Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №5

# по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Обход графа в ширину»

**Выполнили:**

студенты группы 20ВВ3

Пантюшов Егор

Шмелёв Данила

**Приняли:**

Юрова О.В.

Митрохин М.А.

Пенза 2021

# Название

Обход графа в ширину

**Цель работы** – изучение алгоритма обхода графа в ширину.

# Методические указания

# Обход графа в ширину – еще один распространенный способ обхода графов. Основная идея такого обхода состоит в том, чтобы посещать вершины по уровням удаленности от исходной вершины. Удалённость в данном случае понимается как количество ребер, по которым необходимо прейти до достижения вершины.

# Лабораторное задание

# 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++.

**Псевдокод**

Вход: G – матрица смежности графа.

Выход: номера вершин в порядке их прохождения на экране.

Алгоритм ПОШ

1.1. для всех i положим NUM[i] = False пометим как &quot;не посещенную&quot;

1.2. ПОКА существует &quot;новая&quot; вершина v

1.3. ВЫПОЛНЯТЬ BFS (v).

Алгоритм BFS(v):

2.1. Создать пустую очередь Q = {};

2.2. Поместить v в очередь Q.push(v);

2.3. пометить v как &quot;посещенную&quot; NUM[v] = True;

2.4. ПОКА Q != Æ очередь не пуста ВЫПОЛНЯТЬ

2.5. v = Q.front() установить текущую вершину;

2.6. Удалить первый элемент из очереди Q.pop();

2.7. вывести на экран v;

2.8. ДЛЯ i = 1 ДО size\_G ВЫПОЛНЯТЬ

2.9. ЕСЛИ G(v,i) = = 1И NUM[i] = = False

2.10. ТО

2.11. Поместить i в очередь Q.push(i);

2.12. пометить v как &quot;посещенную&quot; NUM[v] = True;

# Листинг

#define \_CRT\_NONSTDC\_NO\_WARNINGS

#define \_CRT\_SECURE\_NO\_WARNINGS

#define HEADER ("Лабораторная работа №5\nВыполнили: Шмелёв Д. и Пантюшов Е.\n\n")

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#include <time.h>

#include <stdbool.h>

#include <queue>

using namespace std;

struct node

{

int vertex;

struct node\* next;

};

struct Graph

{

int numVertices;

struct node\*\* adjList;

};

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

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

graph->adjList[i] = NULL;

return graph;

}

void addEdge(struct Graph\* graph, int i, int j)

{

struct node\* newNode = createNode(j);

if (graph->adjList[i] == 0)

{

graph->adjList[i] = newNode;

newNode = NULL;

}

struct node\* temp = graph->adjList[i];

while (temp->next)

{

temp = temp->next;

}

temp->next = newNode;

newNode = createNode(i);

if (graph->adjList[j] == 0)

{

graph->adjList[j] = newNode;

newNode = NULL;

}

temp = graph->adjList[j];

while (temp->next)

{

temp = temp->next;

}

temp->next = newNode;

}

void printGraph(struct Graph\* graph)

{

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

{

struct node\* temp = graph->adjList[i];

printf("%d-я вершина: ", i);

while (temp)

{

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

temp = temp->next;

}

printf("\n");

}

}

struct qnode

{

int e;

struct qnode\* next;

};

struct q

{

struct qnode\* frnt, \* rear;

};

void Push(int e, struct q\* q)

{

struct qnode\* temp = (struct qnode\*)malloc(sizeof(struct qnode));

temp->e = e;

temp->next = NULL;

if (q->frnt == NULL)

{

q->frnt = temp;

q->rear = temp->next;

}

else

{

q->rear = temp;

q->rear->next = NULL;

}

}

void Pop(struct q\* q)

{

if (!q->frnt)

{

q->frnt = q->frnt->next;

}

}

void bfs(int\*\* G, int\* NUM, int v, int n)

{

clock\_t t1, t2;

float res;

t1 = clock();

int w = v;

queue <int> q1;

q1.push(v);

NUM[v] = true;

while (!q1.empty())

{

v = q1.front();

q1.pop();

printf(" %d ", v);

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

{

if ((G[v][i] == 1) && (NUM[i] == false))

{

q1.push(i);

NUM[i] = true;

}

}

}

t2 = clock();

res = t2 - t1;

printf("\nВремя(библиотека): %f\n", res / 1000);

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

{

NUM[i] = false;

}

t1 = clock();

struct q\* q2 = (struct q\*)malloc(sizeof(struct q));

Push(w, q2);

NUM[w] = true;

while (!q2->frnt)

{

w = q2->frnt->e;

Pop(q2);

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

{

if ((G[w][i] == 1) && (NUM[i] == false))

{

Push(i, q2);

NUM[i] = true;

}

}

}

t2 = clock();

res = t2 - t1;

printf("Время(структура): %f\n", res / 1000);

}

void bfs\_list(struct Graph\* graph, int\* NUM, int v, int n)

{

clock\_t t1, t2;

float res;

t1 = clock();

int w = v;

queue <int> q1;

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

q1.push(v);

