Assignment #2

Algorithm and Analysis COSC 2123/1285

Submission Deadline: 11:59pm Friday, 31st of May, 2019

Members/Student ID: Syed Hariz/S3701799, Simon Hesjevik/S3694451

Introduction

Dijkstra's Pathfinding Algorithm

Dijkstra's pathfinding algorithm is an algorithm that finds the most efficient path between a node and all other nodes in a graph. This can be used to find the shortest path between two points. The algorithm takes into consideration the edge weight between each node which can be used to represent different terrain types such as sand and grass as is stated in the assignment specification.

In this assignment we were asked to implement this algorithm to find the shortest path between two coordinates in a grid. This path can have waypoints that the object needs to reach on its way to the destination coordinate. The same principles in Dijkstra's algorithm applies in a grid context where each vertex has an edge to the adjacent vertex.

Dijkstra's algorithm is the same as BFS (breadth-first search) but each node can have other edge weights than 1. This makes it better for pathfinding in the real world as the edge weights can represent values such as distances. A good example of this is finding the shortest train trip from between suburbs in Melbourne. In this context each station would be a node and the length it takes between stations would be the edge weight.

Dijkstra's Pseudocode

CREATE a set of vertices (sptSet) that keeps track of the vertices you have already calculated

SET the distance values to all vertices to infinite

SET the distance to the start vertex to zero.

WHILE the set (sptSet) does not include all vertices

    SELECT a vertex v that does not have a minimum value

    ADD the vertex to the set

    SET value of v to the distance

    UPDATE distances to all adjacent vertices of v

Analysis

Representation

These set of tasks is implemented by using the pathmap class which is provided. The cost of each cell and its nearby cells location is stored in 2D arrays. A list if coordinated is the final output of the findpath() method which is determined using whether the list of input has waypoint or not. If the input has waypoints, then each waypoint is run through the standard shortest path logic which checks all the nearby neighbors for unvisited nodes and calculates the distance to the unvisited nodes to determine cost. It tends to avoid terrain paths by using their weights.

Task A

In this task we use the Dijkstra algorithm to find the shortest path from the origin to every cell in the map and based on that, figure out the shortest path to the destination. The algorithm involves returning the shortest path to the destination by looking up previous array for each coordinate.

Task B

For this task the cost of each cell which is taken as 1 is replaced by the terrain cost and we use that while calculating the distance for the cells and its neighbours.

Task C

To find the shortest path between any origin and any destination, we first run the Dijkstra algorithm with each origin to find the shortest paths from each origin to every cell in the map. We then construct every possible pairs of origin and destination and calculate the cost of path for each pair based on the shortest path data we collected in the previous step.

Task D

For this task:

- Find the path from source to first waypoint1
- Find path from waypoint1 to waypoint2
- Find path from waypoint2 to destination
- Return the shortest path and its cost

Data Structure

ArrayList

A set of array list is used to store the coordinates of each path, an ArrayList is implemented as a resizeable array. The main focus of this assignment is to add and remove coordinates that were stored in the ArrayList. Storing coordinates from a large grid can give an advantage by since it increases dynamically. Another reason why we have decided to follow this data structure is because ArrayList can dynamically add and remove elements. Since ArrayList uses an array, we can get faster access to the elements. Our implementation includes adding and removing a list of coordinates to find the shortest path from source to destination. Therefore, the algorithm is repeatedly adding and removing coordinates into an ArrayList to store the best path to destination.

Time complexity

add()	O(1)
remove()	O(n)
get()	O(1)

2D array

A 2D array is used to store the Coordinates that were explored and constantly update when the shortest path is found, this would simultaneously update the 2D array of cost from source to destination. Since this assignment provides a grid representation, we have to use 2D array to sync our algorithm with the map.

Time complexity

The time complexity of a 2D array is O(n). Say you have a 2D array of two-D; two-D[r][c]. The time complexity depends on the size of the array. Hence, it would be equal to O(*r*c*). The larger the size *n*, the higher the complexity of time.