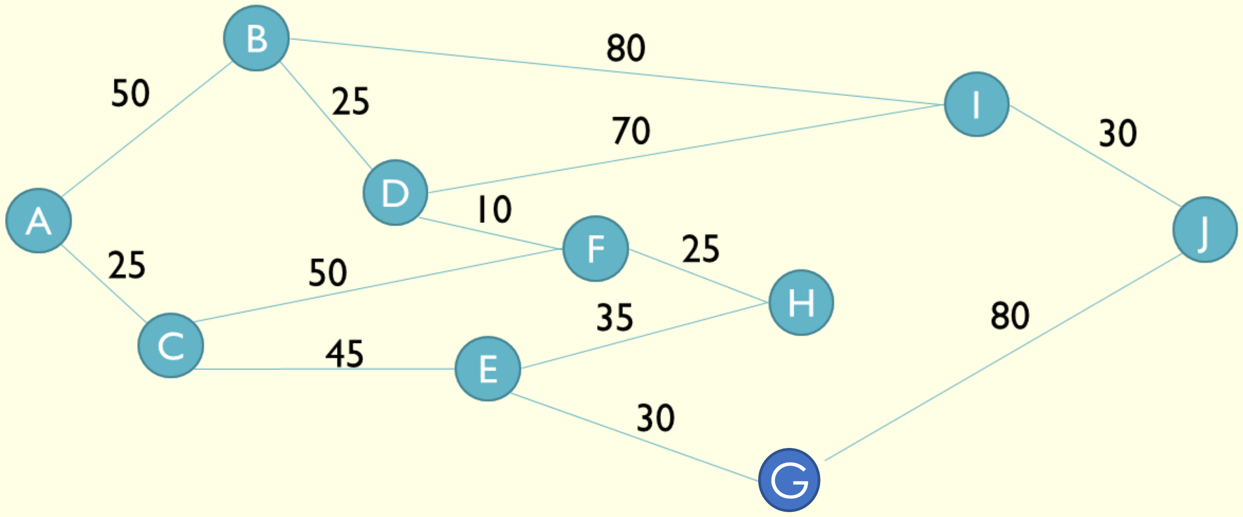
Learning Objectives

Dijkstra's shortest path algorithm

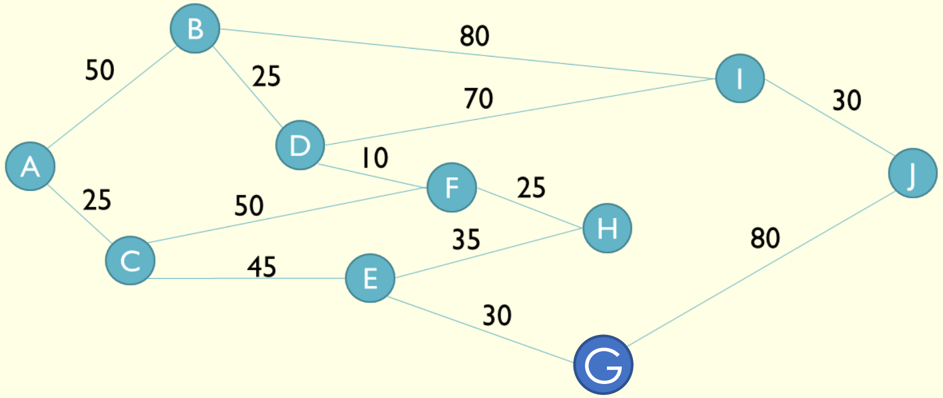
A* algorithm

Using the graph below, we shall use Dijkstra's algorithm to find the shortest path from A to J.
We begin with A as the "Current Node".



Node	Shortest distance from A	Previous Node
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	
G	∞	
H	∞	
I	∞	
J	∞	

- So we begin at A as the current node.
- B becomes = 50 and C = 25
- We mark A as visited and make the node with the shortest time current node – in this case C

