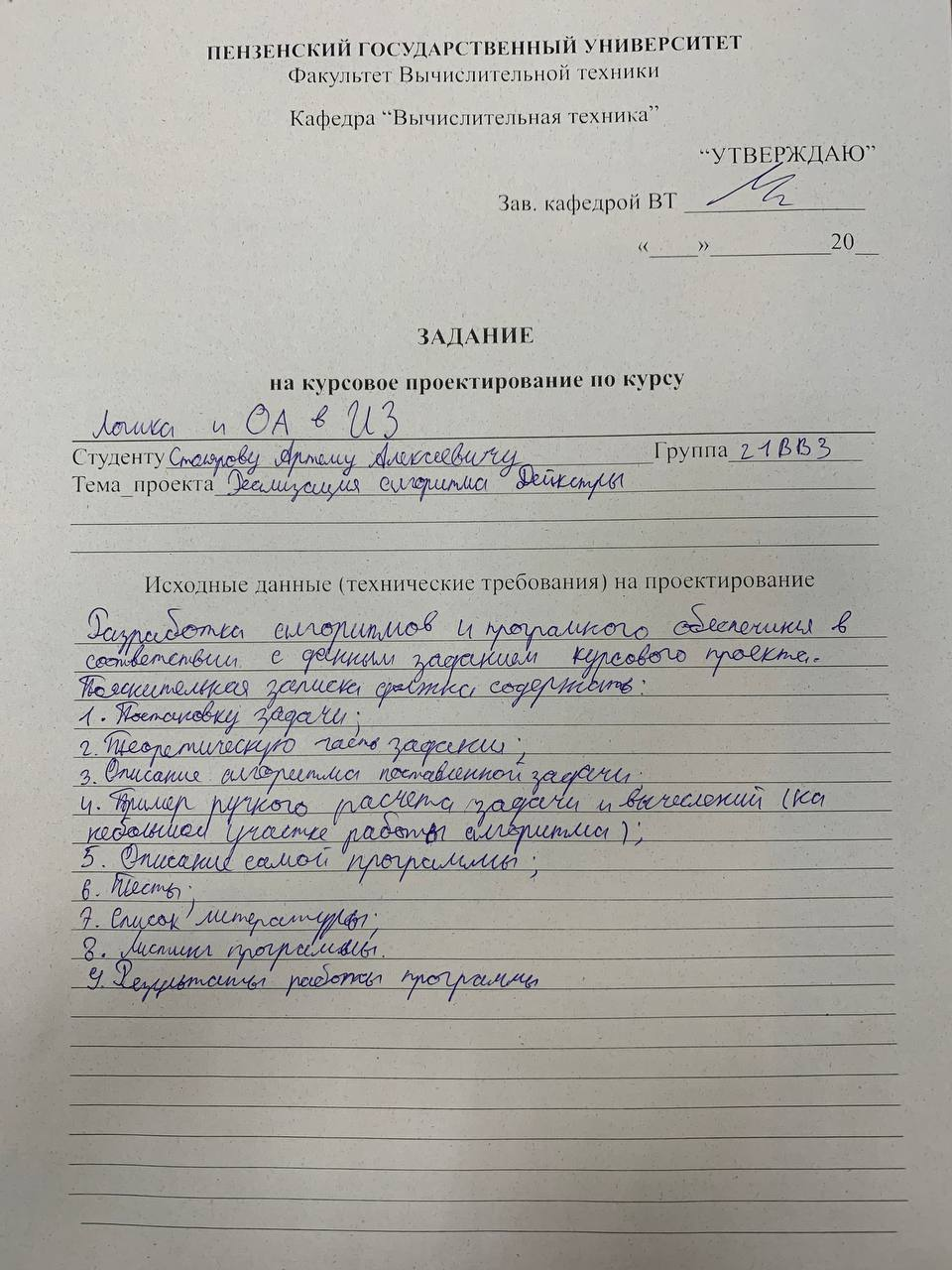
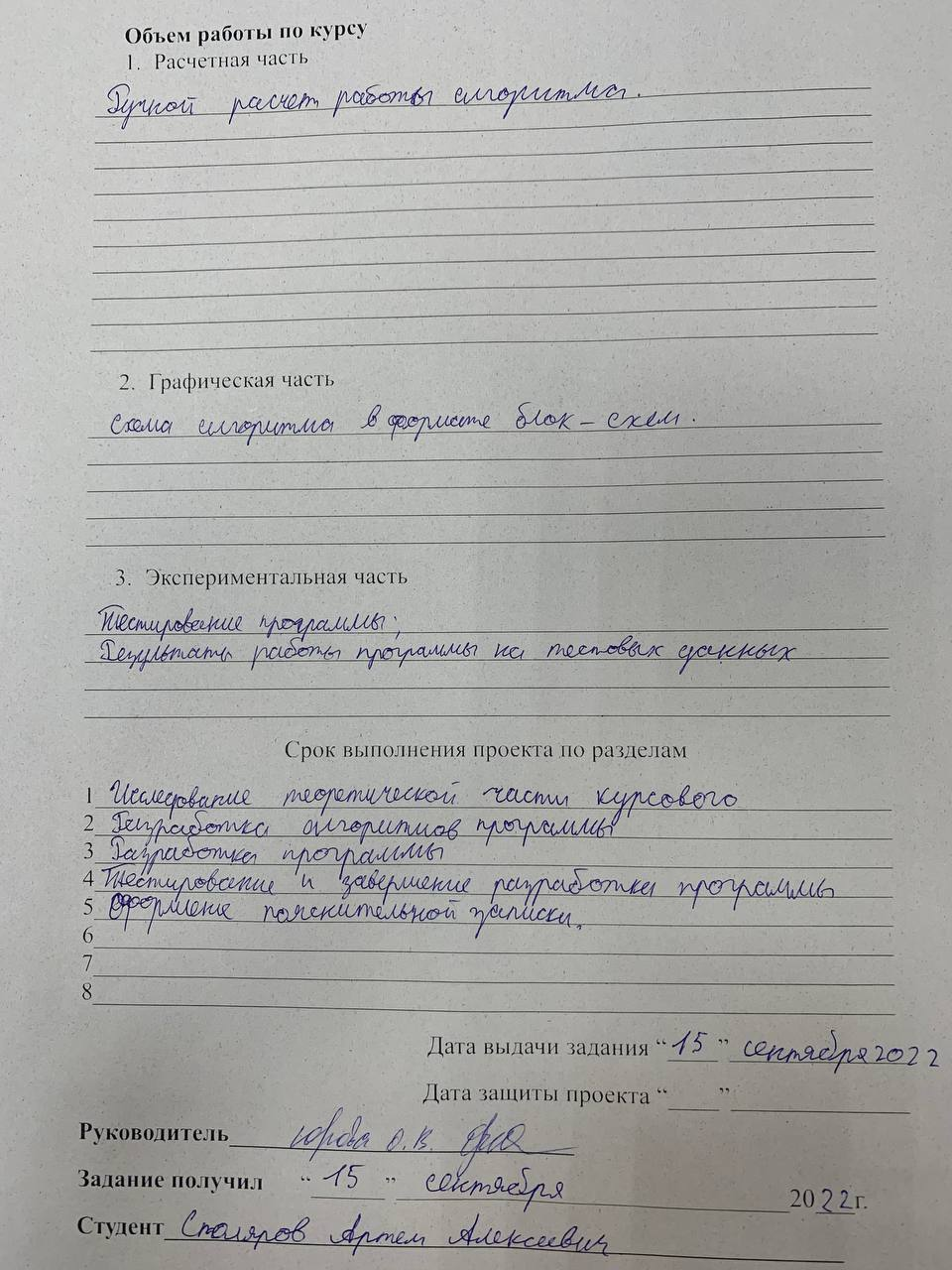


