5. srednješolsko tekmovanje ACM in IJS v znanju računalništva

Šolsko tekmovanje

28. januarja 2011

NASVETI ZA TEKMOVALCE

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.");
   }
}
```

5. srednješolsko tekmovanje ACM in IJS v znanju računalništva

Šolsko tekmovanje

28. januarja 2011

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. Primerjanje oklepajev

Komisija za računalniško tekmovanje ugotavlja, ali sta dve rešitvi naloge morda prepisani, tako, da iz njiju pobriše vse znake *razen* oklepajev `(`, `)`, `[`, `]`, `{` in `}` in nato primerja preostanka. Če sta enaka, šteje rešitvi za sumljivo podobni.

Napiši podprogram Prepisovanje(a, b), ki vrne **true**, če niza a in b predstavljata dve rešitvi naloge, ki sta sumljivo podobni. V nasprotnem primeru naj podprogram vrne **false**.

Tvoj podprogram naj bo takšne oblike:

```
bool Prepisovanje(char *a, char *b);                  /* v C/C++ */
public static bool Prepisovanje(String a, String b);  // v javi
function Prepisovanje(a, b: string): boolean;         { v pascalu }
def Prepisovanje(a, b): ...                           # v pythonu
```

2. Globalno segrevanje

Za naslednjih n let imamo po letih dane ocene, kako se bo zaradi globalnega segrevanja dvigovala gladina morij (predpostaviš lahko, da bo gladina morja vsako leto vsaj tako visoka kot prejšnje leto). Imamo tudi seznam trenutnih nadmorskih višin m obmorskih krajev po vsem svetu. Predpostaviš lahko, da je seznam krajev že urejen naraščajoče po nadmorski višini.

Opiši postopek, ki bo po letih izpisal vrstni red krajev, ki naj bi se vsako leto potopili zaradi dviga gladine morij. Tvoj postopek naj bo učinkovit, tako da bo hitro deloval tudi za velike n in m.

3. Riziko

Dva igralca se igrata naslednjo igro. Prvi igralec vrže tri kocke in jih uredi v nenaraščajočem vrstnem redu po številu pik; označimo jih z a_1, a_2 in a_3 (velja torej $a_1 \geq a_2 \geq a_3$). Drugi igralec vrže dve kocki in ju tudi uredi v nenaraščajočem vrstnem redu pik; označimo ju z b_1 in b_2 (velja torej $b_1 \geq b_2$). Nato igralca primerjata število pik na svojih kockah. Če je $a_1 > b_1$, dobi prvi igralec eno točko, sicer jo dobi drugi. Podobno, če je $a_2 > b_2$, dobi prvi igralec eno točko, sicer jo dobi drugi.

Met petih kock se lahko izide na $6 \cdot 6 \cdot 6 \cdot 6 \cdot 6 = 7776$ različnih načinov. **Napiši program**, ki ugotovi, koliko od teh 7776 izidov pripelje do tega, da dobi prvi igralec dve točki, koliko do tega, da dobi drugi igralec dve točki, in koliko do tega, da dobi vsak od igralcev po eno točko.

4. Preglednice

Programi za delo s preglednicami (angl. *spreadsheet*) pogosto omogočajo izvoz v razne preproste tekstovne formate. Pri tej nalogi predpostavimo, da imamo celo tabelo zapisano kot en sam dolg niz znakov, pri čemer so posamezne vrstice tabele ločene s podpičji („`;`“), posamezne celice znotraj vsake vrstice pa z vejicami („`,`“).

Napiši funkcijo NaslednjiZnak(**char** c), ki ji podajamo tako zapisano tabelo znak za znakom (v parametru c), ta funkcija pa naj kliče funkcijo