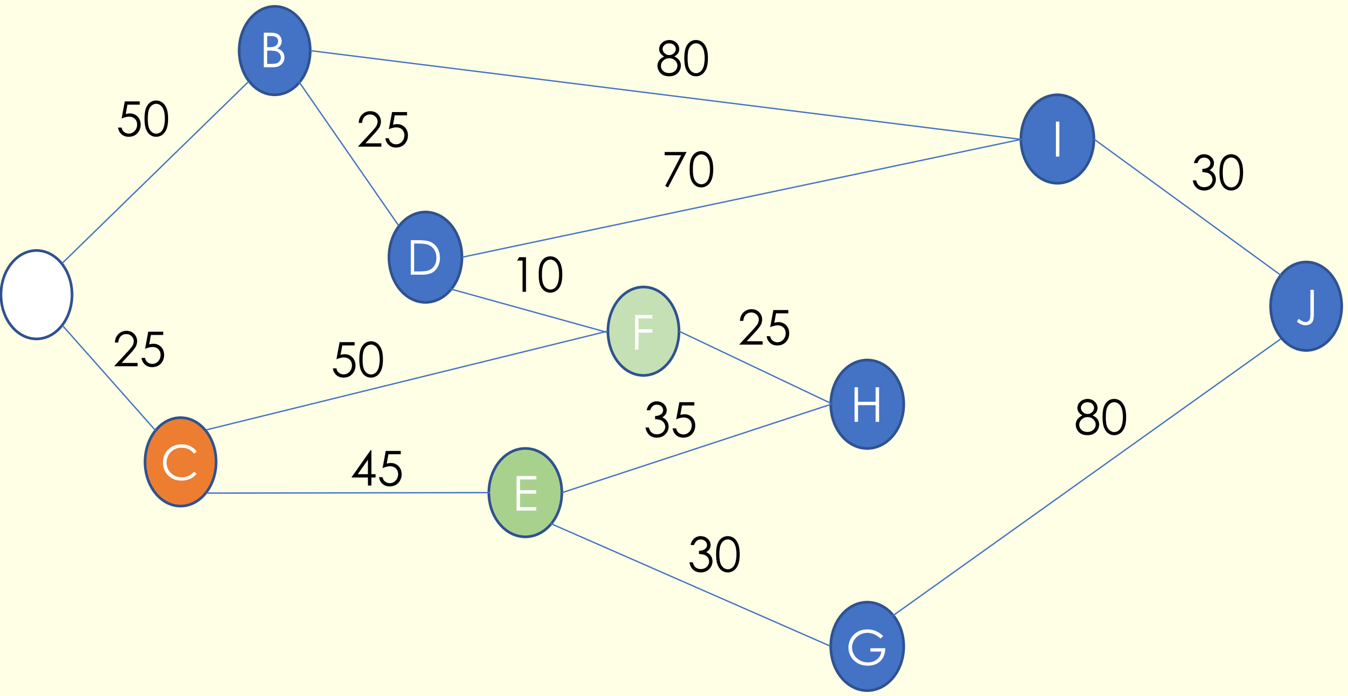


Node	Shortest distance from A	Previous Node
A (c)	0	
B	~~∞~~ 50	A
C	~~∞~~ 25	A
D	∞	
E	∞	
F	∞	
G	∞	
H	∞	
I	∞	
J	∞	

Visited: A

So now C is the current node we look at distance to E and F (remember the distance is from A)

- E is marked as 70 (25 + 45) as distance from A
- F is marked as 75 (25 + 50) as distance from A
- The closest unvisited node is now B so it is marked as current node

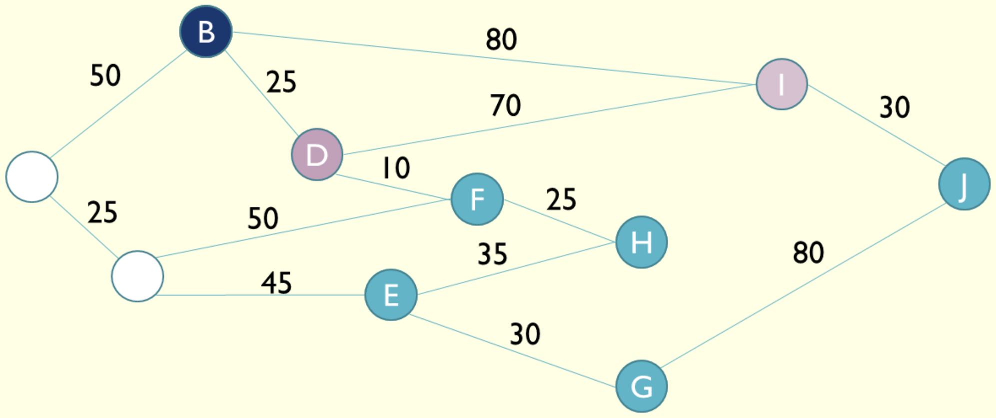


Node	Shortest distance from A	Previous Node
A (v)	0	
B	∞ 50	A
C (c)	∞ 25	A
D	∞	
E	∞ 70	C
F	∞ 75	C
G	∞	
H	∞	
I	∞	
J	∞	

Visited: A C

So now B is the current node we look at distance to I and D (remember the distance is from A)

- D is marked as 75 (50 + 25) as distance from A
- I is marked as 130 (50 + 80) as distance from A
- B will be marked as visited and we move to E

