Zadaci sa rešenjima Državno takmičenje 2008.

zadatak: Trojke

Profesor matematike je postavio Dragančetu sledeći zadatak. Na osnovu datog niza brojeva a_1, a_2, ..., a_n, Draganče treba da za svaku trojku indeksa (i, j, k), gde je $1 \le i < j < k \le n$, napiše na tabli najveći od brojeva a_i, a_j *i* a_k. Zatim treba da izračuna ostatak koji daje zbir svih brojeva koji su napisani na tabli pri deljenju sa 10007. Profesor je obećao Dragančetu peticu za kraj školske godine, ako dobije tačno rešenje pre kraja časa. Pomozite Dragančetu da što brže dobije tačan rezultat.

Ulaz:
(Ulazni podaci se nalaze u datoteci **trojke.in**) U prvom redu ulazne datoteke nalazi se broj n ($3 \le n \le 30000$). U sledećih n redova se nalaze celi brojevi a_1, a_2, ..., a_n, pri čemu je $-100000 \le a_i \le 100000$.

Izlaz:
(Izlazne podatke upisati u datoteku **trojke.out**) U prvom i jedinom redu izlazne datoteke ispisati sumu brojeva napisanih na tabli po modulu 10007.

Primer 1:

trojke.in	trojke.out
4 3 -1 2 2	11

Objašnjenje.
Sve trojke niza brojeva su: (3, -1, 2), (3, -1, 2), (3, 2, 2), (-1, 2, 2). Na tabli su napisani brojevi 3, 3, 3, 2, pa je rešenje u ovom slučaju 11.

Primer 2:

trojke.in	trojke.out
6 8 -10 4 5 2 6	

fajl: trojke.cpp

```
/*
  DRZAVNO TAKMICENJE 2008
  ZADATAK: trojke
  AUTOR: Aleksandar i Andreja Ilic, Nis

  Za svaku uredjenu strago rastucu trojku indeksa (i, j, k)
  treba sabrati max (a_i, a_j, a_k). Na pocetku sortiramo niz
  u nerastucem poredtku. Broj a [1] je najveci u svakoj trojci u
  kojoj se nalazi - dakle tacno (n - 1)(n - 2) / 2 puta. Slicno
  dobijamo da se a [k] pojavljuje (n - k)(n - k - 1) / 2 puta.

    Pri mnozenju moramo paziti da ne dodje do prekoracenja,
  tako da posle svakog sabiranja i mnozenja uzimamo ostatak pri
  deljenju sa 10007.

    Vremenska slozenost gore navedenog algoritma je O(n log n),
  dok je memorijska O (n).
*/

#include <stdlib.h>
#include<stdio.h>
#include <time.h>
#define maxN 30005
#define mod 10007
```

```
int n, sol, a [maxN];
FILE *in, *out;

  // Permutovanje niza
  void mixArray()
  {
        srand(time(0));
    for (int k = 1; k < n; k++)
    {
      int x = rand() % k;
      int tmp = a [x];
            a [x] = a [k];
            a [k] = tmp;
    }
  }

  // Sortiranje niza
  void Sort (int l,  int d)
  {
    int tmp, pivot, i, j;
    if (l < d)
    {
      pivot = a [(l + d) / 2];
      i = l;
      j = d;
      do
      {
        while (a [i] > pivot) i++;
        while (a [j] < pivot) j--;
        if (i <= j)
        {
             tmp = a [i]; a [i] = a [j]; a [j] = tmp;
          i++; j--;
        }
      }
      while (i <= j);
      Sort (l, j);
      Sort (i, d);
      }
  }

  // Glavna fja
  void Solve()
  {
        mixArray();
    Sort(0, n - 1);
    for (int i = 0; i < n; i++)
    {
        a [i] = a [i] % mod;
        if (a [i] < 0)
           a [i] += mod;
        }
    sol = 0;
    for (int i = 0; i < n - 2; i++)
    {
            int tmp = (n - i - 1) * (n - i - 2) / 2;
            if (tmp > mod)
               tmp = tmp % mod;
            sol += tmp * a [i];
            if (sol > mod)
               sol = sol % mod;
    }
  }

    // Unos podataka
  void InPut()
  {
```

```
    in = fopen ("trojke.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 ("trojke.out", "w");
    fprintf (out, "%d\n", (int) sol);
    fclose(out);
  }

int main()
{
  InPut();
  Solve();
  OutPut();
  return 0;
}
```

