# Задача A. Нескучный номер

| | |
|---|---|
| Имя входного файла: | *standard input* |
| Имя выходного файла: | *standard output* |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 512 мебибайт |

Докажем индукцией по $N$, что требуемый номер имеет вид $9898\ldots98$. Действительно, для $N = 1$ утверждение верно (так как все однозначные номера являются нескучными). Пусть утверждение верно для всех $N$, не превосходящих $K$. Рассмотрим $K+1$-значное нескучное число. Очевидно, что число $X$, образованное его первыми $K$ цифрами, также является наибольшим нескучным (иначе возьмём наибольшее нескучное число, допишем к нему какую-то цифру, отличную от последней, и получим заведомо большее нескучное $K+1$ значное число).

Согласно предположению индукции, число $X$ имеет вид $9898\ldots$. Если оно имеет чётную длину (заканчивается на 8), то самым большим $K+1$-значным числом с данными $K$ цифрами будет число $10X+9$, и это число является нескучным (добавленная девятка стоит рядом с восьмёркой). Если оно имеет нечётную длину, то число $10X+9$ не подходит (две девятки подряд). Тогда мы берём следующее по величине число — $10X+8$ и видим, что оно является нескучным (добавленная восьмёрка стоит рядом с девяткой), то есть мы получили наибольшее возможное $K+1$-значное нескучное число. Утверждение доказано.

Для решения задачи достаточно вывести строку из чередующихся девяток и восьмёрок длины $N$, начинающуюся с девятки.

## Код решения на C++

```
#include <stdio.h>

main()
{
    int i, N;
    scanf("%d", &N);
    for (i = 0; i < N; i++)
        printf("%d", 9 - i % 2);
    printf("\n");
    return 0;
}
```

# Задача B. МТСтрока

| | |
|---|---|
| Имя входного файла: | *standard input* |
| Имя выходного файла: | *standard output* |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 512 мебибайт |

Пусть у нас строка является палиндромом. Из определения палиндрома следует, что $S_i = S_{|S|-i-1}$ для всех $i$ от 0 до $|S|-1$. Если строка $S$ имеет чётную длину, то из данного равенства следует, что строка разбивается на пары одинаковых символов (первый и последний, второй и предпоследний и так далее), а раз так, то количество каждого из символов должно быть чётным. Если строка $S$ имеет нечётную длину $2k+1$, то при $i = k$ получится $S_k = S_k$, а при остальных значениях $i$ позиции символов не совпадают, то есть каждый элемент, кроме ровно одного, должен встретиться чётное количество раз, а этот один — нечётное. И наоборот, если у нас есть строка чётной длины, в которой каждый символ встречается чётное количество раз, мы можем составить палиндром, разбив символы на пары одинаковых, и добавляя один символ из пары в начало строки, а один — в конец; аналогично, для строки нечётной длины, в которой ровно один символ встречается нечётное количество раз, мы можем поставить этот символ в середину строки, после чего у нас все символы встречаются чётное количество раз, то есть их можно разбить на пары и добавлять в начало и в конец.

Так что если количество букв, встречающихся нечётное число раз, более единицы, то построить палиндром невозможно. Опишем алгоритм построения палиндрома для строк, в которой не более одной буквы встречается нечётное число раз.

Далее заметим, что как у палиндрома длины $2k$, так и у и палиндрома длины $2k+1$ последние $k$ символов полностью определяются первыми $k$, поэтому, расставив символы из заданной строки по возрастанию (сначала все буквы ‘`C`’, затем все буквы ‘`M`’, затем все буквы ‘`T`’) и отразив относительно середины (если длина нечётная, то буква в середине определяется однозначно, как единственная буква, встречающаяся нечётное число раз), мы получим правильный ответ.

Приведённое решение работает для строк, составленных из любых символов с кодами от 32 до 255 включительно.

