Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №9

по теме «Поиск расстояний в графе»

Выполнили:

Студенты группы 21ВВ2

Козлова К.С.

Кривенкова В.С.

Принял:

Митрохин М.А.

Юрова О.В.

Пенза 2022

**Цель работы:**

Изучить основные алгоритмы поиска расстояний в графе.

**Лабораторные работы:**

Задание 1

Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа G. Выведите матрицу на экран.

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++.

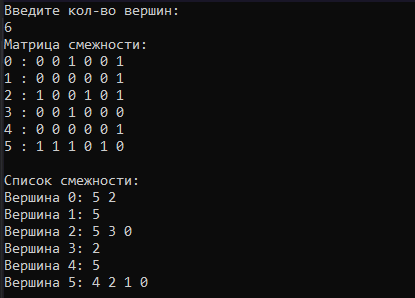
3.\* Реализуйте процедуру поиска расстояний для графа, представленного списками смежности.

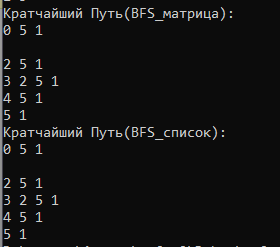
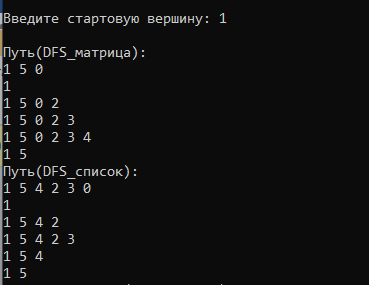
Задание 2\*

Реализуйте процедуру поиска расстояний на основе обхода в глубину.

Реализуйте процедуру поиска расстояний на основе обхода в глубину для графа, представленного списками смежности.

Оцените время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков. **Результат работы программы:**

****

****

**Листинг:**

#include <iostream>

#include <list>

#include <stack>

#include <queue>

using namespace std;

struct node

{

int vertex;

struct node\* next;

};

struct Graph

{

int numVertices;

struct node\*\* adjLists;

};

struct node\* createNode(int v)

{

struct node\* newNode = (node\*)malloc(sizeof(struct node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices)

{

struct Graph\* graph = (Graph\*)malloc(sizeof(struct Graph));

graph->numVertices = vertices;

graph->adjLists = (node\*\*)malloc(vertices \* sizeof(struct node\*));

int i;

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

graph->adjLists[i] = NULL;

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest)

{

// добавление вершины в список элемента

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

}

void printGraph(struct Graph\* graph)

{

for (int v = 0; v < graph->numVertices; v++)

{

struct node\* temp = graph->adjLists[v];

printf("\nВершина %d: ", v);

while (temp)

{

printf("%d ", temp->vertex);

temp = temp->next;

}

}

}

void toString(int\*\* Matrix, int n) {

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

cout << i << " : ";

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

cout << Matrix[i][j] << " ";

}

cout << endl;

}

}

void DFS\_M(int\*\* Matrix, bool\* visited, int n, int start, int dest)

{

visited[start] = true;

cout << start << " ";

if (start == dest)

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

visited[i] = 1;

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

if ((Matrix[start][i] != 0) && (!visited[i]))

DFS\_M(Matrix, visited, n, i, dest);

}

void DFS\_S(int\*\* Matrix, bool\* visited, int n, struct Graph\* graph, int start, int dest)

{

struct node\* spisok = graph->adjLists[start];

struct node\* temp = spisok;

visited[start] = 1;

printf("%d ", start);

if (start == dest)

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

visited[i] = 1;

while (temp != NULL) {

int connectedVertex = temp->vertex;

if (!visited[connectedVertex]) {

DFS\_S(Matrix, visited, n, graph, connectedVertex, dest);

}

temp = temp->next;

}

}

int main()

{

setlocale(0, "");

srand(time(0));

int n;

printf\_s("Введите кол-во вершин: \n");

scanf\_s("%d", &n);

struct Graph\* graph = createGraph(n);

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

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

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

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

Matrix[i][j] = 0; // создание матрицы смежности

}

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

for (int j = i + 1; j < n; j++)

if (0 < rand() % 2) {

Matrix[i][j] = Matrix[j][i] = 1;

addEdge(graph, i, j);

addEdge(graph, j, i);

}

bool\* visited = (bool\*)malloc(n \* sizeof(bool));

printf\_s("Матрица смeжности: \n");

toString(Matrix, n);

printf\_s("\nСписок смежности: ");

printGraph(graph);

int start, end;

printf\_s("\n\nВведите стартовую вершину: "); scanf\_s("%d", &start);

printf\_s("\nПуть(DFS\_матрица): ");

for (end = 0; end < n; end++) {

printf\_s("\n");

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

visited[i] = 0;

DFS\_M(Matrix, visited, n, start, end);

}

printf\_s("\nПуть(DFS\_список): ");

for (end = 0; end < n; end++) {

printf\_s("\n");

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

visited[i] = 0;

DFS\_S(Matrix, visited, n, graph, start, end);

}

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

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

queue<int> queue, q;

// BFS для матрицы смежности

printf\_s("\nКратчайший Путь(BFS\_матрица):");

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

visited[i] = 0;

for (end = 0; end < n; end++) {

int dest1 = end;

printf\_s("\n");

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

dist[i] = INT\_MAX;

pred[i] = -1;

}

visited[start] = true;

dist[start] = 0;

queue.push(start);

while (!queue.empty()) {

int t = queue.front();

queue.pop();

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

if (Matrix[t][i] == 1 && (!visited[i])) {

visited[i] = true;

queue.push(i);

dist[i] = dist[t]++;

pred[i] = t;

if (i == dest1) {

printf("%d ", dest1);

while (pred[dest1] != -1) {

printf("%d ", pred[dest1]);

dest1 = pred[dest1];

}

break;

}

}

}

}

while (queue.size() != 0)

queue.pop();

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

visited[i] = 0;

}

// для списка смежности

printf\_s("\nКратчайший Путь(BFS\_список): ");

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

visited[i] = 0;

for (end = 0; end < n; end++) {

int dest2 = end;

printf\_s("\n");

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

dist[i] = INT\_MAX;

pred[i] = -1;

}

visited[start] = 1;

dist[start] = 0;

q.push(start);

while (!q.empty()) {

int currentVertex = q.front();

q.pop();

struct node\* temp = graph->adjLists[currentVertex];

while (temp != NULL) {

int adjVertex = temp->vertex;

if (!visited[adjVertex]) {

visited[adjVertex] = 1;

q.push(adjVertex);

dist[adjVertex] = dist[currentVertex]++;

pred[adjVertex] = currentVertex;

if (adjVertex == dest2) {

printf("%d ", dest2);

while (pred[dest2] != -1) {

printf("%d ", pred[dest2]);

dest2 = pred[dest2];

}

break;

}

}

temp = temp->next;

}

}

while (queue.size() != 0)

q.pop();

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

visited[i] = 0;

}

free(dist);

free(pred);

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

free(Matrix[i]);

free(Matrix);

free(visited);

return 0;

}

**Вывод:** в ходе данной лабораторной работы мы научились реализовывать основные алгоритмы поиска расстояний в графе.