Министерство образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К курсовому проектированию по курсу «Логика т основа алгоритмизации в инженерных задачах»

На тему «Реализация алгоритма Прима»

Выполнил:
студент группы 21ВВ2
Кривенкова В.С.
Принял:
Юрова О.В

27.12.22
отлично

Пенза 2022

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ _____

«____»_____20__

ЗАДАНИЕ

на курсовое проектирование по курсу

"Логика и основы алгоритмизации в инженерных задачах"

Студенту Кривенковой В.С Группа 22ВВ1.2.

Тема_проекта Реализация алгоритма прима

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программного обеспечения в соответствии с данным заданием курсового проекта.

Пояснительная записка должна содержать:

1. Постановку задачи.
2. Теоретическую часть задания.
3. Описание алгоритма поставленной задачи.
4. Пример ручного расчета задачи и вычислений (на небольшом участке работы алгоритма).
5. Описание самой программы.
6. Тесты
7. Список литературы
8. Листинг программы
9. Результаты работы программы.

Объем работы по курсу

1. Расчетная часть

Ручной расчет работы алгоритма

2. Графическая часть

Схема алгоритма в формате блок-схем

3. Экспериментальная часть

Тестирование программы.
Результаты работы программы на тестовых данных

Срок выполнения проекта по разделам

1 Исследование теоретической части курсового
2 Разработка алгоритмов программы
3 Разработка программы
4 Тестирование и завершение разработки программы
5 Оформление пояснительной записки
6
7
8

Дата выдачи задания "12" 09. 2022

Дата защиты проекта "___" _____

Руководитель _____

Задание получил "12" сентября 2022 г.

Студент Кривенкова В.С.

Содержание

Реферат 5

Введение 6

1. Постановка задачи 7
2. Теоретическая часть задания 8
3. Описание алгоритма программы 9
4. Описание программы 12
5. Тестирование 15
6. Ручной расчёт программы 19

Заключение 20

Список литературы 21

Листинг программы 22

Реферат

Отчет 23 страниц, 10 рисунков

АЛГОРИТМ ПРИМА, ГРАФ, ТЕОРИЯ ГРАФОВ, НЕОРИЕНТИРОВАННЫЙ ГРАФ.

Цель исследования – реализация алгоритма Прима для нахождения минимального остовного дерева.

Достижению данной цели курсовой работы были поставлены и решены следующие задачи: изучить такие основополагающие понятия теории графов, как граф, стоимость графа. Также матрицы смежности и нахождение кратчайшего пути.

Введение

Алгоритм Прима - алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа. Алгоритм впервые был открыт в 1930 году чешским математиком Войцехом Ярником, позже переоткрыт Робертом Примом в 1957 году, и, независимо от них, Э. Дейкстрой в 1959 году. Построение начинается с дерева, включающего в себя одну (произвольную) вершину. В течение работы алгоритма дерево разрастается, пока не охватит все вершины исходного графа. На каждом шаге алгоритма к текущему дереву присоединяется самое лёгкое из рёбер, соединяющих вершину из построенного дерева и вершину не из дерева.

Алгоритм Прима следует общей схеме алгоритма построения минимального остовного дерева: на каждом шаге мы добавляем к строящемуся остову безопасное ребро. Алгоритм Прима относится к группе алгоритмов наращивания минимального остова: на каждом шаге существует не более одной нетривиальной (не состоящей из одной вершины) компоненты связности, и каждый к ней добавляется ребро наименьшего веса, соединяющее вершины компоненты с остальными вершинами. По теореме такое ребро является безопасным.

В качестве среды разработки мною была выбрана среда *MicrosoftVisualStudio2019*, язык программирования – Си.

Целью данной курсовой работы является разработка программы на языке Си, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм Прима.

1. Постановка задачи

Требуется разработать программу построения минимального остовного дерева взвешенного связанного неориентированного графа.

Исходный граф в программе должен задаваться матрицей смежности. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После обработки этих данных на экран должна выводиться матрица смежности. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь.

2. Теоретическая часть задания

Граф Graph (рисунок 1) задается множеством вершин X1, X2, …, Xn и множеством ребер, соединяющих между собой определенные вершины. Ребра из множества A неориентированы, что обозначается линией без стрелки на графе, которая показывает достижимость данной вершины.

