Zadaci sa rešenjima SIO 2009.

zadatak: Igrica

U gradiću u kom žive Kići i Ćiki postoji jedna nagradna igrica. U igrici ima *m* učesnika. Svaki učesnik izabere neko od prvih *n* slova engleske abecede. Onda se od tih slova napravi jedna reč - slovo koje je izabrao prvi učesnik je prvo slovo te reči, itd. Na kraju se prebroji koliko puta se svako od slova pojavljuje u toj reči. Svi oni koji su izabrali slovo koje se pojavljuje neparan broj puta dobijaju brodić.
Učesnici naravno ne znaju koja će slova ostali birati, ali pošto su Kići i Ćiki jako simpatični oni su uspeli da od svakog učesnika saznaju po nekoliko slova koja on sigurno neće izabrati.
I sada oni imaju zadatak za vas: za svako od prvih *n* slova oni će vam reći da li žele da se ono pojavi paran broj puta, neparan broj puta ili im je svejedno. Vi treba da im kažete koliko od reči koje se mogu dobiti u ovoj igrici zadovoljavaju te njihove uslove (pretpostavljamo da niko nije slagao Kićija i Ćikija). Pošto broj reči može biti velik, dovoljno je da im kažete ostatak pri deljenju tog broja sa 10.007.

Ulaz:
(Ulazni podaci se nalaze u datoteci **igrica.in**) U prvom redu ulazne datoteke nalaze se dva broja, *n* i *m* (broj slova iz alfabeta koja mogu biti korišćena i broj učesnika u igri). Sledi *m* redova sa po *n* brojeva u svakom od njih - u *i*-tom od tih redova *j*-ti broj je 0 ako *i*-ti učesnik sigurno neće izabrati *j*-to slovo, odnosno 1 ako možda hoće. U poslednjem redu datoteke nalazi se još *n* brojeva - *j*-ti od njih je 0 ako Kići i ćiki žele da se *j*-to slovo pojavi paran broj puta, 1 ako žele da se pojavi neparan broj puta, odnosno -1 ako im je svejedno.

Izlaz:
(Izlazne podatke upisati u datoteku **igrica.out**) U izlaznoj datoteci treba da se nalazi jedan broj - broj reči koje zadovoljavaju Kićijeve i ćikijeve uslove po modulu 10.007.

Ograničenja:
Neka je k_i broj jedinica u redu datoteke koji odgovara *i*-tom učesniku.

- u 20% test primera: $n \leq 10$, $m \leq 6$, $1 \leq k_i \leq n$
- u 20% test primera: $n \leq 15$, $m \leq 12$, $1 \leq k_i \leq 3$
- u ostalim primerima: $n \leq 15$, $m \leq 50$, $1 \leq k_i \leq n$

Primer 1:

igrica.in	**igrica.out**
5 4 1 0 0 0 1 1 1 1 0 0 1 0 1 0 0 0 0 1 0 0 1 -1 0 -1 -1	3

Objašnjenje.
Prvi učesnik će izabrati neko od slova {A,E}, drugi {A,B,C}, treći {A,C} i četvrti {C}. Traži nam se broj reči u kojima se A pojavljuje neparan a C paran broj puta. Moguće su sledeće reči:

AAAC, AACC, ABAC, ABCC, ACAC, ACCC, EAAC, EACC, EBAC, EBCC, ECAC, ECCC.

Među njima reči koje zadovoljavaju uslov su ABCC, EACC, ECAC - ima ih 3.

fajl: igrica.cpp

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

const int mod = 10007;
const int maxN = 15;
const int maxM = 50;
const int maxDvaNaN = 40000;

int n, m, dvaNaN;
int k[maxM];
int moguceSlovo[maxM][maxN];
int potrebno[maxN];
int koliko[2][maxDvaNaN];
```

```
int main() {
    freopen("igrica.in", "r", stdin);
    scanf("%d%d", &n, &m);
    for (int i = 0; i < m; i++) {
        k[i] = 0;
        for (int j = 0; j < n; j++) {
            int pom;
            scanf("%d", &pom);
            if (pom == 1) {
               moguceSlovo[i][k[i]] = j;
               k[i]++;
            }
        }
    }
    for (int i = 0; i < n; i++)
        scanf("%d", &(potrebno[i]));

    dvaNaN = 1 << n;
    memset(koliko, 0, sizeof(koliko));
    koliko[0][0] = 1;
    int trenutni = 0, prethodni = 1;
    for (int i = 0; i < m; i++) {
        trenutni = 1 - trenutni;
        prethodni = 1 - prethodni;
        for (int j = 0; j< dvaNaN; j++)
            koliko[trenutni][j] = 0;
        for (int j = 0; j < k[i]; j++)
            for (int staraMaska = 0; staraMaska < dvaNaN; staraMaska++) {
                int novaMaska = 1 << moguceSlovo[i][j];
                if ((novaMaska & staraMaska) == 0)
                   novaMaska = staraMaska | novaMaska;
                else novaMaska = staraMaska & (~novaMaska);
                koliko[trenutni][novaMaska] += koliko[prethodni][staraMaska];
                if (koliko[trenutni][novaMaska] >= mod)
                   koliko[trenutni][novaMaska] -= mod;
            }
    }

    int res = 0;
    for (int tek = 0; tek < dvaNaN; tek++) {
        bool zadovoljava = true;
        for (int i = 0; i < n; i++) {
            if ((potrebno[i] == 0) && ((tek & (1 << i)) != 0))
               zadovoljava = false;
            if ((potrebno[i] == 1) && ((tek & (1 << i)) == 0))
               zadovoljava = false;
        }
        if (zadovoljava) {
           res += koliko[trenutni][tek];
           if (res >= mod) res -= mod;
        }
    }

    freopen("igrica.out", "w", stdout);
    printf("%d\n", res);
    return 0;
}
```

zadatak: Oglasna tabla

Tojin Vomić je veoma savestan student. Kako bi što bolje ispratio svoje (i tuđe) napredovanje tokom studija, mnogo vremena provodi pred oglasnom tablom analizirajući tamo prikazane bodove. Nažalost, Tojin je ovog semestra malo popustio s učenjem, pa mu rezultati na oglasnoj tabli baš i ne idu u korist, te je razmišljao kako ne bi bilo loše da se neki od njih sakriju od očiju

javnosti. I baš u tom momentu, dr Grupović je, noseći cedulju na kojoj su rezultati s njegovog poslednjeg kolokvijuma, zamolio Tojina da dotičnu okači na oglasnu tablu. (Sve cedulje na oglasnoj tabli, kao i cedulja profesora Grupovića, okruglog su oblika.)
Tojin je brzo razmišljao: okačiće cedulju tako da pokrije što više već objavljenih rezultata - a što, budući da je izlaznost na kolokvijume prof. Grupovića uvek bila veoma visoka, neće biti teško. No, kako je Tojin ipak student matematike a ne informatike, zamolio je vas da mu pomognete.

Ulaz:

(Ulazni podaci se nalaze u datoteci **tabla.in**) U prvom redu ulazne datoteke nalazi se prirodan broj n ($1 \leq n \leq 100$), koji predstavlja broj rezultata objavljenih na oglasnoj tabli. U svakom od narednih n redova nalaze se po tri realna broja, pri čemu se u i-tom od tih redova nalaze koordinate centra i-te cedulje, kao i njen poluprečnik. Najzad, u poslednjem redu nalazi se još jedan realan broj, koji predstavlja poluprečnik Tojinove/Grupovićeve cedulje. Koordinate su realni brojevi iz intervala [-1000, 1000]. Poluprečnici su realni brojevi iz intervala [1, 1000]. Svi realni brojevi su dati na najviše 2 decimale.

Izlaz:

(Izlazne podatke upisati u datoteku **tabla.out**) U prvi i jedini red izlazne datoteke ispisati jedan ceo broj: najveći mogući broj objavljenih rezultata koje Tojin može da pokrije zabadanjem cedulje. Cedulja se smatra pokrivenom samo ako je u potpunosti skrivena od očiju javnosti, tj. ukoliko nijedan delić ne izviruje.

Primer 1:

tabla.in **tabla.out**

```
3
0 1 1
1 0 1.41
0 -1 1
1.92
```

fajl: tabla.cpp

```
/*
  Author: Slobodan Mitrovic
  e-mail: boba5555@gmail.com
  date: 03.06.2009.
*/
#include <iostream>
#include <cstdio>
#include <cmath>
#include <algorithm>
#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)
#define sqr(a) ((a) * (a))

using namespace std;

const double EPS = 1e-10;

double a[102], b[102], r[102], R;
int n;

struct tDot{
  double x, y;
```

```
  bool ex;
  tDot(){
  }

  tDot(double _x, double _y, bool _ex){
    x = _x;
    y = _y;
    ex = _ex;
  }
};

