УЧЕБНАЯ МАСТЕРСКАЯ

Костин Владимир Андреевич

ГЕНЕРАЦИЯ ПЕРЕСТАНОВОК

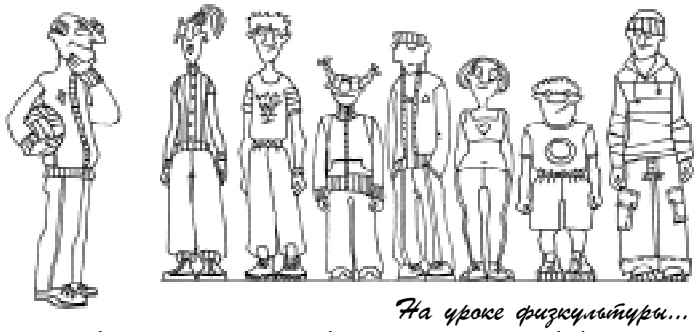
Меня всегда восхищало и восхищает умение Святослава Сергеевича кратко и ясно поставить ту или иную проблему, описать свойства тех или иных объектов.

В начале семидесятых годов, когда программистский мир обсуждал идеи структурного программирования, мне, приученному до этого программированию непосредственно в машинных командах, было не совсем ясно, как это можно обходиться без оператора GO TO. На мой вопрос по этому поводу Святослав Сергеевич дал следующий ответ. Нам нужен GO TO только в том случае, если в программе существует оператор, к которому нам необходимо перейти, по крайней мере, из трех различных мест программы. В каждое из этих мест мы можем попасть, вообще говоря, в зависимости от значения некоторого выражения, структуризация (выделение) которого явно повысит наглядность программы. Нетрудно видеть, что все это легко реализуется с помощью оператора выбора.

После такого объяснения идеи программирования без GO TO для меня стали абсолютно прозрачны.

Среди объектов, изучаемых в дискретной математике, одними из важнейших являются перестановки конечных множеств.

Пример. На уроке физкультуры преподаватель решил посмотреть различные варианты расстановки семи школьников в шеренгу. Каждый подобный вариант расстановки учащихся в математике называют перестановкой данного множества школьников.



На уроке физкультуры...
...варианты расстановки семи школьников в шеренгу.

Определение. Пусть задано некоторое конечное множество. Любое расположение элементов этого множества в определенном порядке будем называть *перестановкой* данного конечного множества. При этом считается, что каждый элемент выбирается только один раз. Число элементов, на котором задаются перестановки, обычно называют *порядком* перестановок.

Простейшая задача, которая может быть сформулирована на перестановках, заключается в следующем:

Пусть множество имеет n элементов, сколько существует различных перестановок этого множества?

Решим эту задачу для нашего примера. Прежде всего, заметим, что на первое место в перестановке можно

выбрать любого из семи школьников. После выбора школьника на первое место вторым мы можем выбрать любого из шести оставшихся учащихся. Таким образом, на первые два места мы можем выбрать школьников $7 \cdot 6 = 42$ различными способами. На первые три места в перестановке можно выбрать школьников $7 \cdot 6 \cdot 5 = 210$ различными способами, а на все семь мест – $7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 5040$ способами. То есть всего существует 5040 различных перестановок семи школьников.

Замечание. Предположим, что для перестроения школьников из одной перестановки в другую требуется 10 секунд. Тогда для перебора всех 5040 различных перестановок семи школьников потребуется 14 часов, что существенно превышает время одного урока физкультуры.

В книге В. Липского [1] приводится другой интересный пример порождения всех перестановок польскими монахами. В 1963 году в течение 17 часов 58 минут и 30 секунд они на восьми колоколах выбили $8! = 40320$ перестановок различных последовательностей звучания колоколов. Этот результат был занесен в книгу рекордов Гиннеса.

