In [10]:

```
from collections import deque

class Graph:
    def __init__(self, adjac_lis): #Constructer to initialise the list
        self.adjac_lis = adjac_lis

    def get_neighbors(self, v): #Function to get the neighbors
        return self.adjac_lis[v]

    def h(self, n): #Heuristic function which is having equal values for all nodes
        H = {'A': 1,'B': 1,'C': 1,'D': 1}
        return H[n]

    def A_Star(self, start, stop):
        open_lst = set([start]) #List of nodes which have been visited, but who's
        closed_lst = set([])  #List of nodes which have been visited and who's neig
        poo = {} #poo has present distances from start to all other nodes
        poo[start] = 0
        par = {} #par contains an adjac mapping of all nodes
        par[start] = start

        while len(open_lst) > 0: #While the open_lst is not empty
            n = None

            #Finding 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

            for (m, weight) in self.get_neighbors(n): # Checking for neighbors in
                if m not in open_lst and m not in closed_lst: #If the current node
                    open_lst.add(m)
                    par[m] = n
                    poo[m] = poo[n] + weight

                else: #Else check if it's quicker to first visit n, then m and if
                    if poo[m] > poo[n] + weight:
                        poo[m] = poo[n] + weight
                        par[m] = n

                        if m in closed_lst: #if the node was in the closed_lst, mo
                            closed_lst.remove(m)
```

```
                            open_lst.add(m)

                #Remove n from the open_lst, and add it to closed_lst because all of h
                open_lst.remove(n)
                closed_lst.add(n)

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

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

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

Out[10]:
```
['A', 'B', 'D']
```