# Турнир Архимеда 2016. Разбор задач.

Представляем вашему вниманию разбор задач Турнира Архимеда по программированию 2016.

Турнир подготовлен под руководством Андрея Станкевича командой школьников Санкт-Петербурга, призёров и победителей Всероссийской олимпиады и международных соревнований по информатике. Турнир готовили: Михаил Анопренко, Арсений Кириллов, Никита Михайлов, Александр Морозов, Анна Никифоровская, Алексей Трилис, Игорь Тух и Михаил Ютман. Мы также благодарим за помощь в подготовке и за прорешивание турнира студентов Николая Будина, Ивана Буракова, Николая Ведерникова, Никиту Костливцева, Апполинарию Романову, Мехрубона Тураева. Координатор проведения на системе Яндекс.Контест: Лидия Перовская. Задачи подготовлены с помощью системы Polygon, автор Михаил Мирзаянов.

Ниже приведены разборы всех задач турнира. Везде мы старались привести как можно более простое и элегантное решение. Если программа не проходит тесты жюри и приведена как неправильное решение, она обведена в красную рамку.

# Разбор задачи «Алёна, помни возраст Вити!»

Автор задачи: Анна Никифоровская
Подготовка тестов и решений: Анна Никифоровская
Автор разбора: Андрей Станкевич

Пусть с тех пор, как Вите было $n$ лет, а его брату $m$, прошло $x$ лет.

**Решение 1, математическое.** Тогда сегодня Вите $n+x$ лет, а его брату $m+x$. По условию $k(n+x)=m+x$, значит $kn+kx=m+x$, то есть $(k-1)x=m-kn$. Таким образом $x=(m-kn)/(k-1)$. Если $m-kn$ не делится на $k-1$, или $m-kn \leqslant 0$, то описанной ситуации произойти не могло, следует вывести $-1$. Иначе выводим ответ: $n+x$.

Приведем код такого решения на языке Python.

```
n = int(input())
m = int(input())
k = int(input())

if m <= k * n or (m - k * n) % (k - 1) != 0:
    print(-1)
else:
    x = (m - k * n) // (k - 1)
    print(n + x)
```

**Решение 2, программистское.** Переберем значение $x$, начиная с 1. Для очередного вариант значения $x$ проверим, выполняется ли равенство $k(n+x)=m+x$. Если это равенство выполнено, то выводим $n+x$. Осталось понять, до какой верхней границы имеет смысл перебирать $x$. Заметим, что если $x>m$, то $k(n+x) \geqslant 2(n+x) > 2x > m+x$, поэтому равенство $k(n+x)=m+x$ выполниться не может. Таким образом, достаточно перебрать $x$ от 1 до $m$. Если ни один $x$ не подошёл, выводим $-1$.

Приведем код такого решения на языке Python.

```
n = int(input())
m = int(input())
k = int(input())

for x in range(1, m + 1):
    if k * (n + x) == m + x:
        print(n + x)
        exit()

print(-1)
```

**Решение 3, даже слишком программистское.** Рассмотрим функцию $f(x)=k(n+x)-m-x$. Заметим, что нам надо найти значение $x$, при котором $f(x)=0$. Также заметим, что $f(x)=kn+kx-m-x=kn-m+(k-1)x$ - возрастающая функция от $x$, поэтому искомое значение можно найти двоичным поиском. Границы двоичного поиска можно установить, используя те же рассуждения, что и во втором варианте решения: ответ лежит между 1 и $m$. Если уже для $x=1$ выполнено $f(x)>0$, то решения нет, аналогично решения нет, если $f(x)=0$ не выполняется для целых $x$.

Наблюдение, что $f(x)$ возрастает можно использовать и при переборе всех значений $x$ в качестве критерия остановки: как только $k(n+x)$ стало больше $m+x$, можно остановиться.

