Министерство науки и высшего образования Российской федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №8

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Обход графа в ширину»

Выполнили ст. группы 22ВВВ1:

Уткин М.М.

Саветкин Д.Д.

Соколовский Е.В

Приняли:

К.э.н., доцент Акифьев И. В.

К.т.н., доцент Юрова О. В.

Пенза 2023

**Цель работы:**

Цель научиться использовать алгоритм 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.

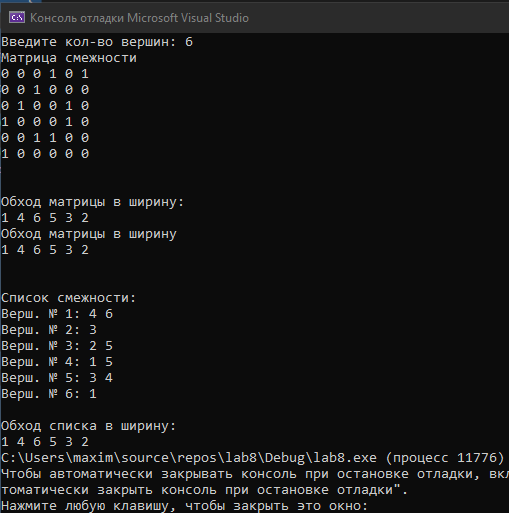
**Задание 2\***

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.

**Ход работы:**

Данный код реализует обход в ширину для графа, заданного матрицей смежности и списком смежности. Создается матрица смежности и список смежности на основе нее. Затем происходит обход в ширину для матрицы смежности и для списка смежности.

**Результаты работы программы:**



**Задание 2:**

**Оценка:**

Временная сложность алгоритма поиска в ширину равна O(V + E), где V — количество вершин (узлов), а E — количество ребер в графе. В обеих ваших реализациях вы используете для обхода графа. Выбор использования стандартной очереди или пользовательского класса очереди повлияет на постоянные факторы временной сложности, но общая сложность по-прежнему будет равна O(V + E).

Однако константы времени выполнения будут различаться из-за разной производительности стандартной очереди std::queue и очереди на связном списке.

При одинаковом количестве вершин и ребер алгоритм на std::queue будет работать быстрее, так как.

Таким образом, для повышения производительности алгоритма лучше использовать std::queue.

**Вывод:**

В ходе выполнения лабораторной работы мы освоили методы обхода графов в ширину, как для представления графа в виде матрицы смежности, так и в виде списка смежности. Мы также освоили реализацию алгоритма обхода в ширину с использованием очереди, созданной с использованием нашей собственной структуры данных "список", которую мы разработали в лабораторной работе №3. В рамках работы мы провели оценку времени выполнения двух вариантов реализации алгоритма обхода в ширину: один с использованием стандартного класса очереди, а другой с использованием нашей собственной реализации очереди. Мы изучили, как эффективно работают эти алгоритмы для графов разных размеров и структур.

**Листинг:**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <stack>

#include <vector>

#include <algorithm>

#include <queue>

using namespace std;

// Структура для представления узла связного списка

struct Node {

int value;

Node\* next;

};

// Класс для реализации очереди на связном списке

class line {

private:

Node\* head;

Node\* tip;

public:

line() {

head = nullptr;

tip = nullptr;

}

// Добавление элемента в конец очереди

void insert(int x) {

Node\* newNode = new Node({ x, nullptr });

// Если очередь не пуста, связываем с последним элементом

if (tip != nullptr) {

tip->next = newNode;

}

// Обновляем указатель на конец очереди

tip = newNode;

// Если очередь была пуста, head тоже указывает на добавленный элемент

if (head == nullptr) {

head = newNode;

}

}

// Удаление элемента из начала очереди

int del() {

if (head == nullptr) {

return -1;

}

Node\* temp = head;

int result = temp->value;

head = head->next;

if (head == nullptr) {

tip = nullptr;

}

delete temp;

return result;

}

bool isEmpty() {

return head == nullptr;

}

};

void evade(vector<vector<int>>& Matr, int n) {

vector<bool> entered(n, false);

vector<int> result;

line q;

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

if (!entered[i]) {

bool isolated = true;

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

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

isolated = false;

break;

}

}

if (!isolated) {

q.insert(i);

entered[i] = true;

while (!q.isEmpty()) {

int v = q.del();

result.push\_back(v);

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

if (Matr[v][j] == 1 && !entered[j]) {

q.insert(j);

entered[j] = true;

}

}

}

}

}

}

// вывод

cout << "\nОбход матрицы в ширину\n";

for (int v : result) {

cout << v + 1 << " ";

}

cout << endl;

}

int main() {

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

int n;

cout << "Введите кол-во вершин: ";

cin >> n;

vector<vector<int>> Matr(n, vector<int>(n));

vector<vector<int>> list(n);

// Матрица 1

cout << "Матрица смежности " << endl;

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

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

if (i == j) {

Matr[i][j] = 0;

}

else {

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

Matr[j][i] = Matr[i][j];

}

}

}

// Вывод матрицы 1

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

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

cout << Matr[i][j] << " ";

}

cout << endl;

}

cout << "\n";

vector<bool> entered(n, false);

cout << "\nОбход матрицы в ширину:\n";

//обход матрицы в ширину

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

if (!entered[i]) {

bool isolated = true;

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

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

isolated = false;

break;

}

}

if (!isolated) {

queue<int> q;

q.push(i);

entered[i] = true;

while (!q.empty()) {

int v = q.front();

q.pop();

cout << v + 1 << " ";

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

if (Matr[v][j] == 1 && !entered[j]) {

q.push(j);

entered[j] = true;

}

}

}

}

}

}

//обход матрицы в ширину на основе списка из ЛР№3

evade(Matr, n);

// Заполнение списка смежности на основе матрицы

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

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

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

list[i].push\_back(j);

//list[j].push\_back(i); // добавляем обратное ребро

}

}

}

// Вывод списка смежности

cout << "\n";

cout << "\nСписок смежности:\n";

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

cout << "Верш. № " << i + 1 << ": ";

for (int j = 0; j < list[i].size(); ++j) {

cout << list[i][j] + 1 << " ";

}

cout << "\n";

}

cout << "\nОбход списка в ширину:\n";

vector<bool> entered2(n, false);

//обход матрицы в ширину списка смежности

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

if (!entered2[i]) {

bool isolated = list[i].empty();

if (!isolated) {

queue<int> q;

q.push(i);

entered2[i] = true;

while (!q.empty()) {

int v = q.front();

q.pop();

cout << v + 1 << " ";

for (int j = 0; j < list[v].size(); j++) {

int u = list[v][j];

if (!entered2[u]) {

q.push(u);

entered2[u] = true;

}

}

}

}

}

}

return 0;

}