Zadaci sa rešenjima Okružno takmičenje 2009.

zadatak: Golf

Đurajger Paunuds je profesionalni igrač golfa. Baš se danas vratio s turnira na kom je izvojevao veoma tešku, ali utoliko slađu pobedu, i sada zbog postturnirskog stresa ima problema s ružnim snovima. Đurajger sanja da se nalazi na ogromnom terenu za golf, na kom ga, sa željom da ga što dalje lansira, juri ogromna palica za golf! Srećom, na ovom ogromnom terenu za golf ima mnogo ogromnih rupa (kao što je očekivano), i Đurajger je svoju šansu video u tome da se sakrije u jednu od njih. On treba da odabere jednu rupu i potrči k njoj pravolinijski. Palica odmah zaključuje ka kojoj rupi Đurajger trči, i potrčaće ka istoj rupi dvaput većom brzinom. Treba pronaći rupu ka kojoj Đurajger treba da potrči kako bi se uspešno spasao, ako takva postoji.

Ulaz:

(Ulazni podaci se nalaze u datoteci **golf.in**) U prvom redu ulazne datoteke nalaze se dva realna broja, koji predstavljaju x-koordinatu i y-koordinatu Đurajgerove početne pozicije. U drugom redu ulazne datoteke nalaze se još dva realna broja, koji predstavljaju x-koordinatu i y-koordinatu početne pozicije palice. U trećem redu nalazi se prirodan broj n ($1 \leq n \leq 1.000.000$), koji predstav a broj rupa na igralixtu. U narednih n redova nalaze se po dva realna broja, pri qemu se u i-tom od tih redova nalaze koordinate i-te rupe.

Izlaz:

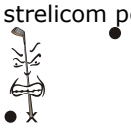
(Izlazne podatke upisati u datoteku **golf.out**) U prvi i jedini red izlazne datoteke upisati redni broj rupe u koju Đurajger treba da se sakrije (ukoliko postoji više takvih, naći bilo koju), odnosno "Nadrljao je!" ukoliko takva rupa ne postoji.

Primer:

golf.in	golf.out
0.0 0.0 5.0 5.0 3 1.0 1.0 9.0 9.0 4.0 5.0	1

Objašnjenje.

Slika ispod prikazuje raspored rupa, kao i pozicije na kojima se nalaze Đurajger i palica za prvi primer. Takođe je strelicom pokazano koja je to rupa spasonosna za Đurajgera.



Primer:

golf.in	golf.out
0.0 0.0 5.0 5.0 2 9.0 9.0 4.0 5.0	Nadrljao je!

fajl: golf.pas

```
var n,i:longint;
    dj1,dj2,p1,p2,x,y:real;
    q:boolean;
    f,g:text;
begin
 assign(f,'golf.in');
 assign(g,'golf.out');
 reset(f);
 rewrite(g);
 readln(f,dj1,dj2);
```

```
 readln(f,p1,p2);
 readln(f,n);
 q:=false;
 i:=1;
 repeat
    readln(f,x,y);
    if sqrt(sqr(p1-x)+sqr(p2-y))-2*sqrt(sqr(dj1-x)+sqr(dj2-y))>=-0.000001 then begin

writeln(g,i);
                                                                                q:=true;
                                                                               end;
    inc(i);
 until (i>n) or q;
 if not q then writeln(g,'Nadrljao je!');
 close(f);
 close(g);
end.
```

zadatak: Rafaelo

Mirko i Slavko su dobili na poklon nekoliko kutija rafelo kuglica. Da se ne bi posvađali oko raspodele, Mirko je predložio sledeće: naizmenično će uzimati (i jesti) po jednu rafaelo kuglicu iz proizvoljne kutije, i onaj ko uzme poslednju kuglu iz neke kutije dobija kao nagradu sve preostale kuglice. Pošto je Mirko predložio način raspodele, Slavko ima prednost da bira da li će prvi početi da uzima, ili će to zadovoljstvo prepustiti Mirku. Naravno, Slavko želi da pojede što više kuglica, pa je na vama da mu došapnete šta da radi (da li da uzima prvi ili drugi).
Pretpostavlja se da i Mirko i Slavko uzimaju tako da pojedu što je više moguće kuglica.

Ulaz:
(Ulazni podaci se nalaze u datoteci rafaelo.in) Ulazna datoteka sadrži tačno tri test primera. Svaki od prva tri reda ulazne datoteke sadrži sledeće podatke: broj kutija K ($2 \leq K \leq 50$), a zatim K brojeva iz opsega [1, 100] (oni predstavljaju količine kuglica u kutijama).

