МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ Федеральное государственное автономное образовательное учреждение высшего образования

**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра: Алгебры, геометрии и дискретной математики**

Направление подготовки: «Прикладная математика и информатика»

Профиль подготовки: «Прикладная математика и информатика (общий профиль)»

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

**Тема:**

**«Сравнительный анализ решений задачи о кратчайших путях методом поиска в ширину для нескольких классов графов»**

Выполнил:

студент группы 381703-1

Шахнов Андрей Владимирович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

Научный руководитель:

Доцент кафедры АГДМ, кандидат физико-математических наук

Сорочан Сергей Владимирович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

Нижний Новгород  
2021

Содержание

[1. Введение 3](#_Toc74252629)

[2. Постановка задачи 4](#_Toc74252630)

[2.1 Основные определения и понятия 4](#_Toc74252631)

[2.2 Способы задания графов 7](#_Toc74252632)

[3. Методы обхода графа 10](#_Toc74252633)

[3.1 Поиск в ширину 10](#_Toc74252634)

[3.2 BFS Дерево 12](#_Toc74252635)

[4. Поиск кратчайшего пути 13](#_Toc74252636)

[5. Результаты экспериментов 15](#_Toc74252637)

[6. Заключение 25](#_Toc74252638)

[7. Литература 26](#_Toc74252639)

[8. Приложение 27](#_Toc74252640)

# Введение

Графы являются очень важным объектом в математике и находят себе применения в совершенно различных сферах нашей жизни. С помощью графов можно легко наблюдать за взаимоотношениями объектов или каких-либо событий в нетривиальных системах. Графы широко востребованы как во многих областях науки, например, в физике, химии, исследовании операций, теории вероятностей, так и в нашей повседневной жизни – карта метро, схема железнодорожных путей, генеалогическое древо человека, и даже созвездия на ночном небе – всё это разновидности графов. В информатике же графы представляют собой очень удобную структуру для хранения данных – вершинами графа может быть что угодно – переменная, список, класс или даже целая структура. Так, например, внутри каждого установщика какой-нибудь программы существует граф зависимостей всех устанавливаемых элементов. В IT сфере графы наиболее востребованы в машинном обучении, каждая нейронная сеть представляет собой граф, описывающий связи между нейронами.

Существует множество способов обхода графа, и в моей работе я использую один из них – поиск в ширину.

Цель работы – исследовать нахождение кратчайшего пути от заданной вершины до любой другой, сравнить значение эксцентриситета вершины в зависимости от типа и плотности графа, сравнить время выполнения поиска в ширину на тех же классах графов при разной вероятности появления ребра между вершинами. В данной работе исследования будут проводиться на обыкновенных, двудольных и расщепляемых классах графов.

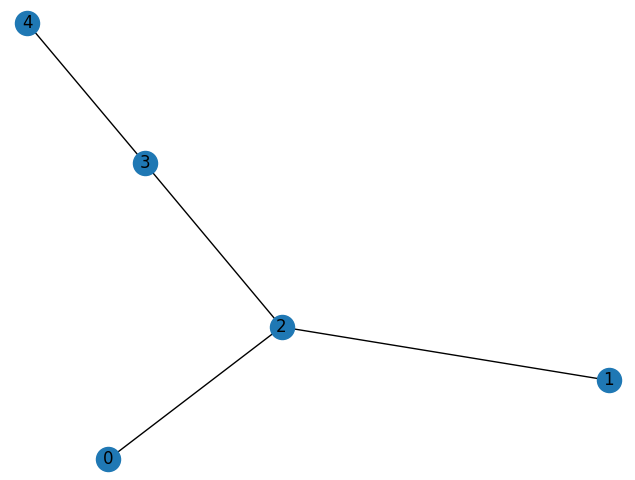
# Постановка задачи

## 2.1 Основные определения и понятия[[1]](#footnote-1)

*Обыкновенным графом* G будем называть такую пару (V, E), где V – конечное непустое множество вершин, а E – пары вершин, или же множество рёбер.

В данной работе будут рассмотрены связные неориентированные графы без петель.   
Неориентированность графа означает, что его рёбра не имеют направлений. Петлёй называют ребро, инцидентное одной и той же вершине.

Для наглядности приведём пример такого графа: G = ({1,2}, {0,2}, {2,3}, {3,4})  
и изобразим его графически:



В приведённом графе ребро (2, 1) будет полностью соответствовать ребру (1, 2)

Введём понятие *эксцентриситета* вершины - это расстояние от вершины A до самой удалённой от неё вершины. В приведённом графе вершины «0», «1», и «4» имеют значение эксцентриситета равное трём. Эксцентриситет вершин «3» и «2» равен двум.

*Радиусом* графа является минимальное значение эксцентриситета в графе.

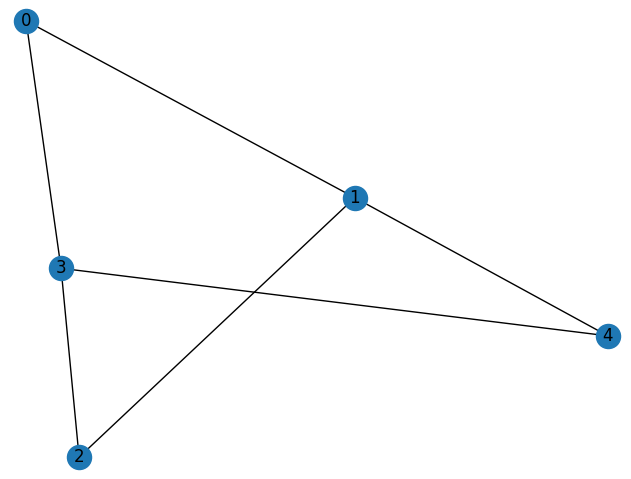
*Центром* графа называют вершину (или множество вершин), чей эксцентриситет равен радиусу графа.

*Диаметром* графа называют максимально возможное расстояние между двумя вершинами. В данном примере вершины «3» и «2» будут являться центром графа. Радиус графа R(G) будет равен двум, а диаметр D(G) будет равен трём.

*Степенью* *вершины* называют количество рёбер, которым принадлежит эта вершина. Например, на приведённом выше графе вершина «0» будет иметь степень один, а вершина «2» будет иметь степень три.