tDot ret1, ret2, ret3, ret4;

void findIntersect(int idx1, int idx2){
  ret1.ex = ret2.ex = ret3.ex = ret4.ex = false;
  double va = -a[idx1], vb = -b[idx1];

  double c = sqr(a[idx2] + va) + sqr(r[idx1]) + sqr(b[idx2] + vb) - sqr(r[idx2]);

  double A = 4.0 * sqr(a[idx2] + va) + 4.0 * sqr(b[idx2] + vb);
  double B = -4.0 * (a[idx2] + va) * c;
  double C = sqr(c) - 4.0 * sqr(b[idx2] + vb) * sqr(r[idx1]);
  double D = sqr(B) - 4.0 * A * C;

  if (fabs(A) < EPS || (fabs(D) > EPS && D < 0.0))
    return;


  double x1 = (-B + sqrt(D)) / (2.0 * A);
  double x2 = (-B - sqrt(D)) / (2.0 * A);

  double y1, y2;
  if (sqr(r[idx1]) >= sqr(x1)){
    y1 = sqrt(sqr(r[idx1]) - sqr(x1));

    if (sqr(r[idx2]) - sqr(x1 - (a[idx2] + va)) - sqr(y1 - (b[idx2] + vb)) <= EPS){
      ret1.ex = true;
      ret1.x = x1 - va;
      ret1.y = y1 - vb;
    }
    y1 = -y1;

    if (sqr(r[idx2]) - sqr(x1 - (a[idx2] + va)) - sqr(y1 - (b[idx2] + vb)) <= EPS){
      ret3.ex = true;
      ret3.x = x1 - va;
      ret3.y = y1 - vb;
    }
  }

  if (sqr(r[idx1]) >= sqr(x2)){
    y2 = sqrt(sqr(r[idx1]) - sqr(x2));

    if (sqr(r[idx2]) - sqr(x2 - (a[idx2] + va)) - sqr(y2 - (b[idx2] + vb)) <= EPS){
      ret2.ex = true;
      ret2.x = x2 - va;
      ret2.y = y2 - vb;
    }

    y2 = -y2;

    if (sqr(r[idx2]) - sqr(x2 - (a[idx2] + va)) - sqr(y2 - (b[idx2] + vb)) <= EPS){
      ret4.ex = true;
      ret4.x = x2 - va;
      ret4.y = y2 - vb;
    }

  }
```

```
}

double dist(double x1, double y1, double x2, double y2){
  return sqrt(sqr(x1 - x2) + sqr(y1 - y2));
}

int cnt(tDot dot){
  int ret = 0;
  FOR (i, n)
    if (dist(dot.x, dot.y, a[i], b[i]) + r[i] < R + EPS)
      ret++;
  return ret;
}

int ret = 0;

void updateRet(tDot dot){
  if (!dot.ex)
    return;

  ret >?= cnt(dot);
}

int main(){
  freopen("tabla.in", "rt", stdin);
  freopen("tabla.out", "wt", stdout);
  scanf("%d", &n);
  FOR (i, n)
   scanf("%lf %lf %lf", &a[i], &b[i], &r[i]);

  scanf("%lf", &R);
  FOR (i, n){
    if (r[i] <= R)
      updateRet(tDot(a[i], b[i], true));

    if (r[i] >= R)
      continue;

    ffor (j, i + 1, n){
      if (r[j] >= R)
        continue;

      a[100] = a[i];
      b[100] = b[i];
      r[100] = R - r[i];

      a[101] = a[j];
      b[101] = b[j];
      r[101] = R - r[j];

      findIntersect(100, 101);
      updateRet(ret1);
      updateRet(ret2);
      updateRet(ret3);
      updateRet(ret4);

    }
  }

  printf("%d \n", ret);
  return 0;
}
```

fajl: tabla.pas

```
var n,i,j,maxpokriva,pokr:longint;
    x,y,r:array[1..100] of real;
    rt,maxx,maxy,veliki_ruzni_izraz,a,b,c,x1,y1,x2,y2:real;
    f:text;
    nnnn:string;
function koliko_pokriva(x1,y1:real):longint;
var k, res:longint;
begin
 res:=0;
 for k:=1 to n do
   if sqr(rt-r[k])-sqr(x1-x[k])-sqr(y1-y[k])>=-0.000001 then
     begin
     inc(res);
     end;
    koliko_pokriva:= res;
end;
procedure proveri_pokrivanje;
begin
 pokr:=koliko_pokriva(x1,y1);
 if pokr>maxpokriva then begin
                          maxpokriva:=pokr;
                          maxx:=x1;
                          maxy:=y1;
                         end;
 pokr:=koliko_pokriva(x2,y2);
 if pokr>maxpokriva then begin
                          maxpokriva:=pokr;
                          maxx:=x2;
                          maxy:=y2;
                         end;
end;
begin
 assign(f,'tabla.in');
 reset(f);
 readln(f,n);
 for i:=1 to n do
   readln(f,x[i],y[i],r[i]);
 readln(f,rt);
 close(f);
 i:=1;
 while (i<=n) do
    if r[i]>rt then begin
                     x[i]:=x[n];
                     y[i]:=y[n];
                     r[i]:=r[n];
                     dec(n);
                    end
               else inc(i);
 maxpokriva:=0;
 for i:=1 to n do
   begin
    pokr:=koliko_pokriva(x[i],y[i]);
    if pokr>maxpokriva then begin
                             maxpokriva:=pokr;
                             maxx:=x[i];
                             maxy:=y[i];
                            end;
   end;
 for i:=1 to n-1 do
   for j:=i+1 to n do
     begin
      veliki_ruzni_izraz:=sqr(rt-r[j])-sqr(rt-r[i])+sqr(x[i])+sqr(y[i])-sqr(x[j])-
sqr(y[j]);
      if y[i]<>y[j] then begin
                          a:=1+sqr((x[i]-x[j])/(y[i]-y[j]));
                          b:=-2*x[i]-2*(x[i]-x[j])/(y[i]-
y[j])*(veliki_ruzni_izraz/(2*(y[i]-y[j]))-y[i]);
```

```
                          c:=sqr(x[i])+sqr(y[i])+sqr(veliki_ruzni_izraz/(2*(y[i]-y[j])))-
veliki_ruzni_izraz/(y[i]-y[j])*y[i]-sqr(rt-r[i]);
                          if sqr(b)-4*a*c>=0 then begin
                                                   x1:=(-b+sqrt(sqr(b)-4*a*c))/(2*a);
                                                   x2:=(-b-sqrt(sqr(b)-4*a*c))/(2*a);
                                                   y1:=veliki_ruzni_izraz/(2*(y[i]-y[j]))-
x1*(x[i]-x[j])/(y[i]-y[j]);
                                                   y2:=veliki_ruzni_izraz/(2*(y[i]-y[j]))-
x2*(x[i]-x[j])/(y[i]-y[j]);
                                                   proveri_pokrivanje;
                                                  end
                               end
                     else if x[i]<>x[j] then begin
                                              x1:=veliki_ruzni_izraz/(2*(x[i]-x[j]));
                                              x2:=x1;
                                              if sqr(rt-r[i])-sqr(x1-x[i])>=0 then begin

y1:=sqrt(sqr(rt-r[i])-sqr(x1-x[i]))+y[i];
                                                                                   y2:=-
sqrt(sqr(rt-r[i])-sqr(x1-x[i]))+y[i];

proveri_pokrivanje;
                                                                                 end
                                            end
     end;
 assign(f,'tabla.out');
 rewrite(f);
 writeln(f,maxpokriva:0);
 close(f);
end.
```

zadatak: Baloni

Mali Đura je dobio sledeći zadatak. Njemu se donose prazni baloni i baloni sa vodom. Kad mu se donese prazan balon, postavlja se pitanje iz koliko najviše donetih balona sa vodom do sada može da istrese svu vodu u upravo doneti prazan balon. Da bi opisali zadatak koji je postavljen malom Đuri, definisaćemo dve operacije

- **N** *v* - donet je nov balon u kome se nalazi *v* litara vode
- **Q** *v* - donet je nov balon zapremine *v* litara, ali prazan. U tom momentu, mali Đura treba da odgovori iz koliko **najviše** do sada donetih balona može da se prespe voda u trenutno donešeni

Napomena. Kada se donese prazan balon, baloni iz kojih može da se prespe voda se ne prazne, samo se daje odgovor koliko najviše može da se isprazni.
Ulaz:
(Ulazni podaci se nalaze u datoteci **baloni.in**) U prvom redu se nalazi prirodan broj M ($1 \leq M \leq 100.000$) koji predstavlja broj operacija. U svakom od narednih M redova se nalaze operacije oblika C v, gde C {**N**, **Q**}. Ako C = **N**, tada $1 \leq v \leq 200.000$. Ako C = **Q**, tada $1 \leq v \leq 100.000.000$.
Izlaz:
(Izlazne podatke upisati u datoteku **baloni.out**) Za svaku operaciju oblika **Q** *v*, redom, u jedan red ispisati koliko najviše do sada donešenih balona sa vodom može da se isprazni u dati balon.
Primer 1:

baloni.in	**baloni.out**
9	0
Q 10	1
N 3	2
Q 10	2
N 7	3
Q 10	
N 2	
Q 10	

