## Авторы:

**Медведев Михаил Геннадиевич** – кандидат физико-математических наук, доцент кафедры математической информатики факультета кибернетики Киевского национального университета имени Тараса Шевченко.

**Присяжнюк Анатолий Васильевич** – учитель-методист высшей категории специализированной школы с углубленным изучением информатики № 17 г. Бердичев Житомирской области.

**Жуковский Сергей Станиславович** – старший преподаватель кафедры прикладной математики и информатики Житомирского государственного университета имени Ивана Франко, учитель информатики городского лицея №25 имени Н. А. Щорса.

## Задачи:

**318, 390, 1184, 1531 – 1539, 1987, 2388, 2390.**

# ОГЛАВЛЕНИЕ

# ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ

Большое количество задач дискретной математики, к которым относятся олимпиадные задачи по информатике, часто требуют совершать перебор различных комбинаторных конфигураций объектов и выбирать среди них наилучшие. Часто знание количества разных вариантов таких конфигураций позволяет реально оценить вычислительную сложность алгоритма.

**Правило сложения.** Если элемент $a \in$ A можно выбрать $m$ способами, а элемент $b \in$ B $n$ способами, то выбор элемента $x \in$ A $\cup$ B можно осуществить $m + n$ способами. При этом множества A и B не должны иметь общих элементов (A $\cap$ B $= \varnothing$).

**Правило умножения.** Если элемент $a \in$ A можно выбрать $m$ способами, а элемент $b \in$ B $n$ способами, то выбор пары $(a, b) \in$ A x B в указанном порядке можно осуществить $m * n$ способами.

***Последовательностью*** называется функция, определенная на множестве натуральных чисел.

***Перестановкой*** конечного множества A называется упорядоченная последовательность всех его элементов, в которой каждый элемент встречается ровно один раз.

Перестановку чисел 1, 2, .., $n$ можно трактовать как биекцию на множестве {1, 2, .., $n$}, которая числу $i$ ставит в соответствие $i$-ый элемент из набора.

Если множество содержит $n$ элементов, то всего существует $n!$ их перестановок.

Перестановка $\pi$ множества A может быть записана в виде подстановки, например:

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \\ y_1 & y_2 & y_3 & y_4 & y_5 \end{pmatrix},$$

где $\{x_1, x_2, x_3, \ldots, x_n\} = \{y_1, y_2, y_3, \ldots, y_n\} =$ A и $\pi(x_i) = y_i$.

Перестановку также можно записать в виде произведения непересекающихся циклов, причём единственным образом с точностью до порядка следования циклов в произведении. Например:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 1 & 6 & 4 & 2 & 3 \end{pmatrix} = (1, 5, 2)\,(3, 6)\,(4)$$

Перестановка $\pi$ называется ***инволюцией***, если $\pi \circ \pi = e$. Каждая инволюция является произведением непересекающихся транспозиций. Например:

$$(5, 7, 4, 3, 1, 8, 2, 6) = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 5 & 7 & 4 & 3 & 1 & 8 & 2 & 6 \end{pmatrix} = (1, 5)\,(2, 7)\,(3, 4)\,(6, 8)$$

***Беспорядком*** называется перестановка без неподвижных точек.

***Сочетанием*** из $n$ элементов по $k$ называется набор $k$ элементов, выбранных из данных $n$ элементов. Наборы, отличающиеся только порядком следования элементов (но не составом), считаются одинаковыми, этим сочетания отличаются от размещений.

Например, наборы {2, 1, 3} и {3, 2, 1} 5-элементного множества {1, 2, 3, 4, 5} являются одинаковыми (однако, как размещения были бы разными) и состоят из одних и тех же элементов {1, 2, 3}.

Количество сочетаний из $n$ по $k$ равно

$$C_n^k = \frac{n!}{k!(n-k)!}$$

Числа $C_n^k$ называются ***биномиальными коэффициентами***, которые появляются в биноме Ньютона

$$(x+y)^n = \sum_{k=0}^{n} C_n^k x^k y^{n-k}$$

**Пример.** Вычислить $\sum_{k=0}^{n} C_n^k$ для заданного натурального $n$.

Из Бинома Ньютона следует, что $\sum_{k=0}^{n} C_n^k = (1+1)^n = 2^n$.

**Пример.** Вычислить $C_n^0 - C_n^1 + C_n^2 - \ldots + (-1)^n C_n^n$ для заданного натурального $n$.

Из Бинома Ньютона следует, что $\sum_{k=0}^{n} (-1)^k C_n^k = (1-1)^n = 0$.

**Пример.** Доказать, что $\left(C_n^0\right)^2 + \left(C_n^1\right)^2 + \left(C_n^2\right)^2 + \ldots + \left(C_n^n\right)^2 = C_{2n}^n$.

Рассмотрим $2n$ объектов $a_i$ и разобьем их пополам: $\{a_1, a_2, \ldots, a_n, a_{n+1}, \ldots, a_{2n}\}$. Для того чтобы выбрать $n$ объектов из $2n$, выберем $k$ ($k \le n$) объектов из левой половины (то есть из множества $\{a_1, a_2, \ldots, a_n\}$) и $n-k$ объектов из правой (из множества $\{a_{n+1}, a_{n+2}, \ldots, a_{2n}\}$). Количество способов совершить такую выборку по правилу умножения равно $C_n^k \cdot C_n^{n-k} = \left(C_n^k\right)^2$. Поскольку $k$ может принимать значения от 0 до $n$, то $\sum_{k=0}^{n} \left(C_n^k\right)^2$ равно количеству способов выбрать $n$ объектов из $2n$, то есть $C_{2n}^n$.

Для реализации функции Cnk вычисления биномиального коэффициента воспользуемся соотношением:

$$C_n^k = \frac{n!}{k!(n-k)!} = \frac{n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-k+1)}{1 \cdot 2 \cdot 3 \cdot \ldots \cdot k}$$

Заведем переменную *res* типа long long, инициализируем ее 1. Умножим ее на $n$ и разделим на 1. Потом умножим на $n-1$ и разделим на 2. Процесс умножений и делений будем продолжать $k$ раз (числитель и знаменатель $C_n^k$ после сокращения содержит $k$ множителей).

Если значение $k$ велико, но $C_n^k$ помещается в int (например $C_{1000000001}^{1000000000}$), то согласно описанного алгоритма придется выполнять $k$ итераций, что приведет к Time Limit. Поэтому при $k > n-k$ следует воспользоваться соотношением $C_n^k = C_n^{n-k}$.

Программа вычисления биномиального коэффициента примет вид:

```
int Cnk(int k, int n)
{
  long long res = 1;
  if (k > n - k) k = n - k;
  for(int i = 1; i <= k; i++)
    res = res * (n - i + 1) / i;
  return (int)res;
}
```

Биномиальные коэффициенты обладают следующими ***свойствами***:

1. $C_n^0 = C_n^n = 1$, $C_n^1 = n$;
2. Симметрия: $C_n^k = C_n^{n-k}$;
3. Внесение-вынесение: $C_n^k = \frac{n}{k} C_{n-1}^{k-1}$;
4. Формула сложения: $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$.

Используя формулу сложения, можно динамически вычислять значения биномиальных коэффициентов.

```
long long cnk[10][10];

long long c(int n, int k)
{
  if (cnk[n][k] > 0) return cnk[n][k];
  if (n - k < k) return c(n,n-k);
  if (!k) return cnk[n][k] = 1;
  return cnk[n][k] = c(n-1,k) + c(n-1,k-1);
}

long long res = c(6,3); // вычисление C_6^3
```

Рассмотрим циклическую реализацию заполнения элементов массива cnk[$n$][$k$] = $C_n^k$. Она является эффективнее рекурсивной, например при обработке больших чисел. Используем равенство $C_n^0 = 1$ и формулу сложения $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$. Таким образом $n$-ая строка массива cnk будет пересчитываться через $(n-1)$ - ую.

| | 0 | 1 | | | | k |
|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | | | | | |
| | 1 | | | | $C_{n-1}^{k-1}$ | $C_{n-1}^{k}$ |
| n | 1 | | | | | $C_n^k$ |

```
void FillCnk(void)
{
  int n, k;
  memset(cnk,0,sizeof(cnk));
  for(n = 0; n < MAX; n++) cnk[n][0] = 1;
  for(n = 1; n < MAX; n++)
  for(k = 1; k <= MAX; k++)
    cnk[n][k] = cnk[n-1][k] + cnk[n-1][k-1];
}
```

После выполнения функции FillCnk массив cnk примет вид:

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 2 | 1 | 0 | 0 | 0 |
| 3 | 1 | 3 | 3 | 1 | 0 | 0 |
| 4 | 1 | 4 | 6 | 4 | 1 | 0 |
| 5 | 1 | 5 | 10 | 10 | 5 | 1 |

**Асимптотические оценки для Биномиального коэффициента.**

**Теорема 1.** Между биномиальными коэффициентами имеет место соотношение:

$$C_n^0 < C_n^1 < \ldots < C_n^{\lfloor n/2 \rfloor} \geq C_n^{\lfloor n/2 \rfloor + 1} > \ldots > C_n^n$$

**Теорема 2.** $C_{2n}^n < 4^n$. Следует из того, что $\sum_{k=0}^{2n} C_{2n}^k = (1+1)^{2n} = 2^{2n} = 4^n$.

**Теорема 3.** $C_{2n}^n > \frac{4^n}{2n+1}$. Следует из того, что $\sum_{k=0}^{2n} C_{2n}^k = 4^n$, а самих коэффициентов $2n+1$.

Поэтому больший из коэффициентов будет больше $\frac{4^n}{2n+1}$.

**Теорема 4. Формула Стирлинга.** $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$.

**Теорема 5.** $C_{2n}^n \approx \frac{\sqrt{4\pi n}\left(\frac{2n}{e}\right)^{2n}}{\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right)^2} = \frac{4^n}{\sqrt{\pi n}}$.

**Теорема 6.** $C_n^k = \frac{n!}{k!(n-k)!} = \frac{n\cdot(n-1)\cdot(n-2)\cdot\ldots\cdot(n-k+1)}{1\cdot 2\cdot 3\cdot\ldots\cdot k} = \frac{n^k}{k!}\left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\ldots\left(1-\frac{k-1}{n}\right)$

$= \frac{n^k}{k!} e^{\ln\left(1-\frac{1}{n}\right)+\ln\left(1-\frac{2}{n}\right)+\ldots+\ln\left(1-\frac{k-1}{n}\right)} \leq$ (воспользуемся неравенством $\ln(1-x) \leq -x$ )

$\frac{n^k}{k!} e^{-\frac{1}{n}-\frac{2}{n}-\ldots-\frac{k-1}{n}} = \frac{n^k}{k!} e^{-\frac{k(k-1)}{2n}}$.

**Теорема 7.** Учитывая, что $\ln(1-x) = -x + o(x^2)$, имеем: $C_n^k = \frac{n^k}{k!} e^{-\frac{1}{n}-\frac{2}{n}-\ldots-\frac{k-1}{n}+O\left(\frac{1}{n^2}+\frac{4}{n^2}+\ldots+\frac{(k-1)^2}{n^2}\right)}$

$= \frac{n^k}{k!} e^{-\frac{k(k-1)}{2n}+O\left(\frac{k^3}{n^2}\right)}$.

**Следствие.** Если $k = o\left(\sqrt{n}\right)$, то $C_n^k \approx \frac{n^k}{k!}$.

Пусть имеются предметы $n$ различных видов. Количество предметов каждого вида неограниченно. Рассмотрим количество расстановок этих предметов, которые имеют длину $k$. Такие расстановки называются ***сочетаниями с повторениями*** из $n$ элементов по $k$, их количество равно

$$\overline{C_n^k} = C_{n+k-1}^{n-1} = C_{n+k-1}^k$$

Пусть имеется $a_1$ предмет первого вида. Поскольку в сочетаниях порядок предметов в расстановке не важен, будем считать что все они находятся в начале. Закодируем эти предметы единицами. Чтобы предметы не перепутались, введем нули-разделители. То есть после $a_1$ единиц поставим 0, после чего запишем $a_2$ единицы, соответствующие $a_2$ предметам второго типа и так далее. Очевидно, что $a_1 + a_2 + \ldots + a_n = k$, каждое $a_i$ может принимать значение от 0 до $k$. Итого имеется всего $k$ предметов и $n-1$ разделитель. Задача свелась к выбору мест для $n-1$ разделителя среди $n+k-1$ позиций.

**Пример.** В магазине имеются 2 типа пирожных. Петя хочет купить три пирожных. Сколькими способами он может это сделать?

Ответ равен $\overline{C_2^3} = C_4^3 = 4$. Действительно, если типы пирожных обозначить через $a$ и $b$, то среди возможных покупок могут быть $aaa$, $aab$, $abb$ и $bbb$.

**Пример.** Сколько существует упорядоченных последовательностей $(a_1, a_2, a_3)$, для которых $a_1 + a_2 + a_3 = 5$, где все $a_i$ целые и неотрицательные?

Ответом будет количество сочетаний из 3 по 5, то есть $\overline{C_3^5} = C_7^5 = 7 * 6 / 2 = 21$.

Искомыми упорядоченными последовательностями будут: (5, 0, 0) и все их перестановки (3 штуки), (4, 1, 0) – 6 штук, (3, 2, 0) – 6 штук, (3, 1, 1) – 3 штуки, (2, 2, 1) – 3 штуки. Итого всего $2 * 6 + 3 * 3 = 12 + 9 = 21$.

***Размещением*** называется расположение «предметов» на некоторых «местах» при условии, что каждое место занято в точности одним предметом и все предметы различны. Более формально, размещением (из $n$ по $k$) называется упорядоченный набор из $k$ различных элементов некоторого $n$-элементного множества.

Например, (2, 5, 4, 1) – это 4-элементное размещение 6-элементного множества {1, 2, 3, 4, 5, 6}. В отличие от сочетаний, размещения учитывают порядок следования предметов. Так, например, наборы (2, 1, 3) и (3, 2, 1) являются различными, хотя состоят из одних и тех же элементов {1, 2, 3} (то есть совпадают как сочетания).

Количество размещений равно убывающему факториалу

$$A_n^k = n\,(n-1)\,(n-2)\ldots(n-k+1) = \frac{n!}{(n-k)!} = C_n^k \cdot k!$$

При $k = n$ количество размещений равно количеству перестановок порядка $n$:

$$A_n^n = P_n = n!$$

***Размещение с повторениями*** или выборка с возвращением – это размещение «предметов» в предположении, что каждый «предмет» может участвовать в размещении несколько раз. По правилу умножения количество размещений с повторениями из $n$ по $k$ равно

$$\overline{A_n^k} = n^k$$

***Мультимножеством*** называется множество, которое может содержать одинаковые элементы. Пусть мультимножество M содержит $n_1$ раз элемент $a_1$, $n_2$ раза элемент $a_2$, … , $n_k$ раз элемент $a_k$. Тогда число его перестановок равно полиномиальному коэффициенту

$$P(n_1, n_2, \ldots, n_k) = \binom{n}{n_1, n_2, \ldots, n_k} = \frac{n!}{n_1!n_2!\ldots n_k!},$$

где $n = n_1 + n_2 + \ldots + n_k$ – общее число элементов в мультимножестве M.

**Связь перестановок с сочетаниями.** Вычислим количество перестановок мультимножества используя сочетания: $n_1$ элементов $a_1$ можно расположить на $n$ местах $C_n^{n_1}$ способами. На следующих $n - n_1$ оставшихся местах можно расположить $n_2$ элементов $a_2$ $C_{n-n_1}^{n_2}$ способами. Рассуждая подобным образом, получим что $n_k$ элементов $a_k$ можно расположить $C_{n-n_1-n_2-\ldots-n_{k-1}}^{n_k}$ способами. Согласно правилу произведения получим, что

$$P(n_1, n_2, \ldots, n_k) = C_n^{n_1} * C_{n-n_1}^{n_2} * \ldots * C_{n-n_1-n_2-\ldots-n_{k-1}}^{n_k} = \frac{n!}{n_1!n_2!\ldots n_k!}$$

**Пример.** Сколько перестановок имеет слово “мама”?

Слово имеет две буквы “м” и две буквы “а”. Поэтому оно имеет $P(2, 2) = \frac{4!}{2!\,2!} = 6$ перестановок.

**Полиномиальная формула.** Формула

$$\left(x_1 + x_2 + \ldots + x_k\right)^n = \sum_{n_1+n_2+\ldots+n_k=n} \frac{n!}{n_1!n_2!\ldots n_k!} x_1^{n_1} x_2^{n_2} \ldots x_k^{n_k}$$

называется ***полиномиальной***, где суммирование выполняется по всем решениям уравнения $n_1 + n_2 + \ldots + n_k = n$ в целых неотрицательных числах, $n_i \geq 0$, $i = 1, 2, \ldots, k$.

***Разбиением*** натурального числа $n$ называется его представление в виде суммы натуральных слагаемых: $n = x_1 + x_2 + \ldots + x_k$ для некоторого натурального числа $k$.

Пусть $a_1, a_2, \ldots, a_n$ – перестановка множества $\{1, 2, \ldots, n\}$. Если $i < j$ и $a_i > a_j$, то пара $(a_i, a_j)$ называется ***инверсией*** перестановки. Каждая инверсия – это пара элементов, которая “нарушает порядок”. Понятие инверсии ввел Г. Крамер в 1750 году, определив детерминант матрицы $n \times n$ следующим образом:

$$\det\begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1n} \\ \ldots & \ldots & & \ldots \\ x_{n1} & x_{n2} & \ldots & x_{nn} \end{pmatrix} = \sum (-1)^{\mathrm{inv}(a_1,a_2,\ldots,a_n)} x_{1a_1} x_{2a_2} \ldots x_{na_n},$$

где сумма берется по всем перестановкам $a_1, a_2, \ldots, a_n$ из $\{1, 2, \ldots, n\}$, а $\mathrm{inv}(a_1, a_2, \ldots, a_n)$ равно числу инверсий в перестановке. Например, перестановка (3, 1, 4, 2) имеет три инверсии: (3, 1), (3, 2), (4, 2).

***Таблицей инверсии*** перестановки $(a_1, a_2, \ldots, a_n)$ называется последовательность $(d_1, d_2, \ldots, d_n)$, где $d_i$ – число элементов, больших $i$ и расположенных левее $i$. Другими словами $d_i$ равно числу инверсий, у которых второй элемент равен $i$. Например, таблицей инверсий для перестановки (3, 1, 4, 2) будет (1, 2, 0, 0). Из определения таблицы инверсий следует, что

$$0 \leq d_1 \leq n-1,\ 0 \leq d_2 \leq n-2,\ \ldots,\ 0 \leq d_{n-1} \leq 1,\ d_n = 0$$

***Построение таблицы инверсий.*** Для получения таблицы инверсий из таблицы перестановки вводим таблицу равной по длине таблице перестановки (ее длина равна $n$) и на $n$-ное место записываем 0. Ищем число $i$ (от $n-1$ до 1) в таблице перестановки, и смотрим: сколько чисел больше $i$ находится слева от числа $i$, полученное число записываем в таблицу инверсий на $i$-ое место.

**Пример.** Таблицей инверсий перестановки (2, 4, 3, 1) будет (3, 0, 1, 0).

***Восстановление таблицы инверсий.*** Для восстановления таблицы перестановки из таблицы инверсий (ее длина равна $n$) создаем таблицу, которую будем расширять по мере добавления в неё чисел. Добавляем в эту таблицу число $i$ (где $i$ изменяется от $n$ до 1) на позицию $k+1$, где $k$ – число в таблице инверсий на $i$-ом месте.

**Пример.** Восстановим перестановку по инверсии (3, 0, 1, 0).
Добавим 4 на позицию 1 (в таблице на 4 месте стоит 0): (4).
Добавим 3 на позицию 2 (в таблице на 3 месте стоит 1): (4, 3).
Добавим 2 на позицию 1 (в таблице на 2 месте стоит 0): (2, 4, 3).
Добавим 1 на позицию 4 (в таблице на 1 месте стоит 3): (2, 4, 3, 1).

***Формула включений-исключений*** позволяет определить мощность объединения конечного числа конечных множеств, которые в общем случае могут пересекаться друг с другом. Например, для двух множеств формула имеет вид:

$$|A \cup B| = |A| + |B| - |A \cap B|$$

В терминах множеств формула включений-исключений имеет вид:

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_i A_i - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \ldots + (-1)^{n-1} |A_1 \cap A_2 \cap \ldots \cap A_n|$$

Сформулируем формулу включений-исключений в ***терминах свойств***. Пусть конечное множество $U$ состоит из $N$ элементов и пусть имеется набор свойств $a_1, a_2, \ldots, a_n$. Каждый элемент множества $U$ может обладать или не обладать любым из этих свойств. Обозначим через $N(a_{i1}, a_{i2}, \ldots, a_{is})$ количество элементов, обладающих свойствами $a_{i1}, a_{i2}, \ldots, a_{is}$ (и возможно некоторыми другими). Обозначим через $N(\overline{a_{i1}}, \overline{a_{i2}}, \ldots, \overline{a_{is}})$ количество элементов, не обладающих ни одним из свойств $a_{i1}, a_{i2}, \ldots, a_{is}$. Тогда имеет место формула:

$$N(\overline{a_{i1}}, \overline{a_{i2}}, \ldots, \overline{a_{is}}) = N - \sum_i N(a_i) + \sum_{i<j} N(a_i a_j) - \ldots + (-1)^n N(a_1 a_2 \ldots a_n)$$

Найдем ***явное выражение функции Эйлера*** $\varphi(n)$. Подсчитаем количество чисел, не больших $n$ и взаимно простых с $n$. Пусть $n = p_1^{a_1} p_2^{a_2} \ldots p_k^{a_k}$ – разложение на простые множители. Число $i$ является взаимно простым с $n$ тогда и только тогда, когда ни один из простых делителей $p_i$ не делит $n$. Пусть свойство $a_i$ означает, что $p_i$ делит $n$. Тогда количество чисел, взаимно простых с $n$, равно $N(\overline{a_1}, \overline{a_2}, \ldots, \overline{a_k})$. При этом количество $N(a_{i1}, a_{i2}, \ldots, a_{is})$ чисел, обладающих свойствами $a_{i1}, a_{i2}, \ldots, a_{is}$, равно $n / p_{i1} p_{i2} \ldots p_{is}$. По формуле включений-исключений находим:

$$\varphi(n) = n - \sum_i n/p_i + \sum_{i,j} n/p_i p_j - \ldots + (-1)^n n/p_1 p_2 \ldots p_k$$

Указанная формула приводится к виду: $\varphi(n) = n \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right)$.

**Пример.** Пусть $n = p^k$. Тогда $\varphi(n) = n - \frac{n}{p} = n\left(1 - \frac{1}{p}\right)$.

Если $n = p^k q^l$, то $\varphi(n) = n - \frac{n}{p} - \frac{n}{q} + \frac{n}{pq} = n\left(1 - \frac{1}{p}\right)\left(1 - \frac{1}{q}\right)$.

***Тождество максимумов и минимумов*** – это математическое соотношение между максимальным элементом конечного множества чисел и минимальными элементами всех его непустых подмножеств. Пусть $x_1, x_2, \ldots, x_n$ – произвольные действительные числа. Тогда имеет место соотношение:

$$\max(x_1, x_2, \ldots, x_n) = \sum_i x_i - \sum_{i<j} \min(x_i, x_j) + \sum_{i<j<k} \min(x_i, x_j, x_k) - \ldots + (-1)^{n-1} \min(x_1, x_2, \ldots, x_n)$$

# УСЛОВИЯ ЗАДАЧ

## 318. Биномиальные коэффициенты 1

Пусть $n$ – целое неотрицательное число. Обозначим:

$$n! = 1 * 2 * ... * n. \ (0! = 1),$$

$$C_n^k = \frac{n!}{k!(n-k)!}$$

По заданным $n$ и $k$ вычислить $C_n^k$.

**Вход.** Входной файл содержит $T \le 50$ тестовых случаев. В первой строке входного файла находится натуральное число T. Каждая из следующих T строк описывает один тестовый случай и содержит числа $n$ и $k$, разделенные пробелом. Эти числа удовлетворяют неравенствам $0 \le n < 2^{64}$ и $0 \le C_n^k < 2^{64}$.

**Выход.** Выходной файл должен состоять из T строк. Каждая строка должна содержать число $C_n^k$ для соответствующего тестового случая из входного файла.

**Пример входа**

```
6
0 0
1 0
1 1
2 0
2 1
2 2
```

**Пример выхода**

```
1
1
1
1
2
1
```

## 390. Анаграммы

Анаграммой слова называется любая перестановка всех букв слова. Например, из слова SOLO можно получить 12 анаграмм: SOLO, LOSO, OSLO, OLSO, OSOL, OLOS, SLOO, LSOO, OOLS, OOSL, LOOS, SOOL. Напишите программу, которая выводит количество различных анаграмм, которые могут получиться из этого слова.

**Вход.** В единственной строке задано слово, количество букв в котором не превышает 14.

**Выход.** Количество различных анаграмм.

**Пример входа**

```
SOLO
```

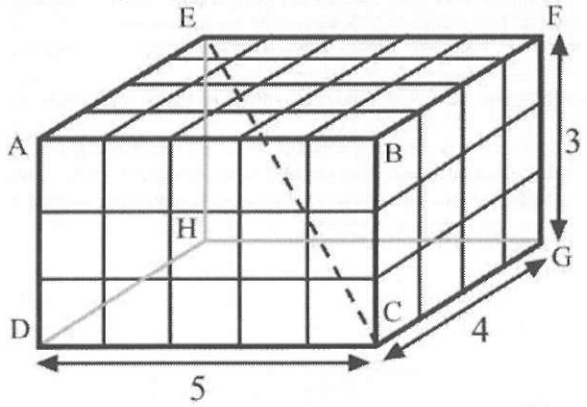
**Пример выхода**

```
12
```

## 1184. Гипер - устройство

Гипер-устройство - это часто используемый термин в научно-фантастических рассказах. Хотя многие считают, что гипер-устройство не возможно, тем не менее существует большое количество теорий, описывающих черные дыры и гипер-устройства. Говорят, что гипер-устройство позволяет путешествовать в более высоких измерениях. В этой задаче Вас следует вычислить стоимость путешествия по большим измерениям согласно теории старого сумасшедшего друга Арифа. Я уверен, что вы знаете кто такой Ариф. Если нет, то можете спросить у своих товарищей по команде.

Пусть $P$ и $Q$ – две точки в $n$ мерном пространстве, координаты которых соответственно равны $P(p_1, p_2, ..., p_n)$ и $Q(q_1, q_2, ..., q_n)$. Универсальное $n$-мерное пространство разбито на маленькие единичные $n$-мерные гиперкубы. Для наглядного примера представленное ниже (5 x 4 x 3) трехмерное пространство разбито на 60 трехмерных единичных кубов (1 x 1 x 1).




Давайте обойдемся без визуализации примеров в больших размерностях. Стоимость путешествия от одной $n$-мерной точки P до другой $n$-мерной точки Q равна "количеству разных $n$-мерных единичных гиперкубов, которое пересекает отрезок, соединяющий эти две точки". Вам следует вычислить эту стоимость. Например, на рисунке сверху стоимость путешествия от C до E равно 10 единицам, так как отрезок EC проходит через 10 разных единичных трехмерных гиперкубов.

**Вход.** Первая строка содержит количество тестов $n$ ($n \le 501$). Каждый тест начинается целым числом $D$ ($0 < D \le 10$) – размерностью, в которой будет измеряться стоимость. Каждая из следующих двух строк содержит $D$ целых чисел. $D$ чисел первой строки задают координаты $P$, а $D$ чисел второй строки задают координаты $Q$. Все числа положительны и являются 32-битовыми знаковыми целыми.

**Выход.** Для каждого теста вывести в одной строке стоимость путешествия из $P$ в $Q$, которая является целым числом.

**Пример входа**

```
2
2
10 10
10 13
1
10
20
```

**Пример выхода**

```
Case 1: 0
Case 2: 10
```

## 1531. Ключи в коробке

Имеется $n$ коробок, пронумерованных с 1 до $n$, а также n ключей, пронумерованных с 1 до $n$. $i$ - ый ключ может открыть только $i$ - ую коробку. Произвольным образом расположим каждый ключ в отдельную коробку, после чего закроем все коробки. Считаем, что любое расположения ключей можно получить с одинаковой вероятностью. Имеются m бомб, каждой из которых можно открыть одну коробку. Открыв при помощи бомбы коробку и достав из нее ключ, возможно, этим ключом можно открыть еще одну коробку (если этот ключ не от коробки, в которой он лежал). Таким образом продолжаем процесс подрыва коробок, доставания ключей и открывание коробок извлеченными ключами.

Найти вероятность того, что можно таким образом открыть все коробки.

**Вход.** Каждая строка содержит два целых числа $n$ ($1 \le n \le 20$) и $m$ ($1 \le m \le n$).

**Выход.** Для каждого теста в отдельной строке вывести вероятность того, что можно открыть все коробки. Выводить искомую вероятность следует в виде дроби "A/B". Значения A и B являются натуральными числами, не содержат ведущих нулей и являются взаимно простыми.

**Пример входа**

```
2 1
3 1
3 2
4 2
```

**Пример выхода**

```
1/2
1/3
5/6
17/24
```

## 1532. Рукопожатие

За круглым столом собрались $n$ бизнесменов. Перед совещанием они должны одновременно пожать друг другу руки. Руки никаких бизнесменов не должны пересекаться. Вычислить количество способов, которыми они смогут пожать друг другу руки.

**Вход.** Каждая строка содержит одно четное натуральное число $n$ ($2 \le n \le 50$).

**Выход.** Для каждого значения $n$ вывести в отдельной строке количество способов, которыми $n$ бизнесменов смогут пожать друг другу руки.

**Пример входа**

```
2
4
8
```

**Пример выхода**

```
1
2
14
```

## 1533. Генерация анаграмм

Вам следует написать программу, которая сгенерирует все возможные слова из заданного набора букв.

Например, из слова "abc" в результате перестановки букв можно получить слова "abc", "acb", "bac", "bca", "cab" и "cba".

Входные слова могут содержать несколько одинаковых букв. Ваша программа не должна выводить одно и то же слово более одного раза. Все слова следует выводить по возрастанию в алфавитном порядке.

**Вход.** Первая строка содержит количество тестов. Каждая следующая строка содержит слово из букв латинского алфавита (от A до Z). Буквы верхнего и нижнего регистра считать различными.

**Выход.** Для каждого теста вывести все возможные слова, которые можно получить из заданных букв, в возрастающем алфавитном порядке. Каждое слово выводить в отдельной строке. В алфавитном порядке буква верхнего регистра меньше соответствующей буквы нижнего регистра.

**Пример входа**

```
2
aAb
abc
```

**Пример выхода**

```
Aab
Aba
aAb
abA
bAa
baA
abc
acb
bac
bca
cab
cba
```

## 1534. Делимость

Имеется набор целых чисел $a_1, a_2, ..., a_n$. Найти количество чисел от $l$ до $r$ включительно, которые делятся хотя бы на одно число из этого набора.

**Вход.** Состоит из нескольких тестов. Первая строка каждого теста содержит два целых числа $l$ ($1 \le l \le 10^9$) и $r$ ($1 \le r \le 10^9$). Следующая строка содержит количество элементов $n$ ($1 \le n \le 18$) в наборе и сам набор этих чисел. Каждое число в наборе принимает значение от 1 до $10^9$.

**Выход.** Для каждого теста в отдельной строке вывести количество чисел от $l$ до $r$ включительно, которые делятся хотя бы на одно из чисел $a_1, a_2, ..., a_n$.

**Пример входа**

```
293 784
1 1
579000 987654
2 1 2
1 1000000000
2 2 3
```

**Пример выхода**

```
492
408655
666666667
```

## 1535. Небоскребы

Линия горизонта в городе содержит n зданий, каждое из которых имеет уникальную высоту от 1 до $n$. Дом виден слева (справа), если левее (правее) его нет дома с большей высотой. Например, если дома имеют порядок {1, 3, 5, 2, 4}, то слева видны три дома с номерами 1, 3, 5, а справа два, номера которых 4 и 5.

Вам известно, что домов всего $n$, *leftSide* домов видны слева, и *rightSide* домов видны справа. Найти количество перестановок домов, которые удовлетворяют этим ограничениям.

**Вход.** Каждая строка является отдельным тестом и содержит значения $n$ ($1 \le n \le 100$), *leftSide* и *rightSide* ($1 \le leftSide, rightSide \le n$).

**Выход.** Для каждого теста в отдельной строке вывести количество перестановок домов, которые удовлетворяют заданным условиям.

**Пример входа**

```
3 2 2
5 3 2
8 3 2
```

**Пример выхода**

```
2
18
4872
```

## 1536. Любимый ребенок мешает

Дети всегда любимые, но иногда они могут заставить Вас вести себя резко. В этой задаче вы увидите, как Тинтин, пятилетний мальчик, создает проблемы для своих родителей. Тинтин - веселый мальчик и всегда занят делами. Но не все что он делает, доставляет радость его родителям. Больше всего ему нравится играть с домашними вещами, как например часы отца или расческа матери. После игры он не возвращает их на место. Тинтин очень умный мальчик с достаточно четкой памятью. Он расстраивает своих родителей тем, что никогда не возвращает вещи, взятые для игры, на первоначальное место.

Подумать только! Как-то утром Тинтин удалось украсть три предмета домашнего обихода. Сколькими способами он может вернуть эти вещи так, чтобы ни один из предметов не попал на свое прежнее место? Тинтин не хочет расстраивать своих родителей и доставлять им неприятности. Поэтому он ничего не прячет в новых местах, а просто переставляет предметы.

**Вход.** Состоит из нескольких тестов. Каждый тест представлен натуральным числом, не большим 800 – количеством вещей, которое Тинтин взял для игры. Каждое число находится в отдельной строке. Последняя строка содержит –1 (минус один) и не обрабатывается.

**Выход.** Для каждого теста вывести количество способов, которыми Тинтин может вернуть взятые вещи.

**Пример входа**

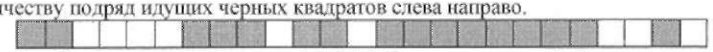
```
2
3
4
-1
```

**Пример выхода**

```
1
2
9
```

## 1537. Полоса

Имеется прямоугольник размера $1 \times n$, квадратики $1 \times 1$ которого закрашены белым или черным цветом. По прямоугольнику можно построить "код" - последовательность чисел, равных количеству подряд идущих черных квадратов слева направо.



Например, код этого прямоугольника 2 3 2 8 1. Однако количество белых квадратов нигде не учитывается (группы черных клеток должны разделяться как минимум одной белой клеткой). Поэтому одному и тому же коду может соответствовать несколько прямоугольников. Например, выше приведенному коду также соответствует прямоугольник

Вам необходимо подсчитать количество прямоугольников, удовлетворяющих заданному коду.

**Вход.** Первая строка содержит количество тестов $t$ ($1 < t < 20$). Каждая из следующих $t$ строк содержит данные для одного теста. Каждый тест начинается длиной прямоугольника $n$ ($1 \le n \le 200$). Затем идет $k$ ($0 \le k \le (n + 1) / 2$) – количество чисел в коде. Далее идут $k$ чисел, описывающих непосредственно код.

**Выход.** Для каждого теста вывести в отдельной строке одно число – количество прямоугольников, удовлетворяющих заданному коду. Ответ всегда помещается в 50 знаковое целое.

**Пример входа**

```
3
4 0
5 2 1 2
4 2 2 2
```

**Пример выхода**

```
1
3
0
```

## 1538. Мысли наоборот

Деление плоскости на части различными фигурами - известная задача в области компьютерных наук. Внизу на рисунке изображено несколько таких диаграмм. На рисунке 1 четыре окружности могут разделить плоскость максимум на 14 областей, четыре эллипса могут разделить плоскость максимум на 26 областей, а три треугольника - на 20 частей. Классическая задача состоит в том, чтобы найти максимальное количество областей, на которое могут разделить плоскость m фигур. Например, для окружностей известна формула $m^2 - m + 2$. В смешанном случае (когда пересекается несколько типов фигур) максимально возможное количество областей найти также не трудно.

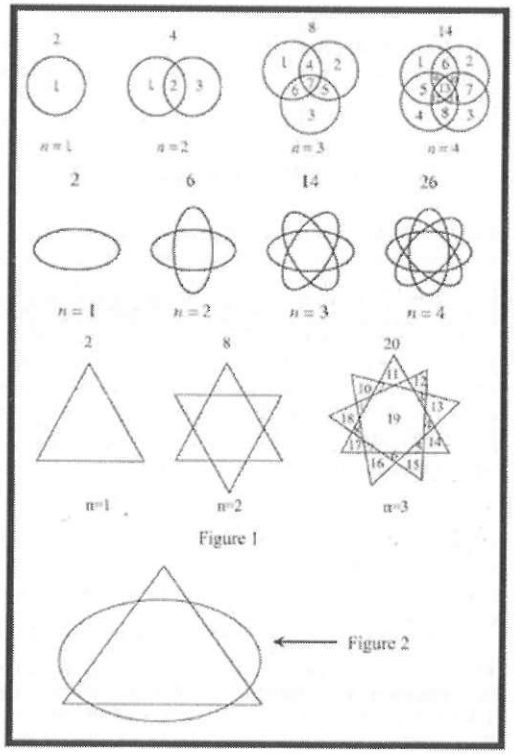



Figure 1




На рисунке 2 восемь областей образованы пересечением одного эллипса и одного треугольника. Здесь Вам следует решить обратную задачу. По заданному максимальному количеству областей следует найти количество эллипсов, окружностей и треугольников, которое могло их образовать.

**Вход.** Состоит из менее чем 300 строк. Каждая строка содержит 32-битовое беззнаковое целое N – максимальное количество областей, образованное $m$ эллипсами, $n$ окружностями и $p$ треугольниками. Последняя строка содержит -1 и не обрабатывается. Все числа кроме -1 в последней строке положительны.

**Выход.** Для каждого теста необходимо вывести две или более строк. Первая строка каждого теста содержит его номер. Каждая из следующих строк содержит три целых числа – возможные значения $m$, $n$ и $p$, для которых образуется максимальное количество областей N. Если существует несколько решений, то их следует отсортировать сначала по возрастанию $m$, а потом по возрастанию $n$. Если решения не существует, то вывести строку "Impossible.". Выводить следует только те решения, для которых $0 \le m < 100$, $0 \le n < 20000$ и $0 \le p < 100$.

**Пример входа**

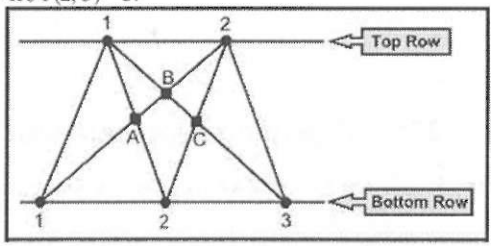
```
20
10
-1
```

**Пример выхода**

```
Case 1:
0 0 3
0 1 2
1 0 2
1 3 0
Case 2:
Impossible.
```

## 1539. Сколько точек пересечения?

Имеются две строки. На верхней строке отмечено $a$ точек, а на нижней $b$ точек. Соединим отрезком каждую точку верхней строки с каждой точкой нижней строки. Точки расположены так, что количество точек, полученных в результате пересечения отрезков, максимально. Для достижения этой цели достаточно чтобы никакие три отрезка не пересекались в одной точке. Точки на верхней и нижней строках в подсчет не включаются, в них могут пересекаться любое количество отрезков. По значениям $a$ и $b$ Вам необходимо вычислить $P(a, b)$ – максимальное количество точек пересечения, расположенное между двумя строками. Например, пусть $a = 2$ и $b = 3$. Из рисунка видно, что $P(2, 3) = 3$.




**Вход.** Каждая строка содержит два натуральных числа $a$ ($0 < a \le 20000$) и $b$ ($0 < b \le 20000$). Последний тест содержит два нуля и не обрабатывается. Входные данные содержат не более 1200 тестов.

**Выход.** Для каждого теста в отдельной строке вывести его номер и значение $P(a, b)$. Результат помещается в 64-битовое знаковое целое.

**Пример входа**

```
2 2
2 3
3 3
0 0
```

**Пример выхода**

```
Case 1: 1
Case 2: 3
Case 3: 9
```

## 1987. Следующая перестановка

На вход программы поступает строка из десятичных цифр. Вывести перестановку этих десятичных цифр, дающую следующее по величине за заданным десятичное число. Например:

123 → 132

279134399742 → 279134423799

Вполне возможно, что входные данные могут содержать набор цифр, не имеющих искомой следующей перестановки. Например, 987.

**Вход.** Первая строка содержит количество тестов $p$ ($1 \le p \le 1000$). Каждая следующая строка представляет собой отдельный тест и содержит его номер и соответствующий набор из не более чем 80 десятичных цифр.

**Выход.** Ответ на каждый тест следует выводить в отдельной строке. Если для заданного набора цифр не существует следующей перестановки, то выведите сначала номер теста и далее через пробел строку BIGGEST. Если же решение существует, то сначала выведите также номер теста, а затем через пробел найденную следующую перестановку входных цифр.

**Пример входа**

```
3
1 123
2 279134399742
3 987
```

**Пример выхода**

```
1 132
2 279134423799
3 BIGGEST
```

## 2388. Номер по перестановке

Дана перестановка из $n$ чисел от 1 до $n$. Найти её номер в лексикографическом порядке.

**Вход.** Первая строка содержит целое число $n$ ($1 \le n \le 12$). В следующей строке задается перестановка из $n$ чисел, разделённых пробелами.

**Выход.** Вывести номер перестановки в лексикографическом порядке.

**Пример входа**

```
3
2 1 3
```

**Пример выхода**

```
3
```

## 2390. Разбиения на слагаемые

Перечислите все разбиения целого положительного числа $n$ на целые положительные слагаемые. Разбиения должны обладать следующими свойствами:

- Слагаемые в разбиениях идут в невозрастающем порядке.
- Разбиения перечисляются в лексикографическом порядке.

**Вход.** Содержит единственное целое число $n$ ($1 \le n \le 40$).

**Выход.** Выведите искомые разбиения по одному в строке.

**Пример входа**

```
4
```

**Пример выхода**

```
1 1 1 1
2 1 1
2 2
3 1
4
```

# АНАЛИЗ ЗАДАЧ

## 318. Биномиальные коэффициенты 1

Вычисления будем проводить в 64-разрядных беззнаковых целых числах (unsigned long long). Поскольку $C_n^k = C_n^{n-k}$, то при $n-k<k$ следует положить $k=n-k$. Очевидно, что

$$C_n^k = \frac{n!}{k!(n-k)!} = \frac{n\cdot(n-1)\cdot(n-2)\cdot\ldots\cdot(n-k+1)}{1\cdot2\cdot3\cdot\ldots\cdot k} = \frac{n}{1}\cdot\frac{n-1}{2}\cdot\frac{n-2}{3}\cdot\ldots\cdot\frac{n-k+1}{k}$$

Присвоим некоторой переменной *res* значение 1, после чего будем ее умножать на $\frac{n-i+1}{i}$ для всех $i$ от 1 до $k$. Каждый раз деление на $i$ будет целочисленным, однако при умножении можно получить переполнение. Пусть $d$ = НОД(*res* , $i$). Тогда операцию

$$res = res * (n-i+1) / i$$

перепишем через

$$res = (res / d) * ((n-i+1) / (i / d))$$

При такой реализации при умножении переполнения не будет (ответ является 64-разрядным беззнаковым целым числом). Заметим, что сначала следует выполнить деление $(n-i+1) / (i / d)$, а потом умножение *res* / $d$ на полученное частное.

## 390. Анаграммы

Пусть мультимножество содержит $n_1$ элементов $a_1$, $n_2$ элемента $a_2$, … , $n_k$ элементов $a_k$. Тогда число его перестановок равно полиномиальному коэффициенту

$$P(n_1, n_2, \ldots, n_k) = \binom{n}{n_1, n_2, \ldots, n_k} = \frac{n!}{n_1!n_2!\ldots n_k!},$$

где $n = n_1 + n_2 + \ldots + n_k$ – общее число элементов в мультимножестве.

Вычислим количество перестановок мультимножества, используя сочетания: $n_1$ элементов $a_1$ можно расположить на $n$ местах $C_n^{n_1}$ способами. На следующих $n - n_1$ оставшихся местах можно расположить $n_2$ элементов $a_2$ $C_{n-n_1}^{n_2}$ способами. Рассуждая подобным образом, получаем, что $n_k$ элементов $a_k$ можно расположить $C_{n-n_1-n_2-\ldots-n_{k-1}}^{n_k}$ способами. Согласно правилу произведения имеем:

$$P(n_1, n_2, \ldots, n_k) = C_n^{n_1} * C_{n-n_1}^{n_2} * \ldots * C_{n-n_1-n_2-\ldots-n_{k-1}}^{n_k} = \frac{n!}{n_1!n_2!\ldots n_k!}$$

## 1184. Гипер - устройство

Построим вектор $d = (|p_1 - q_1|, |p_2 - q_2|, \ldots, |p_n - q_n|) = (d_1, d_2, \ldots, d_n)$. Ответ *res* можно вычислить, используя формулу включения - исключения.

Если $d = (d_1, d_2)$, то $res = d_1 + d_2 - \gcd(d_1, d_2)$.

Если $d = (d_1, d_2, d_3)$, то $res = d_1 + d_2 + d_3 - \gcd(d_1, d_2) - \gcd(d_1, d_3) - \gcd(d_2, d_3) + \gcd(d_1, d_2, d_3)$.

## 1531. Ключи в коробке

Взорвав бомбой коробку, можно достать ключ из другой коробки. Ключом из другой коробки можно открыть следующую. И так далее, пока не будет вскрыта коробка, в которой лежит ключ от первой коробки. Одной бомбой можно открыть такое множество коробок, которые образуют цикл в перестановке ключей. Количество циклов в перестановке ключей равно числу бомб, необходимых для открытия всех коробок.

Например, пусть ключи образуют перестановку (2, 1, 3, 5, 4) = (12)(345). Она состоит из двух циклов, поэтому для извлечения всех ключей требуется две бомбы.

Количество перестановок из $n$ элементов, имеющих $k$ циклов, равно числу Стирлинга первого рода $s(n, k)$, которые вычисляются по рекуррентной формуле:

$$s(n, k) = s(n-1, k-1) + (n-1) * s(n-1, k)$$

Искомая вероятность равна количеству разных перестановок, состоящих из не более чем $m$ циклов, деленная на количество всех перестановок $n!$.

В массиве s[21][21] вычисляются числа Стирлинга, причем s[$n$][$k$] = $s(n, k)$. В переменной $a$ вычисляется сумма $s(n, 1) + s(n, 2) + \ldots + s(n, m)$, в переменной $b$ – число перестановок $n!$. Возвращаем результат в виде строки "$a$/$b$", предварительно сократив числитель и знаменатель на их наибольший общий делитель.

## 1532. Рукопожатие

Пронумеруем бизнесменов, сидящих за столом, числами 1, 2, ..., 2*$p$ = $n$. Обозначим через f($p$) количество способов, которыми они могут пожать друг другу руки согласно условию задачи. Рассмотрим все возможные рукопожатия первого бизнесмена. Он может протянуть руку только тому бизнесмену, который имеет четный номер. Если первый бизнесмен протягивает руку (2 * $k$) - ому, то с одной стороны пересечения их рук останется 2 * $k$ – 2 людей, которые могут пожать руки f($k$ – 1) способами, а с другой стороны 2 * $p$ – 2 * $k$ людей, которые могут пожать руки f($p$ – $k$) способами. Выбирая $k$ = 1, 2, …, $p$, получим:

$$f(p) = f(0) * f(p-1) + f(1) * f(p-2) + \ldots + f(k-1) * f(p-k) + \ldots + f(p-1) * f(0),$$
$$f(1) = 1,\ f(0) = 1$$

То есть $f(p) = \sum_{i=0}^{p-1} f(i)\cdot f(p-i-1)$ являются числами Каталана.

Ответом на задачу будет значение f($n$ / 2).

**Пример.** Значения чисел Каталана вычисляем в массиве cat по приведенной выше формуле:

| $P$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| f($p$) | 1 | 1 | 2 | 5 | 14 | 42 | 132 | 429 | 1430 | 4862 | 16796 |

## 1533. Генерация анаграмм

Сортируем символы входной строки по возрастанию. Далее используем встроенную функцию next_permutation для генерации всех перестановок. При этом следует написать собственную функцию сравнения символов. При стандартном (лексикографическом) сравнении любая буква верхнего регистра будет меньше любой буквы нижнего регистра. То есть при сортировке букв a, A, z, Z, r, R получим слово ARZarz. В задаче следует сортировать (и

генерировать перестановки) в соответствии с порядком AaBbCc…Zz, необходимо из букв a, A, z, Z, r, R получить AaRrZz.

**Пример.** Для строки aAb (1 тест) наименьшей перестановкой будет Aab, а наибольшей baA.

## 1534. Делимость

Воспользуемся фактом, что

numDivisible(L, R, *a*) = numDivisible(1, R, *a*) – numDivisible(1, L - 1, *a*).

Значение numDivisible(1, *n*, *a*) будем вычислять при помощи функции f(*n*, *a*). Рассмотрим процесс вычисления результата в зависимости от количества элементов в массиве *a*.

1. Если *a* содержит один элемент, то ответом будет значение *n* / *a*[0] (округление до целого производится вниз).

2. Пусть *a* содержит два элемента. Ответ будет меньше *n* / *a*[0] + *n* / *a*[1], так как будут существовать числа, которые делятся на *a*[0] и на *a*[1] одновременно. И в приведенной сумме такие числа будут считаться дважды. Количество чисел, делящихся одновременно на *a*[0] и на *a*[1], равно *n* / НОК(*a*[0], *a*[1]). Таким образом, для двух чисел

$$f(n, a) = n / a[0] + n / a[1] - n / НОК(a[0], a[1])$$

3. Пусть *a* содержит три элемента. Тогда согласно принципу включения – исключения

$$f(n, a) = n / a[0] + n / a[1] + n / a[2] -$$
$$- n / НОК(a[0], a[1]) - n / НОК(a[1], a[2]) - n / НОК(a[0], a[2]) + n / НОК(a[0], a[1], a[2])$$

Поскольку по условию задачи массив *a* содержит от 1 до 18 элементов, то перебрать все подмножества множества *a* можно не более чем за $2^{18}$ операций. При этом если подмножество $\{a_{i_1}, a_{i_2}, ..., a_{i_k}\}$ содержит нечетное число элементов, то значение *n* / НОК($a_{i_1}, a_{i_2}, ..., a_{i_k}$) следует прибавлять к накапливаемой сумме (ответу), если четное – то вычитать.

Если при вычислении НОК($a_{i_1}, a_{i_2}, ..., a_{i_k}$) текущее значение НОК($a_{i_1}, a_{i_2}, ..., a_{i_j}$) станет больше *n* для некоторого $j < k$, то процесс вычисления НОК($a_{i_1}, a_{i_2}, ..., a_{i_k}$) можно завершить, так как тогда *n* / НОК($a_{i_1}, a_{i_2}, ..., a_{i_k}$) = 0.

## 1535. Небоскребы

Предположим, что осталось расположить *n* домов, которые следует расположить так, чтобы слева было видно *leftSide*, а справа – *rightSide* домов. Пусть это можно сделать f(*n*, *leftSide*, *rightSide*) способами. Рассмотрим дом с наименьшей высотой. Если его поставить слева, то он будет всегда видим, а оставшиеся дома можно будет расставить f(*n* – 1, *leftSide* – 1, *rightSide*) способами. Если дом с наименьшей высотой поставить справа, то он всегда будет оставаться видимым справа, остальные дома можно будет расставить f(*n* – 1, *leftSide*, *rightSide* – 1) способами. Не с краю наименьший дом можно поставить *n* – 2 способами. В этом случае он не будет видим, поэтому оставшиеся дома можно расставить (*n* – 2) * f(*n* – 1, *leftSide*, *rightSide*) способами. Заметим, что при *n* = 1 единственная возможная расстановка будет лишь при *leftSide* = *rightSide* = 1. Получаем рекуррентное соотношение:

$$f(n, leftSide, rightSide) = f(n-1, leftSide-1, rightSide) + f(n-1, leftSide, rightSide-1) + (n-2) * f(n-1, leftSide, rightSide),$$
$$f(1, 1, 1) = 1,$$
$$f(1, x, y) = 0, \text{ когда } x \text{ и } y \text{ одновременно не равны } 1$$

## 1536. Любимый ребенок мешает

**Первое решение.** Рассмотрим формулу включения - исключения. Пусть S – некоторое множество, P = {$p_1, p_2, ..., p_n$} – свойства, которые могут иметь элементы из S. Обозначим через S($p_1 p_2 ... p_n$) количество элементов из S, которые имеют одновременно свойства $p_1, p_2, ..., p_n$. Тогда количество элементов из S, которые не имеют ни одного свойства $p_i$, равно

$$|S| - \sum_{i=1}^{n} S(p_i) + \sum_{i,j} S(p_i p_j) - ... + (-1)^k \sum_{p_{i_1} \dots p_{i_k}} S(p_{i_1} p_{i_2} ... p_{i_k}) + ... + (-1)^n \sum S(p_1 p_2 ... p_n)$$

Рассмотрим в качестве S множество всех перестановок чисел от 1 до *n*. Тогда $p_i$ – это свойство перестановки оставлять на месте элемент *i*. Беспорядком называется такая перестановка, в которой не существует ни одного свойства $p_i$.

Таким образом S($p_{i_1} p_{i_2} ... p_{i_k}$) = (*n* – *k*)!, а количество перестановок чисел от 1 до *n*, которые оставляют *k* элементов на месте, равно $C_n^k$. Отсюда следует, что количество перестановок - беспорядков равно

$$n! - C_n^1 (n-1)! + C_n^2 (n-2)! + ... + (-1)^k C_n^k (n-k)! + ... (-1)^n =$$
$$n! \left(1 - 1 + \frac{1}{2!} - \frac{1}{3!} + ... + \frac{(-1)^k}{k!} + ... + \frac{(-1)^n}{n!}\right) \approx \frac{n!}{e}$$

**Пример.** Пусть имеется *n* = 3 элемента. Тогда количество перестановок, у которых на первом месте стоит 1, равно 2: это перестановки (1, 2, 3) и (1, 3, 2). Количество перестановок, у которых на втором месте стоит 2, равно 2: это перестановки (1, 2, 3) и (3, 2, 1). Тройка на третьем месте стоит в перестановках (1, 2, 3) и (2, 1, 3). Таким образом, S($p_1$) = S($p_2$) = S($p_3$) = 2.

Перестановкой, в которой на первом месте стоит 1, а на втором 2, является (1, 2, 3). То есть S($p_1 p_2$) = 1. Аналогично S($p_2 p_3$) = S($p_1 p_3$) = 1. Заметим, что S($p_1 p_2 p_3$) = 1. Таким образом, для трех вещей количество перестановок – беспорядков равно

$$3! - 2 - 2 - 2 + 1 + 1 + 1 - 1 = 2$$

Этими двумя беспорядками будут перестановки (2, 3, 1) и (3, 1, 2).

**Теорема.** Обозначим через f(*n*) количество перестановок – беспорядков для множества {1, 2, …, *n*}. Тогда имеет место рекуррентное соотношение (1):

$$f(2n) = 2n * f(2n-1) + 1,$$
$$f(2n+1) = (2n+1) * f(2n) - 1,$$
$$f(1) = 0$$

**Доказательство.** Для четного аргумента имеем:

$$f(2n) = (2n)! \left(1 - 1 + \frac{1}{2!} - \frac{1}{3!} + ... + \frac{1}{(2n)!}\right) = (2n)! \left(1 - 1 + \frac{1}{2!} - \frac{1}{3!} + ... - \frac{1}{(2n-1)!}\right) + 1 =$$
$$(2n) \left((2n-1)! \left(1 - 1 + \frac{1}{2!} - \frac{1}{3!} + ... - \frac{1}{(2n-1)!}\right)\right) + 1 = (2n)\, f(2n-1) + 1.$$

Аналогично получаем для нечетного аргумента:

$$f(2n+1) = (2n+1)! \left(1 - 1 + \frac{1}{2!} - \frac{1}{3!} + ... - \frac{1}{(2n+1)!}\right) =$$
$$(2n+1)! \left(1 - 1 + \frac{1}{2!} - \frac{1}{3!} + ... + \frac{1}{(2n)!}\right) - 1 =$$
$$(2n+1) \left((2n)! \left(1 - 1 + \frac{1}{2!} - \frac{1}{3!} + ... + \frac{1}{(2n)!}\right)\right) - 1 = (2n+1)\, f(2n) - 1.$$

Задача решается вычислением всех значений f(*n*) для $n \le 800$.

