Министерство цифрового развития, связи и массовых коммуникаций

Российской Федерации

Уральский технический институт связи и информатики (филиал) ФГБОУ ВО «Сибирский государственный университет телекоммуникаций и информа-тики» в г. Екатеринбурге (УрТИСИ СибГУТИ)

КАФЕДРА

Информационные системы и технологии

(ИСиТ)

Курсовая работа

по дисциплине «Структуры и алгоритмы обработки данных»

Вариант 2

|  |  |
| --- | --- |
| Выполнил: | студент гр. ПЕ-12б  Цыганков В.С. |
| Проверил: | преподаватель кафедры ИСиТ  Белкина А.В. |

Екатеринбург, 2022

Министерство цифрового развития, связи и массовых коммуникаций

Российской Федерации

Уральский технический институт связи и информатики (филиал) ФГБОУ ВО «Сибирский государственный университет телекоммуникаций и информатики» в г. Екатеринбурге (УрТИСИ СибГУТИ)

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

на курсовую работу

по дисциплине «Структуры и алгоритмы обработки данных»

студента группы ПЕ-12б Цыганкова Валерия Сергеевича

Вариант 2

ИСХОДНЫЕ ДАННЫЕ

1. Написать программу на языке Паскаль, C++ или C# реализующую алгоритмы на графах.

* Определить самый короткий цикл в графе
* Выполнить обход графа в глубину
* Определить кратчайший путь между всеми парами вершин
* Построить минимальное остовное дерево с помощью алгоритма Прима.

2. Входной информацией для программы будет ориентированный граф с числом вершин не более шести, каждое ребро которого имеет определенный неотрицательный вес. Выбрать способ ввода входной информации.

Дата выдачи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Срок сдачи: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Допуск к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель преподаватель \_\_\_\_\_Белкина А.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отзыв руководителя

1 Заключение о степени соответствия выполненной работы техническому заданию

2 Качество произведенных расчетов

3 Качество оформления пояснительной записки, графического материала, наличие ссылок на первоисточники

4 Заключение о возможности допуска курсового проекта (работы) к защите

[Введение 5](#_Toc121303506)

[1. Теоретическая часть. 7](#_Toc121303507)

[1.1 Основные понятия 7](#_Toc121303508)

[1.2 Способы представления графов 8](#_Toc121303509)

[1.3 Определение самого короткого цикла в графе 9](#_Toc121303510)

[1.4 Обход графа в глубину 9](#_Toc121303511)

[1.5 Определение кратчайшего пути между всеми парами вершин 10](#_Toc121303512)

[1.6 Построение минимального остовного дерева с помощью алгоритма Прима. 11](#_Toc121303513)

[2. Практическая часть. 12](#_Toc121303514)

[2.1 Исходные данные 12](#_Toc121303515)

[2.2 Структура данных 12](#_Toc121303516)

[2.3 Результат выполнения 13](#_Toc121303517)

[Заключение 20](#_Toc121303518)

[Список литературы 21](#_Toc121303519)

[Приложения 1 – листинг программы 22](#_Toc121303520)

# Введение

**Существуют много различных подходов к определению графа. Отдельные авторы при определении графа исключают возможность введения нескольких ребер, соединяющих одни и те же вершины. Другие исключают наличие петель т.е. ребер, соединяющих вершину саму с собой, а потом, чтобы определить графы, вводятся специальные термины типа мультиграф и т.п. Таким образом, в настоящее время все еще не сложилось единого стандарта терминологии теории графов.**

**Одним из важнейших разделов дискретной математики является теория графов. С их помощью описывается сложное строение физических, химических, социальных и т.д. объектов. Графы широко используется в градостроительстве при проектировании сетей водо-, газо-, тепло-, электроснабжения.**

**Датой рождения этой теории можно считать 1736 год, когда была опубликована статья Леонарда Эйлера, посвященная решению головоломки под названием «Задача о кёнигсберских мостах». Долгое время методы, аналогичные эйлеровым использовались для исследования подобных развлекательных задач. Но в XIX веке А.Кэли нашел им более достойное определение. С помощью графов он стал описывать химические «цепи».**

**Существующее название закрепилось за этой наукой с 1936 года, после выхода в свет монографии венгерского математика Д.Кёнига. Термин «граф» происходил от греческого слова «пишу». Он говорит о наглядной графической интерпретации, во-первых, основных понятий этой теории, во - вторых она тесно связана с геометрией. И действительно, этот раздел дискретной математики находится на стыке геометрии, топологии, комбинаторики и ряда других математических дисциплин и интенсивно использует их методы.**

**Графы нашли применение практически во всех отраслях научных знаний: физике, биологии, химии, математике, истории, социальных науках, технике и т.п. Наибольшей популярностью теоретико-графовые модели используются при исследовании коммуникационных сетей, систем информатики, химических и генетических структур, электрических цепей и других систем сетевой структуры.**

**Развитие теории графов в основном обязано большому числу всевозможных приложений. По-видимому, из всех математических объектов графы занимают одно из первых мест в качестве формальных моделей реальных систем. Теория графов, которая за последние десять – двадцать лет вступила в новый период интенсивных разработок. В этом процессе явно заметно влияние запросов новых областей: теории игр и программирования, теории передачи сообщений, электрических сетей и контактных цепей, а также проблем психологии и биологии.**

**Целью курсовой работы является реализация графов и операций над ними.**

**Задачи:**

* **изучить основные понятия теории графов;**
* **изучить алгоритмы на графах (обход в глубину и ширину, поиск кратчайшего пути между вершинами и т.п.);**
* **написать программу, реализующую некоторые алгоритмы на графах;**
* **провести тестирование программы.**

# 1. Теоретическая часть.

## 1.1 Основные понятия

Вершина - точка в графе, отдельный объект, для топологической модели графа не имеет значения координата вершины, её расположение, цвет, вкус, размер; однако при решении некоторых задачах вершины могут раскрашиваться в разные цвета или сохранять числовые значения.

Ребро - неупорядоченная пара двух вершин, которые связаны друг с другом. Эти вершины называются концевыми точками или концами ребра. При этом важен сам факт наличия связи, каким именно образом осуществляется эта связь и по какой дороге - не имеет значения; однако рёбра может быть присвоен “вес”, что позволит говорить о “нагруженном графе” и решать задачи оптимизации.

Инцидентность - вершина и ребро называются инцидентными, если вершина является для этого ребра концевой. Обратите внимание, что термин “инцидентность” применим только к вершине и ребру.

Смежность вершин - две вершины называются смежными, если они инцидентны одному ребру.

Смежность рёбер - два ребра называются смежными, если они инцидентны одной вершине.

Петля - ребро, инцидентное одной вершине. Ребро, которое замыкается на одной вершине.

Степень вершины - количество рёбер, инцидентных указанной вершине. По-другому - количество рёбер, исходящих из вершины. Петля увеличивает степень вершины на 2.

Изолированная вершина - вершина с нулевой степенью.

Висячая вершина - вершина со степенью 1.

Полный граф - граф, в котором каждые две вершины соединены одним ребром. Количество ребер в полном графе: N \* (N - 1) / 2.

Регулярный граф - граф, в котором степени всех вершин одинаковые.

Путь или Маршрут - последовательность смежных рёбер. Обычно путь задаётся перечислением вершин, по которым он пролегает.

Длина пути - количество рёбер в пути.

Цикл или Контур - цепь, в котором последняя вершина совпадает с первой.

Длина цикла - количество рёбер в цикле.

Взвешенный граф - граф, в котором у каждого ребра и/или каждой вершины есть “вес” - некоторое число, которое может обозначать длину пути, его стоимость и т. п. Для взвешенного графа составляются различные алгоритмы оптимизации, например поиск кратчайшего пути.

Связный граф - граф, в котором существует путь между любыми двумя вершинами.

Ориентированный граф или Орграф - граф, в котором рёбра имеют направления. Дуга - направленные рёбра в ориентированном графе.

## 1.2 Способы представления графов

Граф (G) — это совокупность вершин (V), и дуг (E), в зависимости от того, как они задаются, выделяются следующие способы машинного представле-ния графа:

матрица смежности для графа из N вершин хранится в виду двумерного массива размером N \* N. Вершины графа в этом случае задаются номерами (индексами строк и столбцов матрицы), а ячейка графа matrix[i, j] отражает наличие дуги между соответствующими вершинами. Например, при наличии дуги в ячейке может быть записана единица (или вес ребра i->j для взвешенно-го графа) , а при отсутствии — ноль;

матрица инцидентности для графа из N вершин и M дуг хранится в виде двумерного массива размером N \* M. Ячейка матрицы matrix[i, j] отражает инцидентность ребра j вершине i, т.е. тот факт, что это ребро выходит или входит в вершину i. Если ребро не связано с вершиной — в соответствующей ячейке матрицы записывается ноль, в противном случае единица (если граф ориентированный, то начало ребра можно отметить -1, а конец 1, если граф взвешенный — единица может быть заменена весом соответствующего ребра).

