Биномиальный коэффициент

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. Первая строка содержит количество тестов t ($t \le 50$). Каждая из следующих t строк содержит два целых числа n и k ($0 \le n < 2^{64}$ и $0 \le C_n^k < 2^{64}$).

Выход. Вывести t строк, каждая из которых содержит значение C_n^k для соответствующего теста.

Пример входа	Пример выхода
6 0 0 1 0 1 1 2 0 2 1 2 2	1 1 1 1 2 1

Вычисления будем проводить в 64-разрядных беззнаковых целых числах (unsigned long long). 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.

Для вычисления C_n^k следует совершить k итераций. Но что делать если следует вычислить $C_{2000000000}^{1999999999}$? Ответ не более 2^{64}, поэтому такие входные данные возможны. Поскольку $C_n^k = C_n^{n-k}$, то при $n - k < k$ следует положить $k = n - k$.

Пример

Рассмотрим следующий пример:

$$C_6^3 = \frac{6}{1}\cdot\frac{5}{2}\cdot\frac{4}{3} = 15\cdot\frac{4}{3}$$

Пусть *res* = 15, и осталось выполнить умножение $res * \frac{4}{3} = 15 * \frac{4}{3}$. Вычислим d = GCD(15 , 3) = 3. Поэтому $15 * \frac{4}{3} = (15 / 3) * \frac{4}{(3/3)} = 5 * \frac{4}{1} = 20$.

Реализация алгоритма

Функция вычисления наибольшего общего делителя.

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

Функция вычисления биномиального коэффициента C_n^k.

```
unsigned long long Cnk(unsigned long long n, unsigned long long k)
{
  unsigned long long CnkRes = 1, t, i;
  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;
}
```

Основная часть программы. Читаем входные данные. Последовательно обрабатываем тесты.

```
scanf("%d",&t);
while(t--)
{
  scanf("%llu %llu",&n,&k);
  res = Cnk(n,k);
  printf("%llu\n",res);
}
```

319. Биномиальные коэффициенты 2

Даны целые неотрицательные числа n и k. Найти разложение биномиального коэффициента C_n^k на простые множители.

Вход. Первая строка содержит количество тестов t ($t \le 10$). Каждая из следующих t строк является отдельным тестом и содержит числа n и k ($0 \le n \le 100000$, $0 \le k \le n$), разделенные пробелом.

Выход. Вывести t строк, каждая из которых содержит разложение числа C_n^k на простые множители для соответствующих входных значений.

Разложение натурального числа N на простые множители следует выводить следующим образом. Если N = 1, то необходимо вывести "1" (без кавычек), иначе пусть N = $p_1^{a_1} \cdot \ldots \cdot p_d^{a_d}$, где p_1, ..., p_d – все различные простые делители числа N, упорядоченные по возрастанию, и a_1, ..., a_d - натуральные числа (a_i равно максимальной степени, в которой p_i делит N). Тогда необходимо вывести строку вида

$$p_1[\hat{}a_1] * p_2[\hat{}a_2] * \ldots * p_d[\hat{}a_d]$$

Здесь $[\hat{}a_i]$ означает, что необходимо не писать $\hat{}a_i$, если $a_i = 1$.

Пример входа	**Пример выхода**
3 1 1 4 2 6 3	1 2 * 3 2^2 * 5

Запишем биномиальный коэффициент в виде

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

Будем раскладывать на множители каждый из множителей, встречающийся как в числителе, так и в знаменателе. Запоминать степени, с которыми входят в разложение C_n^k простые числа, будем в массиве deg: deg[i] ≠ 0 означает, что простое число i входит в разложение C_n^k со степенью deg[i]. При этом для каждого простого множителя i, входящего в числитель выше приведенной дроби, будем увеличивать deg[i] на 1. А если простой множитель i входит в знаменатель дроби, то будем уменьшать deg[i] на 1. Далее по содержимому массива deg выведем разложение числа C_n^k на простые множители.

Реализация алгоритма

В массиве primes отмечаем простые числа решетом Эратосфена. В массиве pr сохраняем простые числа. В массив deg заносим информацию о разложении C_n^k на простые множители: deg[i] ≠ 0 означает, что i – простое число, которое входит в разложение C_n^k со степенью deg[i].

```
#define MAX 100010
int primes[MAX], deg[MAX], pr[100];
```

Раскладываем число n на множители. Если множитель n стоит в биномиальном коэффициенте в числителе, то передаем *flag* = 1, если в знаменателе то *flag* = -1.

```
void factor(int n, int flag)
{
  int i;
  for(i = 0; pr[i] <= (int)sqrt(1.0*n); i++)
    while (n % pr[i] == 0) n /= pr[i], deg[pr[i]] += flag;
  if (n > 1) deg[n] += flag;
}
```

Решето Эратосфена. Строим массив primes, в котором primes[i] = 1, только если число i простое.

```
void gen_primes(void)
{
  int i, j;
  for(i = 0; i < MAX; i++) primes[i] = 1;
  primes[0] = primes[1] = 0;
  for(i = 2; i <= sqrt(1.0*MAX); i++)
    if (primes[i])
      for(j = i * i; j < MAX; j += i) primes[j] = 0;
```

Заносим в массив pr простые числа от 2 до $\lfloor\sqrt{100000}\rfloor$.

```
  for(j = i = 0; i <= sqrt(1.0*MAX); i++)
    if (primes[i]) pr[j++] = i;
```

В конце массива ставим заглушку (число, квадрат которого строго больше MAX).

```
  pr[j] = MAX;
}
```

