Пензенский государственный университет Кафедра «Вычислительная техника»

**Отчет**о выполнении лабораторной работы №5.2 по дисциплине “Логика и основа алгоритмизации в инженерных задачах” на тему**:**

**Обход графа в ширину**

Выполнили студенты гр. 19ВВ4:   
Хлыстов А. Ю.  
Привалов А. Э.

Проверили:   
Юрова О. В.  
Митрохин М. А.

Пенза, 2020 г.

**Название**Обход графа в ширину

**Цель работы**Научиться реализовывать алгоритм обхода в ширину.

**Лабораторное задание**

Задание 1\*  
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.

**Код программы**

**Задание 1\***

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <time.h>

#include <queue>

using namespace std;

struct graph {

int\*\* matrix;

int size;

}m;

typedef struct Node {

int num;

Node\* next;

Node\* back;

}Node;

typedef struct HeadCells {

Node\* start = NULL;

int empty;

int num;

}HeadCells;

Node\* it;

int\* NUM;

HeadCells\*\* headers;

void outputMatrix(graph m) {

printf\_s(" ");

for (int i = 0; i < m.size; i++) {

printf\_s("%2d ", i);

}

printf\_s("\n");

for (int i = 0; i < m.size; i++) {

printf\_s("%2d ", i);

for (int j = 0; j < m.size; j++) {

printf\_s("%2d ", m.matrix[i][j]);

}

printf\_s("\n");

}

}

//Заполнение массива

void fillMatrix(graph m) {

int value;

for (int i = 0; i < m.size; i++) {

for (int j = i + 1; j < m.size; j++) {

value = rand() % 10;

if (value < 3)

value = 0;

else

value = 1;

m.matrix[i][j] = value;

m.matrix[j][i] = m.matrix[i][j];

}

}

}

//Создание двумерного массива

int\*\* createMatrix(int size) {

int\*\* matrix = (int\*\*)malloc(size \* sizeof(int\*));

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

matrix[i] = (int\*)calloc(size, sizeof(int));

}

return matrix;

}

HeadCells\*\* createHList(int size) {

HeadCells\*\* Headers = (HeadCells\*\*)malloc(size \* sizeof(HeadCells\*));

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

Headers[i] = (HeadCells\*)malloc(sizeof(HeadCells));

Headers[i]->start = NULL;

Headers[i]->empty = 1;

}

return Headers;

}

Node\* createNode() {

Node\* N = (Node\*)malloc(sizeof(Node));

N->back = NULL;

N->next = NULL;

return N;

}

void output\_alist(HeadCells\*\* headers, int size) {

int i = 0;

HeadCells\* work\_vertex;

Node\* tmp;

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

work\_vertex = headers[i];

tmp = work\_vertex->start;

printf\_s("%d. ", i);

if (tmp == NULL) {

printf("-");

}

while (tmp != NULL) {

printf\_s("%d ", tmp->num);

tmp = tmp->next;

}

printf\_s("\n");

}

}

void push\_back(HeadCells\* list, int num) {

Node\* tmp;

if (list->empty == 1) {

it->num = num;

it->next = NULL;

list->start = it;

list->empty = 0;

}

else {

tmp = createNode();

tmp->num = num;

tmp->next = NULL;

it->next = tmp;

tmp->back = it;

it = tmp;

}

}

HeadCells\*\* matrix\_to\_list(graph m) {

HeadCells\*\* headers = createHList(m.size);

int\* line;

HeadCells\* workVertex;

for (int i = 0; i < m.size; i++) {

line = m.matrix[i];

workVertex = headers[i];

if (workVertex->empty == 1) {

it = createNode();

}

for (int j = 0; j < m.size; j++) {

if (line[j] == 1) {

push\_back(workVertex, j);

}

else {

continue;

}

}

}

return headers;

}

void BFS(int v, graph m, queue <int> q, int\* NUM) {

Node\* tmp;

q.push(v);

NUM[v] = 1;

while (q.empty() != true) {

v = q.front();

tmp = headers[v]->start;

q.pop();

printf\_s("%d ", v);

while (tmp != NULL) {

if (NUM[tmp->num] == 0) {

q.push(tmp->num);

NUM[tmp->num] = 1;

}

tmp = tmp->next;

}

}

}

bool checkNewVertex(int\* NUM) {

for (int i = 0; i < m.size; i++) {

if (NUM[m.size - i - 1] == 0) {

return true;

break;

}

}

return false;

}

int main()

{

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

queue <int> q;

int\* NUM = (int\*)calloc(m.size, sizeof(int));

int v;

printf\_s("Количество вершин графа:");

scanf\_s("%d", &m.size);

NUM = (int\*)calloc(m.size, sizeof(int));

m.matrix = createMatrix(m.size);

fillMatrix(m);

outputMatrix(m);

printf("\n");

headers = matrix\_to\_list(m);

output\_alist(headers, m.size);

printf\_s("\nНачать обход с вершины:");

scanf\_s("%d", &v);

printf\_s("\nОбход:\n");

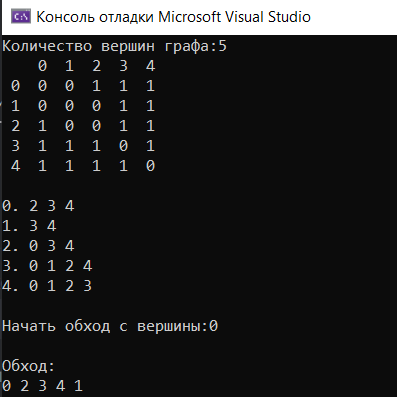
while (checkNewVertex(NUM)) {

BFS(v, m, q, NUM);