Izlaz:
(Izlazne podatke upisati u datoteku rafaelo.out) Za svaki od tri test primera iz ulazne datoteke, u poseban red izlazne datoteke ispisati 1 ako Slavko treba da uzima prvi, odnosno 2 ako treba da uzima drugi.

Primer 1:

rafaelo.in	rafaelo.out
2 2 2	2
3 3 2 1	1
4 3 2 3 3	1

Objašnjenje.
Važi sledeće:

1. kombinacija - iz koje god kutije prvi da uzme, u toj kutiji će ostati 1 kugilca, koju onda uzima drugi i time dobija i sve ostale kuglice
2. kombinacija - prvi može odmah da uzme kuglicu iz poslednje kutije, i time dobija sve ostale kuglice
3. kombinacija - ukoliko neko uzme kuglicu iz kutije koja ima 2 kugle, tada drugi dobija sve ostale. Naizmeničnim uzimanjem iz kutije koja ima 3 kugle, dobijamo da prvi može da pojede više.

fajl: rafaelo.cpp

```
/*
 *
 * Rafaelo kuglice, orkuzno 2008/09
 *
 * Autor : Rajko Nenadov (rajkon@gmail.com)
 *
 */

#include <cstdio>
using namespace std;
```

```
// najveci broj kutija
const int MaxK = 51;

int K;
int kuglice[MaxK];

int resenje[3];

int main()
{
  FILE *f;
  f = fopen("rafaelo.in", "r");
  for (int test = 0; test < 3; test++)
  {
    fscanf(f, "%d", &K);
    for (int i = 0; i < K; i++)
      fscanf(f, "%d", &kuglice[i]);

    // ukoliko neka kutija sadrzija 1 kuglicu, vishe ce pojesti prvi igrac
    bool jedna = false;
    for (int i = 0; !jedna && i < K; i++)
      if (kuglice[i] == 1)
        jedna = true;

    if (jedna)
      resenje[test] = 1;
    else
    {
      // inace, prvi pobedjuje ako je ukupan broj kuglica neparan
      // a drugi ako je paran.
      // objasnjenje :
      // ukoliko neki igrac uzme kuglicu iz kutije u kojoj ima 2 kuglice, pobedjuje njegov
protivnik.
      // kako na pocetku svaka kutija ima bar 2 kuglice, oni ce naizmenicno uzimati
(redosled nije bitan) iz kutije koja ima vise od 2 kuglice.
      // kada se dodje do situacije da se u svakoj kutiji nalaze tacno 2 kuglice, gubi
igrac koji je na potezu.
      // dakle, ukupan broj poteza koji ce se napraviti pre nego sto se dodje do te
situacije je :
      // (kuglice[0] - 2) + ... + (kuglice[K-1] - 2)
      // ukoliko je taj broj paran, znaci da je sledeci na potezu prvi igrac, i da on gubi;
      // ukoliko je neparan, sledeci na potezu je drugi igrac, i onda on gubi.

      int suma = 0;
      for (int i = 0; i < K; i++)
        suma += kuglice[i] - 2;
      if (suma % 2 == 0)
        resenje[test] = 2;
      else
        resenje[test] = 1;
    }
  }
  fclose(f);

  f = fopen("rafaelo.out", "w");
  for (int test = 0; test < 3; test++)
    fprintf(f, "%d\n", resenje[test]);
  fclose(f);

  return 0;
}
```

zadatak: Brojanje

Nakon što su se fino najeli rafaelo kuglica, Mirko i Slavko su odlučili da pomoću testa utvrde koliko je moguće držati koncentraciju sa punim stomakom. Test se sastoji u tome da Mirko govori brojeve Slavku (izgovarajući svaki put jedan od 400 omiljenih brojeva), i u proizvoljnom momentu traži od njega da mu kaže koji je *K*-ti broj po veličini od svih brojeva koje je rekao do tada. Na vama je da pomognete Slavku u odgovaranju na zadata pitanja. Razlog zašto želite da pomognete Slavku nije bitan.

Ulaz:

(Ulazni podaci se nalaze u datoteci brojanje.in) U prvom redu ulazno datoteke nalazi se broj N ($5 \leq N \leq 100.000$). Svaki od narednih N redova ima jedan od dva formata:

- 1 *a* - označava da je Mirko izgovorio broj a ($0 \leq a \leq 65535$).
- 2 *k* - označava da je Mirko tražio od Slavka da mu kaže koji je k-ti broj po veličini (garantuje se da će k biti manje ili jednako trenutnom broju izgovorenih brojeva) Napomenimo još jednom da će broj različitih izgovorenih brojeva biti ne veći od 400 (pojedini brojevi se mogu ponavljati).

