Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №8

по теме «Обход графа в ширину»

Выполнили:

Студенты группы 21ВВ2

Митрошин А.Д.

Сагателов А.К.

Принял:

Митрохин М.А.

Юрова О.В.

Пенза, 2022

**Цель работы:**

Изучить основные алгоритмы обхода графа и научиться решать задачи обхода графа на основе поиска в ширину.

**Лабораторные работы:**

**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.

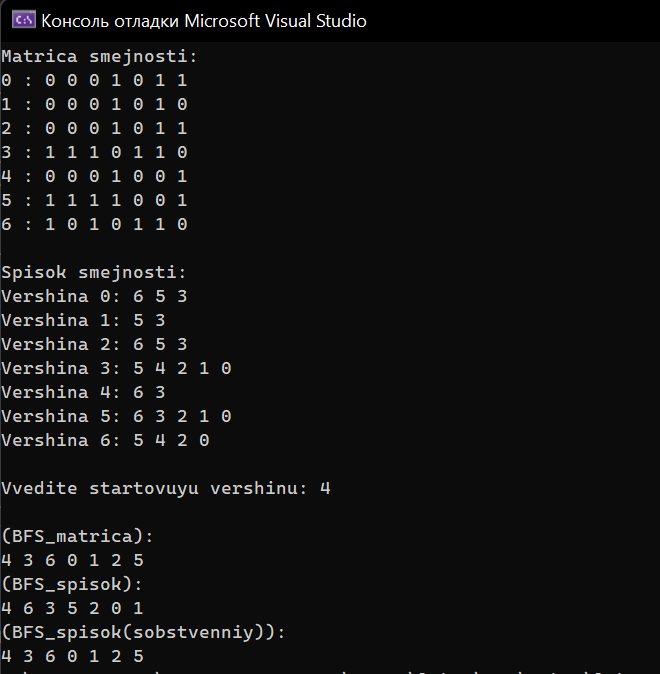
**Тестирование программы:**

|  |  |
| --- | --- |
| Количество вершин | Реализация |
| |  |  |  | | --- | --- | --- | | Матрица смежности  queue | Список смежности | Матрица смежности  MyList | |
| 100 | |  |  |  | | --- | --- | --- | | 0.013 | 0.012 | 0.011 | |
| 200 | |  |  |  | | --- | --- | --- | | 0.026 | 0.026 | 0.024 | |
| 400 | |  |  |  | | --- | --- | --- | | 0.076 | 0.067 | 0.063 | |
| 1 000 | |  |  |  | | --- | --- | --- | | 0.214 | 0.214 | 0.17 | |

Вывод: По приведенным выше замеров можно огласить результат: самый быстрой по реализация функцией оказалась BFS(Матрица смежности MyList), самой медленной стала функция BFS\_M(Матрица смежности

queue)

**Результат работы программы:**



**Листинг:**

#include <iostream>

#include <list>

#include <stack>

#include <queue>

using namespace std;

int\*\* Matrix;

bool\* visited;

const int numVertices = 7;

//реализация списка(двухсвязный)

typedef struct \_Node {

int value;

struct \_Node\* next; //хранит адрес следующего узла

struct \_Node\* prev; //хранит адрес предыдущего узла

} Node;

// структура элемента

typedef struct \_DblLinkedList {

size\_t size;

Node\* head;

Node\* tail;

//В случае, когда в списке нет элементов, оба они равны нулю

} DblLinkedList;

//оздаёт экземпляр структуры DblLinkedList

DblLinkedList\* createDblLinkedList() {

DblLinkedList\* tmp = (DblLinkedList\*)malloc(sizeof(DblLinkedList));

tmp->size = 0;

tmp->head = tmp->tail = NULL;

return tmp;

}

//добавление в конец

void pushBack(DblLinkedList\* list, int value) {

Node\* tmp = (Node\*)malloc(sizeof(Node));

if (tmp == NULL) {

exit(3);

}

tmp->value = value;

tmp->next = NULL;

tmp->prev = list->tail;

if (list->tail) {

list->tail->next = tmp;