# Разбор задачи «Тройной Фибоначчи»

Автор задачи: Игорь Тух
Подготовка тестов и решений: Михаил Ютман
Авторы разбора: Арсений Кириллов, Андрей Станкевич

Попробуем простое решение: найдем все числа Фибоначчи от первого до $R$-го, сложим их в массив и посчитаем, сколько из чисел в интервале от $L$ до $R$ делятся на 3. Напишем такое решение, например, на паскале.

```
program fib3;

var
    fib: array [1..100000] of longint;
    L, R, i, ans: longint;

begin
    read(L, R);

    fib[1] := 1;
    fib[2] := 2;
    for i := 3 to R do
        fib[i] := fib[i - 1] + fib[i - 2];

    ans := 0;
    for i := L to R do
        if fib[i] mod 3 = 0 then
            ans := ans + 1;
    writeln(ans);
end.
```

Программа работает на небольших тестах, но выдает неверный ответ на больших тестах (например, на 12 тесте жюри, если отправить её на проверку). В чём же дело? Посмотрев на содержимое массива `fib`, видно, что числа Фибоначчи растут очень быстро и происходит переполнение типа `longint`. Даже использование 64-битного типа не решает проблему, числа растут слишком быстро.

Что если попробовать написать решение на языке Python, где ограничений по размеру чисел, которые может содержать целочисленный тип, нет? Попробуем.

```
L = int(input())
R = int(input())

fib = [0] * (R + 1)
fib[1] = 1
fib[2] = 2
for i in range(3, R + 1):
    fib[i] = fib[i - 1] + fib[i - 2]

ans = 0
for i in range(L, R + 1):
    if fib[i] % 3 == 0:
        ans += 1
print(ans)
```

К сожалению, эта программа тоже не проходит тесты жюри, на этот раз на больших тестах превышается ограничение по времени — получаются слишком длинные числа, для которых сложение и взятие остатка по модулю 3 работает слишком медленно.

Таким образом, вычислить все числа Фибоначчи с номерами от $L$ до $R$ и для каждого из них проверить, делится ли оно на три, не получается. Нужно применить какой-то другой подход.

**Арифметика остатков.** Первый подход основывается на следующей математической идее. Вместо того, чтобы считать сами числа Фибоначчи, будем считать их остатки от деления на три. Тогда переполнения не произойдет, и числа длинными становиться не будут, поэтому после соответствующей модификации любая из двух предыдущих программ легко решит задачу. Исправим, например, программу на паскале. Получим следующее решение.

```
program fib3;

var
    fib: array [1..100000] of longint;
    L, R, i, ans: longint;

begin
    read(L, R);

    fib[1] := 1;
    fib[2] := 2;
    for i := 3 to R do
        fib[i] := (fib[i - 1] + fib[i - 2]) mod 3;

    ans := 0;
    for i := L to R do
        if fib[i] = 0 then
            ans := ans + 1;
    writeln(ans);
end.
```

**Поиск закономерности.** Попробуем зайти с другой стороны. Выпишем несколько первых чисел Фибоначчи и выделим те из них, которые делятся на 3. Получим последовательность 1, 2, **3**, 5, 8, 13, **21**, 34, 55, 89, **144**, 233, ....

Легко видеть, что, начиная с третьего, каждое четвертое число делится на три. Таким образом, число Фибоначчи делится на 3, если его номер даёт остаток 3 по модулю 4.

Докажем это. Последовательность чисел Фибоначчи по модулю 3 начинается так: 1, 2, 0, 2, 2, 1, 0, 1, 1, 2, .... Заметим, что $F_1 \equiv F_9 \pmod 3$ и $F_2 \equiv F_{10} \pmod 3$. Значит дальше остатки по модулю 3 будут повторяться с периодом 8. Среди первых 8 чисел на 3 делятся только числа Фибоначчи с номерами 3 и 7.

Итак, можно просто посчитать количество чисел, которые дают остаток 3 по модулю 4 на отрезке от $L$ до $R$.

```
L = int(input())
R = int(input())

ans = 0
for i in range(L, R + 1):
    if i % 4 == 3:
        ans += 1
print(ans)
```

Наконец, воспользуемся поводом рассказать об еще одном полезном приёме, который можно (хотя, как мы видели, и не обязательно) применить в этой задаче.

Пусть нам требуется посчитать чисел с каким-либо свойством на отрезке от $L$ до $R$. Тогда можно посчитать количество таких чисел на отрезке от 1 до $R$ и на отрезке от 1 до $L - 1$, и вычесть из первого значения второе. В случае этой задачи получаем такое, совсем простое, решение.

```
L = int(input())
R = int(input())

print((R + 1) // 4 - L // 4)
```

# Разбор задачи «Марсианские нолики»

Автор задачи: Михаил Анопренко
Подготовка тестов и решений: Михаил Анопренко
Автор разбора: Николай Будин

Чтобы решить эту задачу, полезно сначала посмотреть, что происходит в привычной нам десятичной системе счисления. Выпишем несколько первых чисел, которые заканчиваются хотя бы на два нуля: 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, .... Легко понять, что начало этого ряда совпадает с началом натурального ряда, просто в конец каждого числа дописано по два нуля. Таким образом, $i$-е такое число просто равно $100i$.

