

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 «Нахождение минимальных путей в графе»

Выполнила:
студентка группы 18ВВ2
Алпатова А. А.

Принял:
д.т.н., профессор
Митрохин М. А.

Пенза 2019

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет вычислительной техники

Кафедра

Вычислительная техника

«УТВЕРЖДАЮ»

Зав. кафедрой

«__» сентября 201__ г.

ЗАДАНИЕ

на курсовое проектирование по курсу

Логика и основы алгоритмизации вычислительных задач

Студенту Анатолий А. А. Группы 18 ВВ2

Тема проекта Нахождение минимальных путей от фиксированной вершины до произвольной вершины графа (алгоритмы Дейкстры)

Исходные данные (технические требования) на проектирование

Разработать алгоритмы и программное обеспечение в соответствии с данными заданием курсового проекта.

Техническая документация должна содержать:

1. Постановка задачи;
2. Теоретическое решение задачи;
3. Описание алгоритма поставленной задачи;
4. Пример ручного решения задачи и вычислений (на небольшом участке работы алгоритма);
5. Описание самой программы
6. Тесты;
7. Список литературы
8. Листинг программы;
9. Резюме работы программы.

Объем работы по курсу

1. Расчетная часть

Ручной расчет работ алгоритма

2. Графическая часть

Схема алгоритма в формате блок-схем

3. Экспериментальная часть

Тестирование программы;
Разработка работ программы на тестовых данных

Срок выполнения проекта по разделам

- | | | | |
|---|---|---|----------|
| 1 | Исследование термодинамических процессов по учебнику с 02.09.19 | к | 12.10.19 |
| 2 | Разработка алгоритмов программы с 12.10.19 | к | 19.10.19 |
| 3 | Разработка программы | к | 10.11.19 |
| 4 | Тестирование и завершение разработки программы | к | 27.11.19 |
| 5 | Оформление пояснительной записки с 27.11.19 | к | 06.12.19 |

Дата выдачи задания « 2 » сентября 2019 г.

Дата защиты проекта « ___ » декабря 2019 г.

Руководитель Мельников М. А.

Задание получил « 2 » сентября 2019 г.

Студент Антонова А. А.

Содержание

Введение.....	3
1. Постановка задачи.....	4
2. Теоретическая часть задания.....	5
3. Описание алгоритма поставленной задачи.....	7
4. Описание программы.....	10
5. Тестирование программы.....	12
5.1. Тесты	12
5.2. Пример ручного расчета задачи и вычислений	15
Заключение.....	16
Список используемых источников.....	17
Приложение А. Листинг программы.....	18

Введение

Благодаря своему широкому применению, теория о нахождении кратчайших путей в последнее время интенсивно развивается. Задача о поиске кратчайшего пути на графе может быть интерпретирована по-разному и применяться в различных областях: в картографических сервисах для нахождения путей между физическими объектами, недетерминированных машинах для поиска оптимальной последовательности решений для достижения главной цели, для поиска кратчайшего расстояния в сети дорог.

Компьютерная техника и информационные технологии позволяют моделировать объекты, явления и процессы, тем самым упрощая решение различных задач.

Для реализации компьютерной модели, большое значение имеет такое научное направление, как программирование. Без него компьютер это просто набор различных устройств, микросхем, который не может быть полезным.

Целью данной курсовой работы является разработка программы на языке С, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм Дейкстры, осуществляющий поиск минимальных путей.

1 Постановка задачи

Необходимо осуществить программную реализацию алгоритма поиска кратчайшего пути (алгоритма Дейкстры) из фиксированной вершины графа до всех остальных.

Программа должна работать так, чтобы пользователь вводил начальную вершину, от которой необходимо найти все кратчайшие расстояния, вводил вес каждого ребра. После обработки этих данных на экран должна выводиться матрица смежности взвешенного графа с начальной вершиной 1 и вектор, отражающий все кратчайшие расстояния от заданной вершины. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь. Необходимо научиться различать события, возникающие в процессе работы с устройствами ввода. Необходимо однозначно идентифицировать и выполнять те или иные действия в зависимости от выбранной команды.

Данная программа может использоваться в дискретной математике для исследования графов или в качестве наглядного пособия, демонстрирующего применение алгоритма Дейкстры на практике.