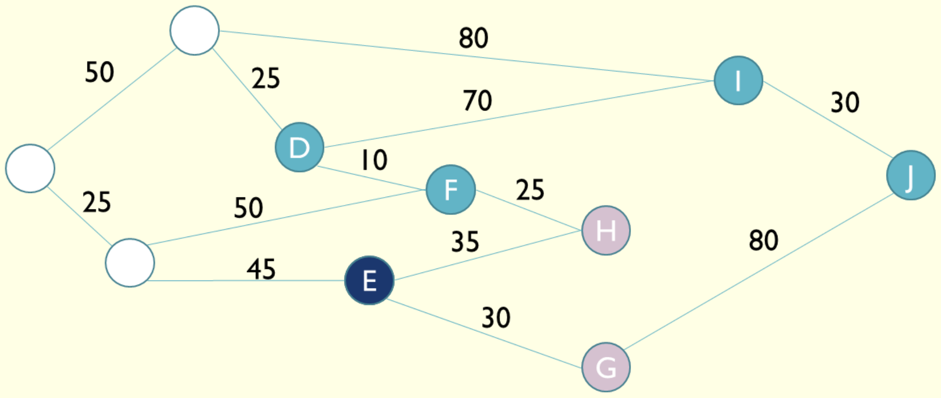


Node	Shortest distance from A	Previous Node
A (v)	0	
B (c)	∞ 50	A
C (v)	∞ 25	A
D	∞ 75	B
E	∞ 70	C
F	∞ 75	C
G	∞	
H	∞	
I	∞ 130	B
J	∞	

Visited: A C B

So now E is the current node we look at distance to H and G (remember the distance is from A)

- G is marked as 100 (25 + 45 + 30) as distance from A
- H is marked as 105 (25 + 45 + 35) as distance from A
- E will be marked as visited and we can move to D or F as they are both the new shortest – so start at D

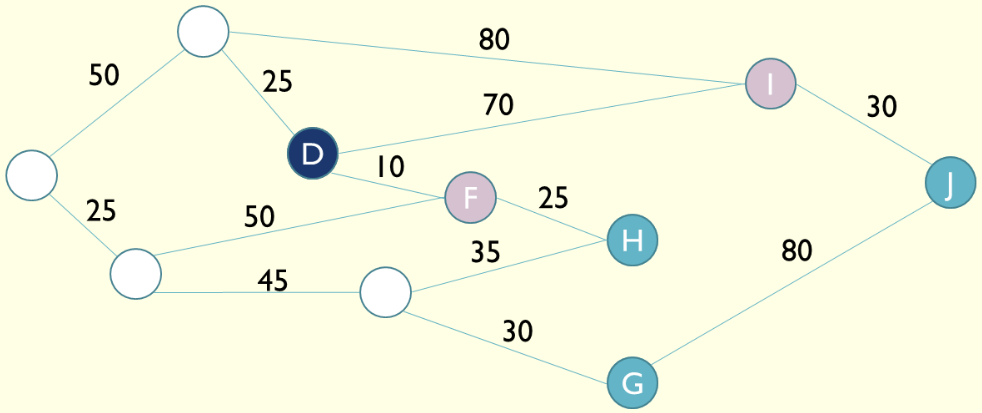


Node	Shortest distance from A	Previous Node
A (v)	0	
B (v)	∞ 50	A
C (v)	∞ 25	A
D	∞ 75	B
E (c)	∞ 70	C
F	∞ 75	C
G	∞ 100	E
H	∞ 105	E
I	∞ 130	B
J	∞	

Visited: A C B E

So now D is the current node we look at distance to I and F (remember the distance is from A)

- I From A via D is 75 + 70 = 145 this is higher than current value of I so no update is done.
- F from A via D is 75 + 10 = 85 which is higher than current value so F is not updated either.
- F now becomes the current node

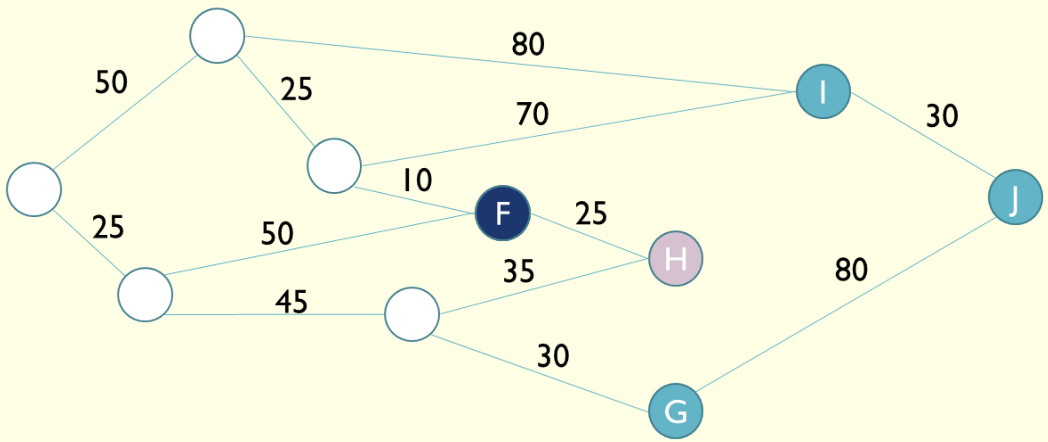


Node	Shortest distance from A	Previous Node
A (v)	0	
B (v)	∞ 50	A
C (v)	∞ 25	A
D (c)	∞ 75	B
E (v)	∞ 70	C
F	∞ 75	C
G	∞ 100	E
H	∞ 105	E
I	∞ 130	B
J	∞	