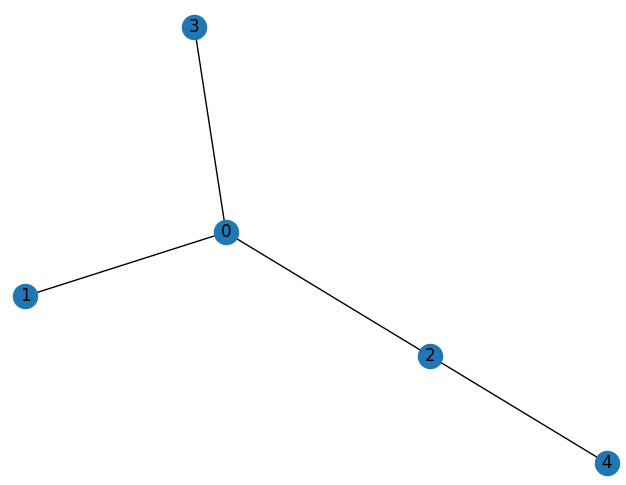
*Двудольным* графом G = (W, E) будем называть такой граф, множество вершин которого можно разбить на две части UV = W таким образом, что ни одна вершина из множества U не соединена с вершинами из множества U, и ни одна вершина из множества V не соединена с вершинами множества V. Множества U и V называются долями графа двудольного графа. Если в двудольном графе каждая вершина одной доли соединена с каждой вершиной другой доли, граф называют полным двудольным графом.

Для наглядности приведём пример полного двудольного графа G = {(0,1), (0,3), (1,2), (1,4), (3,4), (3,2)} и изобразим его графически:



В данном графе вершины «0», «2», и «4» являются вершинами одной доли, а вершины «1» и «3» вершинами второй доли.

Также приведём пример неполного двудольного графа G = {(0,1), (0,3), (0,2), (2,4)} и изобразим его графически:



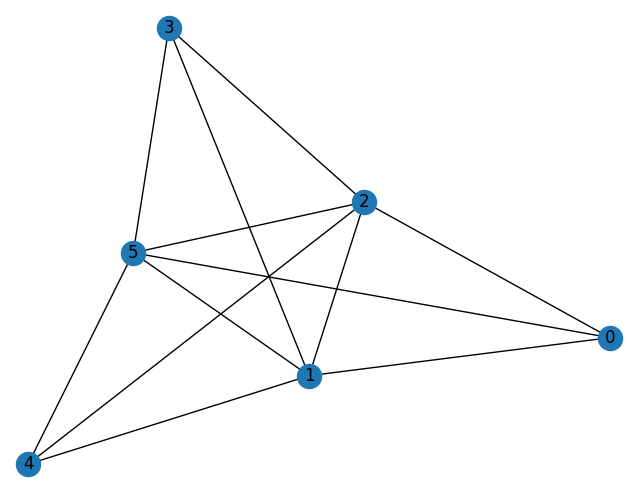
В данном графе вершины «1», «2», и «3» являются вершинами одной доли, а вершины «0» и «4» вершинами второй доли.

*Расщепляемым* графом будем называть такой граф, в котором вершины можно разделить на две части – *клику* и *независимое множество.*

*Кликой* неориентированного графа называют множество вершин, каждая из которых попарно соединена друг с другом. Иными словами, если в графе встречается подграф, являющийся полным графом, множество вершин подграфа будет называться кликой.

*Независимым множеством* называется множество вершин, каждая из которых попарно не соединена ребром, или иными словами, множество, порождающее пустой подграф.

Для наглядности приведём пример полного расщепляемого графа G = {(0, 1), (0, 2), (0, 5), (4, 1), (4, 2), (4, 5), (3, 1), (3, 2), (3,5), (1, 2), (1, 5), (2, 5)}



В данном графе вершины «1», «2» и «5» образуют клику размера три, а вершины «0», «3» и «4» являются независимым множеством.

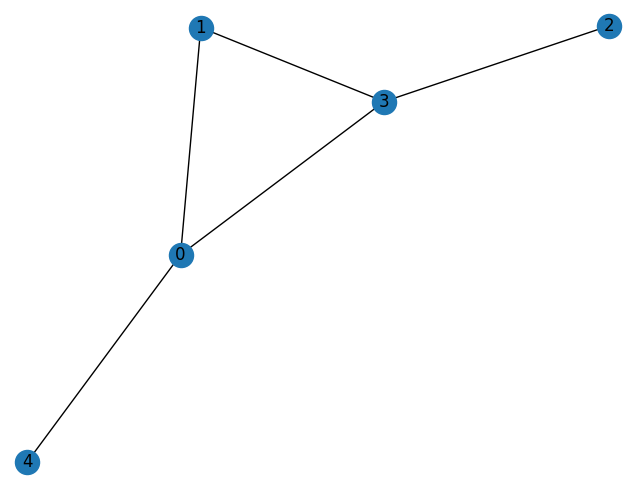
## 2.2 Способы задания графов

На данный момент существует четыре способа представления графов:

1. Графический
2. Список вершин и рёбер
3. Матрица смежности
4. Матрица инцидентности

Рассмотрим каждый из них поподробнее:

1. Графический способ заключается в том, что вершины графа представляются точками на плоскости, а рёбра – линиями, соединяющими соответствующие точки между собой. Для примера представим расщепляемый граф графическим способом:



Назовём этот граф G1 и будем рассматривать все дальнейшие способы основываясь на этом графе

1. Список вершин и рёбер

Чтобы задать граф таким способом, необходимо указать множество вершин графа – множество V = {0, 1, 2, 3, 4} и множество рёбер – множество E = {(0,1), (0, 3), (0, 4), (1, 3), (3, 2)}

1. Матрица смежности

*Матрицей смежности* называется квадратная матрица порядка n, где n – число вершин графа. В матрице столбцы и строки – вершины графа. Если между вершинами есть ребро, в матрице на месте их пересечения стоит единица, если ребра нет – ноль.

Для графа G1 матрица смежности будет выглядеть следующим образом:

Так как в графе отсутствуют петли, на главной диагонали стоят нули. Кроме того, матрица смежности всегда симметрична.

