Оглавление

[**Введение** 3](#_Toc467718754)

[**1 Постановка оптимизационной задачи** 4](#_Toc467718755)

[**2 Описание и анализ алгоритма решения задачи** 7](#_Toc467718756)

[**3 Программная реализация алгоритма** 9](#_Toc467718757)

[**4 Решение контрольного примера** 12](#_Toc467718758)

[**5 Программная документация** 14](#_Toc467718759)

[**Заключение** 16](#_Toc467718760)

[**Библиографический список** 17](#_Toc467718761)

[**А1 Текст программы** 18](#_Toc467718762)

[**А2 Машинное решение** 21](#_Toc467718763)

# **Введение**

Нахождение кратчайшего пути на сегодняшний день является жизненно необходимой задачей и используется практически везде, начиная от нахождения оптимального маршрута между двумя объектами на местности (например, кратчайший путь от дома до университета), в системах автопилота, для нахождения оптимального маршрута при перевозках, коммутации информационного пакета в сетях и т.п.

Кратчайший путь рассматривается при помощи некоторого математического объекта, называемого графом. Поиск кратчайшего пути ведется между двумя заданными вершинами в графе. Результатом является путь, то есть последовательность вершин и ребер, инцидентных двум соседним вершинам, и его длина.

В данной курсовой работе рассматривается решения задачи нахождения кратчайшего пути в графе при помощи алгоритма 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. 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.

# **1 Постановка оптимизационной задачи**

Рассмотрим связный ориентированный граф , где – множество вершин, – множество дуг вида .

Путем в ориентированном графе G называют последовательность дуг , в которой конечная вершина каждой предыдущей дуги совпадает с начальной вершиной следующей дуги. Путь также может определяться как последовательность вершин , соединенных дугами графа. Конечный путь , для которого начальная вершина совпадает с конечной , называется контуром. Путь, в котором никакая дуга не встречается дважды, называется простым.

Длиной пути называют число , равное количеству дуг, составляющих этот путь. Если каждой дуге соответствует некоторое число , называемое длиной или весом дуги (графы со взвешенными душами), то длина пути в таком графе определяется как

Наиболее часто для заданного графа требуется определить кратчайший путь из некоторой выделенной вершины s (источника) в другую заранее заданную вершину t (сток), где . Если значения , рассматривать как стоимости прохождения единицы потока по дугам , то задачу можно сформулировать следующим образом [2, 3]: определить поток единичной мощности из источника s в сток t, который имеет минимальную стоимость.

Соответствующая задача целочисленного линейного программирования (ЦЛП) заключается в определении значений переменных , сопоставленных с дугами и обеспечивающих

при ограничениях

.

Отмечу, что приведенная формулировка задачи поиска кратчайшего пути определяет частный случай, для которого длины дуг всегда неотрицательны, и может быть решена как задача ЦЛП.

В общем случае при построении кратчайших путей выделяют графовые модели, особенности которых отражены на рисунке 1.

C:\Users\Alexey_Perov\Downloads\Scheme.png

Рисунок 1 – Виды графовых моделей для задачи поиска кратчайшего пути

Практически для всех указанных на рисунке 1 разновидностей графов существуют эффективные алгоритмы построения кратчайших путей, учитывающие специфические особенности конкретных моделей. Исключение составляют ориентированные графы, содержащие контуры с суммарной отрицательной длиной (весом). Если такой контур φ существует, то при построении любого пути на графе, проходящего через вершину , можно получить сколь угодно малую длину за счет многократного обхода контура φ. Таким образом, в этом случае кратчайшего пути не существует и можно решать задачу построения кратчайшего простого пути от s к t, которая является значительно более сложной и сводится к нахождению кратчайшего гамильтонова пути [4]. Тем не менее, в работе [2] описан алгоритм поиска простого пути наименьшей длины между вершинами s и t, а в [4] – метод обнаружения контуров суммарного отрицательного веса.