}

list->tail = tmp;

if (list->head == NULL) {

list->head = tmp;

}

list->size++;

}

//удаление первого элемента

void popFront(DblLinkedList\* list) {

Node\* prev;

int tmp;

if (list->head == NULL) {

exit(2);

}

prev = list->head;

list->head = list->head->next;

if (list->head) {

list->head->prev = NULL;

}

if (prev == list->tail) {

list->tail = NULL;

}

tmp = prev->value;

free(prev);

list->size--;

}

////////////////////////////////////////////////////////

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\* createGraph(int vertices)

{

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

graph->numVertices = vertices;

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

int i;

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

graph->adjLists[i] = NULL;

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest)

{

// добавление вершины в список элемента

struct node\* newNode = createNode(dest);

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

graph->adjLists[src] = newNode;

}

void printGraph(struct Graph\* graph)

{

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

{

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

printf("\nVershina %d: ", v);

while (temp)

{

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

temp = temp->next;

}

}

}

void toString(int\*\* Matrix, int n) {

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

cout << i << " : ";

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

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

}

cout << endl;

}

}

void BFS\_M(int start)

{

queue<int> q;

q.push(start);

visited[start] = true;

while (!q.empty()) {

start = q.front();

q.pop();

printf\_s("%d ", start);

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

if (Matrix[start][i] == 1 && (!visited[i])) {

q.push(i);

visited[i] = true;

}

}

}

}

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

{

queue<int> q;

visited[start] = 1;

q.push(start);

while (!q.empty()) {

int currentVertex = q.front();

q.pop();

printf\_s("%d ", currentVertex);

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

while (temp != NULL) {

int adjVertex = temp->vertex;

if (!visited[adjVertex]) {

visited[adjVertex] = 1;

q.push(adjVertex);

}

temp = temp->next;

}

}

}

void BFS(int start)

{

DblLinkedList\* q = createDblLinkedList();

pushBack(q, start);

visited[start] = true;

while (q->head != NULL) {

start = q->head->value;

printf\_s("%d ", start);

popFront(q);

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

if (Matrix[start][i] == 1 && (!visited[i])) {

pushBack(q, i);

visited[i] = true;

}

}

}

}

int main()

{

setlocale(0, "");

srand(time(0));

struct Graph\* graph = createGraph(numVertices);

Matrix = (int\*\*)malloc(sizeof(int\*) \* numVertices);

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

Matrix[i] = (int\*)malloc(sizeof(int\*) \* numVertices);

for (int j = 0; j < numVertices; j++)

{

Matrix[i][j] = 0; // создание матрицы смежности

}

}

for (int i = 0; i < numVertices; i++)

for (int j = i + 1; j < numVertices; j++)

if (0 < rand() % 2)

Matrix[i][j] = Matrix[j][i] = 1, addEdge(graph, i, j), addEdge(graph, j, i); // связи вершин графа

visited = (bool\*)malloc(numVertices \* sizeof(bool));

printf\_s("Matrica smejnosti: \n");

toString(Matrix, numVertices);

printf\_s("\nSpisok smejnosti: ");

printGraph(graph);

int start;

printf\_s("\n\nVvedite startovuyu vershinu: "); scanf\_s("%d", &start);

for (int i = 0; i < numVertices; i++)

visited[i] = 0;

printf\_s("\n(BFS\_matrica): \n");

BFS\_M(start);

for (int i = 0; i < numVertices; i++)

visited[i] = 0;

printf\_s("\n(BFS\_spisok): \n");

BFS\_S(graph, start);

for (int i = 0; i < numVertices; i++)

visited[i] = 0;

printf\_s("\n(BFS\_spisok(sobstvenniy)): \n");

BFS(start);

for (int i = 0; i < numVertices; i++)

free(Matrix[i]);

free(Matrix);

free(visited);

return 0;

}

**Вывод:** Изучили основные алгоритмы обхода графа и научились решать задачи обхода графа на основе поиска в ширину.