**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ**

**«МОСКОВСКИЙ ИНСТИТУТ СТАЛИ И СПЛАВОВ»**

ИНСТИТУТ ИТАСУ

КАФЕДРА Инженерной кибернетики

НАПРАВЛЕНИЕ 01.03.04 Прикладная математика

**КУРСОВАЯ**

**РАБОТА  
по дисциплине "Алгоритмы дискретной математики"**

**на тему:**

Задача о кратчайшем пути

*Выполнили студенты группы БПМ-18-1:*

*Кальянова А.В., Хохлова А.А., Кущ А.А., Уткин Л.В.*

*Отчет подготовила Кущ А.А.*

*Руководитель работы Пышняк М.О.*

*Оценка*

Москва\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020\_\_

Оглавление

[ВВЕДЕНИЕ 3](#_Toc40215444)

[1. АНАЛИТИЧЕСКИЙ ОБЗОР ЛИТЕРАТУРЫ 4](#_Toc40215445)

[1.1 История задачи о кратчайшем пути 4](#_Toc40215446)

[1.2 Основные понятия 4](#_Toc40215447)

[1.3 Алгоритмы решения задачи 5](#_Toc40215448)

[1.3.1 Алгоритм Дейкстры 5](#_Toc40215449)

[1.3.2 Алгоритм Беллмана-Форда 7](#_Toc40215450)

[1.3.3 Метод динамического программирования 8](#_Toc40215451)

[1.4 Современные области применения выбранных методов 9](#_Toc40215452)

[2. ОСНОВНАЯ ЧАСТЬ 10](#_Toc40215453)

[2.1 Содержательная постановка задачи 10](#_Toc40215454)

[2.2 Математическая постановка задачи 10](#_Toc40215455)

[2.2.1 Формулировка в терминах теории графов 10](#_Toc40215456)

[2.2.2 Формулировка в терминах динамического программирования 11](#_Toc40215457)

[2.3 Решение поставленной задачи 12](#_Toc40215458)

[2.3.1 Алгоритм Дейкстры 12](#_Toc40215459)

[2.3.2 Алгоритм Беллмана-Форда 12](#_Toc40215460)

[2.3.3 Сравнение методов решения 13](#_Toc40215461)

[2.4 Реализация методов решения задачи 14](#_Toc40215462)

[Заключение 16](#_Toc40215463)

[Список литературы 17](#_Toc40215464)

[Приложение 18](#_Toc40215465)

# 

# ВВЕДЕНИЕ

Данная работа направлена на изучение различных алгоритмов решения задачи о нахождении кратчайшего пути. Эта тема заинтересовала нас, поскольку вопрос о поиске пути из двух известных точек – А и Б – является актуальным на протяжении всей истории и в последнее время находит применение в новых областях жизни. С появлением компьютерных технологий тривиальная задача по поиску кратчайшего пути переросла в более сложную и требующую использования оптимальных алгоритмов. Они используются для ускорения и упрощения работы различных сервисов, к примеру, навигаторов, автопилотов или при передаче пакетов данных в компьютерных сетях.

Задача о кратчайшем пути является одной из классических в теории графов. Ее решение может быть определено на неориентированном, ориентированном и смешанном графе, и в нашей задаче будет рассматриваться решение относительно неориентированного графа без кратных ребер и петель.

Основной задачей данной курсовой работы является программная реализация алгоритма поиска кратчайшего пути между любыми двумя вершинами неориентированного графа, а также изучение и сравнение основных методов решения задачи о кратчайшем пути.

# 1. АНАЛИТИЧЕСКИЙ ОБЗОР ЛИТЕРАТУРЫ

## 1.1 История задачи о кратчайшем пути

Задача о кратчайшем пути является одной из классических задач теории графов, однако проследить ее точную историю трудно. Можно предположить, что вопросы о поиске пути между двумя известными точками поднимались в более примитивных обществах, чем существующее сейчас, однако по сравнению с другими задачами (к примеру, задачей коммивояжера), математические исследования в области задачи о кратчайшем пути начались относительно поздно. Это может быть связано с тем, что проблема является элементарной и относительно простой на малых множествах, а также имеет различные пути решения.

Толчок к развитию теории графов произошел на рубеже XI-XX столетий, на что оказали свое влияние естественные науки в области электрических сетей, молекулярных схем, а также увеличение работ в области топологии и комбинаторики. Широкое изучение задачи о кратчайшем пути началось во второй половине ХХ века, когда были предложены основные алгоритмы для оптимального решения задачи: алгоритм Дейкстры (1959), алгоритм Беллмана–Форда (1958), алгоритм поиска А\* (1968), алгоритм Флойда–Уоршелла (1962), алгоритм Джонсона (1977), алгоритм Ли (1961), а так же несколько других решений для различных видов графов.

## 1.2 Основные понятия

G – граф – это конечное множество вершин и конечное множество ребер(дуг), т.е. G(X,U,Ф); |U| – порядок (количество вершин), |X| – размерность (количество ребер), Ф — отношение инцидентности. Отношение инцидентности Ф является трехместным отношением Ф(х, u, у), где х, у ∈ X, u ∈ U, которое может либо выполняться (быть истинным), либо не выполняться (быть ложным) и удовлетворяет свойствам: 1) ∀u∈U, ∀х,у∈Х, Ф(х,u,у) — ребро всегда соединяет пару вершин. 2)(Ф(x,u,у)^Ф(х',u,у'))⇒((х=х'^у=у')у(х= у'^у = х')) — ребро и соответствует не более чем одной паре вершин х, у.

Если вершины х и у соединены ребром u, то говорят, что вершины х, у смежные, а ребро u инцидентно вершинам х и у. Два ребра называются смежными, если они имеют общую вершину.[1]

Важно учитывать, имеет ли значение расположение двух концов ребра: если этот порядок не существенен, то ребро считается неориентированным, иначе - ориентированным. Граф называется неориентированным, если каждое его ребро не ориентировано, и ориентированным, если все его ребра ориентированы.

Неориентированный граф называется простым, если он не имеет петель и любая пара вершин соединена не более чем одним ребром.

Путь (маршрут) на графе G(X,U,Ф) определяется последовательностью вершин и ребер х1, u1, х2, u2, х3...хn, un, хn+1, где xi∈X, ui∈U. Ребро ui соединяет вершину хi, с вершиной xi+l. Маршрут называется цепью, если все его ребра различные. Цепь называется простой, если не содержит одинаковых вершин.

Граф называется взвешенным, если каждому его ребру сопоставлено число.[2]

## 1.3 Алгоритмы решения задачи

В зависимости от постановки задачи нахождения кратчайшего пути существуют различные алгоритмы ее решения. Наиболее известные и эффективные из них, которые подходят для нахождения кратчайшего пути от одной из вершин графа до всех остальных – это алгоритм Дейкстры и алгоритм Белмана-Форда.

### 1.3.1 Алгоритм Дейкстры

Алгоритм Дейкстры был предложен голландским ученым Эдсгером Дейкстрой в 1959 году. Он считается одним из наиболее эффективных алгоритмов для нахождения кратчайшего пути в графе, при условии, что все ребра в графе будут с неотрицательными весами.

Данный алгоритм основан на фиксировании минимального пути до вершины. Пошагово проходя по всем вершинам, запоминается минимальное значение расстояния до этой вершины. Чтобы достичь этого, изначальное расстояние начальной точки считается нулем, а все непосещенные принимаются за бесконечности (то есть значение бесконечности показывает, что данная вершина еще не была посещена). Причем каждый шаг данного алгоритма состоит из:

1. Проставления минимального возможного значения расстояния от выбранной клетки до всех соседних вершин (тех к которым существуют ребра от текущей вершины), если они еще не были помечены посещенными. То есть находится расстояние, рассчитанное путем сложения метки расстояния у текущей вершины и длины ребра, соединяющего ее с соседней вершиной. Далее оно сравнивается с меткой расстояния у соответствующей соседней вершины.
2. Обозначения выбранной вершины помеченной.
3. Выбора следующей вершины, если все еще остались непомеченные. Причем следующая вершина обязательно должна быть непосещенной ранее и значение расстояния до нее должно быть минимальным из известных.[3]

Безусловным недостатком этого алгоритма является неверная работа с графами, имеющими ребра с отрицательными весами. Он возникает из-за того, что при выборе вершины, имеющей минимальную метку расстояния, считается что более короткого пути к ней не существует. Но при наличии отрицательных весов возможен вариант того, что на самом деле минимальным расстоянием будет путь через другие вершины, расстояние до которых еще не известно. Поэтому, если необходимо найти кратчайший путь в графе с отрицательными весами следует воспользоваться другим алгоритмом, который обеспечивает корректную работу с ними.

### 1.3.2 Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда был изобретен тремя независимыми математиками: Ричардом Беллманом, Лестером Фордом и Эдвардом Муром в 1958, 1956 и 1957 годах соответственно. Несмотря на это алгоритм редко называют алгоритмом Беллмана-Форда-Мура.

По сравнению с алгоритмом Дейкстры, помощью алгоритма Беллмана-Форда, можно находить кратчайший путь в графах, имеющих отрицательные значения весов. Поэтому он считается более универсальным, но в то же время он значительно медленнее алгоритма Дейкстры. Впрочем, у данного алгоритма также есть ограничение. С его помощью не поддаются решению графы с отрицательными циклами, то есть те, проходя по которым и возвращаясь в исходную точку общий путь уменьшается.

Основная идея алгоритма Беллмана-Форда заключается в том, что нужно пройти по всем ребрам графа n-1 раз, где n – это количество вершин в графе, каждый раз пытаясь уменьшить значение кратчайшего пути до вершины. В то же время можно значительно ускорить работу алгоритма, если завершать его выполнение при отсутствии изменении по окончании итерации.

