МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
Кафедра «Вычислительной техники»

**ОТЧЁТ**

по лабораторной работе №8  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Обход графа в ширину»

Выполнил:

Студент группы 23ВВВ2

Пичаев И. А.

Чупраков С. В.

Приняли:

Митрохин М. А.  
Юрова О.В.

Пенза 2024

**Цель работы**

Приобрести навыки программирования и работы алгоритмов обхода графа в ширину различными способами.

**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**

### **Задание 1.1-3**

Матрица смежности генерируется пи помощи функции rand(). Затем вызывается функция обхода в ширину. Обход в ширину (BFS) в данной программе выполняется в функции BFS следующим образом:

Инициализация:Создается массив visited для отслеживания посещённых вершин. Все элементы массива инициализируются значением false (вершины не посещены).Создается очередь queue для хранения вершин, которые необходимо обработать. Для очереди используется массив queue, а индексы front и rear обозначают начало и конец очереди соответственно.

Начальная вершина (в данном случае с индексом start, равным 0) помечается как посещённая (visited[start] = true) и добавляется в очередь (queue[rear++] = start).

Основной цикл обхода:

В цикле while (front < rear) извлекаются вершины из очереди по очереди для их обработки.

Для текущей вершины, извлечённой из очереди (currentVertex = queue[front++]), выводится её номер (с учётом индексации с 1).

Далее для текущей вершины просматриваются все её соседние вершины, представленные в списке смежности. Для этого используется указатель temp, который перебирает все соседние вершины текущей вершины в списке смежности.

Обработка соседей:

Для каждого соседа вершины, если он ещё не посещён (!visited[adjVertex]), он помечается как посещённый (visited[adjVertex] = true) и добавляется в очередь для дальнейшей обработки (queue[rear++] = adjVertex).

Завершение обхода:

Процесс продолжается до тех пор, пока в очереди не останется вершин (пока front < rear), т.е. все достижимые вершины будут обработаны.

**Задание 2.1-2**

Генерируется граф сразу записывается в виде списка смежности. Первый вариант использует собственную очередь, реализованную через структуру node и связанные элементы.

Второй вариант использует стандартную очередь из библиотеки queue C++.

Процесс обхода в ширину:

В обоих случаях обход начинается с вершины, переданной в качестве стартовой.

Для каждого алгоритма создается очередь, в которую добавляются непосещенные вершины, и выполняется их посещение поочередно.

В процессе обхода проверяется, есть ли ребра между вершинами, и если да, то вершины добавляются в очередь для дальнейшего посещения.

### **Результаты работы программы**

### 

Рисунок 1 — Результаты работы программы 1.2

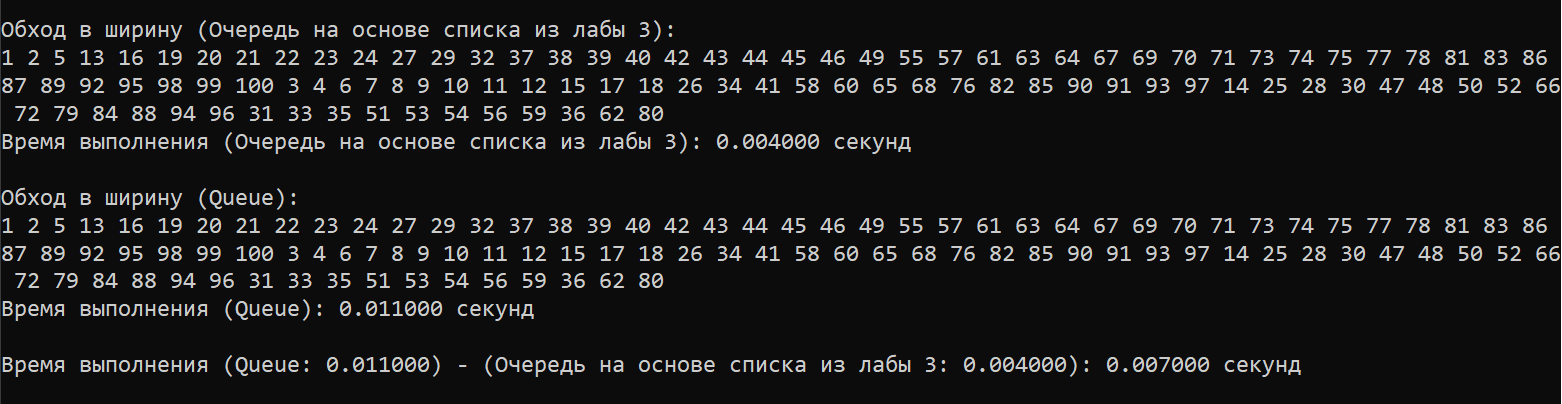


Рисунок 2 — Результаты работы программы 2

По результатам работы программы, обход в ширину с помощью очереди оказался более времязатратным.

**Вывод**

Приобретены навыки программирования и работы алгоритмов обхода графа в ширину различными способами.

