Zadaci sa rešenjima Državno takmičenje 2011.

zadatak: Računi

Đurica voli da obilazi restorane. Tokom godine je obixao puno restorana, i sad je baš ogladneo i zanima ga u koji sledeći da ode. U svim restoranima koje je Đurica posetio ukupno je naručio k različitih stavki, te se svaki račun može opisati kao

$$id\ s_1\ s_2\ \ldots\ s_k.$$

Vrednost *id* je oznaka restorana. Svaka oznaka jedinstveno određuje jedan restoran, i svaki restoran je jedinstveno određen jednom oznakom. Vrednost s_i je 0 ako na tom računu ne postoji i-ta stavka, inače je prirodan broj koji predstavlja koliko je plaćena i-ta stavka. Đurica je sakupio tačno n računa, i njega zanima zbirni račun za svaki restoran. Imajući sve račune za restorane, zbirni račun za restoran *id* se definiše kao račun sa k stavki gde vrednost za i-tu stavku predstavlja sumu cena i-te stavke na svim računima restorana *id*.
Jednom kad Đurica ima sve zbirne račune, on smatra da je najbolji restoran u koji sledeći treba otići onaj koji ima leksikografski najmanji zbirni račun stavki. Đurica je gladan i ne može da razmišlja, te pomozite Đurici i za dati spisak računa ispišite lekikografski najmanji račun.
Tokom određivanja i ispisivanja leksikografski najmanjeg zbirnog računa ne treba uzimati u obzir *id* restorana. Za dva zbirna računa $a_1\ a_2\ \ldots\ a_k$ i $b_1\ b_2\ \ldots\ b_k$ kažemo da je prvi račun leksikografski manji od drugog ako postoji j takvo da $a_i = b_i$, $i < j$, i $a_j < b_j$.

Ulaz:
(Ulazni podaci se učitavaju iz datoteke **racuni.in**.) U prvom redu se nalaze dva prirodna broja n i k ($1 \le n \le 30.000$, $1 \le k \le 30$) U narednih n redova su dati opisi računa u formatu

$$id\ s_1\ s_2\ \ldots\ s_k.$$

Odgovarajuća ograničenja su: $1 \le id \le 1.000.000.000$, $0 \le s_j \le 100.000$.

Izlaz:
(Izlazni podaci se ispisuju u datoteku **racuni.out**.) U prvom i jedinom redu ispisati k brojeva koji predstavljaju leksikografski najmanji zbirni račun.

Primer:

racuni.in	racuni.out
4 2 1 3 4 2 5 1 1 0 8 7 3 20	3 12

Objašnjenje.
Zbirni računi za restorane 1, 2 i 7 su redom 3 12, 5 1 i 3 20.

Primer:

racuni.in	racuni.out
7 4 11 303 0 0 0 2 10 2 8 0 13 20 3 8 1 11 9 4 0 0 2 9 4 0 1 2 3 0 0 5 13 2 3 0 0	22 6 8 1

Objašnjenje.
Restoran sa brojem 13 ima leksikografski najmanji zbirni račun.

