Частное учреждение образования

«Колледж бизнеса и права»

|  |  |
| --- | --- |
|  | УТВЕРЖДАЮ  Ведущий методист колледжа  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Е.В. Паскал  « \_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 года |
| Специальность 2-40 01 01 «Программное обеспечение информационных технологий» | Учебная дисциплина «Основы алгоритмизации и программирование» |

**Лабораторная работа № 39**

**Инструкционно-технологическая карта**

Тема: Разработка и отладка программ, реализующих алгоритмы нахождения кратчайших путей в графе.

Цель: Научиться разрабатывать программы, реализующие алгоритмы нахождения кратчайших путей в графе; научиться выполнять их отладку.

Время выполнения: 2 часа.

1. **Порядок выполнения работы**
2. Изучить теоретические сведения к лабораторной работе.
3. Разработать на языке С++ программу вывода на экран решения задачи в соответствии с вариантом индивидуального задания, указанным преподавателем.
4. Отлаженную, работающую программу сдать преподавателю. Работу программы показать с помощью самостоятельно разработанных тестов.
5. Ответить на контрольные вопросы.
6. **Теоретические сведения**

**Алгоритм Форда-Беллмана и постановка задачи**

Пусть дан ориентированный взвешенный граф G с n вершинами и m рёбрами, и указана некоторая вершина v. Требуется найти длины кратчайших путей от вершины v до всех остальных вершин. Этот алгоритм применим также и к графам, содержащим рёбра отрицательного веса. Но если граф содержит отрицательный цикл, то кратчайшего пути до некоторых вершин может не существовать (по причине того, что вес кратчайшего пути должен быть равен минус бесконечности); впрочем, алгоритм можно модифицировать, чтобы он сигнализировал о наличии цикла отрицательного веса, или даже выводил сам этот цикл.

Будем считать, что граф не содержит цикла отрицательного веса. Заведём массив расстояний d[0, 1, ..., n-1], который после отработки алгоритма будет содержать ответ на задачу. В начале работы мы заполняем его следующим образом: d[v] = 0, а все остальные элементы d[] равны бесконечности.

Сам алгоритм Форда-Беллмана представляет из себя несколько фаз. На каждой фазе просматриваются все рёбра графа, и алгоритм пытается произвести релаксацию (relax, ослабление) вдоль каждого ребра (a, b) стоимости c. Релаксация вдоль ребра – это попытка улучшить значение d[b] значением d[a] + c. Фактически это значит, что мы пытаемся улучшить ответ для вершины b, пользуясь ребром (a, b) и текущим ответом для вершины a.

Утверждается, что достаточно n-1 фазы алгоритма, чтобы корректно посчитать длины всех кратчайших путей в графе (считаем, что циклы отрицательного веса отсутствуют). Для недостижимых вершин расстояние d[] останется равным бесконечности.

Для алгоритма Форда-Беллмана, в отличие от многих других графовых алгоритмов, более удобно представлять граф в виде одного списка всех рёбер (а не n списков рёбер из каждой вершины). В приведённой реализации заводится структура данных edge для ребра. Входными данными для алгоритма являются числа n, m, список eго рёбер, и номер стартовой вершины v. Все номера вершин нумеруются с 0 по n-1. Константа INF обозначает число "бесконечность" – её надо подобрать таким образом, чтобы она заведомо превосходила все возможные длины путей (например, была максимальным числом данного типа).

struct edge

{

int a, b, cost;

};

int n, m, v;

vector<edge> e;//вектор – контейнерный generic-тип данных, динамический массив

const int INF = 1000000000;

void solve()

{

vector<int> d (n, INF);

d[v] = 0;

for (int i=0; i<n-1; ++i)

{

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

{

if (d[e[j].a] < INF)

{

d[e[j].b] = min (d[e[j].b], d[e[j].a] + e[j].cost);

}

}

}

// вывод d, например, на экран

}

Проверка "if (d[e[j].a] < INF)" нужна, только если граф содержит рёбра отрицательного веса: без такой проверки бы происходили релаксации из вершин, до которых пути ещё не нашли, и появлялись бы некорректные расстояния вида (бесконечность – 1), (бесконечность – 2), … и т.д.

