Март 10, 2022

Задача A. Торт от Толи
Задача B. Игра с пол-потолком
Задача C. Найти кратное
Задача D. Три последовательности
Задача E. Подпоследовательности
Задача F. Сумма квадратов
Задача G. Минимальная сумма
Задача H. Буфкрафт
Задача I. Тосты
Задача J. Черный квадрат

1033. Торт от Толи

Толя на день рождения собирается угостить друзей тортом. Известно, что на дне рождения может быть либо n, либо m человек, включая самого именинника. На какое минимальное количество частей ему нужно разрезать торт (не обязательно всех равных), чтобы при любом из указанных количестве собравшихся, все съели торт поровну?

Вход. Два числа m и n ($1 \le m, n \le 30000$).

Выход. Вывести искомое минимальное количество кусочков торта.

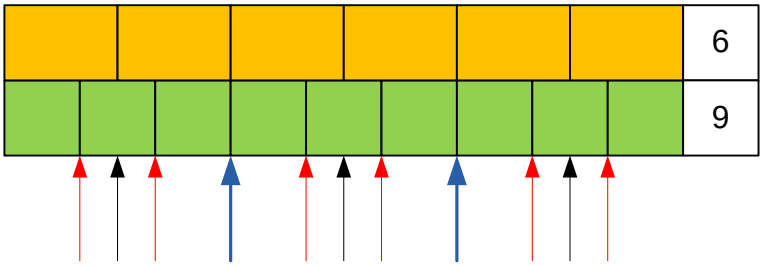
Пример входа	**Пример выхода**
2 3	4

Представим торт в виде отрезка некоторой длины. Отметим на нем разрезы в случае разделения на n равных частей. Затем отметим на нем разрезы в случае разделения на m равных частей. Совершим отмеченные разрезы – они и будут искомыми. Осталось подсчитать количество частей, на которое распадется отрезок. Это количество будет равно

$$n + m - \text{НОД}(n, m)$$

Доказательство. Количество разрезов, которые делят торт на n равных частей, равно $n - 1$. Количество разрезов, которые делят торт на m равных частей, равно $m - 1$. Среди них будет $\text{НОД}(n, m) - 1$ совпадающих разрезов. Итого имеем на торте $(n - 1) + (m - 1) - (\text{НОД}(n, m) - 1) = n + m - 1 - \text{НОД}(n, m)$ разных разрезов. После их совершения торт распадется на $n + m - \text{НОД}(n, m)$ кусков.

Пример



6 + 9 - НОД(6,9) = 12 частей

При разрезании торта на 6 частей имеем 5 разрезов.
При разрезании торта на 9 частей имеем 8 разрезов.
НОД(6, 9) – 1 = 2 разрезов совпадают.
Итого на торте имеется 5 + 8 – 2 = 11 разных разрезов. После их совершения торт распадется на 12 кусков.

Реализация алгоритма
Читаем входные данные. Вычисляем и выводим ответ.

```
scanf("%d %d",&m,&n);
res = m + n - gcd(m,n);
printf("%d\n",res);
```

1565. Игра с пол-потолком

Теорема. Для двух целых чисел x и k всегда существуют такие целые p и q, что

$$x = p\left\lfloor \frac{x}{k} \right\rfloor + q\left\lceil \frac{x}{k} \right\rceil$$

Это довольно известная теорема, но мы не просим Вас её доказывать. Мы хотим Вас попросить сделать кое-что попроще! Зная значения целых x и k, Вы должны найти такие целые p и q, которые удовлетворяют заданному уравнению.

Вход. Первая строка содержит количество тестов t ($1 \le t \le 1000$). Каждая из следующих t строк содержит два положительных целых числа x и k. Известно, что x и k не больше 10^8.

Выход. Для каждого теста выведите в отдельной строке два целых числа p и q. Если существует несколько пар чисел p и q, удовлетворяющих условию, то выведите любую. Известно, что значения $p\lfloor x/k \rfloor$ и $q\lceil x/k \rceil$ являются 64-битными целыми числами.

Пример входа	**Пример выхода**
3 5 2 40 2 24444 6	1 1 1 1 0 6

Если x нацело делится на k, то $\lfloor x/k \rfloor = \lceil x/k \rceil = x / k$. Выбрав $p = 0$, $q = k$, получим: $0 * (x / k) + k * (x / k) = x$.

Пусть x не делится на k. Если $n = \lfloor x/k \rfloor$, то $m = \lceil x/k \rceil = n + 1$. Поскольку НОД($n$, m) = НОД(n , $n + 1$) = 1, то исходя из расширенного алгоритма Евклида, существуют такие целые t и u, что $1 = tn + um$. Помножив равенство на x, получим $x = xtn + xum$, откуда $p = xt$, $q = xu$.

