Виток 2, Шаг 8 –

**Биномиальный коэффициент**

**Правило сложения.** Если элемент *a*  А можно выбрать *m* способами, а элемент *b*  B *n* способами, то выбор элемента *x*  A  B можно осуществить *m* + *n* способами. При этом множества А и В не должны иметь общих элементов (A  B = ).

**Правило умножения.** Если элемент *a*  А можно выбрать *m* способами, а элемент *b*  B *n* способами, то выбор пары (*a*, *b*)  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 π множества A может быть записана в виде подстановки, например:

,

где {*x*1, *x*2, *x*3, …, *xn*} = {*y*1, *y*2, *y*3, …, *yn*} = A и .

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

 = (1, 5, 2) (3, 6) ( 4)

Перестановка π называется ***инволюцией***, если . Каждая инволюция является произведением непересекающихся транспозиций. Например:

(5, 7, 4, 3, 1, 8, 2, 6) =  = (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

 =

Числа  называются ***биномиальными коэффициентами***, которые появляются в биноме Ньютона

(*x* + *y*)*n* = 

**Пример.** Вычислить  для заданного натурального *n*.

Из Бинома Ньютона следует, что  = (1 + 1) *n* = 2*n*.

**Пример.** Вычислить  –  +  – … +  для заданного натурального *n*.

Из Бинома Ньютона следует, что  = (1 – 1) *n* = 0.

**Пример.** Доказать, что  +  +  + … +  = .

Рассмотрим 2*n* объектов *ai* и разобъем их пополам: {*a*1, *a*2, …, *an*, *an*+1, …, *a*2*n*}. Для того чтобы выбрать *n* объектов из 2*n*, выберем *k* (*k* ≤ *n*) объектов из левой половины (то есть из множества {*a*1, *a*2, …, *an* }) и *n* – *k* объектов из правой (из множества {*an*+1, *a n*+2, …, *a*2*n* }). Количество способов совершить такую выборку по правилу умножения равно  = . Поскольку *k* может принимать значения от 0 до *n*, то  равно количеству способов выбрать *n* объектов из 2*n*, то есть .

Для реализации функции Cnk вычисления биномиального коэффициента воспользуемся соотношением:

 == 

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  после сокращения содержит *k* множителей).

Если значение *k* велико, но  помещается в int (например ), то согласно описанного алгоритма придется выполнять *k* итераций, что приведет к Time Limit. Поэтому при *k* > *n* – *k* следует воспользоваться соотношением  = .

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.  =  = 1,  = *n*;

2. Симметрия:  = ;

3. Внесение-вынесение:  = ;

4. Формула сложения:  =  + .

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 

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



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:



**Асимптотические оценки для Биномиального коэффициента.**

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

 <  < … <  ≥  > … > 

**Теорема 2.**  < 4*n*. Следует из того, что  = (1 + 1)2*n* = 22*n* = 4*n*.

**Теорема 3.**  > . Следует из того, что = 4*n*, а самих коэффициентов 2*n* + 1. Поэтому больший из коэффициентов будет больше .

**Теорема 4. Формула Стирлинга.** *n*! ≈ .

Более точное приближение имеет вид:

*n*! ≈ 

**Теорема 5.**   ≈  = .

**Теорема 6.**   == = 

=  ≤ (воспользуемся неравенством ln(1 – *x*) ≤ –*x* )

 = .

**Теорема 7.**  Учитывая, что ln(1 – *x*) = –*x* + o(*x*2), имеем:  = 

= .

**Следствие.** Если *k* = , то  ≈ .