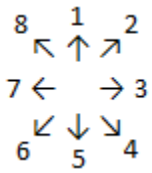
Zadaci sa rešenjima Okružno takmičenje 2011.

zadatak: Osmosmerka

Data je osmosmerka koja umesto slova ima brojeve (od 0 do 10^4). Nisu poznate reči koje treba da se nađu u osmosmerci, ali zato znamo za svaku reč gde počinje, koliko je dugačka i u kom smeru se prostire. Vaš zadatak je da pomoću tih podataka rešite osmosmerku, odnosno da pronađete polja koja ne pripadaju nijednoj od tih reči (polja koja bi ostala neprecrtana).

Ulaz:

(Ulazni podaci se nalaze u datoteci **osmosmerka.in**.) U prvom redu ulazne datoteke se nalaze dimenzije osmosmerke *n* i *m* (*n, m* ≤ 100). Zatim se u svakom od narednih *n* redova nalazi po *m* brojeva - oni predstavljaju sadržaj osmosmerke. Sledi red u kome se nalazi broj *k* (*k* ≤ 10.000), broj reči koje se nalaze u osmosmerci. U svakom od narednih *k* redova se nalaze po četiri broja *i, j, s* i *l*, što znači da odgovarajuća reč počinje sa polja (*i*, *j*), ide u smeru *s* i dužine je *l*. Prvo polje u osmosmerci je polje (1, 1). Smer je broj od 1 do 8 i svaki od njih odgovara smerovima kao na slici.

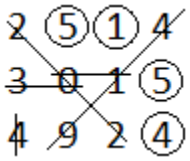


Izlaz:

(Izlazne podatke upisati u datoteku **osmosmerka.out**) U prvom redu izlazne datoteke treba da se nalazi broj *t*, ukupan broj neprecrtanih polja osmosmerke. Zatim u narednih *t* redova treba ispisati vrednosti u tim poljima, onim redosledom kojim se javljaju ako se posmatra red po red osmosmerke, svaki od njih sleva na desno.

Primer:

osmosmerka.in	**osmosmerka.out**
3 4	4
2 5 1 4	5
3 0 1 5	1
4 9 2 4	5
5	4
3 3 8 3	
3 2 2 3	
2 3 7 2	
3 1 1 1	
2 1 3 2	



Objašnjenje.

Na slici je prikazano kako izgleda rešena osmosmerka. Može se videti da su neprecrtani brojevi redom 5, 1, 5, 4.

Napomena.

U 40% test primera postojaće samo smerovi 1, 3, 5, 7.

fajl: osmosmerka.cpp

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

using namespace std;

const int maxN = 100;

int n, m, k, res;
```

```
int tabla[maxN][maxN];
int ti, tj, s, l;
int di[8] = {-1, -1, 0, 1, 1, 1, 0, -1};
int dj[8] = {0, 1, 1, 1, 0, -1, -1, -1};

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

    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &(tabla[i][j]));

    res = n * m;

    scanf("%d", &k);
    for (int t = 0; t < k; t++) {
        scanf("%d%d%d%d", &ti, &tj, &s, &l);
        s--; ti--; tj--;
        for (int tl = 0; tl < l; tl++) {
            if (tabla[ti][tj] >= 0) {
               tabla[ti][tj] = -1;
               res--;
            }
            ti += di[s];
            tj += dj[s];
        }
    }

    printf("%d\n", res);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (tabla[i][j] >= 0)
               printf("%d\n", tabla[i][j]);

    return 0;
}
```

zadatak: Bešenje

Mirko i Slavko igraju igru bešenja. Igra započinje tako što Mirko zamišlja reč iz datog rečnika i zapisuje po jednu crticu za svako slovo zamišljene reči. Tada Slavko počinje sa pogađanjem: Slavko nabraja slova za koja misli da se mogu naći u zamišljenoj reči. Ukoliko Slavko pogodi, svaku crticu koja je pre zamenjivala to slovo Mirko zamenjuje sa njim. Ukoliko se Slavkovo slovo ne nalazi u reči, Mirko dobija jedan poen.

Primer toka igre.

Događaj	Stanje na papiru
Mirko je zamislio reč BAKICA.	- - - - - -
Slavko pita za pojavljivanje slova A	- A - - - A
Slavko pita za pojavljivanje slova E	- A - - - A
Slavko pita za pojavljivanje slova B	B A - - - A

Međutim, Mirko i Slavko su morali da prekinu igru u nekom trenutku. Kako Mirko nije želeo da otkrije svoju reč, Slavko je rešio da sam izračuna koliko reči iz rečnika mogu biti tražene reči. Pomozite Slavku i napišite program koji će rešiti Slavkove muke.

Ulaz:

(Ulazni podaci se nalaze u datoteci **besenje.in**.) U prvom redu ulazne datoteke nalaze se dva prirodna broja n i m ($1 \leq n \leq 100.000$, $0 \leq m \leq 25$) koji predstavljaju broj reči u rečniku i broj slova za koje je Slavko pitao, redom. U drugom redu ulaza nalazi se niz od m velikih slova engleskog alfabeta, razdvojenih po jednim znakom razmaka, koji predstavljaju slova za čije pojavljivanje je Slavko pitao. Sledeći red sadrži reč sastavljenu od velikih slova engleskog alfabeta i znaka "-", koja predstavlja stanje na papiru na kraju

igre. Narednih *n* linija sadrže po jednu reč iz rečnika. Reči su sastavljena od velikih slova engleskog alfabeta dužina ne većih od 30.

Izlaz:

(Izlazne podatke upisati u datoteku **besenje.out**) U prvom i jedinom redu ispisati broj reči koji zadovo avaju dato stanje i niz pogađanja.

Primer:

besenje.in	**besenje.out**
3 3 A C D --CA KUCA ZGRADA MACA	1

Objašnjenje.

Jedina reč koja zadovoljava stanje je reč "KUCA".

fajl: besenje.cpp

```
/*
  Author: Andreja Ilic, PMF Nis
  e-mail: andrejko.ilic@gmail.com
  Complexity: O(n * 30)
*/
#include<stdio.h>
#include<string.h>
#define MAX_LEN 35

int n, m, sol;
char state [MAX_LEN], word [MAX_LEN], pom;
bool guess [26];

  int main()
  {
    FILE *in = fopen ("besenje.in", "r");
    FILE *out = fopen ("besenje.out", "w");

    sol = 0;
    for (int i = 0; i < 26; i++)
      guess [i] = false;

    // Unos stanja i upitanih slova
    fscanf (in, "%d %d\n", &n, &m);
    for (int i = 0; i < m; i++)
    {
      fscanf (in, "%c", &pom);
      guess [pom - 'A'] = true;
      fscanf (in, "%c", &pom);
    }

    fscanf (in, "%s", &state);
    int stateLen = strlen(state);

    // Istipivanje svake reci pojedinacno
    for (int i = 0; i < n; i++)
    {
      fscanf (in, "%s", &word);
      int wordLen = strlen(word);

      // Jednakost duzina
      if (stateLen != wordLen)
        continue;

      // Pokapanje sa stanjem i upitanim slovima
      bool ok = true;
```

```
      for (int i = 0; i < stateLen; i++)
      {
        // Poklapanje sa slovima iz stanja
        if ((state [i] != '-') && (state [i] != word [i]))
        {
          ok = false;
          break;
        }

        // Ukoliko je nepoznato slovo u stanju, nije smelo biti upitano
        if ((state [i] == '-') && (guess [word [i] - 'A']))
        {
          ok = false;
          break;
        }
      }

      if (ok)
        sol++;
    }

    fprintf (out, "%d\n", sol);

    fclose(out);
    fclose(in);

    return 0;
  }
```

zadatak: Upiti

Dat je niz od n brojeva. Nad nizom se izvršavaju, jedan za drugim, m upita jednog od sledeća dva tipa:

- 1 i - seče trenutni niz posle i-tog elementa i zatim drugi deo niza (od (i + 1)-vog elementa do kraja) stavlja na početak, pri čemu se dobija novi niz od n elemenata. Npr. za niz 5 3 10 8 8, upit '1 2' daje 5 3 | 10 8 8 → 10 8 8 | 5 3 → 10 8 8 5 3.
- 2 j - treba odgovoriti koji se broj nalazi na j-toj poziciji u trenutnom nizu.

Odgovoriti na sve upite tipa 2.
Ulaz:
(Ulazni podaci se nalaze u datoteci **upiti.in**.) U prvom redu ulazne datoteke nalaze se 2 prirodna broja n i m koji predstavljaju, redom, broj elemenata niza i broj upita ($1 \leq n, m \leq 10^5$). Sledeći red sadrži n celih brojeva - elemente niza u datom redosledu (svi elementi su iz $[0, 10^9]$). Sledećih m redova sadrže upite već opisanog formata: 'a b' gde je $a \in \{1, 2\}$ i $1 \leq b \leq n$. Upiti se izvršavaju u redosledu datim na ulazu.
Izlaz:
(Izlazne podatke upisati u datoteku **upiti.out**) Za svaki upit tipa 2 iz ulazne datoteke ispisati u novi red izlazne datoteke odgovor na taj upit (odgovore ispisivati u odgovarajućem redosledu). Postojaće bar jedan upit tipa 2.
Primer:

upiti.in	upiti.out
5 4 5 3 10 8 8 2 3 1 2 1 4 2 3	10 8

Objašnjenje.
Na trećoj poziciji u početnom nizu je broj 10. Posle dva sećenja niz postaje 3 10 8 8 5. Na trećoj poziciji u ovom nizu je broj 8.
Napomena.

U 30% test primera biće $n, m \leq 10^3$.

fajl: upiti.cpp

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

const int MaxN = 100010;
const int MaxM = 100010;

int n, m, a, b, x[MaxN], sol[MaxM];
int startIndex;

int main() {

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

  fscanf(inFile, "%d%d", &n, &m);

  // indeksiramo niz od 0 jer je tako lakse
  for (int i = 0; i < n; i++)
    fscanf(inFile, "%d", &x[i]);

  startIndex = 0;

  for (int i = 0; i < m; i++) {
    fscanf(inFile, "%d%d", &a, &b);
    if (a == 1)
      startIndex = (startIndex + b) % n;
    else
      fprintf(outFile, "%d\n", x[(startIndex + b - 1) % n]);
  }

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

fajl: upiti.pas

```
const
  MaxN = 100010;
  MaxM = 100010;

var
  inFile, outFile : text;
  n, m, a, b, startIndex, i : longint;
  x : array[0..MaxN] of longint;
  sol : array[0..MaxM] of longint;

begin

  assign(inFile, 'upiti.in');
        assign(outFile, 'upiti.out');
  reset(inFile); rewrite(outFile);
  read(inFile, n, m);

  // indeksiramo niz od 0 jer je tako lakse
        for i := 0 to n - 1 do
    read(inFile, x[i]);

  startIndex := 0;

  for i := 1 to m do begin
```

```
          read(inFile, a, b);
          if (a = 1) then startIndex := (startIndex + b) mod n
                     else writeln(outFile, x[(startIndex + b - 1) mod n]);
        end;

  close(inFile);
  close(outFile);

end.
```

zadatak: Robot

Imamo robota kome se mogu zadati 3 različite komande:

- P - govori robotu da se pomeri za 1 metar unapred u pravcu u kome je okrenut,
- L - govori robotu da se okrene za 90° u levo u mestu (lokacija mu se ne menja),
- D - govori robotu da se okrene za 90° u desno u mestu (lokacija mu se ne menja).

Dato je *K* blokova od po *N* ovakvih komandi, blokovi se mogu izvršiti u bilo kom redosledu, dok se komande unutar jednog bloka moraju izvršiti u redosledu u kome su date. Svi blokovi se moraju izvršiti. Odrediti u koliko razliqitih redosleda izvršavanja blokova komandi će se robot na kraju naći nazad u istom mestu iz koga je pošao.
Ulaz:
(Ulazni podaci se nalaze u datoteci **robot.in**.) U prvom redu ulazne datoteke sa nalaze dva broja *K* i *N* (1 ≤ *K* ≤ 8, 1 ≤ *N* ≤ 100.000). U sledećih *K* redova nalazi se po *N* znakova od kojih je svaki P, L ili D.
Izlaz:
(Izlazne podatke upisati u datoteku **robot.out**) U prvom i jedinom redu izlaza ispisuje se traženi broj redosleda izvršavanja blokova komandi.
Primer:

robot.in	**robot.out**
3 4 PPLP PLPD LPLD	1

Objašnjenje.
Ako je robot na početku okrenut ka severu, dati su svi mogući redosledi izvršavanja blokova komandi i relativna pozicija robota na kraju u odnosu na početnu tačku.

- 1 - 2 - 3 [PPLP—PLPD—LPLD] 0 metara na sever, 2 metara na zapad
- 1 - 3 - 2 [PPLP—LPLD—PLPD] 0 metara na sever, 0 metara na istok
- 2 - 1 - 3 [PLPD—PPLP—LPLD] 2 metara na sever, 2 metara na zapad
- 2 - 3 - 1 [PLPD—LPLD—PPLP] 0 metara na sever, 4 metara na zapad
- 3 - 1 - 2 [LPLD—PPLP—PLPD] 2 metara na jug, 2 metara na zapad
- 3 - 2 - 1 [LPLD—PLPD—PPLP] 2 metara na jug, 4 metara na zapad

Samo u drugom slučaju kada se blokovi izvršavaju u redosledu 1 - 3 - 2 se robot na kraju nalazi u istoj tački odakle je počeo.
Napomena.
U 30% test primera biće *K* ≤ 6 i *N* ≤ 1000.

fajl: robot.cpp