Izlaz:

(Izlazne podatke upisati u datoteku brojanje.out) Za svaki red iz ulazne datoteke koji je oblika '2 *k*', ispisati u nov red izlazne datoteke odgovor na Mirkovo pitanje. Postojaće bar jedan takav red.

Primer:

brojanje.in	**brojanje.out**
7 1 0 1 1 1 5 2 1 2 3 1 2 2 3	

fajl: brojanje.cpp

```
/*
 *
 *  Brojanje, okruzno 2008/09
 *
 *  Autor : Rajko Nenadov (rajkon@gmail.com)
 *
 */

#include <cstdlib>
#include <cstdio>
#include <iostream>
using namespace std;

struct List
{
  long broj;
  long pojavljivanje;

  List *next;
};

// glava liste u kojoj cuvamo izgovorene brojeve
List *glava;

long N;

/*
 *   ukoliko se 'broj' nalazi u list, polje 'pojavljivanje' elementa u kom se nalazi
povecamo za 1;
 *  inace dodajemo novi element u listu, ali tako da lista ostane sortirana po 'broj'
polju.
 */
```

```
void ubaci(long broj)
{
  if (glava == NULL)
  {
    glava = (List*)malloc(sizeof(List));
    glava->broj = broj;
    glava->pojavljivanje = 1;
    glava->next = NULL;
  }
  else
  {
    List *prev = NULL;
    List *itr = glava;

    while (itr != NULL && (itr->broj < broj))
    {
      prev = itr;
      itr = itr->next;
    }

    if (itr != NULL && (itr->broj == broj))
    {
      // broj postoji u listi
      itr->pojavljivanje++;
    }
    else
    {
      List *novi = (List*)malloc(sizeof(List));
      novi->broj = broj;
      novi->pojavljivanje = 1;

      if (prev == NULL)
      {
        // dodajemo na pocetak liste
        novi->next = glava;
        glava = novi;
      }
      else
      {
        prev->next = novi;
        novi->next = itr;
      }
    }
  }
}


long nadji(long k)
{
  List *itr = glava;
  while (itr->pojavljivanje < k)
  {
    k -= itr->pojavljivanje;
    itr = itr->next;
  }

  return itr->broj;
}


int main()
{
  long long l1 = clock();
  FILE *fin = fopen("brojanje.01.in", "r");
  FILE *fout = fopen("brojanjer.01.out", "w");

  glava = NULL;
```

```
  fscanf(fin, "%ld", &N);
  for (long i = 0; i < N; i++)
  {
    long a, b;
    fscanf(fin, "%d %ld", &a, &b);

    if (a == 1)
      ubaci(b);
    else
      fprintf(fout, "%ld\n", nadji(b));
  }

  fclose(fin);
  fclose(fout);
  cout << clock() - l1 << endl;
  system("pause");
  return 0;
}
```

zadatak: Burići

Dato je *n* burića. U svakom buretu se nalazi određena količina vode. Perica može da probuši ukupno *m* rupa na dnima burića (*m* > *n*). Kroz svaku probušenu rupu izlazi 1 litar vode u sekundi. Perica sve rupe buši istovremeno i želi da ih probuši tako da što pre ni u jednom buretu ne ostane ni malo vode (tj. da sva voda isteče što pre). Odrediti koliko je minimalno vreme posle bušenje nakon koga ni u jedmom buretu neće više biti vode.

Ulaz:

(Ulazni podaci se nalaze u datoteci **burici.in**) U prvom redu tekstualne datoteke nalaze se redom prirodni brojevi *n* (broj burića, *n* ≤ 50.000) i *m* (broj rupa, m ≤ 400.000). U drugom redu nalazi se *n* prirodnih brojeva (svaki je manji ili jednak 2.000.000.000) tako da *i*-ti (1 ≤ *i* ≤ *n*) broj označava broj litara u *i*-tom buretu.

Izlaz:

(Izlazne podatke upisati u datoteku **burici.out**) U prvom redu tekstualne datoteke ispisati jedan realan broj a to je minimalno vreme koje se traži zaokruženo na dve decimale (priznaje se svako rešenje koje se od zvaničnog rešenja razlikuje po apsolutnoj vrednosti za ne više od 0:01).