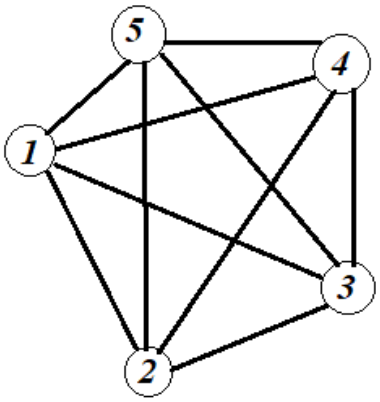

Рисунок 1 – Неориентированный граф

Сам алгоритм имеет очень простой вид. Искомый минимальный остов строится постепенно, добавлением в него рёбер по одному. Изначально остов полагается состоящим из единственной вершины (её можно выбрать произвольно). Затем выбирается ребро минимального веса, исходящее из этой вершины, и добавляется в минимальный остов. После этого остов содержит уже две вершины, и теперь ищется и добавляется ребро минимального веса, имеющее один конец в одной из двух выбранных вершин, а другой - наоборот, во всех остальных, кроме этих двух. И так далее, т.е. всякий раз ищется минимальное по весу ребро, один конец которого - уже взятая в остов вершина, а другой конец - ещё не взятая, и это ребро добавляется в остов (если таких рёбер несколько, можно взять любое). Этот процесс повторяется до тех пор, пока остов не станет содержать все вершины (или, что то же самое, n-1 ребро). В итоге будет построен остов, являющийся минимальным. Если граф был изначально не связен, то остов найден не будет (количество выбранных рёбер останется меньше n-1).

3. Описание алгоритма программы

Для программной реализации алгоритма понадобятся четыре массива:V (int) – массив для отслеживания выбранной вершины (true или false), edge (int)-массив для отслеживания выбранной вершины, G (int) - для возможности динамического ввода, хранения данных.

Имеется граф G. Каждая из вершин, входящая в множество selected, изначально отмечена false, т.е. не посещенная.

В качестве исходного пункта выбирается вершина [0] и ей присваивается, что она посещена: selected [0] = true; затем находятся все соседние вершины в множестве, вычисляется расстояние от вершины, выбранной в начале (нулевой). Если вершина уже находится в множестве, то она отбрасывается, иначе выбираем другую вершину, ближайшую к выбранной вершине на шаге 1. Алгоритм выполняется до тех пор, пока не получится минимальное остовное дерево.

Graph Prima()

1.для i=0 пока i<n делать i=i+1

2. selected[i] = false

3.конец цикла

4. no_edge = 0;

5. selected[0] = true;

6.Пока no_edge< n-1 делать

7. int min = INF;

8. x = 0; y = 0;

9. для i=0 пока i<n делать i=i+1

10. если selected[i]

11. для j=0 пока j<n делать j=j+1

12. если !selected[j] && matr[i][j]

13.если min > matr[i][j]

14. min = matr[i][j];x = i;y = j;

15.конец условия

16.конец условия

17.конец цикла

18.конец условия

19.конец цикла

20.вывод «x – y : matr[x][y]»

21. selected[y] = true;

22. no_edge++

23.конец цикла

24. вывод стоимости дерева

Output()

1.для i=0 пока i<n делать i=i+1

2.для j=0 пока j<n делать j=j+1

3. вывод «matr[i][j]»

4.конец цикла

5. Вывод «\n»

6.конец цикла

4. Описание программы

Для написания данной программы использован язык программирования Си. Язык программирования Си – универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

Данная программа многомодульная, поскольку состоит из нескольких функций: main, Prima, addEdge, toString.

Работа программы начинается с выбора количества вершин. Далее выводится меню. Если пользователь «случайное заполнение матрицы», то выводится матрица смежности, связь ребер и их вес. Если же пользователь выбрал пункт «заполнить самому» ,то на экран выводится запрос ввода двух вершин и вес ребра, по такому принципу заполняется вся матрица . Также предусмотрен выход из программы.

```
int main() {

    setlocale(LC_ALL, "RUS");
    srand(time(NULL));
    int z = 5;
    cout << "Введите кол-во вершин графа: "; cin >> z;
    Graph c(z);

    string s;
    while (true) {
        system("cls");
        cout << "|------------------------|" << endl
            << "|  Количество вершин: " << z << "  |" << endl
            << "|------------------------|" << endl
            << "|          Меню          |" << endl
            << "|------------------------|" << endl
            << "| 1)Случайное заполнение |" << endl
            << "|------------------------|" << endl
            << "| 2)Заполнить самому     |" << endl
            << "|------------------------|" << endl
            << "| 3)Выход                |" << endl
            << "|------------------------|" << endl;

        cout << "       Введите: ";
        cin >> s;
        string q1, q2, q3;
        switch (atoi(s.c_str()))
        {
        case 1:
            system("cls");

            for (int i = 0; i < z; i++)
            {
```

```
            for (int j = i + 1; j < z; j++)
            {
               if (bool(rand() % 2)) {
                  c.addEdge(i, j, rand() % 100);
               }
            }
         }
         cout << "-----------------" << endl << "Матрица смежности: \n";
         //fout << endl << "-----------------" << endl << "Матрица смежности: \n";
         c.toString();
         cout << endl << "-----------------" << endl << "Ответ: \n";
         //fout << endl << "-----------------" << endl << "Ответ: \n";
         c.Prima();

         system("pause");
         break;
      case 2:
         system("cls");
         while (1) {
            system("cls");
            cout << "Для выхода введите *" << endl;
            cout << "Введите 1-вершину: "; cin >> q1;
            if (q1 == "*") break;
            cout << "Введите 2-вершину: "; cin >> q2;
            if (q1 == "*") break;
            cout << "Введите Вес ребра: "; cin >> q3;
            if (q1 == "*") break;
            c.addEdge(atoi(q1.c_str()), atoi(q2.c_str()), atoi(q3.c_str()));
            cout << endl;
            c.toString();
            system("pause");
         }
         system("cls");
         cout << "-----------------" << endl << "Матрица смежности: \n";
         //fout << endl << "-----------------" << endl << "Матрица смежности: \n";
         c.toString();
         cout << endl << "-----------------" << endl << "Ответ: \n";
         //fout << endl << "-----------------" << endl << "Ответ: \n";
         c.Prima();

         system("pause");
         break;
      case 3:
         exit(0);
         break;
      }
      system("cls");
   }

   return 0;
}
```

