Zadaci sa rešenjima Državno takmičenje 2012.

zadatak: Bombone

Mali Đole je u izlogu prodavnice slatkiša ugledao *n* bombona. Bombone su poređane u niz i svaka je predstavljena jednim prirodnim brojem - različiti brojevi označavaju da se radi o različitim vrstama bombona, a isti brojevi označavaju iste vrste bombona. On planira da zgrabi neke od bombona, eventualno plati i kasnije se zasladi.
Radi dobitka na brzini, on želi da zgrabi samo neki **uzastopni podniz** bombona, tj. da izabere indekse *i*, *j* ($1 \le i \le j \le n$) i da pokupi sve bombone koje se nalaze na pozicijama *i*, *i* + 1,..., *j* - 1, *j*. Takođe, pošto ne voli raznolikost, u tom podnizu **ne sme biti više od 3 različite vrste bombona** . Npr. podniz 12434 nije dobar jer sadrži 4 vrste bombona.
Odrediti na koliko načina mali Đole može da se zasladi.

Ulaz.
(Ulazni podaci se uqitavaju iz datoteke **bombone.in**) U prvom redu ulazne datoteke nalazi se jedan prirodan broj *n* koji predstavlja broj bombona u izlogu ($1 \le n \le 10^5$). U sledećem redu se nalaze *n* prirodnih brojeva (ne većih od 10^9) koji predstavljaju odgovarajiće vrste bombona.

Izlaz.
(Izlazni podaci se ispisuju u datoteku **bombone.out**) U prvom i jedinom redu izlazne datoteke ispisati broj uzastpnih podnizova bombona u kojima se pojavljuju najviše 3 različite vrste.

Primer 1.

bombone.in	**bombone.out**
5	13
1 2 4 3 4	

Objašnjenje. Imamo 13 mogućih podnizova sa traženom osobinom: (12434) (12434) (12434) (12434) (12434) (12434) (12434) (12434) (12434) (12434) (12434) (12434) i (12434)

Primer 2.

bombone.in	**bombone.out**
6	21
10 20 10 30 20 20	

Objašnjenje. Kako ukupno imamo samo 3 različite vrste bombona (10, 20 i 30), svaki uzastopni podniz (a njih ima 21) zadovoljava uslove.

Napomena.
U 20% test primera je n ≤ 100.
U 50% test primera je n ≤ 1000.

fajl: bombone.cpp

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

const int MaxN = 100010;

struct Candy
{
  int type;
  int num;
};

int n, differentNum;
int a[MaxN];
Candy B[4];
long long sol;

// da li postoji data vrsta bombona medju trenutnim
int find(int type)
{
  for (int i = 1; i <= 3; i++)
    if (B[i].type == type)
      return i;
  return -1;
}
```

```
// naci slobodno mesto da ubacimo trenutnu vrstu
int freeIndex()
{
  for (int i = 1; i <= 3; i++)
    if (B[i].num == 0)
      return i;
}

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

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

  differentNum = 0;
  for (int i = 1; i <= 3; i++)
  {
    B[i].type = 0;
    B[i].num = 0;
  }

  sol = 0;
  int left = 1;
  for (int right = 1; right <= n; right++)
  {
    int ind = find(a[right]); // da li vrsta a[right] vec postoji
    if (ind != -1) // ako da, povecati brojac
    {
      B[ind].num++;
    }
    else
    {
      if (differentNum == 3)  // ako ne postoji i imamo 3 razlicite vrste, moramo da
izbacujemo
      {
        while (differentNum == 3)
        {
          ind = find(a[left]);
          B[ind].num--;
          left++;
          if (B[ind].num == 0)
          {
            B[ind].type = 0;
            differentNum--;
          }
        }
      }
      ind = freeIndex();        // ubaciti tip a[right] na jedno od 3 slobodna mesta
      B[ind].type = a[right];
      B[ind].num = 1;
      differentNum++;
    }

    sol += (right - left + 1);
  }

  printf("%lld\n", sol);
  return 0;
}
```

fajl: bombone.pas

```
const
  MaxN = 100010;

var
  inFile, outFile : text;
  n, differentNum, i, left, right, ind : longint;
  Kind, Num : array[0..3] of longint;
  a : array[0..MaxN] of longint;
  sol : int64;

