**Графы. Решение практических задач с использованием графов (С++)**

Курсовая работа

Выполнил: студент 1-го курса факультета КНиИТ, группа № 121, Жучков Андрей Сергеевич

Саратовский государственный университет им. Н.Г. Чернышевского

Кафедра теоретических основ информатики и информационных технологий

Саратов 2005

**Введение**

В последнее время исследования в областях, традиционно относящихся к дискретной математике, занимают все более заметное место. Наряду с такими классическими разделами математики, как математический анализ, дифференциальные уравнения, в учебных планах специальности "Прикладная математика" и многих других специальностей появились разделы по математической логике, алгебре, комбинаторике и теории графов. Причины этого нетрудно понять, просто обозначив круг задач, решаемых на базе этого математического аппарата.

**История возникновения теории графов.**

Родоначальником теории графов принято считать математика Леонарда Эйлера (1707-1783). Однако теория графов многократно переоткрывалась разными авторами при решении различных прикладных задач.

Задача о Кенигсбергских мостах. На рис. 1 представлен схематический план центральной части города Кенигсберг (ныне Калининград), включающий два берега реки Перголя, два острова в ней и семь соединяющих мостов. Задача состоит в том, чтобы обойти все четыре части суши, пройдя по каждому мосту один раз, и вернуться в исходную точку. Эта задача была решена (показано, что решение не существует) Эйлером в 1736 году.

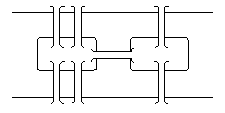


рис. 1

Задача о трех домах и трех колодцах. Имеется три дома и три колодца, каким-то образом расположенные на плоскости. Провести от каждого дома к каждому колодцу тропинку так, чтобы тропинки не пересекались (рис. 2). Эта задача была решена (показано, что решение не существует) Куратовским в 1930 году.

рис. 2

Задача о четырех красках. Разбиение на плоскости на непересекающиеся области называется картой. Области на карте называются соседними, если они имеют общую границу. Задача состоит в раскрашивании карты таким образом, чтобы никакие две соседние области не были закрашены одним цветом (рис. 3). С конца позапрошлого века известна гипотеза, что для этого достаточно четырех красок. В 1976 году Аппель и Хейкен опубликовали решение задачи о четырех красках, которое базировалось на переборе вариантов с помощью компьютера. Решение этой задачи «программным путем» явилось прецедентом, породившим бурную дискуссию, которая отнюдь не закончена. Суть опубликованного решения состоит в том, чтобы перебрать большое, но конечное число (около 2000) типов потенциальных контрпримеров к теореме о четырех красках и показать, что ни один случай контрпримером не является. Этот перебор был выполнен программой примерно за тысячу часов работы суперкомпьютера. Проверить «вручную» полученное решение невозможно – объем перебора выходит далеко за рамки человеческих возможностей. Многие математики ставят вопрос: можно ли считать такое «программное доказательство» действительным доказательством? Ведь в программе могут быть ошибки… Методы формального доказательства правильности программ не применимы к программам такой сложности, как обсуждаемая. Тестирование не может гарантировать отсутствие ошибок и в данном случае вообще невозможно. Таким образом, остается уповать на программистскую квалификацию авторов и верить, что они сделали все правильно.

1

Рис. 3

Основные понятия теории графов

Графом G(V,E) называется совокупность двух множеств – непустого множества V(множества вершин) и множества E двухэлементных подмножеств множества V(E – множество ребер).

Ориентированным называется граф, в котором  - множество упорядоченных пар вершин вида (x,y), где x называется началом, а y – концом дуги. Дугу (x, y) часто записывают как . Говорят также, что дуга ведет от вершины x к вершине y, а вершина y смежная с вершиной x.

Если элементом множества E может быть пара одинаковых (не различных) элементов V, то такой элемент множества E называется петлей, а граф называется графом с петлями (или псевдографом).

Если E является не множеством, а набором, содержащим несколько одинаковых элементов, то эти элементы называются кратными ребрами, а граф называется мультиграфом.

Если элементами множества E являются не обязательно двухэлементные, а любые подмножества множества V, то такие элементы множества E называются гипердугами, а граф называется гиперграфом.

