Министерство 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 Дейкстры»

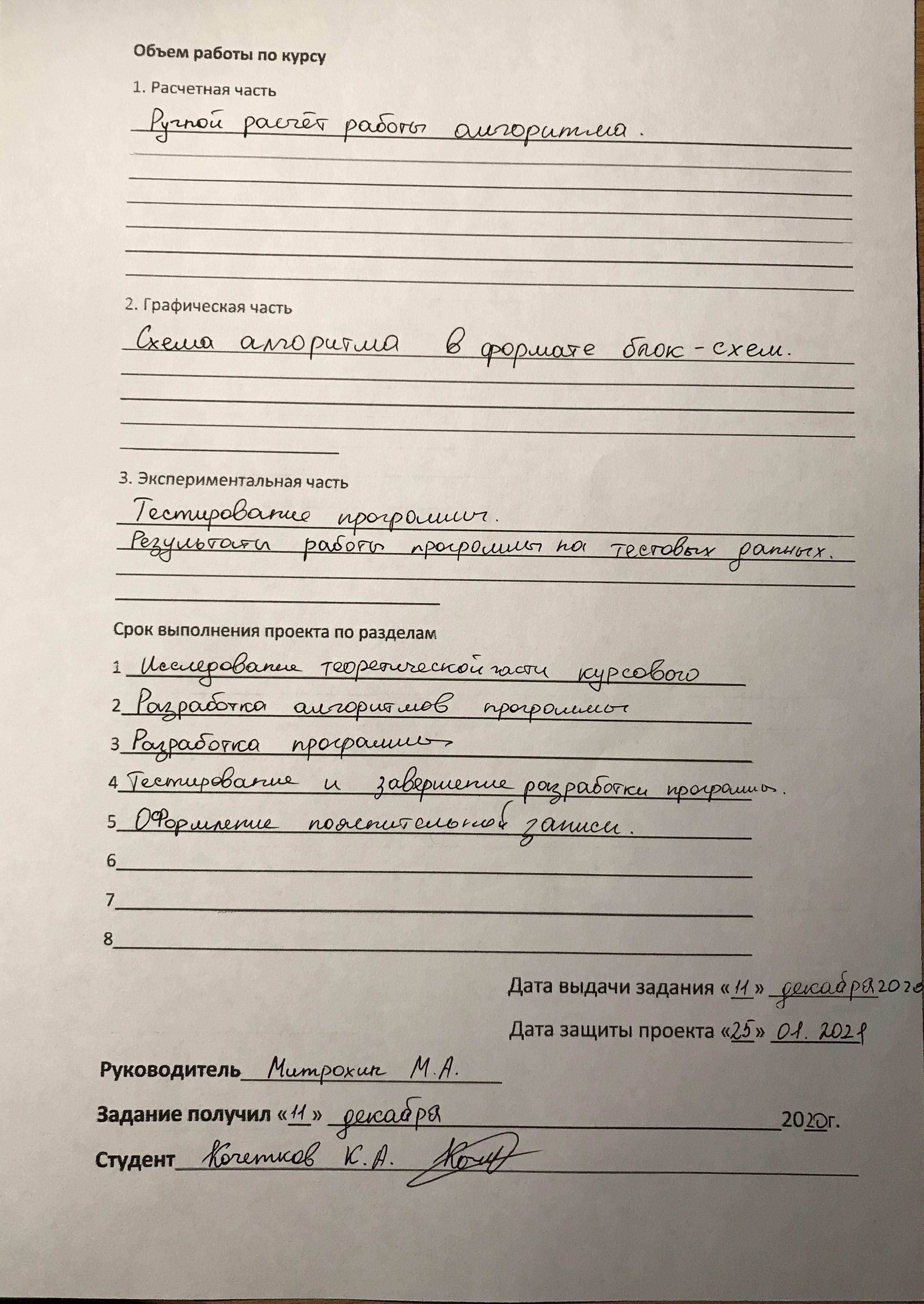
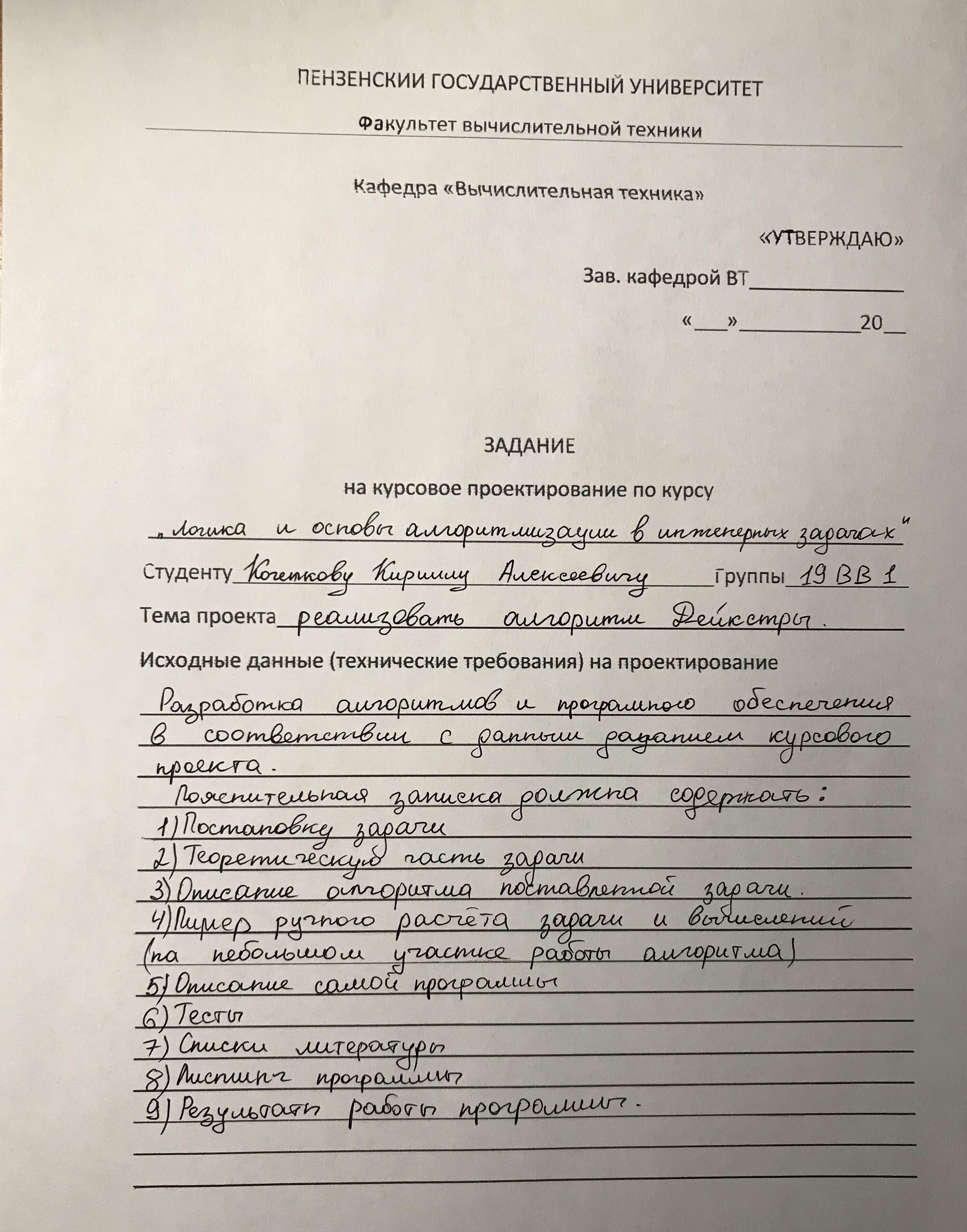
Выполнил студент группы 19ВВ1:

