Комбинаторика. Формула

1289. Ланч

Влад хочет взять с собой для ланча пару фруктов. У него есть a бананов, b яблок и c груш. Сколькими способами он может выбрать 2 разных фрукта из имеющихся у него?

Вход. В одной строке заданы три неотрицательных числа: a, b, c. Все числа не превышают 10^6.

Выход. Количество способов, которыми можно выбрать 2 фрукта разного вида.

Пример входа	Пример выхода
3 4 2	26

Выбрать два разных фрукта можно одним из следующих способов:

- банан и яблоко $a * b$ способами;
- банан и грушу $a * c$ способами;
- яблоко и грушу $b * c$ способами;

Таким образом сделать выбор двух разных фруктов можно $a * b + a * c + b * c$ способами. Учитывая что $a, b, c \le 10^6$, то ответ будет не более $3 * 10^{12}$. Следует воспользоваться типом `long long`.

Реализация алгоритма

Читаем входные данные. Вычисляем и выводим ответ.

```
scanf("%lld %lld %lld",&a,&b,&c);
res = a * b + a * c + b * c;
printf("%lld\n",res);
```

1548. Диагональ

Количество диагоналей у выпуклого n - угольника не менее N. Чему равно минимально возможное значение n?

Вход. Каждая входная стока содержит положительное целое число N ($N \le 10^{15}$) – количество проведенных диагоналей. Значение N = 0 является концом входных данных и не обрабатывается.

Выход. Для каждого теста вывести его номер и минимально возможное число n сторон многоугольника.

Пример входа	**Пример выхода**
10 100 1000 0	Case 1: 7 Case 2: 16 Case 3: 47

Каждая точка многоугольника соединена отрезками с n – 1 остальными точками. Эти отрезки образуют 2 стороны и n – 3 диагонали. Поскольку точек в многоугольнике n, а из каждой точки исходит n – 3 диагонали, то количество диагоналей выпуклого n – угольника равно $n * (n - 3) / 2$ (каждая диагональ подсчитывается дважды).

Если $n * (n - 3) / 2 = N$, то значение n находим из квадратного уравнения

$$n^2 - 3 * n - 2 * N = 0$$

Положительный корень уравнения равен

$$n = \frac{3 + \sqrt{9 + 8N}}{2}$$

Остается округлить сверху вычисленное значение. Поскольку $N \le 10^{15}$, то вычисления следует проводить, используя тип данных ***long long***.

Пример

Рассмотрим второй тест. Для N = 100 получим

$$n = \left\lceil \frac{3 + \sqrt{9 + 8 \cdot 100}}{2} \right\rceil = 16$$

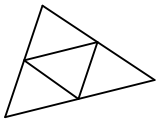
Реализация алгоритма

Читаем входные данные пока N > 0, вычисляем по формуле результат и выводим его с номером теста.

```
int cs = 1;
while(scanf("%lld",&n), n > 0)
{
  res = int(ceil((3 + sqrtl(9.0 + 8*n)) / 2));
  printf("Case %d: %d\n",cs,res);
  cs++;
}
```

2400. Треугольники

Миша любил рисовать треугольники, но делал он это необычным способом. Сначала рисовал произвольный треугольник, потом каждую сторону делил на *n* равных частей и проводил через точки раздела прямые, параллельные сторонам треугольника. В результате получается несколько равных между собой треугольников. Помогите Мише найти наибольшее количество одинаковых треугольников в его финальном рисунке.

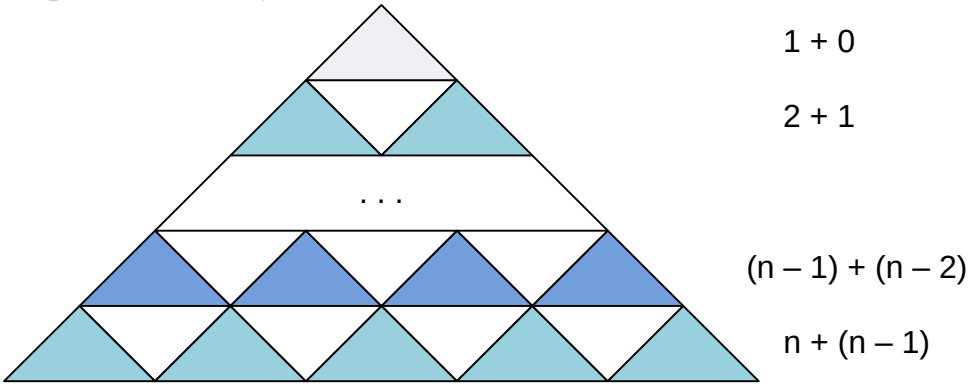


Вход. Целое число *n* ($0 < n < 2 * 10^9$).

Выход. Вывести наибольшее количество равных между собой треугольников.

