Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №8

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

на тему «Обход графа в ширину»

**Выполнили**

студент группы 21ВВ3

Столяров А.

**Приняли:**

Юрова О.В.

Митрохин М.А.

Пенза 2022

# Название

Обход графа в ширину

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

# 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.

# Листинг

#include<iostream>

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<queue>

#include<math.h>

#include<windows.h>

using namespace std;

queue <int> Q;

typedef struct Node {

int data = 0;

Node\* next = NULL;

};

struct Node1 {

int data = 0;

Node1\* next = NULL;

Node1\* perv = NULL;

};

Node1\* start = NULL, \* finish = NULL;

void initList(int var) {

Node1\* newCell = (Node1\*)malloc(sizeof(Node1));

newCell->data = var;

start = newCell;

finish = newCell;

}

void push(int var) {

Node1\* newCell = (Node1\*)malloc(sizeof(Node1));

newCell->data = var;

newCell->next = start;

start->perv = newCell;

start = newCell;

}

int pop() {

Node1\* sw = finish;

if (start == finish) {

start = NULL;

finish = NULL;

}

else finish = sw->perv;

int data = sw->data;

free(sw);

return data;

}

void BFS1(int\*\* G, int\* numG, int v, int total) {

initList(v);

while (start != NULL) {

v = pop();

if (numG[v] == 1) printf(" %d", v + 1);

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

if (G[v][i] == 1) {

if (numG[i] == 1) {

if (start == NULL) initList(i);

else push(i);

}

numG[v] = 0;

}

}

}

}

void BFS(int\*\* G, int\* numG, int v, int total) {

Q = {};

Q.push(v);

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

v = Q.front();

Q.pop();

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

if (G[v][i] == 1) {

if (numG[v] == 1) {

printf(" %d", v + 1);

numG[v] = 0;

}

if (numG[i] == 1) {

Q.push(i);

printf(" %d", i + 1);

numG[i] = 0;

}

}

}

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int total = 0, v = 0, count = 0;

Node\* Cell;

printf("Введите размер матрицы: ");

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

int\* numG = (int\*)malloc(total \* sizeof(int));

int\* vPerv = (int\*)malloc(total \* sizeof(int));

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

vPerv[i] = NULL;

}

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

Node\*\* lastG = (Node\*\*)malloc(total \* sizeof(Node\*));

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

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

lastG[i] = (Node\*)malloc(total \* sizeof(Node));

numG[i] = 1;

}

srand(time(NULL));

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

for (int j = i; j < total; j++) {

if (i == j) {

G[i][j] = 0;

}

else {

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

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

}

}

}

printf("\n Матрица смежности \n");

printf("\n ");

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

printf(" %c", (i + 97));

}

printf("\n ");

for (int i = 0; i < total \* 3; i++) {

printf("\_");

}

printf("\n");

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

printf(" %c | ", (i + 97));

for (int j = 0; j < total; j++) {

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

}

printf("\n");

}

printf("\n Список смежности \n");

printf("\n ");

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

Cell = NULL;

lastG[i] = NULL;

for (int j = 0; j < total; j++) {

if (G[i][j] == 1) {

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

newCell->data = j;

newCell->next = NULL;

if (Cell != NULL) Cell->next = newCell;

else lastG[i] = newCell;

Cell = newCell;

}

}

}

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

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

sw = lastG[i];

printf("%c:", i + 97);

while (sw != NULL) {

printf(" %c", sw->data + 97);

sw = sw->next;

}

free(sw);

printf("\n ");

}

printf("\nОбход по матрице (библиотека queue): ");

clock\_t t1 = clock();

while (v < total) {

if (numG[v] == 1) BFS(G, numG, v, total);

v++;

}

t1 = clock() - t1;

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

numG[i] = 1;

}

Q = {};

v = 0;

clock\_t tList = clock();

printf("\nОбход по списку: ");

while (v < total) {

Cell = lastG[v];

if (Cell != NULL && numG[v] == 1) {

printf(" %c", v + 97);

numG[v] = 0;

}

while (Cell != NULL) {

if (numG[Cell->data] == 1) {

Q.push(Cell->data);

printf(" %c", Cell->data + 97);

numG[Cell->data] = 0;

}

Cell = Cell->next;

if (Cell == NULL && Q.empty() != true) {

v = Q.front();

Q.pop();

}

else if (Cell == NULL && Q.empty() == true) v++;

}

}

tList = clock() - tList;

printf("\nОбход по матрице (своя очередь): ");

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

numG[i] = 1;

}

v = 0;

clock\_t t2 = clock();

while (v < total) {

if (numG[v] == 1) BFS1(G, numG, v, total);

v++;

}

t2 = clock() - t2;

printf("\n\nВремя обхода по матрице(queue): ");

printf("%f", ((float)t1) / CLOCKS\_PER\_SEC);

printf("\nВремя обхода по списку: ");

printf("%f", ((float)tList) / CLOCKS\_PER\_SEC);

printf("\nВремя обхода по матрице(своя очередь): ");

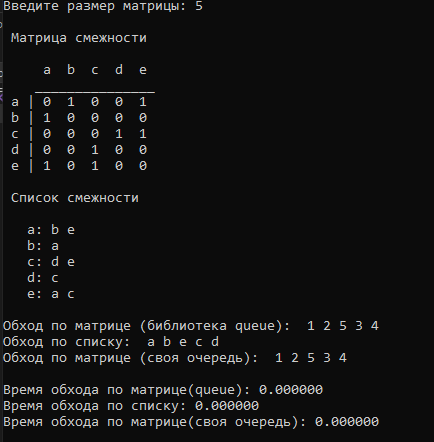
printf("%f\n", ((float)t2) / CLOCKS\_PER\_SEC);

}

# Результат работы программы

**Задание 1**

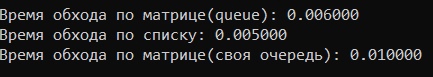
Результаты работы программы показаны на рисунке 1.



**Рисунок 1 – Результат работы программы**

**Задание 2**

Результаты времени выполнения алгоритмов работы используя 100 вершин показан 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. Самый быстрый алгоритм: обход по списку.