Министерство науки и высшего образования Российской Федерации

Пензенский Государственный университет

Кафедра “Вычислительная техника”

**Пояснительная записка**

К курсовому проектированию

По курсу «Логика и основы алгоритмизации в

инженерных задачах»

На тему: «Реализация алгоритма Прима»

Выполнил студент группы 21ВВ3:

Рузляев Д. А.

Приняли:

к.т.н., доцент Юрова О. В.

д. т. н., профессор, зав. кав. ВТ, Митрохин М. А.

Пенза 2022

Оглавление

[Реферат](#_Toc61271564) 4

[Введение](#_Toc61271565) 5

[1.Постановка задачи](#_Toc61271566) 6

[2.Теоретическая часть](#_Toc61271567) 7

[3.Описание алгоритма программы](#_Toc61271568) 9

[4.Описание программы 1](#_Toc61271569)4

[5.Тестирование 1](#_Toc61271570)7

[6.Ручной расчёт программы 2](#_Toc61271571)1

[Заключение 2](#_Toc61271572)2

[Список литературы 2](#_Toc61271573)3

[Приложение А.](#_Toc61271574)[Листинг программы](#_Toc61271575) [2](#_Toc61271574)4

# Реферат

Пояснительная записка к курсовой работе: 27 страниц, 10 рисунков, 1 таблица, 1 приложение.

АЛГОРИТМ ПРИМА, АНАЛИЗ, БЛОК-СХЕМА, ВЕРШИНА, РЕБРО, ОСТОВ, МАТРИЦА СМЕЖНОСТИ, КАРКАС, ВЗВЕШЕННЫЙ ГРАФ.

Объект исследования: алгоритм нахождения минимального остовного дерева ­– алгоритм Прима.

Цель работы – разработка и реализация программы на языке Си, базирующейся на «алгоритме Прима», для поиска минимального остовного дерева взвешенного неориентированного графа. А также, тестирование правильности алгоритма и разработка псевдокода.

В ходе работы была разработана программа и псевдокод для поиска минимального остовного дерева между всеми вершинами графа с помощью алгоритма Прима. Разработки проводились в среде MicrosoftVisualStudio 2022.

В результате при помощи созданной программы была получена возможность нахождения минимального остовного дерева между всеми вершинами. Проанализирована работа алгоритма Прима.

# Введение

Алгоритм Прима — это алгоритм поиска минимального остовного дерева, что принимает граф в качестве входных данных и находит подмножество ребер этого графа, который формирует дерево, включающее в себя каждую вершину, а также имеет минимальную сумму весов среди всех деревьев, которые могут быть сформированы из графа.

Граф можно задавать несколькими путями, однако самый просто способ задания – задание графа матрицей смежности.

В качестве среды разработки была выбрана Microsoft Visual Studio 2022, язык программирования – Си.

Этот алгоритм назван в честь американского математика Роберта Прима (Robert Prim), который открыл его в 1957 г.

Алгоритм широко применяется при проектировании железных дорог, линий электропередачи и других линий коммуникации для предотвращения проблемы построения сетей с минимальными затратами.

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 реализовать программу, которая осуществит поиск минимального остовного дерева.

Проект должен иметь: текстовое или графическое меню, возможность задания пользователем размера графа, возможность выбора автоматического (случайного) или ручного (с клавиатуры или из файла) задания графа (элементов множества), а также возможность сохранения результатов работы программы. 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 должны выводиться матрица смежности, вершина, ближайшая вершина и вес ребра между этими вершинами. Необходимо предусмотреть различные варианты поиска, чтобы программа не выдавала ошибок и работа правильно. Устройство ввода - клавиатура и мышь.

Задание выполняются в соответствии с вариантом №18.

# 2.Теоретическая часть

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

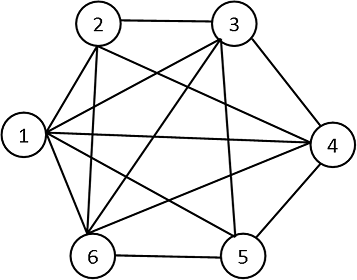


Рисунок 1 – Неориентированный граф.

