## Zadaci sa rešenjima Izborno takmičenje 2011.

### zadatak: Prevara

Mika i Laza igraju igru kartama. Špil sadrži $n$ karata, pri čemu $i$-ta karta vredi $a_i$ poena. Oni izvlače karte u redosledu u kome se nalaze u špilu, počevši od dna. I Mika i Laza na početku imaju po 0 poena i svaki put kada neki igrač izvuče kartu, broj njegovih poena se povećava za vrednost te karte. Prvo izvlači Mika, a zatim izvlači onaj igrač koji u tom trenutku ima manje poena. Ako imaju jednak broj poena, izvlači Mika. Međutim, ono što Laza ne zna a Mika zna je početni raspored karata. Naravno, Mika hoće da vara tako što će preseći špil na nekom mestu i zatim početi igru. Pod sečenjem se podrazumeva standardno sečenje: izabere se proizvoljna pozicija u špilu i sve karte iznad te pozicije se prebace na dno špila, ne menjajući međusobni poredak. Odrediti koliko najviše poena može osvojiti Mika varanjem i na koliko načina može to postići.

**Ulaz:**

(Ulazni podaci se učitavaju iz datoteke **prevara.in**.) U prvom redu ulazne datoteke nalazi se prirodan broj $n \leq 10^5$ - broj karata u špilu. Sledeći red sadrži $n$ prirodnih brojeva $a_i$ - vrednosti karata ($a_i \leq 200$). Karte su date u redosledu od dna do vrha špila.

**Izlaz:**

(Izlazni podaci se ispisuju u datoteku **prevara.out**.) U prvom i jedinom redu izlazne datoteke ispisati dva broja razdvojena razmakom - maksimalan broj poena koje Mika može osvojiti nekim sečenjem i broj načina na koji on to može postići, redom.

**Primer 1:**

| **prevara.in** | **prevara.out** |
|---|---|
| 5<br>2 3 4 5 1 | 9 2 |

**Objašnjenje.**

Ako Mika preseqe posle treće ili posle četvrte karte (2 načina), on osvaja maksimalnih 9 poena.

**Napomena.**

U 10% test primera $n \leq 10^3$.

### fajl: prevara.cpp

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

using namespace std;

const int maxN = 100100;
const int maxD = 410;
const int c = 205;
const int minBesk = -1000000000;

int a[maxN];
int n;

int up[maxN][maxD];
int down[maxN][maxD];


void ucitajPodatke() {
    freopen("prevara.in", "r", stdin);
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%d", &(a[i]));
}

int fmax(int a, int b) {
    return (a > b) ? a : b;
}

void resi() {
    memset(up, 0, sizeof(up));
```

```
  memset(down, 0, sizeof(down));

    for (int j = 0; j < maxD; j++)
        up[n][j] = j;
    for (int i = n - 1; i >= 0; i--)
        for (int j = 0; j < maxD; j++)
            if (j <= c)
                up[i][j] = up[i + 1][j + a[i]];
            else
                up[i][j] = up[i + 1][j - a[i]];
    for (int j = 0; j < maxD; j++)
        down[0][j] = j;
    for (int i = 1; i <= n; i++)
        for (int j = 0; j < maxD; j++) {
            if (down[i - 1][j] <= c)
                down[i][j] = down[i - 1][j] + a[i - 1];
            else
                down[i][j] = down[i - 1][j] - a[i - 1];
        }
}

void sacuvajResenje() {
    int sumaSvih = 0;
    for (int i = 0; i < n; i++)
        sumaSvih += a[i];

    int total = 0, max = -500;
    for (int i = 0; i < n; i++) {
        int td = down[i][up[i][c]] - c;
        int vrednost = (sumaSvih - td) / 2 + td;
        if (vrednost > max) {
           max = vrednost;
           total = 1;
        } else if (max == vrednost) total++;
    }
    freopen("prevara.out", "w", stdout);
    printf("%d %d\n", max, total);
}

int main() {
    ucitajPodatke();
    resi();
    sacuvajResenje();
    return 0;
}
```

## zadatak: Metro

U jednom gradu postoji metro-linija sa *N* stanica, numerisanih brojevima od 1 do *N*. Metro ide pravolinijski, tj. sa stanice broj *i* možemo metroom doći do stanica broj *i* - 1 i *i* + 1 (sa stanice 1 možemo doći samo do stanice 2 a sa stanice *N* samo do stanice *N* -1). Svaka stanica ima svoj ulazni kod - jedan prirodan broj koji se menja svakog dana. Ako imamo kartu sa rednim brojem *k*, mi možemo doći na neku stanicu (sa leve ili desne strane) samo ako *k* deli ulazni kod te stanice. Nas zanima koliko stanica možemo posetiti za različite početne stanice i različite brojeve karata.
Preciznije, dato je *Q* upita jednog od sledeća dva tipa:

- 1 i *x* - promeniti ulazni kod *i*-te stanice na *x*,
- 2 i *k* - ispisati koliko stanica možemo posetiti ako smo trenutno u *i*-toj i imamo kartu sa rednim brojem *k*.

Odgovoriti na sve upite tipa 2. Broj naše karte će uvek deliti ulazni kod početne stanice.
**Ulaz:**

(Ulazni podaci se učitavaju iz datoteke **metro.in**.) U prvom redu ulazne datoteke nalaze se dva prirodna broja, $N$ i $Q$, koji predstavljaju, redom, broj stanica i broj upita ($1 \le N \le 200.000$, $1 \le Q \le 100.000$). Sledeći red sadrži $N$ prirodnih brojeva manjih od $2 \cdot 10^9$ - početne ulazne kodove. Sledićih $Q$ redova sadrže upite u gore opisanom formatu (izvršavaju se u datom redosledu). Za svaki od upita važi $1 \le i \le N$, $2 \le k$, $x \le 2 \cdot 10^9$.

**Izlaz:**

(Izlazni podaci se ispisuju u datoteku **metro.out**.) Za svaki upit tipa 2 ispisati odgovor (ceo broj) u posebnom redu izlazne datoteke. Na upite odgovarati u datom redosledu.

**Primer 1:**

| **metro.in** | **metro.out** |
|---|---|
| 6 4<br>2 25 20 30 19 5<br>2 3 5<br>2 4 6<br>1 5 100<br>2 3 5 | 3<br>1<br>5 |

**Objašnjenje.**

Ako smo na stanici broj 3 (ulazni kod 20) i imamo kartu broj 5, mi možemo posetiti tri stanice - 2, 3 i 4. Iako 5 deli ulazni kod stanice broj 6, mi do nje ne možemo doći jer ne možemo ući na stanicu sa ulaznim kodom 19. U drugom upitu nalazimo se na stanici broj 4 i ne možemo nigde otići. Posle promene ulaznog koda stanice broj 5, mi iz stanice broj 3 sa kartom broj 5 možemo posetiti 5 stanica - 2, 3, 4, 5 i 6.

**Napomena.**

U 20% test primera $Q \le 10^3$. U 40% test primera imamo uvek istu kartu tj. u svim upitima tipa 2 broj $k$ će biti isti (u ovim primerima je $Q$ neparno a u ostalim je parno).

**fajl: metro.cpp**

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

const int MaxN = 1 << 18;  // 2^18

int N, Q, X, Y, Z, sol;
int a[MaxN];

int gcd(int a, int b) {
  if (b == 0)
    return a;
  else
    return gcd(b, a % b);
}

struct SegmentTree {
  int N;
  int t[2 * MaxN];
  int l[2 * MaxN];
  int r[2 * MaxN];
  int offset;

  void init(int a[], int n) {
    N = n;
    offset = MaxN - 1;
    for (int i = 1; i < 2 * MaxN; i++) t[i] = 1;

    for (int i = 1; i <= n; i++)
      t[i + offset] = a[i];
    for (int i = offset + 1; i < 2 * MaxN; i++) {
      l[i] = i;  r[i] = i;
    }

    for (int node = offset; node > 0; node--) {
      t[node] = gcd(t[2 * node], t[2 * node + 1]);
```

```
      l[node] = l[2 * node];
      r[node] = r[2 * node + 1];
    }
  }

  void update(int pos, int val) {
    int i = pos + offset;
    t[i] = val;
    i = i / 2;
    while (i > 0) {
      t[i] = gcd(t[2 * i], t[2 * i + 1]);
      i = i / 2;
    }
  }

  int firstRight(int pos, int k) {
    int i = pos + offset;
    while (i > 1 && t[i] % k == 0) {
      if (pos + offset <= l[i / 2])
        i = i / 2;
      else
        i = r[i] + 1;
      if (i > offset + N) return N;
    }

    while (i <= offset) {
      if (t[2 * i] % k == 0)
        i = 2 * i + 1;
      else
        i = 2 * i;
    }
    return i - offset - 1;
  }

  int firstLeft(int pos, int k) {
    int i = pos + offset;
    while (i > 1 && t[i] % k == 0) {
      if (r[i / 2] <= pos + offset)
        i = i / 2;
      else {
        i = l[i] - 1;
        if (i <= offset) return 1;
      }
    }

    while (i <= offset) {
      if (t[2 * i + 1] % k == 0)
        i = 2 * i;
      else
        i = 2 * i + 1;
    }
    return i - offset + 1;
  }

} metroLine;

int main() {

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

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

  metroLine.init(a, N);
  for (int i = 0; i < Q; i++) {
```

```
    fscanf(inFile, "%d%d%d", &X, &Y, &Z);
    if (X == 1)
      metroLine.update(Y, Z);
    else {
      int r = metroLine.firstRight(Y, Z);
      int l = metroLine.firstLeft(Y, Z);
      fprintf(outFile, "%d\n", r - l + 1);
    }
  }

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

**fajl: metro.pas**

```
const
   MaxN = 1 shl 18;


var
   inFile, outFile : text;
   N, Q, X, Y, Z, sol : longint;
   a : array[0..MaxN] of longint;
   t, l, r : array[0..2*MaxN] of longint;
   N1, offset : longint;
   i, rs, ls : longint;


function gcd(a, b : longint) : longint;
begin
   if (b = 0) then gcd := a
        else gcd := gcd(b, a MOD b);
end;


procedure init(n : longint);
var
   i, node : longint;

begin
   N1 := n;
   offset := MaxN - 1;
   for i := 1 to 2 * MaxN - 1 do t[i] := 1;

   for i := 1 to n do
      t[i + offset] := a[i];

   for i := offset + 1 to 2 * MaxN - 1 do begin
      l[i] := i;
      r[i] := i;
   end;

   for node := offset downto 1 do begin
      t[node] := gcd(t[2 * node], t[2 * node + 1]);
      l[node] := l[2 * node];
      r[node] := r[2 * node + 1];
   end;
end;


procedure update(pos, val : longint);
var
   i : longint;
```

```
begin
   i := pos + offset;
   t[i] := val;
   i := i div 2;
   while (i > 0) do begin
      t[i] := gcd(t[2 * i], t[2 * i + 1]);
      i := i div 2;
   end;
end;



function firstRight(pos, k : longint) : longint;
var
   i : longint;
   found : boolean;

begin
   i := pos + offset;
   found := false;
   while ((i > 1) AND (t[i] MOD k = 0) AND (NOT found)) do begin
      if ((pos + offset) <= l[i DIV 2])
         then i := i DIV 2
      else
         i := r[i] + 1;
      if (i > offset + N1) then found := true;
   end;

   if (NOT found) then begin
      while (i <= offset) do
         if (t[2 * i] MOD k = 0) then i := 2 * i + 1
         else i := 2 * i;
   end;

   if (NOT found) then firstRight := i - offset - 1
   else firstRight := N1;
end;



