

Artificial Intelligence

3.4: Solving Problem by Searching



Today's Topic

- Uninformed Search Strategies
 - BFS, DFS, UCS
 - DPL, ID



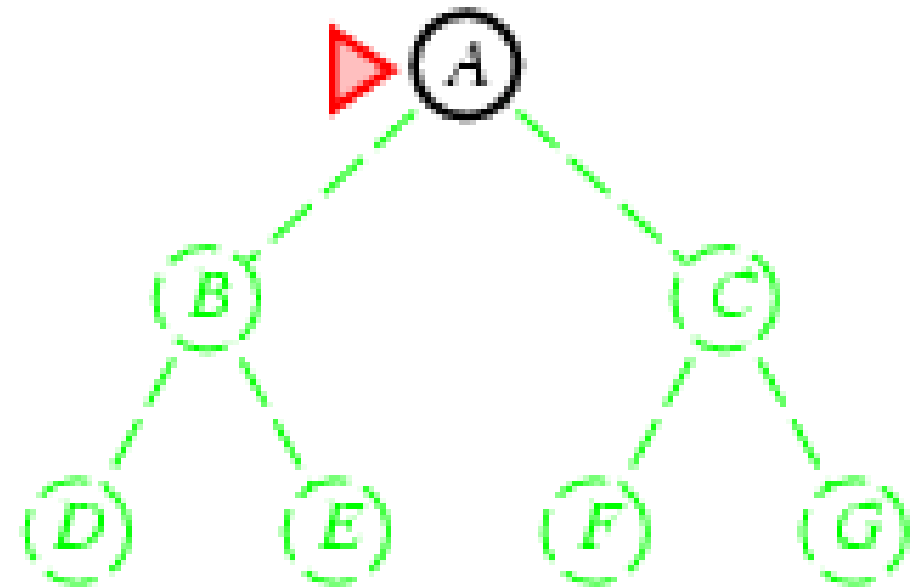
Uninformed search strategies

- **Uninformed** search strategies use only the information available in the problem definition
 - Breadth-first search
 - Uniform-cost search
 - Depth-first search
 - Depth-limited search
 - Iterative deepening search



Breadth-first search

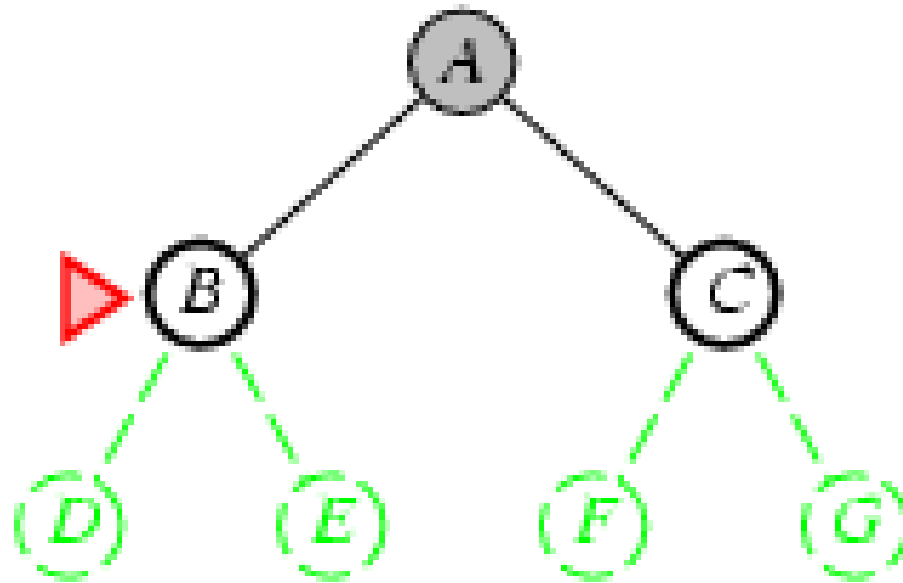
- Expand shallowest unexpanded node
- **Implementation:**
 - QUEUE : FIFO Implementation





Breadth-first search

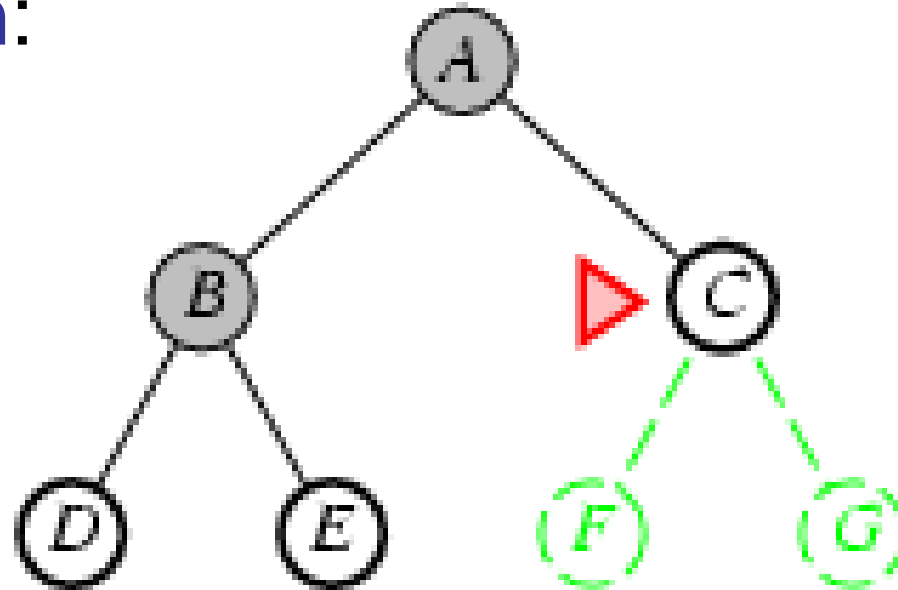
- Expand shallowest unexpanded node
- Implementation:





Breadth-first search

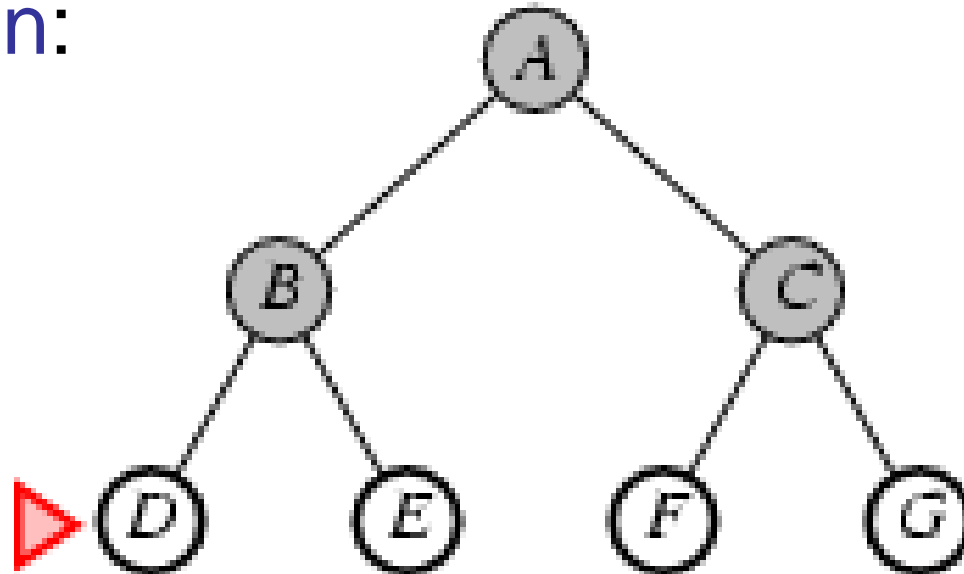
- Expand shallowest unexpanded node
- Implementation:





Breadth-first search

- Expand shallowest unexpanded node
- Implementation:



Breadth-first search

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure  
  node  $\leftarrow$  a node with STATE = problem.INITIAL-STATE, PATH-COST = 0  
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)  
  frontier  $\leftarrow$  a FIFO queue with node as the only element  
  explored  $\leftarrow$  an empty set  
  loop do  
    if EMPTY?(frontier) then return failure  
    node  $\leftarrow$  POP(frontier) /* chooses the shallowest node in frontier */  
    add node.STATE to explored  
    for each action in problem.ACTIONS(node.STATE) do  
      child  $\leftarrow$  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  $\leftarrow$  INSERT(child, frontier)
```

Figure 3.11 Breadth-first search on a graph.



Breadth-first Search: Analysis of BFS

- **Time complexity:** Assume a state space where every state has b successors
 - Assume solution is at depth d
 - **Worst case:** expand all but the last node at depth d
 - Total number of nodes generated:
 - $b + b^2 + b^3 + \dots + b^d = O(b^d)$
- **Space Complexity:** Every node generated must remain in memory so it will be same as time complexity



Properties of breadth-first search

- Complete? Yes (if b is finite)
- Time? $b + b^2 + b^3 + \dots + b^d = O(b^d)$
- Space? $O(b^d)$ (keeps every node in memory)
- Optimal? Yes (if cost = 1 per step)
- **Space** is the bigger problem (more than time)



Using Breadth-first Search

- When is BFS **appropriate**?
 - space is not a problem
 - it's necessary to find the solution with the fewest arcs
 - although all solutions may not be shallow, at least some are
- When is BFS **inappropriate**?
 - space is limited
 - all solutions tend to be located deep in the tree
 - the branching factor is very large



Exponential Growth

- **Exponential growth quickly makes complete state space searches unrealistic**
- **If the branch factor was 10, by level 5 we would need to search 100,000 nodes (i.e. 10^5)**

Exponential Growth

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

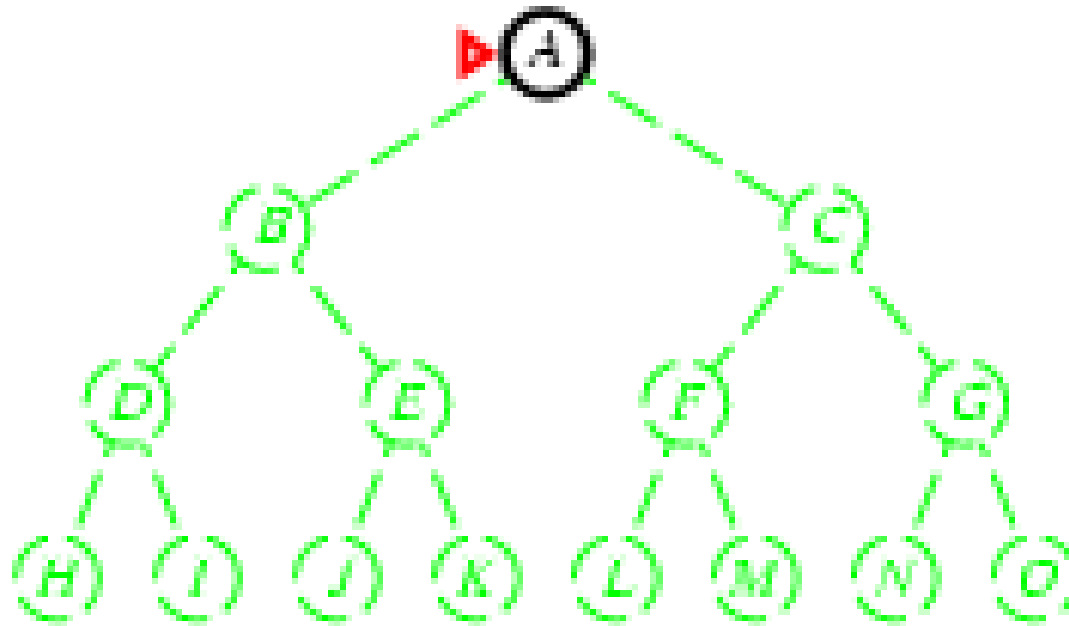
Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

Time and memory requirements for breadth-first search, assuming a branching factor of 10, 100 bytes per node and searching 1000 nodes/second



Depth-first search

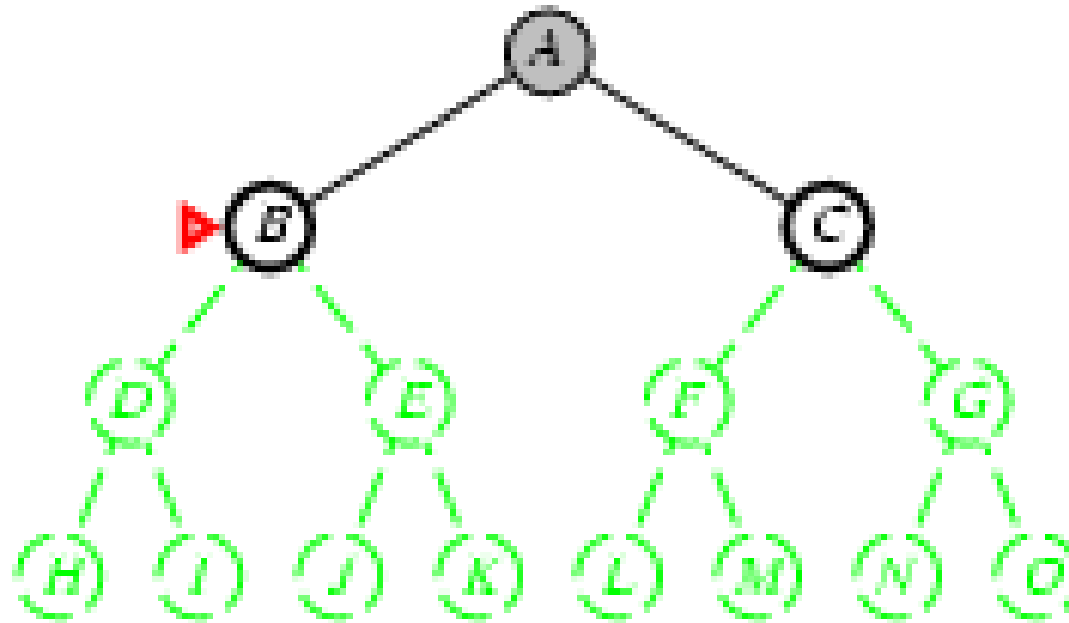
- Expand deepest unexpanded node in the current fringe
- LIFO-Stack
- Implementation:





Depth-first search

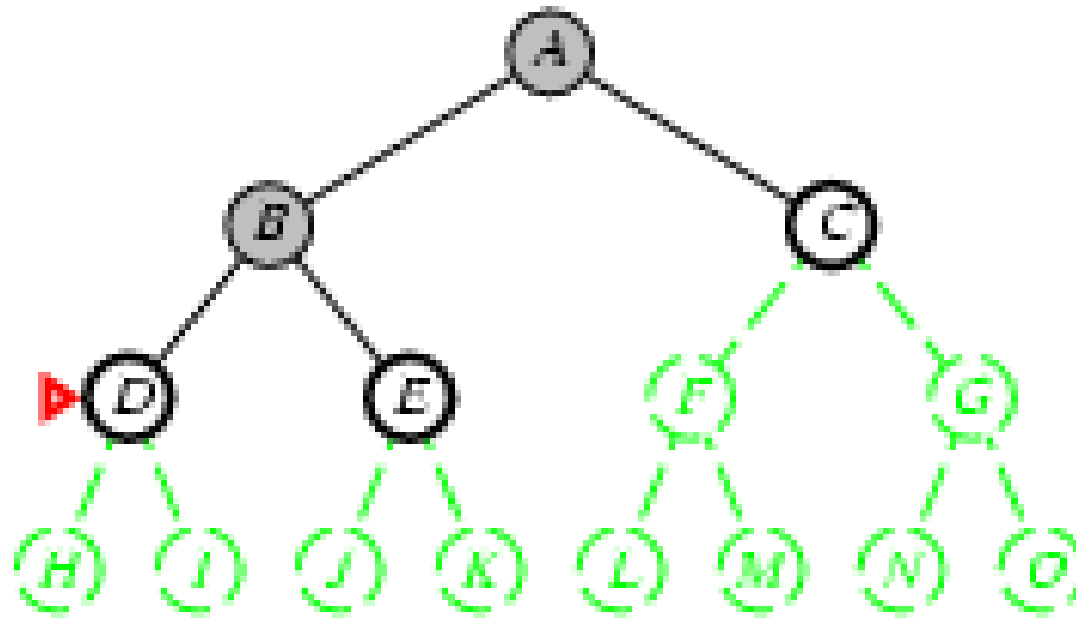
- Expand deepest unexpanded node
- Implementation:





Depth-first search

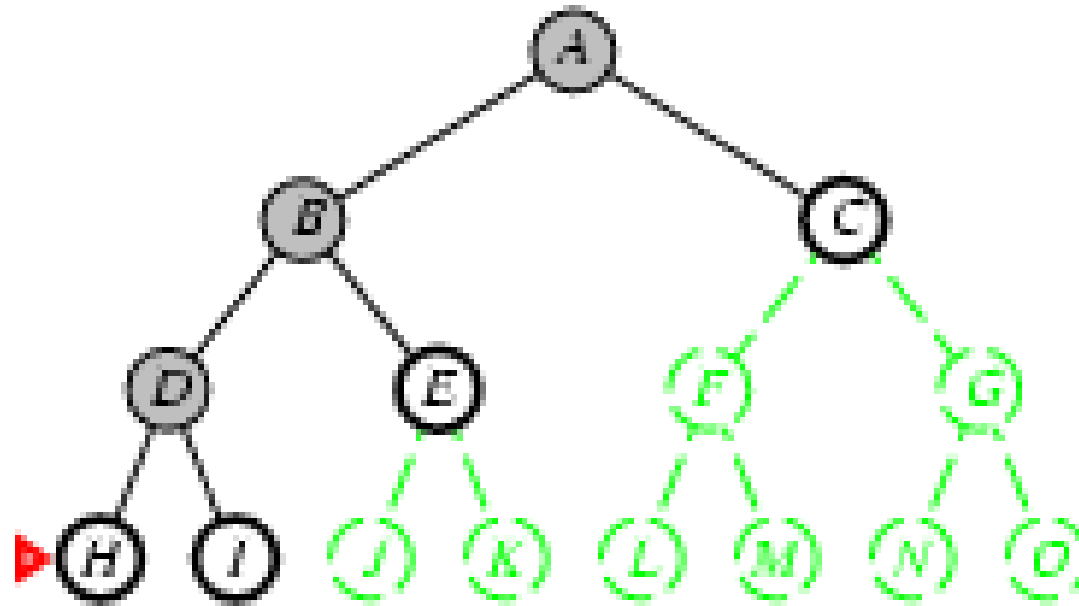
- Expand deepest unexpanded node
