Министерство образования Российской Федерации

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 20ВВ2

Александров В.С.

Кирюткин И.А.

Сафронов Д.В.

Принял:

к.т.н., доцент

Митрохин М.А.

**Пенза 2021**

**Цель работы:** Написать программу, которая осуществляет обход графа в ширину.

**Лабораторные задания:**

**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.

**Листинг программы Bfs1:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

#include <iostream>

#include <stdio.h>

#include <string.h>

#include <cstdlib>

#include <windows.h>

#include <conio.h>

#include <ctime>

#include <cmath>

#include <vector>

#include <queue> // очередь

#define SIZE 40

using namespace std;

struct queue\_str {

int items[SIZE];

int front;

int rear;

};

struct queue\_str\* createQueue();

void enqueue(struct queue\_str\* q, int);

int dequeue(struct queue\_str\* q);

void display(struct queue\_str\* q);

int isEmpty(struct queue\_str\* q);

void printQueue(struct queue\_str\* q);

void BFS1(int mas[7][7]);

int main1();

struct node

{

int vertex;

struct node\* next;

};

struct node\* createNode(int);

struct Graph

{

int numVertices;

struct node\*\* adjLists;

int\* visited;

};

struct Graph\* createGraph(int vertices);

void addEdge(struct Graph\* graph, int src, int dest);

void printGraph(struct Graph\* graph);

void bfs(struct Graph\* graph, int startVertex);

int main()

{

setlocale(LC\_ALL, "Rus");

struct Graph\* graph = createGraph(6);

addEdge(graph, 0, 1);

addEdge(graph, 0, 2);

addEdge(graph, 1, 2);

addEdge(graph, 1, 4);

addEdge(graph, 1, 3);

addEdge(graph, 2, 4);

addEdge(graph, 3, 4);

bfs(graph, 0);

main1();

return 0;

}

void bfs(struct Graph\* graph, int startVertex) {

struct queue\_str\* q = createQueue();

graph->visited[startVertex] = 1;

enqueue(q, startVertex);

while (!isEmpty(q)) {

printQueue(q);

int currentVertex = dequeue(q);

printf("Посетил %d\n", currentVertex);

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

while (temp) {

int adjVertex = temp->vertex;

if (graph->visited[adjVertex] == 0) {

graph->visited[adjVertex] = 1;

enqueue(q, adjVertex);

}

temp = temp->next;

}

}

}

struct node\* createNode(int v)

{

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

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices)

{

struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));

graph->numVertices = vertices;

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

graph->visited = (int\*)malloc(vertices \* sizeof(int));

int i;

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

graph->adjLists[i] = NULL;

graph->visited[i] = 0;

}

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest)

{

// Add edge from src to dest

struct node\* newNode = createNode(dest);

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

graph->adjLists[src] = newNode;

// Add edge from dest to src

newNode = createNode(src);

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

graph->adjLists[dest] = newNode;

}

struct queue\_str\* createQueue() {

struct queue\_str\* q = (struct queue\_str\*)malloc(sizeof(struct queue\_str));

q->front = -1;

q->rear = -1;

return q;

}

int isEmpty(struct queue\_str\* q) {

if (q->rear == -1)

return 1;

else

return 0;

}

void enqueue(struct queue\_str\* q, int value) {

if (q->rear == SIZE - 1)

//printf("\nОчередь заполнена!!");

printf("");

else {

if (q->front == -1)

q->front = 0;

q->rear++;

q->items[q->rear] = value;

}

}

int dequeue(struct queue\_str\* q) {

int item;

if (isEmpty(q)) {

//printf("Очередь пуста");

item = -1;

}

else {

item = q->items[q->front];

q->front++;

if (q->front > q->rear) {

//printf("Сброс очереди");

q->front = q->rear = -1;

}

}

return item;

}

void printQueue(struct queue\_str\* q) {

int i = q->front;

if (isEmpty(q)) {

// printf("Очередь пуста");

