**МИНОБРНАУКИ РОССИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

дисциплина: Дискретная математика

Лабораторная работа №4.5

тема: «**Анализ алгоритмов поиска кратчайшего пути во взвешенном орграфе между заданной парой вершин**»

Выполнил: ст. группы ВТ-22  
Макаров Даниил Сергеевич

Проверил: Рязанов Ю.Д.

Белгород

2018г.

**Задания**

1. Разработать и реализовать алгоритм построения случайного слабо связного взвешенного орграфа с заданным числом вершин *n* и дуг *m*.

2. Реализовать алгоритмы поиска кратчайшего пути во взвешенном

орграфе между заданной парой вершин:

1. Перебор простых цепей.

2. Метод ветвей и границ.

3. Алгоритм Дейкстры.

3. Разработать и написать программу, которая генерирует 100 случайных слабо связных взвешенных орграфов с заданным числом вер шин *n* и дуг *m*, для каждого орграфа находит кратчайший путь между каждой парой вершин тремя алгоритмами и определяет время выполнения каждого алгоритма. Выполнить программу при *n* = 8*,* 9 и 10. Результат для каждого *n* представить в виде таблицы .

n = 8

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Кол-во ребер | n | | (n^2+3n)/6 | | n^2/3 | | (n^2-n)/2 | |
| min | max | min | max | min | max | min | max |
| Перебор простых цепей | 0 | 0 | 0 | 44 | 5 | 369 | 6 | 1681 |
| Метод ветвей и границ | 0 | 16 | 0 | 31 | 0 | 57 | 0 | 44 |
| Флгоритм Дейкстры | 0 | 0 | 0 | 10 | 0 | 4 | 0 | 0 |

n = 9

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Кол-во ребер | n | | (n^2+3n)/6 | | n^2/3 | | (n^2-n)/2 | |
| min | max | min | max | min | max | min | max |
| Перебор простых цепей | 0 | 0 | 0 | 200 | 40 | 2840 | 140 | 16815 |
| Метод ветвей и границ | 0 | 31 | 0 | 30 | 0 | 31 | 0 | 23 |
| Флгоритм Дейкстры | 0 | 6 | 0 | 0 | 0 | 12 | 0 | 1 |

n = 10

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Кол-во ребер | n | | (n^2+3n)/6 | | n^2/3 | | (n^2-n)/2 | |
| min | max | min | max | min | max | min | max |
| Перебор простых цепей | 0 | 5 | 14 | 620 | 290 | 17509 | 1941 | 186618 |
| Метод ветвей и границ | 0 | 31 | 0 | 33 | 0 | 67 | 0 | 31 |
| Флгоритм Дейкстры | 0 | 0 | 0 | 10 | 0 | 0 | 0 | 13 |

**Содержание файла "main.c"**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <limits.h>

#define MAX 1000

#define N 10

void swap(int \*a, int \*b){

int t = \*a;

\*a = \*b;

\*b = t;

}

//Возвращает граф с числом вершин n и ребер m

int \*\*MakeGraf(int n, int m){

int \*\*G = malloc(n \* sizeof(int \*));

//заполнение матрицы смежности

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

G[i] = malloc(n \* sizeof(int));

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

G[i][j] = 0;

}

}

//Количество ребер

int k = 0;

while (k<m){

int x = rand() % n;

int y = rand() % n;

if (x!=y && !G[x][y]){

G[x][y] = G[y][x] = (1 + rand()) % 15;

k++;

}

}

return G;

}

void FreeGraf(int \*\*G, int n){

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

free(G[i]);

}

free(G);

}

//Ссылка на самый короткий путь, и длина этого короткого пути

int \*Wmin = NULL, Lmin = 99999, Nmin = 0;

//Записывает в old информацию из new

void minCopy(int \*\*old, int \*new, int n){

//Память под old перевыделяется, т.к. новая цепь может быть короче

if (\*old != NULL){

free(\*old);

}

Nmin = n;

\*old = malloc(Nmin\*sizeof(int));

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

(\*old)[i] = new[i];

}

}

//Ищет путь методом перебора простых цепей

//G - матрица смежности (взвешенного орграфа)

//n - размер матрицы смежности (кол-во вершин)

//i - заполняемое место (при первом вызове 0)

//S - вершина, из которой ведем поиск

//F - вершина, в которую ищется путь

//L - длина пути (сумма весов)

//W - путь, последовательность вершин

//V - множество посещенных вершин

//При первом вызове необходимо поставить на первое место S, и вызвать с i=1

//FindChains(G, n, 1, S, F, 0, W (W[0] = S), V)

void FindChains(int \*\*G, int n, int i, int S, int F, int L, int \*W, int \*V){

int weight;

//Если уже все вершины просмотрены - выход

if (i==n) return;

//Просматриваем все x, смежные с S и не посещенные

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

if (G[S][x] && !V[x]){

weight = G[S][x];

W[i] = x;

if (x == F){

//Если цепь пришла в F, и цепь короче предыдущей, заменим

if (L+weight < Lmin){

Lmin = L+weight;

minCopy(&Wmin, W, i+1);

}

}else{

V[x] = 1;

FindChains(G, n, i+1, x, F, L+weight, W, V);

V[x] = 0;

}

}

}

}

//Ищет путь методом ветвей и границ

void BranchesBorders(int \*\*G, int n, int i, int S, int F, int L, int \*W, int \*V){

int weight;

//Если уже все вершины просмотрены - выход

if (i==n) return;

//Просматриваем все x, смежные с S и не посещенные

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

if (G[S][x] && !V[x]){

weight = G[S][x];

W[i] = x;

//Если длина пути станет больше минимальной, поиск не имеет смысла

if (L+weight < Lmin){

if (x == F){

//Если цепь пришла в F, и цепь короче предыдущей, заменим

Lmin = L+weight;

minCopy(&Wmin, W, i+1);

}else{

V[x] = 1;

BranchesBorders(G, n, i+1, x, F, L+weight, W, V);

V[x] = 0;

}

}

}

}

}

//Алгоритм Дейкстры - из вершины v1 в v2

int deikstr(int v1, int v2, int\*\* g, int T[], int V[], int \*dl){

//v1 v2 - начальная и конечная вершины, g - граф

//V - множество вершин орграфа

// Т - массив для хранения дерева кратчайших путей

//D = d(xi) - кратчайшее расстояние от вершины v1 к вершине xi

int min,f,i, tmp=(\*dl);

int D[N]; // минимальное расстояние

for(i=0;i<N;i++)

{

D[i]=MAX;

V[i]=0;

T[i]=-1;

}

D[v1]=0;

V[v1]=1;

T[v1]=0;

while(v1!=v2 && f)

{

f=0;

for(i=0;i<N;i++)

{

if(g[v1][i] && D[v1]+g[v1][i]<D[i])

{

D[i]=D[v1]+g[v1][i];

T[i]=v1+1;

}

}

min=MAX;

for(i=0;i<N;i++)

if(V[i]==0 && min>D[i])

{

min=D[i];

v1=i;

f=1;

}

V[v1]=1;

}

int j;

for(i = 0; i < N; i++)

for(j = i+1; j < N; j++)

if(D[i] == D[j] && D[i] != MAX)

return 0;

tmp+=min;

(\*dl)=tmp;

return 1;

}

//Тест для поиска цепей

void Test1(int n, int m, long \*min\_, long \*max\_){

clock\_t start, stop, all;

all = 0;

long min = 99999999;

long max = 0;

int \*W = malloc(n \* sizeof \*W);

int \*V = malloc(n \* sizeof \*V);

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

V[i] = W[i] = 0;

}

for (int i = 0; i<100; i++){

int \*\*G = MakeGraf(n, m);

start=clock();

//Находятся кратчайшие пути между каждой парой вершин

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

W[0] = x;

Lmin = 999999;

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

FindChains(G, n, 1, x, y, 0, W, V);

}

}

FreeGraf(G, n);

stop=clock();

all += (stop-start);

if (all > max){

max = all;

}

if (all < min){

min = all;

}

}

\*min\_ = min;

\*max\_ = max;

}

//Тест для поиска цепей

void Test2(int n, int m, long \*min\_, long \*max\_){

clock\_t start, stop, all;

all = 0;

long min = 99999999;

long max = 0;

int \*W = malloc(n \* sizeof \*W);

int \*V = malloc(n \* sizeof \*V);

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

V[i] = W[i] = 0;

}

for (int i = 0; i<100; i++){

int \*\*G = MakeGraf(n, m);

start=clock();

//Находятся кратчайшие пути между каждой парой вершин

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

W[0] = x;

Lmin = 999999;

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

BranchesBorders(G, n, 1, x, y, 0, W, V);

}

}

FreeGraf(G, n);

stop=clock();

all += (stop-start);

if (all > max){

max = all;

}

if (all < min){

min = all;

}

}

\*min\_ = min;

\*max\_ = max;

}

//Тест для алг. Дейкстры

void Test3(int n, int m, long \*min\_, long \*max\_){

clock\_t start, stop, all;

all = 0;

long min = 99999999;

long max = 0;

for (int i = 0; i<100; i++){

int \*\*G = MakeGraf(n, m);

start=clock();

int dl = 0;

int T[N] = {0};

int V[N] = {0};

//Находятся кратчайшие пути между каждой парой вершин

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

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

deikstr(x, y, G, T, V, &dl);

}

}

FreeGraf(G, n);

stop=clock();

all += (stop-start);

if (all > max){

max = all;

}

if (all < min){

min = all;

}

}

\*min\_ = min;

\*max\_ = max;

}

int main() {

srand(time(NULL));

printf("Chains\n");

long min, max;

for (int n = 8; n<=10; n+=1){

int m = n;

Test1(n, m, &min, &max);

printf("n = %4i, m = %4i, time = %li min, %li max\n", n, m, min, max);

m = (n\*n+3\*n)/6;

Test1(n, m, &min, &max);

printf("n = %4i, m = %4i, time = %li min, %li max\n", n, m, min, max);

m = (n\*n)/3;

Test1(n, m, &min, &max);

printf("n = %4i, m = %4i, time = %li min, %li max\n", n, m, min, max);

m = (n\*n-n)/2;

Test1(n, m, &min, &max);

printf("n = %4i, m = %4i, time = %li min, %li max\n", n, m, min, max);

}

printf("Branches and Borders\n");

for (int n = 12; n<=14; n+=1){

int m = n;

Test2(n, m, &min, &max);

printf("n = %4i, m = %4i, time = %li min, %li max\n", n, m, min, max);

m = (n\*n+3\*n)/6;

Test2(n, m, &min, &max);

printf("n = %4i, m = %4i, time = %li min, %li max\n", n, m, min, max);

m = (n\*n)/3;

Test2(n, m, &min, &max);

printf("n = %4i, m = %4i, time = %li min, %li max\n", n, m, min, max);

m = (n\*n-n)/2;

Test2(n, m, &min, &max);

printf("n = %4i, m = %4i, time = %li min, %li max\n", n, m, min, max);

}

printf("Deikstra\n");

long min, max;

for (int n = 8; n<=10; n+=1){

int m = n;

Test3(n, m, &min, &max);

printf("n = %4i, m = %4i, time = %li min, %li max\n", n, m, min, max);

m = (n\*n+3\*n)/6;

Test3(n, m, &min, &max);

printf("n = %4i, m = %4i, time = %li min, %li max\n", n, m, min, max);

m = (n\*n)/3;

Test3(n, m, &min, &max);

printf("n = %4i, m = %4i, time = %li min, %li max\n", n, m, min, max);

m = (n\*n-n)/2;

Test3(n, m, &min, &max);

printf("n = %4i, m = %4i, time = %li min, %li max\n", n, m, min, max);

}

}