Student name: Boyang YAN
Student number: 4329764
User name: by932

UNIVERSITY OF WOLLONGONG
AUSTRALIA

Assessment 3

* Overall

This project is about find the shortest path and second shortest path using a * algorithms. I have output the the total number of vertexes in the graph, start vertex, goal vertex, the shortest path with length of the path and the number of additional node in the solution tree, as well as the second shortest path with length of the path and the number of additional node in the solution tree.

I am using adjacency matrix representation of graph, if have the edge there are a edge weight will be story in matrix, others will be infinity number mean no edge between this two vertexes, when I have finished read all edge, I will created a undirected graph.

I am using an A * algorithms to find the shortest path and second shortest path.

I am using heap for find which path is the second shortest path.

*question answer

1. If we require that the second shortest path be longer than the shortest path?

The second shortest path must be longer than the shortest path when I first time run A * algorithm will find the shortest path. After when I delete each edge one by one in the shortest path and reused A * algorithm to find the new path shortest must longer than the shortest.

2. If the graph contains cycles?

Before adding a new node to the path array, I will check that the node is not already part of the path. So, this will solve contains cycles.

3. If the graph is undirected?

I have inited graph is undirected graph. At the beginning of create adjacency matrix. I have put both direction with weight. A * algorithms can work on directed graph and undirected graph.

* list of all of data structure used

I am using dynamic 2d array for adjacency matrix. At the beginning of read file, I am already know how many vertices we have got, so using dynamic array is better choice the memory is only allocated once and also can save memory compare with fixed size array..

I am using stack for calculating the shortest path from path record array.

This stack is represented by array can very fast pop and push.

* all of standard algorithms

I am using Manhattan distance for calculate two vertex distance.

Manhattan distance = abs(Vertex one.x - Vertex two.x) + abs(Vertex one.y - Vertex two.y)

In our case, using manhattan distance is better than using Euclidean distance, which is not a suitable search heuristic for this problem and it is a search heuristic, but it is not admissible.

Idea from https://www.cs.ubc.ca/~hutter/teaching/cpsc322/2-Search5.pdf website.

Euclidean distance = sqrt(pow(Vertex one.x - Vertex two.x, 2) + pow(Vertex one.y - Vertex two.y, 2))

I am using heap for find the second shortest path.

When I first get the shortest path, the heap size is number of edge in the shortest path. Each time I will delete one of edge and after run A * algorithm get new path and story into heap. For example, if the shortest path have 4 edges, There are 4 new shortest path will story into the heap. The heap node also record length of the shortest path, which the smallest value will on the top of heap. At the end, the second shortest path will be the top element in the heap.

Student name: Boyang YAN
Student number: 4329764
User name: by932

UNIVERSITY OF WOLLONGONG
AUSTRALIA

Why I have choice using A * algorithm for find the shortest path

A* search only expands a node if it seems promising. It only focuses to reach the goal node from the current node, not to reach every other nodes. This idea come from stackoverflow. It is optimal, if the heuristic function is admissible. The Dijkstra finds the shortest path from source to every other node by considering only real cost. So A * algorithm heuristic function is good to approximate the future cost, as a result need to explore a lot less nodes than Dijkstra. You can see I only cost 0 number of additional node for find the shortest path and cost 9 number of additional node for find the second shortest path. I have used Linear search for find the smallest value in the unselect set. In our case we only have 20 vertexes, so linear search is ok. If we have lots of vertexes, maybe using heap are more suitable.