```
N 3
Q 10
```

Objašnjenje.
Nakon donošenja prvog praznog balona, ne postoji ni jedan donet balon sa vodom, te nije moguće ni jedan presuti u isti. Nakon donošenja poslednjeg praznog balona u njega je moguće presuti vodu iz balona zapremina 3, 2 i 3, što je i najviše u ovom slučaju.

Primer 2:

baloni.in	**baloni.out**
10 N 5 N 8 N 1 Q 20 N 11 Q 8 N 1 N 2 Q 31 N 12	3 2 6

fajl: baloni.cpp

```
/*
  Author: Slobodan Mitrovic
  e-mail: boba5555@gmail.com
  date: 25.05.2009.
  Complexity: O(m log max_v) - in this case, max_v = 200000
*/
#include <iostream>
#include <cstdio>
#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 MAXVAL = 200002;
const int HIGHEST_BIT = 1 << 17;

int bit[MAXVAL];
long long bitWeight[MAXVAL]; // 100000 * 200000 > 2^32

void update(int idx){
  int weight = idx;
  while (idx < MAXVAL){
    bit[idx]++;
    bitWeight[idx] += weight;
    idx += (idx & -idx);
  }
}

struct mpair{
  long long a, b;
  mpair(){
  }

  mpair(long long _a, long long _b){
    a = _a;
    b = _b;
  }
```

```
};

mpair read(int idx){
  long long ret = 0, retW = 0;
  while (idx){
    ret += bit[idx];
    retW += bitWeight[idx];
    idx -= (idx & -idx);
  }

  return mpair(ret, retW);
}

int find(int cumFre){
  int ret = -1, idx = 0, bitMask = HIGHEST_BIT;

    while ((bitMask != 0) && (idx < MAXVAL)){
        int tIdx = idx + bitMask;
        if (tIdx < MAXVAL){
          if (cumFre > bitWeight[tIdx]){
             idx = tIdx;
             cumFre -= bitWeight[tIdx];
          }
          else
        ret = tIdx;
    }
        bitMask >>= 1;
    }

    return ret;
}

int main(){
    freopen("baloni.in", "rt", stdin);
    freopen("baloni.out", "wt", stdout);
    int M, v, ret, canTake, cntOpN = 0, idx;
    mpair rr;
    char op;
    scanf("%d", &M);
    SET(bit, 0);
    SET(bitWeight, 0);
    FOR (i, M){
    scanf("\n%c %d", &op, &v);
    if (op == 'N'){
      update(v);
      cntOpN++;
    }
    else{
      idx = find(v);
      if (idx == -1)
        printf("%d\n", cntOpN);
      else{
        rr = read(idx - 1);
        ret = rr.a;
        v -= rr.b;
        ret += v / idx;
        printf("%d\n", ret);
      }
    }
  }
  return 0;
}
```

zadatak: Torta

Mlađani Zi će uskoro da diplomira. Njegovi drugari su odlučili da mu za poklon naprave beskonačnu veliku tortu. Nakon što su tortu ispekli, nafilovali i ukrasili, drugari su odlučili da tortu iseku u automatskom sekaču. Automatski sekač je naprava koju su Zijevi drugari, elektroničari napravili tokom studija. Sekač radi na sledećem principu: torta se ubaci u sekač, zatim se unese željeni broj rezova *n*. Sekač će zatim seći tortu tačno *n* puta. Ukoliko posmatramo tortu kao beskonačnu pravu, svaki rez predstavlja pravu u toj ravni. Automatski sekač će odabrati položaj *n* rezova nasumično, i nakon završenog sečenja će odštampati izveštaj o položaju svakog od *n* rezova. Sekač je isprogramiran tako da se nikad neće desiti da tri različita reza prolaze kroz istu tačku na torti. U izveštaju sekača svaki rez je opisan dvema tačkama u ravni torte kroz koje prolazi dati rez.
Nakon što su isekli tortu, Zijevi drugari su gledali sekačev izveštaj ne bi li utvrdili na koliko parčića je torta isečena. Obično je mlađani Zi taj koji bi pomogao drugarima u rešavanju sličnih problema. Pošto je torta iznenađenje, drugari su odlučili da vas priupitaju za pomoć.
Pomozite Zijevim drugarima da, koristeći izveštaj sekača odrede na koliko parčića je torta isečena.

Ulaz:

(Ulazni podaci se nalaze u datoteci **torta.in**) U prvom redu ulazne datoteke nalazi se prirodan broj *n* ($1 \leq n \leq 10^6$), koji predstavlja broj rezova koje je sekač napravio. U svakom od narednih *n* redova nalaze se četiri cela broja x_1, y_1, x_2, y_2 razdvojenih razmakom, koji predstavljaju koordinate dveju tačaka kroz koje dati rez prolazi. Ukoliko tortu zamislimo kao beskonačnu ravan, dati rez predstavlja pravu koja prolazi kroz tačke (x_1, y_1) i (x_2, y_2). Svaka od koordinata je u opsegu od -10<sup9< sup=""> do 10^9. Nikoja tri reza neće prolaziti kroz istu tačku na torti.</sup9<>

Izlaz:

(Izlazne podatke upisati u datoteku **torta.out**) U prvi red izlazne datoteke treba upisati prirodan broj *k*, koji predstavlja broj parčića na koje je torta isečena po modulu 1.000.003.

Primer 1:

torta.in	**torta.out**
4 0 0 0 1 1 0 1 1 2 0 2 1 3 0 3 1	5

Primer 2:

torta.in	**torta.out**
3 0 0 1 2 1 2 2 0 0 0 2 0	7

Primer 3:

torta.in	**torta.out**
3 0 0 0 1 1 0 1 1 0 0 1 0	6

fajl: torta.cpp