При представлении графа матрицей смежности информация о ребрах графа хранится в квадратной матрице, где пути из одной вершины в другую и обратно обозначаются единицей, иначе нулем.

На вход алгоритма подаётся связный неориентированный граф. Для каждого ребра задаётся его стоимость.

Сначала берётся произвольная вершина и находится ребро, инцидентное данной вершине и обладающее наименьшей стоимостью. Найденное ребро и соединяемые им две вершины образуют дерево. Затем, рассматриваются рёбра графа, один конец которых — уже принадлежащая дереву вершина, а другой — нет; из этих рёбер выбирается ребро наименьшей стоимости. Выбираемое на каждом шаге ребро присоединяется к дереву. Рост дерева происходит до тех пор, пока не будут исчерпаны все вершины исходного графа.

Результатом работы алгоритма является остовное дерево минимальной стоимости.

# 3.Описание алгоритма программы

Для программной реализации алгоритма понадобятся два массива: selected (int) – массив для отслеживания выбранной вершины (true или false), matr (int) – для возможности динамического ввода, хранения данных.

Имеется граф matr. 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 matr, 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.

Ниже приведен псевдокод функции Alg\_Prima, Output, main.

**Alg\_Prima()**

1.для i=0 пока i<n делать i=i+1

2.selected[i] = false

3.конеццикла

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.конеццикла

**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

**Main()**

1.вывод «меню»

2.вывод «сгенерировать матрицу автоматически»

3.вывод «ввести матрицу с клавиатуры»

4.вывод «выход»

5.d=\_getch();

6.если d==’1’

7.вывод «введите размер матрицы»

8.ввод «n»

9.selected = (int\*)malloc(n \* sizeof(int))

10.matr = (int\*\*)malloc(n \* sizeof(int\*))

11. для i=0 пока i<n делать i=i+1

12.matr[i] = (int\*)malloc(n \* sizeof(int))

13.конец цикла

14. для i=0 пока i<n делать i=i+1

15. для j=0 пока j<n делать j=j+1

16.если i==j

17.matr[i][j]=0

18.конец условия

19 если i<j

20.matr[i][j] = rand()%80

21.matr[j][i] = matr[i][j]

22.конец условия

23. если matr[i][j]>0

24.stp++

25.конец условия

26.конец цикла

27.конец цикла

28. вызов функции Output(matr,n)

29.для i=0 пока i<n делать i=i+1

30. если stp == 0

31. Выход из программы

32.Конец условия

33.Конец цикла

34.вызов функции Alg\_Prima

35.конец условия

36.если d==’2’

37.вывод «введите размер матрицы»

38.ввод «n»

39.selected = (int\*)malloc(n \* sizeof(int));stp = (int\*)malloc(n \* sizeof(int));

40.matr = (int\*\*)malloc(n \* sizeof(int\*))

41. для i=0 пока i<n делать i=i+1

42.matr[i] = (int\*)malloc(n \* sizeof(int))

43.конец цикла

44. для i=0 пока i<n делать i=i+1

45. для j=0 пока j<n делать j=j+1

46.если i==j

47.matr[i][j]=0

48.вывод «matr[i][j]»

49.конецусловия

50. если i<j

51.ввод «matr[]i[j]»

52.matr[j][i] = matr[i][j]

53.конец условия

54. если matr[i][j]>0

55.stp++

56.конец условия

57.конец цикла

58.конец цикла

59.вызов функции Output(matr,n)

60.для i=0 пока i<n делать i=i+1

61. если stp == 0

62. Выход из программы

63.Конец условия

64.Конец цикла

65.вызов функции Alg\_Prima

66.если d==’3’

67.Выход из программы

68. конец условия

# 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.

Проект был создан в виде консольного приложения Win32(VisualC++).

Данная программа многомодульная, поскольку состоит из нескольких функций: main, Alg\_Prima, Output.

