深蓝学院
shenlanxueyuan.com

Autonomous Vehicle Planning and Control

Wu Ning

深蓝学院
shenlanxueyuan.com

Session 8

Vehicle Mission Planning (Route Planning)

Outline

Route Planning/Mission Planning

- Route Planner concept
- Functionality and Challenges

Common method

- Dijkstra based approaches
- A* based approaches

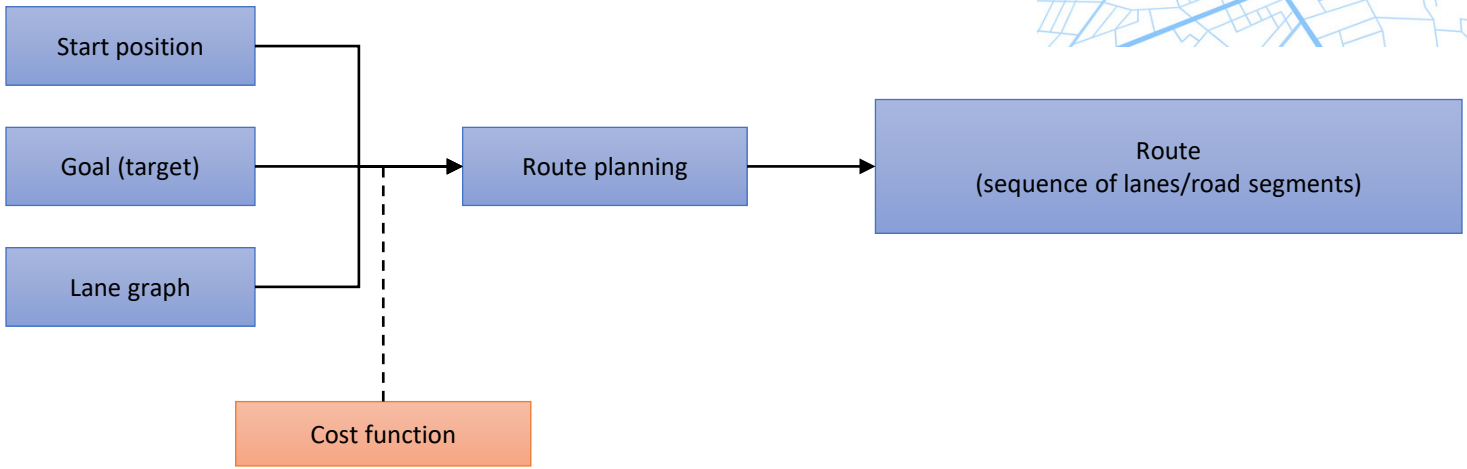
Cost arrangement

Mission Planning

Functionality of mission planning: Find preferred route (or routes) over the road network

Common approach: Graph search (Dijkstra, A*, et al)

- Many algorithms shared with local planning methods

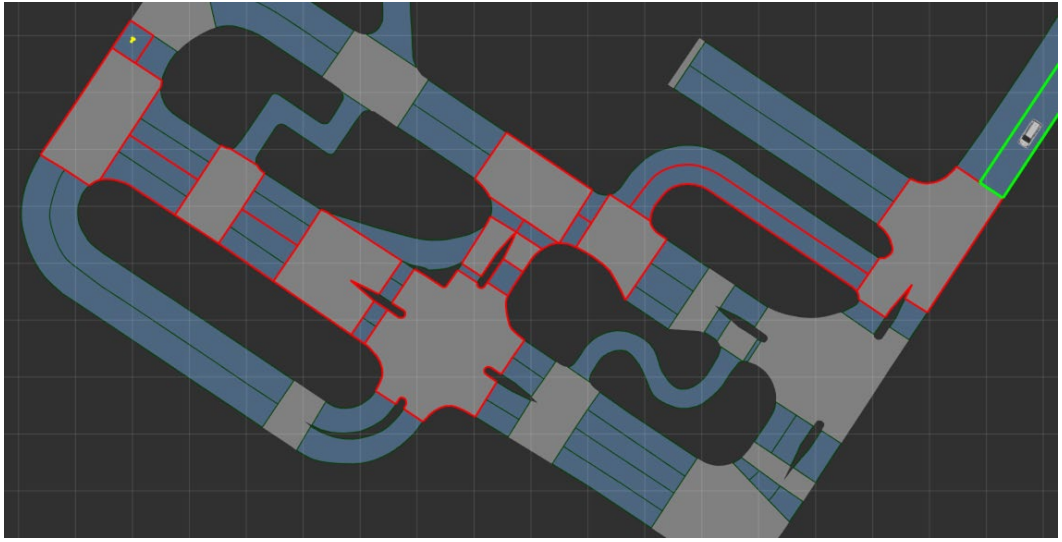


Mission Planning

Example: Car at start, yellow arrow goal, route LaneGroups/intersections highlighted

Design considerations:

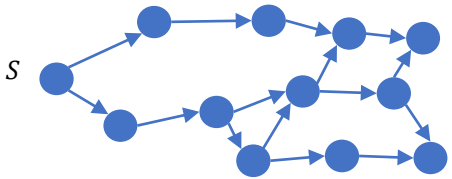
- Edge/Node definition - could plan for lane/connector route rather than LaneGroup/intersection
- Cost function could penalize lane changes, turns (left or right), narrow lanes, etc.
- Interface to Behavior/Local Planners - intermediate goal, or cost to goal?

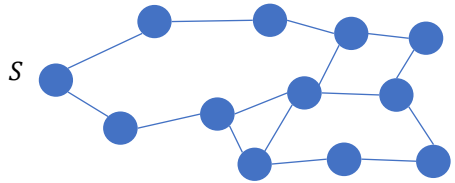


Dijkstra Algorithm

Mission Planning: Dijkstra

Problem:
You are given a directed or undirected weighted graph with n vertices and m edges. The weights of all edges are non-negative. You are also given a starting vertex S. This method discusses finding the lengths of the shortest paths from a starting vertex S to all other vertices, and output the shortest paths themselves.
This problem is also called **single-source shortest paths problem**.





Basics of Dijkstra's Algorithm

- Dijkstra's Algorithm basically starts at the node that you choose (the source node) and it analyzes the graph to find the shortest path between that node and all the other nodes in the graph.
- The algorithm keeps track of the currently known shortest distance from each node to the source node and it updates these values if it finds a shorter path.
- Once the algorithm has found the shortest path between the source node and another node, that node is marked as "visited" and added to the path.
- The process continues until all the nodes in the graph have been added to the path. This way, we have a path that connects the source node to all other nodes following the shortest path possible to reach each node.

Mission Planning: Dijkstra

Let's create an array $d[\,]$ where for each vertex v we store the current length of the shortest path from s to v in $d[v]$. Initially $d[s] = 0$, and for all other vertices this length equals infinity. In the implementation a sufficiently large number (which is guaranteed to be greater than any possible path length) is chosen as infinity.

$$d[v] = \infty, v \neq s$$

In addition, we maintain a Boolean array $u[\,]$ which stores for each vertex v whether it's marked. Initially all vertices are unmarked:

$$u[v] = false$$

The Dijkstra's algorithm runs for n iterations. At each iteration a vertex v is chosen as unmarked vertex which has the least value $d[v]$.

Evidently, in the first iteration the starting vertex s will be selected.

Mission Planning: Dijkstra

The selected vertex v is marked. Next, from vertex v **relaxations** are performed: all edges of the form (v, to) are considered, and for each vertex to the algorithm tries to improve the value $d[to]$. If the length of the current edge equals len, the code for relaxation is:

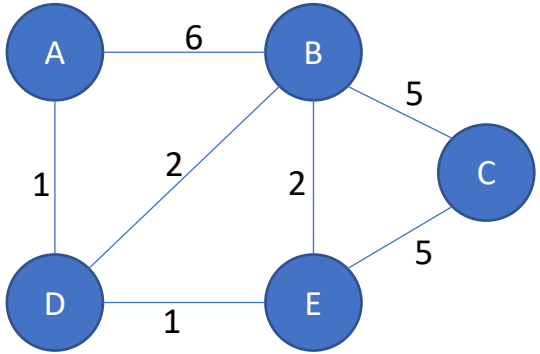
$$d[to] = \min(d[to], d[v] + len)$$

After all such edges are considered, the current iteration ends. Finally, after n iterations, all vertices will be marked, and the algorithm terminates. We claim that the found values $d[v]$ are the lengths of shortest paths from s to all vertices v.

Note that if some vertices are unreachable from the starting vertex s, the values $d[v]$ for them will remain infinite. Obviously, the last few iterations of the algorithm will choose those vertices, but no useful work will be done for them. Therefore, the algorithm can be stopped as soon as the selected vertex has infinite distance to it.

Dijkstra's Shortest Path Algorithm

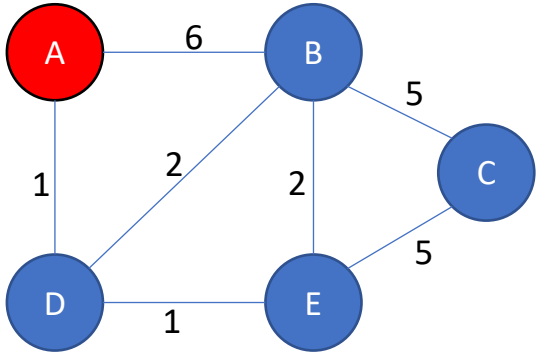
- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited



Vertex	Shortest Distance from A	Previous Vertex
A		
B		
C		
D		
E		

Dijkstra's Shortest Path Algorithm

- **Let distance of start vertex from start vertex = 0**
- **Let distance of all other vertices from start = inf**
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

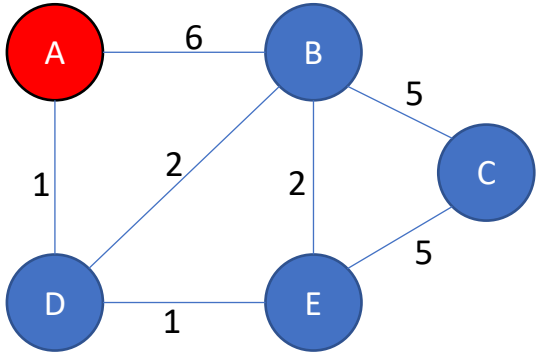


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited=[ ], unvisited = [A,B,C,D,E]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - **Visit the unvisited vertex with the smallest known distance from the start vertex**
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

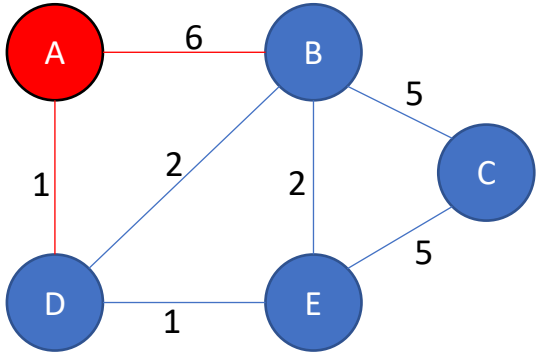


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited=[ ], unvisited = [A,B,C,D,E]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - **For the current vertex, examine its unvisited neighbours**
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited



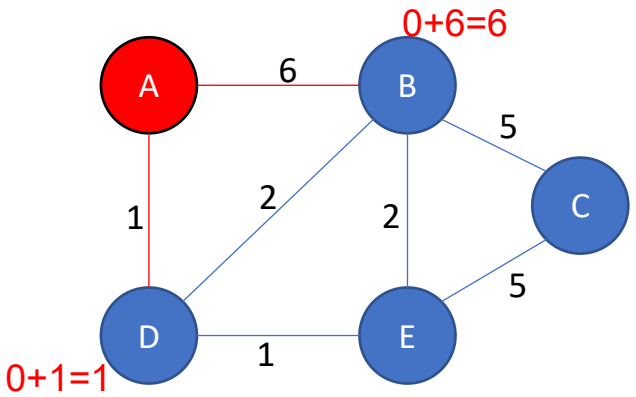
Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited=[ ], unvisited = [A,B,C,D,E]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - **For the current vertex, calculate distance of each neighbour from start vertex**
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

Visited=[ ], unvisited = [A,B,C,D,E]

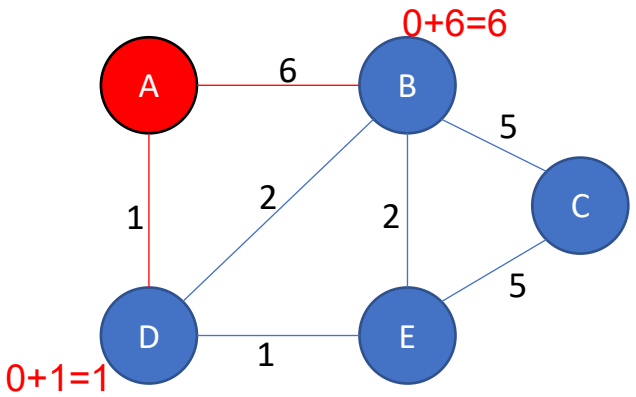


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - **If the calculated distance of a vertex is less that the known distance, update the shortest distance**
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

Visited=[ ], unvisited = [A,B,C,D,E]

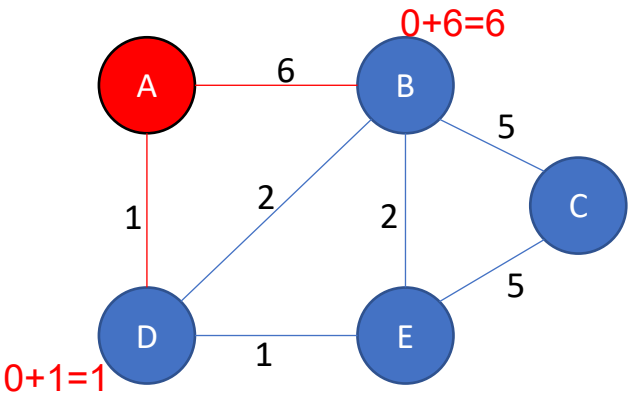


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - **If the calculated distance of a vertex is less that the known distance, update the shortest distance**
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

Visited=[ ], unvisited = [A,B,C,D,E]

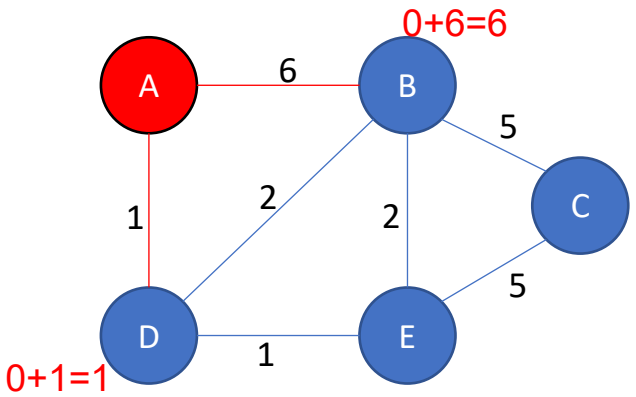


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	6	
C	∞	
D	1	
E	∞	

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - **Update the previous vertex for each of the updated distances**
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

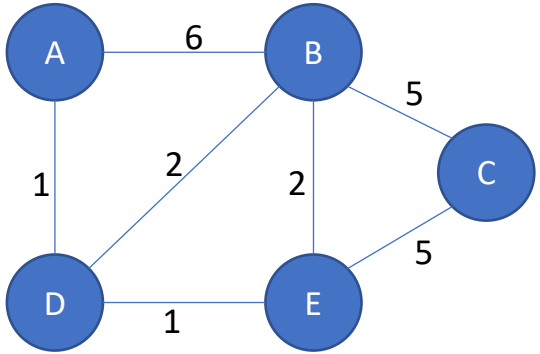
Visited=[ ], unvisited = [A,B,C,D,E]



Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - **Add the current vertex to the list of visited vertices**
- Until all vertices visited

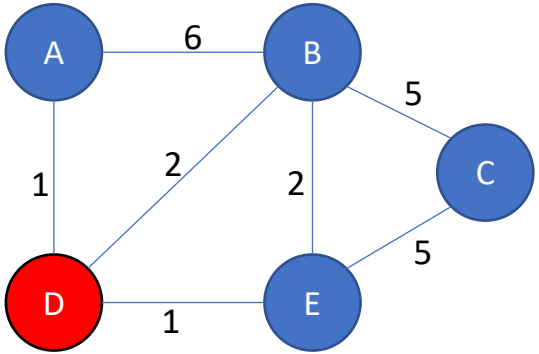


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited=[A], unvisited = [B,C,D,E]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - **Visit the unvisited vertex with the smallest known distance from the start vertex**
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

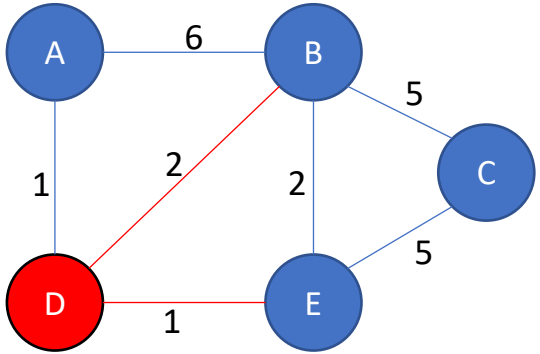


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited=[A], unvisited = [B,C,D,E]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - **For the current vertex, examine its unvisited neighbours**
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

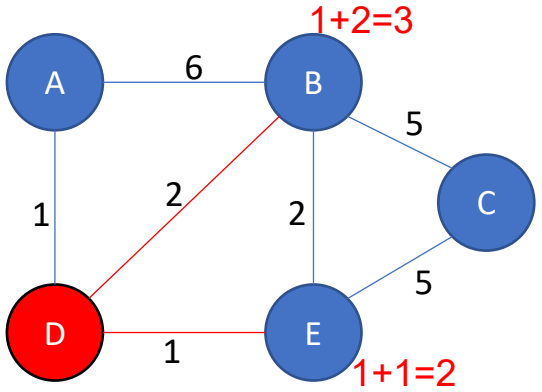


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited=[A], unvisited = [B,C,D,E]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - **For the current vertex, calculate distance of each neighbour from start vertex**
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

