Министерство науки и высшего образования Российской Федерации

Пензенской государственный университет

Кафедра "Вычислительная техника"

**Отчёт**

по лабораторной работе №9

по курсу «Логика и Основы Алгоритмизации в Инженерных Задачах»

на тему «Поиск расстояний в графе»

Выполнил:

Студент группы 23ВВ1

Юров Д.М.

Принял:

Митрохин М.А.

Юрова О.В.

Пенза 2024

**Цель работы:**

1. Изучить и освоить процедуры поиска расстояний в графе на основе алгоритма поиска в ширину.
2. Самостоятельно реализовать процедуры поиска расстояний на основе обхода в глубину для графов, представленных матрицами и списками смежности.
3. Оценить эффективность реализованных процедур.

**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.

**Практическая часть**

**Листинг 1.1:**

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**#include <iostream>**

**using namespace std;**

**int\*\* create\_adjacency\_matrix(int v);**

**int cout\_matrix(int\*\* G, int v);**

**void main()**

**{**

**int vertexes = 0;**

**cout << "input graph size: ";**

**cin >> vertexes;**

**int\*\* M = create\_adjacency\_matrix(vertexes);**

**cout\_matrix(M, vertexes);**

**return;**

**}**

**int\*\* create\_adjacency\_matrix(int v)**

**{**

**srand(time(NULL));**

**int\*\* G = new int\* [v];**

**for (int i = 0; i < v; i++)**

**{**

**G[i] = new int[v];**

**}**

**for (int i = 0; i < v; i++)**

**{**

**for (int j = 0; j < v; j++)**

**{**

**if (j < i)**

**{**

**G[i][j] = G[j][i];**

**}**

**else**

**{**

**G[i][j] = rand() % 2;**

**}**

**}**

**}**

**return G;**

**}**

**int cout\_matrix(int\*\* G, int v)**

**{**

**cout << " ";**

**for (int i = 0; i < v; i++)**

**{**

**cout << i << " ";**

**}**

**cout << endl << endl;**

**for (int i = 0; i < v; i++)**

**{**

**cout << i << " ";**

**for (int j = 0; j < v; j++)**

**{**

**cout << G[i][j] << " ";**

**}**

**cout << endl;**

**}**

**return 1;**

**}**

**Листинг 1.2:**

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**#include <iostream>**

**#include <queue>**

**using namespace std;**

**int\*\* create\_adjacency\_matrix(int v);**

**int cout\_matrix(int\*\* G, int v);**

**void BFS\_matrix(int\*\* g, int v, int size, bool\* vis, int\* d);**

**void main()**

**{**

**int size = 0, to\_start\_with;**

**cout << "input graph size: ";**

**cin >> size;**

**int\*\* M = create\_adjacency\_matrix(size);**

**cout\_matrix(M, size);**

**bool \*visited = new bool[size];**

**int \*depth = new int[size];**

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

**{**

**visited[i] = 0;**

**depth[i] = 0;**

**}**

**cout << endl << "input input number of vertex to star with: ";**

**cin >> to\_start\_with;**

**cout << endl;**

**cout << "Breadth-first matrix search: " << endl;**

**while (!visited[to\_start\_with])**

**{**

**BFS\_matrix(M, to\_start\_with, size, visited, depth);**

**}**

**cout << endl << endl << " Depth of vertexes: " << endl;**

**cout << "vertex depth" << endl;**

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

