Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра “Вычислительная техника”

**ОТЧЕТ**

по лабораторной работе №9

по дисциплине «Логика и основы алгоритмизации в инженерных задачах»

# на тему: «Поиск расстояний в графе»

Выполнили студенты группы 22ВВВ3:

Куракин Н.Н.

Майоров Н.А.

Матюшин К. М.

Приняли:

Юрова О.В.

Акифьев И.В.

Пенза 2023

**Название**

Поиск расстояний.

**Цель работы**

Научиться искать расстояния в графах

**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:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <stdbool.h>

#include <locale.h>

#include <queue>

struct node {

int numNode;

struct node\* nextNode;

};

void ClearLine(struct node\* list, int vertexes) {

if (!list) return;

if (list->nextNode)

ClearLine(list->nextNode, vertexes);

free(list);

}

void ClearList(struct node\* list, int vertexes) {

if (!list || !vertexes) return;

if (vertexes > 1)

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

ClearLine((list + i)->nextNode, vertexes);

}

free(list);

}

int\* CreateGraph(int vertexes) {

int\* Mtrx = (int\*)malloc(sizeof(int) \* vertexes \* vertexes);

if (!Mtrx) return 0;

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

\*(Mtrx + i \* vertexes + i) = 0;

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

\*(Mtrx + i \* vertexes + j) = rand() % 2;

\*(Mtrx + j \* vertexes + i) = \*(Mtrx + i \* vertexes + j);

}

}

return Mtrx;

}

int ModifyGraph(struct node\* list, int\* graph, int vertexes) {

if (!list || !graph) return 0;

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

struct node\* current = list + i;

current->numNode = i + 1;

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

if (\*(graph + i \* vertexes + j)) {

current->nextNode = (struct node\*)malloc(sizeof(struct node));

current = current->nextNode;

current->numNode = j + 1;

}

}

current->nextNode = NULL;

}

return 1;

}

int breadthFirstSearchAdjacencyMatrix(int\* matrix, int startNode, int numNodes, int endNode) {

if (numNodes <= 0 || (startNode + 1) == endNode) return 0;

int\* visited = (int\*)malloc(sizeof(int) \* numNodes), count = 0;

if (!visited) return 0;

std::queue<int>queue;

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

visited[i] = 0;

}

visited[startNode] = 1;

queue.push(startNode);

while (!queue.empty()) {

int currentNode = queue.front();

queue.pop();

count++;

if (\*(matrix + currentNode \* numNodes + endNode - 1)) {

free(visited);

return count;

}

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

if (\*(matrix + currentNode \* numNodes + i) && !visited[i]) {

visited[i] = 1;

queue.push(i);

}

}

}

free(visited);

return 0;

}

int breadthFirstSearchAdjacencyList(struct node\* adjacencyList, int startNode, int numNodes, int endNode) {

if (numNodes <= 0 || (startNode + 1) == endNode) return 0;

int\* visited = (int\*)malloc(sizeof(int) \* numNodes), count = 0;

if (!visited) return 0;

std::queue<int>queue;

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

visited[i] = 0;

}

visited[startNode] = 1;

queue.push(startNode);

while (!queue.empty()) {

int currentNode = queue.front();

struct node\* current = adjacencyList + currentNode;

//printf("%d ", currentNode + 1);

queue.pop();

count++;

do {

if ((current)->numNode == endNode) {

free(visited);

return count;

}

if (!visited[(current)->numNode - 1]) {

visited[(current)->numNode - 1] = 1;

queue.push((current)->numNode - 1);

}

current = current->nextNode;

} while (current);

}

free(visited);

return 0;

}

void ShowGraph(int\* graph, int vertexes) {

if (!graph) return;

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

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

printf("%i ", \*(graph + i \* vertexes + j));

}

printf("|%i", i + 1);

printf("\n");

}

}

void ShowList(struct node\* list, int vertexes) {

if (!list) return;

printf("\n");