Visited: A C B E D

So now F is the current node we look at distance to H as it is the only connected node. (remember the distance is from A)

- H From A via F is 75 + 25 = 100 this value is lower than the current value of H so H is updated and the previous node is changed from E to F
- G or H could be the current node so again we start at the lowest alphabetically G

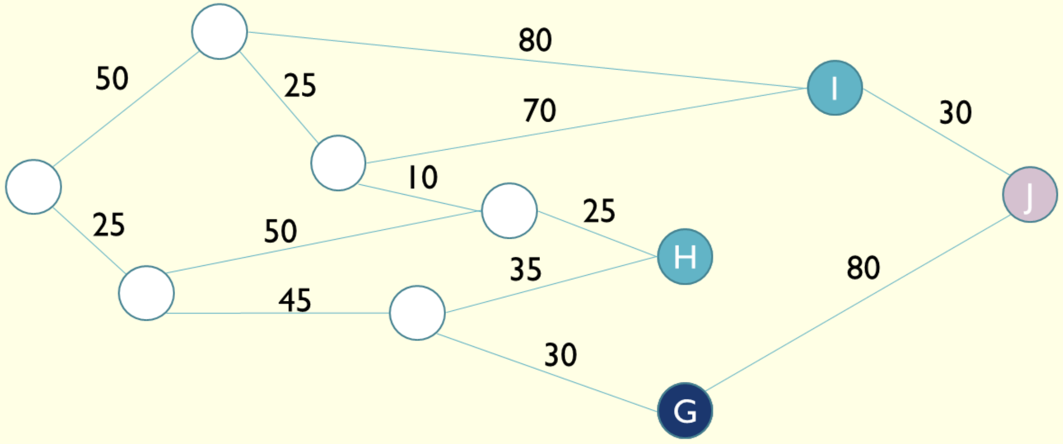


Node	Shortest distance from A	Previous Node
A (v)	0	
B (v)	∞ 50	A
C (v)	∞ 25	A
D (v)	∞ 75	B
E (v)	∞ 70	C
F (c)	∞ 75	C
G	∞ 100	E
H	∞ ~~105~~ 100	~~E~~ F
I	∞ 130	B
J	∞	

Visited: A C B E D F

So now G is the current node we look at distance to J as it is the only connected node. (remember the distance is from A)

- J From A via G is 100 + 80 = 180 J is updates with previous node being G
- H is now the current node

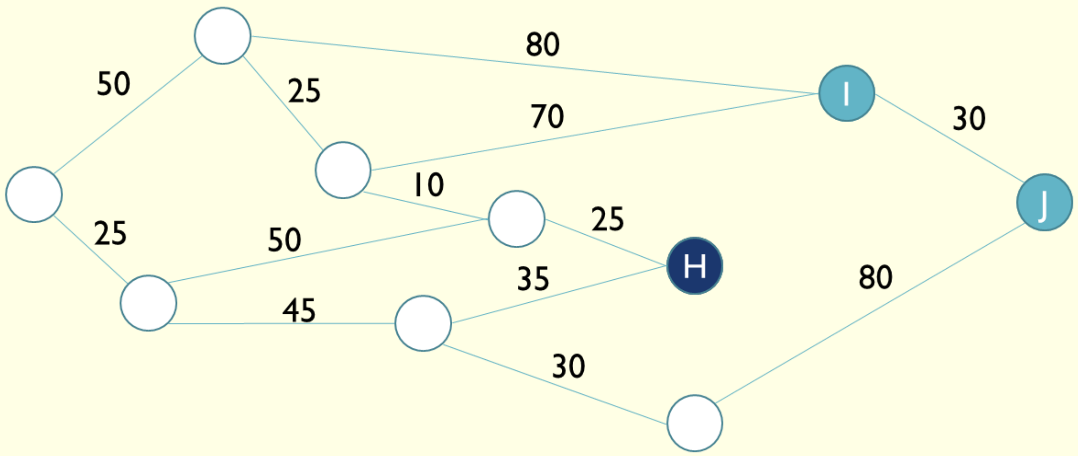


Node	Shortest distance from A	Previous Node
A (v)	0	
B (v)	∞ 50	A
C (v)	∞ 25	A
D (v)	∞ 75	B
E (v)	∞ 70	C
F (v)	∞ 75	C
G (c)	∞ 100	E
H	∞ ~~105~~ 100	~~E~~ F
I	∞ 130	B
J	∞ 180	G

Visited: A C B E D F G

As H has no connected nodes we simply set it to visited.