```
Celica(int vrstica, int stolpec, char *vsebina)
```

čim prebere celotno vsebino celice (poleg vsebine celice naj poda tudi številko vrstice in stolpca, v kateri se nahaja; številčenje vrstic in stolpcev se prične pri 1). Za funkcijo Celica torej predpostavi, da že obstaja; tebi ni treba pisati nikakršne implementacije te funkcije, pač pa jo moraš le poklicati iz svoje funkcije NaslednjiZnak.

Primer: če je tabela predstavljena z nizom

```
Ime,Priimek;Janez,Novak;Ena,,Tri;...
```

lahko pričakuješ naslednje klice funkcije NaslednjiZnak:

```
NaslednjiZnak('I');
NaslednjiZnak('m');
NaslednjiZnak('e');
NaslednjiZnak(',');
NaslednjiZnak('P');
NaslednjiZnak('r');
⋮
```

ki jo moraš napisati tako, da bo klicala funkcijo Celica:

```
Celica(1, 1, "Ime");
Celica(1, 2, "Priimek");
Celica(2, 1, "Janez");
Celica(2, 2, "Novak");
Celica(3, 1, "Ena");
Celica(3, 2, "");
Celica(3, 3, "Tri");
```

Predpostaviš lahko, da je vsebina celice vedno krajša od 100 znakov in da ne vsebuje vejic ali podpičij. Uporabljati smeš tudi globalne spremenljivke, ki jih lahko pred prvim klicem funkcije NaslednjiZnak po svoje inicializiraš.

Še deklaracije v drugih jezikih:

```
procedure NaslednjiZnak(c: char);
procedure Celica(Vrstica, Stolpec: integer; Vsebina: string);

public static void naslednjiZnak(char c);
public static void celica(int vrstica, int stolpec, String vsebina);

def NaslednjiZnak(c): ...
def Celica(vrstica, stolpec, vsebina): ...
```

5. Smučarji

Dan je seznam m otrok in n parov smuči. Smuči so dovolj močne, da lahko zdržijo vsakega otroka; problem je v tem, da lažji otroci ne morejo peljati pretežkih smuči. Za vsakega otroka imamo njegovo težo in za vsak par smuči imamo minimalno težo otroka, ki ju še lahko pelje. **Opiši postopek**, ki iz teh podatkov ugotovi, koliko največ otrok lahko smuča. Poleg tega tudi utemelji, zakaj je tvoja rešitev pravilna (torej zakaj je rezultat, ki ga poišče tvoj postopek, res največje možno število otrok, ki jim je mogoče razdeliti smuči tako, da nihče ni prelahek za dobljeni par smuči).

5. srednješolsko tekmovanje ACM in IJS v znanju računalništva

Šolsko tekmovanje

28. januarja 2011

REŠITVE NALOG ŠOLSKEGA TEKMOVANJA

1. Primerjanje oklepajev

Spodnja rešitev v zanki pregleduje oba podana niza. V vsaki iteraciji poišče naslednji oklepaj (ali zaklepaj) v nizu `a` in naslednji oklepaj (ali zaklepaj) v nizu `b` ter ju primerja med sabo. Čim opazimo kakšno neujemanje (sem šteje tudi možnost, da pridemo pri enem nizu do konca prej kot pri drugem), se ustavimo in vrnemo **false**. Niza sta si sumljivo podobna le, če smo pri obeh prišli hkrati do konca, ne da bi opazili kakšno neujemanje.

```
bool Prepisovanje(const char *a, const char *b)
{
  while (true)
  {
    /* Poiščimo naslednji oklepaj/zaklepaj v vsakem nizu. */
    while (*a && ! strchr("()[]", *a)) a++;
    while (*b && ! strchr("()[]", *b)) b++;
    /* Če smo opazili neujemanje ali prišli do konca nizov, končajmo. */
    if (*a != *b) return false;
    if (! *a) return true;
    /* Premaknimo se naprej od trenutnih oklepajev/zaklepajev. */
    a++; b++;
  }
}
```

2. Globalno segrevanje

Elegantna rešitev je, da se hkrati sprehajamo po letih in po krajih. Za vsako leto se premikamo naprej po seznamu krajev in izpisujemo kraje, ki se tisto leto potopijo; čim pa pridemo do kraja, ki leži tako visoko, da se to leto še ne potopi, lahko s trenutnim letom končamo, pri naslednjem letu pa bomo s pregledovanjem seznama krajev nadaljevali prav tu, kjer smo pri trenutnem letu končali. Postopek lahko opišemo tudi v psevdokodi:

```
postavimo se na začetek seznama krajev;
za vsako leto y od 1 do n:
  naj bo g_y gladina morja v letu y;
  izpiši 'leta y se potopijo:';
  dokler še nismo na koncu seznama krajev:
    naj bo k trenutni kraj v seznamu in naj bo h_k njegova nadmorska višina;
    if h_k ≥ g_y then break;
    izpiši kraj k in se premakni na naslednji kraj v seznamu;
```

Lepo pri tej rešitvi je, da pregleda vsak seznam (let in krajev) le po enkrat.

3. Riziko

Spodnja rešitev s petimi gnezdenimi zankami pregleda vse možne mete petih kock. Pri vsaki kombinaciji uredimo kocke vsakega igralca v padajočem vrstnem redu in nato po pravilih igre pogledamo, kakšen je izid te igre (2:0, 1:1 ali 0:2). Število izidov vsakega tipa hranimo v spremenljivkah `n02`, `n11` in `n20`, ki jih na koncu izpišemo.

```
#include <stdio.h>

int main()
{
  int A1, A2, A3, B1, B2, a1, a2, a3, b1, b2, t, n20 = 0, n11 = 0, n02 = 0;
  for (A1 = 1; A1 <= 6; A1++) for (A2 = 1; A2 <= 6; A2++) for (A3 = 1; A3 <= 6; A3++)
  for (B1 = 1; B1 <= 6; B1++) for (B2 = 1; B2 <= 6; B2++)
  {
    /* Kocke prvega igralca prepišimo iz A1, A2, A3 v a1, a2, a3 in jih uredimo padajoče.
       Enako tudi prepišimo kocki drugega igralca iz B1, B2 v b1, b2 in ju uredimo padajoče. */
    a1 = A1; a2 = A2; a3 = A3; b1 = B1; b2 = B2;
    if (a2 > a1) t = a1, a1 = a2, a2 = t;
    if (a3 > a2) t = a2, a2 = a3, a3 = t;
    if (a2 > a1) t = a1, a1 = a2, a2 = t;
    if (b2 > b1) t = b1, b1 = b2, b2 = t;
    /* Primerjajmo a1 in b1 ter a2 in b2 in povečajmo ustreznega od števcev n20, n11 in n02. */
    if (a1 > b1 && a2 > b2) n20++;
    else if (a1 > b1 || a2 > b2) n11++;
    else n02++;
  }
  printf("%d %d %d\n", n20, n11, n02);
}
```

4. Preglednice

Koristno je imeti globalno spremenljivko, v kateri si shranjujemo vsebino trenutne celice (ki jo naš podprogram NaslednjiZnak dobiva znak po znak); ko pridemo do konca celice (torej ko je naslednji znak vejica ali podpičje), pa jo izpišemo. Naša spodnja rešitev ima v ta namen tabelo vsebina, kazalec trenutniZnak pa označuje naš položaj v tej tabeli. Tudi podatek o tem, v kateri vrstici in stolpcu se trenutno nahajamo, hranimo v globalnih spremenljivkah; ko preberemo vejico, povečamo koordinato stolpca, ko pa preberemo podpičje, povečamo koordinato vrstice in postavimo stolpec spet na 1.

