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:**

студент группы 20ВВ3

Пантюшов Егор

**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

**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.

**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\_SECURE\_NO\_WARNINGS

#define HEADER ("Лабораторная работа №5\nВыполнил: Пантюшов Е.И.\n")

#include <queue>

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#include <time.h>

#include <stdbool.h>

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;

}

struct queue1

{

int element;

struct queue1\* next;

};

struct q

{

struct queue1\* peredn, \* zadn;

};

struct q\* sozd()

{

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

a->peredn = 0;

a->zadn = 0;

return (a);

}

void push(struct q\* a, int x)

{

struct queue1\* nov = (queue1\*)malloc(sizeof(queue1));

nov->element = x;

nov->next = 0;

if (a->peredn == 0)

a->peredn = nov;

else {

if (a->zadn == 0) {

a->zadn = nov;

a->peredn->next = nov;

}

else

{

a->zadn->next = nov;

a->zadn = nov;

}

}

}

void pop(struct q\* a, int x)

{

if (a->peredn)

{

struct queue1\* temp = (queue1\*)malloc(sizeof(queue1));

temp = a->peredn;

a->peredn = temp->next;

free(temp);

}

}

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

{

struct node\* newNode = createNode(j);

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

{

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] == NULL)

{

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");

}

}

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

{

clock\_t t1, t2;

float res;

t1 = clock();

queue <int> q;

q.push(v);

NUM[v] = 1;

while (!q.empty())

{

v = q.front();

q.pop();

printf(" %d ", v);

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

{

if ((G[v][i] == 1) && (NUM[i] == 0))

{

q.push(i);

NUM[i] = 1;

}

}

}

t2 = clock();

res = t2 - t1;

printf("\nВремя работы алгоритма через библиотеку: %f\n", res / 1000);

}

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

{

queue <int> q;

q.push(v);

NUM[v] = 1;

while (!q.empty())

{

v = q.front();

q.pop();

printf("%d ", v);

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

while (list)

{

if (NUM[list->vertex] == 0)

{

q.push(list->vertex);

NUM[list->vertex] = 1;

}

list = list->next;

}

}

}

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

{

clock\_t t1, t2;

float res;

t1 = clock();

struct q\* a = sozd();

push(a, v);

NUM[v] = 1;

while (a->peredn)

{

v = a->peredn->element;

pop(a, v);

printf(" %d ", v);

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

{

if ((G[v][i] == 1) && (NUM[i] == 0))

{

push(a, i);

NUM[i] = 1;

}

}

}

t2 = clock();

res= res = t2 - t1;

printf("\nВремя работы алгоритма через очередь,реализованную самостоятельо %f\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(struct Graph\* graph, int n, int\*\* G)

{

struct node\*\* node = graph->adjList;

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("\n\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);

printf("\n\n");

free(NUM);

}

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

{

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

int v;

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

{

NUM[i] = false;

}

printf("Введите точку входа: ");

scanf("%d", &v);

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

bfs\_sv(G, NUM, v, n);

printf("\n\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\*));

struct Graph\* graph = createGraph(n);

task\_1\_1(G, n);

task\_1\_2(G, n);

task\_1\_3(graph, n, G);

task\_2\_1(graph, n, G);

free(G);

system("PAUSE");

return 0;

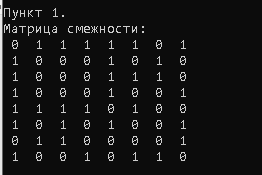
}

# Результат работы программы

**Задание 1**

**Пункт 1.**

Результаты работы программы показаны на рисунке 1.



**Рисунок 1 – Результат работы программы**

**Пункт 2.**

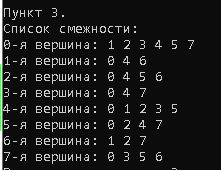
Результаты работы программы показаны на рисунке 2.



**Рисунок 2 – Результат работы программы**

**Пункт 3.**

Результаты работы программы показаны на рисунке 3.

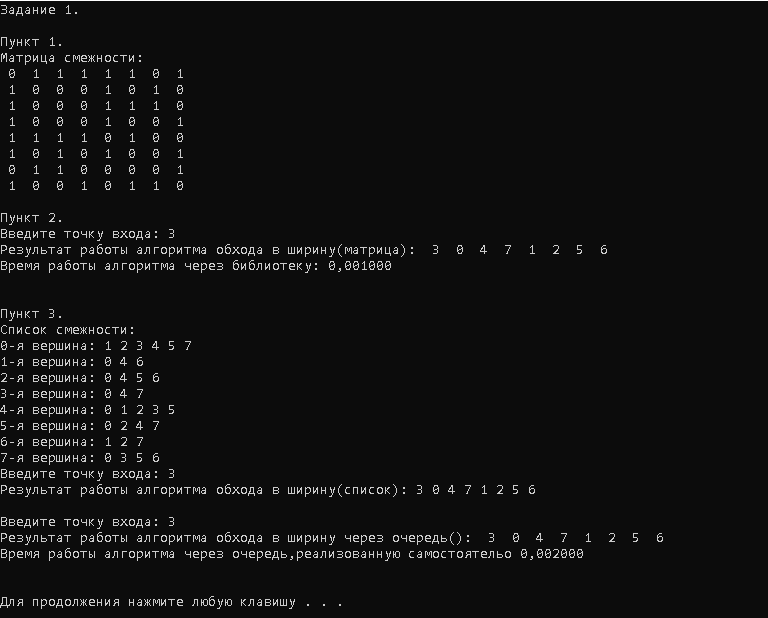


**Рисунок 3 – Результат работы программы**

**Задание 2**

Пункт 1

Результаты работы программы показаны на рисунке 4.



**Рисунок 3 – 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.Работа алгоритма быстрее через встроенную библиотеку.