramo le še izpisati vrednosti obeh spremenljivk.

```
#include <stdio.h>
```

```
int main()
{
  int a[3], b[2], t, ta, tb, i;
  scanf("%d %d %d %d %d", &a[0], &a[1], &a[2], &b[0], &b[1]);
  /* Uredimo kocke prvega igralca padajoče. */
  if (a[0] < a[1]) t = a[1], a[1] = a[0], a[0] = t;
  if (a[1] < a[2]) t = a[2], a[2] = a[1], a[1] = t;
  if (a[0] < a[1]) t = a[1], a[1] = a[0], a[0] = t;
  /* Uredimo kocki drugega igralca padajoče. */
  if (b[0] < b[1]) t = b[1], b[1] = b[0], b[0] = t;
  /* Primerjajmo istoležne kocke obeh igralcev. */
  ta = 0; tb = 0;
  for (i = 0; i < 2; i++)
    if (a[i] > b[i]) ta += 2;
    else if (a[i] < b[i]) tb += 2;
    else ta += 1, tb += 1;
  /* Izpišimo rezultate. */
  printf("Prvi igralec dobi %d točk, drugi pa %d.", ta, tb);
  return 0;
}
```

4. Eksplozija

Označimo koordinate središča eksplozije z (x_e, y_e). Opazimo lahko, da se predmeti, ki so že pred eksplozijo ležali levo od te točke ($x < x_e$), pri eksploziji premaknejo še dlje proti levi; tisti, ki so ležali desno od nje ($x > x_e$), se premaknejo še dlje proti desni; tisti, ki so ležali nad točko eksplozije ($y > y_e$), se pomaknejo še bolj gor; tisti pa, ki so ležali pod točko eksplozije ($y < y_e$), se pomaknejo še bolj navzdol

Oglejmo si zdaj enega od šestnajstih enotskih kvadratkov naše mreže 4×4. Pred eksplozijo je v vsakem od njegovih štirih oglišč stal po en predmet. Kje so zdaj ti predmeti po eksploziji?

Če naš kvadratek leži levo zgoraj od tistega, v katerem je prišlo do eksplozije, se je tisti predmet, ki je bil prej v spodnjem desnem kotu kvadratka, premaknil gor in levo, torej zdaj leži v notranjosti našega kvadratka (ker je dolžina premika le 1/2, je nemogoče, da bi se tak predmet pomaknil čez naš kvadratek — za kaj takega bi moral biti premik daljši od 1).

Podoben razmislek lahko opravimo tudi v primerih, ko leži naš kvadratek desno zgoraj, levo spodaj ali desno spodaj od tistega, v katerem pride do eksplozije. V vsakem primeru torej nek predmet po eksploziji leži v notranjosti našega kvadratka.

Če pa naš kvadratek leži v istem stolpcu kot tisti, v katerem je prišlo do eksplozije, to pomeni, da sta se od predmetov, ki so pred eksplozijo ležali v njegovih ogliščih, leva dva premaknila v levo, desna dva pa v desno, torej zdaj nobeden od njih ne leži v notranjosti našega kvadratka. Podoben razmislek lahko uporabimo tudi v kvadratkih, ki ležijo v isti vrstici kot tisti, v katerem je prišlo eksplozije.

Opazimo lahko tudi, da ni mogoče, da bi se po eksploziji v notranjosti nekega kvadratka nahajal kakšen tak predmet, ki pred eksplozijo ni ležal v enem od oglišč tega kvadratka. Vsi ostali predmeti so namreč od našega kvadratka oddaljeni za vsaj 1 dolžinsko enoto, premaknejo pa se le za 1/2 enote, torej našega kvadratka ne morejo niti doseči, kaj šele se premakniti v njegovo notranjost.

Nalogo lahko torej rešimo tako, da za vsako vrstico in vsak stolpec mreže pogledamo, ali po eksploziji v njej leži kakšen predmet ali ne. Pri natanko eni vrstici in enem stolpcu bomo opazili, da ne vsebuje nobenega predmeta; potem vemo, da je do eksplozije prišlo v središču kvadratka, ki leži na preseku te vrstice in tega stolpca.

Zapišimo našo rešitev še v C-ju. V zanki bomo brali koordinate točk in vsako zaokrožili navzdol na celo število, da vidimo, v notranjosti katere vrstice oz. stolpca leži ta točka. Nato povečamo ustrezni števec točk za tisto vrstico in stolpec (tabeli vrstice in stolpci). Ko na ta način obdelamo vse točke, moramo le še poiskati v vsaki od teh dveh tabel element z vrednostjo 0.

```
#include <stdio.h>
```

```
#include <math.h>

int main()
{
  int vrstice[4], stolpci[4], i, xx, yy;
  double x, y;

  /* Inicializirajmo števce v obeh tabelah na 0. */
  for (xx = 0; xx < 4; xx++) stolpci[xx] = 0;
  for (yy = 0; yy < 4; yy++) vrstice[yy] = 0;
  /* Preberimo koordinate vseh 25 predmetov. */
  for (i = 0; i < 25; i++)
  {
    scanf("%lf %lf", &x, &y);
    /* V kateri vrstici in stolpcu leži zdaj ta predmet? */
    xx = (int) floor(x); yy = (int) floor(y);
    /* Povečajmo števce (če leži zunaj mreže, ga ignorirajmo). */
    if (0 <= xx && xx < 4) stolpci[xx]++;
    if (0 <= yy && yy < 4) vrstice[yy]++;
  }
  /* Poglejmo, v kateri vrstici in stolpcu ni nobenega predmeta. */
  for (xx = 0; xx < 4; xx++) if (stolpci[xx] == 0) x = xx + 0.5;
  for (yy = 0; yy < 4; yy++) if (vrstice[yy] == 0) y = yy + 0.5;
  /* Izpišimo rezultat. */
  printf("Do eksplozije je prišlo v točki (%.1f, %.1f).\n", x, y);
  return 0;
}
```

5. Barvanje plošče

Brez izgube za splošnost lahko predpostavimo, da je začetni položaj plošče enak $z = 0$. Če ni, je vse, kar moramo narediti, to, da tabelo pobarvaj ciklično zamaknemo za z mest navzdol: izsek, ki je bil prej na indeksu z, je zdaj na indeksu 0; tisti, ki je bil prej na indeksu $(z + 1) \bmod n$, je zdaj na indeksu 1 in tako naprej. V nadaljevanju bomo torej predpostavili, da je $z = 0$.

Na število operacij Pobarvaj ne moremo kaj dosti vplivati: vsak izsek, ki bo moral biti v končnem stanju plošče pobarvan, moramo pobarvati vsaj enkrat (na primer takrat, ko pridemo prvič do njega), nobene koristi pa ni od tega, da bi ga pobarvali več kot enkrat. Število izvajanj operacije Pobarvaj bo torej kar enako številu izsekov, ki jih je treba pobarvati črno. Tega ni težko izračunati tako, da se v zanki zapeljemo po tabeli pobarvaj in štejemo vrednosti **true**.

Vplivamo lahko torej le na število zasukov plošče. Rekli bomo, da pri našem barvanju plošče *obiščemo* nek izsek, če se ta izsek vsaj enkrat znajde pod barvno glavo. Ker lahko ploščo vedno obračamo le za en izsek levo ali desno, to pomeni, da obiskani izseki tvorijo neko strnjeno skupino — vsi obiskani izseki se držijo skupaj v enem kosu, ravno tako pa se tudi vsi ostali, neobiskani izseki držijo skupaj v enem kosu. Med obiskanimi izseki je seveda vedno tudi izsek 0, ki je pod barvno glavo že na začetku postopka.

