Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Робототехники и комплексной автоматизации

КАФЕДРА Системы автоматизированного проектирования (РК-6)

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

Студент Никифорова Ирина Андреевна

Группа РК6-71б

Тема лабораторной работы Генерация перестановок

Вариант 15

Студент _____ **Никифорова И. А.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Волосатова Т.М.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2019 г.

Оглавление

Задание на лабораторную работу

Перечислить все перестановки из 5-ти первых натуральных чисел в порядке минимального изменения, используя рекурсивный алгоритм транспозиции смежных элементов.

Используемый комбинаторный алгоритм

Для перечисления перестановок в порядке минимального изменения использовался рекурсивный алгоритм транспозиции смежных элементов. Этот порядок перечисления используется для сокращения вычислительных затрат на генерацию следующей перестановки после предыдущей.

Рекурсивный алгоритм транспозиции смежных элементов состоит из следующей последовательности действий:

1. Генерируется тривиальная одноэлементная перестановка из минимального элемента образующего множества. Эта перестановка образует одноэлементный текущий набор перестановок, который называют транспозитивной последовательностью перестановок (далее ТПП):

$$P_{текущая} = \{ p_{min} \},\ p_{min} = (n_1),\ \text{где } n_1 = min(n_i \in O)$$

2. Каждый элемент ТПП нумеруется слева направо от 1 до N - размера перестановки.

3. Выбирается следующий элемент образующего множества, который будет добавлен в текущие перестановки:

$$n_{i+1} \in O,\ \text{где } n_i \text{ - текущий элемент образующего множества}$$

4. ТПП просматривается по перестановкам с нечетными номерами и в каждую такую перестановку добавляется следующий элемент образующего множества слева направо. Это значит, что в каждую перестановку множества этот элемент добавляется на каждую позицию

последовательно, начиная с минимально допустимой и из одной перестановки получается несколько других большего на один размера:

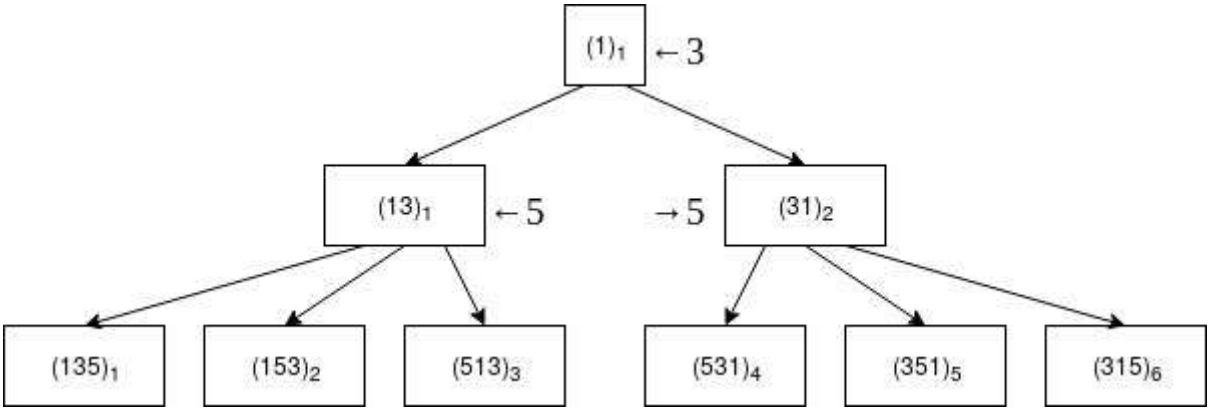
$$P_{неч} = \{ p_k \} \leftarrow n_{i+1}$$

5. Аналогично, в перестановки с четными номерами добавляется следующий элемент образующего множества справа налево:

$$P_{чет} = \{ p_k \} \rightarrow n_{i+1}, \ n_{i+1} \in O$$

6. Пункты 2 - 5 выполняются снова, пока количество элементов в каждой из перестановок не достигнет значения, равного размеру образующего множества.

Пример использования алгоритма для генерации перестановок из элементов множества O = {1, 3, 5}. Представить процесс образования будет удобно в виде дерева, при этом результат перечисления содержится в листьях:



Можно проверить, что перечисление происходит действительно в порядке наименьшего изменения: $1\underline{35}_1 \rightarrow \underline{15}3_2 \rightarrow 5\underline{13}_3 \rightarrow \underline{53}1_4 \rightarrow 3\underline{51}_5 \rightarrow 315_6$.

Описание работы программы

В начале функции *main* в программе определяется текущий (изначально это минимальный) элемент (*n*) образующего множества, состоящая только из него минимальная перестановка (*Pmin*) и ТТП (*TTS* - от *transpositive transposition sequence*).

После этого начинается основной цикл программы, который продолжается, пока размер перестановки в ТТП не станет равен 5 (значение 5 задается символической константой *N*).

Внутри основного цикла сначала выбирается следующий элемент образующего множества, которым будет дополнена каждая перестановка из ТТП. Для этого перезаписывается переменная *n*.

После этого определяется новый размер одной перестановки (*s*) - он увеличивается на 1 по сравнению со старым.

