Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра “Вычислительная техника”

**Отчет**

по лабораторной работе №9

по курсу “ Логика и основы алгоритмизации в инженерных задачах”

на тему “Поиск расстояний в графе”

Выполнили

студенты группы 22ВВП2:

Широкова И.Д.

Перкин П.О.

Приняли

Акифьев И.В.

Юрова О.В.

Пенза 2023

**Задание 1**

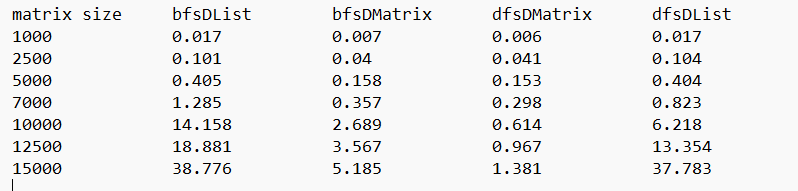
1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием. При реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки С++.

**3.**\* Реализуйте процедуру поиска расстояний для графа, представленного списками смежности.

**Задание 2\***

1. Реализуйте процедуру поиска расстояний на основе обхода в глубину.
2. Реализуйте процедуру поиска расстояний на основе обхода в глубину для графа, представленного списками смежности.
3. Оцените время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков.

**Выполнение 2 задания**



Время работы всех алгоритмов линейно зависит от размера матрицы. DFS и BFS показывают примерно сопоставимую скорость работы. Матричное представление дает преимущество в скорости по сравнению со списочным для обоих алгоритмов.

