

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

“Графы”

Отчет по лабораторной работе №7

По дисциплине

«Структуры и алгоритмы обработки данных в ЭВМ»

Студент гр. 431-3

_____ Д.П. Андреев

« ____ » _____ 2023 г.

Проверил: профессор кафедры АСУ, д.т.н.

_____ А.Н. Горитов

« ____ » _____ 2023 г.

Томск 2023

1.Задание на лабораторную работу

Напишите программу, которая с помощью алгоритма Прима находит минимальное покрывающее дерево для произвольного связного неориентированного графа, заданного списками смежности. Выведите на экран ребра, вошедшие в искомое дерево. Предусмотрите ввод данных из файла. После завершения работы с динамическими структурами данных необходимо освободить занимаемую ими память.

2.Алгоритм решения задачи

Первый шаг — это написание структуры графа. Далее прописываем основные операции инициализации, проверка на пустоту, добавление, удаление, а также дополнительные заполнение из файла и алгоритм Прима. В основной части создаем граф. Заполняем его из файла, после чего передаём в функцию, выполняющую алгоритм Прима. Далее выводим результат на экран.

3.Листинг программы

```
#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

struct Node
{
    int ver1, ver2;
    int weight;//век
    Node* next;
    Node(int _ver1, int _ver2,int _weight)
    {
        ver1 = _ver1;
        ver2 = _ver2;
        weight = _weight;
        next = nullptr;
    }
};
struct List
{
    Node* first;
    Node* last;
    List();//Инициализация
    {
        first = nullptr;
        last = nullptr;
    }
    bool isEmpty()//Проверка на пустоту
    {
        if (first == nullptr)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    void PushBack(int _ver1, int _ver2, int
_weight)//Добавление
    {
        Node* p = new Node(_ver1, _ver2,_weight);

        if (isEmpty() == true)
        {
            first = p;
            last = p;
            return;
        }

        last->next = p;
        last = p;
    }
    void DeleteFirst()//Удаление
    {
        if (isEmpty() == true)
        {
            return;
        }
        Node* p = first;
        first = p->next;
        delete p;
    }
};

struct NodeQ
{
    int data;
    NodeQ* next;
};
struct Queue
{
    NodeQ* firstQ;
    NodeQ* lastQ;

    bool isEmpty(Queue* q)//Проверка очереди на
пустоту
    {
        if (q->firstQ == nullptr)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    void init(Queue* q)//Инициализация очереди
    {
        q->firstQ = nullptr;
        q->lastQ = nullptr;
    }
    List push(Queue* q, List l, int data, bool*
visited)//Вставка нового элемента в очередь
    {
        if (l.first->ver1 == data)
        {
            NodeQ* k = new NodeQ;

            if (k == nullptr)
            {
                cout << "k = nullptr" << endl;
                exit(-1);
            }
            if (q->firstQ == nullptr)
            {
                k->data = 1;
                visited[0] = true;
            }
            else if (visited[l.first->ver2 ] == false)
            {
                if (l.first->weight > l.last->weight)
                {
                    visited[l.last->ver2] = true;
                    k->data = l.last->weight;
                    l.DeleteFirst();
                }
                else
            }
        }
    }
};
```

```

        {
            visited[l.first->ver2] = true;
            k->data = l.first->weight;
            l.DeleteFirst();
        }
    }
    else
    {
        l.DeleteFirst();
        return l;
    }
    k->next = nullptr;
    if (q->firstQ == nullptr)
    {
        q->firstQ = k;
    }
    else
    {
        q->lastQ->next = k;
    }
    q->lastQ = k;

    return l;
}
else
{
    l.DeleteFirst();
    return l;
}
}
int top(Queue* q)//Вернуть первый элемент оче-
реди
{
    if (isEmpty(q) == true)
    {
        cout << "Очередь пуста" << endl;
        exit(0);
    }
    else
    {
        return q->firstQ->data - 1;
    }
}
void pop(Queue* q)
{
    if (isEmpty(q) == 1)
    {
        cout << "Очередь пуста" << endl;
    }
    else
    {
        NodeQ* p = new NodeQ;
        p = q->firstQ;
        q->firstQ = p->next;

        if (q->firstQ == nullptr)
        {
            q->lastQ = nullptr;
        }

        delete p;
    }
}

    }
}
};
List File()
{
    ifstream File("input.txt");
    if (File.is_open())
    {
        List l;
        int ver1, ver2, weight;
        while (!File.eof())
        {
            File >> ver1 >> ver2 >> weight;
            l.PushBack(ver1, ver2, weight);
        }
        return l;
    }
    else
    {
        cout << "Файл не открыт";
        exit(-1);
    }
}

void MST(List graph)
{
    Queue q;
    q.init(&q);
    bool* visited = new bool[5];
    for (int i = 0; i < 5; i++)
    {
        visited[i] = false;
    }
    visited[0] = true;
    int z = 0;
    graph = q.push(&q, graph, z + 1, visited);

    while (!q.isEmpty(&q))
    {
        z = q.top(&q);
        q.pop(&q);
        while (graph.first->next != nullptr)
        {
            if (!visited[graph.first->ver2])
            {
                graph = q.push(&q, graph, z + 1, visited);
            }
        }
    }
}

int main()
{
    setlocale(LC_ALL, "RU");
    List graph=File();
    cout << "Ребра графа, вошедшие в минимальное
стягивающее дерево:" << endl;
    MST(graph);
    return 0;
}

```

4.Пример решения

Входные данные можно увидеть на рисунке 4.1.

```
*input.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
0 1 4
0 2 3
1 2 2
1 3 1
1 4 5
2 3 4
2 4 4
2 4 4
3 4 1
4 5 6
```

Рисунок 4.1— Входные данные из файла input.txt

На следующем рисунке 4.2 можно увидеть граф по введённым данным

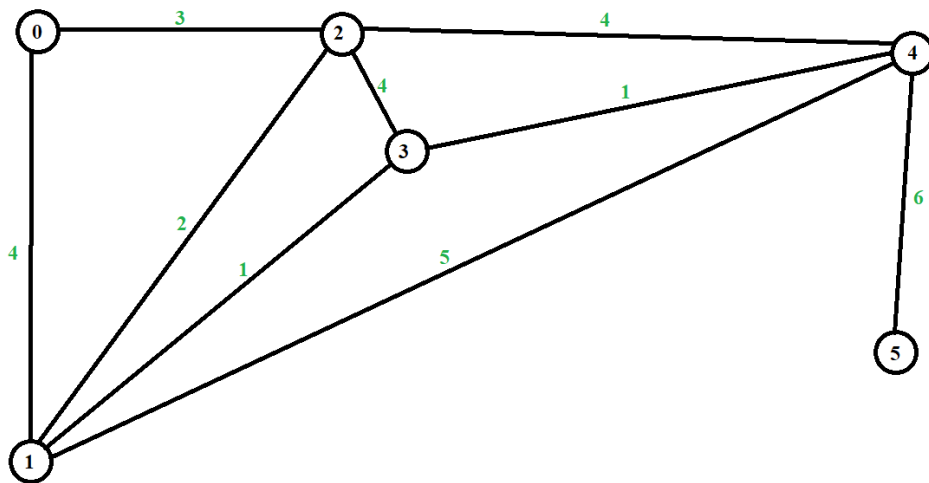
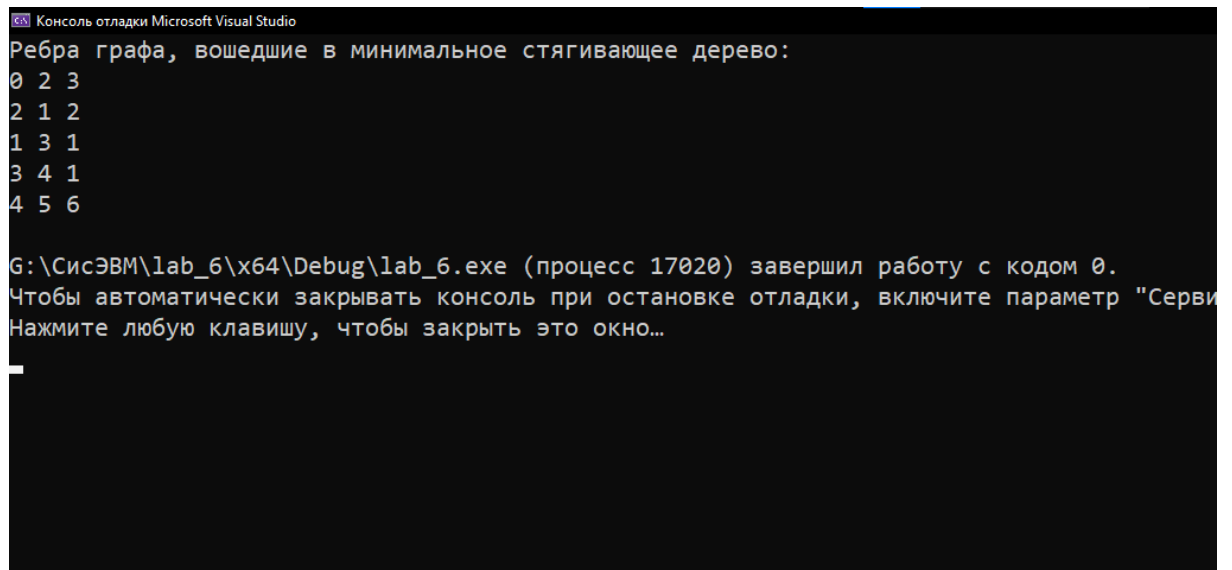


Рисунок 4.2— Полученный граф из входных данных.

Результат программы можно увидеть на рисунке 4.3.



```
Консоль отладки Microsoft Visual Studio
Ребра графа, вошедшие в минимальное стягивающее дерево:
0 2 3
2 1 2
1 3 1
3 4 1
4 5 6

G:\СисЭВМ\lab_6\x64\Debug\lab_6.exe (процесс 17020) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Серви
Нажмите любую клавишу, чтобы закрыть это окно...
```

Рисунок 4.3— Выходные данные после запуска программы

5.Вывод

В результате лабораторной работы были изучены алгоритмы поиска кратчайшего пути в графе, такие как алгоритм Дейкстры, Форда-Беллмана и Флойда-Уоршелла. Так же были изучены алгоритмы нахождения минимального покрывающего дерева.