**{**

**cout << " " << i << " " << depth[i] << endl;**

**}**

**cout << endl << "--------------------------------------------" << endl;**

**return;**

**}**

**int\*\* create\_adjacency\_matrix(int v)**

**{**

**srand(time(NULL));**

**int\*\* G = new int\* [v];**

**for (int i = 0; i < v; i++)**

**{**

**G[i] = new int[v];**

**}**

**for (int i = 0; i < v; i++)**

**{**

**for (int j = 0; j < v; j++)**

**{**

**if (j < i)**

**{**

**G[i][j] = G[j][i];**

**}**

**else**

**{**

**G[i][j] = rand() % 2;**

**}**

**}**

**}**

**return G;**

**}**

**int cout\_matrix(int\*\* g, int v)**

**{**

**cout << " ";**

**for (int i = 0; i < v; i++)**

**{**

**cout << i << " ";**

**}**

**cout << endl << endl;**

**for (int i = 0; i < v; i++)**

**{**

**cout << i << " ";**

**for (int j = 0; j < v; j++)**

**{**

**cout << g[i][j] << " ";**

**}**

**cout << endl;**

**}**

**return 1;**

**}**

**void BFS\_matrix(int\*\* g, int v, int size, bool\* vis, int\* d)**

**{**

**queue<int> q;**

**q.push(v);**

**vis[v] = 1;**

**d[v] = 0;**

**while (!q.empty())**

**{**

**v = q.front();**

**q.pop();**

**cout << v << " -> ";**

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

**{**

**if (g[v][i] == 1 && !vis[i])**

**{**

**d[i] = d[v] + 1;**

**vis[i] = 1;**

**q.push(i);**

**}**

**}**

**}**

**}**

**Листинг 1.3:**

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**#include <iostream>**

**#include <queue>**

**using namespace std;**

**struct Node**

**{**

**int vertex;**

**struct Node\* next;**

**};**

**struct Graph**

**{**

**int vertexes\_amount;**

**struct Node\*\* list;**

**};**

**int\*\* create\_adjacency\_matrix(int v);**

**int cout\_matrix(int\*\* G, int v);**

**void BFSD\_matrix(int\*\* g, int v, int size, bool\* vis, int\* d);**

**struct Graph\* create\_adjacency\_list(int vertexes);**

**struct Node\* create\_vertex(int vertex);**

**void connect\_vertexes(struct Graph\* graph, int coll, int dest);**

**void cout\_adjacency\_list(struct Graph\* graph);**

**void BFSD\_list(struct Graph\* G, int start, int size, bool\* vis, int\* d);**

**void main()**

**{**

**int size = 0, to\_start\_with;**

**cout << "input graph size: ";**

**cin >> size;**

**int\*\* M = create\_adjacency\_matrix(size);**

**cout\_matrix(M, size);**

**bool \*visited = new bool[size];**

**int \*depth = new int[size];**

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

**{**

**visited[i] = 0;**

**depth[i] = 0;**

**}**

**cout << endl << "input input number of vertex to star with: ";**

**cin >> to\_start\_with;**

**cout << endl;**

**cout << "Breadth-first list search: " << endl;**

**while (!visited[to\_start\_with])**

**{**

**BFSD\_matrix(M, to\_start\_with, size, visited, depth);**

**}**

**cout << endl << endl << " Depth of vertexes: " << endl;**

**cout << "vertex depth" << endl;**

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

**{**

**cout << " " << i << " " << depth[i] << endl;**

**}**

**cout << endl << "--------------------------------------------" << endl;**

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

**{**

**visited[i] = 0;**

**depth[i] = 0;**

**}**

**struct Graph\* M1 = create\_adjacency\_list(size);**

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

**{**

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

**{**

**if (M[i][j] == 1)**

**{**

**connect\_vertexes(M1, i, j);**

**}**

**}**

**}**

**cout\_adjacency\_list(M1);**

**cout << endl << "input input number of vertex to star with: ";**

**cin >> to\_start\_with;**

**cout << endl;**

**cout << "Breadth-first list search: " << endl;**

**while (!visited[to\_start\_with])**

**{**

**//BFSD\_matrix(M, to\_start\_with, size, visited, depth);**

**BFSD\_list(M1, to\_start\_with, size, visited, depth);**

**}**

**cout << endl << endl << " Depth of vertexes: " << endl;**

**cout << "vertex depth" << endl;**

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

**{**

**cout << " " << i << " " << depth[i] << endl;**

**}**

**cout << endl << "--------------------------------------------" << endl;**

**return;**

**}**

**int\*\* create\_adjacency\_matrix(int v)**

**{**

**srand(time(NULL));**

**int\*\* G = new int\* [v];**

**for (int i = 0; i < v; i++)**

**{**

**G[i] = new int[v];**

**}**

**for (int i = 0; i < v; i++)**

**{**

**for (int j = 0; j < v; j++)**

**{**

**if (j < i)**

**{**

**G[i][j] = G[j][i];**

**}**

**else**

**{**

**G[i][j] = rand() % 2;**

**}**

**}**

**}**

**return G;**

**}**

**int cout\_matrix(int\*\* g, int v)**

**{**

**cout << " ";**

**for (int i = 0; i < v; i++)**

**{**

**cout << i << " ";**

**}**

**cout << endl << endl;**

**for (int i = 0; i < v; i++)**

**{**

**cout << i << " ";**

**for (int j = 0; j < v; j++)**

**{**

**cout << g[i][j] << " ";**

**}**

**cout << endl;**

**}**

**return 1;**

**}**

**void BFSD\_matrix(int\*\* g, int v, int size, bool\* vis, int\* d)**

**{**

**queue<int> q;**

**q.push(v);**

**vis[v] = 1;**

**d[v] = 0;**

**while (!q.empty())**

**{**

**v = q.front();**

**q.pop();**

**cout << v << " -> ";**

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

