МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт Математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль)/специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Сравнительный анализ алгоритмов поиска максимального потока»

Студент М.М. Баум
(И.О. Фамилия) (личная подпись)

Руководитель М.А. Тренина
(ученая степень, звание, И.О. Фамилия)

Консультант М.В. Дайнеко
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

Аннотация

Темой бакалаврской работы является «Сравнительный анализ алгоритмов поиска максимального потока». Методы решения задачи о нахождении максимального потока актуальны в современной жизни, при планировании и создании новых транспортных сетей. Решив данную задачу можно получить информацию о наиболее рациональных способах перемещения произвольного объекта из исходной позиции в точку назначения.

Объектом исследования являются алгоритмы решения задачи о максимальном потоке. В качестве предмета исследования выступает сравнительный анализ алгоритмов решения задачи о максимальном потоке реализованных на языке программирования Python.

Целью данной работы является разработка и программная реализация алгоритмов решения задачи о максимальном потоке, а также выявление их преимуществ и недостатков.

Для достижения поставленной цели необходимо решить ряд задач:

1. Изучить область объекта исследования;
2. Исследовать алгоритмы для решения задачи максимального потока;
3. Реализовать исследованные алгоритмы на языке Python.
4. Провести анализ результатов;
5. Сделать вывод о проведенной работе.

Первая глава посвящена основным теоретическим сведениям о предметной области. Во второй главе находится описание алгоритмов для решения задачи о максимальном потоке. В третьей главе приводится программная реализация и сравнение полученных результатов.

Бакалаврская работа состоит из введения, 3 глав, заключения и списка использованной литературы с 21 источником. Текст работы представлен на 43 страницах, который содержит 11 иллюстраций, 13 формул, 2 таблицы и 3 приложения.

Abstract

The topic of the graduation work is *Comparative analysis of the algorithms for solving the maximum flow problem*. The methods to solve the maximum flow problem are relevant in modern life when planning and creating new transport networks. By solving this problem people can obtain some information about the most rational ways for an arbitrary object to move from the starting position to the endpoint.

The graduation work consists of an introduction, 3 chapters, a conclusion, 11 figures, 13 formulas, 2 tables, the list of 21 references and 3 appendices.

The object of the research is the algorithms for solving the maximum flow problem.

The subject of research is a comparative analysis of the algorithms for solving the maximum flow problem implemented in Python programming language.

The purpose of this work is to develop algorithms and provide their software implementation to solve the maximum flow problem as well as to identify their advantages and disadvantages.

To achieve this purpose, it is necessary to solve a number of tasks:

1. to study the research object area;
2. to explore the algorithms used to solve the maximum flow problem;
3. to implement the algorithms in Python under consideration.
4. to analyze the results obtained;
5. to draw up a conclusion about the work done.

The first chapter of the investigation examines the basic theoretical foundations related to the subject area.

The second chapter of the research describes the algorithms used to solve the maximum flow problem.

The third chapter of the graduation work ensures software implementation and compares the results obtained.

Содержание

55

Введение

В повседневной жизни нас окружает множество различных типов сетей, включая электрические, телефонные, кабельные, автомобильные, железнодорожные, производственные и компьютерные. Сеть состоит из узлов и соединяющих их звеньев, по которым движется определенный объект. Процесс происходит в условиях определенных ограничений на каждом участке. Для достижения наиболее эффективного использования подобных сетей, необходимо решить задачу нахождения максимального потока, что позволит с максимально возможной нагрузкой эксплуатировать сеть и следовательно, получать максимально возможный результат.

Задача нахождения максимального потока в транспортной сети не теряет своей актуальности в современном мире. При проектировании сети в реальном пространстве, необходимо проведение анализа свойств будущего объекта.

В качестве объекта исследования выступает задача о поиске максимального потока. Предметом исследования являются алгоритмы для нахождения максимального потока.

Целью данной работы является разработка и программная реализация алгоритмов решения задачи о максимальном потоке на языке Python и дополнительной библиотеки pydot. А также проведение сравнительного анализа на основе полученных результатов.

Для достижения поставленной цели необходимо решить ряд задач:

1. Изучить область объекта исследования;
2. Исследовать основные алгоритмы для решения задачи о максимальном потоке;
3. Реализовать исследованные алгоритмы на языке Python;
4. Провести анализ результатов;
5. Сделать вывод о проведенной работе.

1 Исследование предметной области

1.1 Граф

Термин граф, подразумевает наличие наглядной графической интерпретации рассматриваемого объекта.

Графом называется система произвольных объектов (вершин) и их связей (ребра), соединяющих некоторые пары объектов этой системы.

Непустой, но конечный набор вершин (или узлов) вместе с набором ребер, которые соединяют пары различных вершин. Если ребро e соединяет вершины v_1 и v_2, то говорят, что v_1 и v_2 инцидентны e, а вершины – смежными; e – неупорядоченная пара (v_1, v_2).

Граф обычно изображается в графической форме, в которой вершины отображаются в виде точек или других форм, возможно, помеченных для целей идентификации, а края показаны в виде линий, соединяющих соответствующие точки. Если направление добавлено к каждому ребру графа, получается ориентированный граф или орграф. Затем ребра образуют конечный набор упорядоченных пар различных вершин, которые часто называют дугами. В графическом представлении стрелки могут быть размещены на каждом краю. Без указания направления график называется неориентированным.

Хотя эти изображения полезны визуально, они не подходят для компьютерных манипуляций. Более полезные представления используют матрицу инцидентности или матрицу смежности.

Графы используются в вычислениях самыми разными способами: вершины обычно представляют объекты определенного типа, а ребра представляют связи физического или логического характера между вершинами. Таким образом, графы могут использоваться для математического моделирования таких разнообразных объектов, как

компьютер и все подключенные к нему периферийные устройства, сеть компьютеров, деревья синтаксического анализа, логические зависимости между подпрограммами или нетерминальными символами в грамматике, диаграммы, связанные элементы в базах данных и реакционные сети.

Звеном называется ребро графа, у которого есть как минимум две вершины соединенные ребром.

Вершины, прилегающие к одному и тому же ребру, называются смежными. Если ребра имеют направление, то их называют дугами. Граф с такими ребрами называют ориентированным. Если ориентированное ребро начинается и завершается в одной и той же вершине, то это ребро называют петлей.

Кратными ребрами называют такие ребра, у которых совпадают вершины начала и конца. Граф без кратных ребер и петель называется простым.

Степенью вершины называют количество ребер, которым принадлежит эта вершина. Если из вершины не исходит ни одно ребро, то ее называют изолированной. Висячая вершина, это вершина из которой выходит ровно одно ребро.

Путь в графе G = (V, E) из вершины v_s в вершину v_t называют последовательностью вершин и ребер этого графа в которой каждая, кроме последней, вершина соединена с соседней. Если ни одна из вершин не появляется больше одного раза, то такой путь называют простым.

Циклом называют путь, в котором начальная вершина совпадает с конечной. В простом цикле это условие не выполняется для других вершин.

Маршрут – это чередующаяся последовательность вершин и ребер, в которой любые два соседних элемента связаны ребром (инцидентны).

Цепью называют маршрут, все ребра которого различны. Если в маршруте все вершины, кроме крайних различны, то это простая цепь.

Связные вершины, это две вершины графа, которые соединены простой цепью. Граф, у которого все вершины соединены простой цепью, называется связным. Сильно-связный граф – это ориентированный граф, у которого между любыми двумя вершинами существует, соединяющий их, ориентированный путь.

Графы представляют собой одну из самых часто встречающихся комбинаторных моделей. Они возникают везде, где у нас есть какие-то соотношения между парами объектов. С другой стороны, у графов есть нетривиальные общие свойства, которые таким образом, оказываются полезными в самых разных практических ситуациях.

Широким применением графов в информационных технологиях обусловлено добавлением к вышеизложенным определениям понятия графа как структуры данных. В компьютерной науке граф определяется как нелинейная структура данных. Линейные структуры данных характеризуются тем, что связывают элементы отношениями соседних объектов. Линейными структурами являются массивы, таблицы, очереди, стеки, строки и списки. Противоположные нелинейные структуры могут быть представлены в виде структур данных с иерархической системой.

1.2 Сеть и поток

В теории графов, сеть – это граф $G = (V, E)$, где V – множество вершин, а E – множество ребер V - подмножество $V \times V$ - вместе с неотрицательной функцией $c: V \times V \to \mathbb{R}$, называемая функцией емкости. Без ограничения общности можно считать, что если $(u, v) \in E$, то (v, u) также является членом E, поскольку если $(v, u) \notin E$, то мы можем добавить (v, u) к E а затем положим $c(v, u) = 0$.