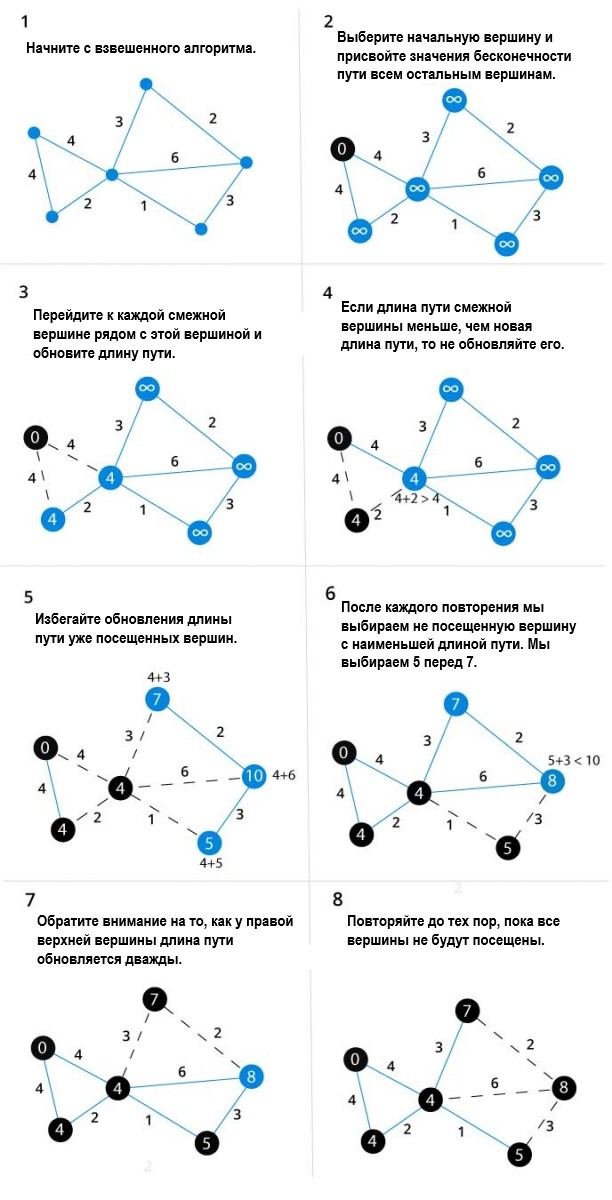
## **Алгоритм Дейкстры**

Рассмотрим пример нахождение кратчайшего пути. Дана сеть автомобильных дорог, соединяющих города области. Некоторые дороги односторонние. Найти кратчайшие пути от центра города до каждого города области. Для решения указанной задачи можно использовать алгоритм Дейкстры – алгоритм на графах, изобретённый нидерландским ученым Э. Дейкстрой в 1959 году. Находит кратчайшее расстояние от одной из вершин графа до всех остальных. Работает только для графов без рёбер отрицательного веса. Пусть требуется найти кратчайшие расстояния от 1-й вершины до всех остальных. Ниже приведены шаги. Входные данные: граф и начальная вершина src. Выходные данные: кратчайшее расстояние до всех вершин от src. Если попадается цикл отрицательного веса, то самые короткие расстояния не вычисляются и выводится сообщение о наличии такого цикла.

1) На этом шаге инициализируются расстояния от исходной вершины до всех остальных вершин как бесконечные, а расстояние до самого src принимается равным 0. Создается массив dist[] размера |V| со всеми значениями, равными бесконечности, за исключением элемента dist[src], где src – исходная вершина.

2) Вторым шагом вычисляются самые короткие расстояния. Следующие шаги нужно выполнять |V|-1 раз, где |V| – число вершин в данном графе. Произведите следующее действие для каждого ребра u-v: если dist[v] > dist[u] + вес ребра uv, то обновите dist[v]: dist [v] = dist [u] + вес ребра uv.

