­­Бюджетное учреждение высшего образования   
Ханты-Мансийского автономного округа   
«Сургутский государственный университет»

Политехнический институт

Кафедра автоматики и компьютерных систем

**Отчет**

по лабораторной работе № 8

по дисциплине «Алгоритмы и структуры данных»

Выполнил: студент группы 609-21,

Шумилов И.Д.

Принял: старший преподаватель кафедры АиКС

Назаров Е.В.

2024 г.

**Цель работы:** изучить основные алгоритмы работы с графами – обход, построение транзитивного замыкания, поиск кратчайших путей, закрепить навыки структурного и объектно-ориентированного программирования.

**Общее задание на работу:**

1. Используя исходный код, приведенный в приложении, разработать функции (методы) реализующие задания, представленные в таблице 1 в соответствии с вариантом задания. При этом необходимо выбрать тип графа и методы, которые позволят эффективно решить поставленную задачу. Решение аргументировать.

**Индивидуальное задание на работу:**

Таблица 1 – Вариант 2 индивидуального ндивидуального задания

|  |  |
| --- | --- |
| № | Задание |
| 2 | Граф представляет карту дорог. Найти кратчайший путь из города A в город B, через город C. Если такой путь не существует, выдать соответствующее сообщение. |

**Листинги функций**

Листинг 1. Функция поиска кратчайшего пути алгоритмом Дейкстра.

Graph Graph::ShortestPath(int From, int To) {

if (From >= 0 && From < \_Size && To < \_Size && To >= 0) {

if (From == To) return Graph(0);

Deikstra\* D = new Deikstra[\_Size];

D[From].Path = 0;

D[From].Vertex = -1;

int U, V = From;

double SPath;

do {

D[V].Label = true;

for (U = 0; U < \_Size; U++) {

if (D[U].Label || \_m[V][U] == DBL\_MAX) continue;

if (D[U].Path > D[V].Path + \_m[V][U]) {

D[U].Path = D[V].Path + \_m[V][U];

D[U].Vertex = V;

}

}

SPath = DBL\_MAX;

V = -1;

for (U = 0; U < \_Size; U++)

if (!(D[U].Label) && D[U].Path < SPath) {

SPath = D[U].Path;

V = U;

}

} while (V != -1 && V != To);

Graph G(\_Size);

V = To;

while (D[V].Vertex != -1) {

G.AddEdge(V, D[V].Vertex, \_m[V][D[V].Vertex]);

V = D[V].Vertex;

}

delete[] D;

return G;

}

return Graph(0);

}

Листинг 2. Функция объединения двух графов (путей).

Graph Graph::Union(Graph G1, Graph G2) {

if (G1.\_Size != G2.\_Size)

return Graph(0);

Graph G(G1.\_Size);

for (int i = 0; i < G1.\_Size; i++) {

for (int j = 0; j < G1.\_Size; j++)

if (G1.\_m[i][j] < G2.\_m[i][j]) {

G.\_m[i][j] = G1.\_m[i][j];

} else {

G.\_m[i][j] = G2.\_m[i][j];

}

}

return G;

}

**Пояснение к решению**

Из курса дискретной математики известно, что кратчайший путь из вершины A в вершину B, проходящий через C равен кратчайшему пути из A в C и из C в B. Таким образом, необходимо найти кратчайшие пути из A в C и из C в B, после чего объединить эти пути.

Для решения задачи был выбран тип неориентированного взвешенного графа, т.к. граф представляет города и дороги между ними, то логично будет предположить, что по дорогам можно перемещаться в обе стороны и, т.к. нужно измерять расстояние между городами, то необходимо хранить длину дорог.

Для нахождения путей был выбран алгоритм Дейкстры, т.к. длина дороги не может быть отрицательной и нас интересует только расстояние пути из одного пункта в другой.

**Пример выполнения реализованного алгоритма**

Листинг 3. Выполняемый код.

// Создание графа с size вершин

int size = 10;

WGraph g(size);

int cityA = 0;

int cityB = 7;

int cityC = 3;

std::cout << "Verticies: " << size << std::endl;

std::cout << "City A: " << cityA << std::endl;

std::cout << "City B: " << cityB << std::endl;

std::cout << "City C: " << cityC << std::endl;

// Заполнение графа случайными значениями

g.Random(1, 5);

// Вывод графа

g.Print();

// Поиск кратчайших путей

Graph g1 = g.ShortestPath(cityA, cityC);

Graph g2 = g.ShortestPath(cityC, cityB);

// Проверка на существование кратчайшего пути через город С

if (g1.EdgeCount() != 0 && g2.EdgeCount() != 0) {

// Объединение путей

Graph path = Graph::Union(g1, g2);

path.Print();

path.PrintPath(cityA, cityB);

std::cout << "Path Length: " << path.PathLength() << std::endl;

} else {

std::cout << "[ERROR] Cannot find path from A to B through C" << std::endl;

}

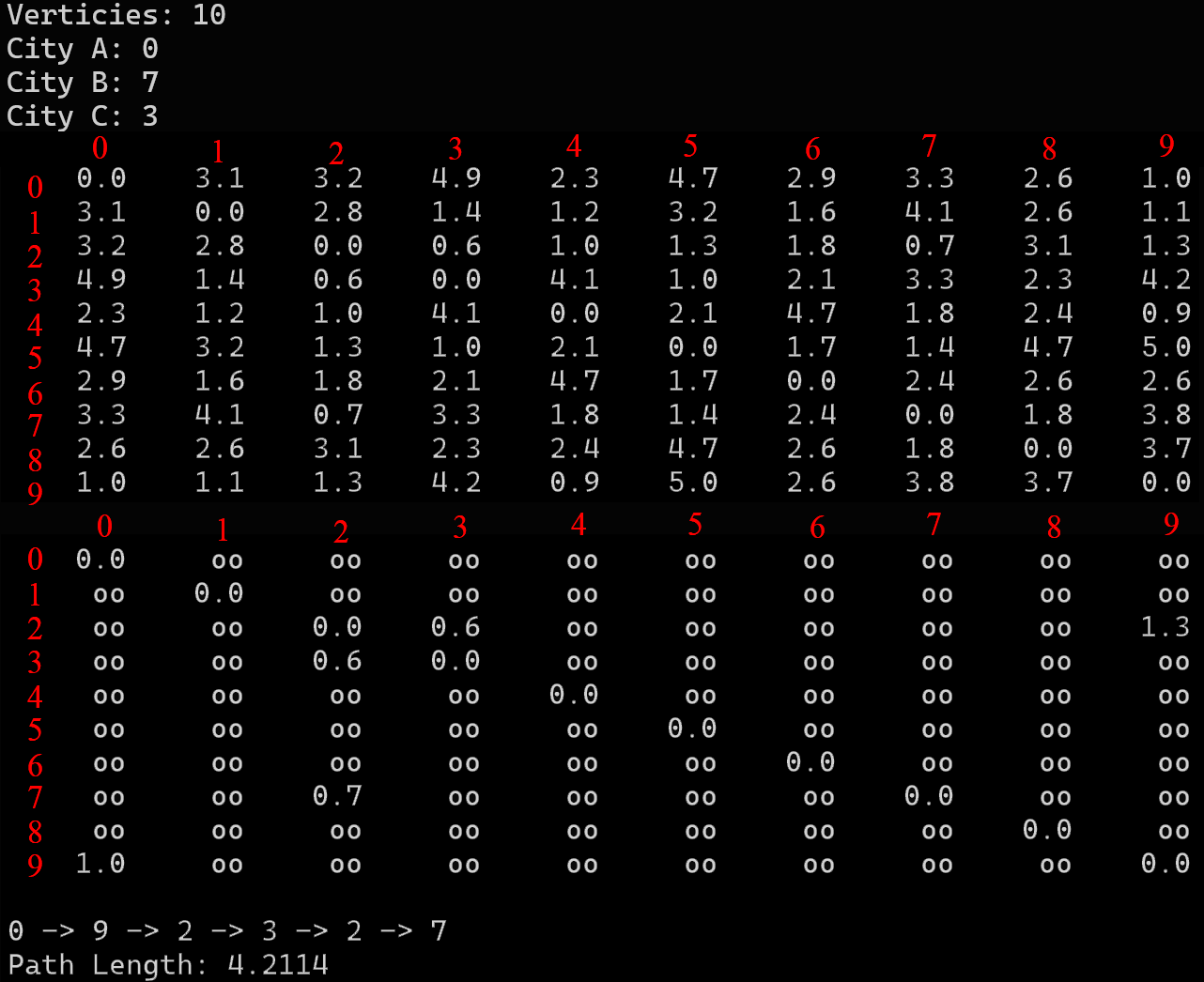


Рис. 1 – Пример работы программы

**Вывод:** были изучены основные алгоритмы работы с графами – обход, построение транзитивного замыкания, поиск кратчайших путей, закреплены навыки структурного и объектно-ориентированного программирования.