}

else {

// printf("\nОчередь содержит \n");

for (i = q->front; i < q->rear + 1; i++) {

// printf("%d ", q->items[i]);

}

}

}

int main1()

{

int mas[7][7] = { { 0, 1, 1, 0, 0, 0, 1 }, // матрица смежности

{ 1, 0, 1, 1, 0, 0, 0 },

{ 1, 1, 0, 0, 0, 0, 0 },

{ 0, 1, 0, 0, 1, 0, 0 },

{ 0, 0, 0, 1, 0, 1, 0 },

{ 0, 0, 0, 0, 1, 0, 1 },

{ 1, 0, 0, 0, 0, 1, 0 } };

printf("\nобход матрици смежности \n");

BFS1(mas);

return 0;

}

void BFS1(int mas[7][7])

{

queue<int> Queue;

int nodes[7]; // вершины графа

for (int i = 0; i < 7; i++)

nodes[i] = 0; // исходно все вершины равны 0

Queue.push(0); // помещаем в очередь первую вершину

while (!Queue.empty())

{ // пока очередь не пуста

int node = Queue.front(); // извлекаем вершину

Queue.pop();

nodes[node] = 2; // отмечаем ее как посещенную

for (int j = 0; j < 7; j++)

{ // проверяем для нее все смежные вершины

if (mas[node][j] == 1 && nodes[j] == 0)

{ // если вершина смежная и не обнаружена

Queue.push(j); // добавляем ее в очередь

nodes[j] = 1; // отмечаем вершину как обнаруженную

}

}

cout << node + 1 << endl; // выводим номер вершины

}

}

**Листинг программы Bfs2:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

#include <iostream>

#include <stdio.h>

#include <string.h>

#include <cstdlib>

#include <windows.h>

#include <conio.h>

#include <ctime>

#include <cmath>

#include <vector>

#include <queue> // очередь

#define SIZE 40

using namespace std;

struct queue\_str {

int items[SIZE];

int front;

int rear;

};

struct queue\_str\* createQueue();

void enqueue(struct queue\_str\* q, int);

int dequeue(struct queue\_str\* q);

void display(struct queue\_str\* q);

int isEmpty(struct queue\_str\* q);

void printQueue(struct queue\_str\* q);

void BFS2(int mas[7][7]);

int main1();

void printGraph(struct Graph\* graph);

void BFS2(int mas[7][7]);

void BFS2(int mas[7][7])

{

struct queue\_str\* q = createQueue();

int nodes[7]; // вершины графа

for (int i = 0; i < 7; i++)

nodes[i] = 0; // исходно все вершины равны 0

enqueue(q, 0); // помещаем в очередь первую вершину

while (!isEmpty(q))

{ // пока очередь не пуста

printQueue(q);

int node = dequeue(q);

nodes[node] = 2; // отмечаем ее как посещенную

for (int j = 0; j < 7; j++)

{ // проверяем для нее все смежные вершины

if (mas[node][j] == 1 && nodes[j] == 0)

{ // если вершина смежная и не обнаружена

enqueue(q, j); // добавляем ее в очередь

nodes[j] = 1; // отмечаем вершину как обнаруженную

}

}

cout << node + 1 << endl; // выводим номер вершины

}

}

struct queue\_str\* createQueue() {

struct queue\_str\* q = (struct queue\_str\*)malloc(sizeof(struct queue\_str));

q->front = -1;

q->rear = -1;

return q;

}

int isEmpty(struct queue\_str\* q) {

if (q->rear == -1)

return 1;

else

return 0;

}

void enqueue(struct queue\_str\* q, int value) {

if (q->rear == SIZE - 1)

//printf("\nОчередь заполнена!!");

printf("");

else {

if (q->front == -1)

q->front = 0;

q->rear++;

q->items[q->rear] = value;

}

}

int dequeue(struct queue\_str\* q) {

int item;

if (isEmpty(q)) {

//printf("Очередь пуста");

item = -1;

}

else {

item = q->items[q->front];

q->front++;

if (q->front > q->rear) {

//printf("Сброс очереди");

q->front = q->rear = -1;

}

}

return item;

}

void printQueue(struct queue\_str\* q) {

int i = q->front;

if (isEmpty(q)) {

// printf("Очередь пуста");

}

else {

// printf("\nОчередь содержит \n");

for (i = q->front; i < q->rear + 1; i++) {

// printf("%d ", q->items[i]);

}

}

}

int main()

{

int mas[7][7] = { { 0, 1, 1, 0, 0, 0, 1 }, // матрица смежности

{ 1, 0, 1, 1, 0, 0, 0 },

{ 1, 1, 0, 0, 0, 0, 0 },

{ 0, 1, 0, 0, 1, 0, 0 },

{ 0, 0, 0, 1, 0, 1, 0 },

{ 0, 0, 0, 0, 1, 0, 1 },

{ 1, 0, 0, 0, 0, 1, 0 } };

BFS2(mas);

printf("%d", clock());

return 0;

}

**Псевдокод Bfs1:**

Вход: G(v,a) – неориентированный граф

Выход: Номер вершины через node.

Алгоритм поиск в ширину:

Помещаем вершины графов в переменную nodes.

Для i от 0 до 7

Все вершины в nodes = 0

В очередь встаёт первая вершина

Пока очередь не пуста

node = Queue.front();

Добавляем в очередь

nodes[node] = 2

Для j от 0 до 7

Проверяем для нее все смежные вершины

Если mas[node][j] == 1 && nodes[j] == 0

Если вершина смежная и не обнаружена

Добавляем ее в очередь

nodes[j] = 1

**Псевдокод Bfs2:**

Вход: G(v,a) – неориентированный граф

Выход: Номер вершины через node.

Алгоритм писка в ширину:

Помещаем вершины графов в переменную nodes.

Для i от 0 до 7

nodes[i] = 0

Помещаем в очередь первую вершину

Пока очередь не пуста

node = dequeue(q);

nodes[node] = 2

Для j от 0 до 7

проверяем для нее все смежные вершины

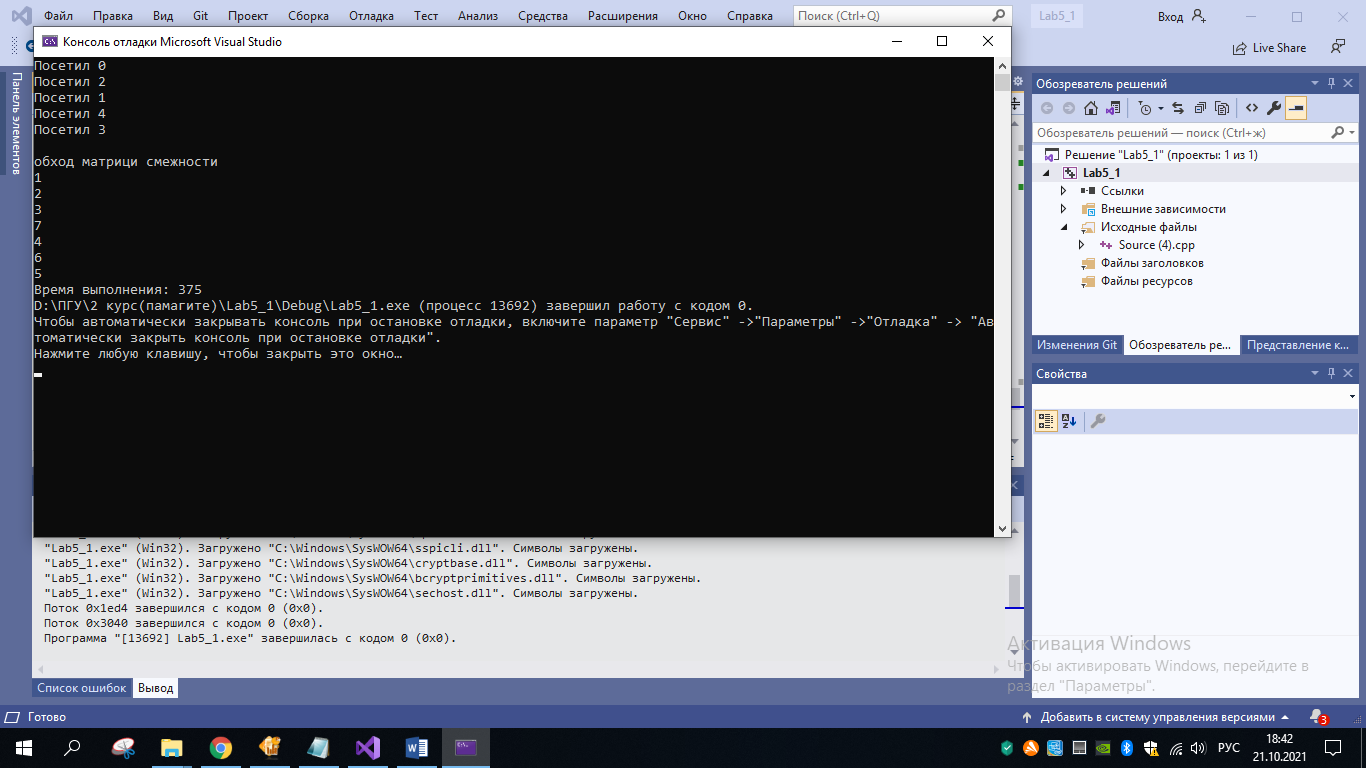
Если (mas[node][j] == 1 && nodes[j] == 0)