function firstLeft(pos, k : longint) : longint;
var
   i : longint;
   found : boolean;

begin
   i := pos + offset;
   found := false;
   while ((i > 1) AND (t[i] MOD k = 0) AND (NOT found)) do begin
      if ((pos + offset) >= r[i DIV 2])
         then i := i DIV 2
      else begin
         i := l[i] - 1;
         if (i <= offset) then found := true;
      end;
   end;

   if (NOT found) then begin
      while (i <= offset) do
         if (t[2 * i + 1] MOD k = 0) then i := 2 * i
         else i := 2 * i + 1;
   end;

   if (NOT found) then firstLeft := i - offset + 1
   else firstLeft := 1;
end;
```

```
begin

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

  read(inFile, N, Q);
  for i := 1 to N do
          read(inFile, a[i]);

        init(N);

  for i := 1 to Q do begin

          read(inFile, X, Y, Z);
          if (X = 1) then update(Y, Z)
          else begin
             ls := firstLeft(Y, Z);
             rs := firstRight(Y, Z);
        writeln(outFile, rs - ls + 1);
     end;

        end;

  close(inFile);
  close(outFile);

end.
```

## zadatak: Kfree

Data je povezana mreža *n* gradova i neki od gradova su međusobno povezani dvosmernim putevima. Rastojanje između dva grada se određuje kao najmanji broj puteva koje treba preći da bi stigli iz jednog grada u drugi. Za skup gradova kažemo da je *k*-slobodan ako je udaljenost svaka dva grada iz tog skupa strogo veća od *k*. Vaš zadatak je da odredite što veći *k*-slobodan skup.
Za ovaj zadatak potrebno je da predate izlazne datoteke za 10 ulaznih datoteka **kfree.01.in**, **kfree.02.in**, ..., **kfree.10.in**, koji se nalaze u arhivi na računaru. Izlazi se pamte u datoteke **kfree.01.out**, **kfree.02.out**, ..., **kfree.10.out**, pri čemu broj u nazivu izlazne datoteke odgovara broju u nazivu ulazne datoteke.

**Ulaz:**
U prvom redu se nalaze prirodni brojevi *n*, *m* i *k*, koji redom predstavljju broj gradova, broj puteva i parametar $k \geq 1$. U sledećih *m* redova se nalaze po dva različita broja *a* i *b* ($1 \leq a, b \leq n$), koji predstavljaju puteve. Gradovi su oznaqeni brojevima od 1 do *n*.

**Izlaz:**
U prvom redu ispisati veličinu *k*-slobodnog skupa. U sledećem redu ispisati indekse gradova koji pripadaju *k*-slobodnom skupu.

**Primer 1:**

```
11 12 2     3
1 3         1 6 11
2 3
3 4
4 5
4 6
5 7
6 9
7 8
8 9
8 10
9 10
7 11
```

**Objašnjenje.**
Ako odaberemo gradove 1, 6 i 11 lako zaključujemo da su gradovi 1 i 11 na rastojanju 4, gradovi 1 i 6 na rastojanju 3 i gradovi 11 i 6 na rastojanju 4. Kako su sva rastojanja strogo veća od 2, dati skup je 3-slobodan.

**Ocenjivanje.**
Broj bodova za svaki test primer se određuje na sledeći način. Neka je *VAL* veličina vašeg *k*-slobodnog skupa, a *OTP* je maksimalna veličina *k*-slobodnih skupova među svim takmičarskim rešenjima, uključujući i komisijsko rešenje. Broj bodova za svaki test primer se računa na sledeći način: ako je $OTP = VAL$ dobijate 10 poena, ako je $OTP = VAL+1$ dobijate 8 poena, ako je $OTP = VAL+2$ dobijate 7 poena, ako je $OTP = VAL+3$ dobijate 4 poena, ako je $OTP = VAL + 4$ dobijate 3 poena, ako je $OTP = VAL + 5$ dobijate 1 poen i ako je $OTP \geq VAL + 6$ dobijate 0 poena.