2 Теоретическая часть задания

Теория графов - один из обширнейших разделов дискретной математики, широко применяется в решении экономических и управленческих задач, в программировании, химии, конструировании и изучении электрических цепей, коммуникации, психологии, психологии, социологии, лингвистике, других областях знаний. Теория графов систематически и последовательно изучает свойства графов. Основателем теории графов считается Леонард Эйлер (1707-1882), решивший в 1736 году известную в то время задачу о кёнигсбергских мостах.

Граф G (рисунок 1) задается множеством точек (вершин) $x_1, x_2, \dots, x_n$ (которое обозначается через X) и множеством линий (ребер) $a_1, a_2, \dots, a_m$ (которое обозначается символом A), соединяющих между собой все или часть этих точек. Таким образом, граф G полностью задается (и обозначается) парой (X, A) . Если ребра из множества A ориентированы, что обычно показывается стрелкой, то они называются дугами, и граф с такими ребрами называется ориентированным графом.

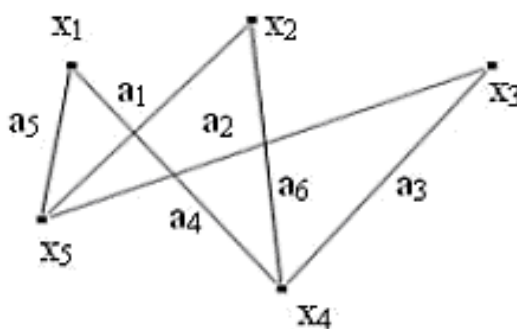


Рисунок 1 – Неориентированный граф

При представлении графа матрицей смежности информация о ребрах графа хранится в квадратной матрице (двумерном списке), где элемент $G[i][j]$ равен 1, если ребра i и j соединены ребром и равен 0 в противном случае.

Если граф неориентированный, то матрица смежности всегда симметрична относительно главной диагонали.

При использовании матрицы смежности удобно проверять соединены ли две вершины ребром – это просмотр одного элемента матрицы $G[i][j]$, но сложнее перебирать все ребра, исходящие из данной вершины (для этого необходимо перебрать все оставшиеся вершины и проверить, соединены ли они ребром). Также матрица смежности требует $O(n^2)$ памяти и может оказаться неэффективным способом хранения дерева или разреженных графов.

При помощи матриц смежности можно представлять и неориентированные графы. В случае матрицы смежности $G[i][j]$ будет равно 1, если есть ребро, начинающееся в вершине i и заканчивающееся в вершине j .

Очень часто рассматриваются графы, в которых каждому ребру приписана некоторая числовая характеристика – вес. Вес может означать длину дороги или стоимость проезда по данному маршруту. Соответствующие графы называются взвешенными.

При представлении графа матрицей смежности вес ребра можно хранить в матрице, то есть $G[i][j]$ в данном случае будет равно весу ребра из i в j . При этом при отсутствии ребра можно хранить специальное значение. Во многих задачах удобно при отсутствии ребра хранить очень большое число, в этом случае отсутствие ребра аналогично наличию ребра очень большой стоимости.

Предусматривается совершение различных операций над графами. Одной из таких задач является поиск кратчайшего пути. Одним из наиболее эффективных алгоритмов нахождения кратчайшего пути является алгоритм Дейкстры, которому посвящена данная курсовая работа.

3 Описание алгоритма поставленной задачи

Для программной реализации алгоритма понадобятся два массива: логический `visited` (`bool`) – для хранения информации о посещенных вершинах и численный `distance` (`int`), в который будут заноситься найденные кратчайшие пути. Итак, имеется граф $G=(V, E)$. Каждая из вершин входящих во множество V , изначально отмечена как не посещенная, т. е. элементам массива `visited` присвоено значение `false`.

Поскольку самые выгодные пути только предстоит найти, в каждый элемент вектора `distance` записывается такое число, которое заведомо больше любого потенциального пути (обычно это число называют бесконечностью, но в программе используют, например максимальное значение конкретного типа данных). В качестве исходного пункта выбирается вершина s и ей приписывается нулевой путь: `distance[s]=0`, т. к. нет ребра из s в s (метод не предусматривает петель).

Далее, находятся все соседние вершины (в которые есть ребро из s) и поочередно исследуются, а именно вычисляется стоимость маршрута из s поочередно в каждую из них.

Но вполне вероятно, что в ту или иную вершину из s существует несколько путей, поэтому цену пути в такую вершину в массиве `distance` придется пересматривать, тогда наибольшее (неоптимальное) значение игнорируется, а наименьшее ставится в соответствие вершине.

После обработки смежных с s вершин она помечается как посещенная: `visited[s] = true`, и активной становится та вершина, путь из s в которую минимален. Далее вершина помечается в массиве `visited` (`true`), активной становится следующая вершина, и вся процедура повторяется для нее. Алгоритм Дейкстры продолжается до тех пор, пока все доступные из s вершины не будут исследованы.

Ниже представлен псевдокод функции `algorithm()`, с помощью которой осуществляется поиск кратчайших путей в программе (алгоритм Дейкстры).

1. пока `s != size`
2. для `i=0` пока `i < size` делать `i=i+1`
3. `distance[i] =` максимальное число
4. `visited[i] =` истина
5. конец цикла
6. `distance[s] = 0`
7. делать пока `mIndex <` максимальное число
8. `mIndex =` максимальное число
9. `min =` максимальное число
10. для `i=0` пока `i < size` делать `i=i+1`
11. если `visited[i] =` истина и `distance[i] = min`
12. `min = distance[i]`
13. `mIndex = i`
14. конец условия
15. конец цикла
16. если `mIndex !=` максимальное число
17. для `i=0` пока `i < size` делать `i=i+1`
18. `V = min + G[mIndex][i]`
19. если `V < distance[i]` `distance[i] = V`
20. конец условия
21. конец цикла
22. `visited[mIndex] =` ложь
23. конец условия
24. конец цикла
25. `s = s+1`
26. вывод "Кратчайшие расстояния от вершины под номером `s`: "
27. для `i=0` пока `i < size` делать `i=i+1`

28. вывод $i+1$
29. для $i=0$ пока $i < \text{size}$ делать $i=i+1$
30. вывод $\text{distance}[i]$
31. конец цикла
32. конец цикла

Полный код программы можно увидеть в Приложении А.

4 Описание программы

Для написания данной программы использован язык программирования Си. Си - язык программирования общего назначения, который очень известен своей эффективностью, экономичностью ресурсов и переносимостью на других программные платформы. Все эти преимущества обеспечивают качество разработки программного обеспечения, их быстроту и сравнительно небольшой размер.

В качестве среды программирования был выбран программный продукт Microsoft Visual Studio 2017. Проект был создан в виде консольного приложения Win32 (Visual C++).

Данная программа является многомодульной, поскольку состоит из нескольких функций: `main`, `inicializaciaM`, `vivodM`, `inicializaciaV`, `algorithm`, `vivodR`.

Работа программы начинается с запроса введения количества вершин. Пользователь будет предупрежден о том, что дробная часть будет отброшена при ее наличии. Затем выводятся элементы массива, а именно матрица смежности графа со случайными значениями веса ребер. Для удобства пользователя все вершины пронумерованы и разделены знаками тире. Ниже можно увидеть оформление матрицы смежности (начальную часть программы).

```
Введите количество вершин графа: 4
Дробная часть будет отброшена!

      -1-  -2-  -3-  -4-
-1-    0    7    1    8
-2-    7    0    2    9
-3-    1    2    0    3
-4-    8    9    3    0
```

Рисунок 2 – Начальная часть программы

Далее предоставляется возможность заменить вес любого ребра случайно сгенерированной матрицы. Таким образом, все значения могут быть заданы пользователем, поскольку замену элементов массива можно сделать неограниченное число раз. При этом в программе будет выведено предупреждение при выборе двух смежных ребер, так как создастся петля. Затем на экран будет выведена новая измененная матрица. На следующем рисунке представлен скриншот, демонстрирующий возможность пользователя многократно изменять веса ребер.

```

Хотите ли Вы заменить вес ребра? Ваш ответ (1 - да, 2 - нет): 1
Введите номер вершины, вес от которой необходимо заменить: 2
Введите номер ее смежной вершины: 2
Вы уверены, что хотите создать петлю? Ваш ответ (1 - да, 2 - нет): 2

Введите номер вершины, вес ребра от которой необходимо заменить: 2
Введите номер ее смежной вершины: 4
Введите неотрицательный вес между вершинами 2 - 4: 12
Хотите ли Вы заменить вес ребра? Ваш ответ (1 - да, 2 - нет): 1

-1-  -2-  -3-  -4-
-1-   0   1   0   2
-2-   1   0   9  12
-3-   0   9   0   4
-4-   2  12   4   0

Введите номер вершины, вес от которой необходимо заменить: 4
Введите номер ее смежной вершины: 2
Введите неотрицательный вес между вершинами 4 - 2: 31
Хотите ли Вы заменить вес ребра? Ваш ответ (1 - да, 2 - нет): 1

-1-  -2-  -3-  -4-
-1-   0   1   0   2
-2-   1   0   9  31
-3-   0   9   0   4
-4-   2  31   4   0

Введите номер вершины, вес от которой необходимо заменить: 3
Введите номер ее смежной вершины: 2
Введите неотрицательный вес между вершинами 3 - 2: 100
Хотите ли Вы заменить вес ребра? Ваш ответ (1 - да, 2 - нет): 2

```

Рисунок 3 – Многократное изменение веса ребер

Кратчайшие пути выводятся аналогично матрицами смежности, что показано ниже (рисунок 4).

```

Кратчайшие расстояния от вершины под номером 1:
- 1 - 2 - 3 - 4 -
  0   1   6   2
Кратчайшие расстояния от вершины под номером 2:
- 1 - 2 - 3 - 4 -
  1   0   7   3
Кратчайшие расстояния от вершины под номером 3:
- 1 - 2 - 3 - 4 -
  6   7   0   4
Кратчайшие расстояния от вершины под номером 4:
- 1 - 2 - 3 - 4 -
  2   3   4   0

```

Рисунок 4 – Вывод кратчайших путей

5 Тестирование программы

5.1 Методика тестирования

Среда разработки Microsoft Visual Studio 2017 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. Также были предусмотрены выводы предупреждений об ошибках при недопустимых действиях пользователя.

Ниже продемонстрирован результат тестирования программы при введении пользователем различных значений.

```

Введите количество вершин графа: 5
Дробная часть будет отброшена!

      -1-  -2-  -3-  -4-  -5-
-1-    0    7    3    6    5
-2-    7    0    4    0    4
-3-    3    4    0    5    8
-4-    6    0    5    0    4
-5-    5    4    8    4    0

Хотите ли Вы заменить вес ребра? Ваш ответ (1 - да, 2 - нет): 1
Введите номер вершины, вес от которой необходимо заменить: -3
Такой вершины не существует!
Введите номер ее смежной вершины: 3
Введите номер ее смежной вершины: 9
Такой вершины не существует!
Введите номер ее смежной вершины: 3
Вы уверены, что хотите создать петлю? Ваш ответ (1 - да, 2 - нет): 2

Введите номер вершины, вес ребра от которой необходимо заменить: 4
Введите номер ее смежной вершины: 3
Введите неотрицательный вес между вершинами 4 - 3: 12

      -1-  -2-  -3-  -4-  -5-
-1-    0    7    3    6    5
-2-    7    0    4    0    4
-3-    3    4    0   12    8
-4-    6    0   12    0    4
-5-    5    4    8    4    0

Хотите ли Вы заменить вес ребра? Ваш ответ (1 - да, 2 - нет): 2

Кратчайшие расстояния от вершины под номером 1:
- 1 - 2 - 3 - 4 - 5 -
  0  7  3  6  5
Кратчайшие расстояния от вершины под номером 2:
- 1 - 2 - 3 - 4 - 5 -
  7  0  4  8  4
Кратчайшие расстояния от вершины под номером 3:
- 1 - 2 - 3 - 4 - 5 -
  3  4  0  9  8
Кратчайшие расстояния от вершины под номером 4:
- 1 - 2 - 3 - 4 - 5 -
  6  8  9  0  4
Кратчайшие расстояния от вершины под номером 5:
- 1 - 2 - 3 - 4 - 5 -
  5  4  8  4  0