Пример входа	**Пример выхода**
2	4

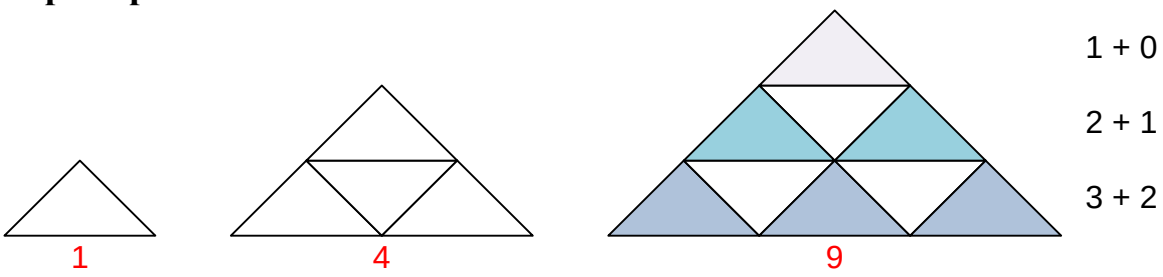
Рассмотрим треугольник, каждая сторона которого поделена на *n* равных частей. Наибольшее количество будет достигаться среди самых маленьких треугольников, равных между собой. Подсчитаем их число.



Разобьем треугольник на n горизонтальных полос, пронумеровав их сверху вниз. Последняя n-ая (нижняя) полоса содержит n треугольников вершиной вверх и $n - 1$ треугольников вершиной вниз. Предпоследняя $(n - 1)$ - ая полоса содержит $n - 1$ треугольников вершиной вверх и $n - 2$ треугольников вершиной вниз. Итого получим $n + (n - 1) + (n - 2) + \ldots + 2 + 1$ треугольников вершиной вверх и $(n - 1) + (n - 2) + \ldots + 2 + 1$ треугольников вершиной вниз. Общее количество одинаковых треугольников равно

$$2 * (n + (n - 1) + (n - 2) + \ldots + 2 + 1) - n = 2 * \frac{n+1}{2} n - n = n^2$$

Пример



При $n = 1, 2, 3$ количество равных треугольников составит 1, 4, 9 соответственно. При $n = 3$ ответ равен $(1 + 2 + 3) + (1 + 2) = 9$.

Реализация алгоритма

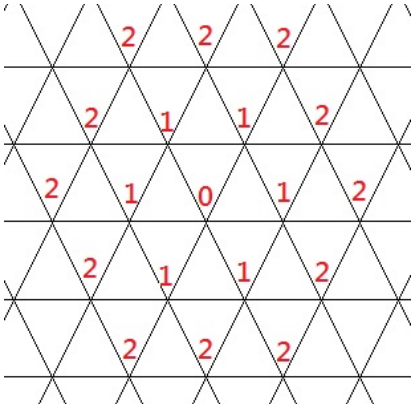
Читаем значение n, вычисляем и выводим ответ.

```
scanf("%lld",&n);
res = n * n;
printf("%lld\n",res);
```

4368. Треугольная паутина

Перед Вами бесконечная треугольная сетка. Она устроена таким образом, что если поджечь какую-нибудь вершину, то эта вершина загорается, в следующую секунду загораются все вершины, соседние непосредственно с данной, далее все вершины, соседние с уже горящими, и т.д. Считайте, что огонь никогда не тухнет.

Изначально подожжена одна вершина. Требуется найти количество горящих вершин через n секунд.



Вход. Одно число n ($0 \le n \le 10^9$).

Выход. Вывести количество горящих вершин через n секунд.

Пример входа 1	**Пример выхода 1**
1	7
Пример входа 2	**Пример выхода 2**
1500	6754501

Первый уровень (вершины, которые загораются через 1 секунду) содержит 6 вершин. Второй уровень содержит 12 вершин, третий 18 и так далее. Получается арифметическая прогрессия с разностью 6.

Чрез n секунд будет гореть $res = 1 + (6 + 12 + 18 + \ldots + (6 + 6 * (n - 1)))$ вершин. Используя формулу суммы арифметической прогрессии $\frac{2a_1 + d(n-1)}{2}n$, получим что

$$res = 1 + \frac{2*6 + 6(n-1)}{2}n = 1 + (6 + 3*(n - 1)) * n$$

Реализация алгоритма

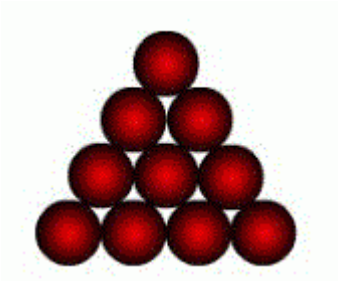
Читаем значение n. Вычисляем и выводим ответ.

```
scanf("%lld",&n);
res = 1 + (6 + 3 * (n - 1)) * n;
printf("%lld\n",res);
```

4538. Вася и шары

Недавно Вася узнал, что с шарами можно играть в очень занимательную игру. В этой игре требуется укладывать шары в виде различных геометрических фигур и тел. Пока Вася занимается укладкой шаров в виде равностороннего треугольника. Но вот незадача: иногда Васе не хватает наличных шаров, и он хочет знать, какова наибольшая сторона такого треугольника, для которого хватит Васиных шаров? Помогите Васе, напишите для него программу, которая будет вычислять n – длину стороны равностороннего треугольника для заданного количества шаров k.

