In [10]:

```
def aStarAlgo(start_node, stop_node):

        open_set = set(start_node)
        closed_set = set()
        g = {} #store distance from starting node
        parents = {}# parents contains an adjacency map of all nodes

        #ditance of starting node from itself is zero
        g[start_node] = 0
        #start_node is root node i.e it has no parent nodes
        #so start_node is set to its own parent node
        parents[start_node] = start_node


        while len(open_set) > 0:
            n = None

            #node with lowest f() is found
            for v in open_set:
                if n == None or g[v] + heuristic(v) < g[n] + heuristic(n)
                    n = v


            if n == stop_node or Graph_nodes[n] == None:
                pass
            else:
                for (m, weight) in get_neighbors(n):
                    #nodes 'm' not in first and last set are added to firs
                    #n is set its parent
                    if m not in open_set and m not in closed_set:
                        open_set.add(m)
                        parents[m] = n
                        g[m] = g[n] + weight


                    #for each node m,compare its distance from start i.e g
                    #from start through n node
                    else:
                        if g[m] > g[n] + weight:
                            #update g(m)
                            g[m] = g[n] + weight
                            #change parent of m to n
                            parents[m] = n

                            #if m in closed set,remove and add to open
                            if m in closed_set:
                                closed_set.remove(m)
                                open_set.add(m)

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

            # if the current node is the stop_node
            # then we begin reconstructin the path from it to the start_no
            if n == stop_node:
                path = []
```

```
58
59                 while parents[n] != n:
60                     path.append(n)
61                     n = parents[n]
62
63                 path.append(start_node)
64
65                 path.reverse()
66
67                 print('Path found: {}'.format(path))
68                 return path
69
70
71             # remove n from the open_list, and add it to closed_list
72             # because all of his neighbors were inspected
73             open_set.remove(n)
74             closed_set.add(n)
75
76         print('Path does not exist!')
77         return None
78
79 #define fuction to return neighbor and its distance
80 #from the passed node
81 def get_neighbors(v):
82     if v in Graph_nodes:
83         return Graph_nodes[v]
84     else:
85         return None
86 #for simplicity we ll consider heuristic distances given
87 #and this function returns heuristic distance for all nodes
88 def heuristic(n):
89         H_dist = {
90             'A': 11,
91             'B': 6,
92             'C': 99,
93             'D': 1,
94             'E': 7,
95             'G': 0,
96
97         }
98
99         return H_dist[n]
100
101 #Describe your graph here
102 Graph_nodes = {
103     'A': [('B', 2), ('E', 6)],
104     'B': [('C', 1),('G', 9)],
105     'C': [('D', 3)],
106     'E': [('D', 6)],
107     'D': [('G', 1)],
108
109 }
110 aStarAlgo('A', 'G')
```

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

Out[10]: ['A', 'B', 'G']