Министерство науки и высшего образования Российской Федерации

Пензенской государственный университет

Кафедра "Вычислительная техника"

**Отчёт**

по лабораторной работе №8

по курсу «Логика и Основы Алгоритмизации в Инженерных Задачах»

на тему «Обход графа в ширину»

Выполнил:

Студент группы 23ВВ1

Юров Д.М.

Принял:

Митрохин М.А.

Юрова О.В.

Пенза 2024

**Цель работы:** изучить алгоритм обхода графа в ширину, написать программу, реализующую обход.

**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 обхода в ширину, реализованную в соответствии с приведенным выше описанием. При реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки С++.

**3.**\* Реализуйте процедуру обхода в ширину для графа, представленного списками смежности.

### **Задание 2\***

1. Для матричной формы представления графов реализуйте алгоритм обхода в ширину с использованием очереди, построенной на основе структуры данных «список», самостоятельно созданной в лабораторной работе № 3.
2. Оцените время работы двух реализаций алгоритмов обхода в ширину (использующего стандартный класс **queue** и использующего очередь, реализованную самостоятельно) 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 << "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;**

**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 BFS\_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 BFS\_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);**

**cout << endl;**

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

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

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

**{**

**visited[i] = 0;**

**depth[i] = 0;**

**}**

**cout << "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;**

**struct Graph\* G1 = 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(G1, i, j);**

**}**

**}**

**}**

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

**cout << endl;**

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

**{**

**visited[i] = 0;**

**depth[i] = 0;**

**}**

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

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

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

**cout << endl;**

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

**BFS\_list(G1, to\_start\_with, size, visited, 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);**

**}**

**}**

**}**

**}**

**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 BFS\_list(struct Graph\* G, int start, int size, bool\* vis, int\* d)**

**{**

**std::queue<int> q;**

**q.push(start);**

**vis[start] = true;**

**d[start] = 0;**

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

**{**

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

**q.pop();**

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

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

**while (l != NULL)**

**{**

**v = l->vertex;**

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

**{**

**q.push(v);**

**vis[v] = true;**

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

**}**

**l = l->next;**

**}**

**}**

**}**

**Листинг 2.1:**

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

**#include <iostream>**

**using namespace std;**

**struct Node**

**{**

**int vertex;**

**struct Node\* next;**

**};**

**struct Graph**

**{**

**int vertexes\_amount;**

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

**};**

**struct Queue**

**{**

**int inf;**

**struct Queue\* next;**

**};**

**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);**

**struct Queue\* make\_struct(int data);**

**int pop\_q();**

**void push\_q(int v);**

**struct Queue\* head = NULL, \* last = NULL, \* f = NULL, \* tmp = NULL;**

**void main()**

**{**

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

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

**cin >> size;**

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

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

**cout << endl;**

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

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

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

**{**

**visited[i] = 0;**

**depth[i] = 0;**

**}**

**cout << "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;**

**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)**

**{**

**struct Queue\* q = NULL;**

**push\_q(v);**

**vis[v] = 1;**

**d[v] = 0;**

**while (!head == NULL)**

**{**

**v = head->inf;**

**pop\_q();**

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

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

**{**

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

**{**

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

**vis[i] = 1;**

**push\_q(i);**

**}**

**}**

**}**

**}**

**struct Queue\* make\_struct(int data)**

**{**

**struct Queue\* new\_str = new struct Queue;**

**new\_str->inf = data;**

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

**return new\_str;**

**}**

**void push\_q(int data)**

**{**

**struct Queue\* p = make\_struct(data);**

**if (head == NULL && p != NULL)**

**{**

**head = p;**

**last = p;**

**}**

**else if (head != NULL && p != NULL)**

**{**

**last->next = p;**

**last = p;**

**}**

**}**

**int pop\_q()**

**{**

**tmp = head->next;**

**int data = head->inf;**

**delete head;**

**head = tmp;**

**return data;**

**}**

**Листинг 2.2:**

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

**#include <iostream>**

**#include <queue>**

**#include <time.h>**

**using namespace std;**

**struct Node**

**{**

**int vertex;**

**struct Node\* next;**

**};**

**struct Graph**

**{**

**int vertexes\_amount;**

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

**};**

**struct Queue**

**{**

**int inf;**

**struct Queue\* next;**

**};**

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

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

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

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

**struct Queue\* make\_struct(int data);**

**int pop\_q();**

**void push\_q(int v);**

**struct Queue\* head = NULL, \* last = NULL, \* f = NULL, \* tmp = NULL;**

**void main()**

**{**

**clock\_t start = 0, end = 0, res\_q = 0, res\_Queue = 0;**

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

**cout << "input input number of vertex to start Broad-first search with: ";**

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

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

**{**

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

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

**//cout << endl;**

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

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

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

**{**

**visited[i] = 0;**

**depth[i] = 0;**

**}**

**//cout << "Breadth-first search using c++ class queue: " << endl;**

**start = clock();**

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

**{**

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

**}**

**end = clock();**

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

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

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

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

**{**

**visited[i] = 0;**

**depth[i] = 0;**

**}**

**//cout << "Breadth-first search using self-made Queue struct: " << endl;**

**start = clock();**

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

**{**

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

**}**

**end = clock();**

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

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

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

**cout << "time spent with using c++ class queue: " << res\_q / 10000.0 << endl;**

**cout << "time spent with using self-made Queue struct: " << res\_Queue / 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\_Queue(int\*\* g, int v, int size, bool\* vis, int\* d)**

**{**

**struct Queue\* q = NULL;**

**push\_q(v);**

**vis[v] = 1;**

**d[v] = 0;**

**while (!head == NULL)**

**{**

**v = head->inf;**

**pop\_q();**

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

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

**{**

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

**{**

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

**vis[i] = 1;**

**push\_q(i);**

**}**

**}**

**}**

**}**

**void BFS\_matrix\_q(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 Queue\* make\_struct(int data)**

**{**

**struct Queue\* new\_str = new struct Queue;**

**new\_str->inf = data;**

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

**return new\_str;**

**}**

**void push\_q(int data)**

**{**

**struct Queue\* p = make\_struct(data);**

**if (head == NULL && p != NULL)**

**{**

**head = p;**

**last = p;**

**}**

**else if (head != NULL && p != NULL)**

**{**

**last->next = p;**

**last = p;**

**}**

**}**

**int pop\_q()**

**{**

**tmp = head->next;**

**int data = head->inf;**

**delete head;**

**head = tmp;**

**return data;**

**}**

**Анализ результатов задания 2.2:**

|  |  |  |
| --- | --- | --- |
| **Размер массива** | **Время выполнения (в мс)** | |
| **class std::queue** | **struct Queue** |
| **1000** | **6** | **5** |
| **2000** | **23** | **22** |
| **3000** | **53** | **51** |
| **4000** | **91** | **88** |
| **5000** | **141** | **141** |
| **6000** | **204** | **199** |
| **7000** | **274** | **272** |
| **8000** | **356** | **348** |
| **9000** | **464** | **454** |
| **10000** | **569** | **553** |

**Вывод:**

1. Были изучены алгоритмы обхода в ширину для матрицы смежности и списков смежности.
2. Алгоритмы обхода были реализованы в созданных в ходе выполнения программах.
3. В качестве очереди для алгоритма обхода в ширину были использованы класс std::queue и самостоятельно созданная структура Queue. Результаты были сравнены на обходах одинаковых матриц разных размеров:   
   Разница между использованием в качестве очереди для алгоритма обхода в ширину класса std::queue и самостоятельно созданной структуры Queue незначительна.

**Дополнительное задание:** реализовать вывод массива номеров вершин, отсортированного по глубинам вершин.

**Листинг:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <queue>

#include <map>

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);

cout << endl;

bool\* visited = new bool[size];

int\* depth = new int[size];

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

{

visited[i] = 0;

depth[i] = 0;

}

cout << "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;

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] = true;

d[v] = 0;

map<int, vector<int>> depth\_map;

while (!q.empty())

{

v = q.front();

q.pop();

depth\_map[d[v]].push\_back(v);

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

{

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

{

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

vis[i] = true;

q.push(i);

}

}

}

cout << endl << "Vertices grouped by depth:" << endl;

for (const auto& depth\_pair : depth\_map)

{

int depth = depth\_pair.first;

const vector<int>& vertices = depth\_pair.second;

cout << "Depth " << depth << ": ";

for (int vertex : vertices)

{

cout << vertex << " ";

}

cout << endl;

}

}

**Результат работы программы:**

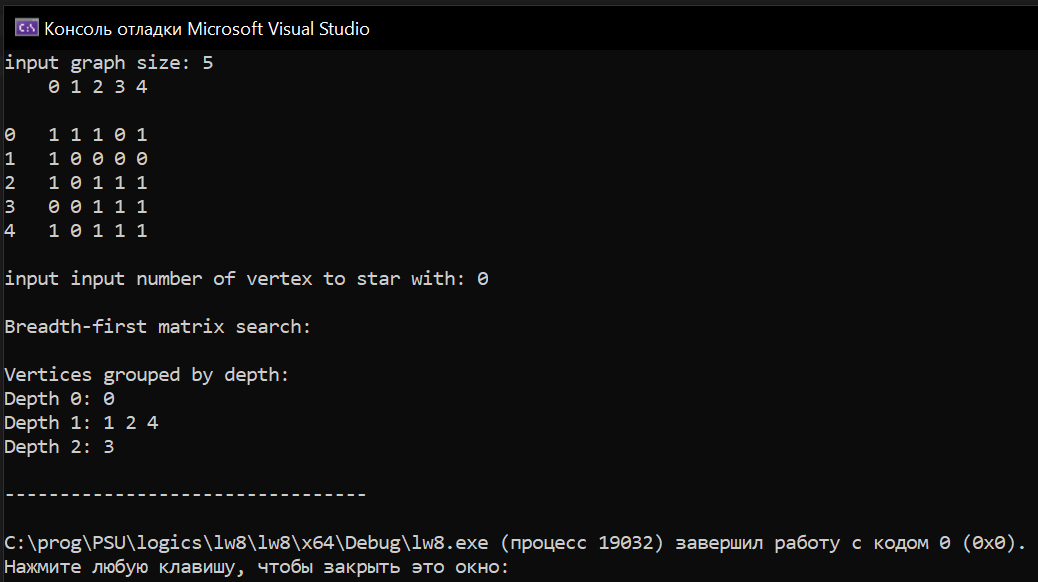


Рисунок - result\_add.\_task

**Вывод:**

1. В ходе выполнения дополнительного задания был изучен и освоен класс map.
2. Был реализован алгоритм вывода массива номеров вершин, отсортированного по глубинам вершин.