- Implementation:





Depth-first search

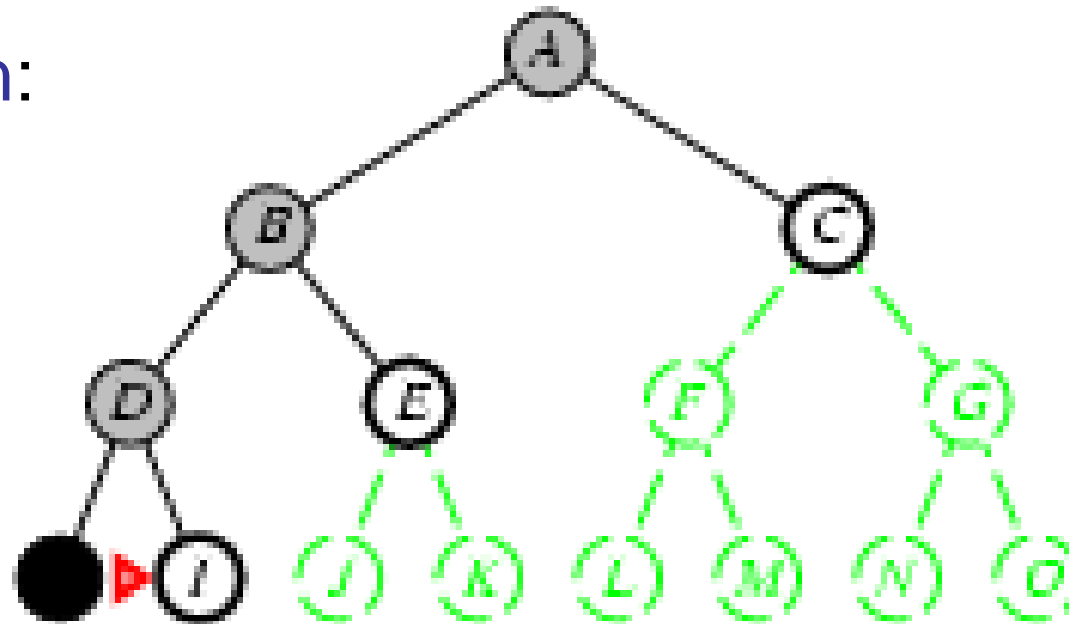
- Expand deepest unexpanded node
- Implementation:





Depth-first search

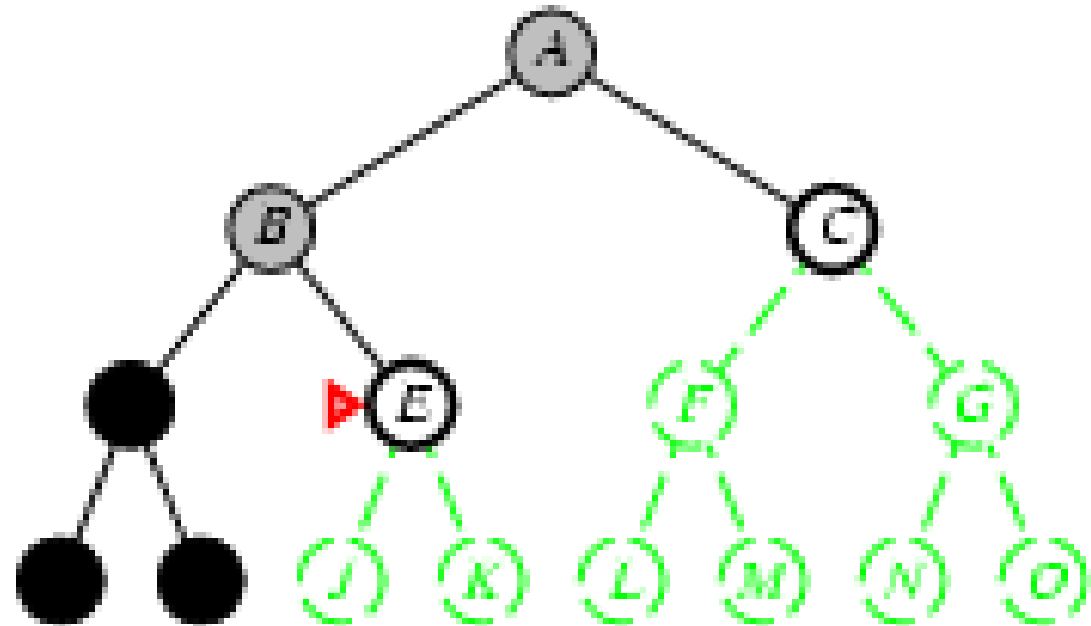
- Expand deepest unexpanded node
- Implementation:





Depth-first search

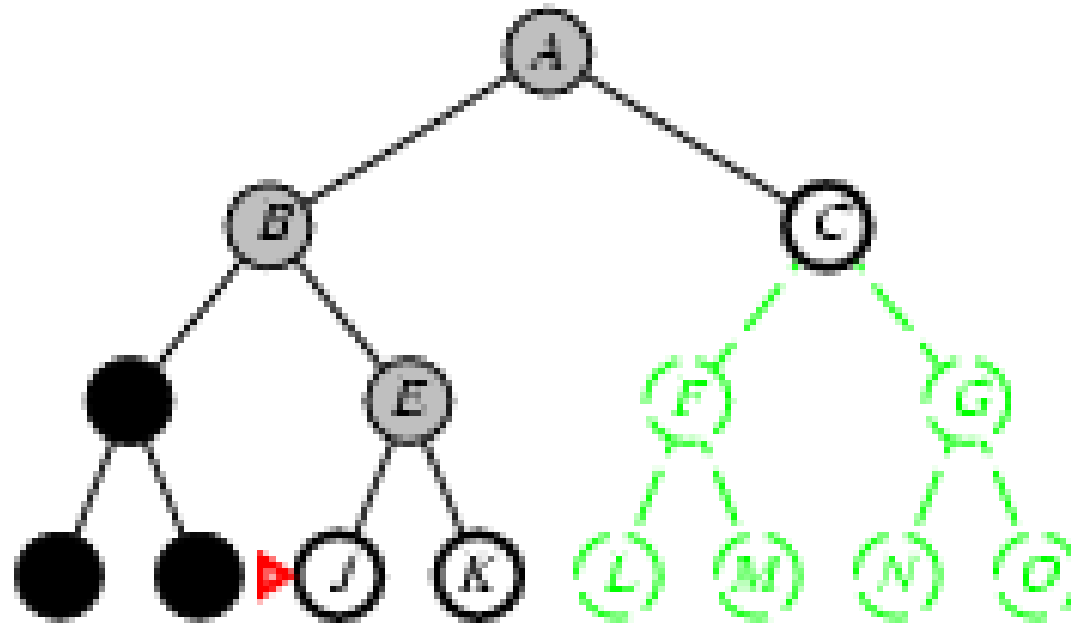
- Expand deepest unexpanded node
- **Implementation:**
 - *fringe* = LIFO queue, i.e., put successors at front





Depth-first search

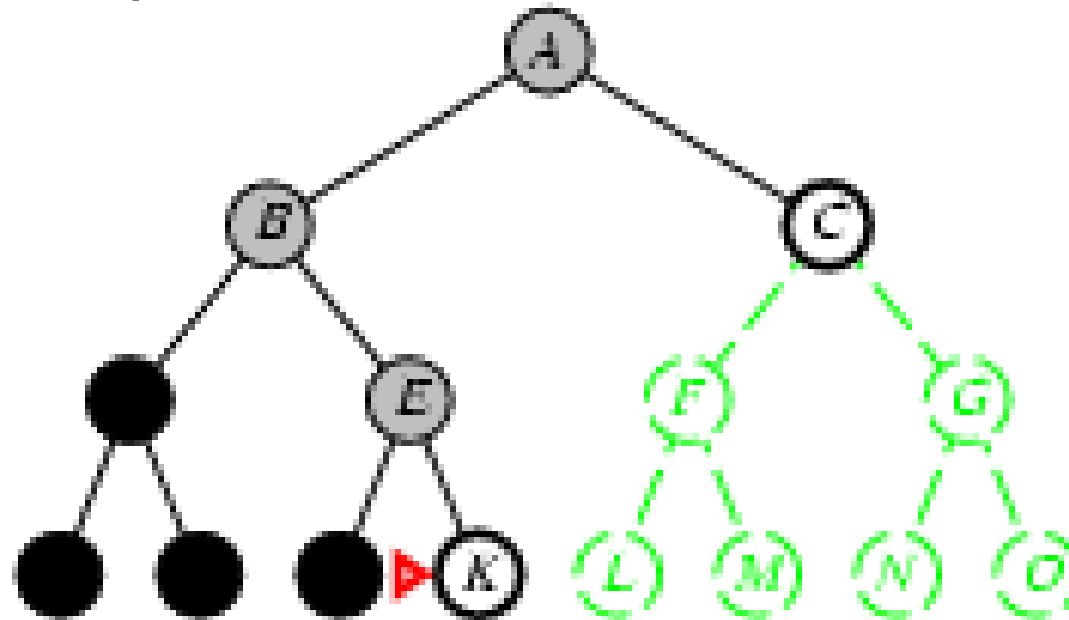
- Expand deepest unexpanded node
- Implementation:





Depth-first search

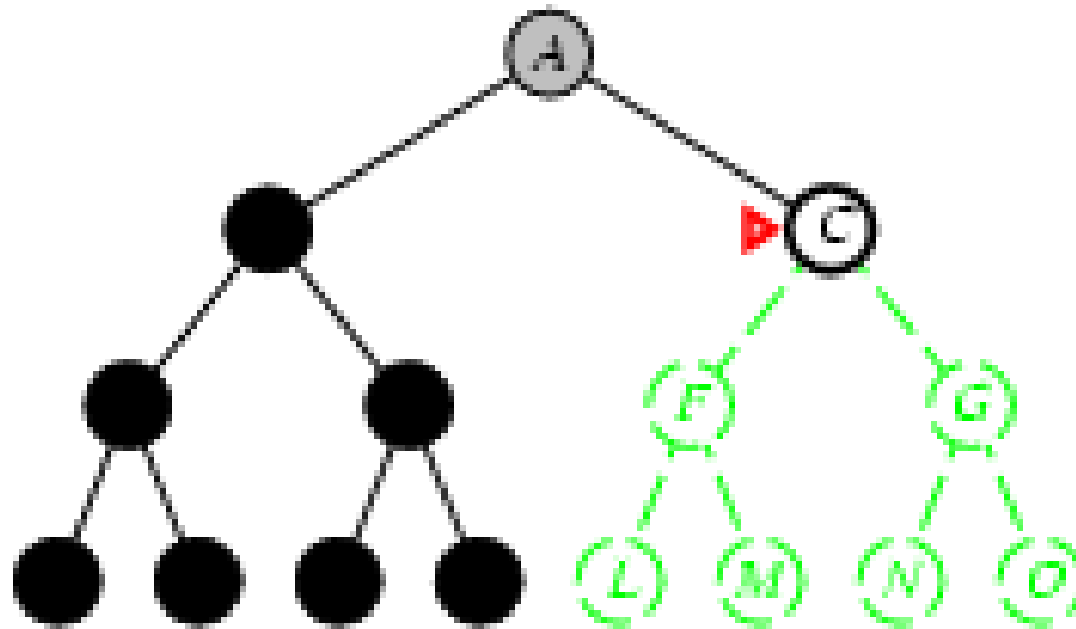
- Expand deepest unexpanded node
- Implementation:





Depth-first search

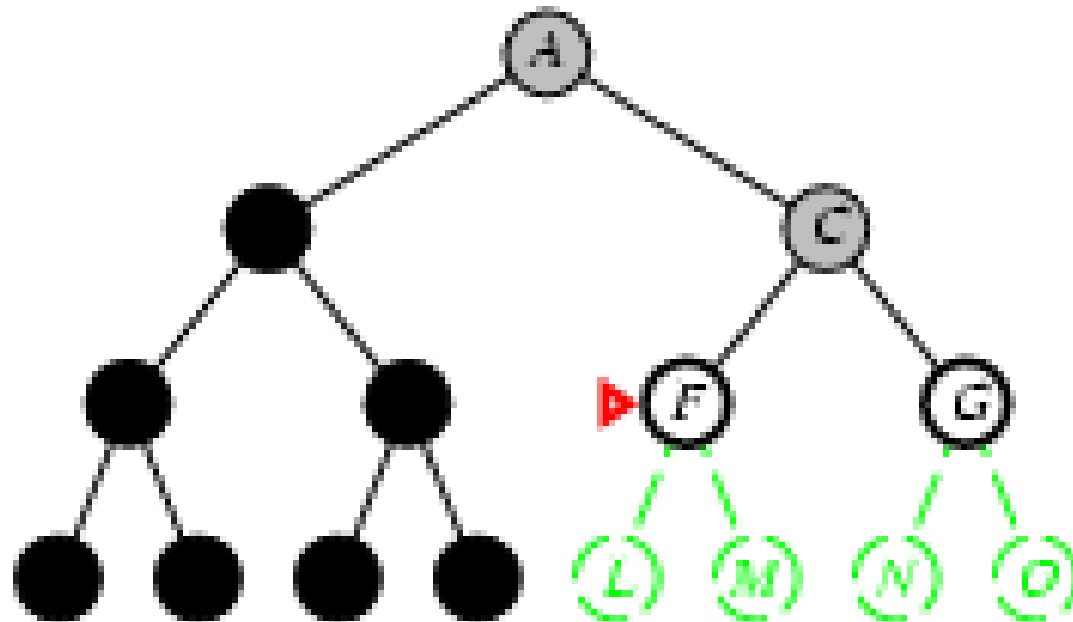
- Expand deepest unexpanded node
- **Implementation:**





Depth-first search

- Expand deepest unexpanded node
- Implementation





-
- ```

graph TD
 A((A)) --- B(())
 A --- C((C))
 B --- D(())
 B --- E(())
 C --- F((F))
 C --- G((G))
 F --- L((L))
 F --- M((M))
 G -.- N((N))
 G -.- O((O))
 style A fill:#ccc
 style B fill:#000
 style C fill:#ccc
 style D fill:#000
 style E fill:#000
 style F fill:#ccc
 style G fill:#fff
 style L fill:#fff
 style M fill:#fff
 style N stroke-dasharray: 5 5
 style O stroke-dasharray: 5 5
 linkStyle 10 stroke:red,stroke-width:2px

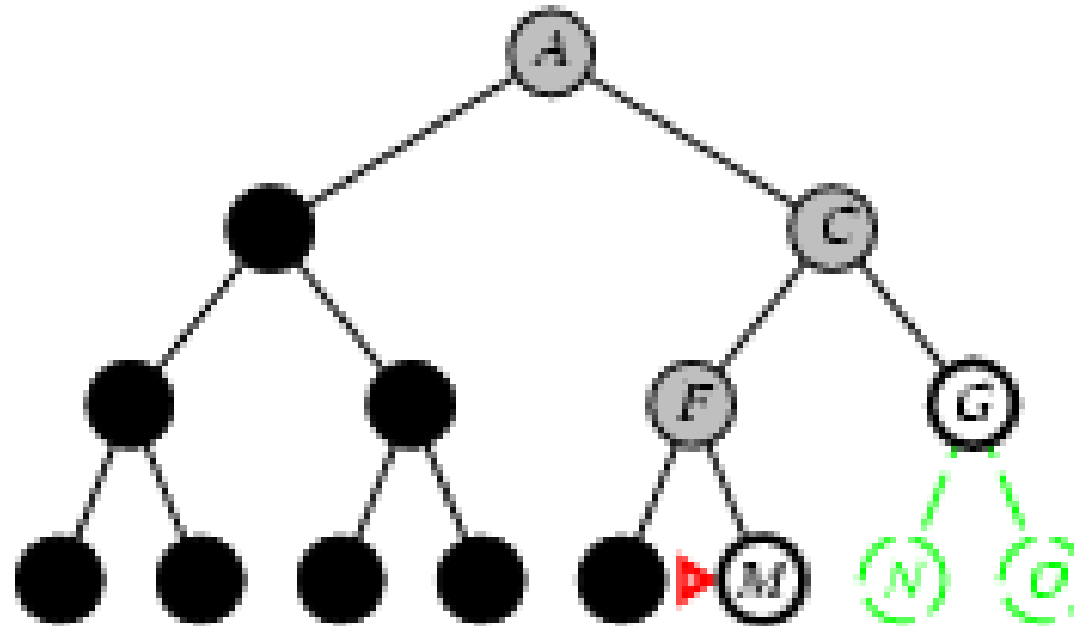
```





# Depth-first search

- Expand deepest unexpanded node
- **Implementation:**
  - *fringe* = LIFO Stack, i.e., put successors at front





# DFS: Evaluation

- DFS Graph version is complete, and **Tree Version is incomplete**
- Why?
- Time Complexity:  $O(b^m)$
- If  $m$  (maximal depth) is much larger than  $d$  (depth of shallowest solution) – time is terrible
  - If there exist multiple solutions, DFS is faster than BFS
- Space Complexity:  $O(bm)$  (in Tree Variant)
- **Non-Optimal**: it might give a solution with a higher cost



# Properties of depth-first search

- Complete? No: fails in infinite-depth spaces,
- Time?  $O(b^m)$ : terrible if  $m$  is much larger than  $d$   
 *$m$ : maximum depth of any node,  $d$ =depth of the shallowest node*
