Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧËТ**  
по лабораторной работе №9  
по дисциплине: «Поиск расстояний в графе»

Выполнили студенты группы 22ВВВ2:  
Демин М. С.

Захаров А. С.

Амиров И. Р.

Приняли:  
Акифьев И. В.

Митрохин М. А.

Пенза 2023

**Название**

Поиск расстояний в графе

**Цель работы**

Разобраться в графах и матрицах. Научиться осуществлять поиск расстояний в графе.

**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.

**Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

#include <windows.h>

#include <queue>

#define MAX\_SIZE 1000

//#define NO\_PRINT

struct node {

int vertex;

struct node \*next;

};

struct node\* createNode(int v) {

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

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph {

int numVertices;

struct node \*\*adjLists;

};

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) {

struct node \*newNode = createNode(dest);

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

graph->adjLists[src] = newNode;

newNode = createNode(src);

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

graph->adjLists[dest] = newNode;

}

void printGraph(struct Graph \*graph) {

#ifdef NO\_PRINT

return;

#endif

int v;

printf("\n");

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

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

printf("\nvertex: %d\n ", v + 1);

while (temp) {

printf("%d ", temp->vertex + 1);

temp = temp->next;

}

}

}

void BFSD(int \*\*G, int size\_G, int v, int DIST[MAX\_SIZE]) {

bool visited[MAX\_SIZE] = {false};

visited[v] = true;

DIST[v] = 0;

int queue[MAX\_SIZE] = {0};

int front = 0, rear = 0;

queue[rear++] = v;

while (front != rear) {

int current\_vertex = queue[front++];

#ifndef NO\_PRINT

printf("%d ", current\_vertex);

#endif

for (int i = 0; i < size\_G; i++) {

if (G[current\_vertex][i] == 1 && !visited[i]) {

queue[rear++] = i;

visited[i] = true;

DIST[i] = DIST[current\_vertex] + 1;

}

}

}

}

int DFS(int \*\*vertex, int first, int \*\*matrix, int count, int clearVertex) {

vertex[1][first] = false;

#ifndef NO\_PRINT

printf("vertex: %d\n", vertex[0][first]);

#endif

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

if (matrix[first][i] == 1 and vertex[1][i] == true)

DFS(vertex, i, matrix, count, clearVertex);

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

if (vertex[1][i] == 0)

clearVertex--;

return clearVertex;

}

void POG(int \*\*matrix, int count, int i) {

int \*\*NUM = (int\*\*) malloc(2 \* sizeof(int\*));

for (int i = 0; i < 2; i++)

NUM[i] = (int\*) malloc(count \* sizeof(int));

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

NUM[0][i] = i + 1;

NUM[1][i] = true;

}

int clearVertex = count;

while (clearVertex > 0) {

while (NUM[1][i] == false) {

i++;

}

clearVertex = DFS(NUM, i, matrix, count, clearVertex);

}

}

int main() {

setlocale(LC\_ALL, "Russian");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setbuf(stdout, NULL);

int size, v = 0;

printf("Enter a count of headers: ");

scanf(" %d", &size);

srand(time(NULL)); //

int \*\*matrix = (int\*\*) calloc(size, sizeof(int\*));

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

matrix[i] = (int\*) calloc(size, sizeof(int));

}

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

for (int j = i + 1; j < size; j++) {

int random = rand() % 2; // 0 1

matrix[i][j] = random;

matrix[j][i] = random; //

}

}

#ifndef NO\_PRINT

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

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

printf("%d ", matrix[i][j]);

}

printf("\n");

}

#endif

printf("Enter the source vertex: ");

scanf("%d", &v);

int DIST[MAX\_SIZE];

for (size\_t i = 0; i < MAX\_SIZE; i++) {

DIST[i] = -1;

}

clock\_t time\_start = clock();

BFSD(matrix, size, v, DIST);

clock\_t time\_end = clock();

double diff = time\_end - time\_start;

printf("\nDIST vector:\n");

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

printf("%d ", DIST[i]);

}

// 1.3

struct Graph \*graph = createGraph(size);

srand(time(NULL));

int \*vertex = (int\*) calloc(size, sizeof(int));

bool edge;

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

vertex[i] = i;

}

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

for (int j = i + 1; j < size; j++) {

edge = rand() % 2;

if (vertex[j] != i and edge == true) {

addEdge(graph, i, j);

}

}

}

printGraph(graph);