Работа программы начинается с запроса генерации матрицы. Если пользователь выбрал сгенерировать матрицу, то на экран выводится запрос на количество вершин в графе. Если же пользователь выбрал пункт «ввести матрицу с клавиатуры», то на экран выводится запрос на количество вершин в графе, а затем элементы массива вводятся с клавиатуры. Также предусмотрен выход из программы.

printf(" М Е Н Ю : \n");

printf(" 1) Сгенерировать матрицу автоматически. (1) \n");

printf(" 2) Ввести матрицу с клавиатуры. (2) \n");

printf(" 3) Выход (3)\n");

d = \_getch();

if (d == '1')

{

printf("\n");

printf(" Введите размер матрицы:");

scanf("%d",&n );

selected = (int\*)malloc(n \* sizeof(int)); //выделение памяти под массив отслеживания вершин

matr = (int\*\*)malloc(n \* sizeof(int\*)); //выделение памяти под массив указателей

for(int i=0; i<n; i++) //выделение памяти для массива значений

{

matr[i] = (int\*)malloc(n \* sizeof(int));

}

for(int i = 0; i < n; i++) // генерация массива случайных чисел

{

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

{

if(i == j)

{

matr[i][j] = 0;

}

if (i < j)

{

matr[i][j] = rand()%80;

matr[j][i] = matr[i][j];

}

}

}

printf("Результат: \n");

Output(matr,n);

}

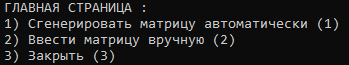


Рисунок 2 – Меню.

После выбора пользователе варианта генерации матрицы происходит ее вывод посредством функции Output:

for(int i = 0; i < n; i++) // вывод сгенерированной матрицы

{

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

{

printf("%3d ",matr[i][j]);

}

printf("\n");

}

printf("\n");

Ниже можно увидеть оформление начального запроса и дальнейшего действия с ним:

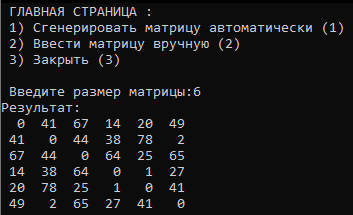


Рисунок 3 – Случайная генерация матрицы.

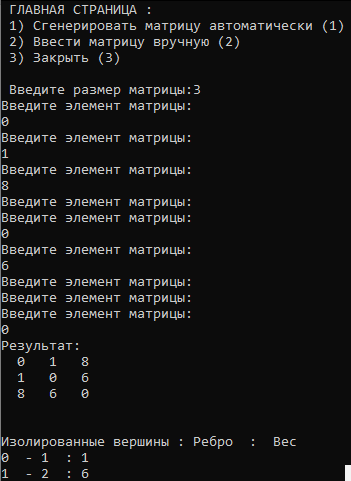


Рисунок 4 – Ручной ввод матрицы с клавиатуры.

# 5.Тестирование

Среда разработки MicrosoftVisualStudio 2022 представляет все средства, необходимые при разработке и отладки многомодульных программ.

Тестирование проводилось в рабочем порядке, в процессе разработки, а также 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.

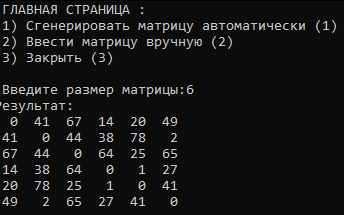


Рисунок 5 – Работа программы при автоматической генерации матрицы 6х6.

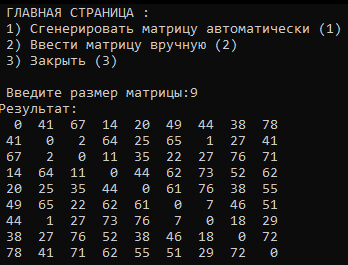


Рисунок 6 – Работа программы при автоматической генерации матрицы 9х9.

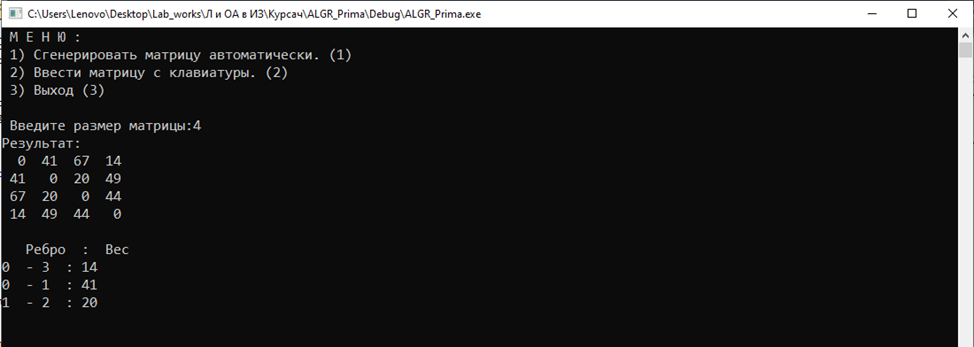


Рисунок 7 – Работа программы при автоматической генерации матрицы 4х4.

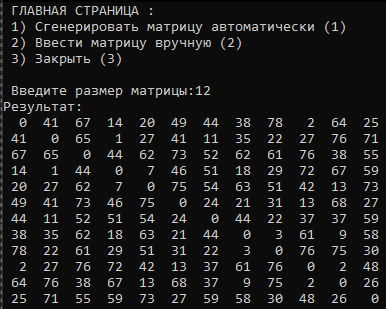


Рисунок 8 – Работа программы при ручном вводе матрицы 12х12.

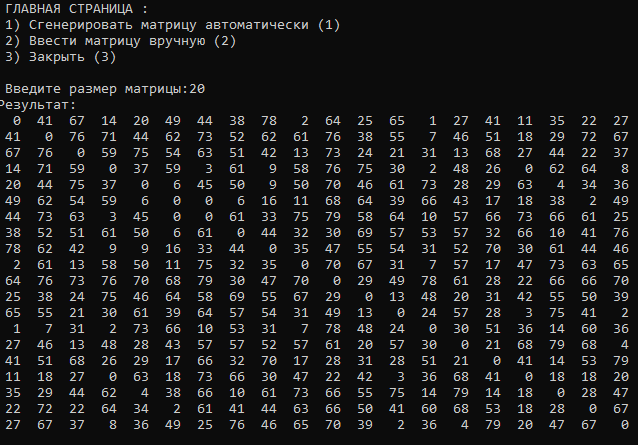


Рисунок 9 – Работа программы при ручном вводе матрицы 20х20.

|  |  |  |
| --- | --- | --- |
| 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 |

Таблица 1 – Описание поведения программы при тестировании.

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 7).

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 три, т.к. его вес меньше всего. Из вершины 3 мы можем направится в пункты 1,2 или же вернуться обратно. Вес этих ребер слишком велик и выгоднее вернуться в нулевую вершину. Оттуда можем попасть в пункт 2 или 1. Вес пункта 2 – 67, а пункта 1 – 41. Более целесообразно идти в пункт 1. Из этой вершины можем попасть в пункт 2. Его вес 20. Остовное дерево равно 75.

Итого:

0 – 3: 14

0 – 1: 41

1 – 2: 20

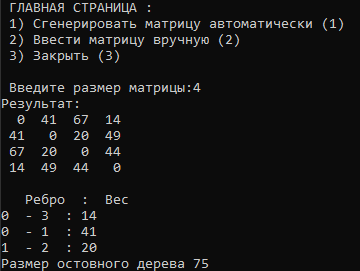


Рисунок 10 — Программное вычисление.

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 2022.

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 — 208 с.
2. Герберт Шилдт «полный справочник по С++» - Вильямс, 2006.
3. З. Оре О. Графы и их применение: Пер. с англ.1965 — 176 с.
4. Одинец В. П. К истории двух знаменитых оптимизационных алгоритмов в теории графов / Одинец В. П. - Математика в высшем образовании, 2013 - 121 с.