1. Матрица инцидентности

*Матрицей инцидентности* называют матрицу, представляющую связь между ребрами и вершинами графа.

Строки матрицы соответствуют рёбрам, а столбцы – вершинам. В нашем случае столбцы будут представлять вершины (0, 1, 2, 3, 4) а строки рёбра (0-1), (0-3), (0-4), (1-3), (2-3)

Для графа G1 матрица инцидентности будет выглядеть следующим образом:

# Методы обхода графа

На данный момент существует множество способов обхода графов, но самыми основными считаются метод поиска в ширину (Breadth-First Search) и метод поиска в глубину (Deep-First Search).

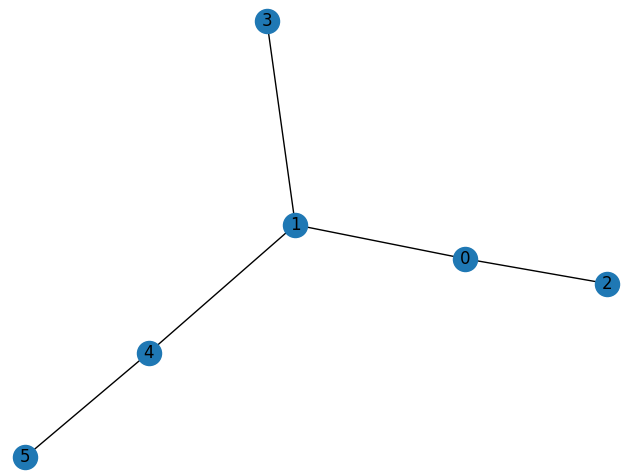
Обход графа – это переход от одной вершины графа к другой с помощью отыскания свойств связей между вершинами. Любая хотя бы раз посещённая вершина запоминается и считается посещённой до конца обхода графа. Каждая непосещённая вершина называется *новой*. Как только в результате обхода мы обнаруживаем непосещённую вершину, она становится *открытой*, и остаётся такой до того момента, пока не будут обнаружены все инцидентные ей рёбра. После того, как все ребра станут известны, вершина будет *закрытой*.

В данной работе все исследования и эксперименты будут основаны на поиске в ширину, поэтому остановимся на нём.

## 3.1 Поиск в ширину

*Поиск в ширину* – алгоритм обхода графа, заключающийся в поиске вершин, основываясь на их удалённости от заданной вершины. Другими словами, после посещения стартовой вершины будут найдены все вершины, которые находятся на расстоянии один от нее, затем два, три и так далее.

Визуализируем обход графа методом поиска в ширину, и пронумеруем вершины так, как они будут посещаться в процессе обхода:



Обозначим V(*x*) как окрестность вершины *x*, а Q – очередь для открытых вершин

**Procedure BFS(𝑎)[[2]](#footnote-2)**

1. Посетить вершину 𝑎
2. 𝑎 ⇒𝑄
3. while 𝑄≠∅ do
4. 𝑥⟸𝑄
5. for 𝑦 ∈𝑉(𝑥) do
6. Исследовать ребро (𝑥, 𝑦)
7. if 𝑦 – новая вершина
8. then посетить вершину
9. 𝑦 ⇐𝑄

Поиск в ширину обладает несколькими свойствами:

1. Поиск завершается по истечении конечного числа шагов

Всякий раз, когда вызывается цикл *while* из очереди исключается одна вершина. Только посещенная вершина присоединяется к очереди. После посещения все новые вершины перестают быть таковыми, отсюда следует вывод, что каждая вершина посещается не более одно раза. Получается, что число повторений цикла *while* не превышает числа вершин графа.

1. В компоненте связности, содержащей вершину 𝑎, будут посещены все вершины и только они.

Любая вершина в графе будет посещена, если у нее существует путь, соединяющий её с вершиной 𝑎.

1. Оценка трудоёмкости алгоритма – 𝑂(𝑚), где 𝑚 – число ребер в компоненте связности, содержащей вершину 𝑎.

В одной компоненте связности каждая вершина будет активной точно один раз. Для активной вершины 𝑥 цикл for выполняется 𝑑𝑒𝑔(𝑥) раз, где 𝑑𝑒𝑔(𝑥) – степень вершины x. Тогда, общее число повторений цикла равно: ∑ 𝑑𝑒𝑔(𝑥) =2𝑚

**Procedure BFS(𝑎)** используется для одной компоненты связности, в которой находится вершина 𝑎. Для полного обхода графа поиском в ширину необходимо добавить пару шагов.

**Алгоритм «Поиск в ширину»[[3]](#footnote-3)**

1. Все вершины отметить как новые

2. Создать пустую очередь 𝑄

3. for 𝑎 ∈𝑉 do if 𝑎 новая then BFS(𝑎)

Трудоёмкость процедуры BFS(𝑎) нам уже известна, посчитаем ее для данного алгоритма. Так как 3 строчка будет повторяться 𝑛 раз (число вершин в графе), то общее время работы всего алгоритма 𝑂(𝑚+𝑛). Однако эта оценка справедлива, если граф задан списками смежности. Для графа, заданного матрицей смежности, оценка трудоёмкости будет выглядеть так: 𝑂(𝑛2). Такое отличие происходит из-за того, что в матрице смежности для просмотра окрестности любой вершины будет затрачиваться время, пропорциональное n.

## 3.2 BFS Дерево[[4]](#footnote-4)

Дерево - это связный граф, не имеющий циклов. Для таких графов применяется своя терминология. Дерево с выделенной вершиной называется корневым деревом, а сама помеченная вершина – корнем. В ребре (𝑥, 𝑦) вершина 𝑥 будет считаться отцом, а вершина 𝑦 её сыном.

Выполним поиск в ширину для связного графа. Рассмотрим каркас графа, построенный прямыми рёбрами, как корневое дерево, где корень – это стартовая вершина. Полученный граф называется BFS – деревом. Чтобы построить его, следует во время обхода графа заполнить таблицу отцов 𝐹, то есть при активной вершине 𝑥, во время посещения вершины 𝑦, присвоить значение 𝐹(𝑦)=𝑥.

# Поиск кратчайшего пути

С задачей поиска в ширину тесно связана задача нахождения кратчайшего пути между вершинами в графе.

