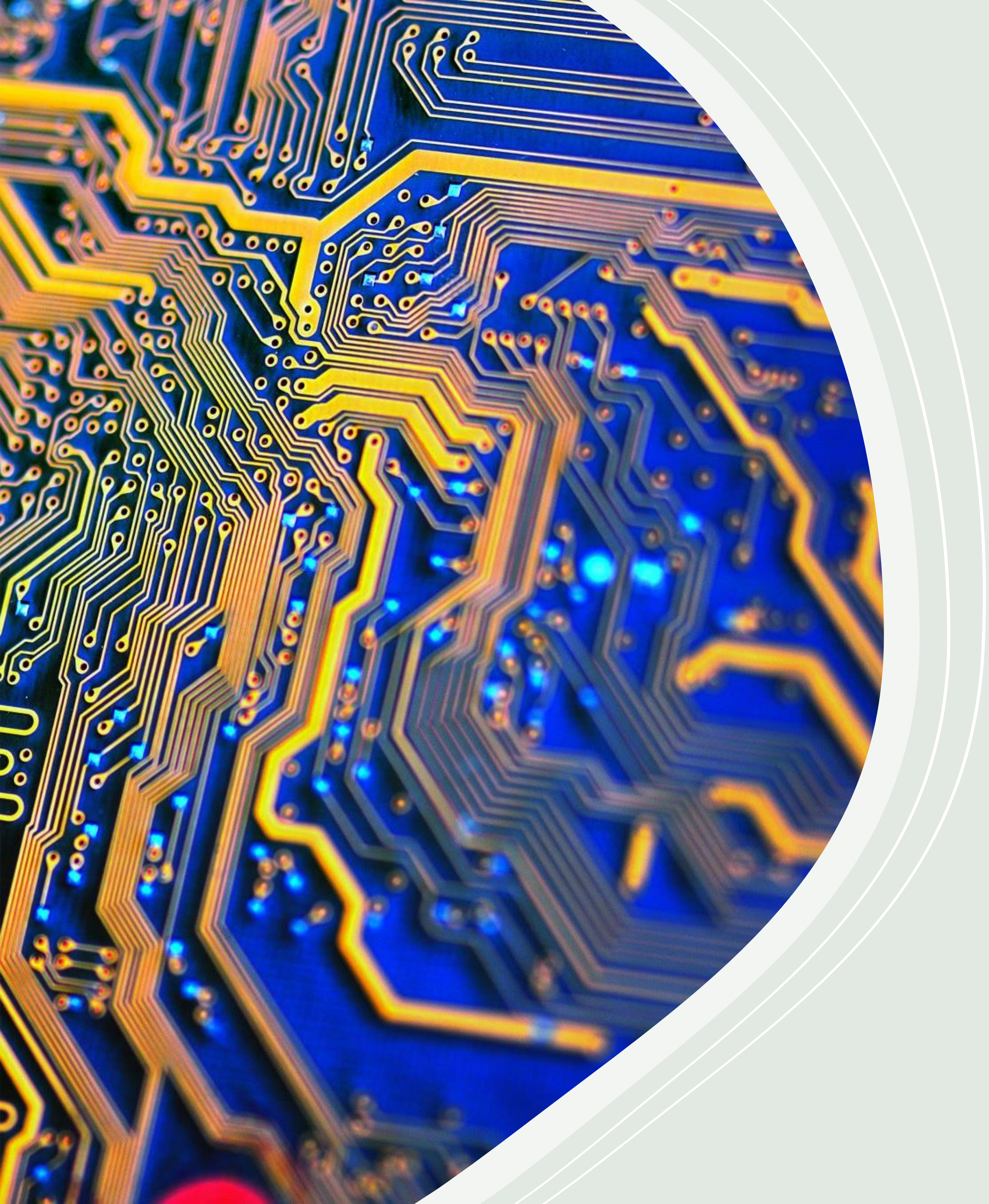


Module 1 Chapter 2

Artificial Intelligence

VBDS 1402

Uninformed Search (DLS, IDDFS, Bidirectional)

In this session you will learn:

Depth Limited Search

Iterative Deepening Depth First Search

Bidirectional search

Depth-first search (Revisit)

1. open = [A]; closed = []

2. open = [B,C,D]; closed = [A]

3. open = [E,F,C,D]; closed = [A,B]

4. open = [I,F,C,D]; closed = [A,B,E]

5. open = [F,C,D]; closed = [A,B,E,I]

6. open = [J,K,C,D]; closed = [A,B,E,I,F]

7. open = [K,C,D]; closed = [A,B,E,I,F,J]

8. open = [O,P,C,D] closed = [A,B,E,I,F,J,K]

9. open = [P,C,D] closed = [A,B,E,I,F,J,K,O]

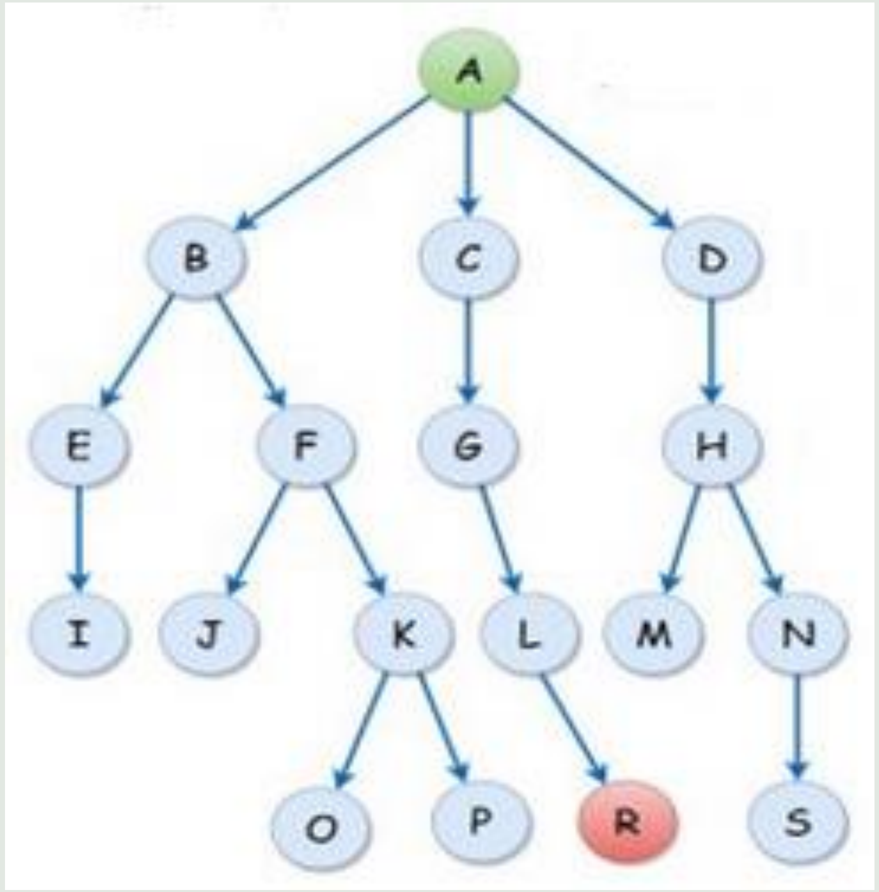
10 open = [C,D] closed = [A,B,E,I,F,J,K,O,P]

11 open = [G,D] closed = [A,B,E,I,F,J,K,O,P,C]

12 open = [L,D] closed = [A,B,E,I,F,J,K,O,P,C,G]

13 open = [R,D] closed = [A,B,E,I,F,J,K,O,P,C,G,L]

14 Next is R, Goal Node Reached



Depth Limited (Uninformed/Blind Search)

Depth-first search with a predetermined limit.

Solves the drawback of the infinite path in the Depth-first search.

Node at the depth limit will treat as it has no successor nodes further.

Depth-limited search can be terminated with two Conditions of failure:

Standard failure value: It indicates that problem does not have any solution.

Cut off failure value: It defines no solution for the problem within a given depth limit.

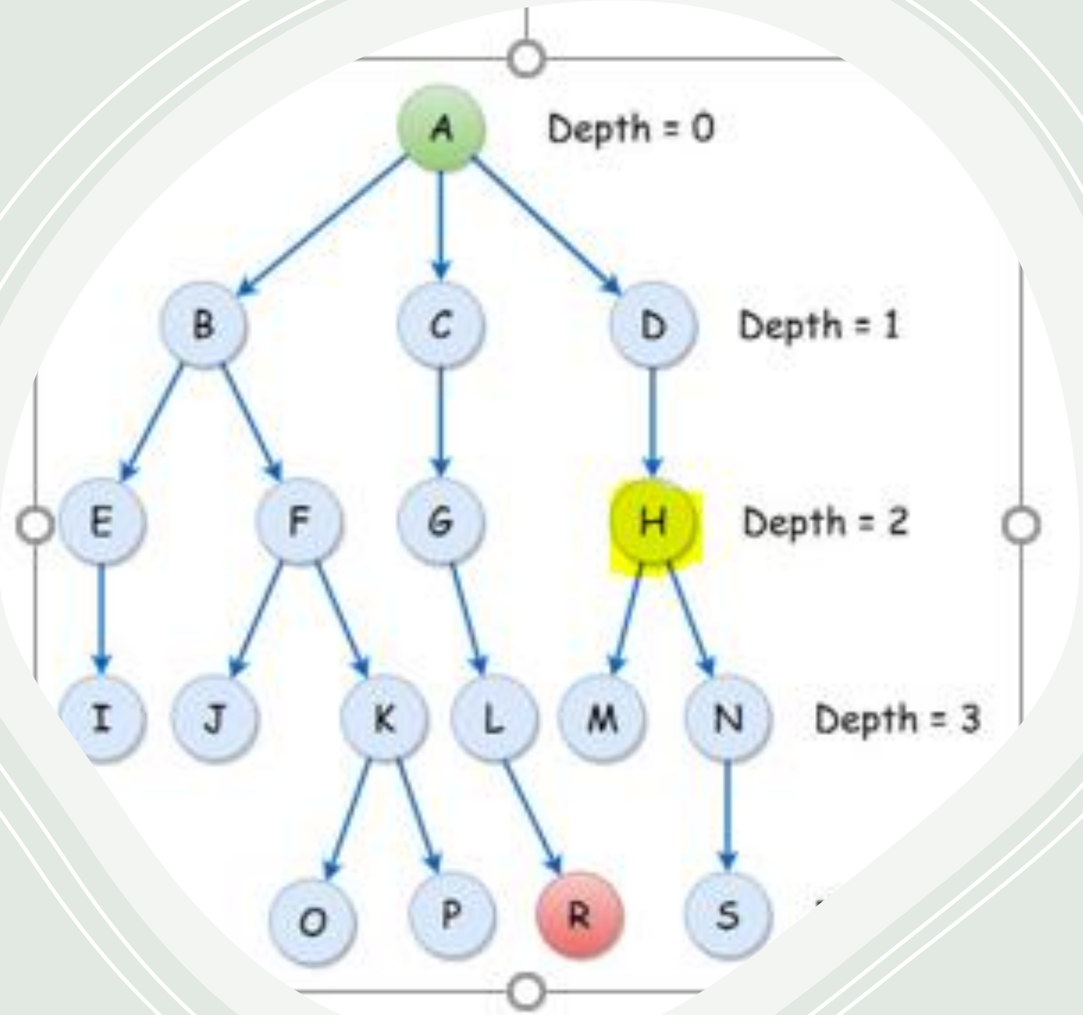
Example

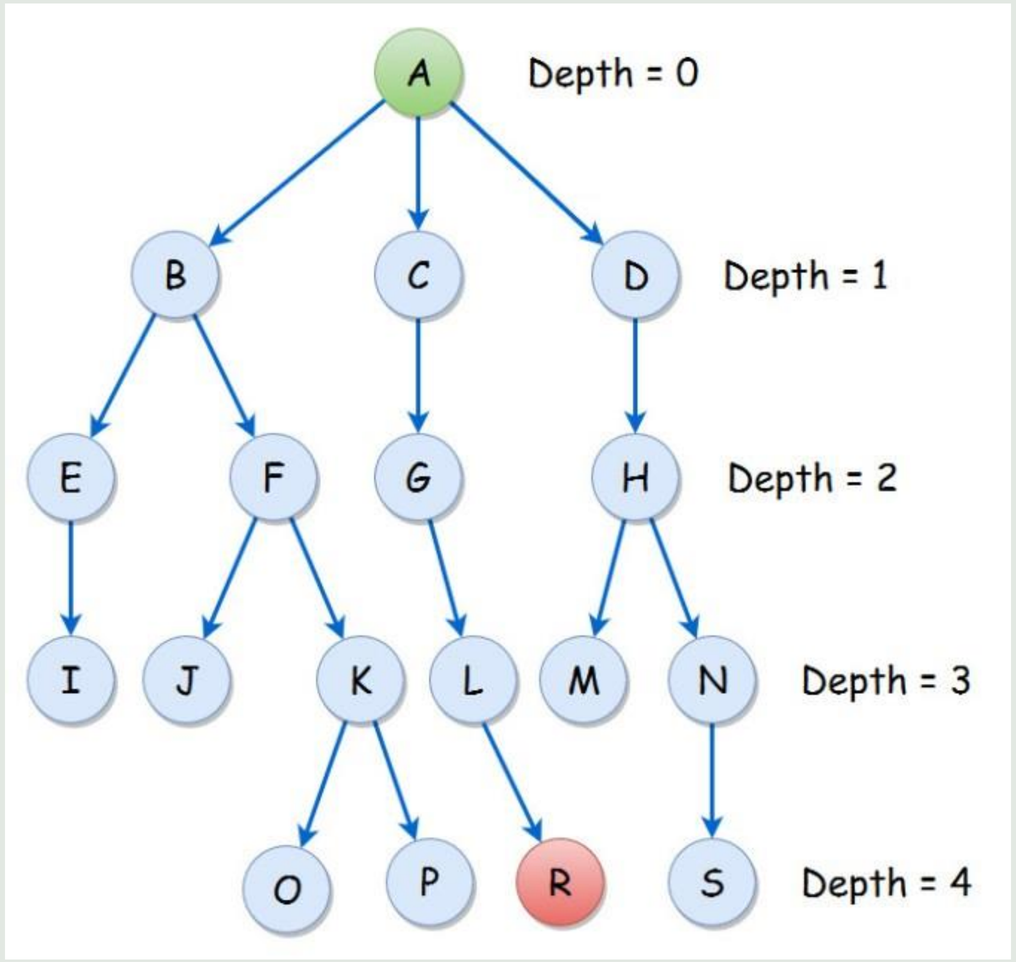
Limit = 2

1. open = [A]; closed = []
2. open = [B,C,D]; closed = [A]
3. open = [E,F,C,D]; closed = [A,B]
4. open = [F,C,D]; closed = [A,B,E]
5. open = [C,D]; closed = [A,B,E,F]
6. open = [G,D]; closed = [A,B,E,F,C]
7. open = [D]; closed = [A,B,E,F,C,G]
8. open = [H]; closed = [A,B,E,F,C.G,D]

Goal Node H Reached

Next is R, Goal Node not Reached





Example

If Depth =0
closed = [A]

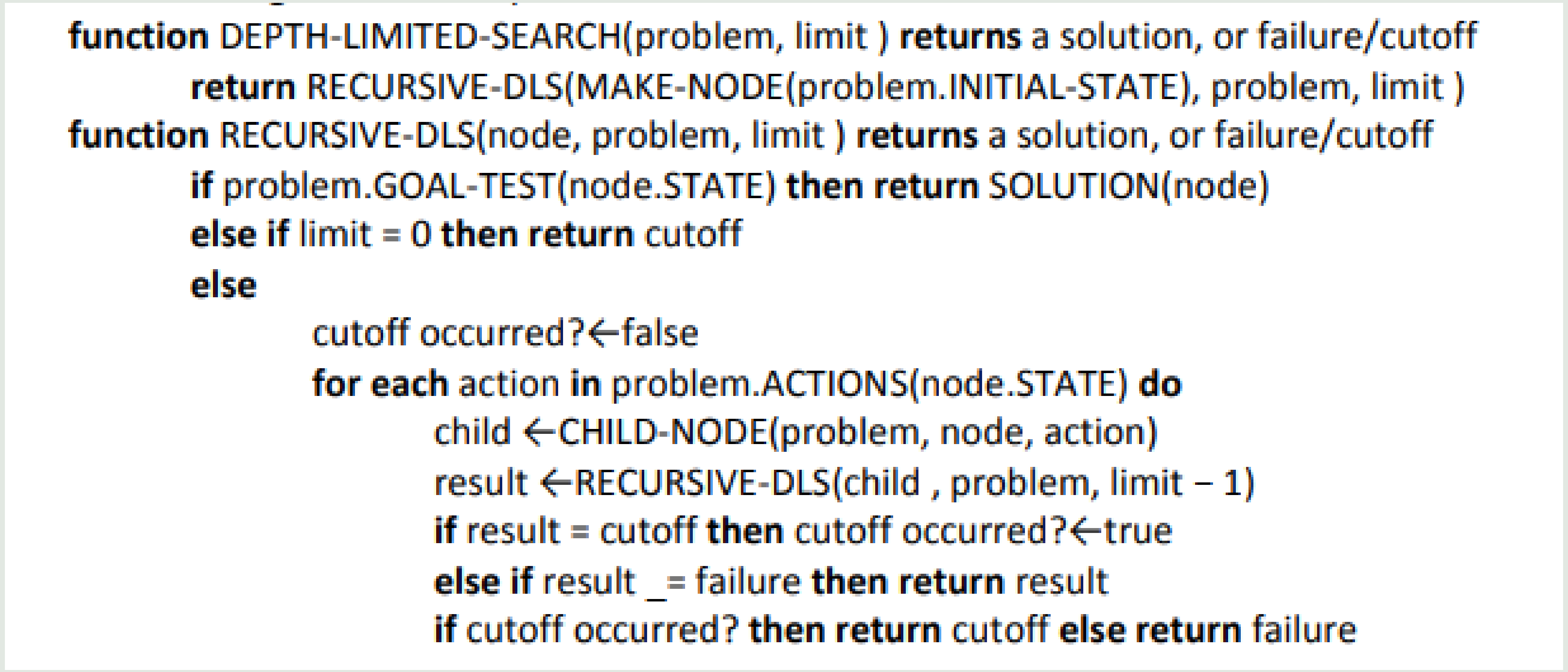
If Depth =1
closed = [A B,C,D]

If Depth =2
closed = [A B E F C G D H]

If Depth =3
closed = [A B E I F J K C G L D H M N]

If Depth =4
closed = [A B E I F J K O P C G L R]

Algorithm



```
function DEPTH-LIMITED-SEARCH(problem, limit ) returns a solution, or failure/cutoff
    return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit )
function RECURSIVE-DLS(node, problem, limit ) returns a solution, or failure/cutoff
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    else if limit = 0 then return cutoff
    else
        cutoff occurred?←false
        for each action in problem.ACTIONS(node.STATE) do
            child ←CHILD-NODE(problem, node, action)
            result ←RECURSIVE-DLS(child , problem, limit − 1)
            if result = cutoff then cutoff occurred?←true
            else if result _= failure then return result
            if cutoff occurred? then return cutoff else return failure
```

Advantages of Depth Limited Search

Better than DFS and requires less time and memory space.

DFS assures that the solution will be found if it exists infinite time.

There are applications of DLS in graph theory particularly similar to the DFS.

In order to combat the disadvantages of DFS, we add a limit to the depth and our search strategy performs recursively down the search tree.

The depth limit is compulsory for this algorithm to execute.

The goal node may not exist in the depth limit set earlier which will push the user to iterate further adding execution time.

The goal node will not be found if it does not exist in the desired limit.

Disadvantages of Depth Limited Search

Iterative deepening DFS (Uninformed/Blind Search)

IDFS = BFS + DFS

This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

Iterative deepening DFS (Uninformed/Blind Search)

It addresses problem of choosing depth bound

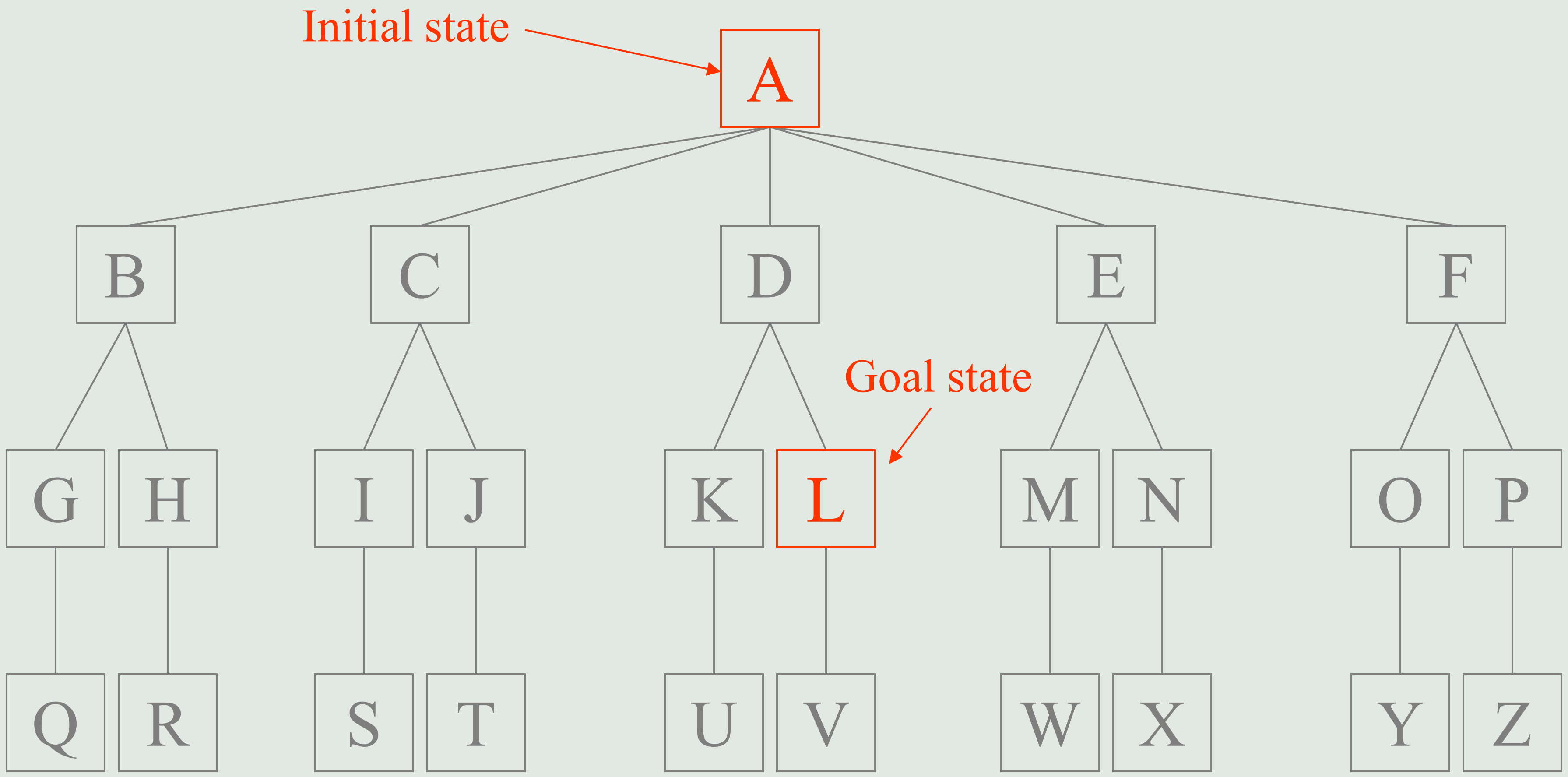
It is complete and finds best solution.

Basic idea is:

- Do DLS for depth n = 0; if solution found, return it; otherwise do DLS for depth n = n + 1; if solution found, return it, etc;
- So, we repeat DLS for all depths until solution found.

Useful if the search space is large and the maximum depth of the solution is not known.

The example node set



Press space to see a IDS of the example node set

A

We begin with the initial state: the node labeled A. This node is added to the queue. Press space to continue

Node A is then expanded and removed from the queue. Press space

As this is the 0th iteration of the search, we cannot search past any level greater than zero. This iteration now ends, and we begin the 1st iteration.

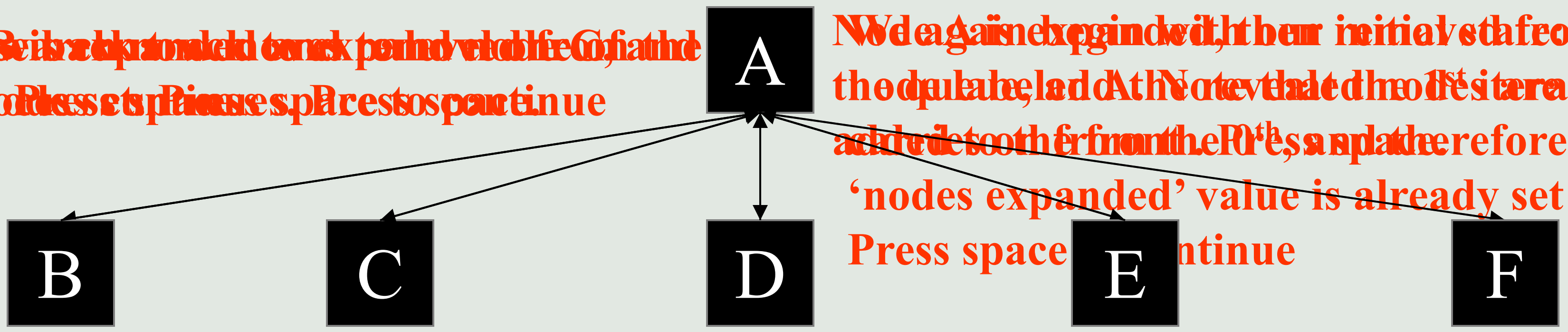
Press space to begin the search

Size of Queue: 0	**Queue: Empty**	
Nodes expanded: 1	**Current Action: Expanding**	**Current level: 0**

ITERATIVE DEEPENING SEARCH PATTERN (0th ITERATION)

[illegible]

We again begin with our initial state: the node labeled A. Note that the 1st iteration [illegible] from the 0th, and therefore the 'nodes expanded' value is already set to 1. Press space to continue



As this is the 1st iteration of the search, we cannot search past any level greater than level one. This iteration now ends, and we begin a 2nd iteration.

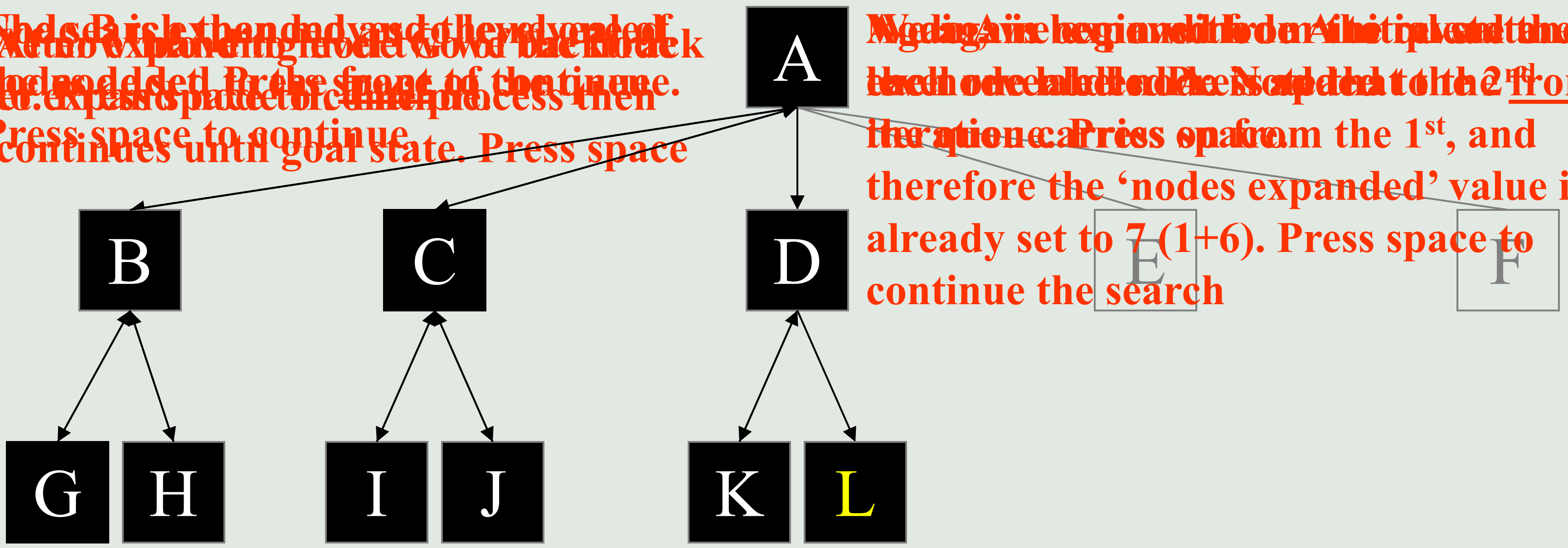
Press space to begin the search

Size of Queue: 0	Queue: Empty
Nodes expanded: 7	Current Action: Expanding
Current level: 1	

ITERATIVE DEEPENING SEARCH PATTERN (1st ITERATION) [illegible] Press space to continue

[illegible] continues until goal state. Press space

[illegible] front of [illegible] from the 1st, and therefore the 'nodes expanded' value is already set to 7 (1+6). Press space to continue the search



Node L is located on the second level and the search returns a solution on its second iteration. Press space to end.

Press space to continue the search

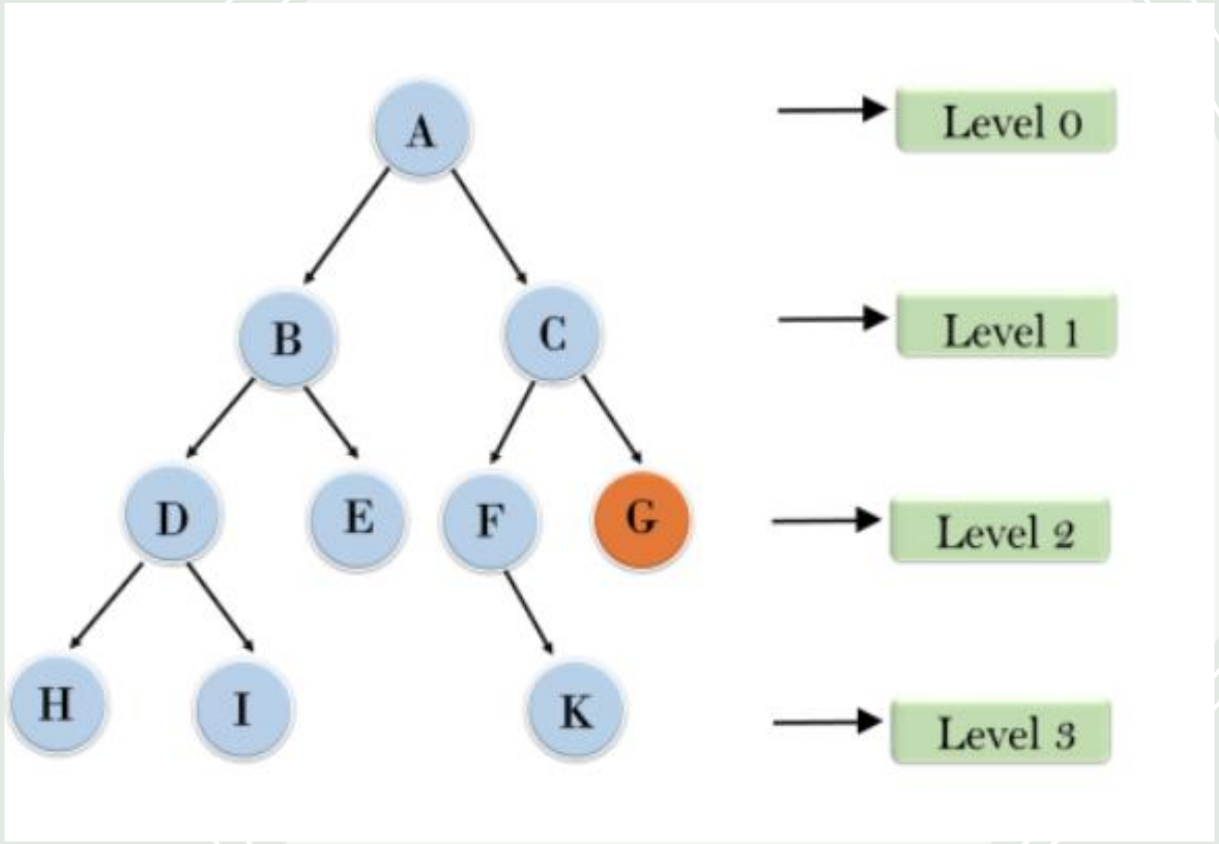
Size of Queue: 0	**Queue: Empty**	
Nodes expanded: 16	**SEARCH FINISHED**	**Current level: 2**

ITERATIVE DEEPENING SEARCH PATTERN (2nd ITERATION)

Example

- 1'st Iteration-----> A
- 2'nd Iteration----> A, B, C
- 3'rd Iteration------>A, B, D, E, C, F, G
- 4'th Iteration------>A, B, D, H, I, E, C, F, K, G

In the fourth iteration, the algorithm will find the goal node.



General Algorithm for Iterative Deepening

```
depth_limit = 0;
while(true) /* infinite loop */
{
result = depth_limited_search(max_depth = depth limit,agenda = initial node);
if result contains goal then return result;
depth limit = depth limit + 1;
}
```

Advantages

It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

IDDFS gives us the hope to find the solution if it exists in the tree.

When the solutions are found at the lower depths say n, then the algorithm proves to be efficient and in time.

The great advantage of IDDFS is found in-game tree searching where the IDDFS search operation tries to improve the depth definition, heuristics, and scores of searching nodes so as to enable efficiency in the search algorithm.

Another major advantage of the IDDFS algorithm is its quick responsiveness. The early results indications are a plus point in this algorithm.

Disadvantages

The main drawback of IDDFS is that it repeats all the work of the previous phase.

The main problem with IDDFS is the time and wasted calculations that take place at each depth.

The IDDFS might fail when the BFS fails. When we are to find multiple answers from the IDDFS, it gives back the success nodes and its path once even if it needs to be found again after multiple iterations. To stop the depth bound is not increased further.

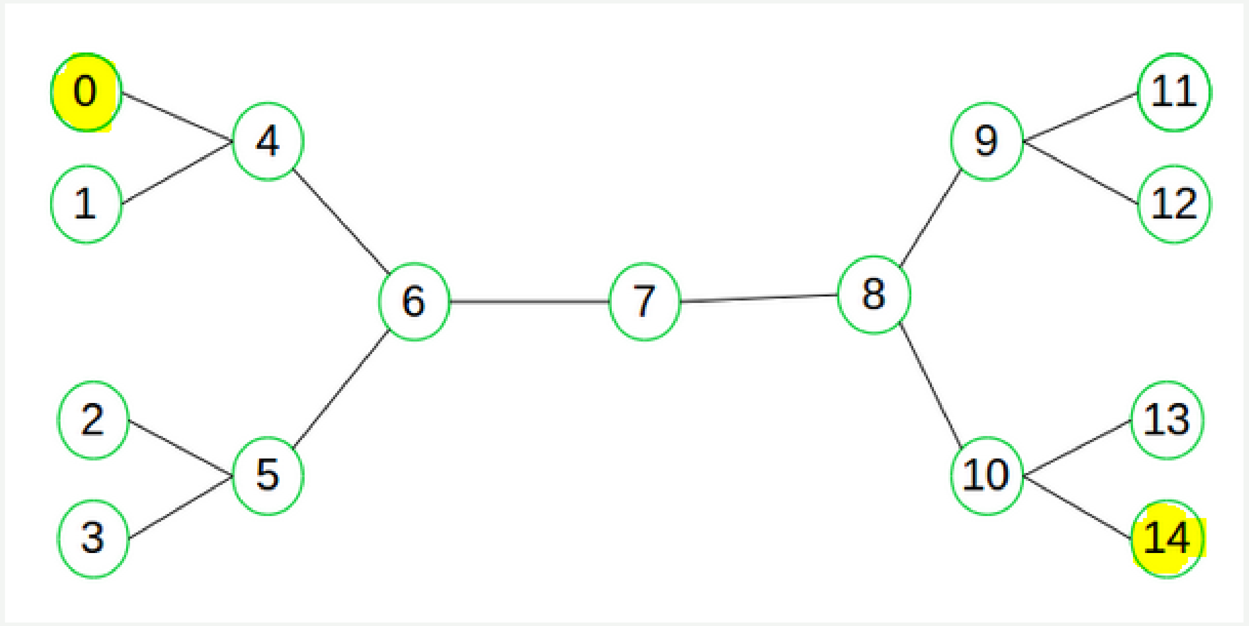
Bidirectional Search

Bidirectional search algorithm runs two simultaneous searches, one form initial state called as forward-search and other from goal node called as backward-search, to find the goal node.

Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex.

