Department of Computer Science and Engineering (Data Science)

Subject: Artificial Intelligence (DJ19DSC502)

AY: 2023-24

NAME:DIVYESH KHUNT **SAPID:60009210116** **BATCH:D12**

Experiment 10

(Planning)

Aim: Implement a plan using AO*.

Theory:

The Depth-first search and Breadth-first search given earlier for OR trees or graphs can be easily adopted by AND-OR graph. The main difference lies in the way termination conditions are determined since all goals following an AND node must be realized; whereas a single goal node following an OR node will do. So for this purpose, we are using AO* algorithm. Like A* algorithm here we will use two arrays and one heuristic function.

OPEN: It contains the nodes that have been traversed but yet not been marked solvable or unsolvable.

CLOSE: It contains the nodes that have already been processed.

AO* Search Algorithm

Step 1: Place the starting node into OPEN.

Step 2: Compute the most promising solution tree say T0.

Step 3: Select a node n that is both on OPEN and a member of T0. Remove it from OPEN and place it in CLOSE

Step 4: If n is the terminal goal node then leveled n as solved and leveled all the ancestors of n as solved. If the starting node is marked as solved then success and exit.

Step 5: If n is not a solvable node, then mark n as unsolvable. If starting node is marked as unsolvable, then return failure and exit.

Step 6: Expand n. Find all its successors and find their h (n) value, push them into OPEN.

Step 7: Return to Step 2.

Step 8: Exit.

Lab Assignment to do:

Consider the use case of a plan to travel from Mumbai to Goa to attend a wedding at Taj Aguada. The plan needs to be decided based on the cost. You can either travel by train or bus or flight and stay in a hotel near or far to the wedding venue. The three options of the venues are Westin, Kennel Worth and Maria Rica hotels. You can choose between a two days package for stay and meal together or separately. Other option for your travel and stay will be a vanity van. There you need to decide if you want to cook or eat outside.

Implement AO* to find the most suitable plan in terms of cost.

```python
H = {'GOA': -1, 'TRAVEL': 8, 'STAY': 8, 'AREOPLANE': 4, 'TRAIN': 6, 'BUS': 7, 'FAR': 10, 'NEAR': 4, '3STAR':4, '5STAR':6,'WITHFOOD':0,'NOFOOD':0}

Conditions = {
 'GOA': { 'AND': ['TRAVEL', 'STAY']},
 'TARVEL': {'OR': ['AREOPLANE', 'TRAIN','BUS']},
 'STAY': {'OR': ['FAR','NEAR']},
 'FAR': {'OR': ['3STAR','5STAR']},
 'NEAR': {'OR': ['3STAR','5STAR']},
 '3STAR': {'OR': ['WITHFOOD','NOFOOD']}
}
```

```python
def Cost(H, condition, weight = 1):
  cost = {}
  if 'AND' in condition:
    AND_nodes = condition['AND']
    Path_A = ' AND '.join(AND_nodes)
    PathA = sum(H[node]+weight for node in AND_nodes)
    cost[Path_A] = PathA

  if 'OR' in condition:
    OR_nodes = condition['OR']
    Path_B =' OR '.join(OR_nodes)
    PathB = min(H[node]+weight for node in OR_nodes)
    cost[Path_B] = PathB
  return cost

def update_cost(H, Conditions, weight=1):
  Main_nodes = list(Conditions.keys())
  Main_nodes.reverse()
  least_cost= {}
  for key in Main_nodes:
    condition = Conditions[key]
    print(key,':', Conditions[key],'>>>', Cost(H, condition, weight))
    c = Cost(H, condition, weight)
    H[key] = min(c.values())
    least_cost[key] = Cost(H, condition, weight)
  return least_cost
```

```python
weight = 1
print('Updated Cost :')
Updated_cost = update_cost(H, Conditions, weight=1)
```

```
Updated Cost :
3STAR : {'OR': ['WITHFOOD', 'NOFOOD']} >>> {'WITHFOOD OR NOFOOD': 1}
NEAR : {'OR': ['3STAR', '5STAR']} >>> {'3STAR OR 5STAR': 2}
FAR : {'OR': ['3STAR', '5STAR']} >>> {'3STAR OR 5STAR': 2}
STAY : {'OR': ['FAR', 'NEAR']} >>> {'FAR OR NEAR': 3}
TARVEL : {'OR': ['AREOPLANE', 'TRAIN', 'BUS']} >>> {'AREOPLANE OR TRAIN OR BUS': 5}
GOA : {'AND': ['TRAVEL', 'STAY']} >>> {'TRAVEL AND STAY': 13}
```

```python
def shortest_path(Start,Updated_cost, H):
  Path = Start
  if Start in Updated_cost.keys():
    Min_cost = min(Updated_cost[Start].values())
    key = list(Updated_cost[Start].keys())
    values = list(Updated_cost[Start].values())
    Index = values.index(Min_cost)

    Next = key[Index].split()
    if len(Next) == 1:

      Start =Next[0]
      Path += '<--' +shortest_path(Start, Updated_cost, H)
    else:
      Path +='<--('+key[Index]+') '
      Start = Next[0]
      Path += '[' +shortest_path(Start, Updated_cost, H) + ' + '
      Start = Next[-1]
      Path += shortest_path(Start, Updated_cost, H) + ']'

  return Path

start_node='GOA'
path = shortest_path(start_node, Updated_cost, H)
print('Shortest Path:', path)
```

```
TAY) [TRAVEL + STAY<--(FAR OR NEAR) [FAR<--(3STAR OR 5STAR) [3STAR<--(WITHFOOD OR NOFOOD) [WITHFOOD + NOFOOD] + 5STAR] + NEAR<--(3STAR OR 5STAR) [3STAR<--(WITHFOOD OR NOFOOD) [WITHFOOD + NOFOOD] + 5STAR]]
```