# **2 Описание и анализ алгоритма решения задачи**

Пусть требуется определить кратчайшие пути между всеми парами вершин и графа . Решение задачи может быть получено n-кратным применением метода Форда – Белмана (для произвольных весов дуг) или метода Дейкстры (для неотрицательных весов). Предельные оценки вычислительной сложности подобных процедур определяются как и операций соответственно. Поэтому в пером случае при большой размерности задачи ее решение крайне затруднительно или даже невозможно.

Более эффективной процедурой поиска кратчайших путей для является алгоритм Флойда, который применяется к графовым моделям как с неотрицательными, так и с произвольными весами дуг. Если граф содержит контур суммарной отрицательной длины, то алгоритм Флойда фиксирует его существование. При этом предельная трудоемкости для случая произвольных весов дуг составляет операций, а для неотрицательных весов время решения примерно на 50% меньше по сравнению с n-кратным применением алгоритма Дейкстры [4].

Алгоритм Флойда базируется на использовании последовательности из n преобразований итераций начальной матрицы весов дуг, элементы которой определяются следующим образом:

На каждой k-й итерации ( элементы матрицы C определяют длины кратчайших путей между каждой парой вершин и при условии, что путь может проходить только через вершины из множества . Таким образом, итоговое значение матрицы C после выполнения n итераций показывает искомые значения длин кратчайших путей .

В основе алгоритма Флойда лежат следующие очевидные уравнения:

Пусть кратчайший путь может содержать промежуточные вершины из множества Тогда второе уравнение показывает, что если путь не содержит вершину , то его длина определяется как В противном случае путь можно разбить на две части: и . Следовательно, его длина составляет . В целом уравнения (1) и (2) позволяют легко вычислить длины кратчайших путей между всеми парами вершин .

# **3 Программная реализация алгоритма**

Таким образом, алгоритм представляет собой последовательность действий:

Алгоритм

Данные: матрица весов дуг ориентированного графа ; произвольные значения для .

Результаты: кратчайшие расстояния между всеми парами вершин .

Шаг 1 (присвоение начальных значений)

Установить номер итерации k=0.

Шаг 2 (итерационный процесс)

Присвоить k = k + 1. Для всех , удовлетворяющих условию , и для всех , таких, что , вычислить новые значения

Все другие элементы матрицы C сохранить без изменений, т.е.

Шаг 3 (проверка условий окончания)

а) Если , то граф G содержит контур отрицательной длины, проходящий через вершину , и задача не имеет решения. Конец алгоритма.

б) Если и k=n, то решение получено. Конец алгоритма.

в) Если и k < n, то переход к шагу 2.

Кратчайший путь можно построить исходя из его длины следующим образом. Сначала определяется предпоследняя вершина пути , предшествующая , по условию . Затем находится вершина , для которой и т.д.

В работах [3, 4, 5] представлен метод построения путей основанных на хранении дополнительной информации об индексах вершин, составляющих тот или иной путь. Для этого требуется ещё одна матрица , элемент которой указывает индекс вершины, непосредственно предшествующей в пути .

Начальные значения для всех присваиваются элементам матрицы V на шаге 1. Обновление матрицы производится на шаге 2 следующим образом:

Полученные значения позволяют построить кратчайший путь

между любой парой вершин, где

Блок схема алгоритма приведена на рисунке 2:

C:\Users\Alexey_Perov\Downloads\Untitled Diagram.png

Рисунок 2 – Блок-схема алгоритма Флойда

# **4 Решение контрольного примера**

В качестве контрольного примера решим задачу, описываемую графом, изображенным на рисунке 3. Необходимо определить кратчайший путь между вершинами 1 и 4.

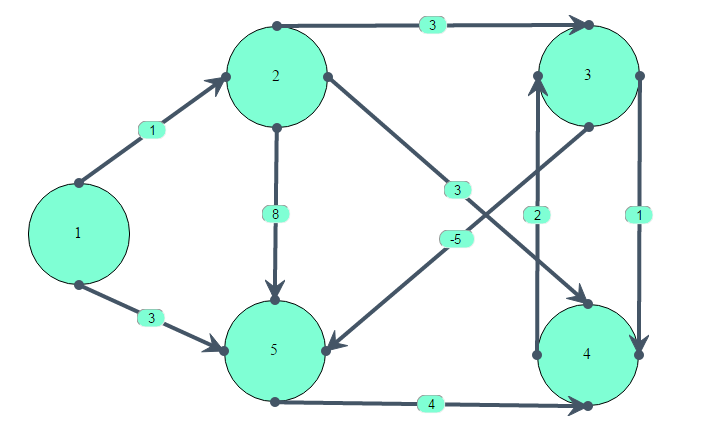


Рисунок 3 – Граф, описывающий контрольный пример.

В данном случае матрица расстояний примет вид:



Далее приведены состояния матрицы C и V на каждом шаге алгоритма:

1.  
2.  
3.  
4.  
5.  

По матрице C определим длину пути:

Кратчайший путь определим по матрице V:

# **5 Программная документация**

Интерфейс программы состоит из 2 главных частей: панели инструментов и рабочего пространства.

Панель инструментов изображена на рисунке 4.



Рисунок 4 – Панель инструментов

Для работы программы необходимо построить ориентированный граф.

Для добавления вершины графа необходимо нажать кнопку «Добавить вершину», при этом в области рабочего пространства появится вершина. Появившуюся вершину можно удобно расположить в области рабочего пространства при помощи манипулятора «мышь» схватившись за вершину и перенеся в любое удобное место в пределах области рабочего пространства.

Для добавления дуги необходимо добавить несколько вершин. При наведении манипулятора «мышь» на вершину появятся несколько якорей. Чтобы добавить дугу, необходимо нажать на якорь и «вытянуть» дугу до якоря другой вершины.

Вес дуги вводится в поле, расположенное в центе дуги.

Для удаления дуги необходимо потянуть дугу со стороны любого якоря, проассоциированных с удаляемой дугой.

Для запуска алгоритма необходимо указать начальную и конечную вершину в полях «Начальная вершина» и «Конечная вершина» соответственно. Алгоритм начинает свою работу по нажатии кнопки «Найти кратчайший путь».

После выполнения работы алгоритма на экран будет выведено оповещение, в котором будут указаны результаты работы алгоритма, а именно: длина кратчайшего пути, вершины, через которые проходит кратчайший путь, матрица расстояний C, а также дополнительная матрица V. Пример вывода результата работы алгоритма представлен на рисунке 5.

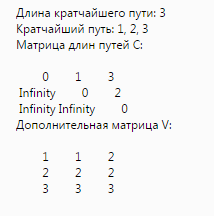


Рисунок 5 – Пример вывода результатов работы алгоритма

После закрытия диалога с результатами работы алгоритма, на схеме будет изменен цвет вершин и дуг которые составляют кратчайший путь от начальной до конечной вершины.

В случае обнаружения контура с отрицательной длиной также будет показано сообщение о невозможности нахождения решения, а также указана вершина, через которую проходит контур с отрицательной длиной. Пример вывода сообщения о нахождении контура отрицательной длины представлен на рисунке 6.



Рисунок 6 – Пример сообщения о нахождении контура отрицательной длины

# **Заключение**

В ходе выполнения курсовой работы был изучен и реализован алгоритм нахождения кратчайшего пути во взвешенном ориентированном графе методом Флойда.

Данный алгоритм подходит для нахождения кратчайшего пути в графах как с неотрицательными, так и с произвольными весами дуг. При этом предельная трудоемкости для случая произвольных весов дуг составляет операций, а для неотрицательных весов время решения примерно на 50% меньше по сравнению с n-кратным применением алгоритма Дейкстры .

Кроме прочего, алгоритм позволяет выявить контуры имеющие отрицательную длину.

В качестве реализации алгоритма было создано браузерное графическое приложение на языке JavaScript, имеющее удобный и понятный интерфейс.

