Program 2
Implement AO* Search algorithm

Algorithm

- AO* algorithm uses the concept of AND-OR graphs to decompose any complex problems into smaller set of problems.
- It's a knowledge based searching technique, where the start and goal nodes is defined, and the best path is found using the heuristic values.
- A* gives the optimal solution. But AO* doesn't guarantee optimal solution. It doesn't explore all the solutions paths.

(1) Create a initial graph with a single node(start node)
(2) Traverse the graph following the current path, accumulating node that has not yet been expanded or solved
(3) Pick any of these nodes and expand it. If it has no successors call this value FUTILITY, otherwise calculate f' for each of the successors.
(4) If f' is 0, then mark the node as SOLVED
(5) Change the value of f' for the newly created node to reflect its successors by back propagation.
(6) Wherever possible use the most promising routes and if a node is marked as SOLVED them mark the parent node as SOLVED
(7) If starting node is SOLVED or value greater than FUTILITY, stop, else repeat from Step 2

Program

```
def recAOStar(n):
    global finalPath
    print("Expanding Node:", n)
    and_nodes = []
    or_nodes = []
    if (n in allNodes):
        if 'AND' in allNodes[n]:
            and_nodes = allNodes[n]['AND']
        if 'OR' in allNodes[n]:
            or_nodes = allNodes[n]['OR']
    if len(and_nodes) == 0 and len(or_nodes) == 0:
        return

    solvable = False
```

```python
    marked = {}

    while not solvable:
        if len(marked) == len(and_nodes) + len(or_nodes):
            min_cost_least, min_cost_group_least = least_cost_group(and_nodes, or_nodes, {})
            solvable = True
            change_heuristic(n, min_cost_least)
            optimal_child_group[n] = min_cost_group_least
            continue
        min_cost, min_cost_group = least_cost_group(and_nodes, or_nodes, marked)
        is_expanded = False
        if len(min_cost_group) > 1:
            if (min_cost_group[0] in allNodes):
                is_expanded = True
                recAOStar(min_cost_group[0])
            if (min_cost_group[1] in allNodes):
                is_expanded = True
                recAOStar(min_cost_group[1])
        else:
            if (min_cost_group in allNodes):
                is_expanded = True
                recAOStar(min_cost_group)
        if is_expanded:
            min_cost_verify, min_cost_group_verify = least_cost_group(and_nodes, or_nodes, {})
            if min_cost_group == min_cost_group_verify:
                solvable = True
                change_heuristic(n, min_cost_verify)
                optimal_child_group[n] = min_cost_group
        else:
            solvable = True
            change_heuristic(n, min_cost)
            optimal_child_group[n] = min_cost_group
        marked[min_cost_group] = 1
    return heuristic(n)


def least_cost_group(and_nodes, or_nodes, marked):
    node_wise_cost = {}
    for node_pair in and_nodes:
        if not node_pair[0] + node_pair[1] in marked:
            cost = 0
            cost = cost + heuristic(node_pair[0]) + heuristic(node_pair[1]) + 2
            node_wise_cost[node_pair[0] + node_pair[1]] = cost
    for node in or_nodes:
        if not node in marked:
            cost = 0
            cost = cost + heuristic(node) + 1
            node_wise_cost[node] = cost
    min_cost = 999999
    min_cost_group = None
    for costKey in node_wise_cost:
        if node_wise_cost[costKey] < min_cost:
            min_cost = node_wise_cost[costKey]
            min_cost_group = costKey
```

```python
    return [min_cost, min_cost_group]


def heuristic(n):
    return H_dist[n]


def change_heuristic(n, cost):
    H_dist[n] = cost
    return


def print_path(node):
    print(optimal_child_group[node], end="")
    node = optimal_child_group[node]
    if len(node) > 1:
        if node[0] in optimal_child_group:
            print("->", end="")
            print_path(node[0])
        if node[1] in optimal_child_group:
            print("->", end="")
            print_path(node[1])
    else:
        if node in optimal_child_group:
            print("->", end="")
            print_path(node)


H_dist = {
    'A': -1,
    'B': 4,
    'C': 2,
    'D': 3,
    'E': 6,
    'F': 8,
    'G': 2,
    'H': 0,
    'I': 0,
    'J': 0
}
allNodes = {
    'A': {'AND': [('C', 'D')], 'OR': ['B']},
    'B': {'OR': ['E', 'F']},
    'C': {'OR': ['G'], 'AND': [('H', 'I')]},
    'D': {'OR': ['J']}
}
optimal_child_group = {}
optimal_cost = recAOStar('A')
print('Nodes which gives optimal cost are')
print_path('A')
print('\nOptimal Cost is :: ', optimal_cost)
```

Result

Expanding Node: A
Expanding Node: B
Expanding Node: C
Expanding Node: D
Nodes which gives optimal cost are
CD->HI->J
Optimal Cost is :: 5