// 2

printf("\n\n--TASK 2--\n\n");

clock\_t time\_start2 = clock();

POG(matrix, size, v);

clock\_t time\_end2 = clock();

double diff2 = time\_end2 - time\_start2;

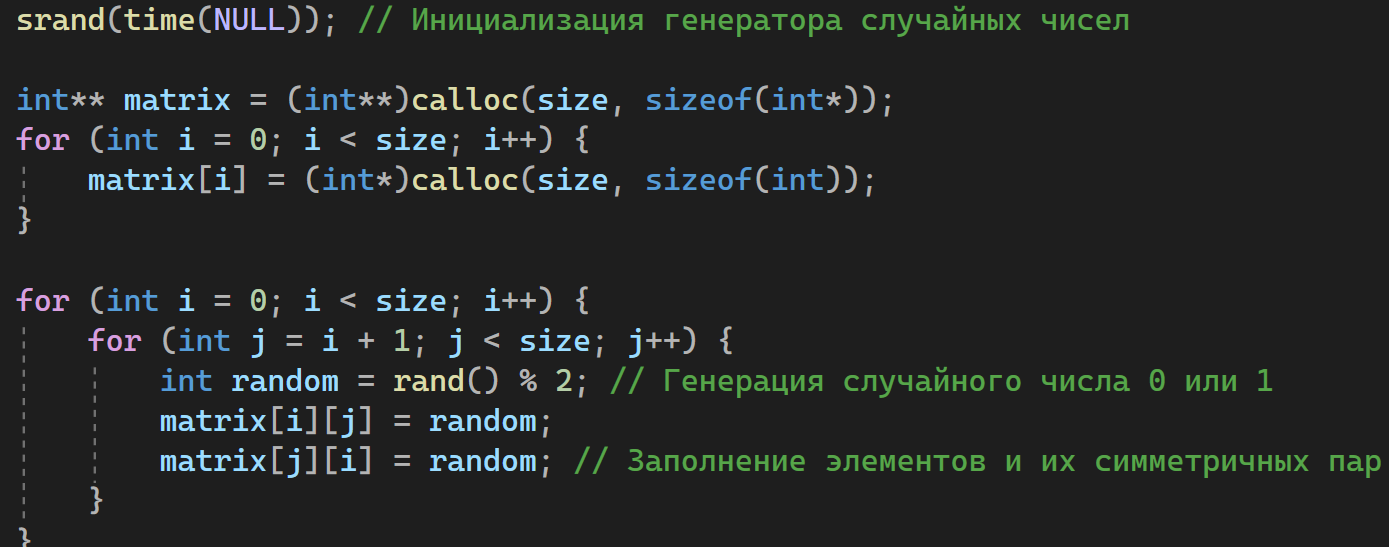
printf("\ntime elapsed: %fms\n", diff);

printf("\ntime elapsed2 : %fms\n", diff2);