Если задана функция F : V → M и/или F : E → M, то множество M называется множеством пометок, а граф называется помеченным (или нагруженным). В качестве множества пометок обычно используются буквы или целые числа. Если функция F инъективна, то есть разные вершины (ребра)имеют разные пометки, то граф называют нумерованным.

Подграфом называется граф G′(V′,E′), где и/или .

Если V′ = V, то G′ называется остовным подграфом G.

Если , то граф G′ называется собственным подграфом графа G.

Подграф G′(V′,E′) называется правильным подграфом графа G(V,E), если G′ содержит все возможные рёбра G.

Степень (валентность) вершины – это количество ребер, инцидентных этой вершине (количество смежных с ней вершин).

Маршрутом в графе называется чередующаяся последовательность вершин и ребер , в которой любые два соседних элемента инциденты.

Если , то маршрут замкнут, иначе открыт.

Если все ребра различны, то маршрут называется цепью.

Если все вершины (а значит, и ребра) различны, то маршрут называется простой цепью.

Замкнутая цепь называется циклом.

Замкнутая простая цепь называется простым циклом.

Граф без циклов называется ациклическим.

Для орграфов цепь называется путем, а цикл – контуром.

V1 V2

V3

V4 V5

рис. 4. Маршруты, цепи, циклы

Пример

В графе, диаграмма которого приведена на рис.4:

v1, v3, v1, v4 – маршрут, но не цепь;

v1, v3, v5, v2, v3, v4 – цепь, но не простая цепь;

v1, v4, v3, v2, v5 – простая цепь;

v1, v3, v5, v2, v3, v4, v1 – цикл, но не простой цикл;

v1, v3, v4, v1 – простой цикл.

Если граф имеет цикл (не обязательно простой), содержащий все ребра графа по одному разу, то такой цикл называется эйлеровым циклом.

Если граф имеет простой цикл, содержащий все вершины графа (по одному разу), то такой цикл называется гамильтоновым циклом.

Деревом называется связный граф без циклов.

Остовом называется дерево, содержащее все вершины графа.

Паросочетанием называется множество ребер, в котором никакие два не смежны.

Паросочетание называется максимальным, если никакое его надмножество не является независимым.

Две вершины в графе связаны, если существует соединяющая их простая цепь.

Граф, в котором все вершины связаны, называется связным.

Граф, состоящий только из изолированных вершин, называется вполне несвязным.

Длиной маршрута называется количество ребер в нем (с повторениями).

Расстоянием между вершинами u и v называется длина кратчайшей цепи , а сама кратчайшая цепь называется геодезической.

Диаметром графа G называется длина длиннейшей геодезической.

Эксцентриситетом вершины v в связном графе G(V,E) называется максимальное расстояние от вершины v до других вершин графа G.

Радиусом графа G называется наименьший из эксцентриситетов вершин.

Вершина v называется центральной, если ее эксцентриситет совпадает с радиусом графа.

Множество центральных вершин называется центром графа.

3

2 2

3 3

3

1. 3

4 4

2

3 3

рис. 5 Эксцентриситеты вершин и центры графов (выделены).

Основные теоремы теории графов

Опираясь на приведенные выше определения теории графов, приведем формулировки и доказательства теорем, которые затем найдут свои приложения при решении задач.

Теорема 1. Удвоенная сумма степеней вершин любого графа равна числу его ребер.

Доказательство. Пусть А1, А2, А3, ..., An — вершины данного графа, a p(A1), p(А2), ..., p(An) – степени этих вершин. Подсчитаем число ребер, сходящихся в каждой вершине, и просуммируем эти числа. Это равносильно нахождению суммы степеней всех вершин. При таком подсчете каждое ребро будет учтено дважды (оно ведь всегда соединяет две вершины).

Отсюда следует: p(A1)+p(А2)+ ... +p(An)=0,5N, или 2(p(A1)+p(А2)+ ... +p(An))=N , где N — число ребер.

Теорема 2. Число нечетных вершин любого графа четно.

Доказательство. Пусть a1, a2, a3, …, ak — это степени четных вершин графа, а b1, b2, b3, …, bm — степени нечетных вершин графа. Сумма a1+a2+a3+…+ak+b1+b2+b3+…+bm ровно в два раза превышает число ребер графа. Сумма a1+a2+a3+…+ak четная (как сумма четных чисел), тогда сумма b1+b2+b3+…+bm должна быть четной. Это возможно лишь в том случае, если m — четное, то есть четным является и число нечетных вершин графа. Что и требовалось доказать.

Эта теорема имеет немало любопытных следствий.

Следствие 1. Нечетное число знакомых в любой компании всегда четно.

Следствие 2. Число вершин многогранника, в которых сходится нечетное число ребер, четно.

Следствие 3. Число всех людей, когда-либо пожавших руку другим людям, нечетное число раз, является четным.

Теорема 3. Во всяком графе с n вершинами, где n больше или равно 2, всегда найдутся две или более вершины с одинаковыми степенями.

Доказательство. Если граф имеет n вершин, то каждая из них может иметь степень 0, 1, 2, ..., (n-1). Предположим, что в некотором графе все его вершины имеют различную степень, то есть, и покажем, что этого быть не может. Действительно, если р(А)=0, то это значит, что А — изолированная вершина, и поэтому в графе не найдется вершины Х со степенью р(Х)=n-1. В самом деле, эта вершина должна быть соединена с (n-1) вершиной, в том числе и с А, но ведь А оказалась изолированной. Следовательно, в графе, имеющем n вершин, не могут быть одновременно вершины степени 0 и (n-1). Это значит, что из n вершин найдутся две, имеющие одинаковые степени.

Теорема 4. Если в графе с n вершинами (n больше или равно 2) только одна пара имеет одинаковую степень, то в этом графе всегда найдется либо единственная изолированная вершина, либо единственная вершина, соединенная со всеми другими.

Доказательство данной теоремы мы опускаем. Остановимся лишь на некотором ее пояснении. Содержание этой теоремы хорошо разъясняется задачей: группа, состоящая из n школьников, обменивается фотографиями. В некоторый момент времени выясняется, что двое совершили одинаковое число обменов. Доказать, что среди школьников есть либо один еще не начинавший обмена, либо один уже завершивший его.

Теорема 5. Если у графа все простые циклы четной длины, то он не содержит ни одного цикла четной длины.

Суть теоремы в том, что на этом графе невозможно найти цикл (как простой, так и непростой) нечетной длины, то есть содержащий нечетное число ребер.

Теорема 6. Для того, чтобы граф был эйлеровым, необходимо и достаточно, чтобы он был связным и все его вершины имели четную степень.

Теорема 7. Для того чтобы на связном графе можно было бы проложить цепь АВ, содержащую все его ребра в точности по одному разу, необходимо и достаточно, чтобы А и В были единственными нечетными вершинами этого графа.

Доказательство этой теоремы очень интересно и характерно для теории графов. Его также следует считать конструктивным (обратите внимание на то, как использована при этом теорема 3.6). Для доказательства к исходному графу присоединяем ребро (А, В); после этого все вершины графа станут четными. Этот новый граф удовлетворяет всем условиям теоремы 3.6, и поэтому в нем можно проложить эйлеров цикл Ψ. И если теперь в этом цикле удалить ребро (А, В), то останется искомая цепь АВ.

На этом любопытном приеме основано доказательство следующей теоремы, которую следует считать обобщением теоремы 7.

Теорема 8. Если данный граф является связным и имеет 2k вершин нечетной степени, то в нем можно провести k различных цепей, содержащих все его ребра в совокупности ровно по одному разу.

Теорема 9. Различных деревьев с n перенумерованными вершинами можно построить nn-2.

По поводу доказательства этой теоремы сделаем одно замечание. Эта теорема известна, в основном, как вывод английского математика А. Кэли (1821—1895). Графы-деревья издавна привлекали внимание ученых. Сегодня двоичные деревья используются не только математиками, а и биологами, химиками, физиками и инженерами (подробнее об этом – в параграфе 6).

Теорема 10. Полный граф с пятью вершинами не является плоским.

Доказательство. Воспользуемся формулой Эйлера: В-Р+Г=2, где В — число вершин плоского графа, Р — число его ребер, Г — число граней. Формула Эйлера справедлива для плоских связных графов, в которых ни один из многоугольников не лежит внутри другого.