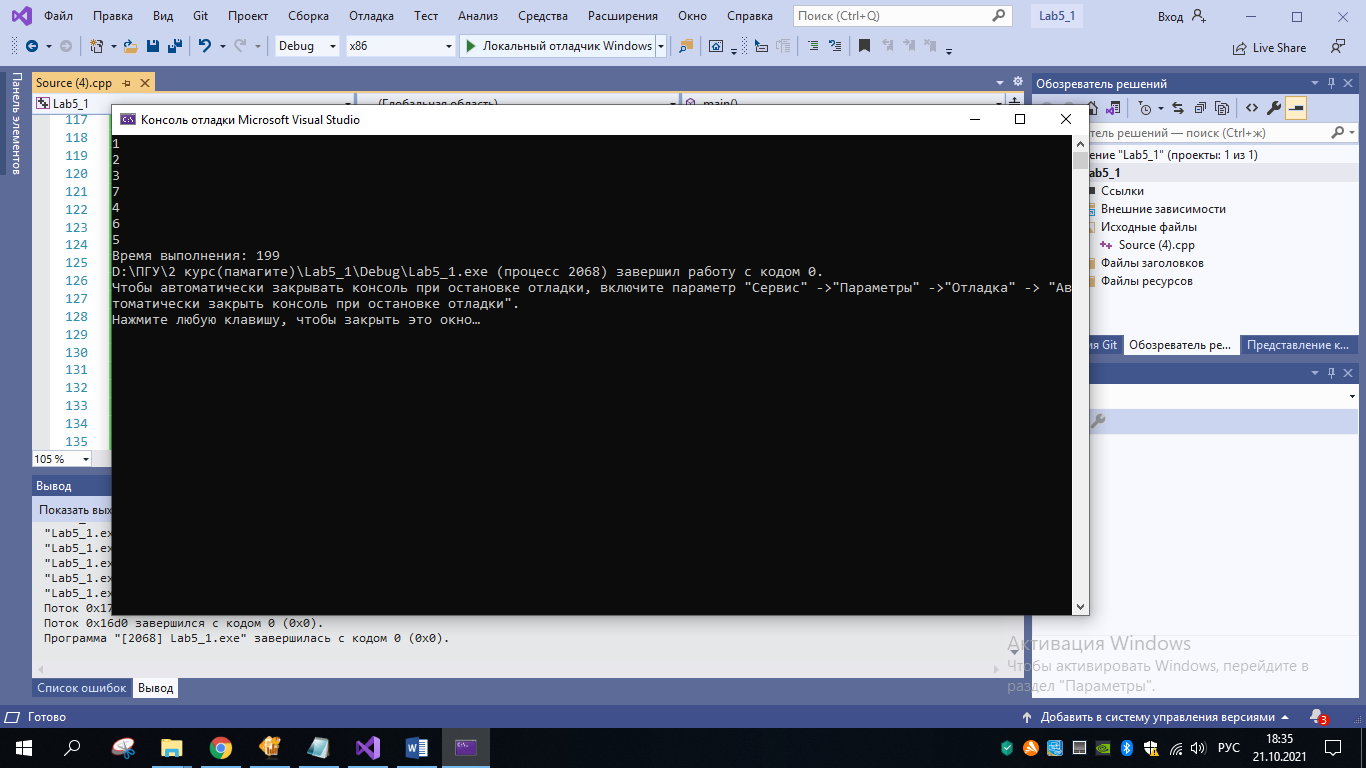
если вершина смежная и не обнаружена

добавляем ее в очередь

отмечаем вершину как обнаруженную

