

**Data Structures and Algorithms Assignment**  
**Implementation and Visualisation of A\* Algorithm**

**Due date: 5pm Friday 12<sup>th</sup> Feb 2016.**

**Deliverables**

Source code must be committed to your github account by 5pm Friday 12<sup>th</sup> February 2016. A signed declaration of originality must also be provided (at the end of this document). You will be required to demonstrate your project to me the following week.

**Part 1**

Implement the A\* algorithm described in Chapter 3: Lecture 17 (slides are available on blackboard). This should be added as a new member function to the Graph class as follows:

```
void aStar( Node* pStart, Node* pDest, void (*pProcess)(Node*),  
           std::vector<Node *>& path );
```

where:

`pStart` is a Graph node indicating the origin of the search

`pDest` is the goal vertex indicating the destination of the search

`void (*pProcess)(Node*)` is a function that shows the node currently being expanded.

`path` is a container (a vector) which holds the best path generated by the algorithm's completion

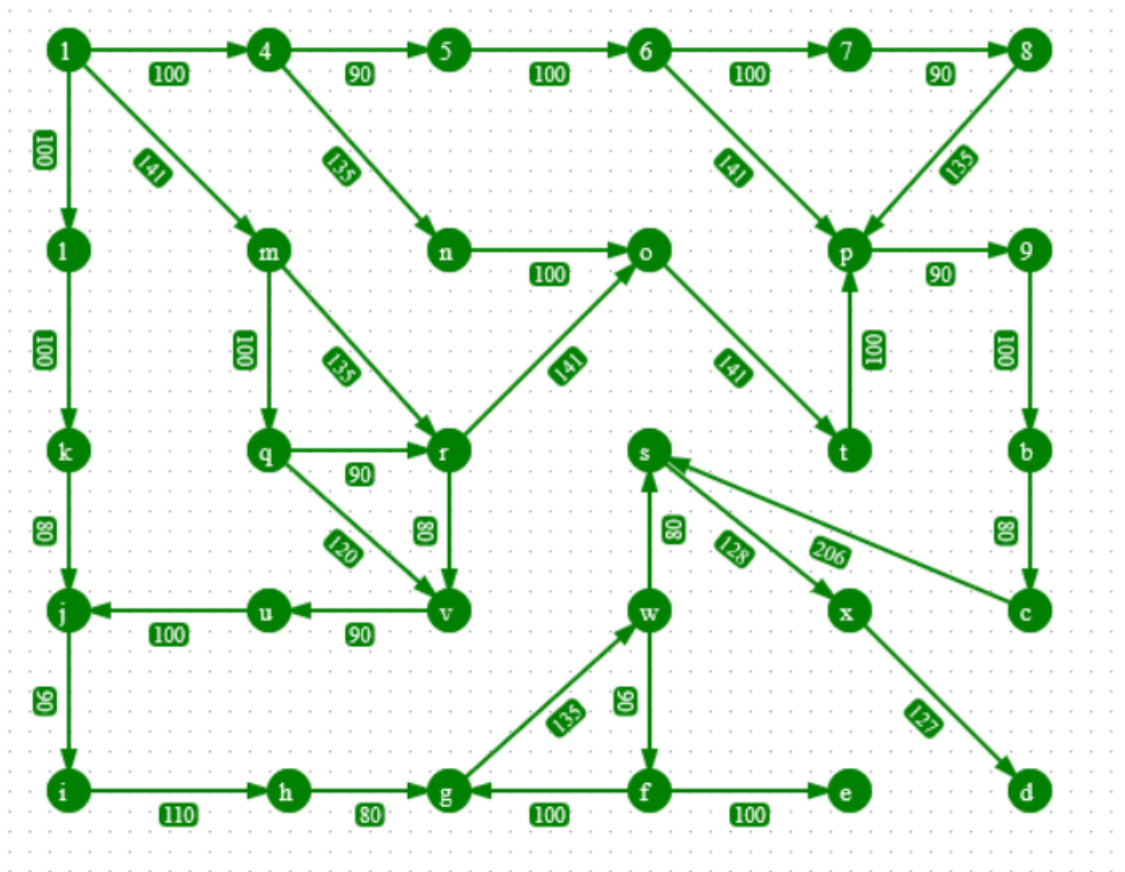
Note that each node now stores three pieces of information: a city name along with values for:

$h(n)$  = estimated cost to the goal from  $n$

$g(n)$  = cost so far to reach  $n$

Construct a graph with the topology as illustrated on the next page - note the graph is taken from here:

<http://mkorytin.blogspot.ie/2012/02/dijkstras-algorithm-html5-simulation.html>



Your implementation should be an undirected version of the above graph, so for example, if there is an arc from 'w' to 's', there is also an arc from 's' to 'w'.  
Note also that node 'l' is not repeated – the node underneath is letter '1'.

