ASSIGNMENT COVERSHEET

PRAGUECITY UNIVERSITY

Student Name: Daniel Shutov	
Class: Algorithms and Data Structures (FT/BL)	
Assignment: Shortest Route	
Lecturer: Dominik Pantůček	Semester: 2201
Due Date: 2022-Jun-07	Actual Submission Date: Submission date

Evidence Produced (List separate items)	**Location (Choose one)**	
	X	Uploaded to the Learning Center (Moodle)
		Submitted to reception
Note: Email submissions to the lecturer are not valid.		

Student Declaration:

I declare that the work contained in this assignment was researched and prepared by me, except where acknowledgement of sources is made. I understand that the college can and will test any work submitted by me for plagiarism.
Note: The attachment of this statement on any electronically submitted assignments will be deemed to have the same authority as a signed statement

Date: March 3, 2023	Student Signature: Daniel Shutov

A separate feedback sheet will be returned to you after your work has been graded.
Refer to your Student Manual for the Appeals Procedure if you have concerns about the grading decision.

Student Comment (Optional)
Was the task clear? If not, how could it be improved?
Was there sufficient time to complete the task? If not, how much time should be allowed?
Did you need additional assistance with the assignment?
Was the lecturer able to help you?
Were there sufficient resources available?
How could the assignment be improved?

Shortest Route

Daniel Shutov

March 3, 2023

Contents

List of Figures

Listings

1 List of symbols and list of acronyms

Graph: As Edpresso (2021) said, a graph is a common data structure that consists of a finite set of nodes (or vertices) and a set of edges connecting them. A pair (x,y) is referred to as an edge, which communicates that the x vertex connects to the y vertex
BFS: As Edpresso (2019) said, is an algorithm for traversing or searching tree or graph data structures. It explores all the nodes at the present depth before moving on to the nodes at the next depth level.

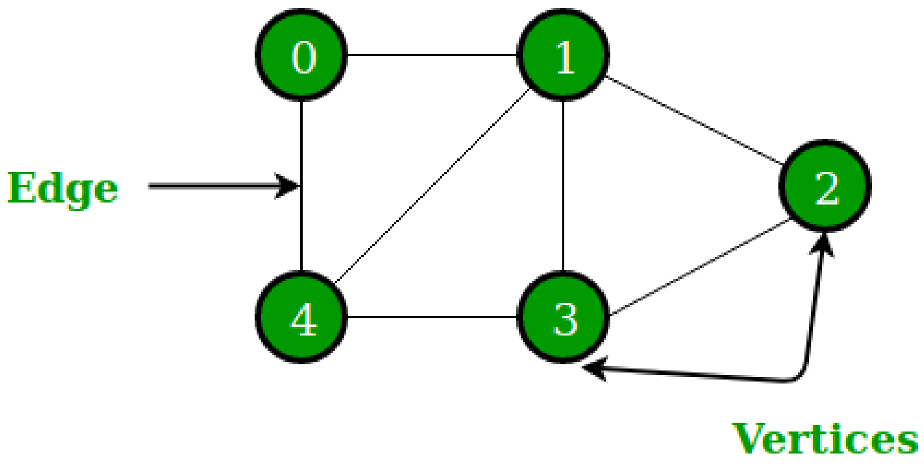
2 Introduction

This technical report will discuss the problem that was assigned in the ICA "How to find the shortest path with and without edge weights". The report must include their time and memory complexity, algorithms that were chosen and their justification and modifications, if possible. The assignment was completed only by using the knowledge that was gained during the classes.

3 Understanding of Graphs

Graphs, as showed in Figure 1, are used to address real-world problems in which the problem area is represented as a network. There are many fields in which graphs can be used to solve different problems. For example, social networks, maps and circuit networks can all be represented by graphs. Graphs are non-linear data structures made up of vertices and edges. Vertices are data storage entities with edges expressing their relationships. Edges might be directed or not.

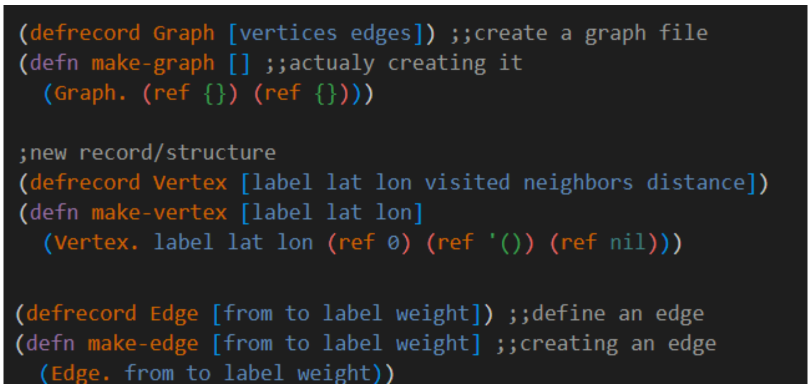
Figure 1: Simple undirected graph



3.1 Creation of a Graph

Before the problem could be solved, there are some necessary steps that must be taken. First, defining the data structure. Second, initializing it with the right values. These two steps are correct to the Graph creation, that consists of vertices and edges that are initialized after definition.

Figure 2: Creation of a Graph



```
(defrecord Graph [vertices edges]) ;;create a graph file
(defn make-graph [] ;;actualy creating it
  (Graph. (ref {}) (ref {})))

;new record/structure
(defrecord Vertex [label lat lon visited neighbors distance])
(defn make-vertex [label lat lon]
  (Vertex. label lat lon (ref 0) (ref '()) (ref nil)))

(defrecord Edge [from to label weight]) ;;define an edge
(defn make-edge [from to label weight] ;;creating an edge
  (Edge. from to label weight))
```

4 Breadth First Search Algorithm

Before tackling the challenges in this technical paper, the breadth-first algorithm (BFS) must be defined as a prerequisite. The proposed solutions and algorithms in the subsequent portions of this work are based on the BFS algorithm. The breadth-first approach is designed to traverse graphs using two types of queues; A queue of open vertices to be processed, and a queue of "marked" vertices that have already been handled. An initial element must be provided. It then represents a starting vertex in the open queue. While the open queue is not empty, the following logic occurs; The first vertex in the open queue is removed from the open queue and utilized as the current vertex. The current vertex's neighbors, which are not yet marked, are then added to the open queue at the end, that means that "first in first out" like line in a supermarket. Then, the current vertex is marked as "visited", processed, and added to the closed queue. The next vertex in the open queue becomes the current vertex, and the loop continues until there are no elements in the open queue.

5 Dijkstra's Algorithm Explanation

Dijkstra's algorithm consists of two phases, marking and tracing: Initially, all vertices will be designated as "unseen" or "unvisited", and then it will receive the input, which consists of the first and last vertices of the path. As was explained by study.com (2021): "Start at the ending vertex by marking it with a distance of 0, because it is 0 units from the end. Call this vertex your current vertex, and put a circle around it indicating as such."
In other words, the last vertex then becomes the "current" vertex. The algorithm identifies all "current" vertex's vertices as "neighbors" and inserts them into a processing queue. As was stated by study.com (2021): "Identify all the vertices that are connected to the current vertex with an edge. Calculate their distance to the end by adding the weight of the edge to the mark on the current vertex. Mark each of the vertices with their corresponding distance, but only change a vertex's mark if it is less than a previous mark. Each time you mark the starting vertex with a mark, keep track of the path that resulted in that mark." It is important to say that this part is relevant only for the second problem that includes the edge weights, and for the first problem it is meaningless. When there are no more adjacent nodes to update, the current vertex is marked as "visited" that means in other words that it is possible theoretically to put "X" on it and dont look at it again, from there a new vertex is chosen from the queue; the process then repeats. After the initial/starting vertex is marked as "visited", the graph is retraced and the shortest path from end to start is repeatedly picked by selecting the vertex neighbor with the shortest distance to the finish.

5.1 Problem 1:

The first problem is to determine the shortest path between two cities, and to show if that there is no path. The shortest path, for this problem, is measured by the amount of cities, represented as vertices, crossed.

5.2 Recommended algorithm

The author chose to use Dijkstra's algorithm to find the shortest path in a graph without edge weights, it will be performed as was explained in section 5 but without the edge weights part, it will be just choosing the neighbors of each vertex with the smallest distance to finish.

5.2.1 Justification

The algorithm that was chosen, Dijkstra's algorithm, is one of the simplest algorithms to implement and understand. Unfortunately, at the time when the author wrote this technical paper, the author understood that without edge weights there is no other information to use, therefore the author did not have any other choice. Simply saying, Dijkstra's algorithm is the best in this case.

5.2.2 Notations

Big O:

- The time complexity for our worst-case scenario is: O(n) (linear).
- The memory complexity for our worst-case scenario is: O(n) (linear).

Big Ω:

- The time complexity for the best-case scenario is: Ω(n) (linear).
- The memory complexity for the best-case scenario is: Ω(1) (constant).

Big Θ:

- The time complexity for our Θ is linear (n). Therefore the algorithm is tightly bound.
- The memory complexity for our Θ is: Ω != O $\implies$ Big Theta does not apply.

5.2.3 Evaluation of efficiency

The author is confident that in terms of algorithm complexity in an unweighted graph, there is no better alternative than the provided algorithm. Although it can be slightly improved by pausing at the marking stage when it reaches the initial vertex. Moreover, it depends only on the size of the graph, therefore in worst case scenarios it will be linear.

5.2.4 Solution for problem 1

The solution can be generated in the .clj file that came with this technical paper.

5.3 Problem 2: Finding the shortest path with edge weights

Find the shortest path between two cities and show that there is no path from a given city to "Prague". This time, the shortest path is measured as the sum of lengths of roads traversed. Road lengths are given as edge weights.

5.3.1 Recommended algorithm

The author chose to use Dijkstra's algorithm to find the shortest path in a graph with edge weights. It is easy to implement, although it could be improved. It will be preformed as was explained in section 5.

5.3.2 Justification

The algorithm is flawless, consistent, straightforward to implement, and does not rely on any additional assumptions. It is crucial to note that the chosen method is not the greatest in many situations, but it always produces accurate results, which is to say that it is enough.

5.3.3 Notations

Big O:

- The time complexity for our worst-case scenario is: O($n^2 + 2E$) (quadratic).
- The memory complexity for our worst-case scenario is: O(n) (linear).

Big Ω:

- The time complexity for the best-case scenario is: $\Omega(N + 2E)$.
- The memory complexity for the best-case scenario is: Ω(n) (linear).

Big Θ:

- The time complexity for our Θ is: Ω != O $\implies$ Big Theta does not apply.
- The memory complexity for our Θ is: Depends on the amount of vertices (v) therefore the algorithm is tightly bound.

5.3.4 Evaluation of efficiency

Dijkstra's algorithm provides an efficient method for traversing large weighted graphs in search of the shortest path, although it is not totally correct, only of course if the weights are correct, for example, if they were computed using great circle distance. This is due to the linear geometry employed implicitly when computing the distance between cities (edge weight) and simply adding them. As theguardian.com (2021) said, "Because the Earth is not flat, the greater the distance, the more erroneous the results will be". Therefore, proper measurements and geometry are required to further improve the outcomes. As stated by geeksforgeeks.org (2020): "Although being the best path finding algorithm around, A* Search Algorithm doesn't produce the shortest path always, as it relies heavily on heuristics / approximations" but it is worth to say that as long as your heuristic function decreases monotonically toward the endpoint and is consistent with edge weights, you will always obtain the proper results.

5.3.5 Solution for problem 2

The solution can be generated in the .clj file that came with this technical paper.

6 Suggested improvements

There are some improvements that can be done:

- Increasing the "user-friendliness" of the code by allowing users to input the name of the city they wish to visit, plus creating a function that check user input, if it is valid. The current application is only capable of working with pre-written (entered manually) start and end locations.
- The algorithm could trade speed for accuracy by giving more weight to the heuristic function than to total distance accumulated so far.
- Using A* instead of Dijkstra's because A* attempts to find a better path by employing a heuristic function which grants precedence to nodes that are believed to be superior to others, whereas Dijkstra's algorithm just explores all possible paths.

7 List of figures

References

Edpresso (June (2021)). “Graph Data Structure”. en. In: URL: `https://www.educative.io/edpresso/what-is-a-graph-data-structure`.

— ((2019)). “What is Breadth First Search?” en. In: URL: `https://www.javatpoint.com/breadth-first-search-algorithm`.

geeksforgeeks.org (June (2020)). “a-search-algorithm”. en. In: URL: `https://www.geeksforgeeks.org/a-search-algorithm/`.

study.com (June (2021)). “Dijkstra-s-algorithm-definition”. en. In: URL: `https://study.com/academy/lesson/dijkstra-s-algorithm-definition-applications-examples.html`.

theguardian.com (June (2021)). “Why the Earth is actually flat”. en. In: URL: `https://www.theguardian.com/science/brain-flapping/2016/jan/26/earth-totally-flat-conspiracy-bob`.