```

Рисунок 5 – Тестирование при различных вводах

Таблица 1 – Описание поведения программы при различных действиях пользователя

Действие пользователя	Ожидаемый результат
Введение корректного количества вершин графа	Инициализация матрицы и вывод матрицы смежности графа
Введение отрицательного количества вершин графа	Вывод сообщения об ошибке и повторение запроса о вводе корректного количества вершин графа
Введение корректного варианта по замене веса ребра	Замена вес ребра или продолжение программы при исходных значениях
Введение некорректного варианта по замене веса ребра	Продолжение программы при исходных значениях
Введение корректных номеров вершин при замене	Продолжение выполнения программы
Введение отрицательных номеров вершин при замене	Вывод сообщения об ошибке и повторение запроса о вводе корректного значения номеров вершин при замене
Ввод корректного веса при замене	Продолжение выполнения программы (предложение изменения веса ребра, затем вывод кратчайших расстояний)
Ввод отрицательного веса при замене	Продолжение выполнения программы при нулевом весе

5.2 Пример ручного расчета задачи и вычислений

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

Метод не предусматривает работу с петлями, поэтому кратчайшие расстояния от вершины до самой себя будут равняться нулю.

Для первой вершины. Между вершинами 1 и 2 расстояние равно 6. Очевидно, что оставшиеся пути окажутся длиннее. Расстояние между 1 и 3 вершиной равно 9. Если путь будет проложен от 1 к 3, а затем ко 2, то расстояние между 1 и 3 будет равно 11. Минимальные расстояния для вершины под номером 1: 0, 6, 9.

Для второй вершины. Как было написано выше, кратчайшее расстояние между 1 и 2 вершиной равно 6. Между 1 и 3: 5 (2-3) или 15 (2-1-3). Минимальные расстояния для вершины под номером 2: 6, 0, 5.

Вычисления для 3 вершины основаны на расчетах расстояний к предыдущим вершинам. Минимальные расстояния для вершины под номером 3: 9, 5, 0.

Таким образом, можно сделать вывод, что программа работает верно.

```
Введите количество вершин графа: 3
Дробная часть будет отброшена!

    -1-  -2-  -3-
-1-    0   6   9
-2-    6   0   5
-3-    9   5   0

Хотите ли Вы заменить вес ребра? Ваш ответ (1 - да, 2 - нет): 2

Кратчайшие расстояния от вершины под номером 1:
- 1 - 2 - 3 -
  0   6   9
Кратчайшие расстояния от вершины под номером 2:
- 1 - 2 - 3 -
  6   0   5
Кратчайшие расстояния от вершины под номером 3:
- 1 - 2 - 3 -
  9   5   0
```

Рисунок 6 – Тестирование работы программы

Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Дейкстры в Microsoft Visual Studio 2017 C++ 6.0.

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

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

В дальнейшем программу можно улучшить, добавив псевдографический интерфейс, вывод сообщений обо всех ошибках и подсказок при неверном действии пользователя, оптимизировав код.

Список используемых источников

1. Камерон П., ван Линт Дж. Теория графов, теория кодирования и блок-схемы - М.:Наука, 1980, 140 стр.
2. Кристофидес Н. Теория графов. Алгоритмический подход. Пер. с англ. - М.:Мир, 1978, 432 с.
3. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с.
4. Рассел, Джесси Алгоритм Дейкстры / Джесси Рассел. - М.: Книга по Требованию, 2012. - 112 с.
5. Уилсон Р. Введение в теорию графов. Пер. с англ. 1977. 208 с.
6. Харви Дейтел, Пол Дейтел. Как программировать на C/C++. 2009 г.

Приложение А. Листинг программы

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <conio.h>
#include <locale>
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <math.h>

int *G, *distance;
bool *visited;
int a = sizeof(int);
int V, mIndex, min, i, j;
int s=0, size, variant=2, uverp;
void vivodM() {

    printf("\n      ");
    for (int j = 0; j < size; j++) {

        printf(" -%-d-  ", j + 1);

    }
    printf("\n\n");

    for (int i = 0; i < size; i++)
    {
        printf(" -%-d- ", i + 1);
        for (int j = 0; j < size; j++) {

            printf("%5d ", *(G + size * i + j));

