

Robotics & Automation

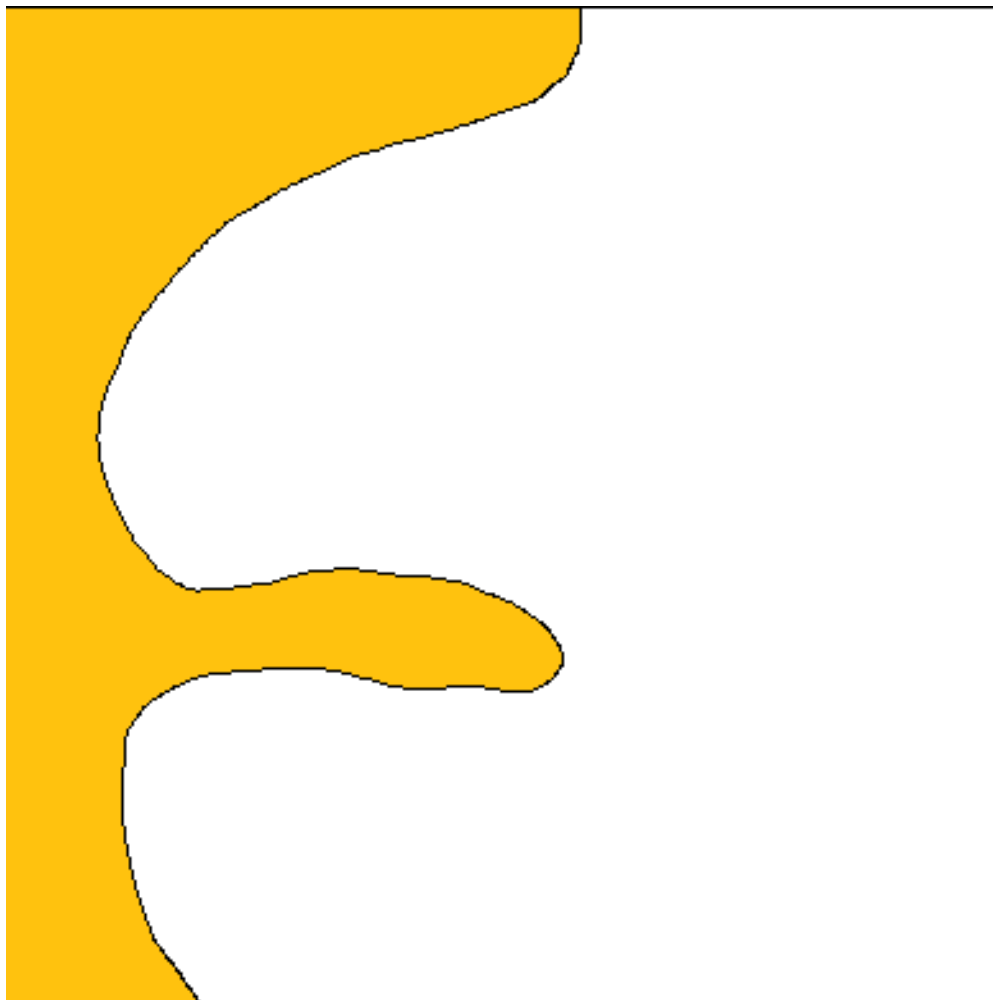
ENGT4314

Tutorial 5

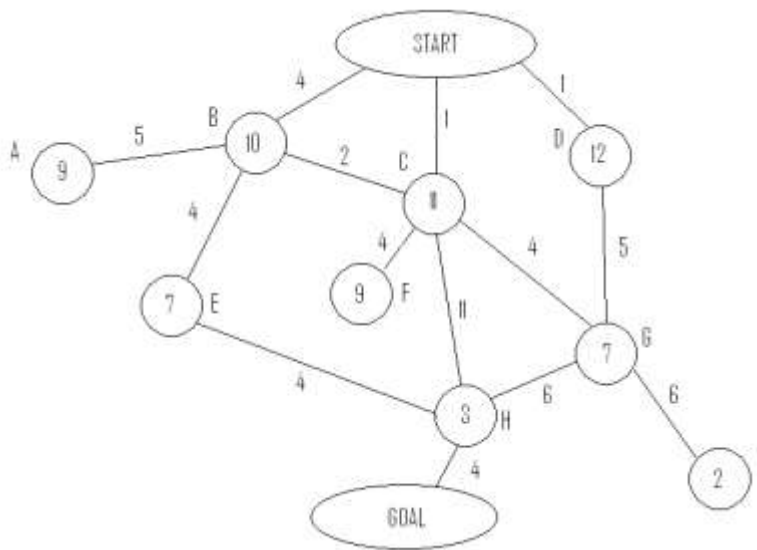
Quadtree

Subdivide an area until resolution limit or the subdivision is either completely unobstructed or obstructed. Then the centre of each final sub-division is a node, and you can then use A* for path planning, etc.

Example: (See recording)



A* Algorithm



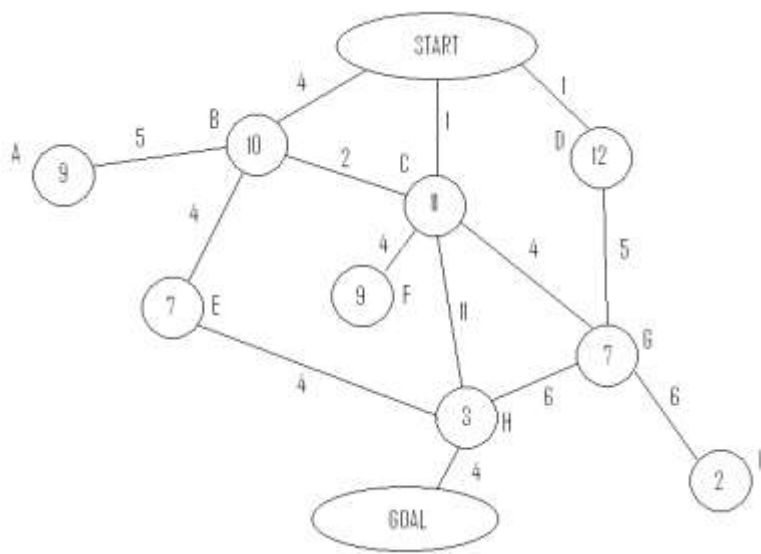
What you require to complete A*:

1. You need to maintain an open and closed set of nodes.
2. The cost from travelling from one node to another (usually length) in the above diagram this is the number above the line, of course you also need to which nodes are adjacent to each other node.
3. The heuristic cost for each node to the goal, which must be less than or equal to the actual cost to goal. Think of this as the distance to the goal as the crow flies.
4. To maintain a back pointer from each node to another node. This will eventually make up the path to goal.

The Process:

1. Select the 'Start' node.
2. Place all nodes that are adjacent to the selected node into the open set, except for closed nodes, and set the back pointer to the selected node.
3. Select the best node from the open set and place it in the closed set. The best node is defined as having the lowest cost, actual cost to start + heuristic to goal.
4. If we have selected the goal then **goto step 7**.
5. For each adjacent node to the selected node do the following.
 - a. If the node is in the closed set we ignore it and continue
 - b. If the node is not in the open set we add it to the open set, and set its back pointer to the selected node.
 - c. If the node is already in the open set we test to see if its current heuristic value + back-path length is larger than the new value. If so update the value and back point.
6. **Go to step 3.**
7. Follow the back pointer from the goal and work your way backwards to the start, this is the optimal path.

Example:



Using the same network, we perform the A* Algorithm on the network above to determine the optimum route.

To keep track of the require variable I will use the following format to record nodes.