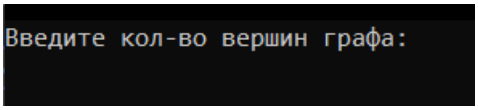

Рисунок 2 – Ввод кол-ва вершин графа.

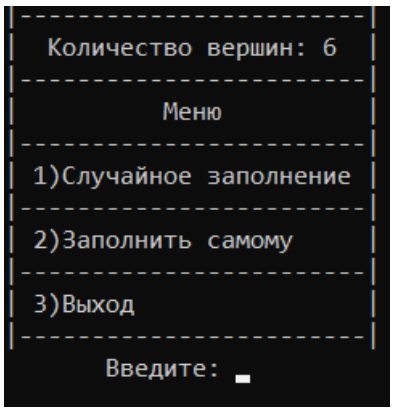

Рисунок 3 – Меню.

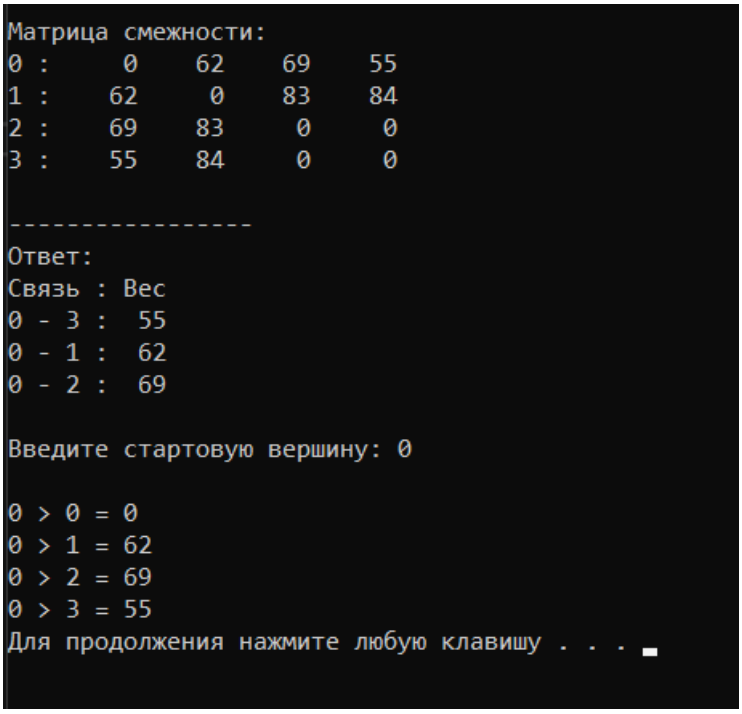

Рисунок 4 – Случайное заполнение матрицы.

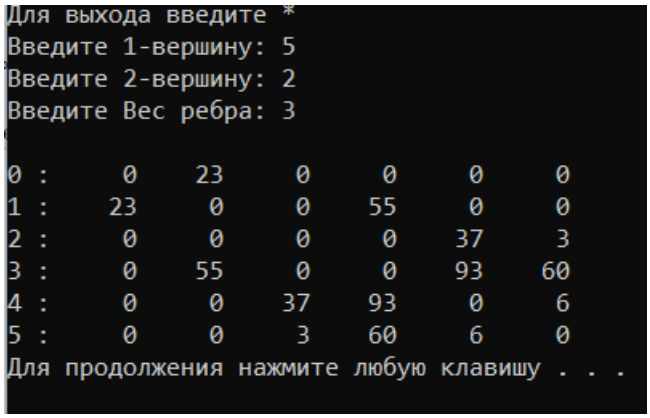

Рисунок 5 – Заполнение матрицы вручную.

5. Тестирование

Среда разработки Microsoft Visual Studio 2019 представляет все средства, необходимые при разработке и отладки многомодульных программ.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество ошибок и проблем, связанных с вводом данных, изменением дизайна вводимых данных, алгоритмом программы, взаимодействием функций.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин.