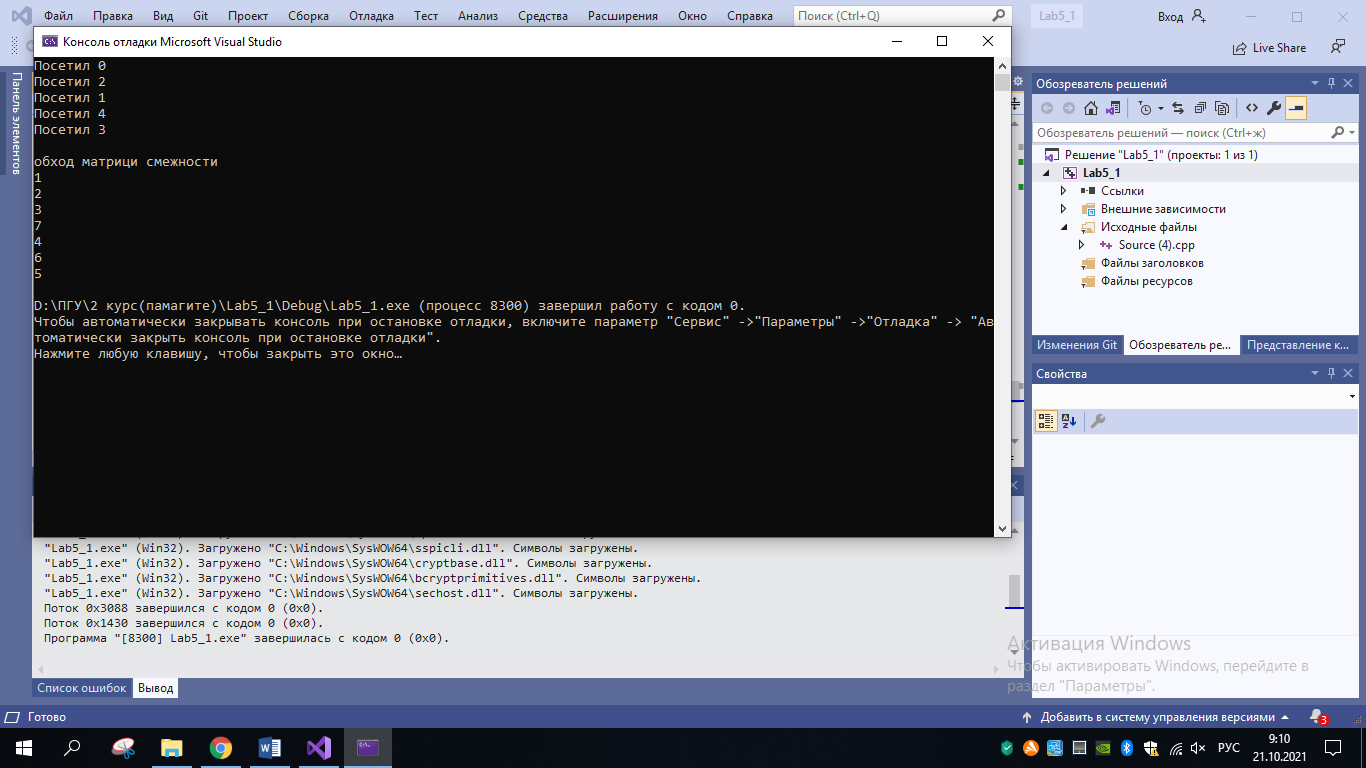
**Сравнение времени выполнения между функциями Bfs1 и Bfs2:**



