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

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовой работе на тему:**

«Решение графовых задач. Нахождение остовного дерева.»

Студент группы ИУ5-34М  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. Р. Юсипов**

(Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю.Е. Гапанюк**

(Подпись, дата) (И.О.Фамилия)

Москва, 2021

**Оглавление**

[**Введение** 3](#_Toc89616467)

[1. Основные понятия 4](#_Toc89616468)

[3. Нахождение кратчайших путей в графе 8](#_Toc89616469)

[Алгоритм Дейкстры 9](#_Toc89616470)

[Схема алгоритма Дейкстры 9](#_Toc89616471)

[4. Нахождение остовного дерева минимального веса 11](#_Toc89616472)

[Алгоритм Прима-Краскала 12](#_Toc89616473)

[5. Алгоритм Прима-Краскала. Решение задач. 14](#_Toc89616474)

[6. Листинг программы 16](#_Toc89616475)

[Заключение 18](#_Toc89616476)

[Список литературы 19](#_Toc89616477)

**Введение**

Теория графов – это область дискретной математики, особенностью которой является геометрический подход к изучению объектов. Теория графов находится сейчас в самом расцвете. Обычно её относят к топологии (во многих случаях рассматриваются лишь топологические свойства графов), однако она пересекается со многими разделами теории множеств, комбинаторной математики, алгебры, геометрии, теории матриц, теории игр, математической логики и многих других математических дисциплин. Основной объект теории графов-граф и его обобщения. Первые задачи теории графов были связаны с решением математических развлекательных задач и головоломок. Одним из первых результатов в теории графов явился критерий существования обхода всех ребер графа без повторений, полученный Л. Эйлером при решении задачи о Кенигсбергских мостах. Существует еще один вид задач, связанных с путешествиями вдоль графов. Речь идёт о задачах, в которых требуется отыскать путь, проходящий через все вершины, причем не более одного раза через каждую. Цикл, проходящий через каждую вершину один и только один раз, носит название гамильтоновой линии. К сожалению, пока еще не найден общий критерий, с помощью которого можно было бы решить, является ли данный граф гамильтоновым, и если да, то найти на нём все гамильтоновы линии. Использование ЭВМ позволило решать возникающие на практике конкретные задачи, связанные с большим объемом вычислений, прежде не поддававшиеся решению. Для ряда экстремальных задач теории графов были разработаны методы их решения, например, один из таких методов позволяет решать задачи о построении максимального потока через сеть. Для отдельных классов графов (деревья, плоские графы и т. д.), которые изучались и ранее, было показано, что решения некоторых задач для графов из этих классов находятся проще, чем для произвольных графов. В теории графов существуют специфические методы решения экстремальных задач. Один из них основан на теореме о максимальном потоке и минимальном разрезе, утверждающей, что максимальный поток, который можно пропустить через сеть из вершины U в вершину V, равен минимальной пропускной способности разрезов, разделяющих вершины U и V. Были построены различные эффективные алгоритмы нахождения максимального потока. Большое значение в теории графов имеют алгоритмические вопросы. Для конечных графов, т. е. для графов с конечным множеством вершин и ребер, как правило, проблема существования алгоритма решения задач, в том числе экстремальных, решается положительно. Решение многих задач, связанных с конечными графами, может быть выполнено с помощью полного перебора всех допустимых вариантов. Однако таким способом удается решить задачу только для графов с небольшим числом вершин и ребер. Поэтому существенное значение для теории графов имеет построение эффективных алгоритмов, находящих точное или приближенное решение. Для некоторых задач такие алгоритмы построены, например, для установления планарности графов, определения изоморфизма деревьев, нахождения максимального потока. Результаты и методы теории графов применяются при решении транспортных задач о перевозках, для нахождения оптимальных решений задачи о назначениях, для выделения «узких мест» при планировании и управлении разработок проектов, при составлении оптимальных маршрутов доставки грузов, а также при моделировании сложных технология, процессов, в построении различных дискретных устройств, в программировании и т. д. За последние десять – двадцать лет теория графов 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.

