Разбор задач

32 Районная олимпиада школьников Красноярского края по информатике, 9-11 классы

(Муниципальный этап ВОШ по информатике)

6 декабря 2018 г.

ID	Задача	Тема	%
1604	A. Максимальная скорость	Задачи для начинающих	10
1583	B. Минимальный циклический сдвиг	Задачи для начинающих	15
1591	C. Уравнение	Простая математика	37
1585	D. Дачный участок	Структуры данных	40
1607	E. Геометрическая прогрессия	Целочисленная арифметика	52
1602	F Сумма в ромбе	Динамическое программирование	55

Автор разбора: Беляев Сергей Николаевич

Средняя сложность задач: 34%

Задача A. Максимальная скорость

(Время: 1 сек. Память: 16 Мб Баллы: 100)

Известно, что некоторый автомобиль начал свое движение со скоростью v_0. После чего он двигался t_1 секунд с ускорением a_1, а затем – t_2 секунд с ускорением a_2. Необходимо найти максимальную скорость автомобиля на описанном выше промежутке его движения.

Входные данные

Входной файл INPUT.TXT содержит целые числа v_0, t_1, a_1, t_2, a_2 ($0 \le v_0 \le 10^6$, $0 < t_1, t_2 \le 10^6$, $|a_1| \le 10^6$, $|a_2| \le 10^6$). Гарантируется, что на данном промежутке движения автомобиль не двигался в обратном направлении.

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное целое число – ответ на задачу.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1 2 3 4 5	27
2	1 2 3 4 -1	7

Система оценки

Решения, работающие для значений во входных данных, не превосходящих 1000 по абсолютной величине, будут оцениваться в 75 баллов.

Задача A. Максимальная скорость

$\mathbf{v_0}$ – скорость в начале пути
$\mathbf{t_1}$, $\mathbf{a_1}$ – время и ускорение при первом равноускоренном движении
$\mathbf{t_2}$, $\mathbf{a_2}$ – время и ускорение при втором равноускоренном движении

Безусловно, что всем известна формула из основ механики, которая позволяет вычислять скорость $\mathbf{v_t}$ после равноускоренного движения материальной точки по прямой с ускорением **a** в течении времени **t** и при начальной скорости $\mathbf{v_0}$:

$$v_t = v_0 + a \times t$$

Поскольку при равноускоренном движении скорость меняется равномерно, то максимальное и минимальное ее значение достигается в одной из двух точек: начале или конце пути. Поэтому нам достаточно рассмотреть скорость в начале пути (v_0), после первого ускорения ($v_0 + a_1 \times t_1$) и после второго ускорения ($v_0 + a_1 \times t_1 + a_2 \times t_2$). Ответом будет служить максимальное среди этих значений.

Алгоритмическая реализация:

```
read(v, t1, a1, t2, a2)
write(max(v, v + t1*a1, v + t1*a1 + t2*a2))
```

В силу ограничений на значения времен и ускорений здесь наиболее эффективно использовать 8-байтовый целый тип (long long в C++ или int64 в Pascal).

Задача A. Максимальная скорость

```
//C++11
#include <bits/stdc++.h>

using namespace std;

int main(){
  long long v0,t1,a1,t2,a2;
  cin >> v0 >> t1 >> a1 >> t2 >> a2;
  cout << max({v0, v0 + a1*t1, v0 + a1*t1 + a2*t2});
  return 0;
}

//Free Pascal
uses Math;

var v0, t1, t2, a1, a2 : int64;

begin
  read(v0, t1, a1, t2, a2);
  write(max(v0, max(v0 + t1*a1, v0 + t1*a1 + t2*a2)))
end.

#Python 3
v0, t1, a1, t2, a2 = map(int, input().split())
print(max(v0, v0 + t1*a1, v0 + t1*a1 + t2*a2))
```

Задача B. Минимальный циклический сдвиг

(Время: 1 сек. Память: 16 Мб Баллы: 100)

Циклическим сдвигом строки s называется строка $s_{k+1}s_{k+2}\ldots s_n s_1 s_2 \ldots s_k$ для некоторого k $(0 \le k < n)$, где n – длина строки s.

Для заданной строки требуется определить ее лексикографически минимальный циклический сдвиг, т.е. необходимо найти среди всех возможных циклических сдвигов строки тот, который идет первым в алфавитном порядке.

Входные данные

В единственной строке входного файла INPUT.TXT записана строка, состоящая из заглавных букв английского алфавита. Длина строки от 1 до 1000 символов.

Выходные данные

В выходной файл OUTPUT.TXT выведите одну строку – минимальный лексикографический циклический сдвиг исходной строки.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	CAB	ABC
2	ABCAAAC	AAACABC

Задача B. Минимальный циклический сдвиг

В силу малых ограничений на длину строки мы можем перебрать все возможные сдвиги и выбрать наименьший среди них:

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

using namespace std;

int main(){
  int i;
  string s, m;

  cin >> s;
  m = s;

  for(i=1; i<s.length(); i++)
    s = s.substr(1)+s[0],
    m = min(m, s);

  cout << m;
  return 0;
}
```

```pascal
//Pascal
var
  i,n : integer;
  s,m : string;

begin
  read(s);
  m := s;
  for i:=1 to length(s)-1 do begin
    s := copy(s,2,length(s)-1)+s[1];
    if s<m then m := s
  end;
  write(m)
end.
```

```python
#Python
s = input()
print(min([s[i:]+s[:i] for i in range(len(s))]))
```

Задача C. Уравнение

(Время: 1 сек. Память: 16 Мб Баллы: 100)

Необходимо написать программу, решающую уравнения вида

$$\frac{a \cdot x + b}{c \cdot x + d} = v.$$

Числа a, b, c, d, v заданы, а x – неизвестно.

Входные данные

Первая строка входного файла INPUT.TXT содержит пять целых чисел, разделенных пробелом: a, b, c, d, v. Все они не превосходят 1000 по абсолютной величине.

Выходные данные

Если указанное уравнение не имеет решений, выведите в выходной файл OUTPUT.TXT слово NONE. Если у уравнения ровно одно решение, то строку вида X = p/q, где p – целое число, q – натуральное, p и q взаимно просты, а дробь p/q является решением уравнения. Если у уравнения более одного решения, выведите слово MULTIPLE.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1 2 3 4 5	X = -9/7
2	1 1 1 1 1	MULTIPLE
3	0 1 0 1 2	NONE

Система оценки

Решения, работающие только в случае существования единственного решения уравнения, будут оцениваться в 50 баллов.

Задача C. Уравнение

a, b, c, d, v – целые коэффициенты

x – неизвестная

Справа представлены несложные математические действия, приводящие к решению данного уравнения. Они верны лишь в том случае, когда уравнение имеет единственный корень. Однако, это не всегда так. Уравнение может не иметь корней или их может быть бесконечно много.

1) Когда c = 0 и d = 0 корней нет, т.к. знаменатель дроби исходного уравнения (1) равен нулю при любом x. Поскольку на ноль делить нельзя, то равенство не может быть верным при любом x.

2) Если в результате мы получаем, что p ≠ 0 и q = 0, то снова видим деление на ноль в (6). Заметим, что при этом не будет выполняться также (5). Т.е. в этом случае корней тоже нет.

3) Далее, если первые два пункта не выполнены и у нас p = 0 и q = 0, то мы получим бесконечное число решений, несмотря на неопределённость в (6). На самом деле это видно из (5): при любом x получаем, что 0 = 0. Однако, следует отметить, что при этом не любое значение x является решением. Об этом как раз следующий случай.

4) Если все предыдущие пункты не выполнены и мы получили определенную дробь в качестве ответа, то это не всегда возможный ответ. Дело в том, что при выполнении (2), (3), (5) и (6) может не выполняться (1). Это возможно только в том случае, когда при данном значении x знаменатель дроби обращается в ноль. Это произойдет, когда c×p+d×q = 0.

Во всех остальных случаях существует единственное решение, которое следует вывести в виде несократимой дроби. Для такого представления можно p и q разделить на НОД(p, q) и в случае отрицательного q поменять знаки у p и q. Для вычисления НОД можно использовать алгоритм Евклида, либо иной неэффективный алгоритм с асимптотикой O(min(p,q)).

1) $\frac{a \cdot x + b}{c \cdot x + d} = v$

2) $a \cdot x + b = v \cdot (c \cdot x + d)$

3) $(a - v \cdot c) \cdot x = v \cdot d - b$

4) $p = v \cdot d - b$

$q = a - v \cdot c$

5) $q \cdot x = p$

6) $x = \frac{p}{q}$

Задача C. Уравнение

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

using namespace std;

int main(){
  int a, b, c, d, v, p, q, gcd;

  cin >> a >> b >> c >> d >> v;

  p = v*d-b;
  q = a-v*c;

  if(c==0 && d==0 || q==0 && p!=0){cout << "NONE"; return 0;}
  if(p==0 && q==0){cout << "MULTIPLE"; return 0;}
  if(c*p+d*q==0){cout << "NONE"; return 0;}

  if(q<0) p =- p, q =- q;
  gcd = abs(__gcd(p,q));

  cout << "X = " << p/gcd << "/" << q/gcd;
  return 0;
}
```

Задача C. Уравнение

```pascal
//Pascal
var a, b, c, d, v, p, q, i : integer;

procedure Answer(ans : string);
begin  write(ans);  halt end;

begin
  read(a, b, c, d, v);

  p := v*d-b;
  q := a-v*c;

  if (c=0) and (d=0) or (q=0) and (p<>0) then Answer('NONE');
  if (p=0) and (q=0) then Answer('MULTIPLE');
  if c*p+d*q=0 then Answer('NONE');

  if q<0 then begin p := -p; q := -q end;

  for i:=2 to 1001000 do
    while (p mod i = 0) and (q mod i = 0) do begin
      p := p div i; q := q div i
    end;

  write('X = ', p, '/', q)
end.
```

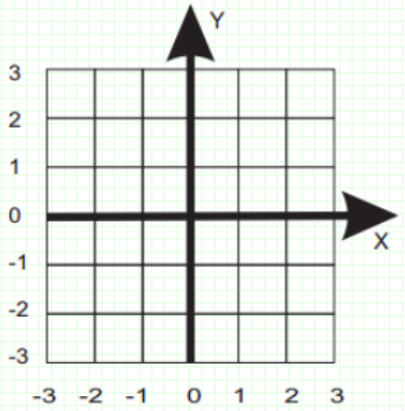
Задача C. Уравнение

```
#Python 3.5
from math import gcd

a, b, c, d, v = map(int, input().split())

p, q = v*d-b, a-v*c

if c==0 and d==0 or q==0 and p!=0:
    print('NONE')
    exit(0)
if p==0 and q==0:
    print('MULTIPLE')
    exit(0)
if c*p+d*q==0:
    print('NONE')
    exit(0)

if q<0:
    p, q = -p, -q

print('X = ', p//gcd(p,q), '/', q//gcd(p,q), sep='')
```

Задача D. Дачный участок

(Время: 2 сек. Память: 16 Мб Баллы: 100)

Недавно Василий Иванович приобрел дачный участок необычной формы. Перед ним встала задача постройки забора, для чего ему потребовалось определить длину периметра данного участка. Он бы и сам справился, если бы участок имел обычную форму прямоугольника, но здесь не все так просто и он решил обратиться к вам за помощью.



Для разметки дачных участков земельный комитет использовал правильную прямоугольную сетку, состоящую из единичных квадратов (квадратных метров). Так, любой участок представляет собой связную область из таких единичных квадратов. Это означает, что из любой точки участка можно попасть в любую другую его точку, не выходя за пределы участка. Логично, что как площадь, так и периметр участка выражаются целыми числами.

По набору координат единичных квадратов, принадлежащих дачному участку Василия Ивановича, необходимо вычислить его периметр.

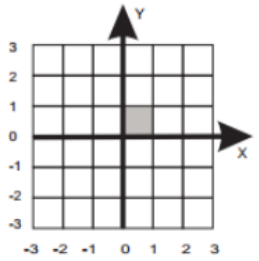
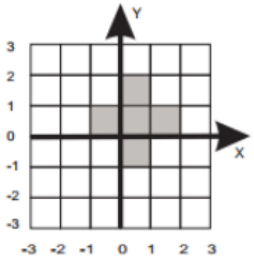
Входные данные

В первой строке входного файла INPUT.TXT указано целое число N ($1 \le N \le 10^5$) – количество единичных квадратов дачного участка. В следующих N строках описываются эти квадраты. Каждый квадрат задается целочисленными координатами (x,y) своего нижнего левого угла. ($-10^9 \le x, y \le 10^9$). Гарантируется, что участок представляет собой связную фигуру.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно целое число – периметр дачного участка.

Примеры

№	INPUT.TXT	OUTPUT.TXT	Пояснение к примеру
1	1 0 0	4	
2	5 0 0 0 -1 -1 0 1 0 0 1	12	

Система оценки

Решения, работающие только на тестах, в которых координаты единичных квадратов не превышают 1000 по абсолютной величине, будут оцениваться в 50 баллов.