Моделируем вычисление биномиального коэффициента. На самом деле раскладываем на множители все множители, стоящие в числителе и знаменателе дроби

$$\frac{n\cdot(n-1)\cdot(n-2)\cdot\ldots\cdot(n-k+1)}{1\cdot2\cdot3\cdot\ldots\cdot k}$$

```
void Cnk(int n, int k)
{
  int i, res = 1;
  memset(deg,0,sizeof(deg));
  for(i = 1; i <= k; i++)
    factor(n-i+1,1), factor(i,-1);
}
```

Основная часть программы. Запускаем решето Эратосфена. Генерируем простые числа.

```
gen_primes();

scanf("%d",&tests);
while(tests--)
{
  scanf("%d %d",&n,&k);
  if (k > n - k) k = n - k;
  if ((n == 1) || !k)
  {
    printf("1\n"); continue;
  }
```

Вычисляем значение C_n^k.

```
  Cnk(n,k);
```

Выводим ответ. Если deg[i] ≠ 0, то простой множитель i входит в разложение C_n^k со степенью deg[i].

```
  for(flag = i = 0; i < MAX; i++)
  if (deg[i])
  {
    if(flag) printf(" * ");
    printf("%d",i); flag = 1;
```

```
      if(deg[i] > 1) printf("^%d",deg[i]);
    }
    printf("\n");
  }
```

3260. Сколько?

Собираясь как-то на экзамен, Петя разложил перед собой n разных шпаргалок со своего “любимого” предмета “Математический анализ”. И, так как на протяжении семестра Петя не учился как следует, шпаргалок оказалось столько, что они все вместе не влазили ни в один карман. Тогда Петя посчитал максимальное количество шпаргалок, которое он может взять с собой на экзамен, и вдруг задумался, а сколько же всего существует способов выбрать нужное количество шпаргалок?

Вход. Содержит общее количество шпаргалок n ($1 \le n \le 12$) и количество шпаргалок k ($0 \le k \le n$), которые Петя может взять с собой.

Выход. Вывести количество способов выбрать k шпаргалок из n.

Пример входа 1
```
3 2
```

Пример выхода 1
```
3
```

Пример входа 2
```
4 1
```

Пример выхода 2
```
4
```

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*, 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).

Для рекурсивной реализации биномиального коэффициента можно воспользоваться соотношением:

$$C_n^k = \begin{cases} C_{n-1}^{k-1} + C_{n-1}^k, n > 0 \\ 1, k = n \text{ или } k = 0 \end{cases}$$

Реализация алгоритма

Вычисление биномиального коэффициента итеративным методом.

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

Основная часть программы. Читаем входные данные.

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

Вычисляем и выводим ответ.

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

Реализация алгоритма – рекурсия с запоминанием

```
#include <stdio.h>
#include <string.h>

int n, k, res;
int dp[35][35];

int Cnk(int n, int k)
{
  if (n == k) return 1;
  if (k == 0) return 1;
  if (dp[n][k] != -1) return dp[n][k];
  return dp[n][k] = Cnk(n - 1, k - 1) + Cnk(n - 1, k);
}

int main(void)
{
  scanf("%d %d",&n,&k);
  memset(dp,-1,sizeof(dp));
  res = Cnk(n,k);
  printf("%d\n",res);
  return 0;
}
```

4008. Биномиальные коэффициенты

Гуннар – достаточно старый и забывчивый исследователь. Сейчас он пишет статью о безопасности в социальных сетях, которая включает в себя некоторые элементы комбинаторики. Он написал программу для вычисления биномиальных коэффициентов, чтобы проверить некоторые расчеты.

Биномиальный коэффициент – это число $C_n^k = \frac{n!}{k!(n-k)!}$, где n и k – неотрицательные числа.

Гуннар использует свою программу для вычисления C_n^k и получил число m в качестве результата. К несчастью, поскольку он забывчивый, он забыл числа n и k, которые он использовал в качестве входных данных. Эти два числа были результатом долгих вычислений, и они были записаны на одном из многочисленных листков, лежащих на его столе. Вместо того чтобы поискать в бумагах, он решил восстановить числа n и k по полученному ответу. Можете ли Вы помочь ему найти все такие возможные значения?

Вход. Первая строка содержит количество тестов, не большее 100. Каждый тест задается в одной строке и содержит целое число m ($2 \le m \le 10^{15}$) – результат программы Гуннара

Выход. Для каждого теста вывести две строки. Первая строка должна содержать количество способов выразить m при помощи биномиального коэффициента. Во второй строке должны располагаться все пары (n, k), удовлетворяющие $C_n^k = m$. Пары следует расположить по возрастанию n, а в случае равенства по возрастанию k. Формат вывода пар приведен в примере.

Пример входа	**Пример выхода**
2 2 15	1 (2,1) 4 (6,2) (6,4) (15,1) (15,14)

Если $C_n^k = m$, то $C_n^{n-k} = m$. Достаточно найти решение $k \le n / 2$ и вместе с парой (k, n) вывести также пару (k, $n - k$). При $k = n / 2$ эти две пары совпадают.

Пусть p – наименьшее число, для которого $C_{2p}^p > m$. Тогда очевидно что $0 \le k < p$.

Зафиксируем такое k что $C_{2k}^k \le m$ и рассмотрим функцию $f(n) = C_n^k$. Тогда при $2k \le n \le m$ функция $f(n)$ является монотонно возрастающей. Следовательно можно решить уравнение $f(n) = m$ бинарным поиском.

Таким образом для решения задачи следует перебрать значения k ($0 \le k < p$) и для каждого такого k решить уравнение $C_n^k = m$ относительно n бинарным поиском. Найденное значение n должно быть целочисленным.

Пример

Рассмотрим уравнение $C_n^k = 3003$. Учитывая что $C_{12}^6 = 924$, а $C_{14}^7 = 3432$, то нам достаточно перебрать $0 \le k \le 6$.

Пусть $k = 2$, рассмотрим уравнение $C_n^2 = 3003$ или $\frac{n(n-1)}{2} = 3003$, $n * (n - 1) = 6006$. Бинарным поиском в промежутке $4 \le n \le 3003$ находим целочисленное решение $n = 78$. Учитывая что $n \ne 2*k$, имеем два решения: $C_{78}^2 = C_{78}^{76} = 3003$.

Реализация алгоритма

Искомые пары будем хранить в векторе пар *res*.

```
vector<pair<long long,long long> > res;
```

Вычисление биномиального коэффициента C_n^k.

```
long long Cnk(long long n, long long k)
{
  long long i, Res = 1;
  if (k > n - k) k = n - k;
  for(i = 1; i <= k; i++)
  {
```

Если на очередной итерации результат превосходит m (мы ищем решение уравнения $C_n^k = m$), то останавливаемся. Таким выходом из функции мы также избежим переполнения.

```
    if (1.0 * Res * (n - i + 1) / i > m) return m + 1;
    Res = Res * (n - i + 1) / i;
  }
  return Res;
}
```

Основная часть программы. Читаем входные данные.

```
scanf("%d",&tests);
while (tests--)
{
  res.clear();
  scanf("%lld",&m);
```

Перебираем значение k от 0 до тех пор пока $C_{2k}^k \le m$.

```
  for(k = 0; Cnk(2*k,k) <= m; k++)
  {
```

Ищем значение n ($2k \le n \le m$) бинарным поиском.

```
    long long lo = 2*k, hi = m;
    while (lo < hi)
    {
      long long n = (lo + hi) / 2;
      if (Cnk(n,k) < m) lo = n + 1; else hi = n;
    }
```

Если $C_{lo}^k = m$, то решение найдено. Заносим в результат одну или две пары решений.

```
    if (Cnk(lo,k) == m)
    {
      res.push_back(make_pair(lo,k));
```

```
      if (lo != 2*k)
        res.push_back(make_pair(lo,lo - k));
    }
  }
```

Сортируем пары.

```
  sort(res.begin(),res.end());
```

Выводим ответ – количество найденных пар и сами пары.

```
  printf("%d\n",res.size());
  for(i = 0; i < res.size(); i++)
    printf("(%lld,%lld) ", res[i].first,res[i].second);
  printf("\n");
}
```

5329. Вечерка

Сколькими способами среди n ЛКШат можно выбрать k таких, которым достанется кефир? Ответ выведите по модулю 9929.

Вход. Целые числа n и k ($0 \le k \le n \le 500$).

Выход. Выведите искомое количество способов по модулю 9929.

Пример входа	**Пример выхода**
6 3	20

Ответом к задаче будет значение $C_n^k \bmod 9929$. Поскольку следует найти биномиальный коэффициент по модулю, постараемся при вычислениях избежать деления. Для этого воспользуемся соотношением $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$, $C_n^0 = 1$.

Задачу можно решить также по формуле

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

заменив деление умножением на обратное число по модулю 9929. Отметим, что число 9929 простое. То есть $n / i = n * (i^{-1}) \bmod 9929$.

По теореме Ферма $i^{9928} \bmod 9929 = 1$ или $(i * i^{9927}) \bmod 9929 = 1$, откуда обратным к i по модулю 9929 будет $i^{9927} \bmod 9929$. Таким образом

$$(i^{-1}) \bmod 9929 = i^{9927} \bmod 9929$$

Реализация алгоритма

Воспользуемся динамикой с запоминанием. Значения $C_n^k \bmod 9929$ будем сохранять в ячейках cnk[k][n].

```
#define MAX 510
#define MOD 9929
int cnk[MAX][MAX];
```

Функция ***c*** вычисляет значение биномиального коэффициента $C_n^k \bmod 9929$.

```
int 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)) % MOD;
}+
```

Основная часть программы. Читаем входные данные. Вычисляем и выводим ответ.

```
memset(cnk,0,sizeof(cnk));
scanf("%d %d",&n,&k);
printf("%d\n",c(n,k));
```

Реализация алгоритма – циклическая

```
#include <stdio.h>
#include <string.h>
#define MAX 502
#define MOD 9929

int cnk[MAX][MAX];
int 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]) % MOD;
}

int main(void)
{
  FillCnk();
  scanf("%d %d",&n,&k);
  printf("%d\n",cnk[n][k]);
  return 0;
}
```

Реализация алгоритма – обратное по модулю

Функция ***pow*** вычисляет $x^n \bmod p$.

```
int pow(int x, int n, int p)
{
```

```
  if (n == 0) return 1;
  if (n % 2 == 0) return pow((x * x) % p, n / 2, p);
  return (x * pow(x, n - 1, p)) % p;
}
```

Функция ***inverse*** вычисляет число, обратное x, по модулю p: $x^{-1} \bmod p$ (p простое).

```
int inverse(int x, int p)
{
  return pow(x, p - 2, p);
}
```

Функция ***Cnk*** вычисляет значение биномиального коэффициента C_n^k по модулю p. Для этого выражение

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

перепишем в виде

$$res = (res * (n - i + 1) * (i^{-1} \bmod p)) \bmod p$$

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

Основная часть программы. Читаем входные данные.

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

Вычисляем и выводим ответ.

```
res = Cnk(n, k, 9929);
printf("%d\n", res);
```

7261. Трудный путь

Вася хорошо выпил и теперь, когда он добрался до своей улицы, он полностью потерял чувство направления. Поскольку он не помнит, с какой стороны его дом, он выбирает направление наобум. Более того, на каждом перекрёстке он с вероятностью 50% продолжает идти вперёд, а иначе разворачивается и идёт назад. Он настолько потерял связь с реальностью, что может даже пройти мимо своего дома и не заметить этого!

Пройдя n кварталов, Вася засыпает прямо на улице. Проснувшись, он задаётся вопросом: какой у него был шанс заснуть рядом с домом? Ведь от перекрёстка, от которого он начал свой путь, до перекрёстка рядом с домом Васи всего m кварталов. Помогите ему.

Вход. В одной строке содержатся два целых числа n и m ($1 \le n, m \le 1000$).

Выход. Выведите одно число – вероятность Васи заснуть на перекрёстке рядом со своим домом. Выведите ответ с абсолютной погрешностью не более 10^{-7}.

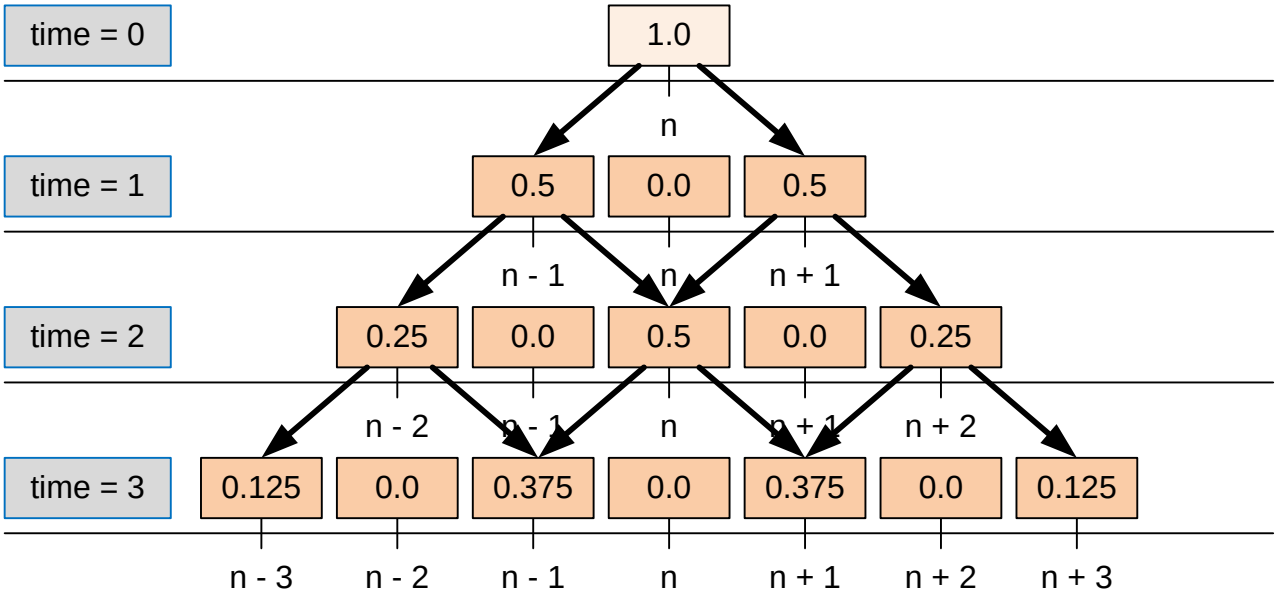
Пример входа 1
```
1 1
```

Пример выхода 1
```
0.5
```

Пример входа 2
```
1000 100
```

Пример выхода 2
```
0.000169397
```

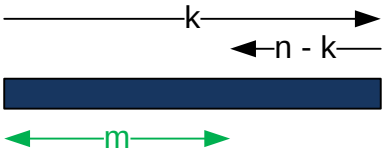
Объявим двумерный массив d, в котором d[*time*][*x*] равно вероятности оказаться в точке с абсциссой x в момент времени *time*. Пусть изначально (в момент времени $t = 0$) Вася находится в точке с абсциссой $x = n$. Тогда d[0][n] = 1.



Вычислим d[i][j] – вероятность того что Вася в момент времени i будет находиться в позиции j. Для этого Васе следует находиться в момент времени $i - 1$ либо в позиции $j - 1$, либо в позиции $j + 1$. Тогда в момент времени i он сможет попасть из них в позицию j с вероятностью 50%. То есть

$$d[i][j] = (d[i - 1][j - 1] + d[i - 1][j + 1]) / 2$$

Рассмотрим математическое решение задачи. Закодируем путь Васи последовательностью из 0 и 1. Пусть 1 соответствует движению вправо, а 0 влево. Пусть из n шагов, которые совершил Вася, k шагов он сделал вправо. Тогда $n - k$ шагов он сделал влево.



Нас интересует вероятность того, что Вася переместился в одну из сторон (например вправо) на m кварталов. Тогда должно выполняться: $m + n - k = k$,

откуда $k = (m + n) / 2$. Количество последовательностей длины n с k единицами равно C_n^k. Поскольку Вася совершил n перемещений, то у него имеется 2^n вариантов выбора различных путей. Следовательно вероятность того что Вася пройдет вправо m кварталов, равна $C_n^k / 2^n$, где $k = (m + n) / 2$. Отметим, что искомая вероятность равна 0, если $m + n$ нечетно. В этом случае Вася просто не сможет попасть домой ($m + n = 2k$, то есть четно).

Пример

Пусть Вася совершит $n = 3$ шага.

Если $m = 1$, то $k = (3 + 1) / 2 = 2$ и вероятность равна $C_3^2 / 2^3 = 3 / 8 = 0.375$.

Если $m = 3$, то $k = (3 + 3) / 2 = 3$ и вероятность равна $C_3^3 / 2^3 = 1 / 8 = 0.125$.

Пусть Вася совершит $n = 4$ шага.

Если $m = 0$, то $k = (4 + 0) / 2 = 2$ и вероятность равна $C_4^2 / 2^4 = 6 / 16 = 0.375$.

Если $m = 2$, то $k = (4 + 2) / 2 = 3$ и вероятность равна $C_4^3 / 2^4 = 4 / 16 = 0.25$.

Если $m = 4$, то $k = (4 + 4) / 2 = 4$ и вероятность равна $C_4^4 / 2^4 = 1 / 16 = 0.0625$.

Реализация алгоритма

Объявим двумерный массив для вычисления вероятностей.

```
#define MAX 2010
double d[MAX][MAX];
```

Читаем входные данные. Инициализируем массив d.

```
scanf("%d %d",&n,&m);
memset(d,0,sizeof(d));
d[0][n] = 1;
```

Пересчитываем вероятности согласно приведенного рекуррентного соотношения.

```
for(i = 1; i <= n; i++)
  for(j = n - i; j <= n + i; j++)
    d[i][j] = (d[i-1][j-1] + d[i-1][j+1]) / 2;
```

Выводим ответ – вероятность того что Вася пройдет в одну из сторон (например вправо) m кварталов и найдет свой дом.

```
printf("%.9lf\n",d[n][n+m]);
```

9892. C0n + … + Cnn

По заданному неотрицательному целому числу n найдите сумму биномиальных коэффициентов

$$C_n^0 + C_n^1 + \ldots + C_n^n$$

Вход. Одно неотрицательное целое число n ($n \le 60$).

Выход. Выведите значение суммы.

Пример входа	**Пример выхода**
2	4

Формула бинома Ньютона имеет вид:

$$(a+b)^n = \sum_{i=0}^{n} C_n^i a^i b^{n-i}$$

Если положить $a = b = 1$, то получим соотношение:

$$(1+1)^n = \sum_{i=0}^{n} C_n^i 1^i 1^{n-i}$$

или

$$2^n = \sum_{i=0}^{n} C_n^i = C_n^0 + C_n^1 + \ldots + C_n^n$$

Таким образом, указанная сумма равна 2^n.

Пример
При $n = 1$: $C_1^0 + C_1^1 = 1 + 1 = 2$;
При $n = 2$: $C_2^0 + C_2^1 + C_2^2 = 1 + 2 + 1 = 4$;
При $n = 3$: $C_3^0 + C_3^1 + C_3^2 + C_3^3 = 1 + 3 + 3 + 1 = 8$;

Реализация алгоритма
Читаем входное значение n.

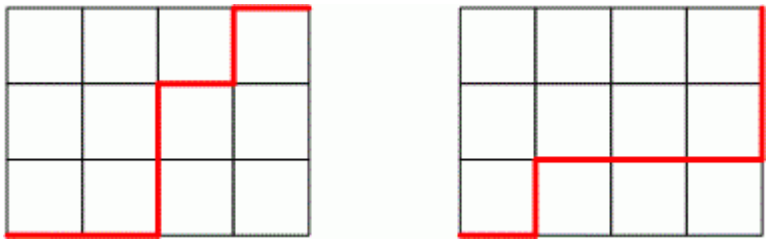
```
scanf("%lld", &n);
```

Вычисляем и выводим значение 2^n.

```
res = 1LL << n;
printf("%lld\n", res);
```

10668. Пути на сетке

У Вас есть лист бумаги, и Вы выбираете на нем прямоугольник размером $n * m$. Назовем этот прямоугольник вместе с линиями, в которых он находится, сеткой. Начиная с левого нижнего угла сетки, Вы перемещаете карандаш в правый верхний угол, следя за тем, чтобы он оставался на линиях и двигался только вправо или вверх. Результат показан слева:



Действительно шедевр, не правда ли? Повторяя процедуру еще раз, Вы получите картинку, показанную справа. Теперь Вы задаетесь вопросом: сколько разных произведений искусства можно таким образом создать?

Вход. Два натуральных числа n и m.

Выход. Выведите количество различных произведений искусства, которые можно создать, используя описанную выше процедуру. Вы можете с уверенностью предположить, что это число соответствует 64 – битовому знаковому целому числу.

Пример входа 1
```
3 4
```

Пример выхода 1
```
35
```

Пример входа 2
```
1 1
```

Пример выхода 2
```
2
```

Искомый путь – ломанная, состоящая из $n + m$ звеньев. n звеньев из них должны идти вертикально, остальные – горизонтально. Количество вариантов выбрать n вертикальных звеньев из $n + m$ равно C_{n+m}^{n}.

Пример

Для первого примера $n = 3$, $m = 4$. Ответ равен $C_7^3 = \frac{7!}{3!\,4!} = \frac{7*6*5}{1*2*3} = 35$.

Для второго примера $n = 1$, $m = 1$. Ответ равен $C_2^1 = \frac{2!}{1!1!} = 2$.

Реализация алгоритма

Функция ***Cnk*** вычисляет значение C_n^k.

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

Читаем входные значения n и m.

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

Вычисляем и выводим ответ.

```
res = Cnk(n + m, m);
printf("%lld\n", res);
```

11262. Ожидаемая минимальная степень

Вам даны два положительных целых числа n и x.

Вы хотите выбрать x различных целых чисел, каждое от 1 до n включительно. Выбор будет сделан равномерно случайным образом. То есть каждое из возможных x-элементных подмножеств целых чисел от 1 до n будет выбрано с равной вероятностью.