**Листинг**

**#include <iostream>**

**#include <ctime>**

**#include <cstdlib>**

**#include <queue>**

**#include <stack>**

**#include <fstream>**

**using namespace std;**

**struct Node {**

**int data;**

**Node\* next;**

**};**

**void printAdjacencyList(Node\*\* adjacencyList, int size) {**

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

**cout << "Вершина " << i << " смежна с: ";**

**Node\* current = adjacencyList[i];**

**while (current != nullptr) {**

**cout << current->data << " ";**

**current = current->next;**

**}**

**cout << endl;**

**}**

**}**

**Node\*\* adjacencyMatrixToAdjacencyList(int\*\* matrix, int size) {**

**Node\*\* adjacencyList = new Node \* [size];**

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

**adjacencyList[i] = nullptr;**

**for (int j = size - 1; j >= 0; j--) {**

**if (matrix[i][j] == 1) {**

**Node\* newNode = new Node;**

**newNode->data = j;**

**newNode->next = adjacencyList[i];**

**adjacencyList[i] = newNode;**

**}**

**}**

**}**

**return adjacencyList;**

**}**

**void freeAdjacencyList(Node\*\* adjacencyList, int size) {**

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

**Node\* current = adjacencyList[i];**

**while (current != nullptr) {**

**Node\* next = current->next;**

**delete current;**

**current = next;**

**}**

**}**

**delete[] adjacencyList;**

**}**

**int\*\* createMatrix(int size) {**

**int\*\* matrix = new int\* [size];**

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

**matrix[i] = new int[size];**

**}**

**return matrix;**

**}**

**void deleteMatrix(int\*\* matrix, int size) {**

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

**delete[] matrix[i];**

**}**

**delete[] matrix;**

**}**

**void fillMatrixRandom(int\*\* matrix, int size) {**

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

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

**if (i == j) {**

**matrix[i][j] = 0;**

**}**

**else {**

**int randomValue = rand() % 2;**

**matrix[i][j] = randomValue;**

**matrix[j][i] = randomValue;**

**}**

**}**

**}**

**}**

**void printMatrix(int\*\* matrix, int size) {**

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

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

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

**cout << ", ";**

**}**

**cout << endl;**

**}**

**}**

**void bfsDistanceList(Node\*\* adjacencyList, int size, int startVertex, int\* distances) {**

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

**distances[i] = -1;**

**}**

**queue<int> q;**

**distances[startVertex] = 0;**

**q.push(startVertex);**

**while (!q.empty()) {**

**int currentVertex = q.front();**

**q.pop();**

**Node\* adjacentNode = adjacencyList[currentVertex];**

**while (adjacentNode != nullptr) {**

**int neighbor = adjacentNode->data;**

**if (distances[neighbor] == -1) {**

**distances[neighbor] = distances[currentVertex] + 1;**

**q.push(neighbor);**

**}**

**adjacentNode = adjacentNode->next;**

**}**

**}**

**}**

**void bfsDistanceMatrix(int\*\* matrix, int size, int startVertex, int\* distances) {**

**bool\* visited = new bool[size];**

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

**distances[i] = -1;**

**visited[i] = false;**

**}**

**queue<int> q;**

**distances[startVertex] = 0;**

**visited[startVertex] = true;**

**q.push(startVertex);**

**while (!q.empty()) {**

**int currentVertex = q.front();**

**q.pop();**

**for (int neighbor = 0; neighbor < size; neighbor++) {**

**if (matrix[currentVertex][neighbor] == 1 && !visited[neighbor]) {**

**distances[neighbor] = distances[currentVertex] + 1;**

**visited[neighbor] = true;**

**q.push(neighbor);**

**}**

**}**

**}**

**delete[] visited;**

**}**

**void dfsDistanceMatrix(int\*\* matrix, int size, int startVertex, int\* distances) {**

**bool\* visited = new bool[size];**

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

**distances[i] = -1;**

**visited[i] = false;**

**}**

**stack<int> s;**

**distances[startVertex] = 0;**

**visited[startVertex] = true;**

**s.push(startVertex);**

**while (!s.empty()) {**

**int currentVertex = s.top();**

**s.pop();**

**for (int neighbor = 0; neighbor < size; neighbor++) {**

**if (matrix[currentVertex][neighbor] == 1 && !visited[neighbor]) {**

**distances[neighbor] = distances[currentVertex] + 1;**

**visited[neighbor] = true;**

**s.push(neighbor);**

**}**

**}**

**}**

**delete[] visited;**

**}**

**void dfsDistanceList(Node\*\* adjacencyList, int size, int startVertex, int\* distances) {**

**bool\* visited = new bool[size];**

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

**distances[i] = -1;**

**visited[i] = false;**

**}**

**stack<int> s;**

**distances[startVertex] = 0;**

**visited[startVertex] = true;**

**s.push(startVertex);**

**while (!s.empty()) {**

**int currentVertex = s.top();**

**s.pop();**

**Node\* adjacentNode = adjacencyList[currentVertex];**

**while (adjacentNode != nullptr) {**

**int neighbor = adjacentNode->data;**

**if (!visited[neighbor]) {**

**distances[neighbor] = distances[currentVertex] + 1;**

**visited[neighbor] = true;**

**s.push(neighbor);**

**}**

**adjacentNode = adjacentNode->next;**

**}**

**}**

**delete[] visited;**

**}**

**int main() {**

**setlocale(LC\_ALL, "Rus");**

**srand(time(0));**

**int choice;**

**do {**

**cout << "Меню:" << endl;**

**cout << "1. генерация графа и поиск расстояний" << endl;**

**cout << "2. тест алгоритмов поиска расстояний" << endl;**

**cout << "3. Выход" << endl;**

**cout << "Пожалуйста, введите номер выбранного пункта: ";**

**cin >> choice;**

**switch (choice) {**

**case 1: {**

**cout << "Вы выбрали пункт 1." << endl;**

**int size;**

**cout << "\nВведите размер матрицы смежности: ";**

**cin >> size;**

**int\*\* adjacencyMatrix = createMatrix(size);**

**fillMatrixRandom(adjacencyMatrix, size);**

**printMatrix(adjacencyMatrix, size);**

**cout << "\nСписок смежности:\n";**

**Node\*\* adjacencyList = adjacencyMatrixToAdjacencyList(adjacencyMatrix, size);**

**printAdjacencyList(adjacencyList, size);**

**int startVertex;**

**cout << "\nВведите начальную вершину для поиска расстояний: ";**

**cin >> startVertex;**

**if (startVertex >= 0 && startVertex < size) {**

**int\* distances = new int[size];**

**bfsDistanceList(adjacencyList, size, startVertex, distances);**

**cout << "\nBFSD list: Расстояния от вершины " << startVertex << " до других вершин:\n";**

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

**cout << "Вершина " << i << ": " << distances[i] << endl;**

**}**

**bfsDistanceMatrix(adjacencyMatrix, size, startVertex, distances);**

**cout << "\nBFSD matrix: Расстояния от вершины " << startVertex << " до других вершин:\n";**

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

**cout << "Вершина " << i << ": " << distances[i] << endl;**

**}**

**dfsDistanceMatrix(adjacencyMatrix, size, startVertex, distances);**

**cout << "\nDFSD matrix: Расстояния от вершины " << startVertex << " до других вершин:\n";**

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

**cout << "Вершина " << i << ": " << distances[i] << endl;**

**}**

**dfsDistanceList(adjacencyList, size, startVertex, distances);**

**cout << "\nDFSD list: Расстояния от вершины " << startVertex << " до других вершин:\n";**

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

**cout << "Вершина " << i << ": " << distances[i] << endl;**

**}**

**delete[] distances;**

**}**

**else {**

**cout << "Ошибка: Введена некорректная начальная вершина." << endl;**

**}**

**deleteMatrix(adjacencyMatrix, size);**

**freeAdjacencyList(adjacencyList, size);**

**break; }**

**case 2: {**

**cout << "Вы выбрали пункт 2." << endl;**

**int sizes[] = { 100, 500, 1000, 2000, 3000, 4000, 5000 };**

**ofstream outputFile("test.txt");**

**if (!outputFile.is\_open()) {**

**cerr << "Failed to open the output file." << endl;**

**return 1;**

**}**

**outputFile << "matrix size\tbfsDList\tbfsDMatrix\tdfsDMatrix\tdfsDList\n";**

**for (int i = 0; i < 7; i++) {**

**int size = sizes[i];**

**int\*\* adjacencyMatrix = createMatrix(size);**

**fillMatrixRandom(adjacencyMatrix, size);**

**Node\*\* adjacencyList = adjacencyMatrixToAdjacencyList(adjacencyMatrix, size);**

**int startVertex = 0;**

**int\* distances = new int[size];**

**clock\_t startTime, endTime;**

**startTime = clock();**

**bfsDistanceList(adjacencyList, size, startVertex, distances);**

**endTime = clock();**

**double bfsDistanceListTime = double(endTime - startTime) / CLOCKS\_PER\_SEC;**

**startTime = clock();**

**bfsDistanceMatrix(adjacencyMatrix, size, startVertex, distances);**

**endTime = clock();**

**double bfsDistanceMatrixTime = double(endTime - startTime) / CLOCKS\_PER\_SEC;**

**startTime = clock();**

**dfsDistanceMatrix(adjacencyMatrix, size, startVertex, distances);**

**endTime = clock();**

**double dfsDistanceMatrixTime = double(endTime - startTime) / CLOCKS\_PER\_SEC;**

**startTime = clock();**

**dfsDistanceList(adjacencyList, size, startVertex, distances);**

**endTime = clock();**

**double dfsDistanceListTime = double(endTime - startTime) / CLOCKS\_PER\_SEC;**

**outputFile << size << "\t\t" << bfsDistanceListTime << "\t\t" << bfsDistanceMatrixTime << "\t\t" << dfsDistanceMatrixTime << "\t\t" << dfsDistanceListTime << "\n";**

**deleteMatrix(adjacencyMatrix, size);**

**freeAdjacencyList(adjacencyList, size);**

**}**

**outputFile.close();**

**break; }**

**case 3:**

**cout << "Выход." << endl;**

**break;**

**default:**

**cout << "Неправильный выбор. Пожалуйста, введите верный номер пункта." << endl;**

**break;**

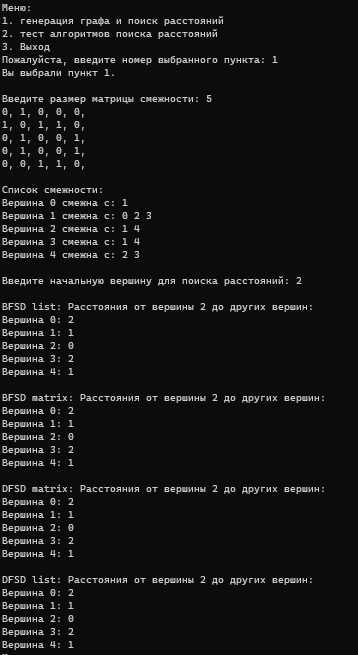
**}**

**} while (choice != 3);**

**return 0;**

**}**

**Результаты работы программы**

****

**Вывод**

В ходе выполнения задания были реализованы процедуры поиска расстояний, как на основе обхода в ширину, так и на основе обхода в глубину. При реализации алгоритмов использовался класс queue из стандартной библиотеки C++, а также списки смежности для обоих методов поиска расстояний. В ходе выполнения второго задания, было произведено сравнение времени работы алгоритмов обхода в ширину и обхода в глубину для графов разных порядков, позволяя оценить эффективность их применения в различных сценариях.