CS561/571 – Artificial Intelligence

Mid-Semester Quiz

Name: **M Maheeth Reddy**	Roll No.: **1801CS31**	Date: **20-Sep-2021**

Ans 1:

Breadth-First Search and Depth-first Search are Uninformed Search Algorithms but there are certain characteristics that make them complimentary.

Breadth-first Search (BFS) expands the fewest nodes among all admissible algorithms using the same cost function but typically requires exponential space.

Depth-first Search needs space only linear in the maximum search depth but expands more nodes than BFS.

Breadth First Search	**Depth First Search**
Strategy: expand shallowest node first	Strategy: expand deepest node first
Implementation: Fringe is a FIFO queue	Implementation: Frontier is a LIFO stack
BFS considers all neighbors so it is not suitable for decision trees used in puzzle games.	DFS is more suitable for decision trees. As with one decision, we need to traverse further to augment the decision. If we reach the conclusion, we won.

We use Depth-First Search when we know that Many solutions exist or to have a good estimate of the depth of solution whereas Breadth-First Search is used when some solutions are known to be shallow.

Ans 2:

Pseudo Code:

Assume source-node as the actual_state and the destination-node as the final_state

Maintain list Close that maintains nodes that are visited and also explored (visiting successors)

Also, maintain a PriorityQueue to store the heuristics of the present state and the node.

Put the actual_state in Priority Queue with the respective heuristics

While priority_queue is not empty:

```
  present_node = get the first node in the prio:rity queue which has the lowest cost
  if (present_node == final_state):
       print("SUCCESS")
       return
 else if(present_node not in Closed):
  Push present node into Closed
  Get all the successors of the node which we find by moving up, down, left, right
  for x in successors:
     if(x == final_state):
        print("SUCCESS")
        return
    Else if (x not in Closed) :
       Push the x   and heuristic of x into the priority_queue
print("Not Possible")
return
```

Suppose,

Start State				Goal State
1 2 3	-->	4 5 6	-->	1 2 3
4 5 6		7 0 8		4 0 6
7 8 0				7 5 8

h1 = 4

h2 = 5

Comparision of Heuristics:

For comparison of the heuristics we found h2 is better than h1 because the no.of states explore in optimal path h2 will be lesser than h1 and also no.of steps for h2 will comparatively less than h1. And it is also observed that time taken is h2 is better compared to h1

Ans 3:

Backward Chaining is more effective compared to Forward Chaining

If we recall Forward Chaining, we have to go through all the propositional logic linearly and match them with the existing database. We will perform a logic whenever it is satisfied and will update all the results to the database until goal is reached.

But in Backward Chaining, we work backwards from a target, then we figure out what are the dependencies. Then, by using inference rules we move backward and determine the facts that satisfy the goal.

We can compile the differences in the following table:

Forward Chaining	**Backward Chaining**
Data Driven	Goal Driven
Complexity linear to Knowledge Base size	Complexity lesser than Knowledge base size
Analogous to Exploration in BFS	Analogous to Greedy DFS
Starts from Known facts	Starts From goal

Since the only computation in Backward chaining happens for the facts necessary for the goal state, it is much more efficient than the forward chaining which computes the inferences for facts not related to the goal state too. Therefore, intuitively we can say that Backward chaining is more effective compared to the forward chaining algorithm.

Consider the following example

In the example shown below for the given propositional logic:

P => Q

L ∩ M => P

A ∩ B => L

A ∩ P => L

B ∩ L => M

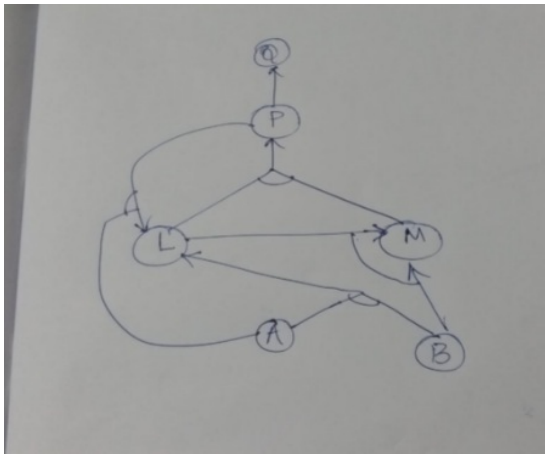
A

B

Forward chaining proves in 5 steps

Backward chaining takes only 4 steps.

So, the difference between the performance is expected to be larger depending on the complexity of the problem. **Hence, backward chaining is more effective compared to the forward chaining algorithm.**


Q
P
L
M
A
B