## Код решения на C++

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

int vals[300];
unsigned char mystring[1000100];

main()
{
    int i, j = 0;
    unsigned char x, mid = 0;

    for (i=0;i<256;i++) vals[i] = 0;

    scanf ("%s",mystring);

    for (i=0;mystring[i]!=0;i++)
     vals[mystring[i]]++;

    for (i = 0; i < 256; i++) {
        if (vals[i] % 2) {
            if (mid) {
```

```
                printf("-1\n");
                exit(0);
            }
            mid = i;
        }
        vals[i] /= 2;
    }

    for (i = 0; i < 256; i++)
        for (j = 0; j < vals[i]; j++)
            printf("%c", (char)i);

    if (mid)
        printf("%c", (char) mid);

    for (; i >= 0; i--)
        for (j = 0; j < vals[i]; j++)
            printf("%c", (char)i);

    printf("\n");

    return 0;
}
```

# Задача C. Подключение

| | |
|---|---|
| Имя входного файла: | *стандартный ввод* |
| Имя выходного файла: | *стандартный вывод* |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 512 мебибайт |

Рассмотрим точку $Z$, которая находится симметрично главному офису относительно существующего кабеля (то есть имеет координаты $(x_2, -y_2)$). Пусть $O$ — точка установки маршрутизатора. Если $G$ находится на существующем кабеле, то $Z$ и $G$ совпадают и $OZ = OG$. Если $OG$ перпендикулярно прямой, то $GZ$ пересекает прямую в точке $O$, следовательно, $OG = OZ$ из свойств симметрии.

Если же треугольник $OGZ$ невырожден, то, так как $G$ и $Z$ по построению находятся на одном и том же расстоянии от кабеля, то у треугольника $OGZ$ высота, проведённая из точки $O$, делит отрезок $GZ$ пополам. Соответственно, треугольник $OGZ$ является равнобедренным и $OZ = OG$. (если точка $G$ лежит на прямой, то $OZ = OG$, так как точки $Z$ и $G$ совпадают, если отрезок $OG$ перпендикулярен от). Таким образом, достаточно найти кратчайшую траекторию между дата-центром и новой точкой; очевидно, что этой траекторией является отрезок прямой, их соединяющий (так как соответствующие точки находятся в разных полуплоскостях, то отрезок обязательно пересечёт существующий кабель). Таким образом, необходимо вычислить квадрат расстояния между точками $(x_1, y_1)$ и $(x_2, -y_2)$. Из теоремы Пифагора получаем, что искомый квадрат расстояния равен $(x_1 - x_2)^2 + (y_1 + y_2)^2$. Учитывая ограничения на координаты, ответ не превосходит $8 \cdot 10^{18}$; таким образом, вычисления можно производить в 64-битных целых (`long long int` в C/C++), после чего вывести ответ и строку, состоящую из десятичной точки и двадцати нулей.

Попытки решения задачи с помощью вещественной арифметики к успеху не приведут, так как накапливающаяся при вычислениях и выводе погрешность исключает точный вывод двадцати нулей. Ещё одной ошибкой может стать использование 32-битных целых вместо 64-битных.

## Код решения на C++

```
#include <iostream>
#include <cstdio>

using namespace std;

long long Sqr(long long x) {
    return x*x;
}

long long xa, ya, xb, yb;

