Artificial Intelligence

Engr. Madeha Mushtaq
Department of Computer Science
Iqra National University

Problem-Solving Agents

- Intelligent agents are supposed to maximize their performance measure.
- Achieving this is sometimes simplified if the agent can adopt a goal and aim at satisfying it.
- Goal formulation, based on the current situation and the agent's performance measure, is the first step in problem solving.
- The solution to any problem is a fixed sequence of actions.
- The process of looking for a sequence of actions that reaches the goal is called search.

Problem-Solving Agents

- A search algorithm takes a problem as input and returns a solution in the form of an action sequence.
- Once a solution is found, the actions it recommends can be carried out.
- A simple problem-solving agent, thus
  - first formulates a goal and a problem,
  - searches for a sequence of actions that would solve the problem,
  - and then executes the actions one at a time.
- When this is complete, it formulates another goal and starts over.

Uniformed Search

- Uniformed search is also known as blind search.
- While searching you have no clue whether one non-goal state is better than any other, your search is blind.
- Various blind strategies:
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Iterative deepening search

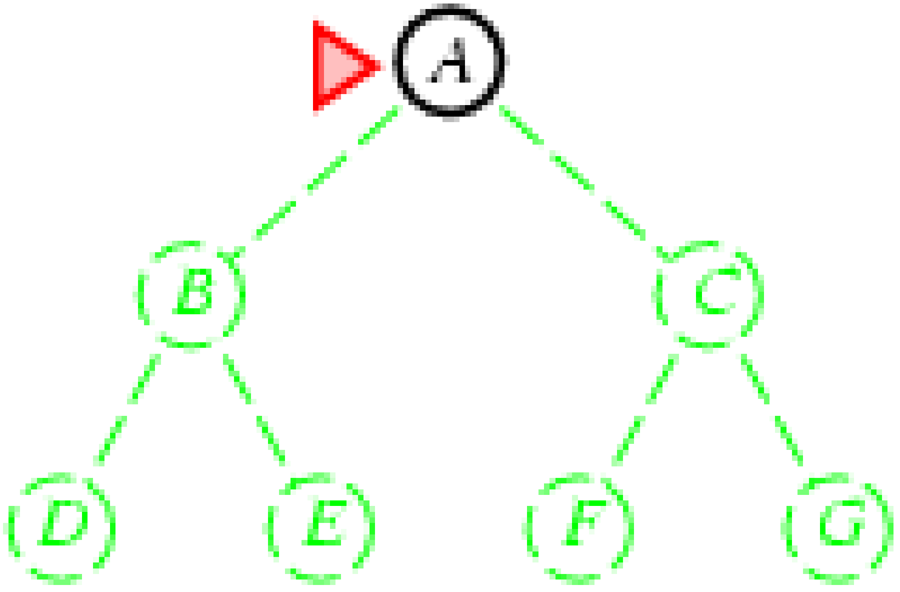
Breadth-first search (BFS)

- Breadth-first search is a simple strategy in which the root node is expanded first, then all successors of the root node are expanded next, then their successors, and so on.
- In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.

Breadth-first search (BFS)

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a first-in-first-out (FIFO) queue, i.e., new successors go at end of the queue.

Is A a goal state?

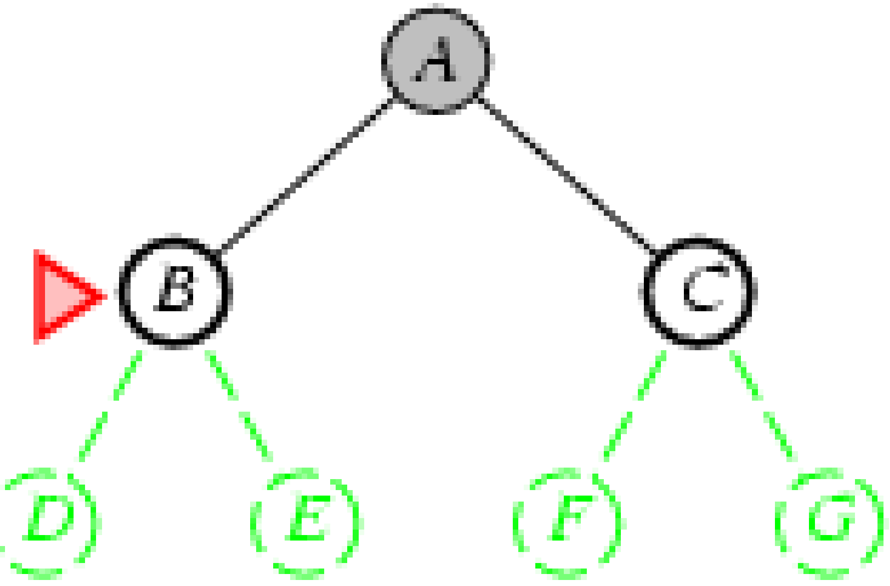


Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end

Expand:
fringe = [B,C]

Is B a goal state?

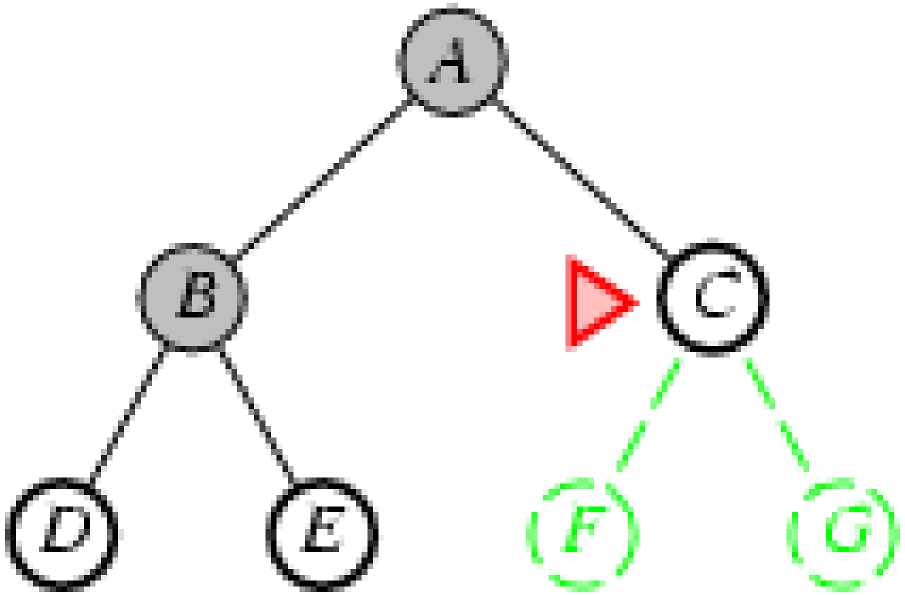


Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end

Expand:
fringe=[C,D,E]

Is C a goal state?