Obiskano območje lahko torej opišemo preprosto tako, da povemo, pri katerem izseku se začne in pri katerem se konča. Recimo, da nazaj od $z = 0$ obsega izseke $n - 1, n - 2, \ldots, n - a$ (če nazaj od $z = 0$ nismo obiskali sploh nobenega izseka, si mislimo $a = 0$), naprej od $z = 0$ pa izseke $1, 2, \ldots, b$ (če naprej od $z = 0$ nismo obiskali sploh nobenega izseka, si mislimo $b = 0$). To je seveda smiselno le, če je $b < n - a$, saj se drugače začneta levi in desni del prekrivati, kar ni smiselno, ker že pri $b = n - a - 1$ obiščemo čisto vse izseke na plošči. Da bo rešitev sploh lahko veljavna, mora tudi veljati, da nobenega od odsekov $b + 1, b + 2, \ldots, n - a - 1$ ni treba pobarvati (saj jih med obračanjem plošče nikoli ne bomo dosegli in jih torej tudi pobarvati ne bi mogli).

Pri danih a in b je, kar se tiče samega barvanja, vseeno, v kakšnem vrstnem redu obiščemo te izseke, saj jih bomo v vsakem primeru lahko pobarvali (vsak izsek, ki ga je treba pobarvati, lahko pobarvamo takrat, ko prvič pridemo do njega). Smiselno je torej, da si zaporedje zasukov izbremo tako, da bomo vse te izseke dosegli s čim manj zasuki plošče. Ena možnost je na primer, da najprej izvedemo b zasukov v levo (pri tem pridejo

pod barvno glavo po vrsti izseki od 1 do b) in nato še $b + a$ zasukov v desno (s čimer pridejo pod barvno glavo po vrsti izseki $b - 1, \ldots, 0, n - 1, \ldots, n - a$). Druga možnost je, da najprej izvedemo a zasukov v desno in nato $a + b$ zasukov v levo. Med tema dvema možnostma vzemimo tisto, ki zahteva manj zasukov — to bo $\min\{2a + b, a + 2b\}$ zasukov.

Ali obstaja še kakšno krajše zaporedje zasukov, s katerim bi lahko obiskali vse te izseke? Razmišljamo lahko takole: vsaj enkrat med našim sukanjem plošče moramo obiskati izsek $n - a$ in vsaj enkrat izsek b. Ločimo dve možnosti: (1) mogoče obiščemo izsek $n - a$ prej kot izsek b. Ker je na začetku naša plošča na položaju 0, izseka $n - a$ ne moremo obiskati prej kot v a zasukih od začetka postopka; ko pa smo enkrat na izseku $n - a$, od tam ne moremo priti do izseka b v prej kot v $a + b$ zasukih. Postopek tega tipa torej gotovo izvede vsaj $2a + b$ zasukov. (2) Mogoče pa obiščemo izsek b prej kot izsek a. Podoben razmislek kot pri (1) nam zdaj pokaže, da tak postopek gotovo izvede vsej $a + 2b$ zasukov. Če ugotovitvi (1) in (2) združimo, vidimo, da je nemogoče, da bi dosegli vse izseke od $n - a$ do b z manj kot $\min\{2a + b, a + 2b\}$ zasuki. Razpored iz prejšnjega odstavka pa je porabil natanko toliko zasukov, torej boljšega razporeda ni.

Vprašanje je še, kako naj si izberemo a in b. Recimo, da je $b > 0$ in da izseka b ni treba pobarvati črno. V tem primeru ni nobene koristi od tega, da izsek b sploh obiščemo — število b lahko zmanjšamo za 1 in potrebno število premikov, $\min\{2a + b, a + 2b\}$, se lahko ob tem le zmanjša ali ostane enako, gotovo pa se ne more povečati. Podobno velja tudi, če je $a > 0$ in $n - a$ ni treba pobarvati črno — takrat lahko zmanjšamo a za 1 in rešitve gotovo ne poslabšamo.