Обобщим теперь это рассуждение на произвольное основание системы счисления и произвольное число нулей. Так как искомое число заканчивается на $n$ нулей в $k$-ичной системе счисления, оно делится на $k^n$. Поэтому ответом является $i$-е натуральное число, делящееся на $k^n$, то есть $i \cdot k^n$.

Решение задачи на языке Python.

```
k = int(input())
n = int(input())
i = int(input())

print(i * k**n)
```

На языках, где значения в целочисленном типе ограничены, следует воспользоваться 64-битным типом данных, чтобы не пострадать от переполнения. Например, решение на паскале может выглядеть так.

```
program zeroes;

var
    i, n, k, j: longint;
    ans: int64;

begin
    read(k, n, i);
    ans := i;
    for j := 1 to n do
        ans := ans * k;
    writeln(ans);
end.
```

На языке C++ следует использовать тип данных `long long`.

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

int main() {
    int k, n, i;
    cin >> k >> n >> i;

    long long ans = i;
    for (int j = 0; j < n; j++) {
        ans = ans * k;
    }

    cout << ans << endl;
}
```

В качестве заключительного замечания отметим, что использовать вещественный тип данных и функцию возведения в степень из стандартный библиотеки паскаля или C++ не следовало. Так как ответ является целым числом, лучше использовать только целочисленные типы в решении.

# Разбор задачи «Спорт или еда»

| | |
|---|---|
| Автор задачи: | Михаил Ютман |
| Подготовка тестов и решений: | Алексей Трилис |
| Автор разбора: | Игорь Тух |

Заметим, что если, Арсений поел, он не может больше тренироваться вообще. Иначе рассмотрим первый момент времени после еды, когда Арсений тренировался. Тогда непосредственно перед этим он поел, получаем противоречие. Тогда нетрудно понять, что расписание будет выглядеть следующим образом: сначала Арсений некоторое, возможно нулевое, время тренировался, а затем все оставшиеся пункты расписания ел.

Переберем сколько пунктов расписания Арсений тренировался. Обозначим это число за $T$. Расписание Арсения однозначно восстанавливается по значению $T$. Все что осталось — научиться определять, сколько среди первых $T$ пунктов Арсений тренировался по первоначальному расписанию, и сколько пунктов расписания среди оставшихся $n - T$ он ел. Если просто пробежаться по расписанию и посчитать, то решение будет работать за $O(n^2)$, что слишком много и не укладывается в ограничение по времени. Рассмотрим более эффективный подход.

Будем использовать префиксные и суффиксные суммы. Перед циклом, перебирающим $T$, посчитаем массивы $pref$ и $suf$, $pref[i]$ будет равно количеству пунктов начального расписания среди первых $i$, когда Арсений ел, и $suf[i]$ будет равно количеству пунктов расписания среди последних $i$ (от $(n - i + 1)$-го до $n$-го, включительно), когда Арсений тренировался.

Тогда интересующее нас количество пунктов, которые нам надо изменить равно $pref[T] + suf[n - T]$. Для восстановления ответа достаточно выбрать минимальное значение этого выражения по всем $T$, таким, что $0 \leqslant T \leqslant n$. Соответствующее расписание будет состоять из $T$ пунктов тренировок в начале и $n - T$ пунктов поедания пищи в конце.

Приведем код программы на языке Python.

```
n = int(input())
s = input()

pref = [0]
suf = [0]
for i in range(n):
    pref.append(pref[i])
    if s[i] == 'e':
        pref[-1] += 1
for i in range(n):
    suf.append(suf[i])
    if s[n - 1 - i] == 't':
        suf[-1] += 1

best = suf[n]
bestT = 0
for T in range(n + 1):
    if pref[T] + suf[n - T] < best:
        best = pref[T] + suf[n - T]
        bestT = T