**Следствие.** Количество беспорядков f($n$) можно также записать в виде

$$f(n) = n * f(n-1) + (-1)^n$$

Также имеет место рекуррентность:

$$f(n) = (n-1) * (f(n-1) + f(n-2))$$

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| f($n$) | 0 | 1 | 2 | 9 | 44 | 265 | 1854 | 14833 | 133496 |

**Второе решение.** Выберем число, которое будет стоять на 1-ой позиции. Это может быть любое число, кроме единицы (имеем $(n-1)$ вариантов для первого числа). Пусть число, которое стоит на первой позиции, равно $k$. Рассмотрим число, которое будет стоять на $k$-й позиции. Есть два варианта:

а) Если на $k$-й позиции стоит 1, то поменяли числа 1 и $k$ местами, и оставшиеся $(n-2)$ числа необходимо расставить на оставшиеся $(n$-2) позиции, то есть задача сведена к подзадаче для $(n-2)$ чисел, так как удалили два числа вместе с соответствующими позициями.

б) Допустим, что на $k$-й позиции стоит не 1, а какое-то другое число. Рассмотрим подзадачу, в которой расставим на позиции со 2-й по $n$-ую числа 1, 2, ... $k-1$, $k+1$, ..., $n$, то есть все числа кроме $k$, причем число 1 стоит не на позиции $k$. Эта подзадача не аналогична исходной, т.к. множество номеров позиций не совпадает с множеством чисел, которые мы расставляем. Но мы можем в этой подзадаче заменить число 1 на число $k$, и при этом условие задачи не нарушится, потому что знаем, что число 1 не стоит на $k$-й позиции, как было оговорено раньше. Таким образом, если заменить в этой подзадаче число 1 на число $k$, то получаем исходную задачу, но для $(n-1)$ чисел. Мы имеем право сделать такую замену, т.к. в этой подзадаче не было 1-й позиции (т.е. после замены не появятся новые варианты перестановки), и заменяемое число не стояло на $k$-й позиции ни в одной из перестановок (то есть замена не приведет к тому, что некоторые перестановки перестанут удовлетворять условию задачи).

Пусть ответом задачи будет f($n$). Есть $(n-1)$ вариантов выбора числа для 1-й позиции. Пусть это число $k$. Если число 1 стоит на $k$-й позиции, то оставшиеся числа можно расставить f($n-2$) способами. Если число 1 стоит не на $k$-й позиции, то количество расстановок чисел {1, 2, ..., $k-1$, $k+1$, ..., $n$} на позициях 2...$n$ будет равняться f($n-1$). Таким образом, имеем рекуррентность (2):

$$f(n) = (n-1) * (f(n-1) + f(n-2)),$$
$$f(1) = 0,\ f(2) = 1$$

**Пример.** Вычислим значения f($i$), используя формулу и рекуррентное соотношение.

| рекуррентность (1) | формула |
|---|---|
| $f(2) = 2 * f(1) + 1 = 1$ | $f(2) = 2!\,(1 - 1 + \frac{1}{2!}) = 1$ |
| $f(3) = 3 * f(2) - 1 = 3 - 1 = 2$ | $f(3) = 3!\,(1 - 1 + \frac{1}{2!} - \frac{1}{3!}) = 3 - 1 = 2$ |
| $f(4) = 4 * f(3) + 1 = 8 + 1 = 9$ | $f(4) = 4!\,(1 - 1 + \frac{1}{2!} - \frac{1}{3!} + \frac{1}{4!}) = 12 - 4 + 1 = 9$ |

| рекуррентность (2) |
|---|
| $f(3) = 2 * (f(1) + f(2)) = 2 * (0 + 1) = 2$ |
| $f(4) = 3 * (f(2) + f(3)) = 3 * (1 + 2) = 9$ |
| $f(5) = 4 * (f(3) + f(4)) = 4 * (2 + 9) = 44$ |

# 1537. Полоса

Пусть имеется $w$ белых квадратов и $g$ групп черных квадратов. Поскольку группы черных квадратов не касаются, то должно существовать как минимум $g-1$ белых квадратов (если таковых не существует – то ответ 0). При этом $w-(g-1)$ белых квадратов будут свободными, и их можно расположить в любом месте по отношению к группам черных квадратов. Количество мест, по которым можно расположить свободные белые квадраты, равно $(g+1)$. Это можно сделать $\overline{C}_{g+1}^{w-(g-1)}$ способами. Здесь $\overline{C}_n^r = C_{n+r-1}^r$ – количество способов расположить $r$ одинаковых объектов по $n$ разным местам, где каждое место может содержать произвольное количество объектов (сочетания с повторениями). Таким образом

$$\overline{C}_{g+1}^{w-(g-1)} = C_{g+1+w-g+1-1}^{w-(g-1)} = C_{w+1}^{w-(g-1)}$$

**Пример.** Рассмотрим второй тест. Существует три полосы длины 5 с кодом 1 2:

Число свободных белых квадратов равно 1, число групп 2. Следовательно количество мест, куда можно поставить 1 свободный белый квадрат, равно 3. Количество вариантов искомой расстановки равно 3, так как свободный квадрат можно поставить на одно из 3 мест.

Воспользуемся формулой. Число белых квадратов $w = 2$, число групп $g = 2$. Количество полос равно $C_{2+1}^{2-(2-1)} = C_3^1 = 3$.

# 1538. Мысли наоборот

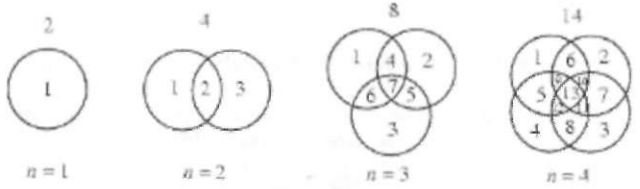
Пусть $n$ фигур разбивают плоскость на f($n$) частей. Одна фигура разбивает плоскость на 2 части. Каждая следующая $n$ - ая фигура должна иметь максимально возможное количество пересечений $k$ с каждой из $(n-1)$ предыдущих фигур. Две окружности могут пересекаться максимум в двух точках ($k = 2$), два эллипса в четырех ($k = 4$), а два треугольника в шести ($k = 6$). Тогда

$$f(n) = f(n-1) + k * (n-1),$$
$$f(1) = 2$$

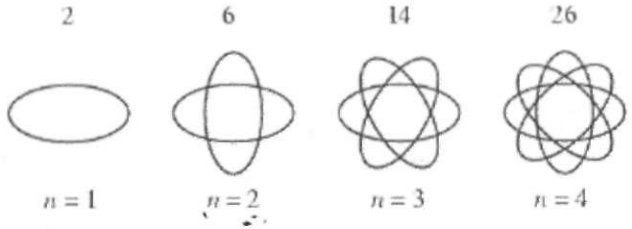
Решим рекуррентное уравнение:

$$f(n) = k * ((n-1) + (n-2) + \ldots + 1) + 2 = k * \frac{n \cdot (n-1)}{2} + 2$$

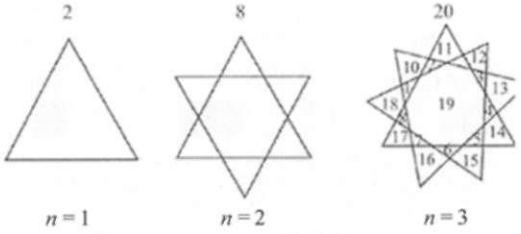
Заметим, что f(0) = 1 для любого $k$ (ноль фигур делят плоскость на одну часть).




Окружности на плоскости




Эллипсы на плоскости




Треугольники на плоскости

Из выше приведенной формулы следует, что $m$ эллипсов, $n$ окружностей и $p$ треугольников могут разделить плоскость максимум на $2 + 2m(m-1) + n(n-1) + 4mn + 3p(p-1) + 6pn + 6pm$ частей. $m$ эллипсов и $n$ окружностей могут иметь максимум $4mn$ точек пересечения, $p$ треугольников и $n$ окружностей или $m$ эллипсов – соответственно $6pn$ и $6pm$ точек пересечения. Приравняем это значение к $s$ и перепишем выражение как квадратное уравнение относительно $n$:

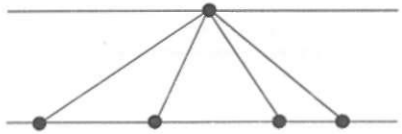
$$n^2 + n(4m + 6p - 1) + 2 + 2m(m-1) + 3p(p-1) + 6pm - s = 0$$

Если дискриминант уравнения является полным квадратом для некоторых целых $m$ и $p$, $0 \le m, p < 100$, то ищем соответственное неотрицательное значение $n$ и проверяем его принадлежность интервалу $0 \le n < 20000$. Остается перебрать все пары $(m, p)$ из заданного интервала и для каждой из них решить квадратное уравнение. Найденные тройки остается упорядочить по заданному в условии задачи правилу.

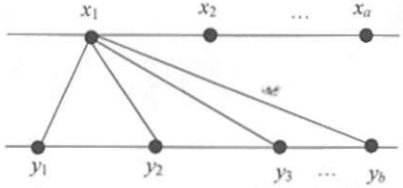
**Пример.** Рассмотрим первый тест. На 20 частей плоскость можно разбить следующими комбинациями фигур: 3 треугольника, 1 окружность и два треугольника, 1 эллипс и два треугольника, 1 эллипс и две окружности.

## 1539. Сколько точек пересечения?

Обозначим через $f(a, b)$ искомое количество точек пересечения. Очевидно, что $f(1, b) = 0$, так как при $a = 1$ никакие два отрезка не пересекаются:



Рассмотрим общий случай. Пусть $x_1, x_2, \ldots, x_a$ – точки на первой прямой, $y_1, y_2, \ldots, y_b$ – точки на второй прямой. Соединим точку $x_1$ с точками $y_1, y_2, \ldots, y_b$. На отрезке $x_1y_1$ точек пересечения не будет. На отрезке $x_1y_2$ будут лежать точки пересечения с отрезками $y_1x_2, y_1x_3, \ldots, y_1x_a$ (всего $a - 1$ точек). На отрезке $x_1y_j$ будут лежать точки пересечения с отрезками $y_kx_k$, где $i < j$, $2 \le k \le a$ (всего $(j-1) * (a-1)$ точек). Количество точек пересечения, которые лежат на отрезках исходящих из $x_1$, равно $(0 + 1 + 2 + \ldots + (b-1)) * (a-1) = b * (b-1) / 2 * (a-1)$.




Итак, из $f(a, b)$ точек $b * (b-1) / 2 * (a-1)$ точек лежат на отрезках исходящих из $x_1$, а остальные точки лежат на отрезках с концами в $x_2, \ldots, x_a$. Имеем рекуррентное соотношение:

$$f(a, b) = b * (b-1) / 2 * (a-1) + f(a-1, b)$$

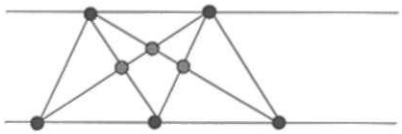
Развернув его, получим:

$$f(a, b) = b * (b-1) / 2 * (a-1) + f(a-1, b) =$$
$$b * (b-1) / 2 * (a-1) + b * (b-1) / 2 * (a-2) + f(a-2, b) = \ldots =$$
$$b * (b-1) / 2 * (a-1) + b * (b-1) / 2 * (a-2) + \ldots + b * (b-1) / 2 * 1 =$$
$$b * (b-1) / 2 * a * (a-1) / 2$$

Таким образом, максимальное количество точек пересечения равно

$$\frac{a \cdot (a-1)}{2} * \frac{b \cdot (b-1)}{2}$$

**Пример.** Рассмотрим второй тест, для которого $a = 2$, $b = 3$. Максимально возможное количество точек пересечения отрезком равно 3 и показано на рисунке:



## 1987. Следующая перестановка

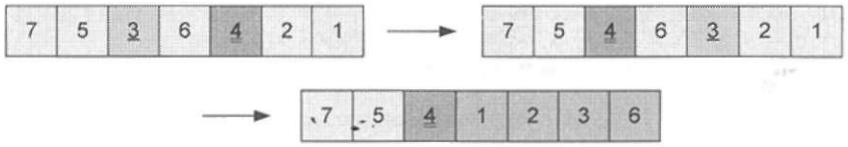
Задачу можно решить при помощи функции next_permutation.

Однако далее опишем алгоритм нахождения лексикографически следующей перестановки. Большей будем называть перестановку, в которой раньше встретился элемент, больший соответствующего ему элемента во второй перестановке. Например, если S = (3, 5, 4, 6, 7), а L = (3, 5, 6, 4, 7), то S < L.

Покажем на примере, как найти перестановку, следующую за P = (5, 6, 7, 4, 3). Просматриваем текущую перестановку справа налево, следя за тем, чтобы каждое следующее число было больше предыдущего. Останавливаемся, когда правило нарушится. Место остановки подчеркнуто: (5, 6, 7, 4, 3). Потом снова просматриваем пройденный путь (справа налево) до тех пор, пока не дойдем до первого числа, которое больше отмеченного. Место второй остановки отметим двойным подчеркиванием: (5, 6, 7, 4, 3). Поменяем местами отмеченные числа: (5, 7, 6, 4, 3). Теперь все числа, расположенные справа от двойного подчеркивания, упорядочим в порядке возрастания. Поскольку они до этих пор были

упорядочены в убывающем порядке, то достаточно перевернуть указанный отрезок. Получим Q = (5, 7, 3, 4, 6). Это и есть перестановка, непосредственно следующая за P.

**Пример.** Найдем перестановку, следующую за P = (7, 5, 3, 6, 4, 2, 1).




## 2388. Номер по перестановке

Пусть $m_1m_2...m_n$ – входная перестановка. Обозначим через $d_i$ количество таких пар $(i, j)$, что $i < j$ и $m[i] > m[j]$ (элементы $m[i]$ и $m[j]$ образуют инверсию).

Тогда номер перестановки в лексикографическом порядке определяется по формуле:

$$d_1 * (n-1)! + d_2 * (n-2)! + \ldots + d_{n-1} * 1! + 1$$

**Пример.** Для перестановки (2, 1, 3) массив $d = (1, 0, 0)$. Номер перестановки равен $1 * 2! + 1 = 3$.

Для перестановки (1, 4, 3, 2) массив $d = (0, 2, 1, 0)$. Номер перестановки равен $0 * 3! + 2 * 2! + 1 * 1! + 1 = 6$.

Для перестановки (3, 2, 1, 4) массив $d = (2, 1, 0, 0)$. Номер перестановки равен $2 * 3! + 1 * 2! + 0 * 1! + 1 = 15$.

Для лексикографически наименьшей перестановки $(1, 2, \ldots, n)$ массив $d = (0, 0, \ldots, 0)$. Ее номер равен 1, так как все слагаемые кроме последнего равны нулю.

Для лексикографически наибольшей перестановки $(n, n-1, n-2, \ldots, 1)$ массив $d = (n-1, n-2, n-3, \ldots, 0)$. Ее номер равен $(n-1) * (n-1)! + (n-2) * (n-2)! + \ldots + 1 * 1! + 1 = n!$ Эту формулу можно доказать методом математической индукции:

***База индукции.*** $1 = 1!$.

***Шаг индукции.*** $n! + n * n! = n! (1 + n) = (n + 1)!$

## 2390. Разбиения на слагаемые

Пусть $n = x_1 + x_2 + \ldots + x_k$ – разбиение числа $n$ на натуральные слагаемые. Исходя из условия задачи, слагаемые любого разбиения удовлетворяют неравенству: $x_1 \geq x_2 \geq \ldots \geq x_k$. $x_1$ может принимать значения от 1 до $n$, а каждое $x_i$ $(2 \leq i \leq k)$ может принимать значение от 1 до $x_{i-1}$. То есть следует сгенерировать все возможные последовательности $x_1, x_2, \ldots, x_k$, удовлетворяющие выше приведенным условиям. Рекурсивность процедуры генерации разбиений состоит в том, что если при разбиении числа $n$ в качестве первого слагаемого выбрано $x_1$ $(x_1 \leq n)$, то далее необходимо генерировать все возможные разбиения числа $n - x_1$ со слагаемыми, не большими $x_1$.

# РЕАЛИЗАЦИЯ ЗАДАЧ

## 318. Биномиальные коэффициенты 1

Функции вычисления наибольшего общего делителя и биномиального коэффициента имеют вид:

```
unsigned long long gcd(unsigned long long a, unsigned long long b)
{
  return (!b) ? a : gcd(b,a % b);
}

unsigned long long Cnk(unsigned long long n, unsigned long long k)
{
  unsigned long long _CnkRes = 1, t, i, i1;
  if (k > n - k) k = n - k;
  for(i = 1; i <= k; i++)
  {
    t = gcd(_CnkRes, i);
    _CnkRes = (_CnkRes / t) * ((n - i + 1) / (i / t));
  }
  return _CnkRes;
}
```

## 390. Анаграммы

Читаем входную строку, сортируем буквы в лексикографическом порядке. Изначально положим *res* = *len*!, где *len* равно длине строки.

```
gets(s); len = strlen(s);
sort(s,s+len); c = 1; res = fact(len);
```

Для каждого набора подряд идущих букв вычисляем его длину *c*. Делим значение *len* на *c*!. По окончанию цикла выводим результат *res*.

```
for(i = 0; i < len - 1; i++)
  if (s[i] == s[i+1]) c++; else res /= fact(c), c = 1;
res /= fact(c);
printf("%lld\n",res);
```

## 1184. Гипер - устройство

Считываем входные данные для каждого теста.

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

Вычисляем массив $d = (|p_1 - q_1|, |p_2 - q_2|, \ldots, |p_n - q_n|) = (d_1, d_2, \ldots, d_n)$.

```
  for(i = 0; i < dim; i++) d[i] = (a[i] > b[i]) ? a[i] - b[i] : b[i] - a[i];
```

```
res = 0;
```

Перебираем все подмножества множества $\{d_1, d_2, ..., d_n\}$. Каждое подмножество генерируется по числу $i$ ($1 \le i \le 2^{dim} - 1$), выбирая из его двоичного представления позиции, на которых стоят единицы.

```
for(i = 1; i < 1<<dim; i++)
{
  ptr = temp = cnt = 0; j = i;
  while(j > 0)
  {
    if (j & 2)
    {
      temp = gcd(temp,d[ptr]);
      cnt++;
    }
    ptr++; j /= 2;
  }
```

Если подмножество состоит из четного количества элементов, то НОД его элементов вычитается их общего результата. Если из нечетного – то прибавляется.

```
  res += (cnt & 2) ? temp : -temp;
}
```

Выводим результат

```
  printf("Case %lld: %lld\n",test+1,res);
}
```

## 1531. Ключи в коробке

Функция getAllKeys по заданным $n$ и $m$ возвращает такие $a$ и $b$, для которых $a / b$ является вероятностью того, что можно открыть все коробки.

```
void getAllKeys(int n, int m, long long &a, long long &b)
{
  int i, k;
  long long d;
  memset(s,0,sizeof(s));
  s[0][0] = 1;
  for(i = 1; i <= n; i++)
    for(k = 1; k <= n; k++)
      s[i][k] = s[i-1][k-1] + (i-1) * s[i-1][k];
```

В переменной $a$ вычисляем сумму $s(n, 1) + s(n, 2) + ... + s(n, m)$

```
  for (a = 0, i = 1; i <= m; i++) a += s[n][i];
```

В переменной $b$ вычисляем факториал числа $n$.

```
  for(b = i = 1; i <= n; i++) b *= i;
```

Сокращаем дробь $a / b$.

```
  d = gcd(a,b); a /= d; b /= d;
}
```

Основной цикл программы.

```
while(scanf("%d %d",&n,&m) == 2)
{
  getAllKeys(n,m,a,b);
  printf("%lld/%lld\n",a,b);
}
```

## 1532. Рукопожатие

В ячейках cat[$i$] будем вычислять $i$-ое число Каталана.

```
long long cat[51];
```

Используя рекуррентную формулу $f(p) = \sum_{i=0}^{p-1} f(i) \cdot f(p-i-1)$, функция countPerfect вычисляет числа Каталана от нулевого до $n$-го.

```
void countPerfect(int n)
{
  int i, j;
  cat[0] = cat[1] = 1;
  for(i = 2; i <= n; i++)
    for(j = 0; j < i; j++)
      cat[i] += cat[j] * cat[i - j - 1];
}
```

Основной цикл программы. Для каждого входного значения $n$ выводим cat[$n$ / 2].

```
countPerfect(50);
while(scanf("%d",&n) == 1)
  printf("%lld\n",cat[n/2]);
```

## 1533. Генерация анаграмм

Функция lt будет использоваться при сортировке и генерации перестановок. Она сравнивает два символа в соответствии с порядком AaBbCc…Zz.

```
int lt(char a, char b)
{
  if (toupper(a) != toupper(b)) return (toupper(a) < toupper(b));
  return (a < b);
}
```

Читаем входную строку, вычисляем ее длину и сортируем символы в алфавитном порядке.

```
scanf("%s", &s);len = strlen(s);
sort(s,s+len,lt);
```

Выводим текущую анаграмму (перестановку символов) и генерируем следующую до тех пор пока это возможно.

```
do {
  printf("%s\n",s);
} while(next_permutation(s,s+len,lt));
```

## 1534. Делимость

Функция f(*n*, *a*) вычисляет значение numDivisible(1, *n*, *a*). Функция gcd вычисляет наибольший общий делитель двух чисел.

```
int f(int N, int *a)
{
  int i, j, bits, temp, res = 0;
  long long lcm;
  for(i = 1; i < (1<<n); i++)
  {
    lcm = 1; bits = 0;
    for(j = 0; j < n; j++)
      if (i & (1 << j))
      {
        bits++;
        temp = gcd(lcm,a[j]);
        lcm = lcm / temp * a[j];
        if (lcm > N) break;
      }
    if (bits % 2) res += N / lcm; else res -= N / lcm;
  }
  return res;
}
```

