**Министерство науки и высшего образования Российской Федерации**

**Пензенский государственный университет**

**Кафедра “Вычислительная техника”**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Реализация Алгоритма Прима»

Выполнил студент группы 23ВВВ4:

Тинчурин Д. А.\_\_\_\_

Приняла:

Юрова О. В.\_\_\_\_

Пенза 2024

Оглавление

[Реферат 3](#_Toc91271104)

[Введение 4](#_Toc91271105)

[1. Постановка задачи 5](#_Toc91271106)

[2. Теоретическая часть 6](#_Toc91271107)

[3. Описание алгоритма программы 7](#_Toc91271108)

[4. Описание программы 12](#_Toc91271109)

[5. Тестирование 15](#_Toc91271110)

[6. Ручной расчёт программы 20](#_Toc91271111)

[Заключение 24](#_Toc91271112)

[Список литературы 25](#_Toc91271113)

[Приложение А 26](#_Toc91271114)

# 

# Реферат

Отчет 30 страниц, 9 рисунков

АЛГОРИТМ ПРИМА

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

В работе представлен алгоритм Прима. Реализовали текстовое меню, где можно выбрать ручной ввод из клавиатуры, автоматический ввод( заполнение массива случайными числами) или выход из программы. Затем мы вводим размер матрицы смежности. Дальше вызывается функция алгоритма Прима. Далее на результат выводится матрица смежности, которую мы ввели вручную или сгенерировали из случайных чисел, ребра минимального остовного дерева и само остовное дерево.

# Введение

В мире компьютерных наук и информационных технологий, эффективное управление ресурсами и поиск оптимальных решений в различных задачах является неотъемлемой частью исследований. Одним из ключевых аспектов в этой области является проблема поиска минимального остовного дерева в связном взвешенном графе, которая находит широкое применение в сетевых технологиях, транспортных системах, телекоммуникациях и многих других областях. Алгоритм Прима, предложенный в 1930 году чешским математиком Войцехом Ярником, представляет собой одно из эффективных решений данной проблемы.

В данной курсовой работе мы будем глубже исследовать алгоритм Прима, изучая его принципы, структуру и алгоритмическую сложность. Мы также рассмотрим его применение в реальных сценариях. Подробное понимание этого алгоритма позволит нам не только эффективно решать задачу минимального остовного дерева, но и применять полученные знания в широком спектре практических областей.

В качестве среды разработки мною была выбрана *Microsoft VisualStudio Code*, 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, осуществляющий нахождение минимального остовного дерева.

# Постановка задачи

Требуется реализовать программу, которая осуществит поиск минимального остовного дерева.

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.

# Теоретическая часть

Граф G (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.

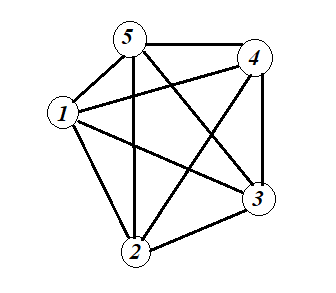


Рисунок 1 – Неориентированный граф

При использовании матрицы смежности для представления графа, информация о рёбрах сохраняется в квадратной матрице. Здесь каждый элемент матрицы обозначает наличие ребра между соответствующими вершинами: единица указывает на наличие ребра, ноль — на его отсутствие.

Входными данными для алгоритма является связный неориентированный граф, где для каждого ребра определена его стоимость. Процесс начинается с выбора произвольной вершины, после чего находится ребро, инцидентное данной вершине и обладающее минимальной стоимостью. Это ребро и соединяемые им две вершины формируют начальное дерево.

Далее рассматриваются рёбра графа, один конец которых уже принадлежит дереву, а другой — нет. Из этих рёбер выбирается ребро с минимальной стоимостью. Выбранное на каждом шаге ребро добавляется к дереву. Процесс продолжается до тех пор, пока все вершины исходного графа не будут включены в дерево. Результатом выполнения алгоритма является остовное дерево минимальной стоимости.

1. **Описание алгоритма программы**

Для программной реализации алгоритма понадобятся два массива: selected (int) – массив для отслеживания выбранной вершины (true или false), 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 G, 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. конец цикла

4. count\_v = 0;

5. selected[0] = true;

6. Пока count\_v < 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. sum = sum + matr[x][y];

21. tree[x][y] = matr[x][y];

22. tree[y][x] = matr[x][y];

23. вывод «x – y : matr[x][y]»

24. selected[y] = true;

25. count\_v++

26. конец цикла

27. вывод стоимости дерева

28. запись стоимости дерева в файл

29. selected[y] = 1;

**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 «Добро пожаловать! Выберите один из пунктов меню, что

начать:»

2. вывод «автоматическая генерация матрицы»

3. вывод «ручной ввод»

4. вывод «завершить сеанс»

5. z =\_getchar();

6. если z==’1’

7. вывод «введите размер матрицы»

8. ввод «n»

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

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

11. для i=0 пока i<n делать i=i+1

12. G[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. G[i][j]=0

18. конец условия

19. если i<j

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

21. G[j][i] = G[i][j]

22. конец условия

23. если G[i][j]>0

24. izol\_v++

25. конец условия

26. конец цикла

27. конец цикла

28. вызов функции Output(G,n)

29. для i=0 пока i<n делать i=i+1

30. если izol\_v == 0

31. Выход из программы

32. Конец условия

33. Конец цикла

34. вызов функции Alg\_Prima

35. вызов функции Output(G,n)

36. конец условия

37. если z==’2’

38. вывод «введите размер матрицы»

39. ввод «n»

40. selected = (int\*)malloc(n \* sizeof(int)); izol\_v = (int\*)malloc(n \* sizeof(int));

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

42. для i=0 пока i<n делать i=i+1

43. G[i] = (int\*)malloc(n \* sizeof(int))

44. конец цикла

45. для i=0 пока i<n делать i=i+1

46. для j=0 пока j<n делать j=j+1

47. если i==j

48. G[i][j]=0

49. вывод «G[i][j]»

50. конец условия

51. если i<j

52. ввод «G[]i[j]»

53. G[j][i] = G[i][j]

54 .конец условия

55. если G[i][j]>0

56. izol\_v++

57 .конец условия

58. конец цикла

59. конец цикла

60. вызов функции Output(G,n)

61. для i=0 пока i<n делать i=i+1

62. если izol\_v == 0

63. Выход из программы

64. Конец условия

65. Конец цикла

66. вызов функции Alg\_Prima

вызов функции Output(tree,n)

68. если z==’3’

69. Выход из программы

70. конец условия

# 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, 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(" Добро пожаловать! Выберите один из пунктов меню, что начать: \n\n");

printf(" 1. Автоматическая генерация матрицы. (нажмите 1) \n");

printf(" 2. Ручной ввод (нажмите 2) \n");

printf(" 3. Завершить сеанс (нажмите 3)\n");

printf("=========================================================================");

z = \_getch();

if (z == '1')

{

printf("\n");

printf("Введите размер матрицы: ");

scanf("%d", &n);

printf("=========================================================================");

izol\_v = (int\*)malloc(n \* sizeof(int));

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

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

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

{

G[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)

{

G[i][j] = 0;

}

if (i < j)

{

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

G[j][i] = G[i][j];

}

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

{

izol\_v[i]++;

}

}

}

printf("\nРезультат: \n");

Output(G, n); //вызов функции печати сгенерированного массива

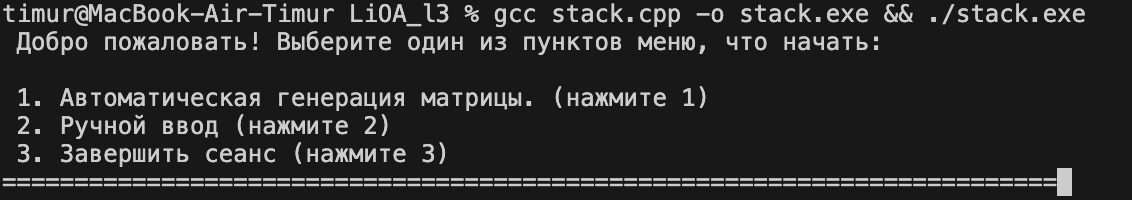


Рисунок 2 – Приветствие.

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:

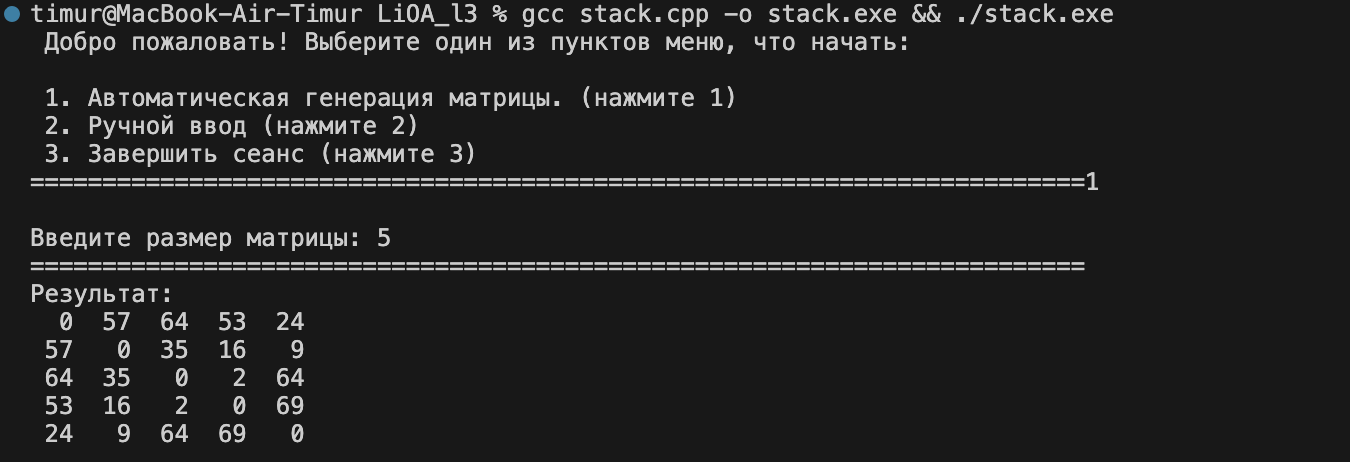


Рисунок 3 – Автоматическая генерация матрицы.

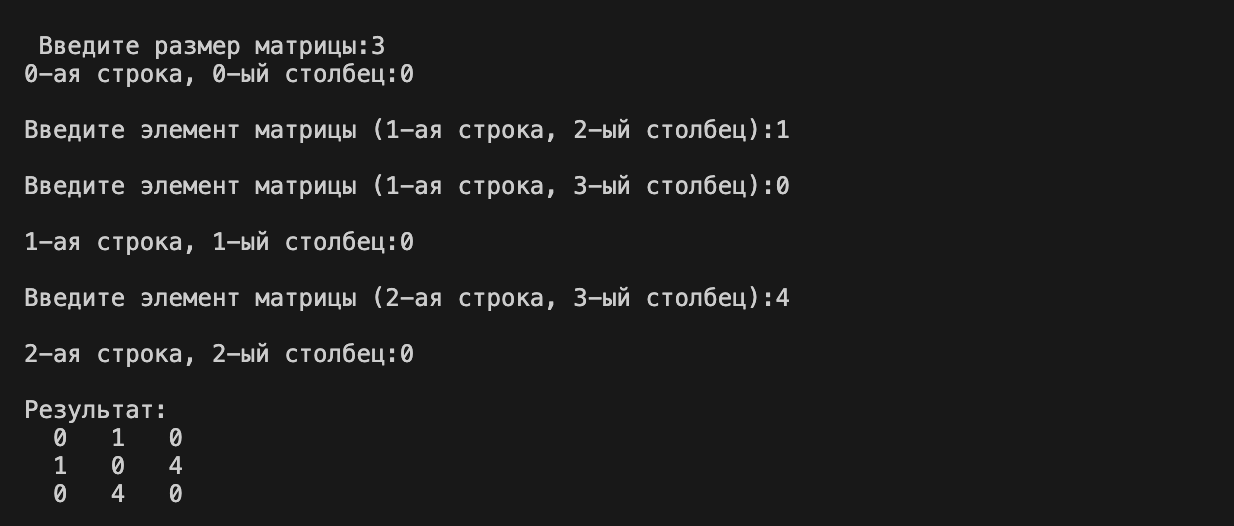


Рисунок 4 – Ручной ввод.

# Тестирование

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

Таблица 1 – Тестовый сценарий

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

Среда разработки Microsoft Visual Studio Code 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 ее запуске пользователем. Тут мы должны выбрать автоматическую генерацию матрицы, либо ручной ввод или же выход из программы.

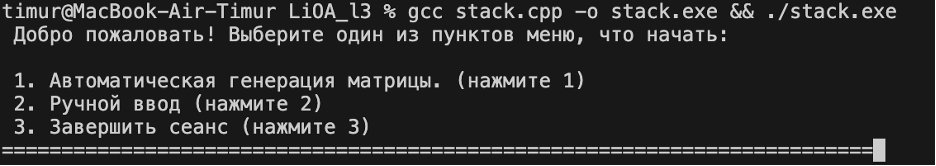


Рисунок 5 – Работа программы

Дальше протестируем работу генерации матрицы, заполненной случайными числами. Сначала нужно ввести размер матрицы и затем выводится матрица смежности.

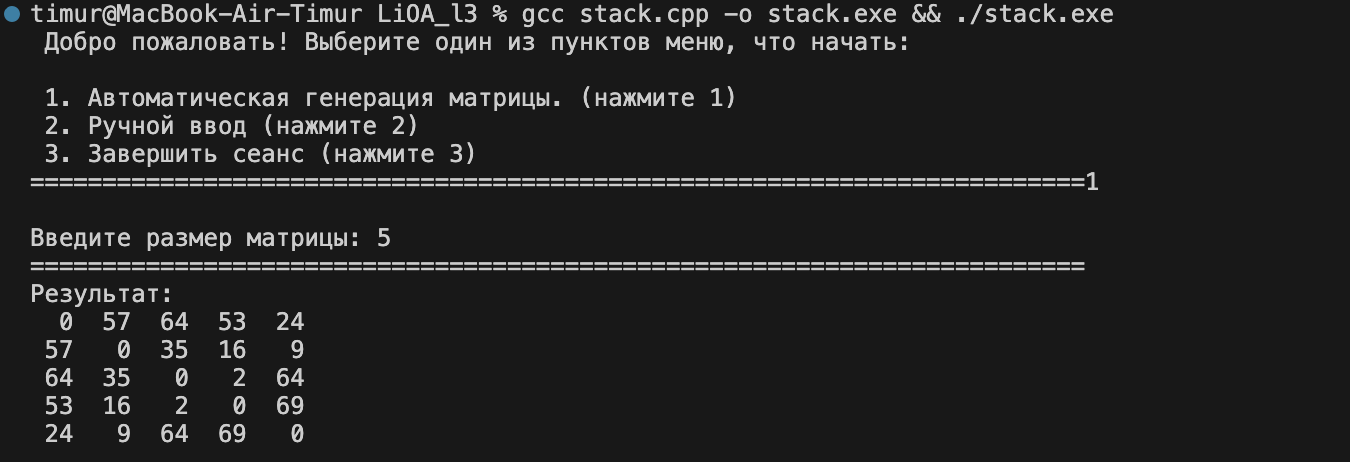


Рисунок 6 – Генерация случайной матрицы 5x5

Теперь проверим работу заполнения матрицы с помощью клавиатуры. Сначала нужно ввести размер матрицы и затем начинается ввод каждой вершины вручную.

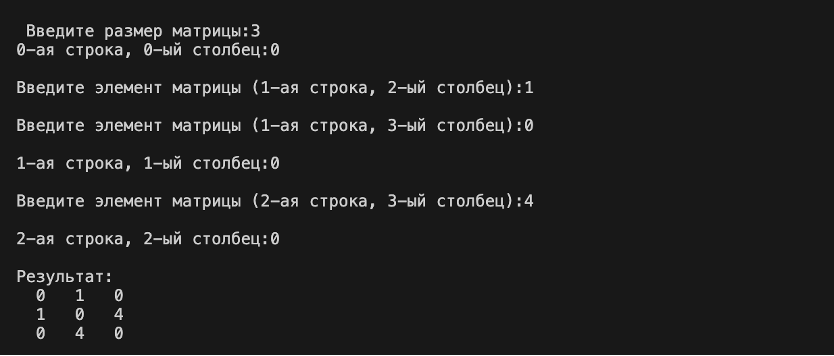


Рисунок 6 - Работа программы при автоматической генерации матрицы 3х3

Теперь проверим правильность вывода результата. Сначала выводятся ребра остовного дерева и вес ребра. Дальше выводится стоимость минимального остовного дерева и матрица минимального остовного дерева.

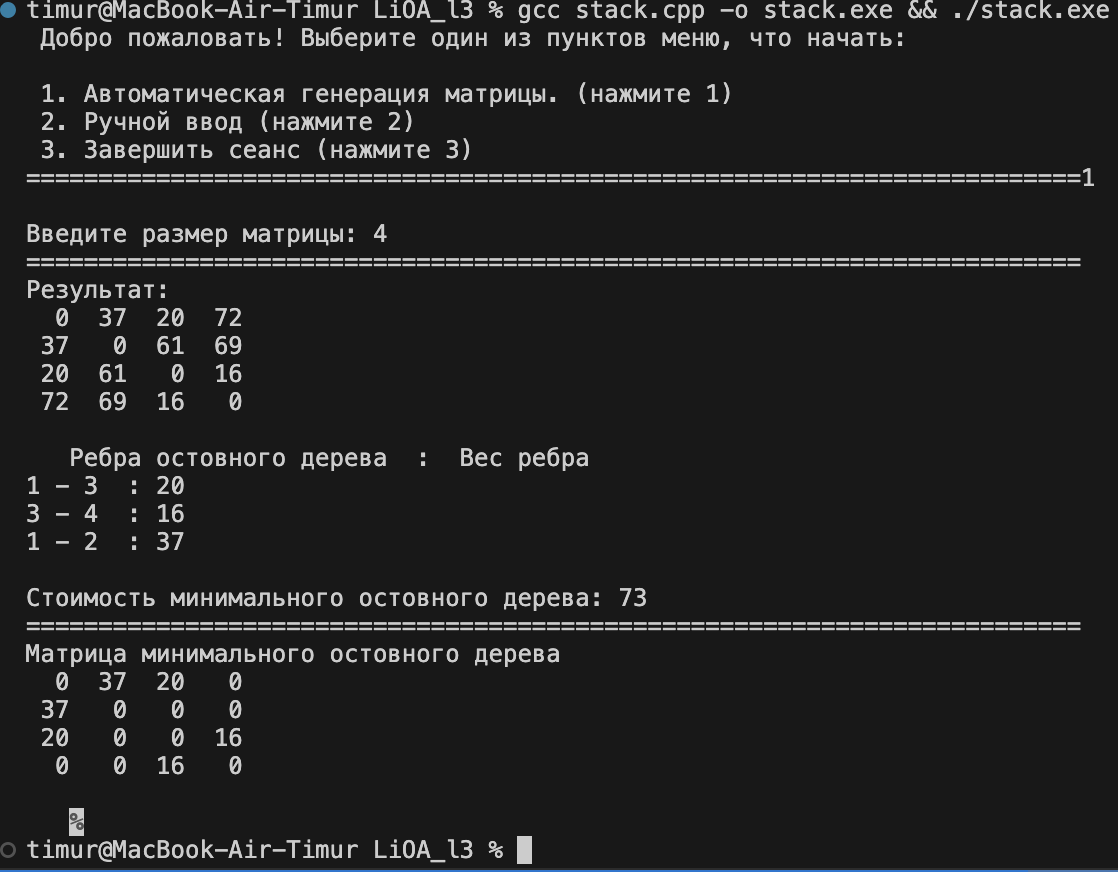


Рисунок 7 - Вывод программы при автоматической генерации матрицы 4х4

Дальше проверим работу проверки на изолированные вершины. В программе учтено, если при проходе строки матрицы G[i][j] только нули(для этого используется переменная stp), то это изолированная вершина и на экран выводится, что при наличии изолированных вершин алгоритм не работает.

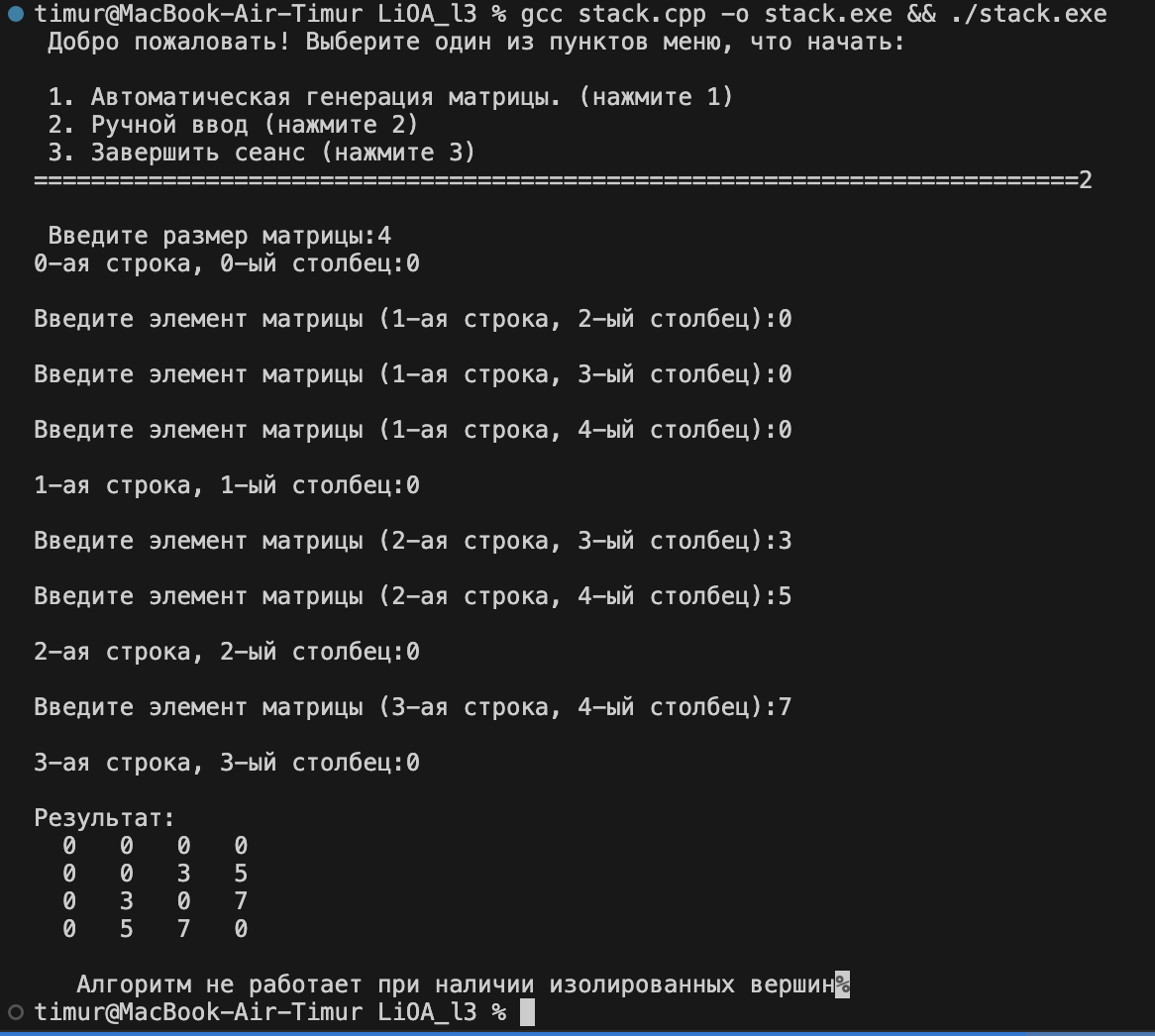


Рисунок 8 - Работа программы на проверку изолированности вершин при ручном вводе матрицы 4х4

Затем проверяем правильность работы самого алгоритма.