print(best)
print("t" * bestT + "e" * (n - bestT))
```

# Разбор задачи «Сборная Юпитера»

| | |
|---|---|
| Автор задачи: | Михаил Анопренко |
| Подготовка тестов и решений: | Михаил Анопренко |
| Автор разбора: | Михаил Ютман |

Это пример «задачи на реализацию», где в условии написано, что требуется сделать, необходимо лишь реализовать описанное.

Для каждого ученика будем хранить структуру, состоящую из трех чисел: номера ученика, номер его класса и его суммарный балл по всем отборочным турам. После этого с помощью написанной вручную или встроенной в язык сортировки отсортируем массив таких структур в порядке убывания суммарного балла. Отберем первую команду, взяв в неё четырех лучших, затем пройдём по ученикам не из 11 класса и отберём четырёх лучших, которые не попали в первую команду. Заметим, что это всегда возможно, так как по условию гарантируется, что есть хотя бы 8 учеников не из 11 класса.

Не забудем перед выводом отсортировать номера участников внутри каждой сборной в порядке возрастания.

Приведем пример программы на языке Python.

```
n = int(input())
alls = []
no11 = []
for i in range(n):
    x = list(map(int, input().split()))
    p = (sum(x[1:]), x[0], i + 1)
    alls.append(p)
    if p[1] != 11:
        no11.append(p)

alls.sort(reverse=True)
no11.sort(reverse=True)

res = [[], []]
for i in range(4):
    res[0].append(alls[i][2])

i = 0
while len(res[1]) < 4:
    if not no11[i][2] in res[0]:
        res[1].append(no11[i][2])
    i += 1

res[0].sort()
res[1].sort()

print('␣'.join(map(str, res[0])))
print('␣'.join(map(str, res[1])))
```

# Разбор задачи «Маша и матрёшки»

| | |
|---|---|
| Автор задачи: | Никита Михайлов |
| Подготовка тестов и решений: | Анна Никифоровская |
| Автор разбора: | Анна Никифоровская |

Заметим, что в одном наборе вложенных друг в друга матрёшек не может быть двух матрёшек одинакового размера. Действительно, в каждую матрёшку непосредственно внутрь неё вкладывается меньшая по размеру, а значит внутри каждой все матрёшки строго меньшие её по размеру, снаружи — строго большие. Таким образом, ответ не может быть больше количества различных размеров матрёшек. С другой стороны, если взять по одной матрёшке каждого имеющегося в наличии размера, то их все можно будет вложить друг в друга, как и требуется по условию.

Итак, ответ — количество различных размеров матрёшек, которые есть у Маши. Осталось понять, как определить это количество.

**Решение 1.** Сначала отсортируем массив размеров матрёшек. Подходит любая сортировка со временем работы $O(n^2)$ и быстрее, например сортировка пузырьком или выбором, а также встроенная в стандартную библиотеку. Теперь, просматривая массив размеров, мы обнаруживаем новый размер матрёшки в первом элементе, а также когда видим значение, не равное предыдущему.

Код программы на паскале с использованием сортировки пузырьком.

```
program matryoska;

var
    a: array [1..1000] of longint;
    i, j, n, t, ans: longint;
begin
    read(n);
    for i := 1 to n do
        read(a[i]);
    for i := 1 to n do
        for j := 1 to n - i do
            if a[j] > a[j + 1] then begin
                t := a[j]; a[j] := a[j + 1]; a[j + 1] := t;
            end;

    ans := 0;
    for i := 1 to n do
        if (i = 1) or (a[i] <> a[i - 1]) then
            inc(ans);

    writeln(ans);
end.
```

Код программы на языке Python с использованием встроенной сортировки.

```
n = int(input())
a = list(map(int, input().split()))
a.sort()

ans = 0
for i in range(n):
    if i == 0 or a[i] != a[i - 1]:
        ans += 1
print(ans)
```

**Решение 2.** Также количество различных элементов можно посчитать следующим образом. Заведем массив *cnt* размером 10 000, изначально заполненный нулями. При считывании массива размеров матрёшек, будем увеличивать на один элемент с номером $a_i$ в массиве *cnt*. Тогда в конце значение $cnt[i]$ будет содержать число различных матрёшек размера $i$. Осталось посчитать количество ненулевых элементов массива *cnt*.

Приведем пример кода на паскале.

