# НАУЧНО-ИССЛЕДОВАТЕЛЬКАЯ ЧАСТЬ

Выполнил: Карандаев В.Ю.

гр. СМ7-121

Проверил: Рубцов В.И.

* Анализ популярных алгоритмов поиска кратчайшего пути.
* Реализация алгоритма Ли (волнового алгоритма) для поиска кратчайшего пути по заданной карте местности

# Анализ популярных алгоритмов поиска кратчайшего пути

Задача о кратчайшем пути — задача поиска самого короткого пути (цепи) между двумя точками (вершинами) на графе, в которой минимизируется сумма весов ребер, составляющих путь.

Кратчайшая (простая) цепь часто называется геодезической.

Задача о кратчайшем пути является одной из важнейших классических задач теории графов. Сегодня известно множество алгоритмов для её решения.

У данной задачи существуют и другие названия: задача о минимальном пути или, в устаревшем варианте, задача о дилижансе.

Значимость данной задачи определяется её различными практическими применениями. Например в GPS-навигаторах, где осуществляется поиск кратчайшего пути между двумя перекрестками. В качестве вершин выступают перекрестки, а дороги являются ребрами, которые лежат между ними. Сумма расстояний всех дорог между перекрестками должна быть минимальной, тогда найден самый короткий путь. В нашем случае мы находим кратчайший путь от точки А до точки Б исходя из заданной карты местности и расположения начальной и конченых точек. В качестве вершин выступают неровности местности, кратчайшие возможные расстояния между этими вершинами являются ребрами.

**Существуют различные постановки задачи о кратчайшем пути:**

* Задача о кратчайшем пути в заданный пункт назначения.

Требуется найти кратчайший путь в заданную вершину назначения t, который начинается в каждой из вершин графа (кроме t). Поменяв направление каждого принадлежащего графу ребра, эту задачу можно свести к задаче о единой исходной вершине (в которой осуществляется поиск кратчайшего пути из заданной вершины во все остальные).

* Задача о кратчайшем пути между заданной парой вершин.

Требуется найти кратчайший путь из заданной вершины u в заданную вершину v.

* Задача о кратчайшем пути между всеми парами вершин.

Требуется найти кратчайший путь из каждой вершины u в каждую вершину v. Эту задачу тоже можно решить с помощью алгоритма, предназначенного для решения задачи об одной исходной вершине, однако обычно она решается быстрее.

В различных постановках задачи, роль длины ребра могут играть не только сами длины, но и время, стоимость, расходы, объем затрачиваемых ресурсов (материальных, финансовых, топливно-энергетических и т. п.) или другие характеристики, связанные с прохождением каждого ребра. Таким образом, задача находит практическое применение в большом количестве областей (информатика, экономика, география и др.).

В связи с тем, что существует множество различных постановок данной задачи, есть наиболее популярные алгоритмы для решения задачи поиска кратчайшего пути на графе:

**Алгоритм Дейкстры** находит кратчайший путь от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса.

**Алгоритм Беллмана — Форда** находит кратчайшие пути от одной вершины графа до всех остальных во взвешенном графе. Вес ребер может быть отрицательным.

**Алгоритм поиска A\*** находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной), используя алгоритм поиска по первому наилучшему совпадению на графе.

**Алгоритм Флойда — Уоршелла** находит кратчайшие пути между всеми вершинами взвешенного ориентированного графа.

**Алгоритм Джонсона** находит кратчайшие пути между всеми парами вершин взвешенного ориентированного графа.

**Алгоритм Ли (волновой алгоритм)** основан на методе поиска в ширину. Находит путь между вершинами s и t графа (s не совпадает с t), содержащий минимальное количество промежуточных вершин (ребер).

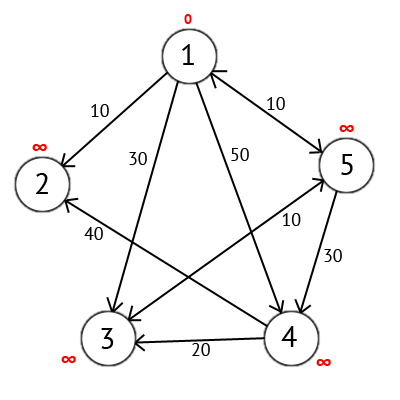
Рассмотрим три самых популярных алгоритма и выберем оптимальный для нашей задачи:

* Алгоритм Дейкстры;
* Алгоритм Беллмана — Форда;
* Алгоритм Ли (волновой алгоритм).

# Алгоритм Дейкстры

Принцип работы алгоритма Дейкстры - нахождение оптимальных маршрутов и их длины между одной конкретной вершиной (источником) и всеми остальными вершинами графа. Недостаток данного алгоритма в том, что он работает некорректно, если граф имеет дуги отрицательного веса.

Присвоим 1-й вершине метку равную 0, потому как эта вершина — источник. Остальным вершинам присвоим метки равные бесконечности.



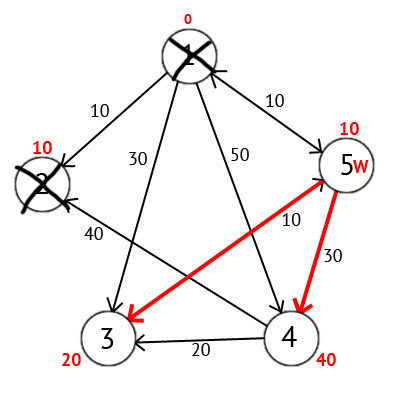
## Рис. #. Начальный этап алгоритма Дейкстры.

Далее выберем такую вершину W, которая имеет минимальную метку (сейчас это вершина 1) и рассмотрим все вершины в которые из вершины W есть путь, не содержащий вершин посредников. Каждой из рассмотренных вершин назначим метку равную сумме метки W и длинны пути из W в рассматриваемую вершину, но только в том случае, если полученная сумма будет меньше предыдущего значения метки. Если же сумма не будет меньше, то оставляем предыдущую метку без изменений.

## image Рис. #. Второй этап алгоритма Дейкстры.

После того как мы рассмотрели все вершины, в которые есть прямой путь из W, вершину W мы отмечаем как посещённую, и выбираем из ещё не посещенных такую, которая имеет минимальное значение метки, она и будет следующей вершиной W. В данном случае это вершина 2 или 5. Если есть несколько вершин с одинаковыми метками, то не имеет значения, какую из них мы выберем как W.

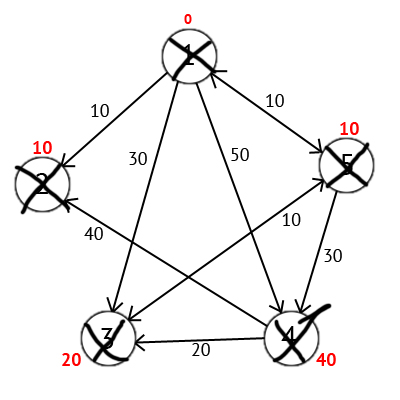
Мы выберем вершину 2. Но из нее нет ни одного исходящего пути, поэтому мы сразу отмечаем эту вершину как посещенную и переходим к следующей вершине с минимальной меткой. На этот раз только вершина 5 имеет минимальную метку. Рассмотрим все вершины в которые есть прямые пути из 5, но которые ещё не помечены как посещенные. Снова находим сумму метки вершины W и веса ребра из W в текущую вершину, и если эта сумма будет меньше предыдущей метки, то заменяем значение метки на полученную сумму.



## Рис. #. Третий этап алгоритма Дейкстры.

Исходя из картинки, мы можем увидеть, что метки 3-ей и 4-ой вершин стали меньше, то есть был найден более короткий маршрут в эти вершины из вершины источника. Далее отмечаем 5-ю вершину как посещенную и выбираем следующую вершину, которая имеет минимальную метку. Повторяем все перечисленные выше действия до тех пор, пока есть непосещенные вершины.

Выполнив все действия, получим такой результат:



## Рис. #. Заключительный этап алгоритма Дейкстры.

Также есть вектор Р, исходя из которого можно построить кратчайшие маршруты. По количеству элементов этот вектор равен количеству вершин в графе, Каждый элемент содержит последнюю промежуточную вершину на кратчайшем пути между вершиной-источником и конечной вершиной. В начале алгоритма все элементы вектора Р равны вершине источнику (в нашем случае Р = {1, 1, 1, 1, 1}). Далее на этапе пересчета значения метки для рассматриваемой вершины, в случае если метка рассматриваемой вершины меняется на меньшую, в массив Р мы записываем значение текущей вершины W. Например: у 3-ей вершины была метка со значением «30», при W=1. Далее при W=5, метка 3-ей вершины изменилась на «20», следовательно мы запишем значение в вектор Р — Р[3]=5. Также при W=5 изменилось значение метки у 4-й вершины (было «50», стало «40»), значит нужно присвоить 4-му элементу вектора Р значение W — P[4]=5. В результате получим вектор Р = {1, 1, 5, 5, 1}.

Зная что в каждом элементе вектора Р записана последняя промежуточная вершина на пути между источником и конечной вершиной, мы можем получить и сам кратчайший маршрут.

# Алгоритм Беллмана — Форда

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.

В отличие от алгоритма Дейкстры, этот алгоритм применим также и к графам, содержащим рёбра отрицательного веса. Впрочем, если граф содержит отрицательный цикл, то, понятно, кратчайшего пути до некоторых вершин может не существовать (по причине того, что вес кратчайшего пути должен быть равен минус бесконечности); впрочем, этот алгоритм можно модифицировать, чтобы он сигнализировал о наличии цикла отрицательного веса, или даже выводил сам этот цикл.

Алгоритм носит имя двух американских учёных: Ричарда Беллмана (Richard Bellman) и Лестера Форда (Lester Ford). Форд фактически изобрёл этот алгоритм в 1956 г. при изучении другой математической задачи, подзадача которой свелась к поиску кратчайшего пути в графе, и Форд дал набросок решающего эту задачу алгоритма. Беллман в 1958 г. опубликовал статью, посвящённую конкретно задаче нахождения кратчайшего пути, и в этой статье он чётко сформулировал алгоритм в том виде, в котором он известен нам сейчас.

## Формулировка задачи

Дан ориентированный или неориентированный граф G со взвешенными рёбрами. Длиной пути назовём сумму весов рёбер, входящих в этот путь. Требуется найти кратчайшие пути от выделенной вершины s до всех вершин графа.

Заметим, что кратчайших путей может не существовать. Так, в графе, содержащем цикл с отрицательным суммарным весом, существует сколь угодно короткий путь от одной вершины этого цикла до другой (каждый обход цикла уменьшает длину пути). Цикл, сумма весов рёбер которого отрицательна, называется отрицательным циклом.

## Решение задачи на графе без отрицательных циклов

Решим поставленную задачу на графе, в котором заведомо нет отрицательных циклов.

Для нахождения кратчайших путей от одной вершины до всех остальных, воспользуемся методом [динамического программирования](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Построим [матрицу](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) Aij, элементы которой будут обозначать следующее: Aij — это длина кратчайшего пути из s в i, содержащего не более *j* рёбер.

Путь, содержащий 0 рёбер, существует только до вершины *s*. Таким образом, Ai0 равно 0 при *i* = *s*, и +∞ в противном случае.

Теперь рассмотрим все пути из *s* в *i*, содержащие ровно *j* рёбер. Каждый такой путь есть путь из j-1 ребра, к которому добавлено последнее ребро. Если про пути длины j-1 все данные уже подсчитаны, то определить *j*-й столбец матрицы не составляет труда.

Так выглядит алгоритм поиска длин кратчайших путей в графе без отрицательных циклов:

**for** v \in V

**do** d[v] \gets +\infty

d[s] \gets 0

**for** i \gets 1 **to** |V| - 1

**do for** (u, v) \in E

**if** d[v] > d[u] + w(u, v)

**then** d[v] \gets d[u] + w(u, v)

**return** d

Здесь *V* — множество вершин графа *G*, *E* — множество его рёбер, а *w* — весовая функция, заданная на ребрах графа (возвращает длину [дуги](https://ru.wikipedia.org/wiki/%D0%94%D1%83%D0%B3%D0%B0_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2)" \l ".D0.B4.D1.83.D0.B3.D0.B0" \o "Дуга (теория графов)), ведущей из вершины *u* в *v*), d - массив, содержащий расстояния от вершины s до любой другой вершины.

Внешний цикл выполняется |V| - 1 раз, поскольку кратчайший путь не может содержать большее число ребер, иначе он будет содержать цикл, который точно можно выкинуть.

Вместо массива *d* можно хранить всю матрицу A, но это требует *O(V²)* памяти. Зато при этом можно вычислить и сами кратчайшие пути, а не только их длины. Для этого заведем матрицу *Pij*.

Если элемент *Aij* содержит длину кратчайшего пути из *s* в *i*, содержащего *j* рёбер, то *Pij* содержит предыдущую вершину до *i* в одном из таких кратчайших путей (ведь их может быть несколько).

Теперь алгоритм Беллмана–Форда выглядит так:

**for** v \in V

**for** i \gets 0 **to** |V| - 1

**do** A_{vi} \gets +\infty

A_{s0} \gets 0

**for** i \gets 1 **to** |V| - 1

**do for** (u, v) \in E

**if** A_{vi} > A_{u, i-1} + w(u, v)

**then** A_{vi} \gets A_{u, i-1} + w(u, v)

P_{vi} \gets u

После выполнения этого алгоритма элементы A_{i, j} содержат длины кратчайших путей от *s* до *i* с количеством ребер *j*, и из всех таких путей следует выбрать самый короткий. А сам кратчайший путь до вершины *i* с *j* ребрами восстанавливается так:

**while** j > 0

p[j] \gets i

i \gets P_{ij}

j \gets j - 1

**return** *p*

## Граф с отрицательными циклами

Алгоритм Беллмана–Форда позволяет очень просто определить, существует ли в графе *G* отрицательный цикл, достижимый из вершины *s*. Достаточно произвести внешнюю итерацию цикла не |V| - 1, a ровно |*V*| раз. Если при исполнении последней итерации длина кратчайшего пути до какой-либо вершины строго уменьшилась, то в графе есть отрицательный цикл, достижимый из *s*. На основе этого можно предложить следующую оптимизацию: отслеживать изменения в графе и, как только они закончатся, сделать выход из цикла (дальнейшие итерации будут бессмысленны).

# Алгоритм Ли

Алгоритм Ли так же известен как Волновой алгоритм или Алгоритм волновой трассировки. Алгоритм работает на дискретном рабочем поле (ДРП), представляющем собой ограниченную замкнутой линией фигуру, не обязательно прямоугольную, разбитую на прямоугольные ячейки, в частном случае — квадратные. Множество всех ячеек ДРП разбивается на подмножества: «проходимые» (свободные), т. е при поиске пути их можно проходить, «непроходимые» (препятствия), путь через эту ячейку запрещён, стартовая ячейка (источник) и финишная (приемник). Назначение стартовой и финишной ячеек условно, достаточно — указание пары ячеек, между которыми нужно найти кратчайший путь.

Алгоритм предназначен для поиска кратчайшего пути от стартовой ячейки к конечной ячейке, если это возможно, либо, при отсутствии пути, выдать сообщение о непроходимости.

Работа алгоритма включает в себя три этапа: инициализацию, распространение волны и восстановление пути.

Во время инициализации строится образ множества ячеек обрабатываемого поля, каждой ячейке приписываются атрибуты проходимости/непроходимости, запоминаются стартовая и финишная ячейки.

Далее, от стартовой ячейки порождается шаг в соседнюю ячейку, при этом проверяется, проходима ли она, и не принадлежит ли ранее меченной в пути ячейке.

Соседние ячейки принято классифицировать двояко: в смысле [окрестности Мура](https://ru.wikipedia.org/wiki/%D0%9E%D0%BA%D1%80%D0%B5%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%9C%D1%83%D1%80%D0%B0" \o "Окрестность Мура) и [окрестности фон Неймана](https://ru.wikipedia.org/wiki/%D0%9E%D0%BA%D1%80%D0%B5%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D1%84%D0%BE%D0%BD_%D0%9D%D0%B5%D0%B9%D0%BC%D0%B0%D0%BD%D0%B0" \o "Окрестность фон Неймана), отличающийся тем, что в окрестности фон Неймана соседними ячейками считаются только 4 ячейки по вертикали и горизонтали, в окрестности Мура — все 8 ячеек, включая диагональные.

При выполнении условий проходимости и непринадлежности её к ранее помеченным в пути ячейкам, в атрибут ячейки записывается число, равное количеству шагов от стартовой ячейки, от стартовой ячейки на первом шаге это будет 1. Каждая ячейка, меченая числом шагов от стартовой ячейки становится стартовой и из неё порождаются очередные шаги в соседние ячейки. Очевидно, что при таком переборе будет найден путь от начальной ячейки к конечной, либо очередной шаг из любой порождённой в пути ячейки будет невозможен.

Восстановление кратчайшего пути происходит в обратном направлении: при выборе ячейки от финишной ячейки к стартовой на каждом шаге выбирается ячейка, имеющая атрибут расстояния от стартовой на единицу меньше текущей ячейки. Очевидно, что таким образом находится кратчайший путь между парой заданных ячеек. Трасс с минимальной числовой длиной пути, как при поиске пути в окрестностях Мура, так и фон Неймана может существовать несколько. Выбор окончательного пути в приложениях диктуется другими соображениями, находящимися вне этого алгоритма.

Результат работы волнового алгоритма представлен на рисунке.

## D:\Бауманка\12 семестр\!Диплом\Научно-исследовательская часть\Lee_wave_4.png

## Рис. #. Результат работы волнового алгоритма.

Основное преимущество данного алгоритма заключается в легкости его реализации. Так же, реализовывая данный алгоритм, мы получаем массив чисел, пронумерованных по номеру движения и имеющий определенные, ранее заложенные действия для нашего робота.

# Реализация Алгоритма Ли для поиска кратчайшего пути по заданной карте местности

Проанализировав основные алгоритмы, выясняем, что самый оптимальный для решения нашей задачи – это Волновой алгоритм или Алгоритм Ли. Реализация в рамках данной дипломной работы будет осуществленная на языке программирования C++.

Как отмечено выше, структура кода разделена на три части: инициализация, распространение волны и восстановление пути.

В первой части инициализации мы дискретизируем карту местности, задаем ее габариты, отмечаем непроходимые и проходимые участки, а так же задаем начало и конец пути.

В части распространения волны мы отмечаем конечную точку *d*, и, начиная с нее, проверяем все соседние ячейки на проходимость. Если ячейка проходима, то присваиваем ей значение *d+1.* И так пока «волна» не доберется до финальной ячейки.

В третьей, заключительной, части восстановления пути, по имеющимся сохраненным данным, мы восстанавливаем путь от начальной точки до конечной, по самому короткому маршруту.

Подробный код представлен в приложении.