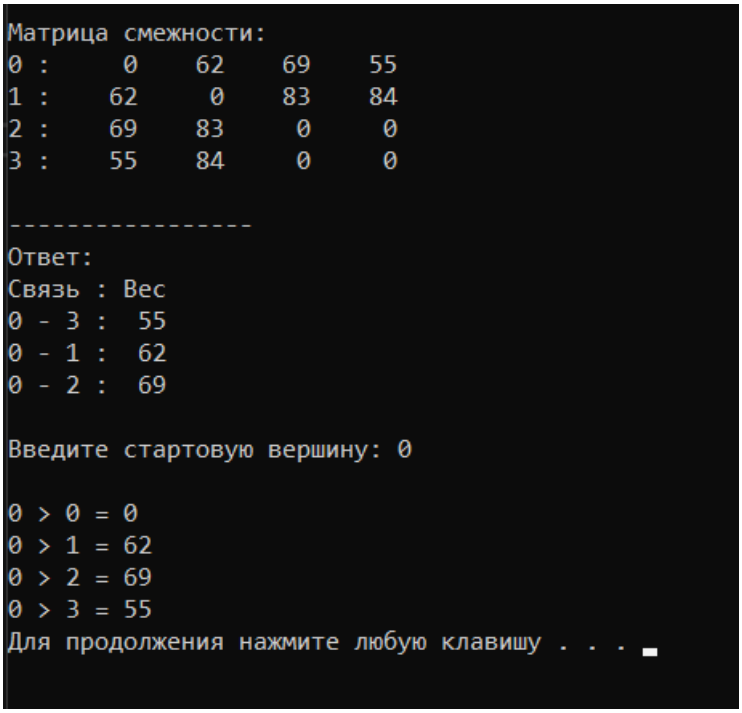

Рисунок 6 - Работа программы при автоматической генерации матрицы 4x4

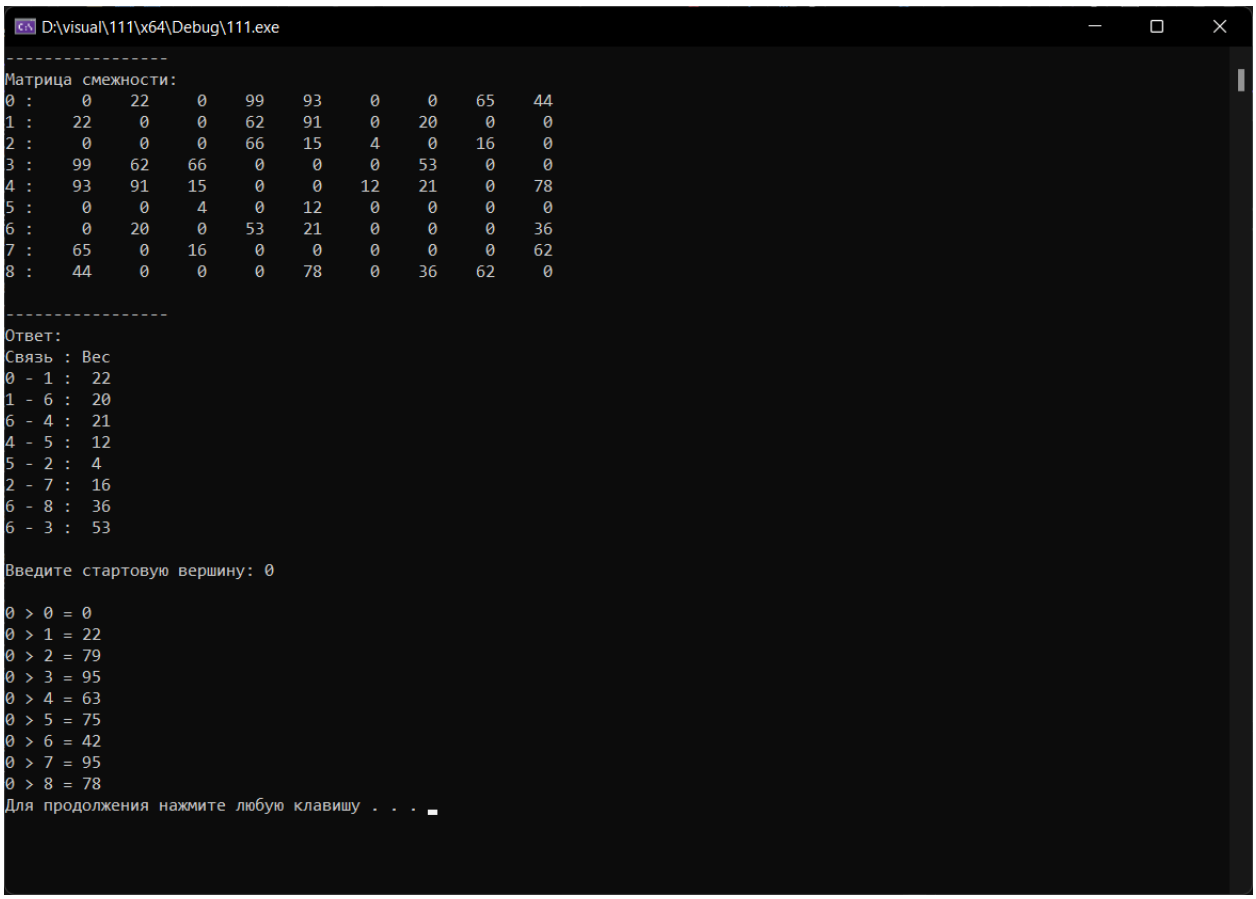

Рисунок 7 - Работа программы при автоматической генерации матрицы 9x9

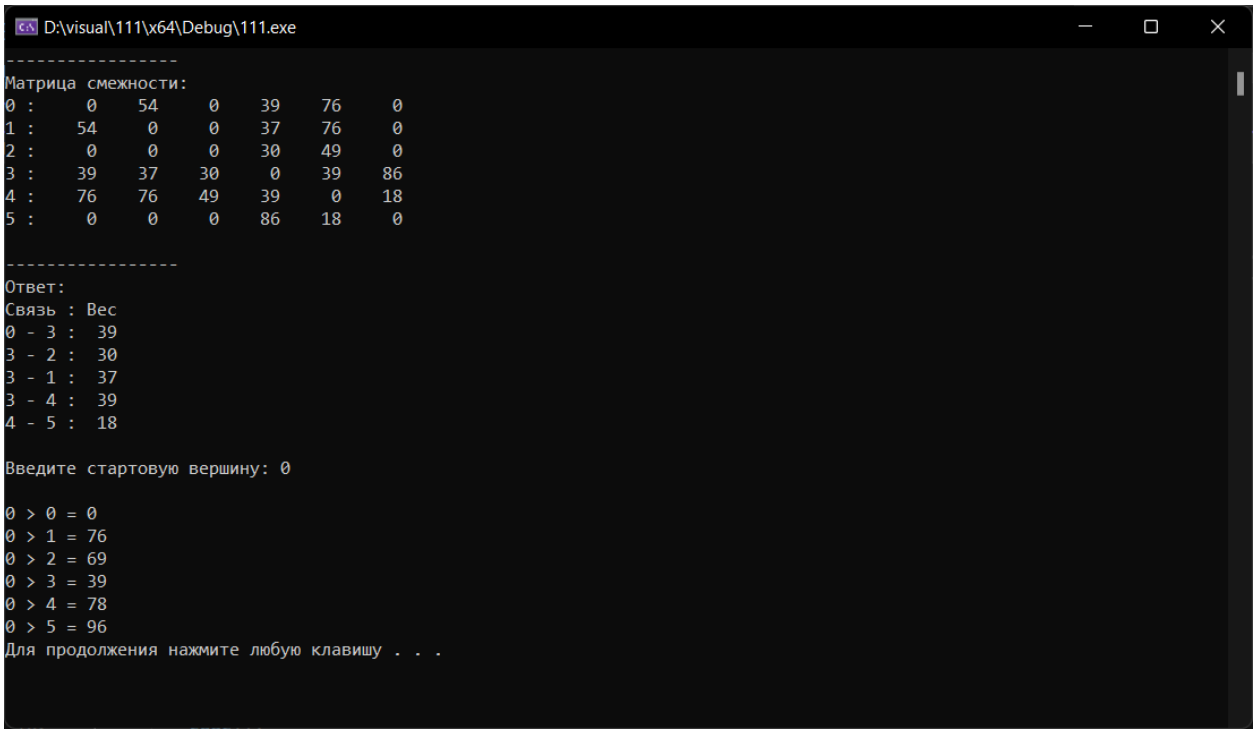

Рисунок 8 - Работа программы при автоматической генерации матрицы 6x6

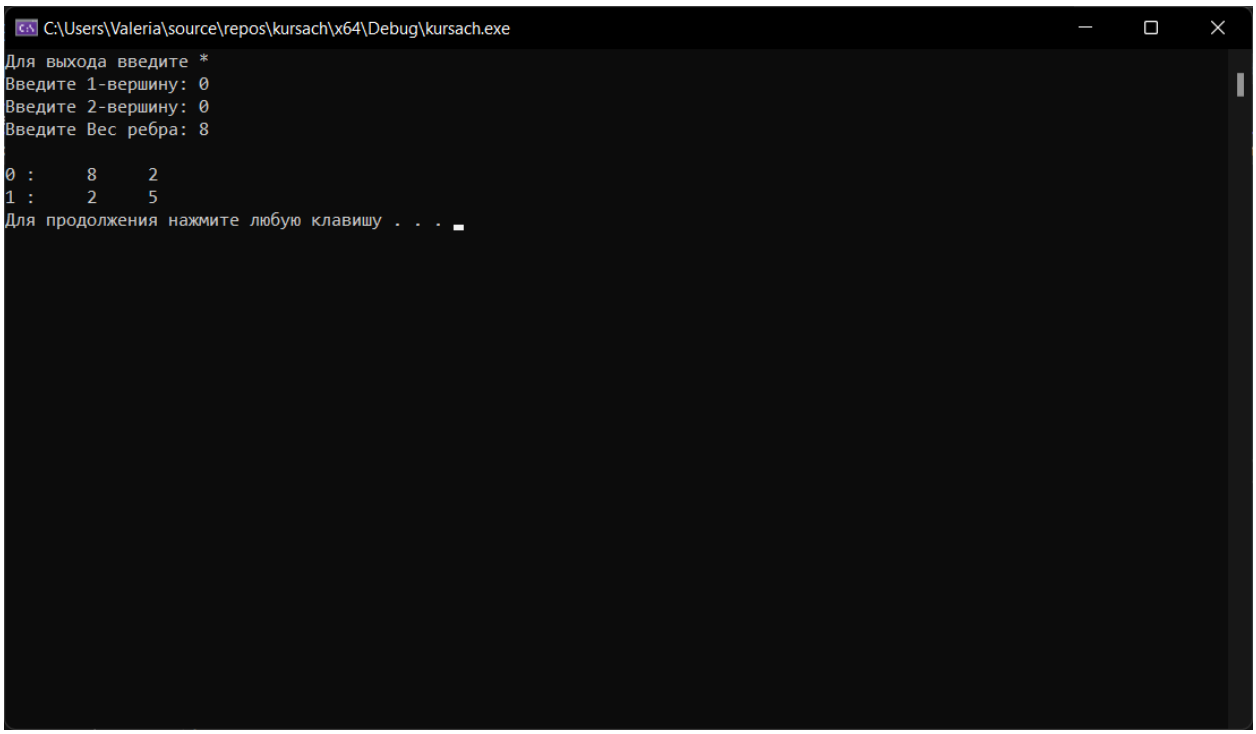