3) На этом шаге сообщается, присутствует ли в графе цикл отрицательного веса. Для каждого ребра u-v необходимо выполнить следующее: если dist[v] > dist[u] + вес ребра uv, то в графе присутствует цикл отрицательного веса. Идея шага 3 заключается в том, что шаг 2 гарантирует кратчайшее расстояние только если граф не содержит цикла отрицательного веса. Если мы снова переберем все ребра и получим более короткий путь для любой из вершин, это будет сигналом присутствия цикла отрицательного веса.

Как это работает? Как и в других задачах динамического программирования, алгоритм вычисляет кратчайшие пути снизу вверх. Сначала он вычисляет самые короткие расстояния, то есть пути длиной не более, чем в одно ребро. Затем он вычисляет кратчайшие пути длиной не более двух ребер и так далее. После *i*-й итерации внешнего цикла вычисляются кратчайшие пути длиной не более*i*ребер. В любом простом пути может быть максимум *|V|-1* ребер, поэтому внешний цикл выполняется именно *|V|-1* раз. Идея заключается в том, что если мы вычислили кратчайший путь с не более чем *i* ребрами, то итерация по всем ребрам гарантирует получение кратчайшего пути с не более чем *i + 1* ребрами.

## **Алгоритм Флойда**

Рассматриваемый алгоритм иногда называют алгоритмом Флойда-Уоршелла. Алгоритм Флойда-Уоршелла является алгоритмом на графах, который разработан в 1962 году Робертом Флойдом и Стивеном Уоршеллом (Варшаллом). Он служит для нахождения кратчайших путей между всеми парами вершин графа.

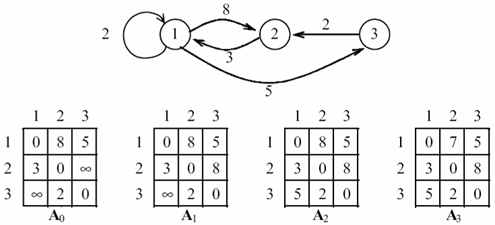
Метод Флойда непосредственно основывается на том факте, что в графе с положительными весами ребер всякий неэлементарный (содержащий более 1 ребра) кратчайший путь состоит из других кратчайших путей.

Этот алгоритм более общий по сравнению с алгоритмом Дейкстры, так как он находит кратчайшие пути между любыми двумя вершинами графа. В алгоритме Флойда используется матрица A размером n на n, в которой вычисляются длины кратчайших путей. Элемент A[i, j] равен расстоянию от вершины i к вершине j, которое имеет конечное значение, если существует ребро (i, j), и равен бесконечности в противном случае.

Основная идея алгоритма Флойда: пусть есть три вершины i, j, k и заданы расстояния между ними. Если выполняется неравенство A[i, k] + A[k, j] < A[i, j], то целесообразно заменить путь i->j путем i->k->j. Такая замена выполняется систематически в процессе выполнения данного алгоритма.

Шаг 0. Определяем начальную матрицу расстояния A0 и матрицу последовательности вершин S0. Каждый диагональный элемент обеих матриц равен 0, таким образом показывая, что эти элементы в вычислениях не участвуют. Полагаем k = 1.

Основной шаг k. Задаем строку k и столбец k как ведущую строку и ведущий столбец. Рассматриваем возможность применения замены описанной выше, ко всем элементам A[i,j] матрицы Ak-1. Если выполняется неравенство A[i,k]+A[k,j]<A[i,j], (i\ne k, j\ne k, i\ne j), тогда выполняем следующие действия: 1) создаем матрицу Ak путем замены в матрице Ak-1 элемента A[i, j] на сумму A[i,k] + A[k,j]; 2) создаем матрицу Sk путем замены в матрице Sk-1 элемента S[i,j] на k. Полагаем k = k + 1 и повторяем шаг k. Таким образом, алгоритм Флойда делает n итераций, после i-й итерации матрица А будет содержать длины кратчайших путей между любыми двумя парами вершин при условии, что эти пути проходят через вершины от первой до i-й. На каждой итерации перебираются все пары вершин и путь между ними сокращается при помощи i-й.