Задача D. Дачный участок

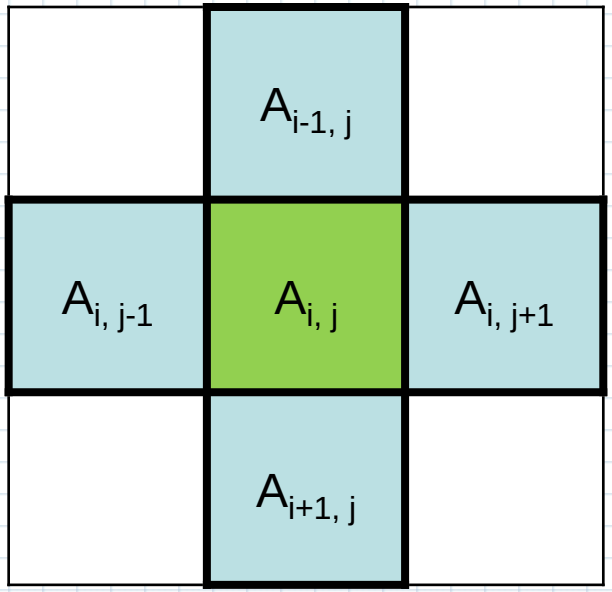
Частичное решение – 50 баллов

Представим карту земельного участка в виде целочисленного двумерного массива A, состоящего из нулей и единиц. Положим A[i][j] = 1, если данная ячейка с координатами (i, j) принадлежит дачному участку, в противном случае A[i][j] = 0. Не составит труда создать и заполнить такой массив.

Очевидно, что между двумя клетками будет элемент забора тогда и только тогда, когда одна из этих клеток принадлежит участку, а другая – нет. Это условие будет выполнено если сумма смежных элементов матрицы равна 1. Ответ на задачу – это количество таких пар, которые можно найти, если рассмотреть все клетки дачного участка и клетки, смежные с ними. Алгоритмическая реализация:

```
read(n, A)
res = 0
foreach (i,j):
  res += A[i][j]*(4 - A[i-1][j] - A[i][j+1] - A[i+1][j] - A[i][j-1])
write(res)
```

Очевидно, что такое решение может работать только при небольших расстояниях между различными клетками, так как в худшем случае для хранения двумерного массива может потребоваться порядка 300 Мб памяти.



Полное решение

Для полного решения следует использовать те же идеи, но координаты дачного участка хранить не в двумерном массиве, а в некотором множестве координат, где каждая координата представляет собой кортеж:

```
set < tuple <int, int> > s
```

Многие современные языки (C++, PascalABC.NET, Python и т.д.) поддерживают подобные структуры данных, что позволяет легко решать подобные задачи.

Задача D. Дачный участок

```
//C++11
#include <bits/stdc++.h>

using namespace std;

int n, x, y, res;
set<tuple<int,int>> s;

int main(){
  cin >> n;

  while(n--)
    cin >> x >> y,
    s.insert({x,y});

  for(auto p : s)
    tie(x,y) = p,
    res += 4 - s.count({x,y-1}) - s.count({x-1,y})
             - s.count({x+1,y}) - s.count({x,y+1});

  cout << res;
  return 0;
}
```

Задача D. Дачный участок

```pascal
//PascalABC.NET 3.4
begin
  var s := new HashSet<(integer, integer)>;

  loop ReadInteger do s.Add((ReadInteger, ReadInteger));

  var res := 0;
  foreach var t in s do begin
    var (x,y) := t;
    res += 4 - ord(s.Contains((x,y-1))) - ord(s.Contains((x-1,y)))
             - ord(s.Contains((x+1,y))) - ord(s.Contains((x,y+1)));
  end;

  write(res)
end.
```

```python
#Python 3
s = set([tuple(map(int, input().split())) for x in range(int(input()))])

res = 0
for x, y in s:
    res += 4 - ((x,y-1) in s) - ((x-1,y) in s) - ((x+1,y) in s) - ((x,y+1) in s)

print(res)
```

Задача E. Геометрическая прогрессия

(Время: 1 сек. Память: 16 Мб Баллы: 100)

Для заданных целых чисел **a** и **n** рассмотрим сумму следующего ряда:

$$S = \sum_{i=0}^{n} a^i = 1 + a + a^2 + a^3 + \ldots + a^{n-1} + a^n.$$

Эта сумма может быть очень большой. Например, при **a** = 10 и **n** = 999999 значение суммы состоит из миллиона цифр!

Требуется найти остаток от деления этой суммы на натуральное число **m**.

Входные данные

В единственной строке входного файла INPUT.TXT даны три целых числа **a**, **n** и **m**, при этом $|\mathbf{a}| \le 10^{18}$, $\mathbf{a} \ne 0$, $0 \le \mathbf{n} \le 10^{18}$, $1 \le \mathbf{m} \le 10^9$.

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное неотрицательное целое число – ответ на задачу.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	-4 10 2	1
2	2 31 10	5
3	10 9999 9	1

Система оценки

Решения, работающие для **a**, **n** и **m**, не превосходящих 10 по абсолютной величине, будут оцениваться в 20 баллов.

Решения, работающие для $\mathbf{n} \le 10^6$, будут оцениваться в 40 баллов.

Решения, работающие для взаимнопростых значений **m** и **a**-1, будут оцениваться в 80 баллов.

Задача E. Геометрическая прогрессия

Решение №1 – 20 баллов

Наиболее простым является следующее частичное решение «в лоб» на C++:

```
//C++ - 20 баллов
#include <bits/stdc++.h>

using namespace std;

int main(){
  long long a, n, m, s=0, p=1;

  cin >> a >> n >> m;

  for(int i=0; i<=n; i++)
    s += p,
    p *= a;

  cout << (s%m+m)%m;
  return 0;
}
```

Решение №2 – 36 баллов

Если использовать формулу суммы геометрической прогрессии, а также один из языков программирования, который поддерживает целочисленную длинную арифметику (например, Python), то можно набрать больше баллов:

```
#Python - 36 баллов
a, n, m = map(int, input().split())
print((a**(n+1)-1)//(a-1)%m)
```

$$S = b_1 \frac{q^n - 1}{q - 1} = \frac{a^{n+1} - 1}{a - 1}$$

Задача E. Геометрическая прогрессия

Решение №3 – 40 баллов

Если при вычислении элементов ряда и суммы каждый раз вычислять все получаемые значения по модулю m, то получим следующее решение:

```
//C++ - 40 баллов
#include <bits/stdc++.h>

using namespace std;

int main(){
  long long a, n, m, s=0, p=1;

  cin >> a >> n >> m;

  a = (a%m+m)%m;

  for(int i=0; i<=n; i++)
    s = (s+p)%m,
    p = p*a%m;

  cout << s;
  return 0;
}
```

Задача E. Геометрическая прогрессия

Решение №4 – 80 баллов

Идея этого решения достаточно проста: здесь следует просто вычислить значение суммы по формуле геометрической прогрессии в кольце вычетов по модулю m.

$$S = b_1 \frac{q^n - 1}{q - 1} = \frac{a^{n+1} - 1}{a - 1}$$

Однако, это не всегда возможно, т.к. для выполнения операции деления необходимо вычисление обратного элемента для (a-1), которого может вовсе не существовать, когда НОД(a-1, m) ≠1 или a=1. Именно поэтому данное решение не является полным. Для вычисления обратного элемента здесь можно использовать как расширенный алгоритм Евклида, так и следствие из теоремы Эйлера.

Далее рассмотрим некоторые теоретические аспекты, используемые в данном решении.

Кольцо вычетов – система операций сложения и умножения над целыми числами по модулю **m**. Все операции и их результат представляются числами диапазона от 0 до **m**-1.

Обратный элемент числа **a** в кольце вычетов по модулю **m** – это такое число a^{-1}, что $a \cdot a^{-1} = 1$. Заметим, что обратный элемент определен только тогда, когда a и m взаимно просты (НОД(a,m) = 1).

Наиболее быстрым способом нахождения обратного элемента является расширенный алгоритм Евклида, но также это возможно реализовать с использованием следующей теоремы:

Теорема Эйлера: $\mathbf{a^{\varphi(m)} = 1}$

Здесь φ(m) – функция Эйлера, которая равна количеству натуральных чисел, не превосходящих m и взаимно простых с m. Если m – простое число, то φ(m) = m-1. Из теоремы Эйлера вытекает, что $a^{-1} = a^{\varphi(m)-1}$ и для вычисления обратного элемента можно воспользоваться алгоритмом бинарного возведения в степень, который будет работать за O(log m).

Задача E. Геометрическая прогрессия

Вычисление функции Эйлера

Функция Эйлера **φ(n)** равна количеству натуральных чисел, не превосходящих n и взаимно простых с n. В частности, **φ(1)=1**.

Поскольку простое число **p** взаимно просто с любым числом, поэтому:

$$\varphi(p) = p-1$$

Далее можно рассмотреть случай, когда n имеет единственный простой делитель p, повторенный несколько раз, т.е. $n = p^k$ (случай, рассмотренный выше, есть частный случай данного, при k=1). Очевидно, что такое число будет взаимно просто со всеми числами, меньше себя, кроме чисел, кратных p. Всего таких чисел p^k-p^{k-1}. Таким образом, верно следующее:

$$\varphi(p^k) = p^k-p^{k-1}$$

Если некоторые p и q взаимно просты, то число p×q будет взаимно просто со всеми числами, меньшими себя, кроме тех, которые кратны хотя бы одному делителю p или хотя бы одному делителю q. Отсюда имеем еще одно свойство функции Эйлера, а именно: для взаимно простых p и q:

$$\varphi(p \cdot q) = \varphi(p) \cdot \varphi(q)$$

Любое число n можно разложить на простые множители. Используя это и идеи, описанные выше получаем следующую формулу:

$$\varphi(n) = \varphi(p_1^{a_1} \cdot \ldots \cdot p_k^{a_k}) = \varphi(p_1^{a_1}) \cdot \ldots \cdot \varphi(p_k^{a_k}) = (p_1^{a_1} - p_1^{a_1-1}) \cdot \ldots \cdot (p_k^{a_k} - p_k^{a_k-1})$$

Задача E. Геометрическая прогрессия

Расширенный алгоритм Евклида позволяет для пары целых неотрицательных чисел **a** и **b** находить не только НОД, но и пару целых коэффициентов **x** и **y** таких, что выполняется следующее равенство:

$$a \cdot x + b \cdot y = \gcd(a,b)$$

Выведем рекуррентную формулу коэффициентов (x,y) для пары (a,b) через коэффициенты (x_1,y_1) пары (b%a,a):

$$(b\%a) \cdot x_1 + a \cdot y_1 = \gcd(a,b)$$

$$b\%a = b - \left\lfloor \frac{b}{a} \right\rfloor \cdot a$$

$$\left(b - \left\lfloor \frac{b}{a} \right\rfloor \cdot a \right) \cdot x_1 + a \cdot y_1 = \gcd(a,b)$$

$$a \cdot \left(y_1 - \left\lfloor \frac{b}{a} \right\rfloor \cdot x_1 \right) + b \cdot x_1 = \gcd(a,b)$$

$$\begin{cases} x = y_1 - \left\lfloor \frac{b}{a} \right\rfloor \cdot x_1 \\ y = x_1 \end{cases}$$

Для нахождения обратного элемента для числа **a** в кольце вычетов по модулю **m** достаточно решить уравнение:

$$a \cdot x + m \cdot y = 1$$

Если взять остаток от деления на **m** от обеих частей, то получим, что **x** - обратный элемент по модулю **m**:

$$a \cdot x = 1 \pmod{m}$$

Следует учесть, что x может быть отрицательным.

```
//Расширенный алгоритм Евклида
int gcdex(int a,int b,int &x,int &y){
  if(a==0){x=0; y=1; return b;}
  int x1,y1,
  d = gcdex(b%a,a,x1,y1);
  x = y1 - b/a*x1;
  y = x1;
  return d;
}
//Нахождение обратного элемента
int inv(int a, int m){
  int x,y;
  if(gcdex(a,m,x,y)!=1) return 0;
    else return (x%m+m)%m;
}
```

Задача E. Геометрическая прогрессия

Бинарное возведение в степень

$$S = \frac{a^{n+1} - 1}{a - 1}$$

Для решения задачи нам необходимо осуществить вычисление числителя дроби в кольце вычетов по модулю **m**. Поскольку **n** может быть велико, то здесь не обойтись без алгоритма бинарного возведения в степень, который также может быть использован для вычисления обратного элемента через функцию Эйлера (ограничения на **m** позволяют это сделать).

Итак, для вычисления некоторого значения $\mathbf{a^n}$ будем использовать рекуррентную формулу:

$$a^n = \begin{cases} \left(a^{\frac{n}{2}}\right)^2, & если\ n - четно \\ a \cdot \left(a^{\frac{n-1}{2}}\right)^2, & если\ n - нечетно \end{cases}$$

Так мы получаем алгоритм, вычисляющий $\mathbf{a^n}$ по модулю **m** и работающий за O(log n):

```
int Power(a, m, n){
  if(n==0) return 1 mod m
  B = Power(a, m, n div 2)
  B2 = B*B mod m
  if(n mod 2 = 1) return B2*a mod m
             else return B2
}
```

Задача E. Геометрическая прогрессия

```
//C++ - 80 баллов
#include <iostream>
#define Int long long

using namespace std;

//Расширенный алгоритм Евклида
Int gcdex(Int a, Int b, Int &x, Int &y){
  if(a==0){
    x=0; y=1;
    return b;
  }
  Int x1,y1,d = gcdex(b%a, a, x1, y1);
  x = y1 - b/a*x1;
  y = x1;
  return d;
}

//Вычисление обратного элемента
Int Inv(Int a, Int m){
  Int x,y;
  gcdex(a,m,x,y);
  return (x%m+m)%m;
}
```

```
//Бинарное возведение a в степень n по модулю m
Int Power(Int a, Int n, Int m){
  if(n==0) return 1;
  if(n%2) return a*Power(a,n-1,m)%m;
  else{
    Int b = Power(a,n/2,m);
    return b*b%m;
  }
}

//Основная функция
int main(){
  Int a, n, m;

  cin >> a >> n >> m;

  a = (a%m+m)%m;

  cout << (Power(a,n+1,m)-1+m)*Inv((a-1+m)%m,m)%m;

  return 0;
}
```

$$S = \frac{a^{n+1} - 1}{a - 1}$$

Задача E. Геометрическая прогрессия

Решение №5 – полное решение

Итак, нам необходимо вычислить сумму следующей геометрической прогрессии:

$$S_n^a = 1 + a + a^2 + a^3 + \ldots + a^{n-1} + a^n$$

Для этого мы можем использовать следующие рекуррентные соотношения:

$$\begin{cases} S_0^a = 1 \\ S_n^a = 1 + a \cdot S_{n-1}^a, \ \textit{если} \ \ n - \textit{чётно} \, (n \geq 2) \\ S_n^a = (1+a) \cdot S_{\frac{n-1}{2}}^{a^2}, \ \textit{если} \ \ n - \textit{нечётно} \end{cases}$$

Покажем, что формулы верны. Первая формула очевидна. Вторую распишем следующим образом:

$$S_n^a = 1 + a \cdot S_{n-1}^a = 1 + a \cdot (1 + a + a^2 + \ldots + a^{n-1}) = 1 + a + a^2 + a^3 + \ldots + a^n = S_n^a$$

В третьей формуле для удобства положим, что n = 2×k+1. Тогда допустимы следующие разложения:

$$\begin{aligned} S_n^a = S_{2k+1}^a = (1+a) \cdot S_k^{a^2} = (1+a) \cdot (1 + a^2 + a^4 + \ldots + a^{2k}) = \\ = (1 + a^2 + a^4 \ldots + a^{2k}) + (a + a^3 + a^5 + \ldots + a^{2k+1}) = \\ = 1 + a + a^2 + a^3 + a^4 + a^5 + \ldots + a^{2k} + a^{2k+1} = S_{2k+1}^a = S_n^a \end{aligned}$$

Используя эти формулы , несложно реализовать решение за O(log n), которое напоминает бинарное возведение в степень, рассмотренное ранее.

Задача E. Геометрическая прогрессия

$$\begin{cases} S_0^a = 1 \\ S_n^a = 1 + a \cdot S_{n-1}^a, \ \textit{если} \ \ n - \textit{чётно} (n \geq 2) \\ S_n^a = (1+a) \cdot S_{\frac{n-1}{2}}^{a^2}, \ \textit{если} \ \ n - \textit{нечётно} \end{cases}$$

```
//C++ - полное решение
#include <iostream>
#define Int long long

using namespace std;

Int sum(Int a, Int n, Int m){
  if(n == 0)  return 1;
  if(n%2)
    return (1+a)*sum(a*a%m, n/2, m)%m;
  else
    return (1+a*sum(a, n-1, m))%m;
}

int main(){
  Int a, n, m;
  cin >> a >> n >> m;
  cout << sum((a%m+m)%m, n, m)%m;
  return 0;
}
```

Задача F. Сумма в ромбе

(Время: 3 сек. Память: 16 Мб Баллы: 100)

Дано клетчатое поле размера N×M, в каждой клетке которого записано целое число. Назовем ромбом с центром в точке (x,y) и целой положительной нечетной диагональю 2×d+1 множество точек (x',y'), которые удовлетворяют неравенству:

$$|x - x'| + |y - y'| \le d.$$

	1	2	3	4	5
1	0	0	1	0	0
2	0	1	1	1	0
3	1	1	-100	1	1
4	0	1	1	1	0
5	0	0	1	0	0

Положим, что все координаты точек целые, тогда каждый ромб содержит ровно $d^2 + (d+1)^2$ точек. Если среди всех точек, принадлежащих ромбу, нет таких, которые лежат за границей заданного поля, то ромб принадлежит клетчатому полю. В дальнейшем будем рассматривать только ромбы, принадлежащие клетчатому полю.

Под суммой в ромбе, будем понимать сумму всех чисел, записанных в ячейках поля, координаты которых принадлежат заданному ромбу.

Например, на рисунке справа показан ромб на клетчатом поле 5×5 с центром в точке (3,3) и диагональю 5, сумма в котором равна -88, а количество клеток, принадлежащих ромбу, равно $2^2 + 3^2 = 13$.

Вам дано клетчатое поле с числами, записанными в его ячейках, и q запросов, каждый из которых характеризуется одним целым числом c_i. Ваша задача среди всех принадлежащих полю ромбов найти такие, которые содержат ровно c_i клеток, среди них найти ромбы с максимальной суммой и найти значение этой максимальной суммы.

Входные данные

В первой строке входного файла INPUT.TXT записаны два натуральных числа N и M – высота и ширина поля (N, M < 512). Каждая из последующих N строк содержит M целых чисел, разделенных пробелом, по модулю не превосходящих 2×10^4 – числа, записанные в ячейках клетчатого поля. В следующей строке указано одно неотрицательное целое число q, не превышающее 2×10^5 – количество запросов. Затем в q строках указано по одному целому числу c_i ($|c_i| \le 10^6$).

Выходные данные

В выходной файл OUTPUT.TXT в ответ на каждый i-й запрос в отдельной строке укажите три числа: сначала n_i – количество различных ромбов, в которых ровно c_i клеток, затем m_i – количество тех из них, сумма в которых максимальна среди всех n_i ромбов, и s_i – значение максимальной суммы соответственно. Если нет ни одного подходящего ромба, то выведите $n_i = m_i = s_i = 0$.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 5 0 0 1 0 0 0 1 1 1 0 1 1 -100 1 1 0 1 1 1 0 0 0 1 0 0 5 0 1 5 13 25	0 0 0 25 12 1 9 4 3 1 1 -88 0 0 0

Система оценивания

Решения, работающие только для N, M ≤ 50 и q ≤ 10, будут оцениваться в 30 баллов.

Решения, работающие только для $c_i \le 40$, будут оцениваться в 40 баллов.

Решения, работающие только для N, M ≤ 50, будут оцениваться в 50 баллов.

Задача F. Сумма в ромбе

Частичное решение: $O(q \times n^2 \times m^2)$ – **30 баллов**

n, m – размеры матрицы
T[1..**n][**1..**m]** – матрица
c – число клеток в рассматриваемом ромбе
d – параметр диагонали ромба, в котором $d^2+(d+1)^2$ клеток
sum – текущая максимальная сумма в рассматриваемом ромбе с некоторой диагональю $2 \times d+1$
cnt – текущее количество найденных ромбов с максимальной суммой sum

Данное решение является прямолинейным и неоптимальным: после чтения матрицы будем последовательно обрабатывать **q** запросов и вычислять ответ для каждого из таковых. Для каждого **c** будем проверять существует ли такое **d**, что **c** = $\mathbf{d}^2+(\mathbf{d}+1)^2$ и $1 \le 2 \times \mathbf{d}+1 \le \min(\mathbf{m},\mathbf{n})$. Если это так, то будем пробегать по всем возможным ромбам, которых ровно $(\mathbf{n}-2 \times \mathbf{d}) \times (\mathbf{m}-2 \times \mathbf{d})$, и вычислять для данного типа ромба значения **sum** и **cnt**, которые и следует вывести в качестве ответа. Если же для заданного **c** такого ромба нет, то будем выводить «0 0 0» в качестве ответа.

Алгоритм вычисления ответа для корректного с может быть записан следующим образом:

```
cnt = 0
for i=d+1..n-d              //пробегаем по всем центрам ромбов (i,j)
  for j=d+1..m-d
    s = 0                   //переменная для накопления суммы клеток ромба
    for y=i-d..i+d          //пробегаем по всем клеткам квадрата,
      for x=j-d..j+d        //содержащего в себе ромб с диагональю 2×d+1
        if(|i-y|+|j-x| ≤ d)           //условие принадлежности ромбу
          s += T[y][x]                //прибавляем элемент ромба к s
    if(cnt=0 or s>sum) cnt=0, sum=s   //запоминаем максимальную сумму
    if(s=sum) cnt++                   //счётчик ромбов с макс. суммой
println((n-2*d)*(m-2*d), cnt, sum)    //вывод ответа
```

Задача F. Сумма в ромбе

Частичное решение: **O(n×m+q) – 40 баллов**

n, m – размеры матрицы
T[1..**n][**1..**m]** – матрица
c – число клеток в рассматриваемом ромбе
d – параметр диагонали ромба, в котором $d^2+(d+1)^2$ клеток
sum[d] – массив максимальных сумм в ромбах для с диагональю 2×d+1 для d=0..3
cnt[d] – массив найденных ромбов с максимальной суммой sum[d] для d=0..3
w[c] – массив, позволяющий по значению с определить значение d, т.е. w[c] = d

Здесь мы используем то, что для **c** ≤ 40 существует всего 4 возможных размера ромба, а именно для **c** = 1, 5, 13 и 25, что соответствует **d** = 0, 1, 2 и 3. Поэтому мы можем за 4 прохода по матрице для каждого **d** вычислить значения **sum[d]** и **cnt[d]** и далее использовать их для обработки каждого запроса за O(1).

Алгоритмическая реализация вышеописанного:

```
read(n,m,T)

for i=1..n
  for j=1..m
    for d=0..min(3,i-1,j-1,n-i,m-j)
      w[d*d+(d+1)*(d+1)] = d
      s = 0
      for y=i-d..i+d
        for x=j-d..j+d
          if(|i-y|+|j-x| ≤ d)
            s += T[y][x]
      if(cnt[d]=0 or s>sum[d])
        cnt[d]=0, sum[d]=s
      if(s=sum[d]) cnt[d]++
```

```
read(q)
for i=1..q
  read(c)
  if(c<1 or c>25 or w[c]=0 and c≠1)
    println("0 0 0")
  else
    d = w[c]
    println((n-2*d)*(m-2*d),cnt[d],sum[d])
```

Задача F. Сумма в ромбе

Частичное решение: $O(n^2 \times m^2 \times (n+m)+q)$ – 50 баллов

Здесь используем тот факт, что возможных значений **d** не может быть больше, чем min(n-1, m-1) div 2. Учитывая ограничение на число запросов **q** имеет смысл реализации предподсчета значений для всех возможных **d**, которые аналогично частичному решению на 40 баллов можно сохранять в отдельных массивах **sum** и **cnt**:

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

using namespace std;

int n,m,i,j,ii,jj,q,c,s,d,
    cnt[130562],sum[130562],
    w[130562], T[511][511];

int main(){
  scanf("%d%d", &n, &m);

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

  for(i=0; i<n; i++)
    for(j=0; j<m; j++)
      for(d=0; d<min({i+1,j+1,n-i,m-j}); d++){
        w[d*d+(d+1)*(d+1)] = d;
        s = 0;
        for(ii=i-d; ii<=i+d; ii++)
          for(jj=j-d; jj<=j+d; jj++)
            if(abs(i-ii)+abs(j-jj)<=d) s += T[ii][jj];
        if(!cnt[d] || s>sum[d]) cnt[d]=0, sum[d]=s;
        if(s==sum[d]) cnt[d]++;
      }

    scanf("%d", &q);
    while(q--){
      scanf("%d", &c);
      if(c<1 || c>130561 || !w[c] && c!=1)
        {printf("0 0 0\n"); continue;}
      d = w[c];
      printf("%d %d %d\n", (n-2*d)*(m-2*d)
                         , cnt[d], sum[d]);
    }

    return 0;
}
```

Задача F. Сумма в ромбе

Полное решение №1: O(N×M×(N+M)+q)

Используем те же идеи, что и в 50-бальном частичном решении, в котором мы вычисляли площадь ромба с диагональю 2×d+1, пробегая по всему квадрату и выполняя при этом $(2\times d+1)^2$ операций. С помощью динамического программирования и предподсчета двух массивов A и B за O(N×M) мы сможем выполнять те же действия за O(1), что приведет к полному решению задачи.

Пусть A[0..N][0..M] и B[0..N][0..M+1] – массивы частичных сумм исходной матрицы T[1..N][1..M]. Элементы исходного массива, входящие в сумму A[i][j] и B[i][j] на рисунках справа обозначены красным и зеленым цветом.

Используем следующие рекуррентные соотношения для вычисления значений матриц A и B:

```
A[i][j] = B[i-1][j] + B[i-1][j+1] - A[i-1][j] + T[i][j]
B[i][j] = A[i][j-1] + A[i][j] - B[i-1][j]
```

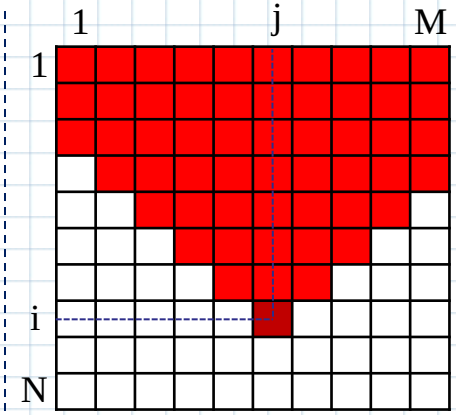
При этом перебор строк и столбцов следует выполнять в строну их увеличения с аккуратным предвычислением границ A[i][0] и B[i][M+1]:

```
A[i][0] = A[i-1][1]
B[i][M+1] = A[i][M]
```

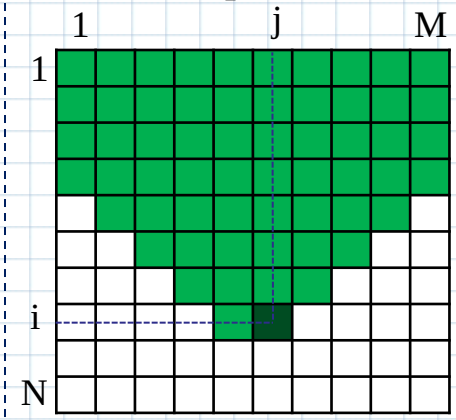
Далее, для вычисления суммы ромба с центром в точке (i,j) и диагональю 2×d+1 воспользуемся следующей формулой:

```
S = A[i+d][j]-B[i-1][j-d]-B[i-1][j+d+1]+A[i-d-1][j]
```

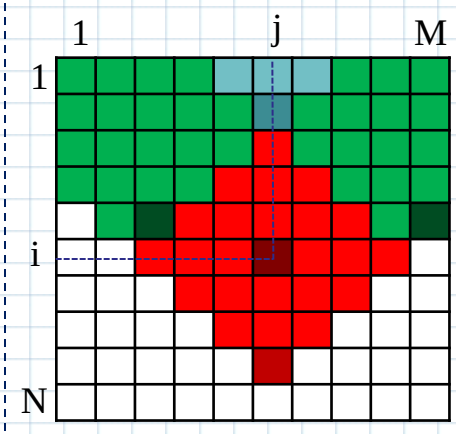
Схема данного вычисления изображена на рисунке справа.



Матрица A



Матрица B



Вычисление площади ромба

Задача F. Сумма в ромбе

```
//C++ - Решение №1
#include <stdio.h>
#include <algorithm>
#define Int long long

using namespace std;

int n,m,x,y,i,j,q,d,c,cnt[256],w[130562];
Int s, a[520][520], b[520][520], sum[256];

int main(){
  scanf("%d%d", &n, &m);

  for(i=1; i<=n; i++){
    a[i][0] = a[i-1][1];
    for(j=1; j<=m; j++)
      scanf("%d", &x),
      a[i][j] = b[i-1][j] + b[i-1][j+1] - a[i-1][j] + x,
      b[i][j] = a[i][j-1] + a[i][j] - b[i-1][j];
    b[i][m+1] = a[i][m];
  }

  for(d=0; d<=(min(n,m)-1)/2; d++)
    w[d*d+(d+1)*(d+1)] = d+1;

  for(i=1; i<=n; i++)
    for(j=1; j<=m; j++)
      for(d=0; d<min({i,j,n-i+1,m-j+1}); d++){
        s = a[i+d][j] - b[i-1][j-d] - b[i-1][j+d+1] + a[i-d-1][j];
        if(!cnt[d] || s>sum[d]) cnt[d]=0, sum[d]=s;
        if(s==sum[d]) cnt[d]++;
      }

  scanf("%d", &q);
  while(q--){
    scanf("%d", &c);
    if(c<1 || c>130561 || !w[c]){
      printf("0 0 0\n");
      continue;
    }
    d = w[c]-1;
    printf("%d %d %lld\n",
      (n-2*d)*(m-2*d), cnt[d], sum[d]);
  }

  return 0;
}
```

Задача F. Сумма в ромбе

Полное решение №2: O(N×M×(N+M)+q)

Это решение более оригинально, чем предыдущее и оно также использует подход ДП для вычисления суммы элементов в ромбе за O(1). Эта задача могла быть намного проще, если бы в ней нужно было суммировать числа в квадратах, стороны которых параллельны исходной таблице T[1..N][1..M] . Для быстрого вычисления сумм можно было бы использовать известный алгоритм RSQ для статического массива, который более детально описан на следующем слайде настоящей презентации.

Однако, в этой задаче мы все же можем применить вышеописанный алгоритм, если преобразуем исходный массив T[1..N][1..M] в массив A[1..N+M-1] [1..N+M-1] путем поворота исходного массива на 45°. Для этого достаточно каждый элемент T[i][j] записать в ячейку A[i+j-1][N-i+j], а не заполненные элементы массива A положить равными нулю. Теперь для вычисления суммы в ромбе исходного массива достаточно посчитать сумму в прямоугольнике массива A, а для этого можно использовать массив сумм B, который позволит сделать это за O(1).

В итоге, для вычисления суммы в ромбе исходного массива с центром (i,j) и диагональю 2×d+1 будем определять центр этого ромба в прямоугольнике массива A, который имеет координаты (x,y) =(N-i+j, i+j-1). Понимая, что лишние нули массива A не влияют на значение суммы, используем следующую формулу:

```
S = B[y+d][x+d] - B[y-d-1][x+d] - B[y+d][x-d-1] + B[y-d-1][x-d-1]
```

Здесь следует также отметить, что данное решение использует больше памяти и работает дольше, чем предыдущее.

	1	j				M
1	1	2	3	4	5	6	7
	8	9	10	11	12	13	14
i	15	16	17	18	19	20	21
	22	23	24	25	26	27	28
	29	30	31	32	33	34	35
	36	37	38	39	40	41	42
N	43	44	45	46	47	48	49

Исходная матрица T

	1								N-i+j				N+M-1
1	0	0	0	0	0	0	1	0	0	0	0	0	0
	0	0	0	0	0	8	0	2	0	0	0	0	0
	0	0	0	0	15	0	9	0	3	0	0	0	0
	0	0	0	22	0	16	0	10	0	4	0	0	0
i+j-1	0	0	29	0	23	0	17	0	11	0	5	0	0
	0	36	0	30	0	24	0	18	0	12	0	6	0
	43	0	37	0	31	0	25	0	19	0	13	0	7
	0	44	0	38	0	32	0	26	0	20	0	14	0
	0	0	45	0	39	0	33	0	27	0	21	0	0
	0	0	0	46	0	40	0	34	0	28	0	0	0
	0	0	0	0	47	0	41	0	35	0	0	0	0
	0	0	0	0	0	48	0	42	0	0	0	0	0
N+M-1	0	0	0	0	0	0	49	0	0	0	0	0	0

Матрица A

1												N+M-1
0	0	0	0	0	0	1	1	1	1	1	1	1
0	0	0	0	0	8	9	11	11	11	11	11	11
0	0	0	0	15	23	33	35	38	38	38	38	38
0	0	0	22	37	61	71	83	86	90	90	90	90
0	0	29	51	89	113	140	152	166	170	175	175	175
0	36	65	117	155	203	230	260	274	290	295	301	301
43	79	145	197	266	314	366	396	429	445	463	469	476
43	123	189	279	348	428	480	536	569	605	623	643	650
43	123	234	324	432	512	597	653	713	749	788	808	815
43	123	234	370	478	598	683	773	833	897	936	956	963
43	123	234	370	525	645	771	861	956	1020	1059	1079	1086
43	123	234	370	525	693	819	951	1046	1110	1149	1169	1176
43	123	234	370	525	693	868	1000	1095	1159	1198	1218	1225

Матрица B

Задача F. Сумма в ромбе

RSQ для одномерного случая

a[1..n] – целочисленный массив

b[0..n] – массив сумм ($b_{[i]} = a_{[1]}+a_{[2]}+\ldots+a_{[i]}$)

```
sub init(){
  for i=1..n
    b[i]=b[i-1]+a[i]
}

int rsq(l,r){
  return b[r]-b[l-1]
}
```

$b_{[8]} = a_{[1]}+\ldots+a_{[8]} = 60$

$a_{[i]}$	5	2	7	11	8	3	9	15	6	12
i	1	2	3	4	5	6	7	8	9	10

$b_{[3]} = 14$ $\quad rsq(4,8) = b_{[8]}-b_{[3]} = 46$

$b_{[i]}$	0	5	7	14	25	33	36	45	60	66	78
i	0	1	2	3	4	5	6	7	8	9	10

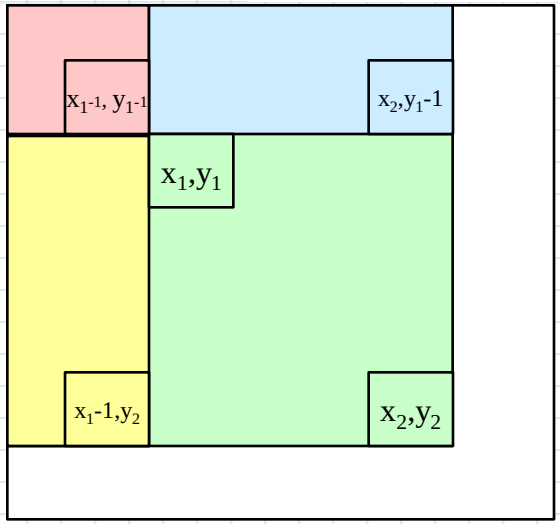
RSQ для двумерного случая

a[1..n] [1..m]– целочисленный массив

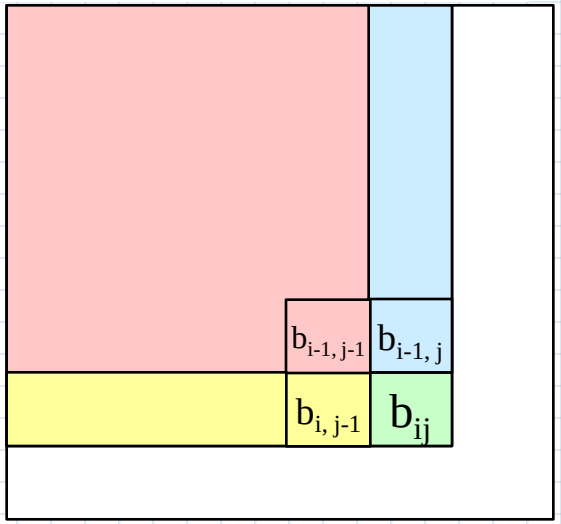
b[0..n] [0..m]– массив сумм

$$b_{[y][x]} = \sum_{\substack{1\le i\le y \\ 1\le j\le x}} a_{[i][j]}$$

```
sub init(){
  for i=1..n
    for j=1..m
      b[i][j] = b[i][j-1]+b[i-1][j] - b[i-1][j-1]+a[i][j]
}

int rsq(x1,y1,x2,y2){return b[y2][x2]-b[y2][x1-1]-b[y1-1][x2]+b[y1-1][x1-1]}
```



Матрица A



Матрица B

Задача F. Сумма в ромбе

```
//C++ - Решение №2
#include <stdio.h>
#include <algorithm>
#define Int long long

using namespace std;

int n,m,x,y,i,j,q,d,c,cnt[256],w[130562];
Int s,a[1022][1022],sum[256];

int main(){
  scanf("%d%d", &n, &m);

  for(i=0; i<n; i++)
    for(j=0; j<m; j++)
      scanf("%lld", &a[i+j+1][n-i+j]);

  for(i=1; i<n+m; i++)
    for(j=1; j<n+m; j++)
      a[i][j] += a[i-1][j]+a[i][j-1]-a[i-1][j-1];

  for(d=0; d<=(min(n,m)-1)/2; d++)
    w[d*d+(d+1)*(d+1)] = d+1;

  for(i=0; i<n; i++)
    for(j=0; j<m; j++)
      for(d=0; d<min({i+1,j+1,n-i,m-j}); d++){
        x = n-i+j;
        y = i+j+1;
        s = a[y+d][x+d]-a[y-d-1][x+d]-a[y+d][x-d-1]+a[y-d-1][x-d-1];
        if(!cnt[d] || s>sum[d]) cnt[d]=0, sum[d]=s;
        if(s==sum[d]) cnt[d]++;
      }

  scanf("%d", &q);
  while(q--){
    scanf("%d", &c);
    if(c<1 || c>130561 || !w[c]){
      printf("0 0 0\n");
      continue;
    }
    d = w[c]-1;
    printf("%d %d %lld\n",
      (n-2*d)*(m-2*d), cnt[d], sum[d]);
  }

  return 0;
}
```