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

Отчет защищен с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель      А. В. Сорокин

(подпись) (и.о.фамилия)

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

дата

Отчет

по дисциплине

ОСНОВЫ МОДЕЛИРОВАНИЯ

лабораторная работа №1

Модели Джонсона задач упорядочения nx2 и nx3

название работы

ЛР 09.03.04.05.001 О

обозначение документа

Студент группы гр. ПИ-92                                              И. И. Шинтяпин

*(подпись*)

Преподаватель доцент, к.т.н.         А. В. Сорокин

должность, ученое звание и.о., фамилия

БАРНАУЛ 2022

# **Аннотация**

В данной лабораторной работе были рассмотрены задачи упорядочения nx2 и nx3 из теории расписаний. Были изучены различные модели и алгоритмы нахождения оптимального решения. Также был изучен вопрос о полном переборе вариантов решения задач упорядочения для поиска оптимального решения, в случае не выполнения необходимого условия сведения задачи nx2 к nx3. Кроме того, был построен график Ганта.

**Оглавление**

[**Аннотация** 2](#_Toc114663704)

[**Оглавление** 3](#_Toc114663705)

[**Задание 1. Задача упорядочивания nx2** 4](#_Toc114663706)

[**Решение задачи задания 1** 5](#_Toc114663707)

[**Алгоритм решения:** 5](#_Toc114663708)

[**Код программы на языке С++** 6](#_Toc114663709)

[**Подключенные библиотеки:** 6](#_Toc114663710)

[**Код описания классов Table и nx2:Table** 6](#_Toc114663711)

[**Код сортировки массива:** 8](#_Toc114663712)

[**Код вывода таблицы:** 9](#_Toc114663713)

[**Код вывода Диаграммы Ганта:** 9](#_Toc114663714)

[**Код Main:** 10](#_Toc114663715)

[**Результаты выполнения задания №1** 11](#_Toc114663716)

[**Задание 2. Задача упорядочивания nx3** 12](#_Toc114663717)

[**Решение задачи задания 2** 13](#_Toc114663718)

[**Алгоритм решения:** 13](#_Toc114663719)

[**Код программы на языке С++** 14](#_Toc114663720)

[**Код описания классов Table и nx3:Table** 14](#_Toc114663721)

[**Код сортировки массива:** 16](#_Toc114663722)

[Описан в реализации 1-го задания 16](#_Toc114663723)

[**Код проверки условия и формирования нового массива** 16](#_Toc114663724)

[**Код вывода таблицы:** 17](#_Toc114663725)

[**Код вывода Диаграммы Ганта:** 18](#_Toc114663726)

[**Код Main:** 19](#_Toc114663727)

[**Результаты выполнения задания №2** 20](#_Toc114663728)

[**Задание 3. Задача упорядочивания nx3 решаемая методом перебора** 21](#_Toc114663729)

[**Решение задачи задания 3** 22](#_Toc114663730)

[**Алгоритм решения:** 22](#_Toc114663731)

[**Код программы на языке С++** 22](#_Toc114663732)

[**Код описания классов Table и nx3:Table** 23](#_Toc114663733)

[**Код метода перестановок и вычисления оптимальной перестановки:** 25](#_Toc114663734)

[**Метод нахождения времени работы:** 25](#_Toc114663735)

[**Код вывода таблицы:** 26](#_Toc114663736)

[**Код вывода Диаграммы Ганта:** 27](#_Toc114663737)

[**Код Main:** 28](#_Toc114663738)

[**Результаты выполнения задания №3** 29](#_Toc114663739)

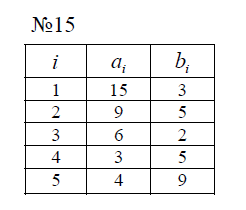
[**Заключение** 30](#_Toc114663740)

[**Список используемых источников** 31](#_Toc114663741)

# **Задание 1. Задача упорядочивания 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).
4. Изучить процесс построения модели задачи Джонсона (nx2) на основе графика Ганта.
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 *n* деталей в обработку на двух станках.
6. Построить график Ганта для оптимальной последовательности запуска деталей своей задачи согласно варианту.
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.

**Вариант 15:**

****

## **Решение задачи задания 1**

### **Алгоритм решения:**

В рамках реализации данной лабораторной работы был создан чисто абстрактный класс Table, хранящий все необходимые данные для реализации алгоритмов для разных задач nx2 и nx3. Данный класс соответствует одной таблице с входными данными. В классе Table присутствует вложенный класс Line\_detail, соответствующий одной строке в таблице, поля: трудоемкость детали a, b и также начальный номер данной строки в массиве. Сам класс Table содержит vector<Line\_detail> list для хранения всех строк таблице.

Изначально происходит считывание данных из файла, затем вывод исходной информации в виде таблицы и диаграммы Ганта, с выводом текущего технологического маршрута, затем в методе sort\_list() в отдельный массив происходит запись четырех переменных: (ai, bi, ai+1, bi+1) затем в методе find\_min() из этих переменных ищется минимальная и если эта переменная равна ai+1 или bi происходит перестановка строк методом swap(). Цикл повторяется по всем элементам массива. В результате получим отсортированную таблицу с оптимальной последовательностью запуска деталей в работу. Далее программа выводит таблицу, оптимальный маршрут и диаграмму Ганта с подсчетом полного времени по формуле:

## **Код программы на языке С++**

Фрагмент, относящийся к Заданию №1.

Все три задачи были реализованы в одном проекте. Если в представленном фрагменте есть код, относящийся к другой задаче, он будет выделен зеленым фоном.

### **Подключенные библиотеки:**

Данный фрагмент относиться ко всем задачам и будет приведён только в задаче №1

#include "stdafx.h"

#include <iostream>

#include <fstream>

#include <string>

#include <Windows.h>

#include <Conio.h>

#include <vector>

### **Код описания классов Table и nx2:Table**

//чисто абстрактный класс таблицы деталей

class Table

{

protected:

class Line\_detail// вложенный класс одной строки таблицы

{

public:

int detail\_a;// трудоемкость детали А

int detail\_b;// трудоемкость детали В

int detail\_c;// трудоемкость детали C (для Nx2 = 0)

int start\_number;// стартовый номер строки деталей

Line\_detail(int a, int b);

Line\_detail()

{

detail\_a = 0;

detail\_b = 0;

detail\_c = 0;

start\_number = 0;

}

Line\_detail::Line\_detail(int a, int b, int number)

{

detail\_a = a;

detail\_b = b;

start\_number = number;

}

};

ifstream \*fin;// указатель на чтение из файла

vector<Line\_detail> list;// массив строк таблицы

void find\_min(int \*array\_comp, int &min\_index);// поиск min индекса среди 4-х эл-ов

int T;// общее время выполнения

// сортировка таблицы для поиска оптимальной последовательности

void sort\_list(vector<Line\_detail> &table);

public:

Table(const string &path);//конструктор

// метод нахождения оптимальной последовательности по заданному алгоритму

virtual void work() = 0;

virtual void print\_table() = 0;// вывод таблицы и текущего маршрута

virtual void print\_Gant() = 0;// вывод диаграммы Ганта для опред таблицы

};

// класс таблицы Nx2, наследник чисто абстрактного класса Table

class nx2 : public Table

{

public:

nx2(const string &path) :Table(path)

{

if (fin->is\_open())

{

fin->close(); // закрываем файл

}

}

// переопределнные методы

void work() override;

void print\_table() override;

void print\_Gant() override;

};

Table::Table(const string &path)

{

T = 0;

Line\_detail \*temp;

string line;

std::auto\_ptr<int> count\_pair(new int);// переменная, необходимая для заполнения таблицы

fin = new ifstream(path);// открываем файл для чтения

if (fin->is\_open())

{

\*fin >> line;

\*count\_pair = atoi(line.c\_str());

// чтение данных из файла

for (int i = 0; i < \*count\_pair; i++)

{

temp = new Line\_detail();

\*fin >> line;

temp->detail\_a = atoi(line.c\_str());

\*fin >> line;

temp->detail\_b = atoi(line.c\_str());

temp->start\_number = list.size() + 1;

list.push\_back(\*temp);

}

}

else

{

cout << "Ошибка открытия файла!" << endl;

\_getch();

exit(1);

}

}

### **Код сортировки массива:**

// сортировка таблицы для поиска оптимальной последовательности

void Table::sort\_list(vector<Line\_detail> &table)

{

int min\_index;

int array\_comp[4];// временный массив для сравнения

// сортировка массива по алгориту Джонсона

for (size\_t i = 0; i < table.size(); i++)

{

for (size\_t j = 0; j < table.size() - i - 1; j++)

{

array\_comp[0] = table.at(j).detail\_a;

array\_comp[1] = table.at(j + 1).detail\_a;

array\_comp[2] = table.at(j + 1).detail\_b;

array\_comp[3] = table.at(j).detail\_b;

find\_min(array\_comp, min\_index);// поиск min индекса среди 4-х

if (min\_index % 2 == 1)// если требуется перестановка

{

swap(table.at(j), table.at(j + 1));

