```python
"""AI&ML
Lab-2
Implement AO* Search Algorithm"""

import time
import os

def get_node(mark_road,extended):
    temp=[0]
    i=0
    while 1:
        current=temp[i]
        if current not in extended:
            return current
        else:
            for child in mark_road[current]:
                if child not in temp:
                    temp.append(child)
            i+=1
def get_current(s,nodes_tree):
    if len(s)==1:
        return s[0]
    for node in s:
        flag=True
        for edge in nodes_tree(node):
            for child_node in edge:
                if child_node in s:
                    flag=False
        if flag:
            return node
def get_pre(current,pre,pre_list):
    if current==0:
        return
```

```
34      for pre_node in pre[current]:
35          if pre_node not in pre_list:
36              pre_list.append(pre_node)
37          get_pre(pre_node,pre,pre_list)
38      return
39  def ans_print(mark_rode,node_tree):
40      print("The final connection is as follow:")
41      temp=[0]
42      while temp:
43          time.sleep(1)
44          print(f"[{temp[0]}]----->{mark_rode[temp[0]]}")
45          for child in mark_rode[temp[0]]:
46              if node_tree[child]!=[[child]]:
47                  temp.append(child)
48          temp.pop(0)
49      time.sleep(5)
50      os.system('cls')
51      return
52  def AOstar(node_trees,h_val):
53      futility=0xfff
54      extended=[]
55      choice=[]
56      mark_rode={0:None}
57      solved={}
58      pre={0:[]}
59      for i in range(1,9):
60          pre[i]=[]
61      for i in range(len(nodes_tree)):
62          solved[i]=False
63      os.system('cls')
64      print("The connection process is as follows")
65      time.sleep(1)
66      while not solved[0] and h_val[0]<futility:
```

```
        node=get_node(mark_rode,extended)
        extended.append(node)
        if nodes_tree[node] is None:
            h_val[node]=futility
            continue
        for suc_edge in nodes_tree[node]:
            for suc_node in suc_edge:
                if nodes_tree[suc_node]==[[suc_node]]:
                    solved[suc_node]=True
        s=[node]
        while s:
            current=get_current(s,nodes_tree)
            s.remove(current)
            origen_h=h_val[current]
            origen_s=solved[current]
            min_h=0xfff
            for edge in nodes_tree[current]:
                edge_h=0
                for node in edge:
                    edge_h+=h_val[node]+1
                if edge_h<min_h:
                    min_h=edge_h
                    h_val[current]=min_h
                    mark_rode[current]=edge
            if mark_rode[current] not in choice:
                choice.append(mark_rode[current])
                print(f"[{current}]---{mark_rode[current]}")
                time.sleep(1)
            for child_node in mark_rode[current]:
                pre[child_node].append(current)
            solved[current]=True
            for node in mark_rode[current]:
                solved[current]=solved[current] and solved[node]
```

```
            if origen_s!=solved[current] or origen_h!=h_val[current]:
                pre_list=[]
                if current!=0:
                    get_pre(current,pre,pre_list)
                s.extend(pre_list)
    if not solved[0]:
        print("The query failed, the path could not be found!")
    else:
        ans_print(mark_rode,nodes_tree)
    return

if __name__=="__main__":
    nodes_tree={}
    nodes_tree[0]=[[1],[4,5]]
    nodes_tree[1]=[[2],[3]]
    nodes_tree[2]=[[3],[2,5]]
    nodes_tree[3]=[[5,6]]
    nodes_tree[4]=[[5],[8]]
    nodes_tree[5]=[[6],[7,8]]
    nodes_tree[6]=[[7,8]]
    nodes_tree[7]=[[7]]
    nodes_tree[8]=[[8]]
    h_val=[3,2,4,4,1,1,2,0,0]
    AOstar(nodes_tree,h_val)

"""
Output:
The connection process is as follows
[0]---[1]
[1]---[2]
[0]---[4, 5]
[4]---[8]
[5]---[7, 8]
```

```
133 The final connection is as follow:
134 [0]----->[4, 5]
135 [4]----->[8]
136 [5]----->[7, 8]
137 """
```