Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение  
высшего образования   
«Алтайский государственный техничеcкий университет им. И. И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с оценкой \_\_\_\_\_

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Отчет

по лабораторной работе № 3

"Управление проектами. Построение линейной диаграммы проекта"

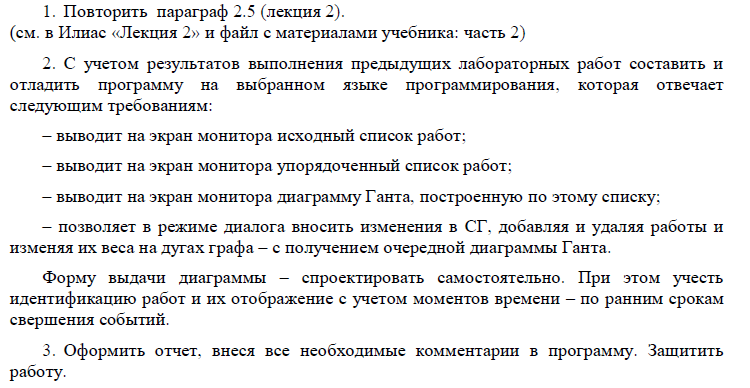
по дисциплине «Разработка и реализация проектов»

Студент группы ПИ-92 Шинтяпин И. И.

Преподаватель доцент, к.э.н. Астахова А.В.

Барнаул 2022

**Задание**



**Решение**

1. **Код программы:**

Файл Graf.h

#pragma once

#include <iostream>

#include <vector>

#include <Conio.h>

#include <fstream>

using namespace std;

struct Table

{

int start\_top;// начальная вершина дуги графа

int end\_top;// конечная вершина дуги графа

int weight;// вес дуги

int number\_sloi;// номер слоя, в котором содержиться конечная вершина дуги

bool metka;// метка вершины, посещена или нет

};

struct Top

{

int number;// шифр события

int tr;// ранний срок свершения события

int number\_sloi;

};

class Graf

{

private:

vector<Table> graf; // вектор, содержащий дуги графа и веса в виде таблицы

vector<Table> struct\_graf;// структурированный СГ

vector<Top> top;// структура, содержащая события и их параметры

int first\_top;// первая вершина графа

int last\_top;// последняя вершина графа

void search\_first\_top(); // поиск первой вершины графа

void graf\_struct();// упорядочивание СГ

void search\_sloi(int \*current\_top, int number\_sloi);// поиск номеров слоев

void input\_top();// нахождение вершин у упорядочивание их в соответствии с номерами слоев

void clear();

public:

Graf(ifstream \*fin);// конструктор

Graf();

void print\_work(const bool flag);// метод печати таблицы путей (работ)

void optimized\_graf();// функция оптимизации (удаления петель, одинаковых работ, поиска конечной вершины СГ)

void search\_tr();// метод поиска раннего срока свершения события

void print\_diagramm();

void push();

void erase();

void change\_weight();

};

Файл Graf.cpp

#include "stdafx.h"

#include "Graf.h"

// конструктор

Graf::Graf(ifstream \*fin)

{

Table \*index = new Table;

// заполение таблицы работ СГ

while (\*fin >> index->start\_top)

{

\*fin >> index->end\_top;

\*fin >> index->weight;

index->metka = false;

index->number\_sloi = 0;

graf.push\_back(\*index);

}

delete index;

// начальные значения первой и послед вершин СГ

first\_top = -500;

last\_top = -500;

}

// метод вывода СГ в консоль

void Graf::print\_work(const bool flag)

{

if (!flag)

{

cout << "Таблица П1:" << endl;

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

printf("\* A \* B \* T \*\n");

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

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

{

printf("\*%10d\*%10d\*%10d\*\n", i->start\_top, i->end\_top, i->weight);

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

}

}

else

{

cout << "Упорядоченный список работ" << endl;

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

printf("\* Номер работы \* A \* B \* T \*\n");

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

for (int i = 0; i <struct\_graf.size(); i++)

