Bibek Kr. Kushwaha (18/10/JC/033)
MCA-IV,

Artificial Intelligence, Assignment -II

(1) Write the depth limited search algo & apply it to the following problem, where nodes 0 and 11 are initial and good nodes, respectively, mentioning the steps of your algo. used in finding the good.

Soln:→ The depth-limited search algo. is similar to depth-first search algo. with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the DFS. In this algo., the node at the depth limit will be treated as if it has no successor nodes further. Depth limited search can be terminated with this conditions of failure:-

i) standard failure value:- It indicates that the problem does not have any solution.

ii) cut-off failure value:- It defines no solution for the problem within a given depth limit.

Advantages:-

(i) Depth limited search has a disadvantage of incompleteness.

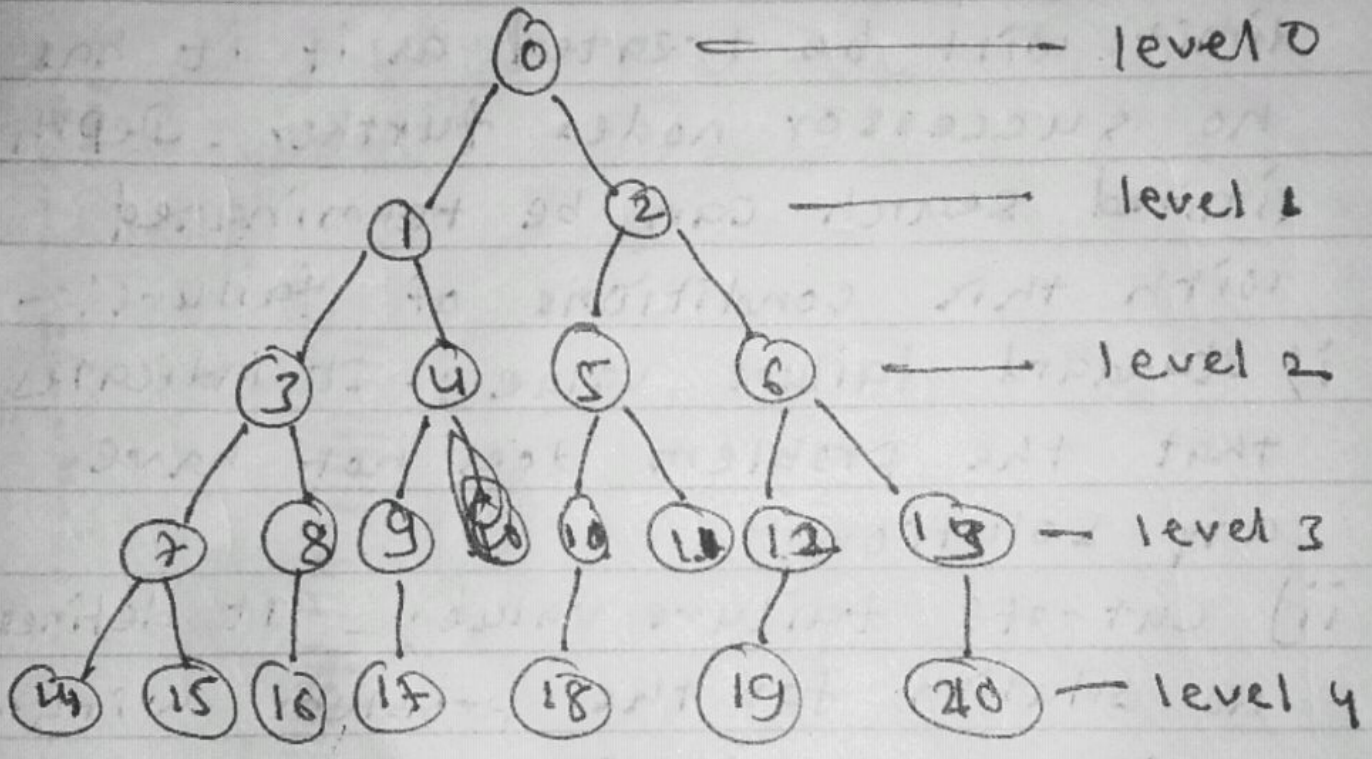
(ii) It may not be optimal if the problem has more than one solution.

Completeness :- DLS Algorithm is complete if the solution is above the depth-limit.

Time complexity :- The complexity of DLS algorithm is $O(b^l)$, where l is the number of levels.

Space complexity :- Space complexity of DLS algorithm is $O(b \times l)$.

Optimal :- Depth limited search algo. may be a special case of DFS but it is not optimal even if $l > d$.