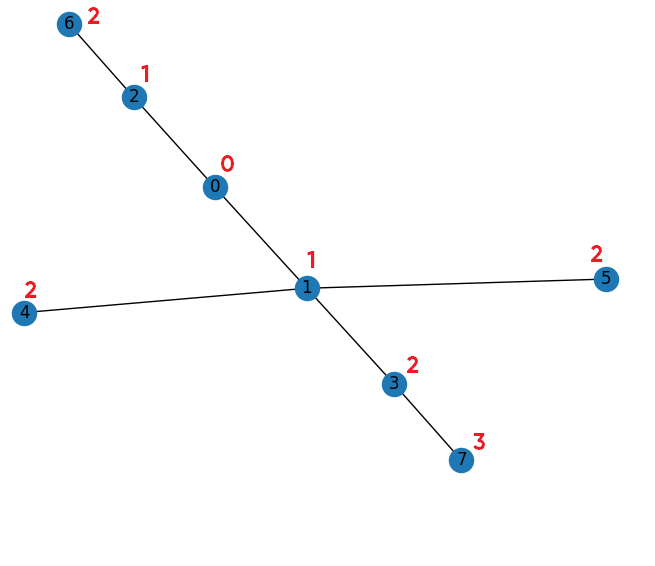
*Кратчайшим путём* из одной вершины в другую будем считать путь минимальной длины между этими вершинами.

Опишем **алгоритм поиска кратчайшего пути**, используемый в моей работе:

Во время проведения поиска в ширину будем выставлять всем обнаруженным вершинам метки, значением которых будет номер итерации, на которых эти вершины были обнаружены и добавлены в очередь.

1. Выбирается корневая вершина (вершина, от которой будет производиться поиск), ей присваивается метка со значением «ноль».
2. Производится поиск всех вершин, смежных с корневой, и их добавление в очередь. Обнаруженным вершинам присваивается метка «один». Значение этой метки есть минимальная длина пути до этой вершины из стартовой.
3. Далее будем производить поиск всех соседей для вершин, добавленных в очередь на предыдущем шаге и присваивать им номер итерации.
4. Алгоритм завершается, когда завершается поиск в ширину – в очереди не осталось вершин, для которых можно обнаружить соседей.

Визуализируем результат работы алгоритма:

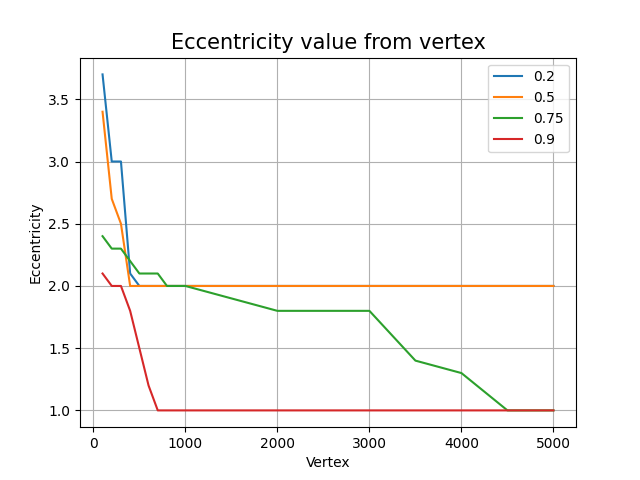
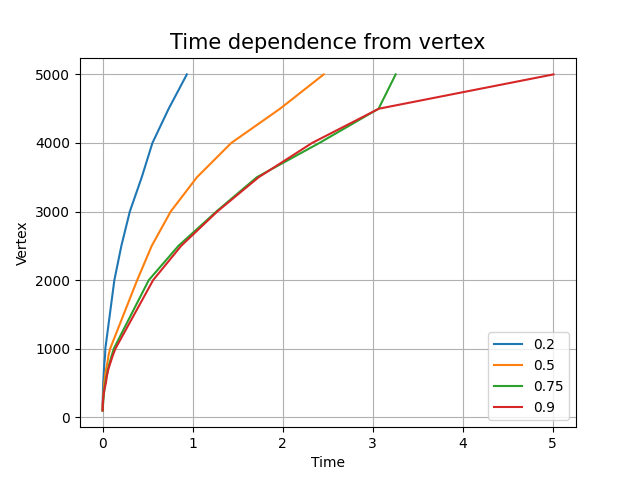


Для того чтобы определить кратчайший путь из стартовой вершины до заданной, достаточно лишь пройти путь из конечной вершины по уменьшению меток. Например, чтобы определить кратчайший путь из вершины «0» в вершину «7», нужно посмотреть на значение метки вершины «7» (равное трём), найти вершину(ы) со значением метки 3-1=2 – такой вершиной будет вершина «3», затем найти вершину со значением метки 2-1=1 – такой вершиной будет вершина «1», а она является смежной с корневой. Таким образом, отсортировав вершины в обратном порядке, получим путь {0, 1, 3, 7}, длина которого будет равна трём.

# Результаты экспериментов

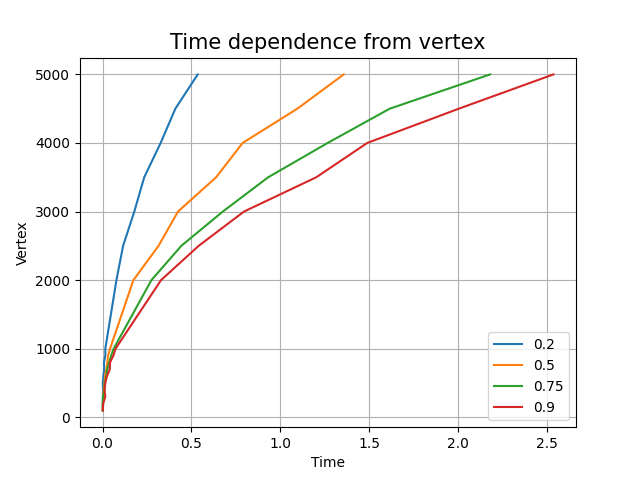
В первом эксперименте производится поиск в ширину для обыкновенных связных графов, построенных по матрице смежности, с разной разреженностью. Разреженность графа – вероятностью отсутствия ребра между вершинами. Чем разреженнее граф, тем меньше в нём рёбер. Вычисления производились для графов с количеством вершин от 100 до 5000. На каждое количество вершин строилось по 10 графов и вычислялось среднее время поиска в ширину, затем результаты отображались на графиках. Также было найдено среднее значение эксцентриситета вершин для каждого числа вершин. Ниже представлены таблица и график зависимости времени работы поиска в ширину от числа вершин в графе с определенной плотностью и график зависимости эксцентриситета от числа вершин:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Число вершин | Среднее время работы | | | |
| Плотность графа | | | |
| 0.2 | 0.5 | 0.75 | 0.9 |
| 100 | 0.0004986 | 0.0003015 | 0.0012964 | 0.0015593 |
| 200 | 0.0013983 | 0.0022400 | 0.0066096 | 0.0031247 |
| 300 | 0.0030916 | 0.0082063 | 0.0122575 | 0.0093736 |
| 400 | 0.0055425 | 0.0140345 | 0.0202034 | 0.0203014 |
| 500 | 0.0077442 | 0.0186746 | 0.0305513 | 0.0374934 |
| 600 | 0.0123514 | 0.0289248 | 0.0452020 | 0.0484200 |
| 700 | 0.0159682 | 0.0399138 | 0.0599137 | 0.0648761 |
| 800 | 0.0214657 | 0.0548526 | 0.0803429 | 0.0891486 |
| 900 | 0.0285560 | 0.0664909 | 0.1018881 | 0.1140392 |
| 1000 | 0.0313105 | 0.0845520 | 0.1251094 | 0.1433185 |
| 2000 | 0.1315479 | 0.3852097 | 0.5139750 | 0.5623549 |
| 2500 | 0.2094345 | 0.5469313 | 0.8439902 | 0.8739775 |
| 3000 | 0.3032871 | 0.7574992 | 1.2648590 | 1.2746792 |
| 3500 | 0.4357714 | 1.0490714 | 1.7141929 | 1.7347203 |
| 4000 | 0.5544304 | 1.4301555 | 2.4090974 | 2.3283137 |
| 4500 | 0.7352420 | 1.9731782 | 3.0673690 | 3.0710709 |
| 5000 | 0.9369348 | 2.4578583 | 3.2570489 | 5.0103962 |



Во втором эксперименте производились аналогичные вычисления для обыкновенных связных графов, заданных списком рёбер.

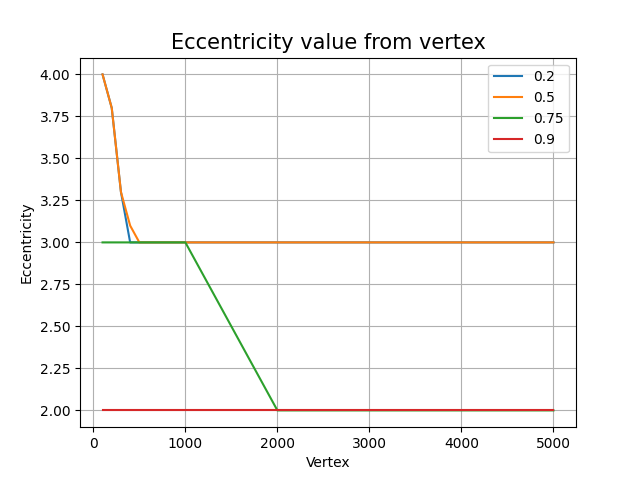
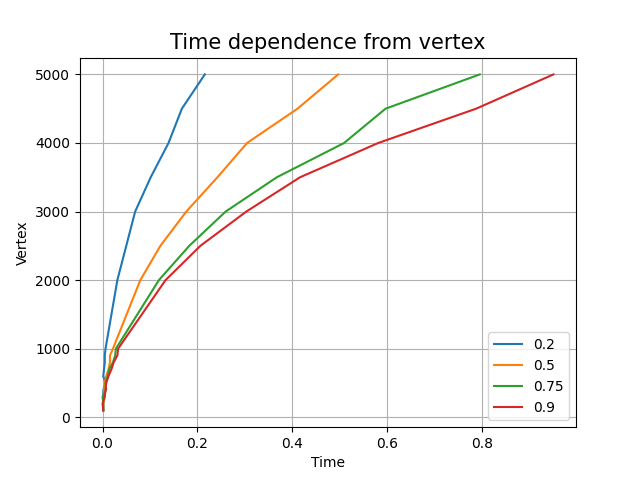
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Число вершин | Среднее время работы | | | |
| Плотность графа | | | |
| 0.2 | 0.5 | 0.75 | 0.9 |
| 100 | 0.0015620 | 0.0005125 | 0.0015621 | 0.0015723 |
| 200 | 0.0031275 | 0.0031210 | 0.0031312 | 0.0031258 |
| 300 | 0.0015622 | 0.0031240 | 0.0031224 | 0.0156146 |
| 400 | 0.0031209 | 0.0062492 | 0.0109468 | 0.0109435 |
| 500 | 0.0015622 | 0.0109383 | 0.0125029 | 0.0156333 |
| 600 | 0.0046910 | 0.0140630 | 0.0218883 | 0.0249915 |
| 700 | 0.0093799 | 0.0218761 | 0.0265629 | 0.0406071 |
| 800 | 0.0078152 | 0.0265627 | 0.0374881 | 0.0421836 |
| 900 | 0.0156225 | 0.0312498 | 0.0484168 | 0.0609272 |
| 1000 | 0.0156247 | 0.0421863 | 0.0624685 | 0.0731261 |
| 2000 | 0.0781008 | 0.1733935 | 0.2749286 | 0.3287643 |
| 2500 | 0.1155993 | 0.3149579 | 0.4428274 | 0.5404993 |
| 3000 | 0.1785984 | 0.4241631 | 0.6762474 | 0.7952489 |
| 3500 | 0.2343164 | 0.6387393 | 0.9313270 | 1.2015882 |
| 4000 | 0.3264856 | 0.7873750 | 1.2653278 | 1.4878900 |
| 4500 | 0.4092822 | 1.0950509 | 1.6143495 | 2.0028306 |
| 5000 | 0.5349989 | 1.3559430 | 2.1788090 | 2.5347759 |



Дальнейшие эксперименты проводились исключительно на графах, заданных списком рёбер.

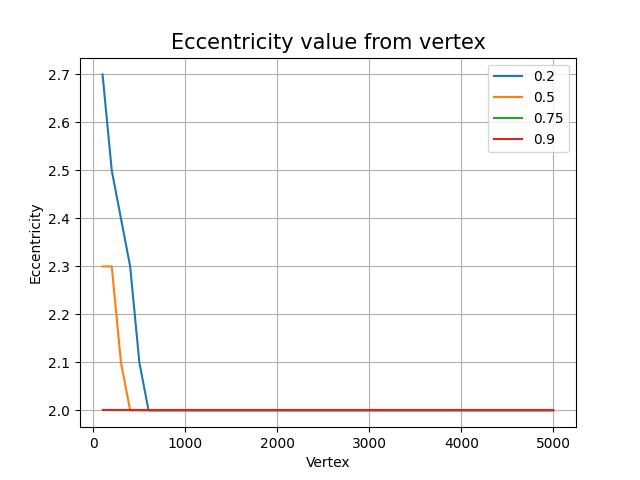
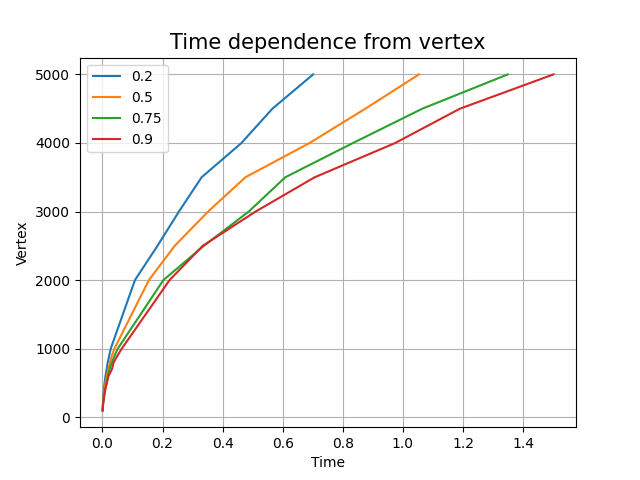
В третьем эксперименте проводились аналогичные вычисления для двудольных графов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Число вершин | Среднее время работы | | | |
| Плотность графа | | | |
| 0.2 | 0.5 | 0.75 | 0.9 |
| 100 | 0.0015621 | 0.0015615 | 0.0015589 | 0.0015641 |
| 200 | 0.0015615 | 0.0015620 | 0.0015619 | 0.0023653 |
| 300 | 0.0015589 | 0.0046859 | 0.0016312 | 0.0046901 |
| 400 | 0.0017543 | 0.0031238 | 0.0078043 | 0.0062490 |
| 500 | 0.0062489 | 0.0046860 | 0.0062502 | 0.0078104 |
| 600 | 0.0015583 | 0.0078086 | 0.0109340 | 0.0124978 |
| 700 | 0.0031213 | 0.0124984 | 0.0140615 | 0.0187509 |
| 800 | 0.0046868 | 0.0156137 | 0.0218790 | 0.0234396 |
| 900 | 0.0046884 | 0.0156226 | 0.0265492 | 0.0312365 |
| 1000 | 0.0062466 | 0.0218813 | 0.0281222 | 0.0327983 |
| 2000 | 0.0312226 | 0.0796695 | 0.1187255 | 0.1327828 |
| 2500 | 0.0499797 | 0.1218541 | 0.1827727 | 0.2061973 |
| 3000 | 0.0687409 | 0.1765171 | 0.2594789 | 0.3030331 |
| 3500 | 0.1015471 | 0.2421258 | 0.3673076 | 0.4162739 |
| 4000 | 0.1390306 | 0.3046041 | 0.5090568 | 0.5811004 |
| 4500 | 0.1671521 | 0.4108352 | 0.5963552 | 0.7873118 |
| 5000 | 0.2155544 | 0.4966590 | 0.7951277 | 0.9502784 |



В четвёртом эксперименте проводились аналогичные вычисления для расщепляемых графов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Число вершин | Среднее время работы | | | |
| Плотность графа | | | |
| 0.2 | 0.5 | 0.75 | 0.9 |
| 100 | 0.0003069 | 0.0006884 | 0.0007013 | 0.0008124 |
| 200 | 0.0015394 | 0.0019266 | 0.0022759 | 0.0022131 |
| 300 | 0.0026953 | 0.0037612 | 0.0050725 | 0.0057224 |
| 400 | 0.0045672 | 0.0065948 | 0.0086456 | 0.0093848 |
| 500 | 0.0070391 | 0.0102974 | 0.0145787 | 0.0142226 |
| 600 | 0.0101920 | 0.0142962 | 0.0179556 | 0.0200365 |
| 700 | 0.0138892 | 0.0193726 | 0.0233248 | 0.0315043 |
| 800 | 0.0175925 | 0.0246977 | 0.0314706 | 0.0372243 |
| 900 | 0.0225637 | 0.0315256 | 0.0405111 | 0.0501233 |
| 1000 | 0.0275158 | 0.0396981 | 0.0508594 | 0.0642329 |
| 2000 | 0.1081408 | 0.1545497 | 0.2026820 | 0.2233534 |
| 2500 | 0.1830810 | 0.2404685 | 0.3376822 | 0.3337672 |
| 3000 | 0.2546111 | 0.3517302 | 0.4869496 | 0.5098368 |
| 3500 | 0.3301415 | 0.4759818 | 0.6086698 | 0.7065004 |
| 4000 | 0.4628097 | 0.6896358 | 0.8341616 | 0.9746884 |
| 4500 | 0.5657108 | 0.8751977 | 1.0648436 | 1.1899045 |
| 5000 | 0.7013343 | 1.0535964 | 1.3490712 | 1.5012792 |



# Заключение

В данной работе был проведён ряд экспериментов, основанных на поиске в ширину и нахождении кратчайших путей на различных классах графов.

Опытным путём был доказан известный факт, что на графах, заданных матрицей смежности поиск в ширину работает гораздо медленнее, чем на графах, заданных списком рёбер. В моём случае, исходя из графиков на страницах 15 и 18, время поиска в ширину на графах, заданных списком смежности, оказалось в 2 раза меньше, чем на заданных матрицей смежности. В связи с этим было принято решение проводить дальнейшие эксперименты пользуясь только списками смежности для оптимизации времени поиска.

