Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Фундаментальные науки
КАФЕДРА Прикладная математика

ОТЧЕТ ПО ПРАКТИКЕ

Студент Марцинович Софья Эльдаровна
фамилия, имя, отчество

Группа ФН2-31Б

Тип практики: Ознакомительная практика

Название предприятия: НУК ФН МГТУ им. Н.Э. Баумана

Студент _____ Марцинович С.Э.
подпись, дата *фамилия и.о.*

Руководитель практики _____ Попов А.Ю.
подпись, дата *фамилия и.о.*

Оценка _____

2019 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»

(МГТУ им. Н.Э. Баумана)

Кафедра «Прикладная математика»

З А Д А Н И Е
на прохождение ознакомительной практики

на предприятии НУК ФН МГТУ им. Н.Э. Баумана

Студент Марцинович Софья Эльдаровна
фамилия, имя, отчество

Во время прохождения ознакомительной практики студент должен

1. Изучить на практике основные возможности языка программирования C++ и систем компьютерной алгебры, закрепить знания и умения, полученные в курсах «Введение в информационные технологии», «Информационные технологии профессиональной деятельности».
2. Изучить способы реализации методов решения задачи поиска минимального пути на графе.
3. Реализовать алгоритм Беллмана - Форда.

Дата выдачи задания «22» сентября 2019 г.

Руководитель практики _____ Попов А.Ю.
подпись, дата *фамилия и.о.*

Студент _____ Марцинович С.Э.
подпись, дата *фамилия и.о.*

Содержание

Введение

Основной целью ознакомительной практики 3-го семестра, входящей в учебный план подготовки бакалавров по направлению 01.03.04 – Прикладная математика, является знакомство с особенностями осуществления деятельности в рамках выбранного направления подготовки и получение навыков применения теоретических знаний в практической деятельности.

В ранее пройденном курсе «Введение в специальность» произошло общее знакомство с возможными направлениями деятельности специалистов в области прикладной математики и получен опыт оформления работ (реферата), который полезен при оформлении отчета по практике.

В рамках освоенного курса «Введение в информационные технологии» изучены основные возможности языка программирования C++ и сформированы базовые умения в области программирования на C++. Задачей практики является закрепление соответствующих знаний и умений и овладение навыками разработки программ на языке C++, реализующих заданные алгоритмы. Кроме того, практика предполагает формирование умений работы с системами компьютерной алгебры и уяснение различий в принципах построения алгоритмов решения задач при их реализации на языках программирования высокого уровня (к которым относится язык C++) и на языках функционального программирования (реализуемых системами компьютерной алгебры).

Формулировка индивидуального задания

Поиск кратчайшего пути на сети - II

Задана сеть из некоторого количества пронумерованных узлов (для которых считаются известными их координаты на плоскости), а также списка ребер, для каждого из которых заданы номера узлов, которые оно соединяет, и некоторое положительное число, определяющее «стоимость» движения по этому ребру. Ребра считаются однонаправленными; если возможно движение в обе стороны, следует задавать два ребра, «стоимость» движения по которым может быть различной.

Найти оптимальные маршруты для нескольких заданных пар вершин.

а) решить задачу «перебором», прокладывая все возможные пути (не заходя дважды в одну и ту же вершину);

б) решить задачу методом Беллмана - Форда (Bellman - Ford).

Структура исходного файла данных:

```
n                 << количество узлов сети
x1 y1             << координаты первого узла сети
…
xn yn             << координаты n-го узла сети
p                 << количество ребер сети
a1 b1 s1          << номера узлов, которые соединяет ребро,
                  и стоимость
                  движения от a1 к b1
…
ap bp sp          << номера узлов, которые соединяет ребро,
                  и стоимость
                  движения от ap к bp
q                 << количество пар точек, для которых надо
                  проложить
                  маршрут
a1 b1             << первая пара узлов для прокладки
                  маршрута
…
aq bq             << q-я пара узлов для прокладки маршрута
```

Структура файла результата:

```
q                 << количество пар точек, для которых
                  надо проложить
                  маршрут
n1                << число узлов в маршруте между первой
                  парой точек
c11 c12 …         << номера узлов, образующих первый
c1(n1)            маршрут
…
nq                << число узлов в маршруте между q-й
                  парой точек
cq1 cq2 …         << номера узлов, образующих q-й маршрут
cq(nq)
```

1. История рассматриваемой задачи

Задача о кратчайшем пути — задача поиска самого короткого пути между двумя точками (вершинами) на графе, в которой минимизируется сумма весов ребер, составляющих путь. Эта задача является одной из важнейших классических задач теории графов. Сегодня известно множество алгоритмов для ее решения.

Задача поиска кратчайшего пути на графе может быть определена для неориентированного, ориентированного или смешанного графа. Далее будет рассмотрена постановка задачи для ориентированного графа, то есть в ходе решения будут учитываться на-правления ребер.

Существуют различные постановки задачи о кратчайшем пути [3]:

- Задача о кратчайшем пути в заданный пункт назначения.
- Задача о кратчайшем пути между заданной парой вершин.
- Задача о кратчайшем пути между всеми парами вершин.

В нашем случае речь пойдет о задаче поиска кратчайшего пути между заданной парой вершин.

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 [4].

Задача находит практическое применение в большом количестве областей (информатика, экономика, география и др.). Например, в GPS-навигаторах осуществляется поиск кратчайшего пути между двумя перекрестками. В качестве вершин выступают перекрестки, а дороги являются ребрами, которые лежат между ними. Если сумма длин дорог между перекрестками минимальна, тогда найденный путь самый короткий [3].

2. Обзор методов решения

В связи с тем, что существует множество различных постановок данной задачи, есть наиболее применяемые алгоритмы для решения задачи поиска кратчайшего пути на графе. Рассмотрим самые популярные из них.

Алгоритм Дейкстры (англ. Dijkstra’s algorithm)

Данный алгоритм был изобретен нидерландским ученым Эдсгером Дейкстрой в 1959 году. Он находит кратчайшие пути от одной из вершин графа до всех остальных. В случае разреженных графов с ребрами неотрицательного веса лучшим выбором считается использование именно этого алгоритма. Алгоритм широко применяется в программировании и технологиях. Например, его используют протоколы маршрутизации [5].

Алгоритм Беллмана - Форда (англ. Bellman - Ford algorithm)

Алгоритм позволяет найти кратчайшие пути от одной вершины графа до всех остальных во взвешенном графе. Вес ребер может быть отрицательным. Особенности использования и реализация будут рассмотрены ниже.

Алгоритм Флойда - Уоршелла (англ. Floyd - Warshall algorithm)

Этот алгоритм был разработан в 1962 году Робертом Флойдом и Стивеном Уоршеллом. Он служит для нахождения кратчайших путей между всеми парами вершин взвешенного ориентированного графа. Алгоритм Флойда - Уоршелла является эффективным для расчета всех кратчайших путей в плотных графах, когда имеет место большое количество пар ребер между парами вершин [5].

Алгоритм Джонсона (англ. Johnson's algorithm)

С помощью алгоритма можно найти кратчайшие пути между всеми парами вершин взвешенного ориентированного графа. Алгоритм либо возвращает матрицу кратчайших расстояний между всеми парами вершин, либо сообщение о том, что в графе существует цикл отрицательной длины. Для разреженных графов он ведет себя быстрее алгоритма Флойда - Уоршелла.

Алгоритм Ли или волновой алгоритм (англ. Lee algorithm)

Данный алгоритм основан на методе поиска в ширину. Он находит путь между вершинами s и t графа (s не совпадает с t), содержащий минимальное количество промежуточных вершин (ребер). Основное применение — трассировки электрических соединений на кристаллах микросхем и на печатных платах. Также используется для поиска кратчайшего расстояния на карте в стратегических играх [3].

3. Метод перебора

3.1. Описание метода перебора

Первой идеей реализации перебора был полный перебор. Планировалось рассмотреть в качестве потенциальных путей все возможные перестановки из порядковых номеров узлов. При генерации перестановок использовалась рекурсивная функция.

На выходе имеется $n!$ перестановок. Такое решение задачи подходит только для графов с малым количеством узлов. Экспериментальным путем было установлено, что программа работает на графах с количеством узлов $n \leq 10$.

Так как область применимости данного алгоритма сильно ограничена, было решено применить другой подход к решению задачи. Рекурсивную функцию, приведенную выше, можно улучшить, если с помощью условий отсечь очевидно несуществующие пути.

В новой рекурсивной функции составляется список из всех потенциально возможных путей. Узел добавляется в путь, если выполняются следующие условия: к этому узлу можно прийти из предыдущего; этого узла еще не было в пути. Таким образом, список из перестановок сокращается до списка из существующих путей. Работа рекурсии будет подробнее рассмотрена в следующем разделе.

По результатам работы рекурсии получается список всех возможных

путей. Из них выбирается путь с минимальным весом.

Такая реализация решения задачи, в отличие от полного перебора перестановок, позволяет работать с большими графами без выброса сообщения об ошибке нехватки памяти.

Недостатком данного подхода является нечувствительность к циклам отрицательного веса. Даже если в тесте имеется такой цикл, программа закончит свое выполнение и запишет в файл вывода найденный путь. Интерпретировать данный ответ можно по договоренности. С одной стороны, минимального пути в графе с циклом отрицательного веса не существует, так как каждый новый проход по циклу уменьшает значение общего веса. С другой стороны, по условию задачи запрещено проходить через один узел дважды. Следовательно, проход по циклу в принципе невозможен и полученный результат можно считать правильным ответом.

3.2. Блок-схема алгоритма метода перебора

Составим блок-схему рекурсивного алгоритма перебора путей.

Обозначим стартовую вершину как *start*, финишную вершину как *finish*. В ходе рекурсии *rec* стартовая вершина меняется, текущую стартовую вершину будем хранить в переменной *s*. При этом финишная вершина постоянна. Выход из рекурсии происходит при переборе всех вершин, смежных со стартовой. Сама блок-схема приведена на рис. 1.

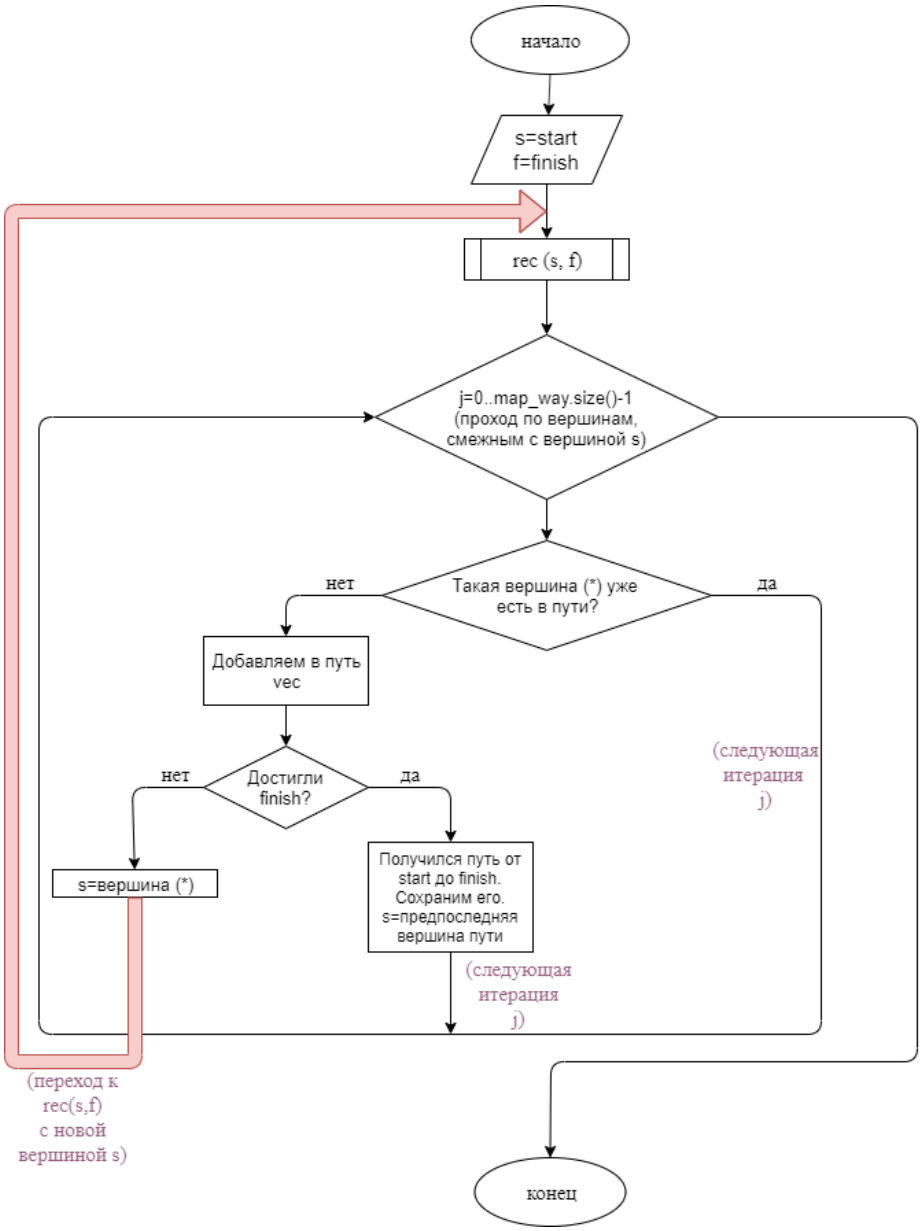


Рис. 1. Блок-схема рекурсивного алгоритма

4. Алгоритм Беллмана - Форда

4.1. Описание алгоритма Беллмана - Форда

История алгоритма связана сразу с тремя независимыми математиками: Лестером Фордом, Ричардом Беллманом и Эдвардом Муром. Форд и Беллман опубликовали алгоритм в 1956 и 1958 годах соответственно, а Мур сделал это в 1957 году. И иногда его называют алгоритмом Беллмана - Форда - Мура [5].

Алгоритм Беллмана - Форда вычисляет во взвешенном графе кратчайшие пути от одной вершины до всех остальных. Он подходит для работы с графами, имеющими ребра отрицательного веса. Но спектр применимости алгоритма затрагивает не все такие графы из-за того, что каждый очередной проход по пути, составленному из ребер, сумма весов которых отрицательна (т. е. по отрицательному циклу), лишь улучшает требуемое значение. Бесконечное число улучшений делает невозможным определение одного конкретного значения, являющегося оптимальным. В связи с этим алгоритм Беллмана - Форда не применим к графам, имеющим отрицательные циклы, но он позволяет определить наличие таковых [6].

Время выполнения программы, имеет порядок $O(n \cdot m)$, где n — это количество вершин, а m — это количество ребер. Наилучшим случаем для алгоритма станет граф, в котором две вершины и отсутствуют ребра.

4.2. Псевдокод алгоритма Беллмана - Форда

Ниже приведен псевдокод алгоритма Беллмана - Форда:

```
procedure shortest-paths (G, l, s)
Input: Directed graph G = (V, E);
edge lengths {l_e: e ∈ E} with no negative cycles;
vertex s ∈ V.
Output: For all vertices u reachable from s, dist(u) is set to
the                distance from s to u.

for all u ∈ V:
    dist(u) = ∞;
    prev(u) = nil;
dist(s) = 0;
repeat |V|-1 times:
    for all e ∈ E:
         update(e);
```

```
procedure update ((u, v) ∈ E)
dist(v) = min{ dist(v), dist(u)+l(u,v)}
```

Пусть V — множество вершин графа G, E — множество его ребер.

Для хранения результатов работы алгоритма создадим вектор `dist(u)`. В каждом его u-ом элементе будет храниться значение кратчайшего пути из вершины s до вершины u (если таковой имеется). Изначально, присвоим элементам вектора `dist(u)` значения равные условной бесконечности (число заведомо большее суммы всех весов), а в элемент `dist(s)` запишем нуль.

Назовем процедурой `update()` релаксацию ребра `(u,v)`. Под релаксацией понимается уменьшение значения `dist(v)` до `dist(u)+l(u,v)`, если второе значение меньше первого.

Внешний цикл выполняется $|V| - 1$ раз, поскольку кратчайший путь не может содержать большее число ребер, иначе он будет содержать цикл, то есть последовательность с повторяющимися узлами.

Внутренний цикл отвечает за проход по всем ребрам. Для каждого ребра выполняется релаксация.

5. Реализация метода перебора

5.1. Особенности реализации на языке C++

При реализации метода перебора было принято решение хранить граф в виде списка смежности [8]. Реализация списка смежности выполнена с помощью словаря *map*, в котором ключ — номер вершины, а значение — список смежных с ней вершин (список реализован с помощью контейнера *vector*).

Пример списка смежности для разреженного графа:

```
2: {11,9}
3: {2,10}
4: {3}
```

```
5: {2,6}
7: {5,8}
8: {1,4}
9: {6,8}
10: {6}
11: {1,5,6}
```

Для реализации метода перебора была использована рекурсивная функция. Предварительно из списка ребер были удалены те ребра, которые соединяют одинаковые пары узлов. В итоге из двух дублирующих друг друга ребер сохраняется то, которое имеет наименьший вес.

Одним из условий добавления узла в путь является то, что до этого в пути он не встречался. Это условие позволяет алгоритму безопасно отработать в случае цикла отрицательного веса, но не дает возможности установить наличие такого цикла.

5.2. Особенности реализации в системе компьютерной алгебры

В системе компьютерной алгебры Wolfram Mathematica перебор всех перестановок был реализован с помощью функции *Permutations*. В отличие от реализованной на языке C++ функции, которая генерирует все перестановки длины *n*, эта функция позволяет генерировать перестановки различной длины (первый элемент – стартовая вершина, последний – финишная вершина). В итоге получается меньшее число перестановок, что сокращает время выполнения программы: вместо *n!* имеем $\sum_{k=2}^{n-2} \frac{(n-2)!}{(n-k)!}$ перестановок.

6. Реализация алгоритма Беллмана - Форда

6.1. Особенности реализации на языке C++

При реализации алгоритма Беллмана - Форда было принято решение хранить граф в виде списка ребер. Был создан вектор, каждый элемент которого хранит в себе информацию об одном ребре: две вершины, инцидентные ребру, и вес ребра. Размер занимаемой памяти: *O(|E|)*.

Рассмотрим задачу восстановления пути. Заведем вектор, в котором для каждой вершины будем хранить ее «предка», то есть предпоследнюю вершину в кратчайшем пути, ведущем в нее. Изначально инициализируем элементы вектора константой *INF* («бесконечность»), то есть заведомо большим значением. После завершения всех релаксаций осуществляется проход по «предкам» и сохраняется пройденный путь, но в обратном порядке. Метод *reverse* вызывается для того, чтобы обратить вектор, хранящий этот путь.

Заметим, что для недостижимых из вершины-старта вершин алгоритм работает корректно: значение пути до них так и останется равным *INF*.

Описанный алгоритм можно несколько ускорить. Иногда ответ можно найти за количество фаз меньшее, чем $(n - 1)$. В этом случае в остальных фазах релаксации не происходят и ребра просматриваются впустую. Будем хранить флаг, который показывает, произошла ли на предыдущей фазе хотя бы одна релаксация. Если нет релаксаций, то можно остановить выполнение алгоритма.

Однако, было принято решение ограничить количество фаз сверху: в случае присутствия отрицательного цикла флаг всегда будет принимать значение *True*, и выход из цикла не произойдет.

Вводим вспомогательную переменную. По ее значению можно найти цикл отрицательного веса. Если после $(n - 1)$-ой фазы выполнится еще одна фаза, и на ней произойдет хотя бы одна релаксация, то граф содержит цикл отрицательного веса. Узлы, в него входящие, восстанавливаются с помощью вектора «предков».

6.2. Особенности реализации в системе компьютерной алгебры

В системе компьютерной алгебры Wolfram Mathematica реализуется хранение графа с помощью *Table*. Координаты представлены в структурах $\{x, y\}$, ребра в структурах $\{node1, node2, weight\}$, пары в структурах

{*node1, node2*}.

Для работы со списками были использованы внутренние функции системы. Таким образом, удалось избавится от циклов, отвечающих за перебор элементов списка.

Файл, полученный в результате выполнения программы, в дальнейшем используется для визуализации найденного минимального пути. О функциях визуализации будет подробнее сказано ниже.

7. Визуализация задачи в системе компьютерной алгебры

В системе компьютерной алгебры Wolfram Mathematica существует встроенная функция для визуализации графа. В качестве аргументов учитываются инцидентность ребер и узлов и вес ребер. Функция не предоставляет возможности стоить граф с учетом его расположения в координатной плоскости. На рис. 2 изображен граф, который будет упоминаться в примере 2 в соответствующем разделе.

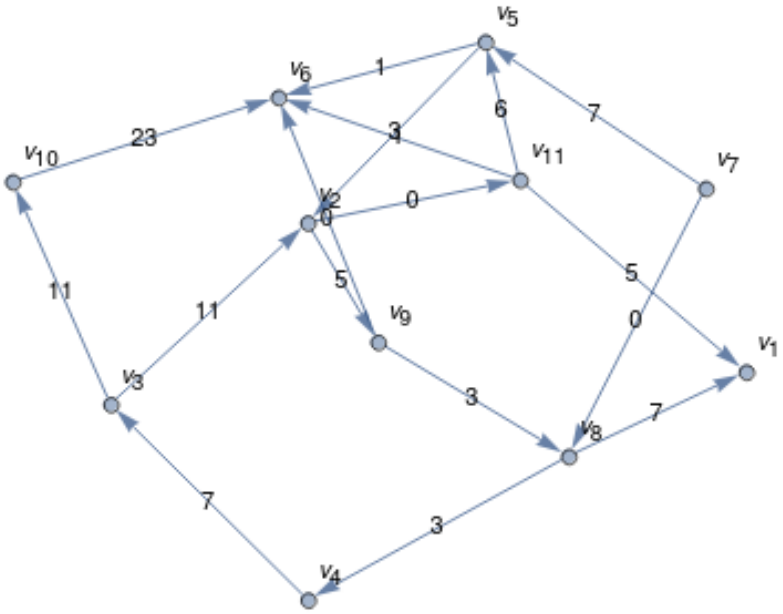


Рис. 2. Построение графа с помощью внутренних средств Wolfram Mathematica

Для визуализации графа с учетом заданных координат узлов была реализована собственная функция. В процессе построения изображения

были использованы такие инструменты Wolfram Mathematica как *Show*, *Graphics* (рис. 3).

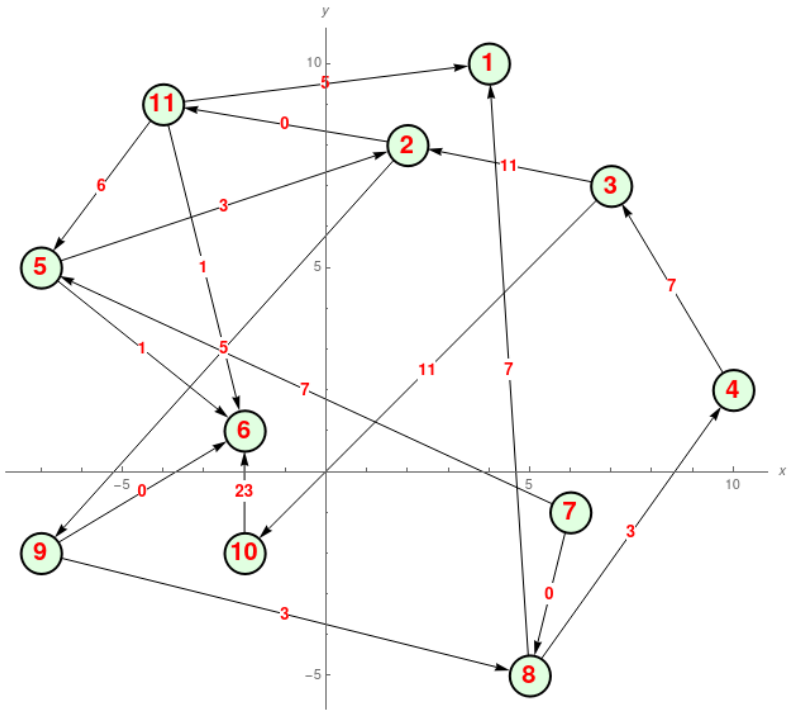


Рис. 3. Построение исходного графа в Wolfram Mathematica

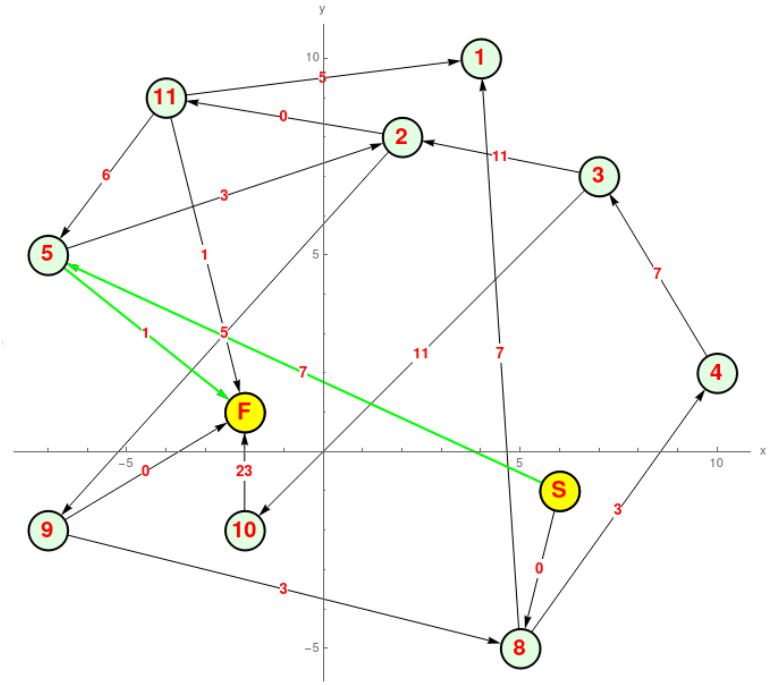


Рис. 4. Построение найденного пути в Wolfram Mathematica

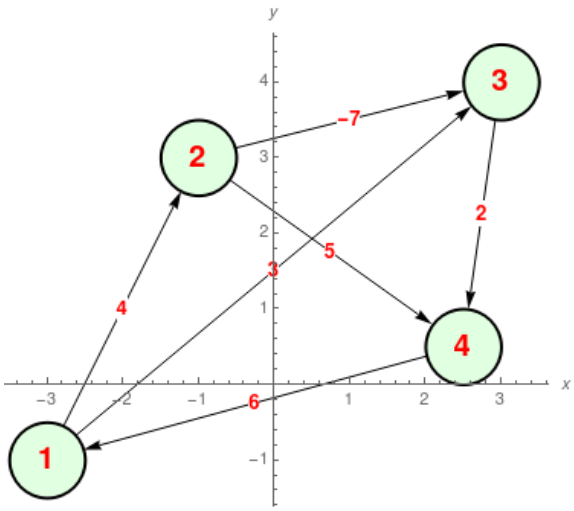
8. Примеры решения модельных задач

Приведенные ниже примеры позволяют сравнить работу двух алгоритмов на тестах с различными входными данными. Для наглядности для каждого теста были приведены иллюстрации, построенные с помощью средств визуализации Wolfram Mathematica.

Пример 1

Данный тест (табл. 1) показывает результаты применения алгоритмов к тесту, не содержащему «исключительные ситуации» (граф не содержит отрицательных циклов, все пути могут быть найдены, данные введены корректно, ответ можно дать однозначно).

Таблица 1. Пример решения

Содержание файла ввода	Содержание файла вывода		Визуализация графа
	Метод перебора	Алгоритм Беллмана - Форда	
4 -2 -2 -3 2 5 6 2 -4 6 1 2 4 1 3 3 2 3 -7 2 4 5 3 4 2 4 1 6 3 1 2 1 3 1 4	3 2 1 2 3 1 2 3 4 1 2 3 4	3 2 1 2 3 1 2 3 4 1 2 3 4	

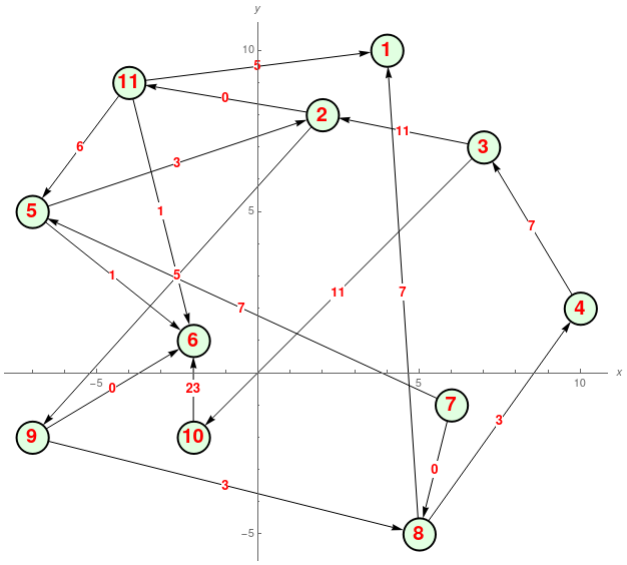
Вывод: файлы вывода для двух алгоритмов совпадают, минимальные пути найдены правильно.

Пример 2

Тест в этом примере (табл. 2) содержит большее количество узлов и

ребер. Он позволяет проверить работу алгоритмов на паре узлов, для которых не найдется ни одного пути.

Таблица 2. Пример решения

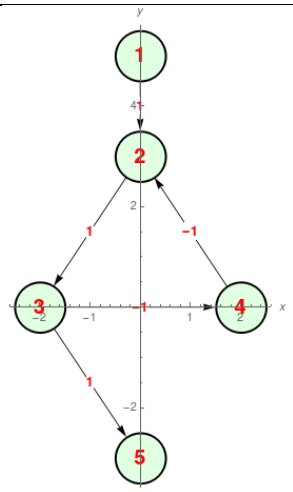
Содержание файла ввода	Содержание файла вывода		Визуализация графа
	Метод перебора	Алгоритм Беллмана - Форда	
11 4 10 2 8 7 7 10 2 -7 5 -2 1 6 -1 5 -5 -7 -2 -2 -2 -4 9 17 11 1 5 11 5 6 11 6 1 2 11 0 2 9 5 3 2 11 3 10 11 4 3 7 5 2 3 5 6 1 7 5 7 7 8 0 8 1 7 8 4 3 9 6 0 9 8 3 10 6 23 3 7 6 8 10 10 11	3 3 7 5 6 4 8 4 3 10 No way	3 3 7 5 6 4 8 4 3 10 No way	

Вывод: файлы вывода для двух алгоритмов совпадают, минимальные пути найдены правильно; в случае пары, для которой нет ни одного пути, выводится сообщение «No way».

Пример 3

Данный тест (табл. 3) содержит цикл отрицательного веса. С помощью этого теста можно проверить поведение алгоритмов при входных данных, для которых решение задачи не может быть найдено.

Таблица 3. Пример решения

Содержание файла ввода	Содержание файла вывода		Визуализация графа
	Метод перебора	Алгоритм Беллмана - Форда	
4 0 5 0 3 -2 0 2 0 0 -3 5 1 2 1 2 3 1 3 4 -1 4 2 -1 3 5 1 1 1 5	1 4 1 2 3 5	1 Negative cycle: 3 4 2 3	

Вывод:

- оба алгоритма завершили работу без аварийных остановок;
- метод перебора не обнаружил цикл, как и предполагалось; путь, найденный методом перебора, можно считать корректным только при условии, что через один узел нельзя проходить дважды;
- алгоритм Беллмана - Форда обнаружил цикл отрицательного веса и вывел узлы, в него входящие.

Сравним время выполнения программ, реализованных на языке C++, для тестов с разным количеством узлов. Из табл. 4 видно, что алгоритм Беллмана - Форда позволяет обрабатывать графы с большим количеством узлов за короткое время.

Табл. 4. Сравнение времени выполнения

Кол-во узлов	Время работы алгоритма (в секундах)		
	Перебор перестановок	Перебор с условиями	Алгоритм Беллмана - Форда
4	1.8e-5 – 3e-5	6.3e-5 – 8.4e-5	3e-6 – 5e-6
10	0.56612	0.00028	1.6e-05
20	-	1.3e-5 – 0.00083	5e-6 – 3.9e-5
124	-	-	0.8e-05 – 0.00011

Заключение

Результатом выполнения практического задания стало знакомство с теорией графов, а конкретно с задачей поиска минимального пути. Для реализации метода перебора и алгоритма Беллмана - Форда были использованы сначала язык программирования C++, а после система компьютерной алгебры Wolfram Mathematica.

Разобранный алгоритм Беллмана - Форда имеет более чем полувековую историю и до сих пор находит широкое применение в информационных технологиях. Благодаря этому практическому заданию были освоены способы реализации этого алгоритма на практике, а также выявлены их преимущества и недостатки.

Список использованных источников

1. Дьяконов В.П. Mathematica 5/6/7. Полное руководство. М.: ДКМ Пресс, 2010. 624с.

2. Доля П.Г. Mathematica для математиков. Часть 2. Графические возможности системы. Харьков: Харьковский Национальный Университет механико-математический факультет, 2015г. 105с.

3. Задача о кратчайшем пути // Википедия. Свободная энциклопедия.
URL: https://ru.wikipedia.org/wiki/ Задача_о_кратчайшем_пути
(дата обращения: 08.12.2019).

4. Задача о нахождении кратчайших путей в графе и методы ее решения// Материалы для обучения lektsii.org.
URL: https://lektsii.org/14-78801.html
(дата обращения: 07.12.2019).

5. Алгоритмы на графах // Информационно-методический ресурс inf-w.ru.
URL: http://inf-w.ru/?page_id=7359
(дата обращения: 25.11.2019).

6. Алгоритм Форда-Беллмана // Образовательный сайт e-maxx.ru.
URL: http://e-maxx.ru/algo/ford_bellman
(дата обращения: 25.11.2019).

7. Bellman-Ford in 5 minutes — Step by step example // Платформа YouTube.
URL: https://www.youtube.com/watch?time_continue=1&v=obWXjtg0L64&feature=emb_logo
(дата обращения: 07.12.2019).

8. Структуры данных для хранения графов: обзор существующих и две «почти новых» // Коллективный блог Хабр.
URL: https://habr.com/ru/post/469967/https://ru.wikipedia.org/wiki
(дата обращения: 23.11.2019).