zadatak: Poruka

Mali Đurica kuca poruku na mobilnom telefonu. On je otkucao *n* redova. Kada mali Đurica kaže da se kursor nalazi na poziciji (*i*, *j*), to znači da se nalazi u *i*-tom redu iza *j*-tog znaka - ako je *j* nula, to znači da je kursor na početku *i*-tog reda. Malog Đuricu zanima koliko najmanje puta treba da pritisne tastere gore, dole, levo, desno (koji služe za pomeranje kursora kroz poruku) tako da od pozicije (*sr*, *sc*) stigne do pozicije (*fr*, *fc*). Kursor sem pozicije sadrži i vrednost poslednje pozicije u redu koja je dobijena pritiskom tastera levo ili desno, čije će značenje detaljno biti opisano u nastavku problema. Ovu vrednost ćemo zvati *poslPoz*. Mali Đurica će Vam reći koliko redova ima u njegovoj poruci i koliko je dugačak svaki red, tj. len_i broj koji predstavlja broj znakova u *i*-tom redu (odnosno dužinu *i*-tog reda). On Vam takođe šalje na koji način funkcionišu njegovi tasteri i njegov mobilni:

1. Sledeći red poslednjeg reda je prvi red; prethodni red prvog reda je poslednji red, odnosno redovi su organizovani ciklično. Prelaskom u sledeći, odnosno prethodni red, kursor će se naći na poziciji koju pokazuje *poslPoz* u skladu sa pravilom 8.
2. Pritiskom na taster dole kursor prelazi u sledeći red; pritiskom na taster gore kursor prelazi u prethodni red.
3. Pritiskom na taster gore ili dole vrednost *poslPoz* se ne menja.
4. Ako se kursor nalazi na početku reda *r*, pritiskom na taster levo kursor prelazi na poziciju iza poslednjeg znaka prethodnog reda i vrednost *poslPoz* se menja na vrednost pozicije iza tog znaka.
5. Ako se kursor ne nalazi na početku reda *r*, tada pritiskom na taster levo za pozicije (*r*, *c*), prelazi na (*r*, *c*-1), a *poslPoz* postaje *c*-1.
6. Ako se kursor nalazi iza poslednjeg znaka reda *r*, pritiskom na taster desno kursor prelazi na početak sledećeg reda i vrednost *poslPoz* postaje 0.
7. Ako se kursor ne nalazi iza poslednjeg znaka reda *r*, tada pritiskom na taster desno sa pozicije (*r*, *c*), prelazi na (*r*, *c*+1), a *poslPoz* postaje *c*+1.
8. Ako se u nekom momentu kursor nađe u redu koji ima *c* karaktera, a vrednost *poslPoz* je veća od *c*, tada će kursor biti na poziciji *c* u tom redu. Vrednost *poslPoz* SE NE MENJA u tom slučaju.

Ulaz:
(Ulazni podaci se nalaze u datoteci **poruka.in**) U prvom redu se nalazi prirodan broj n ($1 \leq n \leq 100$). Potom se u n redova nalaze celi brojevi len_i ($0 \leq len_i \leq 100$) koji redom označavaju dužine redova. Zatim se u n+2-om redu nalaze prirodni brojevi sr ($1 \leq sr \leq n$), sc ($0 \leq sc \leq len_{sr}$), fr ($1 \leq fr \leq n$) i fc ($0 \leq fc \leq len_{fr}$). Prva dva od tih polja označavaju polaznu poziciju (redni broj vrste i kolone), a druga dva završnu poziciju.
Izlaz:
(Izlazne podatke upisati u datoteku **poruka.out**) U izlaznu datoteku ispisati minimalni broj pritisaka tastera da se od pozicije (*sr*, *sc*) dođe do pozicije (*fr*, *fc*) koristeći navedena pravila.

- Maksimalno vreme izvršavanja programa je 0.2 sekunda.

