**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по учебной практике**

Тема: Визуализация алгоритма Дейкстры поиска

Кратчайших путей в графе

|  |  |  |
| --- | --- | --- |
| Студент гр. 7381 |  | Дорох С.В. |
| Студент гр. 7381 |  | Ильясов А.В. |
| Студент гр. 7381 |  | Павлов А.П. |
| Руководитель |  | Жангиров Т.М. |

Санкт-Петербург

2019

**ЗАДАНИЕ**

**на учебную практику**

|  |  |  |
| --- | --- | --- |
| Студент Дорох С.В. группы 7381 | | |
| Студент Ильясов А.В. группы 7381 | | |
| Студент Павлов А.П. группы 7381  Тема практики: разработка визуализатора алгоритма на Java. | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма(ов) на Java с графическим интерфейсом.  Алгоритм: алгоритм Дейкстры. | | |
| Сроки прохождения практики: 01.07.2019 – 14.07.2019 | | |
| Дата сдачи отчета: 08.07.2019 | | |
| Дата защиты отчета: 10.07.2019 | | |
|  | | |
| Студент |  | Дорох С.В. |
| Студент |  | Ильясов А.В. |
| Студент |  | Павлов А.П. |
| Руководитель |  | Жангиров Т.М. |

**Аннотация**

Темой данной учебной практики является командная итеративная разработка визуализатора алгоритма на языке программирования Java. Цель учебной практики – получить практические навыки в визуализации алгоритмов, изучить и получить навыки использования языка программирования Java, получить навыки работы в команде. В работе представлена визуализация алгоритма Дейкстры поиска кратчайших путей в графе.

**Summary**

The topic of this training practice is the team iterative development of the algorithm visualizer in Java programming language. The purpose of the training practice is to get practical skills in visualization of algorithms, to learn and get skills of using Java programming language, to get skills of working in a team. The work presents the visualization of the algorithm of Dijkstra search for the shortest paths in the graph.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 5 |
| 1.1 | Требования к программе | 6 |
| 1.1. | Исходные требования к программе | 7 |
| 1.1.1. | Требования к вводу исходных данных | 7 |
| 1.1.2. | Требования к визуализации | 8 |
| 1.1.3. | Требования к выводу результата | 9 |
| 1.2. | Уточнение требований после сдачи прототипа | 10 |
| 2. | План разработки и распределение ролей в бригаде | 11 |
| 2.1. | План разработки | 11 |
| 2.2. | Распределение ролей в бригаде | 12 |
| 3. | Особенности реализации | 13 |
| 3.1. | Использованные сторонние библиотеки | 13 |
| 3.2. | Использованные структуры данных | 14 |
| 3.3 | uml-диаграммы | 16 |
|  | Заключение | 20 |
|  | Список использованных источников | 21 |

**введение**

Дан ориентированный или неориентированный взвешенный граф с n вершинами и m рёбрами. Веса всех рёбер неотрицательны. Указана некоторая стартовая вершина s. Требуется найти длины кратчайших путей из вершины s во все остальные вершины, а также предоставить способ вывода самих кратчайших путей.

Эта задача называется "задачей о кратчайших путях с единственным источником" (single-source shortest paths problem).

Алгоритм Дейкстры - 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.

Алгоритм Дейкстры относится к так называемым «жадным» алгоритмам. Пусть расстояние от начальной вершины start до вершины i хранится в массиве dist[i]. Начальные значения dist[start]=0, dist[i]=INF для всех остальных вершин i. То есть в самом начале алгоритму известен путь из вершины start до вершины start длины 0, а до остальных вершин кратчайшие пути неизвестны. Между тем алгоритм будет постепенно улучшать значения в массиве dist, в результате получит кратчайшие расстояния до всех вершин.

Основная идея для улучшения называется «релаксацией ребра». Пусть из вершины i в вершину j есть ребро веса w[i][j], при этом выполнено неравенство dist[i] + w[i][j] < dist[j]. То есть можно построить маршрут из начальной вершины до вершины i и добавить к нему ребро из i в j, и суммарная стоимость такого маршрута будет меньше, чем известная ранее стоимость маршрута из начальной вершины в вершину j. Тогда можно улучшить значение dist[j], присвоив dist[j] = dist[i] + w[i][j].

В алгоритме Дейкстры вершины красятся в два цвета, будем говорить, что вершина «неокрашенная» или «окрашенная». Изначально все вершины неокрашенные. Если алгоритм Дейкстры покрасил вершину i, то это означает, что найденное значение dist[i] является наилучшим возможным и в последствии не будет улучшаться, то есть значение dist[i] является кратчайшим расстоянием от начальной вершины до вершины i. Если же вершина не покрашена, то величина dist[i] для такой вершины i равна кратчайшему пути из вершины start до вершины i, который проходит только по покрашенным вершинам (за исключением самой вершины i).

На каждом шаге алгоритма Дейкстры красится одна новая вершина. В качестве такой вершины выбирается неокрашенная вершина i с наименьшим значением D[i]. Затем рассматриваются все ребра, исходящие из вершины i, и производится релаксация этих ребер, то есть улучшаются расстояния до вершин, смежных с i.

Алгоритм заканчивается, когда на очередном шаге не останется неокрашенных вершин или если расстояние до всех неокрашенных вершин будет равно INF (то есть эти вершины являются недостижимыми).

**1. требования к программе**

**1.1. Исходные Требования к программе**

**1.1.1 Требования к вводу исходных данных**

На вход алгоритму Дейкстре подается ориентированный граф с неотрицательными весами ребер.

Ввод компонент графа должен осуществляться с файла, в котором записаны ребра в формате: (from, to, weight), либо же прямо в приложении с возможностью отдельного добавления вершин и ребер.

При попытке добавления вершины, должно выводится окно, сообщающее об этом. Также при попытке ввода ребра должны выводится окна со следующей информацией:

* Отсутствует начальная вершина – при вводе начальной вершины ребра, которая отсутствует в графе;
* Отсутствует конечная вершина – при вводе начальной вершины ребра, которая отсутствует в графе;
* Некорректное значение веса ребра – при вводе отрицательного веса ребра.

**1.1.2 Требования к визуализации**

Алгоритм Дейкстры можно разделить на 3 основных итерации: выбор не просмотренной вершины с наименьшим текущим расстоянии до начальной вершины, выбор ребра, исходящего из текущей просматриваемой вершины и имеющего наименьший вес, релаксация.

Следовательно, необходимо, чтобы программа визуализировала все 3 итерации алгоритма: при выборе следующей текущей вершины, она перекрашивается в другой цвет, при выборе ребра для перехода, оно также перекрашивается в другой цвет, при успешной релаксации, значение текущего расстояния должно обновляться около соответствующей вершины, ну и при завершении просмотра вершины, когда все исходящие из нее ребра просмотрены, она перекрашивается в новый цвет.

**1.1.3 Требования к выводу результата**

В качестве вывода, программа должна выводить кратчайшие расстояния до каждой вершины в графе, кроме начальной, а также сами пути.

**1.2. Уточнение требований после прототипа**

**2. План разработки и распределение ролей в бригаде**

**2.1. План разработки**

**Прототип:**

Будут реализованы все кнопки интерфейса, всплывающие окна с дополнительной информацией.

**Итерация 1:**

Будет реализован ввод произвольного графа графическим способом. Планируется реализовать алгоритм Дейкстры без связи с графический интерфейсом.

**Итерация 2:**

Будет реализована связь графического интерфейса с основным алгоритмом. Добавится подробный вывод информации о работе алгоритма на каждой итерации, а также ввод структуры графа из файла.

**Итерация 3:**

Будет проведена отладка всего проекта.

**2.2. Распределение ролей в бригаде**

В табл. 1 представлены основные обязанности каждого члена бригады.

Таблица 1 – обязанности членов бригады

|  |  |
| --- | --- |
| Имя студента | Обязанность |
| Дорох С.В. | Разработка графического интерфейса.  Ведение Git-репозитория. |
| Ильясов А.В. | Разработка пошаговой работы алгоритма.  Выведение результат программы. |
| Павлов А.П. | Разработка архитектуры проекта(uml-диаграммы).  Разработка связи графического интерфейса и алгоритма. |

**3. Особенности реализации**

**3.1. Использованные сторонние библиотеки**

Использована библиотека JGraphX, из которой взяты классы mxGrpah и mxCell.

mxGrpah – класс, представляющий собой граф, содержащий методы, для удобной реализации алгоритмов на графах.

mxCell – класс с помощью которого можно хранить вершины и ребра.

Эта библиотека была выбрана по 2 основным причинам:

1) удобный и понятный интерфейс;

2) совместимость с java.Swing.

**3.2. Использованные структуры данных**

В проекте реализованы 3 основных класса: Main, GUI, Dijkstra.

**Class Main:**

Главный класс, содержащий метод main, в котором происходит создание объекта графического интерфейса.