Пусть S будет наименьшим целым числом среди x выбранных. Вычислите ожидаемое значение 2^S. Другими словами, определите среднее значение 2 в степени S, где среднее значение берется по всем возможным выборам x различных целых чисел.

Вход. Два натуральных числа: n ($1 \le n \le 50$) и x ($1 \le x \le n$).

Выход. Выведите среднее значение 2 в степени S с 4 десятичными цифрами.

Пример входа 1	**Пример выхода 1**
4 4	2.000000

Пример входа 2	**Пример выхода 2**
3 2	2.666667

Выбрать x различных целых чисел из n можно C_n^x способами.

Рассмотрим, сколько существует подмножеств из x чисел, минимальным элементом в которых будет число i. В такое множество следует выбрать число i и еще $x - 1$ число, каждое из которых больше i. Чисел, больших i, имеется $n - i$. Выбрать из них $x - 1$ число можно C_{n-i}^{x-1} способами. Отметим также, что должно выполняться неравенство $i + x - 1 \le n$, чтобы в подмножестве из x элементов наименьшим было i. Откуда $i \le n - x + 1$.

Для вычисления ожидаемого значения 2^S следует вычислить выражение

$$\frac{\sum_{i=1}^{n} 2^i \cdot C_{n-i}^{x-1}}{C_n^x}$$

При $x - 1 > n - i$ считаем $C_{n-i}^{x-1} = 0$.

Пример

В первом тесте единственная возможная ситуация состоит в том, чтобы выбрать (1, 2, 3, 4). Минимальным является число 1, ожидаемое значение равно $2^1 = 2$.

Во втором тесте имеется три равновероятных сценария: выбрать можно или {1, 2} или {1, 3} или {2, 3}. Соответствующие значения S равны 1, 1 и 2 соответственно. Средним значением 2^S будет $(2^1 + 2^1 + 2^2) / 3 = 8 / 3 = 2.6666666$.

Вычислим указанный ответ по формуле:

$$\frac{\sum_{i=1}^{n} 2^i \cdot C_{n-i}^{x-1}}{C_n^x} = \frac{2^1 \cdot C_2^1 + 2^2 \cdot C_1^1}{C_3^2} = \frac{2 \cdot 2 + 4 \cdot 1}{3} = \frac{8}{3}$$

Реализация алгоритма

Функция ***Cnk*** вычисляет значение C_n^k. Значения биномиального коэффициента будем заносить в ячейки массива dp.

```
long long dp[51][51];

long long Cnk(int n, int k)
{
   if (n == k) return 1;
   if (k == 0) return 1;
   if (dp[n][k] != -1) return dp[n][k];
   return dp[n][k] = Cnk(n - 1, k - 1) + Cnk(n - 1, k);
}
```

Основная часть программы.

```
scanf("%d %d", &n, &x);
memset(dp, -1, sizeof(dp));
```

Вычисляем ответ: $a = \frac{\sum_{i=1}^{n} 2^i \cdot C_{n-i}^{x-1}}{C_n^x}$.

```
a = 0;
for (i = 1; i <= n - x + 1; i++) // x - 1 <= n - i
   a = a + Cnk(n - i, x - 1) * (1LL << i);
res = 1.0 * a / Cnk(n, x);
```

Выводим ответ.

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

11269. Лемурьи вечеринки – базовая

В подчинении у короля лемуров Джулиана есть ровно 2 * k лемуров по 2 лемура каждого из k видов. Джулиан обожает вечеринки, поэтому каждый вечер он устраивает тусовку, однако в VIP-зоне, к сожалению, хватает мест только для него и еще n других лемуров.

Поскольку Джулиан не любит устраивать “одинаковые” вечеринки, то ему каждый день приходится выбирать кого звать в VIP-зону, чтобы наборы лемуров из VIP-зоны никогда не повторялись. Два лемура одного вида считаются неразличимыми. Наборы считаются одинаковыми, если они совпадают как мультимножества видов лемуров.

Помогите Джулиану определить, сколько дней он сможет проводить различные вечеринки. Так как ответ может быть большим, выведите его по модулю 1000000007.

Вход. В одной строке даны два целых числа k и n ($1 \le k \le 500\,000$, $0 \le n \le 2 * k$) – количество видов лемуров и количество мест в VIP-зоне.

Выход. Выведите одно число – ответ на задачу по модулю 1000000007.

Пример входа 1	**Пример выхода 1**
3 3	7

Пример входа 2	**Пример выхода 2**
4 3	16

На n мест некоторые виды будут представлены двумя лемурами, а некоторые одним. Пусть выбранными окажутся i пар лемуров (i видов представлены двумя лемурами), этот выбор можно совершить C_k^i способами. После этого в VIP зоне останется $n - 2i$ мест. Такое количество лемуров можно выбрать из $k - i$ оставшихся видов (по одному лемуру из каждого из $k - i$ оставшихся видов). Такой выбор можно сделать C_{k-i}^{n-2i} способами. Поскольку i может принимать любое значение от 0 до k, получаем ответ

$$\sum_{i=0}^{k} C_k^i \cdot C_{k-i}^{n-2i}$$

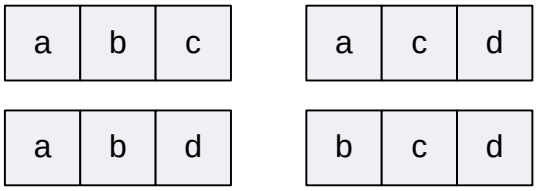
Пример

Рассмотрим второй тест, где имеется $k = 4$ пары лемуров и $n = 3$ места в VIP-зоне. По формуле получим:

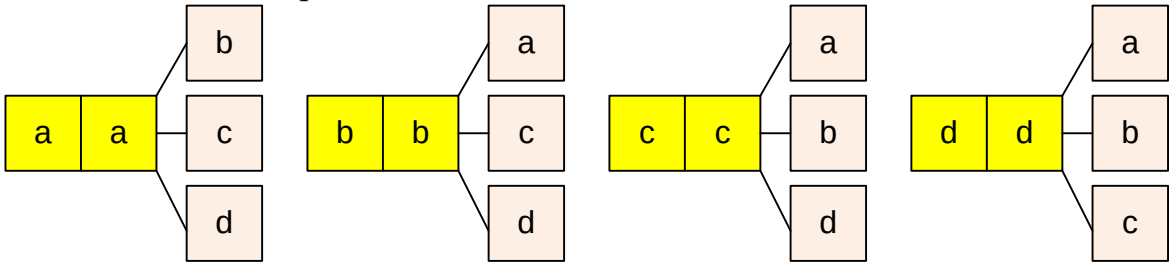
$$\sum_{i=0}^{4} C_4^i \cdot C_{4-i}^{3-2i} = C_4^0 \cdot C_4^3 + C_4^1 \cdot C_3^1 = 4 + 12 = 16$$

Выпишем только ненулевые слагаемые. C_n^k считаем равным нулю, если выполняется одно из трех неравенств: $k < 0$, $n < 0$ или $k > n$.

Обозначим лемуров буквами {*a*, *a*, *b*, *b*, *c*, *c*, *d*, *d*}. Одинаковым буквам соответствуют лемуры одного типа. Слагаемому $C_4^0 \cdot C_4^3 = 4$ соответствует факт, что никакие два лемура одного вида не являются выбранными, каждый из трех выбранных лемуров принадлежит разным видам. Возможными 4 выборками являются:



Рассмотрим слагаемое $C_4^1 \cdot C_3^1 = 12$. Из одного вида выбраны два лемура (4 варианта). На оставшееся одно место выбирается один лемур из трех оставшихся видов. Возможны 12 вариантов:



Очевидно, что два лемура из двух видов на 3 места взять невозможно. Поэтому остальные слагаемые суммы равны нулю.

Реализация алгоритма

Объявим константы.

```
#define MAX 1000001
#define MOD 1000000007
```

Объявим массивы: fact содержит факториалы чисел по модулю MOD, factinv содержит числа, обратные факториалам чисел по модулю MOD:

$$fact[n] = n! \bmod 1000000007$$

$$factinv[n] = n!^{-1} \bmod 1000000007$$

```
typedef long long ll;
ll fact[MAX], factinv[MAX];
```

Функция ***pow*** вычисляет степень числа по модулю: $x^n \bmod p$.

```
ll pow(ll x, ll n, ll p)
{
  if (n == 0) return 1;
  if (n % 2 == 0) return pow((x * x) % p, n / 2, p);
  return (x * pow(x, n - 1, p)) % p;
}
```

Функция ***inverse*** находит число, обратоне x по простому модулю p. Поскольку число p простое, то по теореме Ферма $x^{p-1} \bmod p = 1$ для всякого $1 \le x < p$. Равенство можно переписать в виде $(x * x^{p-2}) \bmod p = 1$, откуда обратным к x является число $x^{p-2} \bmod p$.

```
ll inverse(ll x, ll p)
{
  return pow(x, p - 2, p);
}
```

Функция ***Cnk*** вычисляет биномиальный коэффициент по формуле:

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

```
ll Cnk(int n, int k)
{
  return ((fact[n] * factinv[k]) % MOD * factinv[n - k]) % MOD;
}
```

Основная часть программы. Читаем входные данные.

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

Заполняем массивы факториалов fact и factinv.

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

factinv[0] = 1;
for (i = 1; i < MAX; i++)
   factinv[i] = inverse(fact[i], MOD);
```

Вычисляем ответ по формуле $\sum_{i=0}^{k} C_k^i \cdot C_{k-i}^{n-2i}$. Значение биномиального коэффициента C_n^k считаем равным нулю, если выполняется одно из трех неравенств: $k < 0$, $n < 0$ или $k > n$.