Vidimo torej, da je smiselno za b izbrati le tako številko izseka, ki ga je treba pobarvati črno (poleg tega pa še $b = 0$, kar pokrije možnost, da ploščo od začetnega položaja vrtimo le v desno); in da je potem za a smiselno izbrati število naslednjega izseka (od $b + 1$ do $n - 1$), ki ga je treba pobarvati črno; če pa takega ni, lahko vzamemo $a = n$. Med vsemi tako dobljeni pari (a, b) bomo uporabili tistega, ki nam dá najmanjšo vrednost $\min\{2a + b, a + 2b\}$.

Zapišimo našo rešitev še v C-ju (če odmislimo del, ki bi moral v primerih, ko je prvotni z različen od 0, zamakniti tabelo pobarvaj in potem postaviti z na 0):

```
int KolikoOperacij(int n, bool pobarvaj[])
{
  int stBarvanj = 0, stZasukov = n, na, a, b;
  /* Preštejmo, koliko barvanj potrebujemo. */
  for (b = 0; b < n; b++) if (pobarvaj[b]) stBarvanj++;
  if (stBarvanj == 0) return 0;
  /* Preglejmo vse primerne b in izračunajmo potrebno število zasukov. */
  for (b = 0; b < n; )
  {
    /* b = 0 je koristen tudi, če tega izseka ni treba pobarvati; s tem pokrijemo
       primere, ko sukamo ploščo le desno od začetnega položaja. */
    /* Naj bo „na“ (= n − a) naslednji izsek, ki ga je treba pobarvati. */
    na = b + 1;
    while (na < n && ! pobarvaj[na]) na++;
    a = n − na;
    /* Izračunajmo potrebno število zasukov; če je najboljše doslej,
       si ga zapomnimo. */
    if (2 * a + b < stZasukov) stZasukov = 2 * a + b;
    if (a + 2 * b < stZasukov) stZasukov = a + 2 * b;
    /* Premaknimo se na naslednji izsek, ki ga je treba pobarvati. */
    b = na;
  }
  return stBarvanj + stZasukov;
}
```

11. srednješolsko tekmovanje ACM v znanju računalništva

Šolsko tekmovanje

22. januarja 2016

NASVETI ZA MENTORJE O IZVEDBI TEKMOVANJA IN OCENJEVANJU

Tekmovalci naj pišejo svoje odgovore na papir ali pa jih natipkajo z računalnikom; ocenjevanje teh odgovorov poteka v vsakem primeru tako, da jih pregleda in oceni mentor (in ne npr. tako, da bi se poskušalo izvorno kodo, ki so jo tekmovalci napisali v svojih odgovorih, prevesti na računalniku in pognati na kakšnih testnih podatkih). Pri reševanju si lahko tekmovalci pomagajo tudi z literaturo in/ali zapiski, ni pa mišljeno, da bi imeli med reševanjem dostop do interneta ali do kakšnih datotek, ki bi si jih pred tekmovanjem pripravili sami. Čas reševanja je omejen na 180 minut.

Nekatere naloge kot odgovor zahtevajo program ali podprogram v kakšnem konkretnem programskem jeziku, nekatere naloge pa so tipa „opiši postopek“. Pri slednjih je načeloma vseeno, v kakšni obliki je postopek opisan (naravni jezik, psevdokoda, diagram poteka, izvorna koda v kakšnem programskem jeziku, ipd.), samo da je ta opis dovolj jasen in podroben in je iz njega razvidno, da tekmovalec razume rešitev problema.

Glede tega, katere programske jezike tekmovalci uporabljajo, naše tekmovanje ne postavlja posebnih omejitev, niti pri nalogah, pri katerih je rešitev v nekaterih jezikih znatno krajša in enostavnejša kot v drugih (npr. uporaba perla ali pythona pri problemih na temo obdelave nizov).