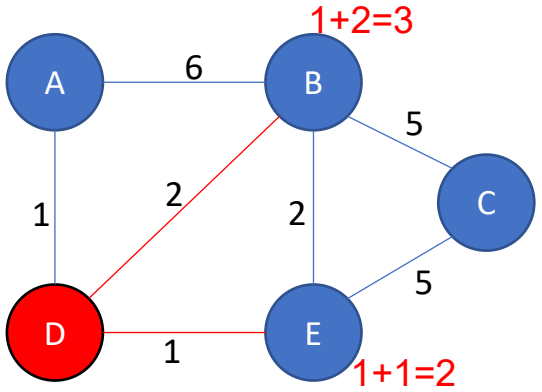


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited=[A], unvisited = [B,C,D,E]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - **If the calculated distance of a vertex is less that the known distance, update the shortest distance**
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

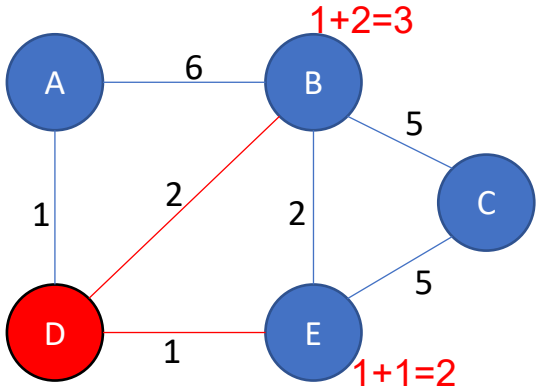


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited=[A], unvisited = [B,C,D,E]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - **If the calculated distance of a vertex is less that the known distance, update the shortest distance**
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

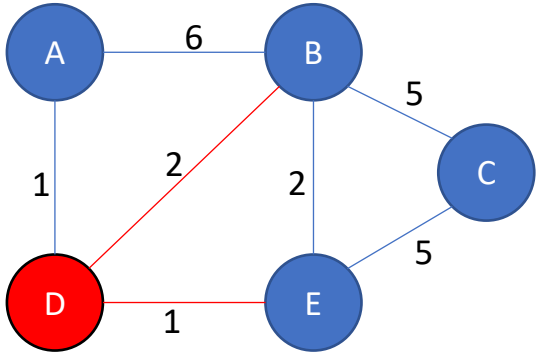


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	A
C	∞	
D	1	A
E	2	

Visited=[A], unvisited = [B,C,D,E]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - **Update the previous vertex for each of the updated distances**
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

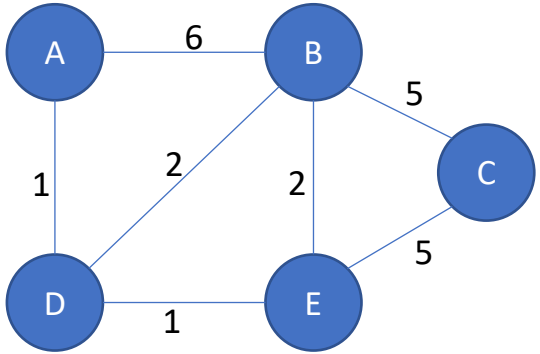


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited=[A], unvisited = [B,C,D,E]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - **Add the current vertex to the list of visited vertices**
- Until all vertices visited

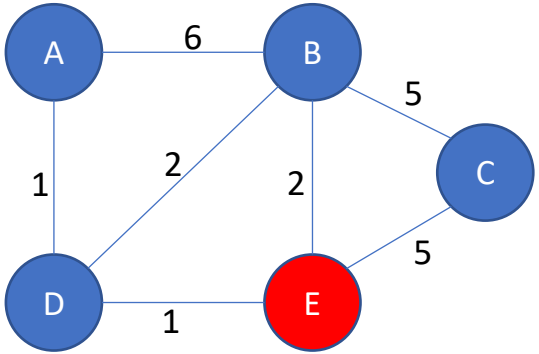


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited=[A, D], unvisited = [B,C,E]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - **Visit the unvisited vertex with the smallest known distance from the start vertex**
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

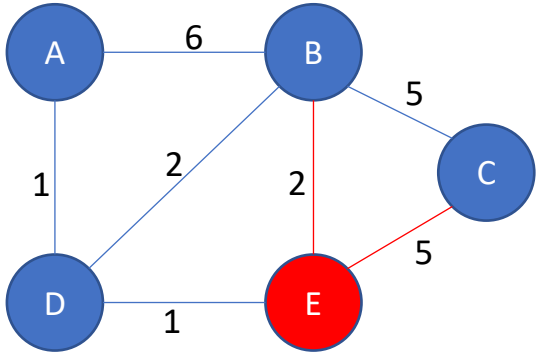


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited=[A, D], unvisited = [B,C,E]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - **For the current vertex, examine its unvisited neighbours**
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

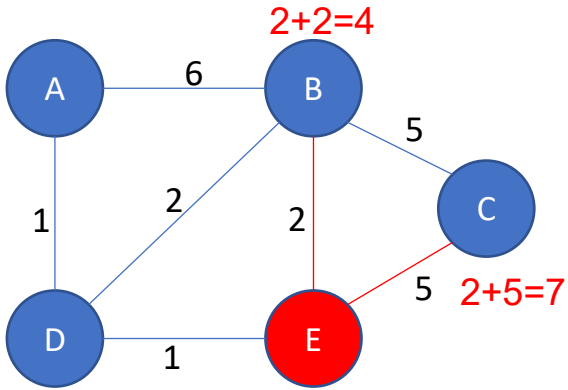


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited=[A, D], unvisited = [B,C,E]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - **For the current vertex, calculate distance of each neighbour from start vertex**
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

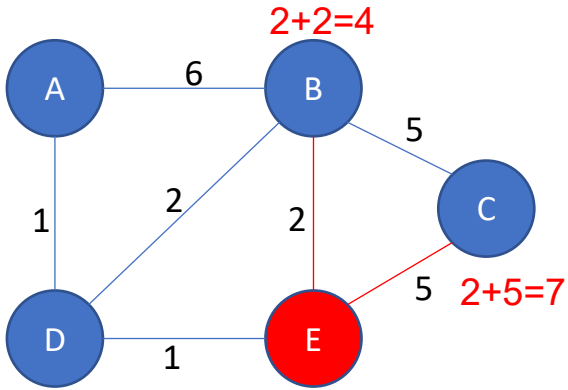


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited=[A, D], unvisited = [B,C,E]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - **If the calculated distance of a vertex is less that the known distance, update the shortest distance**
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

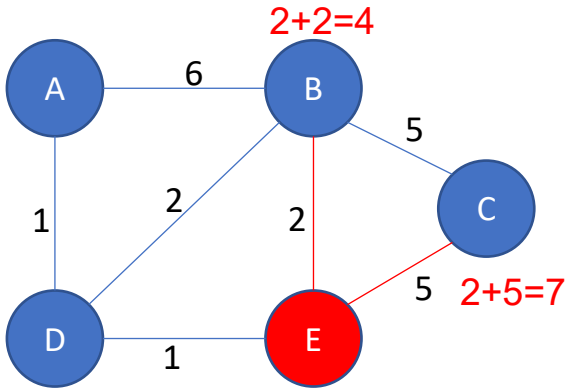


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited=[A, D], unvisited = [B,C,E]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - **If the calculated distance of a vertex is less that the known distance, update the shortest distance**
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

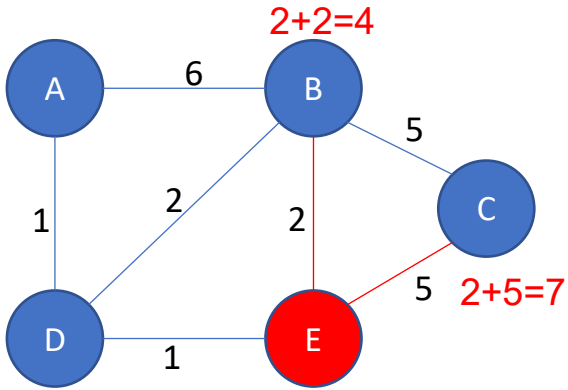


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	7	
D	1	A
E	2	D

Visited=[A, D], unvisited = [B,C,E]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - **Update the previous vertex for each of the updated distances**
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

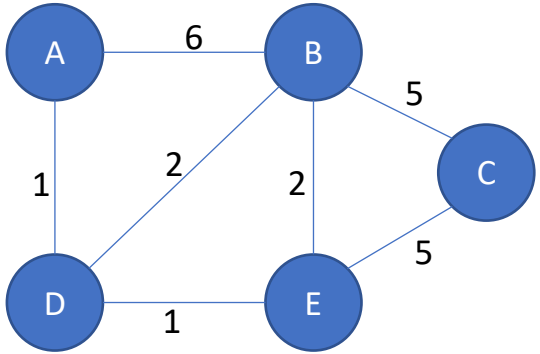


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited=[A, D], unvisited = [B,C,E]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - **Add the current vertex to the list of visited vertices**
- Until all vertices visited

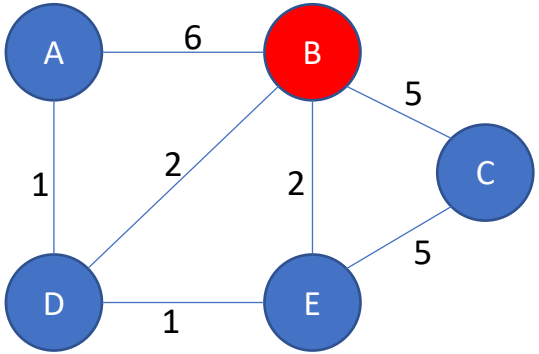


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited=[A, D, E], unvisited = [B,C]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - **Visit the unvisited vertex with the smallest known distance from the start vertex**
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

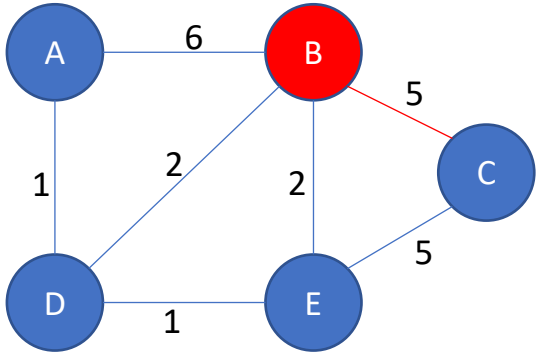


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited=[A, D, E], unvisited = [B,C]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - **For the current vertex, examine its unvisited neighbours**
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

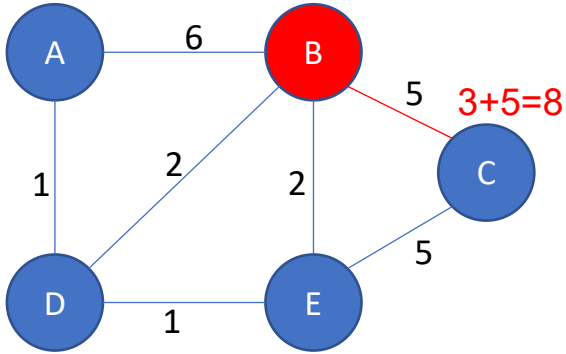


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited=[A, D, E], unvisited = [B,C]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - **For the current vertex, calculate distance of each neighbour from start vertex**
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

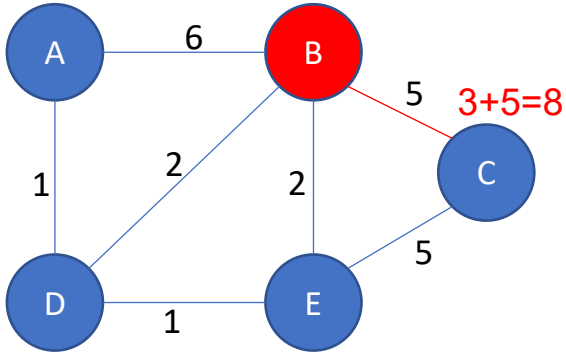


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited=[A, D, E], unvisited = [B,C]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - **If the calculated distance of a vertex is less that the known distance, update the shortest distance**
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

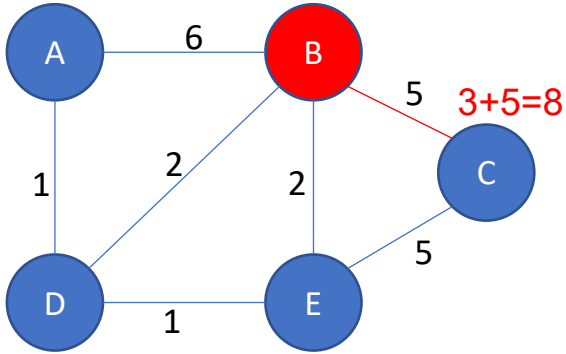


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited=[A, D, E], unvisited = [B,C]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - **Update the previous vertex for each of the updated distances**
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

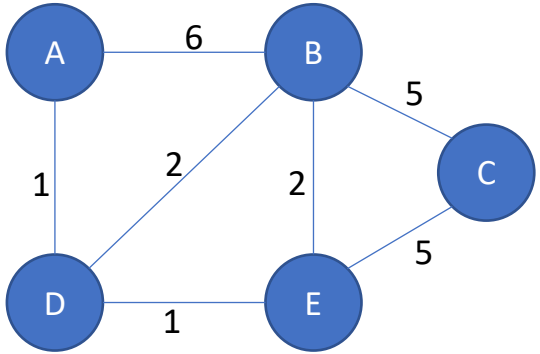


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited=[A, D, E], unvisited = [B,C]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - **Add the current vertex to the list of visited vertices**
- Until all vertices visited

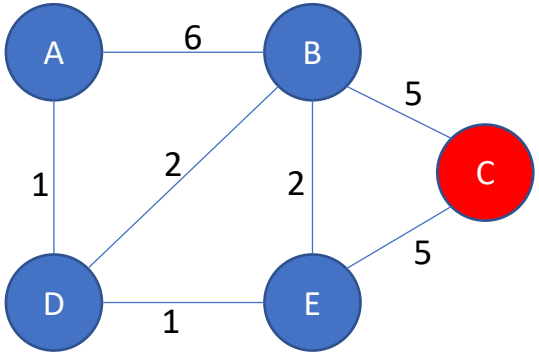


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited=[A, D, E, B], unvisited = [C]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - **Visit the unvisited vertex with the smallest known distance from the start vertex**
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

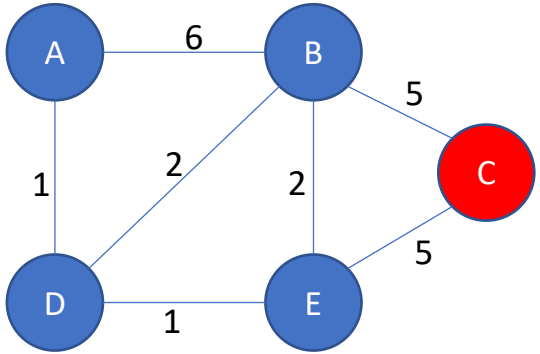


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited=[A, D, E, B], unvisited = [C]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - **For the current vertex, examine its unvisited neighbours**
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - Add the current vertex to the list of visited vertices
- Until all vertices visited

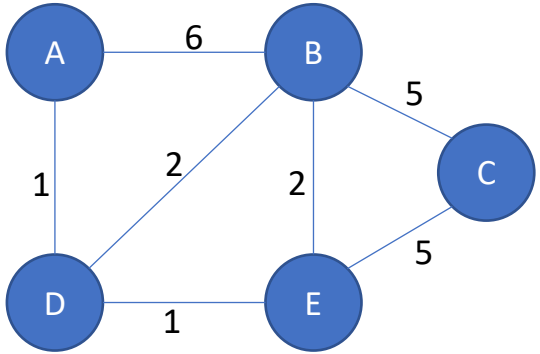


Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited=[A, D, E, B], unvisited = [C]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - **Add the current vertex to the list of visited vertices**
- Until all vertices visited



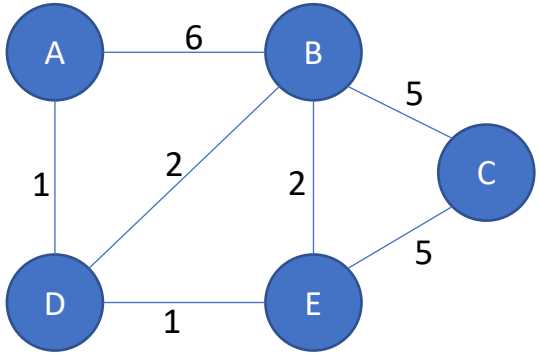
Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited=[A, D, E, B, C], unvisited = [ ]

Dijkstra's Shortest Path Algorithm

- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = inf
- Repeat:
  - Visit the unvisited vertex with the smallest known distance from the start vertex
  - For the current vertex, examine its unvisited neighbours
  - For the current vertex, calculate distance of each neighbour from start vertex
  - If the calculated distance of a vertex is less that the known distance, update the shortest distance
  - Update the previous vertex for each of the updated distances
  - **Add the current vertex to the list of visited vertices**
- Until all vertices visited

Visited=[A, D, E, B, C], unvisited = [ ]



Vertex	Shortest Distance from A	Previous Vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Mission Planning: Dijkstra (Implementation)

Dijkstra's algorithm performs n iterations. On each iteration it selects an unmarked vertex v with the lowest value $d[v]$, marks it and checks all the edges (v, to) attempting to improve the value $d[to]$.

The running time of the algorithm consists of:

- n searches for a vertex with the smallest value d[v] among $O(n)$ unmarked vertices
- m relaxation attempts

For the simplest implementation of these operations on each iteration vertex search requires $O(n)$ operations, and each relaxation can be performed in $O(1)$. Hence, the resulting asymptotic behavior of the algorithm is:

$$O(n^2 + m)$$

This complexity is optimal for dense graph, i.e. when $m \approx n^2$. However in sparse graphs, when m is much smaller than the maximal number of edges n^2, the problem can be solved in $O(n \log n + m)$ complexity. The algorithm and implementation can be found on the article Dijkstra on sparse graphs.

