## Zadaci sa rešenjima Okružno takmičenje 2010.

### zadatak: Imena

Grupa studenata psihologije dobila je zadatak da istraži uticaj imena na popularnost učenika u osnovnoj školi. Posmatrajući grupu osnovaca otkrili su nekoliko načina na koji osnovci računaju sličnost sa svojom simpatijom koji se zasnivaju na brojanju slova u svojim i imenima svojih drugara. Takođe su primetili da se imena nekih učenika mnogo češće nalaze u ovim igricama od drugih. Nakon dugotrajne debate odlučili su da je razlog za ovu popularnost određenih učenika raznovrsnost njihovih imena, odnosno broj različitih slova u imenima učenika. Pomozite im da provere ovu pretpostavku tako što ćete napisati program koji računa ove brojeve.

**Ulaz:**

(Ulazni podaci se nalaze u datoteci **imena.in**) U prvom redu nalazi se broj $n$ ($1 \leq n \leq 1000$) i on predstavlja broj učenika. U svakom od sledećih $n$ redova nalaze se ime i prezime po jednog učenika, bez razmaka između imena i prezimena. Dužine ovih nizova znakova neće biti veće od 100.

**Izlaz:**

(Izlazne podatke upisati u datoteku **imena.out**) U jednom redu izlaza ispisati dva cela broja: prvi koji predstavlja najveći broj različitih slova u imenu i prezimenu nekog učenika, drugi koji predstavlja broj učenika čija imena sadrže taj maksimalni broj slova.

**Napomena.**

Malo i veliko slovo 'a' (a slično i ostali parovi veliko i odgovarajuće malo slovo) se smatraju istim slovom. Sva slova će biti slova engleske abecede.

**Primer:**

| **imena.in** | **imena.out** |
|---|---|
| 5<br>PetarPetrovic<br>MilanMilanovic<br>MilanMilankovic<br>JovanPetrovic<br>JovanJovanovic | 11 1 |

**Objašnjenje.**

Ime JovanPetrovic ima najviše različitih slova, 11, dok sva ostala imena imaju manje slova.

### fajl: imena.cpp

```
#include<stdio.h>

int computeDistinct(char *s)
{
  // init array
  int frequencies [26];
  for (int i = 0; i < 26; i++)
  {
    frequencies[i] = 0;
  }
  // process the name
  bool moreLetters = true;
  int i = 0;
  while (moreLetters)
  {
    if ((s[i] >= 'A') && (s[i] <= 'Z'))
    {
      frequencies[s[i]-'A']++;
    }
    else if ((s[i] >= 'a') && (s[i] <= 'z'))
    {
      frequencies[s[i]-'a']++;
    }
    else
    {
      moreLetters = false;
    }
    i++;
```

```
  }
  // count distinct letters
  int distinct = 0;
  for (int i = 0; i < 26; i++)
  {
    distinct += (frequencies[i]>0);
  }
  return distinct;
}

int main()
{
  FILE *fIn;
  fIn = fopen("imena.in", "r");
  // read N
  int iCount;
  fscanf(fIn,"%d", &iCount);
  // init
  int iMaxDistinct = 0;
  int iNumMax = 0;
  for (int i = 0; i < iCount; i++)
  {
    // read and process individual names
    char pcName[101];
    fscanf(fIn,"%s", pcName);
    int iDistinct = computeDistinct(pcName);
    // update accumulated results
    if (iDistinct == iMaxDistinct)
    {
      iNumMax++;
    }
    if (iDistinct > iMaxDistinct)
    {
      iMaxDistinct = iDistinct;
      iNumMax = 1;
    }
  }
  // output results
  FILE *fOut;
  fOut = fopen("imena.out", "w");
  fprintf(fOut, "%d %d\n", iMaxDistinct, iNumMax);
  fclose(fIn);
  fclose(fOut);
  return 0;
}
```

## zadatak: Puberaks

Proka Pronalazač dugo je živeo sam, ali se ovih dana konačno oženio! Odjednom mu je proradio puberaks, te je sve obavio na brzinu, i još brže dobio *n*-oro divne dečice. Proka smatra da je pravljenje razlike među decom najbolji način za njihovo odgajanje (prema njegovom misljenju, to podstiče zdravu konkurenciju). Jasno nam je da se većina roditelja neće složiti s njim, ali ne treba mu zameriti, ipak je on već zašao u godine a ovo je za njega sasvim novo iskustvo. Evo, baš danas je prodao najnoviji pronalazak i odmah je izdvojio deo zarađenog novca koji će dati deci za džeparac. No, kada je izbrojao izdvojene novčanice, primetio je da ih ima upravo *n* (dakle, isto koliko i dece), kao i da su sve međusobno različitih vrednosti. Tada mu je pala na pamet sledeća ideja: odlučio je da svakom detetu taman da po jednu; štaviše, da bi napravio još veću razliku među decom, odlučio je da ih podeli u dve grupe takve da se prosek džeparca u jednoj grupi što više razlikuje od proseka džeparca u drugoj grupi. Upravo ovde vi uskačete da pomognete Proki.

**Ulaz:**

(Ulazni podaci se učitavaju iz datoteke **puberaks.in**.) U prvom redu ulazne datoteke nalazi se broj Prokine dece, što je ujedno i broj novčanica predviđenih za džeparac, $n$ ($1 \leq n \leq 10.000$ - Proka baš nije gubio vreme!). U idućem redu nalazi se $n$ prirodnih brojeva u rasponu od 1 do 50.000.000, razdvojenih prazninom: to su vrednosti novčanica koje je Proka izdvojio za džeparac.

**Izlaz:**
(Izlazni podaci se ispisuju u datoteku **puberaks.out**.) U prvi i jedini red izlazne datoteke upisati razliku između proseka džeparca u prvoj i drugoj grupi, zaokruženu na dve decimale.
**Primer:**

| **puberaks.in** | **puberaks.out** |
|---|---|
| 5<br>20 50 10 60 70 | 45.00 |

**Objašnjenje.**
Deca su podeljena u dve grupe na sledeći način: dvoje dece čine jednu grupu, troje dece čine drugu. Dvoje dece iz prve grupe dobijaju novčanice u vrednosti od 10 i 20 novčanih jedinica, dok deca iz druge grupe dobijaju tri preostale novčanice. Tada je prosek džeparca u prvoj grupi jednak 15, dok je prosek džeparca u drugoj grupi jednak 60, te razlika između ova dva proseka iznosi 45. Može se proveriti da se nikakvom drugačijom podelom ne može postići veća razlika.
**Napomena.**
U svakoj grupi mora se nalaziti bar po jedno dete (tj., grupa se ne može sastojati od 0 dece).

**fajl: puberaks.pas**

```
var n,i,j,s,s1,pom:longint;
    pr:double;
    a:array[1..10000] of longint;
    f:text;
begin
 assign(f,'puberaks.in');
 reset(f);
 readln(f,n);
 for i:=1 to n do
   read(f,a[i]);
 close(f);
 for i:=1 to n do
   for j:=i+1 to n do
     if a[i]>a[j] then begin
                        pom:=a[i];
                        a[i]:=a[j];
                        a[j]:=pom;
                       end;
 s:=0;
 s1:=0;
 for i:=1 to n do
   s1:=s1+a[i];
 pr:=0;
 for i:=1 to n-1 do
   begin
    s:=s+a[i];
    s1:=s1-a[i];
    if s1/(n-i)-s/i>pr then pr:=s1/(n-i)-s/i;
   end;
 assign(f,'puberaks.out');
 rewrite(f);
 writeln(f,pr:0:2);
 close(f);
end.
```

