

In [1]:

```

1  class Graph:
2      def __init__(self,adjac_lis):
3          self.adjac_lis=adjac_lis
4      def get_neighbors(self,v):
5          return self.adjac_lis[v]
6      def h(self,n):
7          H={
8              'A':1,
9              'B':1,
10             'C':1,
11             'D':1,
12         }
13         return H[n]
14
15     def a_star_algorithm(self,start,stop):
16         open_lst=set([start])
17         closed_lst=set([])
18         poo={}
19         poo[start]=0
20         par={}
21         par[start]=start
22         while len(open_lst)>0:
23             n=None
24             for v in open_lst:
25                 if n==None or poo[v]+self.h(v)<poo[n]+self.h(n):
26                     n=v;
27             if n==None:
28                 print('Path does not exist!')
29                 return None
30             if n==stop:
31                 reconst_path=[]
32                 while par[n]!=n:
33                     reconst_path.append(n)
34                     n=par[n]
35                 reconst_path.append(start)
36                 reconst_path.reverse()
37                 print('Path found:{}'.format(reconst_path))
38                 return reconst_path
39
40             for(m,weight) in self.get_neighbors(n):
41                 if m not in open_lst and m not in closed_lst:

```

```
42         open_lst.add(m)
43         par[m]=n
44         poo[m]=poo[n]+weight
45     else:
46         if poo[m]>poo[n]+weight:
47             poo[m]=poo[n]+weight
48             par[m]=n
49             if m in closed_lst:
50                 closed_lst.remove(m)
51                 open_lst.add(m)
52         open_lst.remove(n)
53         closed_lst.add(n)
54         print('Path does not exist!')
55         return None
56 adjac_lis={
57     'A':[( 'B',1),('C',3),('D',7)],
58     'B':[( 'D',5)],
59     'C':[( 'D',12)]
60 }
61 graph1=Graph(adjac_lis)
62 graph1.a_star_algorithm('A','D')
```

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

Out[1]: ['A', 'B', 'D']

In [2]:

```

1  class Graph:
2      def __init__(self,adjac_lis):
3          self.adjac_lis=adjac_lis
4      def get_neighbors(self,v):
5          return self.adjac_lis[v]
6      def h(self,n):
7          H={
8              'A':1,
9              'B':1,
10             'C':1,
11             'D':1,
12             'E':1,
13             'F':1,
14             'G':1,
15             'H':1,
16             'I':1
17         }
18         return H[n]
19
20     def a_star_algorithm(self,start,stop):
21         open_lst=set([start])
22         closed_lst=set([])
23         poo={}
24         poo[start]=0
25         par={}
26         par[start]=start
27         while len(open_lst)>0:
28             n=None
29             for v in open_lst:
30                 if n==None or poo[v]+self.h(v)<poo[n]+self.h(n):
31                     n=v;
32             if n==None:
33                 print('Path does not exist!')
34                 return None
35             if n==stop:
36                 reconst_path=[]
37                 while par[n]!=n:
38                     reconst_path.append(n)
39                     n=par[n]
40                 reconst_path.append(start)
41                 reconst_path.reverse()

```

```

42         print('Path found:{}'.format(reconst_path))
43         return reconst_path
44
45     for(m,weight) in self.get_neighbors(n):
46         if m not in open_lst and m not in closed_lst:
47             open_lst.add(m)
48             par[m]=n
49             poo[m]=poo[n]+weight
50         else:
51             if poo[m]>poo[n]+weight:
52                 poo[m]=poo[n]+weight
53                 par[m]=n
54                 if m in closed_lst:
55                     closed_lst.remove(m)
56                     open_lst.add(m)
57             open_lst.remove(n)
58             closed_lst.add(n)
59     print('Path does not exist!')
60     return None
61 adjac_lis={
62     'A':[( 'B',1),( 'C',4),( 'E',5)],
63     'B':[( 'A',1),( 'D',2)],
64     'C':[( 'A',4),( 'D',3),( 'F',2),( 'H',3)],
65     'D':[( 'B',2),( 'C',3),( 'F',1)],
66     'E':[( 'A',5),( 'G',3)],
67     'F':[( 'D',1),( 'C',2),( 'H',2)],
68     'G':[( 'E',3),( 'H',2)],
69     'H':[( 'C',3),( 'F',2),( 'I',1),( 'G',2)],
70     'I':[( 'H',1)],
71 }
72 graph1=Graph(adjac_lis)
73 graph1.a_star_algorithm('A','F')

```

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

Out[2]: ['A', 'B', 'D', 'F']