A^* Algorithm

Mission Planning: A*

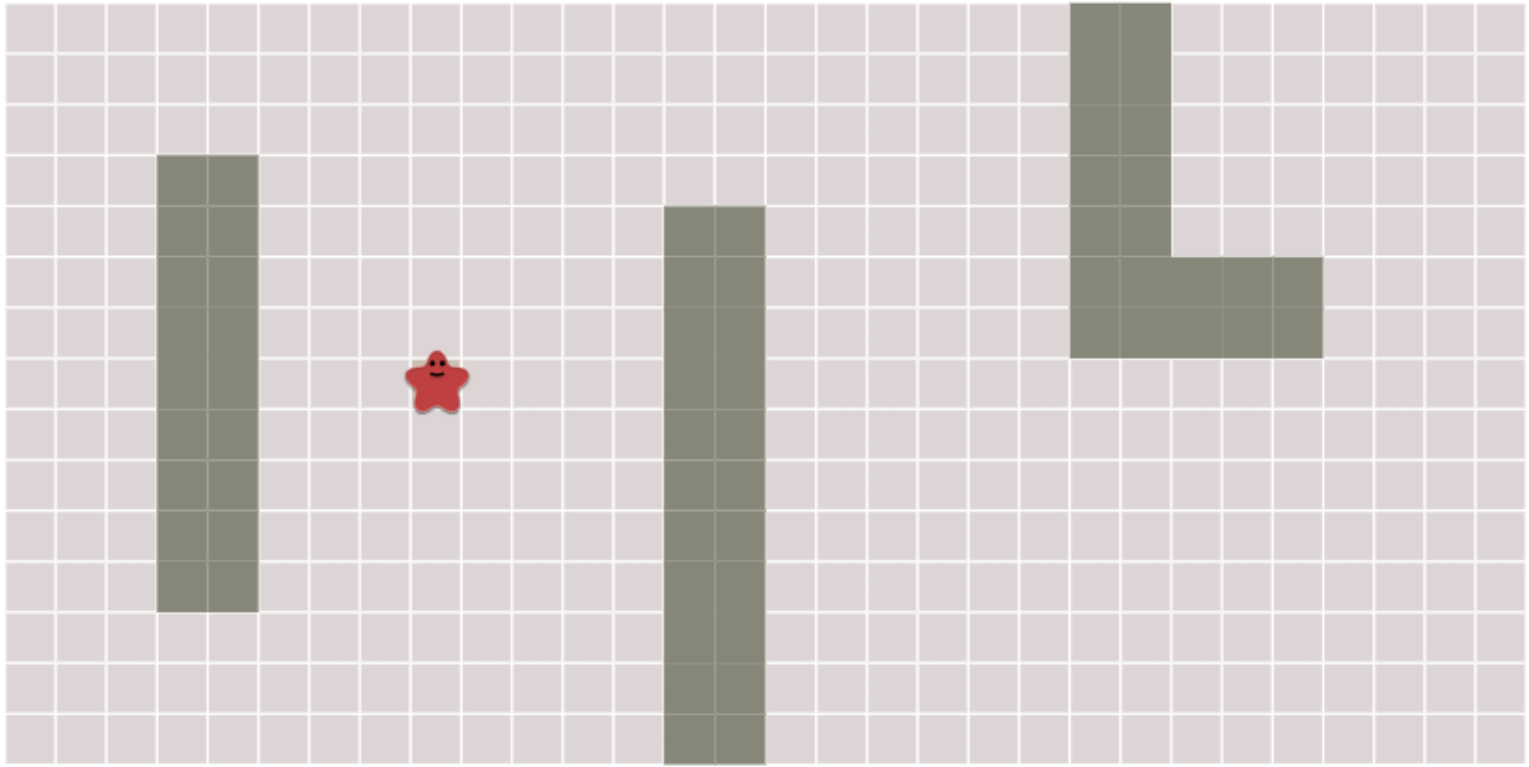
Dijkstra Algorithm:

- Uninformed search algorithms

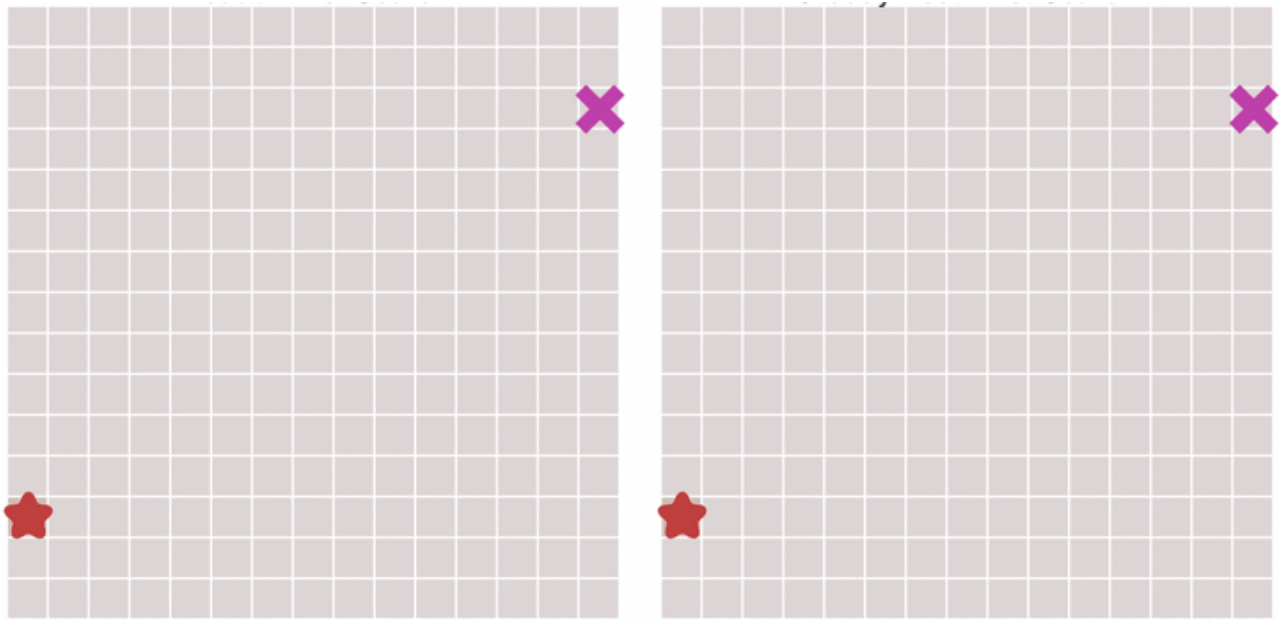
A* Algorithm：

- Uninformed search algorithms
- Heuristic
- A* gives Fast & Optimal results as compared with Dijkstra methods

Mission Planning: A* Breadth First



Mission Planning: A* Best First



Mission Planning: A* heuristic function

$$f(n) = g(n) + h(n)$$

Where

$h(n)$ is the heuristic function;

$g(n)$ is the cost to reach the node 'n' from start state.

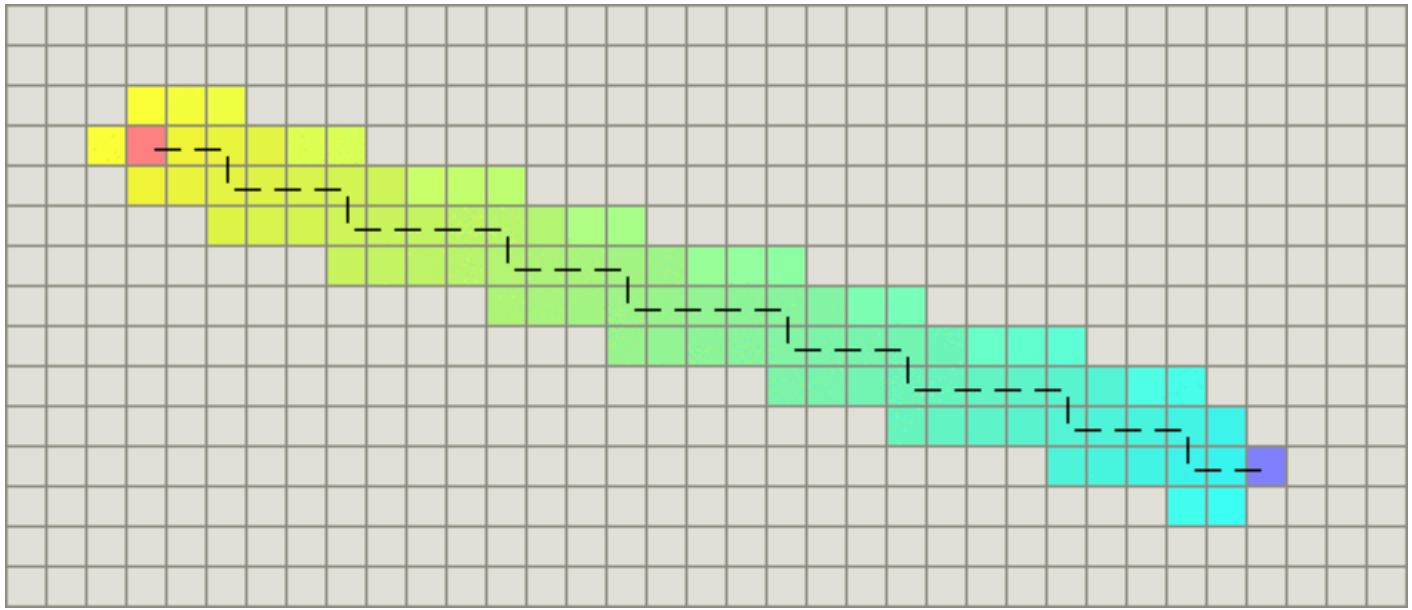
The heuristic can be used to control A*'s behavior.

- At one extreme, if $h(n)$ is 0, then only $g(n)$ plays a role, and A* turns into Dijkstra's Algorithm, which is guaranteed to find a shortest path.
- If $h(n)$ is always lower than (or equal to) the cost of moving from n to the goal, then A* is guaranteed to find a shortest path. The lower $h(n)$ is, the more node A* expands, making it slower.
- If $h(n)$ is exactly equal to the cost of moving from n to the goal, then A* will only follow the best path and never expand anything else, making it very fast. Although you can't make this happen in all cases, you can make it exact in some special cases. It's nice to know that given perfect information, A* will behave perfectly.
- If $h(n)$ is sometimes greater than the cost of moving from n to the goal, then A* is not guaranteed to find a shortest path, but it can run faster.
- At the other extreme, if $h(n)$ is very high relative to $g(n)$, then only $h(n)$ plays a role, and A* turns into Greedy Best-First-Search.

Mission Planning: A*

Manhattan distance:

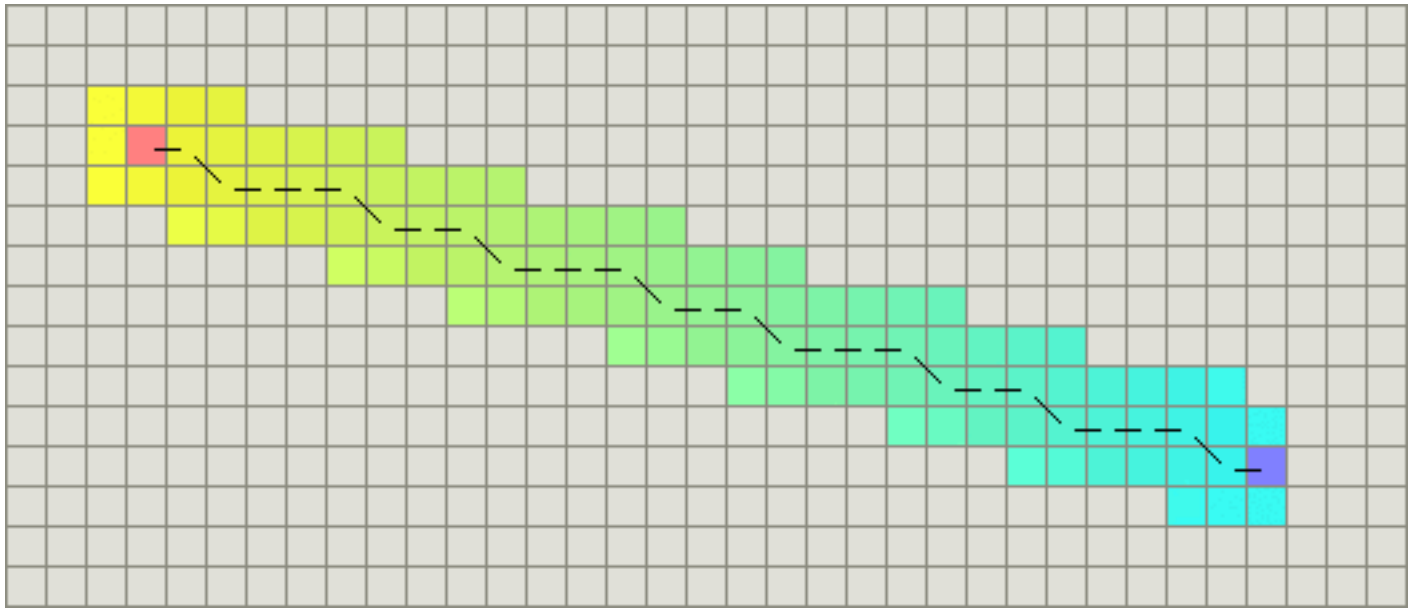
```
function heuristic(node) =
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * (dx + dy)
```



Mission Planning: A*

Diagonal distance:

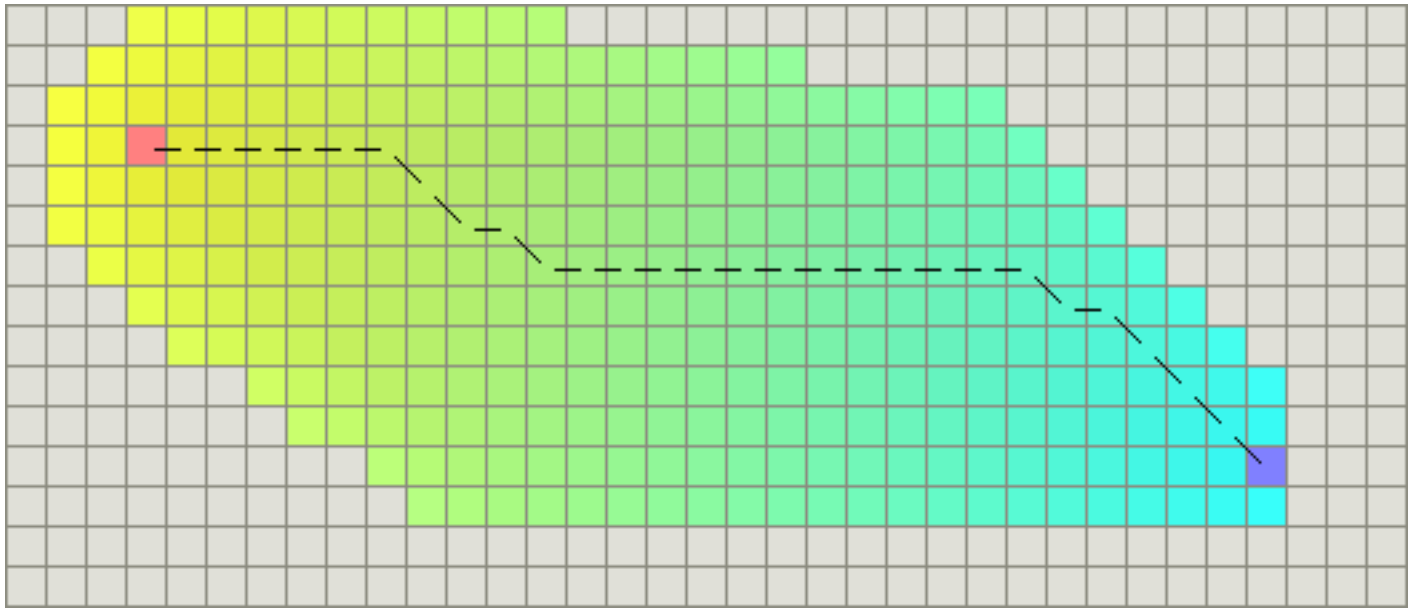
```
function heuristic(node) =
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * (dx + dy) + (D2 - 2 * D) * min(dx, dy)
```



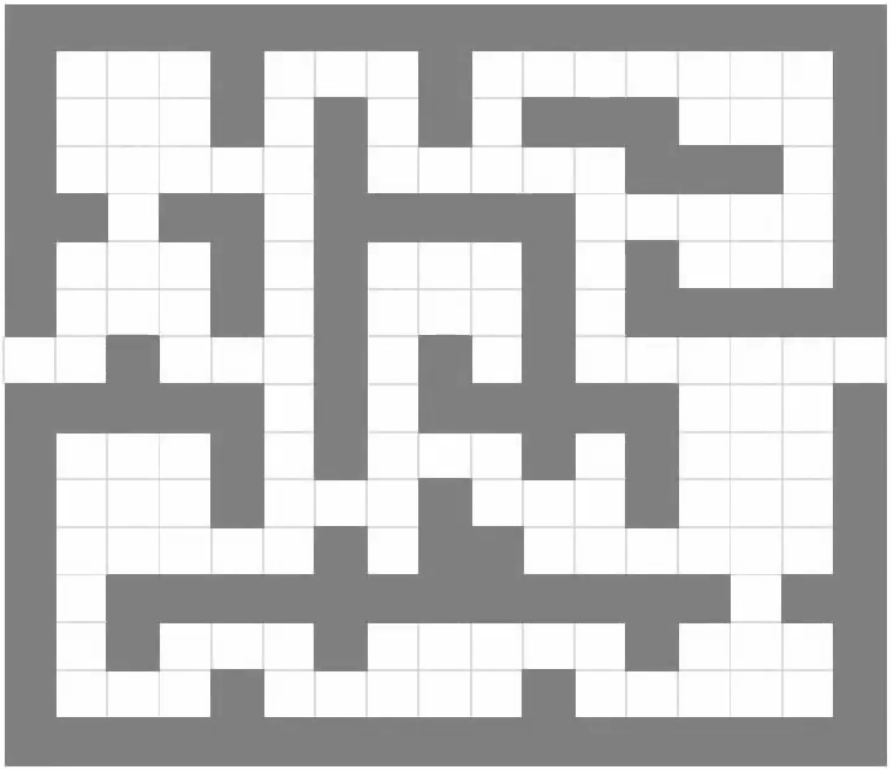
Mission Planning: A*

Euclidean distance:

```
function heuristic(node) =
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * sqrt(dx * dx + dy * dy)
```

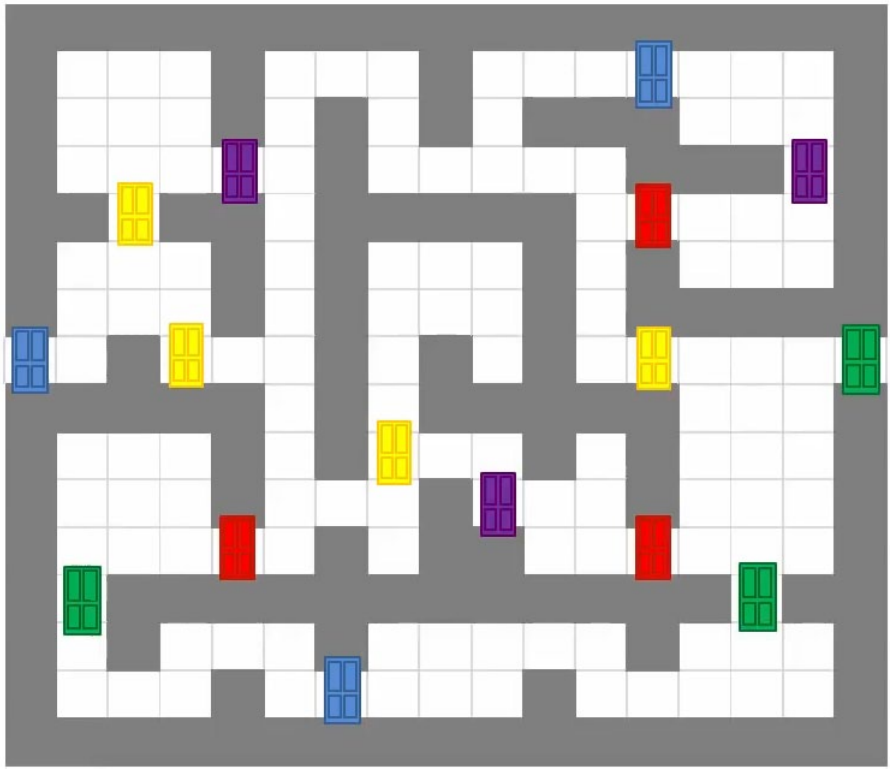


Mission Planning: A* (Example)



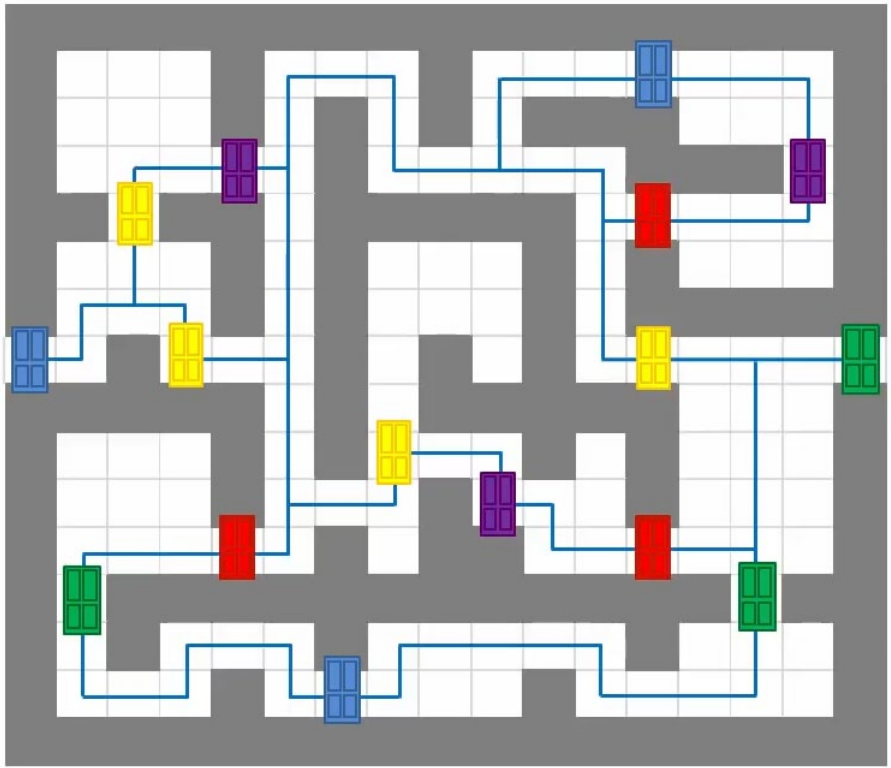
- The graph represent a maze, with rooms and corridors. It could also be genialized to a generic path finding problem.

Mission Planning: A* (Example)



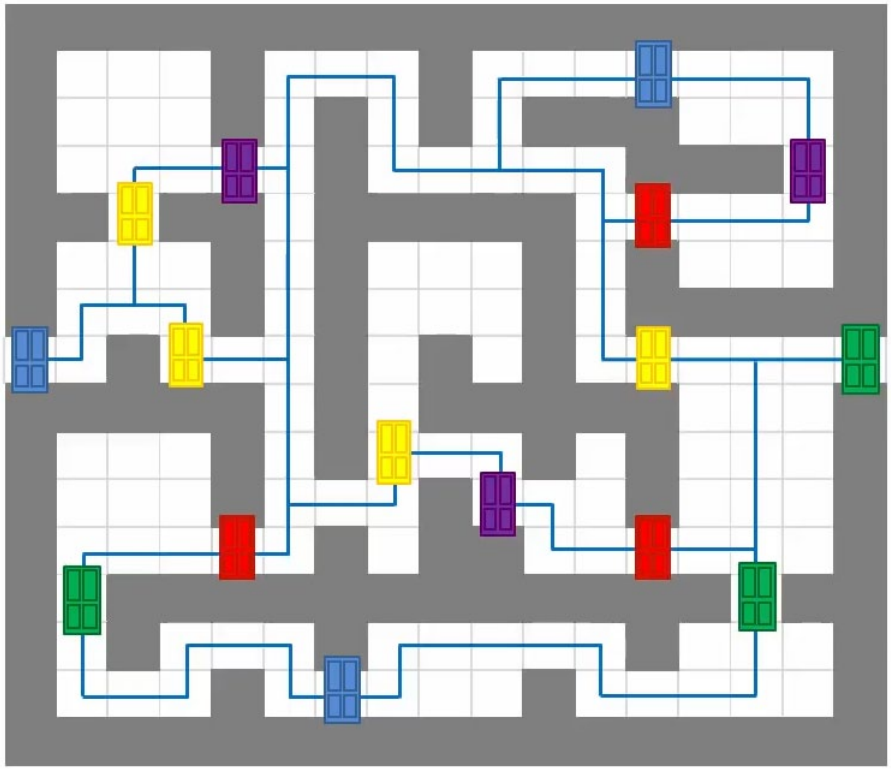
- The graph represent a maze, with rooms and corridors. It could also be genialized to a generic path finding problem.
- There are doors between rooms, which could be generalized to be key waypoints to path through.

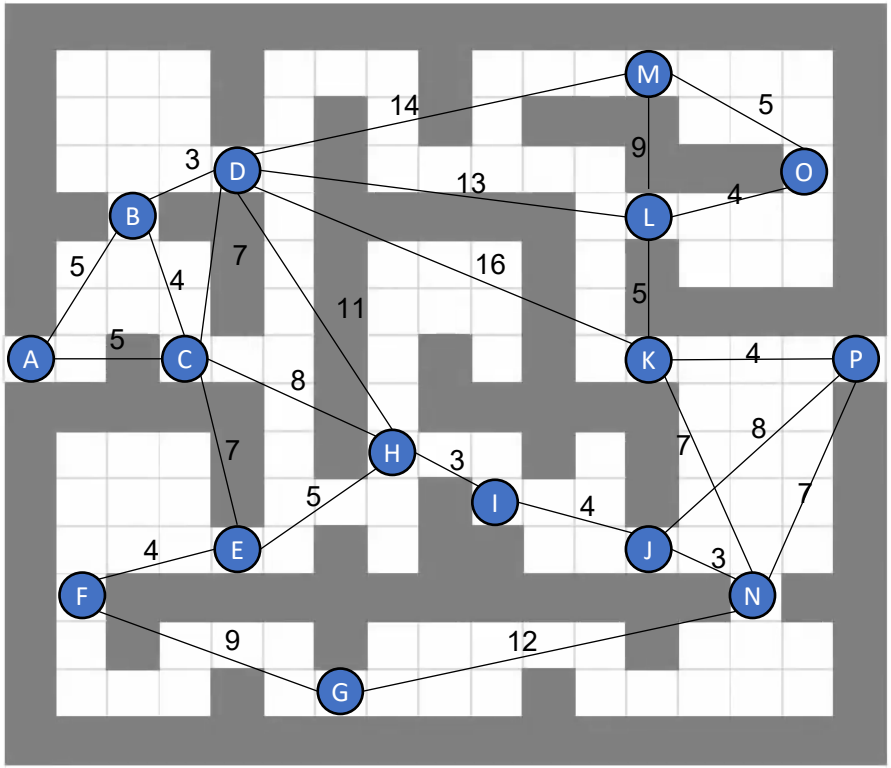
Mission Planning: A* (Example)



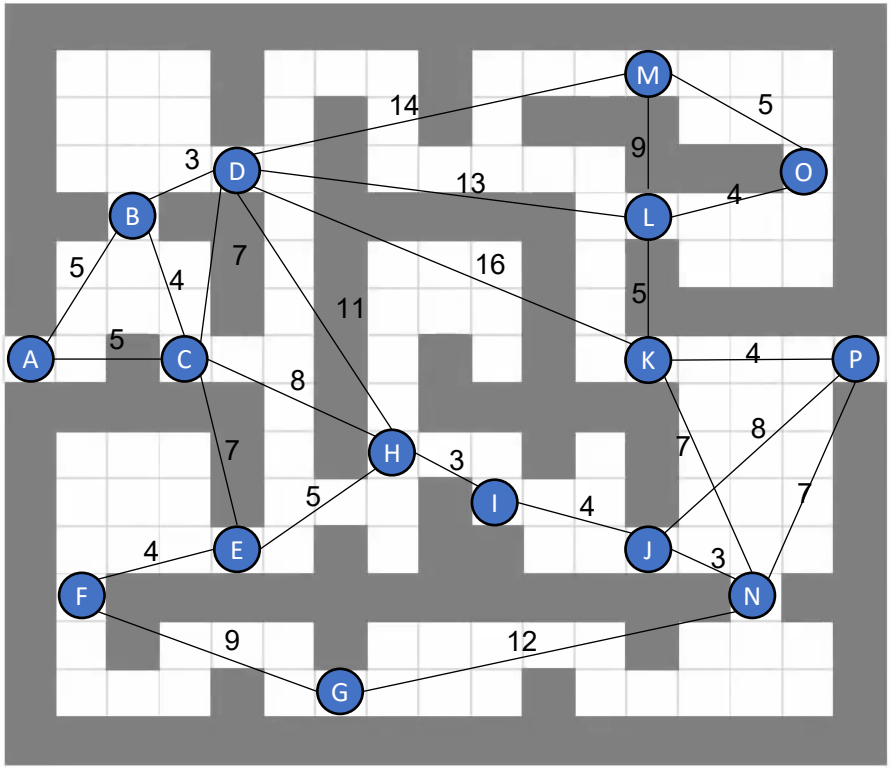
- The graph represent a maze, with rooms and corridors. It could also be genialized to a generic path finding problem.
- There are doors (vertices) between rooms, which could be generalized to be key waypoints to path through.
- Because of the walls, there are only certain ways you can go.
- The weighting for the edges represents the steps to take to reach the next vertex.

Mission Planning: A* (Example)





Mission Planning: A* (Example)



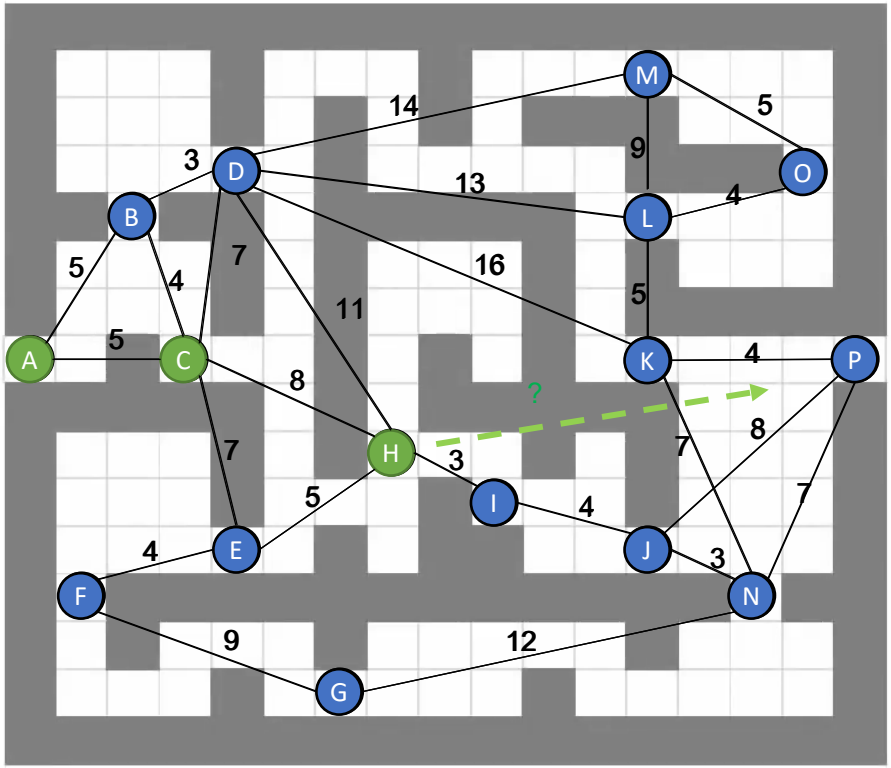
Behind the scenes

- Vertices and edges of undirected weighted graph

Is it enough?

- No
- The A* algorithm depends on so called heuristic estimate
  - **an educated guess**

Mission Planning: A* (Example)



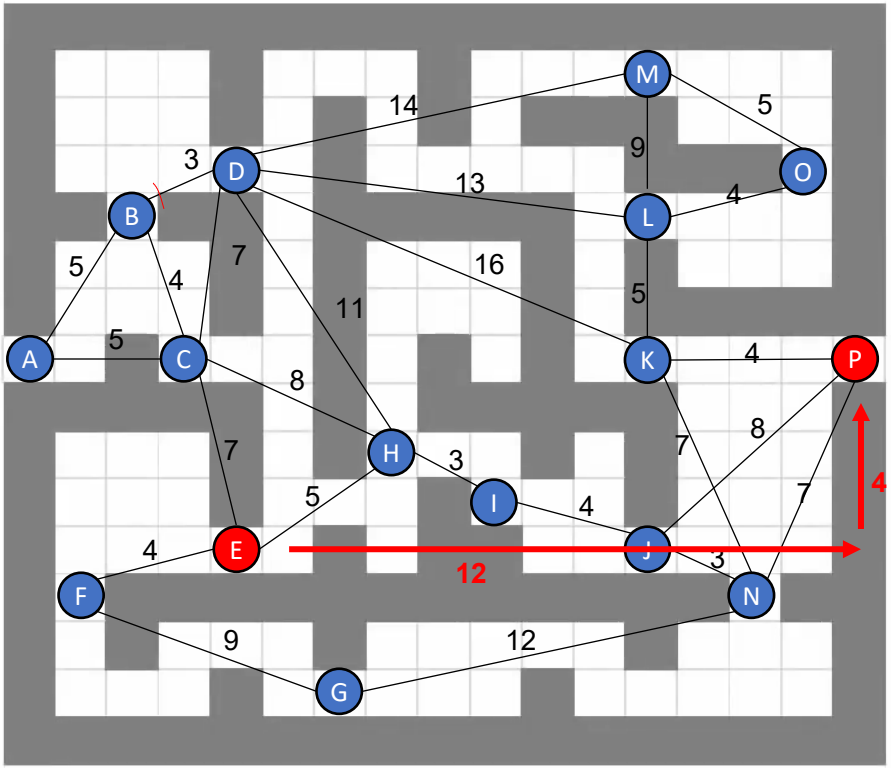
Behind the scenes

- Vertices and edges of undirected weighted graph

Is it enough?

- No
- The A* algorithm depends on so called heuristic estimate
  - **an educated guess**
- Each time we move from one vertex to another, we need an estimate of the remaining distance to the destination

Mission Planning: A* (Example)



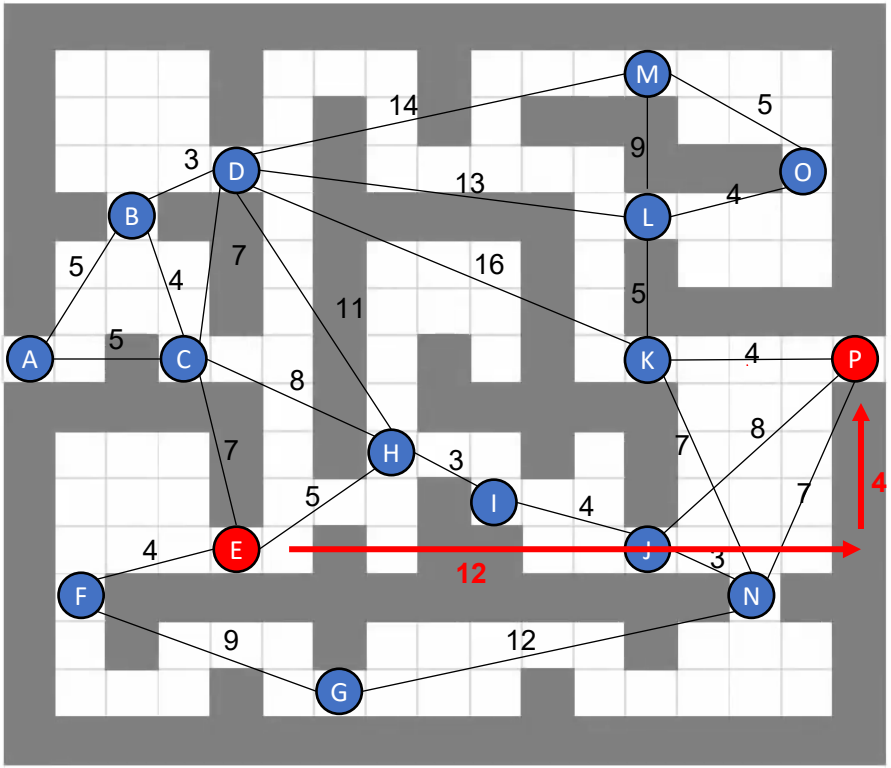
Behind the scenes

- Vertices and edges of undirected weighted graph

Is it enough?

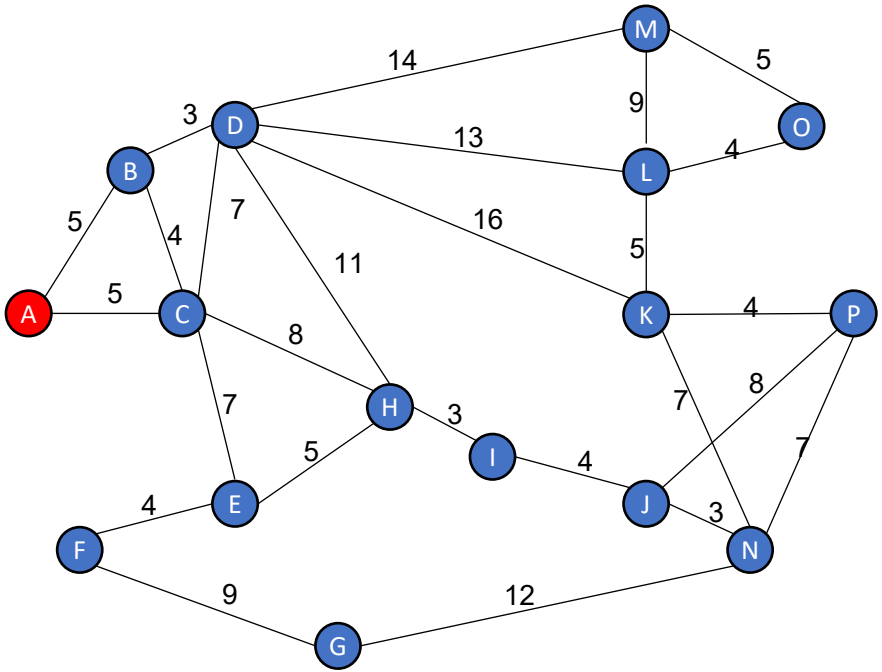
- No
- The A* algorithm depends on so called heuristic estimate
  - **an educated guess**
- Each time we move from one vertex to another, we need an estimate of the remaining distance to the destination
  - Manhattan Distance
  - Diagonal Distance
  - Euclidean distance

Mission Planning: A* (Example)



Vertex	Distance from A (g)	Heuristic Distance (h)	f = g + h	Previous vertex
A		16		
B		17		
C		13		
D		16		
E		16		
F		20		
G		17		
H		11		
I		10		
J		8		
K		4		
L		7		
M		10		
N		7		
O		5		
P		0		

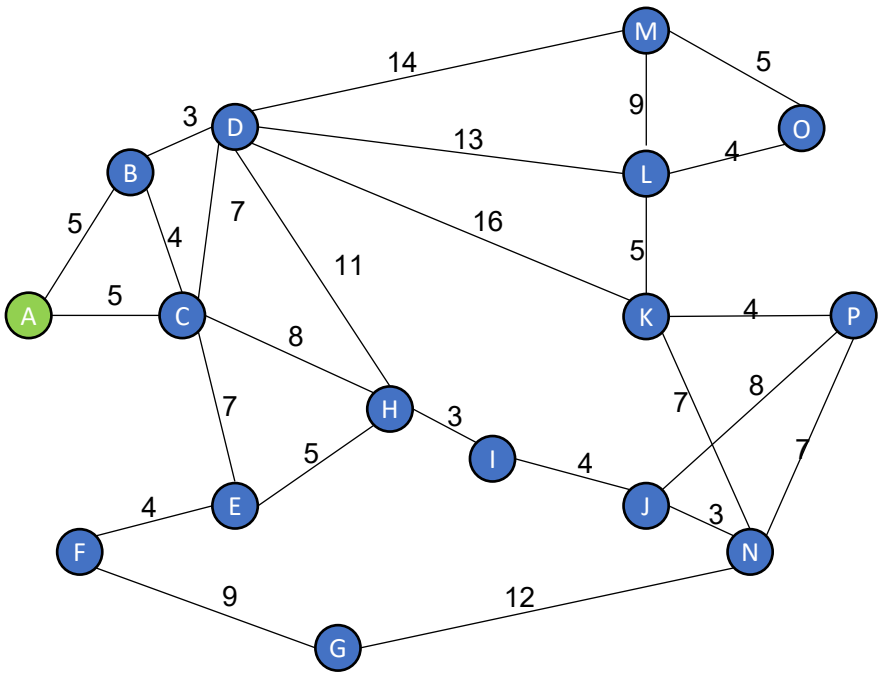
Mission Planning: A* (Example)



Open set: A
Close set:

Vertex	Distance from A (g)	Heuristic Distance (h)	f = g + h	Previous vertex
A		16		
B		17		
C		13		
D		16		
E		16		
F		20		
G		17		
H		11		
I		10		
J		8		
K		4		
L		7		
M		10		
N		7		
O		5		
P		0		

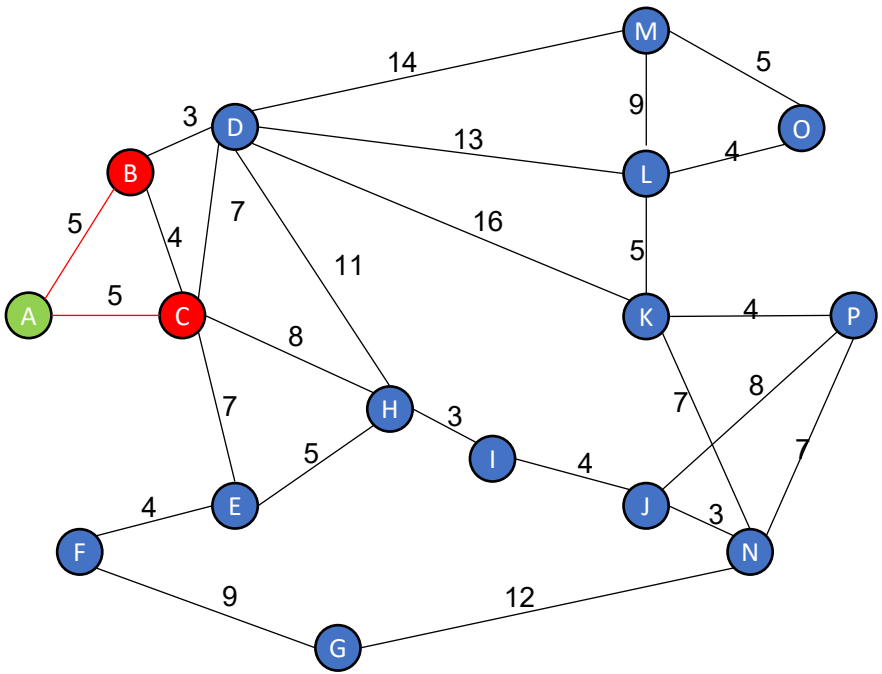
Mission Planning: A* (Example)



Vertex	Distance from A (g)	Heuristic Distance (h)	f = g + h	Previous vertex
A	0	16	16	
B		17		
C		13		
D		16		
E		16		
F		20		
G		17		
H		11		
I		10		
J		8		
K		4		
L		7		
M		10		
N		7		
O		5		
P		0		

Open set: A
Close set:

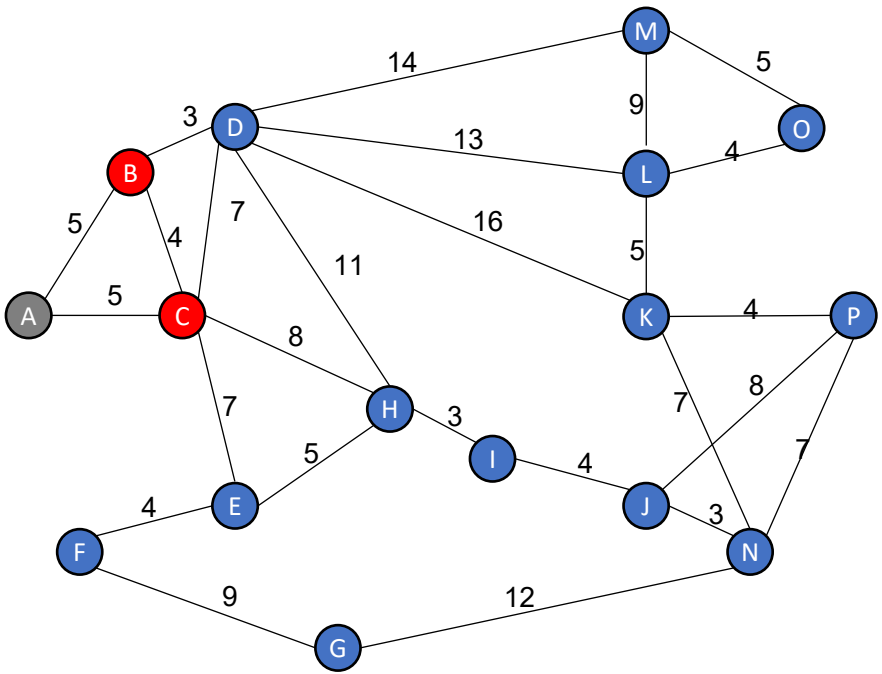
Mission Planning: A* (Example)



Open set: A B C
Close set:

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		16		
E		16		
F		20		
G		17		
H		11		
I		10		
J		8		
K		4		
L		7		
M		10		
N		7		
O		5		
P		0		

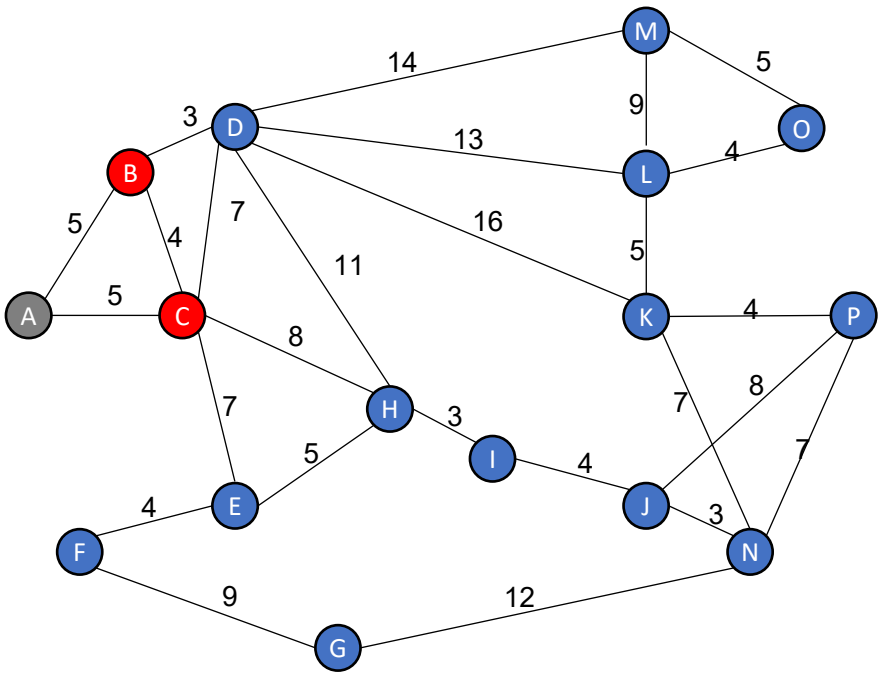
Mission Planning: A* (Example)



Open set: B C
Close set: A

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		16		
E		16		
F		20		
G		17		
H		11		
I		10		
J		8		
K		4		
L		7		
M		10		
N		7		
O		5		
P		0		

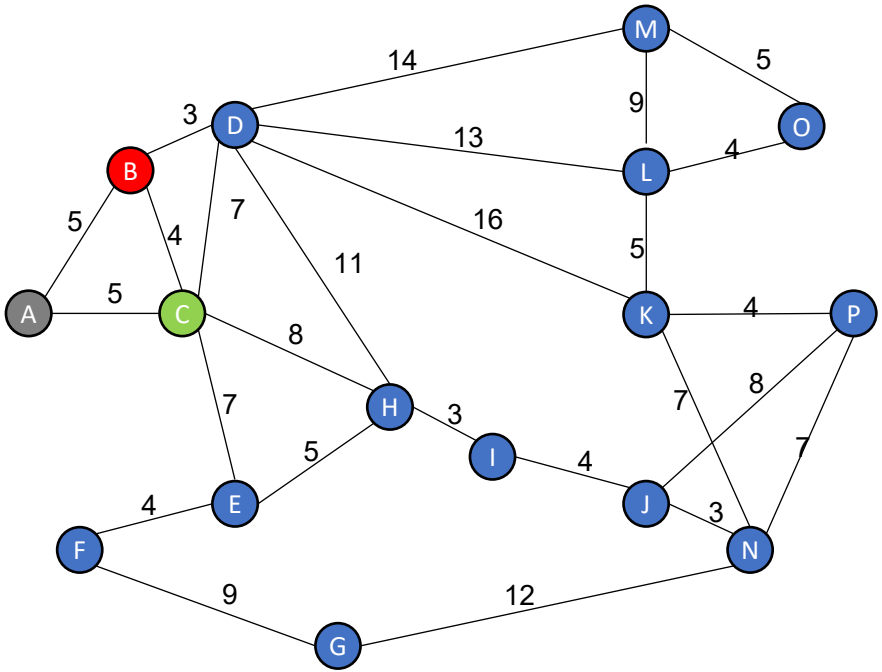
Mission Planning: A* (Example)



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		16		
E		16		
F		20		
G		17		
H		11		
I		10		
J		8		
K		4		
L		7		
M		10		
N		7		
O		5		
P		0		

Open set: B C
Close set: A

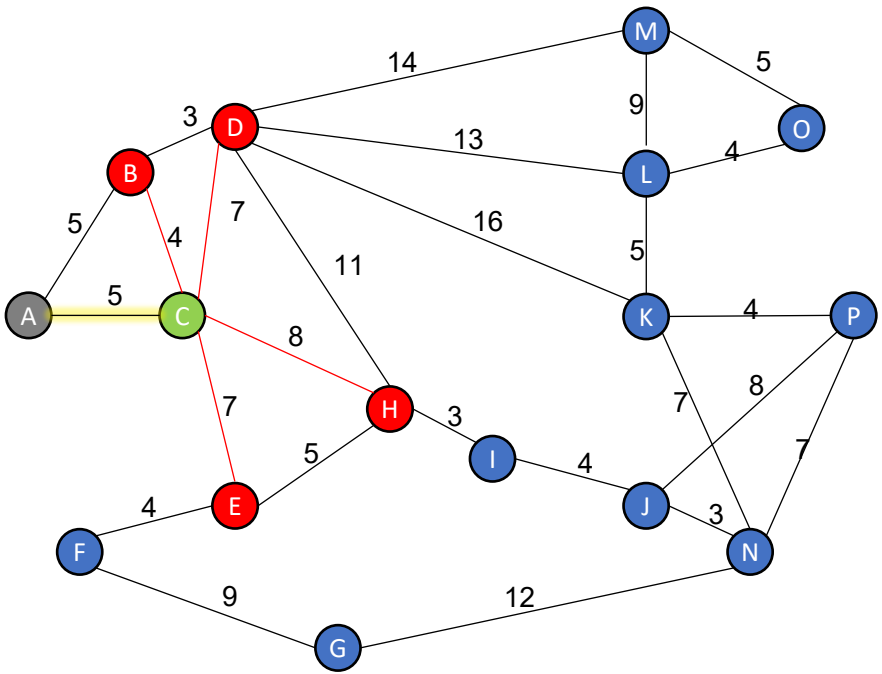
Mission Planning: A*



Open set: B C
Close set: A

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		16		
E		16		
F		20		
G		17		
H		11		
I		10		
J		8		
K		4		
L		7		
M		10		
N		7		
O		5		
P		0		

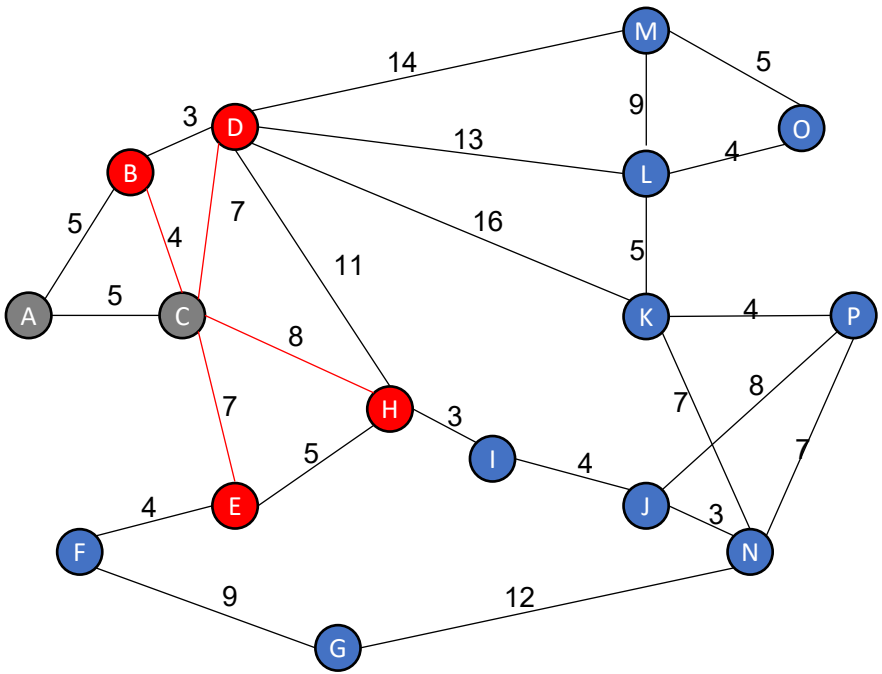
Mission Planning: A* (Example)



Vertex	Distance from A (g)	Heuristic Distance (h)	f = g + h	Previous vertex
A	0	16	16	
B	5 9	17	22 26	A C
C	5	13	18	A
D	12	16	28	C
E	12	16	28	C
F		20		
G		17		
H	13	11	24	C
I		10		
J		8		
K		4		
L		7		
M		10		
N		7		
O		5		
P		0		

Open set: B C D H E
Close set: A

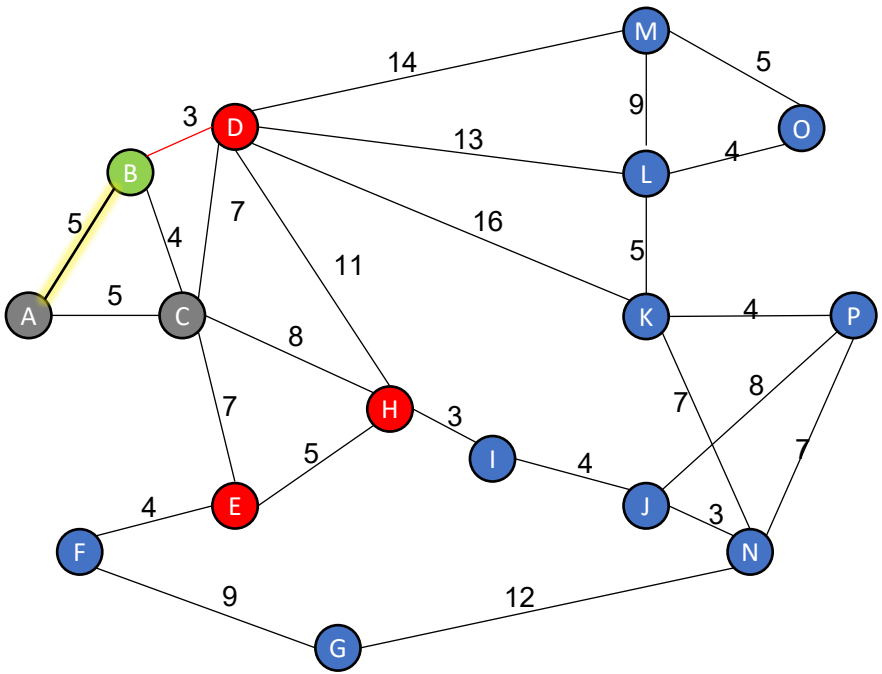
Mission Planning: A* (Example)



Open set: B D H E
Close set: A C

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	12	16	28	C
E	12	16	28	C
F		20		
G		17		
H	13	11	24	C
I		10		
J		8		
K		4		
L		7		
M		10		
N		7		
O		5		
P		0		

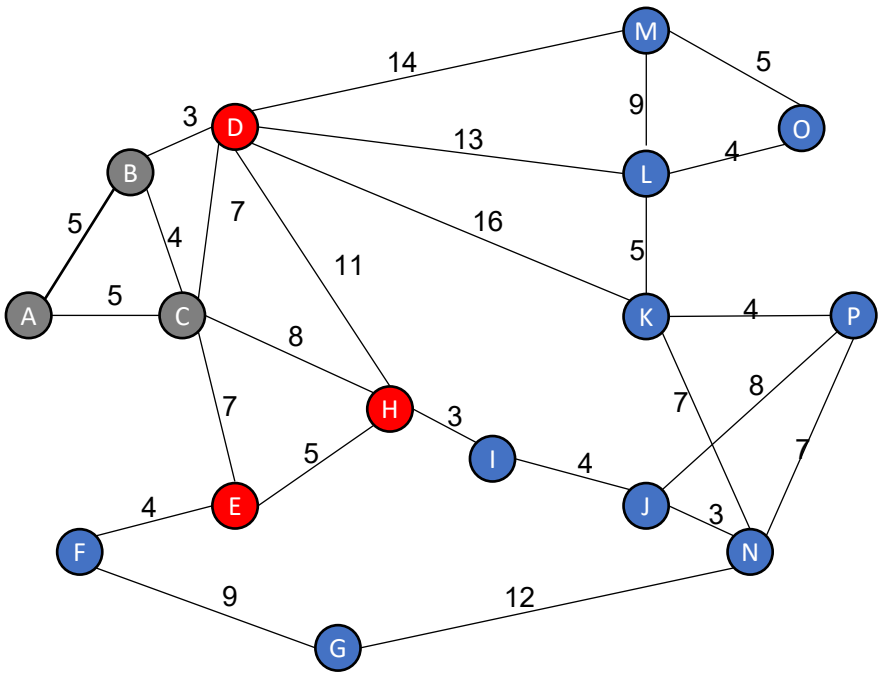
Mission Planning: A* (Example)



Open set: B D H E
Close set: A C

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	12 8	16	28 24	C B
E	12	16	28	C
F		20		
G		17		
H	13	11	24	C
I		10		
J		8		
K		4		
L		7		
M		10		
N		7		
O		5		
P		0		

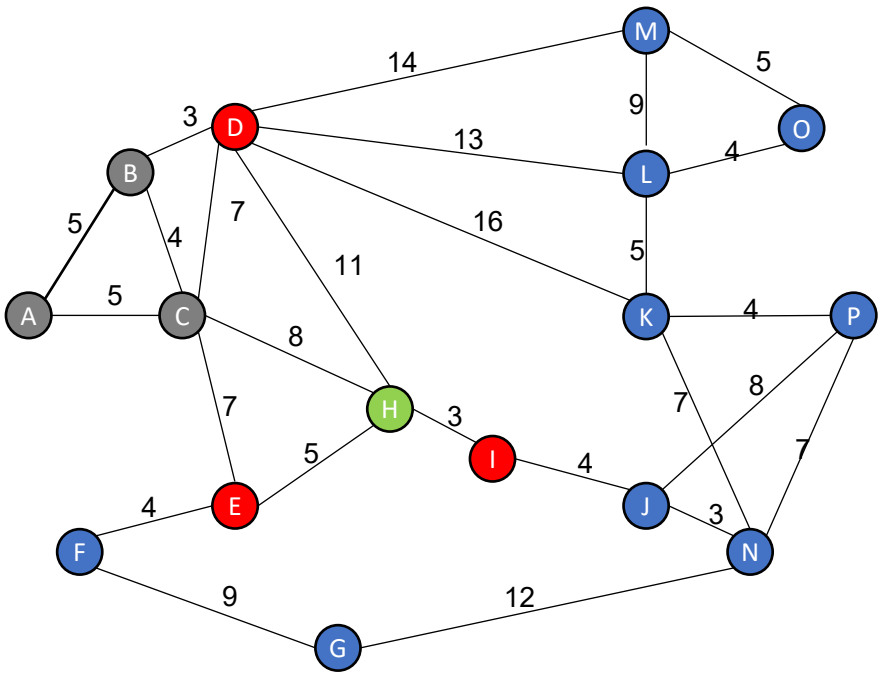
Mission Planning: A* (Example)



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		10		
J		8		
K		4		
L		7		
M		10		
N		7		
O		5		
P		0		

Open set: D H E
Close set: A C B

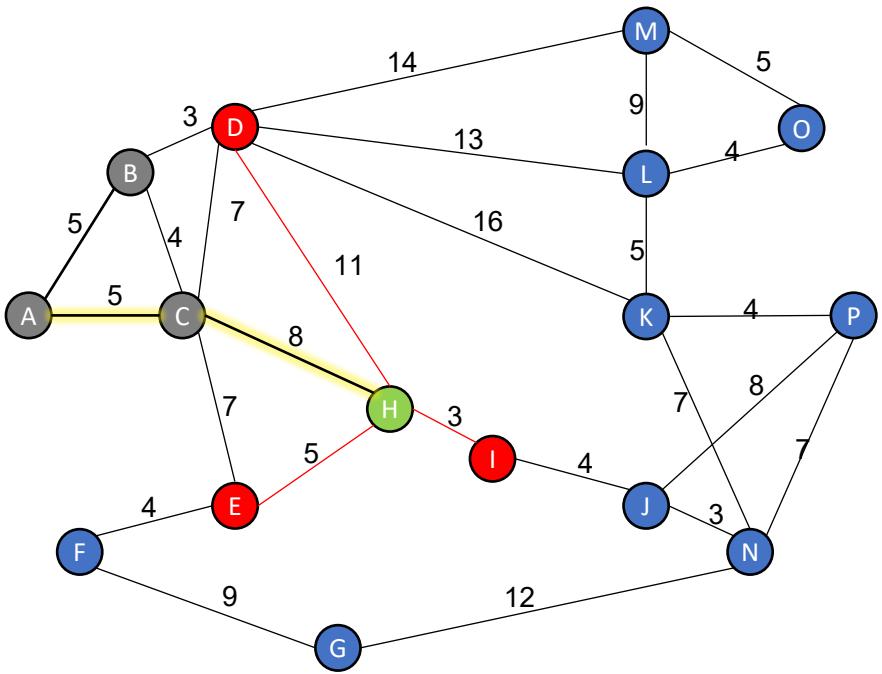
Mission Planning: A* (Example)



Open set: D H E
Close set: A C B

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		10		
J		8		
K		4		
L		7		
M		10		
N		7		
O		5		
P		0		

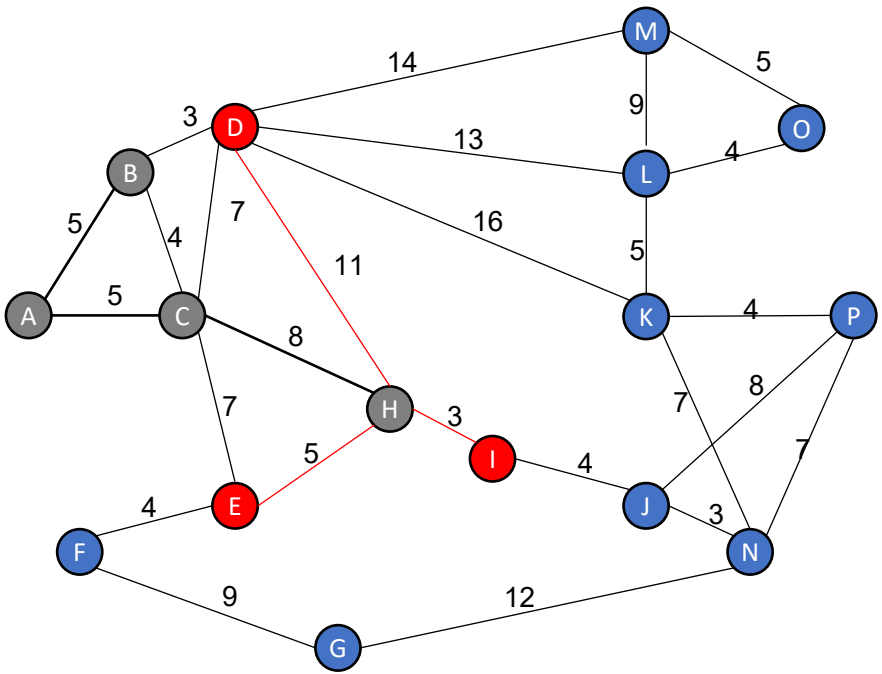
Mission Planning: A* (Example)



Open set: D H E I
Close set: A C B

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 24	16	24 40	B H
E	12 18	16	28 34	C H
F		20		
G		17		
H	13	11	24	C
I	16	10	26	H
J		8		
K		4		
L		7		
M		10		
N		7		
O		5		
P		0		

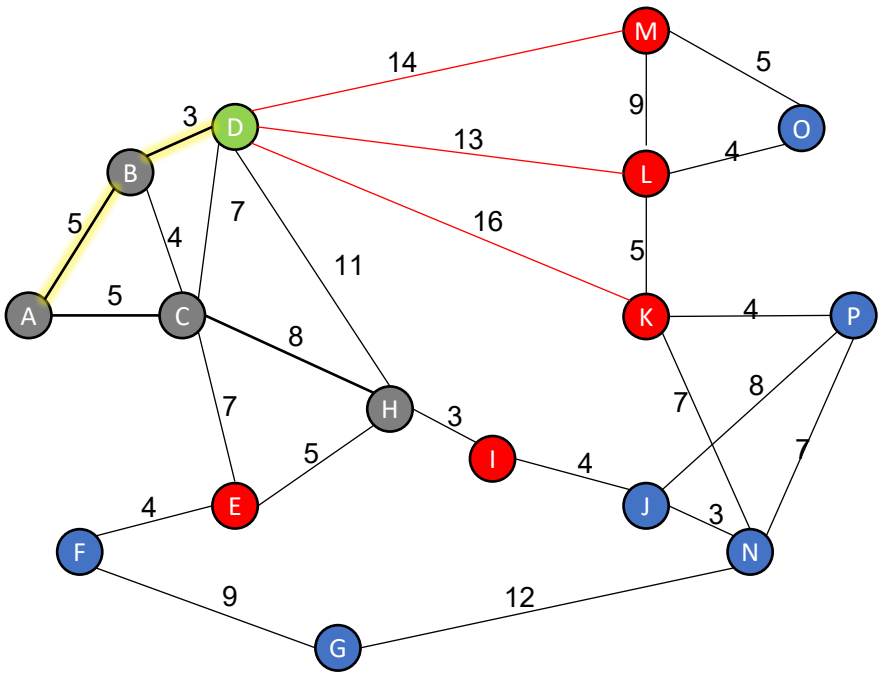
Mission Planning: A* (Example)



Open set: D E I
Close set: A C B H

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		8		
K		4		
L		7		
M		10		
N		7		
O		5		
P		0		

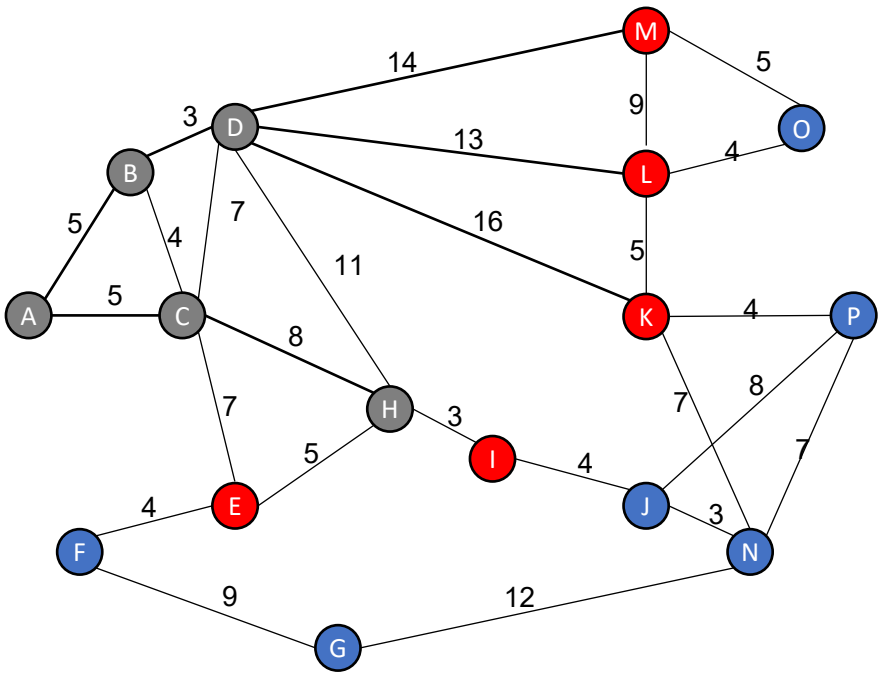
Mission Planning: A* (Example)



Open set: D E I M L K
Close set: A C B H

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		8		
K	24	4	28	D
L	21	7	28	D
M	22	10	32	D
N		7		
O		5		
P		0		

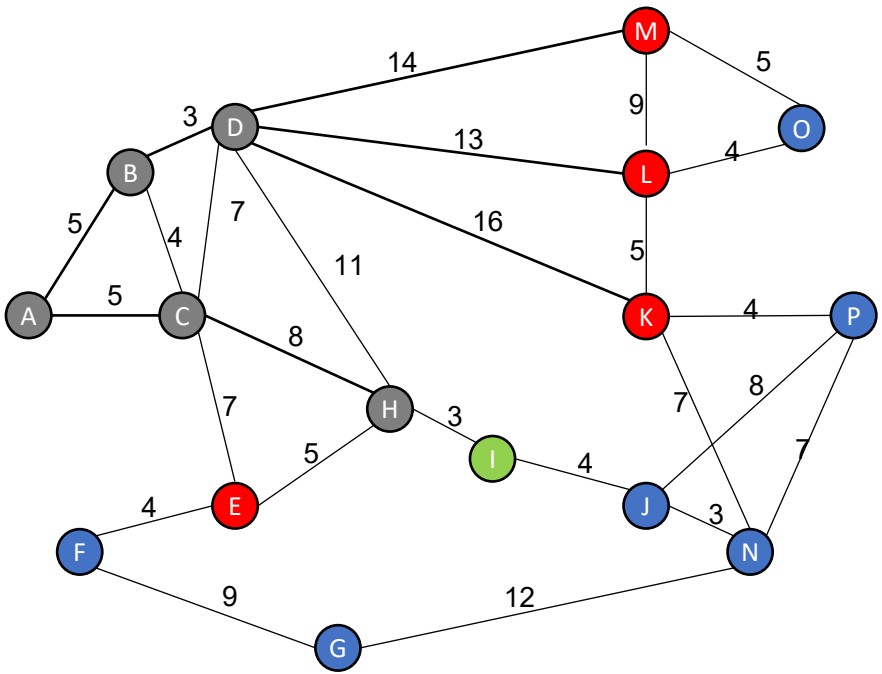
Mission Planning: A* (Example)



Open set: E I M L K
Close set: A C B H D

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		8		
K	24	4	28	D
L	21	7	28	D
M	22	10	32	D
N		7		
O		5		
P		0		

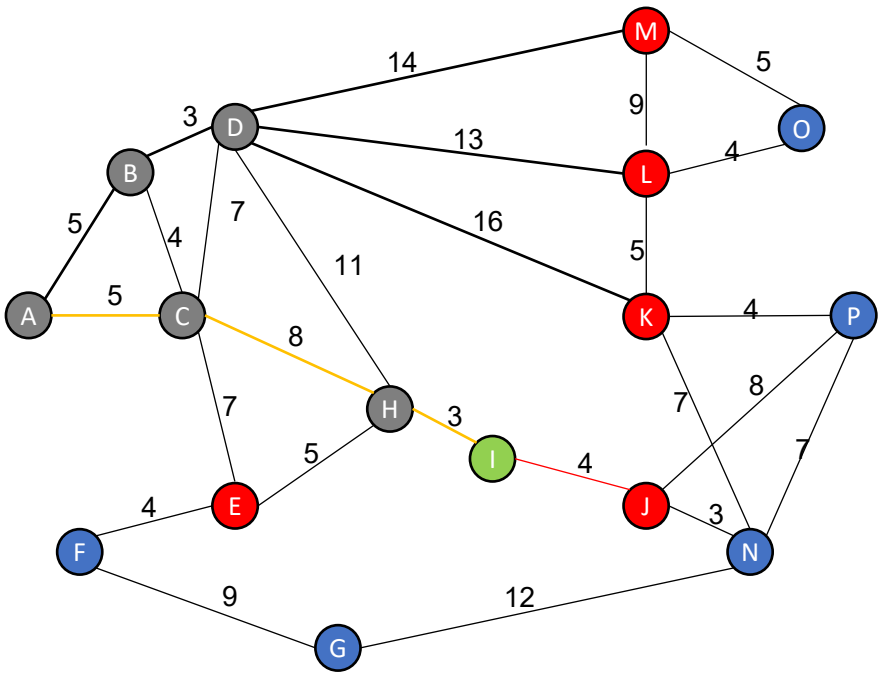
Mission Planning: A* (Example)



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		8		
K	24	4	28	D
L	21	7	28	D
M	22	10	32	D
N		7		
O		5		
P		0		

Open set: E I M L K
Close set: A C B H D

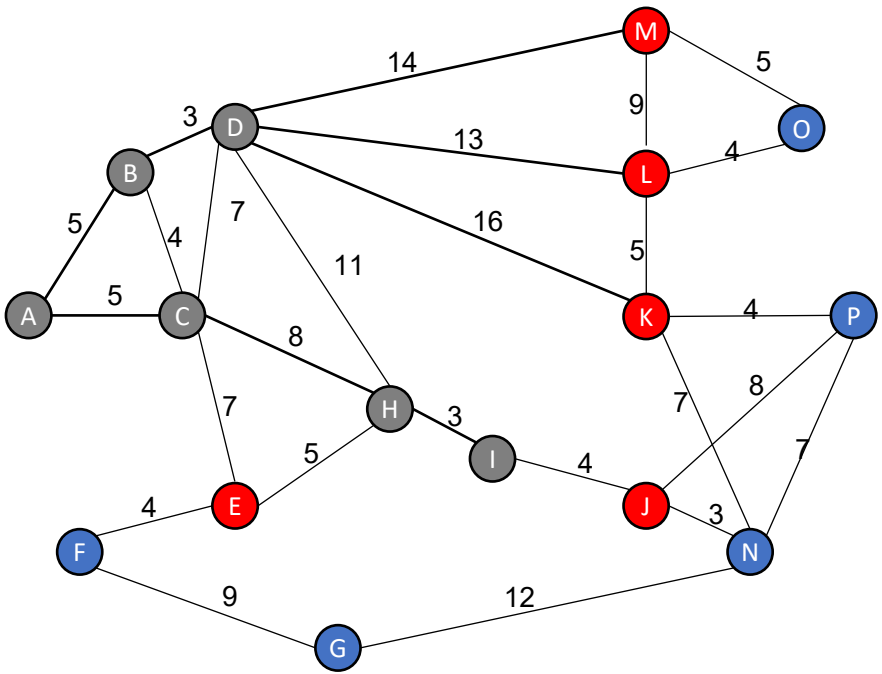
Mission Planning: A* (Example)



Open set: E I M L K J
Close set: A C B H D

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		7		
O		5		
P		0		

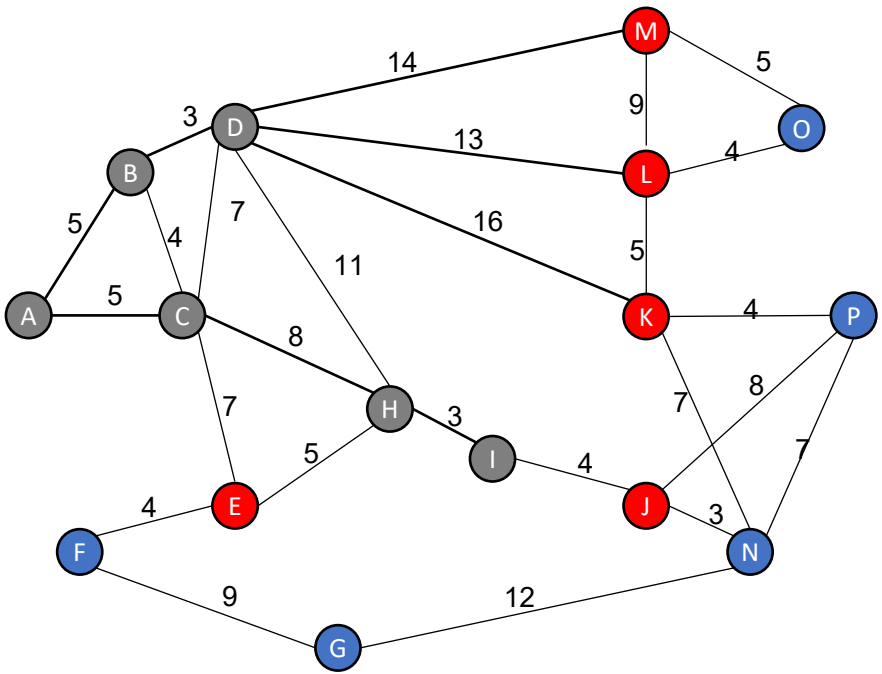
Mission Planning: A* (Example)



Open set: E M L K J
Close set: 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		7		
O		5		
P		0		

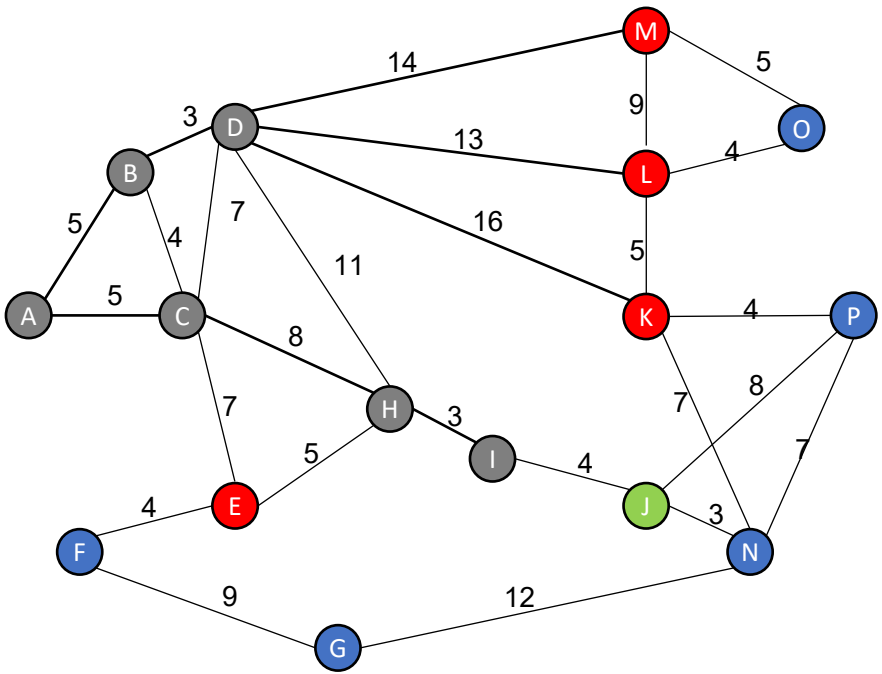
Mission Planning: A* (Example)



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		7		
O		5		
P		0		

Open set: E M L K J
Close set: A C B H D I

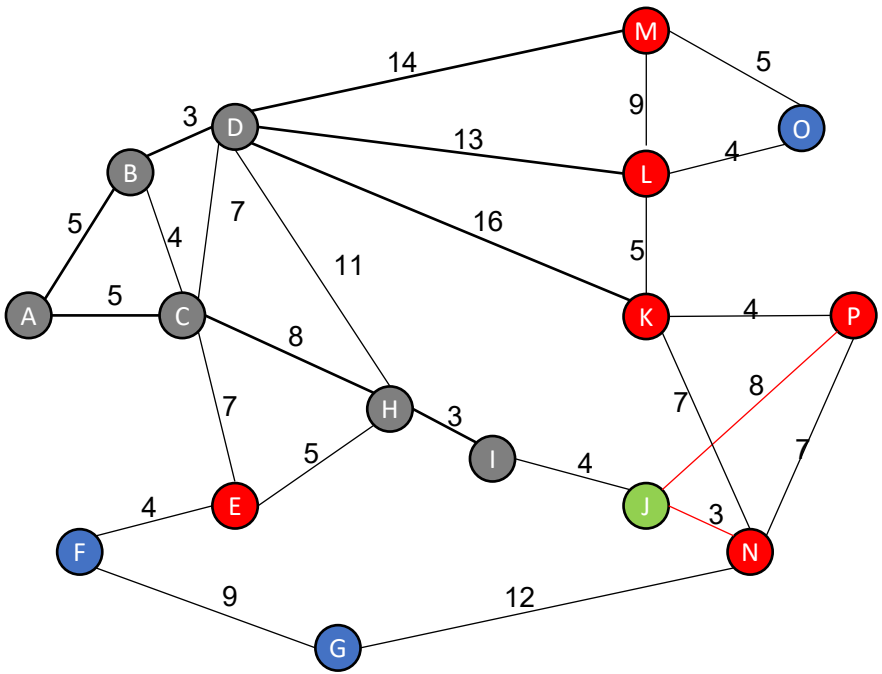
Mission Planning: A* (Example)



Open set: E M L K J
Close set: 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		7		
O		5		
P		0		

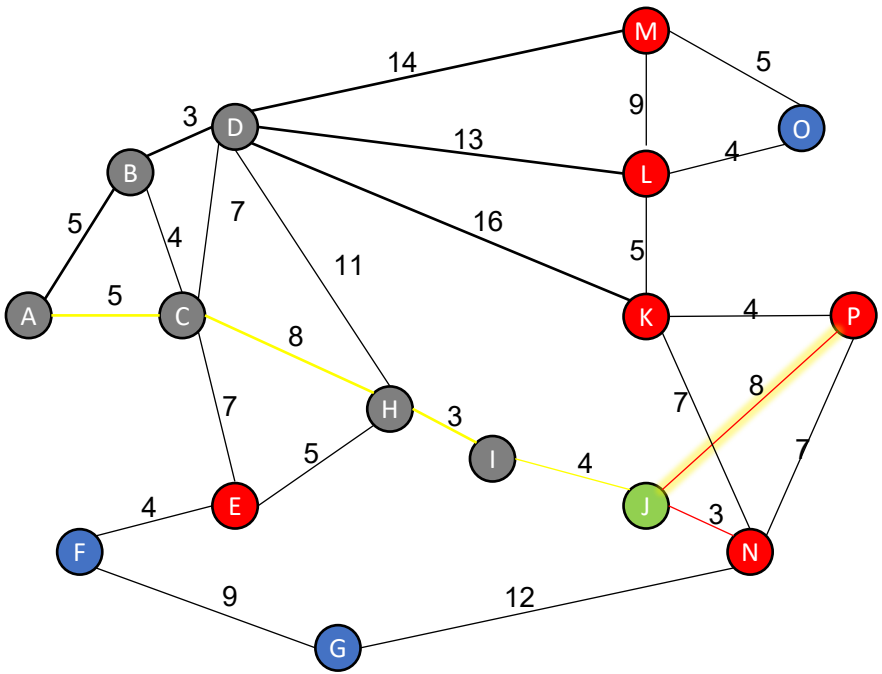
Mission Planning: A* (Example)



Open set: E M L K J P N
Close set: 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		7		
O		5		
P		0		

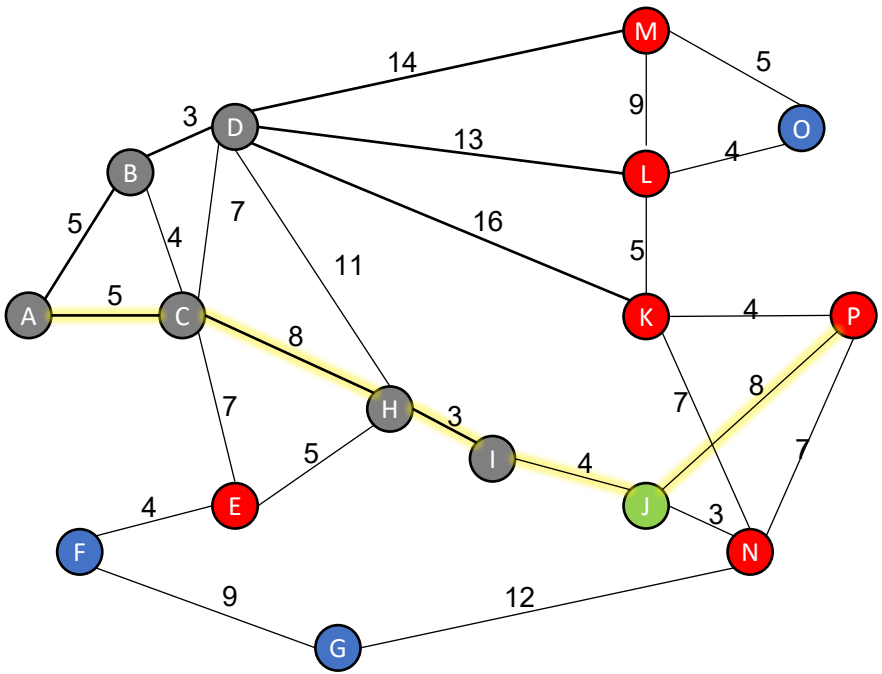
Mission Planning: A* (Example)



Open set: E M L K J P N
Close set: 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

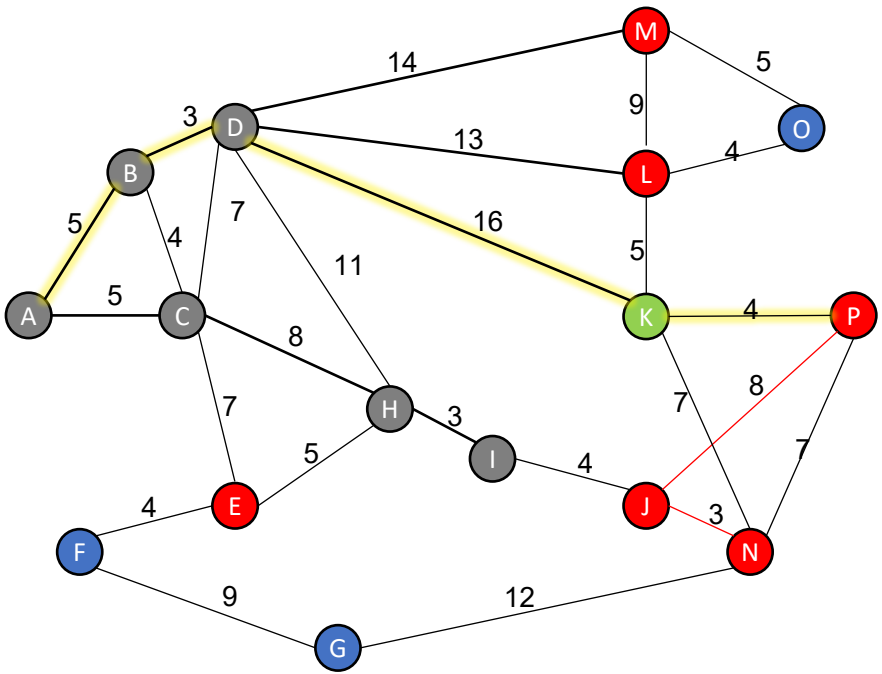
Mission Planning: A* (Example)



Open set: E M L K J P N
Close set: 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

Mission Planning: A* (Example)



Open set: E M L K J P N
Close set: 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		7		
O		5		
P	28	0	28	K

Mission Planning: A* procedure

```
Initialize 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 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
```

Mission Planning: A* Summary

- A* has a wide 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.
- Each node on the path keeps track of the one that came before it.
- A* will always find a solution if one exists.

Mission Planning in Autonomous Vehicle - Lane Graph Cost

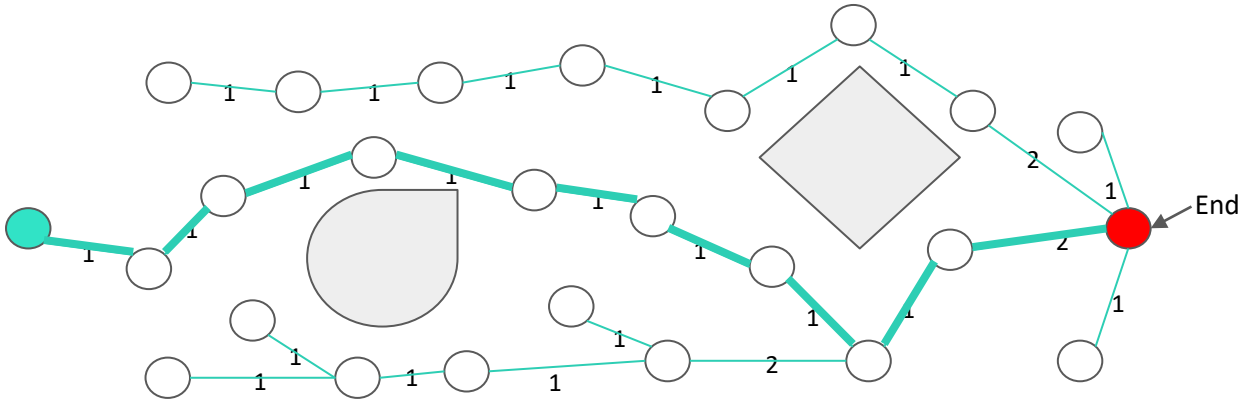
Mission Planning

Tree:

A finite set of Vertices(or nodes) and set of Edges which connect a pair of nodes, where each node has a unique parent.

There's no need to search a tree, instead tree construction algorithms focus on how to generate or select edges which result in good paths (more on this for local planning).

Eg. Shortest path tree to end from any start node (backward tree expansion)



Mission Planning: Data Structures

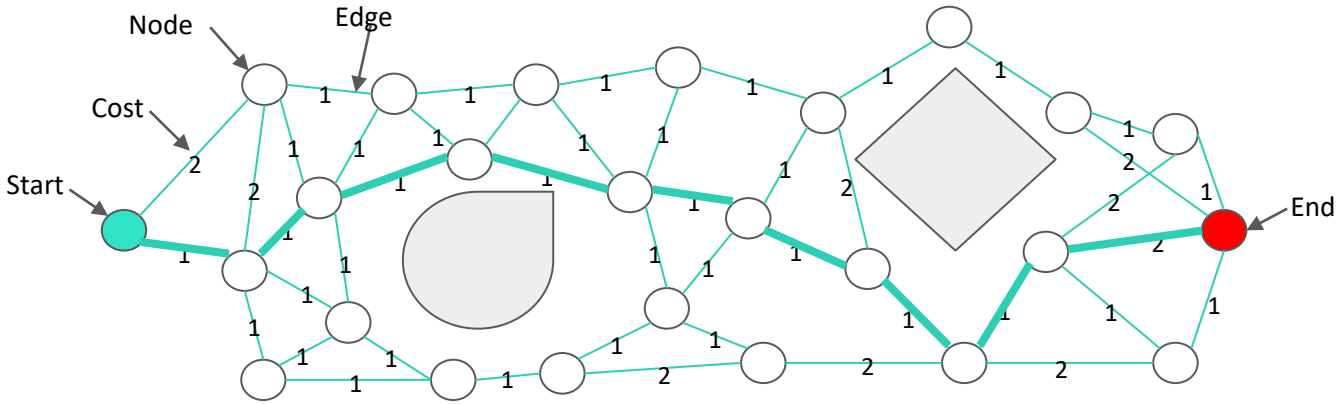
Graph:

A finite set of Vertices(or nodes) and set of Edges which connect a pair of nodes.

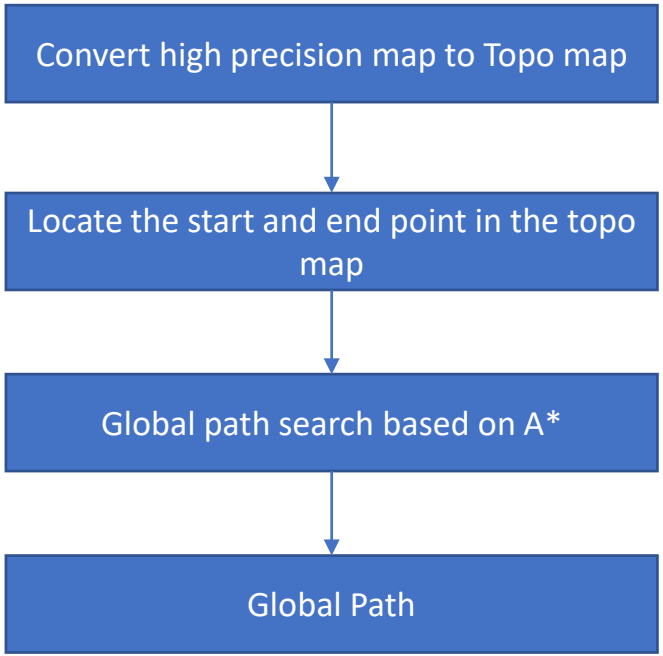
Graph Search Algorithms:

Graph Search Algorithms attempt to find lowest cost solution (sequence of edges).

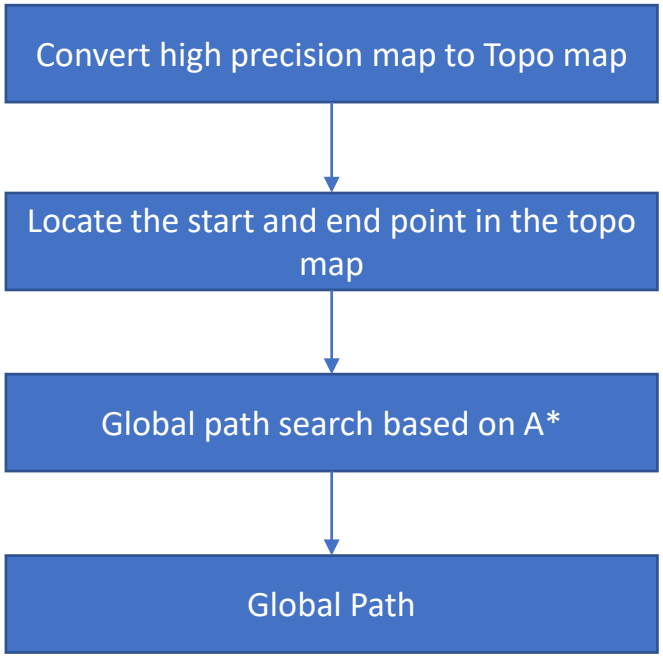
Basic algorithms like Djikstra's or A* (heuristic guided) have many variants,



Mission Planning



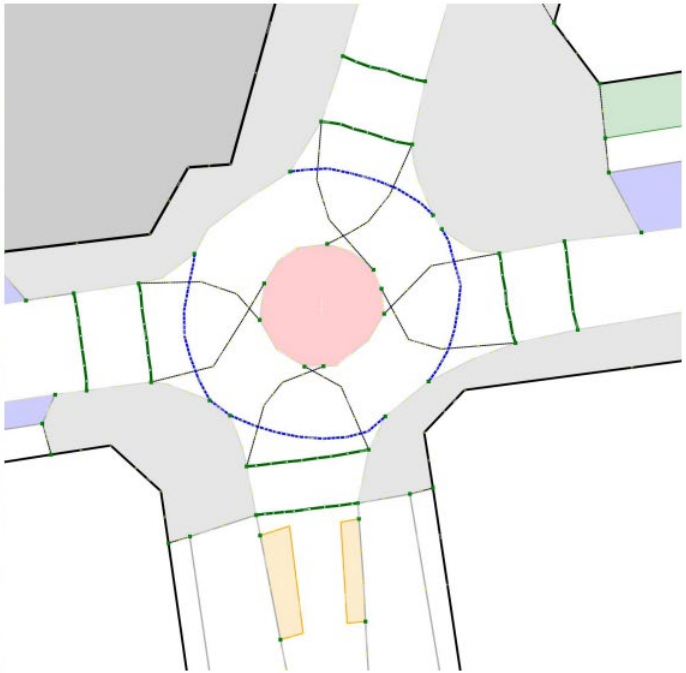
Mission Planning



Mission Planning

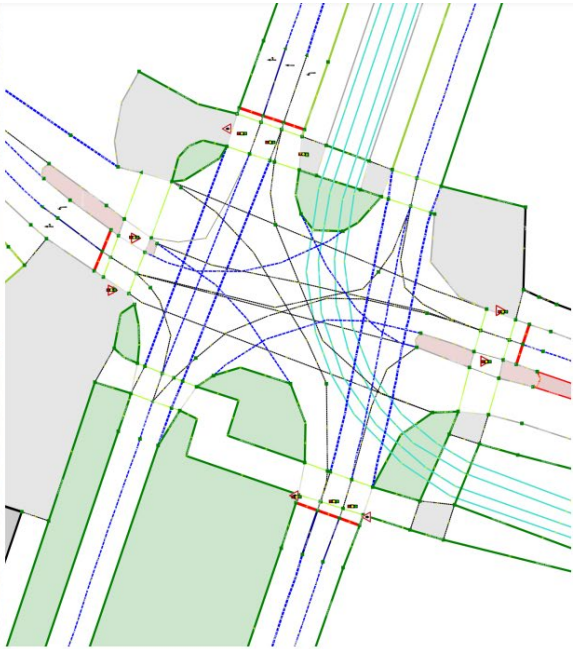
Map example for a roundabout road and resulting topo map.

Satellite Map



Topo Map

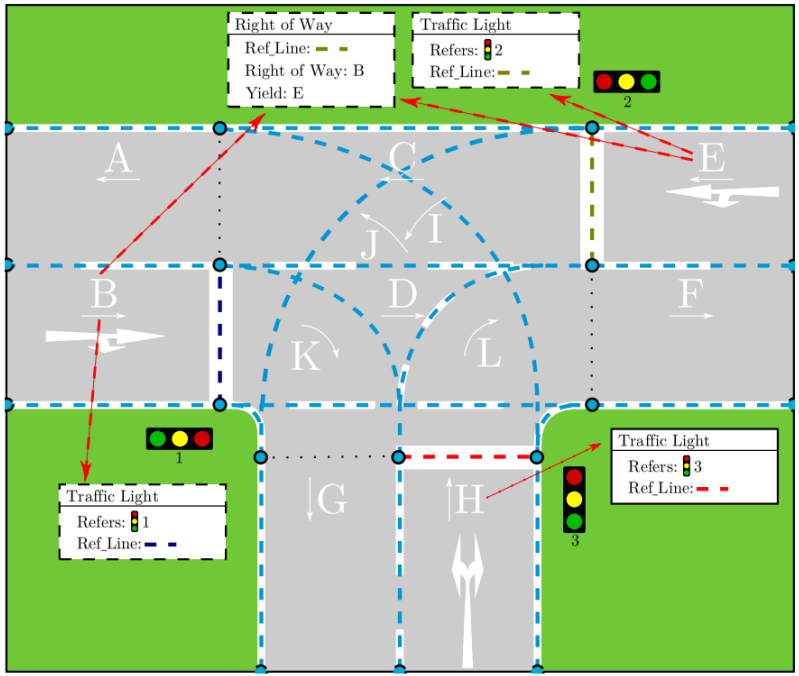
Mission Planning

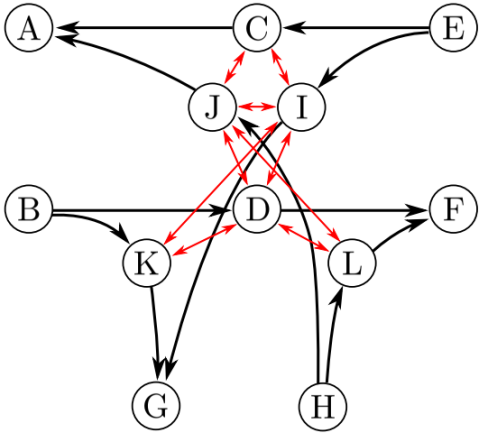


Satellite Map → Topo Map

Mission Planning

Map example for an urban road (intersection) and resulting map structure.

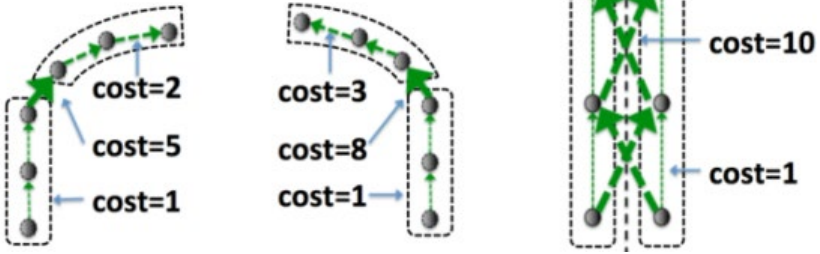




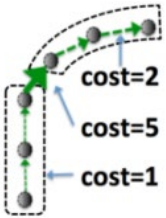
Mission Planning

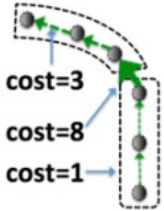
Cost of road graph

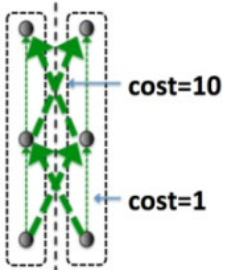
Maneuverer	Cost
Go straight and then turn right	5
Go straight and then turn left	8
During right turn	2
During left turn	3
Stay within the lane	1
Lane change	10



Mission Planning

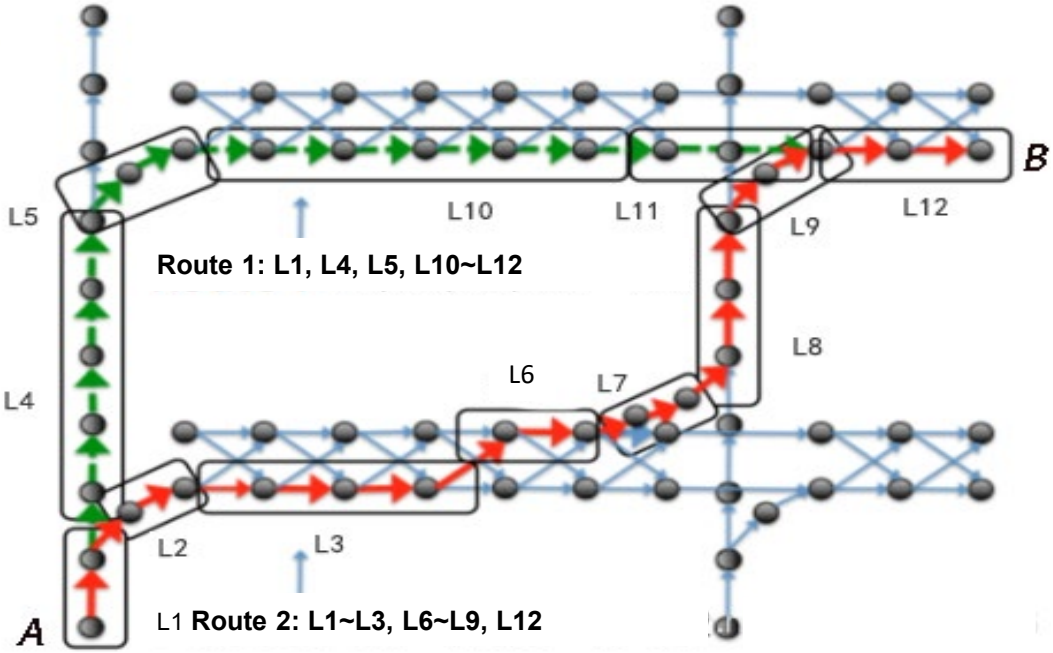




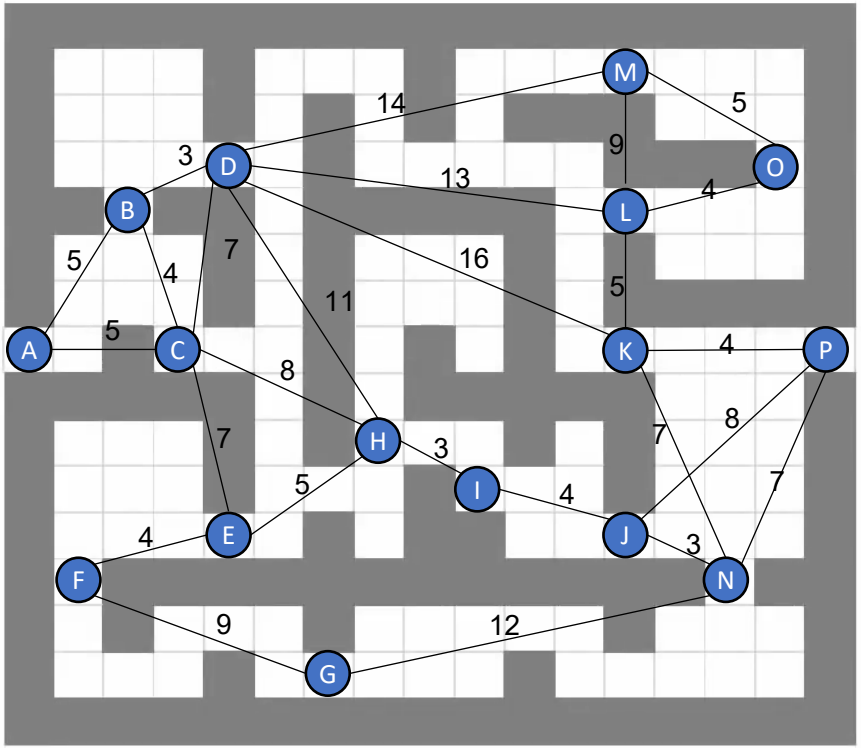


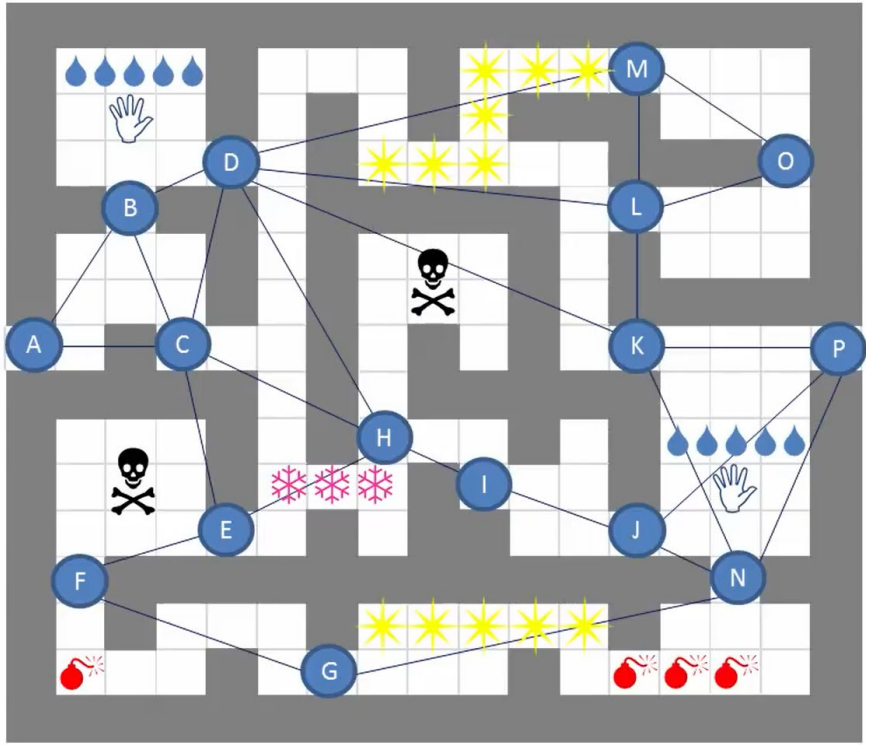
Route 1 Cost = 22

Route 2 Cost = 44



Mission Planning





What if we add more consideration of each path?

- The heuristic distance is changed
- Therefore, final path may be different

Mission Planning: Summary

Mission Planning (Route planning)

- What to Search?
- How to Search?
- What do we get from this functional module?

深蓝学院
shenlanxueyuan.com

Thanks for Listening !