**Class GUI:**

Класс, наследующий класс JApplet. Реализует графический интерфейс.

Компоненты java.swing:

* JButton addVertexButton - кнопка для добавления вершины в граф
* JButton addEdgeButton - кнопка для добавления ребра в граф
* JButton nextButton - кнопка перехода к следующей итерации
* JButton executeButton - кнопка перехода к результату алгоритма
* JButton helpButton - справочная кнопка

Вложенные классы:

Классы, реализующие интерфейс ActionListener и перегружающие метод, вызывающийся по нажатию соответствующей кнопки:

addVertexButtonEventListener

addEdgeButtonEventListener

addEdgeButtonEventListener

nextIterationButton

Execute

RoundedBorder – класс, реализующий интерфейс Border и задающий форму кнопок меню.

Методы класса:

initButtons() - задание формы кнопок и их инициализация.

init() - метод для вызова инициализации графа, кнопок и кругового макета.

initCircleLayout() - метод инициализации кругового макета.

initGraph() - метод инициализации графа.

checkStartVertex() - метод для проверки на наличие в графе начальной вершины.

exceptionChecker(int flag) - метод для вывода сообщения в случае наудачного добавления ребра.

cellFounder(String result) - метод для поиска вершины в графе по её названию.

nextStep(Object cell) – метод, выполняющий 1 шаг алгоритма.

**Class Dijkstra:**

Класс, реализующий алгоритм Дейкстры.

Атрибуты:

enum Steps{UNVISITED\_VERTEX\_SELECTION, NEAREST\_NEIGHBOR\_SELECTION, RELAXATION} – перечисление, хранящее варианты итераций алгоритма.

step – переменная, хранящая элемент Steps, показывающая, какая итерация алгоритма должна выполняться следующей.

HashMap<Object, Double> distance - карта вершин и расстояний до них.

ArrayList<Object> unvisitedVertices - карта вершин и расстояний до них.

HashMap<Object, TreeSet<Object>> outgoingEdges - карта вершин и выходящих из них ребер.

HashMap<Object, Object> parents - список пар вершин, по которым восстанавливаются кратчайшие пути.

mxGraph graph - граф, в котором ищутся кратчайшие пути.

Object source - начальная вершина в алгоритме Дейкстры.

Методы класса:

Dijkstra(mxGraph graph, Object source) – конструктор класса.

getStep() – метод, возвращающий текущее состояние атрибута step.

isNextStep() – метод, проверяющий, не завершился ли алгоритм.

removeVertex(Object vertex, Object edge) – метод удаляющий вершину из списка непросмотренных.

getPaths() – метод, считающий расстояния до всех вершин. Может вызываться, если не требуется визуализация.

selectUnvisitedVertex() – метод выбора следующей просматриваемой вершины из еще непросмотренных.

selectNearestNeighbor(Object vertex) – метод выбора непросмотренной вершины, ближайшей к текущей просматриваемой.

