Unit 2

PAGE NO.
DATE :

Problem Solving.

- The Agents in Artificial Intelligence perfoem some kind of search algorithm to achieve thies tasks.
- The agents implements search algorithm in the background.

*Search Problem:-

A search problem in AI is a framework used to solve problems by finding a sequence of steps or actions.

Components of search problem:

1. State space:-
All the possible places or situations the agent could be in.
For example, if you are toying to solve a maze, the state space includes every possible position in the maze.

2. start state:-
This is where the agent begins. For instance in the maze, the start state is where you first enter the maze.

3. Goal state:-
The goal state is the destination or the answer to the problem.
For the maze, it's the exit.

4. Solution (or Plan):-
A solution is a set of steps or actions the agent takes to move from the start state to the goal state.

PAGE NO.
DATE :

- For example,
in the maze, this might be seeies of "move up", "move right" etc. that leads to the exit.

* How Agent finds the Plan:-
The agent uses a search algorithm to figure out the best path or steps to take.

- The agent explores possible options.
(like paths in the maze)

- It evaluate each option to see if it leads to the goal.
- Once it finds the goal, it stops & outputs the steps it took to get there.

PAGE NO.
DATE :

1) Uninformed Search strategies.

- It is also called as blind search.
- These strategies have no additional information about states.
- These strategies generate successors and distinguish a goal state from a non-goal state.
- These strategies do not use any domain (specific knowledge other than the problem definition itself (e.g. initial state, actions & goal state)
- They work by systematically exploring all possible states, often in a predefined manner, until a solution is found or all possibilities ~~are~~ are exhausted

e.g.

1) Breadth First Search
2) Depth First search.
3) Uniform cost search.
4) Depth Limited search.
5) Iterative Deepening Depth First Search (IDDFS)
6) Bidirectional search.

Breadth First Search:-

- BFS explores all nodes at the present depth level before moving on to the nodes at next depth level.
- The BFS is an instance of the general graph search algorithm.
- In this search the shallowest unexpanded node is choosen for expansion.
- This is achieved very simply by using a FIFO queue for the frontier.
- Frontier is a set of nodes that have been reached but whose children have not yet examined.
- The first node added to the frontier is the first one to be explored.

How BFS works?

1. Initialization:
The BFS algorithm starts by placing the initial node (starting point) in the frontier.

2. Exploration:
The algorithm then pops the first node from the frontier (the shallowest node) to explore it's neighbours.

3. Expansion:
Each unvisited neighbor of the current node is added to the frontier, awaiting future exploration.

4. Termination:
The process continues until the goal state is found or the frontier becomes empty.

Pseudo code for BFS:-

function Breadth First Search (Problem) returns a solution or failure

node ← a node with STATE = problem.Initial-state, path-cost=0
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
frontier ← a FIFO queue with node as the only element.
explored ← an empty set.

loop do
if EMPTY?(frontier) then return failure.
node ← pop(frontier) // choose the shallowest node in frontier.
add node.STATE to explored.
for each action in problem.ACTIONS(node.STATE)
do child ← child.NODE(problem, node, action)

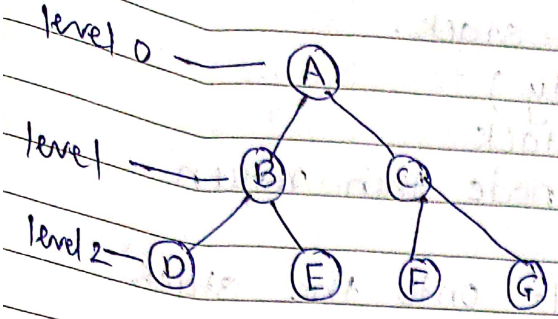
if child.STATE is not in explored or frontier then
if problem.GOAL-TEST(child.STATE) then return
SOLUTION(child)
frontier ← INSERT(child, frontier)

PAGE NO.
DATE :

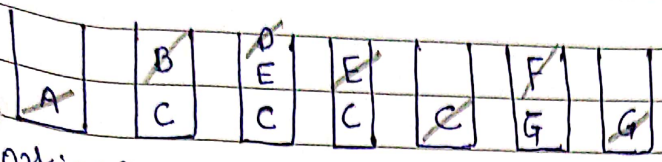
DFS :- (Depth First Search)

→ It is uninformed search.