****

****

**Содержание**

[Реферат 5](#__RefHeading___Toc1765_75800640)

[Введение 6](#__RefHeading___Toc1767_75800640)

[1. Постановка задачи 7](#__RefHeading___Toc1769_75800640)

[2. Теоретическая часть задания 8](#__RefHeading___Toc1771_75800640)

[3. Описание алгоритма программы 15](#__RefHeading___Toc1773_75800640)

[4. Описание программы 19](#__RefHeading___Toc1775_75800640)

[5. Тестирование 23](#__RefHeading___Toc4097_75800640)

[6. Ручной расчѐт задачи 29](#__RefHeading___Toc1779_75800640)

[Заключение 31](#__RefHeading___Toc1781_75800640)

[Список литературы 32](#__RefHeading___Toc1783_75800640)

[Листинг программы. 33](#__RefHeading___Toc1785_75800640)

5

# Реферат

Отчет 38 стр, 28 рисунков.

АЛГОРИТМ ДЕЙКСТРЫ, ГРАФЫ, ОБХОД В ШИРИНУ.

Цель исследования – разработка программы, способная находить кратчайшие пути от одной из вершин графа до всех остальных.

В работе рассмотрены правила построения графа в ручную, автоматически, правила работы алгоритма, записи результатов в файл. Установлено, что с помощью данного алгоритма можно найти кратчайшие пути от одной из вершин графа до всех остальных.

# Введение

Алгоритм Дейкстры позволяет находить кратчайшие пути от одной из вершин графа до всех остальных.

Данный алгоритм работает на ориентированных графах, а так же он подходит для графов, которые не имеют ребра с отрицательным весом. Таким образом, алгоритм может найти путь между двумя вершинами только в том случае, если существует хотя бы один путь между ними. Если это условие не выполняется, то алгоритм отработает корректно, вернув значение «бесконечность» для пары несвязных вершин.

В качестве среды разработки мною была выбрана среда

MicrosoftVisualStudio2022, 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 поиска кратчайших путей в графе.

# 1. Постановка задачи

Требуется разработать программу, которая будет осуществлять поиск кратчайших путей между заданной вершиной и всеми остальными вершинами в графе.

Исходный граф в программе должен задаваться матрицей смежности, причѐм при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности, вводил вершину, от которой бы велся поиск кратчайшего расстояния до остальных вершин графа. После обработки этих данных на экран должна выводиться матрица смежности орграфа, а так же результат поиска кратчайших путей. Проект должен иметь текстовое меню, возможность выбора автоматического или ручного задания графа. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь.

Задания выполняются в соответствии с вариантом №13.

# 2. Теоретическая часть задания

Каждой вершине из V сопоставим метку — минимальное известное расстояние от этой вершины до a.

Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

Инициализация.

Метка самой вершины a полагается равной 0, метки остальных вершин — бесконечности. Это отражает то, что расстояния от a до других вершин пока неизвестны. Все вершины графа помечаются как непосещённые.

Шаг алгоритма.

Если все вершины посещены, алгоритм завершается.

В противном случае, из ещё не посещённых вершин выбирается вершина u, имеющая минимальную метку.

Мы рассматриваем всевозможные маршруты, в которых u является предпоследним пунктом. Вершины, в которые ведут рёбра из u, назовём соседями этой вершины. Для каждого соседа вершины u, кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки u и длины ребра, соединяющего u с этим соседом.

Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину u как посещённую и повторим шаг алгоритма.

### Пример

Рассмотрим выполнение алгоритма на примере графа, показанного на рисунке.

Пусть требуется найти кратчайшие расстояния от 1-й вершины до всех остальных.

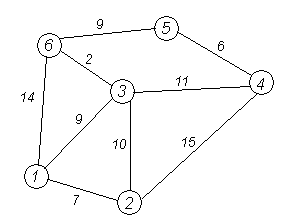


Рисунок 1 – Пример алгоритма

Кружками обозначены вершины, линиями — пути между ними (рёбра графа).

В кружках обозначены номера вершин, над рёбрами обозначен их вес — длина пути.

Рядом с каждой вершиной красным обозначена метка — длина кратчайшего пути в эту вершину из вершины 1.

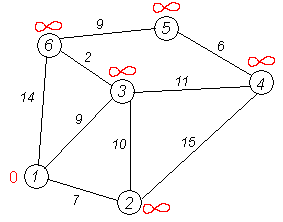


Рисунок 2 – Пример алгоритма

**Первый шаг.**

Минимальную метку имеет вершина 1. Её соседями являются вершины 2, 3 и 6.

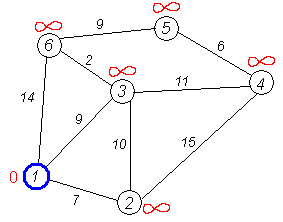


Рисунок 3 – Пример алгоритма

Первый по очереди сосед вершины 1 — вершина 2, потому что длина пути до неё минимальна.

Длина пути в неё через вершину 1 равна сумме значения метки вершины 1 и длины ребра, идущего из 1-й в 2-ю, то есть 0 + 7 = 7.

Это меньше текущей метки вершины 2, бесконечности, поэтому новая метка 2-й вершины равна 7.

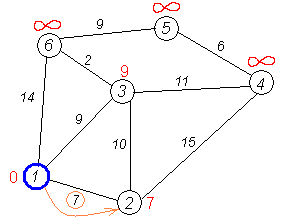


Рисунок 4 – Пример алгоритма

Аналогичную операцию проделываем с двумя другими соседями 1-й вершины — 3-й и 6-й.

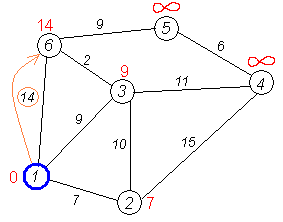


Рисунок 5 – Пример алгоритма

Все соседи вершины 1 проверены.

Текущее минимальное расстояние до вершины 1 считается окончательным и пересмотру не подлежит.

Вычеркнем её из графа, чтобы отметить, что эта вершина посещена.

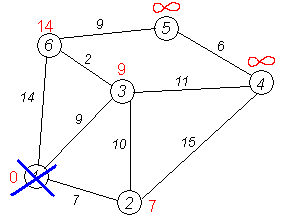


Рисунок 6 – Пример алгоритма

**Второй шаг.**

Снова находим «ближайшую» из непосещённых вершин. Это вершина 2 с меткой 7.

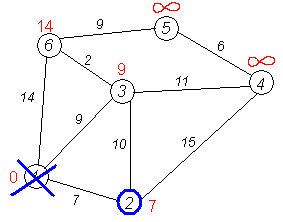


Рисунок 7 – Пример алгоритма

Снова пытаемся уменьшить метки соседей выбранной вершины, пытаясь пройти в них через 2-ю вершину. Соседями вершины 2 являются вершины 1, 3 и 4.

Первый (по порядку) сосед вершины 2 — вершина 1. Но она уже посещена, поэтому с 1-й вершиной ничего не делаем.

Следующий сосед — вершина 3, так как имеет минимальную метку.

Если идти в неё через 2, то длина такого пути будет равна 17 (7 + 10 = 17). Но текущая метка третьей вершины равна 9, а это меньше 17, поэтому метка не меняется.

# 

Рисунок 8 – Пример алгоритма

Ещё один сосед вершины 2 — вершина 4. Если идти в неё через 2-ю, то длина такого пути будет равна сумме кратчайшего расстояния до 2-й вершины и расстояния между вершинами 2 и 4, то есть 22 (7 + 15 = 22).  
Поскольку 22<, устанавливаем метку вершины 4 равной 22.

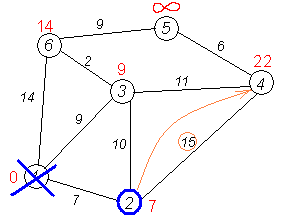


Рисунок 9 – Пример алгоритма

Все соседи вершины 2 просмотрены, замораживаем расстояние до неё и помечаем её как посещённую.

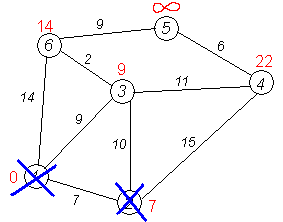


Рисунок 10 – Пример алгоритма

**Третий шаг.**

Повторяем шаг алгоритма, выбрав вершину 3. После её «обработки» получим такие результаты:

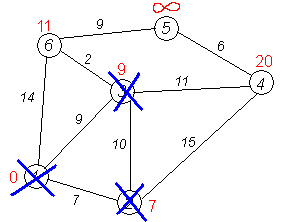


Рисунок 11 – Пример алгоритма

**Дальнейшие шаги.**

Повторяем шаг алгоритма для оставшихся вершин. Это будут вершины 6, 4 и 5, соответственно порядку. Далее придем к следующему виду:

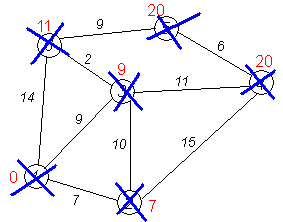


Рисунок 12 – Пример алгоритма

**Завершение выполнения алгоритма.**

Алгоритм заканчивает работу, когда все вершины посещены.

Результат работы алгоритма виден на последнем рисунке: кратчайший путь от вершины 1 до 2-й составляет 7, до 3-й — 9, до 4-й — 20, до 5-й — 20, до 6-й — 11.

Если в какой-то момент все непосещённые вершины помечены бесконечностью, то это значит, что до этих вершин нельзя добраться (то есть граф несвязный). Тогда алгоритм может быть завершён досрочно.

# 3. Описание алгоритма программы

Для программной реализации алгоритма понадобиться: cost[MAX][MAX], distance[MAX], pred[MAX] – хранит предшественника каждого узла; visited[MAX], count – дает количество узлов, просмотренных на данный момент, mindistance, nextnode, i, j;

Первым пунктом создается матрица cost стоимости из матрицы смежности. Стоимость — переход из вершины I в вершину j. Если же между вершинами I и j нет ребра, то С[i][j] будет равно бесконечности.

Вторым пунктом инициализируется массив visited[] нулем for(i=0;i<n;i++) visited[i]=0;

Третьим пунктом ведется проверка, если вершина 0 является исходной вершиной, тогда visited[0] помечаем как 1.

Четвертым пунктом создается матрица расстояний, сохраняя вес вершин от 0 до n-1 из исходной вершины 0. for(i = 1; I<n; i++) distance[i] = cost[0][i]; Изначально расстояние исходной вершины принимается равным 0, то есть distance[0] = 0;

Пятый пункт for(i=1; i<n; i++) : выбирается вершина w так, чтобы distance[w] было минимальным, а visited[w] было равно нулю; Далее рассчитывается кратчайшее расстояние оставшихся вершин от выбранной вершины.

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

if (distance[i] < mindistance && !visited[i])

{

mindistance = distance[i];

nextnode = i;

}

Проверяется, существует ли лучший путь через следующий узел. Только вершины, которые не отмечены как 1 в массиве посещенных visited[], должны учитываться для пересчета расстояния. То есть для каждой вершины v

if(visited[v]==0)

distance[v] = min(distance[v],

distance[w] + cost[w][v])

visited[nextnode] = 1; for (i = 0; i < n; i++)

if (!visited[i])

if (mindistance + cost[nextnode][i] < distance[i])

{

distance[i] = mindistance + cost[nextnode][i];

pred[i] = nextnode;

}

count++;

}

Ниже представлен псевдокод функции Dijkstra ()

1. Объявляем cost[MAX][MAX], distance[MAX], pred[MAX];

visited[MAX], count, mindistance, nextnode, i, j;

1. для (int I = 0; I < n(размерность матрицы смежности);делать i++)
2. {
3. для (int j = 0; j < n;делать j++)
4. {
5. если (G[i][j] == 0)
6. {
7. стоимость(вес)cost[i][j] = бесконечность;
8. }
9. иначе
10. {
11. cost[i][j] = G[i][j];
12. }
13. для (I = 0; I<n; делать i++)
14. {
15. расстояние [I] = стоимость [начальный узел][i];
16. pred[i] = начальный узел
17. посещенные[i] = 0
18. }
19. расстояние[начальный узел] = 0
20. посещенные[начальный узел] = 1
21. count = 1
22. пока (count < n – 1)
23. {
24. минимальная дистанция = бесконечность;
25. для (I = 0; I<n; делать i++)
26. {
27. если (расстояние[i] < минимальное расстояние && !посещенные[i])
28. {
29. минимальное расстояние = расстояние [I]
30. следующий узел = I;
31. }
32. посещенные[следующий узел] = 1;
33. для (I = 0; I<n; делать i++)
34. если (не посещенные[i])
35. если (минимальное расстояние + стоимость[следующий узел][i] < расстояние[i]
36. {
37. расстояние[i] = минимальное расстояние + стоимость [следующий узел][i]
38. pred[i] = следующий узел
39. }
40. делать count++
41. вывод результата и запись его в файл.

# 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 (Visual C++).

Данная программа является многомодульной, поскольку состоит из нескольких функций: dijkstra(), ruchnoivvod(), autovvod.

Работа программы начинается с вывода меню, где предлагается выбрать вариант ввода матрицы смежности (ручной или автоматический) или выйти из программы.

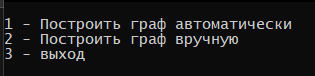


Рисунок 13 – Меню программы

При выборе построения графа автоматически, программа запросит ввести количество вершин графа . После на экран выводится сама матрица смежности и программа попросит ввести начальный узел, от которого будет производиться поиск кратчайшего пути до остальных вершин графа.

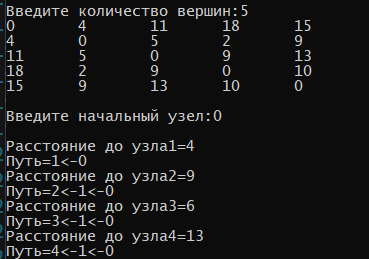


Рисунок 14 – Результат автоматического ввода

printf("Введите количество вершин:");

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

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

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

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

}

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

for (int j = i; j < n; ++j) {

G[i][j] = rand() % 20;

if (i == j) {

G[i][j] = 0;

}

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

}

}

//вывод матрицы смежности:

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

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

printf("%d\t", G[i][j]);

}

printf("\n");

}

printf("\nВведите начальный узел:");

scanf("%d", &u);

}

При выборе построения графа вручную, программа запросит ввести количество вершин графа. После ввода этих данных в программу, в консоль вводим матрицу смежности вручную. Ее заполнение происходит по столбцам. Когда заполнение закончено, программа выводит построенную матрицу смежности, а так же просит ввести начальный узел, от которого будет производиться поиск кратчайшего пути до остальных вершин графа. Затем, программа выводит результат поиска.

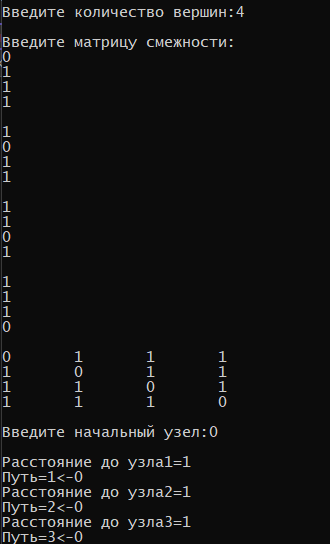


Рисунок 15 – Результат ручного ввода

printf("Введите количество вершин:");

scanf("%d", &n);

printf("\nВведите начальный узел:");

scanf("%d", &u);

printf("\nВведите матрицу смежности:\n");

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

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

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

}

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

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

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

}

printf("\n");

}

При выборе выхода из программы, она завершит свою работу.

case '3':

exit(1);

break;

default:

break;

# 5. Тестирование

Среда разработки MicrosoftVisualStudio 2022 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

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

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод меню программы | Верно |
| Выбор генерации матрицы | Вывод сообщения о количестве вершин в орграфе | Верно |
| Вывод матрицы, введенной автоматически, задание начального узла и поиск кратчайших путей | Вывод матрицы, ввод начального узла вывод и результата поиска | Верно |
| Вывод матрицы, введенной вручную и поиск кратчайших путей | Вывод матрицы, ввод начального узла и вывод результата поиска | Верно |
| Запись в файл матрицы и результата поиска | Запись производится в файл | Верно |

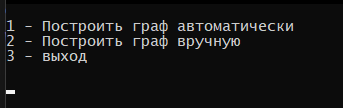


Рисунок 16 – Вывод меню программы



Рисунок 17 – Выбор количества вершин и начального узла

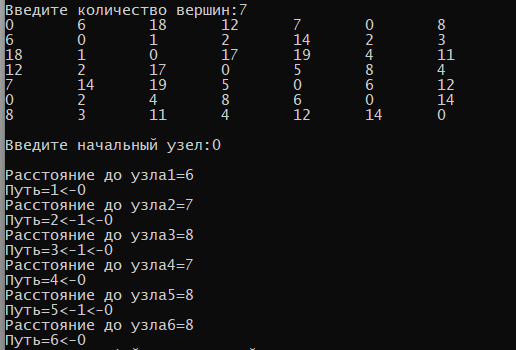


Рисунок 18 – Вывод матрицы и результата поиска

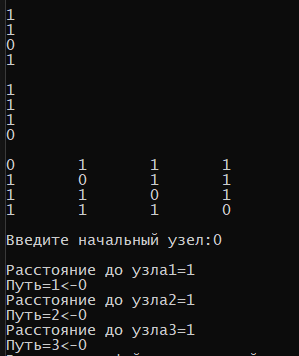


Рисунок 19 – Вывод матрицы и результата поиска (ручной).



Рисунок 20 – Сохранение результата в файл.

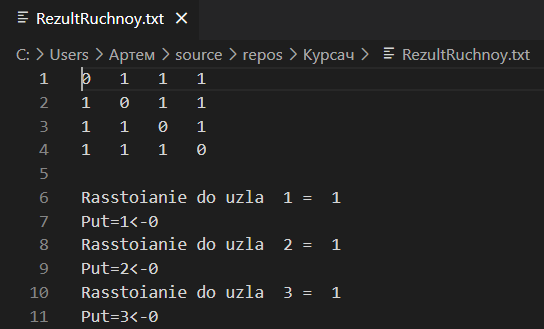


Рисунок 21 – Записанный результат.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин и вывод поиска кратчайших путей.

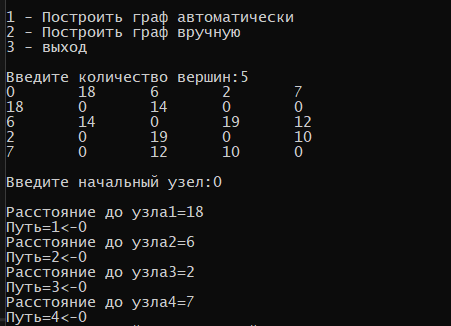


Рисунок 22 – Тестирование при вводе количество вершин = 5 и начальным узлом = 0.

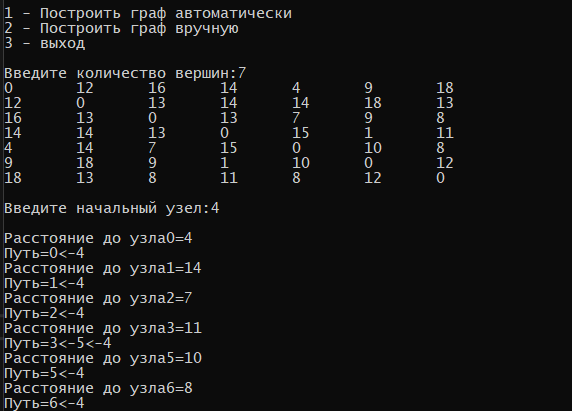


Рисунок 23 – Тестирование при вводе количество вершин = 7 и начальным узлом = 4.

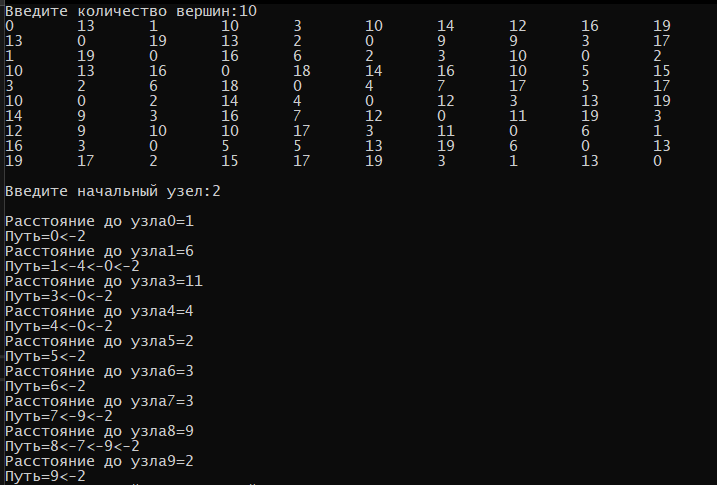


Рисунок 24 – Тестирование при вводе количество вершин = 10 и начальным узлом = 2.

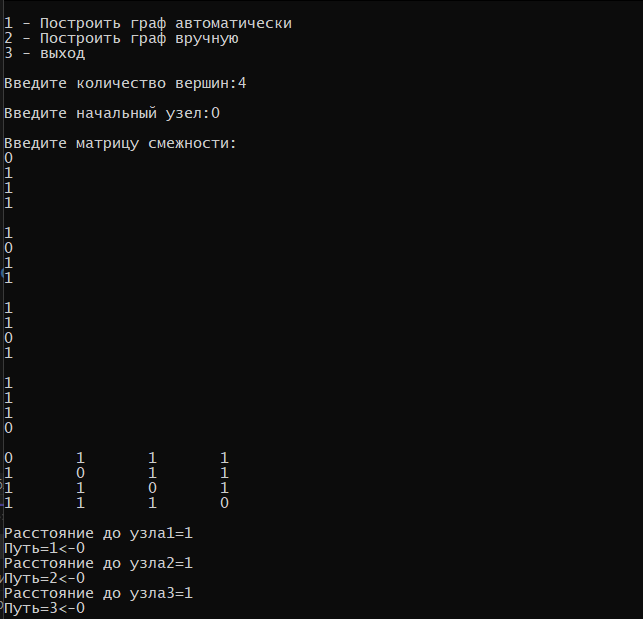


Рисунок 25 – Тестирование ручного ввода при 4 вершинах и начальном узле = 0.

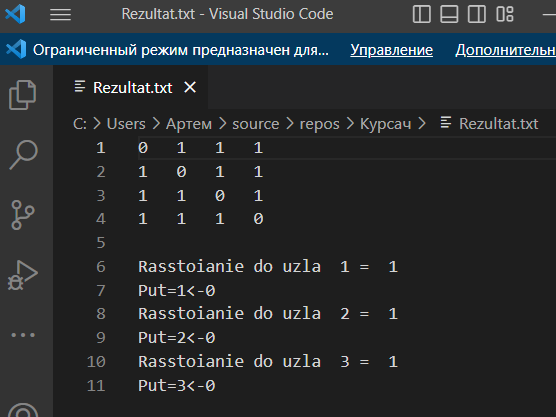


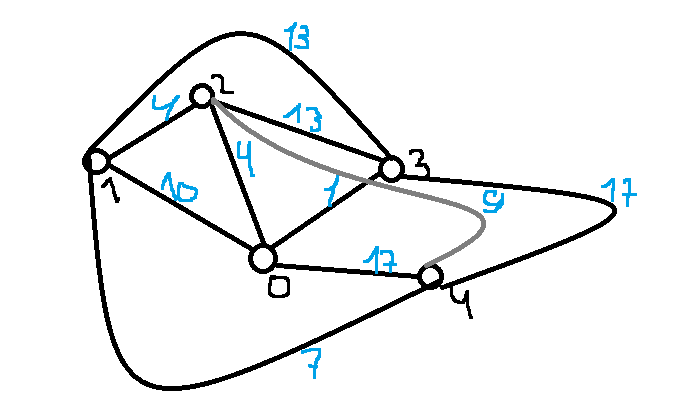
Рисунок 26 – запись результата в файл.

В результате тестирования было выявлено, что программа успешно справляется с поиском кратчайших путей в матрице.

# 6. Ручной расчѐт задачи

Проведем проверку программы посредством ручных вычислений на примере графа с 5 вершинами (рисунок 16).

Для удобства поиска построим граф графически.

Рисунок 27 – Построение графа.

Начинаем обход из 0 вершины в последующие. Проверяем, если есть путь из 0 в другие вершины, то дальше идем. Из 0 в 1 существует путь напрямую, но он равен 10, поэтому ищем самый короткий путь. При проходе 0→ 2→ 1 путь = 8. При 0→ 3 → 1 = 14. Тем самым проходим по всем возможным путям и убеждаемся, что самым коротким путем является 0→ 2→ 1 с расстоянием равным 8. Программа аналогично вывела 8. Проверяем расстояние до 2 узла. 0→ 2 = 4; 0→ 1→ 2 = 14; 0→ 3→ 2 = 14; 0→ 4 → 2 = 26; и тд. Кротчайший путь = 4. Проверяем расстояние до 3 узла. 0→ 3 = 1 Это и будет самый кротчайший путь, так как меньше единицы путь быть не может. Проверяем расстояние до 4 узла. 0→ 4 = 17; 0→1 → 4 = 17; 0→ 2→ 4 = 13; 0→ 3→ 4 = 20; и так далее. Кротчайшим расстоянием будет путь 0→ 2→ 4 = 13. Таким образом, можно сделать вывод, что программа работает верно.

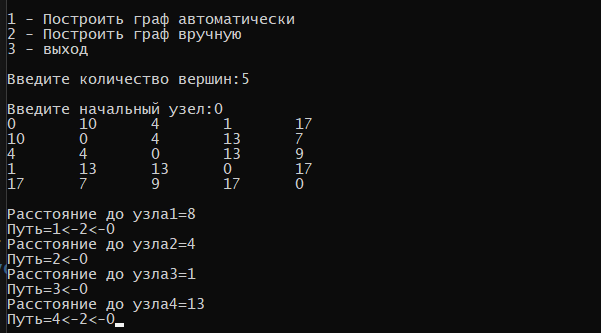


Рисунок 28 – Тестирование работы программы

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм поиска кратчайших путей графов в MicrosoftVisualStudio 2022.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, основанных на теории орграфов. Приобретены навыки по осуществлению алгоритма поиска кратчайших путей. Углублены знания языка программированияCи.

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. Тим Рфгарден «Совершенный алгоритм».
2. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: Построение и анализ - М.: МЦНМО, 2001. - 960 с.
3. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978
4. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006
5. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.
6. Харви Дейтел, Пол Дейтел. Как программировать на C/C++. 2009 г.
7. 3. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с.

**Приложение А.**

# Листинг программы.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<conio.h>

#include <iostream>

#include <Windows.h>

#define INFINITY 9999

#define MAX 100

FILE\* Rezultat\_1,Rezultat\_2,Rezultat\_3;

char RezultatQ[20];

int\*\* G;

int n;

int u;

void dijkstra(int startnode)

{

int cost[MAX][MAX], distance[MAX], pred[MAX];

int visited[MAX], count, mindistance, nextnode, i, j;

//pred[] хранит предшественника каждого узла

//count дает количество узлов, просмотренных на данный момент

//создайте матрицу смежности

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

{

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

{

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

{

cost[i][j] = INFINITY;

}

else

{

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

}

}

}

//инициализировать pred[],distance[] и visited[]

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

{

distance[i] = cost[startnode][i];

pred[i] = startnode;

visited[i] = 0;

}

distance[startnode] = 0;

visited[startnode] = 1;

count = 1;

while (count < n - 1)

{

mindistance = INFINITY;

//следующий узел дает узел на минимальном расстоянии

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

if (distance[i] < mindistance && !visited[i])

{

mindistance = distance[i];

nextnode = i;

}

//проверьте, существует ли лучший путь через следующий узел

visited[nextnode] = 1;

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

if (!visited[i])

if (mindistance + cost[nextnode][i] < distance[i])

{

distance[i] = mindistance + cost[nextnode][i];

pred[i] = nextnode;

}

count++;

}

//выведите путь и расстояние до каждого узла

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

if (i != startnode)

{

printf("\nРасстояние до узла%d=%d", i, distance[i]);

printf("\nПуть=%d", i);

j = i;

do

{

j = pred[j];

printf("<-%d", j);

} while (j != startnode);

}

printf("\n");

printf("Введите имя файла, в который хотите записать ответ:\n");

scanf("%s",&RezultatQ, 20);

Rezultat\_1 = fopen(RezultatQ, "w+");

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

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

fprintf(Rezultat\_1, "%d\t", G[i][j]);

}

fprintf(Rezultat\_1, "\n");

}

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

if (i != startnode)

{

fprintf(Rezultat\_1, "\nRasstoianie do uzla % d = % d", i, distance[i]);

fprintf(Rezultat\_1, "\nPut=%d", i);

j = i;

do

{

j = pred[j];

fprintf(Rezultat\_1,"<-%d", j);

} while (j != startnode);

}

printf("Результат записан!\n");

fclose(Rezultat\_1);

}

void ruchnoivvod() {

//int u;

printf("Введите количество вершин:");

scanf("%d", &n);

printf("\nВведите матрицу смежности:\n");

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

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

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

}

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

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

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

}

printf("\n");

}

//вывод матрицы:

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

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

printf("%d\t", G[i][j]);

}

printf("\n");

}

printf("\nВведите начальный узел:");

scanf("%d", &u);

}

void autovvod() {

srand(time(NULL));

printf("Введите количество вершин:");

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

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

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

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

}

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

for (int j = i; j < n; ++j) {

G[i][j] = rand() % 20;

if (i == j) {

G[i][j] = 0;

}

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

}

}

//вывод матрицы смежности:

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

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

printf("%d\t", G[i][j]);

}

printf("\n");

}

printf("\nВведите начальный узел:");

scanf("%d", &u);

}

int main()

{

setlocale(LC\_ALL, "Russian");

SetConsoleCP(866);

printf("\n1 - Построить граф автоматически\n2 - Построить граф вручную\n3 - выход\n\n");

while (true) {

int l = \_getch();

switch (l) {

case '1':

autovvod();

dijkstra(u);

break;

case '2':

ruchnoivvod();

dijkstra(u);

break;

case '3':

exit(1);

break;

default:

break;

}

}

return 0;

}