A* SEARCH ALGORITHM

PROGRAM:

```
from collections import deque
class Graph:
def __init__(self, adjacency_list):
self.adjacency_list = adjacency_list
def get_neighbors(self, v):
return self.adjacency_list[v]
def h(self, n):
H = {
&#39;A&#39;: 1,
&#39;B&#39;: 1,
&#39;C&#39;: 1,
&#39;D&#39;: 1
}
return H[n]
def a_star_algorithm(self, start_node, stop_node):
open_list = set([start_node])
closed_list = set([])
g = {}
g[start_node] = 0
parents = {}
parents[start_node] = start_node

PRINCIPLES OF ARTIFICIAL INTELLIGENCE
while len(open_list) &gt; 0:
n = None
for v in open_list:
if n == None or g[v] + self.h(v) &lt; g[n] + self.h(n):
n = v
if n == None:
print(&#39;Path does not exist!&#39;)
return None
if n == stop_node:
```

```
reconst_path = []
while parents[n] != n:
reconst_path.append(n)
n = parents[n]
reconst_path.append(start_node)
reconst_path.reverse()
print(&#39;Path found: {}&#39;.format(reconst_path))
return reconst_path
for (m, weight) in self.get_neighbors(n):
if m not in open_list and m not in closed_list:
open_list.add(m)
parents[m] = n
g[m] = g[n] + weight
else:
if g[m] &gt; g[n] + weight:
g[m] = g[n] + weight
parents[m] = n
if m in closed_list:
closed_list.remove(m)
open_list.add(m)
open_list.remove(n)
closed_list.add(n)
print(&#39;Path does not exist!&#39;)
return None
# Sample I/O
adjacency_list = {
&#39;A&#39;: [(&#39;B&#39;, 1), (&#39;C&#39;, 3), (&#39;D&#39;, 7)],
&#39;B&#39;: [(&#39;D&#39;, 5)],
&#39;C&#39;: [(&#39;D&#39;, 12)]
}
graph1 = Graph(adjacency_list)
graph1.a_star_algorithm(&#39;A&#39;, &#39;D&#39;)
```

OUTPUT:

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