Primer 1:

poruka.in	**poruka.out**
5 1 3 1 3 1 4 3 2 3	2

Objašnjenje.
Poruka bi mogla da izgleda ovako
X
XXX
X
XX**X**
X
a kursor se nalazi iza boldovanog slova. Pozicija je (4, 3), *poslPoz* ima vrednost 3. Tada pritiskom gore kursor prelazi na kraj 3. reda i ima poziciju (3, 1), a *poslPoz* ostaje ista. Ponovo pritiskom gore prelazi u 2. red, a kursor prelazi na poziciju u redu koju pokazuje *poslPoz* i kursor se nalazi na željenoj poziciji.

Primer 2:

poruka.in	**poruka.out**
7 10 100 100 100 100 100 100 2 50 3 10	5

Objašnjenje.
Poruka bi mogla da izgleda ovako
XXXXXXXXXX
XX...XX - ukupno 100 znakova X
XX...XX - ukupno 100 znakova X
... - 6 redova, svaki sa 100 znakova X
XX...XX - ukupno 100 znakova X.
Pritiskom na taster gore kursor prelazi na (1, 10); *poslPoz* je 50. Pritiskom na taster levo kursor prelazi na (1, 9); *poslPoz* postaje 9. Pritiskom na taster desno kursor prelazi na poziciju (1, 10); *poslPoz* postaje 10. Pritiskom dva puta na taster dole kursor prelazi na poziciju (2, 10), pa (3, 10).

fajl: poruka.cpp

```
/*
  Drzavno takmicenje 2008
  Zadatak: poruka
  Autor: Slobodan Mitrovic, Deronje

  Zadatak se resava obilaskom svih stanja u sirinu, odnosno primenom BFSa.
  Svako stanje mozemo oznaciti sa (r, c, l) gde je (r, c) pozicija
  kursora, a l vrednost promenljive poslPoz
*/


#include <iostream>
#include <cstdio>
#include <queue>
#include <fstream>

using namespace std;
int dp[100][101][101];
```

```
int main(){
  ifstream fin("poruka.in");
  ofstream fout("poruka.out");
  int n;
  fin >> n;
  int a[n];
  for (int i = 0; i < n; i++)
    fin >> a[i];
  int sr, sc, fr, fc;
  fin >> sr >> sc >> fr >> fc;
  sr--;
  fr--;
  queue <int> qr, qc, ql;
  while (!qr.empty())
    qr.pop();
  while (!qc.empty())
    qc.pop();
  while (!ql.empty())
    ql.pop();
  memset(dp, 255, sizeof(dp));
  dp[sr][sc][sc] = 0;
  qr.push(sr); qc.push(sc); ql.push(sc);
  int v[4][2] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
  int ret = 1 << 30;
  while (!qr.empty()){
    int r = qr.front(), c = qc.front(), l = ql.front();
    if (r == fr && c == fc)
      ret = min(ret, dp[r][c][l]);
    qr.pop(); qc.pop(); ql.pop();
    for (int i = 0; i < 4; i++){
      int rr, cc, ll;
      if (v[i][1] == 0){// idemo gore ili dole
        rr = (r + v[i][0] + n) % n;
        cc = ll = l;
        cc = min(cc, a[rr]);
      }
      else{ // idemo levo ili desno
        cc = ll = c + v[i][1];
        rr = r;
        if (cc > a[rr]){
          rr = (rr + 1) % n;
          cc = ll = 0;
        }
        else if (cc < 0){
          rr = (rr - 1 + n) % n;
          cc = ll = a[rr];
        }
      }
      if (dp[rr][cc][ll] != -1)
        continue;
      dp[rr][cc][ll] = dp[r][c][l] + 1;
      qr.push(rr); qc.push(cc); ql.push(ll);
    }
  }
  fout << ret << endl;
  fin.close();
  fout.close();
  return 0;
}
```

zadatak: Najmanji

Dat je niz brojeva a_1, a_2, ..., a_n. Pod konkatenacijom dva broja x i y podrazumevamo broj xy, koji je dobijen nadovezivanjem cifara broja y posle broja x (npr. konkatenacijom brojeva 123 i 45 dobijamo broj 12345). Veliki broj dobijamo kada dopisujemo brojeve jedan iza drugog u nekom redosledu. Odrediti najmanji broj koji se dobija konkatenacijom svih brojeva a_1, a_2, ..., a_n.

Ulaz:
(Ulazni podaci se nalaze u datoteci **najmanji.in**) U prvom redu ulazne datoteke nalazi se ceo broj n ($2 \leq n \leq 5000$). U sledećih *n* redova se nalaze prirodni brojevi a_1, a_2, ..., a_n ($1 \leq a_i \leq 2000000000$). Brojevi su zadati bez početnih (nevažećih) nula.
Izlaz:
(Izlazne podatke upisati u datoteku **najmanji.out**) U prvom redu izlazne datoteke ispisati najmanji broj koji se dobija konkatenacijom učitanih brojeva.
Ograničenja:

- Maksimalno vreme izvršavanja programa je 1 sekunda.

Primer 1:

najmanji.in	najmanji.out
2 91919 919191	91919191919

Primer 2:

najmanji.in	najmanji.out
5 32 11 987 12 3	1112323987

Objašnjenje:
Najmanji broj dobijamo konkatenacijom redom 11, 12, 32, 3, 987.

fajl: najmanji.cpp

```
/*
  DRZAVNO TAKMICENJE 2008
  ZADATAK: najmanji
  AUTOR: Aleksandar i Andreja Ilic, Nis

    Treba odrediti permutaciju unetih brojeva a [1], a [2], ..., a [n]
  tako da je konkatenirani veliki broj najmanji moguc. Vrsimo zamenu
    dva susedna broja x i y  ako je broj xy veci od yx.

    Zbog ogranicenja u zadatku, brojeve je najbolje posmatrati kao
  stringove ili 64-bitne brojeve.

    Vremenska slozenost je O (n log n), a prostorna O (n).

*/

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <cmath>
#define MAXN 5001
using namespace std;

int n, length;
long a [MAXN];

// Fja uporedjivanja dva broja
int compare (const void * a, const void * b)
{
  __int64 x = *(long*) a;
  __int64 y = *(long*) b;

  long tmp = (long) y;
```

```
  __int64 xy = x;
  while (tmp > 0)
  {
    xy = xy * 10;
    tmp = tmp / 10;
  }
  xy = xy + y;

  tmp = (long) x;
  __int64 yx = y;
  while (tmp > 0)
  {
    yx = yx * 10;
    tmp = tmp / 10;
  }
  yx = yx + x;

  if (xy < yx)
    return -1;
  else if (xy > yx)
    return 1;
  else
    return 0;
}

int main ()
{
  /* Ucitavanje podataka */
  ifstream in ("najmanji.in");
  in >> n;
  for (int i = 0; i < n; i++)
    in >> a [i];
  in.close();

  /* Sortiranje */
  qsort (a, n, sizeof (long), compare);

  /* Stampa rezultata */
  ofstream out ("najmanji.out");
  for (int i = 0; i < n; i++)
    out << a [i];
  out << endl;
   out.close();
    return 0;
}

/*
  DRZAVNO TAKMICENJE 2008
  ZADATAK: najmanji
  AUTOR: Aleksandar i Andreja Ilic, Nis

    Treba odrediti permutaciju unetih brojeva a [1], a [2], ..., a [n]
  tako da je konkatenirani veliki broj najmanji moguc. Vrsimo zamenu
    dva susedna broja x i y  ako je broj xy veci od yx.

    Zbog ogranicenja u zadatku, brojeve je najbolje posmatrati kao
  stringove ili 64-bitne brojeve.

    Vremenska slozenost je O (n log n), a prostorna O (n).

*/

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <cmath>
```

```
#define MAXN 5001
using namespace std;

int n, length;
long a [MAXN];

// Fja uporedjivanja dva broja
int compare (const void * a, const void * b)
{
  __int64 x = *(long*) a;
  __int64 y = *(long*) b;

  long tmp = (long) y;
  __int64 xy = x;
  while (tmp > 0)
  {
    xy = xy * 10;
    tmp = tmp / 10;
  }
  xy = xy + y;

  tmp = (long) x;
  __int64 yx = y;
  while (tmp > 0)
  {
    yx = yx * 10;
    tmp = tmp / 10;
  }
  yx = yx + x;

  if (xy < yx)
    return -1;
  else if (xy > yx)
    return 1;
  else
    return 0;
}

int main ()
{
  /* Ucitavanje podataka */
  ifstream in ("najmanji.in");
  in >> n;
  for (int i = 0; i < n; i++)
    in >> a [i];
  in.close();

  /* Sortiranje */
  qsort (a, n, sizeof (long), compare);

  /* Stampa rezultata */
  ofstream out ("najmanji.out");
  for (int i = 0; i < n; i++)
    out << a [i];
  out << endl;
   out.close();
    return 0;
}
```

zadatak: Let

Ispred Vas se nalazi ekran izdeljen na k vetikalnih traka koje su s leva na desno redom označene brojevima 1, 2, ..., k. Na dnu ekrana se nalazi letelica koja može da se kreće kroz trake (iz jedne trake letelica mozhe da pređe samo u susednu), a kojoj treba 1 sekunda da se pomeri u sledeću traku ekrana. Sa gornjeg dela ekrana pada n dijamanata. Za svaki dijamant je poznata cena c, traka l u kojoj pada (dijamant pada u tačno jednoj traci) i momenat t u kojem će dotaći dno ekrana. Letelica je širine trake i može da pokupi neki dijamant samo ako se celom širinom nalazi u traci u kojoj pada dijamant tačno u vremenu kada dijamant dotakne dno ekrana. Ako letelica u momentu x krene da prelazi u susednu traku, tada će se celom svojom širinom u susednoj traci naći u momentu x+1. Znachi, ako u momentu x sa trake s krene da prelazi u susednu traku $s1$ ($s1$ je ili s-1 ili s+1), tada će u traci s pokupiti dijamante koji u trenutku x, a u $s1$ dijamante koji u trenutku x+1 dotiču dno ekrana. Dok se letelica nalazi u jednoj traci celom širinom, u toj traci može da ostane proizvoljno vreme.

Letelica može da skuplja dijamante t sekundi. U momentu 0 letelica se nalazi u traci 1. Vaš zadatak je da nađete maksimalnu cenu dijamanata koju letelica može da pokupi za dato vreme.

Ulaz:

(Ulazni podaci se nalaze u datoteci **let.in**) U prvom redu se nalaze prirodni brojevi brojevi k ($1 \le k \le 50$), n ($1 \le n \le 100000$) i T ($1 \le T \le 100000$). U narednih n redova se nalazi podaci o dijamantima. U i+1-om redu se nalaze prirodni brojevi c_i ($1 \le c_i \le 1000000$), l_i ($1 \le l_i \le k$) i t_i ($1 \le t_i \le 200000$), koji redom oznachavaju cenu dijamanta, traka u kojoj dijamant pada i momenat u kojem će dijamant dotaći dno ekrana.

Izlaz:

(Izlazne podatke upisati u datoteku **let.out**) U prvom redu ispisati maksimalnu cenu dijamanata koje letelica može da pokupi.

Ogranicenja:

- Maksimalno vreme izvršavanja programa je 1.5 sekunda.

Primer 1:

let.in	**let.out**
5 11 10 10 1 2 10 1 2 200 3 2 50 3 2 50 3 2 10 4 2 10 4 2 200 5 5 50 1 4 10 2 2 10 2 2	500

Primer 2:

let.in	**let.out**
4 9 10 200 4 1 200 4 3 5 1 1 5 1 2 5 1 3 5 1 3 5 1 4 5 1 5 5 1 11	200

Objašnjenje:

Na početku igre letelica počinje kretanje prema traci 4. U traku 4 stiže u 3: sekundi i kupi dijamant cene 200. Nakon toga nema vremena da pokupi još nešto (primetimo da se igra završava posle 10 sekundi, pa dijamant koji će na dno ekrana dospeti u 11. sekundi ne moće biti uhvaćen).

fajl: let.cpp

```
/*
  Drzavno takmicenje 2008
  Zadatak: poruka
  Autor: Slobodan Mitrovic, Deronje

  Zadatak se moze resiti dinamickim programiranjem. Naime, dp[k][t] cuva najvecu vrednost
  koja se moze dobiti u k-toj traci u momentu t.
  Vremena u kojima dijamanti dodiruju dno ekrana se sortiraju u neopadajucem poretku.
  Nakon toga prolazimo kroz sve momente i za svaku traku u datom momentu racunamo
  najbolju vrednost koja je jednaka
  max(ako smo u momentu t-1 bili u susednoj traci pa se pomerili u k;
      ako smo u momentu t-1 bili u traci k) +
    vrednost dijamanat koji su dotakli dno ekrana u k-toj traci u momentu t

  NAPOMENA: Primetimo da se stanje u momentu t racuna samo preko stanja
            u momentu t-1, shodno tome, ne moramo cuvati stanja kroz sve momente
            nego samo za prethodni momenat cime se smanjuje memorijska slozenost.
*/

#include <iostream>
#include <cstdio>

using namespace std;

struct mytype{
  int l, t;
  long long c;
  mytype(){
  }

  friend bool operator <(const mytype &a, const mytype &b){
    if (a.t < b.t)
      return true;
    else if (a.t > b.t)
      return false;
    else
      return a.l < b.l;
  }
};

long long dp[50][100000 + 1];

int main(){
  FILE* fin;
  FILE* fout;
   fin = fopen("let.in", "r");
  fout = fopen("let.out", "w");
  int n, k, T, i;
  fscanf(fin, "%d %d %d", &k, &n, &T);
  mytype d[n];
  for (i = 0; i < n; i++){
    fscanf(fin, "%I64d %d %d", &d[i].c, &d[i].l, &d[i].t);
    d[i].l--;
  }
  sort(d, d + n);
  memset(dp, 128, sizeof(dp));
  dp[0][0] = 0;
  int lidx = 0;
  long long sum;
  long long *cur;
  for (int t = 1; t <= T; t++)
    for (i = 0; i < k; i++){
      sum = 0;
      while (lidx < n && d[lidx].t == t && d[lidx].l == i)
        sum += d[lidx++].c;
```

```
      cur = &dp[i][t];
      *cur = dp[i][t - 1] + sum;
      if (i > 0)
        *cur >?= dp[i - 1][t - 1] + sum;
      if (i + 1 < k)
        *cur >?= dp[i + 1][t - 1] + sum;
    }

  sum = 0LL;
  for (i = 0; i < k; i++)
    sum = max(sum, dp[i][T]);

  fprintf(fout, "%I64d\n", sum);
  fclose(fin);
  fclose(fout);
  return 0;
}
```

zadatak: Ogrlica

Nash mali Draganče se našao u nevolji. Pre par nedelja se dogovorio sa drugarima da na leto ide na more u Abenishbe, ali je ubrzo shvatio da nema dovoljno para. Pa je odlučio da se zaposli i da zaradi te pare. Pošto ga niko nije shvatio ozbiljno da sa 10 godina ume da programira, morao je da se zaposli na nekom mnogo manje zanimljivom mestu - u prodavnici nakita. U toj prodavnici imaju jako veliki izbor - ogrlice, minđušhe, narukvice, prstenje... Našem malom Dragančetu za oko su posebno zapale ogrlice od perli. Perle su poređane u krug, i sa nekih perli iz kruga visi još po jedna niska perli. Svake dve susedne perle su povezane malim konchićem (i sa kruga i sa niski). Draganče je primetio da je ogrlica jako uska, tako da retko koja mušterija može da je stavi oko vrata, tako da i oni kojima se svidi, odustanu posle probavanja. Draganče je smislio kako da reši problem! Makazama će preseći jedan končić (koji spaja dve perle sa kruga), i neke dve perle će da spoji konchićem. Tako će da dobije ogrlicu istog tipa - krug sa visećim niskama perli. Poshto već par nedelja nije ništa programirao, jer svaki dan sedi u prodavnici, zamolio vas je da mu pomognete, i izračunate koliki najveći krug na ogrlici može da dobije, sa jednim sečenjem i jednim spajanjem. Naravno, kada bi znao najveći krug, mogao bi svakoj ogrlici da poveća krug, i samim tim poveća broj mušterija.

Ulaz:

(Ulazni podaci se nalaze u datoteci **ogrlica.in**) U prvom redu ulazne datoteke se nalazi broj n ($n \leq 500000$) koji predstavlja broj perli na krugu. U sledećem redu su dva broja, k i m ($k \leq 10000$, $m \leq 10000000$). U sledećih k redova se nalazi po jedan ceo broj niza x_i ($x_i \leq 2*m$). Niz a_i izračunajte koristeći doele navedeni segment programa (niz x_i ima k elemenata i njhovi indeksi su od 0 do k-1, a a_i ima n elemenata i njihovi indeksi su od 0 do n-1):

```
j = 0;
for (i = 0; i < n; i++) f a[i] = x[j];
         s = (j+1) % k;
         x[j] = ((x[j] ^ x[s]) + 13) % m;
         j = s;
}
```

(% označava ostatak po modulu, a ^ označava bitovski xor) Za Pascal programere bi segment imao sledeći izgled:

```
j := 0;
for i := 0 to n-1 do begin
         a[i] := x[j];
         s := (j+1) mod k;
         x[j] := ((x[j] xor x[s]) + 13) mod m;
         j := s;
end;
```

U ovom kodu je xor oznaka za bitovnu eksluzivnu disjunkciju koja postoji kao operacija u Rascal-u.
Broj a_i predstavlja broj perli u niski ispod i-te perle na krugu. Rešenje ne zavisi od gornje formule, u njoj ne postoje zavisnosti koje bi vam pomogle u rešavanju. Ona služi da ulazna datoteka ne bude prevelika, da bi mogla da se učita u vremenskom ograničenju.

Izlaz:

(Izlazne podatke upisati u datoteku **ogrlica.out**) U prvi red izlazne datoteke ispisati jedan ceo broj - broj perli u najvećem krugu koji se može dobiti jednim sečenjem i jednim spajanjem dve perle date ogrlice.

Ogranicenja:

- Maksimalno vreme izvršavanja programa je 1.5 sekunda.

Primer 1:

ogrlica.in	**ogrlica.out**
12 12 5 3 0 3 2 2 0 2 0 1 0 1 0	17

Objašnjenje:
Ogrlica je prikazana na slici ispod odgovora. Nizovi *x* i *a* su identični. Ako presečemo između 2. i 3. perle sa kruga, i spojimo poslednje perle iz niski ispod 2. i 3. perle, dobijamo dužinu 12 + 3 + 2 = 17.

Primer 2:

ogrlica.in	**ogrlica.out**
8 4 9 3 4 0 5	20

Objašnjenje:
Niz *a*, koji treba da se dobije kada generišete je 3, 4, 0, 5, 2, 8, 0, 2.

fajl: ogrlica.cpp

```
#include <stdio.h>

const int maxn = 1000000;

int x[200];
int a[maxn];
int best[2*maxn];
int koji[2*maxn];

void gen(int* x, int k, int* p, int n, int m) {
      int i,j = 0;
      for (i = 0; i < n; i++) {
          a[i] = x[j];
          printf("niz %d\n", a[i]);
          int s = (j+1) % k;
          x[j] = ((x[j] ^ x[s]) + 13) % m;
          j = s;
      }
}

int main() {
    int i,j,k,n,m,poc,kraj,res;
    freopen("ogrlica.in","r",stdin);
    freopen("ogrlica.out","w",stdout);

    scanf("%d", &n);
    scanf("%d %d", &k, &m);
```

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

    gen(x,k,a,n,m);


    best[0] = n + a[0];
    koji[0] = 0;
    poc = 0;
    kraj = 0;


    for (i=1;i<n;i++)
    {
        while (kraj >=poc && best[kraj]<n-i+a[i])
              kraj--;
        kraj++;
        best[kraj] = n-i+a[i];
        koji[kraj] = i;
    }

    res = a[n-1] + best[poc];

    for (i = 0;i<n;i++)
    {
        if (koji[poc]==i)
           poc++;
        while (kraj >=poc && best[kraj]<-i+a[i])
              kraj--;

        kraj++;
        best[kraj] = -i+a[i];
        koji[kraj] = i;

        int tr = a[i] + i + 1 + best[poc];
        res = res>tr?res:tr;
    }

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