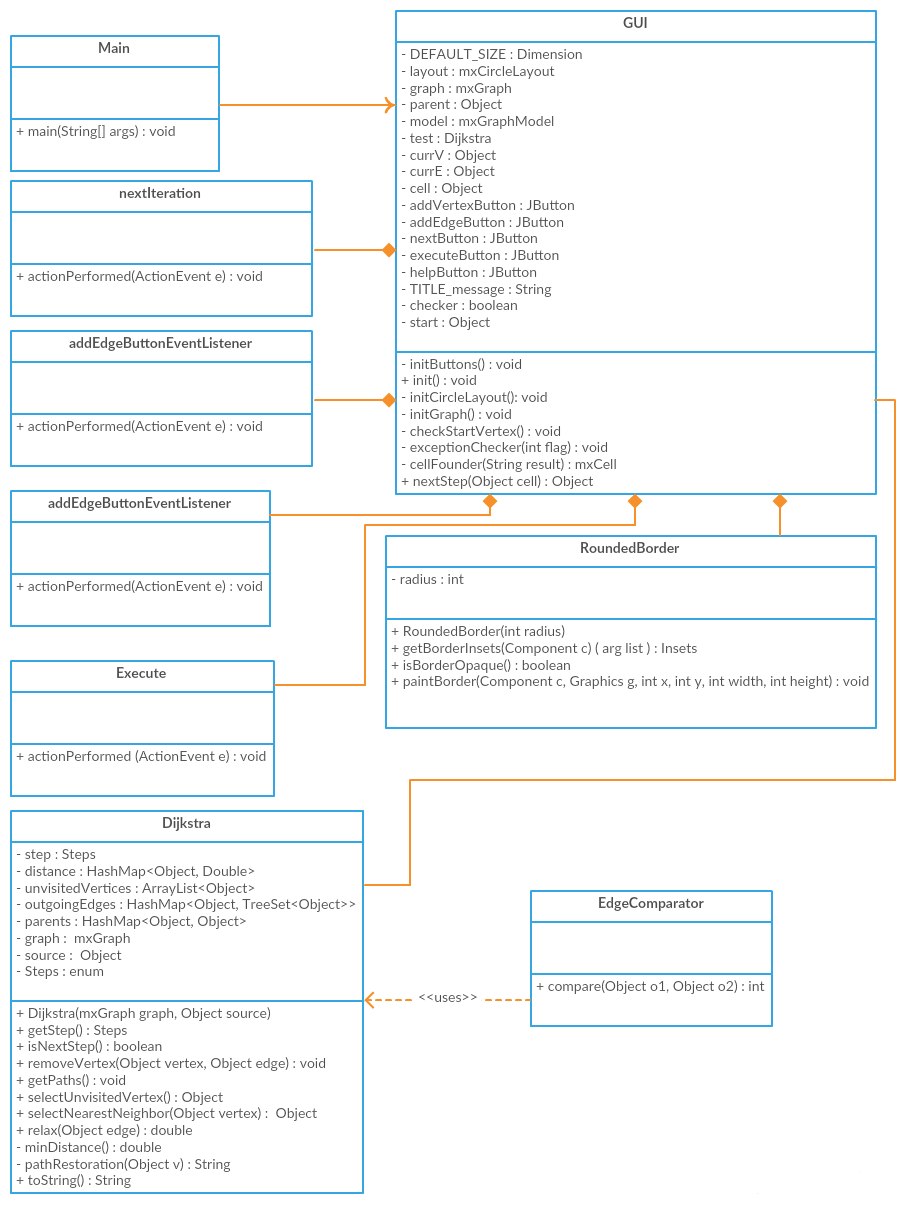
relax(Object edge) – метод обновления расстояния до вершины.

minDistance() – метод поиска кратчайшего расстояния до непросмотренных вершин.

pathRestoration(Object v) – метод восстановления пути до вершины.

toString() – метод, возвращающий строковое состояние объекта(расстояния до вершин и сами пути).