A.B-xx where A is the node, B is the back pointer and xx is the heuristic value + path to start.

Open Set	Closed Set	Comment
B.S-14, C.S-12, D.S-13	S.S,	Adding adjacent nodes to the start.
B.C-13, D.S-13, F.C-14 , H.C-15, G.C-12	S.S, C.S, F.C	C was the best candidate (lowest heuristic + path) so we select it and add adjacent nodes. We also update Node B as it has a lower value through C. F can be ignored as it is a dead end.
B.C-13, D.S-13, H.G-14, I.G-13 ,	S.S, C.S,F.C, G.C, I.G	Selecting G, and add adjacent nodes
D.S-13, H.G-14, A.B-18 , E.B-15,	S.S, C.S, F.C, G.C, I.G, B.C, A.B	Selecting B and add adjacent nodes
H.G-14, E.B-15,	S.S, C.S, F.C, G.C, I.G, B.C, A.B, D.S	Selecting B – No Change
E.B-15, Goal.H-15	S.S, C.S, F.C, G.C, I.G, B.C, A.B, D.S, H.G	Selecting H
Goal.H-15	S.S, C.S, F.C, G.C, I.G, B.C, A.B, D.S, H.G, E.B	Selecting E – No Change
	S.S, C.S, F.C, G.C, I.G, B.C, A.B, D.S, H.G, Goal.H	Selecting Goal, Success...

So we now look at the Goal and find the optimal pass using the back pointer thus the optimal path is in reverse Goal -> H -> G -> C -> Start.