```
#include <cstdio>
#include <cmath>
#include <algorithm>
#include <string>

using namespace std;
```

```
const int MAX_K = 8;

const int XS[] = {0, 1, 0, -1};
const int YS[] = {1, 0, -1, 0};

int k, n;
string blok[MAX_K];


int dx[MAX_K];
int dy[MAX_K];
int ds[MAX_K];

int perm[MAX_K];
bool ubacen[MAX_K];


void izracunajPomeraje()
{
    for(int i = 0; i < k; i++)
    {
        int x = 0;
        int y = 0;
        int s = 0;

        for(int j = 0; j < n; j++)
            if(blok[i][j] == 'P')
            {
                x += XS[s];
                y += YS[s];

            }

            else if(blok[i][j] == 'L')
            {
                s += 3;
                if(s >= 4)
                    s -= 4;
            }
            else
            {
                s += 1;
                if(s >= 4)
                    s -= 4;
            }

        dx[i] = x;
        dy[i] = y;
        ds[i] = s;

    }




}

bool proveri()
{
    int s = 0;
    int x = 0;
    int y = 0;
    for(int i = 0; i < k; i++)
    {
        int tx = dx[perm[i]];
```

```
        int ty = dy[perm[i]];
        for(int j = 0; j < s; j++)
        {
            int t = tx;
            tx = ty;
            ty = -1 * t;
        }

        x += tx;
        y += ty;
        s += ds[perm[i]];
        if(s > 3)
            s -= 4;
    }
    return (x == 0 && y == 0);


}

int permutacije(int t)
{

   if(t == k)
        if(proveri())
            return 1;
        else return 0;

    int broj = 0;
    for(int i = 0; i < k; i++)
        if(!ubacen[i])
        {
            perm[t] = i;
            ubacen[i] = true;
            broj += permutacije(t + 1);
            ubacen[i] = false;
        }
    return broj;
}


int main()
{

    FILE *fin = fopen("robot.in", "r");
    fscanf(fin, "%d %d", &k, &n);
    for(int i = 0; i < k; i++)
    {
        char c = getc(fin);
        while(c != 'P' && c != 'L' && c != 'D')
            c = getc(fin);

        blok[i] = "";
        while(c == 'P' || c == 'L' || c == 'D')
        {
            blok[i] += c;
            c = getc(fin);
        }

    }
    fclose(fin);



    izracunajPomeraje();
    for(int i = 0; i < k; i++)
        ubacen[i] = false;
    int r = permutacije(0);
```

```
    FILE *fout = fopen("robot.out", "w");
    fprintf(fout, "%d\n", r);
    fclose(fout);