        }
        printf("\n\n");
    }
}

void inicializaciaM() {
    //int fl, ce;
    srand(time(NULL));
    printf("Введите количество вершин графа: ");
    scanf("%d", &size);

    while (size < 1) {
        printf("Значения некорректны!\n");
        printf("Введите количество вершин графа: ");
        scanf("%d", &size);
    }

    printf("Дробная часть будет отброшена!");
    scanf("%*[^\\n]");

    printf("\n");
    G = (int*)malloc(size*size * a);
    distance= (int*)malloc(size * a);
    visited= (bool*)malloc(size * a);

    for (i = 0; i < size; i++)        {

        for (j = 0; j < size; j++)
        {
            V = *(G + size * i + j) =*(G + size * j + i) = rand() % 10;
            if (i==j)
                *(G + size * i + j) = 0;

        }
    }
}
```

```

vivodM();

printf("Хотите ли Вы заменить вес ребра? Ваш ответ (1 - да, 2 - нет): ");
scanf("%d", &variant);
if (variant == 1) {
    while (variant == 1)
    {
        printf("Введите номер вершины, вес от которой необходимо заменить: ");
        scanf("%d", &i);
        while ((i < 1) || (i > size)) {
            printf("\nТакой вершины не существует! \n");
            printf("Введите номер ее смежной вершины: ");
            scanf("%d", &i);
        }
        i--;
        printf("Введите номер ее смежной вершины: ");
        scanf("%d", &j);
        while ((j < 1) || (j > size)) {
            printf("\nТакой вершины не существует! \n");
            printf("Введите номер ее смежной вершины: ");
            scanf("%d", &j);
        }
        j--;

        while (i == j) {
            printf("Вы уверены, что хотите создать петлю? Ваш ответ (1 - да, 2 -
нет): ");

            scanf("%d", &uverp);
            printf("\n\n");
            if (uverp == 1) break;
            printf("Введите номер вершины, вес ребра от которой необходимо
заменить: ");

            scanf("%d", &i);
            while ((i < 1) || (i > size)) {
                printf("\nТакой вершины не существует! \n");
                printf("Введите номер ее смежной вершины: ");
                scanf("%d", &i);
            }
            i--;
            printf("Введите номер ее смежной вершины: ");
            scanf("%d", &j);
            while ((j < 1) || (j > size)) {
                printf("\nТакой вершины не существует! \n");
                printf("Введите номер ее смежной вершины: ");
                scanf("%d", &j);
            }
            j--;
        }

        printf("Введите неотрицательный вес между вершинами %d - %d: ", i+1 ,
j+1);

        scanf("%d", &V);

        if (V <= 0) {
            printf("\nЭтот вес будет равен нулю при осуществлении алгоритма!
\n");
        }
        *(G + size * i + j) = V;
        *(G + size * j + i) = V;
    }
}

```

```

        printf("Хотите ли Вы заменить вес ребра? Ваш ответ (1 - да, 2 - нет):
");
        scanf("%d", &variant);
        vivodM();
    }

void algoritm() {
    while ( s != size) {
        for (int i = 0; i < size; i++)
        {
            *(distance + size * i) = INT_MAX;
            *(visited + size * i) = true;
        }
        *(distance + size * s) = 0;

        do {
            mIndex = INT_MAX;
            min = INT_MAX;
            for (int i = 0; i < size; i++)
            {
                if ((*visited + size * i) == true) && (*(distance + size * i) <
min))
                {
                    min = *(distance + size * i);
                    mIndex = i;
                }
            }

            if (mIndex != INT_MAX)
            {
                for (int i = 0; i < size; i++)
                {
                    if (*(G + size * mIndex + i) > 0)
                    {
                        V = min + *(G + size * mIndex+i);
                        if (V < *(distance + size * i))
                        {
                            *(distance + size * i) = V;
                        }
                    }
                }
                *(visited + size * mIndex ) = false;
            } } while (mIndex < INT_MAX);

            s++;
            printf("\nКратчайшие расстояния от вершины под номером %d: \n", s);
            for (int i = 0; i < size; i++)
                printf(" - %3d ", i + 1);
            printf("-\n");
            for (int i = 0; i < size; i++)
                printf(" %5d ", *(distance + size * i));

        }
    }

int main()
{system("chcp 1251");
  system("cls");
  inicializaciaM();
  printf("\n");
  algoritm();
  _getch();
  return 0;
}

```