```
int vrstica = 1, stolpec = 1;
char vsebina[100];
char *trenutniZnak = vsebina;

void NaslednjiZnak(char c)
{
  if (c == ';') /* konec vrstice */
  {
    *trenutniZnak = 0;
    Celica(vrstica, stolpec, vsebina);
    ++vrstica; stolpec = 1;
    trenutniZnak = vsebina;
  }
  else if (c == ',') /* konec celice */
  {
    *trenutniZnak = 0;
    Celica(vrstica, stolpec, vsebina);
    ++stolpec;
    trenutniZnak = vsebina;
  }
  else
    *trenutniZnak++ = c;
}
```

5. Smučarji

Da bo manj pisanja, bomo pri tej rešitvi namesto o minimalni dopustni teži smučarja, ki jo zahteva nek par smuči, govorili kar o teži smuči.

Recimo zdaj, da gledamo otroka A in B, pri čemer je A težji od B. Če ima pri nekem veljavnem razporedu smuči otrok A lažje smuči kot otrok B, si jih lahko izmenjata, pa razpored ostane veljaven in enako dober. Podobno, če je A brez smuči, B pa jih ima, lahko B da svoje smuči A-ju, pa razpored ostane veljaven (in enako dober). Iz tega vidimo, da je dovolj, če se omejimo na takšne razporede, pri katerih dobi smuči le najtežjih nekaj otrok in pri katerih dobijo težji otroci vedno težje smuči kot lažji otroci.

Pregledujmo torej smuči padajoče po teži. Za najtežje smuči nam razmislek iz prejšnjega odstavka pove, da če jih bomo dali sploh kakšnemu otroku, jih moramo dati najtežjemu. Če je celo ta otrok zanje prelahek, vemo, da teh smuči ne moremo uporabiti; če pa je ta otrok zanje dovolj težak, ni nobenega dobrega razloga, da mu jih ne bi dali. Če mu bomo namreč dali neke lažje smuči, bodo tiste najtežje ostale neuporabljene, in tak razpored ostane veljaven (in enako dober), če temu otroku namesto lažjih smuči damo najtežje (saj smo rekli, da je on tudi za te najtežje dovolj težak), tiste lažje pa bodo mogoče prišle prav še za kakšnega lažjega otroka. Z enakim razmislekom nadaljujemo še z ostalimi pari smuči (v padajočem vrstnem redu po minimalni dopustni teži) — vsake damo najtežjemu takemu otroku, ki še ni dobil smuči, če pa je ta otrok prelahek zanje, ostanejo neuporabljene.

5. srednješolsko tekmovanje ACM in IJS v znanju računalništva

Šolsko tekmovanje

28. januarja 2011

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. Primerjanje oklepajev

- V naših rešitvah je naloga rešena tako, da se sprehaja po nizih `a` in `b`, preskakuje ne-oklepajske znake in primerja oklepaje. Za enako dobro naj šteje tudi rešitev, ki bi iz nizov najprej eksplicitno pobrisala ne-oklepajske nize in ju šele potem primerjala; tudi je vseeno, če si v ta name ustvari pomožno kopijo obeh nizov ali pa dela kar s tistima, ki sta podana kot parametra.

- Če bi rešitev zaradi kakšne hude neučinkovitnosti za primerjanje dveh nizov dolžine $O(n)$ porabila več kot $O(n)$ časa, na primer $O(n^2)$ ali kaj podobnega, naj dobi največ 10 točk (če drugače daje pravilne rezultate).

2. Globalno segrevanje

- Vseeno je, ali rešitev predpostavi, da so podatki v tabeli (array), v povezanem seznamu (linked list) ali pa celo v datoteki ali čem podobnem.
- Vseeno je, ali rešitev šteje mesto za potopljeno šele, ko gladina morja postane večja od njegove višine, ali pa že takrat, ko ji postane šele enaka.
- Če gre rešitev pri vsakem letu po celotnem seznamu vseh krajev, naj dobi kvečjemu 10 točk. Z drugimi besedami, radi bi rešitev, ki za m krajev in n let porabi le $O(n + m)$ časa, ne pa $O(n \cdot m)$ časa.