Основные свойства сети:

– имеется только одна вершина, в которую не заходит ни одна дуга, называемая истоком (v_s – source)

– имеется только одна вершина, из которой не выходит ни одна дуга, называемая стоком (v_t – sink)

– каждой дуге e_i принадлежит числовая характеристика $C(e) \geq 0$, называемая пропускной способностью

Поток в сети – это ориентированный граф, в котором каждое ребро имеет пропускную способность и поток. Обычно они используются для моделирования проблем, связанных с транспортировкой предметов между местоположениями, с использованием сети маршрутов с ограниченной пропускной способностью. Примеры включают моделирование трафика в сети дорог, жидкости в сети трубопроводов и электричества в сети компонентов цепи.

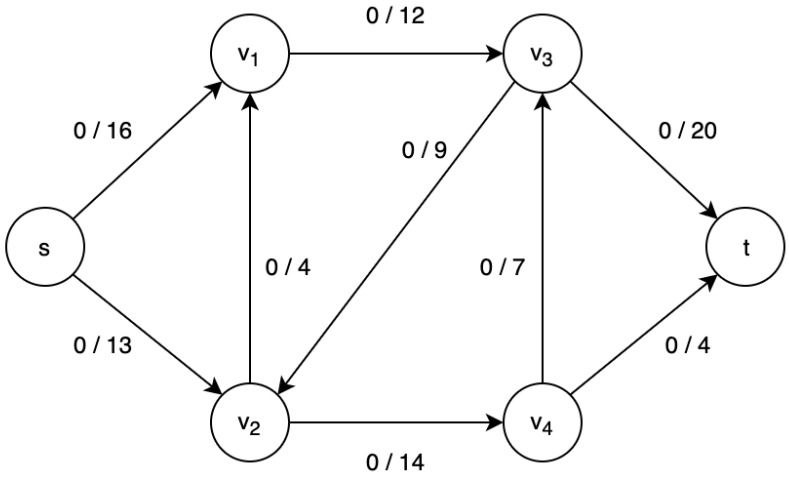

Рисунок 1 – Сеть с нулевым потоком

1.3 Остаточный граф

Интуитивно понятно, что для потоковой сети и потока, остаточная сеть состоит из ребер, в которых можно увеличивать величину потока. Более формально, пусть G = (V, E) потоковая сеть с источником s и стоком t. Пусть

f – поток в сети G. Рассмотрим пару вершин u, v ϵ V. Величина потока, которую можно добавить к текущему потоку на участке от вершины u до вершины v, до достижения максимального ограничения установленной пропускной способностью c(u, v), называется остаточной пропускной способностью ребра (u, v), которая задана следующей формулой:

$$c_f(u, v) = c(u, v) - f(u, v) \quad (1)$$

Например, если в заданной транспортной сети на ребре (u, v) задана пропускная способность c(u, v) = 16 и по нему мы пропустили поток f(u,v)=11, то это означает, что мы еще можем пропустить поток ≤ 5 единицам, прежде чем превысим ограничение пропускной способности на ребре (u, v). В случае, когда по сети пропускают поток с отрицательным значением, остаточная пропускная способность увеличивает свой запас. Например, по ребру (u, v) с начальной пропускной способностью c(u, v) = 16, пропускают поток величиной f(u, v) = –4, то величина его остаточной пропускной способности становится выше начальной и принимает значение в 20 единиц. Ситуацию можно описать следующим образом. От узла v в узел u идет поток f равный 4 единицам, который можно отменить, изменив направление потока. После пустить поток, с той же величиной, из узла u в узел v. Затем увеличить его еще на 16 единиц, прежде чем достигнуть максимально допустимую величину потока на этом участке. В итоге с начальной пропускной способностью в 16 единиц, мы получили 20 единиц потока на участке, начиная увеличивать поток с момента f(u, v) = –4.

Дана потоковая сеть G = (V, E) и поток f. Остаточная сеть G_f, которая индуцирована потоком f, тогда G_f = (V, E_f), где

$$E_f = \{(u, v) \in VxV : c_f(u, v) > 0\} \quad (2)$$

Как было написано выше, каждое ребро в остаточной сети может допускать поток со значением отличным от нуля.

Ребра во множестве E_f являются ребрами в E, либо их противоположностями. Если $f(u,v) < c(u,v)$ для ребра $(u,v) \in E$, то остаточная пропускная способность $c_f(u,v) = c(u,v) - f(u,v) > 0$ и $(u,v) \in E_f$. Если $f(u,v) > 0$ для ребра $(u,v) \in E$, то $f(v,u) < 0$. В этом случае $c_f(v,u) = c(v,u) - f(v,u) > 0$поэтому $(v,u) \in E_f$. Если ни (u, v), ни (v, u) не появляются в исходной сети, то $c(u,v) = c(v,u) = 0$, $f(u,v) = f(v,u) = 0$и $c_f(u,v) = c_f(v,u) = 0$.

Можно сделать вывод, что ребро (u, v) может появится в остаточной сети, только если хотя бы одно из ребер (u, v) и (v, u) появляется в исходной сети и следовательно, $|E_f| \leq 2|E|$. Заметим, что сама остаточная сеть Gf является потоковой сетью с пропускной способностью, заданной потоком f.

1.4 Увеличивающий путь

Дана потоковая сеть G = (V, E) и поток f, увеличивающий путь p – это простой путь из истока s в сток t в остаточной сети G_f. По определению остаточной сети, каждое ребро (u, v) в увеличивающем пути, допускает некоторый дополнительный поток от u к v, с положительной величиной и без превышения ограничения пропускной способности на ребре. Рассматривая остаточную сеть G_f, как потоковую сеть, можно увеличить поток через каждое ребро этого пути до 4 единиц без нарушения ограничения пропускной способности, т. к. наименьшая остаточная пропускная способность на этом пути равна $c_f(v_2, v_3) = 4$. Максимальной величиной, называют величину, с помощью которой можно увеличить поток на каждом ребре увеличивающего пути p в пределах остаточной пропускной способности, задаваемой формулой

$$c_f(p) = min\{c_f(u,v): (u,v) isonp\} \quad (3)$$

1.5 Разрыв сети

Разрезом сети называется множество, которому принадлежит исток и не принадлежит сток. Другими словами, разрез – это минимальное множество дуг, удаление которых «разрывает» все пути, которые соединяют исток и сток. Рассмотрим математическое определение.

Разрезом (S, T) транспортной сети G = (V, E) называется разбиение множества вершин V на множестве S и T = V — S, такие, что s ϵ S, а t ϵ T. Если f – поток, то чистый поток f(S, T) через разрез (S, T) определяется как

$$f(S,T) = \sum_{u\in S}\sum_{v\in T} f(u,v) - \sum_{u\in S}\sum_{v\in T} f(v,u) \quad (4)$$

Пропускной способностью разреза (S, T) является

$$c(S,T) = \sum_{u\in S}\sum_{v\in T} c(u,v) \quad (5)$$

Минимальным разрезом сети является разрез, пропускная способность которого среди всех разрезов сети минимальна.

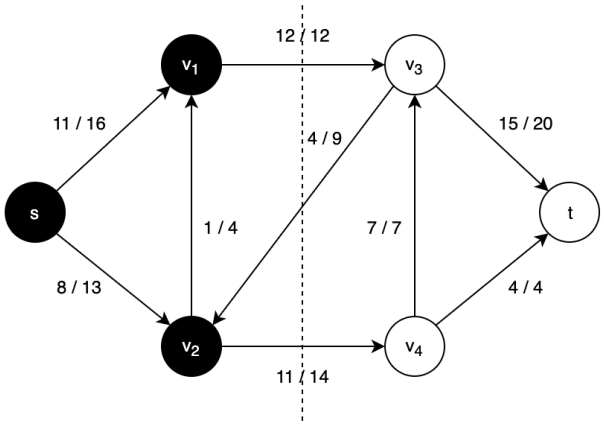

Рисунок 2 – Разрез сети

На рисунке 2 находится пример разреза в сети ({s, v_1, v_2},{v_3, v_4, t}). Для разреза (S, T) в сети, где S = {s, v_1, v_2}, а T = {v_3, v_4, t} чистый поток равен f(S, T) = f(v_1, v_3) + f(v_2, v_4) — f(v_3, v_2) = 12 + 11 — 4 = 9, а пропускная способность составляет c(S, T) = c(v_1, v_3) + c(v_2, v_4) = 12 + 14 = 26.

Величина потока может иметь отрицательное значение, а величина пропускной способности может быть только нулевой или положительной. Другими словами, величина потока через разрез (S, T) состоит из положительных потоков в обоих направлениях. Положительный от S к T прибавляется, а положительный поток от T к S вычитается. С другой стороны, пропускная способность разреза (S, T) вычисляется только по ребрам, идущим от S к T. Ребра, идущие от T к S, не включаются в вычисление c (S, T). Величина максимального потока сети ограничена величиной минимального разреза сети.

Для заданного потока f в сети G с источником s и стоком t справедливо утверждение, что чистый поток через любой разрез одинаков и равен величине потока. Величина любого потока ограничена сверху пропускной способностью произвольного разреза.