Кочетков К.А.

Принял:

Митрохин М.А.

Пенза 2020



**Оглавление:**

Реферат

Введение

1.Постановка задачи

2.Теоретическая часть

3.Описание алгоритма программы

4.Описание программы

5.Тестирование

6.Ручной расчёт программы

Заключение

Список литературы

7.Приложение А

# Реферат

АЛГОРИТМ ДЕЙКСТРЫ

Цель исследования – реализация алгоритма Дейкстры для нахождения кратчайшего пути.

В работе представлен алгоритм Дейкстры.

# Введение

**Алгори́тм Де́йкстры** (англ. *Dijkstra’s algorithm*) — алгоритм на графах, изобретённый нидерландским учёным Эдсгером Дейкстрой в 1959 году. Находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса. Алгоритм широко применяется в программировании и технологиях, например, его используют протоколы маршрутизации OSPF и IS-IS.

В качестве среды разработки мною была выбрана Microsoft VisualStudio2010, 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 Дейкстры, осуществляющий нахождение кратчайшего пути.

# 1.Постановка задачи

Требуется реализовать программу, которая осуществит поиск кратчайшего пути до каждой из вершин.

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 , расстояние до каждой из вершин и путь до конечной вершины. Необходимо предусмотреть, чтобы программа не выдавала ошибок и работа правильно. Устройство ввода -- клавиатура и мышь.

Задание выполняются в соответствии с вариантом №13.

# 2.Теоретическая часть

Граф matr (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.

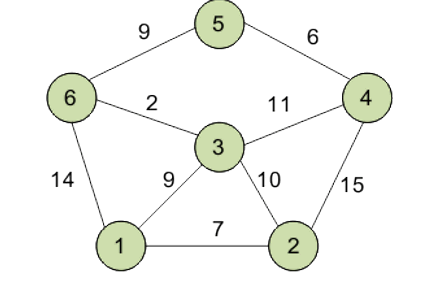


Рисунок 1 – Неориентированный граф

При 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 выберем вершину и проверяем расстояния до её соседей , затем выбираем соседа с наименьшим весом и проверяем всех соседей уже для этой вершины.(Так делаем с каждой из вершин)

Результатом работы программы будут наименьшие расстояния до каждой из вершин.

# 3.Описание алгоритма программы

Для программной реализации алгоритма понадобятся три массива:

d (int) – массив для записи минимальных расстояний,

arr (int) – для возможности динамического ввода , хранения данных,

v (int) – для записей вершин.

Если вершина ещё не посещена и вес < min , тогда эта величина берётся минимальным расстоянием , и если алгоритм не найдёт более короткого пути, тогда эта величина и будет минимальным расстоянием до данной вершины.

Алгоритм выполняется до тех пор, пока не получатся минимальные возможные расстояния.

Ниже приведен псевдокод функции main.

**Main()**

1.вывод «меню»

2.вывод «ввести матрицу с клавиатуры»

3.вывод «сгенерировать матрицу автоматически»

4.вывод «выход»

5.если menu==3

6.exit(0)

7.вывод «введите количество вершин»

8.ввод «m»

9.если menu==2

10. arr = (int\*\*)malloc(m \* sizeof(int));

11. i=0 пока i<m делать i++

12.arr[i]=(int\*)malloc(m \* sizeof(int));

13. j=0 пока j<m делать j++

14. Ввод arr[i][j] при помощи rand

15.конец цикла

16.если menu==1

17. arr = (int\*\*)malloc(m \* sizeof(int));

18. i=0 пока i<m делать i++

19. arr = (int\*\*)malloc(m \* sizeof(int));

20. j=0 пока j<m делать j++

21.arr[i]=(int\*)malloc(m \* sizeof(int));

22.если j<i

23.ввод arr[i][j] с клавиатуры

24.конец условия

25.если i==j

26.arr[i][j]=0

27.конец условия

28.конец цикла

29.i=0 пока i<m делать i++

30. j=0 пока j<m делать j++

31.если i<j

32.arr[i][j]=arr[j][i]

33.конец условия

34.вывод arr[i][j]

35.конец цикла

36.ввод «start»

37.d=(int\*)malloc(m \* sizeof(int));

38.v=(int\*)malloc(m \* sizeof(int));

39.ver=(int\*)malloc(m \* sizeof(int));

40.i=0 пока i<m делать i++

41.d[i] = 10000;

42. v[i] = 1;

43.ver[i]=1;

44.конец цикла

45.d[start] = 0

46.mini = 10000;

47.min = 10000;

48.i=0 пока i<m делать i++

49.если((v[i] == 1) && (d[i]<min))

50.min = d[i];

51.mini = i;

52.конец цикла

53.если (mini != 10000)

54.i=0 пока i<m делать i++

55.если (arr[mini][i] > 0)

56.temp = min + arr[mini][i]

57.если (temp < d[i])

58. d[i] = temp;

59.конец условия

60.конец условия

61.конец цикла

62. v[mini] = 0;

63.конец условия

64.пока (end != start)

65.i=0 пока i<m делать i++

66.если (arr[i][end] != 0)

67.int temp = wes - arr[i][end]

68.если (temp == d[i])

69.wes = temp;

70.end = i

71.ver[k] = i + 1

72.k++

73.конец условия

74.конец условия

75.конец цикла

76. i=k-1 пока i>=0 делать i—

77.вывод пути

78.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.