// da li postoji data vrsta bombona medju trenutnim
function find(x : longint) : longint;
begin
  find := -1;
  for i := 1 to 3 do
    if (x = Kind[i]) then find := i;
end;

// naci slobodno mesto da ubacimo trenutnu vrstu
function freeIndex : longint;
begin
  for i := 1 to 3 do
    if (Num[i] = 0) then freeIndex := i;
end;

BEGIN

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

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

  differentNum := 0;
  for i := 1 to 3 do begin
    Kind[i] := 0;
    Num[i] := 0;
  end;

  sol := 0;
  left := 1;
  for right := 1 to n do begin

    ind := find(a[right]); // da li vrsta a[right] vec postoji
    if (ind <> -1) then Num[ind] := Num[ind] + 1 // ako da, povecati brojac
    else begin
      if (differentNum = 3) then // ako ne postoji i imamo 3 razlicite vrste, moramo da
izbacujemo
        while (differentNum = 3) do begin
          ind := find(a[left]);
          Num[ind] := Num[ind] - 1;
          left := left + 1;
          if (Num[ind] = 0) then begin
            Kind[ind] := 0;
            differentNum := differentNum - 1;
          end;
        end;

      ind := freeIndex;
      Kind[ind] := a[right];
      Num[ind] := 1;
      differentNum := differentNum + 1;
    end;

    sol := sol + (right - left + 1);
  end;
```

```
  writeln(outFile, sol);
  close(inFile);
  close(outFile);

END.
```

zadatak: Ključ

Tajna Komisija vredno je radila na pripremi zadataka za Državno takmičenje i došlo je vreme da se zadaci pošalju na mesta održavanja takmičenja. Naravno, zadaci se čuvaju kao stroga tajna, pa se pre slanja šifruju dobro poznatim algoritmom nastalim u laboratorijama Tajne Komisije.
Algoritmom se šifruje tekst dužine $N \cdot M$ na sledeći način: Matrica sa N redova i M kolona popunjava se slovima teksta red po red, odozgo na dole. Zatim se nekim kolonama u matrici zamene mesta. Formalno, ključ kojim se šifruje je jedna permutacija brojeva od 1 do M. Ključ predstavljamo nizom (a_1, a_2 ,... , a_M), gde a_i predstavlja novu poziciju i-te kolone u matrici. Nakon primene ključa, šifrat (šifrovani tekst) se čita iz matrice istim redosledom kojim je originalni tekst unet u matricu.
Međutim, članovi komisije su malo zaboravni i ne sećaju se ključa koji koriste za šifrovanje zadataka. Naravno, kako je Tajna Komisija veoma odgovorna, ranije je izrađen plan i za ovu situaciju. U tajnom sefu, koji se nalazi negde u zgradi Tajne Komisije (čija lokacija je takođe tajna), sačuvali su jedan tekst sa veoma bitnom osobinom - šifrovanjem tog teksta bilo kojim ključem ne dobija se leksikografski veći šifrat od šifrata dobijenim šifrovanjem pomoću zaboravljenog ključa.
Ako vam je poznata matrica koja se šifruje i činjenica da se njenim šifrovanjem dobija leksikografski najveći mogući šifrat, odredite ključ.
Ulaz.
(Ulazni podaci se učitavaju iz datoteke **kljuc.in**) U ulaznoj datoteci se u prvom redu nalaze dva broja N i M($1 \leq N, M \leq 2.000$), broj redova i broj kolona matrice koja se šifruje, respektivno. U sledećih N redova nalazi se po M malih slova engleske abecede, koja predstavljaju matricu koja se šifruje.
Izlaz.
(Izlazni podaci se ispisuju u datoteku **kljuc.out**) U prvom i jedinom redu izlazne datoteke ispisati M brojeva razdvojenih razmakom, ključ kojim se šifruje matrica. Rešenje će biti jedinstveno.
Primer 1.

kljuc.in	**kljuc.out**
4 3 kom isi jar ulz	3 1 2

Objašnjenje. Primenom ključa 3 1 2 prva kolona se premešta na treću, druga kolona se premešta na prvu, a treća kolona se premešta na drugu poziciju. Čitanjem iz šifrovane matrice dobija se tekst *omksiiarjlzu*. Primenom bilo kog drugog ključa dobija se leksikografski manji šifrat.

fajl: kljuc.cpp