Демонстрация алгоритма Флойда

//Описание функции алгоритма Флойда

void Floyd(int n, int \*\*Graph, int \*\*ShortestPath)

{

int i, j, k;

int Max\_Sum = 0;

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

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

Max\_Sum += ShortestPath[i][j];

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

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

if ( ShortestPath[i][j] == 0 && i != j )

ShortestPath[i][j] = Max\_Sum;

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

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

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

if ((ShortestPath[i][k] + ShortestPath[k][j]) < ShortestPath[i][j])

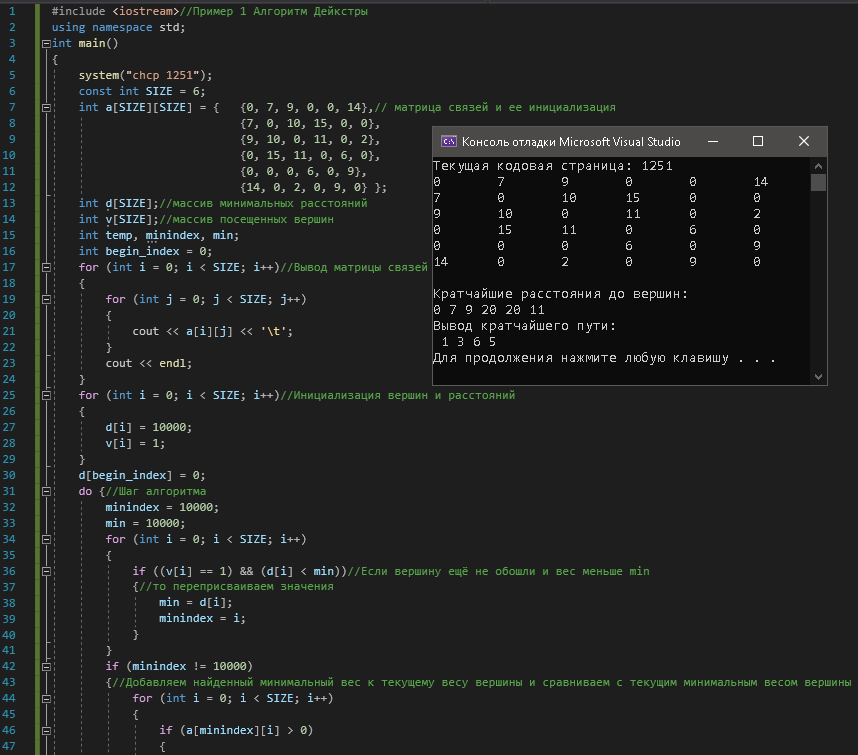
ShortestPath[i][j] = ShortestPath[i][k] + ShortestPath[k][j];