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

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.

for(int i=0;i<1;){

printf("МЕНЮ:\n1. Cгенерировать матрицу смежности самим(1)\n2. Вывести её рандомно(2)\n3. Выход.(3)\n");

scanf("%d",&menu);

if (menu==1||menu==2||menu==3){i++;}else{system("cls");}

}if (menu==3){exit(0);} int m;

int\*\* arr;

printf("Введите количество вершин ");

scanf("%d",&m);

if (menu==2){

printf("Матрица M1:\n");

arr = (int\*\*)malloc(m \* sizeof(int));

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

arr[i]=(int\*)malloc(m \* sizeof(int));

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

if(i<m){arr[i][j]=rand()%10;}

if(i==j){arr[i][j]=0;}

if(j<i){arr[i][j]=arr[j][i];}

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

}

printf("\n");

}}

if (menu==1){

int t;

arr = (int\*\*)malloc(m \* sizeof(int));

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

arr[i]=(int\*)malloc(m \* sizeof(int));

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

if(j<i){ printf("Введите расстояние %d - %d: ", j + 1, i + 1); scanf("%d", &t);arr[i][j]=t;}

if(i==j){arr[i][j]=0;}

}}

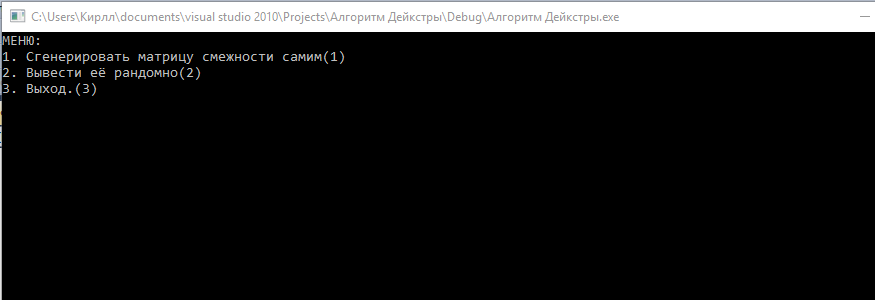


Рис. 2 Меню.

После выбора пользователем варианта генерации матрицы происходит ее вывод:

printf("Матрица M1:\n");

arr = (int\*\*)malloc(m \* sizeof(int));

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

arr[i]=(int\*)malloc(m \* sizeof(int));

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

if(i<m){arr[i][j]=rand()%10;}

if(i==j){arr[i][j]=0;}

if(j<i){arr[i][j]=arr[j][i];}

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

}

printf("\n");

}}

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

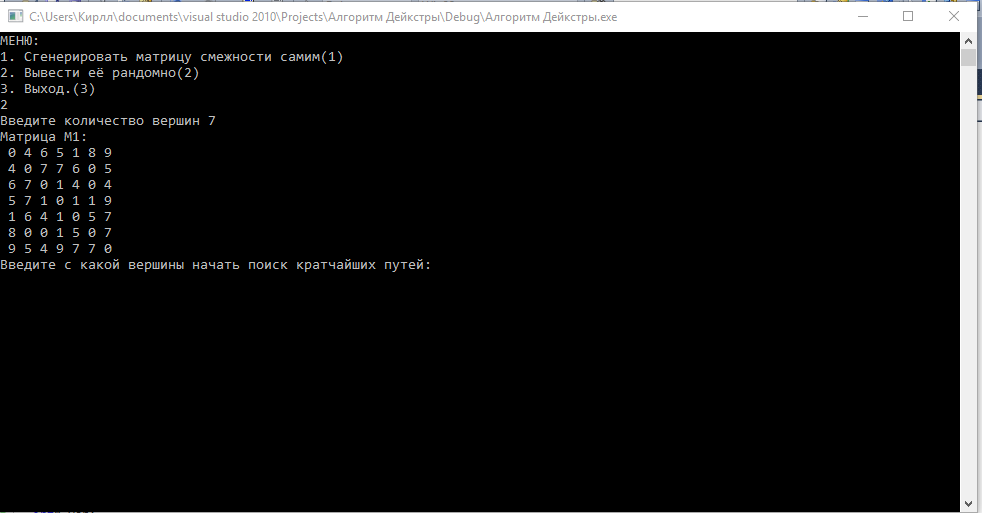


Рис.3 Случайная генерация матрицы.

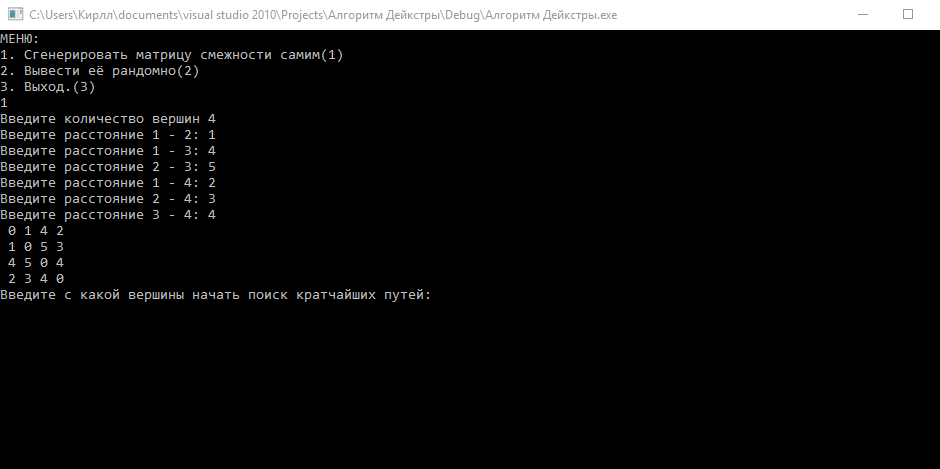


Рис.4 Ввод матрицы с клавиатуры.

# 5.Тестирование

Среда разработки MicrosoftVisualStudio 2010 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.

