Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра «Вычислительная кафедра»

**ОТЧЕТ**

по лабораторной работе №8

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «**Обход графа в ширину**»

Выполнил:

студент группы 22ВВП1

Воробьева М.М.

Приняли:

Акифьев И.В.

Митрохин М.А.

Пенза 2023

**Название**

Обход графа в ширину

**Цель работы:** освоить такую операцию, как обход графов, представленных в виде матрицы смежности и списка смежности в ширину с помощью очереди.

**Лабораторной 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.

Оцените время работы двух реализаций алгоритмов обхода в ширину (использующего стандартный класс **queue** и использующего очередь, реализованную самостоятельно) для графов разных порядков.

**Листинг кода:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <cstdio>

#include <vector>

#include <ctime>

#include <cstdlib>

#include <locale.h>

#include <queue>

using namespace std;

queue<int>Q;

vector<bool> vis;

struct Node {

int inf; // Информационное поле (название объекта)

Node\* next; // Указатель на следующий элемент в очереди

};

void queue\_BFS(int s, const vector<vector<int>> G) {

clock\_t start = clock();

vector<bool> vis(G.size(), 0);

s = 1;

// Инициализация указателей на головы и хвосты как NULL

Node\* heads = NULL;

Node\* lasts = NULL;

Node\* newNode = new Node;

newNode->inf = s;

newNode->next = 0;

heads = lasts = newNode;

while (heads != 0) {

s = heads->inf;

if (!vis[s]) {

printf("->%d ", s);

vis[s] = 1;

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

if (G[s][i] == 1 and !vis[i]) {

Node\* newNode = new Node;

newNode->inf = i;

newNode->next = 0;

if (lasts == nullptr) {

heads = lasts = newNode;

}

else {

lasts->next = newNode;

lasts = newNode;

}

}

}

}

Node\* temp = heads;

heads = heads->next;

delete temp;

}

clock\_t end = clock();

printf("\n\n");

printf("Время работы алгоритма:\n");

printf("%f\n", (double)(end - start) / CLOCKS\_PER\_SEC);

}

void BFS(int s, const vector<vector<int>> G) {

clock\_t start = clock();

vector<bool> vis(G.size(), 0);

s = 1;

vis[s] = 1;

Q.push(s);

while (!Q.empty()) {

s = Q.front();

Q.pop();

printf("->%d ", s);

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

if (G[s][i] == 1 and vis[i] != 1) {

Q.push(i);

vis[i] = 1;

}

}

}

clock\_t end = clock();

printf("\n\n");

printf("Время работы алгоритма:\n");

printf("%f\n", (double)(end - start) / CLOCKS\_PER\_SEC);

printf("\n");

}

void list\_BFS(int s, const vector < vector < int>>& adjacency\_list) {

vector<bool> vis(adjacency\_list.size(), 0);

s = 1;

vis[s] = 1;

Q.push(s);

while (!Q.empty()) {

s = Q.front();

Q.pop();

printf("->%d ", s);

s + 1;

for (int neighbor : adjacency\_list[s]) { //добавляем всех непосещённых соседей.

if (!vis[neighbor]) {

Q.push(neighbor);

vis[neighbor] = 1;

}

}

}printf("\n\n");

}

void printList(const vector<vector<int>>list) {

printf("Список смежности:\n");

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

printf("%d: ", i);

for (int j : list[i]) {

printf("%d ", j);

}

printf("\n");

}

}

vector<vector<int>> matrixToList(const vector<vector<int>>& matrix) {

int num\_vertices = matrix.size() - 1;

vector<vector<int>> adjacency\_list(num\_vertices + 1);

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = 1; j <= num\_vertices; ++j) {

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

adjacency\_list[i].push\_back(j);

}

}

}

return adjacency\_list;

}

int swt = 0;

vector<vector<int>> cmej\_mat(int num\_vertices, double veroj) {

vector<vector<int>> cmej\_mat(num\_vertices + 1, vector<int>(num\_vertices + 1, 0));

int cont = 0;

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = i + 1; j <= num\_vertices; ++j) {

if (static\_cast<double>(rand()) / RAND\_MAX < veroj) {

cmej\_mat[i][j] = 1;

cmej\_mat[j][i] = 1;

cont++;

}

}

}

if (cont == 0) {

swt = 1;

}

printf("Матрица смежности:\n");

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = 1; j <= num\_vertices; ++j) {

printf("%d ", cmej\_mat[i][j]);

}

printf("\n");

}

printf("\n");

return cmej\_mat;

}

int main() {

setlocale(LC\_ALL, "RUS");

int num\_vertices;

double veroj = 0.5;

printf("Введите количество вершин: ");

scanf("%d", &num\_vertices);

srand(static\_cast<unsigned int>(time(nullptr)));

auto matrix = cmej\_mat(num\_vertices, veroj);

auto list = matrixToList(matrix);

if (swt == 1) {

printf("\nОбход произвести нельзя! Первая вершина изолированна\n");

}

else {

printf("------------------------\n");

printf("обход матрицы в ширину:\n");

printf("------------------------\n");

BFS(1, matrix);

printf("----------------------------------\n");

printf("Обход в ширину (список смежности):\n");

printf("----------------------------------\n");

list\_BFS(1, list);

printf("---------------------------------------------\n");

printf("Обход в ширину с помощью собственной очереди:\n");

printf("---------------------------------------------\n");

queue\_BFS(1, matrix);

}

return 0;

}

**Результат работы программы:**



**Вывод:** в ходе выполнения лабораторной работы были освоены навыки обхода графа в ширину, представленного в виде матрицы смежности и списка смежности, с помощью очереди, были сделаны выводы о скорости работы различных алгоритмов на различных графах.