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

struct node\* current = list + i;

while (current) {

printf("|%i|\t->", current->numNode);

current = current->nextNode;

}

printf("\n");

}

}

int main() {

srand(time(NULL));

setlocale(LC\_ALL, "RUS");

int vertexNum = 0, vertexEnd = 0;

printf("Кол-во вершин: ");

if (!scanf("%d", &vertexNum) || vertexNum <= 0) return 1;

vertexNum = vertexNum > 99 ? 99 : vertexNum;

int\* adjacencyMatrix = CreateGraph(vertexNum);

if (!adjacencyMatrix) return 1;

struct node\* adjacencyList = (struct node\*)malloc(sizeof(struct node) \* vertexNum);

if (!ModifyGraph(adjacencyList, adjacencyMatrix, vertexNum)) return 1;

system("cls");

ShowGraph(adjacencyMatrix, vertexNum);

ShowList(adjacencyList, vertexNum);

printf("Поиск расcтояний к вершине: ");

if (!scanf("%d", &vertexEnd) || vertexEnd > vertexNum || vertexEnd <= 0) {

printf("Нет такой вершины");

return 1;

}

printf("\tМатрица смежности:\tСписок смежности\n");

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

printf("%d->%d|\t%d\t\t\t%d\n", i + 1, vertexEnd, breadthFirstSearchAdjacencyMatrix(adjacencyMatrix, i, vertexNum, vertexEnd), breadthFirstSearchAdjacencyList(adjacencyList, i, vertexNum, vertexEnd));

}

free(adjacencyMatrix);

ClearList(adjacencyList, vertexNum);

return 0;

}

Задание 2:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <stdbool.h>

#include <locale.h>

#include <queue>

struct node {

int numNode;

struct node\* nextNode;

};

int\* CreateGraph(int vertexes) {

int\* Mtrx = (int\*)malloc(sizeof(int) \* vertexes \* vertexes);

if (!Mtrx) return 0;

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

\*(Mtrx + i \* vertexes + i) = 0;

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

\*(Mtrx + i \* vertexes + j) = rand() % 2;

\*(Mtrx + j \* vertexes + i) = \*(Mtrx + i \* vertexes + j);

}

}

return Mtrx;

}

int ModifyGraph(struct node\* list, int\* graph, int vertexes) {

if (!list || !graph) return 0;

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

struct node\* current = list + i;

current->numNode = i + 1;

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

if (\*(graph + i \* vertexes + j)) {

current->nextNode = (struct node\*)malloc(sizeof(struct node));

if (!current->nextNode) return 0;

current = current->nextNode;

current->numNode = j + 1;

}

}

current->nextNode = NULL;

}

return 1;

}

void ClearLine(struct node\* list, int vertexes) {

if (!list) return;

if (list->nextNode)

ClearLine(list->nextNode, vertexes);

free(list);

}

void ClearList(struct node\* list, int vertexes) {

if (!list || !vertexes) return;

if (vertexes > 1)

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

ClearLine((list + i)->nextNode, vertexes);

}

free(list);

}

void ShowGraph(int\* graph, int vertexes) {

if (!graph) return;

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

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

printf("%i ", \*(graph + i \* vertexes + j));

}

printf("|%i", i + 1);

printf("\n");

}

}

void ShowList(struct node\* list, int vertexes) {

if (!list) return;

printf("\n");

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

struct node\* current = list + i;

while (current) {

printf("|%i|\t->", current->numNode);

current = current->nextNode;

}

printf("\n");

}

}

int DepthFirstSearch1(int\* graph, int vertexes, int startNode, int endNode) {

if (!graph || startNode == (endNode - 1) || !vertexes) return 0;

int\* visited = (int\*)malloc(sizeof(int) \* vertexes), \* queue = (int\*)malloc(sizeof(int) \* vertexes \* 2), temp = 0, result = INT\_MAX, iQueue = -1, currentNode = startNode, cond = 0;

if (!visited || !queue) return 0;

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

visited[i] = 0;

}

do {

visited[currentNode] = 1;

temp++;

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

if (\*(graph + vertexes \* currentNode + i) && !visited[i]) {

cond = 1;

visited[i] = 1;

if (i == (endNode - 1)) {

result = temp < result ? temp : result;

temp--;

break;

}

queue[++iQueue] = i;

}

}

if (!cond && currentNode == startNode)

return 0;

currentNode = queue[iQueue];

queue[iQueue--] = 0;

} while (iQueue >= 0);

free(visited);

return result;

}

int DepthFirstSearch2(struct node\* graph, int vertexes, int startNode, int endNode) {

if (!graph || startNode == (endNode - 1) || !vertexes) return 0;

int\* visited = (int\*)malloc(sizeof(int) \* vertexes), \* queue = (int\*)malloc(sizeof(int) \* vertexes \* 2), temp = 0, result = INT\_MAX, iQueue = -1, iList = -1, currentNode = startNode, cond = 0;

struct node\* node = graph;

if (!visited || !queue) return 0;

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

visited[i] = 0;

}

do {

visited[currentNode] = 1;

node = &graph[currentNode];

temp++;

while (node->nextNode) {

cond = 1;

node = node->nextNode;

if (!visited[node->numNode - 1]) {

visited[node->numNode - 1] = 1;

if (node->numNode == endNode) {

result = temp < result ? temp : result;

temp--;

break;

}

queue[++iQueue] = node->numNode - 1;

}

}

if (!cond && currentNode == startNode)

return 0;

currentNode = queue[iQueue];

queue[iQueue--] = 0;

} while (iQueue >= 0);

free(visited);

return result;

}

int main() {

srand(time(NULL));

setlocale(LC\_ALL, "RUS");

int vertexNum = 0, vertexEnd = 0;

printf("Кол-во вершин: ");

if (!scanf("%d", &vertexNum) || vertexNum <= 0) return 1;

vertexNum = vertexNum > 99 ? 99 : vertexNum;

int\* adjacencyMatrix = CreateGraph(vertexNum);

if (!adjacencyMatrix) return 1;

struct node\* adjacencyList = (struct node\*)malloc(sizeof(struct node) \* vertexNum);

if (!ModifyGraph(adjacencyList, adjacencyMatrix, vertexNum)) return 1;

system("cls");

ShowGraph(adjacencyMatrix, vertexNum);

ShowList(adjacencyList, vertexNum);

printf("Поиск расcтояний к вершине: ");

if (!scanf("%d", &vertexEnd) || vertexEnd > vertexNum || vertexEnd <= 0) {

printf("Нет такой вершины");

return 1;

}

printf("\tМатрица смежности:\tСписок смежности\n");

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

printf("%d->%d|\t%d\t\t\t%d\n", i + 1, vertexEnd, DepthFirstSearch1(adjacencyMatrix, vertexNum, i, vertexEnd), DepthFirstSearch2(adjacencyList, vertexNum, i, vertexEnd));

}

free(adjacencyMatrix);

ClearList(adjacencyList, vertexNum);

return 0;

}

Задание 3

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <stdbool.h>

#include <locale.h>

#include <queue>

struct node {

int numNode;

struct node\* nextNode;

};

void ClearLine(struct node\* list, int vertexes) {

if (!list) return;

if (list->nextNode)

ClearLine(list->nextNode, vertexes);

free(list);

}

void ClearList(struct node\* list, int vertexes) {

if (!list || !vertexes) return;

if (vertexes > 1)

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

ClearLine((list + i)->nextNode, vertexes);

}

free(list);

}

int\* CreateGraph(int vertexes) {

int\* Mtrx = (int\*)malloc(sizeof(int) \* vertexes \* vertexes);

if (!Mtrx) return 0;

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

\*(Mtrx + i \* vertexes + i) = 0;