**{**

**if (g[v][i] == 1 && !vis[i])**

**{**

**d[i] = d[v] + 1;**

**vis[i] = 1;**

**q.push(i);**

**}**

**}**

**}**

**}**

**struct Graph\* create\_adjacency\_list(int vertexes)**

**{**

**struct Graph\* graph = new struct Graph;**

**graph->vertexes\_amount = vertexes;**

**graph->list = new struct Node\* [vertexes];**

**for (int i = 0; i < vertexes; i++)**

**{**

**graph->list[i] = new struct Node[vertexes];**

**}**

**for (int i = 0; i < vertexes; i++)**

**{**

**graph->list[i] = create\_vertex(i);**

**}**

**return graph;**

**}**

**struct Node\* create\_vertex(int vertex)**

**{**

**struct Node\* new\_node = new struct Node;**

**new\_node->vertex = vertex;**

**new\_node->next = NULL;**

**return new\_node;**

**}**

**void connect\_vertexes(struct Graph\* graph, int coll, int dest)**

**{**

**struct Node\* new\_node = create\_vertex(dest);**

**int i = 0;**

**while (graph->list[i]->vertex != coll)**

**{**

**i++;**

**}**

**struct Node\* tmp = graph->list[i];**

**while (tmp->next != NULL)**

**{**

**tmp = tmp->next;**

**}**

**tmp->next = new\_node;**

**}**

**void cout\_adjacency\_list(struct Graph\* graph)**

**{**

**cout << endl << "adjacency list:" << endl;**

**struct Node\* tmp;**

**for (int i = 0; i < graph->vertexes\_amount; i++)**

**{**

**tmp = graph->list[i];**

**while (tmp)**

**{**

**cout << tmp->vertex;**

**tmp = tmp->next;**

**if (tmp != NULL)**

**{**

**cout << " -> ";**

**}**

**}**

**cout << endl;**

**}**

**}**

**void BFSD\_list(struct Graph\* G, int v, int size, bool\* vis, int\* d)**

**{**

**queue<int> q;**

**q.push(v);**

**vis[v] = 1;**

**d[v] = 0;**

**while (!q.empty())**

**{**

**v = q.front();**

**q.pop();**

**cout << v << " -> ";**

**struct Node\* l = G->list[v];**

**while (l != NULL)**

**{**

**int vert = l->vertex;**

**if (!vis[vert])**

**{**

**q.push(vert);**

**vis[vert] = 1;**

**d[vert] = d[v] + 1;**

**}**

**l = l->next;**

**}**

**}**

**}**

**Листинг 2.1:**

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**#include <iostream>**

**#include <stack>**

**using namespace std;**

**int\*\* create\_adjacency\_matrix(int v);**

**int cout\_matrix(int\*\* g, int v);**

**void DFSD\_matrix(int\*\* g, int s, int size, int\* vis, int\* depth);**

**void main()**

**{**

**int\*\* M;**

**int vertexes = 0;**

**int to\_start\_with = 0;**

**cout << "input graph size: ";**

**cin >> vertexes;**

**M = create\_adjacency\_matrix(vertexes);**

**cout\_matrix(M, vertexes);**

**int\* visited = new int[vertexes];**

**int\* depth = new int[vertexes];**

**for (int i = 0; i < vertexes; i++)**

**{**

**visited[i] = 0;**

**depth[i] = -1;**

**}**

**cout << endl << "input input number of vertex to star with: ";**

**cin >> to\_start\_with;**

**cout << endl << "First-deep search: " << endl;**

**DFSD\_matrix(M, to\_start\_with, vertexes, visited, depth);**

**cout << endl << endl << " Depth of vertexes: " << endl;**

**cout << "vertex depth" << endl;**

**for (int i = 0; i < vertexes; i++)**

**{**

**cout << " " << i << " " << depth[i] << endl;**

**}**

**cout << endl << "--------------------------------------------" << endl;**

**return;**

