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:

студент группы 22ВВВ1

Матросов А.М.

Пяткин М.Э.

Принял:

к.т.н. доцент Юрова О.В.

к.э.н. доцент Акифьев И.В.

Пенза 2023

**Общие сведения**

Обход графа в ширину – еще один распространенный способ обхода графов.

Основная идея такого обхода состоит в том, чтобы посещать вершины по уровням удаленности от исходной вершины. Удалённость в данном случае понимается как количество ребер, по которым необходимо прейти до достижения вершины. Например, если для графа на рисунке 1 начать обход из первой вершины, то вершины 3, 6 и 2 будут находиться на уровне удаленности в 1 ребро, а вершины 5 и 4 на уровне удаленности в 2 ребра.

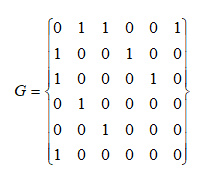
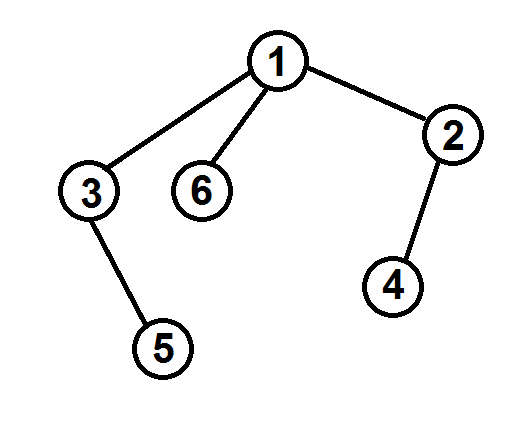


Рисунок 1 – Граф

Тогда при обходе этого графа в ширину, мы сначала посетим вершины первого уровня удаленности (с номерами 2, 3 и 6), и только после того, как закончатся не посещенные вершины на этом уровне, мы перейдем к следующему. На втором уровне мы посетим все вершины, которые удалены от исходной на 2 ребра (вершины 4 и 5).

Так, алгоритм обхода в ширину продолжает осматривать уровень за уровнем, пока не пройдет все доступные вершины.

Чтобы не заходить повторно в уже пройденные вершины, они помечаются, как и в алгоритме обхода в глубину.

Для того, чтобы проход осуществлялся по уровням необходимо хранить информацию о требуемом порядке посещения вершин. Вершины, которые являются ближайшими соседями исходной вершины (из которой начат обход) должны быть посещены раньше, чем соседи соседей и т.д. Такой порядок позволяет задать структура данных «очередь». Просматривая строку матрицы смежности (или список смежности) для текущей вершины мы помещаем всех её ещё не посещенных соседей в очередь. На следующей итерации текущей вершиной становится та, которая стоит в очереди первой и уже её не посещенные соседи будут помещены в очередь. Но место в очереди они займут после тех вершин, которые были помещены туда на предыдущих итерациях.

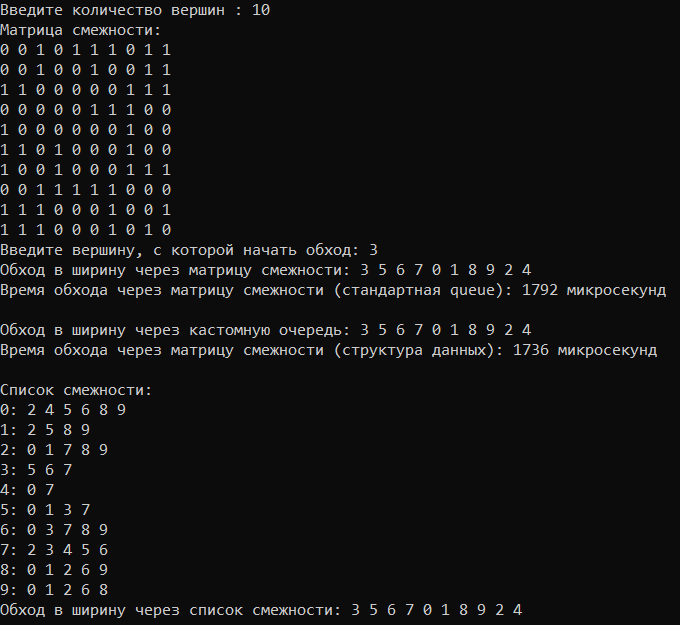
**Задание 1**

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.



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

|  |  |  |
| --- | --- | --- |
| Порядок графа | Время(список) | Время(класс queue) |
| 50 | 0,002 | 0,001 |
| 100 | 0,015 | 0,003 |
| 200 | 0,268 | 0,009 |
| 400 | 4,602 | 0,029 |
| 500 | 11,668 | 0,035 |
| 600 | 23,847 | 0,052 |
| 700 | 63,902 | 0,068 |

### Выводы

В ходе выполнения лабораторной работы была реализована программа, генерирующая матрицу смежности для неориентированного графа, выполняющая обхода графа в ширину, с использованием очереди из стандартной библиотеки c++ и очереди из лабораторной работы №3.

Оценили время работы двух реализаций алгоритмов обхода в ширину для графов разных порядков:

Алгоритм, использующий список из лабораторной работы №3, намного дольше обходит графы больших порядков, но, если порядок графа <100, время работы сравнимо с алгоритмом с использованием очереди из стандартной библиотеки c++.

