

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
ИМ. И. И. ПОЛЗУНОВА»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра Прикладная математика

А.В. Сорокин

**МОДЕЛИ ДЖОНСОНА ЗАДАЧ УПОРЯДОЧЕНИЯ  $n \times 2$  И  $n \times 3$**

МЕТОДИЧЕСКИЕ УКАЗАНИЯ  
к лабораторной работе по дисциплине  
«Основы моделирования»

Барнаул 2021

УДК 519.8

Сорокин А.В. Модели Джонсона задач упорядочения  $px_2$  и  $px_3$ . Методические указания к лабораторной работе по дисциплине «Основы моделирования» / А.В. Сорокин; Алт. госуд. технич. ун-т им. И.И. Ползунова.. - Барнаул, 2021. – 26 с.

В методических указаниях изложен материал по дисциплине «Основы моделирования», используемый для выполнения практических заданий. Материал содержит описание задачи упорядочения на основе производственной задачи – обработки определенного количества деталей на двух или трех обрабатывающих станках. Материал снабжен практическими примерами и вариантами заданий по каждой из рассматриваемых задач упорядочения. Методические указания предназначены для студентов, обучающихся по направлению «Программная инженерия», «Информатика и вычислительная техника».

© Сорокин А.В., 2021

## Содержание

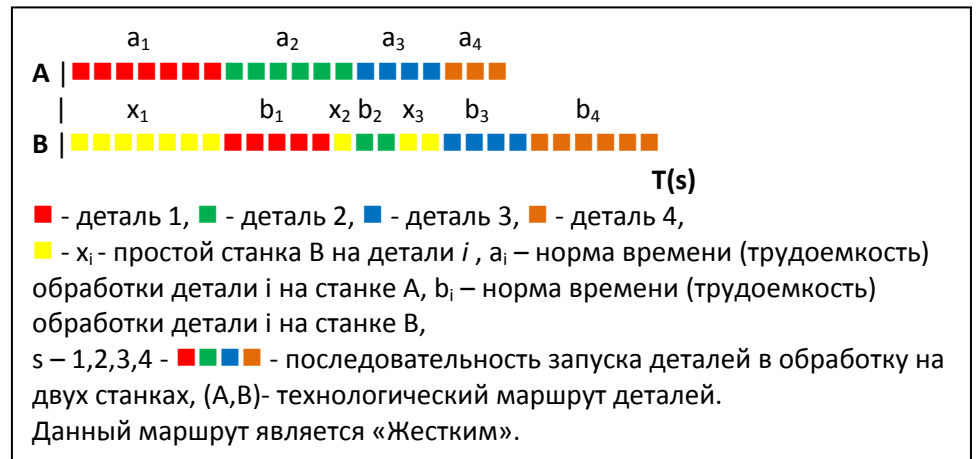
<b>1. Задание 1. Работа с моделью задачи упорядочения Джонсона <math>px2</math></b> .....	4
<b>2. Задание 2. Работа с моделью задачи упорядочения Джонсона <math>px3</math></b> .....	6
<b>3. Задание 3. Не выполнение условия Джонсона. Метод перебора</b> .....	8
<b>4. Алгоритмы получения перестановок</b> .....	9
4.1. Генерация перестановок .....	9
4.2. Перестановки. Идеи реализации .....	12
4.3. Рекурсивный алгоритм перестановки .....	15
4.4. Нерекурсивный алгоритм генерации перестановок .....	15
<b>5. Варианты для задания №1</b> .....	20
<b>6. Варианты для задания №2</b> .....	22
<b>7. Вопросы по данной теме</b> .....	25
<b>8. Список литературы</b> .....	26

## Задание 1. Работа с моделью задачи упорядочения Джонсона nx2

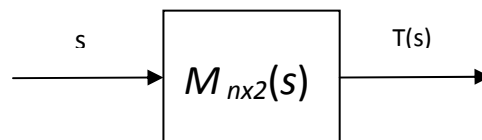
1. Разобраться с построением модели задачи Джонсона-Белмана nx2 из теории расписаний.
2. Ознакомиться с формулировкой задачи Джонсона-Белмана nx2.
3. Научиться строить график Ганта для задачи Джонсона-Белмана nx2 (с n деталями и двумя станками).

Пример 1.1

№	$a_i$	$b_i$
1	7	5
2	6	2
3	1	4
4	3	6



4. Изучить процесс построения модели задачи Джонсона (nx2) на основе графика Ганта



где  $s$  - последовательность запуска деталей в обработку на двух станках,

$$\min_s T(s) = \sum_{i=1}^n b_i + \min_s X(s), \quad X(s) = \sum_{i=1}^n x_i - \text{суммарный простой станка B,}$$

$$X(s) = \max_{1 \leq u \leq n} K(u), \quad K(u) = \sum_{i=1}^u a_i - \sum_{i=1}^{u-1} b_i.$$

Для вычисления значений функции  $K$  можно использовать рекуррентную формулу:

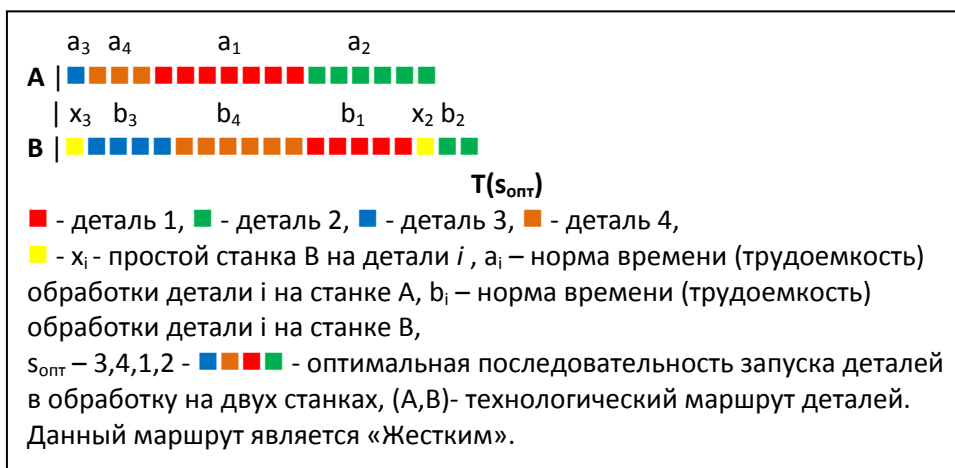
$$K(u) = K(u-1) + a_u - b_{u-1}, \quad K(0) = 0, \quad b_0 = 0, \quad u = 1, 2, \dots, n.$$

5. Изучить возможности поиска оптимального решения задачи Джонсона для двух станков. Изучить подход вывода алгоритма Джонсона и сам алгоритм. Решить задачу упорядочения согласно варианту с использованием алгоритма Джонсона, найдя оптимальный порядок запуска последовательности  $s_{opt}$   $n$  деталей в обработку на двух станках.

6. Построить график Ганта для оптимальной последовательности запуска деталей своей задачи согласно варианту. Используя

### Пример 1.2.

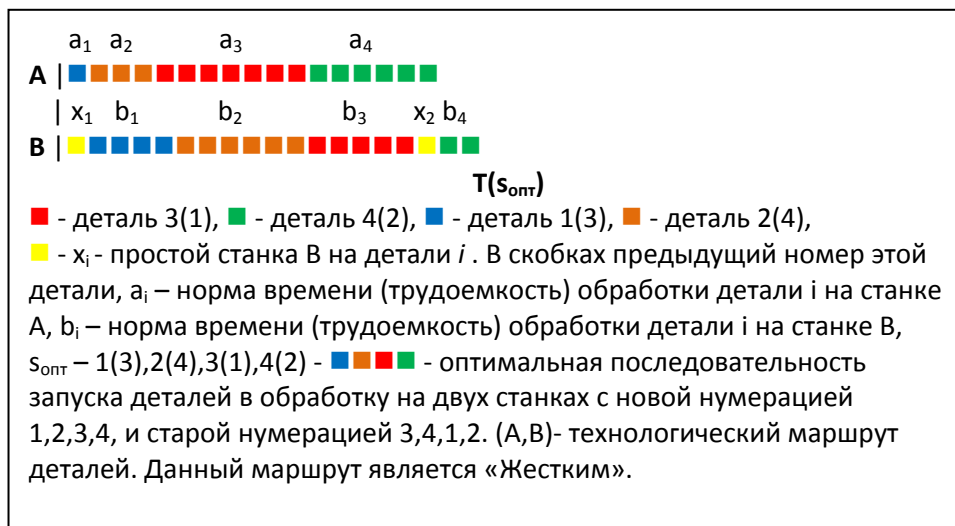
№	a <sub>i</sub>	b <sub>i</sub>
3	1	4
4	3	6
1	7	5
2	6	2



7. Рассчитать оптимальное время окончания обработки всех деталей на двух станках. Это удобно сделать перенумеровывая оптимальный порядок в порядке возрастания индексов и затем использовать формулы из пункта 4. График Ганта тоже можно приводить с новой нумерацией и новым определением цвета.

### Пример 1.3.

№	$a_i$	$b_i$
1	1	4
2	3	6
3	7	5
4	6	2



### Выводимая информация задания 1

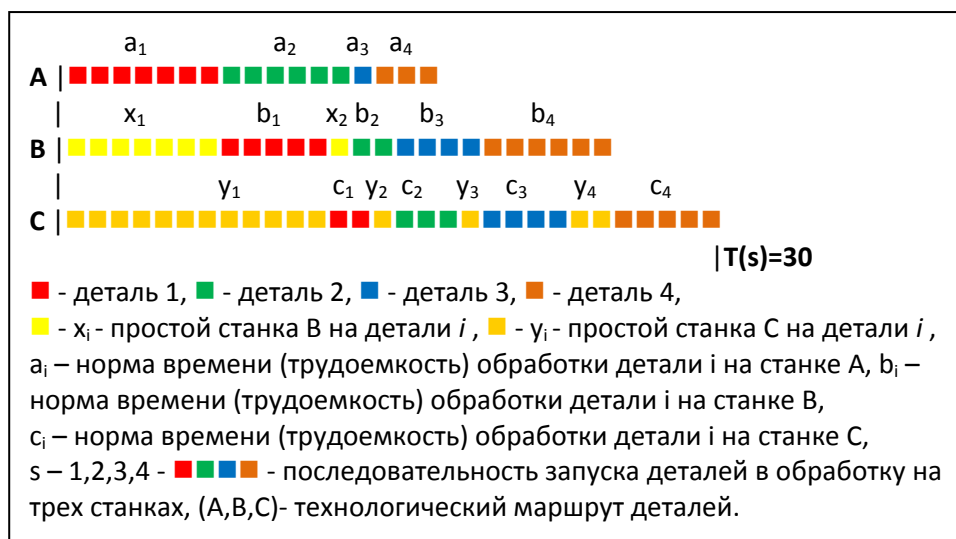
При выполнении программы должны выводиться таблица с исходными значениями с исходным порядком согласно заданному варианту [1], с оптимальным порядком, оптимальная последовательность запуска, графики Ганта для исходной таблицы значений норм времени обработки и для таблицы с оптимальным порядком.

## Задание 2. Работа с моделью задачи упорядочения Джонсона nx3

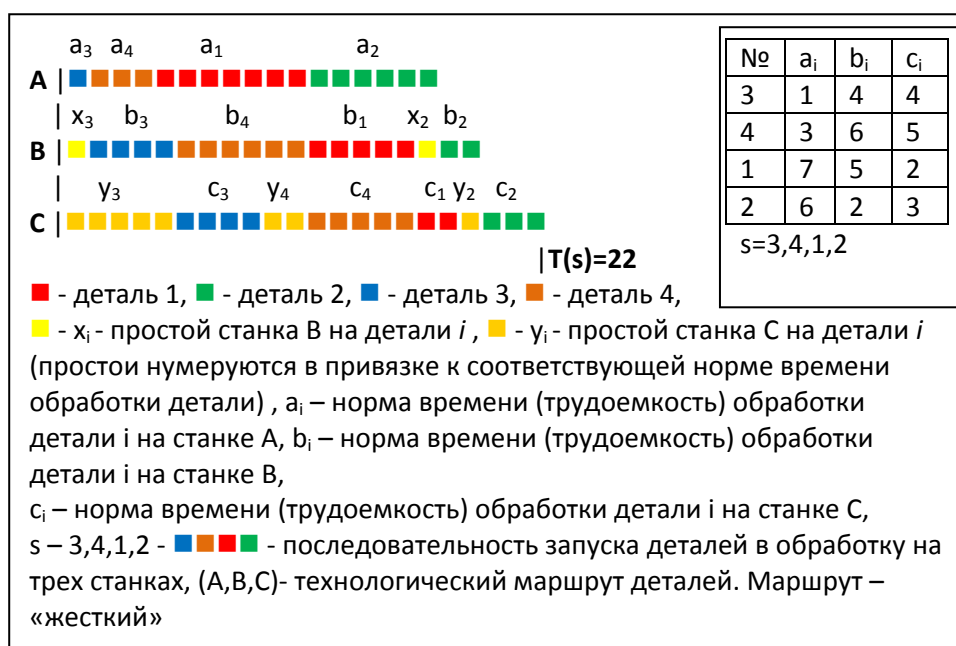
- 1.Разобраться с построением модели задачи Джонсона- Белмана nx3 из теории расписаний.
2. Ознакомиться с формулировкой задачи Джонсона- Белмана nx3.
- 3.Научиться строить график Ганта для задачи Джонсона-Белмана с n деталями и тремя станками.

Пример 2.1  
Таблица 2.1

№	$a_i$	$b_i$	$c_i$
1	7	5	2
2	6	2	3
3	1	4	4
4	3	6	5

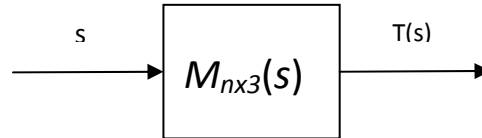


Рассмотрим вычисления  $T(s)$  по графику Ганта для исходной таблицы с последовательностью запуска деталей в обработку  $s=3,4,1,2$



По графику Ганта исходной таблицы 2.1 получим следующее время окончания производственного процесса  $T(s)=30$ , а по графику Ганта таблицы 2.2 с последовательностью запуска  $s=3,4,1,2$  получим следующее время окончания производственного процесса  $T(s)=22$ .

4. Изучить процесс построения модели задачи Джонсона с  $(n \times 3)$  на основе графика Ганта



где  $s$  - последовательность запуска деталей в обработку на трех станках,

$$\min_s T(s) = \sum_{i=1}^n c_i + \min_s Y(s), \quad Y(s) = \sum_{i=1}^n y_i - \text{суммарный простой станка } C,$$

$$Y(s) = \max_{1 \leq v \leq n} \left( \max_{1 \leq u \leq v} K(u) + H(v) \right) = \max_{1 \leq u \leq v \leq n} (K(u) + H(v)), \quad (2.1)$$

$$K(u) = \sum_{i=1}^u a_i - \sum_{i=1}^{u-1} b_i, \quad H(v) = \sum_{i=1}^v b_i - \sum_{i=1}^{v-1} c_i$$

Для вычисления значений функции  $K(u)$  и  $H(v)$  можно использовать рекуррентные формулы:

$$K(u) = K(u-1) + a_u - b_{u-1}, \quad K(0) = 0, \quad b_0 = 0, \quad u = 1, 2, \dots, n.$$

$$H(v) = H(v-1) + b_v - c_{v-1}, \quad H(0) = 0, \quad c_0 = 0, \quad v = 1, 2, \dots, n. \quad (2.2)$$

5. Изучить возможности поиска оптимального решения задачи Джонсона для трех станков. Известно [1], что задачу упорядочения Джонсона для трех станков можно свести к задаче упорядочения Джонсона от двух станков, если выполняется условие:

$$\left( \min_i a_i \geq \max_j b_j \right) \vee \left( \min_k c_k \geq \max_j b_j \right). \quad (2.3)$$

Рассмотрим таблицу 2.3, для которой условие 2.3 выполняется

№	$a_i$	$b_i$	$c_i$
1	7	5	6
2	6	2	7
3	1	4	8
4	3	6	9

преобразуем ее в таблицу 2.4

№	$d_i$	$e_i$
1	12	11
2	8	9
3	5	12
4	9	15

по правилу  $d_i = a_i + b_i$ ,  $e_i = b_i + c_i$ , и к ней применяется алгоритм Джонсона.

В результате находим оптимальную последовательность запуска  $n$  деталей в обработку  $s_{\text{опт}}$ . В нашем случае это последовательность  $s_{\text{опт}}=3,2,4,1$ . Переупорядочим исходную таблицу 2.3 в указанном порядке в таблицу 2.4

таблица 2.4

№	$a_i$	$b_i$	$c_i$
3	1	4	8
2	6	2	7
4	3	6	9
2	7	5	6

Для вычисления  $T(s_{\text{опт}})$  можно использовать упрощенную формулу вычисления простоя

Если выполняется условие (2.2), то формула суммарного простоя  $Y(s)$  станка  $C$  упрощается как показано ниже

$$Y(s) = \max_{1 \leq v \leq n} (\max_{1 \leq u \leq v} K(u) + H(v)) = \max_{1 \leq u \leq v \leq n} (K(u) + H(v)) \Rightarrow Y(s) = \max_{1 \leq u \leq n} (K(u) + H(u)) \quad (2.4)$$

В этом случае простой легко вычисляется с использованием одного оператора цикла.

Произведем расчет суммарного простоя используя формулы (2.2) и (2.4)

$K(1) = a_1 = 1$	$H(1) = b_1 = 4$	$K(1) + H(1) = 5$
$K(2) = 1 + a_2 - b_1 = 1 + 6 - 4 = 3$	$H(2) = 4 + b_2 - c_1 = 4 + 2 - 8 = -2$	$K(2) + H(2) = 1$
$K(3) = 3 + a_3 - b_2 = 3 + 3 - 2 = 4$	$H(3) = -2 + b_3 - c_2 = -2 + 6 - 7 = -3$	$K(3) + H(3) = 1$
$K(4) = 4 + a_4 - b_3 = 4 + 7 - 5 = 6$	$H(4) = -3 + b_4 - c_3 = -3 + 5 - 9 = -7$	$K(4) + H(4) = -1$

Максимальное значение суммы  $K(i) + H(i)$  достигается при  $i = 1$ . И  $Y(s_{\text{опт}})=5$ , а  $T(s_{\text{опт}})=30+5=35$ .

### Выводимая информация задания 2.

При выполнении программы для решения задачи Джонсона согласно заданному варианту [2] должны выводиться условие сведение задачи Джонсона  $n \times 3$  к задаче Джонсона  $n \times 2$ , таблица с исходными значениями с исходным порядком, с оптимальным порядком, оптимальная последовательность запуска, графики Ганта для исходной таблицы значений норм времени обработки и для таблицы с оптимальным порядком.

### Задание 3. Не выполнение условия Джонсона. Метод перебора

Не выполнение условия Джонсона (2.3) не дает возможности свести задачу упорядочения  $n \times 3$  к задаче упорядочения  $n \times 2$ . Для поиска оптимального решения остается лишь метод перебора, который будет гарантировать, что найденное решение будет самым оптимальным. Общее количество вариантов решения задачи равно  $n!$ , где  $n$  – количество деталей. Предположить, что исходный вариант задания 2 не удовлетворяет условию (2.3) и решить задачу методом перебора.

Реализуя метод перебора можно использовать лексиграфический порядок перестановок. Так например для  $n=5$  будем иметь

$$1,2,3,4,5 \rightarrow 1,2,3,5,4 \rightarrow 1,2,4,3,5 \rightarrow 1,2,4,5,3 \rightarrow 1,2,5,3,4 \rightarrow 1,2,5,4,3 \rightarrow 1,3,2,4,5 \rightarrow \dots \rightarrow 5,4,3,2,1.$$

Вместо лексиграфического порядка можно использовать произвольный порядок.

Вычисляя для каждой перестановки  $s$  значение  $T(s)$  мы сможем в конечном итоге найти оптимальное (минимальное  $T(s)$ ) и соответствующее ему  $s$ .



Для создания таблицы исходных данных для задачи Джонсона с тремя станками, необходимо взять соответствующий вариант задания для задачи Джонсона  $px3$  из методички и откорректировать таблицу так, чтобы не выполнялось условие (2.3).

#### **Выводимая информация задания 3.**

При выполнении программы для решения задачи Джонсона не удовлетворяющей условию (2.3) должны выводиться, таблица с исходными значениями с исходным порядком, с оптимальным порядком, оптимальная последовательность запуска, графики Ганта для исходной таблицы значений норм времени обработки и для таблицы с оптимальным порядком.

### **4. Алгоритмы получения перестановок**

#### **4.1. Генерация перестановок**

Перестановка – это комбинация элементов из  $N$  разных элементов взятых в определенном порядке. В перестановке важен порядок следования элементов, и в перестановке должны быть задействованы все  $N$  элементов.

Задача: Найти все возможные перестановки для последовательности чисел 1, 2, 3.  
Существуют следующие перестановки:

1: 1 2 3  
2: 1 3 2  
3: 2 1 3  
4: 2 3 1  
5: 3 1 2  
6: 3 2 1

#### **Перестановки без повторений**

Количество перестановок для  $N$  различных элементов составляет  $N!$ . Действительно:

- на первое место может быть помещен любой из  $N$  элементов (всего вариантов  $N$ ),
- на вторую позицию может быть помещен любой из оставшихся  $(N-1)$  элементов (итого вариантов  $N \cdot (N-1)$ ),
- если продолжить данную последовательность для всех  $N$  мест, то получим:  $N \cdot (N-1) \cdot (N-2) \cdot \dots \cdot 1$ , то есть всего  $N!$  перестановок.

Рассмотрим задачу получения всех перестановок чисел  $1 \dots N$  (то есть последовательности длины  $N$ ), где каждое из чисел входит ровно по 1 разу. Существует множество вариантов порядка получения перестановок. Однако наиболее часто решается задача генерации перестановок в лексикографическом порядке (см. пример выше). При этом все перестановки сортируются сначала по первому числу, затем по второму и т.д. в порядке возрастания. Таким образом, первой будет перестановка  $1 \ 2 \ \dots \ N$ , а последней -  $N \ N-1 \ \dots \ 1$ .

Рассмотрим алгоритм решения задачи. Дана исходная последовательность чисел. Для получения каждой следующей перестановки необходимо выполнить следующие шаги:

- Необходимо просмотреть текущую перестановку справа налево и при этом следить за тем, чтобы каждый следующий элемент перестановки (элемент с большим номером) был не

более чем предыдущий (элемент с меньшим номером). Как только данное соотношение будет нарушено необходимо остановиться и отметить текущее число (позиция 1).

- Снова просмотреть пройденный путь справа налево пока не дойдем до первого числа, которое больше чем отмеченное на предыдущем шаге.
- Поменять местами два полученных элемента.
- Теперь в части массива, которая размещена справа от позиции 1 надо отсортировать все числа в порядке возрастания. Поскольку до этого они все были уже записаны в порядке убывания необходимо эту часть подпоследовательность просто перевернуть.

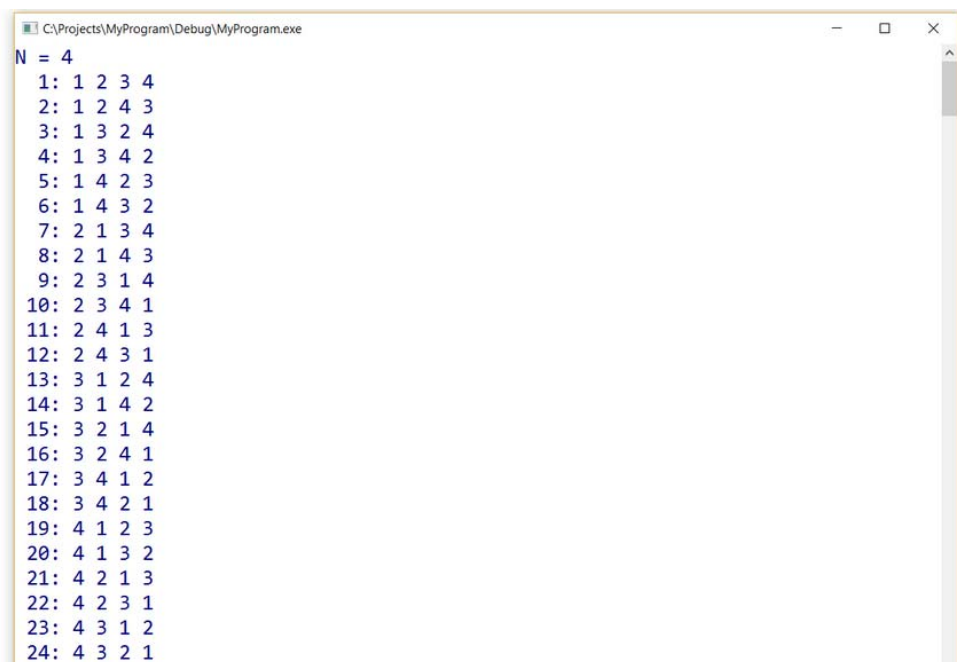
Таким образом мы получим новую последовательность, которая будет рассматриваться в качестве исходной на следующем шаге.

Реализация на C++

```
#include <iostream>
using namespace std;
void swap(int *a, int i, int j)
{
    int s = a[i];
    a[i] = a[j];
    a[j] = s;
}
bool NextSet(int *a, int n)
{
    int j = n - 2;
    while (j != -1 && a[j] >= a[j + 1]) j--;
    if (j == -1)
        return false; // больше перестановок нет
    int k = n - 1;
    while (a[j] >= a[k]) k--;
    swap(a, j, k);
    int l = j + 1, r = n - 1; // сортируем оставшуюся часть последовательности
    while (l < r)
        swap(a, l++, r--);
    return true;
}
void Print(int *a, int n) // вывод перестановки
{
    static int num = 1; // номер перестановки
    cout.width(3); // ширина поля вывода номера перестановки
    cout << num++ << ": ";
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    cout << endl;
}
int main()
{
    int n, *a;
```

```
cout << "N = ";  
cin >> n;  
a = new int[n];  
for (int i = 0; i < n; i++)  
    a[i] = i + 1;  
Print(a, n);  
while (NextSet(a, n))  
    Print(a, n);  
cin.get(); cin.get();  
return 0;  
}
```

### Результат выполнения



```
C:\Projects\MyProgram\Debug\MyProgram.exe  
N = 4  
1: 1 2 3 4  
2: 1 2 4 3  
3: 1 3 2 4  
4: 1 3 4 2  
5: 1 4 2 3  
6: 1 4 3 2  
7: 2 1 3 4  
8: 2 1 4 3  
9: 2 3 1 4  
10: 2 3 4 1  
11: 2 4 1 3  
12: 2 4 3 1  
13: 3 1 2 4  
14: 3 1 4 2  
15: 3 2 1 4  
16: 3 2 4 1  
17: 3 4 1 2  
18: 3 4 2 1  
19: 4 1 2 3  
20: 4 1 3 2  
21: 4 2 1 3  
22: 4 2 3 1  
23: 4 3 1 2  
24: 4 3 2 1
```

## 4.2. Перестановки

### Идеи реализации

#### Представление перестановок в программе

Выберем наиболее простой способ хранить найденные перестановки в программе: массив. Если перестановки предстоит помещать в список, для такого списка подойдёт массив ссылок на анонимные массивы, в каждом из которых будет размещаться перестановка. Вот как будет выглядеть структура, хранящая все шесть перестановок из множества  $S_3$ :  $\langle ([3, 2, 1], [2, 3, 1], [2, 1, 3], [3, 1, 2], [1, 3, 2], [1, 2, 3]) \rangle$ .

#### Рекурсивная реализация

Для реализации рекурсивного алгоритма нужно исходную задачу — поиск перестановок из  $S_n$  — свести к решению таких же задач (поиск перестановок из  $S_k$  для некоторых  $k$ ). Если решение основной задачи оформить в виде процедуры `permutations`, принимающей параметр `$n`, то такая процедура помогала бы сама себе, рекурсивно вызывая себя для решения вспомогательных задач.

Такое представление есть, и оно удивительно просто. Достаточно знать множество  $S_{n-1}$ , чтобы найти  $S_n$ . Из каждой  $(n-1)$ -перестановки  $\sigma \in S_{n-1}$  наделаем  $n$  штук новых  $n$ -перестановок. Для этого в каждую перестановку  $\sigma$  нужно вставить число  $n$  всевозможными способами. Сколько всего таких способов? Конечно,  $n$ . К примеру, для 3-перестановки  $(2, 1, 3)$  после указанных манипуляций получатся 4-перестановки  $(4, 2, 1, 3)$ ,  $(2, 4, 1, 3)$ ,  $(2, 1, 4, 3)$ ,  $(2, 1, 3, 4)$  — всего 4 штуки.

Кстати, мы получили одно из доказательств того факта, что количество  $n$ -перестановок равно  $n!$ . Это доказательство по индукции. Вообще, индукция в математике очень близка по смыслу к рекурсии в информатике.

Параметр  $n$  при вызове процедуры `permutations` — это количество элементов в некотором множестве, поэтому допустимые значения этого параметра — целые неотрицательные числа. Поэтому особого внимания требует граничный случай, когда  $n = 0$ . Что должна возвратить процедура `permutations` в этом случае? Список перестановок пустого множества. Сколькими способами можно упорядочить все (ноль) элементов пустого множества? Только одним — вот таким:  $()$ . Именно так, хоть это и выглядит странно. Таким образом, при  $n = 0$  процедура `permutations` должна

вернуть одноэлементный список перестановок; единственная перестановка в этом списке — это пустой список чисел  $(1, 2, \dots, 0)$ , то есть  $()$ .

Основная часть процедуры `permutations`, таким образом, будет содержать двойной цикл. Во внешнем будут перебираться  $(n-1)$ -перестановки. Во внутреннем будут перебираться все способы, которыми можно вставить число  $n$  в каждую из них.

#### Итеративные реализации

Наблюдения за работой рекурсивной программы выявляют существенный недостаток рекурсивного подхода. Программа некоторое время ничего не выводит на экран, накапливая перестановки в списке. Это означает, что к моменту вывода первой строчки найдены и помещены в массив все  $n!$  перестановок. С учётом того, что  $n!$  очень быстро растёт с ростом  $n$ , мы рискуем очень быстро исчерпать всю доступную память компьютера.

Заметим, что совершенно не обязательно хранить в памяти все перестановки только для того, чтобы вывести их на экран одну за другой. Можно в цикле переходить от текущей перестановки к следующей за ней подобно тому, как в цикле, перебирающем натуральные числа, от текущего числа переходят к следующему, на единицу большему.

Осталось лишь выяснить, какую перестановку из  $S_n$  мы выберем в качестве первой, и каким способом мы станем переходить от текущей перестановки к следующей. Иными словами, нужно описать порядок перебора элементов  $\sigma \in S_n$ .

Совершенно естественно выбрать в качестве начальной перестановки *тождественную*  $(1, 2, \dots, n)$ , которая всегда содержится в  $S_n$ , какое бы  $n$  мы ни взяли.

#### Тасование

Взглянем на  $n$ -элементное множество  $X$  как на карточную колоду. Каждая из перестановок этого множества получается в результате некоторого тасования колоды. Если бы удалось перебрать все возможные варианты тасования, можно было бы получить все перестановки. Такой перебор возможен.

Тасование представим как последовательность из  $n$  транспозиций карт. Сначала последняя карта меняется местами с любой картой из колоды (очень важно не исключить по ошибке случай, когда карта меняется местами сама с собой). На следующем шаге предпоследняя карта меняется с любой, за исключением последней. Потом предпредпоследняя — с любой, кроме двух последних, и так далее. Так мы дойдём до первой карты колоды, которой не останется ничего другого, как обменяться местами самой с собой. Такой подход позволяет представить тасование как  $n$ -элементную последовательность номеров карт  $\langle i_1, i_2, \dots, i_{n-1}, i_n \rangle$ , где  $i_k \leq k$  для всех  $k \in \{1, \dots, n\}$ . Для того, чтобы выполнить закодированное в виде такой последовательности тасование, нужно менять местами карты с номерами  $k$  и  $i_k$ , последовательно уменьшая  $k$  от  $n$  до 1. Например, для последовательности  $\langle 1, 1, 3, 2, 3, 5 \rangle$  получается перестановка  $(4, 1, 6, 2, 3, 5)$  как результат цепочки транспозиций:

$(1, 2, 3, 4, \mathbf{5}, \mathbf{6}),$   
 $(1, 2, \mathbf{3}, 4, \mathbf{6}, 5),$   
 $(1, \mathbf{2}, 6, \mathbf{4}, 3, 5),$   
 $(1, 4, \mathbf{6}, 2, 3, 5),$   
 $(\mathbf{1}, \mathbf{4}, 6, 2, 3, 5),$   
 $(\mathbf{4}, 1, 6, 2, 3, 5).$

Пока что остаётся открытым вопрос о том, как организовать перебор таких последовательностей. Но прежде следует выяснить, имеется ли взаимно-однозначное соответствие между последовательностями и перестановками множества. Иными словами, любая ли перестановка может быть получена таким способом, и не дадут ли разные тасования одну и ту же перестановку?

На второй из этих вопросов ответ отрицательный: нет, не дадут. Пусть имеется два различных тасования  $I = \langle i_1, \dots, i_n \rangle$  и  $J = \langle j_1, \dots, j_n \rangle$ . Тогда найдётся хотя бы одно такое  $k$ , что  $i_k \neq j_k$ . Выберем наибольшее из таких  $k$ . Возьмём колоду и начнём тасовать. Тогда при тасованиях  $I$  и  $J$ , очевидно, на  $k$ -м месте окажутся разные карты, так что разным тасованиям соответствуют разные перестановки.

Осталось доказать, что каждой перестановке отвечает некоторое тасование. Проще всего убедиться в этом подсчётом тасований. Для элементов последовательности, задающей тасование, имеем  $i_1 = 1$ ,  $i_2 \in \{1, 2\}$ ,  $i_3 \in \{1, 2, 3\}$ , и так далее. Таким образом, количество тасований равно  $1 \cdot 2 \cdot \dots \cdot n = n!$ , то есть совпадает с количеством перестановок. С учётом того, что разным тасованиям отвечают разные перестановки, получаем взаимно-однозначное соответствие.

Для перебора тасований мы воспользовались идеями из книги [6]. Начиная с тасования  $\langle 1, 1, \dots, 1 \rangle$ , будем увеличивать элементы этой последовательности на единицу, если это возможно, начиная с самого первого. Если же  $k$ -й элемент невозможно увеличить, не нарушив неравенства  $i_k \leq k$ , мы делаем  $i_k$  равным единице, после чего переходим к следующему. Сразу скажем, что попытка увеличить первый элемент заведомо обречена на провал, поэтому можно начинать прямо со второго. Если уже не осталось элементов, которые можно увеличить, то есть получено тасование  $\langle 1, 2, 3, \dots, n \rangle$ , алгоритм останавливается.

Этот алгоритм очень похож на алгоритм перебора всех  $n$ -значных натуральных чисел в некоторой системе счисления. Начиная с наименьшего числа, мы стараемся увеличить на единицу первую цифру, а когда это становится невозможным, берёмся за вторую, потом за третью, и так далее. Отличие заключается в том ограничении, которое должно выполняться при увеличении очередной цифры: результат должен быть меньше основания системы счисления. В нашем же случае для каждой «цифры» своё ограничение: нельзя превысить номер «разряда» (нумерация начинается с единицы).

Как отмечается в [6], алгоритм перебора чисел реализован в хорошо знакомом всем устройстве — механическом *одометре*, который служит для учёта пробега автомобиля, электрической энергии, воды или газа (рисунок 17.1. «Одометр (с сайта <http://avto-all.com>)»). Одометр состоит из набора колёс с нанесёнными на них цифрами. Колёса связаны таким образом, что полный оборот одного колеса вызывает поворот следующего на одну цифру. Так что когда возможности для увеличения цифр на данном колесе исчерпаны, цифра обнуляется, но зато делается попытка увеличить цифру на следующем колесе. Одометр — ещё одно полезное изобретение Герона Александрийского, сделанное на самой заре автомобилизма — в первом веке нашей эры.

Рисунок 17.1. Одометр (с сайта <http://avto-all.com>)



Если бы на первом колесе одометра была лишь одна цифра, на втором — две, на третьем — три, и так далее, мы получили бы прибор для построения тасований. Нам не удалось найти изображение такого усовершенствованного одометра, так что читателю придётся включить воображение.

Проиллюстрируем алгоритм на примере тасований трёхкарточной колоды. Вот полный перечень тасований, получаемых при его помощи:  $\langle 1, 1, 1 \rangle$ ,  $\langle 1, 2, 1 \rangle$ ,  $\langle 1, 1, 2 \rangle$ ,  $\langle 1, 2, 2 \rangle$ ,  $\langle 1, 1, 3 \rangle$ ,  $\langle 1, 2, 3 \rangle$ .

#### Получение перестановок в лексикографическом порядке

Обратите внимание на то, что в предыдущих версиях программы мы никак не оговаривали порядок, в котором будут выводиться перестановки. Теперь же мы усложним задачу, потребовав, чтобы перестановки выводились в некотором естественном порядке.

Таким естественным способом упорядочить перестановки  $\sigma \in S_n$  будет так называемый *лексикографический порядок*. При сравнении двух перестановок меньшей будет та, у которой на первом месте будет меньшее число. Если числа в первой позиции у обеих перестановок одинаковы, сравниваются вторые числа. Если же и они равны, сравниваются третьи, и так далее. Например,  $\langle 2, 1, 3 \rangle \prec \langle 2, 3, 1 \rangle$ , если знаком  $\prec$  мы обозначим отношение лексикографического «меньше». Приведём полное перечисление перестановок из  $S_3$  в лексикографическом порядке:

$$\langle 1, 2, 3 \rangle \prec \langle 1, 3, 2 \rangle \prec \langle 2, 1, 3 \rangle \prec \langle 2, 3, 1 \rangle \prec \langle 3, 1, 2 \rangle \prec \langle 3, 2, 1 \rangle.$$

Выбор тождественной перестановки в качестве самой первой хорошо согласуется с лексикографическим порядком перебора перестановок, поскольку, очевидно, она является наименьшей в лексикографическом смысле. Между прочим, последней в списке всегда будет перестановка  $\langle n, n-1, \dots, 2, 1 \rangle$ .

Осталось выяснить, как преобразовать некоторую перестановку, чтобы получить из неё следующую в лексикографическом смысле. Введём обозначение  $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_n)$  для текущей перестановки. Заметим, что для самой последней перестановки в списке выполняются неравенства  $\sigma_1 > \sigma_2 > \dots > \sigma_n$ . Если же найдётся такой номер  $i \in \{1, 2, \dots, n\}$ , что  $\sigma_i < \sigma_{i+1}$ ,  $\sigma_{i+1} > \sigma_{i+2} > \dots > \sigma_{n-1} > \sigma_n$ , это означает, что  $\sigma_i$  может быть увеличена при переходе к следующей перестановке за счёт переупорядочения чисел  $\sigma_i, \sigma_{i+1}, \dots, \sigma_n$ . Так как следующая перестановка является ближайшей в лексикографическом смысле, числа  $\sigma_1, \sigma_2, \dots, \sigma_{i-1}$  должны остаться без изменений. Число  $\sigma_i$  меняется местами с наименьшим из чисел  $\sigma_j$ , стоящих правее его ( $j > i$ ), но при этом больших, чем  $\sigma_i$ . Затем числа  $\sigma_{i+1}, \dots, \sigma_n$  переупорядочиваются по возрастанию. Это несложно сделать, поскольку как до обмена  $\sigma_i \leftrightarrow \sigma_j$ , так и после него они упорядочены по убыванию — подумайте, почему. Значит, переупорядочения мы добьёмся, обменяв  $\sigma_{i+1} \leftrightarrow \sigma_n, \sigma_{i+2} \leftrightarrow \sigma_{n-1}, \dots$

В результате всех этих манипуляций перестановка  $\sigma$  превратится в перестановку  $\sigma'$  такую, что  $\sigma \prec \sigma'$ , причём наименьшую возможную.

Продemonстрируем превращение  $\sigma$  в  $\sigma'$  на конкретном примере. Пусть  $\sigma = (4, 2, 5, 3, 1)$ . Ищем  $\sigma_i$ :  $(4, \mathbf{2}, 5, 3, 1)$ . Теперь среди следующих за найденным числом  $\sigma_2 = \mathbf{2}$  найдём наименьшее из чисел, больших чем  $\sigma_2$ . Это  $\sigma_4 = \mathbf{3}$ . Меняем местами  $\sigma_2$  и  $\sigma_4$ :  $(4, \mathbf{3}, 5, \mathbf{2}, 1)$ . Числа, следующие за  $\sigma_2$ , упорядочены в порядке убывания. Переупорядочим их по возрастанию:  $(4, 3, 1, 2, 5)$ . Перестановка  $\sigma'$  готова.

### 4.3. Рекурсивный алгоритм перестановки

```
{ рекурсивные алгоритмы: генерация перестановок }
const n = 3; { количество элементов в перестановке }
var a:array[1..n] of integer;
    index : integer;

procedure generate (l,r:integer);
var i,v:integer;
begin
    if (l=r) then begin
        for i:=1 to n do write(a[i], ' ');
        writeln;
    end else begin
        for i := l to r do begin
            v:=a[l]; a[l]:=a[i]; a[i]:=v; {обмен a[i],a[j]}
            generate(l+1,r);           {вызов новой генерации}
            v:=a[l]; a[l]:=a[i]; a[i]:=v; {обмен a[i],a[j]}
        end;
    end;
end;

begin
    for index := 1 to N do A[index]:=index;
    generate( 1,n );
end.
```

### 4.4. Нерекурсивный алгоритм генерации перестановок

Предлагаемый ниже нерекурсивный алгоритм несколько отличается от изложенных в книге Липского [1] и обнаруженных мной в русскоязычном сегменте интернета. Надеюсь будет интересно.

Кратко постановка задачи. Имеется множество размерности N. Необходимо получить все N! возможных перестановок.

Далее, для простоты, используем в качестве множества целые числа (1..N). Вместо чисел можно использовать любые объекты, т.к. операций сравнения элементов множества в алгоритме нет.

Для хранения промежуточных данных сформируем структуру данных следующего вида:

```
type dtree
    ukaz as integer      ' номер выбранного элемента в списке
    spisok() as integer  ' список доступных значений
end type
```

и заполним ее первоначальными значениями

```
Dim masiv(N-) As dtree ' размерность массива = N-1
For ii = 1 To N - 1
    masiv(ii).ukaz = 1
    ReDim masiv(ii).spisok(N + 1 - ii) ' устанавливаем размерность списка
    For kk = 1 To (N + 1 - ii)
        masiv(ii).spisok(kk) = kk + ii - 1
    Next
```



Next

Номер элемента в массиве `masiv` будем далее называть уровнем.

В список первого уровня заносим все элементы множества. На первом уровне размерность списка равна  $N$  и сам список не изменяется по всему ходу выполнения алгоритма. При первичном заполнении все указатели в массиве устанавливаются на первый элемент в списке.

На каждом следующем уровне его список формируется на основании списка предыдущего уровня, но без одного элемента, который помечен указателем. На предпоследнем уровне ( $N-2$ ) список содержит три элемента. На последнем уровне ( $N-1$ ) список содержит два элемента. Список нижнего уровня формируется как список предыдущего уровня без элемента, на который указывает указатель предыдущего уровня.

В результате первичного заполнения получены две первых перестановки. Это общий массив сформированный на верхних уровнях ( $1 \dots (N-2)$ ) из элементов списка на которые указывают указатели.

```
For ii = 1 To N-2
    massiv(ii).spisok(ukaz)
Next
```

и из списка последнего уровня- две пары элементов в разном порядке ( два хвостика 1 2 и 2 1)

```
+ massiv(N-1).spisok(1) + massiv(N-1).spisok(2)
+ massiv(N-1).spisok(2) + massiv(N-1).spisok(1)
```

Все дальнейшие перестановки формируются также, всегда с предпоследнего уровня ( $N-2$ ), Порядок получения последующих перестановок состоит в том, что находясь на предпоследнем уровне ( $N-2$ ) и сформировав две перестановки пытаемся увеличить указатель выбранного элемента на 1.

Если это возможно, то на последнем уровне меняем список и повторяемся.

Если на предпоследнем уровне увеличить указатель не удастся (перебраны все возможные варианты), то поднимаемся до уровня на котором увеличение указателя (перемещение вправо) возможно. Условие окончания работы алгоритма — указатель на первом уровне выходит за  $N$ .

После сдвига указателя вправо меняем список под ним и двигаемся вниз до предпоследнего уровня ( $N-2$ ) также обновляя списки и устанавливая указатели выбранного элемента в 1.

Более наглядно и понятно работа алгоритма представлена на рисунке ниже ( для размерности множества  $N=5$ ). Номер на рисунке соответствует уровню в описании.

Возможно даже, что кроме рисунка для понимания алгоритма ничего и не надо.



<b>N = 5</b>	номер	указатель	список	номер	указатель	список	номер	указатель	список
состояние	<b>1</b>			<b>2</b>			<b>3</b>		
	1	1	1 2 3 4 5	1	1	1 2 3 4 5	1	1	1 2 3 4 5
	2	1	2 3 4 5	2	1	2 3 4 5	2	1	2 3 4 5
	3	1	3 4 5	3	2	3 4 5	3	3	3 4 5
	4	1	4 5	4	1	3 5	4	1	3 4
результат	1 2 3			1 2 4			1 2 5		
			4 5			3 5			3 4
			5 4			5 3			4 3
состояние	<b>4</b>			<b>5</b>			<b>6</b>		
	1	1	1 2 3 4 5	1	1	1 2 3 4 5	1	1	1 2 3 4 5
	2	2	2 3 4 5	2	2	2 3 4 5	2	2	2 3 4 5
	3	1	2 4 5	3	2	2 4 5	3	3	2 4 5
	4	1	4 5	4	1	2 5	4	1	2 4
результат	1 3 2			1 3 4			1 3 5		
			4 5			2 5			2 4
			5 4			5 2			4 2
состояние	<b>7</b>			<b>8</b>			<b>9</b>		
	1	1	1 2 3 4 5	1	1	1 2 3 4 5	1	1	1 2 3 4 5
	2	3	2 3 4 5	2	3	2 3 4 5	2	3	2 3 4 5
	3	1	2 3 5	3	2	2 3 5	3	3	2 3 5
	4	1	3 5	4	1	2 5	4	1	2 3
результат	1 4 2			1 4 3			1 4 5		
			3 5			2 5			2 3
			5 3			5 2			3 2
...									
состояние	<b>13</b>						<b>(N! / 2) = 60 ( последнее )</b>		
	1	2	1 2 3 4 5				1	5	1 2 3 4 5
	2	1	1 3 4 5				2	4	1 2 3 4
	3	1	3 4 5				3	3	1 2 3
	4	1	4 5				4	1	1 2
результат	2 1 3						5 4 3		
			4 5						1 2
			5 4						2 1

Конечно, при реализации алгоритма можно было использовать и обычный двухмерный массив, тем более что для небольших N выигрыш объема памяти ничего не дает, а на больших N мы можем не дожидаться окончания работы алгоритма.

Один из способов реализации алгоритма на VBA ниже. Для его запуска можно создать книгу Excel с макросами, создать модуль на вкладке разработчик VB и скопировать текст в модуль. После запуска generate() на Лист1 будут выведены все перестановки.

## VBA для Excel

```

Option Explicit
Type dtree
    tek_elem_ukaz As Integer
    spisok() As Integer
End Type
Dim masiv() As dtree
Dim start_print As Integer
Dim N As Integer

Sub generate()
    Dim ii As Integer, kk As Integer, jj As Integer
    Dim uroven As Integer

    Лист1.Cells.Clear
    N = 5
    start_print = 1

```

```

ReDim masiv(N - 1)
' первичное заполнение
For ii = 1 To N - 1
    masiv(ii).tek_elem_ukaz = 1
    ReDim masiv(ii).spisok(N + 1 - ii)
    For kk = 1 To (N + 1 - ii)
        masiv(ii).spisok(kk) = kk + ii - 1
    Next
Next
uroven = N - 2
Do
    ' результат
    Call print_rezult(uroven)
    ' на последнем уровне можно сдвинуться вправо
    If masiv(uroven).tek_elem_ukaz <= (N - uroven) Then
        ' делаем шаг вправо
        ' меняем тек элемент
        masiv(uroven).tek_elem_ukaz = masiv(uroven).tek_elem_ukaz + 1
        ' меняем массив снизу
        Call zap_niz(uroven)
    Else
        ' делаем шаг вверх до первого уровня, где можно сдвинуться вправо
        Do While uroven > 1 And masiv(uroven).tek_elem_ukaz > (N - uroven)
            uroven = uroven - 1
        Loop
        If uroven = 1 And masiv(1).tek_elem_ukaz = N Then
            MsgBox "stop calc"
            Exit Sub ' напечатали все
        End If
        ' делаем шаг вправо на первом снизу доступном уровне
        masiv(uroven).tek_elem_ukaz = masiv(uroven).tek_elem_ukaz + 1
        Call zap_niz(uroven)
        ' заполнение нижних уровней
        Do While uroven < N - 2
            uroven = uroven + 1
            masiv(uroven + 1).tek_elem_ukaz = 1
            ' меняем массив снизу
            For kk = 2 To N - uroven + 1
                masiv(uroven + 1).spisok(kk - 1) = masiv(uroven).spisok(kk)
            Next
        Loop
    End If
Loop
End Sub

Sub print_rezult(ukaz As Integer)
Dim ii As Integer
For ii = 1 To ukaz
    With masiv(ii)
        Лист1.Cells(start_print, ii) = .spisok(.tek_elem_ukaz)
        Лист1.Cells(start_print + 1, ii) = .spisok(.tek_elem_ukaz)
    End With
Next
    With masiv(ukaz + 1)
        Лист1.Cells(start_print, ukaz + 1) = .spisok(1)
        Лист1.Cells(start_print, ukaz + 2) = .spisok(2)
        start_print = start_print + 1
        Лист1.Cells(start_print, ukaz + 1) = .spisok(2)
        Лист1.Cells(start_print, ukaz + 2) = .spisok(1)
        start_print = start_print + 1
    End With
End Sub

Sub zap_niz(ukaz As Integer)
' заполнение нижнего уровня
Dim ii As Integer, wsp1 As Integer
' меняем тек элемент
wsp1 = masiv(ukaz).tek_elem_ukaz
masiv(ukaz + 1).tek_elem_ukaz = 1
' меняем массив снизу
For ii = 1 To wsp1 - 1

```

```
        masiv(ukaz + 1).spisok(ii) = masiv(ukaz).spisok(ii)
    Next
    For ii = wsp1 + 1 To N - ukaz + 1
        masiv(ukaz + 1).spisok(ii - 1) = masiv(ukaz).spisok(ii)
    Next
End Sub
```

**Ссылки:**

[1] В.Липский. Комбинаторика для программистов. -Москва, издательство Мир, 1988.

