ALBUKHARY INTERNATIONAL UNIVERSITY

Artificial Intelligence

LAB SEVEN

Problem Solving Using Search

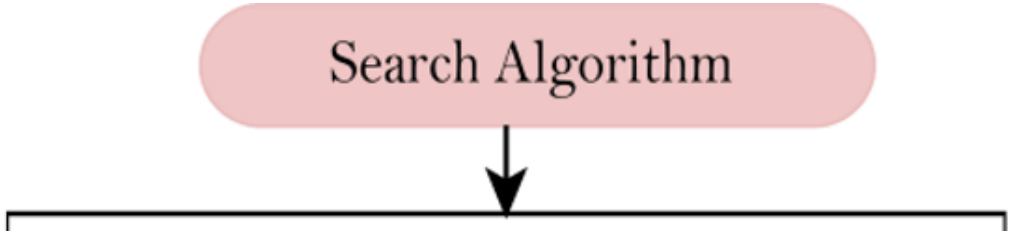
In general, an agent with several immediate options of unknown value can decide what to do by first examining different possible sequences of actions that lead to states of known value, and then choosing the best sequence. This process of looking for such a sequence is called search. A search algorithm takes a problem as input and returns a solution in the form of an action sequence. Once a solution is found, the actions it recommends can be carried out.

Types of search algorithms

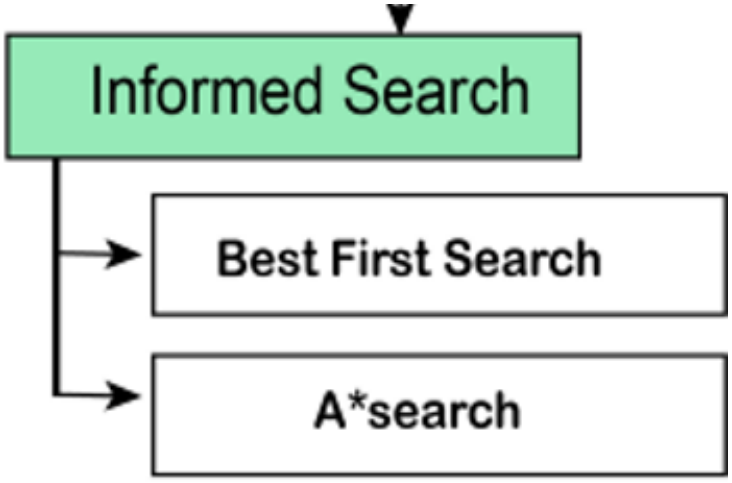
Based on the search problems we can classify the search algorithms into:

1. Uninformed (Blind search) search algorithms
2. Informed search (Heuristic search) algorithms.

As shown in the Figure below:



2. Informed search

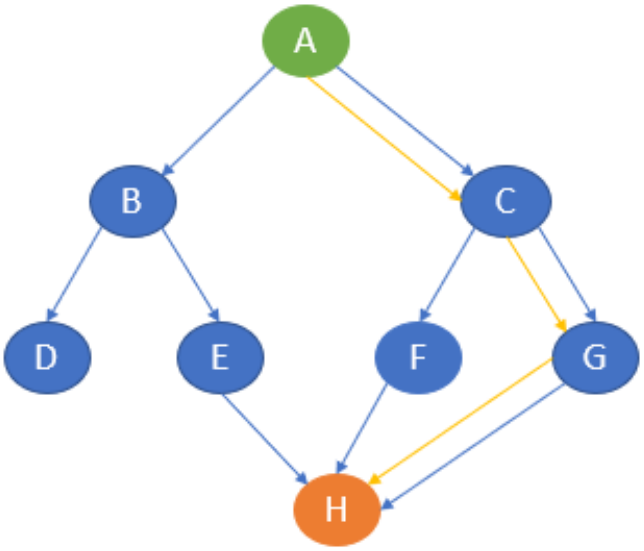


2.1 Best-first Search Algorithm (Greedy search)

Greedy best-first search uses the properties of both depth-first search and breadth-first search. Greedy best-first search traverses the node by selecting the path which appears best at the moment. The closest path is selected by using the heuristic function.

Here, A is the start node and H is the goal node.

Greedy best-first search first starts with A and then examines the next neighbour B and C. Here, the heuristics of B is 12 and C is 4. The best path at the moment is C and hence it goes to C. From C, it explores the neighbours F and G. the heuristics of F is 8 and G is 2. Hence it goes to G. From G, it goes to H whose heuristic is 0 which is also our goal state



In [ ]:

```
graph = {
'A':[('B',12), ('C',4)],
'B':[('D',7), ('E',3)],
'C':[('F',8), ('G',2)],
'D':[],
'E':[('H',0)],
'F':[('H',0)],
'G':[('H',0)]
}
def bfs(start, target, graph, queue=[], visited=[]):
    if start not in visited:
        print(start)
        visited.append(start)
    queue=queue+[x for x in graph[start] if x[0][0] not in visited]
    queue.sort(key=lambda x:x[1])
    if queue[0][0]==target:
        print(queue[0][0])
    else:
        processing=queue[0]
        queue.remove(processing)
        bfs(processing[0], target, graph, queue, visited)
bfs('A', 'H', graph)
```

2.2 A* search algorithm

A* search algorithm is a combination of both uniform cost search and greedy best-first search algorithms. It uses the advantages of both with better memory usage. It uses a heuristic function to find the shortest path. A* search algorithm uses the sum of both the cost and heuristic of the node to find the best path.

Let A be the start node and H be the goal node.

First, the algorithm will start with A. From A, it can go to B, C, H.

Note the point that A* search uses the sum of path cost and heuristics value to determine the path.

Here, from A to B, the sum of cost and heuristics is 1 + 3 = 4.

From A to C, it is 2 + 4 = 6.

From A to H, it is 7 + 0 = 7.

Here, the lowest cost is 4 and the path A to B is chosen. The other paths will be on hold.

Now, from B, it can go to D or E.

From A to B to D, the cost is 1 + 4 + 2 = 7.

From A to B to E, it is 1 + 6 + 6 = 13.

The lowest cost is 7. Path A to B to D is chosen and compared with other paths which are on hold.

Here, path A to C is of less cost. That is 6.

Hence, A to C is chosen and other paths are kept on hold.

From C, it can now go to F or G.

From A to C to F, the cost is 2 + 3 + 3 = 8.

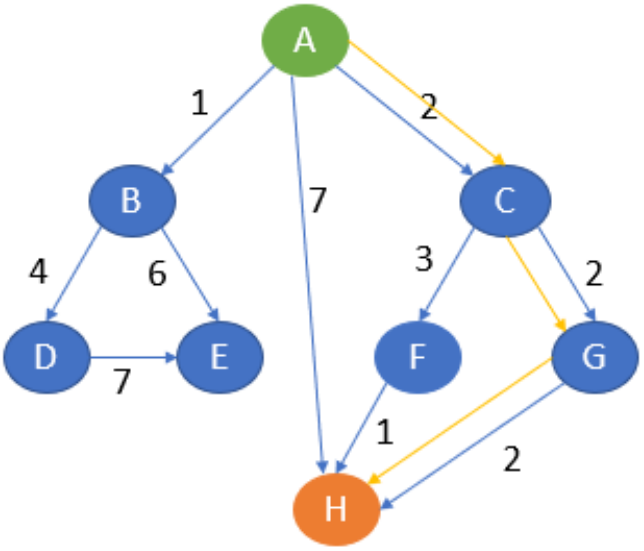
From A to C to G, the cost is 2 + 2 + 1 = 5.

The lowest cost is 5 which is also lesser than other paths which are on hold. Hence, path A to G is chosen.

From G, it can go to H whose cost is 2 + 2 + 2 + 0 = 6.

Here, 6 is lesser than other paths cost which is on hold.

Also, H is our goal state. The algorithm will terminate here.



In [ ]:

```
graph=[['A','B',1,3],
       ['A','C',2,4],
       ['A','H',7,0],
       ['B','D',4,2],
       ['B','E',6,6],
       ['C','F',3,3],
       ['C','G',2,1],
       ['D','E',7,6],
       ['D','H',5,0],
       ['F','H',1,0],
       ['G','H',2, 0]]
temp = []
temp1 = []
for i in graph:
    temp.append(i[0])
    temp1.append(i[1])
nodes = set(temp).union(set(temp1))
def A_star(graph, costs, open, closed, cur_node):
    if cur_node in open:
        open.remove(cur_node)
    closed.add(cur_node)
    for i in graph:
        if(i[0] == cur_node and costs[i[0]]+i[2]+i[3] < costs[i[1]]):
            open.add(i[1])
            costs[i[1]] =  costs[i[0]]+i[2]+i[3]
            path[i[1]] = path[i[0]] + ' -> ' + i[1]
    costs[cur_node] = 999999
    small = min(costs, key=costs.get)
    if small not in closed:
        A_star(graph, costs, open,closed, small)
costs = dict()
temp_cost = dict()
path = dict()
for i in nodes:
    costs[i] = 999999
    path[i] = ' '
open = set()
closed = set()
start_node = input("Enter the Start Node: ")
open.add(start_node)
path[start_node] = start_node
costs[start_node] = 0
A_star(graph, costs, open, closed, start_node)
goal_node = input("Enter the Goal Node: ")
print("Path with least cost is: ",path[goal_node])
```

In [ ]: