DEAKIN UNIVERSITY

DATA STRUCTURES AND ALGORITHMS

ONTRACK SUBMISSION

Problem Solving: Graphs

Submitted By:
Peter STACEY
pstacey
2020/09/23 12:06

Tutor:
Maksym SLAVNENKO

Outcome	**Weight**
Complexity	♦♦♦♦◊
Implement Solutions	♦◊◊◊◊
Document solutions	♦♦♦♦◊

The task involves evaluating the difference between Dijkstra and Floyd's algorithms for a specific problem, with evaluating the different time and space complexity of an adjacency matrix v adjacency list for graph representation and with looking at the different ways that depth-first v breadth-first searches work on a graph. These aspects involve evaluating the memory use and time complexity of algorithms (ULO1), which providing recommendations on a solution (ULO1) and with documenting solution constraints and design decisions (ULO3).

September 23, 2020

Task 9.1

Student Name: Peter Stacey

Student ID: 219011171

Question 1

In an adjacency matrix, a square matrix is used to represent the graph, where the elements of the matrix indicate whether pairs of vertices are adjacent to each other. for each vertex, edges to every adjacent vertex is represented as a 1, while vertices that are not adjacent, are represented as a 0.

As a result, for every vertex, to search the graph, the value of every other vertex in the graph needs to be examined.

Therefore, for an adjacency matrix, the time complexity is *$O(V^2)$.*

However, for an adjacency list, a collection of lists is used to represent the graph, where each list contains the list of neighbours of a specific vertex. Each Vertex has a list to represent it's neighbours. In this case, the time complexity is *$O(V + E)$*.

As a result, in general:

- Adjacency matrix is slower for a breadth-first search, than an adjacency list representation
- The times approach each other when each vertex is connected to every other vertex (eg. mesh network)

Question 2

Floyd's algorithm finds the shortest path between all vertices, while Dijkstra's algorithm finds the shortest path between a single vertex and all other vertices.

In the case of the task, a user needs to find the shortest distance from their current location to each major tourist location. This most closely aligns with the output of Dijkstra's algorithm, however in a real application, tourists might be starting from many different locations, and Dijkstra's algorithm would need to be run multiple times to find all of the paths from every possible starting location.

In this case, factors to consider include:

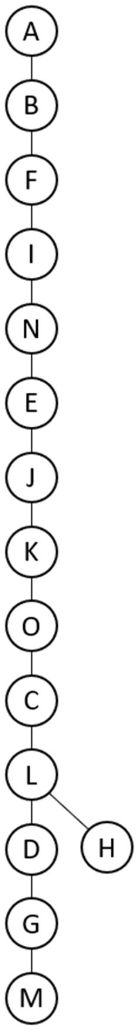
1. Can the results of all paths be cached in the application (or stored persistently)?
    - In this case, it may be best to choose Floyd's algorithm and then just look up the relevant result for the starting position.
2. Does the path need to be calculated every time, because every tourist potentially has a different starting position?
    - In this case, Dijkstra's algorithm will be the required choice, as it focuses only on the start position of the tourist and will generally be faster than Floyd's algorithm.
3. Are tourists likely to change their mind during their trip, and go to each attraction in different orders?

- In this case, Floyd's algorithm might be a better choice, as it will give them more options from a single calculation, while Dijkstra's algorithm would need to be run multiple times and take up additional space to store all results.

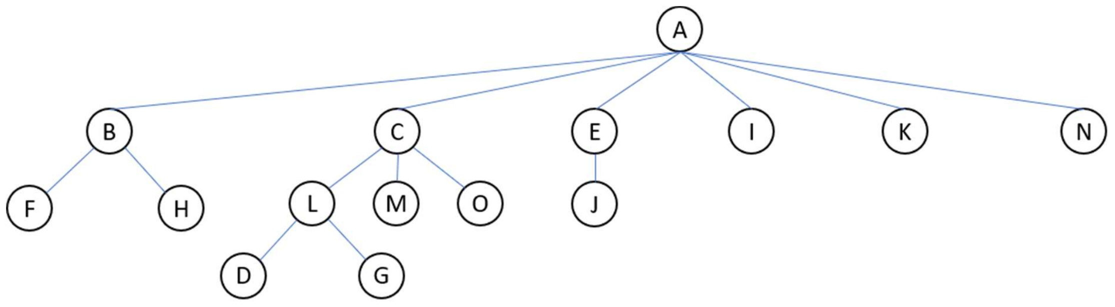
4. How sparse or dense is the graph of edges to vertices?
    - In the case of a sparse graph, where there are few cycles Dijkstra's algorithm may be the better choice.
5. Are tourists likely to back track on their path (ie. Use negative edges in their trip)?
    - In this case, Floyd's algorithm can handle the presence of negative edges, while Dijkstra's algorithm does not.

Question 3

Depth-First Search



Breadth-First Search


A
B
C
E
I
K
N
F
H
L
M
O
J
D
G