Рисунок 9 - Работа программы при ручном вводе матрицы 2x2

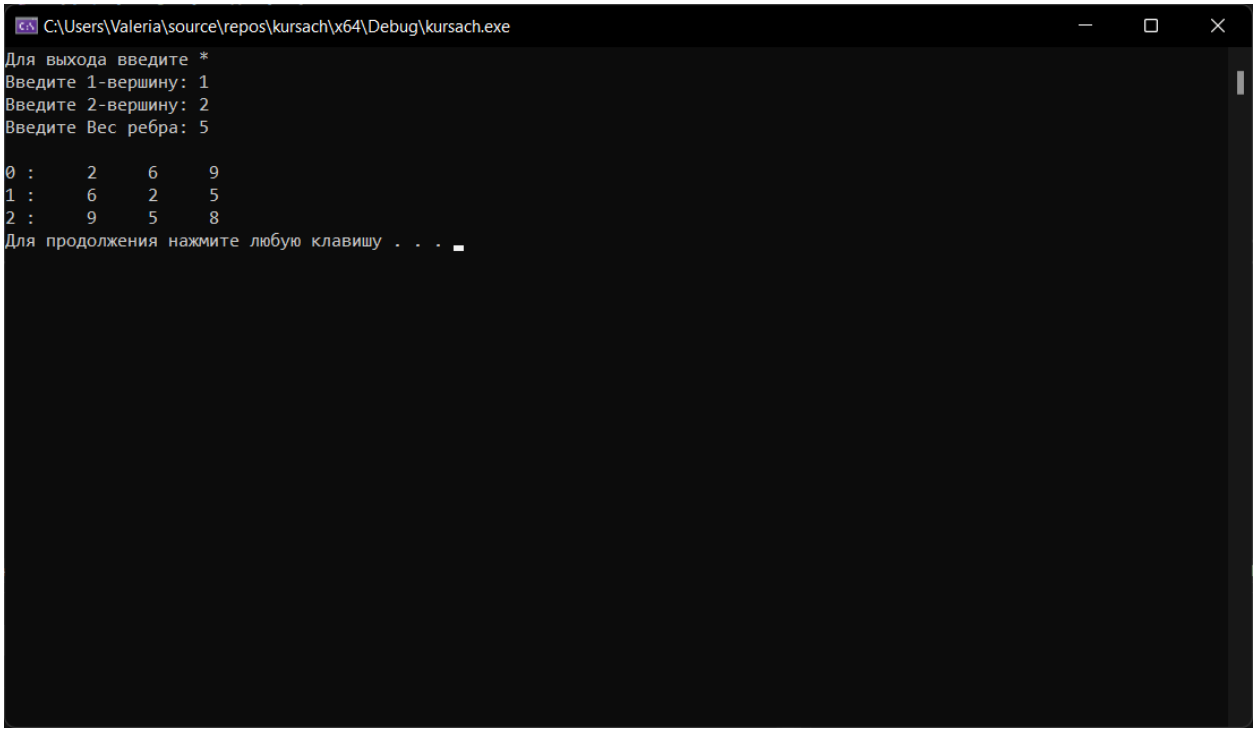

Рисунок 10 - Работа программы при ручном вводе матрицы 3x3

Таблица

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Вывод меню, вывод сообщения «сгенерировать матрицу автоматически» или «ввести матрицу с клавиатуры»	Верно
Выбор генерации матрицы	Вывод сообщения о количестве вершин в графе	Верно
Ввод матрицы с клавиатуры	Вывод сообщения о количестве вершин, ввод элементов, вывод элементов	Верно
Вывод результата	Вывод правильного результата на разно-размерных графах, идентичность с ручным расчетом	Верно
Правильность работы алгоритма	Совпадение ручных расчетов с результатом работы алгоритма	Верно
Проверка на наличие изолированных вершин	Должна выполняться проверка на наличие изолированных вершин	Верно

