Algorithms Project 3 Report

Luis Henrique Baster Massud

Modeling the problem

Before you write a program to solve this problem, you will first write a report describing (in English and pseudocode) how you will solve this problem. This report should answer two basic questions:

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 graph used to model this problem input consists in a matrix that keeps the nodes in the positions from the original problem input and each node has a vector storing the possible nodes it could explore, directionally. The graph is constructed as the algorithm progresses, adding to the vector the nodes that it can access. There is also a stack to keep the solution path and be able to explore other ways if a node does not lead to the answer by popping the node from the stack. The edges are stored in the vector and the vertices are the structures of the nodes themselves containing the amount of “jump” they can provide.

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 Jim must take in order to reach the goal.

The algorithm used to solve this problem is going to be a DFS, using a stack of nodes that contain edges in a vector. It will be implemented using a stack to identify the sequence of moves Jim does to reach the goal like this:

JIMJUMPINGPSEUDO{

Create structure for node having its x, y positions and jump amount.

The structure also contains Booleans for marking the node as not a possibility for others nodes anymore and a vector to keep edges of this node.

Input from the file into a node that fills the matrix, adding to each of the nodes its x, y and jump amount (double for loop (matrix)).

Create a stack to keep track of the sequence of moves needed to reach goal.

Do while solution not found{

Mark starting node as not possible to visit.

Calculate where can the top node in stack can go (edges), add to the possibilities vector in the current node, ordering the top priority as right, down, left, up.

Condition to solution found

Check if possibilities vector for node current top of stack is empty, if it is, this node is not part of the solutions remove from stack and clear all possibilities from node in the top

Else push first possibility to the stack of possible solutions

}

Output file based on stack and its directions.

}