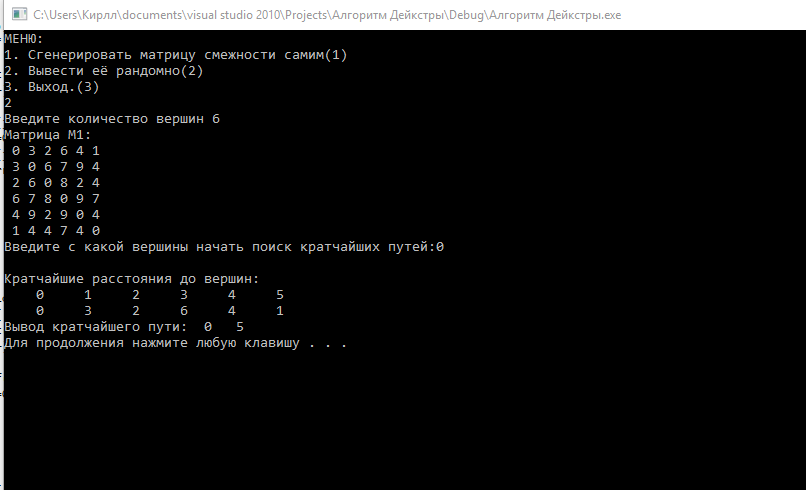


Рисунок 5 – Работа программы при автоматической

генерации матрицы 6х6

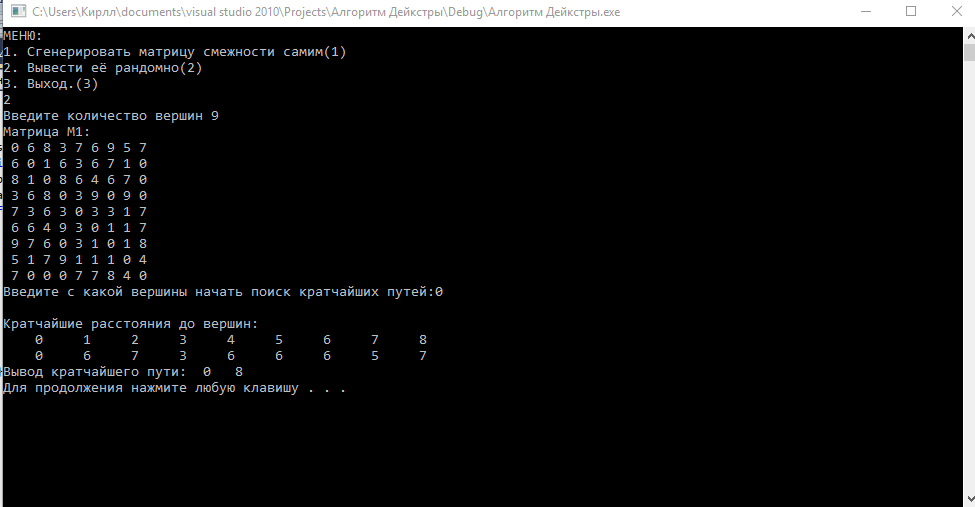


Рисунок 6 - Работа программы при автоматической генерации матрицы 9х9

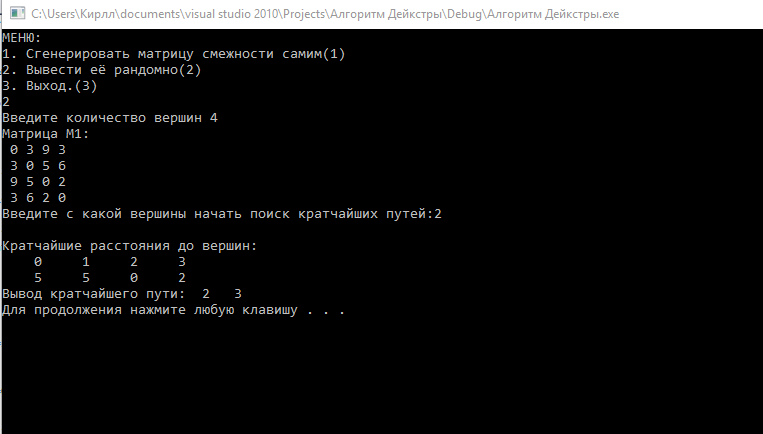


Рисунок 7 - Работа программы при автоматической генерации матрицы 4х4

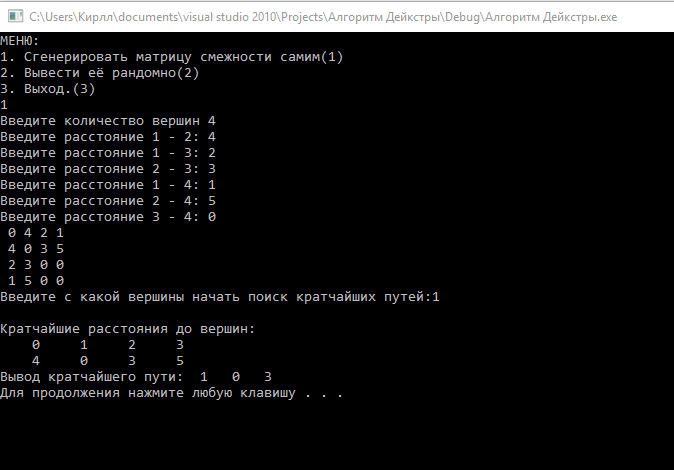


Рисунок 8 - Работа программы при ручном вводе матрицы 4х4

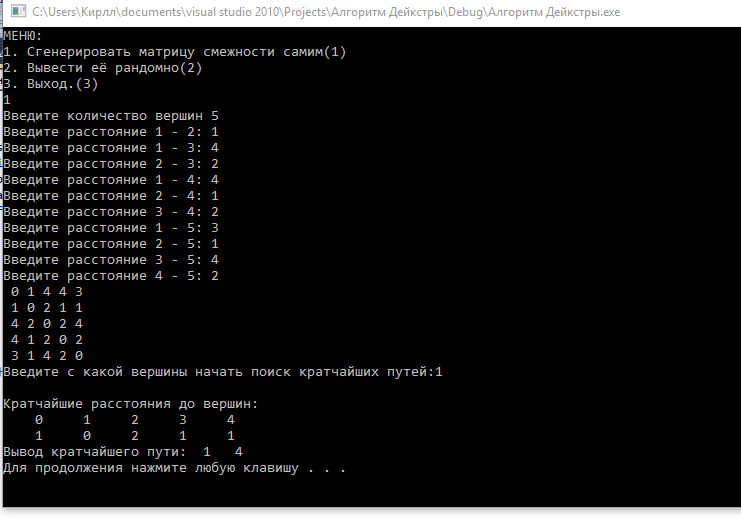


Рисунок 9 - Работа программы при ручном вводе матрицы 5х5

|  |  |  |
| --- | --- | --- |
| 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 7).