1.6 Задача о максимальном потоке

Задача о нахождении максимального потока является классической задачей оптимизации. Подобная задача возникает во многих областях исследования операций, где присутствует модель ориентированного графа с пропускными способностями. Алгоритмы для решения подобной задачи изучаются уже довольно долго и в последнее время эффективность их работы была значительно улучшена.

Транспортная потоковая сеть может быть представлена в виде ориентированного графа. Представьте маршрут произведенного товара от

места производства до места конечного потребления. Источник производит товар с определенной скоростью и с той же скоростью его потребляет сток. Величина потока на разных участках сети, это скорость, с которой движется товар на данном участке. Транспортные потоковые сети полезны, при моделировании ситуации движения воды в водопроводной сети, перемещения детали на сборочном производстве, хождения тока в электросети, движения сообщений в телекоммуникационной сети и во многих других подобных сетях.

Каждое ребро в потоковой сети можно рассматривать как канал передачи объекта. Каждый канал имеет свою пропускную способность. Например, 200 галлонов жидкости в час через трубопровод или напряжение 20 ампер для электрического тока, проходящего через кабель. Вершины графа можно рассматривать, как узлы соединения каналов, через которые проходит поток из источника в сток. Согласно свойству сохранения потока, скорость потока при входе и выходе из вершины должна совпадать.

В задаче о максимальном потоке, нужно найти максимальную скорость доставки объекта из источника в сток через заданную сеть узлов, при этом не нарушая условий ограничения ни на одном из проходимых участков. Подобная проблема является простейшей задачей потоковых сетей, и она может быть успешно решена с помощью применения некоторых алгоритмов. Более того, данные алгоритмы могут быть использованы и для других задач, возникающих при работе с потоковыми сетями.

Пусть граф G = (N, A) – потоковая сеть, где s – источник, а t – сток и пропускная способность c(u, v) с ребром (u, v) ϵ E. Свойства функции потока в сети графа G:

1. Свойство ограничения пропускной способности. Ни на одном ребре поток не может превысить пропускную способность

$$f(u, v) \leq c(u, v) \quad (6)$$

2. Свойство косой симметрии. Поток из u в v должен быть противоположным потоку из v в u

$$f(u, v) = -f(v, u) \quad (7)$$

3. Свойство сохранения потока. Для всех u, v ϵ V, кроме источника и стока

$$\sum_{u,v \in V} f(u, v) = 0 \quad (8)$$

Неотрицательное число f(u, v) называют величиной потока из вершины u в вершину v. Величина потока определяется по формуле

$$|f| = \sum_{v \in V} f(s, v) \quad (9)$$

Решением задачи поиска максимального потока является нахождение максимально возможной величины максимального потока в заданной транспортной сети.

1.7 Теорема Форда-Фалкерсона

Другое название теоремы, теорема о максимальном потоке и минимальном разрезе.

Это теорема о сетевом потоке. Главное утверждение теоремы в том, что максимальный поток через любую сеть от источника к стоку является точной суммой реберных весов, которые при удалении могут стать причиной разрыва соединения источника от стока. Другими словами, для любого графа сети и выбранного источника со стоком, величина максимального потока равна величине потока минимального разреза. Концепт потока может быть

использован в различных сферах деятельности. Например, он может выражать объем воды в водопроводной сети или объем информации, который может проходить по кабелям компьютерной сети.

Пусть граф G = (N, A) – потоковая сеть, где s – источник и t – сток. Все ребра имеют пропускную способность величиной ≥ 0. Ниже два главных определения:

– Возможный поток f в графе G – это присвоение положительных значений к каждому ребру, каждое значение не превышает ограничение, представленное в виде пропускной способности. Кроме того, величина потока, входящего в вершину v и равна величине потоку выходящего из нее, при условии, если эта вершина не является источником или стоком v ≠ s, t. Величина потока соответствует потоку, выходящего из источника.

– Разрез сети G – это результат разбиения множества графа G на два непересекающихся подмножества S и T, таких, что s ϵ S, t ϵ T. Пропускной способностью в данном разрезе, является сумма всех реберных весов из этих подмножествах.

Теорема. Максимальное значение допустимого потока сети G эквивалентно значению минимальной пропускной способности разреза сети G. Более того, если пропускная способность сети G является целочисленной, то максимальный поток в этой сети тоже имеет целочисленное значение.

$$ƒ_{max} = c_f(p) \in G \qquad (10)$$

Доказательство. Начнем с введения произвольного потока в сети. После определим процесс добавления увеличивающих путей, который по завершению приведет к результату в виде получения максимального потока. В заключении, покажем, как этот процесс описывают разрез сети с минимальной пропускной способностью в пути от источника к стоку и что

величины максимального потока и минимальной пропускной способности в пути одинаковы.

Начинаем с любого потока f. Поскольку нулевой поток, это поток, в котором каждое ребро имеет нулевое значение, действителен для любой сети, такой поток должен существовать. Определим ориентированный граф D_f на множестве вершин V со множеством ребер $E = \{\bar{uv} \vee c_f(u,v) > 0\}$. Предположим, что существует путь p из источника s в сток t внутри множества D_f. В графе G, p – это путь, по которому каждое ребро может проводить больше потока, значит p – это увеличивающий путь. Пусть m – это минимальное значение остаточной пропускной способности $c_f(p)$ ориентированных ребер из источника s к стоку t и установим поток $\{f(u,v) \vee \bar{uv}\}$ в обратно ориентированном ребре из источника s к стоку t. В построенном Df минимальное значение пропускной способности должно быть m > 0. Увеличение потока каждого ориентированного ребра на m единиц и уменьшение потока каждого обратно ориентированного ребра на m единиц, это сохраняет неотрицательность величин потоков и ограничение пропускной способности каждого ребра сети. Это увеличение также увеличивает поток из источника s до стока t на положительную величину единиц m.

Обозначим увеличенный поток f'. Поскольку p не является увеличивающим путем в f', увеличение потока на m единиц приводит, либо к тому, что одно из ориентированных вперед ребер достигает максимальную величину пропускной способности или одно из обратно ориентированных ребер сравнивается по значению с нулем.

Выше описанные действия являются итеративными до тех пор, пока в сети не останется увеличивающих путей. Поскольку количество полных путей от s до t конечно в любом конечном графе, поэтому задача будет завершена за конечное количество шагов. В получившемся ориентированном графе G_f обозначим множество вершин, достижимых из источника s

множества R и множество недостижимых вершин из s множества U. Очевидно, что источник s находится во множестве R и во множестве D_f нет путей из s в t, т. к. сток t находится во множестве U. Для каждого ребра из множества вершин R до любой вершины из множества U, каждое ориентированное ребро должно достигнуть максимально возможную величину потока и каждое обратно-направленное ребро должно иметь величину потока равно нулю. Таким образом, увеличивающий поток f' становится максимальным потоком, а разрез сети (R, U) минимальным разрезом. Более того, полученный поток f' совпадает по значению пропускной способности разреза (R, U), что доказывает теорему.

1.6 Выводы по главе

Таким образом, была изучена и описана теоретическая информация, необходимая для выполнения программной реализации алгоритмов поиска максимального потока в транспортной сети. Был полностью рассмотрен граф и его элементы. Исследовано понятие транспортной сети и ее основные свойства. Разобрано понятие разреза сети. Проанализирована теорема Форда-Фалкерсона.

Задача о максимальном потоке в сети можно представить в виде транспортной сети, главной целью которой является снабжение определенного конечного узла. Это сеть узлов, соединенных каналами связи. Один из узлов сети выступает в роли источника и еще один узел представляет собой сток, конечную точку всех путей в сети. Задача о максимальном потоке заключается в определении количества объекта, которые могут пройти по всем возможным путям в сети, из источника в сток.

В следующем разделе будут описаны некоторые алгоритмы для нахождения максимального потока в сетях ориентированного графа. Для выполнение программной реализации будет использоваться язык Python.

2 Программная реализация алгоритмов

2.1 Алгоритм Форда-Фалкерсона

В каждой итерации метода Форда-Фалкерсона, мы находим увеличивающий путь p и увеличиваем поток f на каждом ребре в p, каждый из которых ограничен остаточной пропускной способностью f(p). Данная реализация метода вычисляет максимальный поток в сети графа G = (V, E), обновляя величину потока f(u, v) между каждой парой вершин u и v, соединенных ребром. Если u и v не соединены ребром ни в одном направлении, то мы неявно предполагаем, что f(u, v) = 0. Предполагается, что пропускные способности c(u, v) заданы в графе сети и первоначальные значения потоков, проходящих через ребра сети, равны нулю. Пропускная способность равна нулю $c(u, v) = 0$, при условии отсутствия соединения между вершинами $(u, v)E$. Остаточная пропускная способность рассчитывается по формуле (1). Выражение c_f(p), в коде представляет собой обыкновенную переменную, в которой содержится остаточная пропускная способность текущего пути p.

Увеличивающей цепью является цепь из источника в сток, все дуги которой допустимы. Дугу из вершины источника v_s в вершину стока v_t назовем допустимой, если выполняется одно из следующих условий:

1. f(e) < c(e) и дуга согласованна
2. f(e) > 0 и дуга несогласованна

По увеличивающей цепи можно пустить поток величины Q, где Q = $\min\{q(e_i), 1 \le i \le 1\}$ и q(e) = {c(e) – f(e), если дуга согласованна, f(e), если дуга не согласованна}. Для того, чтобы увеличить величину потока сети на Q, необходимо увеличить на Q поток на каждой согласованной дуге цепи и уменьшить на каждой несогласованной.

Смысл алгоритма в следующем. В начале величина потока является нулевой для всех вершин в графе. Затем величина потока увеличивается на каждой итерации посредством поиска увеличивающего пути. Итерации продолжаются до тех пор, пока можно построить увеличивающий путь от источника к стоку.

1 шаг. Установка нулевого потока.

2 шаг. Поиск произвольного пути в остаточной сети. Остановка алгоритма, в случае не нахождения подобного пути.

3 шаг. Увеличение потока на величину, соответствующей максимально возможной минимальной пропускной способностью C_{min} на этом пути.

4 шаг. Одновременно с предыдущим шагом, уменьшение потока на величину C_{min}в противоположном пути.

5 шаг. В текущей остаточной сети, определяем новые пропускные способности для каждого ребра. Исключаем все ребра с нулевым остатком пропускной способности.

6 шаг. Возврат к шагу 2.

На каждом шаге алгоритма действую те же условия, что и для всех потоков:

– Поток из u в v не превосходит пропускную способность

$$f(u,v) \le c(u,v) \tag{11}$$

– Противоположные потоки симметричны

$$f(u,v) = -f(v,u) \tag{12}$$

– Поток не изменяется при прохождении через узел.

$$\sum_y f(u,v) = 0, f\ \ (u) = f_{out}(u) \tag{13}$$

Алгоритм добавляет поток увеличивающего пути на каждом шаге к уже существующей величине потока.

Если пропускные способности всех ребер, это целые числа, то можно доказать, что и потоки через все ребра всегда будут целыми. Значит на каждом шаге алгоритм увеличивает поток по крайней мере на единицу. Это

означает, что он сойдется не более чем за O(f) шагов, где f – максимальный поток в графе. Можно выполнить каждый шаг за время O(E), где E – число ребер в графе, тогда общее время работы алгоритма будет ограничено O(fE).

Если величина пропускной способности хотя бы одного из ребер, это иррациональное число, то алгоритм будет работать бесконечно, даже не обязательно находя правильный ответ.

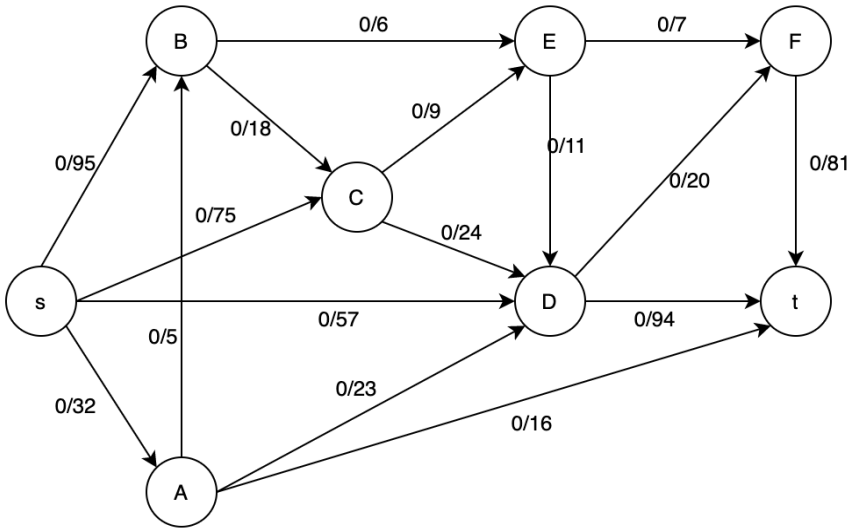

Рисунок 3 – Нулевой поток

Рассмотрим работу алгоритма Форда-Фалкерсона на примере:

Шаг 1. Выбор произвольного пути s → B → E → F → t. Его максимальный поток соответствует минимальной пропускной способности ребра, входящего в этот путь. Ребро (B, E) является узким местом и имеет пропускную способность равную 6. Значит увеличиваем поток на 6. Текущий максимальный поток равен 6. Также подмечаем, что ребро (B, E) можно вычеркнуть из дальнейшего процесса, так как оно достигло своей максимальной пропускной способности. Через это ребро больше не будут проходить удлиняющие пути.

Шаг 2. Выбираем следующий произвольный путь s → C → D → t. Узким место является ребро (C, D). Его пропускная способность равна 24, увеличиваем текущий поток на 24. Максимальный поток составляет 30.

Ребро (C, D) достигло максимального значения и больше не будет использоваться.

Шаг 3. Следующий путь s → D → t. Пропускная способность составляет 57. Текущий максимальный поток равне 87. Вычеркиваем ребро (s, D)/

Шаг 4. Новый произвольный путь s → A → t. Пропускная способность = 16, значит новое значение максимального потока = 103.

Шаг 5. Следующий путь s → A → D → t. На ребро (s, A) уже произошло изменение пропускной способности на прошлой итерации. Теперь оно равно не 32, а 16. Также ребро (D , t) составляет не 94, а 94 — 57 — 24 = 13. Значит это ребро является узким местом. Максимальный поток теперь равен 116. Ребро (D, t) вычеркиваем.

Шаг 6. Путь s → A → D → F → t. Узкое место тут ребро (s, A), так как у него остаток лимита пропускной способности равен 3. Значит увеличиваем поток на 3, теперь он равен 119. Ребро (s, A) достигло максимального значения и больше не будет использоваться.

Шаг 7. Путь s → C → E → F → t. Ребро (E, F) в остатке пропускной способности имеет единицу. Увеличиваем поток на 1. Вычеркиваем это ребро.

Шаг 8. Последний доступный путь s → C → E → D → F — t. Минимальная пропускная способность находится в ребре (C, E) и равна 8. Увеличиваем поток на 8 и получаем максимальный поток в этой сети.

f_{max} = f(s → B → E → F → t) + f(s → C → D → t) + f(s → D → t) + f(s → A → t) + f(s → A → D → t) + f(s → A → D → F → t) + f(s → C → E → F → t) + f(s → C → E → D → F — t) = 6 + 24 + 57 + 16 + 13 + 3 + 1 + 8 = 128

Ниже расположена графическая иллюстрация полученной сети с найденным максимальным потоком.

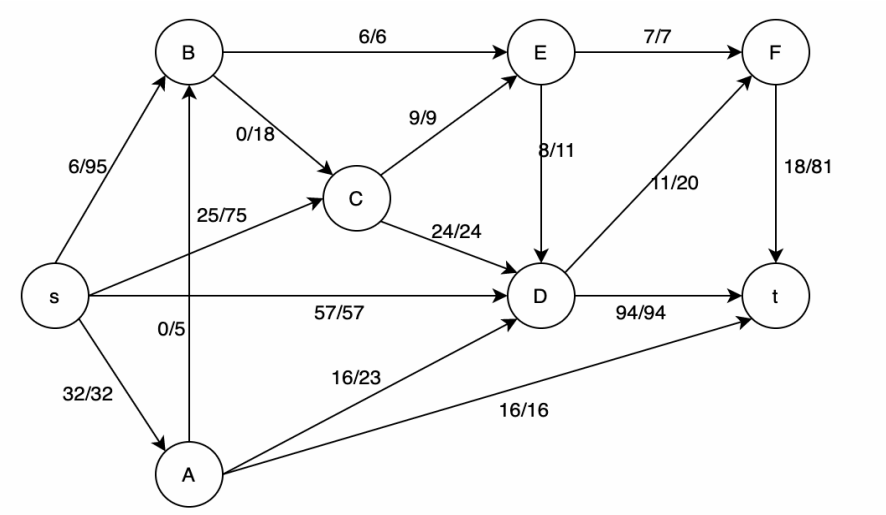

Рисунок 4 – Максимальный поток с величиной 128

Время работы алгоритма Форда-Фулкерсона зависит от того, как быстро определяется увеличивающий путь p. Если он выбран неудачно, то алгоритм может замкнуть и он никогда не завершит свою работу. Значение потока будет увеличиваться бесконечно, не имея никаких ограничений. Если же алгоритм работает, при участии алгоритма поиска в ширину, то он завершит работу за конечное время. Перед доказыванием этого, нужно убедиться, что для каждого произвольно выбранного увеличивающего пути имеются ограничения и все они выражены в целочисленном выражении.

Чаще всего на практике, подобная проблема в поиске максимального потока, возникает при работе с целыми числами. Если величины пропускных способностей выражены рациональными числами, то можно использовать соответствующую шкалу масштабирования для преобразования этих чисел в целый формат. Допуская такое предположение, простая реализация алгоритма Форда-Фулкерсона выполняется за время O(f*E), где f – это величина максимального потока, найденная благодаря алгоритму.

Ford-Fulkerson(G, s, t)

[1] for для каждого ребра $(u, v) \in E[G]$

[2] do $f[u, v] \leftarrow 0$

[3] $f[u,v] \leftarrow 0$

[4] while пока существует путь p из s в t в остаточной сети G_f

[5] do $c_f(p) \leftarrow min\{c_f(u,v): (u,v) \text{впути} p\}$

[6] for для каждого ребра (u,v) в p

[7] do $f[u,v] \leftarrow f[u,v] + c_f(p)$

[8] $f[v,u] \leftarrow -f[u,v]$

Анализ работы алгоритма заключается в следующем: шаги 1 – 3 выполняются за время $O(E)$, в то время как шаги 4 – 8 выполняются в цикле while за время $O(f_{max})$, так как величина потока увеличивается, как минимум на единицу, после каждой итерации.

Работа, выполняемая в цикле while может быть эффективной, при условии, что эффективно организована структура данных в реализуемой сети $G = (V, E)$. Предположим, что мы имеем структуру данных, соответствующую ориентированному графу сети $G' = (V, E')$, где множество $E' = \{(u,v): (u,v) \in E \text{или} (v,u) \in E\}$. Ребра в сети G, также являются ребрами в сети G' и поэтому достаточно просто поддерживать пропускные способности и потоки в этой структуре данных. Для потока f в сети G, ребра остаточной сети G_f состоят из всех ребер (u, v) сети G, таких, что $c(u,v) - f(u,v) \neq 0$. Для поиска пути в остаточной сети происходит за время $O(V + E') = O(E)$, при условии, если используется алгоритм поиска в ширину или глубину. В итоге каждая итерация в цикле while проходит за время $O(E)$, в результате чего общее время работы алгоритма Форда-Фалкерсона составляет $O(f_{max} * E)$.

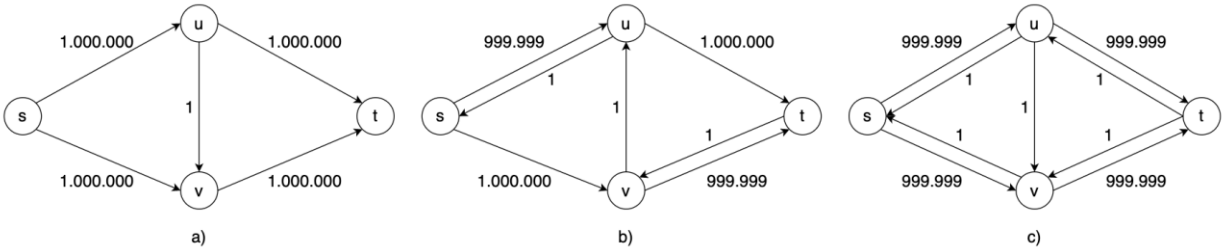

Рисунок 5 – Ограничения в алгоритме Форда-Фалкерсона

При целочисленных величинах пропускных способностей и оптимальном малом значении потока $|f|$, работа алгоритма Форда-Фалкерсона достаточно эффективна. На рисунке 5a изображена сеть для демонстрации сети с очень большой величиной потока $|f|$. Максимальный поток в данной сети составляет значение 2.000.000 единиц. 1.000.000 единиц потока проходят по пути $s \to u \to t$ и еще 1.000.000 единиц потока идут по пути $s \to v \to t$. Если алгоритм Форда-Фалкерсона выберет в качестве первого увеличиваещего пути $s \to u \to v \to t$, как продемонстрировано на рисунке 5a, тогда поток, на первой итерации, примет значение равное 1. На рисунке 5b изображена полученная остаточная сеть. Если на второй итерации будет выбран путь $s \to v \to u \to t$, как показано на рисунке 5b, тогда поток примет значение 2. На рисунке 5c изображена полученная остаточная сеть. Процесс может быть продолжен, на нечетных итерациях будет выбран путь $s \to u \to v \to t$, а на четных $s \to v \to u \to t$. В таком случае будет произведено 2.000.000 итераций, в каждой из которых, поток будет увеличен на 1 единицу.

2.2 Алгоритм Эдмондса-Карпа

Данный алгоритм представляет собой один из вариантов алгоритма Форда-Фалкерсона для поиска максимального потока в сети. Вместо того, чтобы рассматривать произвольный путь как увеличивающий, как это сделано в алгоритме Форда-Фулкерсона, алгоритм Эдмондса-Карпа рассматривает кратчайший путь в качестве увеличивающего пути. Такой путь определяется с помощью использования алгоритма поиска в ширину.

1 шаг. Установка нулевого потока.

2 шаг. Поиск кратчайшего пути в остаточной сети. Остановка алгоритма, в случае не нахождения подобного пути.

3 шаг. Увеличение потока на величину, соответствующей максимально возможной минимальной пропускной способностью C_{min} на этом пути.

4 шаг. Одновременно с предыдущим шагом, уменьшение потока на величину C_{min}в противоположном пути.

5 шаг. В текущей остаточной сети, определяем новые пропускные способности для каждого ребра. Исключаем все ребра с нулевым остатком пропускной способности.

6 шаг. Возврат к шагу 2.

Специализация метода Форда-Фулкерсона на алгоритме Эдмондса-Карпа проста, поскольку поиск кратчайшего пути увеличения – это уточнение поиска любого пути увеличения.

Требуется значительно больше усилий, чтобы показать, что результирующий алгоритм завершается за время $O(V * E)$. Идея доказательства заключается в следующем. Ребра, противоположные ребру кратчайшего пути, не могут лежать на самом кратчайшем пути. При каждом увеличении хотя бы одно ребро остаточного графа, лежащее на кратчайшем пути увеличения меняет свое направление. Таким образом, либо увеличивается длина кратчайшего пути, либо уменьшается количество ребер, лежащих на некотором кратчайшем пути. Поскольку длина кратчайшего пути не превышает V, существует не более $O(V * E)$ итераций.

Демонстрация на примере.

Дана сеть, состоящая из семи вершин, где v_1 является истоком, а v_7 стоком. На рисунке ниже показаны пары f / c, где f – поток ребра, а c – пропускная способность этого ребра.

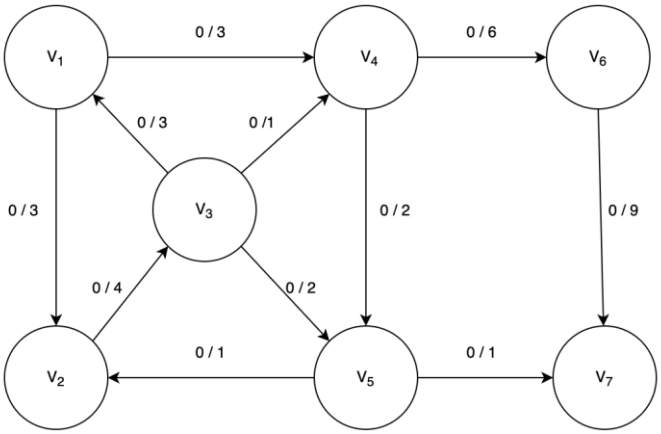

Рисунок 6 – Поток неизвестной величины

1. Цепочка состоит из единственной вершины v_1, которая является посещенной. Предков нет.

2. Цепочка состоит из вершин v_2 и v_4. Посещены вершины v_1, v_2, v_4. Вершины v_2, v_4 имеют предка – вершина v_1.

3. Цепочка состоит из вершин v_4 и v_3. Посещены вершины v_1, v_2, v_3, v_4. Вершины v_2, v_4 имеют предка v_1, вершина v_3 имеет предка v_2.

4. Цепочка состоит из вершин v_3, v_5, v_6. Посещены v_1, v_2, v_3, v_4, v_5, v_6. Вершины v_2, v_4 имеют предка v_1, вершина v_3 имеет предка v_2, вершины v_5, v_6 имеют предка v_4.

5. Вершина v_3 удаляется из цепочки. Из нее ведут ребра только в уже посещенные вершины.

6. При обнаружении ребра (v_5, v_7) цикл останавливается. В цепочке вершины (v_6, v_7). Все вершины посещены. Вершины v_2, v_4 имеют предка v_1, вершина v_3 имеет предка v_2, вершины v_5, v_6 имеют общего предка v_4, вершина v_7 имеет предка v_5.

7. Переходим по предкам: $v_7 \rightarrow v_5 \rightarrow v_4 \rightarrow v_1$. Возвращаем пройденный путь в обратном порядке: $v_1 \rightarrow v_4 \rightarrow v_5 \rightarrow v_7$.

В цепочку последовательно добавляли вершины, достижимые из v_1 ровно за один шаг, два шага и три шага. Кроме того, предок каждой вершины – это вершина, достижимая из v_1 ровно на один шаг быстрее.

2.3 Алгоритм Диница

Алгоритм Диница представляет собой улучшенную версию предыдущего алгоритма, который требует $O(|V|^2|E|)$ операций. Показывает наибольшую эффективность в единичных сетях.

Данный алгоритм состоит из нескольких итераций. На каждой итерации строится остаточная сеть, затем на основе остаточной сети строится слоистая сеть, а в слоистой сети находится блокирующий поток. Найденный блокирующий поток увеличивает ответ на соответствующую величину. Итерации повторяются до тех пор, пока получается находить блокирующий поток.

Главное отличие алгоритма Диница от алгоритма Эдмондса-Карпа заключается в том, что в алгоритме Диница, за одну итерацию находится несколько путей от истока в сток, в отличие от того, что в алгоритме Эдмондса-Карпа находится только один подобный путь.

Шаг 1. Строим минимальную бесконтурную сеть остаточного графа.

Шаг 2. Пока в сети есть путь из v_s в v_t, нужно выполнять следующие шаги:

Шаг 2.1. Находим кратчайший путь из v_s в v_t. Если его нет, выйти из цикла.

Шаг 2.2. На найденном пути в остаточной сети ищем ребро с минимальной пропускной способностью C_{min}.

Шаг 2.3. Для каждого ребра на найденном пути увеличиваем поток на C_{min}, а в противоположном ему – уменьшаем на C_{min}.

Шаг 2.4. Удаляем все ребра, которые достигли насыщения.

Шаг 2.5. Удаляем все тупики (все вершины, кроме стока и источника, из которых не выходят и не входят ребра) вместе со всеми инцидентными им ребрами.

Шаг 2.6. Повторяем предыдущий шаг, пока есть что удалять.

Шаг 3. Если найденный поток ненулевой, добавляем его к общему потоку и возвращаемся на шаг № 1.

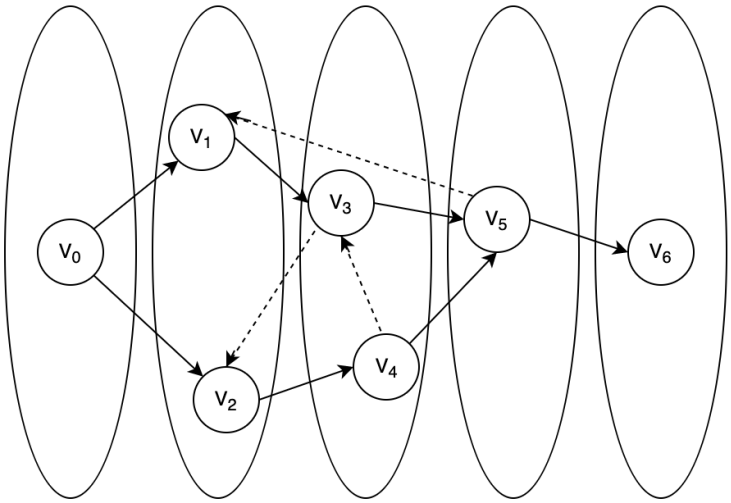

Рисунок 7 – Слоистая сеть

Слоистая сеть – это сеть состоящая только из таких ребре (v_s, v_t) сети G, что $d(v_s) + 1 = d(v_t)$, где d – расстояние от истока.

Пунктирными линиями обозначены ребра, которые не входят в сеть.

2.4 Выводы по главе

В данной главе были описаны все рассматриваемые алгоритмы для решения задачи о поиске максимального потока.

Один из алгоритмов решения подобной задачи, это алгоритм Форда-Фалкерсона. Для построения путей из источника в сток, могут использоваться, как алгоритм поиска в глубину, так и алгоритм поиска в

ширину. Сложность обоих составляет $O(E)$. Если пропускные способности ребер выражены целыми числами, то это означает, что каждую итерацию, величина потока увеличивается не менее, чем на единицу. В остальных случаях сложность алгоритма составит $O(E * F)$. В случае работы с рациональными числами, алгоритм имеет способность завершения. В случае работы с сетью, в которой пропускные способности каналов выражены в иррациональных числах, завершение работы алгоритма не предусмотрено.

Следующий алгоритм, который был рассмотрен, это алгоритм Эдмондса-Карпа. Данный алгоритм является улучшением предыдущего. Все пути он находит с помощью алгоритма поиска в ширину. Сложность составляет $O(V * E^2)$ для всех типов величин в пропускной способности. На каждой итерации, алгоритм находит увеличенный путь и делает его насыщенным. Кроме того, расстояние от источника все время увеличивается на одно ребро.

В алгоритме Диница, каждая итерация состоит из нескольких этапов. На первом этапе строится остаточная сеть. После, с помощью алгоритма поиска в ширину, строится слоистая сеть и в ней происходит поиск блокирующего потока. В завершении итерации происходит суммирование величин текущего потока с величиной блокирующего потока.

Для выполнения программирования методов, описанных в теоретической части работы, был использован язык программирования Python.

3 Тестирование программы и анализ результатов

3.1 Описание программных средств

Для реализации заданной программы был использован язык программирования Python, также его специальная библиотека для отрисовки графов на разном этапе работы в каждом алгоритме для поиска максимального потока.

Библиотека networkx предназначена для создания и управления динамическими сетевыми структурами. Networkx состоит из иерархии пакетов, где верхний уровень каждой представляет собой общие методы управления определенной структуры. В данной был использован пакет для работы с графами. Начальная структура данных представлена в виде матрицы инцидентности, где содержимое каждой ячейки представляет собой значение пропускной способности ребер.

Для визуализации полученных данных были применены элементы библиотеки graphviz (Graph Vizualization Software). Это пакет утилит для автоматической визуализации графов, описанных с помощью кода. Для выполнения работы был использован пакет pydot. В данном пакете находится множество методов для управления состояния структуры созданного графа. Для инициализации графа требуется метод Dot. С помощью метода Node были заданы стили узлов графа. Метод Edge позволил установить стили ребер и зафиксировать формат отображения значений потока и пропускных способностей ребра. Содержимое графа для визуализации можно предоставить несколькими способами. Можно создать внешний текстовый файл и на языке Dot описать структуру желаемого графа, а после с помощью метода graph_from_dot_file загрузить его. Это действие производится вместо инициализации внутри программы. Другой способ создания структуры графа

заключается в том, чтобы вручную описать объект в виде матрицы инцидентности.

3.2 Сравнительный анализ результатов

В алгоритме Форда-Фалкерсона сложность определяется по формуле $O(f_{max} * E)$. На каждой итерации этот алгоритм находит увеличивающий путь (путь от источника к стоку). После этого увеличивает величину потока на максимально допустимую, которая ограничена величиной потока в узком месте. Итерации повторяются до тех пор, пока алгоритм способен найти увеличивающий путь. Этот алгоритм работает только с целочисленными значениями в качестве пропускных способностей. В противном случае он замыкает и бесконечно пытается найти правильный ответ.

Алгоритм Эдмондса-Карпа является улучшенным вариантом предыдущего алгоритма. На каждой итерации, с помощью поиска в ширину, алгоритм ищет кратчайший путь из источника к стоку. Сложность, составляет $O(V * E^2)$.

В остаточной сети для определения расстояния из источника до стока, в алгоритме Диница используется алгоритм поиска в ширину. Далее выполняется построение граф из ребер остаточной сети (между ними растет расстояние на 1). После происходит избавление от тупиковых вершин. На получившемся графе находится кратчайший увеличивающий путь. После этого выполняется операция по исключению ребра с минимальной пропускной способностью из остаточной сети и удаление тупиковых вершин до тех пор, пока есть увеличивающий путь. Сложность определяется по формуле $O(V^2E)$.

Подробное описание работы программы будет продемонстрировано на визуализации состояний графа в алгоритме Форда-Фалкерсона.

В качестве начального условия, дана матрица инцидентности, в ячейках которой зафиксированы значения пропускной способности ребер.

$$G=\begin{bmatrix} 0 & 10 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 10 \\ 0 & 0 & 0 & 6 & 0 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Ниже приведена иллюстрация начального состояния графа.

