A Лотерея

Решение Python

```python
print(0)
```

B Найди кратные

Выведите l и $2l$.

Во-первых, минимальное значение $\frac{y}{x}$, которое можно получить, — это 2, а если какое-либо большее значение подходит, то 2 также подходит. Во-вторых, разность между x и $2x$ увеличивается с ростом x, тем самым понижая вероятность того, что оба числа поместятся в отрезок.

Асимптотика решения: $O(1)$.

Решение C++

```cpp
#include <iostream>

using namespace std;

int main() {
	int t;
	cin >> t;
	while (t--) {
		int l, r;
		cin >> l >> r;
		cout << l << " " << l * 2 << endl;
	}
	return 0;
}
```

Решение Python

```python
T = int(input())
for i in range(T):
	l, r = map(int, input().split())
	print(l, l * 2)
```

C Побег

Каждая сломанная стена увеличивает число связных областей плоскости максимум на 1. Поскольку в конце каждая клетка должна быть в той же области, что и внешнее пространство, в конце должна быть только одна связная область, а значит, ответ хотя бы nm (потому что вначале областей было $nm+1$).

Добиться цели, удалив ровно nm стен, можно, например, сломав верхнюю стену в каждой клетке.

Решение C++

```
#include <iostream>

using namespace std;
long long x, y, n;

int main() {
	cin >> n;
	for (int i = 0; i < n; i++) {
		cin >> x >> y;
		cout << x * y << endl;
	}
}
```

D Математическое ожидание

Если вероятность события a/b, то количество шагов для возникновения этого события b/a. Например для 6-гранного кубика матожидание количества шагов для выпадения 3 равно 6, так как вероятность выпадения 3 равна ⅙.

То есть дано n, нужно вывести n

Решение Python

```
print(input())
```

E Петя и орешки

Нужно посчитать количество орехов, которые не подходят и прибавить 1.

Решение C++

```
#include <iostream>
#include <string>
#include <vector>
#include <set>
#include <algorithm>
#include <map>
#include <stack>
#include <iomanip>
#define ll long long

using namespace std;
ll a, b, c, d, x, y, n, m[200200], k;
```

```
int main() {
	cin >> n >> x;
	for (int i = 0; i < n; i++) {
		cin >> m[i];
	}
	cin >> k;
	for (int i = 0; i < k; i++) {
		cin >> y;
		x -= m[y - 1];
	}
	cout << x + 1;
}
```

Решение Python

```
n, x = map(int, input().split())
ar1 = list(map(int, input().split()))
k = int(input())
ar2 = list(map(int, input().split()))
kek = set()
for elem in ar2:
    kek.add(elem)
ans = 1
for i in range(n):
    if i + 1 not in kek:
        ans += ar1[i]
print(ans)
```

F Вероятность

Нужно посчитать количество положительных исходов и поделить на общее количество (n*m)

Решение C++

```
#include <algorithm>
#include <iostream>

using namespace std;


int main() {

    int n, m, s;
    cin >> n >> m >> s;
    int cnt = 0;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            if (i + j == s)
                cnt++;
    printf("%.12lf\n", cnt * 1.0 / (n * m));

    return 0;

}
```

G Петя, Вася и факториалы

Все факториалы больше 1 четные, поэтому Петя может выиграть только при n = 1, а при n = 2 будет ничья. Во всех остальных случаях он проигрывает.

Решение C++

```
#include <iostream>
#include <string>

using namespace std;
int n,

int main() {
	cin >> n;
	if (n == 1) {
		cout << "Win";
		return 0;
	}
	if (n == 2) {
		cout << "Draw";
		return 0;
	}
	cout << "Lose";
}
```

Решение Python

```
n = int(input())
if n == 1:
    print('Win')
elif n == 2:
    print('Draw')
else:
    print('Lose')
```

H Случайные команды

Если переформулировать задачу в терминах теории графов, то ее можно сформулировать следующим образом: имеется граф, состоящий из *n* вершин и *m* компонент связности. Внутри каждой компоненты связности каждая пара вершин этой компоненты связана ребром. Другими словами, каждая компонента связности является полносвязной. Какое наименьшее и какое наибольшее количество ребер может содержать такой граф? Рассмотрим процесс построения графа из *n* вершин и *m* компонент связности. Для начала предположим, что каждая из *m* компонент содержит ровно одну вершину. Остается распределить оставшиеся *n* - *m* вершин так, чтобы минимизировать или максимизировать количество ребер. Заметим, что при добавлении новой вершины в компоненту связности размера *k*, количество ребер увеличивается

на *k* (новая вершина соединяется с каждой из уже существующих одним ребром). Следовательно, для того, чтобы минимизировать количество образованных ребер на каждом шаге, требуется каждый раз добавлять вершину в компоненту связности наименьшего размера. Если действовать согласно такой стратегии, то после распределения вершин по компонентам связности появится $n \mod m$компонент размера $\lceil \frac{n}{m} \rceil$ и $n - (n \mod m)$ компонент размера $\lfloor \frac{n}{m} \rfloor$. Аналогично, для того, чтобы максимизировать количество ребер, на каждом шаге необходимо добавлять очередную вершину в компоненту связности наибольшего размера. Если действовать согласно такой стратегии, то образуется одна компонента связности размера *n* - *m* + 1, оставшиеся компоненты связности будут состоять из одной вершины. Зная количество компонент связности и их размеры, можно посчитать общее количество ребер. Для полносвязной компоненты, состоящей из *k* вершин, количество ребер равняется $\frac{k \cdot (k-1)}{2}$. Следует помнить про необходимость использовать 64-битный тип данных для хранения количества ребер, которое квадратично зависит от значения *n*.

Решение C++

```
#include<iostream>
using namespace std;

long long comb(long long n)
{
	long long k = n * (n - 1) / 2;
	return k;
}
int main()
{
	long long n, m;
	cin >> n >> m;
	long long min = ((m - (n % m)) * comb(n / m)) + ((n % m) * comb(n / m + 1));
	long long max = comb(n - m + 1);
	cout << min << " " << max << endl;
}
```

Решение Python

```
n,m=map(int,input().split())
a=n//m
print( m*a*(a-1)//2+a*(n%m)  , (n-m)*(n-m+1)//2 )
```

I Петя, Вася, скажи орехам нет

По сути, просто разбор двух случаев.

Решение Python

```
a, b = map(int, input().split())
if a >= b:
    if b == 0:
        print(2 * a + 1)
    else:
        print(0)
else:
    if 2 * a > b:
        print(0)
    else:
        print(b + 1)
```

J Сброс наковальни

В этой задаче нужно было определить вероятность того, что уравнение $x^2 + \sqrt{p} \cdot x + q = 0$ имеет хотя бы один действительный корень, при условии что величины p и q выбирались равновероятно и независимо в своих интервалах [0;a] и [- b;b].

Для этого необходимо и достаточно чтобы дискриминант D = p - 4q был не меньше нуля. Для решения этой задачи можно было нарисовать на плоскости (p, q) прямоугольник с вершинами в точках (0, - b), (a, - b), (a, b) и (0, b) и линию p = 4q. Каждая точка прямоугольника соответствует возможным значениям p и q, а линия делит всю плоскость на две части - где уравнение имеет действительные корни, и где оно их не имеет. Тогда вследствие равновероятности и независимости выбора p и q ответ есть площадь пересечения прямоугольника с областью p ≥ 4q, отнесенная к площади самого прямоугольника, в случае его невырожденности (a, b ≠ 0).

Если ровно одно из чисел a или b равно нулю, прямоугольник вырождается в отрезок, и искомая вероятность равна отношению длины пересечения отрезка с областью p ≥ 4q и всего отрезка. В случае a = b = 0 ответ на задачу, очевидно, равен 1.

