CAB203: Project Report

N10477659, Ash Phillips
28th May, 2021

Table of Contents

1.0. Introduction 2
2.0. Instance Model 3
3.0. Solution Model 4
4.0. Problem Model 5
5.0. Solution Method 6

N10477659
Ashley Phillips

1.0. Introduction

A man named Bob lives in an isolated community, such that everyone in the said community has come to be related to each other over the generations. Bob has created a full family tree, beginning with the first generation, in the form of cards. Each card documents a person's name, the names of their parents, and the names of their children, as seen below:

Name: Alice Father: Arlo Mother: Madeline Children: None	Name: Bob Father: Charlie Mother: Eve Children: None
Name: Eve Father: Oliver Mother: Aurora Children: Bob	Name: Charlie Father: Jack Mother: Luna Children: Bob
Name: Madeline Father: Jack Mother: Aurora Children: Alice	Name: Arlo Father: Oscar Mother: Isla Children: Alice
Name: Oliver Father: Hugo Mother: Rose Children: Eve	Name: Luna Father: Oscar Mother: Isla Children: Charlie
Name: Aurora Father: Hugo Mother: Rose Children: Eve, Madeline	Name: Jack Father: Oscar Mother: Rose Children: Charlie, Madeline
Name: Hugo Father: Unknown Mother: Unknown Children: Oliver, Aurora	Name: Rose Father: Unknown Mother: Unknown Children: Oliver, Aurora, Jack
Name: Isla Father: Unknown Mother: Unknown Children: Luna, Arlo	Name: Oscar Father: Unknown Mother: Unknown Children: Luna, Jack, Arlo

To see how he is most directly related to each person, Bob wants to simplify this tree by creating a new set of cards marked with an asterisk, *, against the family member of each person that links them most directly to Bob. The problem this proposes is that he would like to find the shortest paths between himself and each person in the community so he can use these to mark these most direct links. If multiple people on a person's card have the same linking distance, he does not care which one is marked.

2.0. Instance Model

An instance of this proposed problem can be categorised by a set of dictionary representations of each card and the person to find most direct links for (e.g. Bob). Let D be the set of dictionaries (v, x, y, z) where v is the linking person's name (e.g. Alice), x is the linking person's father, y is the linking person's mother, and z is the set of the linking person's children – the linking person is our goal.

Using the person cards described in Section 1.0, this set is modelled by:

$$D = \{ \{Name: Alice, Father: Arlo, Mother: Madeline, Children: [None]\}, \{Name: Eve, Father: Oliver, Mother: Aurora, Children: [Bob]\}, \dots \}$$

This instance is modelled as (D, s) where s represents the target person we are getting the most direct links for (i.e. the starting person). For the problem presented in Section 1.0, if Bob is the starting person, the instance model is:

$$(\{\{Name: Alice, Father: Arlo, Mother: Madeline, Children: [None]\}, \{Name: Eve, Father: Oliver, Mother: Aurora, Children: [Bob]\}, \dots \}, Bob)$$

3.0. Solution Model

To mark the relation most directly linked to Bob on each person's card, we need to traverse through the family tree to find the shortest path that links to Bob. To do this, we begin by looking at Bob, then looking through each of his neighbouring links (i.e. relatives) at successively increasing distances until reaching the person we are linking to. Therefore, the solution will be a sequence of names $s, a, b, c, \ldots, g$ starting with the person, p, to find the direct links for and ending with the person, g, we were linking, and each name between are who link the two together, i.e. a, b, c, etc.

4.0. Problem Model

To model the familial links of the given instance (D, s), we may use a graph $G = (V, E)$. In this model, V is the set of vertices, and E is the set of edges. Here, the set V holds the names of everyone in the family tree set D, i.e.:

$$V = \{u : (u, v) \in D\} \cup \{v : (u, v) \in D\}$$

And the set E holds all the links between each person in D, i.e.:

$$E = D \cup \{(u, v) : (u, v) \in D\}$$

As all the people in the community are related, as stated in Section 1.0, most of the names will have interconnecting links, and therefore the graph to be connected.

To find the path from s to g, where g is the goal person, this graph can be traversed following a path in G between the two points. The linking distance between adjacent person cards is the same, so we only need to consider how many persons are in the path to determine the distance. The solution that is returned should be the shortest path determined between s and g; that is the sequence of steps (vertices, names) in the order they were traversed, starting at s and ending at g. This can then be used to mark the most direct path with an asterisk. Further analysis is not required to determine if other people on this card would have the same linking distance as, stated in Section 1.0, Bob does not care which path is marked if they are the same distance.

5.0. Solution Method

A *Breath First Search (BFS)* algorithm will be used to determine which is the shortest path between s and g. This search is achieved by retrieving the neighbours, nodes, from each person, starting at the person s (i.e. the stating person given), and progressing outwards to the next neighbours until the goal, g, is found. It then returns the sequence of neighbours, in order, that is the shortest distance to g.

The data to search through is the adjacency list of the graph, implemented as a dictionary, where each key is the node/vertex, and their values are the edges between.

To begin, we perform the BFS algorithm for each vertex in the set D. These vertices, v, are treated as the goal g each time until it is reached.

For each instance of v, the BFS algorithm first sets a queue Q to the set of s.

$$Q = \{\{s\}\}$$

Next, s and the instance of v would be compared to check they were not the same. If they are found to be different, it continues to visit all the neighbouring vertices (i.e. the persons father, mother, and children) of v until we find g.

To do this, the first path P needs to be returned from Q. Then, the last node n of this path P needs to also be returned.

If this node n has not already been explored and was not "Unknown" or "None", all the neighbouring vertices would be added to their own new paths. Each path would now look like:

$$P = \{s, n\}$$

Each of these new paths would then be added to the queue, where $P1$ is the first neighbour path, $P2$ is the second, and so on:

$$Q = \{P1, P2, \ldots\}$$

Once all the neighbour paths for n have been returned, the node needs to be marked as explored. Here, this node gets added to an explored array.

This sequence will keep going until a path that reaches the goal e is found. This path will be the shortest path between s and g.

To summarise, the solution to this problem can be from this instance below:

1. Get a vertex g from the set D.
2. Starting at vertex s, find all the neighbours and add these paths to a queue.
3. Set the vertex who has had the neighbours searched through to explored.
4. Continue to explore though each of the neighbours, neighbours until vertex g is found.
5. Return the path that first reached vertex g, this is the shortest path.