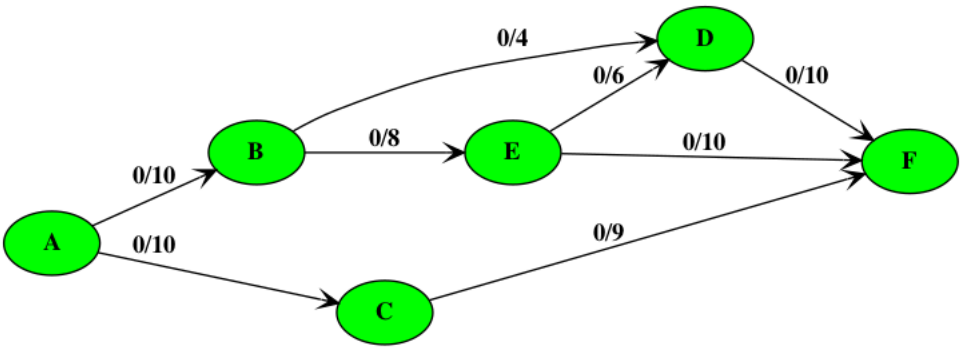

Рисунок 8 – Нулевой поток

После инициализации начального графа прокладывается первый путь из источника к стоку. Величина потока в пути определяется по максимальному значению узкого места, ребра с минимальной пропускной способностью.

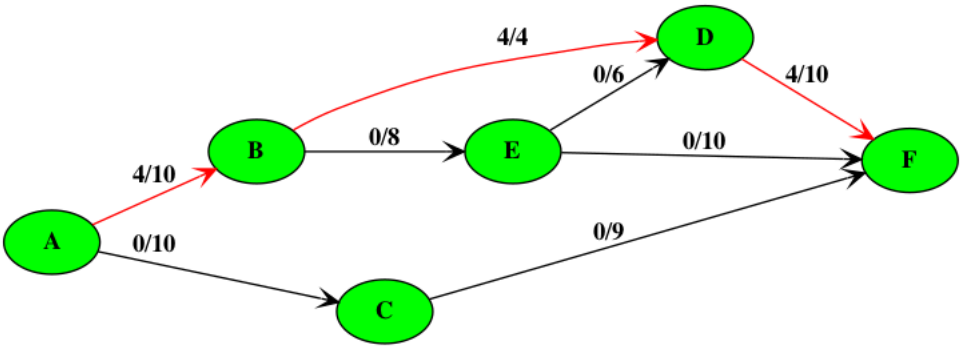

Рисунок 9 – Поток величины 4

На пути из источника A в сток F, через вершины B и D, ребро (B, D) имеет минимальную пропускную способность и является узким местом. Вес этого ребра равен 4, значит путь A → B → D → F будет иметь максимальный поток величиной 4.

После этого подбирается следующий путь.

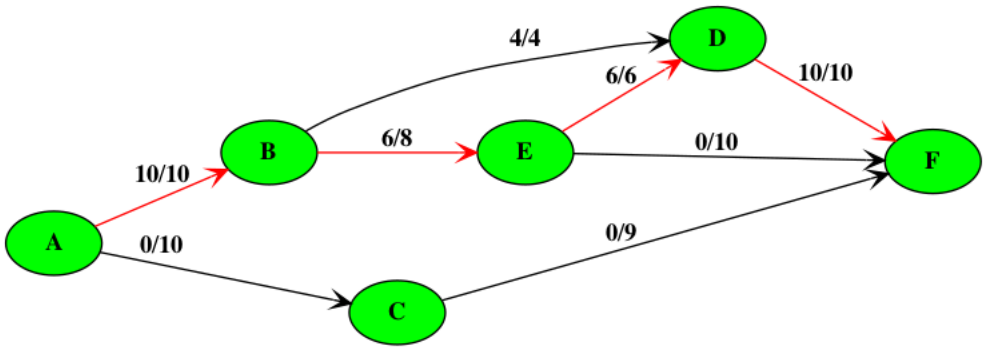

Рисунок 10 – Поток величины 10

На пути A → B → E → D → F находится сразу 3 узких места. Это пути A → B, E → D и D → F. Все они имеют максимальный остаточный запас пропускной способности величиной 6. Данное обстоятельство означает, что можно увеличить максимальный поток на 6 и построить второй путь из источника к стоку.

Ребро (A, B) исчерпало запас пропускной способности, следовательно по нему больше нельзя проходить. Тогда выбираем ребро (A, C). В следующем пути A → C → F слабым место является ребро (C, F) с пропускной способностью 9, поэтому запускаем по нему поток величиной 9.

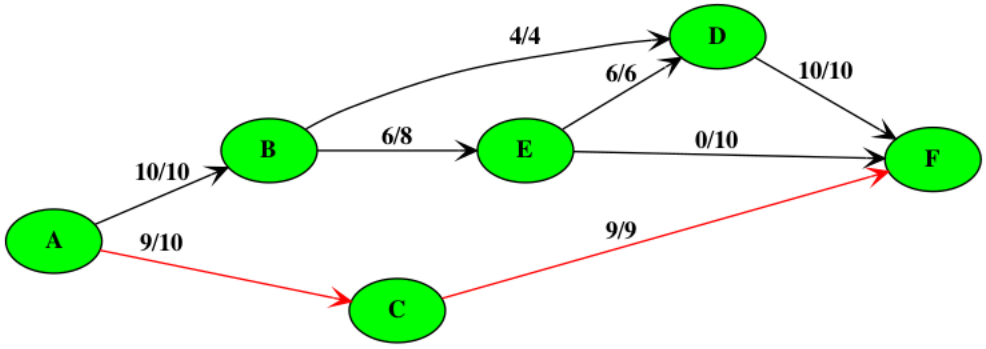

Рисунок 11 – Поток величины 19

Ребро (C, F) достиг максимального значения своей пропускной способности, значит на этом пути больше нет возможности увеличить поток. Максимальное значение достигнуто, хотя ребро (E, F) осталось безучастно.

f_{max}(S, T) = f(A → B → D → F) + f(A → B → E → D → F) + f(A → C → F) = 4 + 6 + 9 = 19.

Использовав условия задачи и полученный результат с величиной максимального потока в задаче из прошлого пункта, вычислим время работы каждого из алгоритмов.

В начальном графе содержится 6 вершин V и 8 ребер E. Также был получен результат со значением максимального потока, который составляет 19. Полученные результаты будут выражены в таблице ниже.

Таблица 1 – Анализ полученных результатов

Название алгоритма	Сложность	Результат
Алгоритм Форда-Фалькерсона	O(f_{max} * E)	19 * 8 = 152
Алгоритм Эдмондса-Карпа	O(V * E^2)	6 * 8^2 = 384
Алгоритм Диница	O(V^2 * E)	6^2 * 8 = 288

Для проведения более точного анализа, проведем оценку графа сети из второго раздела.

$$G = \begin{bmatrix} 0 & 32 & 95 & 75 & 57 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 23 & 0 & 0 & 16 \\ 0 & 0 & 0 & 18 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 24 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 20 & 94 \\ 0 & 0 & 0 & 0 & 11 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 81 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

В данной матрице 8 вершин, включая источник и сток, и 16 ребер. Ниже приведена таблица с полученными результатами.

Таблица 2 – Анализ полученных результатов

Название алгоритма	Сложность	Результат
Алгоритм Форда-Фалькерсона	O(f_{max} * E)	128 * 8 = 1024
Алгоритм Эдмондса-Карпа	O(V * E^2)	8 * 16^2 = 2048
Алгоритм Диница	O(V^2 * E)	8^2 * 16 = 1024

Алгоритм Диница и алгоритм Форда-Фалкерсона справились с задачей за одно и тоже время, в том время как алгоритм Эдмондса-Карпа проводил вычисления намного дольше остальных.

3.3 Выводы по главе

Таким образом, был проведен анализ полученных результатов всеми тремя методами. Во всех случаях была найдена корректная величина

максимального потока. Для анализа были использованы все ранее описанные методы поиска максимального потока в транспортной сети.

В тексте главы продемонстрирован процесс визуализации решения задачи поиска максимального потока. На каждой итерации алгоритма был построен путь из источника к стоку и после был изображен на изображении графа с пропускными способностями и зафиксированной величиной потока на каждом ребре.

Согласно полученным результатам, можно сделать вывод, что для решения задачи из предыдущего пункта, лучшим показал себя алгоритм Форда-Фалькерсона. Данный алгоритм быстрее всех провел все необходимые вычисления. Помимо этого, он же оказался самым простым в реализации.
Для других возможных случаев, где количество вершин будет значительно превышать количество ребер. В такой ситуации будет более эффективным применить алгоритм Диница. В противоположном случае, в котором количество вершин будет превышать количество ребер, более эффективным будет алгоритм Эдмондса-Карпа.

На практики решаются задачи с большим объемом данных и выбор алгоритма для вычислений зависит от количества узлов и связей в сети. С подобными задачами, часто, заметно хуже остальных справляется алгоритм Форда-Фалкерсона. Однако, этот алгоритм остается актуальным, так как прост в реализации и вполне подходит для задач не требующих работы с большим объемом данных.

Заключение

В процессе выполнения задания были исследованы методы поиска максимального потока в транспортных сетях.