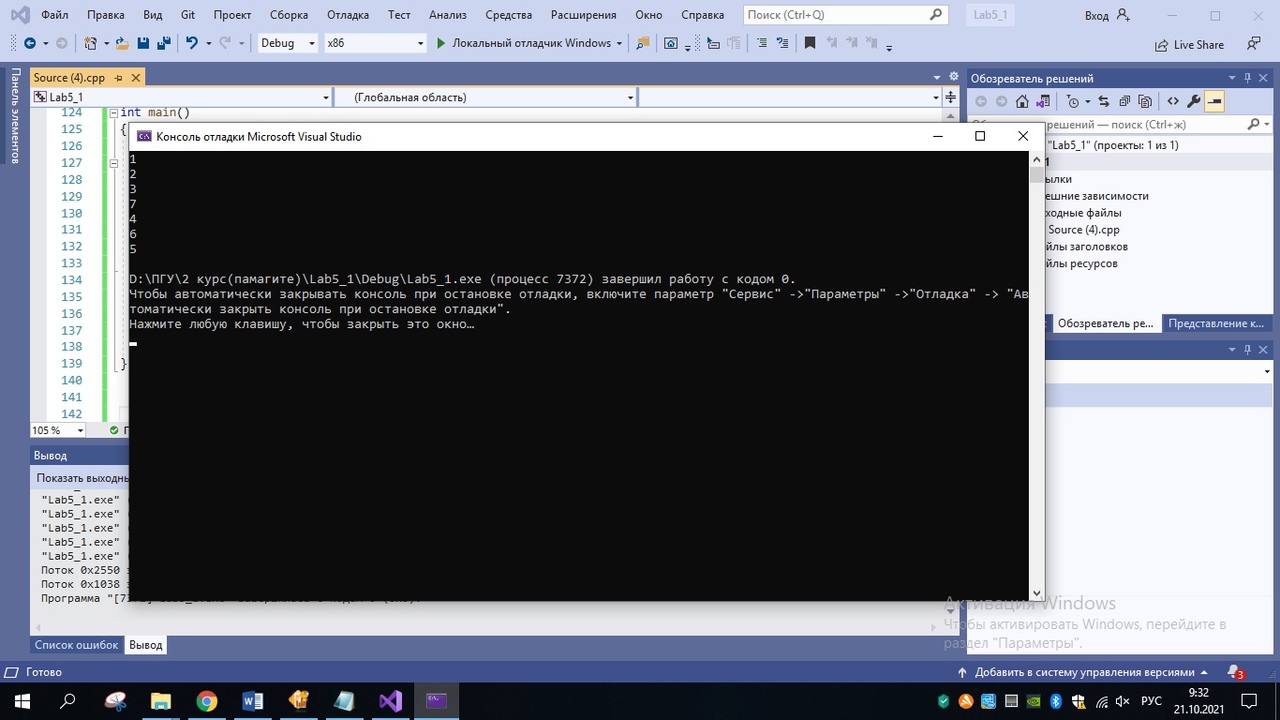


**Результат программы :**

**Bfs1:**



**Bfs2:**



**Вывод:** Написали программу, которая осуществляет обход графа в ширину.