Igli Duro

Dr. William Hendrix

COT 4400

Project 3: Report

1. What type of graph would you use to model the problem input (detailed in the Section 3.1), and how would you construct this graph? (I.e., what do the vertices, edges, etc., correspond to?) Be specific here; we discussed a number of different types of graphs in class.

The type of graph I would use to model "Tarzan and Jojo" mazes would be as such. It would a simple, directed, and weighted, graph with its vertices labeled. Vertices will represent tiles in the maze. Each tile contains directional information of a vine, so edges will be one-directional. Since there are 8 possible movement directions vertices will be labeled with that direction. Each vine carries him 3 or 4 tiles, so the edges will be weighted with that distance. Lastly, the implementation of this graph will be using an adjacency list, as the dimensions of the maze can be quite large (max size of 1000 by 1000) and there also exist empty sections in the maze.

2. What algorithm will you use to solve the problem? Be sure to describe not just the general algorithm you will use, but how you will identify the sequence of moves Tarzan must take in order to reach the goal.

I would use a modified BFS algorithm to find a path from the initial vertex to the target vertex containing Jojo. It would function exactly as a BFS algorithm would but would have the added functionality of using a map to store value for the parent vertex using the child as a key. Once the BFS traversal is finished the algorithm will loop backwards from vertex containing Jojo to the initial vertex. Using the map to look up the parent and the loop would terminate after getting to the initial vertex as it is the root. The algorithm then returns a correct path from the initial vertex to the Jojo vertex. Pseudocode for this algorithm is below.

Input: Graph G

Input: u, v: initial and target vertex in G

Output: jojoPath: a vector containing a sequence of moves to get from u to v.

```
1.  Algorithm: tarzanBFS
2.  vector<bool> discovered;
3.  unordered_map<int,int> pmap;
4.  resize discovered to number of vertices and initialize to false;
5.  queue<int> queue;
6.  Push u onto queue and set discovered[u] = true;
7.  while(queue is not empty) do:
8.  |       i = queue.front();
9.  |       queue.pop();
10. |       getNeighbors(i);
11. |       for(all neighbors of i) do:
12. |       |       if (discovered[neighbor] == false) then;
13. |       |       |       push neighbor onto queue and set discovered[neighbor] = true;
14. |       |       |       pmap[neighbor] = i;
15. |       |       end
16. |       end
17. end
18. flag = true;
19. vector<int> jojoPath;
20. while (flag == true) do:
21. |       add v to jojoPath;
22. |       if (v is in pmap) then;
23. |       |       v = value from pmap.find(v);
24. |       else
25. |       |       flag = false;
26. |       end
27. end
28. reverse(jojoPath);
29. return jojoPath;
```