# 1. Основные понятия

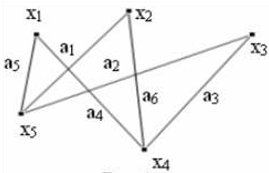


Рис. 1. Граф G.

**Граф** G (рис.1) задается множеством точек (вершин) х1 , х2 ,..., хn . (которое обозначается через Х) и множеством линий (ребер) а1 , а2 ,...,аm . (которое обозначается символом А), соединяющих между собой все или часть этих точек. Таким образом, граф G полностью задается (и обозначается) парой (Х, А).

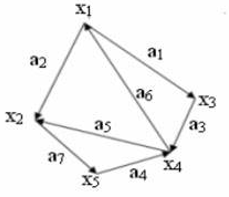


Рис. 2. Ориентированный граф.

Если ребра из множества А ориентированы, что обычно показывается стрелкой, то они называются дугами, и граф с такими ребрами называется **ориентированным** графом, (рис.2).

Например, если дорога имеет не двух-, а одностороннее движение то направление этого движения будет показано стрелкой.

Если ребра не имеют ориентации, то граф называется **неориентированным** (рис 3).

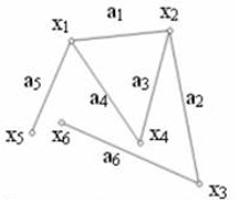


Рис. 3. Неориентированный граф.

В ориентированном графе дуга обозначается упорядоченной парой, состоящей из начальной и конечной вершин (т.е. двумя концевыми вершинами дуги), ее направление предполагается заданным от первой вершины ко второй. Так, например, на рис. 2 обозначение (х1 ,х3 ) относится к дуге а1 .

Другое, употребляемое чаще описание ориентированного графа G состоит в задании множества вершин Х и соответствия Г, которое показывает, как между собой связаны вершины. Соответствие Г называется отображением множества Х в Х, а граф в этом случае обозначается парой G=(Х, Г). Так, например, для графа на рис. 2:

Г (х1 )={х2 , х3 }, т.е. вершины х2 и х3 являются конечными вершинами дуг, у которых начальной вершиной является вершина х1 .

На рис. 3: Г (х1 )={х2 , х4 , х5 }, (неориентированное ребро представляется как две противоположно направленные дуги, соединяющие те же самые вершины.)

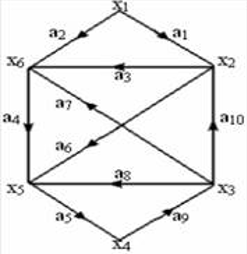


Рис. 4. Пример графа.

**Путем** (или ориентированным маршрутом) ориентированного графа называется последовательность дуг, в которой конечная вершина всякой дуги, отличной от последней, является начальной вершиной следующей.

Так, на рис. 4 последовательности дуг

а6 , а5 , а9 , а8 , а4 . (1)

а1 , а6 , а5 , а9 . (2)

а1 , а6 , а5 , а9 , а10 , а6 , а4 . (3)

являются путями.

**Ориентированной цепью** (орцепью) называется такой путь, в котором каждая дуга используется не больше одного раза. Так, например приведенные выше пути (1) и (2) являются орцепями, а путь (3) не является таким, т.к. дуга а6 в нем используется дважды.

**Простой** орцепью называется такой путь, в котором каждая вершина используется не более одного раза. Например, путь (2) является простой орцепью, а пути (1) и (3) – нет.

**Маршрут** есть неориентированный “двойник” пути, и это понятие рассматривается в тех случаях, когда можно пренебречь направленностью дуг в графе. Таким образом, маршрут есть последовательность ребер а1 , а2 ,..., аq , в которой каждое ребро аq за исключением первого и последнего ребер, связано с ребрами аi-1 и аi+1 своими концевыми вершинами. Последовательности дуг:

а2 , а4 , а8 , а10 , (4)

а2 , а7 , а8 , а4 , а3 , (5)

а10 , а7 , а4 , а8 , а7 , а2 . (6)

в графе, изображенном на рис. 4, являются маршрутами; две точки над символом дуги означают, что ее ориентацией пренебрегают, т.е. дуга рассматривается как неориентированное ребро. Также путь или маршрут можно изображать последовательностью вершин. Например, путь (1) будет выглядеть следующем образом: х2 , х5 , х4 , х3 , х5 , х6 .

Иногда дугам графа приписываются числа, называемые весом, стоимостью, или длиной этой дуги. В этом случае граф называется **графом со взвешенными дугами**. А если вес приписывается вершинам графа, то тогда получается **граф со взвешенными вершинами**. Если в графе веса приписаны и дугам и вершинам, то он называется просто **взвешенным**.

Связный граф без циклов называется **деревом**. Деревья особенно часто возникают на практике при изображении различных иерархий. Например, популярные генеалогические деревья. Пример – генеалогическое дерево. Эквивалентные определения дерева.

1. Связный граф называется деревом, если он не имеет циклов.   
2. Содержит N-1 ребро и не имеет циклов.   
3. Связный и содержит N-1 ребро.   
4. Связный и удаление одного любого ребра делает его несвязным.  
5. Любая пара вершин соединяется единственной цепью.   
6. Не имеет циклов и добавление одного ребра между любыми двумя вершинами приводит к появлению одного и только одного цикла.

1. Геометрический:

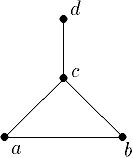
****

Рис. 5. Геометрический способ задания графа.

1. Матрицей смежности:

|  |  |  |  |
| --- | --- | --- | --- |
| A | В | C | D |
| A | | 0 | 1 | 1 | 0 |
| B | | 1 | 0 | 1 | 0 |
| C | | 1 | 1 | 0 | 1 |
| D | | 0 | 0 | 1 | 0 |

**Матрица смежности** – квадратная матрица, размерности, равной количеству вершин. При этом а[ i, j ] – целое число, равное количеству рёбер, связывающих i-ю, j-ю вершину. Если в графе нет петель, то диагональные элементы равны 0. Если рёбра не повторяются, то все элементы 0 или 1. Если граф неориентированный, то матрица симметрична.

1. Матрица инцидентности:

|  |  |  |  |
| --- | --- | --- | --- |
| А | В | С | D |
| A | | 1 | 1 | 0 | 0 |
| B | | 0 | 1 | 1 | 0 |
| C | | 1 | 0 | 1 | 0 |
| D | | 0 | 0 | 1 | 1 |

4. Явное задание графа как алгебраической системы: <{a,b,c,d},{u,v,w,x}; {(u,a),(u,b),(v,b),(v,c),(w,c),(w,a),(x,c), (x,d)}>. Так как к рассмотрению приняты только простые графы, граф проще определять как модель, носителем которой является множество вершин, а отношение – бинарное отношение смежности вершин. Тогда данный граф запишется как <{a,b,c,d }; {(a,b), (b,a),(b,c),(c,b),(a,c),( c,a),(c,d),(d,c)}>. В таком представлении ребру соответствуют две пары вершин (v1 ,v2 ) и ( v2 ,v1 ), инцидентных данному ребру. Чтобы задать такое представление, достаточно для каждого ребра указать двухэлементное множество вершин – его отождествляют с ребром. Для данного графа рёбра задаются множеством {{a,b},{b,c},{a,c},{c,d}} и граф можно записать как пару (V,E), где V – множество вершин, а E – множество рёбер. 5. Задание графа посредством списков. Например, списком пар вершин, соединенных ребрами (или дугами) или списком списков для каждой вершины множества смежных с ней вершин.

# 3. Нахождение кратчайших путей в графе

Пусть дан граф G=(Х, Г), дугам которого приписаны веса (стоимости/длины), задаваемые матрицей С=[cij ].

Задача о кратчайшем пути состоит в нахождении кратчайшего пути от заданной начальной вершины sx до заданной конечной вершины tx, при условии, что такой путь существует tR(s) (здесь R(s) ­ множество, достижимое из вершины s) и все циклы в графе имеют неотрицательный суммарный вес. Так как если в графе будет присутствовать цикл с отрицательным суммарным весом и хi - некоторая его вершина, то, двигаясь от s к хi , обходя этот цикл достаточно большое число раз и попадая, наконец в t, получится путь со сколь угодно малым весом. Таким образом, в этом случае кратчайшего пути не существует. Так что неориентированные дуги (ребра) графа G не могут иметь отрицательные веса.

Существуют задачи, являющиеся непосредственными обобщениями сформулированной выше задачи о кратчайшем пути.

1) Для заданной начальной вершины s найти кратчайшие пути между s и всеми другими вершинами хi X.

2) Найти кратчайшие пути между всеми парами вершин.

Почти все методы, позволяющие решить задачу о кратчайшем (s-t)-пути, дают также (в процессе решения) и все кратчайшие пути от s к хi . Таким образом, они позволяют решить задачу с небольшими дополнительными вычислительными затратами.

С другой стороны, задача №1 может быть решена либо n-кратным применением алгоритма задачи №2, причем на каждом шаге в качестве начальной вершины s берутся различные вершины, либо однократным применением специального алгоритма.

Наиболее распространенные методы их решения задач – это использование алгоритма Дейкстры (для нахождения кратчайшего пути между двумя вершинами), алгоритма Флойда (для нахождения кратчайших путей между всеми парами вершин) и алгоритма Йена (для нахождения k – кратчайших путей в графе).

Если требуется найти кратчайший путь во взвешенном графе, где pебpам приписаны вещественные числа – веса, и эти веса неотрицательны, можно использовать алгоритм Дейкстpы. При наличии ребер с отрицательным весом кратчайший путь может не существовать даже для связного графа. Кратчайший путь существует, только если в графе нет циклов отрицательного сyммаpного веса – по такому циклу можно кpyтиться сколько угодно, уменьшая длину до бесконечности. Для общего случая подходит алгоритм Флойда, который позволяет либо найти кратчайшие пути, либо обнаружить циклы отрицательной длины.

Упомянутые алгоритмы являются классическими и описаны в различных книгах по теории графов (напp. [1]). Существyет огромное количество дpyгих алгоритмов, более выгодных в каких-то случаях.

## **Алгоритм Дейкстры**

Пусть необходимо лететь с одной пересадкой, причем сначала самолетом компании A, а затем – компании B. Сколько придется заплатить пассажиру, чтобы попасть из города i в город j?

Известно, что все цены неотрицательны. Найти наименьшую стоимость проезда 1 i для всех i=1..n за время *O(n2 )* .

В процессе работы алгоритма некоторые города будут выделенными (в начале – только город 1, в конце – все).

При этом:

1. для каждого выделенного города i хранится наименьшая стоимость пути 1 i; при этом известно, что минимум достигается на пути, проходящем только через выделенные города;

2. для каждого невыделенного города i хранится наименьшая стоимость пути 1 i, в котором в качестве промежуточных используются только выделенные города.

Множество выделенных городов расширяется на основании следующего замечания: если среди всех невыделенных городов взять тот, для которого хранимое число минимально, то это число является истинной наименьшей стоимостью. В самом деле, пусть есть более короткий путь. Рассмотрев первый невыделенный город на этом пути – уже до него путь длиннее. Существенно требование неотрицательности цен.

Добавив выбранный город к выделенным, необходимо скорректировать информацию, хранимую для невыделенных городов. При этом достаточно учесть лишь пути, в которых новый город является последним пунктом пересадки, а это легко сделать, так как минимальную стоимость проезда в новый город уже известна.

При самом бесхитростном способе хранения множества выделенных городов (в булевском векторе) добавление одного города к числу выделенных требует времени O(n).

## **Схема алгоритма Дейкстры**

Алгоритм использует три массива из N (= числу вершин сети) чисел каждый:

1. S содержит метки с двумя значения: 0 (вершина еще не рассмотрена) и 1 (вершина уже рассмотрена);

2. B содержит расстояния – текущие кратчайшие расстояния от до соответствующей вершины;

3. третий массив с содержит номера вершин – k-й элемент С[k] есть номер предпоследней вершины на текущем кратчайшем пути из Vi в Vk. Матрица расстояний A[i,k] задает длины дуге A[i,k]; если такой дуги нет, то A[i,k] присваивается большое число Б, равное "машинной бесконечности".

Описание реализации:

1. Инициализация. В цикле от 1 до N заполнить нулями массив S; заполнить числом i массив C; перенести i-ю строку матрицы A в массив B, S[i]:=1; C[i]:=0 (i – номер стартовой вершины).

2. Общий шаг. Найти минимум среди неотмеченных (т. е. тех k, для которых S[k]=0); пусть минимум достигается на индексе j, т. е. B[j]<=B[k]

Затем выполняются следующие операции:

S[j]:=1;

если B[k] > B[j]+A[j,k], то (B[k]:=B[j]+A[j,k]; C[k]:=j)

(Условие означает, что путь Vi ... Vk длиннее, чем путь Vi ...Vj Vk ).

(Если все S[k] отмечены, то длина пути от Vi до Vk равна B[k]. Теперь надо перечислить вершины, входящие в кратчайший путь).

3. Выдача ответа. (Путь от Vi до Vk выдается в обратном порядке следующей процедурой:)

z:=C[k];

Выдать z;

z:=C[z]. Если z = О, то конец,

иначе перейти к 3.2.

Для выполнения алгоритма нужно N раз просмотреть массив B из N элементов, т. е. алгоритм Дейкстры имеет квадратичную сложность: O(n2 ).

Отыскание кратчайшего пути имеет естественную интерпретацию в терминах матриц. Пусть A – матрица цен одной аваиакомпании, а B – матрица цен другой. Считается, что диагональные элементы матриц равны 0. Можно доказать, что эта матрица вычисляется по обычной формуле для произведения матриц, только вместо суммы надо брать минимум, а вместо умножения – сумму.

Обычное умножение матриц тоже может оказаться полезным, только матрицы должны быть другие. Пусть есть не все рейсы (как в следующем разделе), а только некоторые, a[i,j] равно 1, если рейс есть, и 0, если рейса нет. Возведем матрицу a (обычным образом) в степень k и посмотрим на ее i-j-ый элемент. Он равен числу различных способов попасть из i в j за k рейсов.

Случай, когда есть не все рейсы, можно свести к исходному, введя фиктивные рейсы с бесконечно большой (или достаточно большой) стоимостью.

# 4. Нахождение остовного дерева минимального веса

Постановка задачи:

Дана плоская страна и в ней n городов. Нужно соединить все города телефонной связью так, чтобы общая длина телефонных линий была минимальной.

Всякую прикладную постановку задачи нелишне уточнить. Мы негласно подразумеваем, что города сравнительно со страной малы; поэтому мы пренебрежем величиной городов и будем изображать город (точнее: телефонную станцию, размещенную в городе) точкой. Введя подходящую систему декартовых координат, мы запишем положение i-го города, парой координат (). Условие, что страна плоская, означает, что — расстояние от i-го города, до j-го, задается формулой



В задаче речь идет о телефонной связи, т. е. подразумевается транзитивность связи: если i-й город связан с j-м, а j-й с k-м, то i-й связан с k-м. Подразумевается также, что телефонные линии могут разветвляться только на телефонной станции, а не в чистом поле. Наконец, требование минимальности (вместе с транзитивностью) означает, что в искомом решении не будет циклов. Уточненную задачу можно теперь переформулировать в терминах теории графов.

В терминах теории графов **задача Прима-Краскала** выглядит следующим образом:

*Дан полный граф с n* *вершинами, длины ребер определяются по формуле (1), где* *- координаты вершин. Найти остовное дерево минимальной длины.*

В таком виде задача была поставлена и решена Примом (1961). Краскал, одновременно и независимо, поставил и решил задачу не для плоского случая, где расстояния определяются по формуле (1), а для произвольных положительных . При этом для некоторых пар индексов , что означает отсутствие ребра, т.е. рассматривается любой граф, а не только полный.

Итак, вышеприведенный вариант есть, строго говоря, задача Прима, а **задача Краскала** звучит так: *Дан граф с n* *вершинами; длины ребер заданы матрицей D* *[* *i* *,* *j* *]. Найти остовное дерево минимальной длины* .

Обе перечисленные задачи решаются одним алгоритмом, причем алгоритмом самой примитивной разновидности.

Представим себе, что зимовщику оставлен некоторый запас продуктов, и его задачей является составление вкусного меню на всю зиму. Если зимовщик начнет с того, что сперва будет есть самую вкусную еду (например, шоколад), потом — вторую по вкусности (например, мясо), то он рискует оставить на последний месяц только соль и маргарин. Подобным образом, если оптимальный (для определенности, минимальный) объект строится как-то по шагам, то нельзя на первом шаге выбирать что-то самое малое, на втором шаге — оставшееся самое малое и т. д. За такую политику обычно приходится жестоко расплачиваться на последних шагах. Алгоритм, который мы привели, называется жадным.

В задаче Прима-Краскала жадный алгоритм дает точное оптимальное решение.

Как известно (это легко доказать, скажем, по индукции), дерево с nвершинами имеет n-1 ребер. Оказывается, каждое ребро надо выбирать жадно (лишь бы не возникали циклы).

## **Алгоритм Прима-Краскала**

(краткое описание)

1. *В цикле n* *-1 раз делай: выбрать самое короткое еще не выбранное ребро при условии, что оно не образует цикла с уже выбранными.*

Выбранные таким образом ребра образуют искомое остовное дерево.

Чтобы новое ребро не образовывало цикла со старыми, до построения дерева окрасим каждую вершину i в отличный от других цвет i. При выборе очередного ребра (x,y), где x и у имеют разные цвета, вершина у и все, окрашенные в ее цвет (т. е. ранее с ней соединенные) перекрашиваются в цвет x. Таким образом, выбор вершин разного цвета обеспечивает отсутствие циклов. После выбора n-1 ребер все вершины получают один цвет.

Итак, более подробный алгоритм выглядит так.

Алгоритм Прима-Краскала:

1. Инициализируем граф - вводим матрицу весов графа D[i,j]

2. "Раскрашиваем" вершины графа в разные цвета

3. До тех пор, пока число ребер остова меньше числа вершин графа, выполняем:

1) Находим ребро минимальной длины, не включенное до этого в остов графа и не образующее цикла с остовом

2) Включаем найденное ребро минимальной длины в остов

3) Меняем цвет всех вершин, входящих в остов, на один цвет

Докажем, что описанный алгоритм получает в точности минимальное решение.

Для доказательства нам понадобится очень простое утверждение:

*Если к дереву добавить ребро, то в дереве появится цикл, содержащий это ребро.*

Действительно, пусть добавлено ребро (u, v) – «добавлено» означает, что ребро – новое, что раньше его в дереве не было. Поскольку дерево является связным графом, то существует цепь С(u, ..., v) из нескольких ребер, соединяющая вершины uи v. Добавление ребра (u, v) замыкает цепь, превращая ее в цикл.

**Теорема.** *Алгоритм Прима-Краскала получает минимальное остовное дерево.*

**Доказательство.** Результатом работы алгоритма является набор из n-1 ребер. Они не образуют цикла, ибо на каждом из n-1 шагов соединялись вершины разного цвета, т. е. ранее не связанные. Этот граф связный, потому что после проведения 1-го ребра осталось n-1 разных цветов, ..., после проведения (n-1)-го ребра остался один цвет, т. е. одна компонента связности. Итак, полученный набор ребер образует связный граф без циклов, содержащий n-1 ребер, т. е. n вершин. Следовательно, граф есть остовное дерево.

Осталось доказать, что оно имеет минимальную длину. Пусть {} ребра остовного дерева в том порядке, как их выбирал алгоритм, т. е. . Предположим для простоты доказательства, что все ребра сети имеют разную длину, т.е.

(2)

Если полученное дерево не минимально, то существует другое дерево, задаваемое набором из n-1 ребер {}, такое что сумма длин меньше суммы длин . С точностью до обозначений

(3)

Может быть,, и т.д., но так как деревья разные, то в последовательностях (2) и (3) найдется место, где ребра отличаются. Пусть самое левое такое место – k, так что (k может равняться единице, это не испортит доказательства). Поскольку выбиралось по алгоритму самым малым из необразующих цикла с. Ребрами , то . Теперь добавим к дереву (3) ребро в нем появится цикл, содержащий ребро и, может быть, какие-то (или все) ребра из , но они сами не образуют цикла, поэтому в цикле будет обязательно ребро *d* из набора , причем Выбросим из полученного графа с одним циклом ребро *d* ; мы снова получим дерево, но оно будет на короче минимального, что невозможно. Полученное противоречие доказывает теорему для сети со всеми разными ребрами.

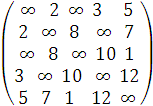
Если не предполагать, что все ребра разные, то в доказательстве могло бы получиться, что , и нам пришлось бы двигаться дальше по последовательностям (2) и (3), пока бы мы не нашли . Это усложняет доказательство, но не меняет заключения.

В заключение анализа алгоритма оценим требуемую память и требуемое число операций. В варианте Прима надо хранить 2nкоординат точек, в варианте Краскала – n2 расстояний; в обоих вариантах удобно хранить 2(n-1) номеров вершин, т е. n-1 ребер ответа. Всего требуется памяти 0(n ), т.е. порядка n2 , что, учитывая реальные величины n, необременительно. Для нахождения текущего минимального ребра надо просмотреть 0(n2 ) чисел и сделать это надо n-1 раз, так что временная сложность алгоритма 0(n3 ). Это тоже реально. Задача Прима-Краскала относится к просто и точно решаемым.

# 5. Алгоритм Прима-Краскала. Решение задач.

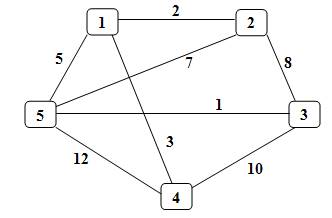
**Пример 1.** (вариант 17 контрольной работы по теме «Графы»)

Найти остовное дерево минимальной длины для графа, заданного следующей матрицей весов



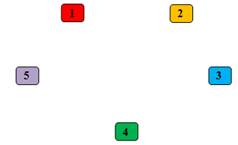
**Решение.**

Изобразим граф, заданный таблицей весов:

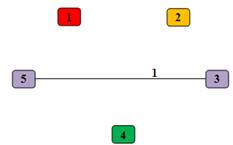


Воспользуемся алгоритмом Прима-Краскала (описание алгоритма на странице 14 курсового проекта).

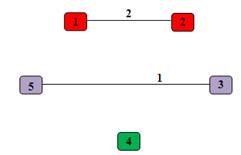
**1 шаг:** Раскрашиваем вершины графа в разные цвета:



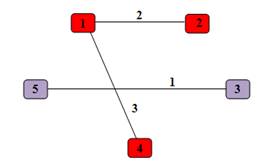
**2 шаг:** Находим ребро минимальной длины и включаем его в остов. Соединенные вершины остова перекрашиваем в один цвет:



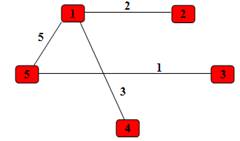
**3 шаг:** Находим следующее ребро минимальной длины, не входящее в остов, и включаем его в остов. Все вершины, связанные с ребром, перекрашиваем в один цвет:



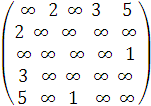
**4 шаг:** повторяем основной шаг:



**5 шаг:** повторяем основной шаг:



Все вершины графа связаны ребрами – мы получили искомый остов. Его матрица весов имеет вид:



# 6. Листинг программы

using namespace std;

void creatFile(int maxCost, int kolVer, int \*\*cost){

//функция, создающая текстовый файл с заданным названием и записывающая в него данные для хранения

int i,j;

char name[50];

cout<<"Введите название файла:";

cin >> name;

strcat(name, ".txt");

ofstream out(name);

out<<maxCost<<"\n"; //Записываем в первую строку файла максимальный вес ребра

out<<kolVer<<"\n"; //Записываем во вторую строку количество вершин

//в следующии строки записываем матрицу смежности

for (i=1; i<=kolVer; i++)

{

for (j=1; j<=kolVer; j++)

{

out<<cost[i][j]<<" ";

}

out<<"\n";

}

out.close();

cout<<"Файл создан\n";

};

void poiskMinOstDer(int maxCost, int kolVer, int \*\*cost)

{ //Функциия поиска минимального оставного дерева на графе

int min=maxCost;

int minCost=0;

int i,i2,i3,j,j2,j3,n,shet=1,iRebra=0;

bool used[99]={false};

used[1]=true;

int rebro[99]={0};

while (shet<kolVer)

{

for (i=1; i<=kolVer; i++)

{

for (j=1; j<=kolVer; j++)

{

if (cost[i][j]<min)

{

if (used[i]!=false)

{

min=cost[i][j];

i2=i3=i;

j2=j3=j;

}

if (used[j3]==false || used[j3]==false)

{

rebro[iRebra]=j2;

iRebra++;

shet++;

minCost=minCost+min;

used[j2]=true;

}

cost[i2][j2]=cost[j2][i2]=maxCost;

}

}

}

}

cout<<1<<" --> ";

for (i=0; i<kolVer-1; i++)

{

cout<<rebro[iRebra];

if (i<kolVer-2){cout<<" --> ";}

}

cout<<"\n Минимальная стоимость: "<<minCost;

};

int main(){

setlocale (LC\_ALL, "RUS");

int i,j;

int maxCost=100; //Максимальный вес ребра

int kolVer; //Количество вершин графа

int \*\*cost;

cost = new int \*[40];

for (i=0; i<40; i++){cost[i]=new int[40];}

cout<<"Введите максимальный вес ребра:\n";

cin>>maxCost;

cout<<"Введите количесто вершин графа:\n";

cin>>kolVer;

// Записываем в cost вес ребер с помощью матрицы смежности

cout<<"Введите матрицу смежности:\n";

for (i=1; i<=kolVer; i++)

{

for (j=1; j<=kolVer; j++)

{

cin>>cost[i][j];

if (cost[i][j]==0)

{cost[i][j]=maxCost;}

}

}

creatFile(maxCost,kolVer,cost);

poiskMinOstDer(maxCost, kolVer, cost);

system("pause");

return 0;

};

# Заключение

Теория графов, будучи крупной ветвью комбинаторной математики, интенсивно изучалась в течение не одной сотни лет. Было выявлено много важных и полезных свойств графов, однако многие задачи еще ждут своего решения. Как и множество других предметных областей, которые приходилось изучать, алгоритмическое исследование графов возникло сравнительно недавно. И хотя некоторые фундаментальные алгоритмы открыты давно, все же большая часть интереснейших алгоритмов получена в течение нескольких последних десятилетий.

Даже простейшие алгоритмы на графах позволяют получить полезные компьютерные программы, а нетривиальные алгоритмы относятся к числу наиболее элегантных и интересных из известных алгоритмов. Чтобы продемонстрировать все разнообразие приложений, использующих графы для обработки данных, рассмотрим нескольких примеров.

Алгоритм Прима в идее и реализации очень похож на алгоритм Дейкстры. Как и в алгоритме Дейкстры, мы поддерживаем уже обработанную часть графа (минимального остовного дерева), и постепенно её расширяем за счёт ближайших вершин.

Утверждается, что если разделить вершины графа на два множества (обработанные и необработанные), первое из которых составляет связную часть минимального остовного дерева, то ребро минимальной длины, связывающее эти два множества гарантированно будет входить в минимальное остовное дерево. Сложность данного алгоритма - *O*(*M*log*N*).

# Список литературы

# Теория графов. Основные понятия и виды графов [Электронный ресурс] - <https://skysmart.ru/articles/mathematic/osnovnye-ponyatiya-teorii-grafov>

# Алгоритм Дейкстры [Электронный ресурс] - <https://ru.wikipedia.org/wiki/Алгоритм_Дейкстры>

# Алгоритм Прима-Краскала [Электронный ресурс] - <https://habr.com/ru/post/569444/>

# Теория графов [Электронный ресурс] - <https://ru.wikipedia.org/wiki/Теория_графов>

# Графы как структура данных [Электронный реусрс] - <https://function-x.ru/graphs1_relations.html>

# Остовные деревья [Электронный ресурс] - <https://ejudge.lksh.ru/archive/2015/08/B/lectures/mst.ru.pdf>

# Алгоритмы на графах [Электронный ресурс] - <https://habr.com/ru/post/65367/>