**}**

**int\*\* create\_adjacency\_matrix(int v)**

**{**

**srand(time(NULL));**

**int\*\* G = new int\* [v];**

**for (int i = 0; i < v; i++)**

**{**

**G[i] = new int[v];**

**}**

**for (int i = 0; i < v; i++)**

**{**

**for (int j = 0; j < v; j++)**

**{**

**if (j < i)**

**{**

**G[i][j] = G[j][i];**

**}**

**else**

**{**

**G[i][j] = rand() % 2;**

**}**

**}**

**}**

**return G;**

**}**

**int cout\_matrix(int\*\* g, int v)**

**{**

**cout << " ";**

**for (int i = 0; i < v; i++)**

**{**

**cout << i << " ";**

**}**

**cout << endl << endl;**

**for (int i = 0; i < v; i++)**

**{**

**cout << i << " ";**

**for (int j = 0; j < v; j++)**

**{**

**cout << g[i][j] << " ";**

**}**

**cout << endl;**

**}**

**return 1;**

**}**

**void DFSD\_matrix(int\*\* g, int s, int size, int\* vis, int\* d)**

**{**

**stack<int> st;**

**st.push(s);**

**vis[s] = 1;**

**d[s] = 0;**

**while (!st.empty())**

**{**

**int v = st.top();**

**st.pop();**

**cout << v << " -> ";**

**for (int i = size - 1; i >= 0; i--)**

**{**

**if (g[v][i] == 1 && vis[i] == 0)**

**{**

**st.push(i);**

**vis[i] = 1;**

**d[i] = d[v] + 1;**

**}**

**}**

**}**

**}**

**Листинг 2.2:**

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**#include <iostream>**

**#include <stack>**

**using namespace std;**

**int\*\* create\_adjacency\_matrix(int v);**

**int cout\_matrix(int\*\* g, int v);**

**void DFSD\_matrix(int\*\* g, int s, int size, int\* vis, int\* depth);**

**struct Node**

**{**

**int vertex;**

**struct Node\* next;**

**};**

**struct Graph**

**{**

**int vertexes\_amount;**

**struct Node\*\* list;**

**};**

**struct Graph\* create\_adjacency\_list(int vertexes);**

**struct Node\* create\_vertex(int vertex);**

**void connect\_vertexes(struct Graph\* graph, int coll, int dest);**

**void cout\_adjacency\_list(struct Graph\* graph);**

**void DFSD\_list(struct Graph\* G, int s, int size, bool\* vis);**

**void main()**

**{**

**int\*\* M;**

**int vertexes = 0;**

**int to\_start\_with = 0;**

**cout << "input graph size: ";**

**cin >> vertexes;**

**M = create\_adjacency\_matrix(vertexes);**

**cout\_matrix(M, vertexes);**

**int\* visited = new int[vertexes];**

**int\* depth = new int[vertexes];**

**for (int i = 0; i < vertexes; i++)**

**{**

**visited[i] = 0;**

**depth[i] = -1;**

**}**

**cout << endl << "input input number of vertex to star with: ";**

**cin >> to\_start\_with;**

**cout << endl << "First-deep search: " << endl;**

**DFSD\_matrix(M, to\_start\_with, vertexes, visited, depth);**

**cout << endl << endl << " Depth of vertexes: " << endl;**

**cout << "vertex depth" << endl;**

**for (int i = 0; i < vertexes; i++)**

**{**

**cout << " " << i << " " << depth[i] << endl;**

**}**

**cout << endl << "--------------------------------------------" << endl;**

**struct