- Če rešitev naredi kakšne predpostavke o največji možni dolžini vhodnih zaporedij, naj se ji odbije največ 5 točk.

3. Riziko

- Če v primerih, ko je $a_1 = b_1$ (ali pa $a_2 = b_2$) rešitev pripiše točko prvemu igralcu (namesto drugemu, kot izhaja iz besedila naloge), naj se ji odbijejo 3 točke.
- Vseeno je, ali rešitev možne kombinacije petih kock našteva z gnezdenimi zankami, z rekurzijo ali še kako drugače.
- Za urejanje kock posameznega tekmovalca lahko uporabi rešitev morebitne podprograme za urejanje iz standardne knjižnice svojega programskega jezika (četudi je to za samo dve ali tri števila neučinkovito).
- Možna napaka pri tej nalogi je, da rešitev pregleduje le takšne kombinacije kock, ki so že urejene padajoče (**for** a1 := 1 **to** 6 **do for** a2 := 1 **to** a1 **do for** a3 := 1 to a2 ...). Če rešitev deluje na ta način, mora pri štetju pravilno upoštevati, da lahko vsaka taka kombinacija nastane z urejanjem več različnih prvotnih kombinacij (največ šestih). Če tega ne dela pravilno, naj dobi največ 10 točk.

4. Preglednice

- Naloga posebej pravi, naj kličemo funkcijo Celica, čim preberemo celotno vsebino celice. Če rešitev kliče funkcijo Celica šele na koncu vrstice ali celo na koncu preglednice, med tem pa si podatke za vse te celice kopiči v pomnilniku, naj se ji odbije 5 točk. Če ob tem naredi še kakšne dodatne predpostavke o največjem številu celic v vrstici/preglednici ali pa o skupni dolžini njihove vsebine, naj se ji odbije še 5 točk.
- Če rešitev pomotoma šteje vrstice in stolpce od 0 namesto od 1 naprej, naj se ji zaradi tega odbije največ 2 točki.

5. Smučarji

- Pri tej nalogi je koristno pregledovati smuči in otroke padajoče po teži. Vseeno je, ali se rešitev ukvarja s tem, kako vhodne podatke urediti, ali pa predpostavi, da so že urejeni; tudi je vseeno, če uporabi za urejanje kakšnega od algoritmov z zahtevnostjo $O(n^2)$ namesto $O(n \log n)$.
- Rešitve z eksponentno časovno zahtevnostjo (npr. take, ki z rekurzijo pregledajo vse možne razporede otrok na smuči) naj dobijo največ 10 točk (če drugače dajejo pravilne rezultate).
- Če rešitev temelji na požrešnem algoritmu (tako kot naša rešitev zgoraj), je vseeno, če je ta implementiran v času $O(n \cdot m)$ namesto $O(n + m)$ (če ne štejemo časa urejanja); tudi ni nujno, da je postopek opisan tako podrobno, da se to sploh vidi iz rešitve (tako kot se tudi ne vidi iz naše rešitve zgoraj).
- Pomembno je tudi, ali se iz odgovora vidi, da ima tekmovalec nekakšen argument za pravilnost svoje rešitve (in je ni zapisal le zato, ker se mu po občutku zdi, da je najbrž pravilna). (To seveda ne pomeni, da pričakujemo formalen dokaz z indukcijo (in mogoče še s protislovjem), kakršne se ponavadi uporablja pri požrešnih algoritmih.)

Težavnost nalog

Državno tekmovanje ACM in IJS 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 in IJS
1. Primerjanje oklepajev	srednje težka naloga v prvi skupini
2. Globalno segrevanje	težja v prvi ali lažja naloga v drugi skupini
3. Riziko	lažja ali srednje težka naloga v drugi skupini
4. Preglednice	srednja naloga v drugi skupini
5. Smučarji	težja naloga v drugi ali lažja v tretji skupini

Če torej na primer nek tekmovalec reši le prvo nalogo in del druge, 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.