int main() {
    cin >> xa >> ya >> xb >> yb;
    cout << Sqr(xb - xa) + Sqr(ya + yb) << ".";
    for (int i = 0; i < 20; ++i)
        cout << '0';
    cout << "\n";
    return 0;
}
```

# Задача D. Сумма десятичных дробей

| | |
|---|---|
| Имя входного файла: | *стандартный ввод* |
| Имя выходного файла: | *стандартный вывод* |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 512 мебибайт |

Идея решения: представим оба заданных числа в виде суммы целой части и правильной дроби, сложим целые части отдельно, дробные части отдельно, приведём сумму дробных частей к правильной дроби (возможно, увеличив сумму целых частей на 1), затем переведём её в десятичную запись.

При этом для вычислений хватит 64-битных целых чисел.

Докажем следующую лемму:

Пусть длина непериодической части $Y$ равна $k$, длина периодической части $Z$ равна $l > 0$. Тогда

$$X.Y(Z) = X + ((10^l - 1)Y + Z)/(10^k \cdot (10^l - 1))$$

Для начала покажем, что $0.(Z) = Z/(10^l - 1)$. Для этого умножим $0.(Z)$ на $10^l$, получим $0.(Z) \cdot 10^l = Z.(Z)$, вычитая $0.(Z)$ из каждой части, получим $0.(Z) \cdot (10^l - 1) = Z$, разделив обе части на $10^l - 1$ (так как периодическая часть непуста, то это всегда возможно), получаем, что $0.(Z) = Z/(10^l - 1)$.

Пусть десятичная запись имеет вид $X.Y(Z)$, где непериодическая часть $Y$ состоит из $k$ цифр, а периодическая часть $Z$ — из $l$ цифр. Тогда дробная часть $0.Y(Z)$ представляется как $Y.(Z)/10^k$, или же $(Y + (Z/(10^l - 1))/10^k = (Y \cdot (10^l - 1) + Z)/(10^k \cdot (10^l - 1)$. Лемма доказана.

Таким образом, знаменатель дроби равен $10^k \cdot (10^l - 1) \leqslant 10^{k+l}$ (равенство достигается, если периодическая часть отсутствует). Так как на дробную часть отводится не более 7 цифр, то знаменатель не превосходит $10^7$, следовательно, при сложении обычных дробей не будут использоваться числа, большие $10^7 \cdot 10^7 = 10^{14}$, что меньше, чем $2^{64}$.

После сложения сокращаем, если требуется, дробь и дробную часть полученной суммы переводим обратно в десятичную запись, например реализовав алгоритм «деления уголком» (делим числитель на знаменатель с остатком, затем умножаем остаток на 10 и снова делим, до тех пор, пока остаток не станет равным нулю — тогда дробь будет непериодической, или же пока какой-либо остаток не повторится в первый раз (для этого храним все предыдущие остатки в массиве).

Оценим максимальную длину требуемого массива (то есть максимальное количество знаков в дробной части). По предыдущим построениям знаменатель дроби является общим кратным $Q_1 = 10^{k_1} \cdot (10^{l_1} - 1)$ и $Q_2 = 10^{k_2} \cdot (10^{l_2} - 1)$. Построим какое-либо число вида $10^x \cdot (10^y - 1)$, которое делится как на $Q_1$, так и на $Q_2$. Отметим, что если $a$ делится на $b$, то $10^a - 1$ делится на $10^b - 1$. Действительно, тогда $a = bc$, и $10^a - 1 = 10^{bc} - 1 = (10^b)^c - 1 = (10^b - 1)((10^b)^{c-1} + \ldots + 10^b + 1)$. Факт, что если $a > b$, то $10^a$ делится на $10^b$, является тривиальным. Но тем самым мы получаем, что число $10^{max(k_1,k_2)} \cdot (10^{l_1 \cdot l_2} - 1)$ делится как на $Q_1$, так и на $Q_2$. Даже грубая оценка сверху на $k_i$ и $l_i$ даёт, что это число меньше $10^7 \cdot (10^{49} - 1)$, то есть в непериодической части будет не более 7 знаков, в периодической — не более 49, то есть хранить надо не более чем $7 + 49 + 1 = 57$ остатков, таким образом, и по памяти, и по времени предложенное решение проходит.

На самом деле оценку максимального знаменателя получившейся суммы можно улучшить до $10^{42} - 1$; желающие могут попробовать доказать этот факт самостоятельно.

## Код решения на C++

```
#include <queue>
#include <cassert>
#include <set>
#include <map>
#include <string>
#include <string.h>
#include <cmath>
#include <vector>
#include <iostream>
```

```
#include <cstdio>

using namespace std;

typedef long long LL;
typedef vector < int >VI;
typedef pair < int, int >PII;

LL gcd(LL a, LL b)
{
    while (b) {
LL r = a % b;
a = b;
b = r;
    }
    return a;
}