Graph\* G1 = create\_adjacency\_list(vertexes);**

**for (int i = 0; i < vertexes; i++)**

**{**

**for (int j = 0; j < vertexes; j++)**

**{**

**if (M[i][j] == 1)**

**{**

**connect\_vertexes(G1, i, j);**

**}**

**}**

**}**

**cout\_adjacency\_list(G1);**

**for (int i = 0; i < vertexes; i++)**

**{**

**visited[i] = 0;**

**depth[i] = -1;**

**}**

**cout << endl << "First-deep search: " << endl;**

**DFSD\_matrix(M, to\_start\_with, vertexes, visited, depth);**

**cout << endl << endl << " Depth of vertexes: " << endl;**

**cout << "vertex depth" << endl;**

**for (int i = 0; i < vertexes; i++)**

**{**

**cout << " " << i << " " << depth[i] << endl;**

**}**

**cout << endl << "--------------------------------------------" << endl;**

**return;**

**}**

**int\*\* create\_adjacency\_matrix(int v)**

**{**

**srand(time(NULL));**

**int\*\* G = new int\* [v];**

**for (int i = 0; i < v; i++)**

**{**

**G[i] = new int[v];**

**}**

**for (int i = 0; i < v; i++)**

**{**

**for (int j = 0; j < v; j++)**

**{**

**if (j < i)**

**{**

**G[i][j] = G[j][i];**

**}**

**else**

**{**

**G[i][j] = rand() % 2;**

**}**

**}**

**}**

**return G;**

**}**

**int cout\_matrix(int\*\* g, int v)**

**{**

**cout << " ";**

**for (int i = 0; i < v; i++)**

**{**

**cout << i << " ";**

**}**

**cout << endl << endl;**

**for (int i = 0; i < v; i++)**

**{**

**cout << i << " ";**

**for (int j = 0; j < v; j++)**

**{**

**cout << g[i][j] << " ";**

**}**

**cout << endl;**

**}**

**return 1;**

**}**

**void DFSD\_matrix(int\*\* g, int s, int size, int\* vis, int\* d)**

**{**

**stack<int> st;**

**st.push(s);**

**vis[s] = 1;**

**d[s] = 0;**

**while (!st.empty())**

**{**

**int v = st.top();**

**st.pop();**

**cout << v << " -> ";**

**for (int i = size - 1; i >= 0; i--)**

**{**

**if (g[v][i] == 1 && vis[i] == 0)**

**{**

**st.push(i);**

**vis[i] = 1;**

**d[i] = d[v] + 1;**

**}**

**}**

**}**

**}**

**struct Graph\* create\_adjacency\_list(int vertexes)**

**{**

**struct Graph\* graph = new struct Graph;**

**graph->vertexes\_amount = vertexes;**

**graph->list = new struct Node\* [vertexes];**

**for (int i = 0; i < vertexes; i++)**

**{**

**graph->list[i] = new struct Node[vertexes];**

**}**

**for (int i = 0; i < vertexes; i++)**

**{**

**graph->list[i] = create\_vertex(i);**

**}**

**return graph;**

**}**

**struct Node\* create\_vertex(int vertex)**

**{**

**struct Node\* new\_node = new struct Node;**

**new\_node->vertex = vertex;**

**new\_node->next = NULL;**

**return new\_node;**

**}**

**void connect\_vertexes(struct Graph\* graph, int coll, int dest)**

**{**

**struct Node\* new\_node = create\_vertex(dest);**

**int i = 0;**

**while (graph->list[i]->vertex != coll)**

**{**

**i++;**

**}**

**struct Node\* tmp = graph->list[i];**

**while (tmp->next != NULL)**