Kjer se v tekmovalčevem odgovoru pojavlja izvorna koda, naj bo pri ocenjevanju poudarek predvsem na vsebinski pravilnosti, ne pa na sintaktični. Pri ocenjevanju na državnem tekmovanju zaradi manjkajočih podpičij in podobnih sintaktičnih napak odbijemo mogoče kvečjemu eno točko od dvajsetih; glavno vprašanje pri izvorni kodi je, ali se v njej skriva pravilen postopek za rešitev problema. Ravno tako ni nič hudega, če npr. tekmovalec v rešitvi v C-ju pozabi na začetku **#include**ati kakšnega od standardnih headerjev, ki bi jih sicer njegov program potreboval; ali pa če podprogram `main()` napiše tako, da vrača **void** namesto **int**.

Pri vsaki nalogi je možno doseči od 0 do 20 točk. Od rešitve pričakujemo predvsem to, da je pravilna (= da predlagani postopek ali podprogram vrača pravilne rezultate), poleg tega pa je zaželeno tudi, da je učinkovita (manj učinkovite rešitve dobijo manj točk).

Če tekmovalec pri neki nalogi ni uspel sestaviti cele rešitve, pač pa je prehodil vsaj del poti do nje in so v njegovem odgovoru razvidne vsaj nekatere od idej, ki jih rešitev tiste naloge potrebuje, naj vendarle dobi delež točk, ki je približno v skladu s tem, kolikšen delež rešitve je našel.

Če v besedilu naloge ni drugače navedeno, lahko tekmovalčeva rešitev vedno predpostavi, da so vhodni podatki, s katerimi dela, podani v takšni obliki in v okviru takšnih omejitev, kot jih zagotavlja naloga. Tekmovalcem torej načeloma ni treba pisati rešitev, ki bi bile odporne na razne napake v vhodnih podatkih.

V nadaljevanju podajamo še nekaj nasvetov za ocenjevanje pri posameznih nalogah.

1. Nadležne besede

- Rešitev sme predpostaviti, da je v vhodnih podatkih vsaj ena beseda (torej da seznam vhodnih besed ni prazen).
- Naš primer rešitve bere vhodne podatke do EOF, enako dobre pa so tudi rešitve, ki predpostavijo, da je konec podatkov označen kako drugače, na primer s prazno vrstico, ali pa da je na začetku vhoda podano najprej število besed.

- Rešitve, ki po nepotrebnem preberejo celoten seznam vhodnih besed v pomnilnik (namesto da bi jih brale in obdelovale sproti), naj se zaradi tega odšteje tri točke.
- Rešitve smejo predpostaviti, da nobena vhodna beseda ni prazna. Z drugimi besedami, vsaka vhodna beseda je dolga vsaj en znak.
- Če ima rešitev pravilno zastavljen mehanizem za ugotavljanje, s katero tipko se tipka določeno črko, vendar ima kakšno napako v s tem povezanih konstantah (kot je npr. tabela `tipka` v naši rešitvi), naj se ji zaradi tega odšteje največ dve točki.

2. Prepisovanje

- Naša rešitev pri tej nalogi sproti bere in pozablja števila iz vhodne datoteke; za enako dobro pa naj velja tudi rešitev, ki bi vsa števila prebrala v tabelo v pomnilniku in jih šele potem začela obdelovati.
- Mogoče je, da so osumljeni trije (ali več) zaporedni učenci. Rešitvam, ki pri štetju osumljenih v takem primeru srednjega od teh učencev štejejo dvojno, naj se zaradi tega odšteje sedem točk.
- Naloga pravi, da učence osumimo prepisovanja, če se dve sosednji števili razlikujeta za t ali manj. Če rešitev pomotoma sumi na prepisovanje le, če se števili razlikujeta za strogo manj kot t, ne pa tudi takrat, ko se razlikujeta za natanko t, naj se ji zaradi tega odšteje dve točki.
- Če poskuša rešitev pomotoma primerjati s prejšnjim učencem tudi prvega (ki prejšnjega učenca sploh nima), naj se ji zaradi tega odšteje tri točke.
- V naši rešitvi za posebno obravnavo prvega učenca poskrbimo tako, da preverjamo, če je indeks `i` že večji od 0 ali še ne. Enako dobra možnost bi bila tudi, da bi spremenljivko `prejsnji` pred glavno zanko inicializirali na neko vrednost, ki je gotovo za več kot t različna od katerega koli veljavnega števila (na primer na $10^9 + t + 1$).
- Če bi slučajno kakšna rešitev imela časovno zahtevnost $O(n^2)$ namesto le $O(n)$, naj dobi največ 13 točk (če je drugače pravilna).