Эту формулу можно доказать методом математической индукции. Это доказательство мы опускаем. Заметим только, что формула справедлива и для пространственных многогранников. Пусть все пять вершин графа соединены друг с другом. Замечаем, что на графе нет ни одной грани, ограниченной только двумя ребрами. Если через φ1 обозначить число таких граней, то φ2=0. Далее рассуждаем от противного, а именно: предположим, что исследуемый граф плоский. Это значит, что для него верна формула Эйлера. Число вершин в данном графе В=5, число ребер Р=10, тогда число граней Г=2-В+Р=2-5+10=7.

Это число можно представить в виде суммы: Г=φ1+φ2+φ3+…, где φ3 – число граней, ограниченных тремя ребрами, φ4 — число граней, ограниченных четырьмя ребрами и т. д.

С другой стороны, каждое ребро является границей двух граней, а поэтому число граней равно 2Р, в то же время 2Р=20=3φ3+4φ4+... . Умножив равенство Г=7=φ3+ φ4 + φ5 + … на три, получим ЗГ=21=3( φ3 + φ4 + φ5 + …).

Ясно, что (3φ3+3φ4+3φ5+…) < (3φ3+4φ4+ 5φ5+…) или 3Г<2Р, но по условию, 2Р=20, а ЗГ=21; поэтому вывод, полученный при введенном нами предположении (граф плоский), противоречит условию. Отсюда заключаем, что полный граф с пятью вершинами не является плоским.

Теорема 11. (Теорема Понтрягина-Куратовского) Граф является плоским тогда и только тогда, когда он не имеет в качестве подграфа полного графа с пятью вершинами.

В заключение этого параграфа, на наш взгляд, следует упомянуть то, что в нем объяснялись только основные теоремы теории графов. Их практическое применение будет рассмотрено в следующих параграфах реферата.

Способы представления графов в компьютере

Конструирование структур данных для представления в программе объектов математической модели – это основа искусства практического программирования. Далее приводится четыре различных базовых представления графов. Выбор наилучшего представления определяется требованиями конкретной задачи. Более того, при решении конкретных задач используются, как правило, некоторые комбинации или модификации указанных представлений, общее число которых необозримо. Но все они так или иначе основаны на тех базовых идеях, которые описаны в этом разделе.

Требования к представлению графов

Известны различные способы представления графов в памяти компьютера, которые различаются объемом занимаемой памяти и скоростью выполнения операций над графами. Представление выбирается, исходя из потребностей конкретной задачи. Далее приведены четыре наиболее часто используемых представления с указанием характеристики n(p,q) – объема памяти для каждого представления. Здесь p – число вершин, а q – число ребер.

Матрица смежности

Представление графа с помощью квадратной булевой матрицы M, отражающей смежность вершин, называется матрицей смежности, где



Для матрицы смежности n(p,q) = O(p2).

Замечание

Матрица смежности неориентированного графа симметрична относительно главной диагонали, поэтому достаточно хранить только верхнюю (или нижнюю) треугольную матрицу.

Матрица инциденций

Представление графа с помощью матрицы H, отражающей инцидентность вершин и ребер, называется матрицей инциденций, где для неориентированного графа



а для орграфа



Для матрицы инциденций n(p,q) = O(pq).

Списки смежности

Представление графа с помощью списочной структуры, отражающей смежность вершин и состоящей из массива указателей на списки смежных вершин, где элемент списка представлен структурой

N : record v : 1..p; n :↑ N end record,

называется списком смежности. В случае представления неориентированных графов списками смежности n(p,q) = O(p+2q), а в случае ориентированных графов n(p,q) = O(p+q).

Массив дуг

Представление графа с помощью массива структур

E : array [1..q] of record b,e : 1..p end record,

отражающего список пар смежных вершин, называется массивом ребер (или, для орграфов, массивом дуг). Для массива ребер (или дуг) n(p,q) = O(2q).

Обзор задач теории графов

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.

Графы нашли применение практически во всех отраслях научных знаний: физике, биологии, химии, математике, истории, лингвистике, 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.

Далее перечислим некоторые типовые задачи теории графов и их приложения:

- Задача о кратчайшей цепи

замена оборудования

составление расписания движения транспортных средств

размещение пунктов скорой помощи

размещение телефонных станций

- Задача о максимальном потоке

анализ пропускной способности коммуникационной сети

организация движения в динамической сети

оптимальный подбор интенсивностей выполнения работ

синтез двухполюсной сети с заданной структурной надежностью

задача о распределении работ

- Задача об упаковках и покрытиях

оптимизация структуры ПЗУ

размещение диспетчерских пунктов городской транспортной сети

- Раскраска в графах

распределение памяти в ЭВМ

проектирование сетей телевизионного вещания

- Связность графов и сетей

проектирование кратчайшей коммуникационной сети

синтез структурно-надежной сети циркуляционной связи

анализ надежности стохастических сетей связи

- Изоморфизм графов и сетей

структурный синтез линейных избирательных цепей

автоматизация контроля при проектировании БИС

- Изоморфное вхождение и пересечение графов

локализация неисправности с помощью алгоритмов поиска МИПГ

покрытие схемы заданным набором типовых подсхем

- Автоморфизм графов

конструктивное перечисление структурных изомеров для

производных органических соединений

синтез тестов цифровых устройств

**Заключение**

В работе были рассмотрены задачи из теории графов, которые уже стали классическими. Особенно часто в практическом программировании возникают вопросы о построении кратчайшего остова графа и нахождении максимального паросочетания. Известно также, что задача о нахождении гамильтонова цикла принадлежит к числу NP-полных, т.е. эффективный алгоритм для ее решения не найден – приведенная программа ищет цикл методом перебора. Все задачи реализованы на языке С++, листинги которых приводятся в приложении А, примеры входных и выходных данных – в приложении Б.

**Список литературы**

Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. – М: МЦНМО, 2001

Н. Кристофидес. Теория графов: алгоритмический подход, Мир, 1978.

Ф.А. Новиков. Дискретная математика для программистов, Питер, 2001.

В.А. Носов. Комбинаторика и теория графов, МГТУ, 1999.

Приложение А

1. Гамильтоновым циклом в графе называется цикл, проходящий через все вершины графа ровно один раз. Для данного графа найти гамильтонов цикл, если он существует.

#include <iostream.h>

#include <stdio.h>

FILE\* fi = fopen("g\_graph.txt","r"); // Файл с матрицей смежности

FILE\* fo = fopen("g\_cycle.txt","w"); // Файл с найденными циклами

bool \*\*graph; //Матрица смежности для хранения графа

int n; //Количество вершин в графе

const int vertex = 1; // первая вершина при поиске

int \*St; //Массив для хранения просмотренных вершин

int \*Nnew; //Массив признаков: вершина просмотрена или нет

void out\_way(int n){ //Процедура вывода Гамильтонова цикла

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

fprintf(fo,"%d\ ",St[i]);

fprintf(fo,"%d\n",vertex);

}

void gamilton\_path(int k){

int v = St[k-1]; // текущая вершина

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

if(graph[v][j]==1) // есть ребро между v и j

if(k==n && j==vertex)out\_way(n); //прошли все вершины

else

if(Nnew[j]){ // вершина не просмотрена

St[k]=j; // добавляем ее к пройденному пути

Nnew[j]=false; // просмотрена

gamilton\_path(k+1); //строим путь дальше

Nnew[j]=true; //возвращаемся назад и строим другие циклы

}

}

int main(){

fscanf(fi,"%d",&n); //Считываем количество вершин

St = new int[n];

Nnew = new int[n];

for(int i = 0;i < n;i++)Nnew[i]=1; // Нет просмотренных вершин

graph = new bool\*[n];

for(int i=0;i<n;i++){

graph[i] = new bool[n]; //выделяем память под строку

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

fscanf(fi,"%d",&graph[i][j]);

}

}

Nnew[vertex]=false; //первая вершина уже просмотрена

St[0]=vertex; // вершина с которой начали проход

gamilton\_path(1);//параметр означает количество пройденных вершин

fcloseall();

return(0);

}

2.Эйлеровым циклом называется цикл в графе, проходящий через все ребра графа ровно один раз. Для данного графа найти эйлеров цикл, если он существует.

#include <iostream.h>

#include <stdio.h>

struct Node{

int inf;

Node \*next;

};

//============================Stack==============================

Node \*init(){ // Инициализация стека

return NULL;

}

void push(Node \*&st,int dat){ // Загрузка числа в стек

Node \*el = new Node;

el->inf = dat;

el->next = st;

st=el;

}

int pop(Node \*&st){ // Извлечение из стека

int value = st->inf;

Node \*temp = st;

st = st->next;

delete temp;

return value;

}

int peek(Node \*st){ // Получение числа без его извлечения

return st->inf;

}

//==============================================================

Node \*\*graph; // Массив списков смежности

const int vertex = 1; // Первая вершина

FILE\* fi = fopen("e\_graph.txt","r"); //Файл с матрицей смежности

FILE\* fo = fopen("e\_cycle.txt","w"); // Результирующий файл

void add(Node\*& list,int data){ //Добавление смежной вершины

if(!list){list=new Node;list->inf=data;list->next=0;return;}

Node \*temp=list;

while(temp->next)temp=temp->next;

Node \*elem=new Node;

elem->inf=data;

elem->next=NULL;

temp->next=elem;

}

void del(Node\* &l,int key){ // Удаление вершины key из списка

if(l->inf==key){Node \*tmp=l; l=l->next; delete tmp;}

else{

Node \*tmp=l;

while(tmp){

if(tmp->next) // есть следующая вершина

if(tmp->next->inf==key){ // и она искомая

Node \*tmp2=tmp->next;

tmp->next=tmp->next->next;

delete tmp2;

}

tmp=tmp->next;

}

}

}

bool eiler(Node \*\*gr,int num){ // Определение эйлеровости графа

int count;

for(int i=0;i<num;i++){ //проходим все вершины

count=0;

Node \*tmp=gr[i];

while(tmp){ // считаем степень

count++;

tmp=tmp->next;

}

if(count%2==1)return 0; // степень нечетная

}

return 1; // все степени четные

}

void eiler\_path(Node \*\*gr){ //Построение цикла

Node \*S = init();// Стек для пройденных вершин

int v=vertex;// 1я вершина (произвольная)

int u;

push(S,v); //сохраняем ее в стек

while(S){ //пока стек не пуст

v = peek(S); // текущая вершина

if(!gr[v]){ // если нет инцидентных ребер

v=pop(S); fprintf(fo,"%d\ ",v); //выводим вершину

}else {

u=gr[v]->inf; push(S,u); //проходим в следующую вершину

del(gr[v],u); del(gr[u],v); //удаляем пройденное ребро

}

}

}

int main(){

int n; // Количество вершин

int zn;// Текущее значение

fscanf(fi,"%d",&n); graph=new Node\*[n];

for(int i=0;i<n;i++)graph[i]=NULL;

for(int i=0;i<n;i++) // заполняем массив списков

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

fscanf(fi,"%d",&zn);

if(zn) add(graph[i],j);

}

if(eiler(graph,n))eiler\_path(graph);

else fprintf(fo,"Граф не является эйлеровым.");

return(0);

}

3.Построить остовное дерево минимальной стоимости для связанного взвешенного графа, используя алгоритм Прима.

#include <iostream.h>

#include <stdio.h>

#include <values.h>

FILE\* fi = fopen("p\_graph.txt","r"); //Входной файл

FILE\* fo = fopen("p\_ostov.txt","w"); //Выходной файл

struct edge{ // Структура для хранения ребра

int b,e;

int weigh;

};

int \*\*graph; // исходный граф

int p; // количество вершин в графе

edge \*SST; // остов

int num\_ver=0; // количество ребер в остове

int \*S; // вершины, включенные в остов

int num\_s=0; //количество вершин в остове

int calc\_ver(){ // подсчитывает число ребер в графе

int i=0,j,res=0;

while(i<p){j=i+1; while(j++<p)if(graph[i][j])res+=1; i++;}

return res;

}

bool added(int v){ // проверяет: добавлена вершина v в остов

for(int i=0;i<num\_s;i++)if(v==S[i])return 1;

return 0;

}

void prim(){ // построение остова

S = new int[calc\_ver()];

int \*alf = new int[p]; // ближайшие вершины, добавленные в остов

int \*bet = new int[p]; // длины ребер, соединяющие вершины с остовом

int u = 0; // произвольная вершина

S[num\_s] = u; num\_s++; // добавляем в остов

for(int v=0;v<p;v++)

if(graph[v][u]){alf[v] = u; bet[v] = graph[u][v];}//u - ближайшая вершина

else {alf[v] = -1; bet[v] = MAXINT;}

int w,x;

for(int i=0;i<p-1;i++){

x = MAXINT;

for(int v=0;v<p;v++){ // поиск ближайшей к остову вершины

if(bet[v]<x && !added(v)){w = v; x = bet[v];}

}

S[num\_s] = w; num\_s++; // добавляем найденную вершину к остову

// и ребро

SST[num\_ver].b=alf[w];SST[num\_ver].e=w;SST[num\_ver].weigh=graph[alf[w]][w];

num\_ver++;

for(int v=0;v<p;v++){

if(graph[v][w] && !added(v))// меняем ближайшую вершину остова

if(bet[v]>graph[v][w]){alf[v]=w; bet[v]=graph[v][w];}

}

}

}

void out(){

int weight=0;

fprintf(fo,"%d\n",num\_ver);

for(int i=0;i<num\_ver;i++){

fprintf(fo,"%2d\ %2d\ %2d\n",SST[i].b,SST[i].e,SST[i].weigh);

weight+=SST[i].weigh;

}

fprintf(fo,"%s%d","Вес найденного остова: ",weight);

}

int main(){

fscanf(fi,"%d",&p); //Считываем количество вершин

graph = new int\*[p];

for(int i=0;i<p;i++){

graph[i] = new int[p]; //выделяем память под строку

for(int j=0;j<p;j++){

fscanf(fi,"%d",&graph[i][j]);

}

}

SST = new edge[calc\_ver()];

prim();

out();

fcloseall();

return 0;

}

4. Построить остовное дерево минимальной стоимости для связанного взвешенного графа, используя алгоритм Краскала.

#include <iostream.h>

#include <stdio.h>

#include <conio.h>

FILE\* fi = fopen("k\_graph.txt","r"); //Входной файл

FILE\* fo = fopen("k\_ostov.txt","w"); //Выходной файл

struct edge{ // Структура для хранения ребра

int beg,end;

int weigh;

};

edge \*E; // массив с ребрами

edge \*SST; //остов

int num\_edge; //количество ребер в остове

int n; //количество ребер

bool \*nov; //признак прохода вершины

void sort(edge \*arr){ //сортировка ребер по весу

edge tmp;

for(int i=0;i<n-1;i++)

for(int j=n-1;j>i;j--)

if(arr[j].weigh<arr[j-1].weigh){

tmp = arr[j-1]; arr[j-1] = arr[j]; arr[j] = tmp;

}

}

int smezh(int v1,edge v2){

//определяет вершину смежную с v1 через ребро v2

if(v1==v2.beg)return(v2.end);

else if(v1==v2.end)return(v2.beg);

else return -1; // вершина v1 и ребро v2 не инцидентны

}

bool cycle(int v,int v0){ //определяет наличие цикла

nov[v] = 0; // v пройдена

for(int i=0;i<num\_edge-1;i++)

if(smezh(v,SST[i])!=-1) //если смежны

if(smezh(v,SST[i])==v0)return 1; //из конца ребра пришли в начало

else if(nov[smezh(v,SST[i])]) // новая вершина

if(cycle(smezh(v,SST[i]),v0))return 1;

nov[v] = 1; //возвращаемся назад

return 0;

}

void kruskal(){

num\_edge = 0; //остов пуст

sort(E); // упорядочивание E по возрастанию веса

for(int i=0;i<n;i++){

SST[num\_edge] = E[i]; num\_edge++; //добавляем ребро

// если появляется цикл - удаляем

if(cycle(SST[num\_edge-1].end,SST[num\_edge-1].beg))num\_edge--;

for(int i=0;i<n;i++)nov[i]=1;

}

}

void out(edge \*mas,int num){ // вывод ребер

int sum=0;

fprintf(fo,"%d\n",num); // количество ребер

for(int i=0;i<num;i++){

fprintf(fo,"%d\ %d\ %3d\n",mas[i].beg,mas[i].end,mas[i].weigh);

sum+=mas[i].weigh;

}

fprintf(fo,"%s%d","Вес найденного остова: ",sum);

}

int main(){

fscanf(fi,"%d",&n); //считываем количество ребер

E = new edge[n];

SST = new edge[n];

nov = new bool[n+1];

for(int i=0;i<n;i++)nov[i] = 1;

for(int i=0;i<n;i++) //заполняем множество ребер

fscanf(fi,"%d%d%d3",&E[i].beg,&E[i].end,&E[i].weigh);

kruskal(); //строим остов

out(SST,num\_edge); //выводим в файл

return 0;

}

5.Для заданного неориентированного графа найдите максимальное паросочетание.

#include <iostream.h>

#include <stdio.h>

#include <conio.h>

FILE\* fi = fopen("m\_graph.txt","r");

FILE\* fo = fopen("m\_par.txt","w");

struct edge{ // ребро графа

int b,e;

};

int n; //количество ребер

edge \*graph; // массив ребер

edge \*matching; // паросочетание

int num\_mat; //количество паросочетаний

bool smezh(edge q1,edge q2){ // 1 - если q1 и q2 смежны, иначе -0

return q1.b==q2.b||q1.b==q2.e||q1.e==q2.b||q1.e==q2.e;

}

void out(edge \*m,int num){

fprintf(fo,"%d\n",num); // количество ребер

for(int i=0;i<num;i++)

fprintf(fo,"%d\ %d\n",m[i].b,m[i].e);

}

bool bad(){//возвращает 1, если в паросочетании есть смежное ребро

for(int i=0;i<num\_mat-1;i++)

if(smezh(matching[i],matching[num\_mat-1]))return 1;

return 0;

}

void solve(){ //находит максимальное паросочетание

num\_mat = 0;

for(int i=0;i<n;i++){

matching[num\_mat]=graph[i];num\_mat++; // добавляем ребро

if(bad())num\_mat--; // если уже есть смежные - удаляем

}

}

int main(){

fscanf(fi,"%d",&n);

graph = new edge[n];

matching = new edge[n];

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

fscanf(fi,"%d%d",&graph[i].b,&graph[i].e);

solve();

out(matching,num\_mat);

fcloseall();

return 0;

}

Приложение Б

Примеры входных и выходных файлов

1.

Входной файл "g\_graph.txt":

5

0 0 1 1 0

0 0 0 1 1

1 0 0 0 1

1 1 0 0 0

0 1 1 0 0

Выходной файл "g\_cycle.txt":

0 2 4 1 3 0

0 3 1 4 2 0

2.

Входной файл " e\_graph.txt ":

5

0 1 1 1 1

1 0 1 1 1

1 1 0 1 1

1 1 1 0 1

1 1 1 1 0

Выходной файл "e\_cycle.txt":

1 4 3 2 4 0 3 1 2 0 1

3.

Входной файл " p\_graph.txt ":

9

0 4 5 2 0 0 0 0 0

4 0 7 0 0 0 6 0 0

5 7 0 3 0 0 0 0 0

2 0 3 0 0 0 0 0 0

0 0 0 0 0 7 8 0 0

0 0 0 0 7 0 1 0 0

0 6 0 0 8 1 0 3 1

0 0 0 0 0 0 3 0 2

0 0 0 0 0 0 1 2 0

Выходной файл "p\_ostov.txt":

8

0 3 2

3 2 3

0 1 4

1 6 6

6 5 1

6 8 1

8 7 2

5 4 7

Вес найденного остова: 26

4.

Входной файл " k\_graph.txt ":

5

0 1 1

1 2 2

2 3 1

0 3 4

1 3 2

Выходной файл "k\_ostov.txt":

3

0 1 1

2 3 1

1 2 2

Вес найденного остова: 4

5.

Входной файл " m\_graph.txt ":

7

0 1

1 4

0 3

3 2

1 3

2 0

2 5

Выходной файл "m\_par.txt":

2

0 1

3 2