## 5. Варианты для задания №1

№1

$i$	$a_i$	$b_i$
1	10	2
2	7	3
3	5	1
4	2	6
5	3	7

№2

$i$	$a_i$	$b_i$
1	12	2
2	7	4
3	4	2
4	2	8
5	1	7

№3

$i$	$a_i$	$b_i$
1	13	3
2	8	2
3	5	3
4	3	6
5	4	7

№4

$i$	$a_i$	$b_i$
1	9	3
2	7	4
3	6	3
4	2	6
5	3	6

№5

$i$	$a_i$	$b_i$
1	14	2
2	7	4
3	4	1
4	1	6
5	3	8

№6

$i$	$a_i$	$b_i$
1	10	4
2	2	5
3	5	3
4	3	6
5	4	8

№7

$i$	$a_i$	$b_i$
1	4	2
2	7	4
3	1	5
4	2	6
5	3	7

№8

$i$	$a_i$	$b_i$
1	8	3
2	7	5
3	3	4
4	2	6
5	1	7

№9

$i$	$a_i$	$b_i$
1	12	4
2	7	2
3	5	8
4	2	7
5	3	9

№10

$i$	$a_i$	$b_i$
1	10	4
2	5	2
3	6	5
4	3	7
5	1	4

№11

$i$	$a_i$	$b_i$
1	8	2
2	7	4
3	6	1
4	2	8
5	4	7

№12

$i$	$a_i$	$b_i$
1	7	3
2	8	4
3	4	1
4	2	9
5	2	7

№ 13

$i$	$a_i$	$b_i$
1	12	3
2	9	4
3	4	2
4	1	7
5	4	8

№14

$i$	$a_i$	$b_i$
1	9	5
2	7	1
3	4	2
4	3	4
5	2	8

№15

$i$	$a_i$	$b_i$
1	15	3
2	9	5
3	6	2
4	3	5
5	4	9

№ 16

$i$	$a_i$	$b_i$
1	11	3
2	12	4
3	6	4
4	2	7
5	1	9

№17

$i$	$a_i$	$b_i$
1	6	2
2	7	3
3	3	5
4	1	7
5	2	6

№18

$i$	$a_i$	$b_i$
1	10	3
2	7	4
3	2	6
4	3	7
5	1	4

№19

$i$	$a_i$	$b_i$
1	11	5
2	8	3
3	3	8
4	4	7
5	2	8

№20

$i$	$a_i$	$b_i$
1	11	2
2	8	1
3	7	3
4	2	9
5	5	5

№21

$i$	$a_i$	$b_i$
1	12	6
2	8	3
3	3	4
4	6	7
5	1	9

№22

$i$	$a_i$	$b_i$
1	10	5
2	7	4
3	3	6
4	5	7
5	2	9

№23

$i$	$a_i$	$b_i$
1	10	5
2	6	7
3	5	6
4	3	9
5	2	8

№24

$i$	$a_i$	$b_i$
1	8	4
2	4	9
3	7	3
4	5	12
5	11	5

№25

$i$	$a_i$	$b_i$
1	8	4
2	5	6
3	7	3
4	5	12
5	10	5

№26

$i$	$a_i$	$b_i$
1	6	4
2	5	6
3	7	2
4	5	12
5	10	7

№27

$i$	$a_i$	$b_i$
1	9	4
2	6	6
3	7	3
4	5	14
5	10	5

## 6. Варианты для задания №2

№1

$i$	$a_i$	$b_i$	$c_i$
1	10	2	4
2	7	3	5
3	5	1	7
4	2	4	5
5	3	3	10

№2

$i$	$a_i$	$b_i$	$c_i$
1	12	2	2
2	7	4	6
3	4	2	7
4	6	3	4
5	5	2	8

№3

$i$	$a_i$	$b_i$	$c_i$
1	13	3	7
2	8	2	9
3	9	3	6
4	7	6	3
5	6	5	4

№4

$i$	$a_i$	$b_i$	$c_i$
1	9	7	9
2	6	4	14
3	7	3	11
4	3	5	10
5	2	8	8

№5

$i$	$a_i$	$b_i$	$c_i$
1	14	2	8
2	7	4	9
3	4	1	11
4	5	6	10
5	3	8	12

№6

$i$	$a_i$	$b_i$	$c_i$
1	10	4	8
2	2	5	12
3	3	3	11
4	5	6	9
5	4	8	10

№7

$i$	$a_i$	$b_i$	$c_i$
1	10	5	6
2	7	4	8
3	9	3	5
4	8	6	11
5	7	7	12

№8

$i$	$a_i$	$b_i$	$c_i$
1	8	3	7
2	7	5	9
3	4	4	8
4	3	6	11
5	2	7	12

№9

$i$	$a_i$	$b_i$	$c_i$
1	12	4	9
2	7	2	11
3	5	8	10
4	4	7	14
5	6	9	12

№10

$i$	$a_i$	$b_i$	$c_i$
1	10	4	7
2	5	3	9
3	7	5	8
4	3	7	12
5	4	6	11

№11

$i$	$a_i$	$b_i$	$c_i$
1	8	2	8
2	6	1	10
3	7	4	9
4	4	8	11
5	2	7	12

№12

$i$	$a_i$	$b_i$	$c_i$
1	7	3	7
2	8	4	9
3	4	6	8
4	2	2	12
5	3	7	10

№13

$i$	$a_i$	$b_i$	$c_i$
1	12	3	9
2	9	4	8
3	4	5	10
4	3	7	12
5	5	8	11

№14

$i$	$a_i$	$b_i$	$c_i$
1	9	5	8
2	7	3	10
3	4	2	9
4	5	4	12
5	2	8	11

№15

$i$	$a_i$	$b_i$	$c_i$
1	15	3	8
2	9	5	12
3	10	7	6
4	9	5	7
5	11	9	13

№16

$i$	$a_i$	$b_i$	$c_i$
1	11	3	4
2	12	5	3
3	9	2	6
4	8	7	9
5	7	4	10

№17

$i$	$a_i$	$b_i$	$c_i$
1	6	2	8
2	1	7	9
3	3	5	7
4	7	3	12
5	2	6	11

№18

$i$	$a_i$	$b_i$	$c_i$
1	10	3	6
2	7	5	9
3	2	4	12
4	3	2	8
5	1	6	7

№19

$i$	$a_i$	$b_i$	$c_i$
1	11	5	9
2	7	3	8
3	5	8	12
4	4	7	11
5	2	4	10

№20

$i$	$a_i$	$b_i$	$c_i$
1	11	6	7
2	8	4	9
3	9	3	8
4	10	2	12
5	5	5	10

№21

$i$	$a_i$	$b_i$	$c_i$
1	12	5	7
2	7	3	8
3	5	6	12
4	3	7	11
5	2	4	10

№22

$i$	$a_i$	$b_i$	$c_i$
1	10	6	6
2	7	5	8
3	9	3	7
4	11	1	11
5	5	5	9

№23

$i$	$a_i$	$b_i$	$c_i$
1	14	3	11
2	6	5	8
3	7	2	9
4	8	6	10
5	10	3	12

№24

$i$	$a_i$	$b_i$	$c_i$
1	13	5	11
2	6	3	9
3	7	4	9
4	8	2	10
5	11	6	8

№25

$i$	$a_i$	$b_i$	$c_i$
1	14	3	8
2	6	6	8
3	7	2	9
4	8	6	10
5	10	3	12

№26

$i$	$a_i$	$b_i$	$c_i$
1	13	4	11
2	6	3	9
3	7	4	9
4	9	3	10
5	11	6	9



## 7. Вопросы по данной теме

1. Какие показатели производственного процесса можно выделить при решении задачи упорядочения?
2. В чем состоит задача упорядочения, исследованная Джонсоном (задача Джонсона)?
3. Что является целевой функцией (оптимизируемым показателем) и оптимизирующей переменной в задаче Джонсона?
4. Какие ограничения имеются в задаче Джонсона и на что они влияют?
5. Откуда выводятся правила Джонсона?
6. Каким образом получается алгоритм Джонсона?
7. В чем суть алгоритма Джонсона?
8. Из каких величин состоит совокупная длительность производственного цикла ( время окончания обработки последней детали на последнем станке)?
9. Как получить математическую модель задачи Джонсона для 2-х станков?
10. Как получить математическую модель задачи Джонсона для 3-х станков?
11. Какие имеются способы определения простоя 2-го станка в задаче Джонсона?
12. Какие имеются способы определения простоя 3-го станка в задаче Джонсона?
13. Какой способ нахождения решения для задачи Джонсона можно предложить помимо использования алгоритма Джонсона?
14. Сколько вариантов последовательностей запуска может быть в задаче Джонсона для 2-х и 3-х станков?
15. Каким образом решается задача Джонсона для 3-х станков?
16. На что влияет условие  $(\min a_i \geq \max b_j)$  или  $(\min c_k \geq \max b_j)$  в задаче Джонсона для 3-х станков?
17. За счет чего происходит сведение задачи упорядочения с тремя станками к задаче упорядочения с двумя станками?
18. Объясните, почему формула вычисления простоя на третьем станке в общем виде

$$Y = \max_{1 \leq u \leq v \leq n} (K(u) + H(v))$$

преобразуется к формуле

$$Y = \max_{1 \leq u \leq n} (K(u) + H(u)) \quad ?$$

19. Почему в общем случае трудоемкость (время) обработки детали не может быть равна нулю?

## **8. Список литературы**

1. Сорокин А.В. Использование алгоритма Джонсона для решения задачи упорядочения. Методические указания к лабораторной работе по дисциплине «Основы моделирования» / А.В. Сорокин; Алт. госуд. технич. ун-т им. И.И. Ползунова.. - Барнаул, 2021. – 22 с.
2. Липский, В. Комбинаторика для программистов. - Москва, издательство Мир, 1988.