Пример. В первом тесте $x = 5$, $k = 2$. Значение x не делится на k нацело. Вычислим $n = \lfloor 5/2 \rfloor = 2$, $m = \lceil 5/2 \rceil = 3$. Решением уравнения $2t + 3u = 1$ будет пара $(t, u) = (-1, 1)$. Умножим уравнение на $x = 5$. Решением уравнения $2p + 3q = 5$ будет пара $(p, q) = (5t, 5u) = (-5, 5)$. Имеет место соотношение:

$$5 = (-5) * \lfloor 5/2 \rfloor + 5 * \lceil 5/2 \rceil = (-5) * 2 + 5 * 3 = -10 + 15$$

Реализация алгоритма

При вычислении используем 64-битовый целый тип ***long long***. Для простоты использования переопределим тип:

```
typedef long long i64;
```

Функция ***gcdext*** представляет собой расширенный алгоритм Евклида.

```
void gcdext(i64 a,i64 b, i64 *d, i64 *x, i64 *y)
{
  i64 s;
  if (b == 0)
  {
    *d = a; *x = 1; *y = 0;
    return;
  }
  gcdext(b,a % b,d,x,y);
  s = *y;
  *y = *x - (a / b) * (*y);
  *x = s;
}
```

Основная часть программы. Читаем количество тестов. Для каждого теста вводим значения x и k.

```
scanf("%d",&tests);
while(tests--)
{
  scanf("%lld %lld",&x,&k);
```

Если x делится на k, то устанавливаем $p = 0$, $q = k$.

```
  if (x % k == 0) { p = 0; q = k;}
  else
  {
```

Иначе вычисляем $n = \lfloor x/k \rfloor$ и $m = \lceil x/k \rceil$, запуская расширенный алгоритм Евклида. Он находит такие t и u, что $1 = tn + um$. Далее находим $p = xt$, $q = xu$ и выводим результат.

```
    n = (int)floor(1.0 * x / k); m = (int)ceil(1.0 * x / k);
    gcdext(n,m,&g,&t,&u);
    p = t * x; q = u * x;
  }
  printf("%lld %lld\n",p,q);
}
```

5621. Найти кратное

Имеется n натуральных чисел, каждое из которых не больше 15000. Они не обязательно различны (два или более числа могут быть одинаковыми). Необходимо выбрать некоторое количество *few* ($1 \le few \le n$) этих чисел так, чтобы их сумма делилась на n (то есть $n * k =$ (сумме выбранных чисел) для некоторого числа k).

Вход. Первая строка содержит число n ($n \le 10000$). Каждая из следующих n строк содержит одно из имеющихся чисел.

Выход. Если требуемое множество чисел не найдено, то вывести 0. Иначе в первой строке вывести количество выбранных чисел, а затем и сами числа (по одному в отдельной строке) в произвольном порядке. Если существует более чем одно множество чисел с требуемыми свойствами, то вывести любое из них.

Пример входа	**Пример выхода**
5 1 2 3 4 1	2 2 3

Пусть $d_1, d_2, \ldots, d_n$ – входные числа. Рассмотрим все частичные суммы $s_i = d_1 + \ldots + d_i$. Поскольку частичных сумм имеется в точности n, то среди всех значений $s_i \bmod n$ обязательно найдутся либо два одинаковых, либо такое i что $s_i \bmod n = 0$.

Если $s_{a-1} \bmod n = s_b \bmod n$ для $a - 1 < b$, то $d_a + \ldots + d_b$ делится на n. Набор чисел $d_a, d_{a+1}, \ldots, d_b$ будет искомым. Если существует такое i, что $s_i \bmod n = 0$, то искомыми будут числа $d_1, d_2, \ldots, d_i$.

Пример. Рассмотрим приведенный пример. Вычислим все частичные суммы:

s_i	$s_i \bmod 5$
$s_1 = 1$	1
$s_2 = 1 + 2 = 3$	3
$s_3 = 1 + 2 + 3 = 6$	1
$s_4 = 1 + 2 + 3 + 4 = 10$	0
$s_5 = 1 + 2 + 3 + 4 + 1 = 11$	1

i	1	2	3	4	5
d_i	1	2	3	4	1
s_i	1	3	6	10	11
$s_i \bmod 5$	1	3	1	0	1

Искомых наборов имеется несколько. Например:

- поскольку $s_1 = s_3$, то $d_2 + d_3 = 5$ делится на 5.
- поскольку $s_1 = s_5$, то $d_2 + d_3 + d_4 + d_5 = 10$ делится на 5.
- поскольку $s_3 = s_5$, то $d_4 + d_5 = 5$ делится на 5.
- поскольку $s_4 = 0$, то $d_1 + d_2 + d_3 + d_4 = 10$ делится на 5.

Реализация алгоритма

Входные числа будем хранить в массиве d. Массив r будем использовать для отмечания частичных сумм: если $s = d_1 + \ldots + d_i$, то положим r[s] = i.

```
#define MAX 10100
int d[MAX], r[MAX];
```

Набором чисел, сумма которых делится на n, будут d[a], d[$a + 1$], …, d[b]. Выводим их количество $b - a + 1$, а также сами числа по одному в одной строке.

```
void print(int a, int b)
{
  printf("%d\n",b - a + 1);
  for(int i = a; i <= b; i++)
    printf("%d\n",d[i]);
}
```

Основная часть программы.

```
memset(r,-1,sizeof(r)); r[0] = 0;
```

```
scanf("%d",&n);
for(sum = 0, i = 1; i <= n; i++)
{
  scanf("%d",&d[i]);
  sum = (sum + d[i]) % n;
```

Если вторично встретилась частичная сумма, равная *sum*, то выводим набор требуемых чисел $d_{r[sum]+1}, \ldots, d_i$.

```
  if (r[sum] != -1)
  {
    print(r[sum]+1,i);
    break;
  }
  else r[sum] = i;
}
```

1765. Три последовательности

Даны три последовательности целых чисел. Найдите длину их наибольшей общей подпоследовательности.

Вход. Содержит описание трех последовательностей. Каждая последовательность задается в двух строках. Первая строка содержит длину последовательности n ($1 \le n \le 100$), а вторая – ее элементы (32-х битовые целые числа).

Выход. Выведите длину наибольшей общей подпоследовательности.

Пример входа	**Пример выхода**
3 1 2 3 3 1 3 2 3 2 1 3	2

Пусть a, b, c – три входные последовательности. Пусть f(i, j, k) равно длине НОП последовательностей a[1..i], b[1..j] и c[1..k]. Значение f(i, j, k) будем хранить в dp[i][j][k].

Если a[i] = b[j] = c[k], то

$$f(i, j, k) = 1 + f(i - 1, j - 1, k - 1)$$

Иначе

$$f(i, j, k) = \max(f(i - 1, j, k), f(i, j - 1, k), f(i, j, k - 1))$$

Реализация алгоритма

Входные три последовательности храним в массивах a, b, c. Для нахождения их НОП объявим массив dp.

```
#define MAX 110
int a[MAX], b[MAX], c[MAX];
int dp[MAX][MAX][MAX];
```

Читаем три входные последовательности.

```
scanf("%d", &n);
for (i = 1; i <= n; i++)
  scanf("%d", &a[i]);
scanf("%d", &m);
for (i = 1; i <= m; i++)
  scanf("%d", &b[i]);
scanf("%d", &k);
for (i = 1; i <= k; i++)
  scanf("%d", &c[i]);
```

Вычисляем НОП, заполняем массив dp.

```
for (u = 1; u <= n; u++)
for (v = 1; v <= m; v++)
for (w = 1; w <= k; w++)
{
  if ((a[u] == b[v]) && (a[u] == c[w]))
    dp[u][v][w] = dp[u - 1][v - 1][w - 1] + 1;
  else
   dp[u][v][w] = max(dp[u - 1][v][w], max(dp[u][v - 1][w], dp[u][v][w
- 1]));
}
```

Выводим ответ.

```
printf("%d\n", dp[n][m][k]);
```

988. Подпоследовательности

Дана последовательность, требуется найти длину наибольшей возрастающей подпоследовательности.

Вход. В первой строке записана длина n ($1 \le n \le 1000$) последовательности. Во второй строке записана сама последовательность. Числа последовательности – целые числа, не превосходящие 10000 по модулю.

Выход. Вывести наибольшую длину возрастающей подпоследовательности.

Пример входа	**Пример выхода**
6 3 29 5 5 28 6	3

Решение за $O(n^2)$. В задаче следует найти длину наибольшей возрастающей подпоследовательности (НВП) для последовательности $x_0, x_1, \ldots, x_{n-1}$.

Объявим массив lis, где lis[i] содержит длину НВП, которую можно выделить среди чисел $x_0, x_1, \ldots, x_i$, и которая обязательно заканчивается элементом x_i. Если массив x содержит n чисел, то ответом задачи будет наибольшее среди чисел $\text{lis}_0, \ldots, \text{lis}_{n-1}$. Значение lis_0 положим равным 1: НВП для последовательности из одного элемента равна этому самому элементу. Следующие значения массива lis вычисляются согласно соотношению:

$$lis[i] = \max_{\substack{j<i \\ x[j]<x[i]}} (lis[j]) + 1$$

Элемент x_i может продолжить только те НВП, которые заканчиваются элементом x_j, для которого $j < i$ и $x_j < x_i$.

Решение за $O(n \log_2 n)$. После обработки i элементов массива x хвост массива lis (ячейки от lis_k до lis_{len-1} для некоторого k) содержит хвост наибольшей возрастающей подпоследовательности, которую можно выделить среди чисел $x_0, x_1, \ldots, x_i$. При этом хвост этой НВП будет лексикографически наименьшим, а значение len равно длине НВП для последовательности $x_0, x_1, \ldots, x_i$.

При обработке очередного элемента x_i вставим его в массив lis при помощи бинарного поиска. Если x_i больше значения lis_{len-1}, то элемент x_i увеличивает на единицу длину НВП, поэтому устанавливаем $\text{lis}_{len} = x_i$ и увеличиваем len на 1. Если $\text{lis}_k < x_i \le \text{lis}_{k+1}$, то заменяем lis_{k+1} на x_i. Эта операция не увеличивает длину НВП, но может лексикографически уменьшить хвост НВП.

После обработки всех элементов массива x значение len содержит длину НВП.

Пример. Для заданной в примере последовательности НВП может быть {3, 5, 6} или {3, 5, 28}. Длина НВП равна 3.

3	29	5	5	28	6

3	29	5	5	28	6

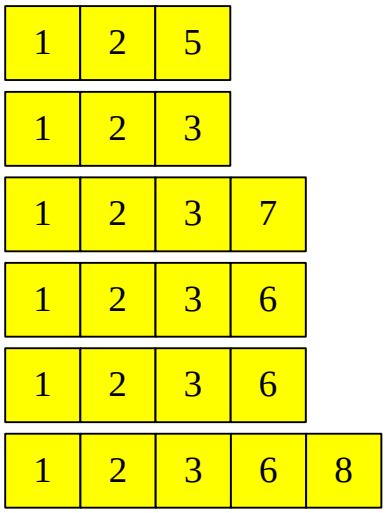
Рассмотрим следующий массив

$x =$

3	1	4	2	5	3	7	6	3	8

Промоделируем на нем работу алгоритма. В каждой новой строке приведем состояние массива после очередной итерации.

3									
1									
1	4								
1	2								



Значение *len* = 5, следовательно длина НВП равна 5. Ею, например, может быть следующая последовательность:

x =	3	1	4	2	5	3	7	6	3	8

Реализация алгоритма

Входную последовательность храним в массиве x. Объявим вспомогательный массив lis.

```
#define MAX 1001
int x[MAX], lis[MAX];
```

Основная часть программы. Читаем входную последовательность.

```
scanf("%d",&n);
for(i = 0; i < n; i++) scanf("%d",&x[i]);
```

Проводим *n* итераций алгоритма.

```
for (len = i = 0; i < n; i++)
{
  pos = lower_bound(lis,lis+len,x[i]) - lis;
  if (pos < len) lis[pos] = x[i]; else lis[len++] = x[i];
}
```

Выводим длину НВП.

```
printf("%d\n",len);
```

Реализация алгоритма – $O(n^2)$

```
#include <cstdio>
#include <vector>
#include <algorithm>
using namespace std;

vector<int> x, lis;
int i, j, n, res;
```

```
int main(void)
{
  scanf("%d", &n);
  x.resize(n); lis.resize(n);
  for (i = 0; i < n; i++)
  {
    scanf("%d", &x[i]);
    lis[i] = 1;
  }

  for (i = 0; i < n; i++)
  for (j = 0; j < i; j++)
    if (x[j] < x[i]) lis[i] = max(lis[i], 1 + lis[j]);

  res = *max_element(lis.begin(), lis.end());
  printf("%d\n", res);
  return 0;
}
```

2414. Сумма квадратов

Обозначим через U(n) общее количество последовательностей, состоящих только из чисел 1 и 2, сумма членов которой, равна n.

По заданному значению n ($1 \le n \le 10^{18}$) определить сумму квадратов чисел U(n) для всех $n = 1, \ldots, n$. Результат выдать по модулю 1000000009.

Вход. Одно число n.

Выход. Вывести ответ задачи.

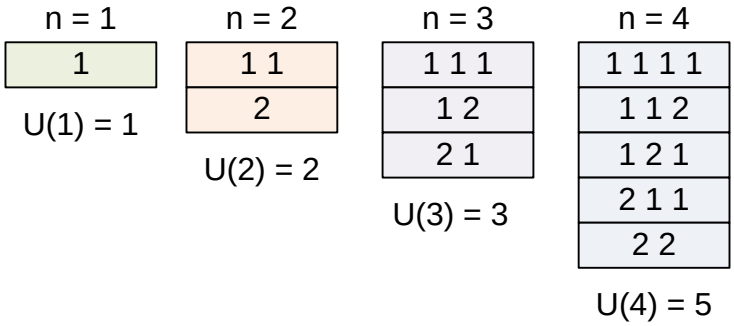
Пример входа	**Пример выхода**
3	14

Рассмотрим, что собой представляет функция U(n). Очевидно, что U(1) = 1 (одна последовательность из одной единицы), U(2) = 2 (две последовательности 1, 1 и 2). Для всех натуральных n U(n) соответствует рекуррентности Фибоначчи:

U(n)	=	1	U(n-1)	+	2	U(n-2)

n	1	2	3	4	5	6
U(n)	1	2	3	5	8	13

Например, для начальных значений n можно получить следующие последовательности:



Числа Фибоначчи имеют следующий вид:

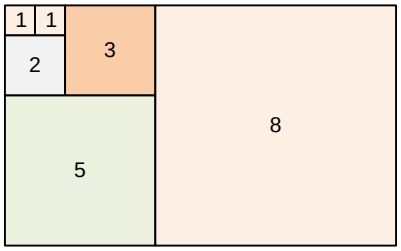
n	0	1	2	3	4	5	6	7
F(n)	0	1	1	2	3	5	8	13

В задаче требуется найти сумму

$$U(1)^2 + U(2)^2 + \ldots + U(n)^2 = F(2)^2 + F(3)^2 + \ldots + F(n+1)^2 = F(n+1) * F(n+2) - 1$$

Осталось реализовать вычисление чисел Фибоначчи $F(n)$ за время $O(\log_2 n)$.

Сумма квадратов чисел Фибоначчи имеет геометрическую интерпретацию. Составим прямоугольник из квадратов с длинами сторон F(1), F(2), …, F(n) как показано на рисунке. Тогда площадь прямоугольника равна F(n) * F(n + 1).



$$F(1)^2 + F(2)^2 + F(3)^2 + F(4)^2 + F(5)^2 + F(6)^2 =$$
$$1^2 + 1^2 + 2^2 + 3^2 + 5^2 + 8^2 = 8 * 13 = F(6) * F(7)$$

Таким образом

$$F(1)^2 + \ldots + F(n)^2 = F(n) * F(n + 1)$$

Эту формулу можно доказать при помощи метода математической индукции:

1. База индукции: $n = 1$. $F(1)^2 = F(1) * F(2)$ или $1^2 = 1 * 1$, что верно.
2. Шаг индукции.

$$F(1)^2 + \ldots + F(n)^2 + F(n + 1)^2 = F(n) * F(n + 1) + F(n + 1)^2 =$$
$$= F(n + 1) * (F(n) + F(n + 1)) = F(n + 1) * F(n + 2)$$

Пример. Для $n = 3$ имеем:

$$U(1)^2 + U(2)^2 + U(3)^2 =$$
$$F(2)^2 + F(3)^2 + F(4)^2 = F(4) * F(5) - 1 = 3 * 5 - 1 = 14$$

Реализация алгоритма

Объявим модуль MOD, вычисления по которому будем производить.

```
#define MOD 1000000009
```

Реализуем вычисление чисел Фибоначчи через возведение матрицы в степень.

```
class Matrix
{
public:
  long long a, b, c, d;
  Matrix(long long a = 1, long long b = 0,
         long long c = 0, long long d = 1)
  {
    this->a = a; this->b = b;
    this->c = c; this->d = d;
  }

  Matrix operator* (const Matrix &x)
  {
    Matrix res;
    res.a = (a * x.a + b * x.c) % MOD;
    res.b = (a * x.b + b * x.d) % MOD;
    res.c = (c * x.a + d * x.c) % MOD;
    res.d = (c * x.b + d * x.d) % MOD;
    return res;
  }

  Matrix operator^ (long long n)
  {
    Matrix x(*this);
    if (n == 0) return Matrix();
    if (n & 1) return x * ((x * x) ^ (n / 2));
    return (x * x) ^ (n / 2);
  }
};

long long fib(long long n)
{
  Matrix res(1, 1, 1, 0);
  res = res ^ n;
  return res.b;
}
```

Читаем входные данные, вычисляем и выводим результирующее значение $F_{n+1} * F_{n+2} - 1$.

```
scanf("%lld",&n);
res = (fib(n+1) * fib(n+2) + MOD - 1) % MOD;
printf("%lld\n",res);
```

Реализация алгоритма – рекурсия с запоминанием

```
#include <cstdio>
#include <map>
#define MOD 1000000009
using namespace std;

map<long long, long long> fib;
```

```
long long f(long long n)
{
  if (n == 0) return 1;
  if (n == 1) return 1;
  if (fib[n] != 0) return fib[n];
  long long k = n / 2;
  if (n % 2 == 0)
    return fib[n] = (f(k - 1) * f(k - 1) + f(k) * f(k)) % MOD;
  return fib[n] = (f(k) * (f(k - 1) + f(k + 1))) % MOD;
}

long long n, res;

int main(void)
{
  scanf("%lld", &n);
  res = (f(n) * f(n + 1) + MOD - 1) % MOD;
  printf("%lld\n", res);
  return 0;
}
```

6198. Минимальная сумма

Имеются два массива натуральных чисел $a_{1..n}$ и $b_{1..n}$. Найти перестановку i_1, i_2, ..., i_n чисел 1, 2, ..., n, для которой сумма

$$a_1 * b_{i1} + ... + a_n * b_{in}$$

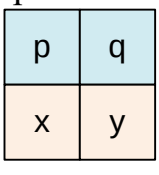
минимальна. В перестановку каждое число должно входить только один раз.

Вход. В первой строке находится количество элементов n ($n \le 100$) в массивах. Во второй строке заданы значения элементов первого массива, а в третьей – второго. Значения элементов массивов не превышают 10^6.

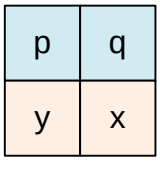
Выход. Вывести минимальное значение искомой суммы.

Пример входа	**Пример выхода**
5 7 2 4 3 10 5 11 6 9 6	165

Рассмотрим случай массивов с двумя элементами. Пусть A = {p, q} ($p < q$), B = {x, y}. Рассмотрим условие, при котором сумма $p * x + q * y$ будет наименьшей. Рассмотрим два варианта перестановок.



$p * x + q * y$



$p * y + q * x$

Неравенство $p * x + q * y < p * y + q * x$ имеет место, если

$$x * (p - q) < y * (p - q)$$

Учитывая что $p \neq q$, разделим неравенство на $(p - q) < 0$. Получим $x > y$.
То есть сумма $p * x + q * y$ (при $p < q$) будет минимальной, если $x > y$.

Рассмотрим пару элементов исходных массивов A и B. Если $a_i < a_j$, $b_i < b_j$, то поменяв местами b_i и b_j, мы уменьшим общую сумму.

Отсортируем массив a по возрастанию, а массив b по убыванию. Тогда получим сумму с наименьшим значением

$$\sum_{i=1}^{n} a_i \cdot b_i$$

Пример. Вычислим сумму для заданных в условии данных.

a_i	7	2	4	3	10
b_i	5	11	6	9	6
$a_i * b_i$	35	22	24	27	60

168

Возьмем пару для которой $a_1 < a_5$ и $b_1 < b_5$. Поменяем местами b_1 и b_5. Сумма уменьшится.

a_i	7	2	4	3	10
b_i	6	11	6	9	5
$a_i * b_i$	42	22	24	27	50

165

Для заданного примера для любой пары (i, j) выполняется: если $a_i < a_j$, то $b_i > b_j$. Следовательно текущая перестановка оптимальная.

Рассмотрим получение минимальной суммы, отсортировав a по возрастанию, а массив b по убыванию.

a_i	2	3	4	7	10
b_i	11	9	6	6	5
$a_i * b_i$	22	27	24	42	50

165

Реализация алгоритма

Объявим входные массивы чисел.

```
#define MAX 101
int a[MAX], b[MAX];
```

Читаем входные данные.

```
scanf("%d",&n);
for(i = 0; i < n; i++)
  scanf("%d",&a[i]);
for(i = 0; i < n; i++)
  scanf("%d",&b[i]);
```

Сортируем первый массив по возрастанию, второй – по убыванию.

```
sort(a,a+n);
sort(b,b+n,greater<int>());
```

Вычисляем искомую минимальную сумму, значение которой может быть 64-битовым целым.

```
for(res = i = 0; i < n; i++)
  res += 1LL * a[i] * b[i];
```

Выводим результат.

```
printf("%lld\n",res);
```

7493. Буфкрафт

Бренда наслаждается новой ролевой игрой Буфкрафт. Щиты, мечи, книги и другая ручная кладь не влияет на статус персонажа в Буфкрафт. Единственный способ увеличить статус Вашего персонажа – забодать его.

В Буфкрафте имеются две бодалки. Прямая бодалка увеличивает базовое значение статуса, в то время как процентная бодалка увеличивает базовое значение на некоторый процент. Если начальное значение статистики Вашего персонажа равно b, после чего Вы его отбодали n прямыми бодалками с силой d_1, d_2, ..., d_n и m процентными бодалками с силой p_1, p_2, ..., p_m, то результирующий статус будет равен $(b + d_1 + d_2 + ... + d_n)(100 + p_1 + p_2 + ... + p_m) / 100$. Отметим, что результат может иметь вид дроби.

К несчастью, у Вашего героя имеются лишь k слотов для бодания, и если Вы забодаете его больше k раз, то лишь последние k боданий останутся активными. Поэтому нет смысла совершать более k боданий одновременно. Одно и то же бодание накладывать более одного раза на персонажа нельзя.

Бренда собирается отправить своего персонажа в поход и хочет увеличить его здоровье до максимально возможного значения. Она имеет в своем распоряжении несколько прямых и процентных боданий и просит Вашей помощи выбрать из них такой набор, который даст персонажу максимально возможное значение здоровья.

Вход. Первая строка содержит четыре целых числа b, k, c_d и c_p – базовое здоровье героя, число слот для бодания, количество доступных прямых боданий и количество доступных процентных боданий.

Следующая строка содержит c_d целых чисел d_i – силы прямых боданий.

Последняя строка содержит c_p целых чисел p_i – силы процентных боданий.

Все числа больше или равны нулю, и не более пятидесяти тысяч.

Выход. В первой строке вывести два числа n и m – количество прямых и процентных боданий ($0 \le n \le c_d$, $0 \le m \le c_p$, $0 \le n + m \le k$), которые будут использованы.

В следующей строке вывести n различных чисел – индексы применяемых прямых боданий (бодания нумеруются с единицы).

В последней строке вывести m разных чисел – индексы применяемых процентных боданий (также нумеруются с единицы).

Результирующее здоровье после применения всех $n + m$ боданий должно быть наибольшим.

Пример входа	**Пример выхода**
70 3 2 2 40 30 50 40	2 1 1 2 1

Чтобы персонаж получил максимально возможное значение здоровья, силы боданий следует выбрать наибольшими. Отсортируем силы прямых и процентных боданий по убыванию. Поскольку имеются лишь k слотов для бодания, мы не знаем сколько следует совершить прямых n и сколько процентных m боданий ($n + m \le k$, $n \le cd$, $m \le cp$). Совершим перебор значений n и m и найдем ту пару, при которой здоровье персонажа будет максимальным.

Реализация алгоритма

Объявим рабочие массивы: силы прямых боданий d и силы процентных боданий p. Силы боданий будем хранить вместе с их индексами. Соответствующие частичные суммы посчитаем в массивах sumd и sump.

```
vector<pair<int,int> > d, p;
vector<int> sumd, sump;
```

Читаем входные данные. Заполняем массивы.

```
scanf("%d %d %d %d",&b,&k,&cd,&cp);
d.resize(cd+1);   p.resize(cp+1);
sumd.resize(cd+1);sump.resize(cp+1);
for(i = 1; i <= cd; i++)
{
  scanf("%d",&d[i].first);
  d[i].second = i;
}
```

```
for(i = 1; i <= cp; i++)
{
  scanf("%d",&p[i].first);
  p[i].second = i;
}
```

Отсосртируем силы боданий по убыванию.

```
sort(d.begin(),d.end(),greater<pair<int,int> >());
sort(p.begin(),p.end(),greater<pair<int,int> >());
```

Вычислим частичные суммы сил боданий.

```
for(i = 1; i <= cd; i++)
  sumd[i] = sumd[i-1] + d[i-1].first;
for(i = 1; i <= cp; i++)
  sump[i] = sump[i-1] + p[i-1].first;
```

Совершим перебор значений n и m ($n + m \le k$), и найдем те, при которых результирующее здоровье персонажа *res* будет наибольшим. Учитываем ограничения: $n \le cd$, $m \le cp$. Сумма наибольших n прямых боданий находится в sumd[n]. Сумма наибольших m процентных боданий находится в sump[m].

```
res = 0;
for(n = 0; n <= min(cd,k); n++)
{
  m = min(k - n,cp);
  s = 1LL * (b + sumd[n]) * (100 + sump[m]);
  if (s > res)
  {
    res = s;
```

Оптимальное количество прямых боданий сохраним в *indn*, а процентных в *indm*.

```
    indn = n; indm = m;
  }
}
```

Выводим ответ – количество прямых и процентных боданий, а также их индексы.

```
printf("%d %d\n",indn,indm);
if (indn > 0)
{
  printf("%d",d[0].second);
  for(i = 1; i < indn; i++)
    printf(" %d",d[i].second);
}
printf("\n");

if (indm > 0)
{
  printf("%d",p[0].second);
  for(i = 1; i < indm; i++)
```

```
        printf(" %d",p[i].second);
    }
    printf("\n");
```

510. Тосты

Вы хотите поджарить несколько тостов для предстоящей вечеринки. Имеется сковорода, на которой может жариться одновременно k тостов. Поджаривание тоста с одной стороны занимает 2 минуты. Будем считать, что операции размещения тоста на сковороде, переворачивания и снятия тоста со сковороды выполняются мгновенно. Напишите программу, вычисляющую минимальное время в минутах для поджаривания n тостов. Тосты нельзя снимать со сковороды раньше или позже 2 минут, необходимых для поджаривания одной стороны. Каждый тост нужно поджарить с обеих сторон.

Вход. В одной строке содержатся два целых числа n и k, разделенных пробелом ($0 \le n \le 1000$, $1 \le k \le 50$) – количество тостов и вместимость сковороды.

Выход. Вывести минимальное время в минутах для поджаривания n тостов.

Пример входа	**Пример выхода**
3 2	6

Если $n = 0$, то ответ 0.

Если количество тостов n не больше вместимости сковороды k, то на поджаривание уйдет 4 минуты.

Поскольку поджаривать тосты следует с обеих сторон, то всего прожарить следует $\lceil 2n/k \rceil$ сторон тостов. Одна прожарка на сковороде занимает 2 минуты времени.Тогда при $n > k$ количество минут, за которое можно поджарить n тостов, равно 2 * $\lceil 2n/k \rceil$.

Реализация алгоритма

Читаем входные данные.

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

Если $n = 0$, то ответ 0.

```
if (!n) res = 0; else
```

Если количество тостов не больше вместимости сковороды, то на поджаривание уйдет 4 минуты.

```
if (n <= k) res = 4; else
{
```

Поджаривать тосты следует с обеих сторон. Всего прожарить следует $\lceil 2n/k \rceil$ сторон тостов. Следует также помнить, что одна прожарка занимает 2 минуты времени.

```
  res = 2 * n / k;
  if (2 * n % k) res++;
  res *= 2;
}
```

Выводим ответ.

```
printf("%d\n",res);
```

2371. Черный квадрат

Вдохновленный шедевром Казимира Малевича "Черный квадрат", Петр Палевич решил создать собственную версию картины. Он приготовил полотно в виде прямоугольной сетки с $m \times n$ белыми квадратами – m строк по n ячеек каждая.

Петр покрасил некоторые клетки в черный цвет так, что черные ячейки сформировали квадрат размером $s \times s$ ячеек. Но на следующий день Петр разочаровался в своем творении и уничтожил его, разрезав полотно горизонтальными полосами размера $1 \times n$, после чего сжег их в камине.

На следующее утро Петр передумал и решил восстановить картину. Он попытался найти ее останки в камине, и, к счастью, одну из полос, а именно k-ую сверху, огонь не тронул.

Теперь Петр задумался, можно ли восстановить картину на основе этой полосы. Помогите ему сделать это.

Вход. Первая строка содержит четыре целых числа: m, n, s и k ($1 \le m, n \le 5000$, $1 \le s \le \min(m, n)$, $1 \le k \le m$).

Вторая строка содержит n символов и описывает k-ую строку картины, '.' означает белую клетку, '*' означает черную клетку.

Выход. Если изображение может быть однозначно восстановлено, то следует вывести "Unique". Если существует несколько вариантов восстановления картины, то вывести "Ambiguous". Если ни одной соответствующей картины не существует, вывести "Impossible".

Пример входа 1
```
4 4 1 2
..*.
```

Пример выхода 1
```
Unique
```

Пример входа 2
```
4 4 2 2
..**
```

Пример выхода 2
```
Ambiguous
```

Вычислим количество черных клеток в *k*-ой строке картины. Все они должны располагаться рядом, их количество должно равняться *s*.

Далее переберем все возможные позиции (*i*, *j*) левого верхнего угла квадрата и посчитаем количество вариантов расположения квадрата. Это число вариантов может равняться 0 (ответ Impossible), 1 (ответ Unique), или быть больше 1 (ответ Ambiguous).

Реализация алгоритма

k-ую строку картины будем хранить в массиве ss.

```
#define MAX 5010
char ss[MAX];
```

Читаем входные данные. Уменьшим значение *k* на 1 чтобы нумерация строк начиналась с нуля.

```
scanf("%d %d %d %d\n",&m,&n,&s,&k);
k--;
gets(ss);
```

Определим положение черных клеток: самая левая клетка будет в позиции *start*, самая правая в *finish*. Вычислим общее количество черных клеток *sum*.

```
sum = 0; start = finish = -1;
for(i = 0; i < n; i++)
  if (ss[i] == '*')
  {
    sum++;
    if (start == -1) start = i;
    finish = i;
  }
```

Рассмотрим случай, когда в *k*-ой строке имеются черные клетки. Тогда их количество *sum* должно равняться длине отрезка [*start*, *finish*], а также значению *s*.

```
if ((sum > 0) && ((sum != finish - start + 1) || (sum != s)))
{
  puts("Impossible");
  return 0;
}
```

Переберем все возможные позиции (*i*, *j*) левого верхнего угла квадрата и посчитаем количество вариантов *pos* расположения квадрата.

```
pos = 0;
for (i = 0; i < m - s + 1; i++)
for (j = 0; j < n - s + 1; j++)
```

Если *k*-ая строка расположена среди строк [*i*, *i* + *s* – 1], то она должна содержать черные клетки, причем самая левая позиция *start* черных клеток должна равняться *j*.

```
if ((i <= k) && (k <= i + s - 1))
{
  if (j == start) pos++;
}
else
{
```

Иначе *k*-ая строка не должна содержать черные клетки (в этом случае *start* остается равным -1).

```
  if (start == -1) pos++;
}
```

В зависимости от значения *pos* выводим ответ.

```
if (pos == 0)
  puts("Impossible");
else if (pos == 1)
  puts("Unique");
else
  puts("Ambiguous");
```