```
#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;

const int kMaxSize = 2001;

struct column
{
    char characters[kMaxSize];
    int originalIdx;
};

char matrix[kMaxSize][kMaxSize];
column columns[kMaxSize];
int n, m;
```

```
int key[kMaxSize];

bool cmp(const column& first, const column& second)
{
    return strcmp(first.characters, second.characters) > 0;
}

int main()
{
    freopen("kljuc.in", "r", stdin);
    freopen("kljuc.out", "w", stdout);
    scanf("%d %d", &n, &m);
    for (int i = 0; i < n; i++)
    {
        scanf("%s", matrix[i]);
    }
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            columns[i].characters[j] = matrix[j][i];
        }
        columns[i].characters[n] = 0;
        columns[i].originalIdx = i;
    }
    sort(columns, columns + m, cmp);
    for (int i = 0; i < m; i++)
    {
        key[columns[i].originalIdx] = i + 1;
    }
    for (int i = 0; i < m; i++)
    {
        printf("%d ", key[i]);
    }
    return 0;
}
```

fajl: kljuc.pas

```
const
  MaxN = 2012;

var
        inFile, outFile : text;
  n, m, i, j: longint;
  s : array[0..MaxN, 0..MaxN] of char;
  ind, sol : array[0..MaxN] of longint;

function less(a, b : longint) : boolean;
var
  i : longint;
  q : boolean;
begin
        q := false;
  for i := 1 to n do
  begin
    if (s[a][i] > s[b][i]) then q := true;
    if (s[a][i] < s[b][i]) then q := false;
    if (s[a][i] <> s[b][i]) then break;
  end;

  less := q;
end;

procedure QS(l, r : longint);
```

```
var
        i , j, mid, x, tmp2 : longint;
        tmp : char;
begin
  i := l; j := r;
        mid := (l + r) div 2;
        for x := 1 to n do s[m + 1][x] := s[mid][x];
        repeat
    while (less(i, m + 1)) do i := i + 1;
    while (less(m + 1, j)) do j := j - 1;
    if (i <= j) then
    begin
      for x := 1 to n do
      begin
        tmp := s[i][x]; s[i][x] := s[j][x]; s[j][x] := tmp;
      end;
      tmp2 := ind[i]; ind[i] := ind[j]; ind[j] := tmp2;
                        i := i + 1; j := j - 1;
    end;
  until (i > j);

  if (l < j) then QS(l, j);
  if (i < r) then QS(i, r);
end;

BEGIN

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

  readln(inFile, n, m);
  for i := 1 to n do
  begin
    for j := 1 to m do
      read(inFile, s[j][i]);
    readln(inFile);
  end;

  for i := 1 to m do
    ind[i] := i;
  QS(1, m);
  for i := 1 to m do
    sol[ ind[i] ] := i;

  for i := 1 to m - 1 do
    write(outFile, sol[i], ' ');
  writeln(outFile, sol[m]);

  close(inFile);
  close(outFile);

END.
```

zadatak: LepBroj

Za prirodan broj kažemo da je **lep** ukoliko je u njemu rastojanje između svake dve iste cifre bar 10. Rastojanje između dve cifre jednako je broju cifara između njih + 1 (recimo, u broju 12342, rastojanje između cifara '2' je 3).
Imamo početni broj od *n* cifara i želimo da od njega napravimo lep broj. U jednom potezu moguće je obrisati bilo koju cifru početnog broja i na njeno mesto upisati neku drugu cifru. Koliko je najmanje poteza potrebno da bi dobili lep broj?
Ulaz.

(Ulazni podaci se uqitavaju iz datoteke **lepbroj.in**) U prvom redu ulazne datoteke nalazi se jedan prirodan broj *n* koji predstavlja broj cifara početnog broja ($1 \leq n \leq 10^6$). U sledećem redu se nalazi početni broj. Između cifara ne postoji razmak i broj može imati vodeće nule.

Izlaz.

(Izlazni podaci se ispisuju u datoteku **lepbroj.out**) U prvom i jedinom redu izlazne datoteke ispisati najmanji broj poteza potrebnih da se od datog broja dobije lep broj.

Primer 1.

lepbroj.in	**lepbroj.out**
8 00346731	2

Objašnjenje. Ukoliko obrišemo cifru 0 na drugoj poziciji i umesto nje napišemo cifru 2 i obrišemo cifru 3 na sedmoj poziciji i napišemo cifru 5, dobijamo lep broj 02346751. Moguće je dobiti i druge lepe brojeve ali ne u manjem broju poteza.

fajl: lepbroj.cpp

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

const int MaxN = 1000010;

char s[MaxN];
int m[12][12];
int p[12];
bool mark[12];
int n, sol;

void compareSolution()
{
  int curr = 0;
  for (int i = 0; i < 10; i++)
    curr += m[ p[i] ][i];

  if (curr < sol)
    sol = curr;
}

void permutations(int k)
{
  if (k >= 10)
  {
    compareSolution();
  }
  else
  {
    for (int i = 0; i < 10; i++)
      if (!mark[i])
      {
        p[k] = i;
        mark[i] = true;
        permutations(k + 1);
        mark[i] = false;
      }
  }
}

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

  fscanf(inFile, "%d", &n);
  fscanf(inFile, "%s", s);
```

```
  memset(m, 0, sizeof(m));
  for (int i = 0; i < n; i++)
    m[s[i] - '0'][i % 10]++;

  for (int i = 0; i < 10; i++)
    for (int j = 0; j < 10; j++)
    {
      m[i][j] = (n / 10) - m[i][j];
      if (1 <= j + 1 && j + 1 <= n % 10) m[i][j]++;
    }

  sol = n + 1;
  memset(mark, false, sizeof(mark));
  permutations(0);

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

  fclose(inFile);
  fclose(outFile);

  return 0;
}
```

fajl: lepbroj.pas

```
const
  MaxN = 1000010;

var
        inFile, outFile : text;
  n, i, j, curr, sol : longint;
  m : array[0..12, 0..12] of longint;
  p : array[0..12] of longint;
  mark : array[0..12] of boolean;
       c : char;

procedure compareSolution;
begin
  curr := 0;
  for i := 0 to 9 do
    curr := curr + m[ p[i] ][i];

  if (curr < sol) then sol := curr;
end;

procedure permutations(k : longint);
var
  i : longint;
begin
  if (k >= 10) then compareSolution
  else
    for i := 0 to 9 do
      if (NOT mark[i]) then
      begin
        p[k] := i;
        mark[i] := true;
        permutations(k + 1);
        mark[i] := false;
      end;
end;

BEGIN

  assign(inFile, 'lepbroj.in');
```

```
  assign(outFile, 'lepbroj.out');
  reset(inFile); rewrite(outFile);

  fillchar(m, sizeof(m), 0);

  readln(inFile, n);
  for i := 1 to n do
  begin
    read(inFile, c);
    m[ord(c) - 48][i mod 10] := m[ord(c) - 48][i mod 10] + 1;
  end;

  for i := 0 to 9 do
    for j := 0 to 9 do
    begin
      m[i][j] := (n div 10) - m[i][j];
      if ((1 <= j + 1) and (j + 1 <= n mod 10)) then
        m[i][j] := m[i][j] + 1;
    end;

  sol := n + 1;
  fillchar(mark, sizeof(mark), false);
  permutations(0);

  writeln(outFile, sol);

  close(inFile);
  close(outFile);

END.
```

zadatak: Go

Perica i Jovica su nedavno saznali za drevnu kinesku igru Go i odlučili da odigraju partiju. Igra im se toliko svidela da nisu ni proučili sva pravila detaljno, već igraju po svojim, malo izmenjenim, pravilima.
Perica i Jovica igraju na tabli dimenzija *N x N*. Perica je beli igrač i njegovi kamenčići su bele boje, dok Jovica ima kamenčiće crne boje. Igrači igraju naizmenično, a u svakom potezu igrač, čiji je red, postavlja jedan svoj kamenčić na slobodno polje.
Svako polje je susedno sa najviše četiri polja - gore, dole, levo i desno. Polja na ivicama table imaju tri, a ćopkovi samo dva susedna polja. Prazno polje susedno nekom polju na kome se nalazi kamenčić predstavlja jednu slobodu tog kamenčića. Dva susedna kamenčića iste boje su povezana, a svi međusobno povezani kamenčići čine jednu grupu. Takođe, broj praznih susednih polja svih kamenčića jedne grupe predstavlja broj sloboda te grupe. Ukoliko, posle nekog poteza, neke protivničke grupe izgube sve svoje slobode, svi kamenčići tih grupa se uklanjaju. Ako postavljanje kamenčića dovodi do gubitka slobode i protivničkih i svojih grupa, uklanjaju se samo protivničke grupe. Međutim, potez može da dovede i do uklanjanja sopstvenih kamenčića ako se tim potezom gube slobode samo sopstvene grupe.
Partija se zahuktala i na potezu je Perica - beli igrač. Perica želi da odigra takav potez da nakon postavljanja belog kamenčića i eventualnog uklanjanja grupa, razlika belih i crnih kamenčića bude što veća. Odnosno, ako sa *b* označimo broj belih kamenčića, a sa *c* broj crnih kamenčića, Perica želi da izraz *b - c* bude što veći. Odredite kolika će ova razlika da bude ako Perica odigra najbolji potez.
Ulaz.
(Ulazni podaci se uqitavaju iz datoteke **go.in**) U ulaznoj datoteci se u prvom redu nalazi jedan broj *N* (1 ≤ *N* ≤ 1.000), koji predstavlja dimenzije table. U sledećih *N* redova nalazi se po *N* znakova 'B', 'C' ili '.', koji predstaljaju stanje table u trenutku kad je Perica na potezu. Svaki znak 'B' predstavlja po jedan beli kamenčić, svaki znak 'C' predstavlja po jedan crni kamenčić, dok znak '.' predstavlja prazno polje. Postojaće bar jedno prazno polje i na tabli neće biti nijedna grupa koja nema nijednu slobodu.
Izlaz.
(Izlazni podaci se ispisuju u datoteku **go.out**) U prvom i jedinom redu izlazne datoteke ispisati razliku belih i crnih kamenčića nakon najboljeg poteza.
Primer 1.

go.in	**go.out**
7 .BBBB..	16

```
BCCCCBC
BCB.CBC
BCCCCBB
BBBBB..
....B.C
CB.....
```

Objašnjenje. Na tabli se nalazi 19 belih i 14 crnih kamenčića. Od svih poteza, najbolji je postavljanje belog kamenčića na polje u trećem redu i četvrtoj koloni, što dovodi do toga da jedna bela i jedna crna grupa gube sve svoje slobode, pa se uklanja samo crna grupa od 10 kamenčića. Nakon ovog poteza, na tabli se nalazi 20 belih i 4 crna kamenčića, pa je razlika 16.