fajl: racuni.cpp

```
#include <iostream>
#include <cstdio>
#include <fstream>
#include <vector>
#include <cmath>
#include <time.h>
#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 = 30000;
const int MAXK = 30;

struct recipe{
  int id, idx;
  recipe(){
  }

  recipe(int _id, int _idx){
    id = _id;
    idx = _idx;
  }

  friend bool operator <(const recipe &a, const recipe &b){
    return a.id < b.id;
  }
};

int val[MAXN][MAXK];

recipe recipes[MAXN];

long long sum[MAXN][MAXK];

int main(){
  char filein[32] = "racuni.in";
  char fileout[32] = "racuni.out";
  FILE *fin;
  FILE *fout;

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

  int n, k, id;
  fscanf(fin, "%d %d", &n, &k);
  FOR (i, n){
    fscanf(fin, "%d", &id);
    recipes[i] = recipe(id, i);
    FOR (j, k)
      fscanf(fin, "%d", &val[i][j]);
  }

  sort(recipes, recipes + n);
  int idx = 0, j, cntIds = 0;
  while (idx < n){
    FOR (i, k)
      sum[cntIds][i] = 0LL;
    j = idx;
    while (j < n && recipes[idx].id == recipes[j].id){
      FOR (i, k)
        sum[cntIds][i] += val[recipes[j].idx][i];
      j++;
    }
    idx = j;
    cntIds++;
  }

  long long ret[k];
  ret[0] = 1LL << 60LL;
  FOR (i, cntIds){
    bool found = false;
```

```
    FOR (j, k)
      if (ret[j] > sum[i][j]){
        found = true;
        break;
      }
      else if (ret[j] < sum[i][j])
        break;
    if (found)
      FOR (j, k)
        ret[j] = sum[i][j];
  }

  FOR (i, k)
    fprintf(fout, "%lld ", ret[i]);

  fprintf(fout, "\n");
  fclose(fin);
  fclose(fout);
  return 0;
}
```

zadatak: Sumhem

U teoriji informacija, Hemingovo rastojanje (eng. *Hamming distance*) između dva stringa jednakih dužina je broj pozicija na kojima se odgovarajući simboli razlikuju. Drugim rečima, ovo rastojanje meri minimalni broj zamena potrebnih za transformaciju jednog stringa u drugi.
U zadatku ćemo posmatrati 32-bitne brojeve, gde se Hemingovo rastojanje ham(*a*, *b*) računa kao broj bitova na kojima se brojevi *a* i *b* razlikuju. Na primer, Hemingovo rastojanje za brojeve 93 = 00...0010**1**1**10**1 i 75 = 00...0010**0**1**01**1 je 3, pošto se brojevi razlikuju na drugom, trećem i petom bitu s desna.
Dat je niz celih brojeva *a* dužine *n*. Odrediti zbir Hemingovih rastojanja svih parova elemenata niza, odnosno

$$\Sigma_{i<j} \text{ham}(a[i], a[j]).$$

Ulaz:
(Ulazni podaci se učitavaju iz datoteke **sumhem.in**.) U prvom redu se nalazi prirodan broj *n* ($1 < n < 100.000$). U sledećih *n* redova se nalazi po jedan broj i oni predstavljaju niz *a* ($0 < a[i] < 2^{31} - 1$).
Izlaz:
(Izlazni podaci se ispisuju u datoteku **sumhem.out**.) U prvom i jedinom redu ispisati sumu Hemingovih rastojanja svaka dva broja u nizu.
Primer:

sumhem.in	sumhem.out
4 1 9 4 10	14

Objašnjenje.
Hemingova rastojanja za svaki par brojeva iz niza su jednaka *ham*(1, 9) = 1, *ham*(1, 4) = 2, *ham*(1, 10) = 3, *ham*(9, 4) = 3, *ham*(9, 10) = 2 i *ham*(4, 10) = 3. Suma svih rastojanja je upravo 14.
Primer:

sumhem.in	sumhem.out
7 128 7 0 99 18 10 1984	84

Napomena.
U 40% test primera, broj *n* će biti manji od 3.000.