- I becomes the current node.

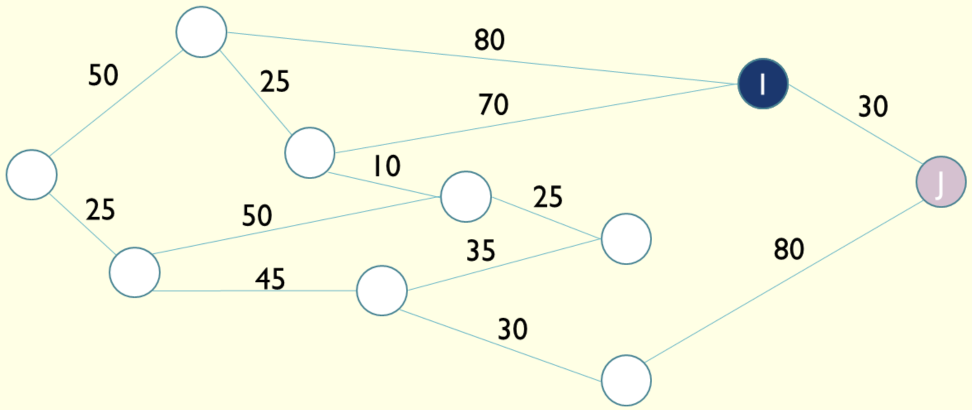


Node	Shortest distance from A	Previous Node
A (v)	0	
B (v)	∞ 50	A
C (v)	∞ 25	A
D (v)	∞ 75	B
E (v)	∞ 70	C
F (v)	∞ 75	C
G (v)	∞ 100	E
H (v)	∞ ~~105~~ 100	~~E~~ F
I	∞ 130	B
J	∞ 180	G

Visited: A C B E D F G H

With I as the current node we look at the distance to J via I.

- J via I is 130 + 30 = 160
- As 160 is lower than current value we update as follows.
- And we set J to current node.



Node	Shortest distance from A	Previous Node
A (v)	0	
B (v)	∞ 50	A
C (v)	∞ 25	A
D (v)	∞ 75	B
E (v)	∞ 70	C
F (v)	∞ 75	C
G (v)	∞ 100	E
H (v)	∞ ~~105~~ 100	~~E~~ F
I (c)	∞ 130	B
J	∞ ~~180~~ 160	~~G~~ I

Visited: A C B E D F G H I

- So we have now we know the time shortest time from A to J is 160.
- We now need to look at the path – Previous node to J is I, from I is B and B is A.
- So the shortest path is A, B, I J.

Node	Shortest distance from A	Previous Node
A (v)	0	
B (v)	∞ 50	A
C (v)	∞ 25	A
D (v)	∞ 75	B
E (v)	∞ 70	C
F (v)	∞ 75	C
G (v)	∞ 100	E
H (v)	∞ ~~105~~ 100	~~E~~ F
I (v)	∞ 130	B
J (c)	∞ ~~180~~ 160	~~G~~ I

Visited: A C B E D F G H I J

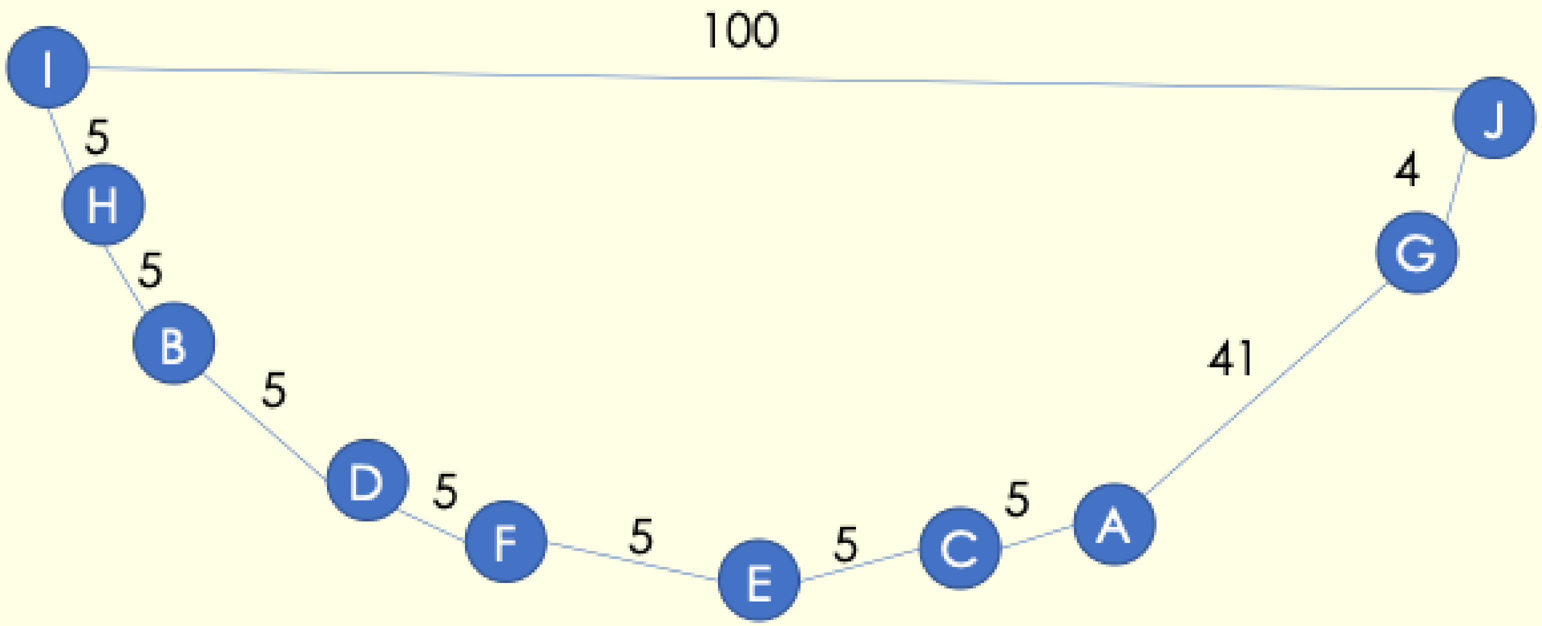
Dijkstra's Algorithm

- The algorithm is as follows:

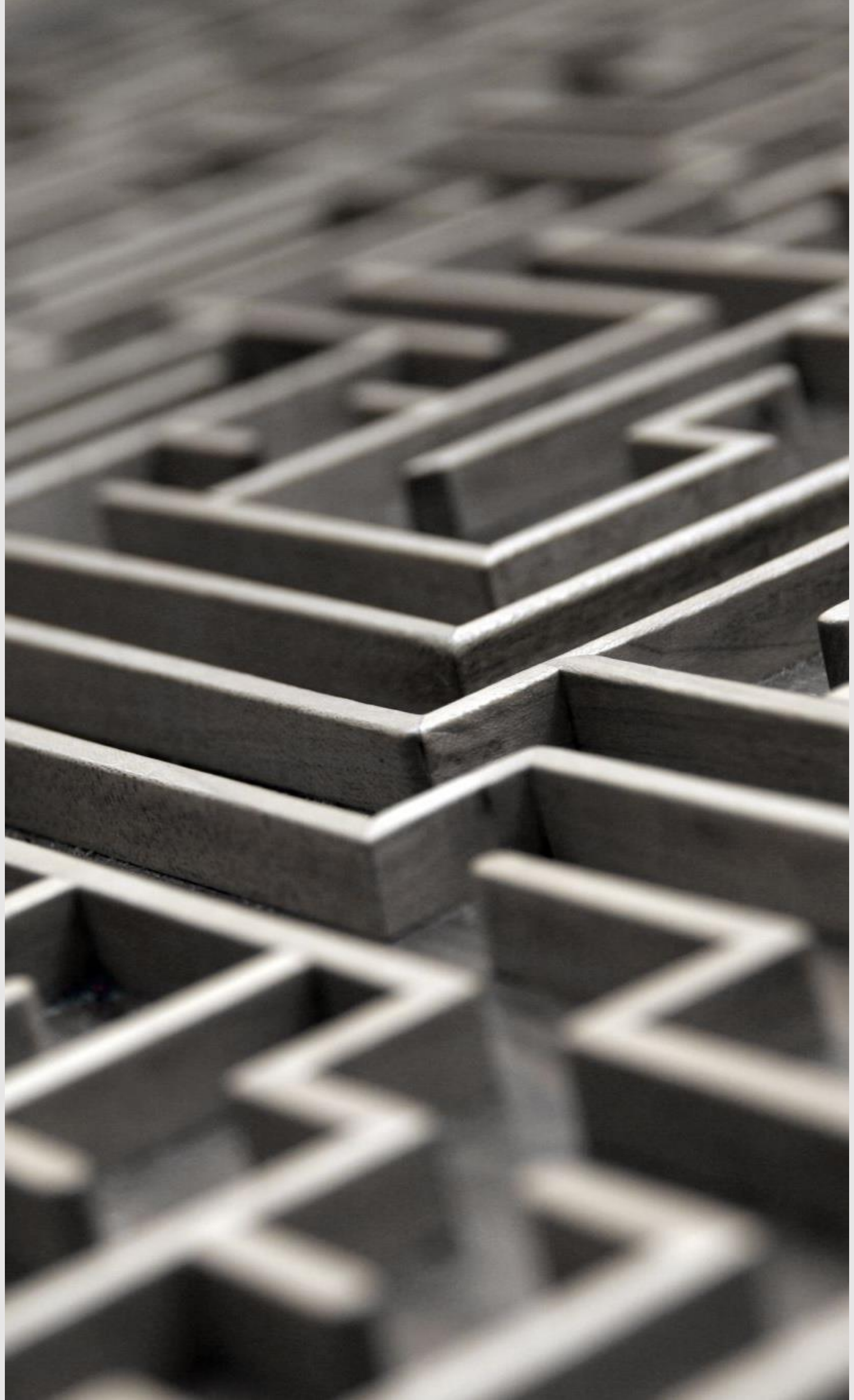
```
Mark the start node as a distance of 0 from itself and all
other nodes as an infinite distance from the start node.

WHILE the destination node is unvisited:

   Go to the closest unvisited node to A (initially this will
   be A itself) and call this the current node.

   FOR every unvisited node connected to the 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
```

Dijkstra’s Algorithm

- Of course that works so long as it’s not obvious.



- Here you can see the shortest route is from A G to J – 45.
- But using Dijkstra’s Algorithm, you would need to visit every node to work that out.



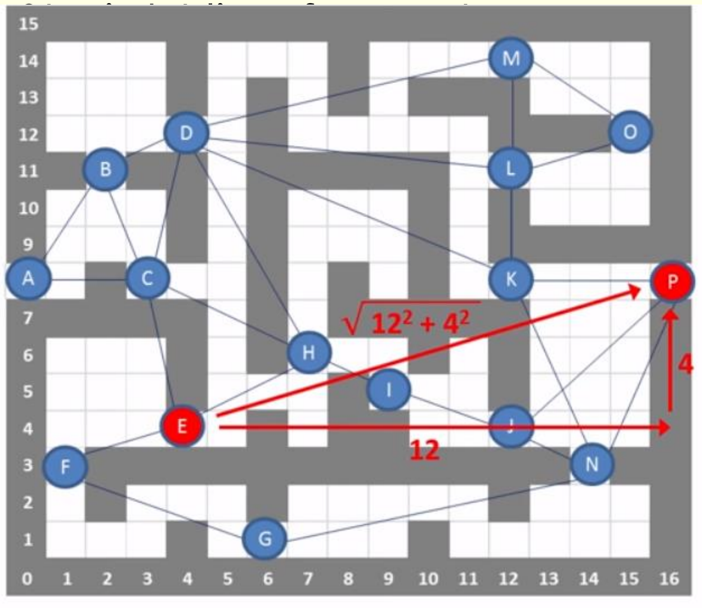
A* Search

- The A* (or A Star) Search is an alternative algorithm that can be used for finding the shortest path.
- It performs better than Dijkstra's algorithm because of its use of **heuristics.**
- Heuristics is when existing experience is used to form judgement, the so called "rule of thumb".

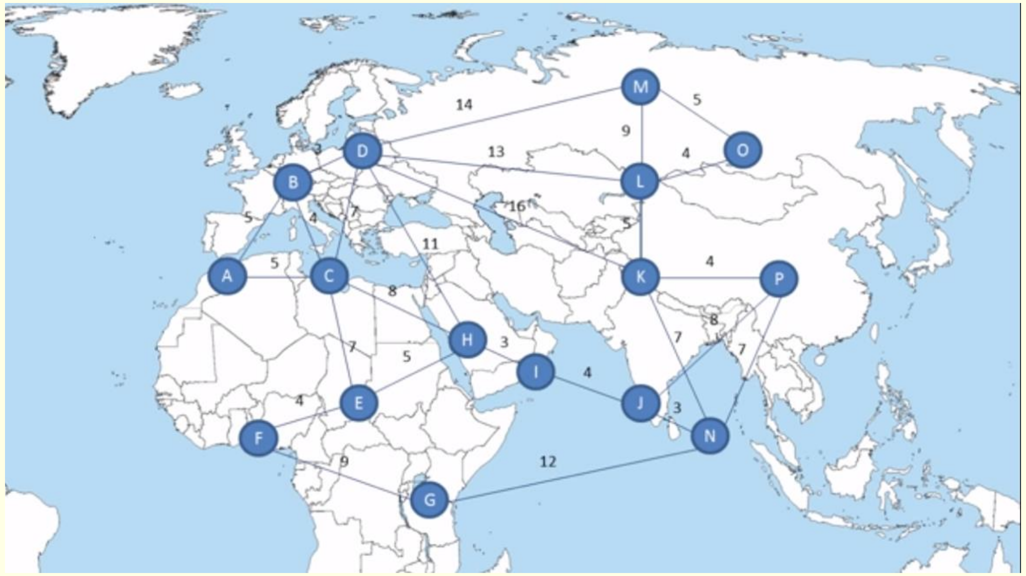
Heuristics

- Depend on the problem
- Working out an estimated distance from a node to the end node

A maze could use the Manhattan Distance
Number of steps (12 + 4)
Using x, y coordinates
Or Euclidean Distance as a heuristics

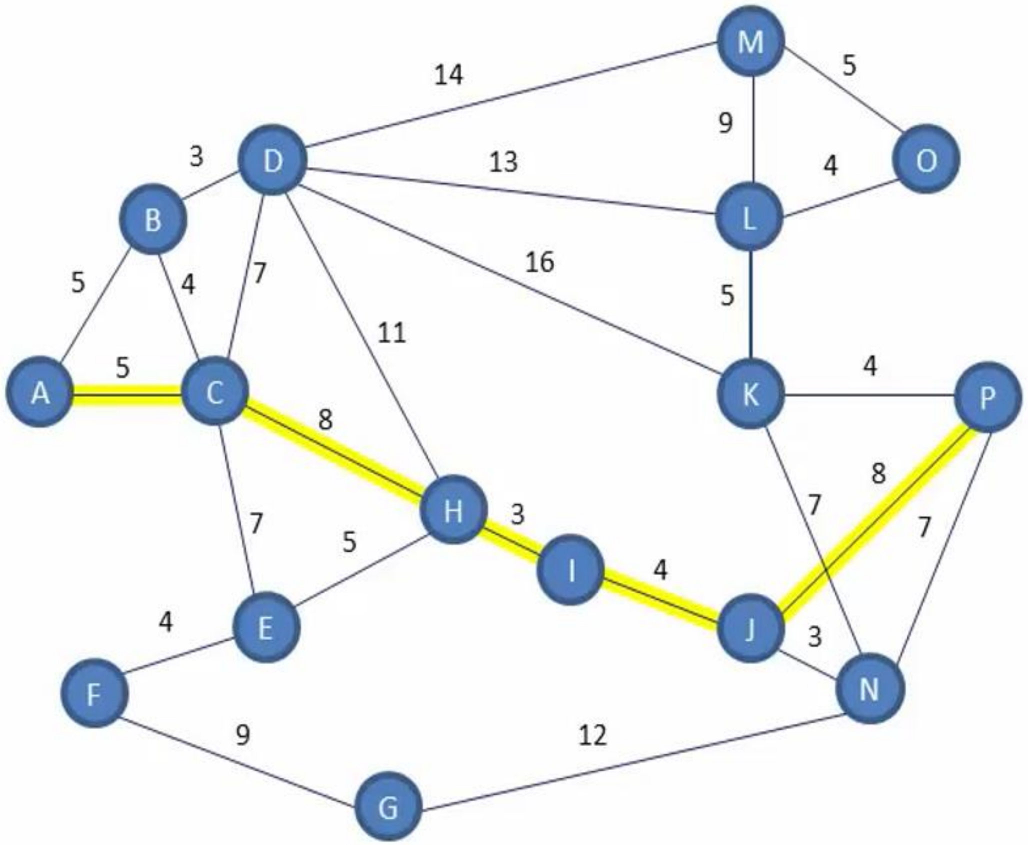


On a map the heuristic distance is in kilometres from a node to the end node
Using latitude and longitude



Open E M L K J P N

Closed A C B H D I



Vertex	Distance from A (g)	Heuristic distance (h)	f = g + h	Previous vertex
A	0	16	16	
B	5	17	22	A
C	5	13	18	A
D	8	16	24	B
E	12	16	28	C
F		20		
G		17		
H	13	11	24	C
I	16	10	26	H
J	20	8	28	I
K	24	4	28	D
L	21	7	28	D
M	22	10	32	D
N	23	7	30	J
O		5		
P	28	0	28	J

A* Search

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

The Pseudocode for a the A* Search looks like this:

```
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
```

A* Algorithm

```
Initialise open and closed lists
Make the start vertex current
Calculate heuristic distance of start vertex to destination (h)
Calculate f value for start vertex (f = g + h, where g = 0)
WHILE current vertex is not the destination
    FOR each vertex adjacent to current
        IF vertex not in closed list and not in open list THEN
            Add vertex to open list
            Calculate distance from start (g)
            Calculate heuristic distance to destination (h)
            Calculate f value (f = g + h)
            IF new f value < existing f value or there is no existing f value THEN
                Update f value
                Set parent to be the current vertex
            END IF
        END IF
    NEXT adjacent vertex
    Add current vertex to closed list
    Remove vertex with lowest f value from open list and make it current
END WHILE
```

Dijkstra Vs A*

- 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

Summary

- A* has a wise range of applications
- A* finds the shortest path between two vertices
- A* does not have to visit all vertices, ideally
- A* picks the most promising looking node next
- The better the heuristic, the quicker A* finds the path
- Heuristic is problem specific
- Open nodes known as "the fringe" or "the frontier"
- List of open nodes can be implemented as a priority queue (so next node with the smallest F value is chosen next.)
- Each node on the path keeps track of the one that came before it