Алгоритм, использующий очередь из стандартной библиотеки, очень быстро обходит графы больших порядков.

### Листинг

#include <iostream>

#include <vector>

#include <queue>

#include <cstdlib>

#include <ctime>

#include <chrono>

using namespace std;

using namespace std::chrono;

struct Queue {

int front, rear, size;

unsigned capacity;

int\* array;

};

Queue\* createQueue(unsigned capacity) {

Queue\* queue = new Queue();

queue->capacity = capacity;

queue->front = queue->size = 0;

queue->rear = capacity - 1;

queue->array = new int[(queue->capacity \* sizeof(int))];

return queue;

}

int isFull(Queue\* queue) {

return (queue->size == queue->capacity);

}

int isEmpty(Queue\* queue) {

return (queue->size == 0);

}

void enqueue(Queue\* queue, int item) {

if (isFull(queue)) return;

queue->rear = (queue->rear + 1) % queue->capacity;

queue->array[queue->rear] = item;

queue->size = queue->size + 1;

}

int dequeue(Queue\* queue) {

if (isEmpty(queue)) return -1;

int item = queue->array[queue->front];

queue->front = (queue->front + 1) % queue->capacity;

queue->size = queue->size - 1;

return item;

}

vector<vector<int>> generateRandomGraph(int n) {

vector<vector<int>> graph(n, vector<int>(n, 0));

srand(time(0));

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

for (int j = i + 1; j < n; j++) {

int random = rand() % 2;

graph[i][j] = random;

graph[j][i] = random;

}

}

return graph;

}

void printMatrix(vector<vector<int>>& matrix) {

cout << "Матрица смежности:" << endl;

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

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

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

}

cout << endl;

}

}

void BFSUsingMatrix(vector<vector<int>>& graph, vector<bool>& visited, int startVertex) {

queue<int> q;

q.push(startVertex);

visited[startVertex] = true;

while (!q.empty()) {

int currentVertex = q.front();

q.pop();

cout << currentVertex << " ";

for (int i = 0; i < graph[currentVertex].size(); i++) {

if (graph[currentVertex][i] == 1 && !visited[i]) {

q.push(i);

visited[i] = true;

}

}

}

}

void BFSUsingMatrixWithTime(vector<vector<int>>& graph, vector<bool>& visited, int startVertex) {

auto start = high\_resolution\_clock::now();

BFSUsingMatrix(graph, visited, startVertex);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

cout << "\nВремя обхода через матрицу смежности (стандартная queue): " << duration.count() << " микросекунд" << endl;

}

void BFSUsingCustomQueue(vector<vector<int>>& graph, vector<bool>& visited, int startVertex) {

int n = graph.size();

Queue\* q = createQueue(n);

enqueue(q, startVertex);

visited[startVertex] = true;

while (!isEmpty(q)) {

int currentVertex = dequeue(q);

cout << currentVertex << " ";

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

if (graph[currentVertex][i] == 1 && !visited[i]) {

enqueue(q, i);

visited[i] = true;

}

}

}

}

void BFSUsingCustomQueueWithTime(vector<vector<int>>& graph, vector<bool>& visited, int startVertex) {

auto start = high\_resolution\_clock::now();

BFSUsingCustomQueue(graph, visited, startVertex);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

cout << "\nВремя обхода через матрицу смежности (структура данных): " << duration.count() << " микросекунд" << endl;

}

void printAdjacencyList(vector<vector<int>>& graph) {

cout << "Список смежности:" << endl;

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

cout << i << ": ";

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

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

}

cout << endl;

}

}

void BFSUsingLists(vector<vector<int>>& graph, vector<bool>& visited, int startVertex) {

queue<int> q;

q.push(startVertex);

visited[startVertex] = true;

while (!q.empty()) {

int currentVertex = q.front();

q.pop();

cout << currentVertex << " ";

for (int i = 0; i < graph[currentVertex].size(); i++) {

int adjacentVertex = graph[currentVertex][i];

if (!visited[adjacentVertex]) {

q.push(adjacentVertex);

visited[adjacentVertex] = true;

}

}

}

}

int main() {

setlocale(LC\_ALL, "Russian");

int n;

cout << "Введите количество вершин : ";

cin >> n;

vector<vector<int>> adjacencyMatrix = generateRandomGraph(n);

printMatrix(adjacencyMatrix);

vector<bool> visited(n, false);

int startVertex;

cout << "Введите вершину, с которой начать обход: ";

cin >> startVertex;

if (startVertex < 0 || startVertex >= n) {

cout << "Некорректная вершина. Пожалуйста, введите вершину от 0 до " << n - 1 << "." << endl;

return 0;

}

cout << "Обход в ширину через матрицу смежности: ";

BFSUsingMatrixWithTime(adjacencyMatrix, visited, startVertex);

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

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

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

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

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

}

}

}

visited.assign(n, false);

cout << "\nОбход в ширину через кастомную очередь: ";

BFSUsingCustomQueueWithTime(adjacencyMatrix, visited, startVertex);

cout << endl;

printAdjacencyList(adjacencyList);

visited.assign(n, false);

cout << "Обход в ширину через список смежности: ";

BFSUsingLists(adjacencyList, visited, startVertex);

cout << endl;

return 0;

}