LL Pow(LL a, int y)
{
    LL res = 1;
    for (; y; y >>= 1) {
if (y & 1)
    res *= a;
a *= a;
    }
    return res;
}

struct F {
    typedef const F & CF;
    LL u, d;
   F(LL u = 0, LL d = 1):u(u), d(d) {
Norm();
    } void Norm() {
if (d < 0) {
    u = -u;
    d = -d;
}
LL g = gcd(u, d);
u /= g;
d /= g;
    }
    friend F operator +(CF a, CF b) {
return F(a.u * b.d + b.u * a.d, a.d * b.d);
    }
    friend F operator -(CF a, CF b) {
return F(a.u * b.d - b.u * a.d, a.d * b.d);
    }
    friend F operator *(CF a, CF b) {
return F(a.u * b.u, a.d * b.d);
    }
    friend F operator /(CF a, CF b) {
return F(a.u * b.d, a.d * b.u);
    }
    static F Parse(const string & s) {
vector < pair < LL, LL > >v(3);
int j = 0;
for (int i = 0; i < 3; i++) {
    while (j < s.size() && s[j] >= '0' && s[j] <= '9') {
```

```
v[i].first = v[i].first * 10 + s[j] - '0';
v[i].second++;
j++;
    }
    j++;
}


F res = F(v[0].first);
res = res + F(v[1].first) / F(Pow(10, v[1].second));
F k = F(Pow(10, v[2].second));
if (v[2].second > 0) {
    res += F(v[2].first) * k / (k - 1) / F(Pow(10, v[1].second + v[2].second));
}
return res;
    }
};


int printdec(F toprint)
{
    LL rests[100];
    char ans[100];
    LL rest;

    for (int i = 0; i < 100; i++)
rests[i] = 0;

    printf("%d", toprint.u / toprint.d);
    rest = toprint.u % toprint.d;
    if (rest == 0)
return (0); // integer? Leaving
    ans[0] = '.';
    int num = 1;
    rests[1] = rest;
    while (1) {
rest *= 10;
ans[num++] = '0' + rest / toprint.d;
rest = rest % toprint.d;
if (rest == 0) {
    ans[num] = 0;
    printf("%s", ans);
    return (0); // finite? Leaving
}
for (int j = 1; j < num; j++)
    if (rests[j] == rest) {
ans[num++] = ')';
ans[num++] = 0;
ans[j] += 12;
for (int i = 0; i < strlen(ans); i++) {
    if (ans[i] > '9') {
printf("(");
ans[i] -= 12;
    }
    printf("%c", ans[i]);
}
fflush(NULL);
return (0);
    }
rests[num] = rest;
```

```
    }
}

int main()
{
    string a;
    cin >> a;
    F res = F::Parse(a);
    cin >> a;
    F res1 = F::Parse(a);

    F sum = res + res1;


    printdec(sum);
    cout << endl;
    return 0;
}
```

# Задача E. Мобильное ориентирование

| | |
|---|---|
| Имя входного файла: | *standard input* |
| Имя выходного файла: | *standard output* |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 512 мебибайт |

Задача решается с помощью бинарного поиска по ответу. Сперва найдем координату $y$ искомой точки. Зафиксируем какой-нибудь $x$ для всех запросов, пока будем искать $y$ (можно заметить, что правильный $y$ будет найден вне зависимости от выбора этого $x$). Сначала сделаем произвольный запрос, а отрезок поиска положим равным отрезку от 0 до $10^9$ (максимально возможной координаты).

Пусть $m$ — текущая середина отрезка поиска, $prev$ — координата предыдущего запроса. Сообщается, какое число из $m$ и $prev$ ближе к правильному. Если $m$ ближе и $m > prev$, то серединой отрезка между $m$ и $prev$ можно ограничить левую границу бинпоиска (обозначим ее $s$). Если $m$ ближе и $m < prev$, то, наоборот, серединой ограничивается правая граница (обозначим $f$). В противном случае $prev$ ближе, если $m > prev$, то ограничивается $f$, иначе $s$.

Отрезок поиска будет, таким образом, все время сужаться и постепенно выродится в точку, дав нам правильную координату $y$. Нужно аккуратно следить за тем, чтобы $m$ не совпадал с $prev$, так как два одинаковых запроса подряд не несут информации.

Далее абсолютно аналогичным образом находится правильный $x$.

Время работы алгоритма может быть оценено сверху как $4 \cdot \log_2(MaxCoord)$, где $MaxCoord$ — максимально возможная координата.

## Код решения на C++