{

printf("\*%14d\*%5d\*%5d\*%5d\*\n", i + 1, struct\_graf[i].start\_top, struct\_graf[i].end\_top, struct\_graf[i].weight);

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

}

cout << "\nНачальная вершина: " << first\_top << endl;

cout << "Конечная вершина: " << last\_top << endl << endl;

}

}

void Graf::print\_diagramm()

{

printf("Диаграмма Ганта:\n");

printf(" день: \*%d ", 1);

int count = 10;

int temp=9;

for (int i = 2; i < 129; i++)

{

if (i % temp == 0)

{

printf("%d", count);

count += 10;

if (count == 110)

{

temp--;

printf(" ");

}

}

else

{

printf(" ");

}

}

printf("\n");

for (int i = 0; i < struct\_graf.size(); i++)

{

printf("Работа%4d -%3d\*", struct\_graf[i].start\_top, struct\_graf[i].end\_top);

for (int j = 0; j < top.size(); j++)

{

if (top[j].number == struct\_graf[i].start\_top)

{

for (int k = 0; k < top[j].tr; k++)

{

printf(" ");

}

}

}

for (int j = 0; j < struct\_graf[i].weight; j++)

{

printf("#");

}

printf("\n");

}

}

// функция оптимизации (удаления петель, одинаковых работ, поиска конечной вершины СГ)

void Graf::optimized\_graf()

{

for (int i = 0; i < graf.size(); i++)

{

graf[i].metka = false;

}

search\_first\_top(); // поиск первой вершины графа

// итерация по таблице путей СГ (внешний цикл)

one:

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

{

if (i->start\_top == i->end\_top)// условие нахождения петли

{

cout << "Найдена петля, автоматическое удаление работы" << endl;

graf.erase(i);// удаление записи в таблице

goto one;// возврат в начало цикла

}

two:

for (auto j = graf.begin(); j != graf.end(); j++)// вложенный цикл для сравнения всех вершин СГ

{

if ((i->start\_top == j->start\_top) && (i->end\_top == j->end\_top) && (i != j))// условие дублирования работы

{

cout << "ошибка, работа " << i->start\_top << "->" << i->end\_top << " дублируется" << endl;

if (i->weight != j->weight)// если веса разные

{

cout << "но имеет 2 веса:" << endl;

cout << "1) " << i->weight << endl;

cout << "2) " << j->weight << endl;

cout << "выбрать нужное ( по умолчанию удалиться 2) ): ";

switch (\_getche())

{

case '1': // удаление 1 работы

graf.erase(i);

cout << endl;

goto one;

break;

default:// по умолчанию удаляется 2 работа

graf.erase(j);

cout << endl;

goto two;

break;

}

}

else// если вес совпадает, удаление одной из работ

{

cout << "с одним весом: " << i->weight << endl;

cout << "автоматическое удаление одной работы" << endl;

graf.erase(j);

goto two;

}

}

}

}

// поиск последней вершины СГ

bool flag\_last\_top;// флаг нахождения послед вершины

three:

int size = graf.size();// запоминание кол-во строк таблицы до поиска

for (int i = 0; i < size; i++)// внешний цикл по таблице работ СГ

{

flag\_last\_top = true;// начальное значение флага

for (int j = 0; j < size; j++)// вложенный цикл для сравнения вершин СГ

{

// условие, что вершина не последняя

if ((graf[i].end\_top == graf[j].start\_top) && (graf[i].end\_top != graf[j].end\_top))

{

flag\_last\_top = false;// обнуление флага

break;// завершение цикла

}

}

if (flag\_last\_top && !graf[i].metka)// если найдена последняя вершина

{

if (last\_top == -500)// проверка, первая эта вершина или нет

{

// если вершина первая, запоминание ее шифра и отметка всех работ с этим шифром как пройденные

last\_top = graf[i].end\_top;

for (int temp = 0; temp < graf.size(); temp++)

{

if (graf[temp].end\_top == graf[i].end\_top)

{

graf[temp].metka = true;