### **Задание 1.1-3**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <vector>

#include <queue>

#include <ctime>

#include <cstdlib>

#include <windows.h>

#define MAX\_VERTICES 100

struct node {

int vertex;

struct node\* next;

};

struct Graph {

int numVertices;

struct node\*\* adjLists;

};

struct node\* createNode(int v) {

struct node\* newNode = (node\*)malloc(sizeof(struct node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createAGraph(int vertices) {

struct Graph\* graph = (Graph\*)malloc(sizeof(struct Graph));

graph->numVertices = vertices;

graph->adjLists = (node\*\*)malloc(vertices \* sizeof(struct node\*));

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

graph->adjLists[i] = NULL;

}

return graph;

}

void addEdge(struct Graph\* graph, int s, int d) {

struct node\* temp = graph->adjLists[s];

while (temp) {

if (temp->vertex == d) {

return;

}

temp = temp->next;

}

struct node\* newNode = createNode(d);

newNode->next = graph->adjLists[s];

graph->adjLists[s] = newNode;

temp = graph->adjLists[d];

while (temp) {

if (temp->vertex == s) {

return;

}

temp = temp->next;

}

newNode = createNode(s);

newNode->next = graph->adjLists[d];

graph->adjLists[d] = newNode;

}

struct Graph\* edges(int\*\* G, int size) {

struct Graph\* graph = createAGraph(size);

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

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

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

addEdge(graph, i, j);

}

}

}

return graph;

}

void printGraph(struct Graph\* graph) {

for (int v = 0; v < graph->numVertices; v++) {

struct node\* temp = graph->adjLists[v];

printf("\nVertex %d: ", v + 1);

while (temp) {

printf("%d -> ", temp->vertex + 1);

temp = temp->next;

}

printf("\n");

}

printf("\n");

}

int\*\* createG(int size) {

int\*\* G = NULL;

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

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

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

}

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

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

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

if (i == j) G[i][j] = 0;

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

}

}

return G;

}

void printG(int\*\* G, int size) {

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

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

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

}

printf("\n");

}

printf("\n");

return;

}

void BFS(struct Graph\* graph, int start) {

bool visited[MAX\_VERTICES] = { false };

int queue[MAX\_VERTICES];

int front = 0, rear = 0;

visited[start] = true;

queue[rear++] = start;

printf("Результатобхода в ширину: : ", start + 1);

while (front < rear) {

int currentVertex = queue[front++];

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

struct node\* temp = graph->adjLists[currentVertex];

while (temp) {

int adjVertex = temp->vertex;

if (!visited[adjVertex]) {

visited[adjVertex] = true;

queue[rear++] = adjVertex;

}

temp = temp->next;

}

}

printf("\n");

}

int main(void) {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int sizeG1 = 0;

int\*\* G1 = NULL;

printf("Введите количество вершин графа: ");

scanf("%d", &sizeG1);

G1 = createG(sizeG1);

printf("1 graf\n");

printG(G1, sizeG1);

struct Graph\* graph1 = edges(G1, sizeG1);

printGraph(graph1);

BFS(graph1, 0);

}

Задание 2.1-2

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <iostream>

#include <vector>

#include <queue>

#include <time.h>

#include <cstdlib>

#include <windows.h>

using namespace std;

struct node {

int inf;

struct node\* next;

};

struct node\* head = NULL, \* last = NULL;

void BFSqueue(vector < vector < int >> G, int start) {

int n = G.size();

vector<bool> visited(n, false);

queue<int> Q;

Q.push(start);

visited[start] = true;

while (!Q.empty()) {

int v = Q.front();

Q.pop();

cout << v + 1 << " ";

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

if (G[v][i] == 1 && !visited[i]) {

Q.push(i);

visited[i] = true;

}

}

}

}

void enqueue(int value) {

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

if (p == NULL) {

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

exit(1);

}

p->inf = value;

p->next = NULL;

if (last == NULL) {

head = p;

last = p;

}

else {

last->next = p;

last = p;

}

}

int isQueueEmpty(void) {

return head == NULL;

}

int dequeue(void) {

if (isQueueEmpty()) {

printf("Очередь пуста\n");

return -1;

}

struct node\* temp = head;

int value = temp->inf;

head = head->next;

if (head == NULL) {

last = NULL;

}

free(temp);

return value;

}

void BFS(int\*\* G, int n, int start) {

int\* visited = (int\*)malloc(n \* sizeof(int));

for (int i = 0; i < n; i++) visited[i] = 0;

enqueue(start);

visited[start] = 1;

while (!isQueueEmpty()) {

int v = dequeue();

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

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

if (G[v][i] == 1 && !visited[i]) {

enqueue(i);

visited[i] = 1;

}

}

}

free(visited);

}

int\*\* generateG(int versh) {

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

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

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

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

G[i][j] = 0;

}

}

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

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

if (rand() % 2 == 1) {

G[i][j] = 1;

G[j][i] = 1;

}

}

}

return G;

}

void printG(int\*\* G, int versh) {

printf("Матрица смежности: \n");

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

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

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

}

printf("\n");

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(5435234623462343726);

int versh;

printf("Введите количество вершин графа: ");

scanf("%d", &versh);

int\*\* G = NULL;

G = generateG(versh);

printG(G, versh);

printf("\nОбход в ширину (Очередь на основе списка из лабы 3):\n");

clock\_t start, end;

start = clock();

BFS(G, versh, 0);

end = clock();

float bfs = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\nВремя выполнения (Очередь на основе списка из лабы 3): %f секунд\n", bfs);

vector < vector < int >> G1(versh, vector<int>(versh, 0));

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

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

G1[i][j] = G[i][j];

}

}

cout<< endl << "Обход в ширину (Queue):" << endl;

start = clock();

BFSqueue(G1, 0);

end = clock();

float bfsQ = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\nВремя выполнения (Queue): %f секунд\n", bfsQ);

printf("\nВремя выполнения (Queue: %f) - (Очередь на основе списка из лабы 3: %f): %f секунд\n", bfsQ, bfs, bfsQ - bfs);

}