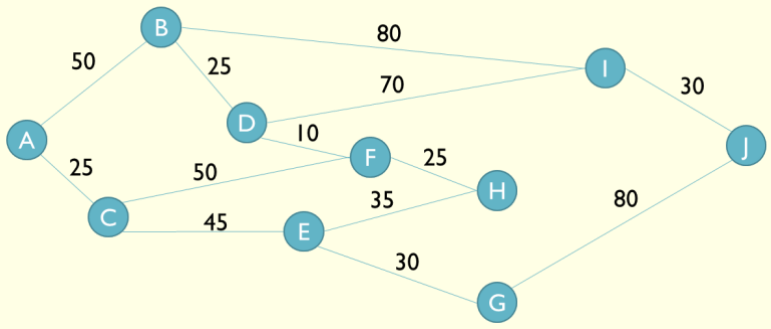
Shortest Path Algorithms

Dijkstra's shortest path algorithm

- The purpose of Dijkstra's algorithm finds the shortest path between nodes / vertices in a weighted graph.
- Selects the unvisited node with the shortest path.
- Calculates the distance to each unvisited neighbor
- Updates the distance of each unvisited neighbor if smaller
- Once all neighbours have been visited mark node as visited



Node	F = Distance from A	Previous Node
A (v)	0	-
B (v)	∞ 75	A
C (v)	∞ 10	A
D (v)	∞ 95 80	F B
E (v)	∞ 65	C
F (v)	∞ 40	C
G (v)	∞ 50	F
H (v)	∞ 90	G
I (v)	∞ 90	D
J (c)	∞ 135 105	E H

Shortest Path is A – C – F – G – H – J

Algorithm

#Next Current Node will be the node with the **lower F value**

Mark the start node as a distance of 0 from itself and all other nodes as an infinite distance from the start node.

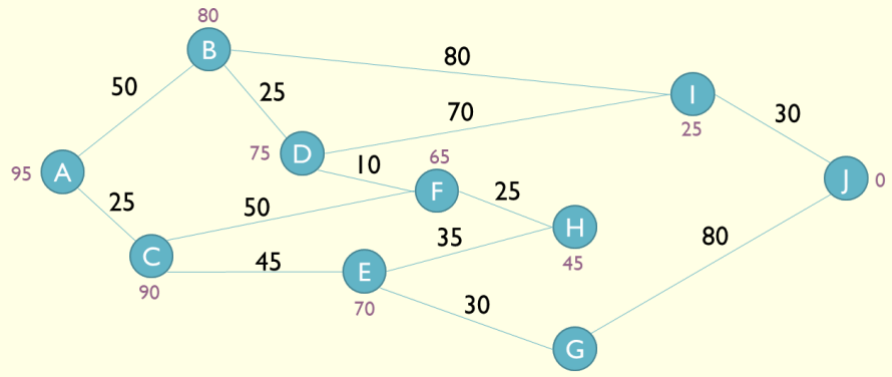
WHILE the destination node is unvisited:

Go to the closest **unvisited node** to A (initially this will be A itself)
Call this the current node.
FOR every unvisited node connected to current node
Calculate the distance to the current plus the distance of the edge to unvisited
If this distance is less than the currently recorded shortest distance, make it the
new shortest distance.
NEXT Connected node
Mark the current node as visited.
ENDWHILE

A*

The A* Algorithm is a general path-finding algorithm which is an improvement of Dijkstra's Algorithm. A heuristic is also used - usually approximate distance from a node to the final node .

This aims to make the shortest path finding process more Efficient and much quicker. How effective the A* algorithm is, however, depends largely on the accuracy of the heuristics used.



Node	Path Distance (g)	Heuristic Distance (h)	f=g+h	Previous Node
A	0	95	95	
B	50	80	130	A
C	25	90	115	A
D	75	75	150	B
E	70	70	140	C
F	75	65	140	C
G	100	50	150	E
H	100	45	145	F
I	130	25	155	B
J	180 160	0	180 160	G I

Shortest Path is A – C – F – G – H – J

A* Algorithm

#**Will not visit all nodes**. Will follow the route with the **lowest F value.** Stops when current node is the destination node

Begin at the start node and make this the current node.
WHILE the destination node is unvisited
FOR each open node directly connected to the current node.
Add to the list of open nodes
Add the distance from the start (g) to the heuristic estimate of the distance (h).
Assign this value (f) to the node.
NEXT connected node
Make the unvisited node with the **lowest F value** the current node
ENDWHILE

Summary

Heuristic helps produce a solution in a faster time
A* uses estimated distance from final node
Dijkstra uses a weight/distance

A* chooses which path to take next based on lowest current distance travelled.
A* does not have to visit all vertices to find a solution

Difference in programming complexity is minimal