```
#include<bits/stdc++.h>

using namespace std;
typedef long long lint;
typedef long double ldb;
typedef unsigned long long uli;

#define F(i, l, r) for(auto i = l; i != r; i++)
#define Df(i, l, r) for(auto i = l; i != r; i--)
#define asg assign
#define all(x) x.begin(),x.end()
#define y1 adjf

int
ask (int x, int y)
{
  printf ("%d %d\n", x, y);
  fflush (stdout);
  int res;
  scanf ("%d", &res);
  return res;
}

const int maxcoord = 1000000000;

int l, r, ansx, ansy;
bool wasl = true;

int
```

```
main ()
{
  ask (0, 0);
  l = 0, r = maxcoord;
  while (r - l > 1)
    {
      if (wasl)
{
  int mid = (r + l) / 2;
  int res = ask (r, 0);
  if (res == 1)
    l = mid + 1;
  else
    r = mid;
  wasl = !wasl;
  ask (r, 0);
}
      else
{
  int mid = (r + l) / 2 + (r + l) % 2;
  int res = ask (l, 0);
  if (res == 1)
    r = mid - 1;
  else
    l = mid;
  wasl = !wasl;
  ask (l, 0);
}
    }
  if (l == r)
    {
      ansx = l;
    }
  else
    {
      ask (l, 0);
      if (ask (r, 0) == 1)
{
  ansx = r;
}
      else
{
  ansx = l;
}
    }
  ask (ansx, 0);
  wasl = true;
  l = 0, r = maxcoord;
  while (r - l > 1)
    {
      if (wasl)
{
  int mid = (r + l) / 2;
```

```
  int res = ask (ansx, r);
  if (res == 1)
    l = mid + 1;
  else
    r = mid;
  wasl = !wasl;
  ask (ansx, r);
}
      else
{
  int mid = (r + l) / 2 + (r + l) % 2;
  int res = ask (ansx, l);
  if (res == 1)
    r = mid - 1;
  else
    l = mid;
  wasl = !wasl;
  ask (ansx, l);
}
    }
  if (l == r)
    {
      ansy = l;
    }
  else
    {
      ask (ansx, l);
      if (ask (ansx, r) == 1)
{
  ansy = r;
}
      else
{
  ansy = l;
}
    }
  printf ("A %d %d\n", ansx, ansy);
  fflush (stdout);
  return 0;
}
```

# Задача F. Наибольший общий делитель

| | |
|---|---|
| Имя входного файла: | *standard input* |
| Имя выходного файла: | *standard output* |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 512 мебибайт |

В данной задаче требовалось найти НОД всех чисел, которые можно получить из данного числа $N$ путем всевозможных перестановок его цифр. Обозначим такое множество чисел как $S_N$, а искомый НОД — как $ans$.

1. Сразу рассмотрим тривиальный случай: если все цифры числа $N$ одинаковые, то ответом является само число $N$.

2. Пусть в числе $N$ существуют хотя бы две различные цифры $i$ и $j$, $i < j$. Тогда искомый НОД не превосходит 81. Действительно, рассмотрим два числа $N_{ji}$ и $N_{ij}$, которые состоят из цифр числа $N$ и совпадают по всем позициям, кроме последних двух, причем $N_{ji}$ заканчивается на цифры $j$ и $i$ (именно в таком порядке), а $N_{ij}$ — на цифры $i$ и $j$. Другими словами:

$$N_{ji} = b_1b_2...b_{k-2}ji,$$
$$N_{ij} = b_1b_2...b_{k-2}ij,$$

где $b_1$, $b_2$, ..., $b_{k-2}$, $i$, $j$ — цифры числа $N$. Так как $N_{ji}$ и $N_{ij}$ делятся на $ans$, то на $ans$ делится и $N_{ji} - N_{ij} = 10 * j + i - 10 * i - j = 9 * (j - i) \leqslant 9 * j \leqslant 81$; следовательно, $ans$ не превосходит 81, QED.

Таким образом, мы можем перебрать все числа от 81 до 1 и проверить для каждого из них, является ли это число делителем всех чисел множества $S_N$; максимальное подходящее число и будет числом $ans$. Но каким образом выполнить такую проверку?

3. Заметим, что некоторое число $t$ является общим (не обязательно наибольшим) делителем всех чисел множества $S_n$, если и только если выполняются следующие два условия:

   - 1) $N$ делится на $t$;
   - 2) для любых двух цифр $i, j, 0 \leqslant i < j \leqslant 9$, присутствующих в числе $N$, верно, что число $9 * (j - i)$ делится на $t$.

Необходимость этих двух утверждений следует из очевидных соображений и рассуждений пункта 2. Достаточность же можно доказать следующим образом. Будем говорить, что число $N_2$ можно получить из $N_1$ с помощью транспозиции, если $N_1$ и $N_2$ отличаются друг от друга перестановкой ровно одной пары соседних цифр; другими словами, $N_1$ и $N_2$ можно представить в следующем виде:

$$N_1 = b_1b_2...b_{l-3}ijb_lb_{l+1}...b_k,$$
$$N_2 = b_1b_2...b_{l-3}jib_lb_{l+1}...b_k,$$

где $b_1$, $b_2$, ..., $b_{l-3}$, $i$, $j$, $b_l$, ..., $b_k$ — цифры числа $N$. Заметим, что $N_1 - N_2 = 9 \cdot (i - j) \cdot 10^{k-l+1}$. Отсюда и из утверждения 2 следует, что $N_1 - N_2$ делится на $t$, а $N_1$ и $N_2$ имеют одинаковый остаток при делении на $t$. Так как любое число множества $S_N$ можно получить из числа $N$ с помощью транспозиций, то все числа множества $S_N$ имеют такой же остаток от деления на $t$, что и $N$; утверждение 1 завершает доказательство.

Таким образом, чтобы узнать, является ли $t$ общим делителем чисел множества $S_N$, достаточно проверить условия 1 и 2. Детали реализации оставляем читателю.

## Код решения на C++