**zadatak: Palindromski niz**

Niz brojeva je palindromski, ako se čita isto i sa leve i sa desne strane. Na primer, nizovi {1, 5, 1} i {10, 9, 9, 10} su palindromski, dok {1, 2, 3, 1} i {20, 2} nisu palindromski.
Na početku je dat niz prirodnih brojeva *a* dužine *n*. U jednom koraku dozvoljeno je zameniti dva susedna broja njihovom sumom (obrisati dva susedna broja *a*[i] i *a*[i + 1] i na njihovom mestu upisati *a*[i] + *a*[i + 1]). Odrediti najveću moguću dužinu palindromskog niza, koji se može dobiti primenom ove operacije proizvoljan broj puta.
**Ulaz:**

(Ulazni podaci se nalaze u datoteci **pniz.in**) U prvom redu se nalazi prirodan broj $n$ ($1 \leq n \leq 100.000$). U drugom redu se nalaze $n$ prirodnih brojeva $a$[i] ($1 \leq a$[i] $\leq 10.000$), koji predstavljaju elemente niza $a$.
**Izlaz:**
(Izlazne podatke upisati u datoteku **pniz.out**) U prvom i jedinom redu ispisati jedan prirodan broj koji predstavlja najveću dužinu palindromskog niza koji se može dobiti od polaznog niza.
**Primer:**

| **imena.in** | **imena.out** |
|---|---|
| 6<br>10 10 10 20 20 30 | 4 |

**Objašnjenje.**
Zamenimo prva dva broja i dobijamo niz {20, 10, 20, 20, 30}. Zatim menjamo opet prva dva broja i dobijamo palindromski niz {30, 20, 20, 30} dužine četiri.
**Primer:**

| **imena.in** | **imena.out** |
|---|---|
| 5<br>1 2 3 4 5 | 1 |

**Objašnjenje.**
Jedini palindromski niz koji možemo dobiti je {15}.

**fajl: pniz.cpp**

```
#include<stdio.h>

#define MAXN 100001

int n, sol;
int a [MAXN];

int mmain ()
{
  FILE *in = fopen ("palindrom.in", "r");
  fscanf (in, "%d", &n);
  for (int i = 0; i < n; i++)
    fscanf (in, "%d", &a [i]);
  fclose (in);

  int left = 0;
  int right = n - 1;
  while (left < right)
  {
    if (a [left] < a [right])
    {
      a [left + 1] += a [left];
      left++;
    }
    else if (a [left] > a [right])
    {
      a [right - 1] += a [right];
      right--;
    }
    else
    {
      sol += 2;
      left++;
      right--;
    }
  }
  if (left == right)
    sol++;

  FILE *out = fopen ("palindrom.out", "w");
  fprintf (out, "%ld\n", sol);
  fclose (out);
```

```
  return 0;
}
```

## zadatak: Robotić

Robotić *WALL-E* se nalazi na deponiji smeća koja je prikazana u obliku matrice dimenzije *n* x *m*. Kako je jedna od karakteristika deponija neurednost, neka polja su nepodesna za kretanje, tako da *WALL-E* ne može prelaziti preko njih. *WALL-E*-u je dosadno pa je rešio da se malo poigra. Naime, zadaje sebi niz *d* prirodnih brojeva dužine *k* koji označavaju broj koraka. Robotić može sa polja u matrici da se kreće u četiri pravca: gore, dole, levo i desno. U *i*-tom trenutku *WALL-E* mora da napravi *d[i]* koraka u jednom od četiri pravaca. Naravno, u toku tih *d[i]* koraka *WALL-E* ne sme preći preko polja na kojima je zabranjeno kretanje. *WALL-E*-ja zanima na koliko različitih polja može biti u *k*-tom trenutku ukoliko kretanje započinje sa startnog polja (*startX, startY*).

**Ulaz:**

(Ulazni podaci se učitavaju iz datoteke **robotic.in**.) U prvom redu nalaze se četiri prirodna broja *n*, *m*, *startX* i *startY* ($1 \leq n,m \leq 200$, $1 \leq startX \leq$ n, $1 \leq startY \leq$ m) koji predstavljaju dimenzije matrice i koordinate startnog polja. Narednih *n* redova sadrže po *m* brojeva 1 ili 0, koji opisuju polja matrice: 0 označava da se preko polja može preći, a 1 označava polje na kome je zabranjeno kretanje. U (*n* + 2)-om redu se nalazi prirodni broj *k* ($1 \leq k \leq 200$) koji označava dužinu niza *d*. Naredni red sadrži *k* prirodnih brojeva odvojenih po jednim znakom razmaka koji označavaju elemente niza *d*.