Работая с приведенными матрицами возможно, например, найти в графе кратчайшие пути между вершинами, построить минимальные остовы и т.д., однако, у них есть недостатки:

избыточность. Зачастую в графах ребра существуют между небольшим (количеством вершин), поэтому в матрице смежности будет огромное количество нулей. В матрице инцидентности в каждом столбце может быть лишь два ненулевых значения (т.к. у дуги два конца). На хранение нулей тратится память, что может быть существенно при обработки больших графов;

недостаточная расширяемость. В матрицу смежности можно без проблем добавлять новые дуги, но, чтобы добавить вершину нужно создавать новую матрицу большего размера и копировать в нее данные из старой. Это работает очень медленно при больших матрицах. В матрице инцидентности такие проблемы возникнут как при добавлении дуг, так и при добавлении вершин.

Списки смежности

Данный способ представления больше подходит для разреженных графов, то есть графов у которых количество рёбер гораздо меньше чем количество вершин в квадрате (|E| << |V|2).

В данном представлении используется массив Adj содержащий |V| списков. В каждом списке Adj[v] содержатся все вершины u, так что между v и u есть ребро. Память, требуемая для представления равна O (|E| + |V|) что является лучшим показателем чем матрица смежности для разреженных графов.

Главный недостаток этого способа представления в том, что нет быстрого способа проверить существует ли ребро (u, v).

## 1.3 Определение самого короткого цикла в графе

Циклами графа называют замкнутые обходы без повторного прохода по ребру или посещения вершины дважды, за исключением начальной и конечной вершин. Циклы можно описать набором рёбер.

Для решения данной задачи можно воспользоваться модифицированным алгоритмом Флойда-Уоршелла, алгоритм нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе, подробнее о нём поговорим дальше.

В нашем случае graph[i][i] мы будем считать не 0, а также, как и все остальные недоступные вершины - INF, тогда мы с лёгкостью получим самые короткие циклы из каждой вершины, останется лишь выбрать минимальную из них.

## 1.4 Обход графа в глубину

Обход в глубину (поиск в глубину, англ. Depth-First Search, DFS) — один из основных методов обхода графа, часто используемый для проверки связности, поиска цикла и компонент сильной связности и для топологической сортировки.

Общая идея

Общая идея алгоритма состоит в следующем: для каждой не пройденной вершины необходимо найти все не пройденные смежные вершины и повторить поиск для них.

Как понятно из названия, суть алгоритма заключается в том, что мы максимально углубляемся в граф, и потом уже возвращаемся.

Пошаговое представление:

1. Выбираем любую вершину из еще не пройденных, обозначим ее как u.
2. Запускаем процедуру dfs(u)

* Помечаем вершину u как пройденную
* Для каждой не пройденной смежной с u вершиной (назовем ее v) запускаем dfs(v)

1. Повторяем шаги 1 и 2, пока все вершины не окажутся пройденными.

## 1.5 Определение кратчайшего пути между всеми парами вершин

Для определения кратчайших путей между всеми парами вершин используется вышеупомянутый алгоритм Флойда-Уоршелла.

Алгоритм Флойда – Уоршелла – динамический алгоритм вычисления значений кратчайших путей для каждой из вершин графа. Метод работает на взвешенных графах, с положительными и отрицательными весами ребер, но без отрицательных циклов, являясь, таким образом, более общим в сравнении с алгоритмом Дейкстры, т. к. последний не работает с отрицательными весами ребер, и к тому же классическая его реализация подразумевает определение оптимальных расстояний от одной вершины до всех остальных.

Для реализации алгоритма Флойда – Уоршелла сформируем матрицу смежности D[][] графа G=(V, E), в котором каждая вершина пронумерована от 1 до |V|. Эта матрица имеет размер |V|´|V|, и каждому ее элементу D[i][j] присвоен вес ребра, идущего из вершины i в вершину j. По мере выполнения алгоритма, данная матрица будет перезаписываться: в каждую из ее ячеек внесется значение, определяющее оптимальную длину пути из вершины i в вершину j (отказ от выделения специального массива для этой цели сохранит память и время).

Теперь, перед составлением основной части алгоритма, необходимо разобраться с содержанием матрицы кратчайших путей. Поскольку каждый ее элемент D[i][j] должен содержать наименьший из имеющихся маршрутов, то сразу можно сказать, что для единичной вершины он равен нулю, даже если она имеет петлю (отрицательные циклы не рассматриваются), следовательно, все элементы главной диагонали (D[i][i]) нужно обнулить.

А чтобы нулевые недиагональные элементы (матрица смежности могла иметь нули в тех местах, где нет непосредственного ребра между вершинами i и j) сменили по возможности свое значение, определим их равными бесконечности, которая в программе может являться, например, максимально возможной длинной пути в графе, либо просто – большим числом.

## 1.6 Построение минимального остовного дерева с помощью алгоритма Прима.

Остовное дерево - ациклический связный подграф данного связного неориентированного графа, в который входят все его вершины. Можно сказать, что остовное дерево состоит из некоторого подмножества рёбер графа, таких, что из любой вершины графа можно попасть в любую другую вершину, двигаясь по этим рёбрам, и в нём нет циклов, то есть из любой вершины нельзя попасть в саму себя, не пройдя какое-то ребро дважды. Минимальное остовное дерево (или минимальное покрывающее дерево) в связанном взвешенном неориентированном графе - остовное дерево этого графа, имеющее минимальный возможный вес, где под весом дерева понимается сумма весов, входящих в него рёбер.

Этот алгоритм назван в честь американского математика Роберта Прима (Robert Prim), который открыл этот алгоритм в 1957 г. Впрочем, ещё в 1930 г. этот алгоритм был открыт чешским математиком Войтеком Ярником (Vojtěch Jarník). Кроме того, Эдгар Дейкстра (Edsger Dijkstra) в 1959 г. также изобрёл этот алгоритм, независимо от них.

Описание алгоритма

Сам алгоритм имеет очень простой вид. Искомый минимальный остов строится постепенно, добавлением в него рёбер по одному. Изначально остов полагается состоящим из единственной вершины (её можно выбрать произвольно). Затем выбирается ребро минимального веса, исходящее из этой вершины, и добавляется в минимальный остов. После этого остов содержит уже две вершины, и теперь ищется и добавляется ребро минимального веса, имеющее один конец в одной из двух выбранных вершин, а другой — наоборот, во всех остальных, кроме этих двух. И так далее, т.е. всякий раз ищется минимальное по весу ребро, один конец которого — уже взятая в остов вершина, а другой конец — ещё не взятая, и это ребро добавляется в остов (если таких рёбер несколько, можно взять любое). Этот процесс повторяется до тех пор, пока остов не станет содержать все вершины (или, что то же самое, n-1 ребро).

В итоге будет построен остов, являющийся минимальным. Если граф был изначально не связен, то остов найден не будет (количество выбранных рёбер останется меньше n-1).

# 2. Практическая часть.

## 2.1 Исходные данные

Исходными данными является ориентированный граф. В графе 7 вершин. Исходный граф:

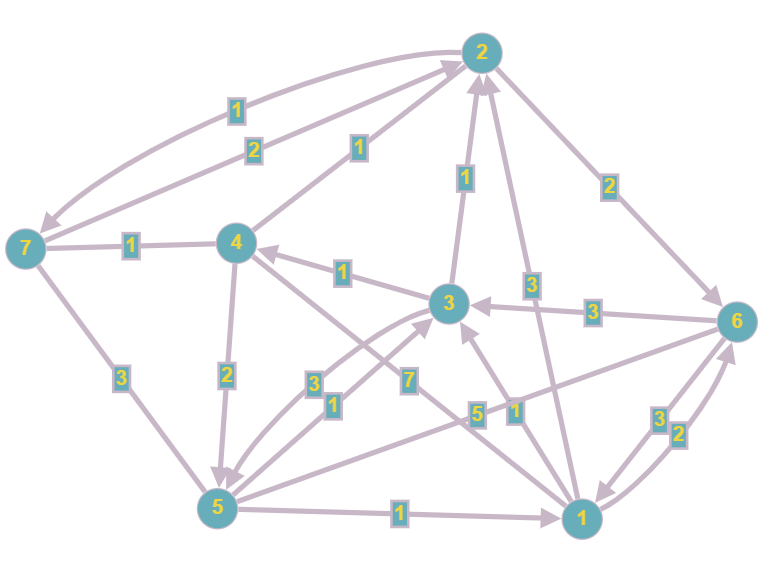


Рисунок 1 - Исходный граф

Для представления графа в программе используется двумерный массив (матрица смежности):

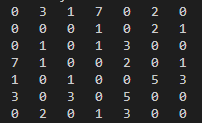


Рисунок 2 - Матрица смежности

## 2.2 Структура данных

Для хранения данных в программе используется массив graph размером n\*n, определенный как целочисленный двумерный массив, где n – количество вершин.

## 2.3 Результат выполнения

В шапке программы подключены базовые библиотеки языка C++ и описаны некоторые глобальные константы, такие как наш граф, INF, и n (количество вершин).

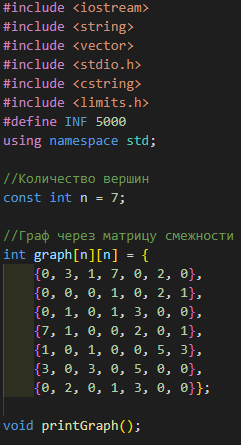


Рисунок 3 – Шапка программы

Также в этой шапке определены все функции (и их переменные), которые будут переопределены в коде уже после основной функции.

Для взаимодействия в программе реализовано меню, в нём можно вызвать все базовые операции с графом:

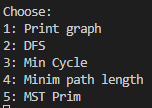


Рисунок 4 – Меню

Меню реализовано с помощью конструкции switch case, которая представляет собой множественный if, где в каждом значении case проверяется полученное(введённое) значение.

Вывод графа реализован с помощью функции printGraph():

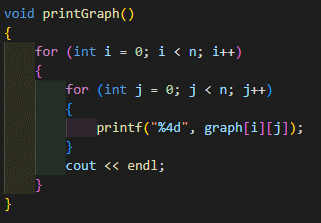


Рисунок 5 - Вывод графа

Для поиска в глубину была создана функция DFS и visitedFalse, первая соответственно реализует алгоритм обхода, а вторая обнуляет массив visited, который является ключевым в работе DFS.

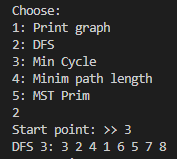


Рисунок 6 – Результат DFS

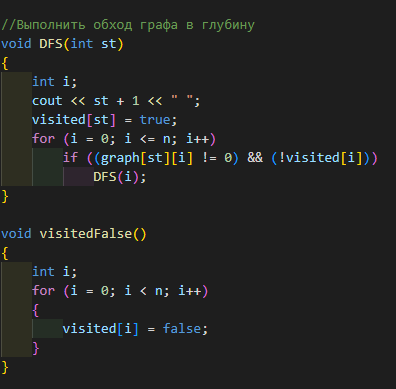


Рисунок 7 - код DFS и visitedFalse

Чтобы найти кратчайшие циклы была создана функция, выполняющая алгоритм Флойда-Маршалла, в ней же происходить запоминания пути цикла и вывод.

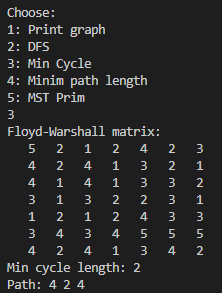


Рисунок 8 - Результат поиска минимального цикла

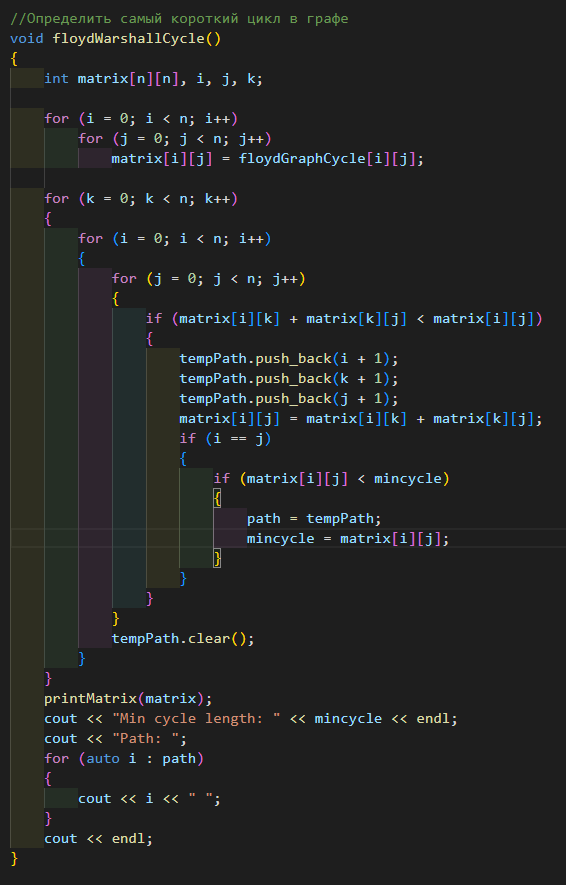


Рисунок 9 – Код функции для поиска кратчайшего цикла

Также для корректной работы алгоритма Флойда-Маршалла граф нужно видоизменить, в данном случае все 0 поменять на INF, за это отвечает функция toFloydCycle, а также для упрощения вывода матрицы функция printMatrix

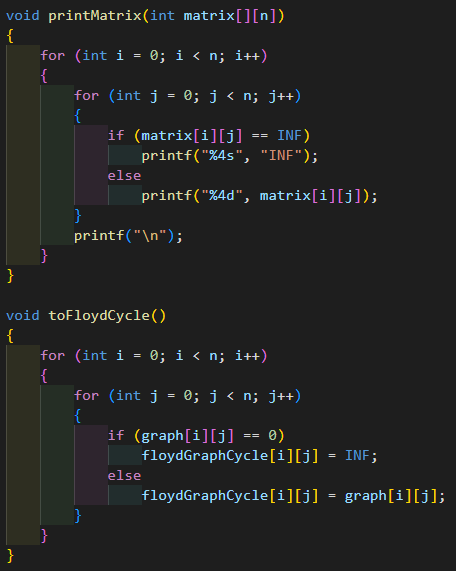


Рисунок 10 - Код для создания нужного вида графа и вывода матрицы

Для поиска кратчайших путей также был применён алгоритм Флойда-Маршалла. В отличии от прошлой реализации вершины graph[i][i] мы будем оставлять нулями, а также нам не нужен будет вывод пути кратчайшего пути. Функцию printMatrix менять нет смысла, так как вид матрицы остаётся таким же.

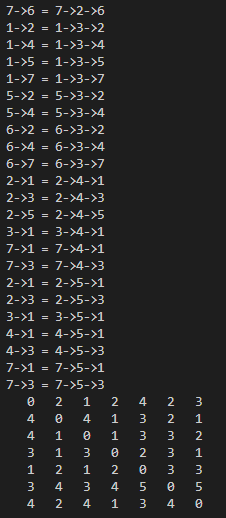


Рисунок 11 - Результат работы алгоритма Флойда-Маршалла

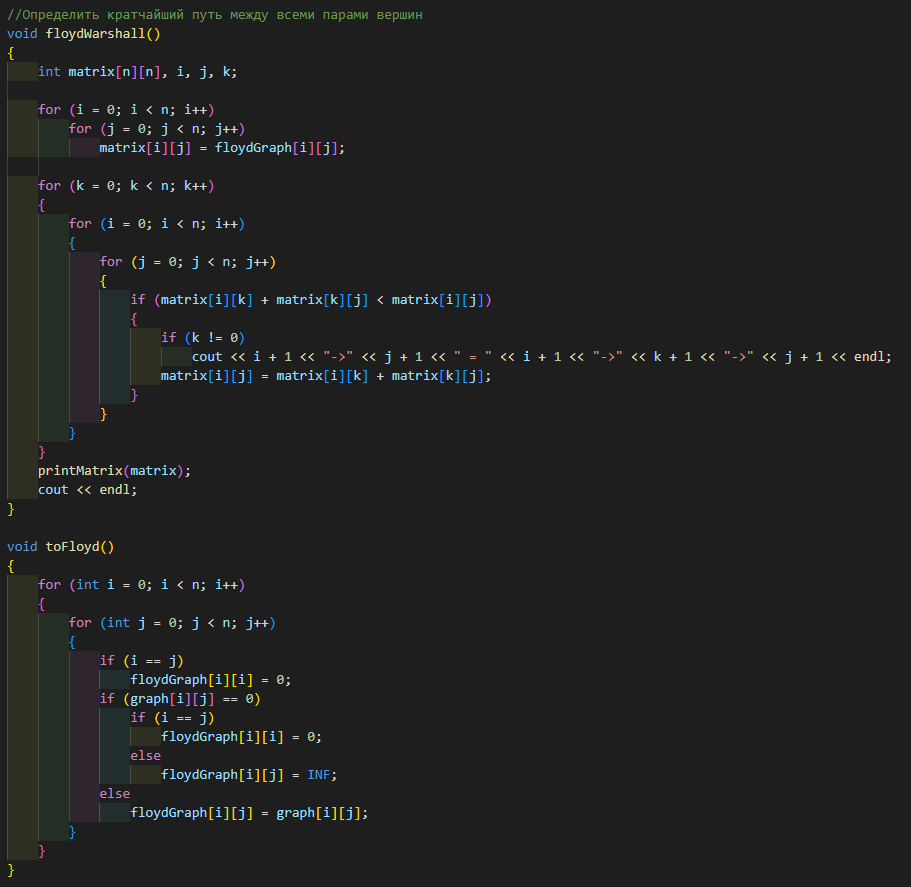


Рисунок 12 - Код алгоритма Флойда-Маршалла

Нахождение минимального остовного дерева выполнено с помощью функции primMST().

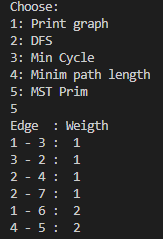


Рисунок 13 - МОД с помощью алгоритма Прима

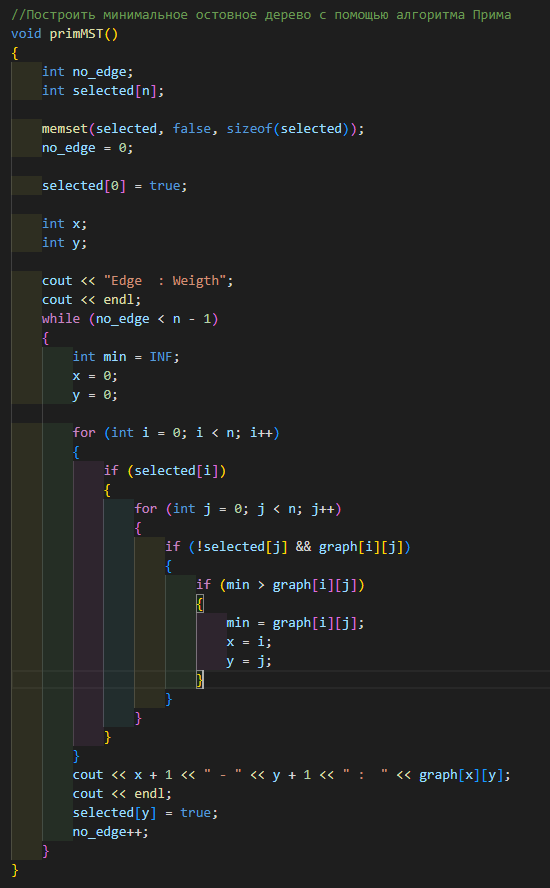


Рисунок 14 - Код для построения МОД с помощью алгоритма Прима

# Заключение

В ходе выполнения курсовой работы были изучены и реализованы графы, реализация осуществлена с помощью языка С++. Также на графах были использованы алгоритмы: для обхода графа в глубину, поиска кратчайшего цикла, определения кратчайших путей между всеми парами вершин и построение минимального остовного дерева с помощью алгоритма Прима.

На протяжении более сотни лет развитие теории графов определялось в основном проблемой четырёх красок. Решение этой задачи в 1976 году оказалось поворотным моментом истории теории графов, после которого произошло её развитие как основы современной прикладной математики. Универсальность графов незаменима при проектировании и анализе коммуникационных сетей.

Теория графов, как математическое орудие, приложима как к наукам о поведении (теории информации, кибернетике, теории игр, теории систем, транспортным сетям), так и к чисто абстрактным дисциплинам (теории множеств, теории матриц, теории групп и так далее).

За последние же годы графы стали одним из самых развивающихся разделов математики. Для примера возьмём карту метро или электричек, все они представляются набором цветных линий (направлений маршрутов) и точек на пересечении этих линий (станций или вокзалов). И всё это возможно благодаря развитию графов.

# Список литературы

1. Макоха А. Н., Сахнюк П. А., Червяков Н. И. Дискретная математика: Учеб. пособие. — М. ФИЗМАТЛИТ, 2005 — 368 с.
2. Дж. Макконелл Анализ алгоритмов. Активный обучающий подход. — 3-е дополненное издание. М: Техносфера, 2009. -416с.
3. Дубяго, К. Д. Графы – многофункциональный инструмент любого человека / К. Д. Дубяго, А. А. Белозеров, Е. А. Матвеева, В. В. Краснова, 2016. — № 6.1 (9.1). — С. 26-29.
4. Ойстин Оре. Теория графов — 1968. 352 с.
5. Ховрина Вера Владимировна, Гуровиц Владимир Михайлович. Графы. — МЦНМО, 2019, — 32 с.

# Приложения 1 – листинг программы

#include <iostream>

#include <string>

#include <vector>

#include <stdio.h>

#include <cstring>

#include <limits.h>

#define INF 5000

using namespace std;

//Количество вершин

const int n = 7;

//Граф через матрицу смежности

int graph[n][n] = {

    {0, 3, 1, 7, 0, 2, 0},

    {0, 0, 0, 1, 0, 2, 1},

    {0, 1, 0, 1, 3, 0, 0},

    {7, 1, 0, 0, 2, 0, 1},

    {1, 0, 1, 0, 0, 5, 3},

    {3, 0, 3, 0, 5, 0, 0},

    {0, 2, 0, 1, 3, 0, 0}};

void printGraph();

// DFS

bool \*visited = new bool[n];

void DFS(int st);

void visitedFalse();

// FLOYD-WARSHALL MIN CYCLE

int floydGraphCycle[n][n];

vector<int> tempPath, path;

int mincycle = INF;

void toFloydCycle();

void floydWarshallCycle();

void printMatrix(int matrix[][n]);

// FLOYD-WARSHALL

int floydGraph[n][n];

void toFloyd();

void floydWarshall();

// PRIM

void primMST();

int main()

{

    int i, j, start;

    printGraph();

    visitedFalse();

    int loop = 1, command = 0;

    while (loop == 1)

    {

        cout << "Choose: " << endl;

        cout << "1: Print graph" << endl;

        cout << "2: DFS" << endl;

        cout << "3: Min Cycle" << endl;

        cout << "4: Minim path length" << endl;

        cout << "5: MST Prim" << endl;

        cin >> command;

        switch (command)

        {

        case 1:

            printGraph();

            break;

        case 2:

            cout << "Start point: >> ";

            cin >> start;

            cout << "DFS " << start << ": ";

            DFS(start - 1);

            cout << endl;

            break;

        case 3:

            cout << "Floyd-Warshall matrix: " << endl;

            toFloydCycle();

            floydWarshallCycle();

            break;

        case 4:

            cout << "Floyd-Warshall matrix: " << endl;

            toFloyd();

            floydWarshall();

            break;

        case 5:

            primMST();

        default:

            break;

        }

        cout << "1 - continue" << endl;

        cin >> loop;

        cout << "---------------------" << endl;

    }

    delete[] visited;

    return 0;

}

void printGraph()

{

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            printf("%4d", graph[i][j]);

        }

        cout << endl;

    }

}

//Выполнить обход графа в глубину

void DFS(int st)

{

    int i;

    cout << st + 1 << " ";

    visited[st] = true;

    for (i = 0; i <= n; i++)

        if ((graph[st][i] != 0) && (!visited[i]))

            DFS(i);

}

void visitedFalse()

{

    int i;

    for (i = 0; i < n; i++)

    {

        visited[i] = false;

    }

}

//Определить самый короткий цикл в графе

void floydWarshallCycle()

{

    int matrix[n][n], i, j, k;

    for (i = 0; i < n; i++)

        for (j = 0; j < n; j++)

            matrix[i][j] = floydGraphCycle[i][j];

    for (k = 0; k < n; k++)

    {

        for (i = 0; i < n; i++)

        {

            for (j = 0; j < n; j++)

            {

                if (matrix[i][k] + matrix[k][j] < matrix[i][j])

                {

                    tempPath.push\_back(i + 1);

                    tempPath.push\_back(k + 1);

                    tempPath.push\_back(j + 1);

                    matrix[i][j] = matrix[i][k] + matrix[k][j];

                    if (i == j)

                    {

                        if (matrix[i][j] < mincycle)

                        {

                            path = tempPath;

                            mincycle = matrix[i][j];

                        }

                    }

                }

            }

            tempPath.clear();

        }

    }

    printMatrix(matrix);

    cout << "Min cycle length: " << mincycle << endl;

    cout << "Path: ";

    for (auto i : path)

    {

        cout << i << " ";

    }

    cout << endl;

}

void printMatrix(int matrix[][n])

{

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (matrix[i][j] == INF)

                printf("%4s", "INF");

            else

                printf("%4d", matrix[i][j]);

        }

        printf("\n");

    }

}

void toFloydCycle()

{

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (graph[i][j] == 0)

                floydGraphCycle[i][j] = INF;

            else

                floydGraphCycle[i][j] = graph[i][j];

        }

    }

}

//Определить кратчайший путь между всеми парами вершин

void floydWarshall()

{

    int matrix[n][n], i, j, k;

    for (i = 0; i < n; i++)

        for (j = 0; j < n; j++)

            matrix[i][j] = floydGraph[i][j];

    for (k = 0; k < n; k++)

    {

        for (i = 0; i < n; i++)

        {

            for (j = 0; j < n; j++)

            {

                if (matrix[i][k] + matrix[k][j] < matrix[i][j])

                {

                    if (k != 0)

                        cout << i + 1 << "->" << j + 1 << " = " << i + 1 << "->" << k + 1 << "->" << j + 1 << endl;

                    matrix[i][j] = matrix[i][k] + matrix[k][j];

                }

            }

        }

    }

    printMatrix(matrix);

    cout << endl;

}

void toFloyd()

{

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (i == j)

                floydGraph[i][i] = 0;

            if (graph[i][j] == 0)

                if (i == j)

                    floydGraph[i][i] = 0;

                else

                    floydGraph[i][j] = INF;

            else

                floydGraph[i][j] = graph[i][j];

        }

    }

}

//Построить минимальное остовное дерево с помощью алгоритма Прима

void primMST()

{

    int no\_edge;

    int selected[n];

    memset(selected, false, sizeof(selected));

    no\_edge = 0;

    selected[0] = true;

    int x;

    int y;

    cout << "Edge  : Weigth";

    cout << endl;

    while (no\_edge < n - 1)

    {

        int min = INF;

        x = 0;

        y = 0;

        for (int i = 0; i < n; i++)

        {

            if (selected[i])

            {

                for (int j = 0; j < n; j++)

                {

                    if (!selected[j] && graph[i][j])

                    {

                        if (min > graph[i][j])

                        {

                            min = graph[i][j];

                            x = i;

                            y = j;

                        }

                    }

                }

            }

        }

        cout << x + 1 << " - " << y + 1 << " :  " << graph[x][y];

        cout << endl;

        selected[y] = true;

        no\_edge++;

    }

}