Ниже приведён пример укладки шаров в виде равностороннего треугольника:

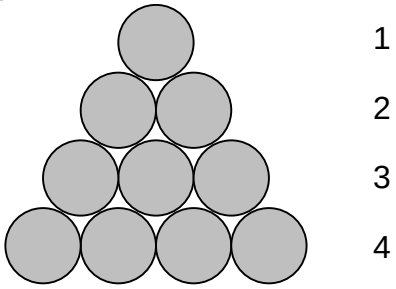


Вход. Натуральное число k ($0 \le k \le 2 * 10^8$) – имеющееся количество шаров.

Выход. Вывести число n – ответ задачи.

Пример входа	**Пример выхода**
5	2

Если n – длина стороны треугольника, то для полной его укладки требуется $1 + 2 + \ldots + n = n * (n + 1) / 2$ шаров.



В наличии имеется k шаров. Решим уравнение $n * (n + 1) / 2 = k$ и округлим неотрицательный корень вниз до ближайшего целого.

Решим квадратное уравнение: $n^2 + n - 2k = 0$, $d = 1 + 8k$, $n = \left(-1+\sqrt{1+8k}\right)/2$.

Ответом будет значение $\left\lfloor\left(-1+\sqrt{1+8k}\right)/2\right\rfloor$.

Реализация алгоритма

Читаем значение k. Вычисляем и выводим ответ.

```
scanf("%d",&k);
n = int((-1 + sqrt(1.0 + 8*k)) / 2.0);
printf("%d\n",n);
```

5716. Карточные домики

Славик мечтает стать артистом оригинального жанра, поэтому постоянно занимается манипуляциями с разнообразными предметами. В данный момент он всё своё свободное время занят построением карточных домиков. Ему уже даже удалось построить в домашних условиях 4-х этажный домик, на что у него ушло 26 карт. А на уроке технологии ему удалось тайком от учителя продемонстрировать своё умение одноклассникам и построить домик в 3 этажа,

на что было потрачено всего 15 карт. “Художества” будущего артиста запечатлены на фотографиях из его мобильного телефона ниже.

В данный момент Славика интересует вопрос математического содержания: А сколько нужно иметь карт, чтобы построить домик в n этажей? Он просит Вас помочь ему, так как с математикой у Славика пока проблемы...

Вход. Целое неотрицательное число n – количество этажей в домике, который планируется построить. Известно, что это число не превышает 10^8.

Выход. Вывести количество карт, необходимое для постройки домика в n этажей.

Пример входа	**Пример выхода**
3	15

Пусть f(h) равно количеству карт, из которых можно построить карточный домик высоты h. Например, f(1) = 2, f(2) = 7, f(3) = 15.

Для построения домика высоты h у основания следует поставить h домиков высоты 1 ($2h$ карт) и h – 1 перекрытий. На основание ставится домик высоты h – 1, на что потребуется f(h – 1) карт. Отсюда f(h) = $2h$ + h – 1 + f(h – 1) или

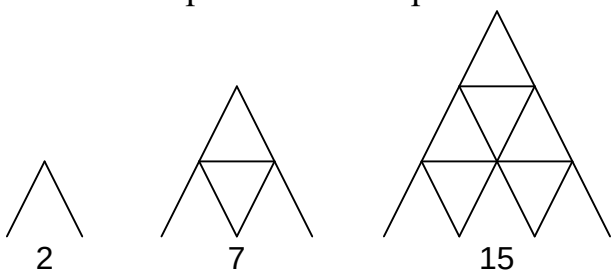
$$f(h) = 3h - 1 + f(h-1)$$

Выведем явную формулу.

$$f(h) = 3h - 1 + 3(h-1) - 1 + 3(h-2) - 1 + \ldots + 3 * 1 - 1 =$$
$$= 3\,(h + (h-1) + (h-2) + \ldots + 1) - h =$$
$$= 3*\frac{h+1}{2}h - h = \frac{3h^2+3h}{2} - \frac{2h}{2} = \frac{3h^2+h}{2}$$

Пример

Для домика высоты 1 достаточно 2 карты. Для домика высоты 2 необходимо 7 карт. Домик высоты 3 можно построить из 15 карт.



Реализация алгоритма

Читаем высоту домика n. Вычисляем и выводим количество карт в нем.

```
scanf("%lld", &n);
res = n * (3 * n + 1) / 2;
printf("%lld\n", res);
```

7817. Хорошее число

"Хорошим" будем считать число, которое состоит только из нечетных цифр. Например число 157953 хорошее, а число 2452117 нет. Необходимо выяснить, сколько существует n - значных хороших чисел.

Вход. Одно целое неотрицательное число n ($1 \le n \le 20$).

Выход. Вывести количество хороших чисел.

Пример входа	**Пример выхода**
4	625

Всего имеется пять нечетных цифр: 1, 3, 5, 7 и 9. Каждая из этих цифр может находиться в любой позиции n - значного числа. То есть поскольку в каждой из n позиций может находиться одна из 5 цифр, то количество хороших чисел равно 5^n.

Реализация алгоритма
Читаем значение n.

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

В переменной *res* вычисляем значение 5^n.

```
res = 1;
for(i = 0; i < n; i++)
 res *= 5;
```

Выводим ответ.