```
program matryoska2;

const
    MAX = 10000;

var
    i, k, n, ans: longint;
    cnt: array [1..MAX] of longint;

begin
    readln(n);
    for i := 1 to n do
    begin
        read(k);
        inc(cnt[k]);
    end;

    ans := 0;
    for i := 1 to MAX do
        if cnt[i] > 0 then inc(ans);
    writeln(ans);
end.
```

# Разбор задачи «ASCII-графика»

| | |
|---|---|
| Автор задачи: | Михаил Анопренко, на базе идей Владимира Гуровица |
| Подготовка тестов и решений: | Михаил Анопренко, использованы материалы NEERC 2011, подготовленные Павлом Мавриным |
| Автор разбора: | Андрей Станкевич |

При кажущейся внешней простоте, попытка выделить стороны многоугольника из ASCII-картинки оказывается довольно неприятной и может привести к программе в 50–100 строк. Однако простой трюк позволяет радикально упростить решение и получить простую и короткую программу.

Будем вместо подсчёта числа сторон многоугольника считать число его вершин. Чтобы определить вершину многоугольника, достаточно посмотреть на два соседних символа картинки. Например, если два соседних по горизонтали символа «/» и «\», то это показывает, что в верхней точке общей стороны этих ячеек находится вершина многоугольника. Всего, таким образом, получается четыре случая, два для соседних по горизонтали ячеек и два для соседних по вертикали.

Рассмотрим пример программы на Паскале.

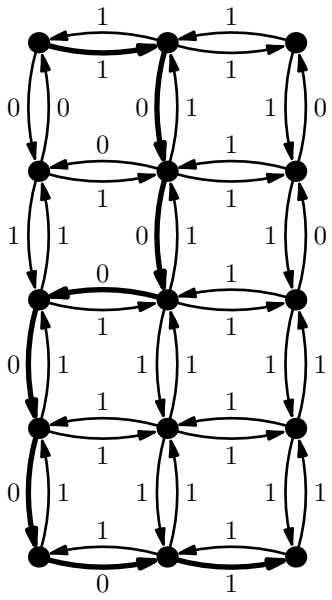
```
program ascii;

var
    i, j, h, w, ans: longint;
    a: array[1..200] of string;

begin
    readln(h, w);
    for i := 1 to h do
        readln(a[i]);

    ans := 0;
    for i := 1 to h do
        for j := 1 to w do begin
            if (j < w) and (a[i][j] = '/') and (a[i][j + 1] = '\') then
                inc(ans);
            if (i < h) and (a[i][j] = '/') and (a[i + 1][j] = '\') then
                inc(ans);
            if (j < w) and (a[i][j] = '\') and (a[i][j + 1] = '/') then
                inc(ans);
            if (i < h) and (a[i][j] = '\') and (a[i + 1][j] = '/') then
                inc(ans);
        end;

    writeln(ans);
end.
```

# Разбор задачи «Робот»

| | |
|---|---|
| Автор задачи: | Михаил Ютман |
| Подготовка тестов и решений: | Арсений Кириллов |
| Автор разбора: | Арсений Кириллов, Андрей Станкевич |

Один из подходов к этой задаче — использовать какой-нибудь из базовых алгоритмов теории графов. Давайте представим полигон в виде ориентированного взвешенного графа, где вершинам будут соответствовать клетки, а рёбрам — переходу из клетки в соседнюю. Тогда веса рёбер будут 0 и 1. Необходимое количество литров топлива соответствует минимальному расстоянию от стартовой вершины до конечной.

Например, для теста из примера получается следующий граф, кратчайший путь помечен жирными ребрами:



Для поиска кратчайшего пути в таком графе можно применить разные подходы.

Стандартным подходом является модификация алгоритма *поиска в ширину*. Вспомним, как работает этот алгоритм. Вершины графа хранятся в очереди, исходно туда помещается стартовая вершина. Извлекая очередную вершину из очереди, просматриваем вершины, в которые из неё идет ребро и добавляем в конец очереди ранее не посещенные из этих вершин. Алгоритм поиска в ширину корректно работает только для случая *невзвешенных* графов, когда веса всех ребер равны 1. Это не наш случай: в этой задаче веса ребер могут быть 0 и 1.

Можно модифицировать поиск в ширину для того, чтобы он работал для такого варианта весов рёбер. Будем вместо очереди использовать *дек*. Если ребро, которое мы просматриваем, имеет вес 0, то, добавляем вершину, в которую это ребро ведёт, в начало дека, а для рёбер веса 1, как и раньше добавляем в конец. После такой модификации расстояние до всех вершин будет найдено правильно.

Заметим, что основным достоинством поиска в ширину является линейное время работы. В то же время в этой задаче ограничения на входные данные довольно маленькие, поэтому можно было применить и другие, менее эффективные алгоритмы. Например, алгоритм Дейкстры, алгоритм Флойда или алгоритм Форда-Беллмана. Много доступного теоретического материала про алгоритмы на графах можно, например, найти на сайте `http://informatics.mccme.ru`.

Однако решать эту задачу можно было и не формулируя в явном виде ее в виде задачи поиска кратчайшего пути в графе. Заведём массив $dist[i][j]$, в котором будем хранить минимальное количество литров топлива, которое необходимо потратить, чтобы добраться от стартовой клетки до клетки $(i, j)$. Исходно заполним массив большими числами, а ячейку, соответствующую стартовой клетке, заполним значением 0. Будем повторять следующее действие: пройдем по всем клеткам поля и попробуем переместить из них робота по каждому из четырех направлений. Пусть мы рассматриваем клетку $(i, j)$ и $dist[i][j] = u$. Для клетки, в которую мы попадаем, заменим текущее значение

в массиве *dist* на минимум из него и значения $u$ при перемещении по направлению ускорителя, на минимум из него и значения $u + 1$ при перемещении в любом другом направлении. Повторяем это действие, пока массив dist не перестанет меняться.

Приведем пример кода на языке Python, реализующего эту идею. Код также демонстрирует некоторые идеи, помогающие реализовать обходы на прямоугольных таблицах, так что полезно изучить его и с этой точки зрения.