NUM[v] = true;

while (!q1.empty())

{

v = q1.front();

node = graph->adjList[v];

q1.pop();

printf(" %d ", v);

while (node)

{

if (NUM[node->vertex] == false)

{

q1.push(node->vertex);

NUM[node->vertex] = true;

}

node = node->next;

}

}

t2 = clock();

res = t2 - t1;

printf("\nВремя(библиотека): %f\n", res / 1000);

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

{

NUM[i] = false;

}

t1 = clock();

struct q\* q2 = (struct q\*)malloc(sizeof(struct q));

Push(w, q2);

NUM[w] = true;

while (!q2->frnt)

{

w = q2->frnt->e;

node = graph->adjList[w];

Pop(q2);

while (node)

{

if (NUM[node->vertex] == false)

{

Push(node->vertex, q2);

NUM[node->vertex] = true;

}

node = node->next;

}

}

t2 = clock();

res = t2 - t1;

printf("Время(структура): %f\n\n", res / 1000);

}

void task\_1\_1(int\*\* G, int n)

{

printf("\nЗадание 1.\n\nПункт 1.\nМатрица смежности:\n");

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

{

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

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

{

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

}

}

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

{

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

{

if (i == j)

{

G[i][j] = 0;

}

else

{

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

}

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

}

printf("\n");

}

}

void task\_1\_2(int\*\* G, int n)

{

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

int v;

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

{

NUM[i] = false;

}

printf("\nПункт 2.\nВведите точку входа: ");

scanf("%d", &v);

printf("Результат работы алгоритма обхода в ширину(матрица): ");

bfs(G, NUM, v, n);

free(NUM);

}

void task\_1\_3(int\*\* G, int n)

{

struct Graph\* graph = createGraph(n);

int v;

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

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

{

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

{

if ((G[i][j] == 1) && (i < j))

{

addEdge(graph, i, j);

}

}

}

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

{

NUM[i] = false;

}

printf("\nПункт 3.\nСписок смежности: \n");

printGraph(graph);

printf("Введите точку входа: ");

scanf("%d", &v);

printf("Результат работы алгоритма обхода в ширину(список): ");

bfs\_list(graph, NUM, v, n);

free(NUM);

}

int main()

{

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

printf(HEADER);

printf("Введите размерность матрицы: ");

int n;

scanf("%d", &n);

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

task\_1\_1(G, n);

task\_1\_2(G, n);

task\_1\_3(G, n);

system("PAUSE");

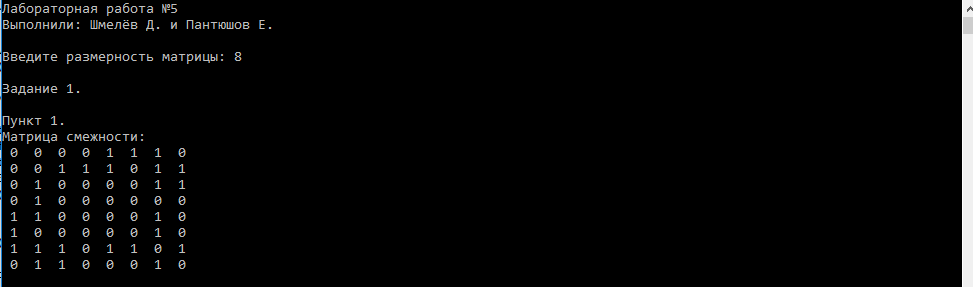
return 0;

}

# Результат работы программы

**Пункт 1.**

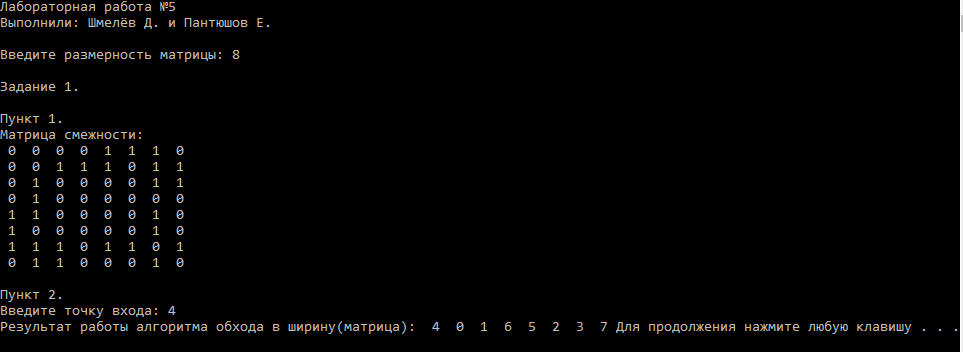
Результаты работы программы показаны на рисунке 1.



**Рисунок 1 – Результат работы программы**

**Пункт 2.**

Результаты работы программы показаны на рисунке 2.



**Рисунок 2 – Результат работы программы**

# Вывод

# В ходе выполнения лабораторной работы была разработана программа, в который был реализован алгоритм обхода графа в ширину. При разработке данного алгоритма была изучена такая структура данных как очередь.