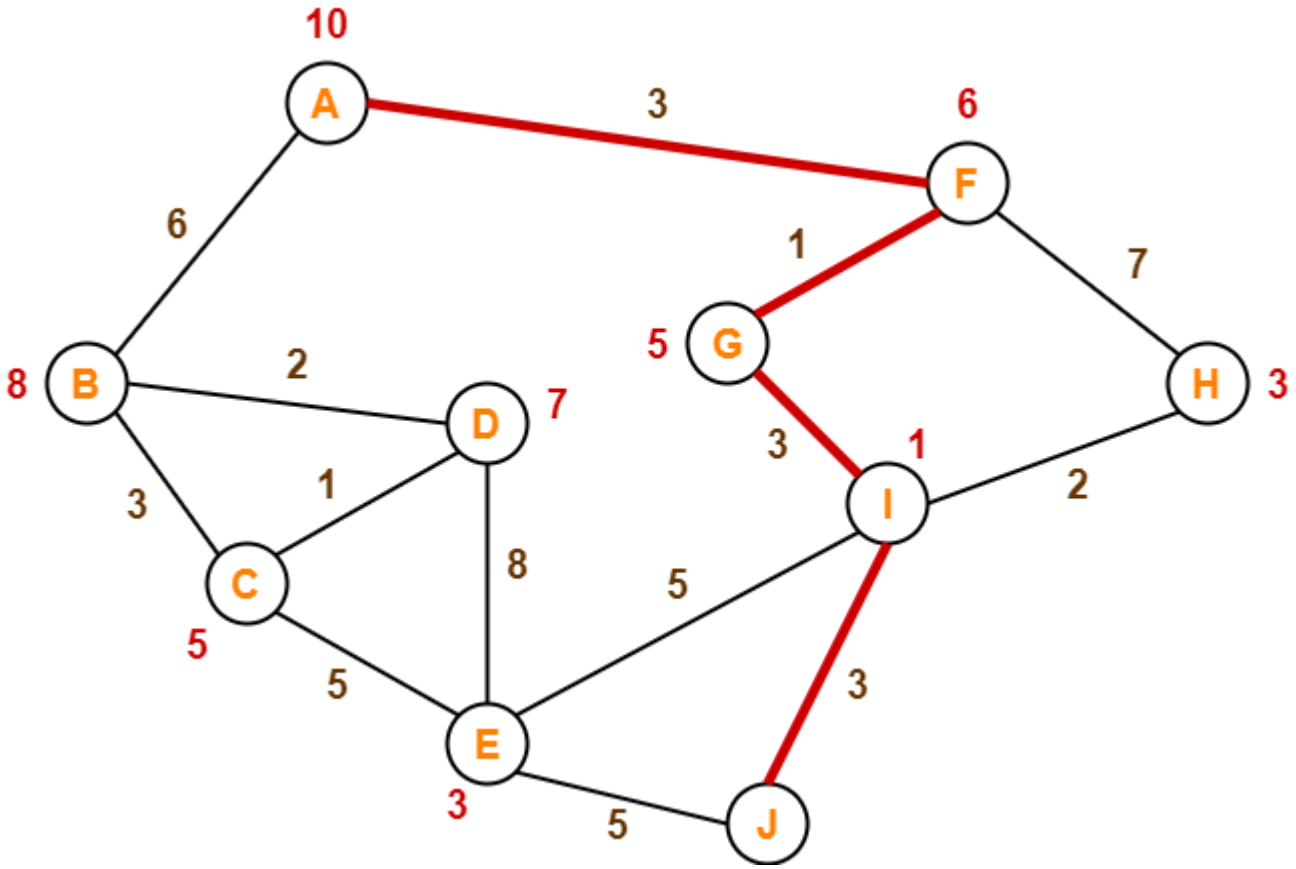
A* Algorithm

The most important advantage of A* search algorithm which separates it from other traversal techniques is that it has a brain. This makes A* very smart and pushes it much ahead of other conventional algorithms

1. GENERATE A LIST of all possible next steps towards goal from current position
2. STORE CHILDREN in priority queue based on distance to goal, closest first
3. SELECT CLOSEST child and REPEAT until goal reached or no more children




```
from collections import deque

class Graph:
    def __init__(self, adjac_lis):
        self.adjac_lis = adjac_lis

    def get_neighbors(self, v):
        return self.adjac_lis[v]

    # This is heuristic function which is having equal values for all nodes
    def h(self, n):
        H = {
```

```
            'A': 1,
            'B': 1,
            'C': 1,
            'D': 1
        }

        return H[n]

    def a_star_algorithm(self, start, stop):
        # In this open_lst is a lisy of nodes which have been visited, but who's
        # neighbours haven't all been always inspected, It starts off with the start
#node
        # And closed_lst is a list of nodes which have been visited
        # and who's neighbors have been always inspected
        open_lst = set([start])
        closed_lst = set([])

        # poo has present distances from start to all other nodes
        # the default value is +infinity
        poo = {}
        poo[start] = 0

        # par contains an adjac mapping of all nodes
        par = {}
        par[start] = start

        while len(open_lst) > 0:
            n = None

            # it will find a node with the lowest value of f() -
            for v in open_lst:
                if n == None or poo[v] + self.h(v) < poo[n] + self.h(n):
                    n = v;

            if n == None:
                print('Path does not exist!')
                return None

            # if the current node is the stop
            # then we start again from start
            if n == stop:
                reconst_path = []

                while par[n] != n:
                    reconst_path.append(n)
                    n = par[n]

                reconst_path.append(start)

                reconst_path.reverse()

                print('Path found: {}'.format(reconst_path))
```

```
                return reconst_path

        # code here

        print('Path does not exist!')
        return None


adjac_lis = {
    'A': [('B', 1), ('C', 3), ('D', 7)],
    'B': [('D', 5)],
    'C': [('D', 12)]
}
graph1 = Graph(adjac_lis)
graph1.a_star_algorithm('A', 'D')
```

```
 Path found: ['A', 'B', 'D']
 ['A', 'B', 'D']
```

** Task**

Complete A star algorithm and display the path found

```
#code here
```