Primer:

burici.in	**burici.out**
3 9 6 10 2	2

fajl: burici.cpp,

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


int n,m;
double *L;
double broj;

//prvo resenje ---------------------------------------
//za 100 poena
struct cvor
{
  double lit;
  int br;
};

void zam(cvor *a,cvor *b)
{
  cvor pom=*a;
```

```
  *a=*b;
  *b=pom;
}

void f1()
{
  cvor *A=(cvor *)malloc(n*sizeof(cvor));
  int i;
  for (i=0;i<n;i++)
  {
    A[i].lit=L[i];
    A[i].br=1;
    int j=i;
    while (j>0 && A[(j-1)/2].lit<A[j].lit)
    {
      zam(A+(j-1)/2,A+j);
      j=(j-1)/2;
    }
  }
  for (i=0;i<m-n;i++)
  {
    A[0].br++;
    int j=0,da=1;
    while (da)
    {
      if (j*2+1>=n)
        da=0;
      else
        if (j*2+2==n)
          if (A[j].lit/A[j].br<A[j*2+1].lit/A[j*2+1].br)
          {
            zam(A+j,A+j*2+1);
            j=j*2+1;
          }
          else
            da=0;
        else
          if (A[j*2+1].lit/A[j*2+1].br>A[j*2+2].lit/A[j*2+2].br)
            if (A[j].lit/A[j].br<A[j*2+1].lit/A[j*2+1].br)
            {
              zam(A+j,A+j*2+1);
              j=j*2+1;
            }
            else
              da=0;
          else
            if (A[j].lit/A[j].br<A[j*2+2].lit/A[j*2+2].br)
            {
              zam(A+j,A+j*2+2);
              j=j*2+2;
            }
            else
              da=0;
    }
  }
  broj=A[0].lit/A[0].br;
  free(A);
}
//-------------------------------------------------------

//drugo resenje ------------------------------------------
//za sto poena
void f2()
{
  double g=L[0],d=0;
  int i;
  for (i=1;i<n;i++)
```

```
    if (g<L[i])
      g=L[i];
  while (g-d>0.0001)
  {
    double br1=(g+d)/2;
    int br2=0;
    for (i=0;i<n;i++)
      br2+=ceil(L[i]/br1);
    if (br2<=m)
      g=br1;
    else
      d=br1;
  }
  broj=g;
}

//------------------------------------------------------

void main()
{

  FILE *dat=fopen("Bure.in","r");
  fscanf(dat,"%d%d",&n,&m);
  L=(double *)malloc(n*sizeof(double));
  int i;
  for (i=0;i<n;i++)
    fscanf(dat,"%lf",L+i);
  fclose(dat);

  //f1();
  f2();




  dat=fopen("Bure.out","w");
  fprintf(dat,"%.2lf",broj);
  fclose(dat);