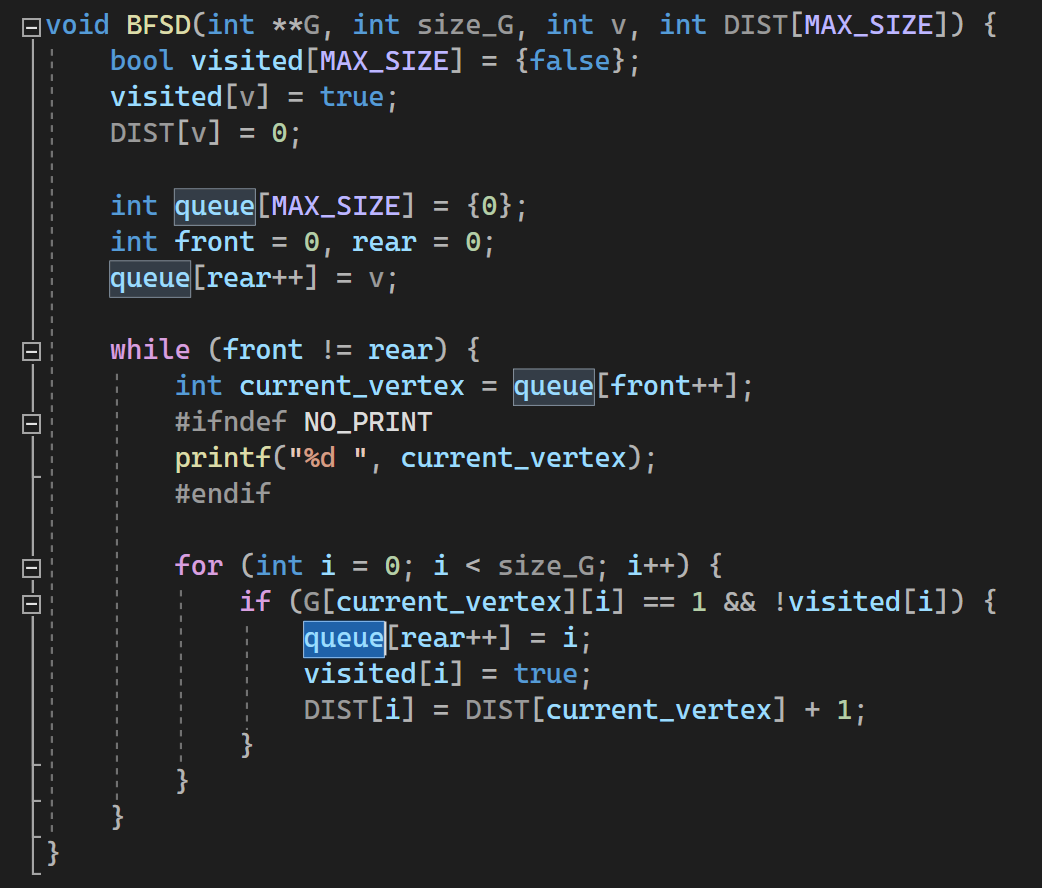
}

**Задания**

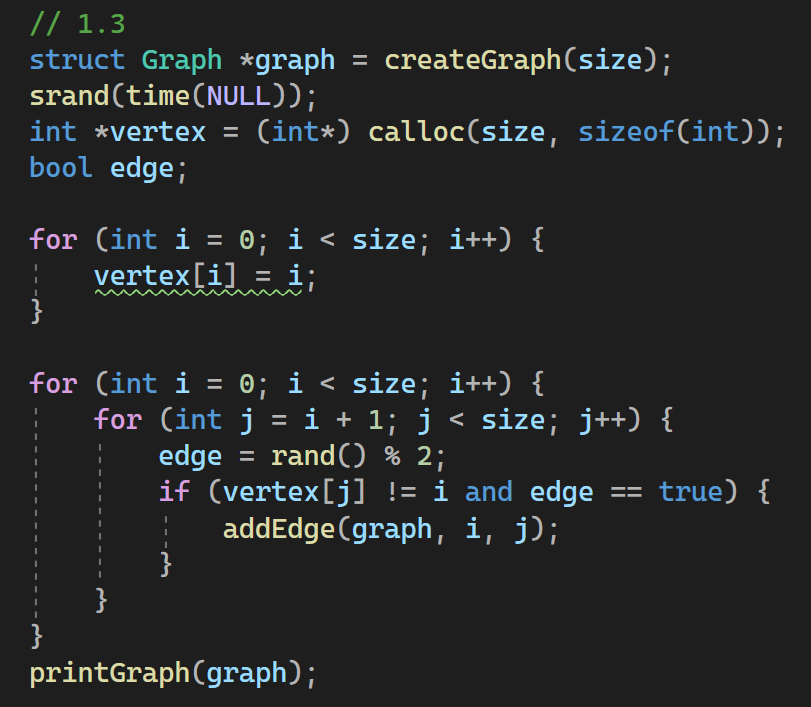
* 1. Матрица смежности



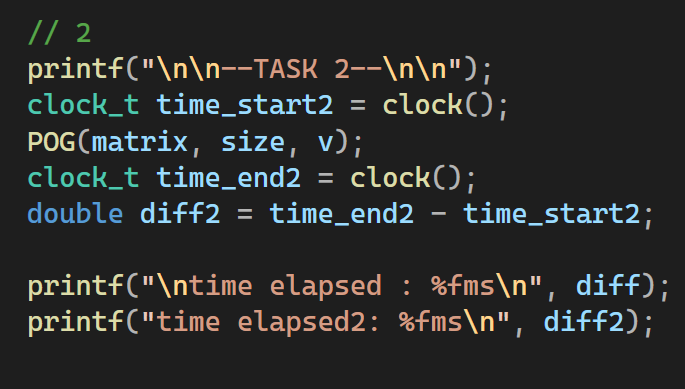
* 1. Процедура поиска расстояний(queue(cpp))