В результате тестирования было выявлено, что программа успешно проверяет данные на соответствие необходимым требованиям.

6. Ручной расчёт программы

Проведем проверку программы посредством ручных вычислений на примере графа с четырьмя вершинами (рисунок 6).

Начинаем обход из нулевой вершины, проверяем есть ли путь в другие вершины, если да, то идем. В нашем случаем можем направится в вершины с номерами: 1,2,3. Выгоднее всего будет направится в пункт 1, т.к. его вес меньше всего. Из вершины 1 мы можем направится в пункты 2,3. Выберем пункт 2, так путь в него наименьший. Из пункта 2 мы можем пойти в пункт 3 или вернуться обратно в пункт 1 и затем пойти в пункт 3, но уже по другому пути. Программа выберет 2-й вариант, потому что именно таким образом мы получим кратчайшее ребро.

Итого:

0-1: 12

0-2: 45

0-3: 46

Результат ручных расчетов совпадает с результатом работы алгоритма, таким образом можно сделать вывод, что программа работает верно.

Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Прима в *MicrosoftVisualStudio 2019*.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежности, а также работы с новыми алгоритмами. Углублены навыки знания языка программирования Си.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Потому что программа работает в консольном режиме, не добавляющем к сложности языка сложность программного оконного интерфейса.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

Список литературы

1. Уилсон Р. Введение в теорию графов.Пер.с анг.1977.208с.
2. Герберт Шилдт «полный справочник по С++» - вильямс,2006
3. З. Оре О. Графы и их применение: Пер.сангл.1965.176с

Листинг программы

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
using namespace std;
#define INF 0x3f3f3f3f // бесконечность
ofstream fout("cppstudio.txt");

class Graph {
private:

   int** G; // матрица смежности
   int V; //кол-во вершин
public:
   //конструктор
   Graph(int V)
   {
      this->V = V;
      G = new int* [V];
      for (int i = 0; i < V; i++) {
         G[i] = new int[V];
         for (int j = 0; j < V; j++)
            G[i][j] = 0;
      }
   }
   //Деструктор(освобождение памяти)
   ~Graph() {
      for (int i = 0; i < V; i++)
         delete[] G[i];
      delete[] G;
   }
   //добавление
   void addEdge(int i, int j, int w) {
      G[i][j] = G[j][i] = w;
   }
   //вывод на экран матрицы
   void toString() {
      for (int i = 0; i < V; i++) {
         cout << i << " : ";
         for (int j = 0; j < V; j++) {
            cout << setw(5) << G[i][j] << " ";
            fout << setw(5) << G[i][j] << " ";
         }
         cout << endl;
         fout << endl;
      }
   }
   //алгоритм Прима
   void Prima();
   //алгоритм дейстра(из 10 лабораторной)
   void Dijkstra(int st);
};

void Graph::Prima() {
   Graph Gr(V); // в этот граф запишется наше дерево(иначе сжатый граф по условия алгоритма)
   int edge; // количество ребер
   edge = 0; // установила количество ребер равным 0

   // создать массив для отслеживания выбранной вершины
   // selected(выбранный) станет true, в противном случае false
   int* selected = new int[V];
```

```
    // изначально установить выбранное значение false (заполнение всего массива false)
    memset(selected, 0, (V * 2) * 2);

    // количество egde в минимальном связующем дереве будет
    // всегда меньше, чем (V -1), где V - количество вершин в графе

    // выберите 0-ю вершину и сделайте ее истинной
    selected[0] = true;

    int x, y;  // строка, номер столбца

    // печать по краю и весу
    cout << "Связь" << " : " << "Вес" << endl;
    while (edge < V - 1) {

        //Для каждой вершины в множестве S нашла все смежные вершины
         // вычислила расстояние от вершины , выбранной на шаге 1.
         // если вершина уже находится в наборе S, отбросываю ее в противном случае
         //выбираю другую вершину, ближайшую к выбранной вершине на шаге 1.