}

}

}

else// если уже есть послед вершина

{

cout << "Вершины графа (события) с шифрами " << last\_top << " и " << graf[i].end\_top << " конечные." << endl;

cout << "Ввести фиктивную конечную вершину или, если вершина уже фиктивна, добавить работу?(1-да, 2-нет): ";

char c = \_getche();

cout << endl;

if (c == '1')

{

// создание фиктивной вершины

Table \*index = new Table;

index->start\_top = last\_top;

index->end\_top = 100;

index->weight = 0;

graf.push\_back(\*index);

index->start\_top = graf[i].end\_top;

graf.push\_back(\*index);

delete index;

if (last\_top != 100)

{

last\_top = 100;

}

for (auto temp = 0; temp < graf.size(); temp++)

{

if (graf[temp].end\_top == graf[i].end\_top)

{

graf[temp].metka = true;

}

}

}

else// удаление одной из вершин

{

graf.erase(graf.begin() + i);

last\_top = -500;

for (int i = 0; i < graf.size(); i++)// итерация по таблице вершин графа

{

graf[i].metka = false;

}

goto three;

}

}

}

}

for (int i = 0; i < graf.size(); i++)// итерация по таблице вершин графа

{

graf[i].metka = false;

}

graf\_struct();

}

// метод поиска первой вершины

void Graf::search\_first\_top()

{

bool flag\_search;// флаг, отвечающий за определение начальной вершины

int size = graf.size();

for (auto i = 0; i < size; i++)// итерация по таблице вершин графа

{

flag\_search = true;

for (auto j = 0; j < size; j++)// вторая итерация для поиска первой вершины

{

// условие, что в вершину входит дуга, значит это не стартовая вершина, обнуление флага

if ((graf[i].start\_top == graf[j].end\_top) && (graf[i].start\_top != graf[j].start\_top))

{

flag\_search = false;

break;

}

}

if (flag\_search && !graf[i].metka)

{

// если это первая начальная вершина

if (first\_top == -500)

{

first\_top = graf[i].start\_top;

}

else

{

// создание фиктивной вершины

Table \*index = new Table;

index->start\_top = -100;

index->end\_top = first\_top;

index->weight = 0;

graf.push\_back(\*index);

index->end\_top = graf[i].start\_top;

graf.push\_back(\*index);

delete index;

if (first\_top != -100)

{

first\_top = -100;

}

}

for (auto temp = 0; temp < graf.size(); temp++)

{

if (graf[temp].start\_top == graf[i].start\_top)

{

graf[temp].metka = true;

}

}

}

}

for (auto i = 0; i < graf.size(); i++)// итерация по таблице вершин графа

{

graf[i].metka = false;// обнуление меток

}

}

void Graf::graf\_struct()// метод упорядочивания СГ

{

vector<int> top;// очередь для хранения вершин СГ

top.push\_back(first\_top);// положили в очередь первую вершину

int size = graf.size();// запоминание размера таблицы для оптимизации программы

while (struct\_graf.size() != size)// пока новый СГ не сформирован

{

for (int i = 0; i < size; i++)// цикл по элементам таблицы СГ

{

if (top.at(0) == graf.at(i).start\_top&&!graf.at(i).metka)// если найдена следующая вершина и данная работа не посещена

{

struct\_graf.push\_back(graf.at(i));// добавление текущей строки таблицы в новую таблицу СГ

top.push\_back(graf.at(i).end\_top);// добавление в очередь конечной вершины текущей строки

graf.at(i).metka = true;// работа посещена

}

}

top.erase(top.begin());// после цикла удаление послед вершины

}

}

void Graf::input\_top()

{

for (int i = 0; i < struct\_graf.size(); i++)

{

struct\_graf[i].metka = false;

}

int size = struct\_graf.size();

Top index;// текущая вершина для добавления к таблице вершин

index.number = first\_top;

index.number\_sloi = 1;

top.push\_back(index);

int current\_top = first\_top;

int number\_sloi = 2;

search\_sloi(&current\_top, number\_sloi);// поиск второго слоя

while (current\_top != last\_top)// пока текущая вершина не равна конечной

{

for (int i = 0; i < size; i++)// цикл по элементам таблицы путей

{

if (struct\_graf[i].number\_sloi == number\_sloi)// если найдена вершина с необходимым слоем

{

current\_top = struct\_graf[i].end\_top;// запоминание в текущей новую врешину

if (current\_top == last\_top)

{

break;

}

search\_sloi(&current\_top, number\_sloi + 1);// поиск текущего слоя

}

}

number\_sloi++;// переход к след слою

}

for (int i = 2; i < number\_sloi; i++)// цикл заполнения таблице вершин в соответствии с номерами слоев по возрастанию

{

for (int j = 0; j < struct\_graf.size(); j++)

{

if (struct\_graf[j].number\_sloi == i)

{

index.number = struct\_graf[j].end\_top;

index.number\_sloi = struct\_graf[j].number\_sloi;

top.push\_back(index);

}

}

}

}

void Graf::search\_sloi(int \*current\_top, int number\_sloi)

{

int size = struct\_graf.size();

bool flag;

for (int i = 0; i < size; i++)

{

if (struct\_graf[i].start\_top == \*current\_top)// если найдена дуга с текущим начальным событием

{

struct\_graf[i].metka = true;// дуга посещена

}

}

for (int i = 0; i < size; i++)// цикл по элементам таблице путей

{

flag = true;

if (\*current\_top == struct\_graf[i].start\_top)// если найдена дуга с текущим начальным событием

{

for (int j = 0; j < size; j++)// вложенный цикл по элементам таблице путей

{

if ((struct\_graf[i].end\_top == struct\_graf[j].end\_top) && (i != j) && !struct\_graf[j].metka)// обнуление флага если в вершину входит дуга, кроме текущей

{

flag = false;

}

}

if (flag)

{

struct\_graf[i].number\_sloi = number\_sloi;// если в данную вершину не входят дуги кроме текущей, присваивание ей номера слоя

}

}

}

}

// метод поиска раннего срока свершения события

void Graf::search\_tr()

{

input\_top();// нахождение вершин

top.at(0).tr = 0;// ранний срок начальной вершины равен 0

int size = top.size();

for (int i = 1; i < size; i++)// цикл по элементам таблице путей

{

top[i].tr = 0;// присваивание каждой вершине в начале наименьшего срока для сравнения

for (int j = 0; j < struct\_graf.size(); j++)// цикл по таблице работ

{

if (top[i].number == struct\_graf[j].end\_top)//если найдена след работа

{

for (int k = 0; k < size; k++)// вложенный цикл по вершинам

{

if (top[k].number == struct\_graf[j].start\_top&&top[i].tr < top[k].tr + struct\_graf[j].weight)// условие находения стартовой вершины текущей работы и проверка суммы

{

top[i].tr = top[k].tr + struct\_graf[j].weight;// присваиваение нового раннего срока, меньше предыдущего

break;

}

}

}

}

}

}

// добавление работы в таблицу

void Graf::push()

{

Table index;

system("cls");

print\_work(1);// печать таблицы работ

cout << "Добавление работы" << endl;

cout << "Введите данные:" << endl;

cout << "Начальное событие: ";

cin >> index.start\_top;

cout << "Конечное событие: ";

cin >> index.end\_top;

cout << "Время работы: ";

cin >> index.weight;

struct\_graf.push\_back(index);// добавление новой работы

clear();

cout << "Работа успешно добавлена";

\_getch();

}

// удаление работы из таблицы

void Graf::erase()

{

int number;

system("cls");

print\_work(1);

cout << "Удаление работы" << endl;

cout << "Введите номер работы:" << endl;

cin >> number;

number--;

struct\_graf.erase(struct\_graf.begin() + number);// удаление нужной работы

clear();

cout << "Работа успешно удалена";

\_getch();

}

// замена времени работы

void Graf::change\_weight()

{

int number;

system("cls");

print\_work(1);

cout << "Изменение времени работы" << endl;

cout << "Введите номер работы:" << endl;

cin >> number;

number--;

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

printf("\* Номер работы \* A \* B \* T \*\n");

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

printf("\*%14d\*%5d\*%5d\*%5d\*\n", number + 1, struct\_graf[number].start\_top, struct\_graf[number].end\_top, struct\_graf[number].weight);

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

cout << "Введите новое время: ";

cin >> struct\_graf[number].weight;// ввод нового времени работы

clear();

cout << "Время успешно поменяно";

\_getch();

}

void Graf::clear()

{

graf.clear();// очистка начальной таблицы

for (int i = 0; i < struct\_graf.size(); i++)// заполнение начальной таблицы, таблицей с новыми данными

{

struct\_graf[i].metka = false;// обнуление флага

struct\_graf[i].number\_sloi = 0;// обнуление номера слоя

graf.push\_back(struct\_graf[i]);// добавление работы

}

top.clear();// очистка таблицы событий

struct\_graf.clear();//очистка структурированной таблицы

first\_top = -500;

last\_top = -500;

}

Файл Main.cpp

#include "stdafx.h"

#include <iostream>

#include <Conio.h>

#include "Graf.h"

#include <Windows.h>

using namespace std;

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

{

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

SetConsoleOutputCP(1251);

system("color F0");

ifstream fin("input.dat");// чтение данных из файла input.dat

if (fin.is\_open() == false)

{

cout << "Ошибка открытия файла\nзавершение работы" << endl;

\_getch();

return 1;

}

Graf graf(&fin);// создание и инициализация СГ

char c;

fin.close();

do

{

graf.print\_work(0);// вывод таблицы работ

graf.optimized\_graf();// оптимизация СГ, построение частично структурированной таблицы

graf.search\_tr();// поиск раннего срока свершения события

graf.print\_work(1);// вывод таблицы параметров работ

graf.print\_diagramm();

cout << endl << "Выберите действие:" << endl;

cout << "1) добавить работу" << endl;

cout << "2) удалить работу" << endl;

cout << "3) изменить время" << endl;

cout << "0) выход из программы" << endl;

// выбор действия

switch (c = \_getche())

{

case '1':

graf.push();// добавление работы

break;

case '2':

graf.erase();// удаление работы

break;

case '3':

graf.change\_weight();// замена времени

break;

default:

break;

}

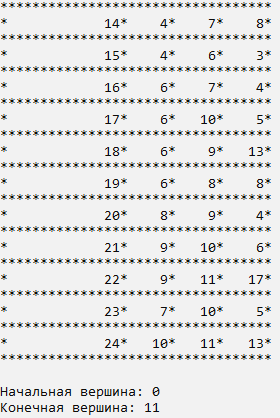
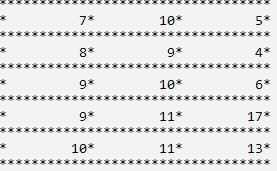
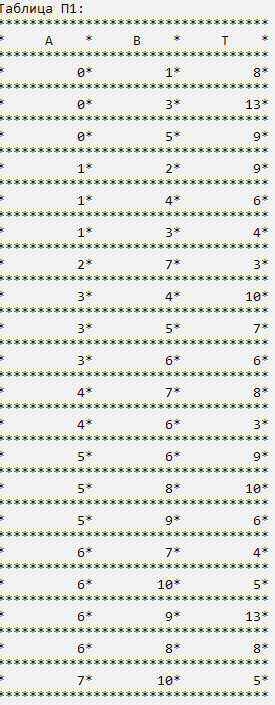
system("cls");

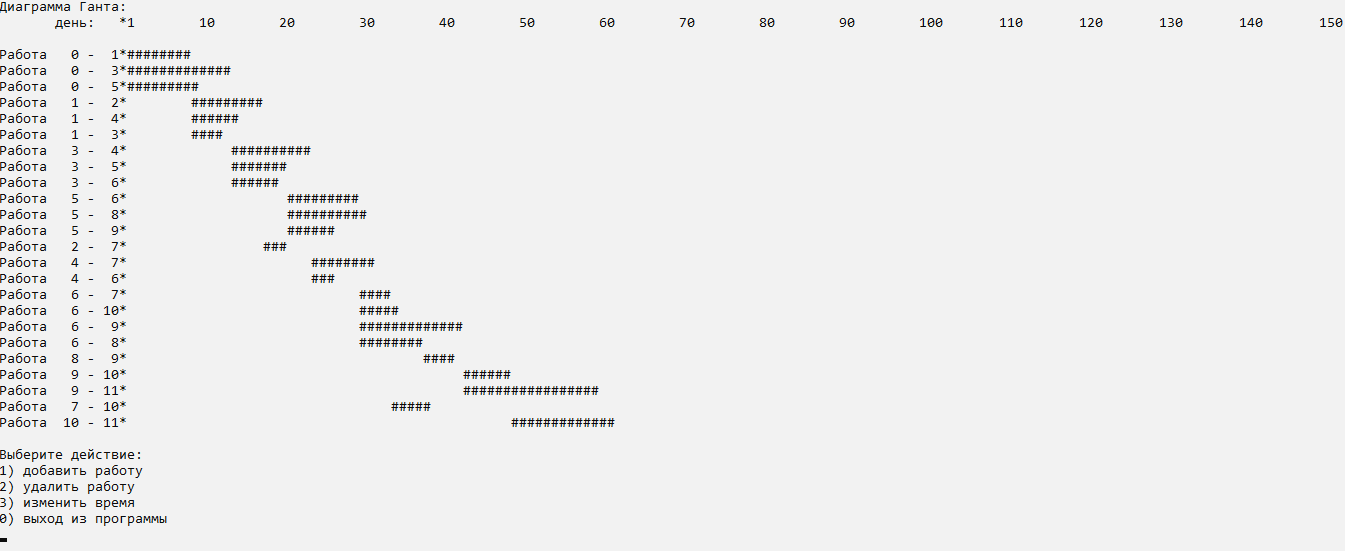
} while (c != '0');// цикл пока не выход из программы

return 0;

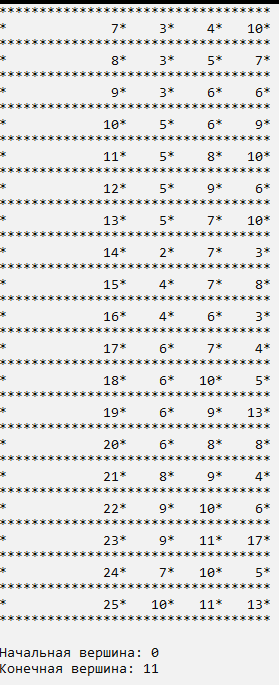
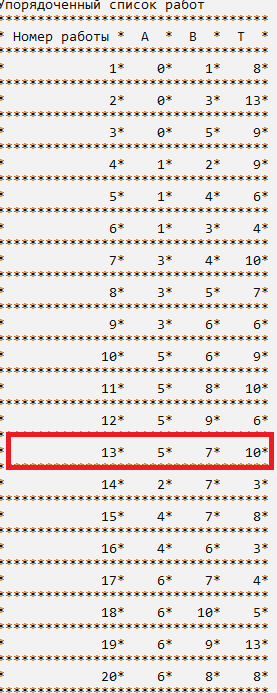
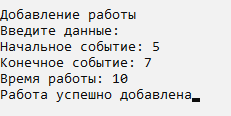
}

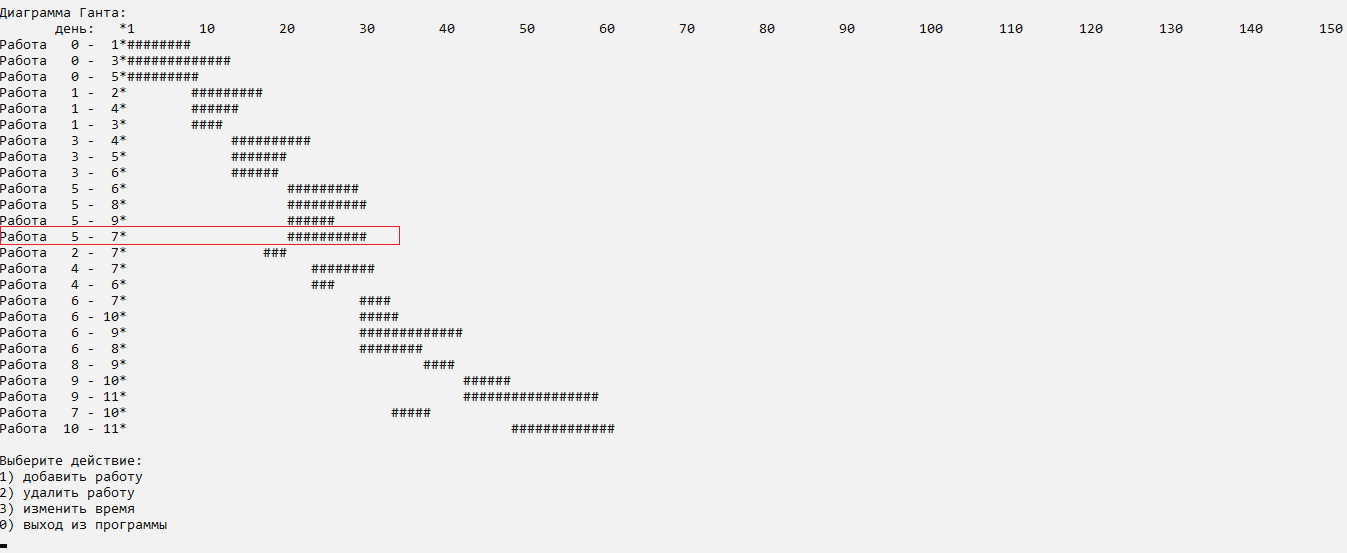
1. **Тесты программы**



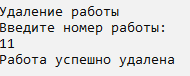


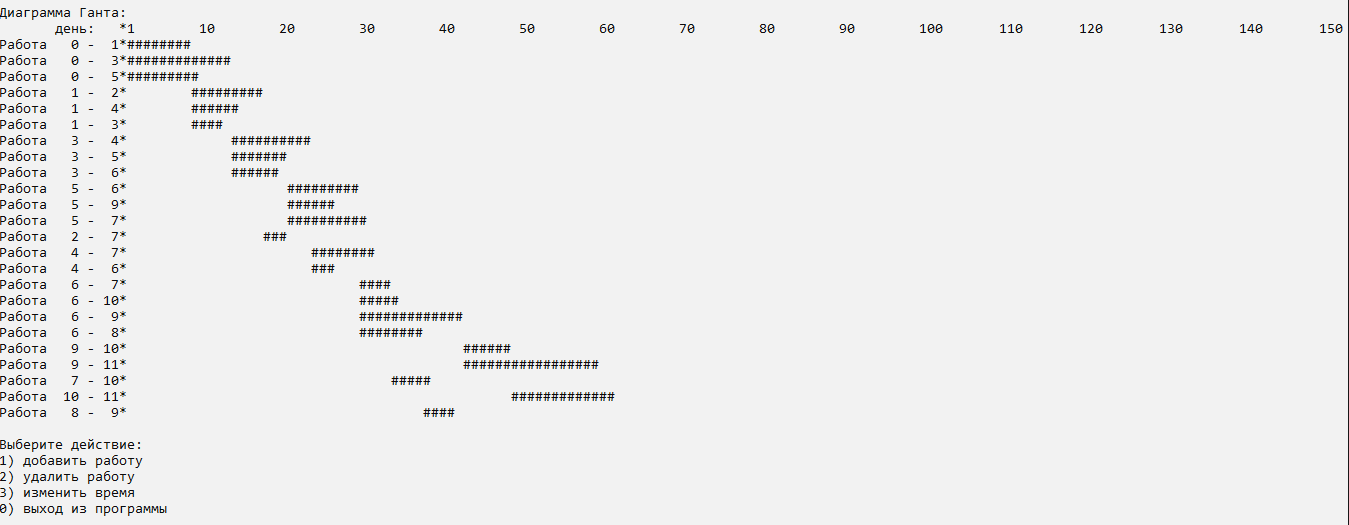
Добавление работы





Удаление работы





Замена времени работы

