Генерация k-элементных подмножеств

Сочетания (без повторений) из n по **k** – это подмножества, образованные **k** элементами из **n** возможных.

Количество сочетаний определяется по формуле

$C_n^k = n!/((n-k)!k!)$.

Задача 1. Для заданных n и k перечислить все k-элементные подмножества множества 0..n-1

Пример

Пусть n=5, k=3. Тогда имеем следующие подмножества ***M*** множества {0,1,2,3,4}:

0 1 2	0 2 3	1 2 3	2 3 4
0 1 3	0 2 4	1 2 4	
0 1 4	0 3 4	1 3 4	

Количество подмножеств $C_5^3 = 5!/(2!*3!) = 10$.

Пример. Дано **n** предметов, каждый из которых характеризуется весом $\mathbf{w_i}$. Необходимо выбрать **k** предметов так, чтобы суммарный вес этого набора не превышал **W**.

Алгоритм генерации лексикографическом порядке:

1. выберем наименьшее *в лексикографическом порядке подмножество из* ***k*** *элементов* $\{0, 1, \ldots, k-1\}$ из имеющегося множества $M= \{0, 1, \ldots, n-1\}$. Полученное подмножество будем хранить в $\{p_0, p_1, \ldots, p_{k-1}\}$;

В нашем примере для n=5 и k=3 имеем {***0,1,2***};

2. от конца к началу текущего подмножества $\{p_0, p_1, \ldots, p_{k-1}\}$ ищем первый элемент p_i который можно увеличить на ***1***.

- если такого элемента нет, мы получили последнее подмножество из ***k*** элементов {n-k, … , n-2, n-1}. В нашем примере {***2,3,4***} – ***генерация закончена***;
- иначе:

3. увеличиваем $p_i= p_i +1$. Всем последующим элементам $p_{i+1}, p_{i+2}, \ldots, p_{k-1}$ присваиваем значение большее на ***1*** чем предыдущий, $p_{i+1}.= p_i+1$;

4. выполняем пункт ***2***.

```
#include <iostream>
using namespace std;
typedef int vec[20];
vec a; int n,k;
void Print(vec p,int k)
{
 int i;
 for (i=0; i<k; i++)
 cout<<p[i]<<" ";
 cout<<endl;
}
void setnk(int n, int k) {
 int i,j,flag; vec p;
 for (i=0;i<k;i++) p[i]=i;
 Print(p,k);
 do
 {flag=0;
  for (i=k-1;i>=0;i--)
  if (p[i]<n-k+i) {
   p[i]=p[i]+1;
   for (j=i+1;j<k;j++)
   p[j]=p[j-1]+1;
   Print(p,k);
   falg=1;break;
  }
 } while(flag);
}
int main()
{
   cin>>n>>k;
   setnk(n,k);
   return 0;
}
```

Задача 2. По номеру L соответствующее сочетание (подмножество). Рассмотрим решение на примере:

M={0,1,2,3,4,5}, N=6 ; k=3; L=15.

Нумерация с нуля.

№	Текущее	K=|m| (m - множество используемых цифр)		L old	L new	Получено
1	?**	?=0, m={1, 2, 3, 4, 5} ?=1, m={2, 3, 4, 5}		15	15 – 10 = 5	1**
2	1?*	? = 2, m={3, 4, 5} ? = 3, m={4, 5}		5	5 – 5 = 0	14*
3	14?	?=5, M={}	0	0		145

0) **0**12	4) **0**23	8) **0**35	12) 1**2**5	16) 234
1) **0**13	5) **0**24	9) **0**45	13) 1**3**4	17) 235
2) **0**14	6) **0**25	10) **1**2**3**	14) 1**3**5	18) 245
3) **0**15	7) **0**34	11) 1**2**4	15) **145**	19) 345

Задача 3. По сочетанию (подмножеству) получить его номер.
Рассмотрим решение на примере:
Дано подмножество ***{1,4,5}*** множества **M=*{0,1,2,3,4,5}***, найти порядковый номер в подмножествах по k=**3** элемента.

1. Первая цифра 1, т.е. подмножества начинающиеся с 0, использованы. Их количество $C_5^2 = 10$;
2. Вторая цифра равна 4, т.е. подмножества начинающиеся с 2 и 3 использованы. Их количество $C_3^1 + C_2^1 = 5$;
3. Третья цифра равна 5 и она первая среди возможных цифр на 3 месте.

Порядковый номер подмножества будет C_5^2 + C_3^1+C_2^1 + 1 = 10 + 5 + 1 = 16, а т.к. нумерация начинается с 0, ответ **15**

0) **012**	4) **023**	8) **035**	12) 1**25**	16) 234
1) **013**	5) **024**	9) **045**	13) 1**34**	17) 235
2) **014**	6) **025**	10) 1**23**	14) 1**35**	18) 245
3) **015**	7) **034**	11) 1**24**	15) **145**	19) 345

Генерация всех подмножеств

Задача 1. Пусть $A=\{a_0,a_1,\ldots,a_{n-1}\}$ – множество целых чисел. Построить все его непустые подмножества.

Для построения всех подмножеств можно использовать функцию setnk(n,k):

```
For (k=1;k>=n;k++) setnk(n,k);
```

т.е. генерируем по одному, по два и т.д. до ***n*** элементов.

Битовый алгоритм. Поставим в соответствие каждому элементу множества 0 или 1. То есть каждому подмножеству соответствует *n*-значное число в двоичной системе счисления. Отсюда следует, что полный перебор всех подмножеств данного множества соответствует перебору всех чисел в двоичной системе счисления от 1 до $2^n - 1$:

$$\underbrace{00\ldots0}_{n-1}1 \text{ от 1 до } \underbrace{11\ldots1}_{n}.$$

Легко подсчитать и количество различных подмножеств данного множества, оно равно $2^n - 1$ (или 2^n, с учетом пустого множества).

Очевидно, что за 1 секунду мы можем сгенерировать множество из $n \le 20$ элементов.

```
#include <iostream>
using namespace std;
typedef int vec[18];
vec a; int n;
void subset(vec a, int n) {
   for (int mask=1;mask<1<<n;mask++){
   for(int i=0; i<n; i++)
   if((1<<i & mask)>0) cout<<a[i]<<' ';
cout<<endl;
 }
}
int main() {
   cin>>n;
   for(int i=0; i<n; i++) a[i]=i;
   subset(a, n);  // n – количество
элементов
  return 0;
}
```

Размещения с повторениями

Размещения с повторениями из **n** по **k** – это последовательности длины **k**, в которых **n** возможных элементов могут повторяться.

Пример. Для n=4 и k=3, имеем:

0,0,0	0,2,0	1,0,0	1,2,0	2,0,0	2,2,0	3,0,0	3,2,0
0,0,1	0,2,1	1,0,1	1,2,1	2,0,1	2,2,1	3,0,1	3,2,1
0,0,2	0,2,2	1,0,2	1,2,2	2,0,2	2,2,2	3,0,2	3,2,2
0,0,3	0,2,3	1,0,3	1,2,3	2,0,3	2,2,3	3,0,3	3,2,3
0,1,0	0,3,0	1,1,0	1,3,0	2,1,0	2,3,0	3,1,0	3,3,0
0,1,1	0,3,1	1,1,1	1,3,1	2,1,1	2,3,1	3,1,1	3,3,1
0,1,2	0,3,2	1,1,2	1,3,2	2,1,2	2,3,2	3,1,2	3,3,2
0,1,3	0,3,3	1,1,3	1,3,3	2,1,3	2,3,3	3,1,3	3,3,3

На каждой из **k** позиций в последовательности независимо от других может находиться любой из **n** элементов, поэтому по правилу произведения общее количество размещений – $\mathbf{n^k} = 4^3 = 64$.

Задача 1. Перечислить все последовательности длины ***k*** из чисел ***0..n-1***.

Алгоритм генерации размещения лексикографическом порядке

1. Первой будет последовательность <0,0,…,0>, печатаем. Будем хранить последнюю напечатанную последовательность в массиве ***x[0]..x[k-1]***;

2. С конца ищем элемент который можно увеличить. Так как последним является последовательность ***<n-1,n-1,…,n-1>***, мы можем увеличить найденный элемент только до ***n-1***:

   - если такого элемента нет, мы получили последнюю последовательность ***<n-1,n-1,…,n-1>***, печатаем и заканчиваем работу;

   иначе

   - увеличиваем его на ***1***, а за ним стоящие элементы приравниваем ***0***, печатаем;

3. повторяем пункт 2.

```
#include <iostream>
using namespace std;
typedef int vec[20];
vec a; int n,k;
void Print(vec p, int k) {
  int i;
  for (i=0; i<k; i++) cout<<p[i]<<" ";
  cout<<endl;
}
void razm(int n, int k) {
  int i,j,flag; vec p;
  for (i=0; i<k; i++) p[i]=0;
  Print(p,k);
  do  {
    flag=0;
    for (i=k-1; i>=0; i--)
     if (p[i]<n-1) {
       p[i]=p[i]+1;
       for (j=i+1;j<k;j++) p[j]=0;
       Print(p,k);  flag=1;  break;
     }
  } while (flag);
}
int main() {
  cin>>n>>k;
  razm(n,k);
  return 0;
}
```

Размещения без повторений

Задача 1. Сгенерировать все размещения для заданного множества {0, 1, 2,…, N-1} и все размещения по K элементов в лексикографичес-ком порядке.

Пример. N = 4, K = 3.

012	031	120	201	230	310
013	032	123	203	231	312
021	102	130	210	301	320
023	103	132	213	302	321

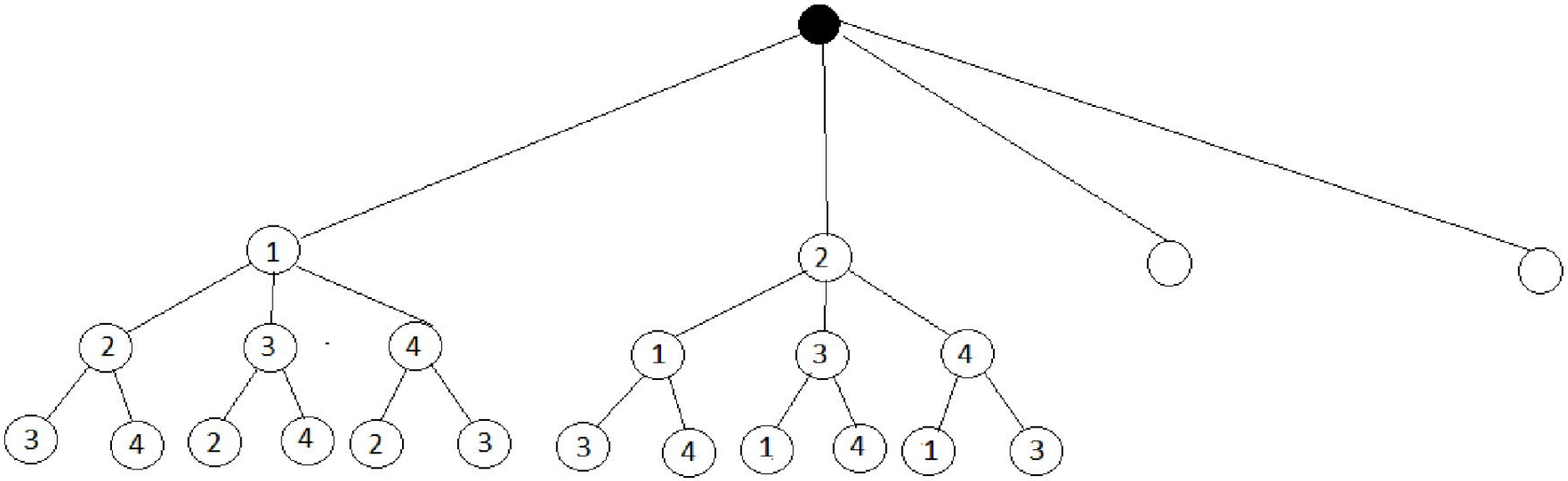
Количество размещений равна $A_N^K = \frac{N!}{(N-K)!}, A_4^3 = 24.$

Не рекурсивный алгоритм.

1. Строим наименьшее в лексикографическом порядке размещение 1,2,3,…,K. Из оставшихся элементов строим множество свободных элементов {K+1, K+2,…,N}. Обрабатываем полученное размещение.
2. Ищем в множестве свободных элементов наименьшее число большее последнего элемента temp = a[K].
3. Если такой элемент есть, заменяем последний элемент найденным числом. Это число удаляем из множества свободных элементов и добавляем в множество свободных элементов число temp. Переходим в начало пункта 2.

3. С конца ищем элемент a[i] < a[i+1].
4. Если такой элемент есть, в множество свободных элементов добавляем a[i+1]. В множестве свободных элементов ищем наименьшее число большее temp = a[i]. Заменяем a[i] найденным числом и добавляем в множество свободных элементов temp. В элементы a[i+1], a[i+2],…,a[K] записываем в возрастающем порядке меньшие элементы множества свободных элементов удаляя их из этого множества. Переходим в пункт 2.
5. Если не найдено, заканчиваем поиск следующего размещения.

Рекурсивный алгоритм



Рассмотрим п*рограмму* ***рекурсивного алгоритма:***

```
#include <iostream>
#include <set>
using namespace std;
int n, m, i, a[15];
set <int> s;
void Print() {
  int i;
  for(i=1;i<=m;i++) cout<<a[i]<<' ';   cout<<endl;
}
void Solve(int k){
int i;
  for(i=1;i<=n;i++)
  if (s.find(i)==s.end()) {
  s.insert(i); a[k]=i;
    if (k<m) Solve(k+1); else Print();
s.erase(i);
  }
}
```

```
void main(){
cin>>n>>m;
  Solve(1);
}
```