Министерство науки и высшего образования Российской федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №9

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Поиск расстояний в графе»

Выполнили ст. группы 22ВВВ1:

Уткин М.М.

Саветкин Д.Д.

Соколовский Е.В

Приняли:

К.э.н., доцент Акифьев И. В.

К.т.н., доцент Юрова О. В.

Пенза 2023

**Цель работы:**

Научиться пользоваться алгоритмом поиска расстояний в графе.

**Лабораторное 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 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.

.

**Ход работы:**

Ввод данных:

Пользователь задает количество вершин графа (n).

Генерация матрицы смежности:

Случайным образом генерируется матрица смежности (Matr). Значения в матрице принимают значения 0 или 1, обозначая отсутствие или наличие ребра между соответствующими вершинами.

Создание списка смежности и графа:

На основе сгенерированной матрицы смежности создаются список смежности (list) и граф (graph).

Поиск расстояний с использованием матрицы смежности:

Пользователь вводит начальную и конечную вершины для определения расстояния между ними в графе, представленном матрицей смежности.

Выполняется обход в ширину (RGM(graph, start1, end1, distances)) для нахождения кратчайшего пути.

Выполняется обход в глубину (GGM(graph, start2, end2, distances2)) для нахождения пути с использованием поиска в глубину.

Создание и вывод списка смежности:

Программа создает список смежности (list) на основе матрицы смежности и выводит его на экран.

Поиск расстояний с использованием списка смежности:

Пользователь вводит начальную и конечную вершины для определения расстояния между ними в графе, представленном списком смежности.

Выполняется обход в ширину (GGM\_list(list, start3, end3, distances3)) для нахождения кратчайшего пути.

Выполняется обход в глубину (DFS\_list(list, start4, end4, distances4)) для нахождения пути с использованием поиска в глубину.

Измерение времени выполнения обходов:

Программа измеряет время выполнения обхода в ширину и глубину для матрицы смежности с использованием хронометража.

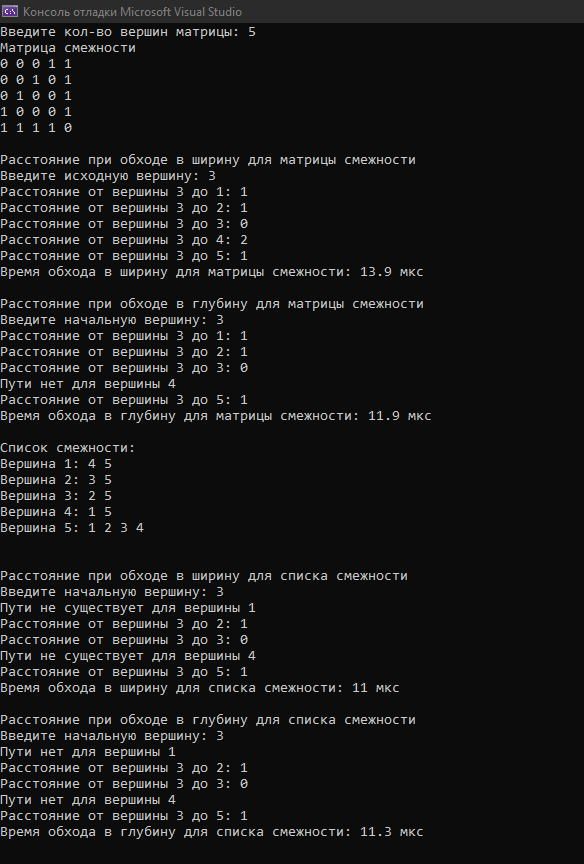
Вывод результатов:

Программа выводит кратчайшие расстояния для обходов в ширину и глубину для матрицы смежности, список смежности и время выполнения каждого обхода.

Окончание работы программы:

Программа завершает выполнение, выводя результаты на экран и завершая работу.

**Результаты работы программы:**



**Таблица: Время работы реализаций алгоритмов поиска расстояний:**

|  |  |
| --- | --- |
| Наименование | Время выполнения |
| Обход в ширину для матрицы смежности | 13.9мкс |
| Обход в глубину для матрицы смежности | 11.9мкс |
| Обход в ширину для списка смежности | 11мкс |
| Обход в глубину для списка смежности | 11.3мкс |

**Вывод:**

В результате выполнения лабораторной работы были изучены и реализованы методы поиска расстояний в графах. Обходы в ширину и в глубину представляют собой эффективные инструменты для нахождения кратчайших путей между вершинами. Работа с графом в форме матрицы смежности и списка смежности позволяет выбирать наилучший способ представления данных, а измерение времени выполнения обходов выявляет разницу в эффективности алгоритмов. Это важный критерий при выборе метода для работы с обширными графами.