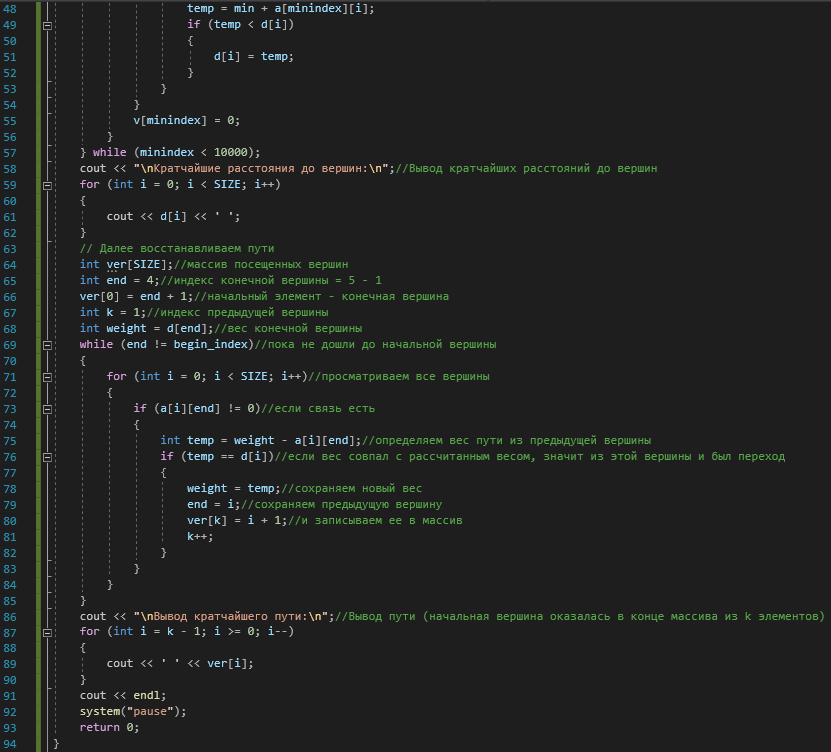
}

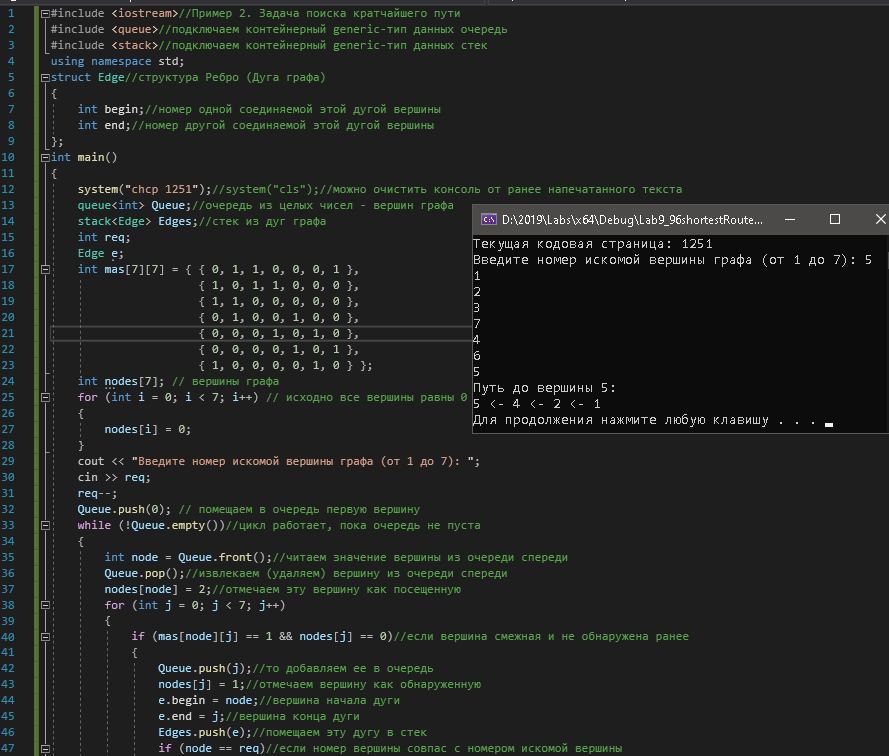
Заметим, что если граф неориентированный, то все матрицы, получаемые в результате преобразований симметричны и, следовательно, достаточно вычислять только элементы, расположенные выше главной диагонали.