        int min = INF; x = 0; y = 0;

        for (int i = 0; i < V; i++) {
            if (selected[i]) {
                for (int j = 0; j < V; j++) {
                    if (!selected[j] && G[i][j]) { // не выбрано и есть край
                        if (min > G[i][j]) {
                            min = G[i][j];
                            x = i;
                            y = j;
                        }
                    }
                }
            }
        }
        Gr.addEdge(x, y, G[x][y]); // добавляем путь в наш граф.
        cout << x << " - " << y << " :  " << G[x][y] << endl;
        fout << x << " - " << y << " :  " << G[x][y] << endl;
        selected[y] = true;
        edge++;
    }
    //сложность n^2+e
    delete[] selected;

    int st;
    cout << endl << "Введите стартовую вершину: "; cin >> st;
    fout << endl << "Стартовая вершина: " << st << endl;
    cout << endl;
    Gr.Dijkstra(st);
}

void Graph::Dijkstra(int st)
{
    int* distance = new int[V];
    bool* visited = new bool[V];
    int count, index;

    for (int i = 0; i < V; i++)
        distance[i] = INT_MAX, visited[i] = false;

    distance[st] = 0;

    for (count = 0; count < V - 1; count++)
    {
        int min = INT_MAX;
        for (int i = 0; i < V; i++)
```

```
            if (!visited[i] && distance[i] <= min)
            {
                min = distance[i]; index = i;
            }

        int u = index;
        visited[u] = true;
        for (int i = 0; i < V; i++)
            if (!visited[i] && G[u][i] && distance[u] != INT_MAX && distance[u] + G[u][i] < distance[i])
                distance[i] = distance[u] + G[u][i];
    }

    for (int i = 0; i < V; i++)
        if (distance[i] != INT_MAX) {
            cout << st << " > " << i << " = " << distance[i] << endl;
            fout << st << " > " << i << " = " << distance[i] << endl;
        }
        else {
            cout << st << " > " << i << " = " << "маршрут недоступен" << endl;
            fout << st << " > " << i << " = " << "маршрут недоступен" << endl;
        }
    delete[] distance;
    delete[] visited;
}

int main() {
    /*
    { 0, 9, 75, 0, 0 },
    { 9, 0, 95, 19, 42 },
    { 75, 95, 0, 51, 66 },
    { 0, 19, 51, 0, 31 },
    { 0, 42, 66, 31, 0 }

    c.addEdge(0 ,1, 9);
    c.addEdge(0, 2, 75);
    c.addEdge(1, 2, 95);
    c.addEdge(1, 3, 19);
    c.addEdge(1, 4, 42);
    c.addEdge(2, 3, 51);
    c.addEdge(2, 4, 66);
    c.addEdge(3, 4, 31);
    */
    setlocale(LC_ALL, "RUS");
    srand(time(NULL));
    int z = 5;
    cout << "Введите кол-во вершин графа: "; cin >> z;
    Graph c(z);

    string s;
    while (true) {
        system("cls");
        cout << "|-----------------------|" << endl
            << "|  Количество вершин: " << z << "  |" << endl
            << "|-----------------------|" << endl
            << "|          Меню         |" << endl
            << "|-----------------------|" << endl
            << "| 1)Случайное заполнение |" << endl
            << "|-----------------------|" << endl
            << "| 2)Заполнить самому     |" << endl
            << "|-----------------------|" << endl
            << "| 3)Выход                |" << endl
            << "|-----------------------|" << endl;

        cout << "       Введите: ";
        cin >> s;
        string q1, q2, q3;
```

```
        switch (atoi(s.c_str()))
        {
        case 1:
            system("cls");

            for (int i = 0; i < z; i++)
            {
                for (int j = i + 1; j < z; j++)
                {
                    if (bool(rand() % 2)) {
                        c.addEdge(i, j, rand() % 100);
                    }
                }
            }
            cout << "-----------------" << endl << "Матрица смежности: \n";
            fout << endl << "-----------------" << endl << "Матрица смежности: \n";
            c.toString();
            cout << endl << "-----------------" << endl << "Ответ: \n";
            fout << endl << "-----------------" << endl << "Ответ: \n";
            c.Prima();

            system("pause");
            break;
        case 2:
            system("cls");
            while (1) {
                system("cls");
                cout << "Для выхода введите *" << endl;
                cout << "Введите 1-вершину: "; cin >> q1;
                if (q1 == "*") break;
                cout << "Введите 2-вершину: "; cin >> q2;
                if (q1 == "*") break;
                cout << "Введите Вес ребра: "; cin >> q3;
                if (q1 == "*") break;
                c.addEdge(atoi(q1.c_str()), atoi(q2.c_str()), atoi(q3.c_str()));
                cout << endl;
                c.toString();
                system("pause");
            }
            system("cls");
            cout << "-----------------" << endl << "Матрица смежности: \n";
            fout << endl << "-----------------" << endl << "Матрица смежности: \n";
            c.toString();
            cout << endl << "-----------------" << endl << "Ответ: \n";
            fout << endl << "-----------------" << endl << "Ответ: \n";
            c.Prima();

            system("pause");
            break;
        case 3:
            exit(0);
            break;
        }
        system("cls");
    }

    fout.close();
    return 0;
}
```