## Zadaci sa rešenjima Okružno takmičenje 2008.

### zadatak: Najslabija karika

U popularnom televizijskom kvizu "Najslabija karika" nekoliko takmičara redom u krug odgovaraju na postavljena pitanja. Prvi tačan odgovor donosi jedan dinar, dok svaki sledeći tačan odgovor u nizu donosi duplo više poena nego prethodni. Kada neko odgovori netačno - lanac se prekida i sledeći tačan odgovor donosi 1 dinar. Kada takmičar kaže "Banka", sav novac u trenutnom lancu tačnih odgovora se dodaje na ukupnu zagarantovanu sumu, a naredni tačan odgovor ponovo vredi 1 dinar. Ako takmičari u nizu daju 15 tačnih odgovora, a između nije bilo bankiranja - nagrada iznosi neverovatnih 1000000000 (milijardu) dinara, bez obzira da li je neko rekao "Banka" ili su pre toga takmičari osvojili nešto novca. Vama su na raspolaganju podaci iz prethodne emisije. Treba odrediti koliko dinara su takmičari zaradili.

**Ulaz:**

(Ulazni podaci se nalaze u datoteci **karika.in**) U prvom redu ulazne datoteke se nalazi niz znakova. Svaki od znakova je iz skupa {'0', 'X', 'B'}, (sva tri moguća znaka su velika slova, što znaci da prvi od tri znaka nije cifra nula, vec slovo O) i oni predstavljaju tačan odgovor, netačan odgovor i banku. Dužina niza znakova je manja ili jednaka od 100000.

**Izlaz:**

(Izlazne podatke upisati u datoteku **karika.out**) U izlaznu datoteku treba upisati broj dinara koji su takmičari zaradili.

**Primer 1:**

| **karika.in** | **karika.out** |
|---|---|
| OOOBXOXBXOOBOX | 10 |

**Objašnjenje.**

Posle bankiranja, takmičari redom dobijaju 7 + 0 + 3 = 10 dinara.

**Primer 2:**

| **karika.in** | **karika.out** |
|---|---|
| XOOBOOOOOOOOOOOOOOOOXOOB | 1000000000 |

**Objašnjenje.**

Takmičari imaju niz od 15 tačnih odgovora, pa zarađuju 1000000000 dinara.

### fajl: karika.cpp

```
/*
  Zadatak najlakse resavamo, tako sto ucitavamo redom karaktere iz ulaznog fajla i odmah ih
procesiramo.
  Ako naidjemo na karakter koji je razlicit od X, O ili B, prekidamo izvrsavanje i stampamo
resenje.
  U promenljivoj money cuvamo trenutnu zaradu igraca. Nju azuriramo samo kada neko kaze
banka i dodajemo
  1 + 2 + 4 + ... + 2^n = 2^(n + 1) - 1 dinara, gde je n duzina lanca tacnih odgovora.
Ukoliko je duzina
  trenutnog lanca 15, tada postavljamo osvojenu sumu novca na 1.000.000.000 i izlazimo iz
while petlje.
*/

#include <stdio.h>
long money, n;
char c;

int main ()
{
  FILE *in = fopen ("karika.in", "r");
  money = 0;
  n = 0;
  while (true)
  {
    c = fgetc (in);
    if (n == 15)
    {
      money = 1000000000;
```

```
      break;
    }
    if (c == 'X')
      n = 0;
    else if (c == 'O')
      n++;
    else if (c == 'B')
    {
      money += (1 << n) - 1;
      n = 0;
    }
    else
      break;
  }
  fclose (in);
  FILE *out = fopen ("karika.out", "w");
  fprintf (out, "%ld\n", money);
  fclose (out);
  return 0;
}
```

## zadatak: Igra

Na papiru je napisan broj $N$ ($1 \leq N \leq 10^7$). Dva igrača naizmenično rade sledeće: igrač koji je na potezu bira neki delilac broja koji je napisan na papiru (delilac ne sme biti 1 ili broj sa papira), računa količnik broja sa papira i izabranog delioca, briše broj koji se nalazi na papiru, i piše dobijeni količnik. Ukoliko igrač ne može da odigra svoj potez (tj. jedini delioci su 1 i broj sa papira), on je pobednik. Napisati program koji treba da učita niz brojeva i za svaki od tih brojeva odredi koji igrač pobeđuje u partiji na čijem početku se na papiru nalazi taj broj. Smatrati da oba igrača igraju optimalno (povlače najbolje poteze).