**Izlaz:**

(Izlazni podaci se ispisuju u datoteku **robotic.out**.) U prvom i jedinom redu ispisati jedan prirodan broj koji predstavlja broj različitih polja na kojima *WALL-E* može da bude u *k*-tom trenutku.

**Primer:**

| **robotic.in** | **robotic.out** |
|---|---|
| 3 3 1 1<br>0 1 0<br>0 0 0<br>0 0 0<br>3<br>1 2 1 | 3 |

**Objašnjenje.**

*WALL-E* može završiti na poljima sa koordinatama (1, 3), (2, 2) i (3, 3). Navedene krajnje pozicije kao i putanje kojima dolazi do njih su prikazani na slici.

## fajl: robotic.cpp

```
/*
  Author: Andreja Ilic, PMF Nis
  e-mail: ilic_andrejko@yahoo.com

  Algorithm:
  Zadatak resavamo BFS pretragom. Za svaki trenutak pamtimo listu
  moguci pozicija, na osnovu kojih racunamo pozicije za naredni
  trenutak. Ispitivanje moguceg poteza sa polja (x, y) na polje
  (x1, y1) svodimo na ispitivanje da li je suma elemenata u toj
  podmatrici 0 ili ne. Ovo mozemo  odraditi pomocu predprocesiranja
  u O(1) (pamcenjem prefiksnih suma sa redove i kolone).

  Complexity: O(n^2 * k)
*/


#include<stdio.h>
#include<iostream>
#include<vector>
using namespace std;

#define MAX_N 205

struct cell
{
```

```
  int x, y;
} Cell;

int n, m, startX, startY, sol = 0, k;
int board [MAX_N][MAX_N], d [MAX_N], prefixRow [MAX_N][MAX_N], prefixColumn [MAX_N][MAX_N];
int dx [] = {0, 1, 0, -1};
int dy [] = {1, 0, -1, 0};

  // Unos podataka
  void input()
  {
    FILE *in = fopen ("robotic.in", "r");
    fscanf (in, "%d %d %d %d", &n, &m, &startX, &startY);
    for (int i = 1; i <= n; i++)
      for (int j = 1; j <= m; j++)
        fscanf(in, "%d", &board[i][j]);
    fscanf(in, "%d", &k);
    for (int i = 0; i < k; i++)
      fscanf (in, "%d", &d [i]);
    fclose(in);
  }

  // Ispis resenja
  void output()
  {
    FILE *out = fopen("robotic.sol", "w");
    fprintf (out, "%d\n", sol);
    fclose(out);
  }

  // Inicijalizacija prefiksnih suma za redove i kolone
  void intializationOfPrefixSums()
  {
    for (int i = 1; i <= n; i++)
    {
      prefixRow [i][0] = 0;
      for (int j = 1; j <= m; j++)
        prefixRow [i][j] = prefixRow [i][j - 1] + board [i][j];
    }
    for (int j = 1; j <= m; j++)
    {
      prefixColumn [j][0] = 0;
      for (int i = 1; i <= n; i++)
        prefixColumn [j][i] = prefixColumn [j][i - 1] + board [i][j];
    }
  }

  // Funkcija minimuma
  int min(int a, int b)
  {
    if (a < b)
      return a;
    return b;
  }

  // Funkcija maxkimuma
  int max(int a, int b)
  {
    if (a > b)
      return a;
    return b;
  }

  // Suma elemenata u matrici od polja (a,b) do (c,d)
  int sum (int a, int b, int c, int d)
  {
    if (a == c)
```

```
    return prefixRow [a][max(b, d)] - prefixRow [a][min(b, d) - 1];
  return prefixColumn [b][max(a, c)] - prefixColumn[b][min(a, c) - 1];
}

// Glavna metoda
void solve()
{
  intializationOfPrefixSums();
  cell q [2][MAX_N * MAX_N];
  int num [2];

  q [0][0].x = startX; q [0][0].y = startY;
  num [0] = 1;

  bool mark [MAX_N][MAX_N];
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= m; j++)
      mark [i][j] = false;

  for (int index = 0; index < k; index++)
  {
    int  nextIndex = (index + 1) % 2;
    for (int i = 0; i < num[index % 2]; i++)
      mark [q [(index % 2)][i].x][q [(index % 2)][i].y] = false;

    int t = 0;
    num [nextIndex] = 0;
    while (t < num [index % 2])
    {
      cell currentCell = q [index % 2][t];
      int x = currentCell.x, y = currentCell.y;

      for (int i = 0; i < 4; i++)
      {
        int x1 = x + dx [i] * d [index];
        int y1 = y + dy [i] * d [index];
        if ((1 <= x1) && (x1 <= n) && (1 <= y1) && (y1 <= m))
          if ((! mark [x1][y1]) && (sum (x, y, x1, y1) == 0))
          {
            q [nextIndex][num [nextIndex]].x = x1;
            q [nextIndex][num [nextIndex]].y = y1;
            num[nextIndex]++;
            mark [x1][y1] = true;
          }
      }
      t++;
    }
  }

  sol = num [k % 2];
}

int main()
{
  input();
  solve();
  output();

  return 0;
}
```