fajl: go.cpp

```
#include <cstdio>
#include <queue>

using namespace std;

const int kMaxSize = 1001;
const int kInfinity = 1000000000;

char table[kMaxSize][kMaxSize];
int visited[kMaxSize][kMaxSize];
int blackValues[kMaxSize][kMaxSize], whiteValues[kMaxSize][kMaxSize];
int componentSize[kMaxSize * kMaxSize];
int n, blackCount, whiteCount = 1, sol, bfsCount;
int bestBlack, bestWhite = kInfinity;
queue<pair<int, int> > bfsQueue;
int dr[] = {-1, 0, 1, 0};
int dc[] = {0, 1, 0, -1};

bool isConnected(int row, int column, int type)
{
    return row >= 0 && row < n && column >= 0 && column < n && table[row][column] == type;
}

void bfs(int row, int column, char stone)
{
    pair<int, int> liberty(-1, -1);
    int liberties = 0, groupSize = 0;
    bfsCount++;
    bfsQueue.push(make_pair(row, column));
    visited[row][column] = bfsCount;
    while (!bfsQueue.empty())
    {
        pair<int, int> current = bfsQueue.front();
        bfsQueue.pop();
        groupSize++;
        for (int i = 0; i < 4; i++)
        {
            if (visited[current.first + dr[i]][current.second + dc[i]] != bfsCount)
            {
                if (isConnected(current.first + dr[i], current.second + dc[i], stone))
                {
                    bfsQueue.push(make_pair(current.first + dr[i], current.second +
dc[i]));
                    visited[current.first + dr[i]][current.second + dc[i]] = bfsCount;
                }
                if (isConnected(current.first + dr[i], current.second + dc[i], '.'))
                {
                    visited[current.first + dr[i]][current.second + dc[i]] = bfsCount;
                    liberty = make_pair(current.first + dr[i], current.second + dc[i]);
                    liberties++;
                }
```

```
            }
        }
    }
    componentSize[bfsCount] = liberties;
    if (liberties == 1)
    {
        if (stone == 'C')
        {
            blackValues[liberty.first][liberty.second] += groupSize;
        }
        else
        {
            whiteValues[liberty.first][liberty.second] += groupSize;
        }
    }

}

int main()
{
    freopen("go.in", "r", stdin);
    freopen("go.out", "w", stdout);
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        scanf("%s", table[i]);
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (table[i][j] != '.')
            {
                if (visited[i][j] == 0)
                {
                    bfs(i, j, table[i][j]);
                }
                if (table[i][j] == 'C')
                {
                    blackCount++;
                }
                else
                {
                    whiteCount++;
                }
            }
        }
    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (table[i][j] == '.')
            {
                if (blackValues[i][j] > bestBlack)
                {
                    bestBlack = blackValues[i][j];
                }
                if (whiteValues[i][j] == 0)
                {
                    bool free = false;
                    for (int k = 0; k < 4; k++)
                    {
                        if (isConnected(i + dr[k], j + dc[k], '.'))
                        {
                            free = true;
```

```
                        }
                    }
                    if (free)
                    {
                        whiteValues[i][j] = -1;
                    }
                }
                if (blackValues[i][j] == 0)
                {
                    bool connects = false;
                    for (int k = 0; k < 4; k++)
                    {
                        if (isConnected(i + dr[k], j + dc[k], 'B') &&
                            componentSize[visited[i + dr[k]][j + dc[k]]] > 1)
                        {
                            connects = true;
                        }
                    }
                    if (connects)
                    {
                        whiteValues[i][j] = -1;
                    }
                    if (whiteValues[i][j] < bestWhite)
                    {
                        bestWhite = whiteValues[i][j] + 1;
                    }
                }
            }
        }
    }
    sol = bestBlack > 0 ? whiteCount - blackCount + bestBlack:
            whiteCount - blackCount - bestWhite;
    printf("%d\n", sol);
    return 0;
}
```

zadatak: Vojnici

Tajna Komisija je ove godine dobila zadatak od Vojske Srbije da napiše jedan mali program koji će pomoći da vojska napravi što bolju formaciju pešadije u raznim prilikama. Međutim, Tajna Komisija je prezauzeta oko pripreme državnog takmičenja, pa je zamolila vas da napišete taj program, a zauzvrat ćete dobiti poene na takmičenju.

