Министерство науки и высшего образования РФ
Пензенский государственный университет
Кафедра «Вычислительная техника»

ОТЧЕТ по
лабораторной работе №8
по дисциплине «логика и основы алгоритмизации в инженерных задачах»
на тему «Обход графа в ширину»

Выполнили:
студенты группы 24ВВВ4
Королёв Д.В
Алешин К.А

Приняли:
Юрова О.В
Деев М.В

Пенза 2024

Цель работы – научиться на практике реализации алгоритма обхода графа в ширину

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.

Листинг

Задание 1

1-2)

```
#include <iostream>

#include <queue>

#include <cstdlib>

#include <ctime>


#define MAX_VERTICES 100


void generate_adjacency_matrix(int n, int
matrix[MAX_VERTICES][MAX_VERTICES]) {

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

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

            matrix[i][j] = 0;

        }

    }


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

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

            int edge = rand() % 2;
```

```
            matrix[i][j] = edge;
            matrix[j][i] = edge;
        }
    }
}

void bfs(int matrix[MAX_VERTICES][MAX_VERTICES], int n, int start_vertex) {
    bool visited[MAX_VERTICES] = {false};
    std::queue<int> q;

    visited[start_vertex] = true;
    q.push(start_vertex);

    std::cout << "BFS обход начиная с вершины " << start_vertex << ": ";

    while (!q.empty()) {
        int vertex = q.front();
        q.pop();
        std::cout << vertex << " ";

        for (int i = 0; i < n; i++) {
            if (matrix[vertex][i] == 1 && !visited[i]) {
                visited[i] = true;
                q.push(i);
```

```
            }
        }
    }
    std::cout << std::endl;
}

int main() {
    int n;

    while(1) {
        std::cout << "Введите размер матрицы: ";
        std::cin >> n;

        if (n <= 0) {
            break;
        }

        int matrix[MAX_VERTICES][MAX_VERTICES];

        srand((time(NULL)));

        generate_adjacency_matrix(n, matrix);

        std::cout << "Adjacency Matrix:\n";
        for (int i = 0; i < n; i++) {
```

```
            for (int j = 0; j < n; j++) {
                std::cout << matrix[i][j] << " ";
            }
            std::cout << "\n";
        }

        int start_vertex;
        std::cout << "Введите стартовую вершину для BFS (0 до " << n-1 << "): ";
        std::cin >> start_vertex;
        bfs(matrix, n, start_vertex);
        std::cout << "\n\n";
    }
    return 0;
}
```

3)

```
#include <iostream>
#include <vector>
#include <queue>

struct Node {
    int vertex;
    Node* next;
};
```

```
struct Graph {
    int numVertices;
    Node** adjLists;
};

Node* createNode(int vertex) {
    Node* newNode = new Node();
    newNode->vertex = vertex;
    newNode->next = nullptr;
    return newNode;
}

Graph* createGraph(int vertices) {
    Graph* graph = new Graph();
    graph->numVertices = vertices;



    graph->adjLists = new Node*[vertices];

    for (int i = 0; i < vertices; i++) {
        graph->adjLists[i] = nullptr;
    }
```

```
    return graph;
}

void addEdge(Graph* graph, int src, int dest) {
    Node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

// обход в ширину
void bfs(Graph* graph, int startVertex) {
    std::vector<bool> visited(graph->numVertices, false);
    std::queue<int> queue;

    visited[startVertex] = true;
    queue.push(startVertex);

    while (!queue.empty()) {
        int currentVertex = queue.front();
```

```
            queue.pop();
            std::cout << currentVertex << " ";

            Node* adjList = graph->adjLists[currentVertex];
            while (adjList != nullptr) {
                int connectedVertex = adjList->vertex;
                if (!visited[connectedVertex]) {
                    visited[connectedVertex] = true;
                    queue.push(connectedVertex);
                }
                adjList = adjList->next;
            }
        }
    }

    int main() {
        int vertices = 10; // количество вершин
        Graph* graph = createGraph(vertices);

        addEdge(graph, 0, 1);
        addEdge(graph, 0, 4);
        addEdge(graph, 1, 2); // ребро между 1 и 2
        addEdge(graph, 1, 3);
```

```
addEdge(graph, 1, 4);
addEdge(graph, 2, 3);
addEdge(graph, 3, 4);
addEdge(graph, 4, 5);
addEdge(graph, 5, 7);
addEdge(graph, 2, 8);

std::cout << "обход в ширину начиная с вершины 0:\n";
bfs(graph, 0);
std::cout << "\n";

// Освобождение памяти
for (int i = 0; i < vertices; i++) {
    Node* adjList = graph->adjLists[i];
    while (adjList != nullptr) {
        Node* temp = adjList;
        adjList = adjList->next;
        delete temp;
    }
}
delete[] graph->adjLists;
delete graph;
```

```
    return 0;
}
```