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

\*(Mtrx + i \* vertexes + j) = rand() % 2;

\*(Mtrx + j \* vertexes + i) = \*(Mtrx + i \* vertexes + j);

}

}

return Mtrx;

}

int ModifyGraph(struct node\* list, int\* graph, int vertexes) {

if (!list || !graph) return 0;

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

struct node\* current = list + i;

current->numNode = i + 1;

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

if (\*(graph + i \* vertexes + j)) {

current->nextNode = (struct node\*)malloc(sizeof(struct node));

current = current->nextNode;

current->numNode = j + 1;

}

}

current->nextNode = NULL;

}

return 1;

}

int breadthFirstSearchAdjacencyMatrix(int\* matrix, int startNode, int numNodes, int endNode) {

if (numNodes <= 0 || (startNode + 1) == endNode) return 0;

int\* visited = (int\*)malloc(sizeof(int) \* numNodes), count = 0;

if (!visited) return 0;

std::queue<int>queue;

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

visited[i] = 0;

}

visited[startNode] = 1;

queue.push(startNode);

while (!queue.empty()) {

int currentNode = queue.front();

queue.pop();

count++;

if (\*(matrix + currentNode \* numNodes + endNode - 1)) {

free(visited);

return count;

}

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

if (\*(matrix + currentNode \* numNodes + i) && !visited[i]) {

visited[i] = 1;

queue.push(i);

}

}

}

free(visited);

return 0;

}

int breadthFirstSearchAdjacencyList(struct node\* adjacencyList, int startNode, int numNodes, int endNode) {

if (numNodes <= 0 || (startNode + 1) == endNode) return 0;

int\* visited = (int\*)malloc(sizeof(int) \* numNodes), count = 0;

if (!visited) return 0;

std::queue<int>queue;

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

visited[i] = 0;

}

visited[startNode] = 1;

queue.push(startNode);

while (!queue.empty()) {

int currentNode = queue.front();

struct node\* current = adjacencyList + currentNode;

//printf("%d ", currentNode + 1);

queue.pop();

count++;

do {

if ((current)->numNode == endNode) {

free(visited);

return count;

}

if (!visited[(current)->numNode - 1]) {

visited[(current)->numNode - 1] = 1;

queue.push((current)->numNode - 1);

}

current = current->nextNode;

} while (current);

}

free(visited);

return 0;

}

int DepthFirstSearch1(int\* graph, int vertexes, int startNode, int endNode) {

if (!graph || startNode == (endNode - 1) || !vertexes) return 0;

int\* visited = (int\*)malloc(sizeof(int) \* vertexes), \* queue = (int\*)malloc(sizeof(int) \* vertexes \* 2), temp = 0, result = INT\_MAX, iQueue = -1, currentNode = startNode;

if (!visited || !queue) return 0;

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

visited[i] = 0;

}

do {

visited[currentNode] = 1;

temp++;

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

if (\*(graph + vertexes \* currentNode + i) && !visited[i]) {

visited[i] = 1;

if (i == (endNode - 1)) {

result = temp < result ? temp : result;

temp--;

break;

}

queue[++iQueue] = i;

}

}

currentNode = queue[iQueue];

queue[iQueue--] = 0;

} while (iQueue >= 0);

//free(queue);

free(visited);

return result;

}

int DepthFirstSearch2(struct node\* graph, int vertexes, int startNode, int endNode) {

if (!graph || startNode == (endNode - 1) || !vertexes) return 0;

int\* visited = (int\*)malloc(sizeof(int) \* vertexes), \* queue = (int\*)malloc(sizeof(int) \* vertexes \* 2), temp = 0, result = INT\_MAX, iQueue = -1, iList = -1, currentNode = startNode;

struct node\* node = graph;

if (!visited || !queue) return 0;

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

visited[i] = 0;

}

do {

visited[currentNode] = 1;

node = &graph[currentNode];

temp++;

while (node->nextNode) {

node = node->nextNode;

if (!visited[node->numNode - 1]) {

visited[node->numNode - 1] = 1;

if (node->numNode == endNode) {

result = temp < result ? temp : result;

temp--;

break;

}

queue[++iQueue] = node->numNode - 1;

}

}

currentNode = queue[iQueue];

queue[iQueue--] = 0;

} while (iQueue >= 0);

//free(queue);

free(visited);

return result;

}

void ShowGraph(int\* graph, int vertexes) {

if (!graph) return;

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

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

printf("%i ", \*(graph + i \* vertexes + j));

}

printf("|%i", i + 1);

printf("\n");

}

}

void ShowList(struct node\* list, int vertexes) {

if (!list) return;

printf("\n");

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

struct node\* current = list + i;

while (current) {

printf("|%i|\t->", current->numNode);

current = current->nextNode;

}

printf("\n");

}

}

int main() {

srand(time(NULL));

setlocale(LC\_ALL, "RUS");

char tablest[] = "+-------+---------------+---------------+\n"

"|n\t|Матрица\t|Списки\t\t|\n"

"|\t+-------+-------+-------+-------+\n"

"|\t|Глубин\t|Ширина\t|Глубин\t|Ширина\t|\n"

"+-------+-------+-------+-------+-------+\n",

tableen[] = "|%d\t|%0.4f\t|%0.4f\t|%0.4f\t|%0.4f\t|\n"

"+-------+-------+-------+-------+-------+\n";

int vertexNum = 0, startNode = 0, count = 1, temp = 0;

printf(tablest);

for (vertexNum = 500; vertexNum != 5500; vertexNum += 500) {

int\* adjacencyMatrix = CreateGraph(vertexNum), temp = 0;

double time[4] = { 0 };

clock\_t start, end;

if (!adjacencyMatrix) return 1;

struct node\* adjacencyList = (struct node\*)malloc(sizeof(struct node) \* vertexNum);

if (!ModifyGraph(adjacencyList, adjacencyMatrix, vertexNum)) return 1;

start = clock();

DepthFirstSearch1(adjacencyMatrix, vertexNum, 0, vertexNum); //матрица - глубина

end = clock();

time[0] = (end - start) / 1000.0;

start = clock();

DepthFirstSearch2(adjacencyList, vertexNum, 0, vertexNum); //матрица - глубина

end = clock();

time[2] = (end - start) / 1000.0;

start = clock();

breadthFirstSearchAdjacencyMatrix(adjacencyMatrix, 0, vertexNum, vertexNum); //матрица - ширина

end = clock();

time[1] = (end - start) / 1000.0;

start = clock();

breadthFirstSearchAdjacencyList(adjacencyList, 0, vertexNum, vertexNum); //матрица - ширина

end = clock();

time[3] = (end - start) / 1000.0;

printf(tableen, vertexNum, time[0], time[1], time[2], time[3]);

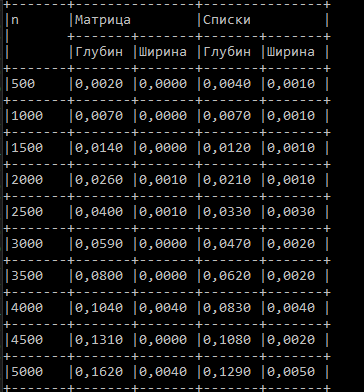
free(adjacencyMatrix);

ClearList(adjacencyList, vertexNum);

}

return 0;

}

  
По результатам тестированния программы мы можем сделать вывод о том, что поиск элемента на основе алгоритма обхода графа в ширину быстрее, чем на основе алгоритма обхода графа в глубину.

Вывод: Наша бригада освоила алгоритм поиска расстояний в графах. Мы реализовали процедуру поиска расстояний на основе обхода в глубину, а также реализовали процедуру поиска расстояний для графа, представленного списками смежности.