fajl: sumhem.cpp

```
/*
  Problem: Dat je niz a duzine n <= 100.000 prirodnih brojeva manjih od 2^31.
  Naci sumu Hemingovih rastojanja svaka dva broja a [i] i a [j], gde je i < j.
  Hemingovo rastojanje za dva broja se definise kao broj bitova na kojima se ti brojevi
razlikuju.

  Resenje: Kvadratno resenje u O (32 n^2) donosi 40 poena.
  Za linearno resenje O (32 n) je potrebno primetiti da svaki bit mozemo posmatrati
nezavisno.
  Za i-ti bit, 0 <= i <= 31, neka count predstavlja koliko brojeva imaju 1 na i-toj
poziciji.
  Tada na ukupnu sumu dodajemo tacno count * (n - count).
  U resenju koristimo neoznacene brojeve sa 64 bita i idemo do bita najvece tezine.

  Autor: Aleksandar Ilic
*/

#include <stdio.h>
#define MAXN 100001

unsigned int a [MAXN];
int n;
long long sum;

long long check ()
{
  long long sum = 0;
  for (int i = 0; i < n; i++)
    for (int j = i + 1; j < n; j++)
    {
      unsigned int x = a [i] ^ a [j];
      while (x > 0)
      {
        sum++;
        x = x & (x - 1);
      }
    }
  return sum;
}

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

  int max = a [0];
  for (int i = 0; i < n; i++)
    if (max < a [i])
      max = a [i];
  int max_bit = 1;
  while ((max_bit < 32) && (max > (1 << max_bit)))
    max_bit++;

  sum = 0;
  unsigned int pow = 1;
  for (int i = 0; i < max_bit; i++)
  {
    int count = 0;
    for (int j = 0; j < n; j++)
      if ((a [j] & pow) != 0)
        count++;
    sum = sum + (long long)count * (long long)(n - count);
```

```
    pow = pow << 1;
  }
  //printf ("%lld %lld\n", sum, check ());

  FILE *out = fopen ("sumhem.out", "w");
  fprintf (out, "%lld\n", sum);
  fclose (out);
  return 0;
}
```

zadatak: Papirići

Perica je oduvek voleo igre na sreću, pa je smislio jednu za svog druga Jovicu. Perica uzme *n* papirića, na svakom papiriću napiše po jedan broj i okrene ih tako da Jovica ne zna koji su brojevi napisani. Zatim Jovica mora da rasporedi sve papiriće na proizvoljan broj gomila. Svaka gomila mora da ima najmanje 2, a najviše *k* papirića. Na kraju Perica otkrije sve papiriće, te računa broj poena koji je Jovica osvojio.
Perica računa poene na sledeći način: Pronađe najveći i najmanji broj na jednoj gomili i izračuna njihovu razliku. Kada izračuna razliku najvećeg i najmanjeg za svaku gomilu, sabere sve razlike. Dobijena vrednost predstavlja osvojene poene. Što je broj poena manji, to je Jovica uspešniji.
Znajući koje brojeve je Perica napisao na papiriće, odredite koliko najmanje poena Jovica može da sakupi.
Ulaz:
(Ulazni podaci se učitavaju iz datoteke **papirici.in**.) U prvom redu ulaza nalaze se dva broja *n* i *k* (2 ≤ *k* ≤ *n* ≤ 100.000) koji predstavljaju, redom, broj papirića i maksimalnu veličinu gomila. U drugom redu se nalazi *n* celih brojeva, koji predstavljaju brojeve napisane na papirićima. Svi brojevi će biti u intervalu $[-2^{31}, 2^{31} - 1]$.
Izlaz:
(Izlazni podaci se ispisuju u datoteku **papirici.out**.) U prvom i jedinom redu izlaza ispisati najmanji broj poena koji može da se sakupi. Rešenje će uvek postojati.
Primer:

papirici.in	papirici.out
7 3 6 0 5 2 8 0 -2	7

Objašnjenje.
Na prvu gomilu se stave papirići sa brojevima 0 i 2, na drugu gomilu papirići sa 6, 5 i 8, a na treću papirići sa 0 i -2. Ukupan broj poena je (2 - 0) + (8 - 5) + (0 - (-2)) = 7
Napomena.
U 30% test primera biće *k* ≤ 100.