Основная часть программы.

```
while(scanf("%d %d",&l,&r) == 2)
{
  scanf("%d",&n);
  for(i = 0; i < n; i++) scanf("%d",&a[i]);
  res = f(r,a) - f(l - 1,a);
  printf("%d\n",res);
}
```

## 1535. Небоскребы

Объявим трехмерный массив dp, ячейка dp[*i*][*j*][*k*] которого будет сохранять значение f(*i*, *j*, *k*).

```
#define MAX 101
int dp[MAX][MAX][MAX];
```

Функция f(*n*, *leftSide*, *rightSide*) возвращает количество способов, которыми можно расположить *n* домов, так, чтобы слева было видно *leftSide*, а справа *rightSide* домов.

```
int f(int n, int leftSide, int rightSide)
{
  if (n == 1) return (leftSide == 1 && rightSide == 1) ? 1 : 0;
  if ((leftSide < 1) || (rightSide < 1)) return 0;
  if (dp[n][leftSide][rightSide] != -1) return dp[n][leftSide][rightSide];
  dp[n][leftSide][rightSide] = (int)(((long long)f(n-1,leftSide-1,rightSide) +
            (long long)f(n-1,leftSide,rightSide-1) +
            (n-2)*(long long)f(n-1,leftSide,rightSide)) % 1000000007);
  return dp[n][leftSide][rightSide];
}
```

Основная часть программы.

```
while(scanf("%d %d %d",&n,&leftSide,&rightSide) == 3)
{
  memset(dp,-1,sizeof(dp));
  res = f(n,leftSide,rightSide);
  printf("%d\n",res);
}
```

## 1536. Любимый ребенок мешает

В массиве p[801][MAX], MAX = 2000 будем хранить значения f(*n*) (p[*n*] = f(*n*)). pt[*i*] будет содержать количество цифр в числе f(*i*), m – рабочий массив.

```
#define MAX 2000
int ptr, n, i, m[MAX];
int p[801][MAX], pt[MAX];
```

Функция mult(int n) будет пересчитывать значение f(*n*) в соответствии с приведенным выше рекуррентным соотношением (1).

```
void mult(int n)
{
  int i, carry, temp;
  int res[MAX];
  carry = 0;
  for(i = 0; i <= ptr; i++)
  {
    temp = (carry+m[i]*n);
    res[i] = temp % 10;
    carry = temp / 10;
  }
  while(carry > 0)
  {
    res[++ptr] = carry % 10;
    carry /= 10;
  }
  memcpy(m,res,sizeof(res));
  if (n % 2)
  {
    i = 0; while(!m[i]) {m[i] = 9; i++;}; m[i]--;
  } else
  {
    i = 0; while(m[i] == 9) {m[i] = 0; i++;}; m[i]++;
  }
}
```

Основная часть программы. Для каждого *i* (2 ≤ *i* ≤ 800) вычисляем значение f(*i*) в массиве *m* и сохраняем его в ячейке p[*i*].

```
memset(m,0,sizeof(m)); ptr = 0;
for(i = 2; i <= 800; i++)
{
   mult(i);
   memcpy(p[i],m,sizeof(m));
   pt[i] = ptr;
}
```

Для каждого входного значения $n$ выводим содержимое p[$n$].

```
while (scanf("%d",&n),n!=-1)
{
  if (n == 1) printf("0"); else
  for(i = pt[n]; i >= 0; i--) printf("%d",p[n][i]);printf("\n");
}
```

## 1537. Полоса

Технически задача сводится к вычислению биномиального коэффициента, значение которого не помещается в 64-битный целый тип. Воспользуемся классом BigInteger.

```
int n, g, w, group[200];
int i, j, tests;

class BigInteger{  .  . .};

BigInteger Cnk(int k, int n)
{
  BigInteger res(1);
  for(int i = 1; i <= k; i++)
    res = res * (n - i + 1) / i;
  return res;
}
```

Читаем количество тестов *tests*. Для каждого теста считываем код в массив *group*, параллельно суммируя количество черных квадратов. Вычитаем его из $n$, получим число белых квадратов $w$. Если количество белых квадратов $w$ меньше $g - 1$, то полосы с таким кодом не существует. Иначе вычисляем и выводим результат $C_{w+1}^{w-(g-1)}$.

```
scanf("%d",&tests);
for(i = 0; i < tests; i++)
{
  scanf("%d %d",&n,&g);
  w = 0;
  for(j = 0; j < g; j++)
  {
    scanf("%d",&group[j]);
    w += group[j];
  }
  w = n - w;
  if (w < g - 1) printf("0");
  else Cnk(w-g+1,w+1).print();
  printf("\n");
}
```

## 1538. Мысли наоборот

Читаем входные данные и выводим номер теста *CaseNo*.

```
CaseNo = 1;
while(scanf("%d",&s), s > 0)
{
  printf("Case %d:\n",CaseNo++);
```

Если входное значение $s$ равно 1, то ответом будут три нуля:

```
  if (s == 1)
  {
    printf("0 0 0\n");continue;
  }
```

Введем переменную флаг *found*, который будет равен 1, если для заданного $s$ будет найдена хотя бы одна тройка - решение. Изначально присвоим found значение 0.

```
    found = 0;
```

Совершаем перебор всех возможных значений $m$, $p$ ($0 \le m, p < 100$). По ходу вычисляем максимальное количество частей, на которое фигуры делят плоскость. Если на каком-то этапе значение суммы (res или res1) станет большей $s$, то выходим из цикла (нет смысла перебирать последующие значения соответствующей переменной).

```
  for(m = 0; m < 100; m++)
  {
    res = 2 + 2 * m * (m - 1);
    if (res > s) break;
    mas.clear();
    for(p = 0; p < 100; p++)
    {
      res1 = res + 3 * p * (p - 1) + 6 * m * p;
      if (res1 > s) break;
```

Имеются значения $m$ и $p$. Решаем приведенное выше квадратное уравнение относительно $n$. Вычисляем корень из дискриминанта det. Если дискриминант является полным квадратом (дробная часть числа det равна 0), то находим положительный корень уравнения $n$ и проверяем его принадлежность интервалу $0 \le n < 20000$. Для каждого $m$ будем накапливать соответствующие пары – решения ($n$, $p$) в переменной mas типа вектор: `vector<pair<int, int> > mas`.

```
      b = 4 * m + 6 * p - 1;
      det = sqrt(1.0 * b * b - 4 * (res1 - s));
      if (fabs(det - (int)det) < 1e-12)
      {
        n = (int)((-b + det) / 2);
        if ((n < 0) || (n >= 20000)) break;
        mas.push_back(make_pair(n,p));
      }
    }
```

Если для текущего значения $m$ найдена хотя бы одна пара решений ($n$, $p$) (массив mas не пустой), то отсортировать и вывести все тройки – решения. Сортировка по умолчанию будет производиться по первому элементу пары (значению $n$).

```
    if (mas.size())
    {
      sort(mas.begin(),mas.end());
      for(i = 0; i < mas.size(); i++)
      printf("%d %d %d\n",m,mas[i].first,mas[i].second);
      found = 1;
    }
  }
```

Если решений не найдено, то вывести соответствующее сообщение.

```
  if (!found) printf("Impossible.\n");
}
```

## 1539. Сколько точек пересечения?

Читаем входные данные и для каждого теста вычисляем результат по выше приведенной формуле. При заданных ограничениях на $a$ и $b$ в умножении переполнения не будет, если вычисления проводить в 64-битовом целочисленном типе.

```
cs = 1;
while(scanf("%lld %lld",&a,&b), a + b > 0)
{
  res = a * (a - 1) * b * (b - 1) / 4;
  printf("Case %d: %lld\n",cs++,res);
}
```

## 1987. Следующая перестановка

Решение при помощи функции next_permutation:

```
char s[1001];
scanf("%d",&tests);
while(tests--)
{
  scanf("%d %s",&cs,s);
  len = strlen(s);
  flag = next_permutation(s,s+len);
  printf("%d ",cs);
  if (!flag) puts("BIGGEST"); else puts(s);
}
```

Функция NextPerm генерирует лексикографически следующую перестановку.

```
int NextPerm(char *s)
{
  int Ipos, Jpos;
  Ipos = len - 2; Jpos = len - 1;
```

*Ipos* указывает на место первой остановки: s[*Ipos*] подчеркнуто один раз.

```
  while((s[Ipos] >= s[Ipos+1]) && (Ipos >= 0)) Ipos--;
  if (Ipos < 0) return 0;
```

*Jpos* указывает на место второй остановки: s[*Jpos*] подчеркнуто два раза.

```
  while(s[Jpos] <= s[Ipos]) Jpos--;
```

Меняем местами значения s[*Ipos*] и s[*Jpos*].

```
  swap(s[Ipos],s[Jpos]);
```

Переворачиваем хвост перестановки, находящийся правее от двойного подчеркивания.

```
  Jpos = len - 1;
  for(Ipos++; Ipos < Jpos; Ipos++, Jpos--) swap(s[Ipos],s[Jpos]);
  return 1;
}
```

Основная часть программы.

```
scanf("%d",&tests);
while(tests--)
{
  scanf("%d %s",&cs,s); len = strlen(s);
  printf("%d ",cs);
  if (NextPerm(s)) puts(s); else puts("BIGGEST");
}
```

## 2388. Номер по перестановке

Входную перестановку храним в массиве m. В массиве fact вычислим факториалы чисел: fact[$i$] = $i$!

```
#define MAX 13
int fact[MAX], m[MAX];
```

Основная часть программы. Читаем входную перестановку в массив m.

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

Заполняем массив факториалов чисел.

```
fact[0] = 1;
for(i = 1; i <= n; i++) fact[i] = fact[i-1] * i;

for(res = 0, i = 1; i <= n; i++)
{
```

Для заданного значения $i$ в переменной *inv* подсчитываем количество $d_i$ таких элементов m[$j$], для которых $i < j$ и m[$i$] > m[$j$].

```
  for(inv = 0, j = i + 1; j <= n; j++)
    if (m[j] < m[i]) inv++;
```

Добавляем к результату значение $d_i * (n - i)!$

```
  res += inv * fact[n-i];
}
```

Выводим ответ.

```
printf("%d\n",res + 1);
```

# 2390. Разбиения на слагаемые

В массиве x будем генерировать очередное разбиение числа *n*.

```
int x[50];
```

Процедура генерации разбиений Partitions имеет три аргумента: *pos* – текущая позиция в массиве *x*, *max* – максимально возможное слагаемое, которое может находиться в позиции *pos*, *number* – число, которое разбивается.

```
void Partitions(int pos, int max, int number)
{
  int i;
  if (!number)
  {
```

Если разбиваемое число *number* равно нулю, то выводим текущее разбиение, построенное в массиве x.

```
    printf("%d",x[0]);
    for (i = 1; i < pos; i++) printf(" %d",x[i]);
    printf("\n");
  } else
```

В позиции *pos* массива x может находиться любое число от 1 до min(*max*, *number*).

```
  for (i = 1; i <= min(max,number); i++)
  {
    x[pos] = i;
    Partitions(pos+1,i,number-i);
  }
}
```

Основная часть программы. Читаем входное значение *n* и запускаем генерацию всех разбиений.

```
scanf("%d",&n); Partitions(0,n,n);
```