```
#include <iostream>
#include <stdio.h>
#include <string>
#include <vector>
#include <cstring>
using namespace std;

template < typename T > T abs(T a)
{
    if (a < 0)
return -a;
    return a;
}

string s;
bool ok;
int i, j;
bool raz[10];
int x;
int ost;

int main()
{
    cin >> s;
    ok = true;
    for (i = 1; i < s.length(); i++)
if (s[i] != s[i - 1]) {
    ok = false;
    break;
}
    if (ok)
cout << s << endl;
    else {
for (i = 0; i < s.length(); i++)
    for (j = i; j < s.length(); j++)
raz[abs((int) s[i] - (int) s[j])] = true;

for (x = 81; x >= 1; x--) {
    ok = true;
    for (i = 1; i < 10; i++)
if (raz[i])
    if ((9 * i) % x) {
ok = false;
break;
    }
    if (!ok)
continue;

    ost = 0;
    int pow = 1;
    for (i = s.length() - 1; i >= 0; --i) {
ost += (s[i] - '0') * pow;
pow *= 10;
```

```
ost %= x;
pow %= x;
    }
    if (ost)
ok = false;
    if (ok)
break;
}
cout << x << endl;
    }
    return 0;
}
```

# Задача G. Социальная сеть

| | |
|---|---|
| Имя входного файла: | *стандартный ввод* |
| Имя выходного файла: | *стандартный вывод* |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 512 мебибайт |

В задаче требуется построить граф с $N$ вершинами и $M$ рёбрами, в котором степень каждой вершины не менее, чем $K$.

Найдём оценку на $M$ по $N$ и $K$. Так как из каждой вершины выходит не менее $K$ рёбер, то всего мы имеем не менее $NK/2$ рёбер (делим пополам, так как каждое ребро соединяет ровно две точки). Тем самым $M \geqslant NK/2$.

Требуемый в задаче граф будем строить следующим образом: сначала построим граф с $(N \cdot K + 1)/2$ рёбрами, удовлетворяющий требованиям задачи, а затем оставшиеся рёбра будем добавлять произвольным образом, соединяя точки, которые ещё не соединены.

Приведём один из возможных способов построения.

Реализуем способ построения графа, содержащего $N$ вершин, степень которого равна двум. Для этого будем соединять первую вершину с $(l+1)$-й, вторую с $(l+2)$-й и так $N$ раз (если $l+i > N$, то берём остаток от деления $l+i$ на $N$). Для $l < N/2$ этот алгоритм работает и даёт для различных $l$ попарно непересекающиеся наборы рёбер. Обозначим этот способ за $cycle(l)$ (вообще говоря, в результате работы $cycle(l)$ не обязательно получится один цикл: может получиться набор циклов, если $l$ не взаимно просто с $N$). Усовершенствуем $cycle(l)$ следующим образом: пусть функции `int advcycle(l,t)` на вход дополнительно подаётся оставшееся количество рёбер $t$, тогда при $t \geqslant N$ функция работает так же, как $cycle(l)$, и возвращает $t - N$, при $t < N$ вершины соединяются не $N$, а $t$ раз (то есть строится часть цикла) и возвращается 0 (таким образом реализуется соединение «оставшихся» точек).

В случае чётного $K$ просто присваиваем $t = M$ и повторяем `t=advcycle(l,t)`, начиная с $l = 1$ и увеличивая $l$ на 1 до тех пор, пока $t$ не станет равным нулю; в случае же нечётного $K$ предварительно надо сделать дополнительные построения.

В случае нечётного $K$ и чётного $N$ сначала соединяем первую вершину с $(N/2+1)$-й, вторую с $(N/2+2)$-й и так далее $N/2$ раз. Получаем граф с $N/2$ рёбрами, степень всех вершин которого равна 1. Дополнение данного графа до полного представляет собой объединение $cycle(l)$; таким образом, после этого просто присваиваем $t$ значение $M - N/2$, $l = 1$ и выполняем `t=advcycle(l,t)` до тех пор, пока $t$ не станет равным нулю.

В случае нечётного $K$ и нечётного $N$ на первом шаге используется следующая модификация `advcycle`: обозначим за $D$ остаток от деления $M$ на $N$ (то есть $M = Nx + D$, $0 \leqslant D < N$), после чего $D$ раз повторяем следующее действие: соединяем $i$-ю вершину с $(i+1)$-й, затем увеличиваем $i$ на 2 (если значение $i$ и/или $i+1$ стало больше $N$, то берём остаток от деления соответствующего числа на $N$).

После этой операции останется $M - D$ вершин; по выбору $D$ $M - D$ делится на $N$, в силу чего пример строится повторением $cycle(l)$ соответствующее количество раз, начиная с $l = 2$, если $D \neq 0$, и с $l = 1$ в противном случае.

Покажем, что данный способ работает.

Если остаток меньше, чем $D = N/2$, то $M < xN + N/2$; преобразовав к $M < (2x+1)N/2$, получаем, что $K < 2x + 1$. Так как $K$ нечётно, то $K < 2x$, то есть при построении $x$ циклов требование задачи будет выполнено при повторении $cycle(l)$, а на первом шаге будут расставлены «оставшиеся» рёбра.

Если же $D \geqslant (N+1)/2$, то после $(N+1)/2$ шагов будет построен граф с $(N+1)/2$ рёбрами, степень всех вершин которого не меньше 1; после повторения $cycle(l)$ $x$ раз степень каждой вершины не будет менее $2x+1$. Построением, аналогичным проведённому в предыдущем абзаце, доказывается, что $K \leqslant 2x + 1$; тем самым требование задачи выполняется.

При реализации похожих идей распространённой ошибкой оказывается следующая: делать дополнительные построения не в начале, а в конце (точнее даже после того, как $cycle(1)$ построен). Например, ошибка возникает при построении графа с 6 вершинами, 9 рёбрами и минимальной степенью 3 следующим образом: сначала строится $cycle(1)$, затем в $cycle(2)$ берутся вершины «через одну» (детальный разбор этого примера предлагается проделать самостоятельно, представив вершины графа, например, как вершины правильного шестиугольника).

## Код решения на C++