fajl: papirici.cpp

```
/*
  Autor: Vanja Petrovic Tankovic
*/
#include <cstdio>
#include <algorithm>

#define INF 100000000000ll
#define MAXN 100001

using namespace std;

int n,k,niz[MAXN];
long long razlika[MAXN],resenje;

int main()
{
  freopen("papirici.in","r",stdin);
  freopen("papirici.out","w",stdout);

  scanf("%d %d",&n,&k);
  for (int i=0; i<n; i++)
    scanf("%d",&niz[i]);
```

```
  sort(niz,niz+n);

  if (k==2)
    for (int i=0; i<n; i+=2)
      resenje+=(long long)niz[i+1]-niz[i];
  else
  {
    razlika[0]=INF;
    razlika[1]=(long long)niz[1]-niz[0];
    razlika[2]=(long long)niz[2]-niz[0];
    for (int i=3; i<n; i++)
      if (razlika[i-2]-niz[i-1]<razlika[i-3]-niz[i-2])
        razlika[i]=(long long)niz[i]-niz[i-1]+razlika[i-2];
      else
        razlika[i]=(long long)niz[i]-niz[i-2]+razlika[i-3];
    resenje=razlika[n-1];
  }

  printf("%u",resenje);
  return 0;
}
```

zadatak: Mastilo

Nabavili smo jednu od super-upijajućih krpi sa jednog od tv shop-ova i želimo da je testiramo. Krpa je pravougaonog oblika i ima šare u obliku vertikalnih i horizontalnih linija koje je dele na $n \times m$ podudarnih kvadratića. Test se sastoji od prolivanja mastila na neke od $n \times m$ kvadratića krpe (svaki kvadratić je ili uprljan mastilom ili potpuno čist na početku) i merenja vremena za koje će krpa upiti svo mastilo. Naravno, ispostavlja se da je krpa skupa i beskorisna. Osim xto ne upija leto kako treba, ona doprinosi njegovom razlivanju. Preciznije, svake sekunde leto se razliva na susedne, čiste kvadratiće (tj. svaki čist kvadratić koji u toj sekundi ima **bar jednog uprljanog suseda** postaje uprljan mastilom) a **u isto vreme** krpa upija ono što je do tada bilo uprljano mastilom (tj. **svaki** uprljani kvadratić postaje čist). Dva kvadratića su susedna ako dele zajedničku stranicu.
Pošto smo razočarani krpom, a ne želimo da nam eksperiment propadne, odlučili smo da predvidimo kako će krpa izgledati posle t sekundi.

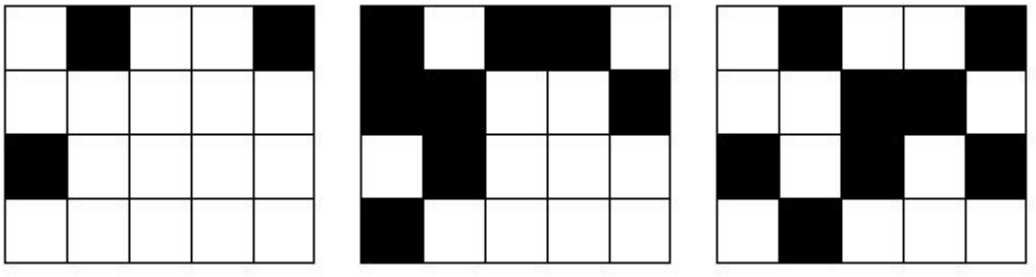
Ulaz:
(Ulazni podaci se učitavaju iz datoteke **mastilo.in**.) U prvom redu ulazne datoteke nalaze se 3 prirodna broja n, m i t koji predstavljaju, redom, dimenzije krpe i vreme (u sekundama) posle kog nas interesuje izgled krpe ($1 \leq n, m \leq 10^3$, $1 \leq t \leq 10^6$). Sledećih n redova sadrže niz od m cifara iz skupa {0, 1} **bez razmaka** - izgled krpe nakon početnog prolivanja mastila. 1 označava da se na odgovarajućem mestu nalazi uprljan kvadratić, a 0 čist kvadratić.

Izlaz:
(Izlazni podaci se ispisuju u datoteku **mastilo.out**.) U prvih n redova izlazne datoteke ispisati po m cifara iz skupa {0, 1} bez razmaka - izgled krpe nakon t sekundi. 1 i 0 imaju isto značenje kao na ulazu.

Primer:

mastilo.in	mastilo.out
4 5 2	01001
01001	00110
00000	10101
10000	01000
00000	

Objašnjenje.
Za dati test primer, promena krpe je prikazana na slikama:



Početak Posle 1 sekunde Posle 2 sekunde

Napomena.
U 25% test primera biće $1 \leq n, m, t \leq 200$. Zbog veličine ulaza, C/C++ kodovi bi trebalo da učitavaju cele redove.

fajl: mastilo.cpp

```
#include <cstdlib>
#include <cstdio>

const int MaxN = 1010;
const int inf = 1000000000;

const int dx[] = {0, 1, 0, -1};
const int dy[] = {1, 0, -1, 0};

int n, m, t, color;
char s[MaxN];
bool a[MaxN][MaxN], tmp[MaxN][MaxN];
int Qx[MaxN*MaxN], Qy[MaxN*MaxN];
int d[MaxN][MaxN];

bool inside(int x, int y) {
  return (0 <= x && x < n && 0 <= y && y < m);
}

void initialStep() {
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++) {
      tmp[i][j] = false;
      if (!a[i][j])
        for (int k = 0; k < 4; k++)
          if (inside(i + dx[k], j + dy[k]))
            tmp[i][j] = tmp[i][j] || a[ i + dx[k] ][ j + dy[k] ];
    }
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
      a[i][j] = tmp[i][j];
}

void BFS() {
  int first = -1, last = 0;
  int x, y, x1, y1;

  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
      if (a[i][j]) {
        Qx[last] = i;  Qy[last] = j;
        d[i][j] = 0;
        last++;
      }
      else d[i][j] = inf;
```

```
  while (first < last - 1) {
    first++;
    x = Qx[first]; y = Qy[first];
    for (int i = 0; i < 4; i++) {
      x1 = x + dx[i];  y1 = y + dy[i];
      if (inside(x1, y1) && d[x1][y1] == inf) {
        d[x1][y1] = d[x][y] + 1;
        Qx[last] = x1;  Qy[last] = y1;
        last++;
      }
    }
  }
}

int main() {

  FILE* inFile = fopen("mastilo.in", "r");
  FILE* outFile = fopen("mastilo.out", "w");

  fscanf(inFile, "%d%d%d", &n, &m, &t);
  for (int i = 0; i < n; i++) {
    fscanf(inFile, "%s", s);
    for (int j = 0; j < m; j++)
      a[i][j] = (s[j] == '1');
  }

  initialStep();
  BFS();

  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      if (d[i][j] > t - 1)
        color = 0;
      else
        color = (t - d[i][j]) % 2;
      if (color == 0) fprintf(outFile, "0");
      else fprintf(outFile, "1");
    }
    fprintf(outFile, "\n");
  }

  fclose(inFile);
  fclose(outFile);
  return 0;
}
```

zadatak: Leto

Katarina obožava da putuje i već je počela da pravi planove za ovo leto. Napravila je spisak sa n egzotičnih mesta koja bi volela da poseti. Za svako mesto poznato je koliko dana bi ona tamo ostala, to su vrednosti t_i, i pri tome za svako $i \neq j$ važi $t_i \neq t_j$.
Katarina ima p prijatelja sa kojima može da putuje. Međutim, ne žele svi prijate i da idu na sva mesta, već postoji p parova brojeva b_i i e_i ($1 \leq b_i \leq e_i \leq n$). Oni nam govore da i-ti prijatelj želi da ide samo na mesta sa rednim brojem j takvim da važi $b_i \leq j \leq e_i$.
Potrebno je pronaći koliko najviše dana Katarina može da provede na putovanjima, ako moraju da budu zadovoljeni sledeći uslovi:

- Katarina nigde ne može da putuje sama.
- Sa svakim prijateljem ona može da ide na najviše jedno mesto.
- Svako mesto ona može posetiti najviše jednom.

Ulaz:

(Ulazni podaci se učitavaju iz datoteke **leto.in**.) U prvom redu ulazne datoteke se nalazi broj n ($n \leq 5.000$), broj mesta. U sledećem redu nalazi se n brojeva razdvojenih razmacima, to su vrednosti t_i ($1 \leq t_i \leq 100.000$). Sledi red u kome se nalazi broj p ($p \leq 5.000$), broj prijatelja. U svakom od narednih p redova nalaze se po dva broja, b_i i e_i.

Izlaz:

(Izlazni podaci se ispisuju u datoteku **leto.out**.) U izlaznu datoteku treba upisati samo jedan broj, koliko najviše dana mogu trajati sva Katarinina putovanja zajedno.

Primer:

leto.in	**leto.out**
8 2 3 1 4 6 5 20 9 5 5 6 2 5 1 2 8 8 8 8	23

Objašnjenje.

Postoji 8 mesta, na prvom se ostaje 2 dana, na drugom 3 dana, itd. Katarina ima 5 prijatelja. Prvi prijatelj želi da ide na mesto 5 ili 6, drugi na neko od mesta 2, 3, 4 i 5, itd. Optimalno rešenje dobija se na primer ako sa prvim prijateljem ode na šesto mesto (5 dana), sa drugim na peto mesto (6 dana), sa trećim na drugo (3 dana), i sa petim na osmo mesto (9 dana). To ukupno daje 5 + 6 + 3 + 9 = 23 dana.

Napomena.

U 60% test primera biće $n, p \leq 1.000$, a u 30% $n, p \leq 200$.

fajl: leto.cpp

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

using namespace std;

const int maxN = 5000;

typedef struct { int t, index; } Mesto;
bool compareMesta(Mesto const& m1, Mesto const& m2) {
    return (m1.t > m2.t);
}
typedef struct { int b, e, index; } Interval;
bool compareIntervali(Interval const& i1, Interval const& i2) {
    return (i1.e < i2.e) || (i1.e == i2.e && i1.b > i2.b);
}

int n, p;
Mesto mesta[maxN];
Interval intervali[maxN];

int res;

int intervaluDodeljen[maxN], tmp[maxN];

void ucitajPodatke() {
    freopen("leto.in", "r", stdin);
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &(mesta[i].t));
        mesta[i].index = i;
    }
    scanf("%d", &p);
    for (int i = 0; i < p; i++) {
        scanf("%d%d", &(intervali[i].b), &(intervali[i].e));
        intervali[i].b--;
```

```
        intervali[i].e--;
    }
}

void sortirajMesta() {
    sort(mesta, mesta + n, compareMesta);
}
void sortirajIntervale() {
    sort(intervali, intervali + p, compareIntervali);
}

bool uIntervalu(int mesto, int interval) {
    return intervali[interval].b <= mesto && mesto <= intervali[interval].e;
}

int probajDaDodas(int mesto) {
    int interval = 0;
    while (interval < p) {
        tmp[interval] = intervaluDodeljen[interval];
        if (uIntervalu(mesto, interval))
            if (tmp[interval] == -1) {
               tmp[interval] = mesto;
               return interval;
            }
            else if (tmp[interval] > mesto) {
                int pom = tmp[interval];
                tmp[interval] = mesto;
                mesto = pom;
            }
        interval++;
    }
    return -1;
}

void resi() {
  for (int i = 0; i < p; i++)
    intervaluDodeljen[i] = -1;
    for (int i = 0; i < n; i++) {
    int dodatU = probajDaDodas(mesta[i].index);
    if (dodatU != -1) {
      res += mesta[i].t;
      for (int j = 0; j <= dodatU; j++)
        intervaluDodeljen[j] = tmp[j];
    }
    }
}

void sacuvajResenje() {
    freopen("leto.out", "w", stdout);
    printf("%d\n", res);
}

int main() {
    ucitajPodatke();
    sortirajMesta();
    sortirajIntervale();

    resi();
    sacuvajResenje();

    return 0;
}
```