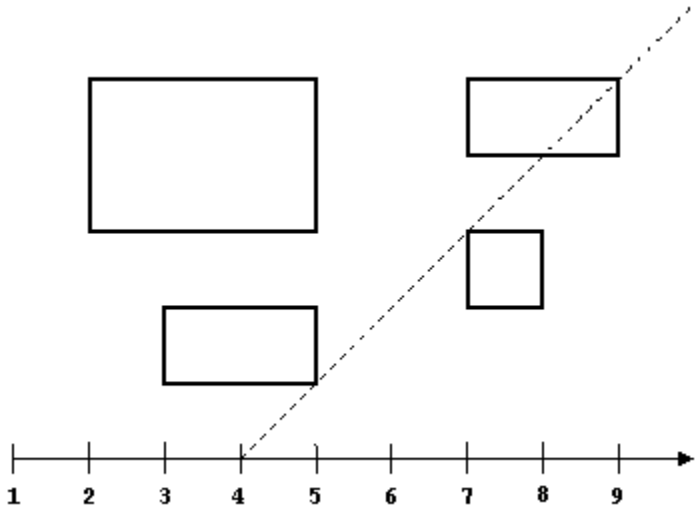
## zadatak: Poljane

Đurica putuje vozom kroz Vojvodinu i uživa razgledajući predivne poljane. Tokom putovanja on često (i ovaj put) fiksira položaj glave i razgleda, ne bi li bio siguran da neće propustiti ni jedan metar prelepog prizora,

kao i da će svakom pokloniti isto vremena. Iz tih razloga, položaj njegove glave ostaje fiksiran tokom putovanja. Poljane su oblika pravougaonika, sa stranicama paralelnim ili normalnim na prugu.
Radi lakšeg opisa, pretpostavićemo da se pruga nalazi na *x*-osi, a poljane u prvom kvadrantu koordinatnog sistema. Svaka poljana je predstavljena svojim donjim-levim i gornjim-desnim temenom. Đuricin pogled ćemo predstaviti polupravom koja menja svoju početnu tačku duž *x*-ose u pozitivnom smeru. Nagib poluprave na *x*-osu je izražen u stepenima, a ima vrednost *alfa*.
Nakon svog putovanja, Đurica se zamislio i zapitao koji je najveći broj poljana koje je on u nekom momentu posmatrao, tj. presekao pogledom. Pošto ćete vi dobiti opis poljana i ugao nagiba *alfa*, pomozite Đurici i izračunajte koji je najveći broj poljana u nekom momentu koje je Đurica presekao pogledom.
**Ulaz:**
(Ulazni podaci se nalaze u datoteci **poljane.in**) U prvom redu nalaze se prirodni brojevi $n$ ($1 \le n \le 100.000$) i *alfa* ($1 \le alfa \le 90$), koji predstav aju broj po ana i ugao nagiba Đuricinog pogleda u stepenima, redom. U narednih $n$ redova su data po četiri prirodna broja $x_1$, $y_1$, $x_2$, $y_2$ ($1 \le x_1 < x_2 \le 100.000$, $1 \le y_1 < y_2 \le 100.000$) koji označavaju donje-levo i gornje-desno teme poljana, redom - u $i$-tom redu koordinate za $i$-tu poljanu.
**Izlaz:**
(Izlazne podatke upisati u datoteku **poljane.out**) U prvom i jedinom redu ispisati jedan ceo broj koji predstavlja najveći broj poljana koje je Đurica u nekom momentu video.
**Primer:**

