Program 1
Implement A* Search algorithm

- A* algorithm is an Artifical intelligence problem used for pathfinding and graph traversals.
- This is the advanced form of BFS algorithm, which searches for the shorter path first then, the longer paths.
- One important aspect of A* is f = g + h. The f, g, and h variables are in our Node class and get calculated every time we create a new node.
  - G is the cost of moving from the start node to the current node. Basically, it is the sum of all the nodes that have been visited since leaving the start node.
  - h is called as the Heuristic value, it is estimated cost of moving from the current node to the final node. Since, the actual cost cannot be calculated until the final cell is reached.
  - F is the sum of g and h

Algorithm

```
1) Create an open list and a closed list. Put the starting node on the open list
2) While the open list is not empty
    (a) Find the node with the least f on the open list, call it “q”
    (b) Pop q off the open list
    (c) Generate q’s neighbors and set their parent to q
    (d) For each neighbor
        i)   if neighbor is the stop-node, then stop the search
        ii)  else, compute both the g and h for neighbor
             neighbor.g = q.g + weight
             neighbor.h = distance from neighbor to the stop-node
             neighbor.f = neighbor.g + neighbor.h
        iii) if a node with the same position as neighbor is in the open list which has lower
             f than neighbor, skip this neighbor
        iv)  if a node with the same position as neighbor is in the closed list which has a
             lower f than the neighbor, skip this neighbor
        End loop
    (e) Push q on the closed list
End Loop
```

Program

```
def AStart(startnode, stopnode):
    openlist = set(startnode)
    closelist = set()
    g = {}  # Stores the heurisitc values
    g[startnode] = 0
    parents = {}
    parents[startnode] = startnode
    while len(openlist) > 0:
        node = None
        for vertex in openlist:
            if node == None or g[vertex] + heuristic(vertex) < g[node] < heuristic(node):
                node = vertex

        if node == stopnode or Graph_nodes[node] == None:
            pass

        else:
            for (neighbor, weight) in get_neighbors(node):
                if neighbor not in openlist and neighbor not in closelist:
                    openlist.add(neighbor)
                    parents[neighbor] = node
                    g[neighbor] = g[node] + weight
                else:
                    if g[neighbor] > g[node] + weight:
                        g[neighbor] = g[node] + weight
                        parents[neighbor] = node
                        if neighbor in closelist:
                            closelist.remove(neighbor)
                            openlist.add(neighbor)

        if node == None:
            print('Path not found')
            return None

        if node == stopnode:
            path = []
            while parents[node] != node:
                path.append(node)
                node = parents[node]
            path.append(startnode)
            path.reverse()
            print('Path found: {}'.format(path))
            return path

        openlist.remove(node)
        closelist.add(node)
    print('Path doesn\'t exist')
    return None


def get_neighbors(vertex):
    if vertex in Graph_nodes:
        return Graph_nodes[vertex]
    else:
        return None
```

```
def heuristic(node):
    H_dist = {
        'A': 10,
        'B': 8,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }
    return H_dist[node]


Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('C', 3), ('D', 2)],
    'C': [('D', 1), ('E', 5)],
    'D': [('C', 1), ('E', 8)],
    'E': [('I', 5), ('J', 5)],
    'F': [('G', 1), ('H', 7)],
    'G': [('I', 3)],
    'H': [('I', 2)],
    'I': [('E', 5), ('J', 3)],
}

AStart('A', ‘J')
```

```
Path found: ['A', 'F', 'G', 'I', 'J']
Process finished with exit code 0
```