```
h, w = map(int, input().split())
a = [input() for i in range(h)]

dx = [-1, 0, 1, 0]
dy = [0, -1, 0, 1]
sides = "NWSE"

FAR = 10**9
dist = [[FAR] * w for i in range(h)]
dist[0][0] = 0

changed = True
while changed:
    changed = False
    for i in range(h):
        for j in range(w):
            for d in range(4):
                ni = i + dx[d]
                nj = j + dy[d]
                if (0 <= ni < h) and (0 <= nj < w):
                    cost = 0 if a[i][j] == sides[d] else 1
                    if dist[ni][nj] > dist[i][j] + cost:
                        dist[ni][nj] = dist[i][j] + cost
                        changed = True
print(dist[-1][-1])
```

# Разбор задачи «Нужно больше золота»

| | |
|---|---|
| Автор задачи: | Игорь Тух |
| Подготовка тестов и решений: | Никита Михайлов |
| Автор разбора: | Игорь Тух |

Обозначим суммарную ценность всех магических артефактов за $s$, а конечную магическую силу героя за $p$.

Докажем сначала, что в оптимальном решении Петя сначала активирует некоторые артефакты с помощью магии, а затем активирует оставшиеся артефакты с помощью силы.

Действительно, пусть в момент активации с помощью силы некого артефакта ценности $w$ магическая сила героя равнялась $x$. Так как ценности всех артефактов положительны, то $x \leqslant p$, тогда и $xw \leqslant pw$. То есть Петя, активируя артефакты с помощью силы уже после того, как он активировал все артефакты, которые он планировал активировать с помощью магии, получит монет не меньше, чем если бы он активировал их в другой момент. При этом он должен активировать все артефакты, так как если активировать с помощью силы один из еще не активированных артефактов, то количество полученных монет не уменьшится.

Итак, пусть суммарная ценность активированных героем Пети с помощью магии артефактов равняется $p$, тогда суммарная ценность активированных с помощью силы артефактов будет равна $s-p$, тогда Петя получит ровно $(s-p)p$ золотых монет. Осталось найти оптимальное значение $p$. Для этого научимся для всех $p$ из диапазона от 0 до $s$ определять, может ли герой Пети обладать конечной силой магии $p$. Это можно сделать с помощью динамического программирования. Аналогичный метод применяется при решении задачи о рюкзаке.

Решение с помощью динамического программирования устроено следующим образом: обозначим как $dp[i][p]$ логическое значение, верно ли, что можно активировав с помощью магии некоторые из первых $i$ артефактов, получить силу героя $p$. Тогда $dp[0][0] = \text{True}$, а $dp[0][p] = \text{False}$ для $p > 0$. Формула же для пересчёта выглядит так. Если $p < w_i$, то $dp[i][p] = dp[i-1][p]$, а иначе:

$$dp[i][p] = dp[i-1][p] \texttt{ or } dp[i-1][p-w_i]$$

Теперь переберем все значения $p$ от 0 до $s$ и выберем то, для которого $dp[n][p] = \text{True}$ и значение $p(s-p)$ максимально.

Приведем код решения на языке Python.

```
n = int(input())
w = list(map(int, input().split()))

s = sum(w)

dp = [[False] * (s + 1) for i in range(n + 1)]
dp[0][0] = True
for i in range(1, n + 1):
    for j in range(s + 1):
        if w[i - 1] > j:
            dp[i][j] = dp[i - 1][j]
        else:
            dp[i][j] = dp[i - 1][j] or dp[i - 1][j - w[i - 1]]

ans = 0
for p in range(s + 1):
    if dp[n][p] and p * (s - p) > ans:
        ans = p * (s - p)
print(ans)
```

# Разбор задачи «Башни»

| | |
|---|---|
| Автор задачи: | Михаил Анопренко |
| Подготовка тестов и решений: | Михаил Анопренко |
| Авторы разбора: | Иван Бураков, Андрей Станкевич |

Переформулируем условие математически. Задан массив целых чисел $h$, длины $n$. Необходимо найти количество троек $i < j < k$, для которых верно, что $h_i < h_j < h_k$.

Заметим, что просто перебрать все тройки в этой задаче не получается, решение совершает слишком много действий и не укладывается в отведенное время. Тем не менее, рассмотрим программу на C++, которая перебирает все тройки, поскольку она поможет нам перейти к более быстрому решению.

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

int main() {
    int n;
    cin >> n;
    vector<int> h(n);
    for (int i = 0; i < n; i++) {
        cin >> h[i];
    }

    long long ans = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            for (int k = j + 1; k < n; k++) {
                if (h[i] < h[j] && h[j] < h[k])
                    ans++;
            }
        }
    }

    cout << ans << endl;
}
```

Заметим, что проверка $h[i] < h[j]$ во внутреннем цикле не зависит от $k$ и её можно вынести из цикла. Получаем другое кубическое решение, которое тоже не укладывается в ограничение по времени.

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

int main() {
    int n;
    cin >> n;
    vector<int> h(n);
    for (int i = 0; i < n; i++) {
        cin >> h[i];
    }

    long long ans = 0;
```

```
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (h[i] < h[j]) {
                for (int k = j + 1; k < n; k++) {
                    if (h[j] < h[k])
                        ans++;
                }
            }
        }
    }

    cout << ans << endl;
}
```

Однако уже видна оптимизация, которая позволит получить решение за $O(n^2)$. Заметим, что внутренний цикл по $k$ добавляет к ответу число таких $k$, что $j < k$ и $h[j] < h[k]$. Посчитаем это количество заранее для всех $j$ в массиве $cnt$. Этот предподсчёт можно выполнить за $O(n^2)$. Теперь вместо цикла по $k$ будем просто прибавлять к ответу $cnt[j]$. Получаем следующее решение, которое укладывается в установленные ограничения.

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

int main() {
    int n;
    cin >> n;
    vector<int> h(n);
    for (int i = 0; i < n; i++) {
        cin >> h[i];
    }

    vector<int> cnt(n);
    for (int j = 0; j < n; j++) {
        for (int k = j + 1; k < n; k++) {
            if (h[j] < h[k])
                cnt[j]++;
        }
    }

    long long ans = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (h[i] < h[j]) {
                ans += cnt[j];
            }
        }
    }

    cout << ans << endl;
}
```

Рассмотрим еще один подход к решению за $O(n^2)$, который основывается на другой идее. Переберём $j$. Тогда количество троек, в которых $j$ участвует в качестве среднего элемента, равно произ-

ведению количества $i$, таких что $i < j$ и $h[i] < h[j]$, и количества $k$, таких что $j < k$ и $h[j] < h[k]$. Количество таких $i$ и $k$ можно найти за один линейный проход.

Получаем следующую программу.

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

int main() {
    int n;
    cin >> n;
    vector<int> h(n);
    for (int i = 0; i < n; i++) {
        cin >> h[i];
    }

    long long ans = 0;
    for (int j = 0; j < n; j++) {
        int cnti = 0;
        for (int i = 0; i < j; i++)
            if (h[i] < h[j])
                cnti++;
        int cntk = 0;
        for (int k = j + 1; k < n; k++)
            if (h[j] < h[k])
                cntk++;
        ans += cnti * cntk;
    }

    cout << ans << endl;
}
```

Отметим напоследок, что у этой задачи есть решение и за $O(n \log n)$, но его написание при $n \leqslant 8000$ не требовалось.