```
res = 0;
for (i = 0; i <= k; i++)
{
  if (n - 2 * i < 0 || k - i < 0 || n - 2 * i > k - i) continue;
  res = (res + Cnk(k, i) * Cnk(k - i, n - 2 * i)) % MOD;
}
```

Выводим ответ.

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

11270. Неисчислимые пути

Задан прямоугольник размером $n * m$. Прямоугольник размером $a * b$ отрезается от правого верхнего угла. Сколько способов имеется у муравья добраться из верхнего левого угла в нижний правый, если он может двигаться по линиям сетки только вправо или вниз.

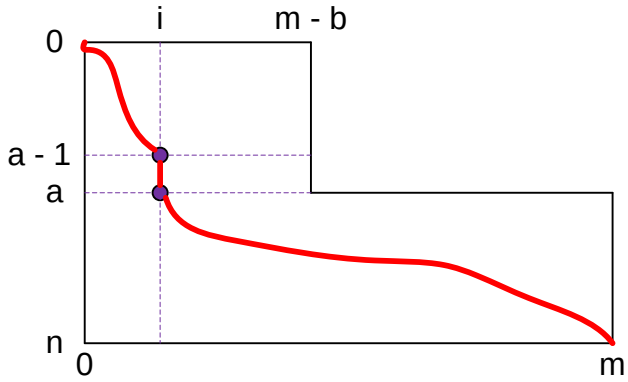
Вход. Первая строка содержит количество тестов t. Каждая из следующих t строк содержит четыре целых числа n, m, a, b ($2 \le n, m \le 400000$, $1 \le a < n$, $1 \le b < m$).

Выход. Для каждого теста выведите в отдельной строке количество способов, которыми муравей может добраться до нижнего правого угла при заданных условиях.

Пример входа	**Пример выхода**
3 2 2 1 1 4 5 2 2 7 7 6 6	5 105 50

Рассмотрим целый (без вырезов) прямоугольник размером $n * m$. Вычислим количество путей от клетки (0, 0) до клетки (n, m). Поскольку путь между указанными точками имеет длину $n + m$, и при этом содержит m горизонтальных сегментов, то число путей равно ways(n, m) = C_{n+m}^{m}. Если путь рассматривать как выбор n вертикальных сегментов из $n + m$, то число путей равно C_{n+m}^{n}. Однако заметим, что эти значения равны так как $C_{n+m}^{m} = C_{n+m}^{n+m-m} = C_{n+m}^{n}$.

Теперь рассмотрим прямоугольник размером $n * m$ с вырезанным правым верхом размером $a * b$.



Путь из (0, 0) в (n, m) разобъем на три части:

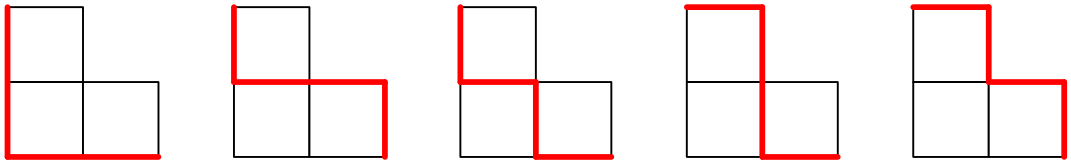
- двигаемся из (0, 0) в (a – 1, i), где $0 \le i \le m - b$;
- двигаемся из (a – 1, i) в (a, i);
- двигаемся из (a, i) в (n, m);

Количество путей из (0, 0) в (a – 1, i) равно ways(a – 1, i).
Количество путей из (a, i) в (n, m) равно ways(n – a, m – i).
Таким образом число путей из (0, 0) в (n, m) равно

$$\sum_{i=0}^{m-b} ways(a-1,i) \cdot ways(n-a,m-i)$$

Пример

Рассмотрим прямоугольник 2 * 2, у которого вырезан правый верхний угол размером 1 * 1. У муравья имеется 5 способов добраться из верхнего левого угла в нижний правый.



Реализация алгоритма

Объявим константы.

```
#define MAX 1000001
#define MOD 1000000007
```

Объявим массивы: fact содержит факториалы чисел по модулю MOD, factinv содержит числа, обратные факториалам чисел по модулю MOD:

$$fact[n] = n! \bmod 1000000007$$
$$factinv[n] = n!^{-1} \bmod 1000000007$$

```
typedef long long ll;
ll fact[MAX], factinv[MAX];
```

Функция ***pow*** вычисляет степень числа по модулю: $x^n \bmod p$.

```
ll pow(ll x, ll n, ll p)
{
  if (n == 0) return 1;
  if (n % 2 == 0) return pow((x * x) % p, n / 2, p);
  return (x * pow(x, n - 1, p)) % p;
}
```

Функция ***inverse*** находит число, обратоне x по простому модулю p. Поскольку число p простое, то по теореме Ферма $x^{p-1} \bmod p = 1$ для всякого $1 \le x < p$. Равенство можно переписать в виде $(x * x^{p-2}) \bmod p = 1$, откуда обратным к x является число $x^{p-2} \bmod p$.

```
ll inverse(ll x, ll p)
{
  return pow(x, p - 2, p);
}
```

Функция ***Cnk*** вычисляет биномиальный коэффициент по формуле:

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

```
ll Cnk(int n, int k)
{
  return ((fact[n] * factinv[k]) % MOD * factinv[n - k]) % MOD;
}
```

Функция ***ways*** вычисляет количество путей на решетке размером n * m от клетки (0, 0) до клетки (n, m). Поскольку путь между указанными точками имеет длину $n + m$, и при этом содержит m горизонтальных сегментов, то число путей равно C_{n+m}^m.

```
ll ways(int n, int m)
{
  return Cnk(n + m, m);
}
```

Основная часть программы. Заполняем массивы факториалов fact и factinv.

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

factinv[0] = 1;
for (i = 1; i < MAX; i++)
   factinv[i] = inverse(fact[i], MOD);
```

Читаем входные данные.

```
scanf("%d", &tests);
while (tests--)
{
  scanf("%d %d %d %d", &n, &m, &a, &b);
```

В переменной *res* вычисляем ответ по формуле.

```
  res = 0;
  for (i = 0; i <= m - b; i++)
    res = (res + ways(a - 1, i) * ways(n - a, m - i)) % MOD;
```

Выводим ответ.

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

11271. Сумма квадратов Cnk

По заданному неотрицательному целому числу n найдите сумму биномиальных коэффициентов

$$\left(C_n^0\right)^2+\left(C_n^1\right)^2+\left(C_n^2\right)^2+\ldots+\left(C_n^n\right)^2$$

Вход. Одно неотрицательное целое число n ($n \le 60$).

Выход. Выведите значение суммы.

Пример входа	**Пример выхода**
3	20

Рассмотрим $2n$ объектов a_i и разобьём 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.

Пример

Для $n = 3$ ответ равен $C_6^3 = \frac{6!}{3!\cdot 3!} = \frac{4\cdot 5\cdot 6}{6} = 20$;

Реализация алгоритма

Функция ***Cnk*** вычисляет значение биномиального коэффициента C_n^k.

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

Основная часть программы. Читаем входное значение n.

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

Вычисляем и выводим ответ.

```
res = Cnk(2*n, n);
printf("%lld\n", res);
```