Exploring Programming In Ruby, Practical 7 - Marc Laffan (15202421)

Q 2. *The way sixdegrees.rb searches through the space of actors and films is ok, but probably not the most efficient. Can you propose two modifications to the program that might search in a different way. Just propose the algorithm, you don't have to implement it.*

The current tree traversal algorithm used by the program is known as the breadth-first searching algorithm. This is where neighbour elements are fully-explored before moving onto the neighbours of these nodes themselves. I will explore two alternatives to this algorithm; the depth-first and Monte Carlo searching algorithms.

Depth First

The depth-first algorithm approach could be thought of as the opposite of the breadth-first. Where the breadth-first algorithm explores the current set of elements before moving onto their neighbours, the depth-first algorithm takes the first element of a set then recursively explores the neighbours of that set. I say recursively because the neighbours first neighbour is also searched, following by that neighbours first neighbour and so on until the search reaches its lowest depth. From there, the previous element will have its remaining neighbours explored. This process is repeated all the way back to the original element, where the algorithm will continue on to the next element on the same level as the element whose depth has been fully searched. The pseudo-code for this algorithm can be seen below:

```
for each actor{
        while actor has children{
                store children in array
                while array is not empty{
                        check child
                }
        }
}
```

Monte Carlo

The Monte Carlo algorithm is somewhat similar to the depth-first algorithm, with the exception that the navigation to the lowest element depth is affected by the likelihood of each element being the goal. To put it in simpler terms, the algorithm will always "choose" the element most likely to be the element we are looking for (an actor/film match). The pseudocode for this algorithm can be seen below:

```
for each actor{
        while actor has children{
                store children in array
                while array is not empty{
                        if child is most probable {
                        check child
                        }
                }
        }
}
```