После этих подготовительных действий, создается новый вектор ТТП (*TTS_new*). Это необходимо, т.к. на место каждой перестановки после наших действий должны будут встать сразу несколько новых. Проще всего не удалять старую перестановку и заменять ее новыми, а просто создать новый вектор и записывать в него элементы в нужном порядке.

Далее определяются начальные позиции для вставки в векторы перестановок (*pos_even* и *pos_odd* для четных и нечетных перестановок соответственно). Если элемент ТПП имеет четный номер (от единицы - в программе элементы нумеруются от 0, поэтому *i* % 2 == 0 означает нечетный номер от единицы), то на его замену генерируется несколько новых перестановок с помощью функции *get_even_transposition*. Аналогично, *get_odd_transposition* - для нечетных. Эти функции записывают новый элемент образующего множества на нужную позицию в новой перестановке и дополняют ее элементами старой в правильном порядке.

В конце цикла старая ТТП заменяется на сгенерированную и цикл продолжается. Когда цикл закончен, ТТП выводится с нумерацией перестановок от единицы.

Листинг программы

```
#include <iostream>
#include <vector>

#define N 5

using namespace std;

void print_transposition( vector <int> transposition) {
    for (int i = 0; i < transposition.size(); i++) {
        cout << transposition[i];
    }
    cout << endl;
}


void print_TTS(vector <vector <int> > TTS) {
    for (int i = 0; i < TTS.size(); i++) {
        cout << i + 1 << ". ";
        print_transposition(TTS[i]);
    }
    cout << endl;
}

vector<int> get_odd_transposition(int s, int pos_odd, int n, vector<int>&
prev_transposition) {
    vector <int> new_transposition(s);
    new_transposition[pos_odd] = n;
    for (int j = 0, k = 0; j < s; j++) {
        if (j != pos_odd) {
            new_transposition[j] = prev_transposition[k];
            k++;
```

```
        }
    }

    return new_transposition;
}

vector<int> get_even_transposition(int s, int pos_even, int n, vector<int>&
prev_transposition) {
    vector <int> new_transposition(s);
    new_transposition[pos_even] = n;
    for (int j = 0, k = 0; j < s; j++) {
        if (j != pos_even) {
            new_transposition[j] = prev_transposition[k];
            k++;
        }
    }

    return new_transposition;
}

int main() {
    int n = 1;
    vector<int> Pmin = {n};
    vector< vector<int> > TTS = { Pmin };

    while(TTS[0].size() != N) {
        n = n + 1;
        int s = TTS[0].size() + 1;

        vector < vector<int> > new_TTS;
        for (int i = 0; i < TTS.size(); i ++) {
            int pos_even = 0, pos_odd = s - 1;
```

```
            if (i % 2 == 0) {
                while(pos_odd >= 0) {
                    new_TTS.push_back(get_odd_transposition(s, pos_odd, n,
                                                            TTS[i]));
                    pos_odd -= 1;
                }
            } else {
                while(pos_even < s) {
                    new_TTS.push_back(get_even_transposition(s, pos_even, n,
                                                             TTS[i]));
                    pos_even += 1;
                }
            }
        }

        TTS = new_TTS;
    }

    print_TTS(TTS);
    return 0;
}
```

Результаты работы программы

Результат работы программы представлен в таблице:

1. 12345	51. 34125	101. 42135
2. 12354	52. 34152	102. 42153
3. 12534	53. 34512	103. 42513
4. 15234	54. 35412	104. 45213
5. 51234	55. 53412	105. 54213
6. 51243	56. 54312	106. 52413
7. 15243	57. 45312	107. 25413
8. 12543	58. 43512	108. 24513
9. 12453	59. 43152	109. 24153
10. 12435	60. 43125	110. 24135
11. 14235	61. 43215	111. 21435
12. 14253	62. 43251	112. 21453
13. 14523	63. 43521	113. 21543
14. 15423	64. 45321	114. 25143
15. 51423	65. 54321	115. 52143
16. 54123	66. 53421	116. 52134
17. 45123	67. 35421	117. 25134
18. 41523	68. 34521	118. 21534
19. 41253	69. 34251	119. 21354
20. 41235	70. 34215	120. 21345
21. 41325	71. 32415	
22. 41352	72. 32451	
23. 41532	73. 32541	
24. 45132	74. 35241	
25. 54132	75. 53241	
26. 51432	76. 53214	
27. 15432	77. 35214	
28. 14532	78. 32514	
29. 14352	79. 32154	
30. 14325	80. 32145	
31. 13425	81. 23145	
32. 13452	82. 23154	
33. 13542	83. 23514	
34. 15342	84. 25314	
35. 51342	85. 52314	
36. 51324	86. 52341	
37. 15324	87. 25341	
38. 13524	88. 23541	
39. 13254	89. 23451	
40. 13245	90. 23415	
41. 31245	91. 24315	
42. 31254	92. 24351	
43. 31524	93. 24531	
44. 35124	94. 25431	
45. 53124	95. 52431	
46. 53142	96. 54231	
47. 35142	97. 45231	
48. 31542	98. 42531	
49. 31452	99. 42351	
50. 31425	100. 42315	