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 1,4. Выберем пункт 4, так путь в него наименьший. Из пункта 1 мы можем пойти в пункт 2, и этот путь на эту вершину наименьший.

Итого: 1-3: 20; 3-4: 16; 1-2: 37

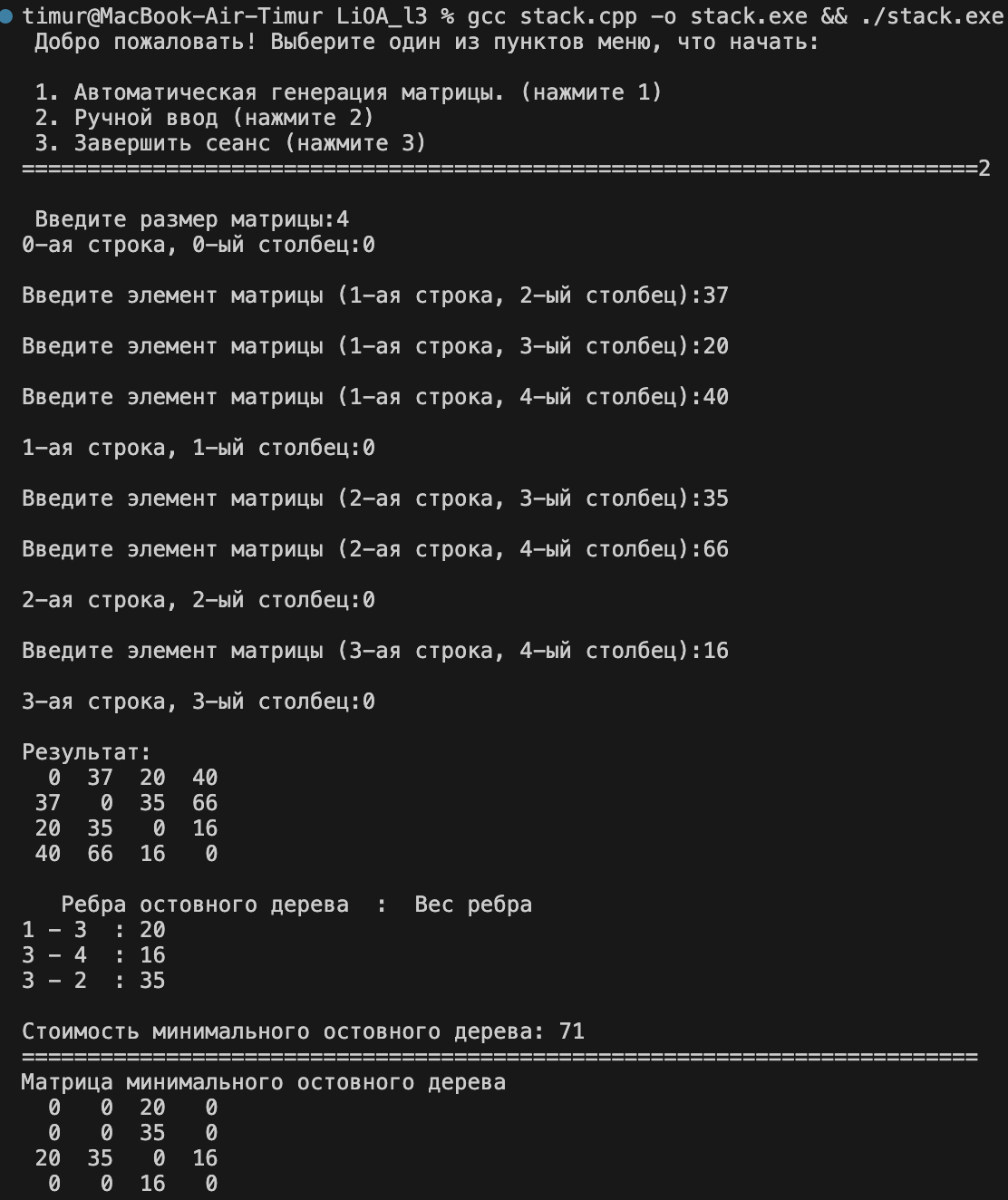


Рисунок 9 – Проверка работы алгоритма Прима

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

# Ручной расчёт программы

Проведем проверку программы посредством ручных вычислений на примере графа с четырьмя вершинами.

