**Министерство науки и высшего образования Российской Федерации**

**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 студ. группы 24ВВВ4:

Агуреев А.П.

Невежин И.С.

Приняли:

к.э.н. доцент Акифьев И.В.\_\_\_\_

к.т.н доцент Юрова О.В.\_\_\_\_

Пенза 2025

**Цель работы:**

Изучить и практически освоить алгоритм обхода 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.

**Результат работы программы:**

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Вывод:**

Изучили и практически освоили реализацию алгоритма обхода графа в ширину.

**Листинг:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <iostream>

#include <vector>

#include <time.h>

#include <queue>

using namespace std;

struct node\_queue {

int index;

struct node\_queue\* next;

};

struct node {

int index;

struct node\* next\_node;

};

struct Graph {

vector<node\*>vertexes;

};

struct node\_queue\* head\_queue = NULL;

struct node\_queue\* last\_queue = NULL;

void BFS\_list\_struct(Graph\* G, int start);

void BFS\_list(Graph\* G, int start);

void add\_el\_queue(int index);

void del\_queue();

void BFS\_struct(vector<vector<int>>& M, int start);

void BFS(vector<vector<int>>& M, int start);

void print\_G(Graph\* G);

void add\_el(struct Graph\* G, int vertex, int new\_index);

int main()

{

clock\_t start, end;

setlocale(LC\_ALL, "Russian");

Graph G;

int sizeM = 0, number = 0;

cout << "Введите количество вершин у графа:\n";

cin >> sizeM;

G.vertexes.resize(sizeM, nullptr);

cout << "Матрица смежности:\n";

vector <vector<int>> M(sizeM, vector<int>(sizeM, 0));

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

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

if (i == j) {

M[i][j] = 0;

}

else {

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

M[j][i] = M[i][j];

}

}

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

printf("%3d ", M[i][j]);

if (M[i][j]) {

if (i == j) {

continue;

}

add\_el(&G, i, j);

}

}

cout << '\n';

}

cout << "Список смежности:\n";

print\_G(&G);

cout << "Введите номер вершины:\n";

cin >> number;

number--;

cout << "Обход графа в ширину с использованием библиотеки очереди:\n";

cout << "Для матрицы:\n";

start = clock();

BFS(M, number);

end = clock();

double time\_spend = (double)(end - start) / CLOCKS\_PER\_SEC;

cout << "\nВремя работы: " << time\_spend << "\n";

cout << "Для списка смежности:\n";

start = clock();

BFS\_list(&G, number);

end = clock();

time\_spend = (double)(end - start) / CLOCKS\_PER\_SEC;

cout << "\nВремя работы: " << time\_spend << "\n\n";

cout << "Обход графа в ширину с использованием структуры очереди:\n";

cout << "Для матрицы:\n";

start = clock();

BFS\_struct(M, number);

end = clock();

time\_spend = (double)(end - start) / CLOCKS\_PER\_SEC;

cout << "\nВремя работы: " << time\_spend << "\n";

cout << "Для списка смежности:\n";

start = clock();

BFS\_list\_struct(&G, number);

end = clock();

time\_spend = (double)(end - start) / CLOCKS\_PER\_SEC;

cout << "\nВремя работы: " << time\_spend << "\n\n";

}

void BFS\_struct(vector<vector<int>>& M, int start) {

vector<bool> visited(M.size(), false);

add\_el\_queue(start);

visited[start] = true;

while (head\_queue != NULL) {

int current = head\_queue->index;

del\_queue();

cout << current + 1 << " ";

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

if (M[current][i] == 1 && !visited[i]) {

add\_el\_queue(i);

visited[i] = true;

}

}

}

}

void BFS(vector<vector<int>>& M, int start) {

vector<bool> visited(M.size(), false);

queue<int> q;

q.push(start);

visited[start] = true;

while (!q.empty()) {

int current = q.front();

q.pop();

cout << current + 1 << " ";

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

if (M[current][i] == 1 && !visited[i]) {

q.push(i);

visited[i] = true;

}

}

}

}

void BFS\_list\_struct(Graph\* G, int start) {

vector <bool> visited(G->vertexes.size(), false);

head\_queue = nullptr;

last\_queue = nullptr;

add\_el\_queue(start);

visited[start] = true;

while (head\_queue != nullptr) {

int current = head\_queue->index;

del\_queue();

cout << current + 1 << " ";

node\* tmp = G->vertexes[current];

while (tmp != nullptr) {

if (visited[tmp->index] == false) {

add\_el\_queue(tmp->index);

visited[tmp->index] = true;

}

tmp = tmp->next\_node;

}

}

}

void BFS\_list(Graph\* G, int start) {

vector<bool> visited(G->vertexes.size(), false);

queue<int> q;

q.push(start);

visited[start] = true;

while (!q.empty()) {

int current = q.front();

q.pop();

cout << current + 1 << " ";

node\* tmp = G->vertexes[current];

while (tmp != nullptr) {

if (!visited[tmp->index]) {

q.push(tmp->index);

visited[tmp->index] = true;

}

tmp = tmp->next\_node;

}

}

}

void add\_el(struct Graph\* G, int vertex, int new\_index) {

node\* p = (node\*)malloc(sizeof(node));

p->index = new\_index;

if (G->vertexes[vertex] == nullptr) {

G->vertexes[vertex] = p;

p->next\_node = nullptr;

}

else {

node\* tmp = G->vertexes[vertex];

while (tmp->next\_node != nullptr) {

tmp = tmp->next\_node;

}

tmp->next\_node = p;

p->next\_node = nullptr;

}

}

void print\_G(Graph\* G) {

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

node\* tmp = G->vertexes[i];

cout << "Вершина " << i + 1 << ": ";

while (tmp != nullptr) {

cout << tmp->index + 1 << " ";

tmp = tmp->next\_node;

}

cout << '\n';

}

}

/\* Последовательное добавление в список элемента (в конец) \*/

void add\_el\_queue(int index) {

struct node\_queue\* pointer = NULL;

if ((pointer = (node\_queue\*)malloc(sizeof(struct node\_queue))) == NULL) {

printf("Ошибка при распределении памяти\n");

exit(1);

}

pointer->index = index;

pointer->next = NULL;

if (head\_queue == NULL && pointer != NULL) { // если списка нет, то устанавливаем голову списка

head\_queue = pointer;

last\_queue = pointer;

}

else if (head\_queue != NULL && pointer != NULL) { // список уже есть, то вставляем в конец

last\_queue->next = pointer;

last\_queue = pointer;

}

return;

}

/\* Удаление элемента с начала очереди \*/

void del\_queue() {

if (head\_queue == NULL) {

printf("Список пуст\n");

return;

}

struct node\_queue\* struc = head\_queue;

head\_queue = head\_queue->next;

free(struc);

}