# **Библиографический список**

1. Алгоритмы построения путей на графах: Методические указания к практическим занятиям / Сост: С.В. Скворцов, В.И. Хрюкин, Л.Б. Михеева. Рязань: РГРТА, 2004.
2. Йенсен П., Барнес Д. Потоковое программирование. М.: Радио и связь Наук 1984. 392 с.
3. Пападимитриу Х., Стайглиц К. Комбинаторная оптимизация. Алгоритмы и сложность. М.: Мир, 1980. 476 с.
4. Кристофидес Н. Теория графов. Алгоритмический подход М.: Мир, 1978. 432 с.
5. Ху Т. Целочисленное программирование и потоки в сетях. М.: Мир, 1974. 520 с.

# **А1 Текст программы**

**controller.js**

**var *dotAmount*** = 0;  
  
**$**(**document**).ready(**function**() {  
 **var** *preventNonDigits* = **function**(event) {  
 **return** event.**charCode** >= 48 && event.**charCode** <= 57;  
 };  
 **$**(**"#firstTop"**).on(**"keypress"**, *preventNonDigits*);  
 **$**(**"#secondTop"**).on(**"keypress"**, *preventNonDigits*);  
});  
  
**function** *addDot*() {  
 **var** newDot = **$**(**"<div class='connectorHolder' data-number='"** + ++***dotAmount*** + **"'><div class='dot-number'>"** + ***dotAmount*** + **"</div></div>"**);  
 **$**(**".container"**).append(newDot);  
  
 *addNewEndpoints*(newDot);  
  
 newDot.on(**"mouseover"**, **function**() {  
 **$**(**".jsplumb-endpoint"**).addClass(**"jsplumb-hover"**);  
 });  
  
 newDot.on(**"mouseout"**, **function**() {  
 **$**(**".jsplumb-endpoint"**).removeClass(**"jsplumb-hover"**);  
 });  
  
 **jsPlumbInstance**.**draggable**(newDot, {  
 **containment**:**true** });  
  
 **window**.**model**.addDot();  
}  
  
**function** *addNewEndpoints*(item) {  
 **var** endpointCommonOptions = {  
 **isSource**: **true**,  
 **isTarget**: **true** };  
 **jsPlumbInstance**.addEndpoint(item, {**anchor**: **"Top"**}, endpointCommonOptions);  
 **jsPlumbInstance**.addEndpoint(item, {**anchor**: **"Bottom"**}, endpointCommonOptions);  
 **jsPlumbInstance**.addEndpoint(item, {**anchor**: **"Left"**}, endpointCommonOptions);  
 **jsPlumbInstance**.addEndpoint(item, {**anchor**: **"Right"**}, endpointCommonOptions);  
}  
  
**function** *getDotByNumber*(number) {  
 **return $**(**"[data-number="** + number + **"]"**)[0];  
}  
  
**function** *getWeightOverlay*(source, target) {  
 **return jsPlumbInstance**.getConnections({**source**: source, **target**: target})[0].getOverlay(**"weightInput"**);  
}  
  
**function** *showShortestPath*(pathes, begin, end) {  
 **var** shortestPath = [*getDotByNumber*(end)];  
  
 (**function** *getNextDot*(begin, end) {  
 **if** (begin != end) {  
 **var** number = pathes[begin - 1][end - 1];  
 **var** nextDot = *getDotByNumber*(number);  
 shortestPath.push(*getWeightOverlay*(nextDot, shortestPath[shortestPath.**length** - 1]).getElement());  
 shortestPath.push(nextDot);  
 *getNextDot*(begin, number);  
 }  
 })(begin, end);  
  
 (**function** *highlightPath*(shortestPath) {  
 **if** (shortestPath.**length** > 0) {  
 setTimeout(**function**() {  
 **$**(shortestPath.pop()).addClass(**"highlighted"**);  
 *highlightPath*(shortestPath);  
 }, 700);  
 }  
 })(shortestPath);  
  
}  
  
**function** *getShortestPath*(pathes, begin, end) {  
 **var** shortestPath = [end];  
  
 (**function** *getNextDot*(begin, end) {  
 **if** (begin != end) {  
 **var** number = pathes[begin - 1][end - 1];  
 shortestPath.push(number);  
 *getNextDot*(begin, number);  
 }  
 })(begin, end);  
  