Сравнив графики зависимости времени от числа вершин можно сделать вывод, что быстрее всего поиск в ширину работает на двудольных графах. При плотности графа 90% поиск в ширину на обыкновенных графах, заданных списком рёбер составляет примерно 2.5 секунды, 1.5 секунды на расщепляемых графах и около одной секунды на двудольных графах для 5000 вершин. При количестве вершин меньше одной тысячи время работы программы практически идентично.

Сравнив графики зависимости эксцентриситета от числа вершин можно сделать вывод, что с увеличением плотности графа значение эксцентриситета уменьшается. Для обыкновенных графов увеличение плотности графа приближает его к полному, и эксцентриситет стремится к одному. Интересное наблюдение – во время увеличения плотности графа до 50% его значение стремилось к 2.0, а уже при 75% стало стремиться к одному. Похожая ситуация происходит с двудольными графами, разве что минимальное значение эксцентриситета в двудольном графе не может быть меньше двух. Значение эксцентриситета для полного двудольного графа всегда равно двум. По графику на странице 20 видно, что с увеличением плотности до 50% значение стремится к 3.0, а до 90% к 2.0. В расщепляемых графах значение эксцентриситета для разреженных графов в моей программе получилось равно 3м, и с увеличением плотности графа оно также, как и в двудольном графе, стремится к двум.

# Литература

1. Алексеев, В.Е. Графы. Модели вычислений. Структуры данных: Учебник / В.Е. Алексеев, В.А. Таланов. – Нижний Новгород: изд-во ННГУ, 2004. – 291 с.

2. Алексеев, В.Е. Теория графов: Учебное пособие / В.Е. Алексеев, Д.В. Захарова. – Нижний Новгород: изд-во Нижегородский госуниверситет, 2017. –119 с.

3. Харари Ф. Теория графов / Харари Ф. – Москва: изд-во Мир, 1973. – 300 с.

4. Оре О. Графы и их применение / Оре О. – Москва: изд-во Мир, 1965. – 175 с.

5. Берж К. Теория графов и ее применения / Берж К. – Москва: изд-во Книга по Требованию, 2013. – 318 с.

# Приложение

**Файл plot\_builder.py для построения графиков**

import matplotlib.pyplot as plot

data\_n = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 2500, 3000, 3500, 4000, 4500, 5000]

data\_t1 = [3.7, 3.0, 3.0, 2.1, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0]

data\_t2 = [3.4, 2.7, 2.5, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0]

data\_t3 = [2.4, 2.3, 2.3, 2.2, 2.1, 2.1, 2.1, 2.0, 2.0, 2.0, 1.8, 1.8, 1.8, 1.4, 1.3, 1.0, 1.0]

data\_t4 = [2.1, 2.0, 2.0, 1.8, 1.5, 1.2, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

plot.title('Eccentricity value from vertex', fontsize=15)

plot.plot(data\_n, data\_t1, label='0.2')

plot.plot(data\_n, data\_t2, label='0.5')

plot.plot(data\_n, data\_t3, label='0.75')

plot.plot(data\_n, data\_t4, label='0.9')

plot.grid()

plot.legend()

plot.ylabel('Eccentricity')

plot.xlabel('Vertex')

plot.show()

**Файл bi\_graph.py для построения двудольных графов**

import networkx as nx

import random

import time

import numpy as np

import matplotlib.pyplot as plot

import pylab as plt

from time\_counter import time\_counter

import random

data\_n = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 2500, 3000, 3500, 4000, 4500, 5000]

data\_test = [1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000]

data\_t = []

def generate\_bigraph(n):

sides = dict()

sides[1] = []

sides[2] = []

keys = sides.keys()

for k in range(0, n):

node = random.randint(1, 2)

sides[node].append(k)

g = nx.Graph()

for new\_node in sides[1]:

g.add\_node(new\_node)

for new\_node in sides[2]:

g.add\_node(new\_node)

for q in sides[1]:

for j in sides[2]:

probability = random.random()

if probability <= 0.9:

g.add\_edge(q, j)

isolate = list(nx.isolates(g))

if isolate:

for i in isolate:

g.add\_edge(i, 1)

nx.draw(g, with\_labels=True)

plt.savefig("plot.png")

plt.show()

return g

def eccent(g):

end = 1

eccentricity = dict()

lens = []

isolate = list(nx.isolates(g))

if isolate:

for i in isolate:

g.add\_edge(i, end)

for i in range(n - 1):

start = 0

p = nx.shortest\_path(g, source=0, target=end)

end += 1

if len(p) in eccentricity:

eccentricity[len(p)].append(p)

else:

eccentricity[len(p)] = []

eccentricity[len(p)].append(p)

lens.append(len(p))

max\_len = max(lens) - 1

"""cent = nx.center(g)

print("Center - ", cent)

diam = nx.diameter(g)

print("Diameter - ", diam)

rad = nx.radius(g)

print("Radius - ", rad)"""

return max\_len

d = dict()

for n in data\_n:

a = []

d[n] = list()

for i in range(10):

graph = generate\_bigraph(n)

time = time\_counter(graph)

ewa = eccent(graph)

a.append(time)

d[n].append(ewa)

print(a)

print(d)

t = np.sum(a)

print("sum = ", t)

print("average = ", t/10)

data\_t.append(t/10)

print(data\_t)

#data\_t = sorted(data\_t)

data\_to\_dataset = str(data\_t) + "\n" + str(data\_n) + "\n"

data\_path = str(d) + "\n"

with open("dataset.txt", "a") as file:

file.write(data\_to\_dataset)

with open("dataset\_ways.txt", "a") as file:

file.write(data\_path)

plot.plot(data\_t, data\_n)

plot.ylabel('Vertex')

plot.xlabel('Time')

plot.show()

**Файл split\_graph.py для построения расщепляемых графов**

import networkx as nx

import random

import time

import numpy as np

import matplotlib.pyplot as plot

import pylab as plt

from cycler import cycler

from time\_counter import time\_counter

data\_n = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 2500, 3000, 3500, 4000, 4500, 5000]

data\_t = []

def generate\_splitgraph(n):

sides = dict()

sides[1] = []

sides[2] = []

keys = sides.keys()

for k in range(0, n):

node = random.randint(1, 2)

#if len(sides[node]) == n // 2:

#for j in keys:

#if j != node:

#sides[j].append(k)

#else:

if k == n-1:

if not sides[2]:

sides[2].append(k)

elif len(sides[2]) == n-1:

sides[1].append(k)

else:

sides[node].append(k)

else:

sides[node].append(k)

g = nx.Graph()

for new\_node in sides[1]:

g.add\_node(new\_node)

for new\_node in sides[2]:

g.add\_node(new\_node)

for q in sides[1]:

for j in sides[2]:

probability = random.random()

if probability <= 0.9:

g.add\_edge(q, j)

for l in sides[1]:

for e in sides[1]:

g.add\_edge(l, e)

isolate = list(nx.isolates(g))

for node in isolate:

if node in sides[2]:

g.add\_edge(node, sides[1][0])

return g

def eccent(g):

end = 1

eccentricity = dict()

lens = []

isolate = list(nx.isolates(g))

if isolate:

for i in isolate:

g.add\_edge(i, end)

for i in range(n - 1):

start = 0

p = nx.shortest\_path(g, source=0, target=end)

end += 1

if len(p) in eccentricity:

eccentricity[len(p)].append(p)

else:

eccentricity[len(p)] = []

eccentricity[len(p)].append(p)

lens.append(len(p))

max\_len = max(lens) - 1

return max\_len

d = dict()

for n in data\_n:

a = []

d[n] = list()

for i in range(10):

graph = generate\_splitgraph(n)

ewa = eccent(graph)

time = time\_counter(graph)

a.append(time)

d[n].append(ewa)

print(a)

print(d)

t = np.sum(a)

print("sum = ", t)

print("average = ", t/10)

data\_t.append(t/10)

print(data\_t)

#data\_t = sorted(data\_t)

data\_to\_dataset = str(data\_t) + "\n" + str(data\_n) + "\n"

data\_path = str(d) + "\n"

with open("dataset.txt", "a") as file:

file.write(data\_to\_dataset)

with open("dataset\_ways.txt", "a") as file:

file.write(data\_path)

#plot.title('Chart price', fontsize=17)

#plot.plot(data\_t, data\_n, label='first')

#plot.legend()

#plot.ylabel('Vertex')

#plot.xlabel('Time')

#plot.show()

**Файл time\_counter.py для расчета времени выполнения программы**

import networkx as nx

import random

import time

import numpy as np

import matplotlib.pyplot as plot

def time\_counter(graph):

root = 0

start\_time = time.time()

edges = nx.bfs\_edges(graph, root)

nodes = [root] + [v for u, v in edges]

times = time.time() - start\_time

return times

**Файл graph.py для построения обыкновенных графов по матрице смежности и списку рёбер**

import networkx as nx

import numpy as np

import matplotlib.pyplot as plot

from time\_counter import time\_counter

data\_n = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 2500, 3000, 3500, 4000, 4500, 5000]

data\_t = []

flag = 0

check = 0

def eccent(g):

end = 1

eccentricity = dict()

lens = []

isolate = list(nx.isolates(g))

if isolate:

for i in isolate:

g.add\_edge(i, end)

for i in range(n - 1):

start = 0

p = nx.shortest\_path(g, source=0, target=end)

end += 1

if len(p) in eccentricity:

eccentricity[len(p)].append(p)

else:

eccentricity[len(p)] = []

eccentricity[len(p)].append(p)

lens.append(len(p))

average\_len = sum(lens)/len(lens)

return average\_len

d = dict()

for n in data\_n:

p = 1

a = []

d[n] = list()

for u in range(10):

G = nx.fast\_gnp\_random\_graph(n, p)

#root = 0

#nx.write\_edgelist(G, 'edge\_list2.txt')

#G3 = nx.read\_edgelist('edge\_list2.txt')

#A = nx.adjacency\_matrix(G)

#start\_time = time.time()

#G4 = nx.Graph(A)

#nx.bfs\_edges(G4,0)

#start\_time = time.time()

#nx.write\_edgelist(G,'edge\_list2.txt')

#edges = nx.bfs\_edges(G,root)

#nx.bfs\_edges(G3,0)

#nodes = [root] + [v for u, v in edges]

time = time\_counter(G)

ewa = eccent(G)

a.append(time)

d[n].append(ewa)

print(a)

print(d)

t = np.sum(a)

print("sum = ", t)

print("average = ", t / 10)

data\_t.append(t / 10)

print(data\_t)

data\_to\_dataset = str(data\_t) + "\n" + str(data\_n) + "\n"

data\_path = str(d) + "\n"

with open("dataset.txt", "a") as file:

file.write(data\_to\_dataset)

with open("dataset\_ways.txt", "a") as file:

file.write(data\_path)

plot.plot(data\_t,data\_n)

plot.ylabel('Vertex')

plot.xlabel('Time')

plot.show()

**Файл draw\_graph.py для визуализации графов**

import networkx as nx

import time

import pylab as plt

import os

g = nx.Graph()

nodes = [0,1,2,3,4,5]

for i in nodes:

g.add\_node(i)

g.add\_edge(0, 1)

g.add\_edge(0, 2)

g.add\_edge(1, 3)

g.add\_edge(1, 4)

g.add\_edge(1, 5)

g.add\_edge(2, 6)

g.add\_edge(3, 7)

nx.draw(g, with\_labels=True)

plt.savefig("plot.png")

plt.show()

1. Алексеев В.Е., Таланов В.А. Графы и алгоритмы. Структуры данных. Модели вычислений, с. 7. [↑](#footnote-ref-1)
2. Алексеев В.Е., Таланов В.А. Графы и алгоритмы. Структуры данных. Модели вычислений, с. 43-44 [↑](#footnote-ref-2)
3. Алексеев В.Е., Таланов В.А. Графы и алгоритмы. Структуры данных. Модели вычислений, с. 44 [↑](#footnote-ref-3)
4. Алексеев В.Е., Таланов В.А. Графы и алгоритмы. Структуры данных. Модели вычислений, с. 45 [↑](#footnote-ref-4)