Виток 2, Шаг 9 –

**Перестановки, генерация подмножеств**

Пусть ищутся все перестановки чисел *x*1, …, *xn*. Поменяем местами числа *x*1 и *xi*, 1  *i*  *n*, а потом рекурсивно сгенерируем все перестановки чисел *x*2, …, , *xi*-1, *x*1, *xi*+1, … , *xn*. Такая процедура будет генерировать перестановки не в лексикографическом порядке.

void permut(int k)

{

 int i;

 if (k == n)

 {

 for(i = 1; i <= n; i++) printf("%d ",x[i]);

 printf("\n"); return;

 }

 for(i = k; i <= n; i++)

 {

 swap(x[i],x[k]);

 permut(k+1);

 swap(x[i],x[k]);

 }

}

Процедура swap(*i*, *j*) переставлят местами элементы *i* и *j*. Перед запуском процедуры генерации перестановок следует занести переставляемые элементы в массив х:

 scanf("%d",&n);

 for(i = 1; i <= n; i++) x[i] = i;

 permut(1);

Пусть f(*xi*, …, *xj*) генерирует все перестановки чисел от *xi* до *xj*. Тогда например

f(*x*1, *x*2, *x*3) = *x*1 f(*x*2, *x*3) + *x*2 f(*x*1, *x*3) + *x*3 f(*x*2, *x*1)

В свою очередь

f(*x*2, *x*3) = *x*2 f(*x*3) + *x*3 f(*x*2), f(*x*1, *x*3) = *x*1 f(*x*3) + *x*3 f(*x*1), f(*x*2, *x*1) = *x*2 f(*x*1) + *x*1 f(*x*2)

Так, например, все перестановки последовательности (1, 2, 3) будут сгенерированы в виде

(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 2, 1), (3, 1, 2)

Опишем другой алгоритм, в котором перестановки генерируются лексикографически. 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.**

|  |  |
| --- | --- |
| предыдущая перестановка | следующая перестановка |
| (4, 7, 1, 3, 5, 2, 6) | (4, 7, 1, 3, 5, 6, 2) |
| (4, 7, 1, 3, 5, 6, 2) | (4, 7, 1, 3, 6, 2, 5) |
| (4, 5, 3, 1, 7, 6, 2) | (4, 7, 3, 2, 1, 6, 7) |
| (3, 6, 4, 7, 5, 2, 1) | (3, 6, 5, 1, 2, 4, 7) |

Функция permut находит перестановку, следующую после той, которая задается в массиве *m*. Элементы, участвующие в перестановке, находятся в ячейках массива с индексами от 1 до *n*.

void permut(int n)

{

 int first, second;

 first = n - 1;

 while (x[first] > x[first+1]) first--;

 second = n;

 while (x[second] < x[first]) second--;

 swap(x[first],x[second]);

 reverse(&x[first+1],&x[n+1]);

}

Функция print выводит содержимое массива *m* на экран:

void print(void)

{

 for(int i = 1; i <= n; i++) printf("%d ",x[i]);

 printf("\n");

}

Пусть fact(*n*) – функция вычисления факториала числа *n*. Генерация перестановок в лексикографическом порядке совершается следующим образом: вводим количество переставляемых элементов *n*, заполняем массив *m* числами от 1 до *n* в возрастающем порядке, и запускаем цикл, состоящий из функций печати и генерации следующей за ней перестановки.

 scanf("%d",&n);

 for(i = 1; i <= n; i++) x[i] = i;

 for(i = 1; i <= fact(n); i++)

 {

 print();

 permut(n);

 }