 **return** shortestPath.reverse();  
}  
  
**function** *arrayToString*(array) {  
 **var** result = **""**;  
 **$**.each(array, **function**(i, row) {  
 **$**.each(row, **function**(j, element) {  
 result += **new** Array(9 - element.toString().**length**).fill(**" "**).join(**""**) + element;  
 });  
 result += **"\n"**;  
 });  
 **return** result;  
}

**model.js**

**function** *Model*() {  
  
 **var** weights = [];  
 **var** pathes = [];  
  
 **this**.addDot = **function**() {  
 **for** (**var** i = 0; i < weights.**length**; i++) {  
 weights[i].push(**Infinity**);  
 pathes[i].push(i + 1);  
 }  
 **var** newRow = (**new** Array(weights.**length**)).fill(**Infinity**);  
 newRow.push(0);  
 weights.push(newRow);  
 pathes.push(**new** Array(pathes.**length** + 1).fill(pathes.**length** + 1));  
 };  
  
 **this**.setConnectionWeight = **function** (source, target, weight) {  
 weights[source][target] = weight;  
 };  
  
 **this**.getWeights = **function**() {  
 **return** weights;  
 };  
  
 **this**.getPathes = **function**() {  
 **return** pathes;  
 };  
  
 **this**.calculate = **function**() {  
 **$**(**".highlighted"**).removeClass(**"highlighted"**);  
 **if** (*validateInitialConditions*()) {  
 **var** firstTop = +**$**(**"#firstTop"**).val();  
 **var** secondTop = +**$**(**"#secondTop"**).val();  
 **var** result = **$**.extend(**true**, [], weights);  
 **var** solutionPathes = **$**.extend(**true**, [], pathes);  
 **for** (**var** k = 0; k < result.**length**; k++) {  
 **for** (**var** i = 0; i < result.**length**; i++) {  
 **for** (**var** j = 0; j < result[i].**length**; j++) {  
 **if** (result[i][j] > result[i][k] + result[k][j]) {  
 result[i][j] = result[i][k] + result[k][j];  
 solutionPathes[i][j] = k + 1;  
 }  
 **if** (i == j && result[i][j] < 0) {  
 alert(**"Обнаружен контур отрицательной длины! Дальнейшие вычисления невозможны.\n"** +  
 **"Вершина, через которую проходит контур отрицательной длины: "** + (k + 1));  
 **return false**;  
 }  
 }  
 }  
 }  
 *showShortestPath*(solutionPathes, firstTop, secondTop);  
 alert(**"Длина кратчайшего пути: "** + result[firstTop - 1][secondTop - 1] + **"\n"** +  
 **"Кратчайший путь: "** + *getShortestPath*(solutionPathes, firstTop, secondTop).join(**", "**) + **"\n"** +  
 **"Матрица длин путей С:"** + **"\n"** + arrayToString(result) +  
 **"Дополнительная матрица V:"** + **"\n"** + arrayToString(solutionPathes));  
 }  
 };  
  
 **function** *validateInitialConditions*() {  
 **var** isValid = **true**;  
 **if** (weights.**length** > 1) {  
 **var** firstTop = **$**(**"#firstTop"**).val();  
 **var** secondTop = **$**(**"#secondTop"**).val();  
  
 **if** (firstTop === **""** || secondTop === **""**) {  
 alert(**"Укажите начальную и конечную вершины"**);  
 firstTop === **""** ? **$**(**"#firstTop"**).**focus**() : **$**(**"#secondTop"**).**focus**();  
 isValid = **false**;  
 }  
  
 **if** (+firstTop > weights.**length** + 1 || +secondTop > weights.**length** + 1) {  
 alert(**"Введите номер вершины в диапазоне от "** + 1 + **" до "** + weights.**length**);  
 isValid = **false**;  
 }  
 } **else** {  
 alert(**"Сначала добавьте несколько вершин"**);  
 }  
 **return** isValid;  
 }  
}  
  
**window**.**model** = **new** *Model*();

# **А2 Машинное решение**