**Ulaz:**

(Ulazni podaci se nalaze u datoteci **igra.in**) U prvom redu ulazne datoteke se nalazi broj $M$ ($5 \leq M \leq 20$). U narednih $M$ redova se nalazi po jedan broj $N_i$ ($1 \leq N_i \leq 10^7$).

**Izlaz:**

(Izlazne podatke upisati u datoteku **igra.out**) U izlaznu datoteku treba upisati $M$ redova. U $i$-tom redu ispisati broj 1, ako prvi igrač pobeđuje kad je početni broj na papiru $N_i$, a 2, ako drugi igrač pobeđuje.

**Primer 1:**

| **igra.in** | **igra.out** |
|---|---|
| 5 | |
| 1 | |
| 4 | |
| 5 | |
| 10 | |
| 25 | |

## fajl: igra.cpp

```
#include <iostream>
#include <cstdio>
using namespace std;


int Solve(int n){
  int cnt = 0;

  for (long k = 2; cnt <= 2 && 2 * k <= n; k++){
    long tmp = n;
    while (tmp % k == 0){
      cnt++;
      tmp /= k;
    }
  }
  if (cnt == 2) return 2;
```

```
  else return 1;
}

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

  int test;
  fscanf(f, "%ld", &test);
  for (int i = 0; i < test; i++){
    int n;
    fscanf(f, "%ld", &n);
    fprintf(fout, "%ld\n", Solve(n));
  }

  fclose(f); fclose(fout);
  return 0;
}
```

## zadatak: Saksije

Baštovan Toma je rešio da ulepša svoje prostrano dvorište tako što će ga ukrasiti cvećem. On naime želi da najpre postavi $n$ saksija u red ($1 \leq n \leq 1000000$) i da u svakoj od ovih $n$ saksija bude $k$ kilograma zemlje ($0 \leq k \leq 1000$), a da potom u njima sadi raznorazno cveće. On je zato od firme koja se bavi prodajom saksija naručio $n$ saksija takvih da se u svakoj od tih $n$ saksija nalazi $k$ kilograma zemlje. Ta firma je bila toliko ljubazna da mu ne samo donese te saksije već i da ih postavi u red. Međutim, sutradan je Toma doživeo veliki šok. Neke saksije imaju više, a neke manje od $k$ kilograma zemlje. Toma se ipak malo smirio kada je primetio da je ukupna težina zemlje u svim saksijama $k \cdot n$ kilograma - pa se ipak ova greška da ispraviti. Toma želi da prebacivanjem zemlje iz jedne u drugu saksiju učini da u svakoj saksiji bude $k$ kilograma zemlje, a da se pri tome najmanje umori. Toma zna da za prebacivanje $x$ kilograma zemlje iz saksije koja je $i$-ta po redu u saksiju koja je $j$-ta po redu utroši $x \cdot |i - j|$ džula energije. Odrediti koliko je minimalno energije koju Toma mora potrošiti da bi ispravio grešku firme koja mu je prodala saksije.

**Ulaz:**

(Ulazni podaci se nalaze u datoteci **saksije.in**) U prvom redu tekstualne datoteke nalaze se redom celi brojevi $n$ i $k$. U drugom redu ove datoteke nalazi se $n$ celih brojeva koji su veći ili jednaki 0. Naime $i$-ti broj ($1 \leq i \leq n$) u drugom redu označava koliko se kilograma zemlje nalazi u $i$-toj saksije kada se gleda sa prozora Tomine spavaće sobe sa leva na desno.

**Izlaz:**

(Izlazne podatke upisati u datoteku **saksije.out**) U izlaznu datoteku treba upisati jedan broj koji je jednak $W$ mod 1000000000, gde je $W$ jednak minimalnom broju Džula koje Toma mora potrošiti da bi ispravio grešku firme od koje je kupio saksije.

**Primer:**

| **saksije.in** | **saksije.out** |
|---|---|
| 6 4<br>5 6 2 1 7 3 | 8 |

## fajl: saksije.cpp

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

void main()
{
  FILE *dat;
  dat=fopen("Saksije.in","r");
  int n,k;
  fscanf(dat,"%d%d",&n,&k);
  int *A,i;
  A=(int *)malloc(n*sizeof(int));
  for (i=0;i<n;i++)
    fscanf(dat,"%d",A+i);
  fclose(dat);
```

```
  int suma=0;
  for (i=0;i<n-1;i++)
  {
    if (A[i]>k)
    {
      suma=(suma+A[i]-k)%1000000000;
      A[i+1]+=A[i]-k;
    }
    else
    {
      suma=(suma+k-A[i])%1000000000;
      A[i+1]-=k-A[i];
    }
  }

  dat=fopen("Saksije.out","w");
  fprintf(dat,"%d\n",suma);
  fclose(dat);
}
```

## zadatak: Kamenčići

Mali Đurica je našao *N* kamenčića. Na svakom kamenčiću je napisano jedno veliko slovo engleske abecede (slova od 'A' do 'Z'). On je izmerio svaki kamenčić i utvrdio njegovu masu u gramima. Nakon toga je napisao reč dužine *M* koja se sastoji samo od velikih slova engleske abecede. Malog Đuricu zanima koja je najmanja masa kamenčića, koje treba da izabere, tako da kad ih nekako složi u jedan red dobije prethodno napisanu reč.
Pošto mali Đurica želi da se igra, ovaj problem je prepustio Vama.
**Ulaz:**
(Ulazni podaci se nalaze u datoteci **kamen.in**) U prvom redu ulazne datoteke se nalaze celi brojevi *N* ($1 \leq N \leq 50000$) i *M* ($1 \leq M \leq 50000$). U narednih *N* redova se nalaze po jedno slovo $s_i$ i jedan ceo broj $t_i$, gde $s_i$ predstavlja slovo na *i*-tom kamenčiću ($s_i$ će uvek biti veliko slovo engleske abecede), a $t_i$ ($1 \leq t_i \leq 50000$) masu *i*-tog kamenčića. Potom se učitava *M* redova koji sadrže po jedno slovo i to su redom slova reči koju je napisao mali Đurica.
**Izlaz:**
(Izlazne podatke upisati u datoteku **kamen.out**) U prvom redu datoteke ispisati najmanju ukupnu masu kamenčića koje je potrebno koristiti da bi se njihovim slaganjem dobila napisana reč. Ako je nemoguće dobiti napisanu reč ispisati -1.
**Ogranicenja:**

- $1 \leq N \leq 50000$
- $1 \leq M \leq 50000$
- $1 \leq t_i \leq 50000$
- $\texttt{'A'} \leq s_i \leq \texttt{'Z'}$
- vremensko ogranicenje za izvršavanje programa je 1 s.

**Primer 1:**

| **kamen.in** | **kamen.out** |
|---|---|
| 5 3<br>A 10<br>B 40<br>C 30<br>B 20<br>E 50<br>C<br>E<br>B | 100 |

**Primer 2:**

| **kamen.in** | **kamen.out** |
|---|---|

```
14 11
A 1
I 2
K 3
L 4
O 5
P 6
N 7
F 8
R 9
O 10
M 11
T 12
I 13
Z 14
I
N
F
O
R
M
A
T
I
K
A
```

**fajl: kamen.cpp**

```
#include <iostream>
#include <cstdio>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>

using namespace std;


int main(){
  ofstream fout("kamen.out");
  ifstream fin("kamen.in");
  int n, m;
  fin >> n >> m;
  vector <int> ic[26];
  int cnt[26];
  for (int i = 0; i < 26; i++){
    ic[i].clear();
    cnt[i] = 0;
  }
  int ret = 0, t;
  char cc;
  for (int i = 0; i < n; i++){
    fin >> cc >> t;
    ic[cc - 'A'].push_back(t);
  }
  for (int i = 0; i < 26; i++)
    sort(ic[i].begin(), ic[i].end());
  for (int i = 0; i < m; i++){
    fin >> cc;
    cnt[cc - 'A']++;
```

```
  }
  for (int i = 0; i < 26 && ret != -1; i++)
    for (int j = 0; j < cnt[i]; j++)
      if (ic[i].size() == j){
        ret = -1;
        break;
      }
      else
        ret += ic[i][j];
  fout << ret << endl;
  fout.close();
  fin.close();
  return 0;
}
```

## zadatak: Krtice

Dve grupe krtica su se našle na susednim stranicama pravougaone livade oivičene ogradom. Svaka krtica je pronašla po jednu rupu na ogradi. U trenutku kad je počela da pada kiša krtice su počele da kopaju tunele u pravcu normalnom na ogradu. Krtica kopa tunel dok ne dođe na već iskopan tunel (svi tuneli su na istoj dubini), a onda prestaje i počinje da se kreće već iskopanim tunelom. Ako je data brzina kojom svaka krtica kopa tunel, izračunati koliko dugo će svaka krtica kopati tunel pre nego što naiđe na već iskopan tunel. Ako neka krtica nikada neće naići na iskopan tunel, ispisati -1.

**Ulaz:**

(Ulazni podaci se nalaze u datoteci **krtice.in**) U prvom redu datoteke nalaze se dva cela broja: $n_1$ - broj krtica u prvoj grupi (one koje se nalaze na jednoj strani livade) i $n_2$ - broj krtica u drugoj grupi (krtice koje se nalaze na drugoj strani livade) ($0 < n_1, n_2 < 5000$). U sledećih $n_1$ redova se nalaze podaci o krticama iz prve grupe (u jednom redu su podaci o jednoj krtici). U svakom redu se nalaze po dva realna broja koji označavaju udaljenost krtice od ugla livade (na početku, pre nego što započne kopanje kanala) i brzinu kojom ta krtica kopa tunel. U sledećih $n_2$ redova se nalaze podaci o krticama iz druge grupe. Udaljenosti i brzine su između 0 i 1000000.

**Izlaz:**

(Izlazne podatke upisati u datoteku **krtice.out**) U svakom od $n_1 + n_2$ redova treba ispisati po jedan broj koji označava vreme koliko odgovarajuća krtica (iz ulazne datoteke) sama kopa tunel. Vremena odgovaraju krticama po redosledu kojim su navedene u ulazu, zaokruženo na dve decimale. Dozvoljena greška je 0.01.

**Primer:**

| **krtice.in** | **krtice.out** |
|---|---|
| 2 3<br>1 4<br>2 5.1<br>4.5 3.6<br>1 1<br>5 0.5 | 4.50<br>0.88<br>-1.00<br>1.00<br>-1.00 |

**Napomena:**

Nije moguće da dve krtice stignu u istom trenutku kopajući tunele.

## fajl: krtice.cpp

```
#include <algorithm>
#include <iostream>
#include <cstdlib>

#define TOLERANCIJA 0.01f

using namespace std;

typedef struct TKrtica {
  double brzina;
  double poz;
  double vreme;
  int id;
} Krtica;
```

```
bool porediPoPoziciji(const Krtica& left, const Krtica& right) {
  return left.poz < right.poz;
}

bool porediPoId(const Krtica& left, const Krtica& right) {
  return left.id < right.id;
}

int brojKrticaH;
int brojKrticaV;
Krtica* h;
Krtica* v;

void ulaz() {
  freopen("krtice.in", "r", stdin);
  freopen("krtice.out", "w", stdout);
  scanf("%d %d", &brojKrticaV, &brojKrticaH);
  v = (Krtica*) malloc(sizeof(Krtica) * brojKrticaV);
  h = (Krtica*) malloc(sizeof(Krtica) * brojKrticaH);

  for (int i=0; i<brojKrticaV; i++) {
    scanf("%lf %lf", &v[i].poz, &v[i].brzina);
    v[i].vreme = 0.0f;
    v[i].id = i;
  }

  for (int i=0; i<brojKrticaH; i++) {
    scanf("%lf %lf", &h[i].poz, &h[i].brzina);
    h[i].vreme = 0.0f;
    h[i].id = i;
  }
}

void obrada() {
  sort(h, h+brojKrticaH, porediPoPoziciji);
  sort(v, v+brojKrticaV, porediPoPoziciji);
  int nextV = 0;
  int nextH = 0;

  while (nextV < brojKrticaV || nextH < brojKrticaH) {
    if (nextV == brojKrticaV) {
      h[nextH++].vreme = -1.0;
    } else if (nextH == brojKrticaH) {
      v[nextV++].vreme = -1.0;
    } else {
      double tV = h[nextH].poz / v[nextV].brzina;
      double tH = v[nextV].poz / h[nextH].brzina;
      if (tV > tH) {
        v[nextV++].vreme = tV;
      } else if (tV < tH) {
        h[nextH++].vreme = tH;
      } else {
        fprintf(
        stderr,
        "Nepravilan slucaj! Krtice %d i %d se susrecu u istom trenutku %lf!",
        nextV, nextH, tV);
        h[nextH].vreme = v[nextV].vreme = -1.0;
      }
    }
  }
}

void izlaz() {
  sort(h, h+brojKrticaH, porediPoId);
  sort(v, v+brojKrticaV, porediPoId);
```

```
  for (int i=0; i<brojKrticaV; i++)
    printf("%lf\n", v[i].vreme);

  for (int i=0; i<brojKrticaH; i++)
    printf("%lf\n", h[i].vreme);
}

bool jednako(double a, double b) {
  return (a + TOLERANCIJA > b) && (b + TOLERANCIJA > a);
}

void tester(const char* testovi) {
  char ulaz[512];
  char izlaz[512];

  FILE* fTestovi = fopen(testovi, "r");
  int count = 1;

  while (!feof(fTestovi)) {
    if (fscanf(fTestovi, "%s %s", ulaz, izlaz) <= 0)
      break; // kraj testova
    printf("Testiram za: Ulaz %s Izlaz: %s Rezultat: ", ulaz, izlaz);
    try {
      // pripremi ulaz
      fclose(fopen("krtice.out", "w"));
      FILE* fUlaz = fopen(ulaz, "r");
      FILE* fKrticeIn = fopen("krtice.in", "w");

      int ch;
      while ((ch = fgetc(fUlaz)) != EOF) {
        fputc(ch, fKrticeIn);
      }

      fclose(fKrticeIn);
      fclose(fUlaz);
      // pokreni program
      system("krtice.exe");
      // otvori izlaz programa i tacan izlaz
      FILE* ispravno = fopen(izlaz, "r");
      FILE* izlazPrograma;

      try {
        izlazPrograma = fopen("krtice.out", "r");
      } catch (exception) {
        fclose(ispravno);
        throw "Greska kod otvaranja fajla";
      }
      // poredi izlaze programa
      while (true) {
        double a, b;
        int fa, fb;

        fa = fscanf(ispravno, "%lf", &a);
        fb = fscanf(izlazPrograma, "%lf", &b);

        if ( (fa <= 0 && fb > 0) || (fa > 0 && fb <= 0) ) {
          throw "Razlicit EOF";
        } else if (fa < 0 && fb < 0) {
          break;
        }

        if (!jednako(a,b)) {
          printf("\n%lf != %lf\n", a, b);
          throw "Neispravan izlaz";
        }
      }
      printf("OK %d\n", count++);
```

```
    } catch (const char* exception) {
      printf("FAILED\n");
      printf("\tZASTO: %s\n", exception);
    }
  }

  fclose(fTestovi);
}

int main(int argc, char** argv) {
  if (argc == 1) {
    ulaz();
    obrada();
    izlaz();
  } else {
    tester(argv[1]);
  }
  return 0;
}
```