**{**

**tmp = tmp->next;**

**}**

**tmp->next = new\_node;**

**}**

**void cout\_adjacency\_list(struct Graph\* graph)**

**{**

**cout << endl << "adjacency list:" << endl;**

**struct Node\* tmp;**

**for (int i = 0; i < graph->vertexes\_amount; i++)**

**{**

**tmp = graph->list[i];**

**while (tmp)**

**{**

**cout << tmp->vertex;**

**tmp = tmp->next;**

**if (tmp != NULL)**

**{**

**cout << " -> ";**

**}**

**}**

**cout << endl;**

**}**

**}**

**void DFSD\_list(struct Graph\* G, int s, int size, bool\* vis)**

**{**

**std::stack<int> st;**

**st.push(s);**

**vis[s] = true;**

**while (!st.empty())**

**{**

**int v = st.top();**

**st.pop();**

**cout << v << " -> ";**

**Node\* l = G->list[v];**

**while (l != NULL)**

**{**

**int neighbor = l->vertex;**

**if (!vis[neighbor])**

**{**

**st.push(neighbor);**

**vis[neighbor] = true;**

**}**

**l = l->next;**

**}**

**}**

**}**

**Листинг 2.3:**

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**#include <iostream>**

**#include <queue>**

**#include <stack>**

**#include <time.h>**

**using namespace std;**

**int\*\* create\_adjacency\_matrix(int v);**

**int cout\_matrix(int\*\* G, int v);**

**void BFS\_matrix(int\*\* g, int v, int size, bool\* vis, int\* d);**

**void DFSD\_matrix(int\*\* g, int s, int size, bool\* vis, int\* depth);**

**void main()**

**{**

**clock\_t start = 0, end = 0, res\_BFS = 0, res\_DFS = 0;**

**int size, to\_start\_with;**

**cout << endl << "input input number of vertex to star with: ";**

**cin >> to\_start\_with;**

**for (size = 1000; size <= 10000; size = size + 1000)**

**{**

**int\*\* M = create\_adjacency\_matrix(size);**

**//cout\_matrix(M, size);**

**bool\* visited = new bool[size];**

**int\* depth = new int[size];**

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

**{**

**visited[i] = 0;**

**depth[i] = -1;**

**}**

**//cout << "Breadth-first search: " << endl;**

**start = clock();**

**while (!visited[to\_start\_with])**

**{**

**BFS\_matrix(M, to\_start\_with, size, visited, depth);**

**}**

**end = clock();**

**res\_BFS = end - start;**

**/\***

**cout << endl << endl << " Depth of vertexes: " << endl;**

**cout << "vertex depth" << endl;**

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

**{**

**cout << " " << i << " " << depth[i] << endl;**

**}**

**cout << endl << "--------------------------------------------" << endl;**

**\*/**

**start = 0, end = 0;**

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

