Содержание

Введение

. Граф и его элементы

.1 Основные понятия

.2 Ориентированный граф

.3 Неориентированный граф

.4 Смежность

.5 Маршруты и пути

. Постановка задачи коммивояжера и алгоритмы решения

.1 Задача коммивояжера

.2 Методы решения задачи коммивояжера

. Понятия транспортной сети

.1 Понятие увеличивающая дуга, цепь, разрез

. Алгоритм Флойда-Уоршелл

. Постановка задачи

. Решение задачи аналитическим методом

. Создание приложения для решения задачи

.1 Описание алгоритма

.3 Тестирование программы

.4 Руководство пользователя

Заключение

Список использованных источников

Приложение A. Расчет элементов для матриц

# Введение

Теория графов представляет собой раздел математики, имеющий широкие практические приложения. Теория графов - область дискретной математики, особенностью которой является геометрический подход к изучению объектов. Во многих случаях жизни в частности, в проектной практике нам приходится рисовать на бумаге точки, изображающие города, отдельные объекты, железнодорожные узлы и т.п., и соединять эти точки линиями или стрелками, обозначающими некоторые связи или отношения. Такие схемы под различными названиями встречаются всюду, электрические цепи, в физике - радиосхемы; в экономике - диаграммы организации работ и т.д. В основном все схемы рассматриваются в абстрактной форме как самостоятельные математические объекты, названные графами. Такой подход к изучаемым объектам дает возможность теории графов иметь самые разнообразные и многочисленные приложения, в том числе и в области градостроительного проектирования и научных исследований. Первая работа по теории графов, принадлежавшая Эйлеру, появилась в 1736 году и была связана c решением задачи о Кёнигсбергских мостах. Однако, эта работа Эйлера была единственной в течение почти ста лет. Вначале теория графов казалась незначительным разделом математики, так как имела дело лишь c математическими развлечениями и головоломками, например, в схемотехнике, топология межсоединений элементов на печатной плате или микросхеме, задача o перевозках, задача o кругосветном путешествии и др. Развитие математики и ее приложении послужило сильным толчком к развитию теории графов. Уже в середине Х века графы использовались при построении эхем электрических цепей и молекулярных схем. Впервые термин «граф» ввел в 1936г. венгерский математик Денеш Кёниг. B его работе теория графов была представлена как отдельная математическая дисциплина, находящая в настоящее время широкое применение в автоматике, телемеханике, кибернетике, электронике, физике, экономике, психологии, биологии и других областях науки.

1. Граф и его элементы

1.1 Основные понятия

Граф - это множество точек, называемых вершинами, и множество линий, называемых ребрами, которые соединяют пары вершин (или вершину саму с собой).

Геометрически граф можно представить как набор вершин (точек), определенные пары которых соединены линиями.

Например: сеть дорог, соединяющая города 1, 2, 3, 4, 5 можно представить в виде графа следующим образом: города обозначим точками (вершинами), а дороги неориентированными линиями (рисунок 1). Неориентированные линии означают наличие двустороннего движения между соответствующей парой городов. Пересечения линией не считаются вершинами.



Рисунок 1 - Граф сети дорог

Рассмотрим другой пример, ориентированный граф (рисунок 2).



Рисунок 2 - Граф

Таким образом, математически граф можно рассматривать как множество кружков и множество соединяющих их линий. Кружки называются вершинами графа, линии со стрелками - дугами, без стрелок -ребрами.
В зависимости от наличия или отсутствия ориентации у пар элементов из множества V различают графы:

а) ориентированные <http://graphsmaster.com/orgraph.php>,

б) неориентированные (реберные),

в) смешанные.

## 1.2 Ориентированный граф

Граф называется ориентированным или орграфом, если некоторые ребра имеют направление (рисунок 3). Это означает, что в орграфе некоторая вершина может быть соединена с другой вершиной, а обратного соединения нет.



Рисунок 3 - Ориентированный граф

## 1.3 Неориентированный граф

В неориентированном графе отношения симметричны, то есть (u, v) = (v, u). В неориентированном графе нет дуг, связи называют ребрами.



Рисунок 4 - Неориентированный граф

## 1.4 Смежность

задача коммивояжер транспортный приложение

Пусть D1, D2- вершины, е = (D1, D2) - соединяющее их ребро (рисунок 5). Тогда вершина D1 и ребро е инцидентны, вершина D2 и ребро е также инцидентны.



Рисунок 5 - Смежные вершины

Две вершины называются смежными, если они соединены ребром/дугой (рисунок 5).

Два ребра называются смежными, если они соединены вершиной/узлом (рисунок 6).



Рисунок 6 - Смежные ребра

Смежные вершины - две вершины, инцидентные одному ребру.

Смежные ребра - два ребра, инцидентные одной вершине.

Множество вершин, смежных с вершиной V, называется множеством смежности вершины V и обозначается Г+(V) (рисунок 7).



Рисунок 7 - Множество смежности

## .5 Маршруты и пути

Одно из наиболее простых свойств, которым может обладать граф это свойство быть связным <http://graphsmaster.com/svjaznost.php>. В данном разделе рассматриваются основные структурные свойства связных и несвязных графов.

Маршрут в графе - чередующаяся последовательность вершин и ребер.

Эта последовательность начинается и кончается вершиной, и каждое ребро последовательности инцидентно двум вершинам, одна из которых непосредственно предшествует ему, а другая непосредственно следует за ним (рисунок 8).



Рисунок 8 Граф для иллюстрации маршрутов

Указанный маршрут соединяет вершины Do и Dn, и его можно обозначить Do,D1,D2,...,Dn (наличие ребер подразумевается). Эта последовательность иногда называется (Do-Dn) - маршрутом. Маршрут замкнут, если D = Dn, и открыт в противном случае.

Цепь - маршрут с различными ребрами.

Простая цепь-маршрут с различными вершинами (а значит, и ребрами).

Цикл- замкнутая цепь.

В помеченном графе J на рисунке 8 D1 D2 D3 D4 - маршрут, который не является цепью, а D1 D2 D4 D3 D2 - цепь, где D1 D2 D4 D3 - простая цепь и D2 D4 D3 D2 - простой цикл.

Обозначим через Jn граф состоящий из одного простого цикла с n вершинами, и через Pn простую цепь с n вершинами. Jn часто называют треугольником.

2. Постановка задачи коммивояжера и алгоритмы решения

## 2.1 Задача коммивояжера

Задача коммивояжера - задача математического программирования по определению оптимального маршрута движения коммивояжера, цель которого состоит в том, чтобы посетить все объекты, записанные в задании, за кратчайший срок и с наименьшими затратами. В теории графов - это поиск пути, связывающего два или более узла, с использованием критерия оптимальности.

Задача коммивояжера является типичной задачей оптимизации, которая широко применяется при разработке программного обеспечения. Задача о коммивояжере является упрощенной моделью для многих других задач дискретной оптимизации, а также часто является подзадачей. В своей области (оптимизации дискретных задач) она служит своеобразным катализатором, стимулирующим разработку наиболее эффективных методов, алгоритмов и способов их машинной реализации.

Задача коммивояжера формулируется очень просто: на плоскости (в пространстве) расположены N городов, заданы расстояния между каждой парой городов. Требуется найти маршрут минимальной длины с посещением каждого города ровно один раз и с возвращением в исходную точку.

В задаче коммивояжера целевой функцией, которую надо минимизировать, является стоимость обхода.

Особенностью задачи о коммивояжере является необходимость дополнительно учитывать расстояния от города до города, которые предполагаются известными. Эти «расстояния» можно заменить на количество затраченного времени, стоимость проезда или предполагать другие произвольные значения. В общем случае мы даже не предполагаем, что стоимость проезда из I в J обязательно совпадает со стоимостью обратного проезда из I в J. Данная задача соединяет в себе простоту условия и сложность решения, обусловленную большими размерами поискового пространства.

На графах задача формулируется следующим образом: требуется найти гамильтонов цикл наименьшей стоимости во взвешенном полном графе. Т.е. выйдя из стартовой вершины, посетить каждую вершину графа ровно один раз и вернуться в начальную по кратчайшему пути.

.2 Методы решения задачи коммивояжера

Задачи коммивояжера решаются посредством различных методов, выведенных в результате теоретических исследований. Все эффективные методы (сокращающие полный перебор) - методы эвристические. В большинстве эвристических методов находится не самый эффективный маршрут, а приближённое решение. Зачастую востребованы так называемые any-time алгоритмы, то есть постепенно улучшающие некоторое текущее приближенное решение.

Выделяют следующие группы методов решения задач коммивояжера, которые относят к простейшим:

а) Полный перебор.

Полный перебор (или метод «грубой силы») - метод решения задачи путем перебора всех возможных вариантов. Сложность полного перебора зависит от количества всех возможных решений задачи. Если пространство решений очень велико, то полный перебор может не дать результатов в течение нескольких лет или даже столетий.

б) Случайный перебор.

Обычно выбор решения можно представить последовательностью выборов. Если делать эти выборы с помощью какого-либо случайного механизма, то решение находится очень быстро, так что можно находить решение многократно и запоминать «рекорд», т. е. наилучшее из встретившихся решений. Этот наивный подход существенно улучшается, когда удается учесть в случайном механизме перспективность тех или иных выборов, т. е. комбинировать случайный поиск с эвристическим методом и методом локального поиска. Такие методы применяются, например, при составлении расписаний для Аэрофлота.

в) Жадные алгоритмы (метод ближайшего соседа, метод включения ближайшего города, метод самого дешевого включения).

Жадный алгоритм - алгоритм нахождения наикратчайшего расстояния путём выбора самого короткого, ещё не выбранного ребра, при условии, что оно не образует цикла с уже выбранными рёбрами. «Жадным» этот алгоритм назван потому, что на последних шагах приходится жестоко расплачиваться за жадность. При решении задачи коммивояжера жадный алгоритм превратится в стратегию «иди в ближайший (в который еще не входил) город». Жадный алгоритм, очевидно, бессилен в этой задаче. Рассмотрим для примера сеть (рисунок 9), представляющую узкий ромб. Коммивояжер стартует из города 1. Алгоритм «иди в ближайший город» выведет его в город 2, затем 3, затем 4; на последнем шаге придется платить за жадность, возвращаясь по длинной диагонали ромба. В результате получится не кратчайший, а длиннейший тур.



Рисунок 9 - граф для примера

3. Понятия транспортной сети

Транспортной сетью называется конечный Связный орграф G(V, E) без петель, каждой дуге которого поставлено в соответствие некоторое неотрицательное число c(ei), называемое пропускной способностью дуги, и существует:

- ровно одна вершина Vo = S, в которую не заходит ни одна дуга, называемая источником или началом сети;

- ровно одна вершина Vn=t, из которой не выходит ни одной дуги; эта вершина называется стоком или концом сети.

Потоком сети называется неотрицательная функция f(1) такая, что f(e) меньше или равно c(e). (Поток не может превышать пропускную способность дуги.)

Дуга e=(Vi,Vj) называется насыщенной потоком f, если f(Vi, Vj) = c(Vi, Vj)=c(e) (Поток называется полным, если содержит насыщенную дугу f(e)=c(e).)

3.1 Понятие увеличивающая дуга, цепь, разрез

Рассмотрим данные понятия на примере:

Разрезом L сети S(N, U) называется множество насыщенных дуг, отделяющих источник s от стока t.

Пусть задана сеть S=(N, U) с множеством вершин N и множеством дуг U.

Определение: дуга u є U, соединяющая вершины i є N, j є N сети S, называется допустимой дугой, если она обладает одним из следующих свойств:

а) направление дуги совпадает с направлением потока, и значение потока по этой дуге меньше ее пропускной способности:

u=(i, j), j(u)<c(u) (1)

б) направление дуги противоположного направлению потока, и величина потока отлична от нуля:

u=(j, i), j(u)>0 (2)

Дуги, обладающие первым свойством, называют увеличивающими; дуги, обладающие вторым свойством, - уменьшающими.

Увеличивающей цепью, соединяющей вход и выход сети, называется простая цепь, все дуги которой являются допустимыми.

Пример 1: построим увеличивающую цепь для сети S=(N, U), представленной на рисунке 10.



Рисунок 10 - увеличивающая цепь

Над каждой дугой указана ее пропускная способность, в скобках - поток по этой дуге.

Цепь (s, 1, 2, 4, t) является увеличивающей, так как все дуги - допустимые:

- дуга (s, 1) - увеличивающая, так как она проходит по направлению потока, и поток по ней меньше ее пропускной способности:

< 10;

- дуга (1,2) - также увеличивающая дуга: 12 < 15;

- дуга (2,4) - уменьшающая, так как она проходит против потока и поток по ней 3 > 0;

- дуга (4, t) - увеличивающая: 1 < 4.

4. Алгоритм Флойда-Уоршелл

Алгоритм Флойда является одним из методов поиска кратчайших путей в графе. В отличии от алгоритма Дейкстры, который позволяет при доведении до конца построить ориентированное дерево кратчайших путей от некоторой вершины, метод Флойда позволяет найти длины всех кратчайших путей в графе.

Прежде чем представлять алгоритмы, необходимо ввести некоторые обозначения. Перенумеруем вершины исходного графа целыми числами от 1 до N. Обозначим через di,jm длину кратчайшего пути из вершинм i в вершину j, который в качестве промежуточных может содержать только первые m вершин графа. (Напомним, что промежуточной вершиной пути является любая принадлежащая ему вершина, не совпадающая с его начальной или конечной вершинами.) Если между вершинами i и j не существует ни одного пути указанного типа, то условно будем считать, что di,jm=∞. Из данного определения величин di,jm следует, что величина di,j0, представляет длину кратчайшего пути из вершины i в вершину j, не имеющего промежуточных вершин, т. е. длину кратчайшей дуги, соединяющей i с j (если такие дуги присутствуют в графе). для любой вершины i положим di,im= 0. Отметим далее, что величина di,jmпредставляет длину кратчайшего пути между вершинами i и j.

Обозначим через Dm матрицу размера NxN, элемент (i, j) которой совпадает с di,jm. Если в исходном графе нам известна длина каждой дуги, то мы можем сформировать матрицу D0. Наша цель состоит в определении матрицы DN, представляющей кратчайшие пути между всеми вершинами рассматриваемого графа.

В алгоритме Флойда в качестве исходной выступает матрица D0. Вначале из этой матрицы вычисляется матрица D1. Затем по матрице D1 вычисляется матрицав D2 и т. д. Процесс повторяется до тех пор, пока по матрице DN-1 не будет вычислена матрица DN.

Рассмотрим основную идею, лежащую в основе алгоритма Флойда. Суть алгоритма Флойда заключается в проверке того, не окажется ли путь из вершины i в вершину j короче, если он будет проходить через некоторую промежуточную вершину m. Предположим, что нам известны:

1) кратчайший путь из вершины i в вершину m, в котором в качестве промежуточных допускается использование только первых (m - 1) вершин;

2) кратчайший путь из вершины m в вершину j, в котором в качестве промежуточных допускается использование только первых (m - 1) вершин;

) кратчайший путь из вершины i в вершину j, в котором в качестве промежуточных допускается использование только первых (m - 1) вершин.

Поскольку по предположению исходный граф не может содержать контуров отрицательной длины, один из двух путей - путь, совпадающий с представленным в пункте 3, или путь, являющийся объединением путей из пунктов 1 и 2 - должен быть кратчайшим путем из вершины i в вершину j, в котором в качестве промежуточных допускается использование только первых m вершин. Таким образом,

di,jm=min{ di,mm-1+ dm,jm-1; di,jm-1}, (3)

где di,jm - элемент матрицы Dm, di,jm-1 - элементы матрицы Dm-1 найденой на предыдущем шаге алгоритма.

Из соотношения видно, что для вычисления элементов матрицы Dm необходимо располагать лишь элементами матрицы Dm-1. Более того, соответствующие вычисления могут быть проведены без обращения к исходному графу. Теперь мм в состоянии датьформальное описание алгоритма Флойда для нахождения на графе кратчайших путей между всеми парами вершин.

Алгоритм:

) перенумеровать вершины графа от 1 до N целыми числами, определить матрицу D0, каждый элемент di,j которой есть длина кратчайшей дуги между вершинами i и j. Если такой дуги нет, положить значение элемента равным ∞. Кроме того, положить значения диагонального элемента di,iравным 0;

) для целого m, последовательно принимающего значения 1...N определить по элементам матрицы Dm-1элементы Dm;

) алгоритм заканчивается получением матрицы всех кратчайших путей DN, N - число вершин графа. Для определения по известным элементам матрицы Dm-1 элементов матрицы Dm в алгоритме Флойда применяется рекурсивное соотношение указанное в формуле (3).

5. Постановка задачи

Найти путь наименьшей длины между вершинами 1 и 8. Построить коммуникационную сеть минимальной длины, используя Алгоритм Флойда-Уоршелл.



Рисунок 11 - Граф задачи с весом ребер

# 6. Решение задачи аналитическим методом

Обозначим через di,jm длину кратчайшего пути из вершины i в вершину j, который в качестве промежуточных может содержать только первые m вершин графа.

На основании исходных данных формируем матрицу длин кратчайших дуг D0 (Таблица 1), каждый элемент которой равен длине кратчайшей дуги между вершинами i и j. Если такой дуги нет, положим значение элемента равным ∞.

Таблица 1- Матрица D0

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D0= |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  | 1 | 0 | 1 | 5 | 9 | 9 | ∞ | ∞ | ∞ |
|  | 2 | 1 | 0 | 1 | ∞ | ∞ | 1 | ∞ | ∞ |
|  | 3 | 5 | 1 | 0 | ∞ | ∞ | 2 | ∞ | 1 |
|  | 4 | 9 | ∞ | ∞ | 0 | 9 | ∞ | 5 | ∞ |
|  | 5 | 9 | ∞ | ∞ | 9 | 0 | ∞ | 4 | 4 |
|  | 6 | ∞ | 1 | 2 | ∞ | ∞ | 0 | ∞ | 5 |
|  | 7 | ∞ | ∞ | ∞ | 5 | 4 | ∞ | 0 | 8 |
|  | 8 | ∞ | ∞ | 1 | ∞ | 4 | 5 | 8 | 0 |

Представим матрицу D1 (Таблица 2). включив в нее рассчитанные элементы из приложения A.

Таблица 2- Матрица D1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D1= |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  | 1 | 0 | 1 | 5 | 9 | 9 | ∞ | ∞ | ∞ |
|  | 2 | 1 | 0 | 1 | 10 | 10 | 1 | ∞ | ∞ |
|  | 3 | 5 | 1 | 0 | 14 | 14 | 2 | ∞ | 1 |
|  | 4 | 9 | 10 | 14 | 0 | 9 | ∞ | 5 | ∞ |
|  | 5 | 9 | 10 | 14 | 9 | 0 | ∞ | 4 | 4 |
|  | 6 | ∞ | 1 | 2 | ∞ | ∞ | 0 | ∞ | 5 |
|  | 7 | ∞ | ∞ | ∞ | 5 | 4 | ∞ | 0 | 8 |
|  | 8 | ∞ | ∞ | 1 | ∞ | 4 | 5 | 8 | 0 |

Представим матрицу D2 (Таблица 3). включив в нее рассчитанные элементы из приложения.

Таблица 3 - Матрица D2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  D2= |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  | 1 | 0 | 1 | 2 | 9 | 9 | 2 | ∞ | ∞ |
|  | 2 | 1 | 0 | 1 | 10 | 10 | 1 | ∞ | ∞ |
|  | 3 | 2 | 1 | 0 | 11 | 11 | 2 | ∞ | 1 |
|  | 4 | 9 | 10 | 11 | 0 | 9 | 11 | 5 | ∞ |
|  | 5 | 9 | 10 | 11 | 9 | 0 | 11 | 4 | 4 |
|  | 6 | 2 | 1 | 2 | 11 | 11 | 0 | ∞ | 5 |
|  | 7 | ∞ | ∞ | ∞ | 5 | 4 | ∞ | 0 | 8 |
|  | 8 | ∞ | ∞ | 1 | ∞ | 4 | 5 | 8 | 0 |

Представим матрицу D3 (Таблица 4). включив в нее рассчитанные элементы из приложения.

Таблица 4 - Матрица D3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D3= |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  | 1 | 0 | 1 | 2 | 9 | 9 | 2 | ∞ | 3 |
|  | 2 | 1 | 0 | 1 | 10 | 10 | 1 | ∞ | 2 |
|  | 3 | 2 | 1 | 0 | 11 | 11 | 2 | ∞ | 1 |
|  | 4 | 9 | 10 | 11 | 0 | 9 | 11 | 5 | 12 |
|  | 5 | 9 | 10 | 11 | 9 | 0 | 11 | 4 | 4 |
|  | 6 | 2 | 1 | 2 | 11 | 11 | 0 | ∞ | 3 |
|  | 7 | ∞ | ∞ | ∞ | 5 | 4 | ∞ | 0 | 8 |
|  | 8 | 3 | 2 | 1 | 12 | 4 | 3 | 8 | 0 |

Представим матрицу D4 (Таблица 5). включив в нее рассчитанные элементы из приложения.

Таблица 5 - Матрица D4

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D4= |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  | 1 | 0 | 1 | 2 | 9 | 9 | 2 | 14 | 3 |
|  | 2 | 1 | 0 | 1 | 10 | 10 | 1 | 15 | 2 |
|  | 3 | 2 | 1 | 0 | 11 | 11 | 2 | 16 | 1 |
|  | 4 | 9 | 10 | 11 | 0 | 9 | 11 | 5 | 12 |
|  | 5 | 9 | 10 | 11 | 9 | 0 | 11 | 4 | 4 |
|  | 6 | 2 | 1 | 2 | 11 | 11 | 0 | 16 | 3 |
|  | 7 | 14 | 15 | 16 | 5 | 4 | 16 | 0 | 8 |
|  | 8 | 3 | 2 | 1 | 12 | 4 | 3 | 8 | 0 |

Представим матрицу D5 (Таблица 6). включив в нее рассчитанные элементы из приложения.

Таблица 6 - Матрица D5

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D5= |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  | 1 | 0 | 1 | 2 | 9 | 9 | 2 | 13 | 3 |
|  | 2 | 1 | 0 | 1 | 10 | 10 | 1 | 14 | 2 |
|  | 3 | 2 | 1 | 0 | 11 | 11 | 2 | 15 | 1 |
|  | 4 | 9 | 10 | 11 | 0 | 9 | 11 | 5 | 12 |
|  | 5 | 9 | 10 | 11 | 9 | 0 | 11 | 4 | 4 |
|  | 6 | 2 | 1 | 2 | 11 | 11 | 0 | 15 | 3 |
|  | 7 | 13 | 14 | 15 | 5 | 4 | 15 | 0 | 8 |
|  | 8 | 3 | 2 | 1 | 12 | 4 | 3 | 8 | 0 |

Представим матрицу D6. (Таблица 7). включив в нее рассчитанные элементы из приложения.

Таблица 7 - Матрица D6

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D6= |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  | 1 | 0 | 1 | 2 | 9 | 9 | 2 | 13 | 3 |
|  | 2 | 1 | 0 | 1 | 10 | 10 | 1 | 14 | 2 |
|  | 3 | 2 | 1 | 0 | 11 | 11 | 2 | 15 | 1 |
|  | 4 | 9 | 10 | 11 | 0 | 9 | 11 | 5 | 12 |
|  | 5 | 9 | 10 | 11 | 9 | 0 | 11 | 4 | 4 |
|  | 6 | 2 | 1 | 2 | 11 | 11 | 0 | 15 | 3 |
|  | 7 | 13 | 14 | 15 | 5 | 4 | 15 | 0 | 8 |
|  | 8 | 3 | 2 | 1 | 12 | 4 | 3 | 8 | 0 |

Представим матрицу D7. (Таблица 8). включив в нее рассчитанные элементы из приложения.

Таблица 8 - Матрица D7

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D7= |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  | 1 | 0 | 1 | 2 | 9 | 9 | 2 | 13 | 3 |
|  | 2 | 1 | 0 | 1 | 10 | 10 | 1 | 14 | 2 |
|  | 3 | 2 | 1 | 0 | 11 | 11 | 2 | 15 | 1 |
|  | 4 | 9 | 10 | 11 | 0 | 9 | 11 | 5 | 12 |
|  | 5 | 9 | 10 | 11 | 9 | 0 | 11 | 4 | 4 |
|  | 6 | 2 | 1 | 2 | 11 | 11 | 0 | 15 | 3 |
|  | 7 | 13 | 14 | 15 | 5 | 4 | 15 | 0 | 8 |
|  | 8 | 3 | 2 | 1 | 12 | 4 | 3 | 8 | 0 |

Представим матрицу D8, включив в нее рассчитанные элементы.

Таблица 9 - Матрица D8

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D8= |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  | 1 | 0 | 1 | 2 | 9 | 7 | 2 | 11 | 3 |
|  | 2 | 1 | 0 | 1 | 10 | 6 | 1 | 10 | 2 |
|  | 3 | 2 | 1 | 0 | 11 | 5 | 2 | 9 | 1 |
|  | 4 | 9 | 10 | 11 | 0 | 9 | 11 | 5 | 12 |
|  | 5 | 7 | 6 | 5 | 9 | 0 | 7 | 4 | 4 |
|  | 6 | 2 | 1 | 2 | 11 | 7 | 0 | 11 | 3 |
|  | 7 | 11 | 10 | 9 | 5 | 4 | 11 | 0 | 8 |
|  | 8 | 3 | 2 | 1 | 12 | 4 | 3 | 8 | 0 |

В результате, нами получена матрица длин кратчайших путей между каждой парой вершин графа. Ниже представлена таблица путей. Каждый элемент Cij таблицы, это путь из вершины i в вершину j:

Таблица 10- Таблица путей

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i/j | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | - | d1-2=1-2 | d1-3=1-2-3 | d1-4=1-4 | d1-5=1-2-3-8-5 | d1-6=1-2-6 | d1-7=1-2-3-8-7 | d1-8=1-2-3-8 |
| 2 | d2-1=2-1 | - | d2-3=2-3 | d2-4=2-1-4 | d2-5=2-3-8-5 | d2-6=2-6 | d2-7=2-3-8-7 | d2-8=2-3-8 |
| 3 | d3-1=3-2-1 | d3-2=3-2 | - | d3-4=3-2-1-4 | d3-5=3-8-5 | d3-6=3-6 | d3-7=3-8-7 | d3-8=3-8 |
| 4 | d4-1=4-1 | d4-2=4-1-2 | d4-3=4-1-2-3 | - | d4-5=4-5 | d4-6=4-1-2-6 | d4-7=4-7 | d4-8=4-1-2-3-8 |
| 5 | d5-1=5-8-3-2-1 | d5-2=5-8-3-2 | d5-3=5-8-3 | d5-4=5-4 | - | d5-6=5-8-3-6 | d5-7=5-7 | d5-8=5-8 |
| 6 | d6-1=6-2-1 | d6-2=6-2 | d6-3=6-3 | d6-4=6-2-1-4 | d6-5=6-3-8-5 | - | d6-7=6-3-8-7 | d6-8=6-3-8 |
| 7 | d7-1=7-8-3-2-1 | d7-2=7-8-3-2 | d7-3=7-8-3 | d7-4=7-4 | d7-5=7-5 | d7-6=7-8-3-6 | - | d7-8=7-8 |
| 8 | d8-1=8-3-2-1 | d8-2=8-3-2 | d8-3=8-3 | d8-4=8-3-2-1-4 | d8-5=8-5 | d8-6=8-3-6 | d8-7=8-7 | - |

В результате следуя из таблицы 10 и таблицы 9, кратчайший путь из вершины 1 в вершину 8, проходчит через вершины 1-2-3-8 и имеет вес равный 3.

# . Создание приложения для решения задачи

## 7.1 Описание алгоритма

Ниже приведен алгоритм (рисунок 3) работы программы в упрощенном виде, для лучшего восприятия хода работы программы.



Рисунок 12 - Алгоритм работы программы

7.2 Разработка программы

Для программной реализации алгоритма Флойда был использован язык программирования Delphi 7. Для оформления формы были использованы такие компоненты как: button (кнопка возврата условия, а так же для нахождения пути), StringGrid (отвечает за количество вершин), edit (для установки из какой в какую вершину будем искать кратчайший путь ), label (комментарий), LabelPath (вывод ответа), mainmenu (выпадающее меню).

В программном коде, было использовано всего две функции для обращения к алгоритму Флойда и восстанавливленния самого кратчайшего пути между любыми двумя заданными вершинами.

Остальной код программы состоит из процедур, отвечающих за функции такие как:

- чтение матрицы из Edit - procedure TForm1.InputMatrix;

- нахождение перспективной пары из множества конкурирующих пар - procedure TForm1.Konkurir;

- привидение (вычитание минимума элементов) матрицы, нахождение нижней оценки (границы) - procedure TForm1.Etap;

- вычеркивание из матрицы строки и столбца, определение новой диагонали - procedure TForm1.DelStrStolb;

- нахождение оптимального пути - procedure TForm1.OpredilPuti;

- проверка на замкнутость пути - procedure ProverkaIskl;

- процедура, описывающая нажатие на кнопку "найти" - procedure TForm3.Button2Click;

- увеличение/уменьшение количествава вершин указанных в Edit1 - procedure TForm3.Button1Click;

- несколько простейших процедур, описывающих выпадающее меню.

В практически всех процедурах используются циклы с параметрами, для перебора всех имеющихся строк и столбцов.

## 7.3 Тестирование программы

Введем исходные данные в матрицу StrinGrid (рисунок 13).



Рисунок 13 - Ввод исходных данных

Вводим значения в edit’ы, от какой и до какой вершины желаем найти кратчайший путь (рисунок 14).



Рисунок 14 - Ввод вершин

Нажимаем кнопку «Найти» и смотрим результат (рисунок 15).



Рисунок 15 - Результат работы

Длина пути, найденное программным способом, совпадает с аналитическим. Можно сделать вывод, что и программное и аналитическое решение является верным.

## 7.4 Руководство пользователя

Запустив приложение Floid.exe, откроется окно позволяющее решать задачи методом алгоритма Флойда-Уоршелла

Все что вам требуется - это:

- ввести количество вершин;

- ввести вершины между которыми желаем найти кратчайший путь;

- нажать кнопку Найти;

- посмотреть результат работы программы.

При необходимости можно очистить форму при помощи кнопки «Очистить».

В представленной программе можно ввести данные задачи двумя способами: вручную, либо программно, нажав на кнопку «Задача».

Заключение

Теория графов находит широкое применение в различных областях науки и техники, будь то физика, химия, математика и пр.

В процессе выполнения данной курсовой работы, цель была достигнута, поставленные задачи выполнены.

В начале курсовой работы была рассмотрена теория графов в общем, изучены понятия путей и маршрутов, постановку задач коммивояжера, понятие транспортной сети и алгоритм Флойда-Уоршелла. Данная мне тема «Задачи на графах» может найти свое применение в программах для транспортных и коммуникационных сетей, таких как коммутация информационного пакета в Internet, где вершины - роутеры, а ребра это связи между роутерами. Таким образом, чем короче будет путь, тем быстрее пакет достигнет пункта назначения и меньше будут информационные потери.

В данной курсовой работе реализованы поставленные задачи с помощью языка программирования Delphi, был разработан программный продукт «Алгоритм Флойда» для нахождения кратчайших путей графа между его вершинами. Преимуществом данной программы является то, что в ней реализованы принципы динамического программирования, то есть, пользователь сам определяет количество вершин графа, количество ребер (дуг) и их вес. В программе использован алгоритм Флойда-Уоршелла, который помогает последовательно вычислить все значения длин кратчайших путей между вершинами.

# Список использованных источников

1 Бурков, В.Н., Заложнев, А.Ю., Новиков, Д.А. Теория графов в управлении организационными системами. - Москва: Синтег, 2001.

 Новиков, Ф.А. Дискретная математика для программистов. - Санкт-Петербург: Питер, 2002.

 Новиков, Ф.А. Дискретная математика: Учебник для вузов. Стандарт третьего поколения. Санкт-Петербург: Питер, 2011.

 Прилуцкий, М.Х. Математические основы информатики. - Нижний Новгород, 2000.

 Партыка, Т.Л., Попов, И.И. Математические методы- Москва, 2005.