Zadatak se sastoji u tome da N vojnika treba rasporediti u R redova, tačno po C vojnika u svakom redu, tako da je razmak između svaka dva susedna vojnika jednak i razmak između svaka dva reda jednak. Formalnije, vojnike možemo rasporediti u koordinatnu mrežu veličine R x C tako da se u svakoj tački (x, y) za ($1 \le x \le R$, $1 \le y \le C$) nalazi tačno jedan vojnik.

Formacija je bolja ako je međusobna preglednost vojnika bolja. Odnosno, ako označimo sa $V(x,y)$ broj vojnika koje vidi vojnik koji se nalazi u tački (x, y), onda suma

$S = \Sigma_{1 \le x \le R,\ 1 \le y \le C} V(x,y)$

treba da bude što veća. Dva vojnika vide jedan drugog ukoliko se na pravoj liniji između njih (između tačaka u kojima se nalaze) ne nalazi ni jedan drugi vojnik.

Za dati broj vojnika N ispisati R i C za koje je suma S najveća moguća, i tu sumu S.

Ulaz.

(Ulazni podaci se učitavaju iz datoteke **vojnici.in**) U prvom i jedinom redu ulazne datoteke se nalazi broj N ($1 \le N \le 300.000$), broj vojnika za koje treba napraviti formaciju.

Izlaz.

(Izlazni podaci se ispisuju u datoteku **vojnici.out**) U prvom i jedinom redu izlazne datoteke ispisati brojeve R , C i S, tražene brojeve iz zadatka, razdvojene jednim razmakom.

Ukoliko ima više rešenja, ispisati bilo koje.

Primer 1.

vojnici.in	**vojnici.out**
12	3 4 98

Objašnjenje. Moguće podele vojnika su 1 x 12, 2 x 6, 3 x 4, 4 x 3, 6 x 2 i 12 x 1. Za podele 3 x 4 i 4 x 3 suma *S* je najveća moguća, tj. 98.

Primer 2.

vojnici.in	**vojnici.out**
4	2 2 12

Primer 2.

vojnici.in	**vojnici.out**
5	5 1 8

Napomena.

U 70% test primera je $N \leq 100.000$

fajl: vojnici.cpp

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

const int MaxN = 1000010;

struct Matrix
{
  int rowNum;
  int colNum;
  bool m[MaxN];

  Matrix() {}

  void reset(int rows, int cols)
  {
    memset(m, false, sizeof(m));
    rowNum = rows;
    colNum = cols;
  }

  bool get(int i, int j)
  {
    return m[(i - 1) * colNum + (j - 1)];
  }

  void set(int i, int j, bool val)
  {
    m[(i - 1) * colNum + (j - 1)] = val;
  }
};

Matrix M;

long long count(int X, int Y)
{
  long long X1 = X, Y1 = Y;
  long long sol = 2 * (X1 * (Y1 - 1) + Y1 * (X1 - 1));
  int x1, y1;

  M.reset(X, Y);
  for (int x = 1; x < X; x++)
    for (int y = 1; y < Y; y++)
    {
      if (!M.get(x, y))
      {
        sol += 4*(X1 - x)*(Y1 - y);
        x1 = x; y1 = y;
        while (x1 < X && y1 < Y)
        {
          M.set(x1, y1, true);
```