}
```

zadatak: Magacin

Dobili ste posao magacionera u novom magacinu kutija! Magacin je ogroman i u njemu se nalazi *n* gomila kutija, u svakoj gomili su kutije poređane jedna na drugu. Međutim, neke gomile su previsoke a neke preniske, a vi volite red, pa ste rešili da prerasporedite neke kutije.
Na raspolaganju vam je viljuškar koji odjednom može da prenosi tačno *k* kutija (ni manje ni više). Prema tome, u jednom prebacivanju možete uzeti tačno *k* kutija sa neke gomile (koja ima bar *k* kutija) i prebaciti ih na bilo koju drugu gomilu. Želite da izvršite nekoliko prebacivanja tako da na kraju dobijete *n* što približnijih gomila, tj. da razlika broja kutija na najvećoj i najmanjoj gomili bude minimalna. Odredite tu razliku.
Ulaz:
(Ulazni podaci se nalaze u datoteci **magacin.in**.) U prvom redu ulazne datoteke nalaze se 2 prirodna broja *n* i *k* koji predstavljaju, redom, broj gomila i kapacitet viljuškara ($1 \leq n, k \leq 10^6$). Sledeći red sadrži *n* brojeva a_i razdvojenih razmakom - broj kutija na odgovarajućim gomilama ($1 \leq a_i \leq 10^9$).
Izlaz:
(Izlazne podatke upisati u datoteku **magacin.out**) U prvom i jedinom redu izlazne datoteke ispisati minimalnu moguću razliku između broja kutija na najvećoj i najmanjoj gomili posle optimalnog niza prebacivanja.
Primer:

magacin.in	**magacin.out**
5 7 20 3 8 19 29	6

Objašnjenje.
Ukoliko prebacimo 7 kutija sa prve na drugu gomilu, 7 kutija sa pete na drugu i 7 kutija sa pete na treću, dobijamo gomile 13 17 15 19 15 gde je razlika između najveće i najmanje 19 - 13 = 6. Nijedan drugi niz prebacivanja ne daje manju razliku.

fajl: magacin.cpp

```
#include <cstdlib>
#include <cstdio>
#include <memory.h>

const int MaxN = 1000100;
const int MaxK = 1000010;

int n, k, a[MaxN], r[MaxN], sorted[MaxN], c[MaxK];

int main() {

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

  fscanf(inFile, "%d%d", &n, &k);
  for (int i = 1; i <= n; i++)
    fscanf(inFile, "%d", &a[i]);

  long long sum = 0;
  for (int i = 1; i <= n; i++) {
    r[i] = a[i] % k;
    sum += a[i] / k;
```

```
  }

  // sortiramo niz ostataka counting sortom
  memset(c, 0, sizeof(c));
  for (int i = 1; i <= n; i++)
    c[ r[i] ]++;
  for (int i = 1; i < k; i++)
    c[i] = c[i] + c[i - 1];
  for (int i = n; i >= 1; i--) {
    sorted[ c[ r[i] ] ] = r[i];
    c[ r[i] ]--;
  }

  int position = sum % n;

  if (position == 0)
    fprintf(outFile, "%d\n", sorted[n] - sorted[1]);
  else
    fprintf(outFile, "%d\n", sorted[position] + k - sorted[position + 1]);

  fclose(inFile);
  fclose(outFile);

  return 0;
}
```

fajl: magacin.pas

```
const
  MaxN = 1000100;
  MaxK = 1000010;

var
  inFile, outFile : text;
  n, k, i, position : longint;
  a, r, sorted : array[0..MaxN] of longint;
  c : array[0..MaxK] of longint;
  sum : int64;


begin
        assign(inFile, 'magacin.in');
        assign(outFile, 'magacin.out');
        reset(inFile); rewrite(outFile);

  readln(inFile, n, k);
        for i := 1 to n do
          read(inFile, a[i]);

  sum := 0;
        for i := 1 to n do begin
          r[i] := a[i] mod k;
          sum := sum + (a[i] div k);
        end;

  // sortiramo niz ostataka counting sortom
  fillchar(c, sizeof(c), 0);

        for i := 1 to n do
          c[ r[i] ] := c[ r[i] ] + 1;
        for i := 1 to k - 1 do
          c[i] := c[i] + c[i - 1];
        for i := n downto 1 do begin
          sorted[ c[ r[i] ] ] := r[i];
          c[ r[i] ] := c[ r[i] ] - 1;
```

```
        end;

  position := sum mod n;

  if (position = 0) then
          writeln(outFile, sorted[n] - sorted[1])
  else
    writeln(outFile, sorted[position] + k - sorted[position + 1]);

  close(inFile);
  close(outFile);
end.
```