* 1. Процедура поиска расстояний для графа, представленного списками смежности



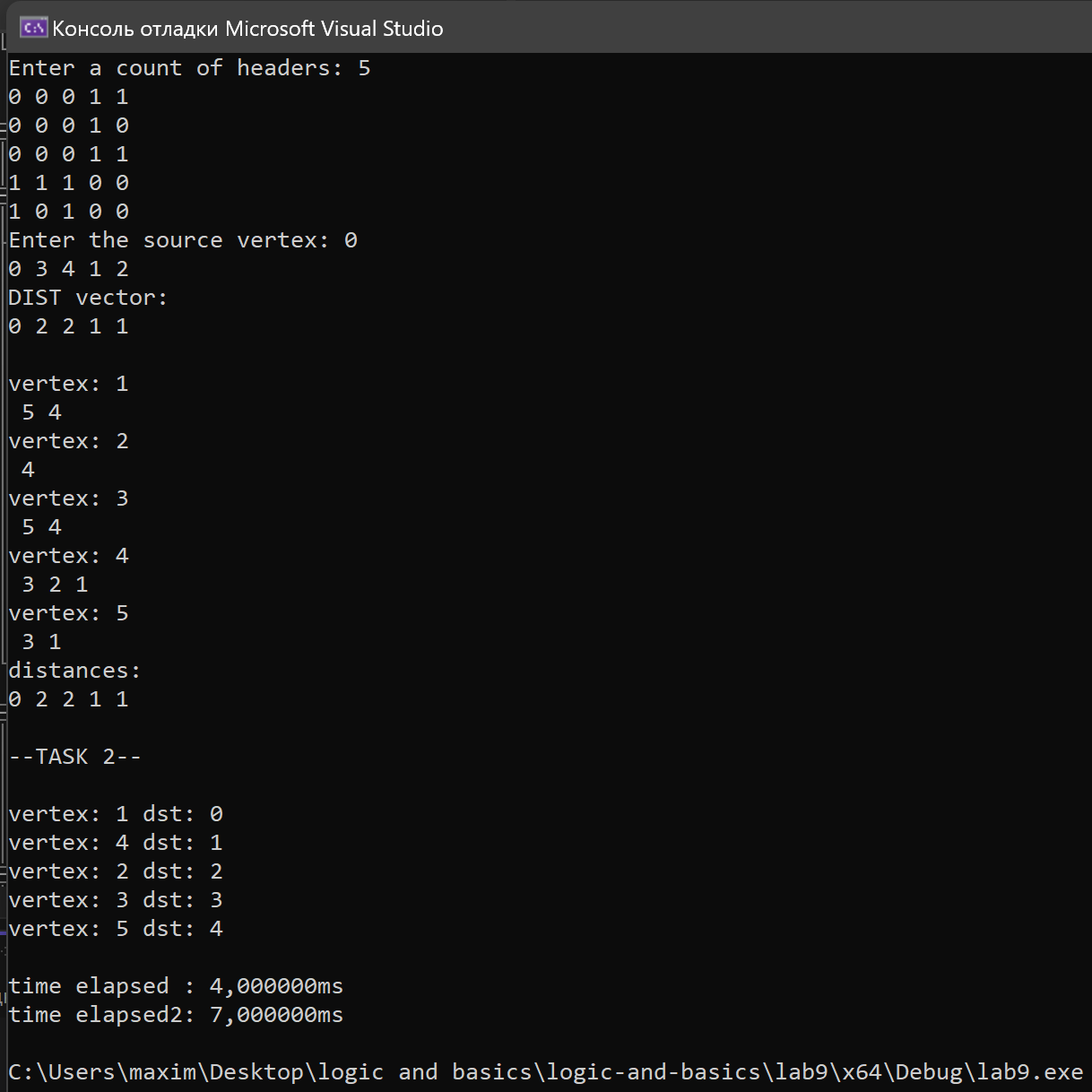
2.2.1 Процедура поиска расстояний на основе обхода в глубину



2.3 Время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков

Проведя тесты, пришли к выводу, что скорость выполнения поиска пути через обход в ширину более медленная, но алгоритм дает кратчайший путь, а поиск через обход в глубину напротив более быстрый, но выдает не всегда оптимальный маршрут.

**Результат работы программы**

****

**Вывод**

Разобрались с графами и матрицами, разработали и реализовали поиск расстояний в графе.