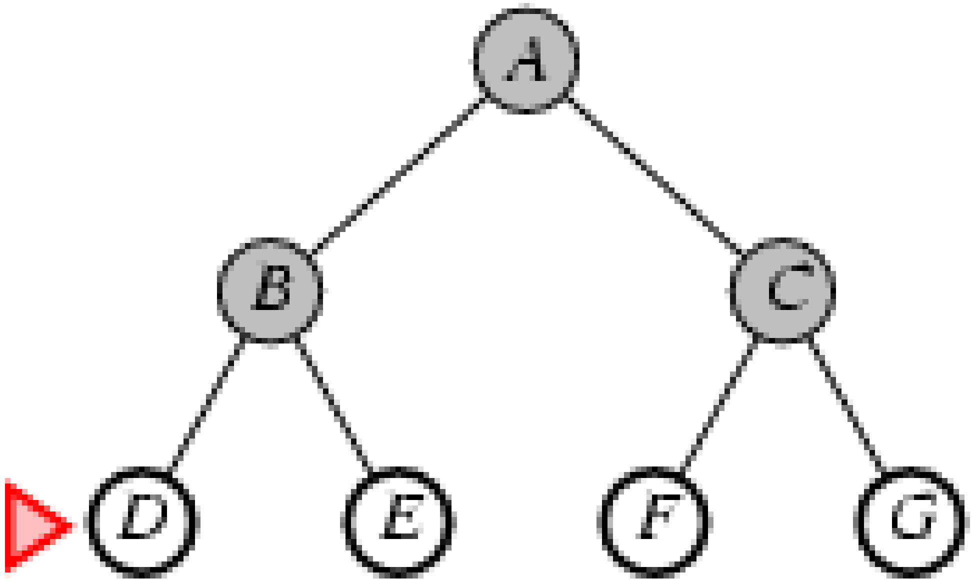


Breadth-first search

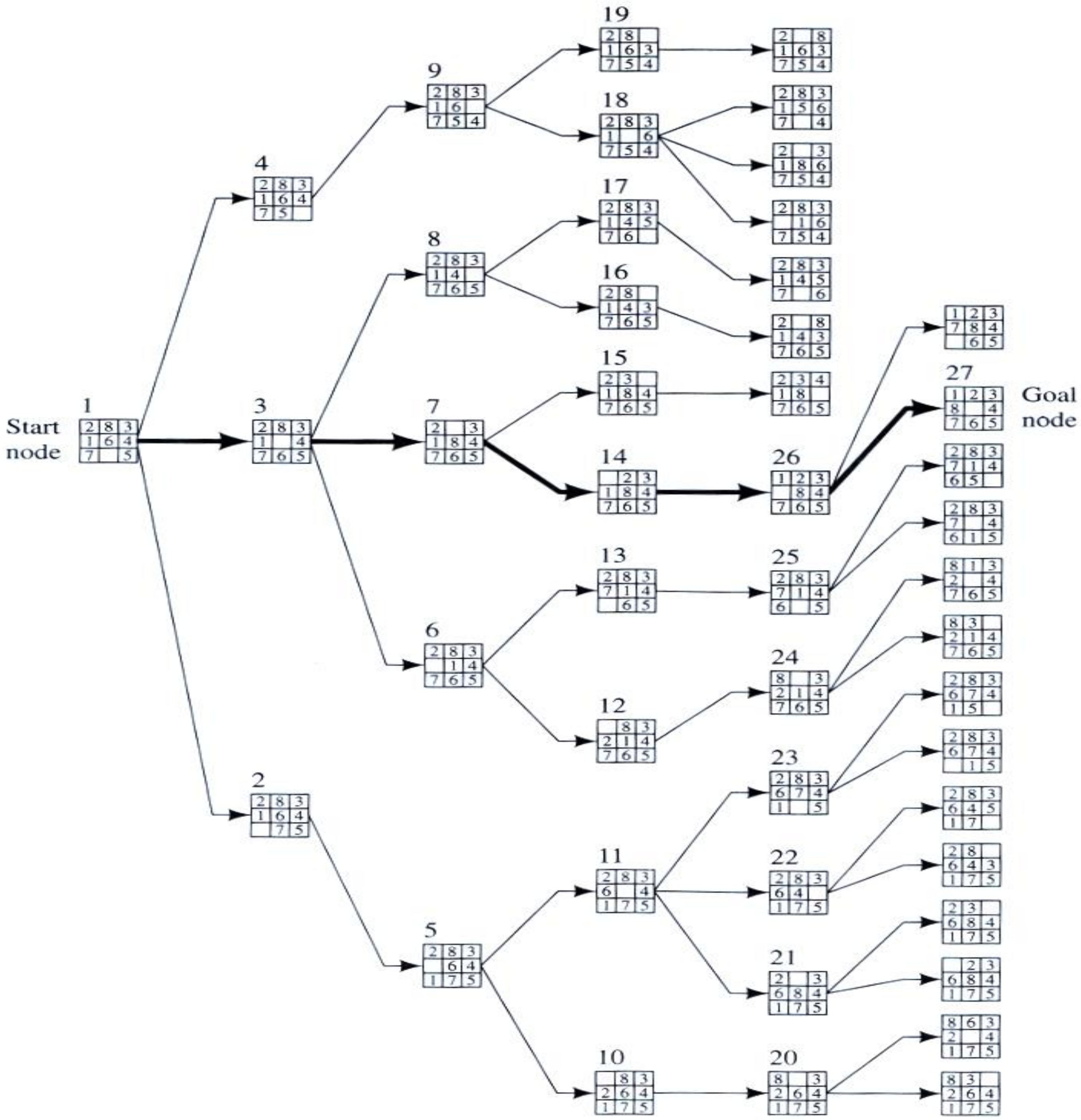
- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end

Expand:
fringe=[D,E,F,G]

Is D a goal state?