```
#include <iostream>
#include <cmath>
#include <vector>
#include <algorithm>
#include <fstream>

using namespace std;

int main() {

  ifstream fin("torta.in");
  ofstream fout("torta.out");

  int n; fin >> n;
  vector<double> angle;
  for (int i=0; i<n; ++i) {
```

```
    double x1, y1, x2, y2;
    fin >> x1 >> y1 >> x2 >> y2;
    angle.push_back(atan2(x1-x2, y1-y2));
  }

  sort(angle.begin(), angle.end());
  long long tot = 1, curr = 1;
  long long longN=n;
  tot+= longN*(longN+1)/2;
  for (int i=1; i<n; ++i, ++curr)
    if (angle[i] - angle[i-1] > 1.0e-6) {
      tot-=curr*(curr-1)/2;
      curr = 0;
    }
  tot-=curr*(curr-1)/2;
  tot %= 1000003;

  fout << tot << endl;

  fin.close(); fout.close();

}
```

zadatak: Čokoladomat

Noćica mnogo voli čokoladu. Na sreću, odmah na ulazu firme u kojoj radi, "Megahard", nalazi se automat za čokoladu, čokoladomat. Njena firma se bavi proizvodnjom operativnog sistema "Dors" i, logično, softver na čokoladomatu zasnovan je upravo na tom operativnom sistemu. Nažalost, mnogi tvrde da je ovaj operativni sistem daleko inferioran u odnosu na konkurentski, čija je maskota polarni medved (a koji mnogo voli Noćičin prijatelj Tugomir, što je i predmet njihovih čestih prepirki, ali to nije bitno za ovu priču). Upravo da propagatori konkurencije ne bi dobili za pravo, ono što se jutros desilo s čokoladomatom u "Megahardu" mora ostati tajna — a pošto znamo da ste vi savesna deca, koja neće okolo širiti tračeve (a i budući da Noćica traži pomoć od vas, pa i nema baš neki izbor), ispričaćemo vam.
Softver u čokoladomatu načisto je pobrljavio. Umesto da iznose ubačenih apoena sabere (kako bi bilo prirodno), on na sve njih primeni operaciju XOR. Zaposleni u firmi to su brzo shvatili, ali proći će dosta vremena dok ne stignu rezervni delovi, što je Noćicu bacilo u očaj.
Srećom, uprava firme donela je odluku da se čokoladomat može koristiti i dok ovako pogrešno obračunava ubačen iznos. To je malo olakšalo muke Noćici, ali i dalje je u nedoumici: ukoliko dođe pred čokoladomat s određenom količinom apoena, šta od toga treba da ubaci kako bi dobila što više čokolade, tj. kako bi čokoladomat zaračunao što je veći mogući iznos? Postoji još jedna začkoljica: pauze za uzimanje čokolade u "Megahardu" ne traju baš dugo, pa Noćica nema vremena da polagano prebira po novčaniku. Jedino može stići da odabere dve novčanice, i sve što je u novčaniku između njih (uključujući njih) stavi u čokoladomat. Jasno, Noćica je vrhunska programerka, te joj u normalnim okolnostima ovo ne bi predstavljalo nikakav problem, ali kako ne može da funkcioniše bez čokolade, preklinje vas da joj pomognete.
Ulaz:
(Ulazni podaci se nalaze u datoteci **cokolada.in**) U prvom redu ulazne datoteke nalazi se prirodan broj n ($1 \leq n \leq 250.000$), koji predstavlja broj apoena koje Noćica ima kod sebe (neki od njih mogu biti i jednaki). U narednih n redova nalazi se po jedan ceo broj iz intervala [0,60.000], pri čemu svaki od njih predstavlja vrednost po jednog Noćičinog apoena.
Izlaz:
(Izlazne podatke upisati u datoteku **cokolada.out**) U prvi red i jedini red izlazne datoteke treba upisati jedan ceo broj, koji predstavlja maksimalnu moguću svotu koju Noćici može zaračunati čokoladomat.
Primer 1:

cokolada.in	**cokolada.out**
5 13 3 11 18 12	30

fajl: cokolada.cpp