→ It operates in LIFO manner.

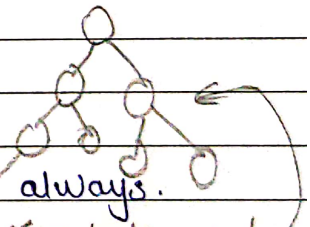


A B D E C F G



*Working :-

- Start with node A.
- Put into stack.
- Remove from stack and add children's of A.
- You can add childrens of A into the stack in any order.
- Now remove topmost element i.e. B from the stack and add it's children into the stack by keeping C as it is.
- Iterate above steps till your stack becomes empty by traversing all nodes.



- The DFS works vertically.
- It doesn't guarantee solution always.

If there is "branch (infinite) then it will never reach to goal.

∞ (infinity)

* Time complexity :-

$O(V+E)$

V - no. of nodes (vertices)

E - no. of edges.

Time complexity in terms of AI :-

- $O(b^d)$

b → branch factor. d → depth

branch factor means for every node how many childrens are there. (In above example for every node there are two childrens & depth is 2).

PAGE NO.

DATE :

Algorithm:-

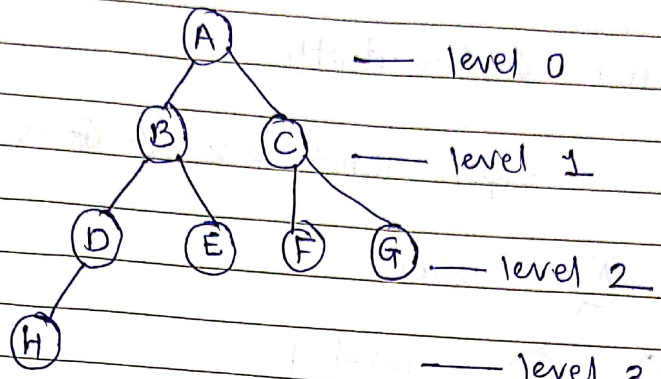
1. Push the root node on a stack.
2. While (stack is not empty)
   a) pop node from the stack.
   b) If node is a goal node then return Success.
3. Push all children of node onto the stack.
4. return failure.

PAGE NO.
DATE :

DLS:- (Depth Limited Search)

- It is uninformed search.
- To solve the problem of DFS with infinite branch of nodes. We can use : DLS.
- In DLS, we set a depth limit.

consider following example.



- Suppose goal state is G. here the depth limit is set to 2.
- Then DLS will not go to the level 3 to search for goal state.
- In above example, the goal state is available at level 2 & depth limit is also set to 2.

- The DLS teeminates if,
  i) The solution is found.
  ii) If there is no solution in depth limit

Advantages:-

i) Solve problem of infinite deep path with no solution of DFS.
ii) Memory efficient

Disadvantages:-

It can be incomplete if solution is below depth limit.

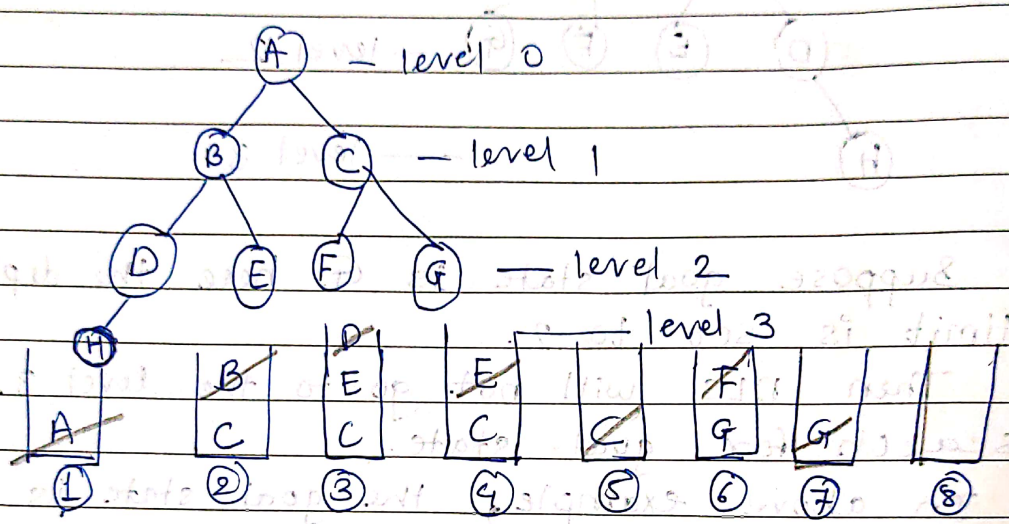
ii) Not necessary it will always give best solution.

Time complexity:-

- $O(b^d)$

b - branch factor & d → depth.

Working:- Depth limit = 2. & G is your goal state.



At step 3. When we process D. We need to add it's children to stack but depth limit is set to 2. So we can't add H to the stack.

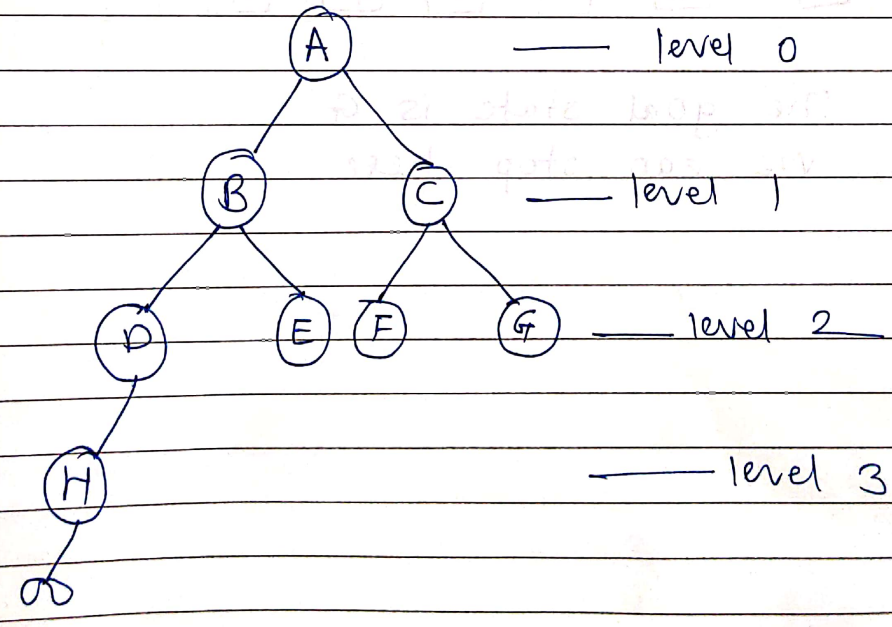
PAGE NO.
DATE :

* Iteeative Deepening Depth First Search:-

- It is uninformed search.
- This algorithm starts with depth limit of zero, if goal not found, it increments this limit iteeatively.
- Each iteeation peefoems a DFS search upto the current depth limit.

Algorithm:-

1. Start at the root node.
2. Peefoem DFS with Depth limit (L).
3. Increment depth afte each iteeation, incement depth limit by 1.
4. Continue this process until the goal node is found or the seeech space is exhausted.



Advantages:
It inheeits advantages of DFS & DLS of fast seeching & less memory.

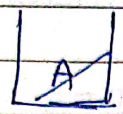
Disadvantages:
Keeps repeating process of previous phase.

DATE :

Working :-

Initially depth limit is zero. & goal state is G.

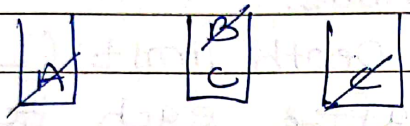
L = 0



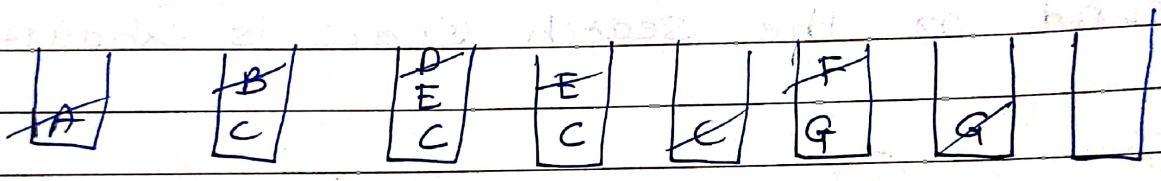
- add check if we got the solution. if yes then stop ⊗ if no, the increase the limit by 1.

~~L = 0~~

L = 1



L = 2



The goal state is G.
So, we can stop here.