Example BFS


Start node
Goal node
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

Breadth-first search

- Technically, breadth-first search is optimal if the path cost is a non-decreasing function of the depth of the node.
- The most common such scenario is that all actions have the same cost.
- So far, the news about breadth-first search has been good.
- The news about time and space is not so good.
- Imagine searching a uniform tree where every state has b successors.
- The root of the search tree generates b nodes at the first level, each of which generates b more nodes, for a total of b^2 at the second level.
- Each of these generates b more nodes, yielding b^3 nodes at the third level, and so on.

Breadth-first search

- Now suppose that the solution is at depth d.
- In the worst case, it is the last node generated at that level.
- Then the total number of nodes generated is $b + b^2 + b^3 + \cdots + b^d = O(b^d)$.
- Space Complexity is $O(b^{d+1})$ (keeps every node in memory, either in fringe or on a path to fringe).
- BFS is optimal if we guarantee that deeper solutions are less optimal, e.g. step-cost=1).
- So space is the bigger problem (more than time).

Uniform-cost search

- When all step costs are equal, breadth-first search is optimal because it always expands the shallowest unexpanded node.
- By a simple extension, we can find an algorithm that is optimal with any step-cost function.
- Instead of expanding the shallowest node, uniform-cost search expands the node n with the lowest path cost g(n).
- This is done by storing the frontier as a priority queue ordered by g.

Uniform-cost search

- Uniform-cost search is optimal in general.
- Uniform-cost search does not care about the number of steps a path has, but only about their total cost.
- Therefore, it can get stuck in an infinite loop if there is a path with an infinite sequence of zero-cost actions.
- Uniform-cost search is guided by path costs rather than depths, so its complexity is not easily characterized in terms of b and d.

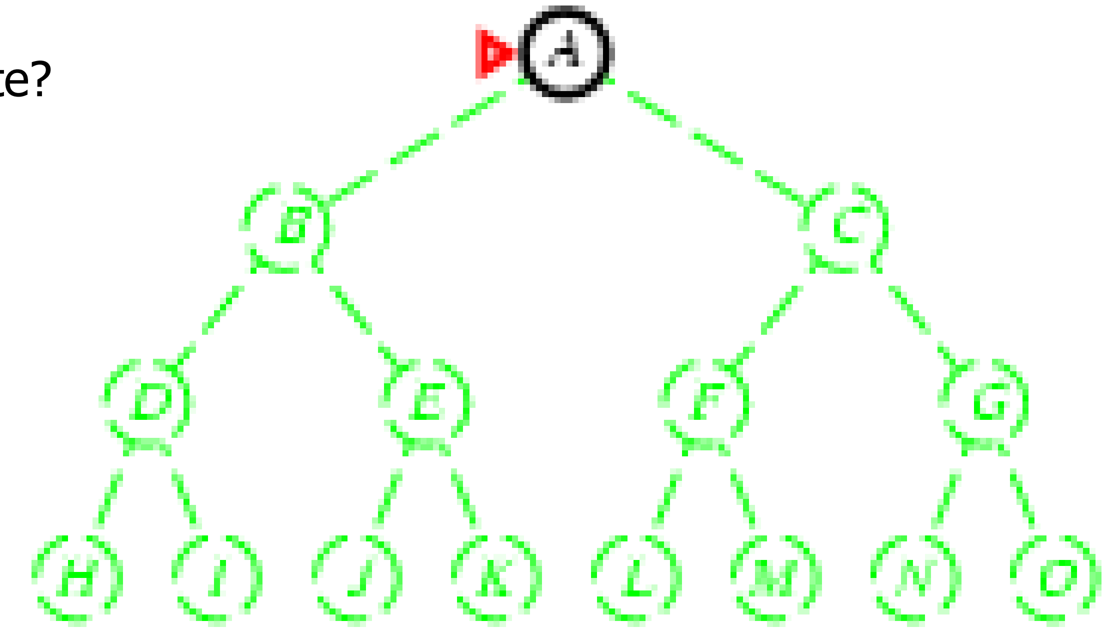
Depth-first search

- Depth-first search always expands the deepest node in the current frontier of the search tree.
- The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors.
- As those nodes are expanded, they are dropped from the frontier, so then the search “backs up” to the next deepest node that still has unexplored successors.

Depth-first search

- Expand *deepest* unexpanded node
- Implementation:
  - *fringe* = Last In First Out (LIPO) queue, i.e., put successors at front

Is A a goal state?

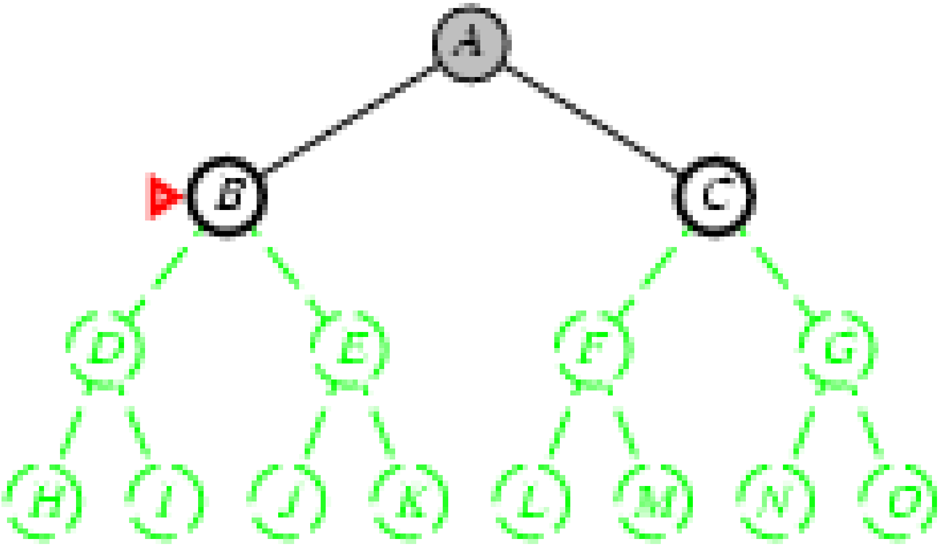


Depth-first search

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

queue=[B,C]

Is B a goal state?