```
          x1 += x; y1 += y;
        }
      }
    }

  return sol;
}

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

  int n = 0;
  scanf("%d", &n);

  long long sol = 0, curr = 0;
  int x, y;
  for (int i = 1; i * i <= n; i++)
  {
    if (n % i == 0)
    {
      curr = count(i, n / i);
      if (curr > sol)
      {
        sol = curr;
        x = i;
        y = n / i;
      }
    }
  }

  printf("%d %d %lld\n", x, y, sol);
  return 0;
}
```

fajl: vojnici.pas

```
const
  MaxN = 1000010;

var
  inFile, outFile : text;
  rowNum, colNum, x, y, n, i : longint;
  m : array[0..MaxN] of boolean;
  sol, curr : int64;

procedure resetMatrix(rows, cols : longint);
begin
  fillchar(m, sizeof(m), false);
  rowNum := rows;
  colNum := cols;
end;

function get(i, j : longint) : boolean;
begin
  get := m[(i - 1) * colNum + (j - 1)];
end;

procedure put(i, j : longint; val : boolean);
begin
  m[(i - 1) * colNum + (j - 1)] := val;
end;

function count(X, Y : longint) : int64;
```

```
var
  X1, Y1, sol : int64;
  i, j, dx, dy : longint;
begin
  X1 := X; Y1 := Y;
  sol := 2 * (X1 * (Y1 - 1) + Y1 * (X1 - 1));

  resetMatrix(X, Y);
  for i := 1 to X - 1 do
    for j := 1 to Y - 1 do
      if (NOT get(i, j)) then
      begin
        sol := sol + 4*(X1 - i)*(Y1 - j);
        dx := i; dy := j;
        while ((dx < X) and (dy < Y)) do
        begin
          put(dx, dy, true);
          dx := dx + i; dy := dy + j;
        end;
      end;

  count := sol;
end;

BEGIN

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

  readln(inFile, n);
  sol := 0;
  i := 1;

  while (i * i <= n) do
  begin
    if (n mod i = 0) then
    begin
      curr := count(i, n div i);
      if (curr > sol) then
      begin
        sol := curr;
        x := i; y := n div i;
      end;
    end;
    i := i + 1;
  end;

  writeln(outFile, x, ' ', y, ' ', sol);
  close(inFile);
  close(outFile);

END.
```

fajl: vojnici.alternativno.cpp

```
#include<stdio.h>
using namespace std;

int x,y;
long long d[1000555];
bool nzd[1000555];

int get(int r, int c) {
    return r*y+c;
```

```
}

int main() {

    freopen("vojnici.in", "r", stdin);
    freopen("vojnici.out", "w", stdout);

    int n,i,j,r,c,l,k,pr,pc,resx,resy;
    long long s,res;

    scanf("%d", &n);

    resx = 1;
    resy = n;
    res = (2*(n-2) + 2);

    for(i=2; i*i<=n; i++) if (n % i == 0) {

        x = i; y = n/i;

        for(j=0; j<=n; j++) { d[j]=0; nzd[j]=true; }

        for(r=1; r<x; r++) {
            for(c=1; c<y; c++) {
                //d[get(r,c)] = d[get(r-1,c)] + d[get(r,c-1)] - d[get(r-1,c-1)] + (
nzd[get(r,c)] == true );
                d[r*y+c] = d[(r-1)*y + c] + d[r*y + c-1] - d[(r-1)*y + c-1] + ( nzd[r*y+c]
== true );

                for(k=2; k*r < x && k*c < y; k++) {
                    nzd[ k*r*y + k*c ] = false;
                }
            }
        }

        s=0;
        for(r=0; r<x; r++) {
            for(c=0; c<y; c++) {
                s+=4;
                if (r==0) s--;
                if (r==x-1) s--;
                if (c==0) s--;
                if (c==y-1) s--;
                //s += d[get(r,c)] + d[get(x-r,c)] + d[get(r,y-c)] + d[get(x-r,y-c)];
                s += d[r*y+c] + d[(x-r)*y + c] + d[r*y + y-c] + d[(x-r)*y + y-c];
            }
        }

        if (s > res) {
            resx=x; resy=y;
            res=s;
        }
    }

    printf("%d %d %I64d\n", resx, resy, res);

    return 0;
}
```