```
/*
  Author: Slobodan Mitrovic
  e-mail: boba5555@gmail.com
  date: 31.05.2009.
*/
#include <iostream>
#include <cstdio>
#include <cmath>
#include <algorithm>
#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 TREE_SIZE = 1 << 17;

bool tree[TREE_SIZE];

void add(int val){
  int idx = 1, bit = 1 << 16;
  FOR (i, 17){
    tree[idx] = true;
    bit >>= 1;
    idx <<= 1;
    idx += ((val & bit) != 0);
  }
}

int read(int val){
  int idx = 1, cld, cb, ob, ret = val, goonB;

  int bit = 1 << 15;

  FOR (i, 16){
    idx <<= 1;
    cb = (val & bit) != 0; // current bit
    ob = cb ^ 1; // opposite bit
    if (tree[idx + ob])
      goonB = ob;
    else
      goonB = cb;

    idx += goonB;
    ret ^= (goonB * bit);

    bit >>= 1;
  }

  return ret;
}

unsigned short int s;
bool ex[1 << 16];
int n, a;

int main(){
  freopen("cokoladomat.in", "rt", stdin);
  freopen("cokoladomat.out", "wt", stdout);
  SET(tree, 0);
  SET(ex, 0);
  scanf("%d", &n);
```

```
  s = 0;
  add(0);
  int ret = 0;
  FOR (i, n){
    scanf("%d", &a);
    s ^= a;
    if (!ex[s]){
      ex[s] = true;
      ret >?= read(s);
      add(s);
    }
  }

  printf("%d\n", ret);
  return 0;
}
```

zadatak: Niz

Dat je niz *a* celih brojeva dužine *n*. Pod transformacijom niza podrazumevamo povećavanje ili smanjivanje jednog elemenat niza za 1. Odrediti minimalni broj transformacija niza potrebnih da se on prevede u strogo rastući niz *b*. Za svaki element niza prikazati razliku: *b[i] - a[i]*.

Ulaz:

(Ulazni podaci se nalaze u datoteci **niz.in**) U prvom redu se nalazi prirodni broj *n* ($1 \leq n \leq 5.000$) koji označava broj elemenata niza. U narednom redu se nalazi *n* celih brojeva koji predstavljaju elemente niza. Apsolutna vrednost elemenata nije veća od 1.000.000.000.

Izlaz:

(Izlazne podatke upisati u datoteku **niz.out**) U prvom redu treba ispisati minimalni broj transformacija potrebnih za dobijanje strogo rastućeg niza. U narednom redu štampati *n* brojeva odvojenih jednim razmakom koji predstavljaju promene brojeva početnog niza. Ukoliko promene nisu jedinstvene štampati bilo koju.

Primer 1:

niz.in	**niz.out**
4	4
1 1 1 1	-1 0 1 2

Primer 2:

niz.in	**niz.out**
5	5
1 1 1 6 4	-1 0 1 0 3

Objašnjenje.

U primeru 1 opisanim transformacijama bi dobili rastući niz *b* = 0, 1, 2, 3. U drugom primeru rešenje nije jedinstveno, npr. promene su mogle biti i: -1, 0, 1, -1, 2.

Napomena.

U 40% test primera apsolutna vrednost elemenata niza neće biti veća od 2.000.

fajl: niz.cpp