3. Riziko

- Pri tej rešitvi podrobnosti branja vhodnih podatkov in izpisa rezultatov niso pomembne, tako da je vseeno, kako program bere vhodne podatke, ali pri tem tudi še kaj izpiše (npr. vprašanja uporabniku, ki naj bi vnašal podatke prek konzole), kako točno je formatiran izpis rezultatov ipd. Enako dobra bi bila tudi rešitev, ki bi predpostavila, da so podatki že na voljo v nekih spremenljivkah oz. kot parametri podprograma, pomembno je le, da ne predpostavi, da so podatki že urejeni.
- Če rešitev uredi kocke posameznega igralca naraščajoče namesto padajoče, naj se ji zaradi tega odšteje dve točki. Če na urejanje popolnoma pozabi, naj se ji odšteje pet točk.
- Naša rešitev ureja kocke vsakega od igralcev sama, enako dobro pa je tudi, če rešitev za urejanje uporabi kakšne funkcije iz standardne knjižnice svojega programskega jezika.
- Če rešitev sama implementira urejanje, je vseeno, po kakšnem postopku ureja (če je le rezultat pravilen). Naš primer rešitve uporablja bubble sort, možni pa so seveda še mnogi drugi postopki.

4. Eksplozija

- Poudarek pri tej nalogi je na ideji rešitve, ne na podrobnostih implementacije (in še posebej ne na obravnavanju morebitnih zaokrožitvenih napak pri delu s števili s plavajočo vejico). Pomembno pa je, da tekmovalčev odgovor vsebuje nekakšno utemeljitev, iz katere je razvidno, da je tekmovalec razumel, *zakaj* je njegov postopek pravilen, in ne le, da ima pač občutek, da je pravilen.

- Poleg postopka, ki je opisan v naših rešitvah, so seveda možne še drugačne rešitve, ki tudi lahko dosežejo vse točke, če dajejo pravilne rezultate (in so dobro utemeljene). Na primer, ena možnost je, da točke (po eksploziji) uredimo po y, jih v tem vrstnem redu razdelimo na skupine po 5 točk in nato vsako skupino uredimo še po x. Pokazati je mogoče, da tako dobimo enak vrstni red točk, kot če bi enako urejanje izvedli po eksploziji — najprej imamo torej koordinate predmeta, ki je bil pred eksplozijo na $(0, 0)$, nato tistega, ki je bil pred eksplozijo na $(0, 1)$ in tako naprej. Zdaj lahko za vsak predmet potegnemo premico skozi njegov prvotni položaj in njegov položaj po eksploziji ter poiščemo presečišče vseh tako dobljenih premic; tam je prišlo do eksplozije.

- Opazimo lahko, da je pri tej nalogi mogočih pravzaprav le 16 položajev eksplozije. Možna rešitev je torej tudi ta, da za vsako od teh 16 možnosti izračuna položaje vseh točk po eksploziji, nato pa vhodni nabor 25 točk primerja z vsemi 16 razporedi, da vidi, s katerim se ujema (pri tem mora paziti tudi na dejstvo, da so točke v vhodnih podatkih lahko poljubno premešane). Tudi taka rešitev lahko dobi vse točke, če je pravilno zamišljena in utemeljena. Če je primerjanje vhodnega razporeda z ostalimi 16 razporedi izvedeno na zelo neučinkovit način (na primer tako, da vhodni razpored premešamo na 25! možnih načinov), naj taka rešitev dobi največ 10 točk (če je drugače pravilna).