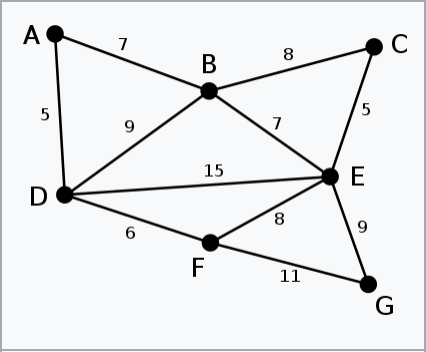


Рисунок 10 – Неориентированный граф

A=1-ая вершина;

B=2-ая вершина;

C=3-ая вершина;

D=4-ая вершина;

E=5-ая вершина;

F=6-ая вершина;

G=7-ая вершина;

.В качестве начальной произвольно выбирается вершина D. Каждая из вершин A, B, E и F соединена с D единственным ребром. Вершина A — ближайшая к D, и выбирается как вторая вершина вместе с ребром AD. Следующая вершина — ближайшая к любой из выбранных вершин D или A. B удалена от D на 9 и от A — на 7. Расстояние до E равно 15, а до F — 6. F является ближайшей вершиной, поэтому она включается в дерево F вместе с ребром DF.

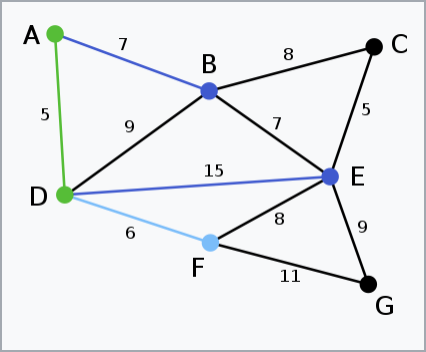


Рисунок 11- Выбор ребер по минимальной длине

Аналогичным образом выбирается вершина B, удаленная от A на 7. В этом случае есть возможность выбрать либо C, либо E, либо G. C удалена от B на 8, E удалена от B на 7, а G удалена от F на 11. E — ближайшая вершина, поэтому выбирается E и ребро BE.

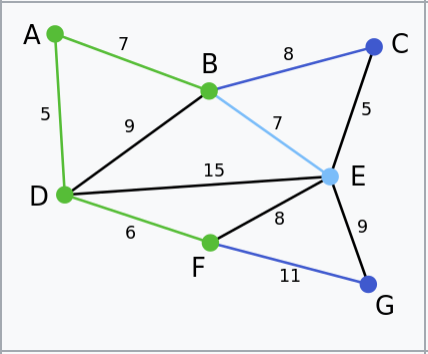


Рисунок 11- Выбор ребер по минимальной длине

Здесь доступны только вершины C и G. Расстояние от E до C равно 5, а до G — 9. Выбирается вершина C и ребро EC. Единственная оставшаяся вершина — G. Расстояние от F до неё равно 11, от E — 9. E ближе, поэтому выбирается вершина G и ребро EG.

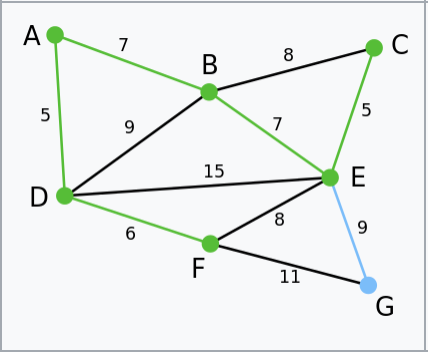


Рисунок 12 - Выбор ребер по минимальной длине

Выбраны все вершины, минимальное остовное дерево построено (выделено зелёным). В этом случае его вес равен 39.

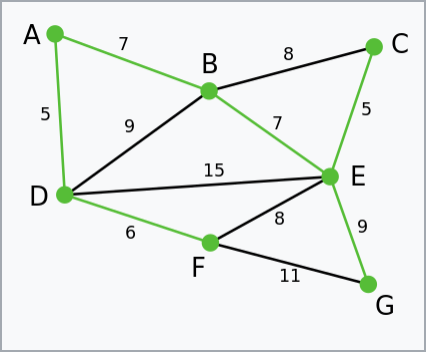


Рисунок 13 - Выбор ребер по минимальной длине

Сверим с результатом вывода на программе.

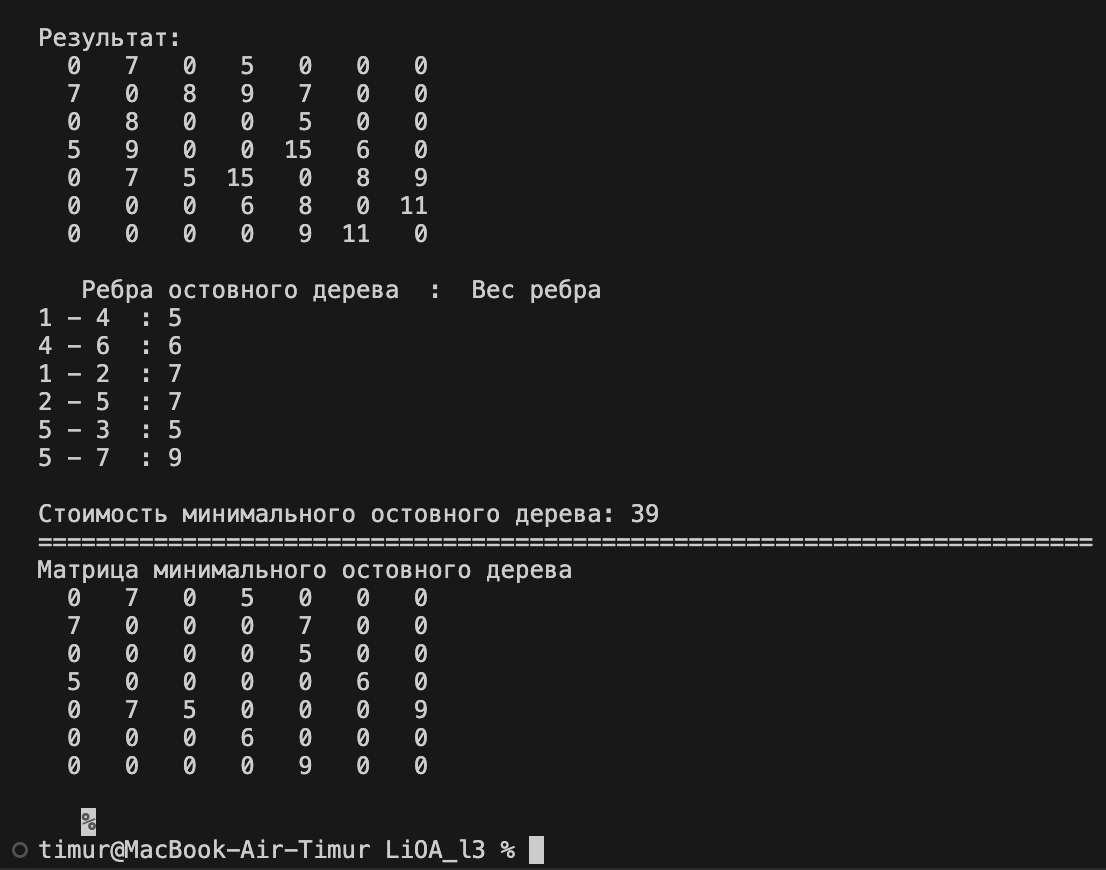


Рисунок 14 – Результат программы

Результат программы сходится с результатом ручного расчета.

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Прима в *Microsoft Visual Studio Code*.

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. Герберт Шилдт «полный справочник по С++» - вильямс,2006
2. З. Оре О. Графы и их применение: Пер.сангл.1965.176с.
3. Стивен Прата «Язык программирования C» Диалектика,2018
4. Уилсон Р. Введение в теорию графов.Пер.с анг.1977.208с.

# Приложение А

Листинг программы

#include <conio.h>

#include <iostream>

#include <locale.h>

#include <windows.h>

#include <cstring>

#include <fstream>

using namespace std;

#define INF 9999999

FILE\* file = fopen("text.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 sum = 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]) // если отмечена

{

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

{

if (!selected[j] && matr[i][j]) // не отмечена и есть ребро между

{

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

{

min = matr[i][j];

x = i;

y = j;

}

}

}

}

}

sum = sum + 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("\nСтоимость минимального остовного дерева: %d", sum);

fprintf(file, "\nСтоимость минимального остовного дерева: %d\n", sum);

fprintf(file, "\n=========================================================================");

printf("\n=========================================================================\n");

}

void Output(int\*\* matr, int n) // вывод сгенерированной матрицы

{

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()

{

srand(time(0));

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;

printf(" Добро пожаловать! Выберите один из пунктов меню, что начать: \n\n");

printf(" 1. Автоматическая генерация матрицы. (нажмите 1) \n");

printf(" 2. Ручной ввод (нажмите 2) \n");

printf(" 3. Завершить сеанс (нажмите 3)\n");

printf("=========================================================================");

d = \_getch();

if (d == '1')

{

printf("\n");

printf("Введите размер матрицы: ");

scanf("%d", &n);

printf("=========================================================================");

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("\nРезультат: \n");

Output(matr, n); //вызов функции печати сгенерированного массива

//printf("\nIzolir versh:\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); //вызов функции реализации алгоритма Прима

}

if (d == '2')

{

printf("\n");

printf(" Введите размер матрицы:");

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

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

}

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

{

stp[i]++;

}

}

}

printf("Результат: \n");

Output(matr, n); // вызов функции вывода результата генерации

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); // алгоритм прима

}

if (d == '3')

{

exit(0);

}

return 0;

}