- Space?  $O(bm)$ , i.e., linear space! ( $bm+1$ )
- Optimal? No



# Depth-first Search: When it is appropriate?

## **Appropriate**

- Space is restricted (complex state representation e.g., robotics)
- There are many solutions, perhaps with long path lengths, particularly for the case in which all paths lead to a solution

## **Inappropriate**

- Cycles
- There are shallow solutions



# Why DFS need to be studied and understood?

- It is simple enough to allow you to learn the basic aspects of searching (When compared with breadth first)
  - It is the basis for a number of more sophisticated / useful search algorithms



## Variant of DFS: Backtracking

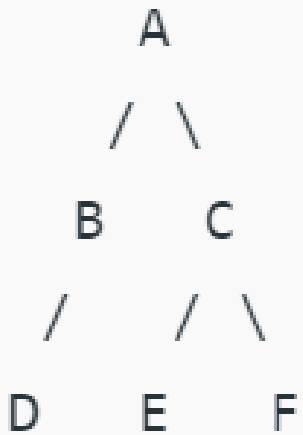
- Less Memory usage
- Only one successor generated at a time
- Each partially expanded node remembers which successor to generate next
- Memory :  $O(m)$  instead of  $O(bm)$
- Example (N-queen problem, incremental approach)



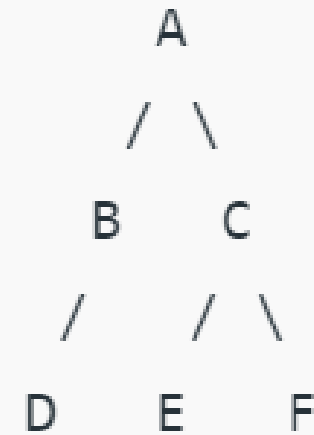
# BFS

## vs

# DFS



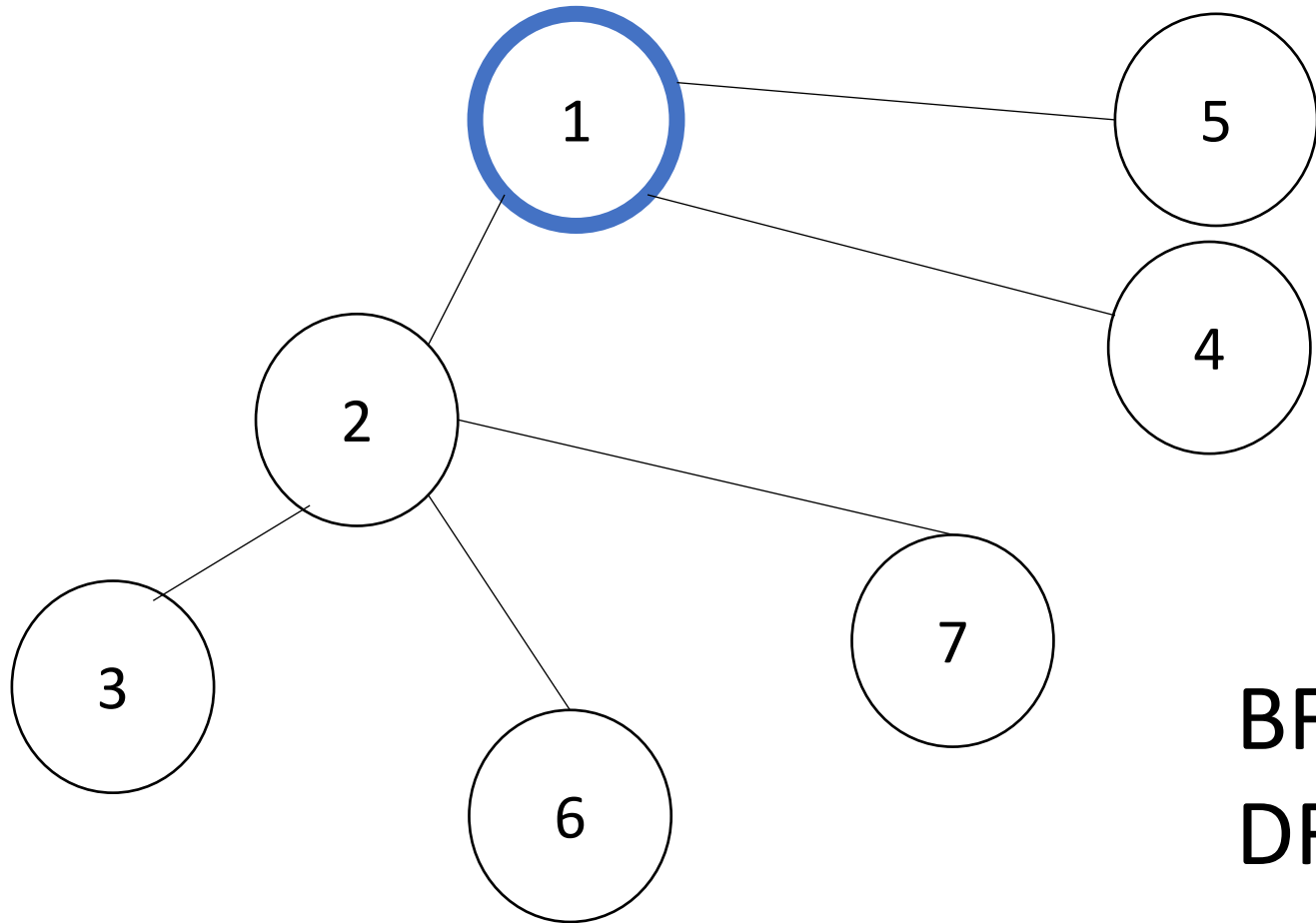
A, B, C, D, E, F



A, B, D, C, E, F



# Apply BFS and DFS on following Graph



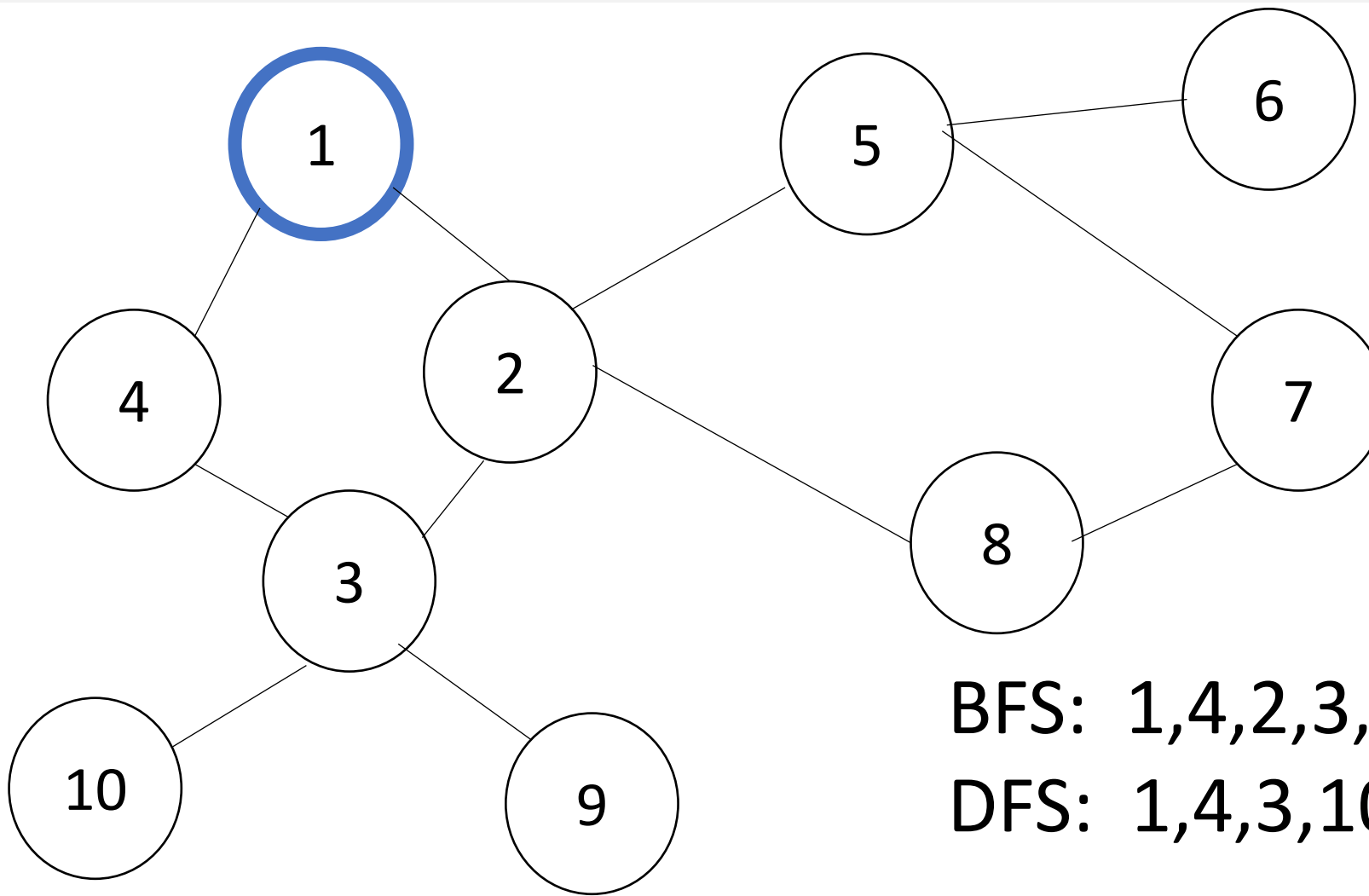
**BFS:**

**DFS:**





# Apply BFS and DFS on following Graph



BFS: 1,4,2,3,5,8,10,9,7,6

DFS: 1,4,3,10,9,2,8,7,5,6