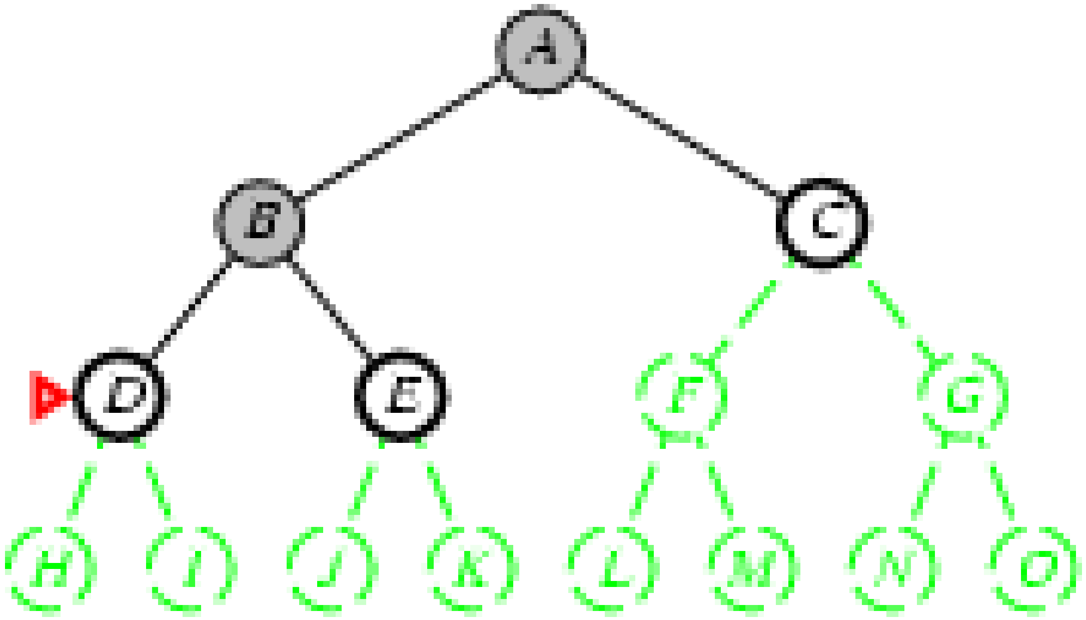


Depth-first search

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

queue=[D,E,C]

Is D = goal state?

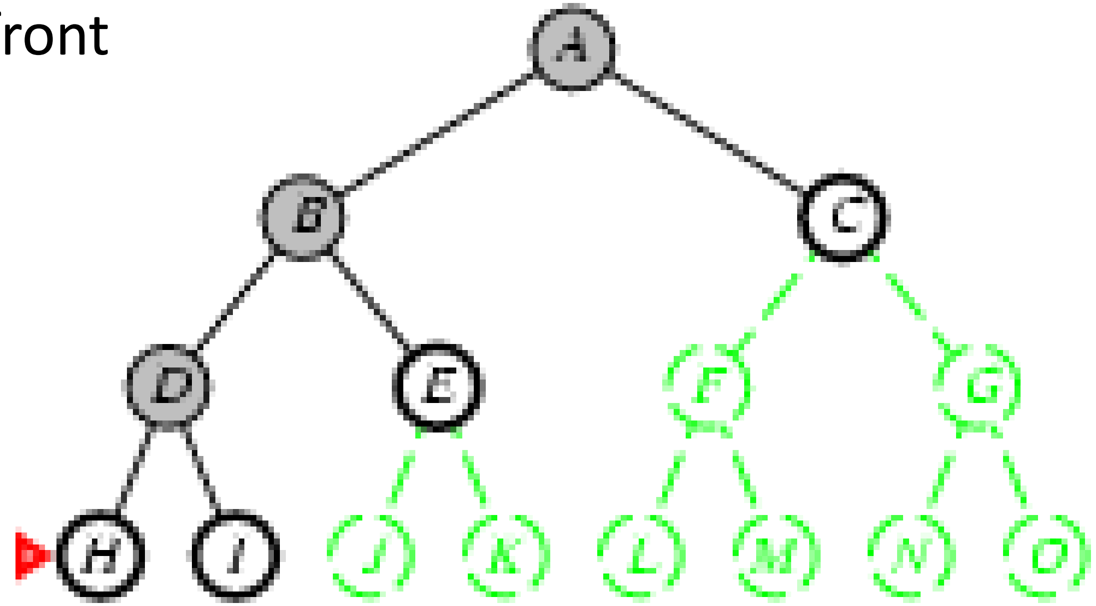


Depth-first search

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

queue=[H,I,E,C]

Is H = goal state?

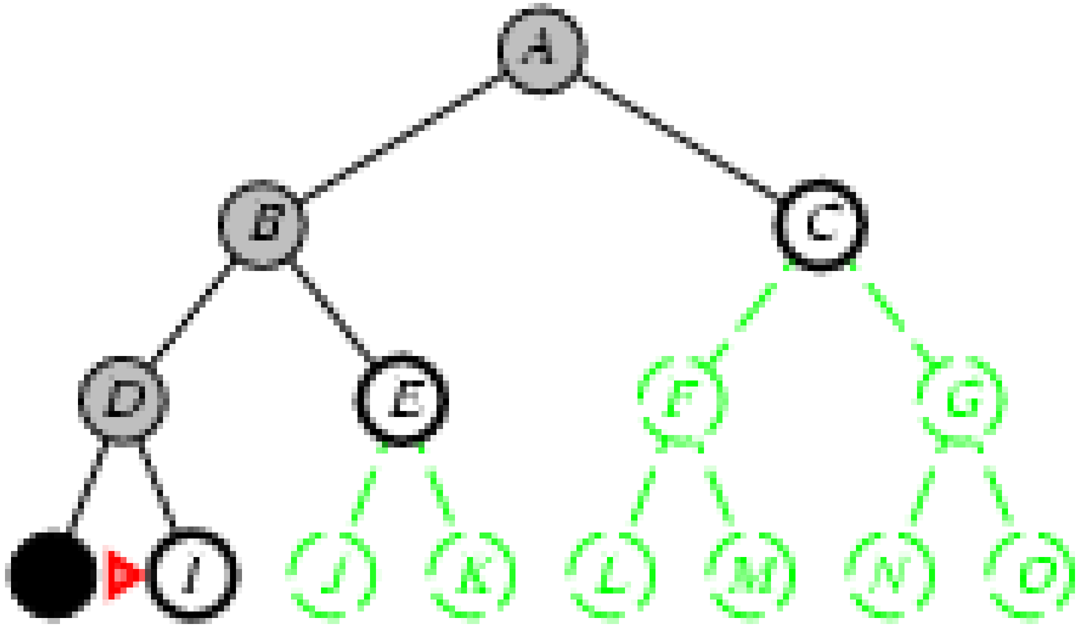


Depth-first search

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

queue=[I,E,C]

Is I = goal state?

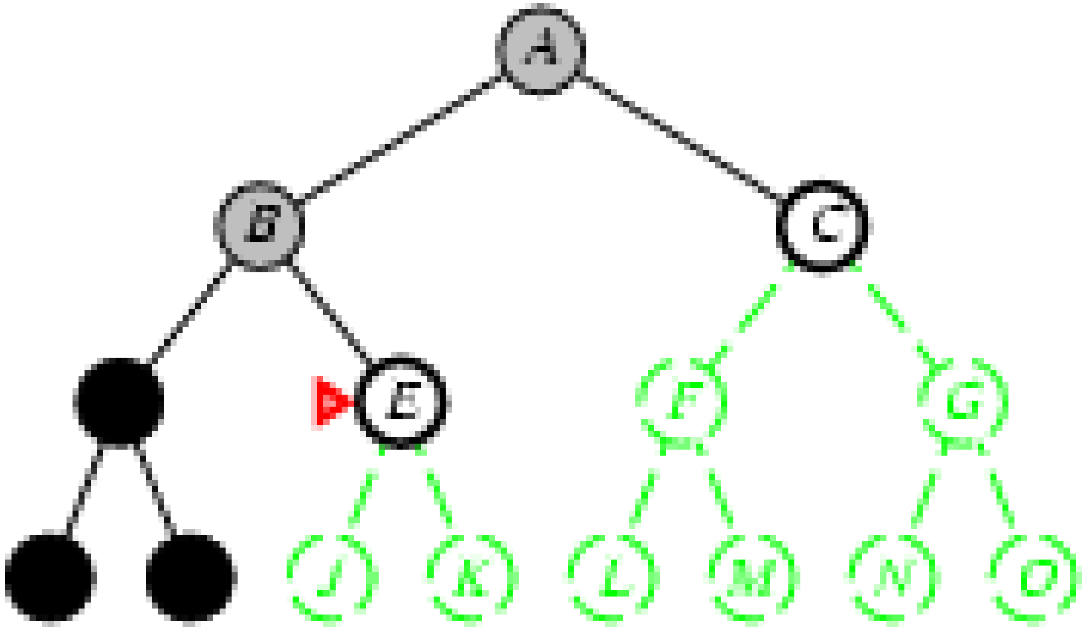


Depth-first search

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

queue=[E,C]

Is E = goal state?

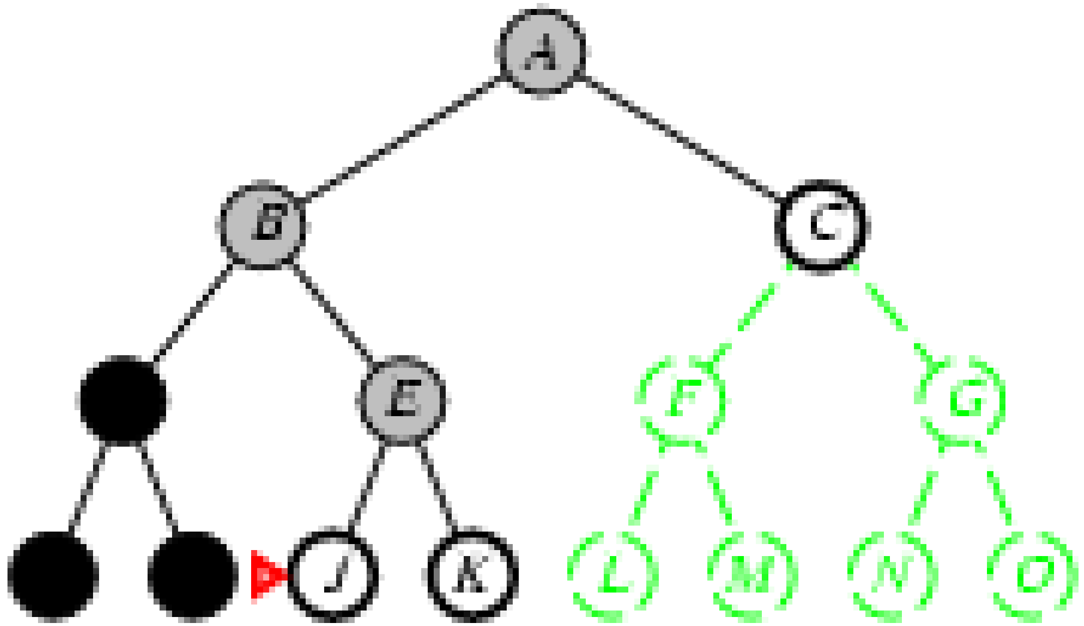


Depth-first search

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

queue=[J,K,C]

Is J = goal state?

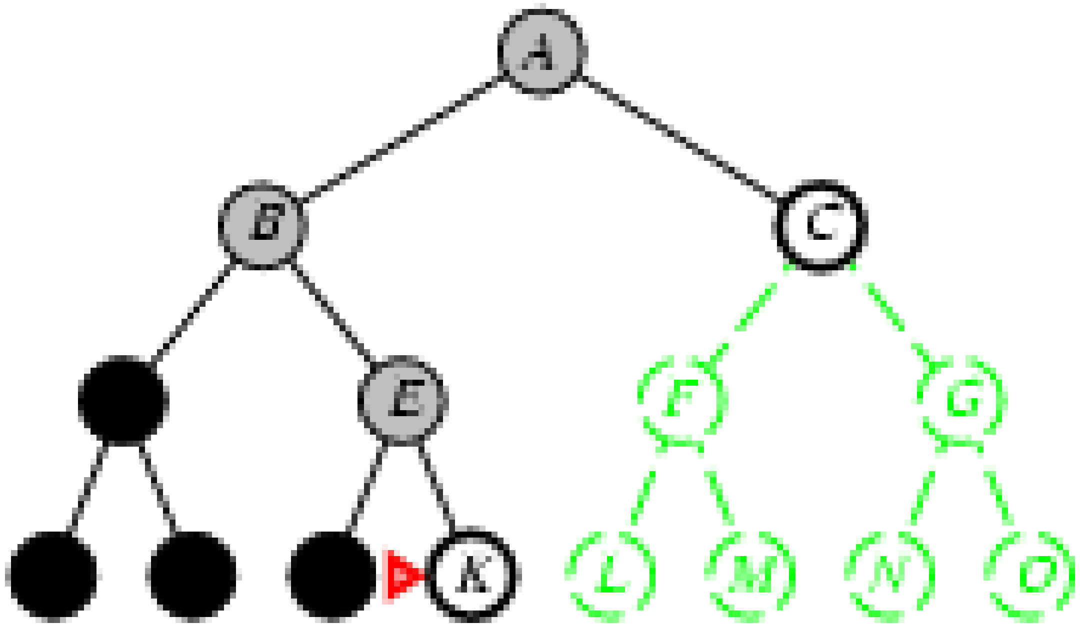


Depth-first search

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

queue=[K,C]

Is K = goal state?

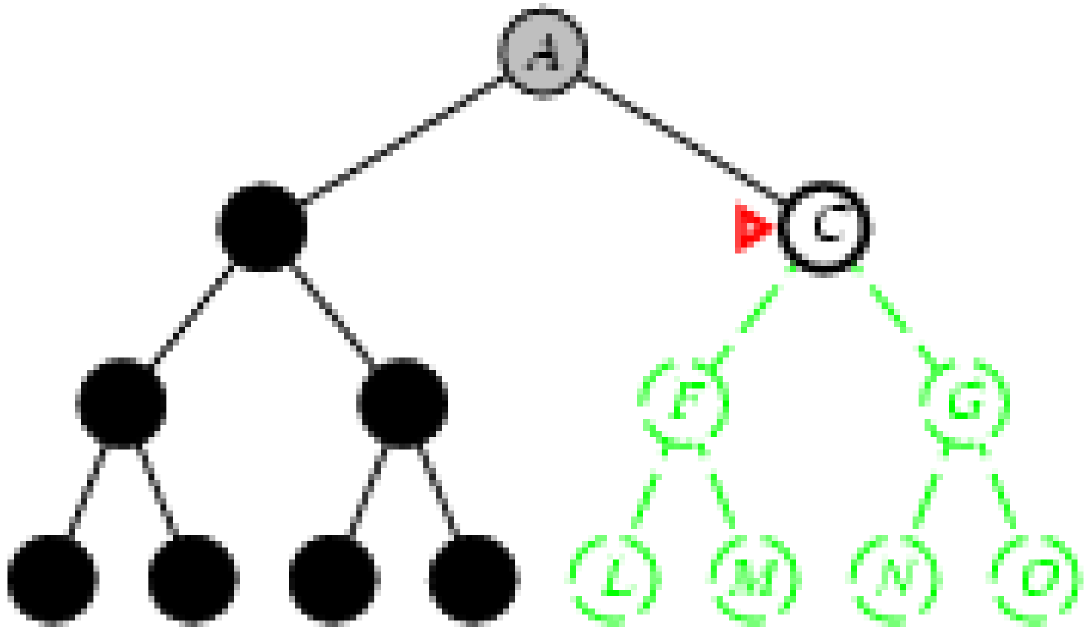


Depth-first search

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

queue=[C]

Is C = goal state?

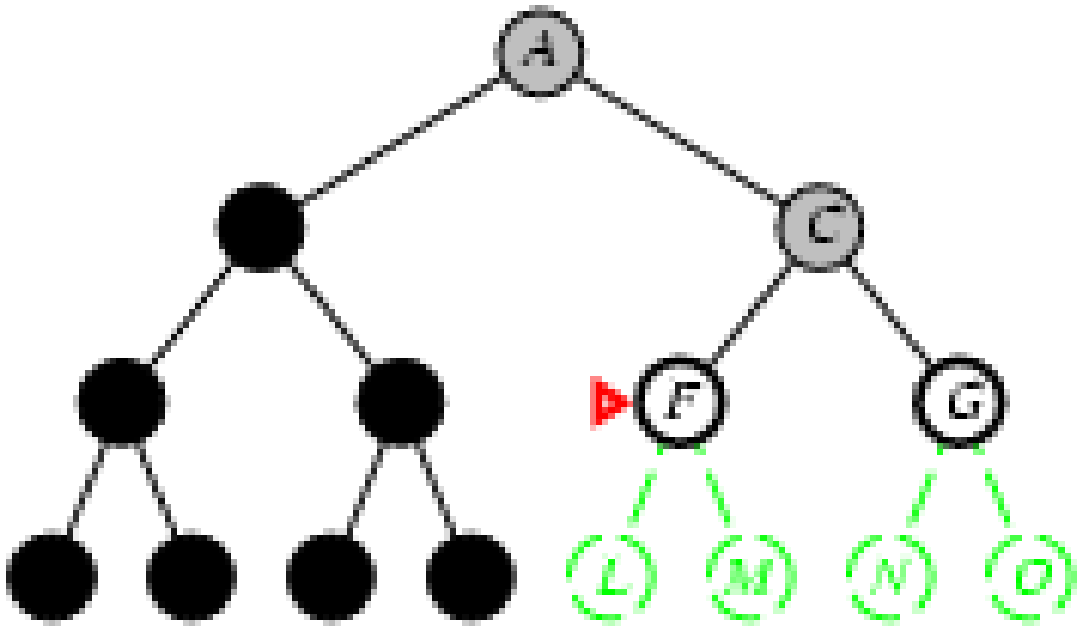


Depth-first search

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

queue=[F,G]

Is F = goal state?

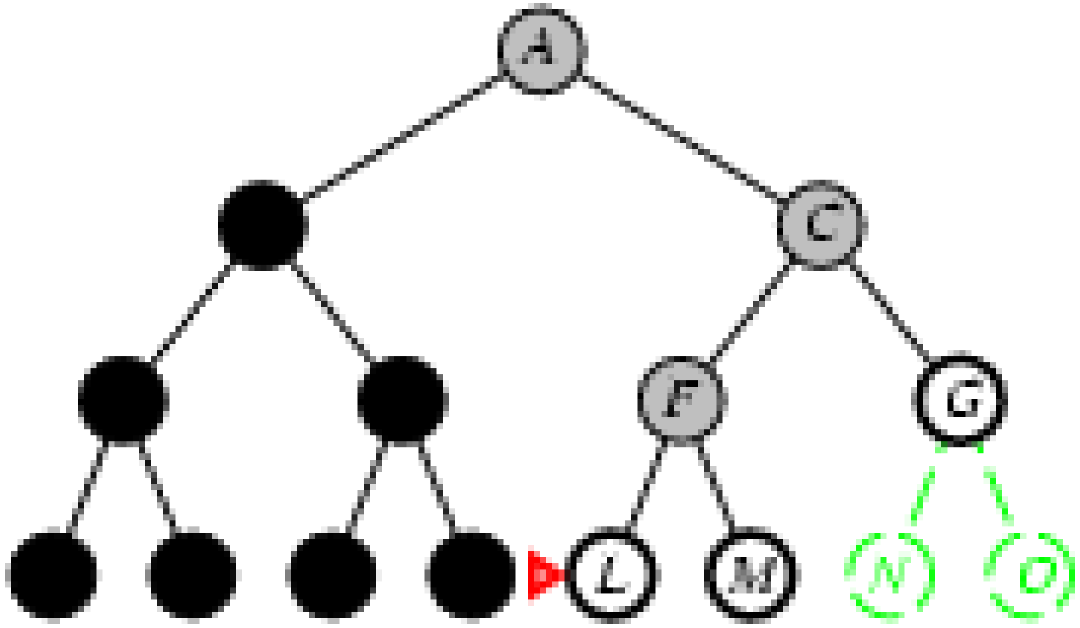


Depth-first search

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

queue=[L,M,G]

Is L = goal state?