Асимптотическая сложность решения - O(t)

Решение C++

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

using namespace std;

int main()
{
    int t;
    scanf("%d", &t);
    while (t--) {
        double a, b;
        scanf("%lf%lf", &a, &b);
        if (a * b) {
            if (b >= a / 4) {
                printf("%0.6f\n", 0.5 + a / (16 * b));
            }
            else {
                printf("%0.6f\n", 1 - b / a);
            }
        }
        else {
            if (b == 0)
                printf("1\n");
            else
                printf("0.5\n");
        }
    }
    return 0;
}
```

Решение Python

```
    for i in range(int(input())):
      a,b = (map(int,input().strip().split(' ')))
      if(b==0):
        print(1)
      elif(a==0):
        print(.5)
      elif(a<=4*b):
        print(((8*b)+a)/16/b)
      else:
        print((a-b)/a)
```

K Мешок мышек

Пусть на каком-то шаге итерации в мешке остается B черных и W белых мышек и вероятность попасть в это состояние равна P (изначально P = 1, W и B равны начальным значениям). Вероятность вытащить белую мышь на этом шаге равна P * W / (B + W) (это уже не условная вероятность "при условии того, что мы находимся в этом состоянии", а абсолютная). Если очередь принцессы тянуть, то эта вероятность прибавляется к вероятности ее победы, если очередь дракона, вероятность победы принцессы не меняется. Для перехода на следующий шаг итерации нам нужно, чтобы игра не закончилась, то есть на

текущем шаге вытащили черную мышь, и количество черных мышей уменьшилось, а вероятность попасть на новый шаг итерации умножилась на B / (B + W). Проитерировав так до того, как черные мыши закончатся, получим ответ.

Увы, мышки в мешке явно ведут себя не так флегматично, как шары. Это вносит элемент неопределенности — мы не знаем точно, в какое состояние перейдет игра после того, как дракон и принцесса вытянут своих мышей и перейдут на следующий этап игры. Поэтому решение должно быть рекурсивным (или динамическим программированием, кто как любит). Ходы принцессы и дракона обрабатываются аналогично, но после этого нужно скомбинировать результаты решения подзадач (W — 1, B — 2) и (W, B — 3). Код функции:

Сложность рекурсии с мемоизацией равна количеству разных пар аргументов, с которыми ее можно вызвать, т.е. O(WB).

Решение C++

```
map<pair<int, int>, double> memo;

double p_win_1_rec(int W, int B) {
    if (W <= 0) return 0;
    if (B <= 0) return 1;
    pair<int, int> args = make_pair(W, B);
    if (memo.find(args) != memo.end()) {
        return memo[args];
    }
    // we know that currently it's player 1's turn
    // probability of winning from this draw
    double ret = W * 1.0 / (W + B), cont_prob = B * 1.0 / (W + B);
    B--;
    // probability of continuing after player 2's turn
    cont_prob *= B * 1.0 / (W + B);
    B--;
    // and now we have a choice: the mouse that jumps is either black or white
    if (cont_prob > 1e-13) {
        double p_black = p_win_1_rec(W, B - 1) * (B * 1.0 / (W + B));
        double p_white = p_win_1_rec(W - 1, B) * (W * 1.0 / (W + B));
        ret += cont_prob * (p_black + p_white);
    }
    memo[args] = ret;
    return ret;
}
```

Решение Python

```
w, b = [int(i) for i in input().split()]


dp = [[-1 for i in range(b+1)] for j in range(w+1)]

def f(w, b):
    #if w<0 or b<0:
    #    return 0
    if w==0 and b==0:
        return 0
    if b<=0:
        return 1
    if w == 0:
        return 0
    if dp[w][b] != -1:
        return dp[w][b]
    ans = w/(w+b)
    if w>=1 and b>=2:
        ans += (b/(w+b))*((b-1)/(w+b-1))*((w/(w+b-2)) * f(w-1, b-2))
    if b>=3:
        ans += (b/(w+b))*((b-1)/(w+b-1))*((b-2)/(w+b-2))* f(w, b-3)
    dp[w][b] = ans
    return ans

print(f(w, b))
```