```
#include <stdio.h>

int loop(int n, int step, int rest)
{
    int i, l = n;
    if (rest < l)
l = rest;
    for (i = 0; i < l; i++)
printf("%d %d\n", i + 1, (i + step) % n + 1);
    return (rest - n);
}

main()
{
    int m, n, k, i;

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

    if (n % 2 == 0) {
if (k % 2) {
    for (i = 0; i < n / 2; i++)
printf("%d %d\n", i + 1, (i + n / 2) % n + 1);
    m -= n / 2;
}
for (i = 1; m > 0; m = loop(n, i++, m));
    } else {
i = 1;
if (m % n) {
    for (i = 0; i < m % n; i++)
printf("%d %d\n", (2 * i) % n + 1, (2 * i + 1) % n + 1);
    i = 2;
    m -= m % n;
}
for (; m > 0; m = loop(n, i++, m));
    }
    return (0);
}
```

# Задача H. Волшебный бублик

| | |
|---|---|
| Имя входного файла: | *standard input* |
| Имя выходного файла: | *standard output* |
| Ограничение по времени: | 20 секунд |
| Ограничение по памяти: | 512 мебибайт |

Из каждой позиции существуют всего 16 возможных ходов. Решение «в лоб» ведёт к проверке $16^{12}$ последовательностей ходов, что не укладывается в оганичения по времени.

Для решения этой задачи используем метод «meet-in-the-middle». Построим все последовательности из 6 и менее ходов, начиная с данной позиции, после чего для каждой достижимой позиции запишем минимальное количество ходов, которое понадобилось для её достижения.

Аналогично построим все последовательности из 6 и менее ходов, начиная с собранной головоломки. Для каждой достижимой позиции также запишем минимальное количество ходов.

Рассмотрим все позиции, помеченные как достижимые как с начала, так и с конца, и выберем среди них ту, сумма ходов для которой минимальна.

В этом случае количество операций будет равно $16^6 \cdot \log(16^6)$, что в ограничения по времени уже укладывается.

## Код решения на C++

```
#include <cstdio>
#include <queue>
#include <map>
using namespace std;
typedef unsigned long long ST;
const ST T = (0x55 << 16) | (0xAA << 8) | (0xFF);

void
relax (map < ST, int >&D, queue < ST > &Q, ST x, int nd)
{
  int &od = D[x];
  if (!od)
    {
      od = nd;
      if (nd <= 6)
Q.push (x);
    }
}

void
go (ST S, map < ST, int >&D)
{
  queue < ST > Q;
  Q.push (S);
  D[S] = 1;
  while (!Q.empty ())
    {
      ST x = Q.front ();
      Q.pop ();
      int d = D[x];
      for (int i = 0; i < 4; ++i)
{
  ST mask = 0x000000FFULL << (8 * i);
  ST b = x & mask;
```

```
  relax (D, Q, (x & ~mask) | ((b >> 2 | b << 6) & mask), d + 1);
  relax (D, Q, (x & ~mask) | ((b >> 6 | b << 2) & mask), d + 1);
  mask = 0x03030303ULL << (2 * i);
  b = x & mask;
  relax (D, Q, (x & ~mask) | ((b >> 8 | b << 24) & mask), d + 1);
  relax (D, Q, (x & ~mask) | ((b >> 24 | b << 8) & mask), d + 1);
}
    }
}

int
main (void)
{
  ST S = 0;
  for (int i = 0; i < 4; ++i)
    {
      char row[10];
      scanf ("%s", row);
      for (int j = 0; j < 4; ++j)
{
  ST b = 0;
  switch (row[j])
    {
    case 'Y':
      ++b;
    case 'B':
      ++b;
    case 'G':
      ++b;
    }
  S = (S << 2) | b;
}
    }
  map < ST, int >distS, distT;
  go (S, distS);
  go (T, distT);
  int ans = 20;
for (auto a:distS)
    if (a.first == T)
      ans = a.second - 1;
    else
      {
auto b = distT.find (a.first);
if (b != distT.end ())
  ans = min (ans, a.second + b->second - 2);
      }
  printf ("%d\n", ans);
  return 0;
}
```