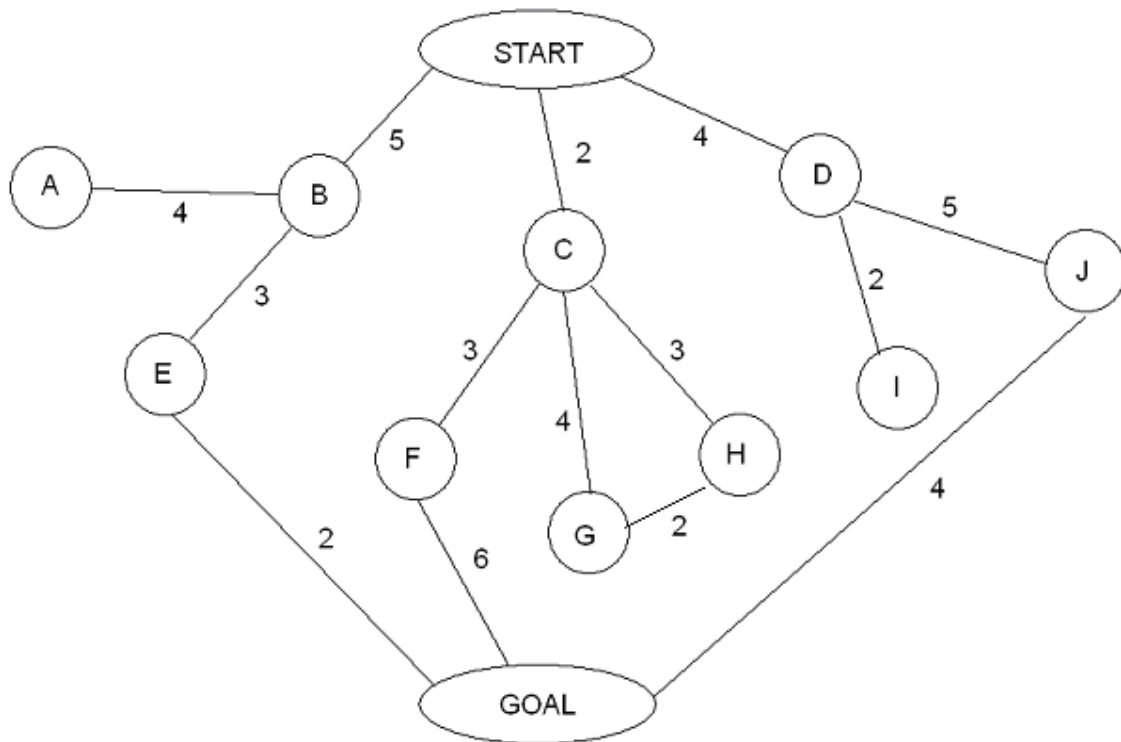
Dijkstra

Dijkstra's Algorithm is much the same as A* except without the heuristics, this means that we cannot ignore nodes in the closed set. Thus taking the procedure from before

The Process:

1. Select the 'Start' node.
2. Place all nodes that are adjacent to the selected node into the open set, except for closed nodes, and set the back pointer to the selected node.
3. Select the best node from the open set and place it in the closed set. The best node is defined as having the lowest cost, path to start.
4. If we have selected the goal then **goto step 7**.
5. For each adjacent node to the selected node do the following.
 - a. If the node is not in the open set we add it to the open set, and set its back pointer to the selected node.
 - b. If the node is already in the open/closed set we test to see if its current heuristic value + back-path length is larger than the new value. If so update the value and back point.
6. **Go to step 3**.
7. Follow the back pointer from the goal and work your way backwards to the start, this is the optimal path.

Example:



Open	Closed	
B.S-5, C.S-2, D.S-4	S.S-0,	
B.S-5, D.S-4, F.C-5, G.C-6, H.C-5	S.S-0,C.S-2	Selecting C
B.S-5, F.C-5, G.C-6, H.C-5, J.D-9, I.D-6	S.S-0,C.S-2,D.S-4, I.D-6,	Selecting D
F.C-5, G.C-6, H.C-5, J.D-9, A.B-9, E.B-8	S.S-0,C.S-2,D.S-4, I.D-6,B.S-5, A.B-9	Selecting B
G.C-6, H.C-5, J.D-9, E.B-8, Goal.F-11	S.S-0,C.S-2,D.S-4, I.D-6,B.S-5, A.B-9, F.C-5	Selecting F
G.C-6, J.D-9, E.B-8, Goal.F-11	S.S-0,C.S-2,D.S-4, I.D-6,B.S-5, A.B-9, F.C-5, H.C-5	Selecting H – No Change
J.D-9, E.B-8, Goal.F-11	S.S-0,C.S-2,D.S-4, I.D-6,B.S-5, A.B-9, F.C-5, H.C-5, G.C-6	Selecting G – No Change
J.D-9, Goal.E-10	S.S-0,C.S-2,D.S-4, I.D-6,B.S-5, A.B-9, F.C-5, H.C-5, G.C-6, E.B-8	Selecting E – Notice Goal Changed
Goal.E-10	S.S-0,C.S-2,D.S-4, I.D-6,B.S-5, A.B-9, F.C-5, H.C-5, G.C-6, E.B-8, J.D-9	Selecting J – No Change
	S.S-0,C.S-2,D.S-4, I.D-6,B.S-5, A.B-9, F.C-5, H.C-5, G.C-6, E.B-8, J.D-9, Goal.E-10	Selecting Goal -Finished

Thus the optimal path in reverse order is Goal <- E <- B <- S

PGM file access

PGM files store images as ASCII and are an easy way of reading / writing images for the purpose of generating a quadtree (Lab4).

You will need to add the `pgmimage.c` and `pgmimage.h` to your project; this code does all the IO and reads the images in to arrays for you. These will be available online in a few days.

To use make sure you include `pgmimage.h`.

`IMAGE img_open(char* file Name)` - opens an image and returns a pointer of type `IMAGE`.

`COLS(IMAGE* img)` returns the number of columns in the image

`ROWS(IMAGE* img)` returns the number of rows in the image.

`img_getpixel(IMAGE* img, int row, int col)` - Returns the pixel as an integer.