Implement A* Search algorithm.

```
class Graph:
    def __init__(self,adjac_lis):
        self.adjac_lis = adjac_lis
    def get_neighbours(self,v):
        return self.adjac_lis[v]
    def h(self,n):
        H={'A':1,'B':1, 'C':1,'D':1}
        return H[n]
    def a_star_algorithm(self,start,stop):
        open_lst = set([start])
        closed_lst = set([])
        dist ={}
        dist[start] = 0
        prenode ={}
        prenode[start] =start
        while len(open_lst)>0:
            n = None
            for v in open_lst:
                if n==None or
dist[v]+self.h(v)<dist[n]+self.h(n):
                    n=v;
            if n==None:
                print("path doesnot exist")
```

```
            return None
        if n==stop:
            reconst_path=[]
            while prenode[n]!=n:
                reconst_path.append(n)
                n = prenode[n]
            reconst_path.append(start)
            reconst_path.reverse()
            print("path found:{}".format(reconst_path))
            return reconst_path
        for (m,weight) in self.get_neighbours(n):
            if m not in open_lst and m not in closed_lst:
                open_lst.add(m)
                prenode[m] = n
                dist[m] = dist[n]+weight
            else:
                if dist[m]>dist[n]+weight:
                    dist[m] = dist[n]+weight
                    prenode[m]=n
                    if m in closed_lst:
                        closed_lst.remove(m)
                        open_lst.add(m)
        open_lst.remove(n)
        closed_lst.add(n)
```

```
        print("Path doesnot 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')
```

Output

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

['A', 'B', 'D']
```