}

}

**Результаты работы программы**



**Задание 2\***

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <time.h>

#include <queue>

using namespace std;

clock\_t start, stop;

struct graph {

int\*\* matrix;

int size;

}m;

int\* NUM;

void outputMatrix(graph m) {

printf\_s(" ");

for (int i = 0; i < m.size; i++) {

printf\_s("%2d ", i);

}

printf\_s("\n");

for (int i = 0; i < m.size; i++) {

printf\_s("%2d ", i);

for (int j = 0; j < m.size; j++) {

printf\_s("%2d ", m.matrix[i][j]);

}

printf\_s("\n");

}

}

//Заполнение массива

void fillMatrix(graph m) {

int value;

for (int i = 0; i < m.size; i++) {

for (int j = i + 1; j < m.size; j++) {

value = rand() % 10;

if (value < 3)

value = 0;

else

value = 1;

m.matrix[i][j] = value;

m.matrix[j][i] = m.matrix[i][j];

}

}

}

//Создание двумерного массива

int\*\* createMatrix(int size) {

int\*\* matrix = (int\*\*)malloc(size \* sizeof(int\*));

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

matrix[i] = (int\*)calloc(size, sizeof(int));

}

return matrix;

}

typedef struct Node {

int num;

Node\* next;

Node\* back;

}Node;

typedef struct que {

Node\* qstart;

Node\* qend;

int count;

}que;

Node\* createNode() {

Node\* N = (Node\*)malloc(sizeof(Node));

N->back = NULL;

N->next = NULL;

return N;

}

que\* create\_queue() {

que\* q = (que\*)malloc(sizeof(que));

q->count = 0;

q->qstart = NULL;

q->qend = NULL;

return q;

};

void qpush(que\* q, int num) {

Node\* Element = createNode();

Element->num = num;

if (q->count == 0) {

q->qstart = Element;

q->qend = Element;

q->count++;

}

else {

if (q->count == 1) {

q->qend = Element;

q->qend->back = q->qstart;

q->qstart->next = q->qend;

q->count++;

}

else {

q->qend->next = Element;

q->qend = Element;

q->count++;

}

}

};

int qfront(que\* q) {

int val;

if (q->count == 0) {

return NULL;

}

val = q->qstart->num;

return val;

};

int qpop(que\* q) {

Node\* n;

if (q->count == 0) {

return 0;

}

if (q->count == 1) {

q->qstart = NULL;

q->qend = NULL;

q->count--;

return 1;

}

else {

n = q->qstart;

q->qstart = q->qstart->next;

q->qstart->back = NULL;

free(n);

q->count--;

return 1;

}

};

void BFS(int v, graph m, queue <int> q, int\* NUM) {

q.push(v);

NUM[v] = 1;

while (q.empty() != true) {

v = q.front();

q.pop();

printf\_s("%d ", v);

for (int i = 0; i < m.size; i++) {

if (m.matrix[v][i] == 1 && NUM[i] == 0) {

q.push(i);

NUM[i] = 1;

}

}

}

}

void BFS\_h(int v, graph m, que\* q, int\* NUM) {

qpush(q, v);

NUM[v] = 1;

while (q->count != 0) {

v = qfront(q);

qpop(q);

printf\_s("%d ", v);

for (int i = 0; i < m.size; i++) {

if (m.matrix[v][i] == 1 && NUM[i] == 0) {

qpush(q, i);

NUM[i] = 1;

}

}

}

}

bool checkNewVertex(int\* NUM) {

for (int i = 0; i < m.size; i++) {

if (NUM[m.size - i - 1] == 0) {

return true;

break;

}

}

return false;

}

int main()

{

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

queue <int> q;

que\* qh = create\_queue();

int\* NUM = (int\*)calloc(m.size, sizeof(int));

int\* NUMh = (int\*)calloc(m.size, sizeof(int));

int v;

printf\_s("Количество вершин графа:");

scanf\_s("%d", &m.size);

NUM = (int\*)calloc(m.size, sizeof(int));

NUMh = (int\*)calloc(m.size, sizeof(int));

m.matrix = createMatrix(m.size);

fillMatrix(m);

//outputMatrix(m);

printf("\n");

printf\_s("\nНачать обход с вершины:");

scanf\_s("%d", &v);

printf\_s("\nОбход (queue):\n");

start = clock();

while (checkNewVertex(NUM)) {

BFS(v, m, q, NUM);

}

stop = clock();

printf("\nВремя выполнения: %d", (stop - start));

printf\_s("\nОбход (ручная очередь):\n");

start = clock();

while (checkNewVertex(NUMh)) {

BFS\_h(v, m, qh, NUMh);

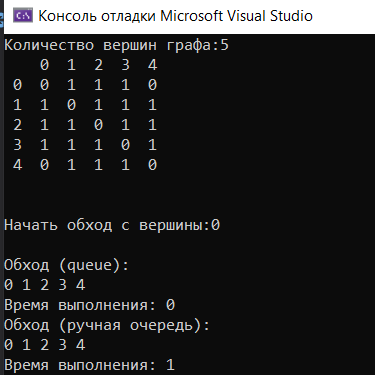
}

stop = clock();

printf("\nВремя выполнения: %d", (stop - start));

}

**Результаты работы программы**





**Выводы**В ходе выполнения лабораторной работы была разработана программа, выполняющая алгоритм обхода графа в ширину.