}

}

}

}

// поиск min индекса среди 4-х эл-ов

void Table::find\_min(int \*array\_comp, int &min\_index)

{

int min = array\_comp[0];

min\_index = 0;

for (int i = 1; i < 4; i++)

{

if (min>array\_comp[i])

{

min = array\_comp[i];

min\_index = i;

}

}

}

### **Код вывода таблицы:**

void nx2::print\_table()

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* № \* A \* B \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

int index = 1;

for (auto i = list.begin(); i != list.end(); i++)

{

printf("\*%5d(%d) \*%8d \*%8d \*\n", index, i->start\_number, i->detail\_a, i->detail\_b);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

index++;

}

cout << endl << "S -> ";

for (size\_t i = 0; i < list.size(); i++)

{

cout << i + 1 << "(" << list.at(i).start\_number << "), ";

}

cout << endl;

}

### **Код вывода Диаграммы Ганта:**

void nx2::print\_Gant()

{

cout << "Диаграмма Ганта" << endl;

cout << "A: ";

for (auto i = list.begin(); i != list.end(); i++)

{

for (int j = 1; j <= i->detail\_a; j++)

{

cout << i->start\_number;

}

}

cout << endl << "B: ";

int X = list.begin()->detail\_a;

T = 0;

int sum\_a = X;

int sum\_b = 0;

int sum\_X = 0;

for (auto i = list.begin(); i != list.end(); i++)

{

for (int j = 1; j <= X; j++)

{

cout << "x";

T++;

}

for (int j = 1; j <= i->detail\_b; j++)

{

T++;

cout << i->start\_number;

}

if (i + 1 != list.end())

{

sum\_a += (i + 1)->detail\_a;

sum\_b += (i)->detail\_b;

sum\_X += X;

X = sum\_a - sum\_b - sum\_X;

if (X < 0) X = 0;

}

}

cout << endl << "T = " << T << endl;

}

### **Код Main:**

int \_tmain(int argc, \_TCHAR\* argv[])

{

SetConsoleCP(1251);// подключение русскоязычного ввода/вывода

SetConsoleOutputCP(1251);

system("Color F0");

char choise;

do{

system("cls");

cout << "Выберите кол-во станков:" << endl;

cout << "1. nx2" << endl;

cout << "2. nx3" << endl;

cout << "3. Выход" << endl;

choise = \_getch();

switch (choise)

{

case '1':

count\_nx2();

\_getch(); break;

case '2':

count\_nx3();

\_getch(); break;

default:

break;

}

} while (choise != '3');

return 0;

}

//////////////////nx2//////////////////////

void count\_nx2()

{

system("cls");

auto\_ptr<Table> table\_nx2(new nx2("date\_nx2.txt"));

cout << "Исходные данные:" << endl;

table\_nx2->print\_table();

table\_nx2->print\_Gant();

table\_nx2->work();

cout << endl << "Оптимальная последовательность:" << endl;

table\_nx2->print\_table();

table\_nx2->print\_Gant();

}

## **Результаты выполнения задания №1**

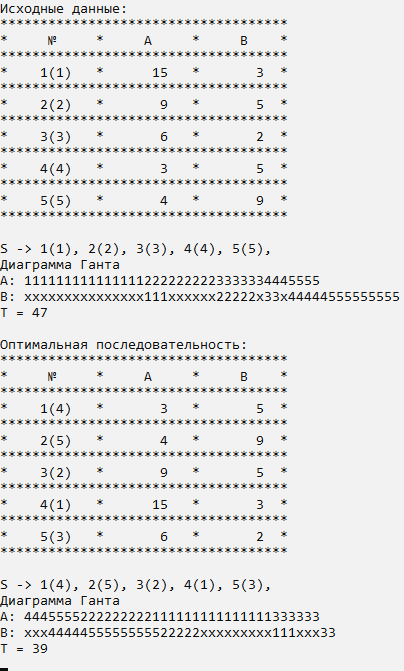
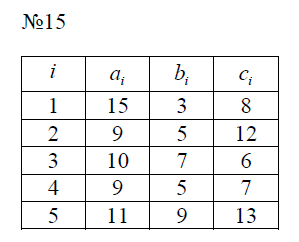


Рис 1. Исходная таблица значений; исходный технологический маршрут, график Ганта, построенный на исходных данных, исходное полное время работы; оптимальная таблица; оптимальный технологический маршрут, график Ганта, оптимальное время обработки деталей.

# **Задание 2. Задача упорядочивания 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.
4. Изучить процесс построения модели задачи Джонсона с (nx3) на основе графика Ганта.
5. Изучить возможности поиска оптимального решения задачи Джонсона для трех станков.

**Вариант 15:**

****

## **Решение задачи задания 2**

### **Алгоритм решения:**

После считывание данных, в методе check\_condition() происходит проверка таблицы nx3 на выполнения условия:

Если истина, то задача сводится к решению алгоритмом Джонсона. Происходит формирование новой таблицы nx2, по правилу:



Далее к сформированной таблице nx2 применяется метод Джонсона, описанный в реализации 1-го задания. После оптимизации таблицы nx2 происходят необходимые перестановки в исходной таблице nx3, чтобы номера строк таблиц nx2 и nx3 совпадали, таким образом получим оптимальное решение, которое затем будет выведено в консоль.

График Ганта в данном случае для первых двух станков выводится аналогично первому заданию, в случае с третьим станком, простои вычисляются по формуле:

## **Код программы на языке С++**

Фрагмент, относящийся к Заданию №2.

### **Код описания классов Table и nx3:Table**

//чисто абстрактный класс таблицы деталей

class Table

{

protected:

class Line\_detail// вложенный класс одной строки таблицы

{

public:

int detail\_a;// трудоемкость детали А

int detail\_b;// трудоемкость детали В

int detail\_c;// трудоемкость детали C (для Nx2 = 0)

int start\_number;// стартовый номер строки деталей

Line\_detail(int a, int b);

Line\_detail()

{

detail\_a = 0;

detail\_b = 0;

detail\_c = 0;

start\_number = 0;

}

Line\_detail::Line\_detail(int a, int b, int number)

{

detail\_a = a;

detail\_b = b;

start\_number = number;

}

};

ifstream \*fin;// указатель на чтение из файла

vector<Line\_detail> list;// массив строк таблицы

void find\_min(int \*array\_comp, int &min\_index);// поиск min индекса среди 4-х эл-ов

int T;// общее время выполнения

// сортировка таблицы для поиска оптимальной последовательности

void sort\_list(vector<Line\_detail> &table);

public:

Table(const string &path);//конструктор

// метод нахождения оптимальной последовательности по заданному алгоритму

virtual void work() = 0;

virtual void print\_table() = 0;// вывод таблицы и текущего маршрута

virtual void print\_Gant() = 0;// вывод диаграммы Ганта для опред таблицы

};

// класс таблицы Nx3, наследник чисто абстрактного класса Table

class nx3 : public Table

{

public:

nx3(const string &path) :Table(path)

{

//std::ifstream fin(path); // окрываем файл для чтения

string line;

if (fin->is\_open())

{

for (auto i = list.begin(); i != list.end(); i++)

{

\*fin >> line;

i->detail\_c = atoi(line.c\_str());

}

}

if (fin->is\_open())

{

fin->close(); // закрываем файл

}

}

// переопределнные методы

void work() override;

void print\_table() override;

void print\_Gant() override;

private:

// проверка необходимого условия для работы алгоритма Джонсона для Nx3

bool check\_condition();

void algorithm\_Johnsona();// реализация алгоритма Джонсона для Nx3

void transposition();// алгоритм перестановок

bool next\_set();// метод реализации перестановок в таблице

// проверка что новая перестановка предпочтительней предыдущей по времени

int check\_time();

};

Table::Table(const string &path)

{

T = 0;

Line\_detail \*temp;

string line;

std::auto\_ptr<int> count\_pair(new int);// переменная, необходимая для заполнения таблицы

fin = new ifstream(path);// открываем файл для чтения

if (fin->is\_open())

{

\*fin >> line;

\*count\_pair = atoi(line.c\_str());

// чтение данных из файла

for (int i = 0; i < \*count\_pair; i++)

{

temp = new Line\_detail();

\*fin >> line;

temp->detail\_a = atoi(line.c\_str());

\*fin >> line;

temp->detail\_b = atoi(line.c\_str());

temp->start\_number = list.size() + 1;

list.push\_back(\*temp);

}

}

else

{

cout << "Ошибка открытия файла!" << endl;

\_getch();

exit(1);

}

}

### **Код сортировки массива:**

### Описан в реализации 1-го задания

### **Код проверки условия и формирования нового массива**

void nx3::work()

{

// проверка необходимого условия для работы алгоритма Джонсона для Nx3

if (check\_condition())

{

cout << "Условие (min\_a >= max\_b) || (min\_c >= max\_b) выполняется" << endl;

cout << "Задача решается методом Джонсона" << endl;

algorithm\_Johnsona();

}

else

{

// решение через метод перестановок

cout << "Условие (min\_a >= max\_b) || (min\_c >= max\_b) не выполняется" << endl;

cout << "Задача решается методом перебора" << endl;

transposition();

}

}

// реализация алгоритма Джонсона для Nx3

void nx3::algorithm\_Johnsona()

{

//формирования новой таблицы через суммирование эл-ов

vector<Line\_detail> list\_1 = list;

for (auto i = list\_1.begin(); i != list\_1.end(); i++)

{

i->detail\_a += i->detail\_b;

i->detail\_b += i->detail\_c;

}

sort\_list(list\_1);

for (size\_t i = 0; i < list.size(); i++)

{

if (list\_1.at(i).start\_number>i + 1)

{

swap(list.at(i), list.at(list\_1.at(i).start\_number - 1));

}

}

}

### **Код вывода таблицы:**

void nx3::print\_table()

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* № \* A \* B \* C \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

int index = 1;

for (auto i = list.begin(); i != list.end(); i++)

{

printf("\*%5d(%d) \*%8d \*%8d \*%8d \*\n", index, i->start\_number, i->detail\_a, i->detail\_b, i->detail\_c);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

index++;

}

cout << endl << "S -> ";

for (size\_t i = 0; i < list.size(); i++)

{

cout << i + 1 << "(" << list.at(i).start\_number << "), ";

}

cout << endl;

}

### **Код вывода Диаграммы Ганта:**

void nx3::print\_Gant()

{

cout << "Диаграмма Ганта" << endl;

cout << "A: ";

for (auto i = list.begin(); i != list.end(); i++)

{

for (int j = 1; j <= i->detail\_a; j++)

{

cout << i->start\_number;

}

}

cout << endl << "B: ";

int X = list.begin()->detail\_a;

T = 0;

int sum\_a = X;

int sum\_b = 0;

int sum\_X = 0;

for (auto i = list.begin(); i != list.end(); i++)

{

for (int j = 1; j <= X; j++)

{

cout << "x";

}

for (int j = 1; j <= i->detail\_b; j++)

{

cout << i->start\_number;

}

if (i + 1 != list.end())

{

sum\_a += (i + 1)->detail\_a;

sum\_b += (i)->detail\_b;

sum\_X += X;

X = sum\_a - sum\_b - sum\_X;

if (X < 0) X = 0;

}

}

cout << endl << "C: ";

int Ta = list.at(0).detail\_a;

X = Ta;

int Tb = list.at(0).detail\_b + X;

int Y = Tb;

T = 0;

for (auto i = list.begin(); i != list.end(); i++)

{

for (int j = 0; j < Y; j++)

{

cout << "Y";

T++;

}

for (int j = 0; j < i->detail\_c; j++)

{

T++;

cout << i->start\_number;

}

if (i + 1 != list.end())

{

Ta += (i + 1)->detail\_a;

X = Ta - Tb;

if (X < 0)X = 0;

Tb += X + (i + 1)->detail\_b;

Y = Tb - T;

if (Y < 0)Y = 0;

}

}

cout << endl << "T = " << T << endl;

}

### **Код Main:**

//////////////////nx3//////////////////////

void count\_nx3()

{

system("cls");

auto\_ptr<Table> table\_nx3(new nx3("date\_nx3.txt"));

cout << "Исходные данные:" << endl;

table\_nx3->print\_table();

table\_nx3->print\_Gant();

table\_nx3->work();

cout << endl << "Оптимальная последовательность:" << endl;

table\_nx3->print\_table();

table\_nx3->print\_Gant();

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

SetConsoleCP(1251);// подключение русскоязычного ввода/вывода

SetConsoleOutputCP(1251);

system("Color F0");

char choise;

do{

system("cls");

cout << "Выберите кол-во станков:" << endl;

cout << "1. nx2" << endl;

cout << "2. nx3" << endl;

cout << "3. Выход" << endl;

choise = \_getch();

switch (choise)

{

case '1':

count\_nx2();

\_getch(); break;

case '2':

count\_nx3();

\_getch(); break;

default:

break;

}

} while (choise != '3');

return 0;

}

## **Результаты выполнения задания №2**

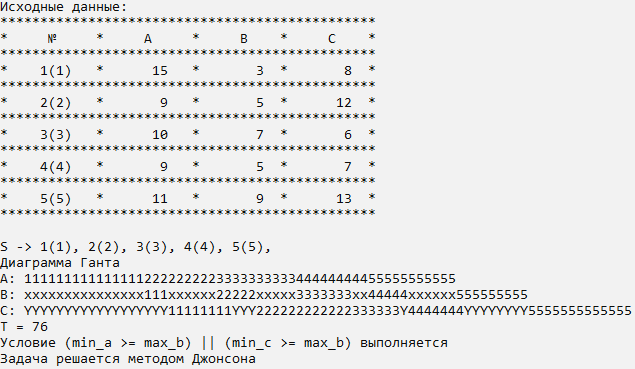


Рис 1. Исходная таблица значений, исходный маршрут, диаграмма Ганта, текущее время работы и проверка условия сводимости

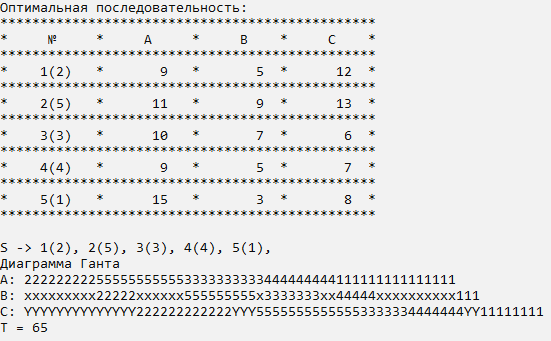
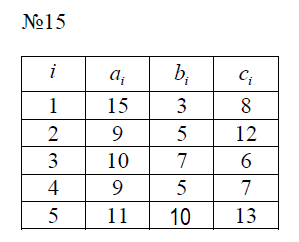


Рис 2. Таблица оптимальных значений, оптимальная последовательность, диаграмма Ганта, оптимальное время работы.

# **Задание 3. Задача упорядочивания nx3 решаемая методом перебора**

Не выполнение условия Джонсона не дает возможности свести задачу упорядочения nx3 к задаче упорядочения nx2. 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.

**Вариант 15:**

****

В данном варианте а5 c 9 заменим на 3, чтобы условие сводимости не выполнялось**.**

## **Решение задачи задания 3**

### **Алгоритм решения:**

Также проверим таблицу nx3 на выполнение условия сводимости к nx2

Если условие не выполняется, то запускается метод transposition() для нахождения оптимального решение через перестановки. Сначала происходит вычисление текущего времени работы в методе check\_time() по формуле, описанной в реализации второго задания и данное время запоминается в переменную. Исходная таблица также запоминается. Затем просходит цикл по перестановкам на каждой итерации которого находится новая перестановка и проверяется значение времени. Если новое время раблоты меньше предыдущего, то новая перестановка и время запоминаются в соответствующие переменные. В конце цикла получим таблицу с оптимальным временем работы. Сами перестановки находятся в методе next\_set(). Суть алгоритма заключается в перестановке строк таблицы всеми возможными способами, общее кол-во перестановок равно n!

Далее, также, как и в прошлых методах происходит вывод оптимальной таблицы, маршрута, диаграммы Ганта и времени работы в консоль.

## **Код программы на языке С++**

Фрагмент, относящийся к Заданию №3.

### **Код описания классов Table и nx3:Table**

//чисто абстрактный класс таблицы деталей

class Table

{

protected:

class Line\_detail// вложенный класс одной строки таблицы

{

public:

int detail\_a;// трудоемкость детали А

int detail\_b;// трудоемкость детали В

int detail\_c;// трудоемкость детали C (для Nx2 = 0)

int start\_number;// стартовый номер строки деталей

Line\_detail(int a, int b);

Line\_detail()

{

detail\_a = 0;

detail\_b = 0;

detail\_c = 0;

start\_number = 0;

}

Line\_detail::Line\_detail(int a, int b, int number)

{

detail\_a = a;

detail\_b = b;

start\_number = number;

}

};

ifstream \*fin;// указатель на чтение из файла

vector<Line\_detail> list;// массив строк таблицы

void find\_min(int \*array\_comp, int &min\_index);// поиск min индекса среди 4-х эл-ов

int T;// общее время выполнения

// сортировка таблицы для поиска оптимальной последовательности

void sort\_list(vector<Line\_detail> &table);

public:

Table(const string &path);//конструктор

// метод нахождения оптимальной последовательности по заданному алгоритму

virtual void work() = 0;

virtual void print\_table() = 0;// вывод таблицы и текущего маршрута

virtual void print\_Gant() = 0;// вывод диаграммы Ганта для опред таблицы

};

// класс таблицы Nx3, наследник чисто абстрактного класса Table

class nx3 : public Table

{

public:

nx3(const string &path) :Table(path)

{

//std::ifstream fin(path); // окрываем файл для чтения

string line;

if (fin->is\_open())

{

for (auto i = list.begin(); i != list.end(); i++)

{

\*fin >> line;

i->detail\_c = atoi(line.c\_str());

}

}

if (fin->is\_open())