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

8595. Собаки и обезьяны

У Барыша есть n собак и m обезьян. Он хочет выстроить их в одну линию. Но он не хочет, чтобы в каком-либо месте стояло подряд две собаки или две обезьяны, потому что в таком случае они начинают драться. Сколько существует различных вариантов построения, таких чтобы ни обезьяны, ни собаки не дрались. Ответ выведите по модулю $10^9 + 7$. Имейте в виду, что собаки и обезьяны между собой различаются.

Вход. Два числа n и m ($1 \le n, m \le 10^5$).

Выход. Выведите количество различных вариантов построения обезьян и собак по модулю $10^9 + 7$.

Пример входа 1
```
2 2
```

Пример выхода 1
```
8
```

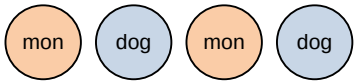
Пример входа 2
```
3 2
```

Пример выхода 2
```
12
```

Пример входа 3
```
1 8
```

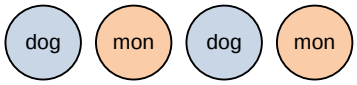
Пример выхода 3
```
0
```

Если $m > n + 1$ или $m < n - 1$, то требуемой расстановки не существует, ответ 0. Если $m = n$, то собак и обезьян следует расположить чередуя друг с другом. Например, на четных местах можно поставить собак ($n!$ вариантов, так как все собаки различные), на нечетных – обезьян ($m!$ вариантов, обезьяны различные).



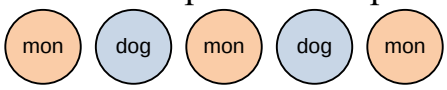
2! * 2! = 2 * 2 = 4

Аналогично можно на нечетных местах поставить собак, а на четных обезьян. Итого при $m = n$ имеем $2 * n! * m!$ вариантов.



2! * 2! = 2 * 2 = 4

Если собак на 1 больше обезьян (или соответственно обезьян на 1 больше чем собак), то количество вариантов их расположения равно $n! * m!$



3! * 2! = 6 * 2 = 12

Реализация алгоритма

Вычисления следует проводить по модулю MOD = $10^9 + 7$.

```
#define MOD 1000000007
```

Функция ***fact*** вычисляет факториал $n!$

```
long long fact(int n)
{
  long long res = 1;
  for (int i = 2; i <= n; i++)
    res = (res * i) % MOD;
  return res;
}
```

Читаем входные данные.

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

Если собак на 2 больше чем обезьян или обезьян на 2 больше чем собак, то такая расстановка невозможна.

```
if (m > n + 1 || m < n - 1)
  res = 0;
```

Если собак и обезьян одинаковое количество.

```
else if (m == n)
  res = (fact(n) * fact(n) * 2) % MOD;
else
```

Если собак на 1 больше чем обезьян или обезьян на 1 больше чем собак.

```
  res = (fact(n) * fact(m)) % MOD;
```

Выводим ответ.

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

9592. Конкатенация двух палиндромов

Найдите количество способов, которыми можно построить строку длины n используя k латинских букв (размер алфавита равен k) в виде конкатенации двух непустых палиндромов.

Вход. Два натуральных числа n и k ($1 \le n \le 10^5$, $1 \le k \le 26$).

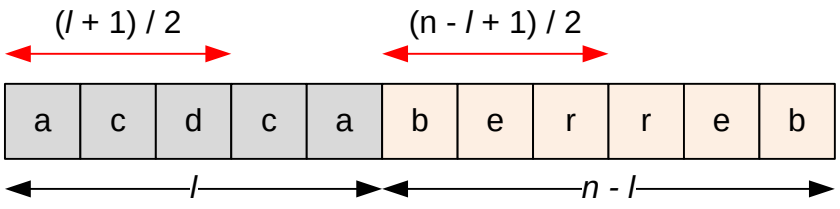
Выход. Выведите количество способов построить заданную строку. Ответ вывести по модулю $10^9 + 7$.

Пример входа 1	**Пример выхода 1**
4 1	3

Пример входа 2	**Пример выхода 2**
3 2	8

Представим строку длины n в виде конкатенации двух непустых палиндромов длин l и $n - l$. В палиндроме длины l первые $(l + 1) / 2$ букв можно выбрать произвольным образом (любую из k имеющихся), а остальные буквы следует выбрать так чтобы получить палиндром. Это можно совершить $k^{(l+1)/2}$ способами.

Аналогично в палиндроме длины $n - l$ первые $(n - l + 1) / 2$ букв можно выбрать произвольным образом, а из остальных следует образовать палиндром. Что можно совершить $k^{(n-l+1)/2}$ способами.



Построить конкатенацию двух палиндромов с длинами l и $n - l$ можно

$$k^{(l+1)/2} * k^{(n-l+1)/2}$$

способами. Поскольку ни один из палиндромов не должен быть пустым, то $1 \le l \le n - 1$. Общее количество возможных палиндромов равно

$$\sum_{l=1}^{n-1} k^{\frac{l+1}{2}} \cdot k^{\frac{n-l+1}{2}}$$

Все операции следует проводить по модулю $10^9 + 7$.

Реализация алгоритма

Вычисление ведется по модулю $10^9 + 7$. Определим модуль.

```
#define MOD 1000000007
```

Функция ***powmod*** вычисляет значение $x^n \bmod m$.

```
long long powmod(long long x, long long n, long long m)
{
  if (n == 0) return 1;
  if (n % 2 == 0) return powmod((x * x) % m, n / 2, m);
  return (x * powmod(x, n - 1, m)) % m;
}
```

Основная часть программы. Читаем входные данные.

```
scanf("%d %d", &n, &k);
res = 0;
for (l = 1; l < n; l++)
  res = (res + powmod(k, (l + 1) / 2, MOD) *
               powmod(k, (n - l + 1) / 2, MOD)) % MOD;
```

Выводим ответ.

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

9594. ABA

Задана строка символов. Сколько подпоследовательностей “ABA” встречается в ней? Буквы “ABA” не обязательно должны идти непосредственно друг за другом. Однако их порядок должен быть соблюден.

Вход. Одна строка, состоящая только из заглавных латинских символов. Длина строки не больше 10^6.

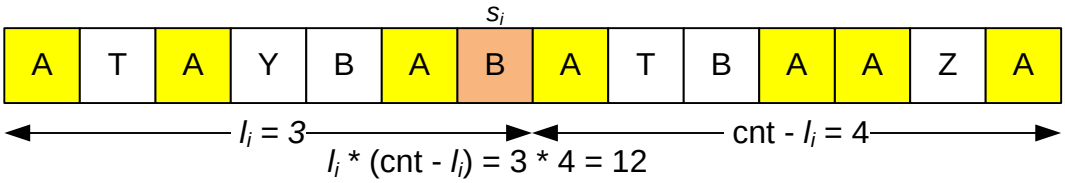
Выход. Выведите количество подпоследовательностей “ABA” в строке.

Пример входа 1	**Пример выхода 1**
YARBRBAQ	2

Пример входа 2
ABAYBATATBBAZA

Пример выхода 2
29

Пусть $s_0s_1\ldots s_{n-1}$ – входная строка. Пусть l[i] содержит количество букв A в позициях от нулевой до i-ой включительно. Пусть cnt равно количеству букв A в слове. Тогда справа от позиции i находится cnt – l[i] букв A.

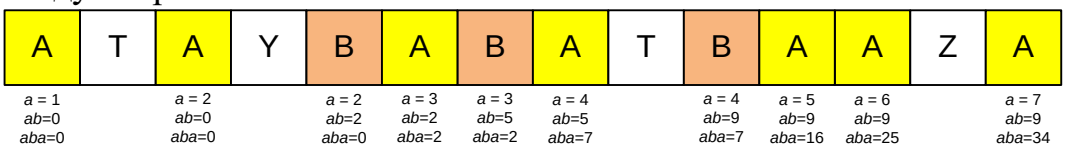


Если в i-ой позиции находится буква B, то слева от нее находится l[i] букв A, а справа (cnt – l[i]) букв A. Количество подпоследовательностей “ABA” у которых буква B находится в i-ой позиции, равно l[i] * (cnt – l[i]).

Второе решение. Пройдем по строке слева направо и в переменной a будем подсчитывать количество букв A.

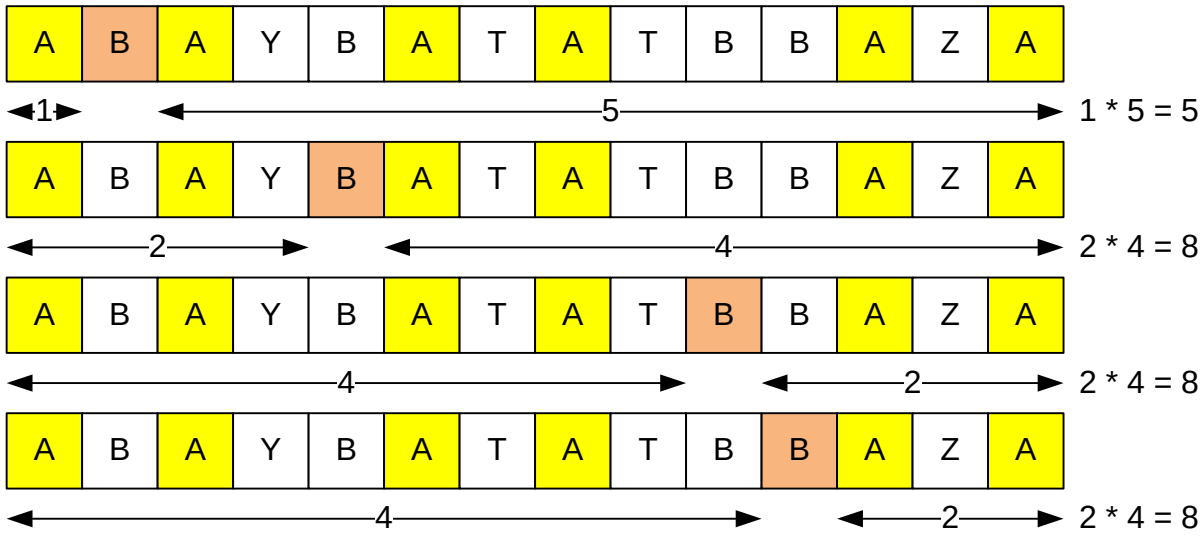
Для каждой буквы B в строке у нас образуется a слов AB (перед текущей буквой B имеется в точности a букв A). В переменной ab будем подсчитывать количество слов AB, которые встретились. Для каждой буквы B к ab следует прибавить a.

Для каждой текущей буквы A перед ней имеется ab слов AB. В переменной aba подсчитываем количество слов ABA, которые встретились. Для каждой буквы A к aba следует прибавить ab.



Пример

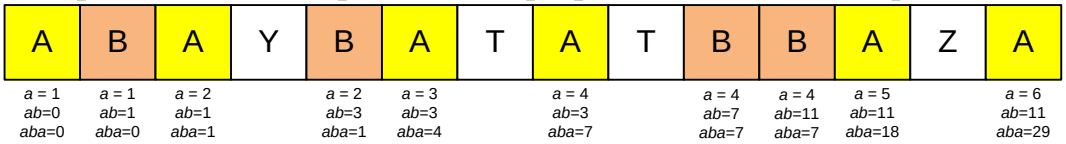
Рассмотрим второй тест.



Количество подпоследовательностей “ABA” равно

$$1 * 5 + 2 * 4 + 2 * 4 + 2 * 4 = 5 + 8 + 8 + 8 = 29$$

Рассмотрим значения переменных при решении задачи вторым способом.



Реализация алгоритма

Читаем входную строку *s*.

```
cin >> s;
cnt = 0; len = s.size();
l.resize(len);
```

Для каждой позиции *i* занесем в l[*i*] количество букв A в позициях от нулевой до *i*-ой включительно. По окончанию цикла *cnt* содержит количество букв A.

```
for (i = 0; i < len; i++)
{
  if (s[i] == 'A') cnt++;
  l[i] = cnt;
}
```

В переменной *res* подсчитываем ответ.

```
res = 0;
```

Для каждой позиции *i*, в которой находится буква B, подсчитываем количество подпоследовательностей “ABA”.

```
for (i = 0; i < len; i++)
  if (s[i] == 'B') res += 1LL * l[i] * (cnt - l[i]);
```

Выводим ответ.

```
cout << res;
```

Реализация алгоритма – комбинаторный подсчет

Читаем входную строку.

```
cin >> s;
a = ab = aba = 0;
```

Перебираем символы строки. Пересчитываем переменные *a*, *ab* и *aba*.

```
for (i = 0; i < s.size(); i++)
  if (s[i] == 'A')
  {
    a++;
    aba += ab;
```

```
    }
    else
    if (s[i] == 'B') ab += a;
```

Выводим ответ.

```
cout << aba << endl;
```

11177. Корова

Беси стоит перед огромным камнем в середине своего любимого поля. На камне – шифровка на древнем языке, алфавит которого состоит только из трёх букв C, O, W. Беси интересно, сколько раз встретилось слово COW в тексте.

Бесси не возражает если другие букв встречаются между C O W. Также Беси считает разными слова, в которых отличается хоть одна буква. Например COW встречается только один раз в слове CWOW, два раза в слове CCOW и 8 раз в слове CCOOWW.

По заданному тексту шифровки помогите Беси посчитать сколько раз появится слово COW.

Вход. Первая строка содержит одно целое число n ($n \le 10^5$). Вторая строка содержит строку из n символов, каждый их которых либо C, либо O, либо W.

Выход. Выведите количество раз, которое COW появляется как подпоследовательность, необязательно непрерывная, во входной строке.

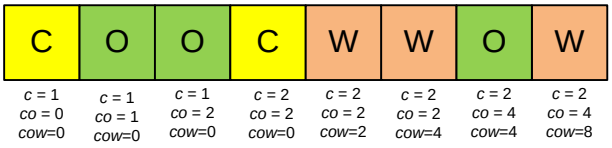
Заметим, что ответ может быть очень большим, поэтому используйте 64-битный целочисленный тип.

Пример входа	**Пример выхода**
12 COWCOWCOWCOW	20

Пройдем по строке слева направо и в переменной *c* будем подсчитывать количество букв C.

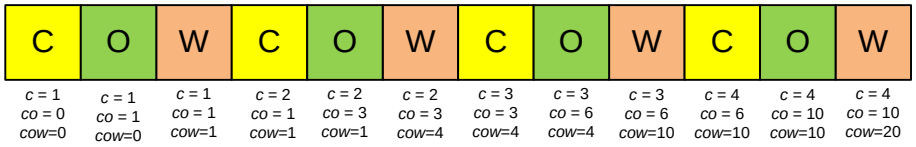
Для каждой буквы O в строке у нас образуется *c* слов CO (перед текущей буквой O имеется в точности *c* букв C). В переменной *co* будем подсчитывать количество слов CO, которые встретились. Для каждой буквы O к *co* следует прибавить *c*.

Для каждой текущей буквы W перед ней имеется *co* слов CO. В переменной *cow* подсчитываем количество слов COW, которые встретились. Для каждой буквы W к *cow* следует прибавить *co*.



Пример

Рассмотрим значения переменных *c*, *co* и *cow* при решении задачи.



Реализация алгоритма

Читаем входные данные. Инициализируем переменные.

```
cin >> n >> s;
c = co = cow = 0;
```

Последовательно обрабатываем буквы строки. В зависимости от текущей буквы s[*i*] увеличиваем переменную *c*, *co* или *cow*.

```
for (i = 0; i < n; i++)
  if (s[i] == 'C') c++; else
  if (s[i] == 'O') co += c; else
  if (s[i] == 'W') cow += co;
```

Выводим ответ.

```
cout << cow << endl;
```

11258. Количество прямоугольников

Сетка состоит из *n* * *m* единичных квадратов. Сколько различных прямоугольников можно изобразить на ней?

Стороны прямоугольников должны быть параллельны линиям сетки. Прямоугольник должен покрывать целое число полных квадратов. Два прямоугольника одинакового размера считаются разными, если они расположены в разных местах сетки.

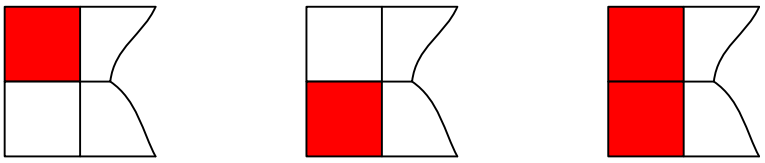
Вход. Два целых числа *n* и *m* ($n, m \le 10^5$).

Выход. Выведите количество различных прямоугольников, которые можно изобразить на сетке.

Пример входа	**Пример выхода**
2 3	18

Расмотрим размер прямоугольника *a* * *b*, изображенного на сетке. Рассмотрим, сколькими способами можно выбрать значение *a*. Сетка содержит *n* строк из квадратов, а следовательно в сетке имеется *n* + 1 горизонтальная линия. Пронумеруем их числами от 1 до *n* + 1. Выберем две линии *i* и *j* (*i* < *j*). Проведем

между ними отрезок – это и будет вертикальная сторона прямоугольника *a*. Количество способов выбрать два числа *i* и *j* (*i* < *j*) из множества {1, 2, …, *n* + 1} равно C_{n+1}^2. Например, для *n* = 2 имеется 3 варианта выбрать вертикальную сторону прямоугольника:



Аналогично в сетке имеется *m* + 1 вертикальная линия. Количество способов выбрать горизонтальную сторону прямоугольника равно C_{m+1}^2.

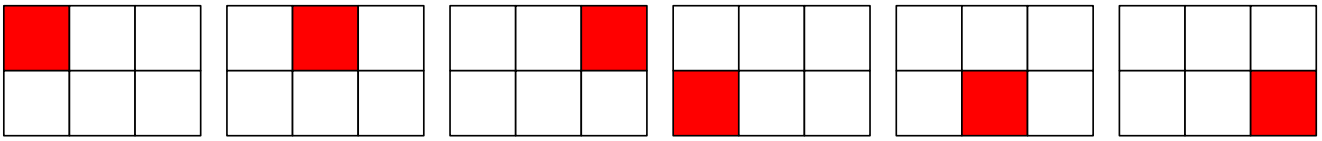
Следовательно количество различных прямоугольников на сетке равно

$$C_{n+1}^2 * C_{m+1}^2 = \frac{(n+1)n}{2} \cdot \frac{(m+1)m}{2}$$

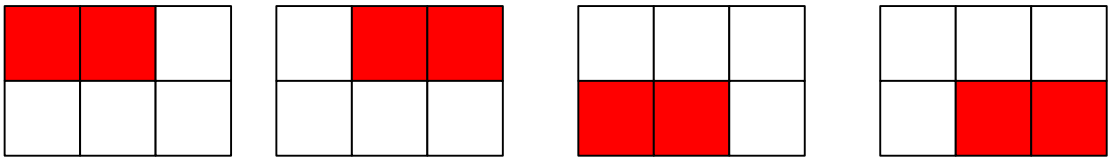
Пример

Для прямоугольника 2 * 3 имеются:

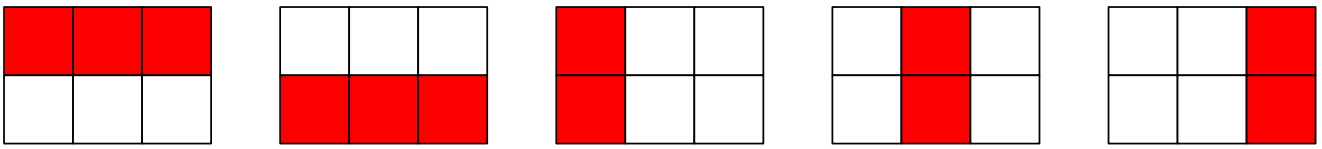
- 6 единичных прямоугольников;



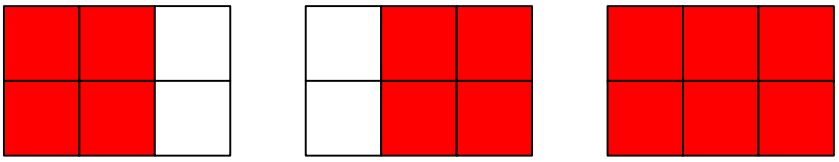
- 4 прямоугольника размером 1 * 2;



- 2 горизонтальных прямоугольника 1 * 3;
- 3 вертикальных прямоугольника 2 * 1;



- 2 прямоугольника 2 * 2;
- 1 прямоугольник 2 * 3;



Итого получилось $C_3^2 * C_4^2 = 3 * 6 = 18$ прямоугольников.

Реализация алгоритма
Читаем входные данные.

```
cin >> n >> m;
```

Вычисляем ответ $res = C_{n+1}^2 * C_{m+1}^2$.

```
a = (n + 1) * n / 2;
b = (m + 1) * m / 2;
res = a * b;
```

Выводим ответ.

```
cout << res << endl;
```

11259. Минимальная триангуляция

Вам задан правильный многоугольник из n вершин, пронумерованных от 1 до n против часовой стрелки. Триангуляция данного многоугольника – это набор треугольников такой, что каждая вершина любого из треугольников является вершиной первоначального многоугольника, не существует пары треугольников имеющих положительную площадь пересечения, и площадь объединения треугольников равна площади многоугольника. Вес триангуляции – это сумма весов треугольников из которых она состоит, где весом треугольника является произведение меток его вершин.

Найдите минимальный вес среди всех триангуляций заданного многоугольника.

Вход. Одно целое число n ($3 \le n \le 500$) – количество вершин в правильном многоугольнике.

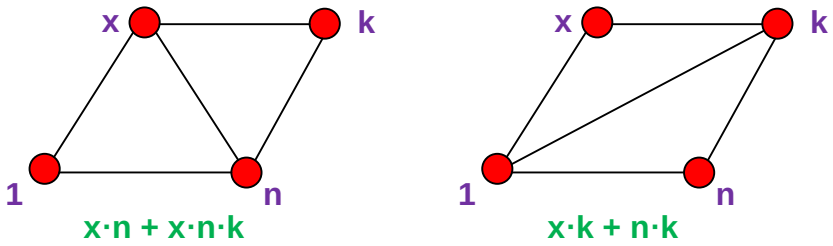
Выход. Выведите минимальный вес среди всех триангуляций заданного многоугольника.

Пример входа 1	**Пример выхода 1**
3	6
Пример входа 2	**Пример выхода 2**
4	18

Рассмотрим треугольник, содержащий сторону (1, n). Пусть его третьей вершиной будет x, то есть рассматриваемый треугольник имеет вид (1, n, x).

Если $x = n - 1$, то отбросив треугольник $(1, n, n - 1)$, перейдем к $(n - 1)$ – угольнику.

Пусть $1 < x < n - 1$. Рассмотрим треугольник (n, x, k), где $x < k < n$.

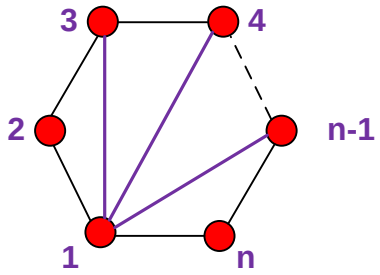


Заменим пару треугольников $(1, x, n)$ и (n, x, k) на пару $(1, x, k)$ и $(1, n, k)$. Суммарный вес треугольников уменьшится, так как $xn + xnk > xk + nk$. После перегруппировки получим

$$x(n - k) + nk(x - 1) > 0$$

Неравенство справедливо, так как $n - k > 0$ и $x - 1 > 0$.

Отсюда следует, что треугольник $(1, x, n)$ выгоднее заменить на $(1, k, n)$, где $k > x$. Отсюда следует, что в качестве x выгодно взять $x = n - 1$. Триангуляция с минимальным весом получится, если провести все диагонали из вершины 1.

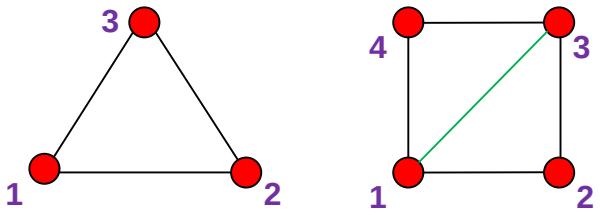


Минимальный вес равен

$$1 * 2 * 3 + 1 * 3 * 4 + \ldots + 1 * (n - 1) * n = \sum_{i=2}^{n-1} i(i+1)$$

Пример

В первом тесте задан треугольник с метками 1, 2, 3. Его вес равен 1 * 2 * 3 = 6.



Второй тест представляет собой квадрат с метками 1, 2, 3, 4. Минимальный вес получится для триангуляции, в которой проведена диагональ (1, 3). Он равен

$$1 * 2 * 3 + 1 * 3 * 4 = 6 + 12 = 18$$

Реализация алгоритма
Читаем входные данные.

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

Вычисляем ответ.

```
res = 0;
for (i = 2; i < n; i++)
  res += 1ll * i * (i + 1);
```

Выводим ответ.

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