| poljane.in | poljane.out |
|---|---|
| 4 45<br>3 1 5 2<br>7 2 8 3<br>7 4 9 5<br>2 3 5 5 | 3 |

**Objašnjenje.**




Kada se Đurica bude nalazio na koordinati 4 *x*-ose, videće prve tri poljane date na ulazu. Poljani pripada i rub pravougaonika koji je ograničava. Dovoljno je posmatrati samo jednu tačku, na primer teme, poljane da se smatralo da pogled preseca poljanu.

**fajl: poljane.cpp**

```
#include <iostream>
#include <cstdio>
#include <fstream>
#include <vector>
#include <cmath>
#define ffor(_a,_f,_t) for(int _a=(_f),__t=(_t);_a<__t;_a++)
#define all(_v) (_v).begin() , (_v).end()
#define sz size()
#define pb push_back
#define SET(__set, val) memset(__set, val, sizeof(__set))
```

```
#define FOR(__i, __n) ffor (__i, 0, __n)

using namespace std;

const int MAXN = 100000;
const double PI2 = acos(0.0);

int xx1, yy1, xx2, yy2, alpha;

struct myt{
  double pos;
  bool start;

  myt(){
  }

  myt(double _pos, bool _start){
    pos = _pos;
    start = _start;
  }

  friend bool operator <(const myt &a, const myt &b){
    if (fabs(a.pos - b.pos) > 1e-9)
      return a.pos < b.pos;

    return a.start && !b.start;
  }
};

myt points[MAXN << 1];

/*
  Za datu tacku (a, b), zelimo da za pravu
  sa nagibom alpha koja sadrzi tacku (a, b)
  nadjemo mesto presecanja na x-osi.
*/
double calc(double a, double b){
  if (alpha == 90)
    return a;

  // 90 : PI2 = alpha : rad - pretvaramo u radijane
  double rad = PI2 * alpha / 90.0;

  // b / c = tg alpha; b i c su stranice
  // pravouglog trougla kojeg posmatramo,
  // tj. onog koji ima jedan ugao alpha,
  // ugao naspram stranice a. Resenje ce
  // biti a - c
  double c = b / tan(rad);
  return a - c;
}

int main(){
  char filein[16] = "poljane.in";
  char fileout[16] = "poljane.sol";
  FILE *fin;
  FILE *fout;


  fin = fopen(filein, "r");
  fout = fopen(fileout, "w");

  int n;
  fscanf(fin, "%d %d", &n, &alpha);
  FOR (i, n){
    fscanf(fin, "%d %d %d %d", &xx1, &yy2, &xx2, &yy1);
    points[i << 1] = myt(calc(xx1, yy1), true);
```

```
    points[(i << 1) + 1] = myt(calc(xx2, yy2), false);
  }
  sort(points, points + (n << 1));

  int ret = 0, cnt = 0;
  FOR (i, n << 1){
    if (points[i].start)
      cnt++;
    else
      cnt--;

    ret >?= cnt;
  }

  fprintf(fout, "%d\n", ret);

  fclose(fin);
  fclose(fout);

  return 0;
}
```