{

fin->close(); // закрываем файл

}

}

// переопределнные методы

void work() override;

void print\_table() override;

void print\_Gant() override;

private:

// проверка необходимого условия для работы алгоритма Джонсона для Nx3

bool check\_condition();

void algorithm\_Johnsona();// реализация алгоритма Джонсона для Nx3

void transposition();// алгоритм перестановок

bool next\_set();// метод реализации перестановок в таблице

// проверка что новая перестановка предпочтительней предыдущей по времени

int check\_time();

};

Table::Table(const string &path)

{

T = 0;

Line\_detail \*temp;

string line;

std::auto\_ptr<int> count\_pair(new int);// переменная, необходимая для заполнения таблицы

fin = new ifstream(path);// открываем файл для чтения

if (fin->is\_open())

{

\*fin >> line;

\*count\_pair = atoi(line.c\_str());

// чтение данных из файла

for (int i = 0; i < \*count\_pair; i++)

{

temp = new Line\_detail();

\*fin >> line;

temp->detail\_a = atoi(line.c\_str());

\*fin >> line;

temp->detail\_b = atoi(line.c\_str());

temp->start\_number = list.size() + 1;

list.push\_back(\*temp);

}

}

else

{

cout << "Ошибка открытия файла!" << endl;

\_getch();

exit(1);

}

}

### **Код метода перестановок и вычисления оптимальной перестановки:**

// метод перестановок

void nx3::transposition()

{

int T = check\_time();

int current\_T;

vector<Line\_detail> temp\_table = list;

while (next\_set())

{

current\_T = check\_time();

if (T>current\_T)

{

T = current\_T;

temp\_table = list;

}

}

list = temp\_table;

}

// метод реализации перестановок в таблице

bool nx3::next\_set()

{

int j = list.size() - 2;

while (j != -1 && list.at(j).start\_number >= list.at(j+1).start\_number) j--;

if (j == -1)

return false; // больше перестановок нет

int k = list.size() - 1;

while (list.at(j).start\_number >= list.at(k).start\_number) k--;

swap(list.at(j), list.at(k));

int l = j + 1, r = list.size() - 1; // сортируем оставшуюся часть последовательности

while (l < r)

swap(list.at(l++), list.at(r--));

return true;

}

### **Метод нахождения времени работы:**

// проверка что новая перестановка предпочтительней предыдущей по времени

int nx3::check\_time()

{

int Ta = list.at(0).detail\_a;

int X = Ta;

int Tb = list.at(0).detail\_b + X;

int Y = Tb;

int Tc = Y + list.at(0).detail\_c;

// подсчет текущего времени выполнения

for (auto i = list.begin(); i != list.end(); i++)

{

if (i + 1 != list.end())

{

Ta += (i + 1)->detail\_a;

X = Ta - Tb;

if (X < 0)X = 0;

Tb += X + (i + 1)->detail\_b;

Y = Tb - Tc;

if (Y < 0)Y = 0;

Tc += Y + (i + 1)->detail\_c;

}

}

return Tc;

}

### **Код вывода таблицы:**

void nx3::print\_table()

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* № \* A \* B \* C \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

int index = 1;

for (auto i = list.begin(); i != list.end(); i++)

{

printf("\*%5d(%d) \*%8d \*%8d \*%8d \*\n", index, i->start\_number, i->detail\_a, i->detail\_b, i->detail\_c);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

index++;

}

cout << endl << "S -> ";

for (size\_t i = 0; i < list.size(); i++)

{

cout << i + 1 << "(" << list.at(i).start\_number << "), ";

}

cout << endl;

}

### **Код вывода Диаграммы Ганта:**

void nx3::print\_Gant()

{

cout << "Диаграмма Ганта" << endl;

cout << "A: ";

for (auto i = list.begin(); i != list.end(); i++)

{

for (int j = 1; j <= i->detail\_a; j++)

{

cout << i->start\_number;

}

}

cout << endl << "B: ";

int X = list.begin()->detail\_a;

T = 0;

int sum\_a = X;

int sum\_b = 0;

int sum\_X = 0;

for (auto i = list.begin(); i != list.end(); i++)

{

for (int j = 1; j <= X; j++)

{

cout << "x";

}

for (int j = 1; j <= i->detail\_b; j++)

{

cout << i->start\_number;

}

if (i + 1 != list.end())

{

sum\_a += (i + 1)->detail\_a;

sum\_b += (i)->detail\_b;

sum\_X += X;

X = sum\_a - sum\_b - sum\_X;

if (X < 0) X = 0;

}

}

cout << endl << "C: ";

int Ta = list.at(0).detail\_a;

X = Ta;

int Tb = list.at(0).detail\_b + X;

int Y = Tb;

T = 0;

for (auto i = list.begin(); i != list.end(); i++)

{

for (int j = 0; j < Y; j++)

{

cout << "Y";

T++;

}

for (int j = 0; j < i->detail\_c; j++)

{

T++;

cout << i->start\_number;

}

if (i + 1 != list.end())

{

Ta += (i + 1)->detail\_a;

X = Ta - Tb;

if (X < 0)X = 0;

Tb += X + (i + 1)->detail\_b;

Y = Tb - T;

if (Y < 0)Y = 0;

}

}

cout << endl << "T = " << T << endl;

}

### **Код Main:**

//////////////////nx3//////////////////////

void count\_nx3()

{

system("cls");

auto\_ptr<Table> table\_nx3(new nx3("date\_nx3.txt"));

cout << "Исходные данные:" << endl;

table\_nx3->print\_table();

table\_nx3->print\_Gant();

table\_nx3->work();

cout << endl << "Оптимальная последовательность:" << endl;

table\_nx3->print\_table();

table\_nx3->print\_Gant();

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

SetConsoleCP(1251);// подключение русскоязычного ввода/вывода

SetConsoleOutputCP(1251);

system("Color F0");

char choise;

do{

system("cls");

cout << "Выберите кол-во станков:" << endl;

cout << "1. nx2" << endl;

cout << "2. nx3" << endl;

cout << "3. Выход" << endl;

choise = \_getch();

switch (choise)

{

case '1':

count\_nx2();

\_getch(); break;

case '2':

count\_nx3();

\_getch(); break;

default:

break;

}

} while (choise != '3');

return 0;

}

## **Результаты выполнения задания №3**

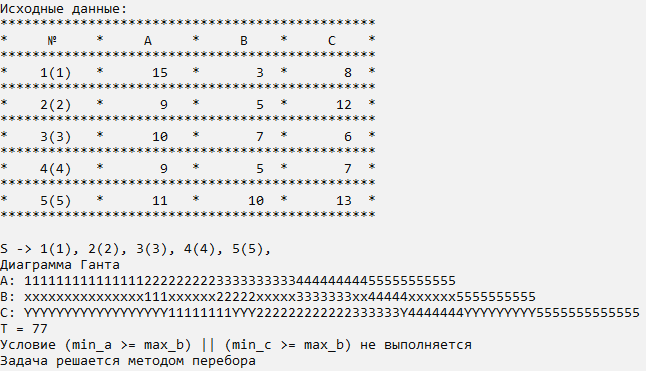


Рис 1. Исходная таблица значений, исходный маршрут, диаграмма Ганта, текущее время работы и проверка условия сводимости

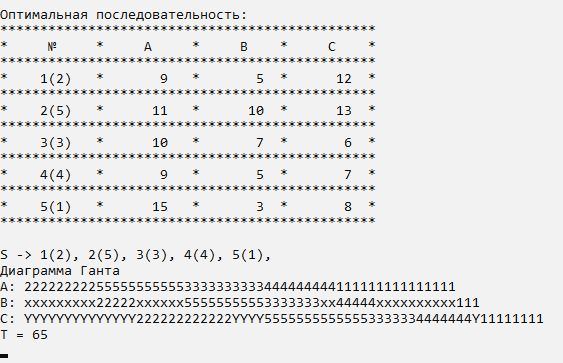


Рис 2. Таблица оптимальных значений, оптимальная последовательность, диаграмма Ганта, оптимальное время работы.

# **Заключение**

В данной лабораторной были рассмотрены задачи упорядочивания nx2 и nx3 из теории расписаний, были изучены соответствующие модели и алгоритмы нахождения оптимального решения. При выполнении лабораторной работы были изучены методы полного перебора для нахождения оптимального решения. Кроме того был реализован вывод последовательности работ в виде диаграммы Ганта для таблиц nx2 и nx3.

В процессе работы возникла проблема построение диаграммы Ганта для таблицы nx3.

# **Список используемых источников**

1. Сорокин А.В. Использование алгоритма Джонсона для решения задачи упорядочения. Методические указания. - Барнаул: АлтГТУ, 2015. – 32 с. – [Электронный ресурс]. – url: http://elib.altstu.ru/eum/download/pm/Sorokin\_alg\_Johnson.pdf
2. Алпатов, Ю.Н. Математическое моделирование производственных процессов [Электронный ресурс] : учеб. пособие / Ю.Н. Алпатов. — Электрон. дан. — Санкт-Петербург : Лань, 2018. — 136 с. — Режим доступа: https://e.lanbook.com/book/107271. — Загл. с экрана.