}
```

zadatak: Vojnici

Perica igra jednu igru na svom računaru. On ima n svojih vojnika od kojih svaki ima neku jačinu. Dato je i n protivničkih vojnika od kojih svaki takođe ima neku jačinu. Jačine tih $2n$ vojnika su različite (tj. ne postoje dva vojnika sa jednakim jačinama). Perica treba da uradi sledeću stvar: treba da sastavi n parova vojnika tako da se svaki par sastoji od jednog njegovog i jednog protivničkog vojnika i da se svaki od $2n$ vojnika pojavljuje u tačno jednom paru. I tada kreće bitka. U svakom od n dvoboja (u i-tom dvoboju ($1 \leq i \leq n$) učestvuju vojnici i-tog para) pobeđuje vojnik koji je jači. Za svakog od n protivničkih vojnika data su po dva broja: jedan koji govori koliko Perica dobija poena ukoliko njegov (Peričin) vojnik pobedi tog vojnika i drugi koji govori koliko Perica gubi poena ukoliko njegov vojnik izgubi od tog vojnika. Perica na početku ima 0 poena. Odrediti koliki je maksimalan broj poena koji Perica može skupiti (taj broj može biti i negativan).

Ulaz.

(Ulazni podaci se nalaze u datoteci **vojnici.in**) U prvom redu tekstualne datoteke nalazi se prirodan broj n ($n \leq 2.000$). U drugom redu nalazi se n prirodnih brojeva: i-ti od tih brojeva ($1 \leq i \leq n$) predstavlja jačinu i-tog Peričinog vojnika (svaki od brojeva je manji od ili jednak 2.000.000.000). U trećem redu nalazi se n prirodnih brojeva: i-ti broj u tom redu ($1 \leq i \leq n$) predstavlja jačinu i-tog protivničkog vojnika (svaki od brojeva je manji ili jednak 2.000.000.000). U četvrtom redu nalazi se n prirodnih brojeva: i-ti broj ($1 \leq i \leq n$) predstavlja broj poena koji Perica dobija ukoliko je taj protivnički vojnik poražen (svaki od brojeva je manji od ili jednak 1.000). U petom redu nalazi se n prirodnih brojeva: i-ti broj ($1 \leq i \leq n$) predstavlja

broj poena koji Perica gubi ukoliko je taj protivnički vojnik u dvoboju u kome je učestvovao izašao kao pobednik (svaki od brojeva je manji ili jednak 1.000).
Izlaz.
(Izlazne podatke upisati u datoteku **vojnici.out**) U prvom redu tekstualne datoteke ispisati jedan ceo broj a to je maksimalan broj poena koji Perica može skupiti.
Primer 1.

vojnici.in	**vojnici.out**
3 9 12 3 4 5 6 10 2 7 5 3 1	

fajl: vojnici.cpp

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


int n;
int *A,*B,*D,*G;
int broj;

void zam(int *a,int *b)
{
  int pom=*a;
  *a=*b;
  *b=pom;
}

//prvo resenje ----------------------------------------
void sortA(int *A,int n)
{
  int br=A[n/2];
  int i=0,j=n-1;
  while (i<=j)
  {
    while (A[i]<br)
      i++;
    while (A[j]>br)
      j--;
    if (i<=j)
    {
      zam(A+i,A+j);
      i++;
      j--;
    }
  }
  if (j>0)
    sortA(A,j+1);
  if (n-i>1)
    sortA(A+i,n-i);
}

void sortB(int *B,int *D,int *G,int n)
{
  int br=B[n/2];
  int i=0,j=n-1;
  while (i<=j)
  {
    while (B[i]<br)
      i++;
    while (B[j]>br)
      j--;
```

```
    if (i<=j)
    {
      zam(B+i,B+j);
      zam(G+i,G+j);
      zam(D+i,D+j);
      i++;
      j--;
    }
  }
  if (j>0)
    sortB(B,D,G,j+1);
  if (n-i>1)
    sortB(B+i,D+i,G+i,n-i);
}


void f1()
{
  sortA(A,n);
  sortB(B,D,G,n);

  int *P1=(int *)malloc(n*sizeof(int));
  int *P2=(int *)malloc(n*sizeof(int));
  int i,j;
  for (i=0;i<n;i++)
  {
    int *pom=P2;
    P2=P1;
    P1=pom;
    if (A[i]>B[0])
      P1[0]=D[0];
    else
      P1[0]=-G[0];
    for (j=1;j<=i;j++)
    {
      P1[j]=P1[j-1]-G[j];
      if (A[i]>B[j] && P2[j-1]+D[j]>P1[j])
        P1[j]=P2[j-1]+D[j];
    }
  }
  broj=P1[n-1];
}
//-------------------------------------------------------


//drugo resenje ----------------------------------------
void sortC(int *B,int *D,int *G,int n)
{
  int br=D[n/2]+G[n/2];
  int i=0,j=n-1;
  while (i<=j)
  {
    while (D[i]+G[i]<br)
      i++;
    while (D[j]+G[j]>br)
      j--;
    if (i<=j)
    {
      zam(B+i,B+j);
      zam(G+i,G+j);
      zam(D+i,D+j);
      i++;
      j--;
    }
  }
  if (j>0)
    sortC(B,D,G,j+1);
```

```
  if (n-i>1)
    sortC(B+i,D+i,G+i,n-i);
}

void f2()
{
  sortC(B,D,G,n);
  broj=0;
  int i,j,da,k,m;
  m=n;
  for (i=n-1;i>=0;i--)
  {
    da=0;
    for (j=0;j<m;j++)
      if (A[j]>B[i])
        if (!da)
        {
          da=1;
          k=j;
        }
        else
          if (A[j]<A[k])
            k=j;
    if (da)
    {
      m--;
      zam(A+k,A+m);
      broj+=D[i];
    }
    else
      broj-=G[i];
  }
}
//-------------------------------------------------------



void main()
{
  FILE *dat=fopen("Vojnici.in","r");
  fscanf(dat,"%d",&n);
  int i;
  A=(int *)malloc(n*sizeof(int));
  B=(int *)malloc(n*sizeof(int));
  D=(int *)malloc(n*sizeof(int));
  G=(int *)malloc(n*sizeof(int));
  for (i=0;i<n;i++)
    fscanf(dat,"%d",A+i);
  for (i=0;i<n;i++)
    fscanf(dat,"%d",B+i);
  for (i=0;i<n;i++)
    fscanf(dat,"%d",D+i);
  for (i=0;i<n;i++)
    fscanf(dat,"%d",G+i);
  fclose(dat);

  f1();
  //f2();



  dat=fopen("Vojnici.out","w");
  fprintf(dat,"%d",broj);
  fclose(dat);


}
```