- Naloga pravi, da so predmeti v naših vhodnih podatkih lahko poljubno premešani. Rešitev, ki predpostavi, da so koordinate predmetov po eksploziji navedene v nekem konkretnem (in koristnem) vrstnem redu (npr. tako, da najprej pride položaj predmeta, ki je bil pred eksplozijo na $(0, 0)$, nato položaj predmeta, ki je bil pred eksplozijo na $(0, 1)$ itd.), naj dobi največ 13 točk (če je drugače pravilna).

5. Barvanje plošče

- Za vsak primer poudarimo, da naloga ne zahteva, naj rešitev izpiše ali kako drugače zgenerira konkretno zaporedje operacij, ki bi ustrezno pobarvalo ploščo — rešitev mora le izračunati najmanjše potrebno število operacij.

- Postopek, kot smo ga opisali v C-ju na koncu naše rešitve, ima časovno zahtevnost $O(n)$. Z malo manj pazljivosti pri implementaciji zanke, ki mora pregledati vse b in pri vsakem najti primeren a, si lahko predstavljamo rešitve s časovno zahtevnostjo $O(n^2)$ ali celo $O(n^3)$. Take rešitve naj pri tej nalogi dobijo največ 18 točk (če so drugače pravilne).

- Glede utemeljitve pravilnosti rešitve pričakujemo predvsem nekakšen razmislek o tem, da nam smeri obračanja plošče ni treba spremeniti več kot enkrat.

- Ekstremno neučinkovite rešitve — npr. če bi nekdo generiral vsa možna zaporedja $2n$ ali manj operacij in preverjal, če pravilno pobarvajo ploščo (in med takimi izpisal dolžino najkrajšega) — naj dobijo največ 8 točk (če so drugače pravilne).

- Če rešitev ne deluje za poljuben z, ampak le za eno konkretno vrednost z-ja (na primer $z = 0$) in pri tem ne razloži, kako lahko vhodne primere z drugačnim z predelamo na to konkretno vrednost, naj se ji odšteje dve točki.

Težavnost nalog

Državno tekmovanje ACM v znanju računalništva poteka v treh težavnostnih skupinah (prva je najlažja, tretja pa najtežja); na tem šolskem tekmovanju pa je skupina ena sama, vendar naloge v njej pokrivajo razmeroma širok razpon zahtevnosti. Za občutek povejmo, s katero skupino državnega tekmovanja so po svoji težavnosti primerljive posamezne naloge letošnjega šolskega tekmovanja:

Naloga	Kam bi sodila po težavnosti na državnem tekmovanju ACM
1. Nadležne besede	lažja do srednja naloga v prvi skupini
2. Prepisovanje	srednja v prvi ali lahka naloga v drugi skupini
3. Riziko	srednja v prvi ali lažja naloga v drugi skupini
4. Eksplozija	srednja do težja naloga v drugi skupini
5. Barvanje plošče	težja v drugi ali srednja v tretji skupini

Če torej na primer nek tekmovalec reši le eno ali dve lažji nalogi, pri ostalih pa ne naredi (skoraj) ničesar, to še ne pomeni, da ni primeren za udeležbo na državnem tekmovanju; pač pa je najbrž pametno, če na državnem tekmovanju ne gre v drugo ali tretjo skupino, pač pa v prvo.

Podobno kot prejšnja leta si tudi letos želimo, da bi čim več tekmovalcev s šolskega tekmovanja prišlo tudi na državno tekmovanje in da bi bilo šolsko tekmovanje predvsem v pomoč tekmovalcem in mentorjem pri razmišljanju o tem, v kateri težavnostni skupini državnega tekmovanja naj kdo tekmuje.