**{**

**visited[i] = 0;**

**depth[i] = -1;**

**}**

**//cout << "Deep-first search: " << endl;**

**start = clock();**

**DFSD\_matrix(M, to\_start\_with, size, visited, depth);**

**end = clock();**

**res\_DFS = end - start;**

**cout << "---------------------------------" << endl;**

**cout << "matrix size: " << size << endl;**

**cout << "time spent on Breadth-first search: " << res\_BFS / 10000.0 << endl;**

**cout << "time spent on Deep-first search: " << res\_DFS / 10000.0 << endl;**

**delete[] M;**

**delete[] visited;**

**delete[] depth;**

**}**

**return;**

**}**

**int\*\* create\_adjacency\_matrix(int v)**

**{**

**srand(time(NULL));**

**int\*\* G = new int\* [v];**

**for (int i = 0; i < v; i++)**

**{**

**G[i] = new int[v];**

**}**

**for (int i = 0; i < v; i++)**

**{**

**for (int j = 0; j < v; j++)**

**{**

**if (j < i)**

**{**

**G[i][j] = G[j][i];**

**}**

**else**

**{**

**G[i][j] = rand() % 2;**

**}**

**}**

**}**

**return G;**

**}**

**int cout\_matrix(int\*\* g, int v)**

**{**

**cout << " ";**

**for (int i = 0; i < v; i++)**

**{**

**cout << i << " ";**

**}**

**cout << endl << endl;**

**for (int i = 0; i < v; i++)**

**{**

**cout << i << " ";**

**for (int j = 0; j < v; j++)**

**{**

**cout << g[i][j] << " ";**

**}**

**cout << endl;**

**}**

**return 1;**

**}**

**void BFS\_matrix(int\*\* g, int v, int size, bool\* vis, int\* d)**

**{**

**queue<int> q;**

**q.push(v);**

**vis[v] = 1;**

**d[v] = 0;**

**while (!q.empty())**

**{**

**v = q.front();**

**q.pop();**

**//cout << v << " -> ";**

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

**{**

**if (g[v][i] == 1 && !vis[i])**

**{**

**d[i] = d[v] + 1;**

**vis[i] = 1;**

**q.push(i);**

**}**

**}**

**}**

**}**

**void DFSD\_matrix(int\*\* g, int s, int size, bool\* vis, int\* d)**

**{**

**stack<int> st;**

**st.push(s);**

**vis[s] = 1;**

**d[s] = 0;**

**while (!st.empty())**

**{**

**int v = st.top();**

**st.pop();**

**//cout << v << " -> ";**

**for (int i = size - 1; i >= 0; i--)**

**{**

**if (g[v][i] == 1 && vis[i] == 0)**

**{**

**st.push(i);**

**vis[i] = 1;**

**d[i] = d[v] + 1;**

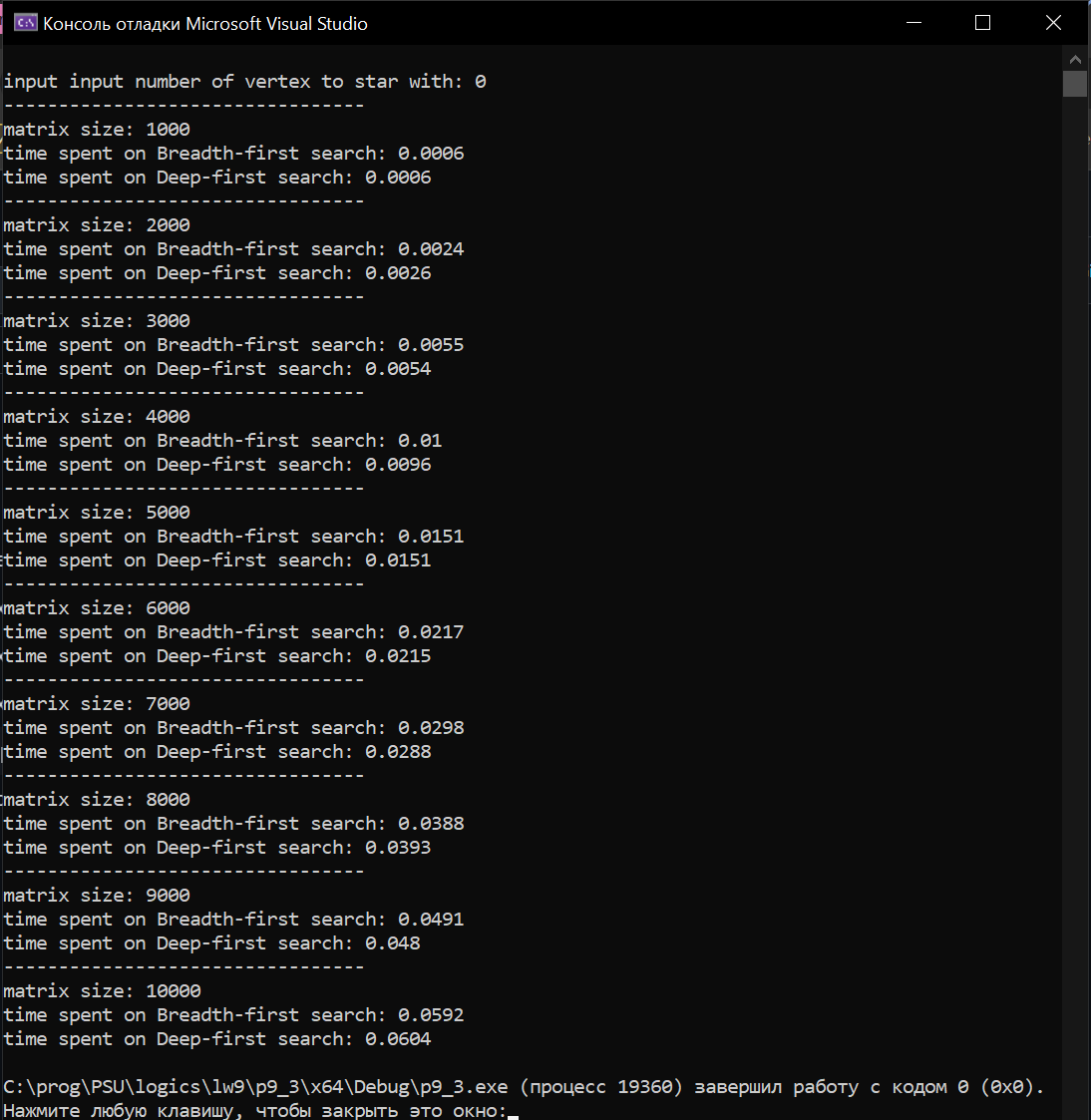
**}**

**}**

**}**

**}**

**Результат работы программы 2.3:**

****

**Анализ результатов задания 2.3:**

|  |  |  |
| --- | --- | --- |
| **Размер массива** | **Время выполнения (в мс)** | |
| **В ширину** | **В глубину** |
| **1000** | **6** | **6** |
| **2000** | **24** | **26** |
| **3000** | **54** | **55** |
| **4000** | **100** | **96** |
| **5000** | **151** | **151** |
| **6000** | **217** | **215** |
| **7000** | **298** | **288** |
| **8000** | **388** | **393** |
| **9000** | **491** | **480** |
| **10000** | **592** | **604** |

**Вывод:**

1. Были изучены и освоены процедуры поиска расстояний в графе для матриц и списков смежности на основе алгоритма поиска в ширину.
2. Самостоятельно реализованы процедуры поиска расстояний на основе обхода в глубину для графов, представленных матрицами и списками смежности.
3. Результаты сравнения процедур поиска расстояний на основе обхода в глубину и на основе обхода в ширину были проанализированы:  
   Разница незначительна.