In order to test your algorithm, assume the desired route is from '1' to 'v'. You need to apply a heuristic to each node on this route, i.e. each node has to have an *estimate* of the path cost to 'v'.

To simulate this, in your main function, run the UCS algorithm with 'v' as the starting point and '1' as the destination. As you expand each node from 'v', compute  $h(n)$  as 90% of the path cost to reach  $n$ . For example:

Current	Neighbour	Actual Path Cost	Estimated path cost - $h(n)$ values
-----	-----	-----	-----
v	r	80	90% of 80 = 72
v	q	120	90% of 120 = 108
v	u	90	90% of 90 = 81
r	o	80+141	90% of (80 + 141) = 199
r	m	...	...
r	q	...	...

This will give us a list of  $h(n)$  values for each node on the path from '1' to 'v'.  
Remember the heuristic should always underestimate slightly.

Test your algorithm between any two vertices and check the results are correct before attempting part 2.

## Part 1 Marking Scheme (50% total)

Component	Weighting
Uses UCS to set the initial $h(n)$ values for each node	10
Initial $g(n)$ values set correctly for each node	5
Priority queue sorts nodes where node with lowest evaluation function $f(n)$ has highest priority	10
A child (neighbouring) node is <u>only</u> evaluated if its previous pointer is not equal to the node at the top of the priority queue (pq.top) and the child node is still in the priority queue.	5
A node's $f(n)$ value is only updated if a lower path cost is discovered (previous pointer field is also updated).	5
At end of each iteration, current node is removed from the queue and the node with the lowest $f(n)$ value is selected for expansion	5
Algorithm terminates if destination node not found (i.e. priority queue is empty) OR highest priority element is the destination node	5
Input vector holds best path generated by algorithm	5

## Part 2

Using the SFML library, draw your graph structure to approximate scale by using circles for nodes. For the above graph, consider the distances as pixel values. This is important as later the A\* heuristic will use the Euclidean distance between nodes. You may wish to add additional information to the nodes text file such that each node contains the screen coordinates of where it should be drawn, e.g.:

Node	Screen Coords	
-----	-----	
1	100,100	(top left corner of circle's bounding box)
2	200, 100	
..		
8	580, 100	

Inside each circle, show the name of each city, along with the current path cost i.e.  $g(n)$  (initially, this will be unknown).

Draw edges between nodes to show the relevant connections. Assume the graph is undirected, so a single edge is bi-directional. You may wish to use a similar approach for drawing the edges as for the nodes, i.e. store the screen coordinates in the arcs text file:

From	To	Weight	LineFrom	LineTo	
-----	-----	-----	-----	-----	
1	4	100	110, 105	200, 105	(assuming circle diameter = 10)

The weight (path cost) should be illustrated above each edge.

Allow the user select an origin node and a destination node. Use colouring or shading to highlight the selection - it should not be possible to select more than two nodes and the origin and destination must be different.

Once the origin and destination nodes are selected, compute the heuristic  $h(n)$  value for each node in the graph as the Euclidean distance from the centre of a node's circle to the centre of the destination node's circle. Display the  $h(n)$  value below the  $g(n)$  value at each node.

Provide a start button and a reset button. The start button will initiate the search, and show what nodes are expanded by A\* between the origin and destination (again highlight the relevant nodes using colour). As each node is expanded, show the updated  $g(n)$  values. Once the destination is found, set each node/circle to a specific colour along the optimal search path (if the destination is not reachable, do nothing).

The reset button will reset the graph state so the user can select new origin and destination nodes.

## Part 2 Marking Scheme (50% total)

Component	Weighting
Nodes visually represented as circles showing city names, $g(n)$ values.	5
Edges drawn with weight information shown and Graph drawn to approximate scale	5
User can select origin and destination nodes with visual confirmation	5
$h(n)$ values are calculated and shown for each node once the origin and destination nodes are selected.	5
A* runs when start button pressed, nodes that are expanded during the search are coloured and updated $g(n)$ values shown.	10
Once destination is reached, nodes along the optimal path are highlighted.	10
Reset button resets graph state to enable selection of new origin/destination nodes.	10

Each component will be graded 0 to 3 where:

- 0 means nothing of value;
- 1 means significant problems with the implementation;
- 2 minor problems with the implementation;
- 3 fully working implementation.

***Declaration on Originality***

DECLARATION

I declare that the work I am submitting for assessing by the Institute examiner(s) is entirely my own work, save for any portion(s) thereof where the author or source has been duly referenced and attributed.

I further state that I have read, and am familiar with, the Institute Examination and Assessment Regulations and that I am not, so far as I am aware, in breach of any of these regulations.

Student Name:.....Student ID Number:.....

Course of Study:.....

SIGNED: \_\_\_\_\_DATE: \_\_\_\_\_