# Приложение А

# Листинг программы

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <conio.h>

#include <iostream>

#include <locale.h>

#include <windows.h>

#include <cstring>

#include <fstream>

using namespace std;

#define INF 9999999

#define V 5

FILE\* file = fopen("prym zapysal.txt", "a");

void Alg\_Prima(int n, int\*\* matr, int\* selected) {

int no\_edge;

// Изначально установим false

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

{

selected[i] = false;

}

no\_edge = 0;

// число ребер в минимальном остовном дереве будет

// всегда меньше (V -1), где V-число вершин в

//графе

selected[0] = true;

int x; // номер строки

int y; // номер столбца

int tr = 0;

printf("Ребро : Вес\n");

while (no\_edge < n - 1) {

int min = INF;

x = 0;

y = 0;

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

{

if (selected[i])

{

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

{

if (!selected[j] && matr[i][j]) // не выбрана и есть край

{

if (min > matr[i][j])

{

min = matr[i][j];

x = i;

y = j;

}

}

}

}

}

tr += matr[x][y];

printf("%d ", x); printf(" - "); printf("%d ", y);

printf(" : "); printf("%d\n", matr[x][y]);

fprintf(file, "%d ", x); fprintf(file, " - "); fprintf(file, "%d ", y);

fprintf(file, " : "); fprintf(file, "%d\n", matr[x][y]);

selected[y] = true;

no\_edge++;

}

printf("Размер остовного дерева %d\n", tr);

}

void Output(int\*\* matr, int n)

{

for (int i = 0; i < n; i++) // вывод сгенерированной матрицы

{

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

{

printf("%3d ", matr[i][j]);

}

printf("\n");

}

printf("\n ");

}

int main()

{

setlocale(LC\_ALL, "Russian");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int\*\* matr;//указатель для массива указателей

int\* selected; // Массив для отслеживания вершины

int\* stp;

int n, d;

d = 0;

while (d != 3) {

printf(" ГЛАВНАЯ СТРАНИЦА : \n");

printf(" 1) Сгенерировать матрицу автоматически (1) \n");

printf(" 2) Ввести матрицу вручную (2) \n");

printf(" 3) Закрыть (3)\n");

d = \_getch();

if (d == '1')

{

printf("\n");

printf(" Введите размер матрицы:");

scanf\_s("%d", &n);

stp = (int\*)malloc(n \* sizeof(int));

selected = (int\*)malloc(n \* sizeof(int)); //выделение памяти под массив отслеживания вершин

matr = (int\*\*)malloc(n \* sizeof(int\*)); //выделение памяти под массив указателей

for (int i = 0; i < n; i++) //выделение памяти для массива значений

{

matr[i] = (int\*)malloc(n \* sizeof(int));

}

for (int i = 0; i < n; i++) // генерация массива случайных чисел

{

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

{

if (i == j)

{

matr[i][j] = 0;

}

if (i < j)

{

matr[i][j] = rand() % 80;

matr[j][i] = matr[i][j];

}

if (matr[i][j] > 0)

{

stp[i]++;

}

}

}

printf("Результат: \n");

Output(matr, n);

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

{

if (stp[i] == 0)

{

printf("Алгоритм не работает при наличии изолированных вершин");

exit(0);

}

}

Alg\_Prima(n, matr, selected);

}

if (d == '2')

{

printf("\n");

printf(" Введите размер матрицы:");

scanf\_s("%d", &n);

stp = (int\*)malloc(n \* sizeof(int));

selected = (int\*)malloc(n \* sizeof(int)); //выделение памяти под массив отслеживания вершин

matr = (int\*\*)malloc(n \* sizeof(int\*)); //выделение памяти под массив указателей

for (int i = 0; i < n; i++) //выделение памяти для массива значений

{

matr[i] = (int\*)malloc(n \* sizeof(int));

}

for (int i = 0; i < n; i++) // генерация массива случайных чисел

{

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

{

printf("Введите элемент матрицы:\n");

if (i == j)

{

matr[i][j] = 0;

printf("%d\n", matr[i][j]);

}

if (i < j)

{

scanf\_s("%d", &matr[i][j]);

matr[j][i] = matr[i][j];

}

if (matr[i][j] > 0)

{

stp[i]++;

}

}

}

printf("Результат: \n");

Output(matr, n);

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

{

if (stp[i] == 0)

{

printf("Алгоритм не работает при наличии изолированных вершин");

exit(0);

}

}

Alg\_Prima(n, matr, selected);

}

if (d == '3')

{

exit(0);

}

}

\_getch();

return 1;

}