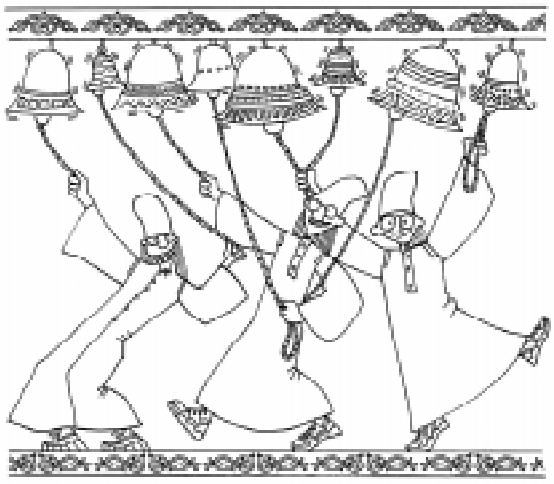
Упражнение. Докажите, что существует $1 \cdot 2 \cdot \ldots \cdot n = n!$ различных перестановок n-го порядка.

В данной статье мы не будем изучать общую теорию перестановок даже на элементарном уровне. Кого интересует эта тема, можно с ней познакомиться, например, по книге [2], в которой с ориентацией на школьников излагаются достаточно интересные свойства перестановок. Например, в ней с использованием теории перестановок описан алгоритм преобразования кубика Рубика из произвольного состояния в состояние, когда все его грани одного цвета. Нашей целью будет более узкая задача – рассмотреть алгоритмы порождения на компьютере всех перестановок n-го порядка. В качестве базового множества, на котором строятся перестановки, мы будем рассматривать множество чисел $\{1, 2, \ldots, n\}$. Такие алгоритмы генерации всех перестановок необходимы при решении задач переборного типа, в которых решение данной задачи представляет собой некоторую перестановку, обладающую конкретным заданным свойством. Для поиска искомой перестановки мы перебираем все возможные перестановки и проверяем для каждой из них выполнение этого конкретного свойства.

Например, предположим, мы разобрали кубик Рубика на отдельные части и собрали его случайным образом. После этого мы хотим проверить, можно ли его преобразовать таким образом, чтобы все его грани были одного цвета. Моделирование подобной задачи на компьютере явно потребует генерации перестановок.

Последовательное генерирование перестановок определяет на множестве всех перестановок некоторый порядок, а именно: пусть f и g перестановки, тогда $f < g$, если в этой генерации перестановка f появляется раньше перестановки g.

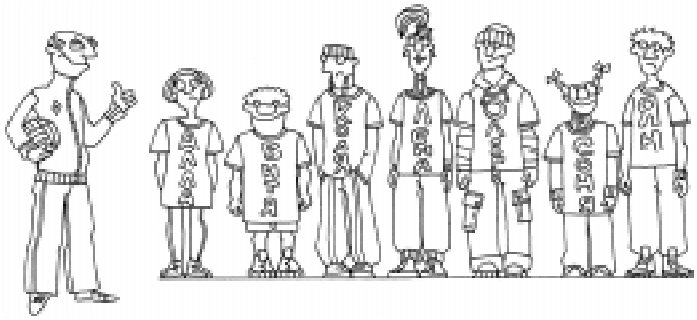
С другой стороны, при проектировании решения задачи обычно имеются вполне определенные аргументы в пользу выбора той или иной упорядоченности перестановок при генерации. Фактически эта упорядоченность определяет алгоритм генерации перестановок. Чаще всего эти алгорит-



... в течение 17 часов 58 минут и 30 секунд они на восьми колоколах выбили 8!=40320 перестановок различных последовательностей звучания колоколов.

мы строятся по схеме, когда каждая последующая перестановка вычисляется как некоторая функция от предыдущей.

Замечание. Интересен вопрос, какого порядка перестановки можно генерировать в разумное время на современных ЭВМ? Вследствие того, что общее число перестановок n-го порядка равно $n!$, современные ЭВМ позволяют генерировать перестановки не более чем 16-го порядка (попробуйте это обосновать!).



...только алгоритмы генерации всех перестановок в лексикографическом порядке.

Мы рассмотрим только алгоритмы генерации всех перестановок в лексикографическом порядке. Этот порядок свойствен расположению слов в различных словарях, поэтому его часто называют словарным. Он характеризуется тем, что буквы алфавита считаются упорядоченным множеством, а слова в словаре располагаются от меньших букв к большим. Слова, начинающиеся с одинаковых букв, располагаются в зависимости от упорядоченности вторых букв в слове, и так далее. Для перестановок множества $\{1, 2, ..., n\}$ числа считаются упорядоченными естественным образом. Формально можно дать следующее определение лексикографического порядка для перестановок.

Определение. Пусть $f = <a_1, ..., a_n>$, $g = <b_1, ..., b_n>$, будем говорить, что $f < g$ в лексикографическом порядке, если существует $k \geq 1$ такое, что $a_k \leq b_k$ и при $k > 1$ $a_q = b_q$ для $q < k$.

Пример. При $n = 4$ в лексикографическом порядке перестановки располагаются так, как показано на рисунке 1.

...предположим, мы разобрали кубик Рубика

1. <1, 2, 3, 4>	7. <2, 1, 3, 4>	13. <3, 1, 2, 4>	19. <4, 1, 2, 3>
2. <1, 2, 4, 3>	8. <2, 1, 4, 3>	14. <3, 1, 4, 2>	20. <4, 1, 3, 2>
3. <1, 3, 2, 4>	9. <2, 3, 1, 4>	15. <3, 2, 1, 4>	21. <4, 2, 1, 3>
4. <1, 3, 4, 2>	10. <2, 3, 2, 4>	16. <3, 2, 4, 1>	22. <4, 2, 3, 1>
5. <1, 4, 2, 3>	11. <2, 4, 1, 3>	17. <3, 4, 1, 2>	23. <4, 3, 1, 2>
6. <1, 4, 3, 2>	12. <2, 4, 3, 1>	18. <3, 4, 2, 1>	24. <4, 3, 2, 1>

Рисунок 1.

Лексикографический порядок может быть интерпретирован так. Пусть каждая перестановка рассматривается как целое число, записанное в n-ичной позиционной системе (с цифрами '0' $\leftrightarrow$ 1, ..., 'n – 1' $\leftrightarrow n$). Тогда генерация их в лексикографическом порядке – это перечисление в порядке возрастания чисел, состоящих из n разных цифр.

Наша цель заключается в построении алгоритма генерации всех перестановок в лексикографическом порядке для произвольного n. Для этого мы выясним, как должна выглядеть следующая перестановка, если мы знаем текущую, при условии, что текущая перестановка не является последней в генерации. Рассмотрим подробнее свойства лексикографического порядка генерации перестановок:

Л1. В первой перестановке элементы располагаются в возрастающей последовательности, в последней – в убывающей (докажите это свойство для произвольного n).

Л2. Последовательность всех перестановок можно разбить на n блоков длины $(n – 1)!$, соответствующих возрастающим значениям элемента в первой позиции. Остальные $n – 1$ позиций блока, содержащего элемент p в первой позиции, определяют

последовательность перестановок множества $\{1, ..., n\}/\{p\}$ в лексикографическом порядке.

Это свойство легко иллюстрируется приведенным примером генерации перестановок 4-го порядка. Нетрудно видеть, что все перестановки 4-го порядка разбиты на четыре столбца, при этом у перестановок первого столбца на 1-ой позиции расположен элемент 1, у второго – элемент 2, и так далее. Кроме того, в каждом столбце элементы, расположенные в перестановках со 2-ой по 4-ую позиции, образуют перестановки этих элементов в лексикографическом порядке. Для первого столбца перестановки элементов 2, 3, 4; для второго – 1, 3, 4; для третьего – 1, 2, 4; для четвертого – 1, 2, 3. Отметим также, что в каждом столбце элементы, расположенные со 2-ой по 4-ую позиции в первой перестановке, образуют возрастающую последовательность, а в последней перестановке эти же элементы расположены в убывающей последовательности (свойство Л1 лексикографического порядка).

Остальные n – 1 позиций блока ... определяют последовательность перестановок множества ... в лексикографическом порядке.

Таким образом, если мы рассмотрим перестановки каждого столбца, то для элементов, расположенных со 2-ой по 4-ую позиции, полностью выполняются свойства Л1 и Л2. Это замечание приводит к следующему обобщению свойства Л2 для перестановок произвольного порядка:

Л3. Последовательность всех перестановок можно разбить на $n*(n-1)*...*(n-k+1)$ блоков выбором значений $p_1, ..., p_k$ элементов, расположенных на первых k позициях. При этом блок $p_1, ..., p_k$ предшествует блоку $q_1, ..., q_k$, если $p_1, ..., p_k$ меньше $q_1, ..., q_k$ в лексикографическом порядке. Кроме того, для перестановок каждого такого обобщенного блока элементы, расположенные с $k+1$-ой по n-ую позиции, представляют собой генерацию перестановок этих элементов в лексикографическом порядке.

Теперь мы готовы сформулировать самое важное свойство лексикографического порядка, на основе которого легко преобразовать текущую перестановку в следующую. Это свойство можно записать так:

Л4. Любая текущая перестановка является заключительной для некоторого обобщенного блока. Этот блок определяется элементами текущей перестановки, расположенными на позициях в конце перестановки и представляющими собой максимально возможную убывающую последовательность значений элементов перестановки. Справедливость последнего замечания следует из свойства Л1 лексикографического порядка.

В дальнейшем максимально возможную убывающую последовательность значений элементов перестановки, расположенную на позициях в конце перестановки, будем называть «хвостом» перестановки.

Примеры. Рассмотрим приведенную выше генерацию перестановок 4-го порядка, тогда:

– перестановка <2,1,4,3> является заключительной для блока, состоящего из перестановок 7. <2, 1, 3, 4> и 8. <2, 1, 4, 3>, элементы 4,3 образуют ее хвост;

– перестановка <3, 1, 2, 4> является заключительной для блока, состоящего из одной перестановки 13. <3, 1, 2, 4>, ее хвост состоит из одного элемента – 4;

– перестановка <2, 4, 3, 1> является заключительной для второго столбца приведенной генерации, ее хвост – 4, 3, 1;

– перестановка <4, 3, 2, 1> является заключительной для всей генерации перестановок 4-го порядка, ее хвост совпадает со всей перестановкой.

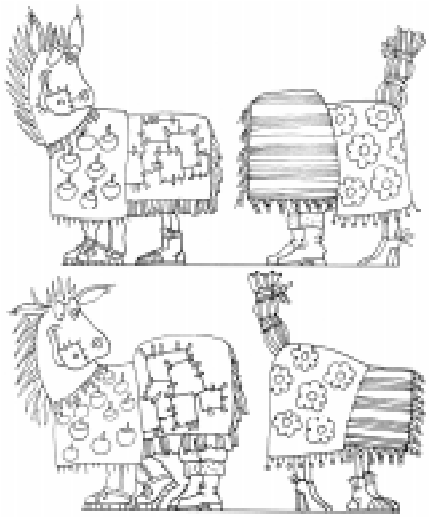
Как же воспользоваться этим свойством, чтобы преобразовать текущую перестановку в следующую. Это можно сделать по следующему алгоритму:

1. Выделяем хвост текущей перестановки.

2. Если он не совпадает со всей перестановкой, то ищем в хвосте первый с конца перестановки элемент, больший элемента перестановки, расположенного непосредственно перед ее хвостом (если перестановка совпадает со своим хвостом, то она является заключительной во всей генерации).

3. Меняем местами элемент, найденный в предыдущем пункте, с элементом, расположенным непосредственно перед хвостом перестановки (перемену местами двух элементов перестановки в теории перестановок обычно называют *транспозицией* этих элементов).

4. Располагаем все элементы преобразованного в пункте 3 хвоста перестановки в обратном порядке (инвертирование преобразованного хвоста перестановки).



Выделяем хвост текущей перестановки.

Примеры. Перестановка <2, 1, 4, 3> преобразуется по вышеприведенному алгоритму в перестановку <2, 3, 1, 4>, а перестановка <3, 1, 2, 4> в <3, 1, 4, 2>. Рассмотрим перестановку 15-порядка <15, 2, 4, 3, 1, 13, 7, 10, 14, 12, 11, 9, 8, 6, 5>, она преобразуется в перестановку <15, 2, 4, 3, 1, 13, 7, 11, 5, 6, 8, 9, 10, 12, 14>.

Упражнение. В какие перестановки преобразуются следующие перестановки:

а) $n = 3$, <2,3,1>;

б) $n = 5$, <2,5,4,3,1>;

в) $n = 7$, <4,5,2,3,1,6,7>;

г) $n = 8$, <2,4,3,6,8,7,5,1>.

Замечание 1. Можно провести формальное доказательство того факта, что преобразованная по данному алгоритму перестановка действительно совпадает со следующей после текущей перестановки в лексикографическом порядке. Однако мы такое доказательство опустим.

Замечание 2. Приведенный выше алгоритм преобразования текущей перестановки в следующую формально можно записать так:

Пусть $p = \langle p_1, ..., p_k, ..., p_j, ..., p_n \rangle$, где $1 \le k < n$, $p_k < p_{k+1}$, и для q таких, что $k < q < n$, следует $p_q > p_{q+1}$

(если $p = \langle n, n-1, ..., 1 \rangle$, то $k = 0$),

$j > k$ и $p_j > p_k$ и для $q : j < q \le n$ следует $p_q < p_k$;

тогда следующая за p перестановка при $k \ne 0$ имеет вид

$\langle p_1, ..., p_{k-1}, p_j, p_n, p_{n-1}, ..., p_{j+1}, p_k, p_{j-1}, ..., p_{k+1} \rangle$.

Упражнение. Постройте генерацию всех перестановок 3-го порядка в лексикографическом порядке.

Теперь мы легко можем написать на языке Паскаль алгоритм генерации всех перестановок в лексикографическом порядке. Он основан на поиске k и j в текущей перестановке, транспозиции элементов p_k и p_j, и инвертировании ее «хвоста» (см. листинг 1).

Комментарий. Нулевой элемент включен в массив p для того, чтобы обеспечить конец цикла {поиск k} после генерации последней перестановки.

Упражнение. Протестируйте приведенную выше программу при $n = 3$.

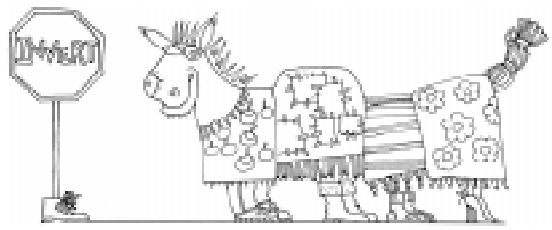
Замечание. Для читателей, знакомых с программированием рекурсивных программ на языке Паскаль, не нужно было выяснять, как выглядит следующая перестановка после текущей в лексикографическом порядке. Они могли бы построить соответствующую рекурсивную программу непосредственно на основе свойств Л1–Л3. Эта рекурсивная программа может быть написана так, как показано в листинге 2.

Комментарий. Процедура INVERT служит для восстановления первоначальной перестановки (свойство Л1) после генерации всех перестановок данного обобщенного блока. Процедура LEC осуществляет либо печать перестановки (строка {1}), если все n позиций уже сформированы, либо (по свойству Л2) генерирует перестановки $n - k + 1$ порядка как последовательность $n - k + 1$ блоков перестановок $n - k$ порядка с возрастающим по значению элементом на k позиции.

Тест $n = 3$ (см. рисунок 2).

Упражнение. Проведите формальное доказательство правильности алгоритма процедуры LEC.

Указание. Методом математической индукции докажите, что если $p[k] < ... < p[n]$, то вызов LEC(k) приводит к генерированию всех перестановок множества $p[k]$, ..., $p[n]$ в лексикографическом порядке при неизменных значениях $p[1]$, ..., $p[k - 1]$.



Процедура INVERT служит для восстановления первоначальной перестановки...

Замечание. Для сравнения различных подходов к решению одной и той же задачи в программировании введено понятие вычислительной сложности программы. Обычно временная вычислительная сложность программ, представленных на языке высокого уровня, оценивается как порядок роста числа исполняемых операторов программы в зависимости от некоторого параметра исходных данных [3]. Однако в алгоритмах генерации перестановок такой подход малоэффективен, так как в процессе работы любого алгоритма генерации всех перестановок порождается $n!$ перестановок, то есть временная вычислительная сложность всегда будет, по крайней мере, $O(n!)$ – величина слишком быстро растущая. Любая «экономия» в реализации будет сказываться только на коэффициенте пропорциональности при $n!$. Поэтому, для того чтобы удобнее было сравнивать различные алгоритмы генерации перестановок, обычно вводят другие критерии оценки вычислительной слож-

{2}	$k = 1$	$i = 3$	
{2}	$k = 2$	$i = 3$	
{1}	$k = 3$		вывод <1 2 3>
{3}	$k = 2$	$i = 3$	$p = 1\ 3\ 2$
{4}	$k = 2$		$p = 1\ 3\ 2$
{2}	$k = 2$	$i = 2$	
{1}	$k = 3$		вывод <1 3 2
{3}	$k = 1$	$i = 3$	$p = 2\ 3\ 1$
{4}	$k = 1$		$p = 2\ 1\ 3$
{2}	$k = 2$	$i = 2$	
{2}	$k = 2$	$i = 3$	
{1}	$k = 3$		вывод<2 1 3>
{3}	$k = 2$	$i = 3$	$p = 2\ 3\ 1$
{4}	$k = 2$		$p = 2\ 3\ 1$
{2}	$k = 2$	$i = 2$	
{1}	$k = 3$		вывод <2 3 1>
{3}	$k = 1$	$i = 2$	$p = 3\ 2\ 1$
{4}	$k = 1$		$p = 3\ 1\ 2$
{2}	$k = 1$	$i = 3$	
{3}	$k = 2$	$i = 3$	
{1}	$k = 3$	$i = 3$	вывод <3 1 2>
{3}	$k = 2$	$i = 3$	$p = 3\ 2\ 1$
{4}			$p = 3\ 2\ 1$
{2}	$k = 2$	$i = 2$	
{3}	$k = 3$		вывод <3 2 1>

Рисунок 2. Тест $n = 3$.

Листинг 1.

```
program LEX;
 const  n=...;       {порядок перестановок}
 var p : array [0..n] of 0..n;  {текущая перестановка}
        k : 0..n; j,r,m : 1..n;
 begin
   for k:=0 to n do p[k]:=k;    {задание начальной перестановки}
   k:=1;
   while k<> 0 do
     begin
      for k:=1 to n do write(p[k]); writeln;   {вывод перестановки}
       k:=n-1; while p[k]>p[k+1] do k:=k-1;    {поиск k}
       j:=n;  while p[k]>p[j] do j:=j-1;       {поиск j}
       r:=p[k]; p[k]:=p[j]; p[j]:=r;           {транспозиция p_k и p_j}
       j:=n; m:= k+1;
       while  j>m do             {инвертирование хвоста перестановки}
        begin r:=p[j]; p[j]:=p[m]; p[m]:=r; j:=j-1; m:=m+1 end
     end
 end.
```

Листинг 2.

```
program LEX1 (output);
   const n=...;   {n порядок перестановок}
   var p : array [1..n] of 1..n;
       i,r : 1..n;
   procedure INVERT (m : integer); {инвертирование p[m]...p[n] }
       var i,j: 1..n;
       begin i:=m; j:=n;
          while i<j do begin r:=p[i]; p[i]:=p[j]; p[j]:=r;
                             i:=i+1; j:=j-1
                       end
       end {INVERT};
 procedure Lec (k:integer);
   var i : 1..n;
   begin
      if  k=n then
  {1}      begin for  i:=1 to n do write (p[i]); writeln end
        else
           for  i:=n downto k do
  {2}        begin
               LEC (k+1);
               if i>k then
                      begin
                         r:=p[i]; p[i]:=p[k]; p[k]:=r;
  {3}                   INVERT (k+1)
  {4}                 end
             end
  end {LEC};
 begin
    for i:=1 to n do p[i]:=i;
    LEC (1)
 end.
```

1. <1, 2, 3, 4>	7. <1, 2, 4, 3>	13. <1, 3, 4, 2>	19. <2, 3, 4, 1>
2. <2, 1, 3, 4>	8. <2, 1, 4, 3>	14. <3, 1, 4, 2>	20. <3, 2, 4, 1>
3. <1, 3, 2, 4>	9. <1, 4, 2, 3>	15. <1, 4, 3, 2>	21. <2, 4, 3, 1>
4. <3, 1, 2, 4>	10. <4, 1, 2, 3>	16. <4, 1, 3, 2>	22. <4, 2, 3, 1>
5. <2, 3, 1, 4>	11. <2, 4, 1, 3>	17. <3, 4, 1, 2>	23. <3, 4, 2, 1>
6. <3, 2, 1, 4>	12. <4, 2, 1, 3>	18. <4, 3, 1, 2>	24. <4, 3, 2 ,1>

Рисунок 3.

ности. Здесь разумно ввести два критерия – количество транспозиций элементов перестановки, выполняемых в среднем при генерации одной перестановки, и аналогичное среднее число вызовов процедуры LEC как функции от n-порядка перестановок. С оценками вычислительной сложности генерации перестановок в лексикографическом порядке можно познакомиться в [4].

Наряду с лексикографическим порядком, достаточно часто встречается генерирование перестановок в антилексикографическом порядке.

Определение. Пусть $f = <a_1, ..., a_n>$, $g = <b_1, ..., b_n>$, будем говорить, что $f < g$ в антилексикографическом порядке, если существует $k \le n$ такое, что $a_k > b_k$ и $a_q = b_q$ для $q > k$.

Пример. При $n = 4$ в антилексикографическом порядке перестановки располагаются так, как показано на рисунке 3.

Упражнение.

1. Сформулируйте свойства А1–А3 антилексикографического порядка, аналогичные свойствам Л1–Л3 для лексикографического порядка.

2. Определите соотношения между некоторой перестановкой и непосредственно следующей за ней в антилексикографическом порядке.

3. Постройте нерекурсивный алгоритм ANTYLEX, порождающий все перестановки в антилексикографическом порядке (сравните с [1]).

4. Постройте рекурсивный алгоритм ANTYLEX1, порождающий все перестановки в антилексикографическом порядке.

Для читателей, интересующихся другими подходами к генерации перестановок, а также генерациями различных теоретико-множественных объектов, можно порекомендовать монографию [4], которую отличает удачное сочетание компактности изложения с весьма подробной разработкой темы.

Литература.

1. *Липский В.* Комбинаторика для программистов. Пер. с польск. М.: Мир, 1988.

2. *Калужнин Л.А.*, *Сущанский В.И.* Преобразование и перестановки. Пер. с украинск. М.: Наука, 1985.

3. *Ахо А.*, *Хопкрофт Дж.*, *Ульман Дж.* Построение и анализ вычислительных алгоритмов. Пер. с англ. М.: Мир, 1979.

4. *Рейнгольд Э.*, *Нивергельт Ю.*, *Део Н.* Комбинаторные алгоритмы теория и практика. Пер. с англ. М.: Мир, 1980.

Костин Владимир Андреевич, доцент кафедры информатики математико-механического факультета СПБГУ.