После изучения необходимых источников литературы, были реализованы алгоритмы для решения задачи о поиске максимального потока. Во время этого процесса были пополнены знания о возможностях языка программирования Python и его дополнительной библиотеке pydot.

В ходе работы был выполнен ряд поставленных в начале работы задач:

1. Изучена область объекта исследования;
2. Разобрана теорема о максимальном потоке и минимальном разрезе
3. Исследованы алгоритмы поиска максимального потока;
4. Выполнена реализация программы;
5. Проведен сравнительный анализ полученных результатов;
6. Сформирован вывод о проведенной работе.

Для проведения сравнительного анализа полученных результатов, был выполнен поиск максимального потока на основе различных входящих данных графа, представленных в виде матриц смежности.

Рассматриваемые матрицы отличались по количеству ребер и вершин. Это потребовалось для достижения необходимого результата и формирования понимания о недостатках одних и преимуществ других алгоритмов в различных ситуациях. Граф сети был представлен в виде матрицы смежности, в ячейках которой были указаны пропускные способности каждого из ребер.

В исследуемых алгоритмах, для обхода узлов сети были использованы алгоритм поиска в глубину и алгоритм поиска в ширину.

Все поставленные задачи были выполнены. Созданная программа позволяет определить максимальный поток в сети на основе входящих данных.

Список используемой литературы

1. Андерсон Д. Дискретная математика. – М.: Издательство Вильямс, 2019. – С. 960. – ISBN 978-5-907144-07-1.
2. Асанов М.О., Баранский В.А., Расин В. Дискретная математика: Графы, матроиды, алгоритмы. – Спб.: Издательство Лань, 2020. – С. 364. – ISBN 978-5-8114-4998-9.
3. Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы. – М.: Издательство Вильямс,2016. – С. 400. – ISBN 978-5-8459-1610-5.
4. Берцун В.Н. Математическое моделирование на графах. – Томск: Издательство НТЛ, 2006. – С. 88. – ISBN 5-89503-312-1.
5. Букатов А.А., Гуда С.А. Компьютерные сети. – Спб: Издательство Питер, 2019. – С. 496. – ISBN 978-5-4461-1338-5.
6. Дорофеева В.И. Алгоритмы на графах и их приложения. – Орел: ОГУ, 2002. – С. 48
7. Иванов Б.Н. Дискретная математика. Алгоритмы и программы. – М.: Лаборатория базовых знаний, 2003. – С. 289. – ISBN 5-93208-093-0.
8. Кормен Т.Х., Лейзерсон Ч. Алгоритмы: построение и анализ. – М.: Издательство Вильямс, 2011. – С. 1296. – ISBN 978-5-8459-0857-5.
9. Новиков Ф.А. Дискретная математика для программистов. – Спб: Издательство Питер, 2008. – С. 384. – ISBN 978-5-91180-759-7.
10. Овчинникова Е.В., Судоплатов С.В. Элементы дискретной математики. – М.: Издательство Юрайт, 2019. – С. 279. – ISBN 978-5-534-00871-5.
11. Окулов С. Программирование алгоритмов. – М.: Бином. Лаборатория знаний, 2007. – С. 384. – ISBN 978-5-94774-689-1.
12. Романовский И. Дискретный анализ. – Спб: Издательство БХВ-Петербург, 2016. – С. 352. – ISBN 978-5-7940-0152-5.

13. Свами М. Графы, Сети и алгоритмы. – М.: Издательство Наука, 2012. – С. 450. – ISBN 978-5-458-33391-7.

14. Теория графов [Электронный ресурс]: Онлайн курс. / Образовательный сайт Coursera. – Режим доступа: https://www.coursera.org/learn/teoriya-grafov (дата обращения: 2.03.2021)

15. Харари Ф. Теория графов. – М.: Издательство УРСС, 2018. – С. 304. – ISBN 978-5-9710-5127-5.

16. Apostol K. Ford-Fulkerson Algorithm. – International Book Market Service Limited, 2012. – 60 p. – ISBN 978-6-13-628748-5.

17. Cormen T.H. Introduction to algorithms. – MIT Press, 2001. – 1179 p. – ISBN 978-0-26-203293-3.

18. Heineman G.T. Algorithms in a nutshell. – O`Reilly, 2008. – 506 p. – ISBN 978-0-59-651624-6.

19. Introduction to Maxflow [Электронный ресурс]: Онлайн курс. / Образовательный сайт Coursera. – Режим доступа: https://www.coursera.org/lecture/algorithms-part2/introduction-to-maxflow-jZVUm (дата обращения: 19.03.2021)

20. Kleinberg J., Tardos E. Algorithm Design. – Pearson Education Inc., 2006. – 984 p. – ISBN 978-0-13-213108-7.

21. Lawler E. Combinatorial optimization: Networks and Matroids. – Dover Publications Inc, 2003. – 374 p. – ISBN 978-0-48-641453-9.

Приложение А

Алгоритм Форда-Фалкерсона

```
# поиск пути методом поиска в глубину
def dfs(C, F, s, t):
        # добавление в стек корневой вершины
        stack = [s]
        paths={s:[]}
        # условие окончания работы (если дошли из истока в сток, то выходим)
        if s == t:
                return paths[s]
        while(stack):
                # вытаскивание вершины из стека
                u = stack.pop()
                # обход графа
                for v in range(len(C)):
                        # поиск смежных вершин
                        if(C[u][v]-F[u][v]>0) and v not in paths:
                                # запись найденной вершины
                                paths[v] = paths[u]+[(u,v)]
                                print(paths)
                                # конец обхода при нахождение стока
                                if v == t:
                                        return paths[v]
                                # добавление обнаруженной вершины в стек
                                stack.append(v)
        return None

counter = 0

def FordFulkerson(C, s, t):
        # фиксирование длины строки
        n = len(C)
        F = [[0] * n for i in range(n)]
        # поиск в глубину для нахождения путей от истока до стока
        path = dfs(C, F, s, t)
        while path != None:
                # поиск узкого места
                flow = min(C[u][v] - F[u][v] for u,v in path)
                for u,v in path:
                        F[u][v] += flow
                        F[v][u] -= flow
                path = dfs(C,F,s,t)
        return sum(F[s][i] for i in range(n))
```

Приложение Б

Алгоритм Эдмондса-Карпа

```
# поиск смежных вершин с помощью алгоритма поиска в ширину
def bfs(C, F, s, t):
        # добавление истока в очередь
        queue = [s]
        paths = {s:[]}
        # условие окончания работы (если дошли из истока в сток, то выходим)
        if s == t:
            return paths[s]
        while queue:
            # вытаскивание вершины из очереди
            u = queue.pop(0)
            # обход графа
            for v in range(len(C)):
                    # поиск смежных вершин
                    if(C[u][v]-F[u][v]>0) and v not in paths:
                        # запись найденной вершины
                        paths[v] = paths[u]+[(u,v)]
                        # конец обхода при нахождение стока
                        if v == t:
                            return paths[v]
                        # добавление обнаруженной вершины в очередь
                        queue.append(v)
        return None

def EdmondsKarp(C, s, t):
        # фиксирование длины строки
        n = len(C)
        F = [[0] * n for i in range(n)]
        # поиск в ширину для нахождения путей от истока до стока
        path = bfs(C, F, s, t)
        while path != None:
            # поиск узкого места
            flow = min(C[u][v] - F[u][v] for u,v in path)
            for u,v in path:
                F[u][v] += flow
                F[v][u] -= flow
            path = bfs(C, F, s, t)
        return sum(F[s][i] for i in range(n))
```

Приложение В

Алгоритм Диница

```
def Bfs(C, F, s, t):
        # фиксирование длины строки (список смежности вершины)
        n = len(C)
        queue = []
        # добавление корневой вершины в очередь
        queue.append(s)
        global level
        # инициализация массива уровней с общим значением 0 для всех ячеек
        level = n * [0]
        # фиксирование первого уровня
        level[s] = 1
        while queue:
                # вытаскивание текущей вершины
                k = queue.pop(0)
                # обход графа
                for i in range(n):

                        if (F[k][i] < C[k][i]) and (level[i] == 0):
                                level[i] = level[k] + 1
                                queue.append(i)
        return level[t] > 0

def Dfs(C, F, k, cp):
        tmp = cp
        if k == len(C)-1:
            return cp
        for i in range(len(C)):
            if (level[i] == level[k] + 1) and (F[k][i] < C[k][i]):
                f = Dfs(C,F,i,min(tmp,C[k][i] - F[k][i]))
                F[k][i] = F[k][i] + f
                F[i][k] = F[i][k] - f
                tmp = tmp - f
        return cp - tmp

def Dinitz(C,s,t):
        n = len(C)
        F = [n*[0] for i in range(n)]
        flow = 0
        while(Bfs(C,F,s,t)):
               flow = flow + Dfs(C,F,s,100000)
        return flow
```