Свой расчёт мы начинаем со 2 вершины , как и было задано программе. Сразу видно что кратчайший путь из 2 вершины в 3 = 2, что соответствует работе нашего алгоритма. Если войти в 3 вершину то мы замечаем путь из 3 вершины в 0 и сумма такого пути будет = 5 , такой же результат и получился у нашей программы. Ну и осталось посетить вершину 1 , короче пути чем из вершины 2 в 1 нет , и путь будет равен 5 , это соответствует результату нашуй программы.

2-3 = 2

2-0 = 5 (маршрутом 2-3-0)

2-1 = 5

2-2 = 0 (мы с неё начинали)

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 Дейкстры в MicrosoftVisualStudio 2010.

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. Эдгар Дейкстра - Дисциплина программирования

2.Герберт Шилдт «полный справочник по С++» - вильямс,2006

3.Оре О. Графы и их применение: Пер.сангл.1965.176с

# 7.Приложение А

**Листинг программы:**

#include <iostream>

#include <stdio.h>

#include <locale>

#include <malloc.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

int main()

{

system("chcp 1251");

system("cls");

srand(time(NULL));

int menu;

for(int i=0;i<1;){

printf("МЕНЮ:\n1. Cгенерировать матрицу смежности самим(1)\n2. Вывести её рандомно(2)\n3. Выход.(3)\n");

scanf("%d",&menu);

if (menu==1||menu==2||menu==3){i++;}else{system("cls");}

}if (menu==3){exit(0);}

int m;

int\*\* arr;

printf("Введите количество вершин ");

scanf("%d",&m);

if (menu==2){

printf("Матрица M1:\n");

arr = (int\*\*)malloc(m \* sizeof(int));

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

arr[i]=(int\*)malloc(m \* sizeof(int));

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

if(i<m){arr[i][j]=rand()%10;}

if(i==j){arr[i][j]=0;}

if(j<i){arr[i][j]=arr[j][i];}

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

}

printf("\n");

}}

if (menu==1){

int t;

arr = (int\*\*)malloc(m \* sizeof(int));

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

arr[i]=(int\*)malloc(m \* sizeof(int));

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

if(j<i){ printf("Введите расстояние %d - %d: ", j + 1, i + 1); scanf("%d", &t);arr[i][j]=t;}

if(i==j){arr[i][j]=0;}

}

}

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

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

if(i<j){arr[i][j]=arr[j][i];}

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

}

printf("\n");

}}

int\* ver;

int\* d;

int\* v;

int temp, mini, min;

int start;

printf("Введите с какой вершины начать поиск кратчайших путей:");

scanf("%d",&start);

d=(int\*)malloc(m \* sizeof(int));

v=(int\*)malloc(m \* sizeof(int));

ver=(int\*)malloc(m \* sizeof(int));

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

{

d[i] = 10000;

v[i] = 1;

ver[i]=1;

}

d[start] = 0;

do {

mini = 10000;

min = 10000;

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

{

if ((v[i] == 1) && (d[i]<min))

{

min = d[i];

mini = i;

}

}

if (mini != 10000)

{

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

{

if (arr[mini][i] > 0)

{

temp = min + arr[mini][i];

if (temp < d[i])

{

d[i] = temp;

}

}

}

v[mini] = 0;

}

} while (mini < 10000);

printf("\nКратчайшие расстояния до вершин: \n");

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

printf("%5d ", i);}

printf("\n");

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

printf("%5d ", d[i]);

int end = m-1;

ver[0] = end + 1;

int k = 1;

int wes = d[end];

while (end != start)

{

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

if (arr[i][end] != 0)

{

int temp = wes - arr[i][end];

if (temp == d[i])

{

wes = temp;

end = i;

ver[k] = i + 1;

k++;

}

}

}

printf("\nВывод кратчайшего пути:");

for (int i = k - 1; i >= 0; i--)

printf("%3d ", ver[i]-1);

printf("\n");

system("pause");

}