**3.3. uml-диаграммы**

рисунок 1 – uml-диаграмма классов

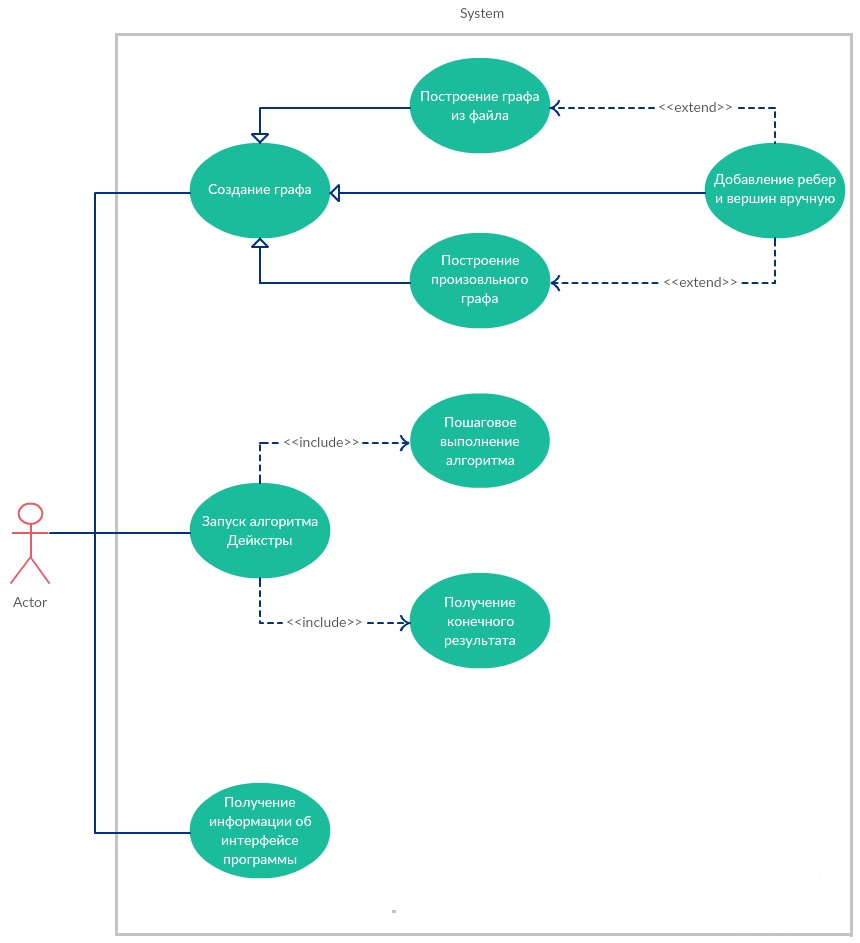


рисунок 2 – use case диаграмма

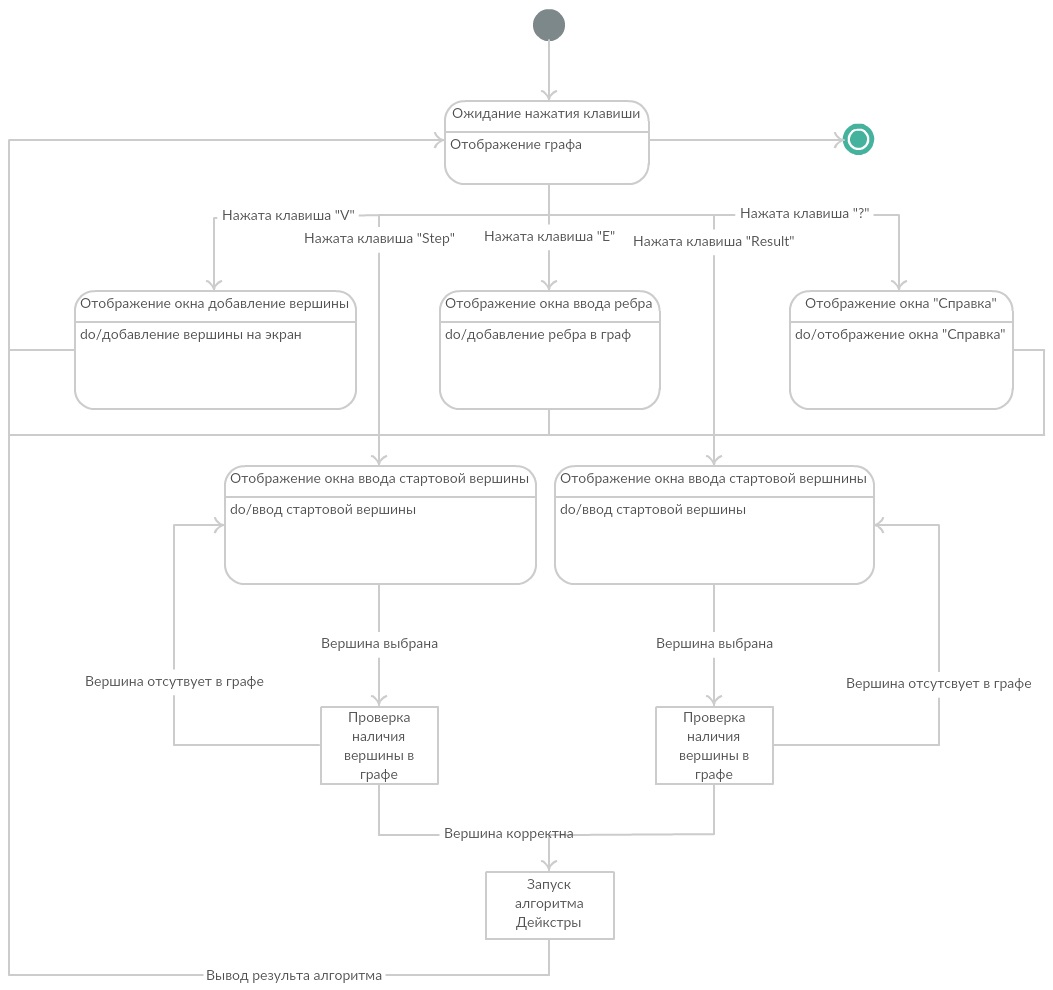


Рисунок 3 – statechart диаграмма

**заключение**

В ходе данной учебной практики были изучены основы программирования на языке Java, пройден интерактивный курс «Java.Базовый курс». После чего была разработана программа, которая с помощью алгоритма Дейкстры находит кратчайшие пути в графе из начальной вершины с визуализацией шагов алгоритма.

**список использованных источников**

1)https://docs.oracle.com/en/java/

2) https://jgraph.github.io/mxgraph/docs/tutorial.html

3) https://www.baeldung.com/jgrapht