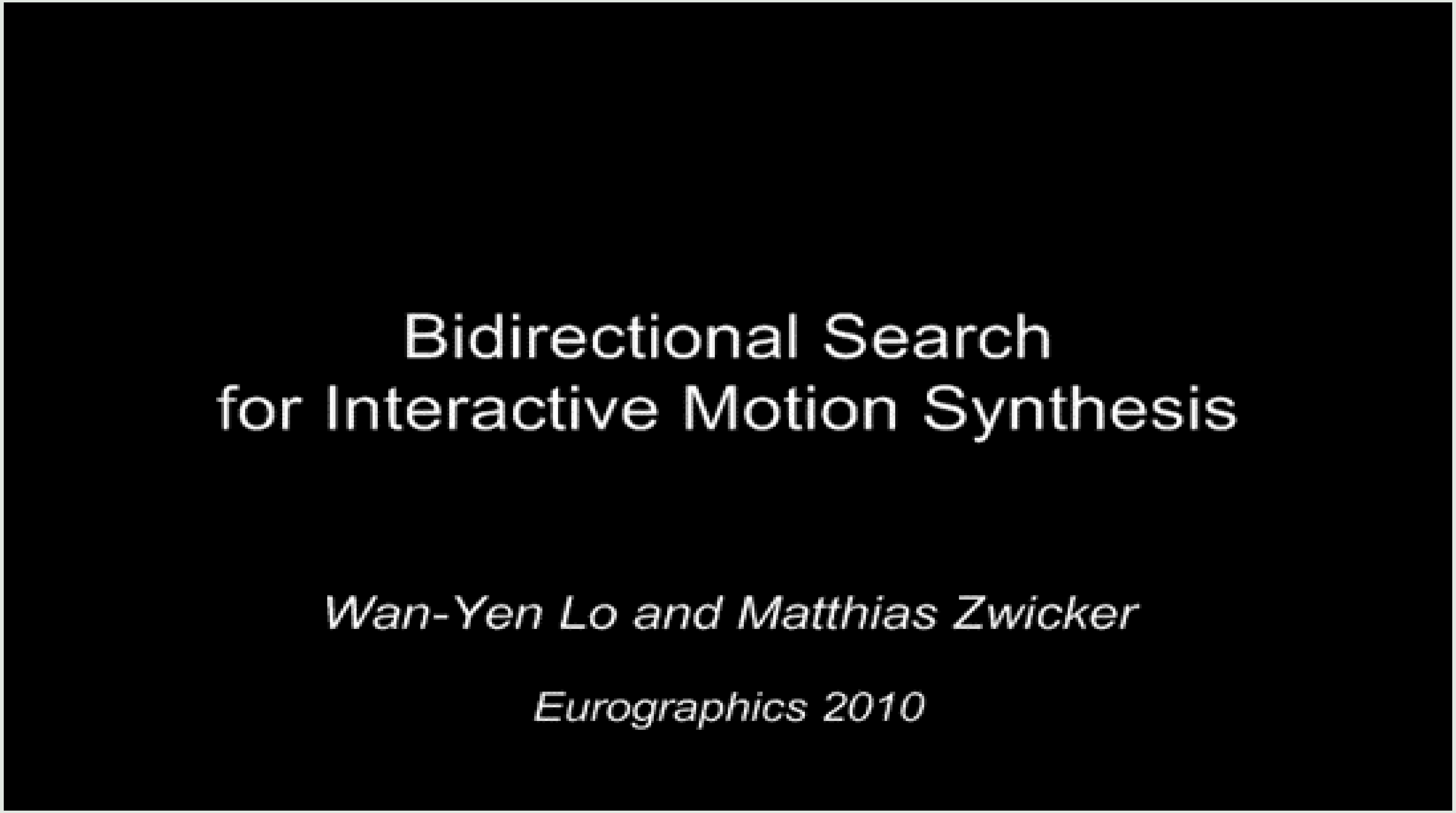
The search stops when these two graphs intersect each other.

Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.



EXAMPLE

Bi-Directional Search for Interactive Motion


Bidirectional Search
for Interactive Motion Synthesis

Wan-Yen Lo and Matthias Zwicker

Eurographics 2010

Bi - directional search: https://www.youtube.com/watch?v=NCkdU5Rub5U (Interactive Motion Synthesis)

Bidirectional Search Advantages & Disadvantages

Advantages:

Bidirectional search is fast.

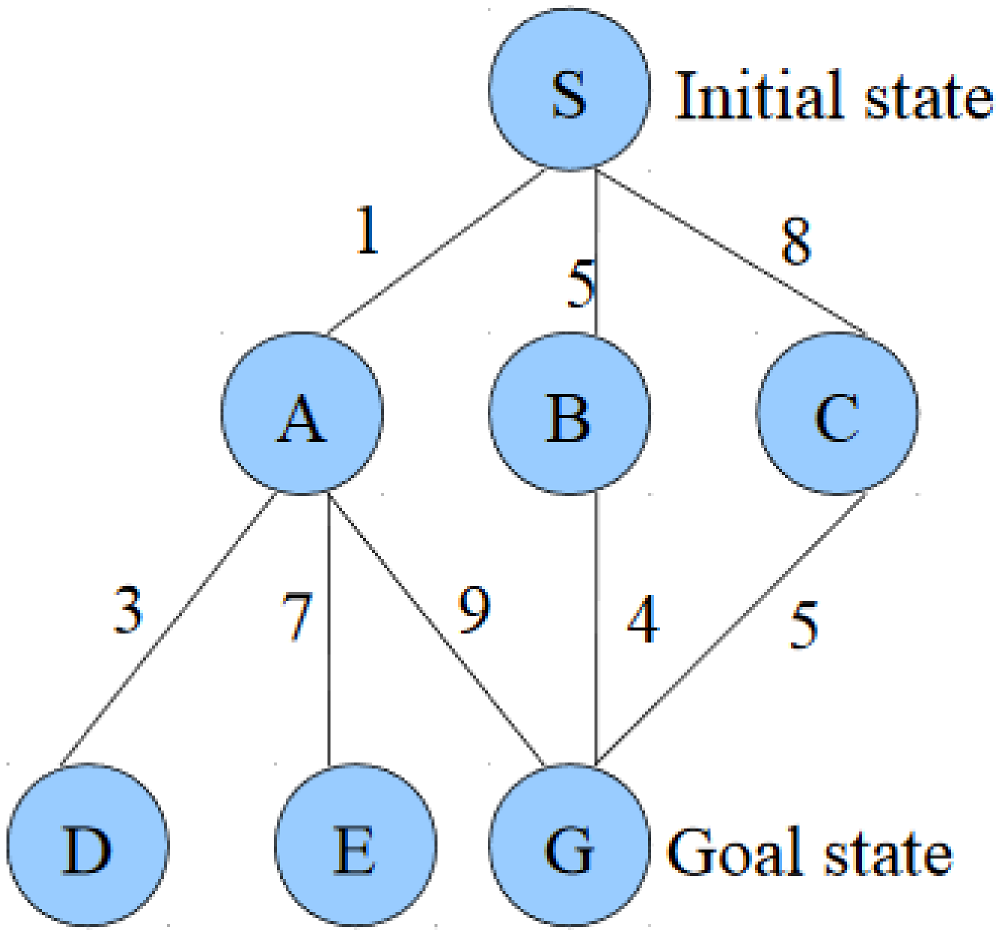
Bidirectional search requires less memory

Disadvantages:

Implementation of the bidirectional search tree is difficult.

In bidirectional search, one should know the goal state in advance**.**

Find Node expanded by BFS, DFS, UCS, IDDFS



(All edges are directed, pointing downwards)

THANK
YOU!