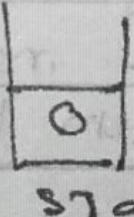
Given initial node = 0, good node = 11

let limit = 3.

step 1) Visit 0, PUSH 0

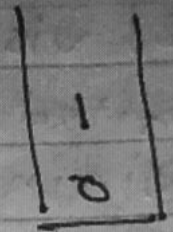
visited = {0}

Adjoining node = {1, 2}

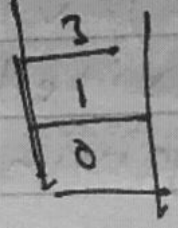


STACK

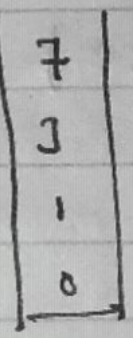
step 2) visit 1, PUSH 1
visited = {0, 1}
Adjoining nodes: {3, 4}



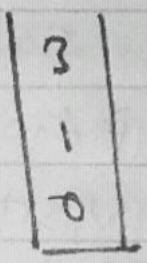
step 3) visit 3, PUSH 3
visited = {0, 1, 3}
Adjoining nodes = {7, 8}



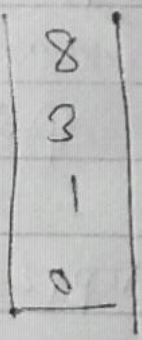
step 4) visit 7, PUSH 7
visited = {0, 1, 3, 7}
Adjoining nodes = { } [∵ limit = 3]



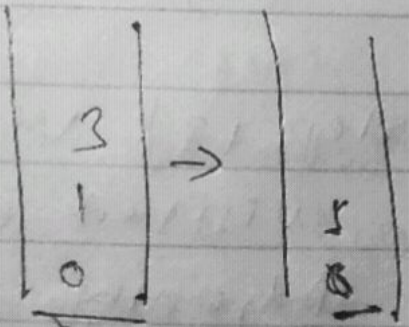
step 5) POP (7)
visit 3
Adjoining nodes = {7, 8}



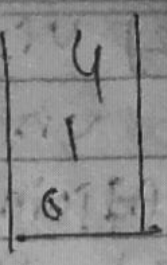
step 6) visit 8, PUSH 8
visited = {0, 1, 3, 7, 8}
Adjoining nodes = { } [∵ limit = 3]



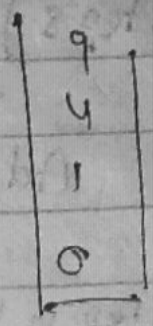
step 7) POP (8)
visit 3
POP (3)
visit 1
Adjoining nodes {3, 4}



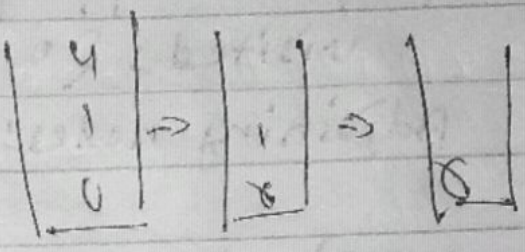
step-8) visit 4, PUSH 4
visited = {0, 1, 3, 7, 8, 4}
Adjoining nodes = {9}



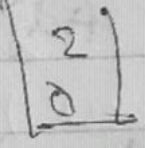
step-9) visit 9, PUSH 9
visited = {0, 1, 3, 7, 8, 4, 9}
Adjoining nodes = { } (∵ limit = 3)



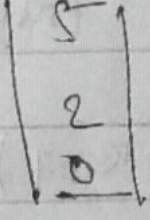
step 10) pop(9)
visit 4
pop(4)
visit(1)
pop(1) visit 0
Adjoining nodes: {1, 2}



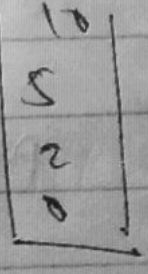
step 11) visit 2, PUSH 2
visited = {0, 1, 3, 7, 8, 4, 9, 2}
Adjoining nodes = {5, 6}



step 12) visit 5, PUSH 5
visited = {0, 1, 3, 7, 8, 4, 9, 2, 5}
Adjoining nodes = {10, 11}



step 13) visit 10, PUSH 10
visited = {0, 1, 3, 7, 8, 4, 9, 2, 5, 10}
Adjoining nodes = { }



step 14) pop(10)
visit 5
Adjoining nodes = {10, 11}

5
2
0

step 15) visit 11, push 11
visited = {0, 1, 3, 7, 8, 4, 9, 2, 5, 10, 11}
Goal search!

11
5
2
0

Nodes visited (path followed) to reach the goal =
{0, 1, 3, 7, 8, 4, 9, 2, 5, 10, 11}.

(8) write the depth first iterative deepening search algo. and apply it to the following problem, mentioning the steps followed in finding the goal.

Soln:- Depth first iterative deepening search algo:—

The iterative deepening depth first search algo. is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit

until a goal is found.
This algorithm performs depth first search upto a certain depth limit, then it keeps increasing the depth limit after each iteration ~~with~~ until the goal node is found.
This algo is a combination of BFS's fast search and DFS's memory efficiency

Advantages:-
It is a combination of the benifits of BFS (in terms of fast search) and DFS (in terms of memory ~~limit~~ efficiency).

Dis-advantages:-
The main drawback is that it repeats the work of the prev. phase.

Completeness:- The algo. is complete if the branching factor is finite.

Time complexity :- let b be the branching factor and d be the depth. Then the worst case time complexity is $O(b^d)$.

Space complexity :-

The space complexity is $O(bd)$.

Optimal :-

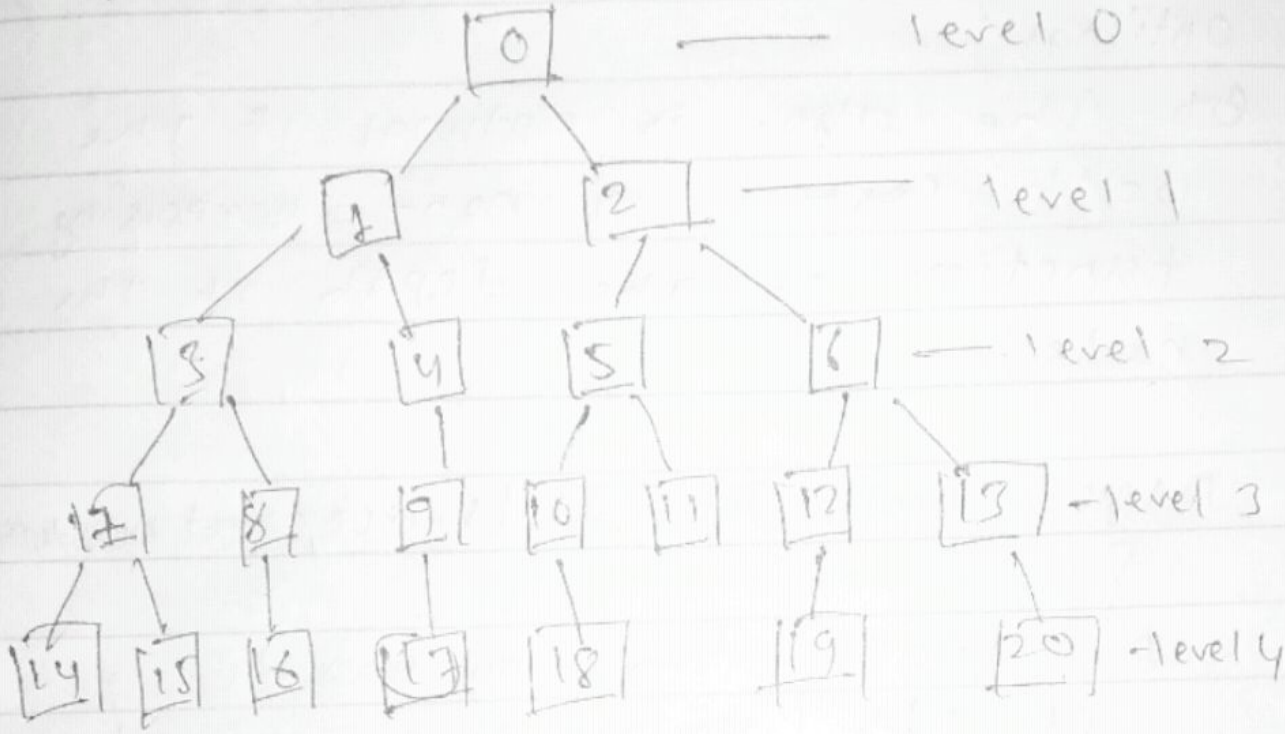
The algo. is optimal if the path cost is a non-decreasing function of the depth of the node.

```
Algo. bool DFIDS (source, target, max_depth)
  {
  for limit from 0 to max_depth
    if DLS (source, target, limit)==true
      return true;
  return false;
  }

bool DLS (source, target, limit)
    if (source == target)
        return true;
    if (limit <= 0)
        return false;
```

for each adjacent i of source
if dls(i, target, limit - 1)
return true;
return false;

Given tree;



Depth considered	Depth first iterative deepening search [Nodes]
0	0
1	0, 1, 2
2	0, 1, 3, 4, 2, 5, 6
3	0, 1, 3, 7, 8, 4, 9, 2, 5, 10, (11), 6, 12, 13

Hence, the goal node is found in the 4th iteration.

The algorithm returns true once the goal node is found.

There are no further levels to be searched once the good node is found.