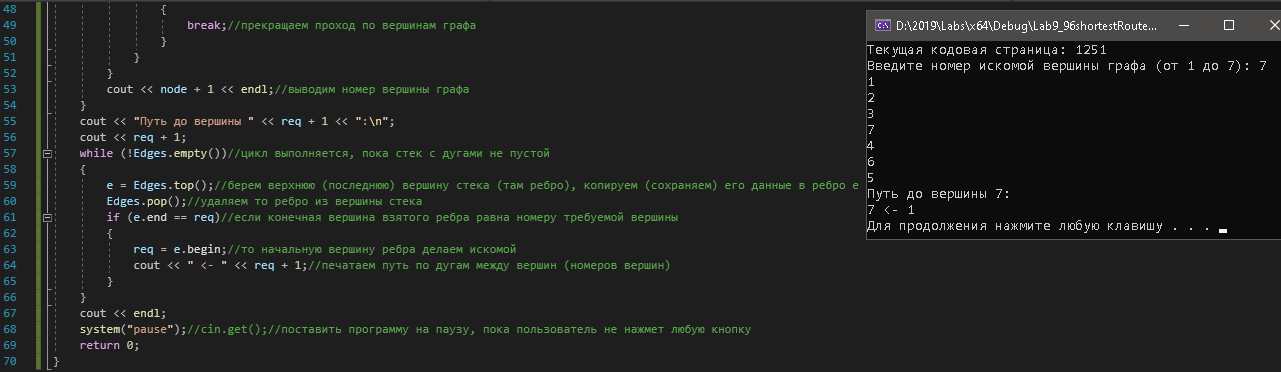
Если граф представлен матрицей смежности, то время выполнения этого алгоритма имеет порядок O(n3), поскольку в нем присутствуют вложенные друг в друга три цикла.

1. **Пример выполнения программы**









1. **Задания по вариантам**

Варианты индивидуальных заданий:

1. Реализовать алгоритм Флойда (отдельной функцией) для графа записанного в виде матрицы:
2. Реализовать алгоритм Форда-Беллмана (отдельной функцией) для графа записанного в виде матрицы:
3. Реализовать алгоритм Дейкстры (отдельной функцией) для графа записанного в виде матрицы:
4. Реализовать алгоритм Флойда (отдельной функцией) для графа записанного в виде матрицы:
5. Реализовать алгоритм Форда-Беллмана (отдельной функцией) для графа записанного в виде матрицы:
6. Реализовать алгоритм Дейкстры (отдельной функцией) для графа записанного в виде матрицы:
7. Реализовать алгоритм Флойда (отдельной функцией) для графа записанного в виде матрицы:
8. Реализовать алгоритм Форда-Беллмана (отдельной функцией) для графа записанного в виде матрицы:
9. Реализовать алгоритм Дейкстры (отдельной функцией) для графа записанного в виде матрицы:
10. Реализовать алгоритм Флойда (отдельной функцией) для графа записанного в виде матрицы:
11. Реализовать алгоритм Форда-Беллмана (отдельной функцией) для графа записанного в виде матрицы:
12. Реализовать алгоритм Дейкстры (отдельной функцией) для графа записанного в виде матрицы:
13. Реализовать алгоритм Флойда (отдельной функцией) для графа записанного в виде матрицы:
14. Реализовать алгоритм Форда-Беллмана (отдельной функцией) для графа записанного в виде матрицы:
15. Реализовать алгоритм Дейкстры (отдельной функцией) для графа записанного в виде матрицы:
16. **Контрольные вопросы**
17. Дайте определение понятию «путь в графе».
18. Дайте определение понятию «взвешенный граф».
19. Зачем может понадобиться находить кратчайший путь между двумя вершинами графа?
20. Опишите алгоритм Флойда. В чем его суть?
21. Опишите алгоритм Форда-Беллмана. В чем его суть?
22. Опишите алгоритм Дейкстры. В чем его суть?

**Литература**

**Дейтел,** Х.М. Как программировать на С++ / Х.М. Дейтел, П.Дж. Дейтел . – М. : Бином-Пресс , 2018 . – 1456 с.

**Павловская**, Т.А. С++. Объектно-ориентированное программирование : практикум / Т.А. Павловская, Ю.А. Щупак . – СПб. : Питер , 2019 . – 265 с.

**Страуструп**, Б. Язык программирования С++ / Б. Страуструп . – СПб. : Бином-Пресс , 2019 . – 1054 с.

Преподаватель Шаляпин Ю.В.

|  |
| --- |
| Рассмотрено на заседании цикловой  комиссии ПОИТ № 10  Протокол №\_\_\_\_от «\_\_\_»\_\_\_\_\_\_\_\_\_\_2021 г.  Председатель ЦК \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |