РАЗДЕЛ III.
ТЕХНОЛОГИИ И МЕТОДИКИ ОБУЧЕНИЯ

УДК 519.16

Л.И. Фёдоров
Государственный университет управления

ГЕНЕРАТОР ПЕРЕСТАНОВОК ТРАНСПОЗИЦИЕЙ СОСЕДНИХ ЭЛЕМЕНТОВ В MATHCAD

Аннотация. В статье рассматривается алгоритм генерации перестановок и приводится текст программы, работающей в среде MATHCAD, приводятся примеры использования данного алгоритма в некоторых учебных задачах.
Ключевые слова: перестановка, мобильность перестановки, направление перестановки, транспозиция, алгоритм, программа.

L. Fedorov
State University of Management

GENERATOR OF PERMUTATIONS BY TRANSPOSITION OF NEIGHBORING ELEMENTS IN MATHCAD

Abstract. The article explains the algorithm for generating permutations, and the text of the program that is running in an environment MATHCAD, there are examples of the use of this algorithm in some learning tasks.
Keywords: Permutation, mobility changes the direction of the permutation, transposition, algorithm, program mobility permutation, the direction of the permutation

Опыт использования MathCad в учебных целях при параллельном изучении разделов курса математики и информатики отражен в литературе [1] и безусловно заслуживает внимания. Цель данной работы дать по возможности доступное для учащихся и преподавателей информатики описание данного алгоритма генерирования перестановок транспозицией только соседних элементов и реализация этого алгоритма в среде MathCad. Это позволит с успехом применять его при изучении разделов дискретной математики и линейной алгебры, где он может быть использован при переборе вариантов. Алгоритм с транспозицией соседних элементов имеет реализации в C, Python, Java и др. Хорошее

описание алгоритма и реализация (без представления исходного кода) дается в [2].. Реализация данного алгоритма в среде MathCad является частью указанной цели , так как программный код в этой среде хорошо согласуется с используемым в литературе псевдокодом для описания алгоритмов [3,4].

Об алгоритме: Дано целое положительное число n, этот алгоритм генерирует список перестановок {1, ..., n} , то есть , получив на входе n = 3, этот алгоритм выдаст:

$$\mathbf{Permutation}(3) = \begin{matrix} 1 & 1 & 3 & 3 & 2 & 2 \\ 2 & 3 & 1 & 2 & 3 & 1 \\ 3 & 2 & 2 & 1 & 1 & 3 \end{matrix}$$

Авторы [5] при описании алгоритма использовали термины направление и мобильность чисел в перестановке**.**

Направление

Вначале числа располагаются в порядке возрастания, и каждому из элементов изначально не присваивается направлении влево(Left) . Символ <= перед каждым номером ниже указывает на направление, связанное с этим числом

<=[1] <=[2] <=[3]

Само число и символ справа или слева от числа будет обозначать, что его направление Right или соответстенно Left В приведенном ниже примере, направление номер 3 является Right.

[3]=> <=[2] <=[1]

Мобильность

Этот алгоритм использует термин мобильность целого числа. Целое число называется мобильным, если к числу на его направлении примыкает число меньшее, чем это число. Если целое находится на правой колонке и имеет направление вправо, то это не мобильное число. Если целое находится на крайней левой колонке, и имеет направление влево, это не мобильное.

Алгоритм

I. Разместить цифры из диапазона 1 ... N в порядке возрастания и присвоить каждому направление влево Left .

II. Найти наибольшее мобильное целое и поменять его с соседним элементом на его направлении без изменения направления любого из этих двух.

III. При этом, если наибольшее мобильное целое достигло места, где это число уже не мобильно, то приступить к поиску следующего наибольшего мобильного целого.

IV. После каждой транспозиции проверить, есть ли число, большее, чем текущее мобильное целое. Если есть одно или несколько тако-

вых, то изменить направление у всех из них. (смотрите пример для n=4, приводимый ниже).

Алгоритм останавливается, когда нет более мобильных целых чисел.

Алгоритм в действии

<=[1]	**<=[2]**	**<=[3]**
<=[1]	**<=[2]**	**<=[2]**
<=[3]	**<=[1]**	**<=[2]**

Теперь 3 больше не мобильно, так как оно находится в крайней левой колонке и имеет направление влево. Поэтому следует переходить к следующему наибольшему мобильному числу, которым является 2.

[3]=>	**<=[2]**	**<=[1]**

Теперь, число [3] изменило свое направление в связи с этапом (IV) алгоритма . Когда **<=[2]** местами с **<=[1]**, алгоритм проверяет (не только среди мобильных), есть ли номер больше, чем [2] Поэтому <=[3] направление изменилось на [3]= >

Алгоритм с этого момента продолжается с [3]= >, так как оно снова стать наибольшим мобильным целым.

<=[2]	**[3]=>**	**<=[1]**
<=[2]	**[1]=>**	**[3]=>**

Алгоритм завершен, так как ни одно из чисел больше не мобильно. Причины, почему они не являются мобильными следующие**:**

<=[2] больше не мобильно, так как оно в самом левом столбце и имеет направление влево.

[1]=> больше не мобильно, так как нет номера на его направлении меньшего, чем оно само.

[3]=> больше не мобильно , так как оно в правой колонке и имеет направление вправо.

Поэтому алгоритм завершает работу, выдав 6 перестановок **.**

Реализация алгоритма в MathCad приведена ниже.

Этап поиска наибольшего мобильного числа реализован как подпрограмма FindM**.**

Глобальные константы Left и Right равны соответственно 1 и 0 и введены для большей читаемости программы.

```
Permutation(n) :=
    for i ∈ 1 .. n
        (V_i ← i   D_i ← Left)
    m ← FindM(n, V, D)
    A ← V
    while m ≠ 0
        if D_m = Left
            (r ← V_m   V_m ← V_{m−1}   V_{m−1} ← r)
            (r ← D_m   D_m ← D_{m−1}   D_{m−1} ← r)
            m ← m − 1
        otherwise
            (r ← V_m   V_m ← V_{m+1}   V_{m+1} ← r)
            (r ← D_m   D_m ← D_{m+1}   D_{m+1} ← r)
            m ← m + 1
        for i ∈ 1 .. n
            if V_i > V_m
                D_i ← Right  if D_i = Left
                D_i ← Left   otherwise
        A ← augment(A, V)
        m ← FindM(n, V, D)
    return A

FindM(n, V, D) :=
    MaxIndex ← 0
    MaxValue ← −1
    for i ∈ 1 .. n
        continue  if (i = 1 ∧ D_i = Left) ∨ (i = n ∧ D_i = Right)
        otherwise
            if V_{i−1} < V_i ∧ MaxValue ≤ V_i  if D_i = Left
                MaxIndex ← i
                MaxValue ← V_i
            if V_{i+1} < V_i ∧ MaxValue ≤ V_i  if D_i = Right
                MaxIndex ← i
                MaxValue ← V_i
    return MaxIndex
```

Проиллюстрируем работу алгоритма с n = 4.

1. **<=[1] <=[2] <=[3] <=[4]** **<=[4]** движется влево
2. **<=[1] <=[2] <=[4] <=[3]**
3. **<=[1] <=[4] <=[2] <=[3]**
4. **<=[4] <=[1] <=[2] <=[3]** **<=[4]** больше не мобильно

<=[4] больше не мобильно, так как оно в находится самом левом столбце и указывает влево.

Осуществляется транспозиция **<=[3]** с **<=[2]** , а направление **[4]** меняется и становится снова мобильным.

5. **[4]=> <=[1] <=[3] <=[2]**

6. **<=[1] [4]=> <=[3] <=[2] [4]=> движется вправо**

7. **<=[1] <=[3] [4]=> <=[2]**

8. **<=[1] <=[3] <=[2] [4]=> [4]=>** опять не мобильно

[4]=> больше не мобильно, так как оно в правой позиции и его направление вправо**.**

Осуществляется транспозиция **<=[3] с <=[1],** и изменяется направление **[4]** к **<[=4]**

9. **<=[3] <=[1] <=[2] <=[4]**

10. **<=[3] <=[1] <=[4] <=[2] <=[4]** движется влево

11. **<=[3] <=[4] <=[1] <=[2]**

12. **<=[4] <=[3] <=[1] <=[2] <=[4]** больше не мобильно

<=[4] больше не мобильно, так как оно располагается на левом конце и его направление влево

<=[3] стал не мобильным, так как нет никаких чисел меньших, чем оно на его направлении.

Осуществляется транспозиция **<=[2] с <=[1]** и нужно

(а) изменить направление **[4]** на **[4]=>**

(б) изменить направление **[3]** на **[3]=>**

Когда числа поменялись, направление всех чисел, которые больше, чем это число должно быть изменено.

13. **[4]=> [3]=> <=[2] <=[1]**

14. **[3]=> [4]=> <=[2] <=[1]**

15. **[3]=> <=[2] [4]=> <=[1]**

16. **[3]=> <=[2] <=[1] [4]=>**

[4] => больше не мобильно, так как оно на правой колонке и указывает вправо.

Осуществляется транспозиция **[3]=>** с **<=[2] ,** а 4 изменяет направление **.**

17. **<=[2] [3]=> <=[1] <=[4]**

18. **<=[2] [3]=> <=[4] <=[1]**

19. **<=[2] <=[4] [3]=> <=[1]**

20. **<=[4] <=[2] [3]=> <=[1]** 4 больше не мобильно, так как оно на самом левом столбце и имеет направление влево.

Осуществляется транспозиция **[3]=> с <=[1]** и изменение **<=[4]** на **[4]=> .**

21. **[4]=> <=[2] [1]=> <=[3]**

22. **<=[2] [4]=> [1]=> <=[3]**
23. **<=[2] [1]=> [4]=> <=[3]**
24. **<=[2] [1]=> <=[3] [4]=>**

Все числа не мобильны и, следовательно, алгоритм завершается после того как сгенерируется 4! = 24 перестановки.

В [6] для генерации с помощью транспозиции соседних элементов вместо поиска нибольшего мобильного производится вычисление места и направления транспозиции. Приводим этот вариант алгоритма на MathCad'e. Он показывает лучшие характеристики по времени.

```
Perm(n) := | for i ∈ 1 .. n
           |   (P_i ← i  C_i ← 1  PR_i ← true)
           | C_n ← 0
           | A ← P
           | i ← 1
           | while i < n
           |     | (i ← 1  x ← 0)
           |     | while C_i = n − i + 1
           |     |     | PR_i ← ¬PR_i
           |     |     | C_i ← 1
           |     |     | x ← x + 1  if PR_i
           |     |     | i ← i + 1
           |     | if i < n
           |     |     | k ← C_i + x  if PR_i
           |     |     | k ← n − i + 1 − C_i + x  otherwise
           |     |     | (w ← P_k  P_k ← P_{k+1}  P_{k+1} ← w)
           |     |     | C_i ← C_i + 1
           |     |     | A ← augment(A, P)
           | return A
```

В этом варианте булевская переменная PR для каждого i в диапазоне от 1 до n задает направление. Переменная Ci задает позицию элемента 1 относительно блоков чисел, которых единица разделяет.

Генератор перестановок может быть использован в задачах , в которых нужен перебор вариантов, например в задаче коммивояжера, задаче о назначениях. В учебных целях он может быть использован для вычисления определителя матриц . В MathCad вычисление определителя матрицы может быть записано либо формулой

$$\mathbf{Determinant}(\mathbf{a}) := \sum_{\mathbf{k}=1}^{\mathbf{n}!} \left[(-1)^{\mathbf{k}+1} \cdot \prod_{\mathbf{i}=1}^{\mathbf{n}} \mathbf{a}_{\mathbf{i},\left(\mathbf{p}^{\langle \mathbf{k} \rangle}\right)_{\mathbf{i}}} \right]$$

либо в виде вычисляемой функции

```
Determinant(a) := | n ← rows(a)
                  | P ← Perm(n)
                  | Σ ← 0
                  | for k ∈ 1..n!
                  |     | Π ← 1
                  |     | for i ∈ 1..n
                  |     |     | j ← (p^⟨k⟩)_i
                  |     |     | Π ← Π · a_{i,j}
                  |     | Σ ← Σ + (-1)^{k+1} · Π
                  | return Σ
```

Для вычисления перманента матрицы можно использовать функцию:

```
Перманент(a) := | n ← rows(a)
                | p ← Perm(n)
                | Σ ← 0
                | for k ∈ 1..n!
                |     | Π ← 1
                |     | for i ∈ 1..n
                |     |     | j ← (p^⟨k⟩)_i
                |     |     | if a_{i,j} = 0
                |     |     |     | Π ← 0
                |     |     |     | continue
                |     | Σ ← Σ + Π
                | return Σ
```

Примечание:

Строго говоря, реализация генератора перестановок в той форме, которая представлена, нельзя назвать Генератором. По принятой терминологии генератор строится из трех функций – GetFirst, IsLast и Getnext, используемых в конструкции цикла-

```
GenP (n) := | p ← GetFirst
            | while  ¬Islast (p)
            |     | p ← Getnext
            |     | UseP (p)
```

Получается, что использование UseP происходит внутри генератора. В нашем случае перестановки создаются сразу в полном объеме, как это реализовано в рассмотренных вариантах и примерах с определителем и перманентом матрицы.

ЛИТЕРАТУРА

1. Плис А.И., Сливина Н.А. Mathcad: математический практикум для экономистов и инженеров .: Финансы и статистика, 1999. 656 стр.
2. http://tropenhitze.wordpress.com/2010/01/25/steinhaus-johnson-trotter-permutation-algorithm-explained-and-implemented-in-java/ Steinhaus Johnson Trotter permutation algorithm explained and implemented in Java.
3. Кормен Т. Лейзерсон Ч. Ривест Р. Штайн К. Алгоритмы. Построение и анализ. Издательство. Вильямс,. 2007 год.
4. А. Левитин Название: Алгоритмы: введение в разработку и анализ Издательство: Вильямс Год: 2006
5. Steinhaus, Hugo (1964), One hundred problems in elementary mathematics, New York: Basic Books, pp. 49–50
6. В. Липский Название: Комбинаторика для программистов Издательство: Мир Год: 1988

УДК 37.022

П.Д. Рабинович
Московский государственный областной университет

СОЗДАНИЕ МОТИВИРУЮЩЕЙ ИНТЕРАКТИВНОЙ СРЕДЫ РАННЕГО ЛИЧНОСТНОГО И ПРОФЕССИОНАЛЬНОГО САМООПРЕДЕЛЕНИЯ ДЕТЕЙ И ПОДРОСТКОВ, РАЗВИТИЯ У НИХ МНОЖЕСТВЕННОГО ИНТЕЛЛЕКТА, ИНТЕРЕСА К ЕСТЕСТВЕННЫМ НАУКАМ И НАУЧНО-ТЕХНИЧЕСКОМУ ТВОРЧЕСТВУ

Аннотация. В статье рассматриваются вопросы формирования мотивирующей интерактивной среды раннего личностного и профессионального самоопределения детей и подростков, развития у них множе-