Также существует модифицированная версия алгоритма Беллмана-Форда, которая позволяет обнаружить цикл отрицательного веса в графе. Для этого количество итераций в алгоритме не ограничивалось на количестве вершин в графе, уменьшенном на единицу, и проводилась еще одна дополнительная итерация. Соответственно, если на этой последней итерации веса определенных вершин обновились, то кратчайшего пути до них не существует из-за наличия отрицательного цикла на пути к ним.[4]

### 1.3.3 Метод динамического программирования

В реализации данных алгоритмов решения задачи нахождения кратчайшего пути является целесообразным выполнение операций шаг за шагом, иными словами разбиение задачи на подзадачи. Для этого хорошо подходит метод динамического программирования – это способ решения сложных и обычно объемных задач, основанный на представлении оптимального многошагового решения задачи, то есть разделение сложных задач на более простые. За счет этого происходит сокращение времени нужного на их выполнение. Объединение решений этих более простых задач в итоге приводит к готовому решению всей задачи в целом. Так как подзадачи зачастую повторяются, то достаточно будет решить их только один раз, что приводит к уменьшению объема необходимых вычислений, особенно когда количество повторяющихся задач достаточно велико.

Большой вклад в метод динамического программирования внес Ричард Беллман и его ученики. На сегодняшний день основными идеями этого метода являются:

1)    Оптимальность (использование оптимальных решений подзадач в общем решении, которое из них состоит).

2)    Мемоизация (сохранение результатов вычислений подзадач и использование их вместо повторного вычисления). [5]

Важным достоинством метода динамического программирования является возможность теоретической оценки объема необходимых вычислений. В то же время у этого метода имеется недостаток – это необходимость в слишком большом объеме памяти.

Применимость метода динамического программирования в задаче нахождения кратчайшего пути заключается в просматривании вершин по шагам. В то же время при нахождении в определенной вершине графа известен кратчайший путь до нее из изначальной позиции, то есть используется оптимальное решение.

## 1.4 Современные области применения выбранных методов

Алгоритм Дейкстры используется во многих областях программирования и технологий. Например, такие протоколы динамической маршрутизации, как OSPF и IS-IS, основаны на состоянии (link-based). Такой тип протоколов создает карту соединений сети в форме графа от каждого из узлов и с помощью алгоритма Дейкстры вычисляет наилучший путь передачи. Также очень часто его используют в сервисах и устройствах, которым необходимо проложить путь и определить наилучший маршрут следования. Немаловажную роль он играет в организации транспортных потоков и потоков населения, например при их эвакуации из мест бедствия.

Динамическое программирование в свою очередь применяется к задачам, в которых искомый ответ состоит из частей, каждая из которых представляет собой оптимальное решение некоторой подзадачи. Соответственно его применение целесообразно при проектировании решения сложных и многошаговых задач.

# 2. ОСНОВНАЯ ЧАСТЬ

## 2.1 Содержательная постановка задачи

Имеется список городов (вершин), список возможных перелетов (соединяющие ребра) и их неотрицательные координаты, которые преобразуются в расстояния между ними (веса ребер). Требуется решить задачу о кратчайшем пути для любых двух городов, то есть найти кратчайший путь (минимальное расстояние) между городами. Для решения задачи использовать оптимальный алгоритм – алгоритм Дейкстры – и методы динамического программирования.

В процессе работы над алгоритмом Дейкстры реализовать алгоритм Беллмана-Форда и протестировать оба варианта решения задачи о кратчайшем пути на разных условиях, чтобы убедиться, что основной выбранный алгоритм – алгоритм Дейкстры – является наиболее оптимальным.

## 2.2 Математическая постановка задачи

### 2.2.1 Формулировка в терминах теории графов

Имеется граф G(X,V) с неориентированными ребрами, V – множество вершин, X – множество ребер.

Пусть V = {0, 1, …, b}, X = {(ik) | k – i = aj; j = 1, …, n}. Каждой дуге (ik) с j–номером, где aj = k – i (разность номеров вершин) приписывается ее длина cj = c (ik), т.е. существует b вершин, соединенных ai ребрами, каждое из которых имеет длину сi.

Маршрутом (или путем) в графе G является конечная последовательность вершин и ребер (V0x1, V1x2, …, xiVi), в которой Vi ∈ V; xi ∈ X; i = 0, …,l; V0 = 0; Vi = b. L(p) – длина пути из вершины V0 в V1.

Для любого пути от V0 = 0 до Vi = b имеем aj = b. Требуется найти кратчайший путь из вершины в вершину.

### 2.2.2 Формулировка в терминах динамического программирования

Важная частью применения динамического программирования в данной задаче состоит в следующем: если последовательность вершин (V0,V1, …,Vi) определяет кратчайший путь от V0 до Vi, то последовательность (V0,V1, …, Vi) также определяет кратчайший путь от V1 до Vi. Это позволяет просматривать вершины по шагам, рассматривая на i–м шаге только те вершины, которые удалены от V0 на i дуг.

Ak – множество вершин, находящихся на шаге k, т.е.

Ak = {Vik | Vik ∈ V}

В каждую из вершин Vik входит, по крайней мере, одна дуга из множества вершин Ak-1 на стадии k–1, при этом A0 = {V0 = 0} и AN = {VN = b}.

На всех стадиях k в качестве параметров состояния Sk выбираются вершины на предыдущей стадии, то есть Sk = Vik–1, причем количество состояний равно |Ak–1|. Пусть Fk(Vik–1) – длина кратчайшего маршрута от вершины Vi k–1 до вершины VN = b. Из вершины Vi k–1 идут дуги в вершины Vjk.

Каждая дуга (ik–1, jk) характеризуется ее длиной (параметром) C(ik–1, jk), для отсутствующих дуг C = ∞. Индекс jk определяет вершину Vjk, в которую попадает дуга (i k–1, jk). С другой стороны количество таких вершин зависит от индекса i k–1, то есть Vjk (i k–1, jk) – это множество вершин, в которые входят дуги, выходящие из вершины Vi k–1. Имеем функциональное уравнение:

Fk (Vi k–1) = minj(k) {C (ik–1, jk) + Fk+1 (Vjk (ik–1, jk))}

k = 1,…,N; FN+1 = 0; V0 = 0

В соответствии с принципом ДП процесс оптимизации разбит на N стадий. Перебирая в процедуре минимизации индексы jk, рассматриваются различные вершины на стадии k, то есть разные функции Fk+1.

## 2.3 Решение поставленной задачи

Изначально программа считывает список городов с их координатами, список возможных путей и значения начальной и конечной вершин. Рассчитываются расстояния между городами. После чего с помощью выбранного алгоритма вычисляется кратчайший путь от начальной до конечной точки заданного маршрута.

### 2.3.1 Алгоритм Дейкстры

В случае с алгоритмом Дейкстры сначала происходит проверка на наличие смежных с начальной вершиной ребер. При их наличии рассматривается ближайшая. Если текущая суммарная длина до нее меньше той, которая была ранее отмечена минимальной в рассматриваемой вершине, то это значение меняется на новое значение суммарного пути. Вершина добавляется в конец стека если не отмечена посещенной. Эти же действия повторяются со всеми смежными ребрами. После этого вершина отмечается посещенной и выбирается следующая непосещенная вершина из стека. Действия повторяются пока существуют непосещенные соседние города (стек не пустой).

### 2.3.2 Алгоритм Беллмана-Форда

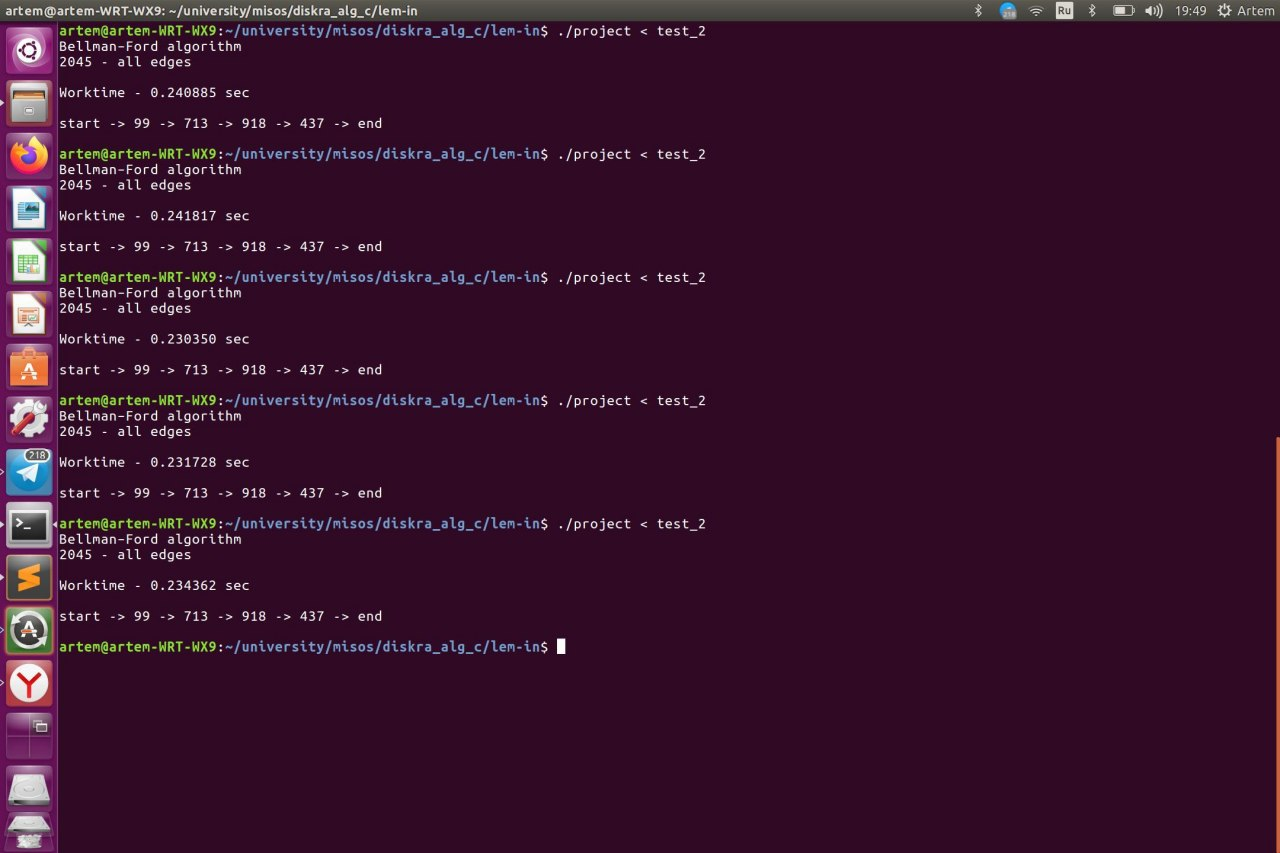
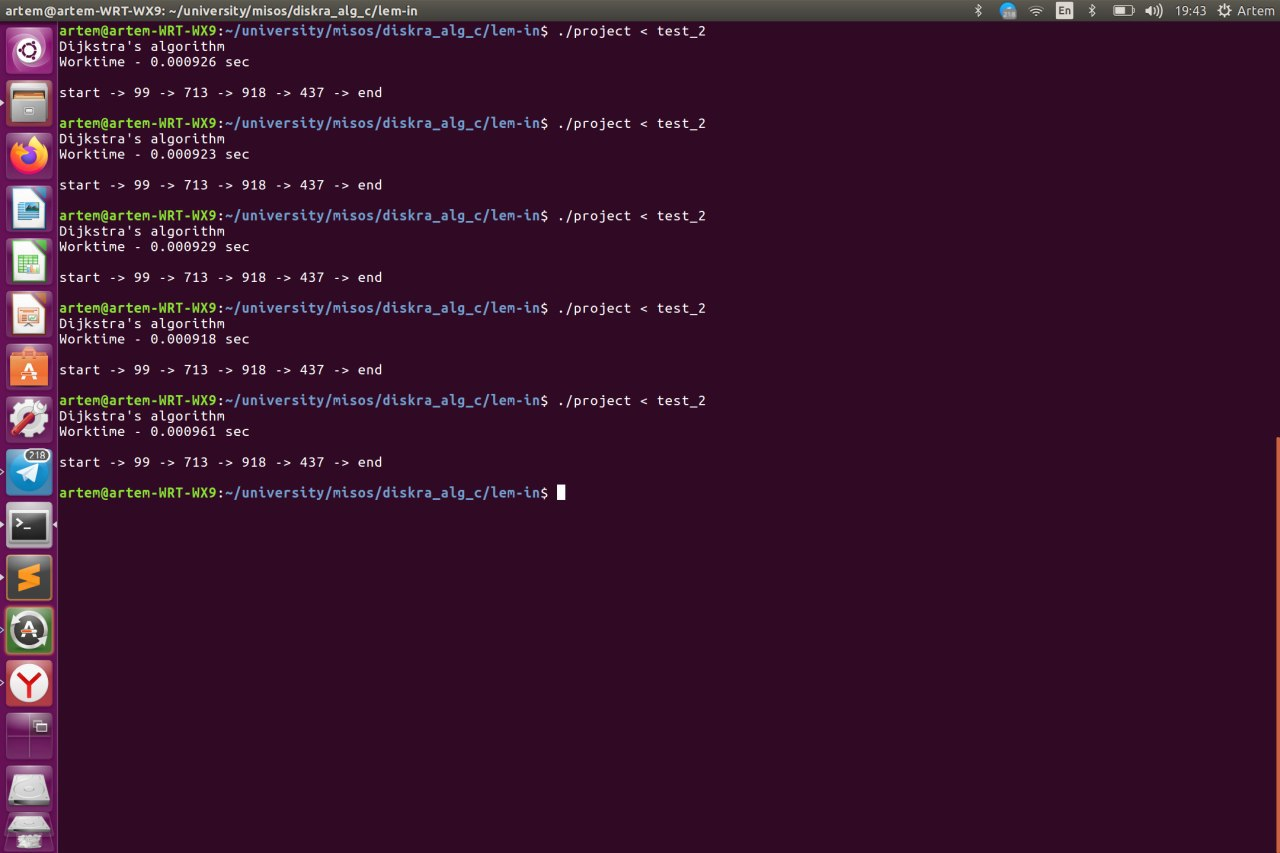
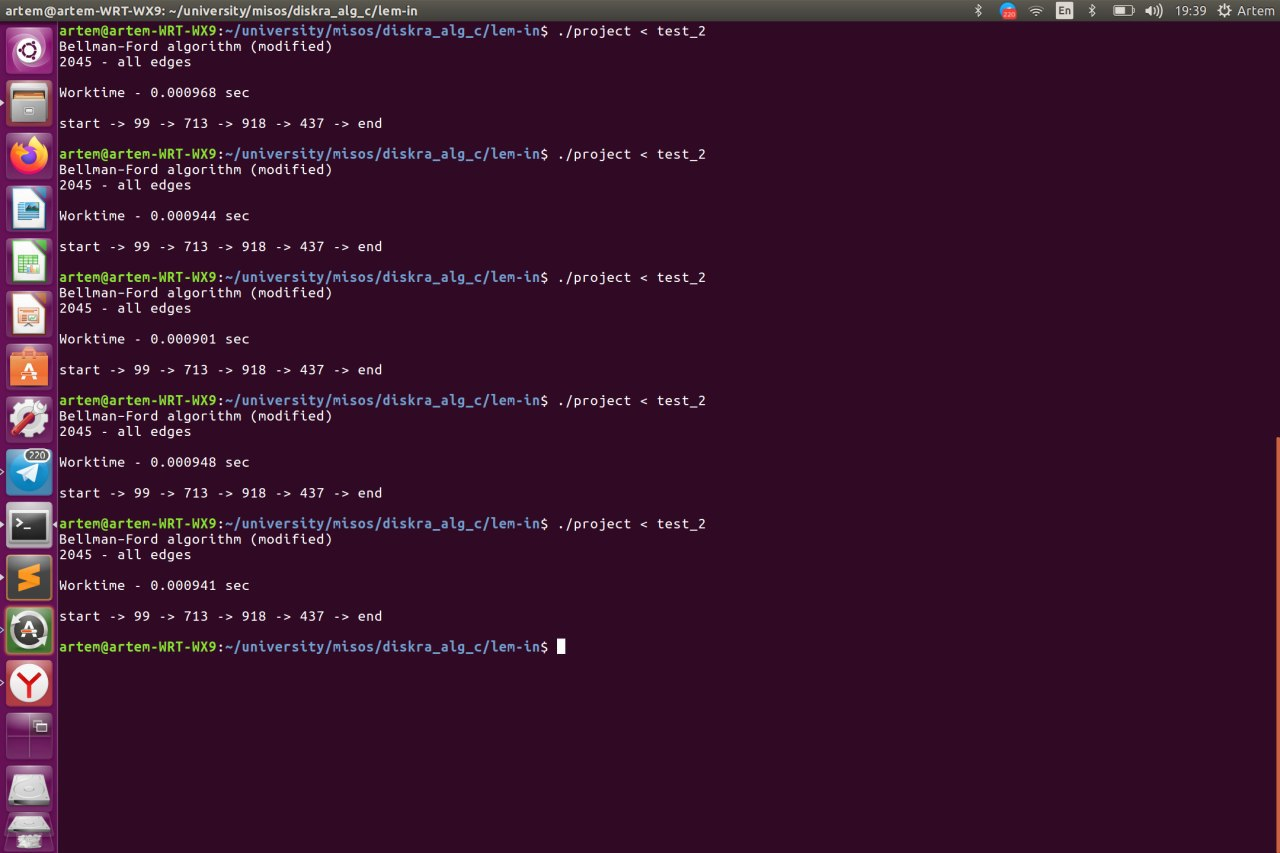
При использовании алгоритм Беллмана-Форда так же проверяется наличие смежных ребер. В данном случае мы проходим по всем вершинам и проставляем текущие длины до их соседних вершин, если они меньше того значения, которое там было до этого.

Также исследуется усовершенствованный алгоритм Беллмана-Форда, в котором завершается выполнение программы при отсутствии изменении по окончании итерации. Это достигается за счет добавления счетчика изменений (если он после очередной итерации равен нулю, то алгоритм завершается).

### 2.3.3 Сравнение методов решения

Сравнение методов реализовано на сравнении времени, которое необходимо для того, чтобы получить результат с помощью выбранных алгоритмов. В то же время тесты проводятся на одном вычислительном устройстве и с одинаковыми данными в тестах с одинаковыми количествами ребер и вершин графа.

Для сравнения методов используется функция clock(), которая возвращает время на момент ее вызова. Соответственно разница между временем начала выполнения алгоритма и временем получения результата его выполнения и будет временем требующемся алгоритму для работы.

Для тестов были сгенерированы и записаны в файлы 4 карты с разным количеством городов, их координат, начальных и конечных точек маршрута и соединяющими путями. Было проведено по пять тестов на этих четырех разных картах для каждого из алгоритмов, при этом происходит считывание исходных данных из файлов. Ниже приведены примеры выполнения одного из таких тестов на 3 алгоритмах.

В каждом таком блоке подсчитано средние значения полученных результатов. Результаты были занесены в таблицу:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** |  | test\_0 | test\_1 | test\_2 | test\_3 |
| **Bellman-Ford** | 1 | 0,36690 | 275,33910 | 0,24090 | 38,87630 |
| 2 | 0,37360 | 292,84790 | 0,24180 | 42,26852 |
| 3 | 0,36800 | NULL | 0,02304 | NULL |
| 4 | 0,36570 | NULL | 0,23170 | NULL |
| 5 | 0,36790 | NULL | 0,23440 | NULL |
| res | **0,36842** | **284,09350** | **0,19437** | **40,57241** |
| **Bellman-Ford (modified)** | 1 | 0,00790 | 0,04320 | 0,00096 | 0,01690 |
| 2 | 0,00780 | 0,03760 | 0,00094 | 0,01700 |
| 3 | 0,00800 | 0,03560 | 0,00091 | 0,01700 |
| 4 | 0,00790 | 0,03530 | 0,00095 | 0,01710 |
| 5 | 0,00790 | 0,03750 | 0,00094 | 0,01730 |
| res | **0,00790** | **0,03784** | **0,00094** | **0,01706** |
| **Dijkstra** | 1 | 0,00100 | 0,28000 | 0,00093 | 0,07180 |
| 2 | 0,00094 | 0,28490 | 0,00092 | 0,07170 |
| 3 | 0,00100 | 0,28380 | 0,00093 | 0,07260 |
| 4 | 0,00098 | 0,29410 | 0,00092 | 0,07120 |
| 5 | 0,00093 | 0,28390 | 0,00096 | 0,07490 |
| res | **0,00097** | **0,28534** | **0,00093** | **0,07244** |
| paths between cities |  | 3545 | 39825 | 2045 | 19973 |
| cities |  | 1500 | 25000 | 1000 | 10000 |

Очевидно, что алгоритм Беллмана-Форда работает существенно медленнее алгоритма Дейкстры. В то же время величина скорость измененной версии алгоритма Беллмана-Форда менее предсказуема нежели алгоритма Дейкстры. Соответственно, порой на вычисление кратчайшего пути может понадобиться существенно большее время по сравнению вариантом, в котором используется алгоритм Дейкстры. Эти недостатки обоих вариантов алгоритма Беллмана-Форда показывают, что использование алгоритма Дейкстры предпочтительнее.

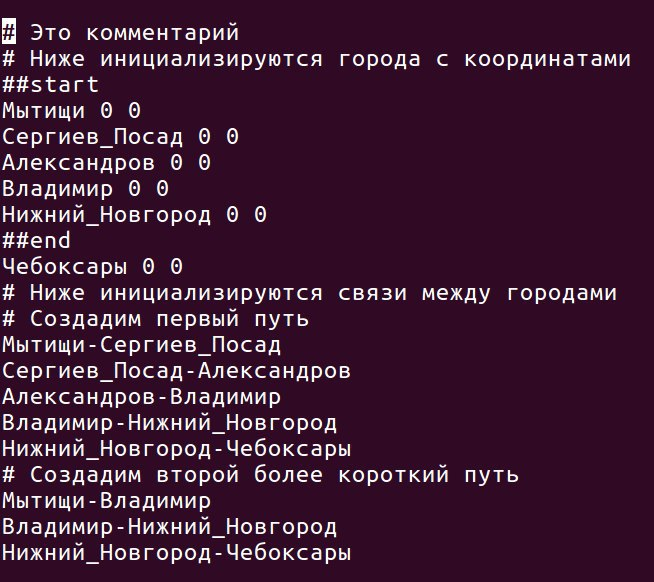
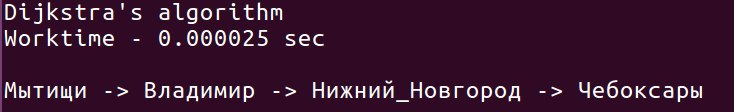
Исходя из полученных результатов тестирования можно сделать вывод что алгоритм Дейкстры оптимальный для решения задачи нахождения кратчайшего пути в неориентированном графе, в котором отсутствуют ребра с отрицательными весами.

## 2.4 Реализация методов решения задачи

Для реализации алгоритмов использовался язык программирования C. Для компиляции требуется IDE или только терминал, в котором будет использоваться утилита make. Листинг кода реализации алгоритмов находится в Приложении. Код всей программы расположен на веб-сервисе GitHub:

https://github.com/artemkush1/coursework\_discrete\_math.git

Кроме считывания карты из файла, реализован вариант отображения результата, который позволяет вводить необходимые данные карты самостоятельно с клавиатуры. Результат работы такой программы:

1. Создадим небольшую карту в правильном формате[[[1]](#footnote-1)]
2. Запустим алгоритм и найдем кратчайший путь

# Заключение

В данной курсовой работе были разобраны и реализованы два наиболее популярных и эффективных алгоритма нахождения кратчайшего пути, такие как алгоритм Дейкстры и алгоритм Беллмана-Форда. Скорости работы этих алгоритмов были проверены на разных объемах данных, исходя из полученных результатов получилось, что алгоритм Дейкстры действительно более подходящий для решения поставленной задачи. Также была написана программа, которая реализует нахождение кратчайшего пути в неориентированном графе, вершины которого даны в виде координат, и использует при этом оптимальный из исследуемых алгоритмов.

# Список литературы

1. Омельченко, А.В. Теория графов. – М.: МЦНМО, 2018

2. Новиков, Ф.А. Дискретная математика для программистов, 3-е издание. –Спб.: Питер, 2009

3. Cormen, Thomas H. / Leiserson, Charles E. Introduction to Algorithms, Dijkstra’s algorithm. – 2001

4. Статья «Алгоритм Форда-Беллмана»:

<https://e-maxx.ru/algo/ford_bellman>

5. Статья «Динамическое программирование»: <https://ru.wikipedia.org/wiki/Динамическое_программирование>

# Приложение

Ниже представлен код алгоритма Дейкстры с разбором:

static void alg\_deikstri(void)

{

// Инициализация переменных

t\_tmp \*start\_stack;

t\_tmp \*curr\_stack;

t\_next \*curr\_neigh;

t\_tmp \*new\_stack;

// Выделяем память под структуру (односвязный список), являющейся своеобразным кешем,

// и все параметры делаем NULL, дабы избежать набора мусора

start\_stack = ft\_memalloc(sizeof(t\_tmp));

curr\_stack = start\_stack; // Переводим указатель на начало структуры

curr\_stack->room = g\_lemin->start; // Переводим указатель на город “Старт”

// Пока кеш не пустой

while (curr\_stack)

{

curr\_neigh = curr\_stack->room->next; // Переводим указатель на соседний город

// Пока существуют соседние города

while (curr\_neigh)

{

// Проверяем, меньше ли вес перехода из текущего города в соседний

if (curr\_stack->room->min\_w + curr\_neigh->weight < curr\_neigh->room->min\_w)

{

// Присваиваем новый вес

curr\_neigh->room->min\_w = curr\_stack->room->min\_w + curr\_neigh->weight;

// Указываем новый предыдущий город

curr\_neigh->room->prev = curr\_stack->room;

// Проверяем, не посещали ли мы город ранее, дабы далее добавить в стек

if (!curr\_neigh->room->superpos)

{

// Пробежались до конца стека

new\_stack = curr\_stack;

while (new\_stack->next)

new\_stack = new\_stack->next;

// Выделили память под новый город в списке и добавили его в конец стека

new\_stack->next = ft\_memalloc(sizeof(t\_tmp));

new\_stack->next->room = curr\_neigh->room;

}

// Запрет на последующее добавление города в стек

curr\_neigh->room->superpos = 1;

}

// Переводим указатель на другой соседний город

curr\_neigh = curr\_neigh->next;

}

// Переводим указатель на следующую ячейку стека

curr\_stack = curr\_stack->next;

}

// Когда цикл завершится (закончится стек), мы выйдем из функции

}

Ниже представлен код алгоритма Беллмана-Форда с разбором:

// Передаю в функция список, содержащий в себе все города в порядке считывания из файла

static int alg\_bell\_ford(t\_tmp \*start)

{

// Инициализация переменных

t\_tmp \*curr;

t\_room \*prev\_r;

t\_next \*curr\_n;

t\_next \*prev\_n;

int counter;

curr = start; // Переводим указатель на первый город списка

counter = 0; // Счетчик. Считает, у скольких городов был пересчитан вес

while (curr)

{

curr\_n = curr->room->next; // Переводим указатель на соседний город

// Пока существуют соседние города

while (curr\_n)

{

prev\_r = curr\_n->room; // Передвигаем указатель на предыдущий город

// минимальный вес;

// город не является финишем;

// все веса в городах изначально равны (INT\_MAX / 2), дабы не уйти в переполнение

if (prev\_r->min\_w + curr\_n->weight < curr->room->min\_w

&& prev\_r != g\_lemin->finish && prev\_r->min\_w != (INT\_MAX / 2))

{

counter++;

curr->room->min\_w = prev\_r->min\_w + curr\_n->weight; // Присваиваем новый вес

curr->room->prev = prev\_r; // Указываем новый предыдущий город

}

// Переводим указатель на другой соседний город

curr\_n = curr\_n->next;

}

// Переводим указатель на следующий город

curr = curr->next;

}

// Когда цикл завершится (закончится список),

// мы выйдем из функции и вернем количество городов с измененным весом.

// Если counter == 0, то алгоритм Беллмана-Форда можно смело останавливать

return (counter);

}

1. Комментарий оставляется через символ ‘#’;

   после ‘##start’ и ‘##end’ записываются города, выступающие в роли старта и финиша;

   после инициализации всех городов записываются ребра (дороги) в формате ‘городо1-городо2’. [↑](#footnote-ref-1)