Задание 2

1)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX_VERTICES 100

typedef struct {
    int items[MAX_VERTICES];
    int head, tail;
} Queue;

void initQueue(Queue* q) {
    q->head = -1;
    q->tail = -1;
}

int isEmpty(Queue* q) {
    return q->head == -1;
}
```

```
void push(Queue* q, int value) {
    if (q->tail == MAX_VERTICES - 1) {
        printf("Очередь переполнена!\n");
        return;
    }
    if (q->head == -1) {
        q->head = 0;
    }
    q->tail++;
    q->items[q->tail] = value;
}

int pop(Queue* q) {
    if (isEmpty(q)) {
        printf("Очередь пуста!\n");
        return -1;
    }
    int item = q->items[q->head];
    if (q->head >= q->tail) {
        q->head = q->tail = -1; // Сброс очереди
    } else {
        q->head++;
    }
    return item;
}
```

```
void generate_adjacency_matrix(int n, int
matrix[MAX_VERTICES][MAX_VERTICES]) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            matrix[i][j] = 0;
        }
    }

    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            int edge = rand() % 2;
            matrix[i][j] = edge;
            matrix[j][i] = edge;
        }
    }
}

void bfs(int matrix[MAX_VERTICES][MAX_VERTICES], int n, int
start_vertex) {
    int visited[MAX_VERTICES] = {0};
    Queue q;
    initQueue(&q);

    visited[start_vertex] = 1;
    push(&q, start_vertex);
```

```
    printf("обход в ширину начиная с вершины %d: ",
start_vertex);

    while (!isEmpty(&q)) {
        int vertex = pop(&q);
        printf("%d ", vertex);

        for (int i = 0; i < n; i++) {
            if (matrix[vertex][i] == 1 && !visited[i]) {
                visited[i] = 1;
                push(&q, i);
            }
        }
    }
    printf("\n");
}

int main() {
    int n;

    while(1) {
        printf("Введите размер матрицы: ");
        scanf("%d", &n);

        if (n <= 0) {
```

```
            break;
        }

        int matrix[MAX_VERTICES][MAX_VERTICES];

        srand(time(NULL));

        generate_adjacency_matrix(n, matrix);

        printf("Adjacency Matrix:\n");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                printf("%d ", matrix[i][j]);
            }
            printf("\n");
        }

        int start_vertex;
        printf("Введите стартовую вершину для BFS (0 до %d):
", n-1);
        scanf("%d", &start_vertex);
        bfs(matrix, n, start_vertex);
        printf("\n\n");
    }
    return 0;
}
```

2)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>

#define MAX_VERTICES 1000

typedef struct {
    int items[MAX_VERTICES];
    int head, tail;
} Queue;

void initQueue(Queue* q) {
    q->head = -1;
    q->tail = -1;
}

int isEmpty(Queue* q) {
    return q->head == -1;
}

void push(Queue* q, int value) {
    if (q->tail == MAX_VERTICES - 1) {
        printf("Очередь переполнена!\n");
```

```
        return;
    }
    if (q->head == -1) {
        q->head = 0;
    }
    q->tail++;
    q->items[q->tail] = value;
}

int pop(Queue* q) {
    if (isEmpty(q)) {
        printf("Очередь пуста!\n");
        return -1;
    }
    int item = q->items[q->head];
    if (q->head >= q->tail) {
        q->head = q->tail = -1; // Сброс очереди
    } else {
        q->head++;
    }
    return item;
}

void generate_adjacency_matrix(int n, int
matrix[MAX_VERTICES][MAX_VERTICES]) {
    for (int i = 0; i < n; i++) {
```

```
        for (int j = 0; j < n; j++) {
            matrix[i][j] = 0;
        }
    }

    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            int edge = rand() % 2;
            matrix[i][j] = edge;
            matrix[j][i] = edge;
        }
    }
}

void bfs_standard_queue(int
matrix[MAX_VERTICES][MAX_VERTICES], int n, int start_vertex)
{
    bool visited[MAX_VERTICES] = {false};
    int queue[MAX_VERTICES];
    int head = 0, tail = 0;

    visited[start_vertex] = true;
    queue[tail++] = start_vertex;

    //printf("BFS обход начиная с вершины %d: ",
start_vertex);
```

```
    while (head < tail) {
        int vertex = queue[head++];
        //printf("%d ", vertex);

        for (int i = 0; i < n; i++) {
            if (matrix[vertex][i] == 1 && !visited[i]) {
                visited[i] = true;
                queue[tail++] = i;
            }
        }
    }
    printf("\n");
}


void bfs_custom_queue(int
matrix[MAX_VERTICES][MAX_VERTICES], int n, int start_vertex)
{
    int visited[MAX_VERTICES] = {0};
    Queue q;
    initQueue(&q);

    visited[start_vertex] = 1;
    push(&q, start_vertex);

   // printf("Обход в ширину начиная с вершины %d: ",
start_vertex);
```

```
    while (!isEmpty(&q)) {
        int vertex = pop(&q);
        //printf("%d ", vertex);

        for (int i = 0; i < n; i++) {
            if (matrix[vertex][i] == 1 && !visited[i]) {
                visited[i] = 1;
                push(&q, i);
            }
        }
    }
    printf("\n");
}

void measure_time(int n) {
    int matrix[MAX_VERTICES][MAX_VERTICES];
    generate_adjacency_matrix(n, matrix);

    int start_vertex = 0; // начальный узел

    // время для стандартной очереди
    clock_t start = clock();
    bfs_standard_queue(matrix, n, start_vertex);
    clock_t end = clock();
    double time_taken_standard = ((double)(end - start)) /
CLOCKS_PER_SEC;
```

```
    // время для самодельной очереди
    start = clock();
    bfs_custom_queue(matrix, n, start_vertex);
    end = clock();
    double time_taken_custom = ((double)(end - start)) /
CLOCKS_PER_SEC;


    printf("Размер графа: %d, Время стандартной очереди: %f,
Время самодельной очереди: %f\n", n, time_taken_standard,
time_taken_custom);
}


int main() {
    srand(time(NULL)); // Инициализация генератора случайных
чисел
    for (int n = 10; n <= 1000; n += 100) {
        measure_time(n);
    }
    return 0;
}
```

Рисунок 1 – Результат выполнения 1-го задания 1-2 часть

```
Введите размер матрицы: 10
Adjacency Matrix:
0 0 0 1 1 1 1 1 0 1
0 0 0 0 1 1 1 1 1 1
0 0 0 0 1 1 0 1 0 0
1 0 0 0 1 1 1 1 1 1
1 1 1 1 0 0 0 0 0 1
1 1 1 1 0 0 1 1 0 0
1 1 0 1 0 1 0 0 0 0
1 1 1 1 0 1 0 0 1 0
0 1 0 1 0 0 0 1 0 0
1 1 0 1 1 0 0 0 0 0
Введите стартовую вершину для BFS (0 до 9): 0
BFS обход начиная с вершины 0: 0 3 4 5 6 7 9 8 1 2
```

Рисунок 2 – Результат выполнения 1-го задания 3 часть

```
обход в ширину начиная с вершины 0:
0 4 1 5 3 2 7 8
```

Рисунок 3 – Результат выполнения 2-го задания 1 часть

```
Введите размер матрицы: 10
Adjacency Matrix:
0 0 0 0 0 0 0 0 1 0
0 0 0 1 0 1 0 1 1 0
0 0 0 1 1 1 1 0 0 1
0 1 1 0 0 1 1 1 1 0
0 0 1 0 0 1 1 0 1 1
0 1 1 1 1 0 0 0 1 0
0 0 1 1 1 0 0 0 1 1
0 1 0 1 0 0 0 0 0 1
1 1 0 1 1 1 1 0 0 0
0 0 1 0 1 0 1 1 0 0
Введите стартовую вершину для BFS (0 до 9): 0
обход в ширину начиная с вершины 0: 0 8 1 3 4 5 6 7 2 9
```

Рисунок 4 – Результат выполнения 2-го задания 2 часть

```
Размер графа: 10, Время стандартной очереди: 0.000031, Время самодельной очереди: 0.000004

Размер графа: 110, Время стандартной очереди: 0.000081, Время самодельной очереди: 0.000081

Размер графа: 210, Время стандартной очереди: 0.000300, Время самодельной очереди: 0.000280

Размер графа: 310, Время стандартной очереди: 0.000601, Время самодельной очереди: 0.000620

Размер графа: 410, Время стандартной очереди: 0.001023, Время самодельной очереди: 0.001072

Размер графа: 510, Время стандартной очереди: 0.001580, Время самодельной очереди: 0.001650

Размер графа: 610, Время стандартной очереди: 0.002271, Время самодельной очереди: 0.002283

Размер графа: 710, Время стандартной очереди: 0.003509, Время самодельной очереди: 0.003207

Размер графа: 810, Время стандартной очереди: 0.003979, Время самодельной очереди: 0.004036

Размер графа: 910, Время стандартной очереди: 0.004985, Время самодельной очереди: 0.005068
```

Вывод: В ходе выполнения лабораторной работы были изучены на практике навыки реализации алгоритма обхода графа в ширину.