```
/*
  Author: Andreja Ilic, PMF Nis
  e-mail: ilic_andrejko@yahoo.com
  Complexity: O(n^2)
*/
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include <algorithm>

using namespace std;
#define MAX_N 5005

int a [MAX_N], c [MAX_N], b [MAX_N], n, m, p [MAX_N][MAX_N], h [MAX_N];
long long d [MAX_N][MAX_N], sol;
```

```
FILE *in, *out;

    // Funkcija uporedjivanje za sortiranje
    int compare (const void * a, const void * b)
    {
        return (*(int*)a - *(int*)b );
    }

    // Inicijalizacija niza c = razlicite vrednosti elemenata
    // niza a, sortirani u rastucem poretku
    void initializationOfSteps()
    {
        for (int i = 0; i < n; i++)
        {
            c [i] = a [i];
        }
        sort(c, c + n);

    // izbacivanje istig vrednosti iz niza c
    // m = broj elemenata niza c
        m = 1;
        for (int i = 1; i < n; i++)
        {
            if (c [i] != c [i - 1])
            {
                c [m++] = c [i];
            }
        }
    }

    // Glavna metoda za racunanje resenja
    void solve()
    {
        // prebacivanje rada bez uslova stroge nejednakosti
        for (int i = 0; i < n; i++)
        {
            a [i] = a [i] - i;
        }

    // inicijalizacija mogucih vrednosti niza b
        initializationOfSteps();

        d [0][0] = abs (a [0] - c [0]);
        p [0][0] = 0;

        // Inicijalizacija prve kolone
        for (int i = 1; i < n; i++)
        {
            d [i][0] = d [i - 1][0] + abs (a [i] - c [0]);
            p [i][0] = 0;
        }

        // Inicijalizacija prve vrste
        for (int j = 1; j < m; j++)
        {
            if (abs (a [0] - c [j]) < d [0][j - 1])
            {
                d [0][j] = abs (a [0] - c [j]);
                p [0][j] = j;
            }
            else
            {
                d [0][j] = d [0][j - 1];
                p [0][j] = p [0][j - 1];
            }
        }
```

```
    // Inicijalizacija matrice d
    for (int i = 1; i < n; i++)
    {
        for (int j = 1; j < m; j++)
        {
            if (d [i][j - 1] <= d [i - 1][j] + abs(a [i] - c [j]))
            {
      // u slucaju da niko od elemnata ne uzima vrednost c [j]
                d [i][j] = d [i][j - 1];
                p [i][j] = p [i][j - 1];
            }
            else
            {
      // u slucaju da element sa indeksom i uzima vrednost c [j]
                d [i][j] = d [i - 1][j] + abs (a [i] - c [j]);
                p [i][j] = j;
            }
        }
    }

    // Racunanje minimalnog broja transformacija
    sol = d [n - 1][m - 1];
// currentIndex = index vrednosti elementa, preko niza c
    int currentIndex = p [n - 1][m - 1];
    for (int j = m - 2; j >= 0; j--)
    {
        if (sol > d [n - 1][j])
        {
            sol = d [n - 1][j];
            currentIndex = p [n - 1][j];
        }
    }

    // Nalazenje pomeraja za svaki element
    for (int i = n - 1; i >= 0; i--)
    {
        b [i] = c [currentIndex];
        if (i == 0)
            break;
        int lastIndex = currentIndex;
  // updatovanje currentIndex-a
        int currentMin = d [i - 1][currentIndex];
        currentIndex = p [i - 1][currentIndex];
        for (int j = lastIndex - 1; j >= 0; j--)
        {
            if (currentMin > d [i - 1][j])
            {
                currentMin = d [i - 1][j];
                currentIndex = p [i - 1][j];
            }
        }
    }
}

// Unos podataka
void input()
{
    in = fopen ("niz.in", "r");
    fscanf (in, "%d", &n);
    for (int i = 0; i < n; i++)
    {
        fscanf (in, "%d", &a [i]);
    }
    fclose(in);
}

// Ispis resenja
```

```
void output()
{
    out = fopen ("niz.out", "w");
    fprintf (out, "%lld\n", sol);
    for (int i = 0; i < n; i++)
        fprintf (out, "%d ", (b [i] - a [i]));
    fprintf(out, "\n");
    fclose(out);
}

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

    return 0;
}
```