**Листинг:**

[#include](https://vk.com/im?sel=457408366&st=%23include) <iostream>  
[#include](https://vk.com/im?sel=457408366&st=%23include) <cstdlib>  
[#include](https://vk.com/im?sel=457408366&st=%23include) <stack>  
[#include](https://vk.com/im?sel=457408366&st=%23include) <vector>  
[#include](https://vk.com/im?sel=457408366&st=%23include) <algorithm>  
[#include](https://vk.com/im?sel=457408366&st=%23include) <queue>  
[#include](https://vk.com/im?sel=457408366&st=%23include) <chrono>  
  
using namespace std;  
  
void printMatrix(const vector<vector<int»& matrix) {  
for (const auto& row : matrix) {  
for (int element : row) {  
cout « element « " ";  
}  
cout « endl;  
}  
cout « "\n";  
}  
  
void printAdjacencyList(const vector<vector<int»& adjacencyList) {  
for (int i = 0; i < adjacencyList.size(); ++i) {  
cout « "Вершина " « i + 1 « ": ";  
for (int neighbor : adjacencyList[i]) {  
cout « neighbor + 1 « " ";  
}  
cout « "\n";  
}  
cout « "\n";  
}  
  
void breadthFirstSearch(const vector<vector<int»& graph, int start, int end, vector<int>& distances) {  
queue<int> q;  
q.push(start);  
distances[start] = 0;  
  
while (!q.empty()) {  
int v = q.front();  
q.pop();  
  
if (v == end) break;  
  
for (int to : graph[v]) {  
if (distances[to] == -1) {  
q.push(to);  
distances[to] = distances[v] + 1;  
}  
}  
}  
}  
  
void depthFirstSearch(const vector<vector<int»& graph, int start, int end, vector<int>& distances) {  
stack<int> s;  
s.push(start);  
distances[start] = 0;  
  
while (!s.empty()) {  
int v = [s.top](https://vk.com/away.php?to=http%3A%2F%2Fs.top&cc_key=)();  
s.pop();  
  
if (v == end) break;  
  
for (int to : graph[v]) {  
if (distances[to] == -1) {  
distances[to] = distances[v] + 1;  
s.push(to);  
}  
}  
}  
}  
  
int main() {  
setlocale(LC\_ALL, "Russian");  
srand(time(NULL));  
  
int n;  
cout « "Введите кол-во вершин матрицы: ";  
cin » n;  
  
vector<vector<int» matrix(n, vector<int>(n));  
vector<vector<int» adjacencyList(n);  
vector<vector<int» graph(n, vector<int>(n));  
  
// Матрица смежности  
cout « "Матрица смежности " « endl;  
  
for (int i = 0; i < n; i++) {  
for (int j = i; j < n; j++) {  
if (i == j) {  
matrix[i][j] = 0;  
}  
else {  
matrix[i][j] = rand() % 2;  
matrix[j][i] = matrix[i][j];  
}  
}  
}  
  
// Вывод матрицы смежности  
printMatrix(matrix);  
  
// Заполнение графа на основе матрицы смежности  
for (int i = 0; i < n; ++i) {  
for (int j = 0; j < n; ++j) {  
if (matrix[i][j] == 1) {  
graph[i].push\_back(j);  
}  
}  
}  
  
// Нахождение расстояния при обходе в ширину для матрицы смежности  
cout « "Расстояние при обходе в ширину для матрицы смежности\n";  
  
int start1;  
cout « "Введите исходную вершину: ";  
cin » start1;  
start1--;  
  
vector<int> distances(n, -1);  
auto startTimeBFSMatrix = chrono::high\_resolution\_clock::now();  
breadthFirstSearch(graph, start1, n - 1, distances);  
auto endTimeBFSMatrix = chrono::high\_resolution\_clock::now();  
  
auto durationBFSMatrix = endTimeBFSMatrix - startTimeBFSMatrix;  
chrono::duration<double, micro> microsecondsBFSMatrix = chrono::duration\_cast<chrono::duration<double, micro»(durationBFSMatrix);  
  
for (int end1 = 0; end1 < n; ++end1) {  
if (distances[end1] != -1) {  
cout « "Расстояние от вершины " « start1 + 1 « " до " « end1 + 1 « ": " « distances[end1] « endl;  
}  
else {  
cout « "Пути не существует для вершины " « end1 + 1 « endl;  
}  
}  
  
cout « "Время обхода в ширину для матрицы смежности: " « microsecondsBFSMatrix.count() « " мкс" « endl;  
  
fill(distances.begin(), distances.end(), -1);  
  
cout « "\n";  
  
// Обход в глубину для матрицы смежности  
cout « "Расстояние при обходе в глубину для матрицы смежности\n";  
  
int start2;  
cout « "Введите начальную вершину: ";  
cin » start2;  
start2--;  
  
// Нахождение расстояния при обходе в глубину для матрицы смежности  
vector<int> distances2(n, -1);  
auto startTimeDFSMatrix = chrono::high\_resolution\_clock::now();  
depthFirstSearch(graph, start2, n - 1, distances2);  
auto endTimeDFSMatrix = chrono::high\_resolution\_clock::now();  
  
auto durationDFSMatrix = endTimeDFSMatrix - startTimeDFSMatrix;  
chrono::duration<double, micro> microsecondsDFSMatrix = chrono::duration\_cast<chrono::duration<double, micro»(durationDFSMatrix);  
  
for (int end2 = 0; end2 < n; ++end2) {  
if (distances2[end2] != -1) {  
cout « "Расстояние от вершины " « start2 + 1 « " до " « end2 + 1 « ": " « distances2[end2] « endl;  
}

else {  
cout « "Пути нет для вершины " « end2 + 1 « endl;  
}  
}  
  
cout « "Время обхода в глубину для матрицы смежности: " « microsecondsDFSMatrix.count() « " мкс" « endl;  
  
fill(distances2.begin(), distances2.end(), -1);  
  
// Заполнение списка смежности на основе матрицы  
for (int i = 0; i < n; ++i) {  
for (int j = 0; j < n; ++j) {  
if (matrix[i][j] == 1) {  
adjacencyList[i].push\_back(j);  
}  
}  
}  
  
// Вывод списка смежности  
cout « "\nСписок смежности:\n";  
printAdjacencyList(adjacencyList);  
  
cout « "\n";  
  
// Нахождение расстояния при обходе в ширину для списка смежности  
cout « "Расстояние при обходе в ширину для списка смежности\n";  
  
int start3;  
cout « "Введите начальную вершину: ";  
cin » start3;  
start3--;  
  
vector<int> distances3(n, -1);  
auto startTimeBFSList = chrono::high\_resolution\_clock::now();  
breadthFirstSearch(adjacencyList, start3, n - 1, distances3);  
auto endTimeBFSList = chrono::high\_resolution\_clock::now();  
  
auto durationBFSList = endTimeBFSList - startTimeBFSList;  
chrono::duration<double, micro> microsecondsBFSList = chrono::duration\_cast<chrono::duration<double, micro»(durationBFSList);  
  
for (int end3 = 0; end3 < n; ++end3) {  
if (distances3[end3] != -1) {  
cout « "Расстояние от вершины " « start3 + 1 « " до " « end3 + 1 « ": " « distances3[end3] « endl;  
}  
else {  
cout « "Пути не существует для вершины " « end3 + 1 « endl;  
}  
}  
  
cout « "Время обхода в ширину для списка смежности: " « microsecondsBFSList.count() « " мкс" « endl;  
  
fill(distances3.begin(), distances3.end(), -1);  
  
// Нахождение расстояния при обходе в глубину для списка смежности  
cout « "\nРасстояние при обходе в глубину для списка смежности\n";  
  
int start4;  
cout « "Введите начальную вершину: ";  
cin » start4;  
start4--;  
  
vector<int> distances4(n, -1);  
auto startTimeDFSList = chrono::high\_resolution\_clock::now();  
depthFirstSearch(adjacencyList, start4, n - 1, distances4);  
auto endTimeDFSList = chrono::high\_resolution\_clock::now();  
  
auto durationDFSList = endTimeDFSList - startTimeDFSList;  
chrono::duration<double, micro> microsecondsDFSList = chrono::duration\_cast<chrono::duration<double, micro»(durationDFSList);  
  
for (int end4 = 0; end4 < n; ++end4) {  
if (distances4[end4] != -1) {  
cout « "Расстояние от вершины " « start4 + 1 « " до " « end4 + 1 « ": " « distances4[end4] « endl;  
}  
else {  
cout « "Пути нет для вершины " « end4 + 1 « endl;  
}  
}  
  
cout « "Время обхода в глубину для списка смежности: " « microsecondsDFSList.count() « " мкс" « endl;  
  
cout « "\n";  
  
return 0;  
}