Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №9

по курсу “Логика и основа алгоритмизации в ИЗ”

на тему “Поиск расстояния в графе”



Пенза 2022

**Название**

Определение характеристик графов.

**Цель работы:** выполнить лабораторные указания 1-2 выполняя обход графа в ширину.

**Лабораторное задание:**

**Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием. При реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки С++.

**3.** Реализуйте процедуру поиска расстояний для графа, представленного списками смежности.

**Задание 2**

1. Реализуйте процедуру поиска расстояний на основе обхода в глубину.
2. Реализуйте процедуру поиска расстояний на основе обхода в глубину для графа, представленного списками смежности.
3. Оцените время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков.

**Задание 1:**

**Листинг**

#include <iostream>

#include <locale>

#include <queue>

#include <stack>

using namespace std;

int main()

{

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

int count = 2;

cout << "Введите размер матрицы: ";

cin >> count;

clock\_t start, end; // объявляем переменные для определения времени выполнения

/\*

1. Сгенерируйте (используя генератор случайных чисел)

матрицу смежности для неориентированного графа G.

Выведите матрицу на экран.

\*/

int\*\* arr1 = new int\* [count];

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

arr1[i] = new int[count];

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

arr1[i][j] = 0;

}

}

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

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

int p = rand() % 101;

if (i == j) {

arr1[i][j] = 0;

continue;

}

if (p > 70) {

int weight = rand() % 10;

arr1[i][j] = 1;

arr1[j][i] = 1;

}

}

}

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

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

cout << arr1[i][j] << ' ';

}

cout << endl;

}

/\*

2. Для сгенерированного графа осуществите процедуру

поиска расстояний, реализованную в соответствии

с приведенным выше описанием.

\*/

queue<int> Queue;

int\* visited = new int[count];

cout << endl << endl;

for (int h = 0; h != count; h++) {

start = clock();

for (int i = 0; i != count; i++) visited[i] = -1;

cout << "Вершина " << h << endl;

Queue.push(h);

visited[h] = 0;

while (!Queue.empty()) {

int node = Queue.front();

Queue.pop();

for (int i = 0; i != count; i++) {

if (arr1[node][i] == 1 && visited[i] == -1) {

Queue.push(i);

visited[i] = visited[node] + 1;

}

}

}

for (int i = 0; i != count; i++) cout << visited[i] << ' ';

cout << endl;

double time = 0.0;

end = clock();

time += (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Время работы: %lf в секундах\n\n", time);

}

/\*3.

Реализуйте процедуру поиска расстояний для

графа, представленного списками смежности.

\*/

cout << endl << endl << "Доп. Задание" << endl;

vector< vector< int> > arr2;

arr2.resize(count);

for (int i = 0; i != count; i++) {

for (int j = 0; j != count; j++) {

if (arr1[i][j] != 0) {

arr2[i].push\_back(j);

}

}

}

for (int i = 0; i < arr2.size(); i++) {

for (int j = 0; j < arr2[i].size(); j++) {

cout << arr2[i][j] << ' ';

}

cout << endl;

}

cout << endl << endl;

for (int h = 0; h < count; h++) {

for (int i = 0; i != count; i++) visited[i] = -1;

cout << "Вершина " << h << endl;

Queue.push(h);

visited[h] = 0;

while (!Queue.empty()) {

int node = Queue.front();

Queue.pop();

for (int i = 0; i != arr2[node].size(); i++) {

if (visited[arr2[node][i]] == -1) {

Queue.push(arr2[node][i]);

visited[arr2[node][i]] = visited[node] + 1;

}

}

}

for (int i = 0; i != count; i++) cout << visited[i] << ' ';

cout << endl << endl;

}

/\*

1. Реализуйте процедуру поиска

расстояний на основе обхода в глубину.

\*/

cout << endl << endl << "Обход в глубину" << endl << endl;

stack<int> Stack;

int\* top = new int[count];

for (int h = 0; h != count; h++) {

start = clock();

cout << "Вершина " << h << endl;

for (int q = 0; q != count; q++) {

if (h == q) continue;

for (int i = 0; i != count; i++) {

top[i] = 1000;

}

Stack.push(h);

top[h] = 0;

while (!Stack.empty()) {

int node = Stack.top();

Stack.pop();

if (node == q) continue;

for (int i = 0; i != count; i++) {

if (arr1[node][i] == 1 && top[i] > top[node]) {

Stack.push(i);

top[i] = top[node] + 1;

}

}

}

cout << top[q] << ' ';

}

cout << endl;

double time = 0.0;

end = clock();

time += (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Время работы: %lf в секундах\n\n", time);

}

/\*

2. Реализуйте процедуру поиска расстояний на

основе обхода в глубину для графа,

представленного списками смежности.

\*/

cout << endl << endl << endl;

for (int h = 0; h != count; h++) {

cout << "Вершина " << h << endl;

for (int q = 0; q != count; q++) {

if (h == q) continue;

for (int i = 0; i != count; i++) {

top[i] = 1000;

}

Stack.push(h);

top[h] = 0;

while (!Stack.empty()) {

int node = Stack.top();

Stack.pop();

if (node == q) continue;

for (int i = 0; i != arr2[node].size(); i++) {

if (top[arr2[node][i]] > top[node]) {

Stack.push(arr2[node][i]);

top[arr2[node][i]] = top[node] + 1;

}

}

}

cout << top[q] << ' ';

}

cout << endl;

}

return 0;

}

**Результаты работы программы:**

Результаты работы программы показан на рисунке 1.

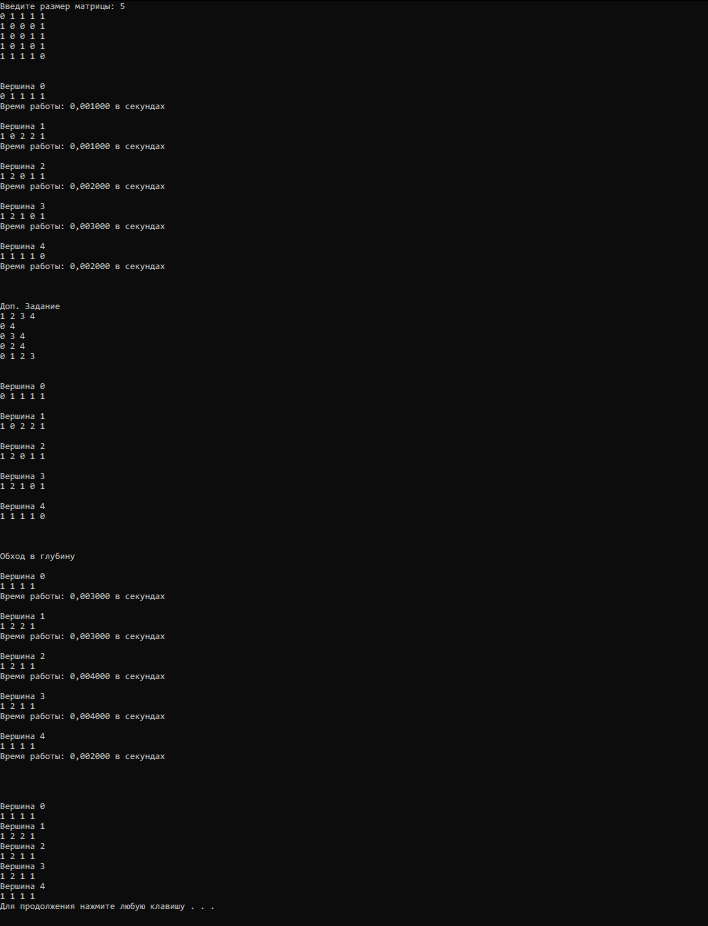
****

Рис. 1

Результат получился правильный, все действия совершены верно.

Результат работы программы, показанный на рисунке 1.

**Задание 2:**

**Листинг**

#include <iostream>

#include <locale>

#include <queue>

#include <stack>

using namespace std;

int main()

{

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

int count = 2;

cout << "Введите размер матрицы: ";

cin >> count;

clock\_t start, end; // объявляем переменные для определения времени выполнения

/\*

1. Сгенерируйте (используя генератор случайных чисел)

матрицу смежности для неориентированного графа G.

Выведите матрицу на экран.

\*/

int\*\* arr1 = new int\* [count];

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

arr1[i] = new int[count];

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

arr1[i][j] = 0;

}

}

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

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

int p = rand() % 101;

if (i == j) {

arr1[i][j] = 0;

continue;

}

if (p > 70) {

int weight = rand() % 10;

arr1[i][j] = 1;

arr1[j][i] = 1;

}

}

}

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

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

cout << arr1[i][j] << ' ';

}

cout << endl;

}

/\*

2. Для сгенерированного графа осуществите процедуру

поиска расстояний, реализованную в соответствии

с приведенным выше описанием.

\*/

queue<int> Queue;

int\* visited = new int[count];

cout << endl << endl;

for (int h = 0; h != count; h++) {

start = clock();

for (int i = 0; i != count; i++) visited[i] = -1;

cout << "Вершина " << h << endl;

Queue.push(h);

visited[h] = 0;

while (!Queue.empty()) {

int node = Queue.front();

Queue.pop();

for (int i = 0; i != count; i++) {

if (arr1[node][i] == 1 && visited[i] == -1) {

Queue.push(i);

visited[i] = visited[node] + 1;

}

}

}

for (int i = 0; i != count; i++) cout << visited[i] << ' ';

cout << endl;

double time = 0.0;

end = clock();

time += (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Время работы: %lf в секундах\n\n", time);

}

/\*3.

Реализуйте процедуру поиска расстояний для

графа, представленного списками смежности.

\*/

cout << endl << endl << "Доп. Задание" << endl;

vector< vector< int> > arr2;

arr2.resize(count);

for (int i = 0; i != count; i++) {

for (int j = 0; j != count; j++) {

if (arr1[i][j] != 0) {

arr2[i].push\_back(j);

}

}

}

for (int i = 0; i < arr2.size(); i++) {

for (int j = 0; j < arr2[i].size(); j++) {

cout << arr2[i][j] << ' ';

}

cout << endl;

}

cout << endl << endl;

for (int h = 0; h < count; h++) {

for (int i = 0; i != count; i++) visited[i] = -1;

cout << "Вершина " << h << endl;

Queue.push(h);

visited[h] = 0;

while (!Queue.empty()) {

int node = Queue.front();

Queue.pop();

for (int i = 0; i != arr2[node].size(); i++) {

if (visited[arr2[node][i]] == -1) {

Queue.push(arr2[node][i]);

visited[arr2[node][i]] = visited[node] + 1;

}

}

}

for (int i = 0; i != count; i++) cout << visited[i] << ' ';

cout << endl << endl;

}

/\*

1. Реализуйте процедуру поиска

расстояний на основе обхода в глубину.

\*/

cout << endl << endl << "Обход в глубину" << endl << endl;

stack<int> Stack;

int\* top = new int[count];

for (int h = 0; h != count; h++) {

start = clock();

cout << "Вершина " << h << endl;

for (int q = 0; q != count; q++) {

if (h == q) continue;

for (int i = 0; i != count; i++) {

top[i] = 1000;

}

Stack.push(h);

top[h] = 0;

while (!Stack.empty()) {

int node = Stack.top();

Stack.pop();

if (node == q) continue;

for (int i = 0; i != count; i++) {

if (arr1[node][i] == 1 && top[i] > top[node]) {

Stack.push(i);

top[i] = top[node] + 1;

}

}

}

cout << top[q] << ' ';

}

cout << endl;

double time = 0.0;

end = clock();

time += (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Время работы: %lf в секундах\n\n", time);

}

/\*

2. Реализуйте процедуру поиска расстояний на

основе обхода в глубину для графа,

представленного списками смежности.

\*/

cout << endl << endl << endl;

for (int h = 0; h != count; h++) {

cout << "Вершина " << h << endl;

for (int q = 0; q != count; q++) {

if (h == q) continue;

for (int i = 0; i != count; i++) {

top[i] = 1000;

}

Stack.push(h);

top[h] = 0;

while (!Stack.empty()) {

int node = Stack.top();

Stack.pop();

if (node == q) continue;

for (int i = 0; i != arr2[node].size(); i++) {

if (top[arr2[node][i]] > top[node]) {

Stack.push(arr2[node][i]);

top[arr2[node][i]] = top[node] + 1;

}

}

}

cout << top[q] << ' ';

}

cout << endl;

}

return 0;

}

**Результаты работы программы:**

Результаты работы программы показан на рисунке 1.

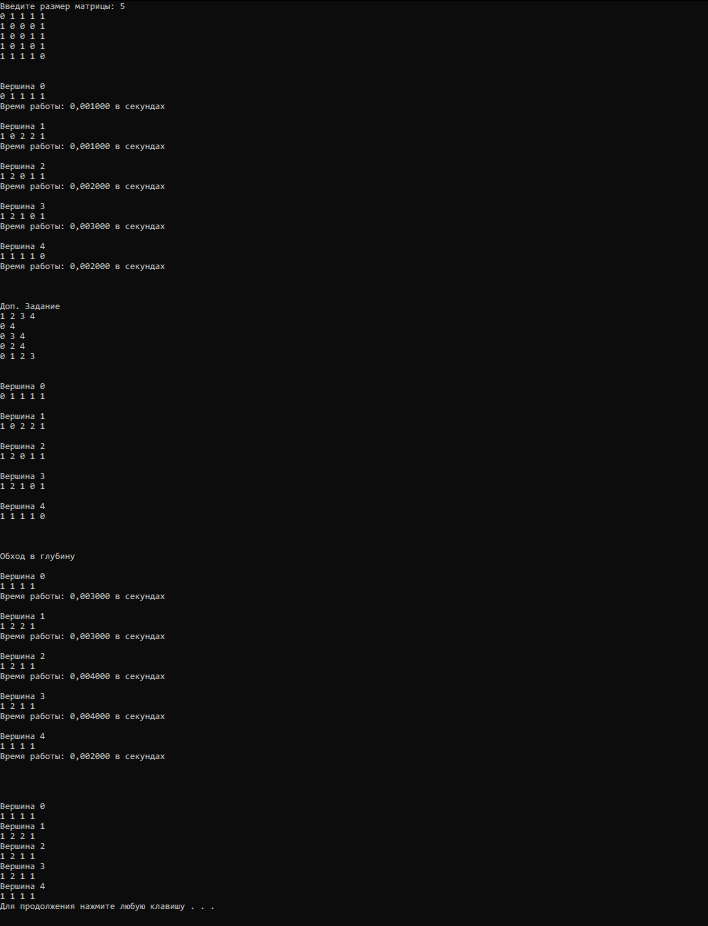
****

Рис. 1

Результат получился правильный, все действия совершены верно.

Результат работы программы, показанный на рисунке 1.