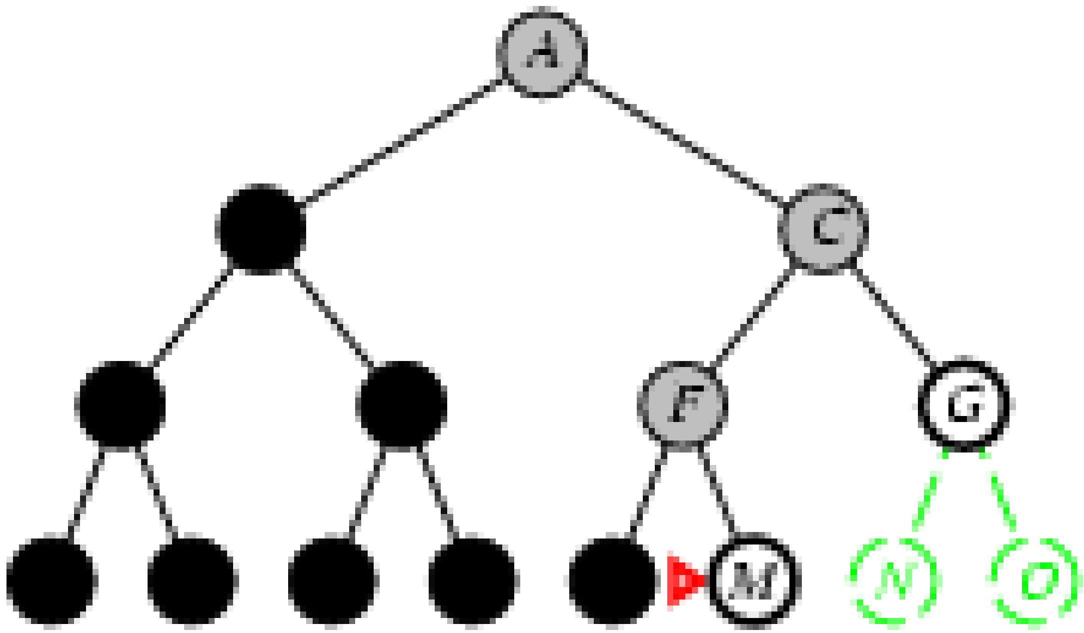


Depth-first search

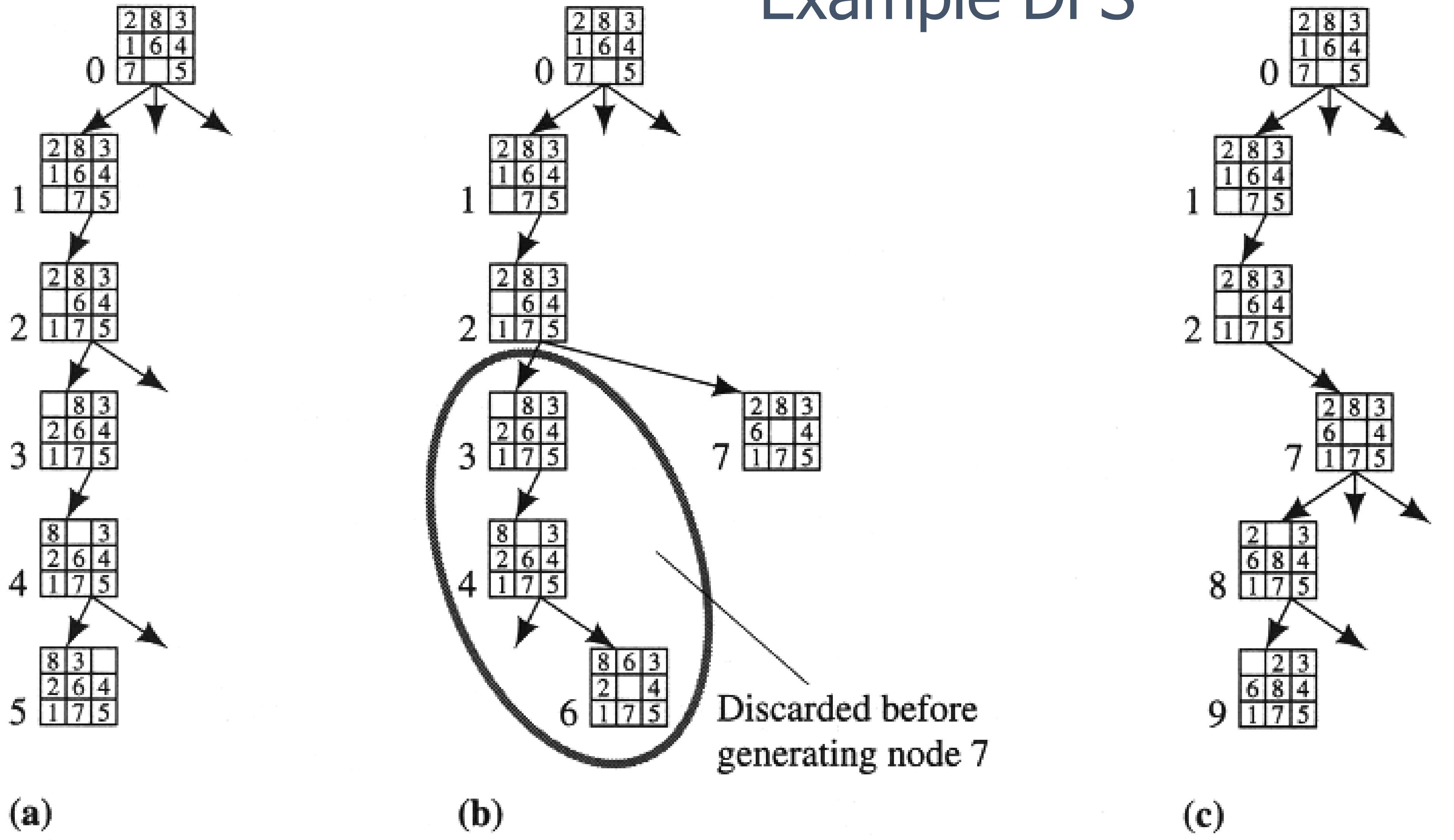
- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[M,G]

Is M = goal state?



Example DFS



Generation of the First Few Nodes in a Depth-First Search

Depth-first search

- DFS is not complete, it fails in infinite-depth spaces
- Time Complexity is $O(b^m)$ with m=maximum depth
- terrible if *m* is much larger than *d*
  - but if solutions are dense, may be much faster than breadth-first
- Space complexity is $O(bm)$, i.e., linear space!
- (we only need to remember a single path + expanded unexplored nodes)
- It is not optimal (It may find a non-optimal goal first).

Iterative deepening search (IDS)

- To avoid the infinite depth problem of DFS, we can decide to only search until depth L, i.e. we don’t expand beyond depth L.
- Depth-Limited Search

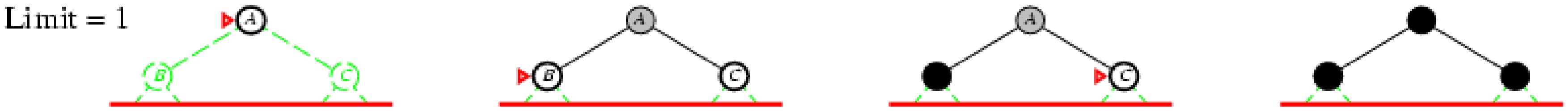
- What if solution is deeper than L? → Increase L iteratively.
- Iterative Deepening Search

- As we shall see: this inherits the memory advantage of Depth-First search.

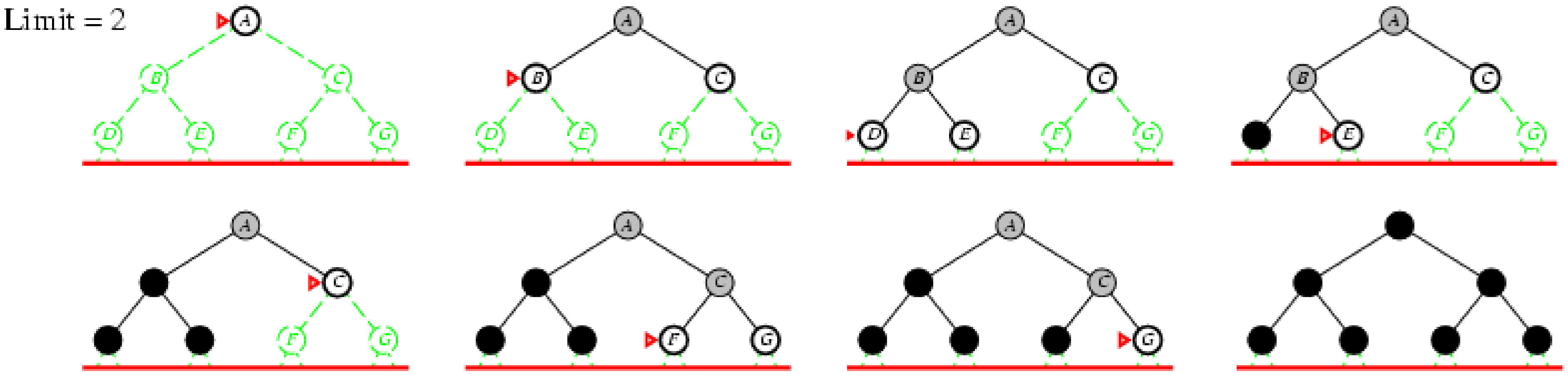
Iterative deepening search L=0

Limit = 0

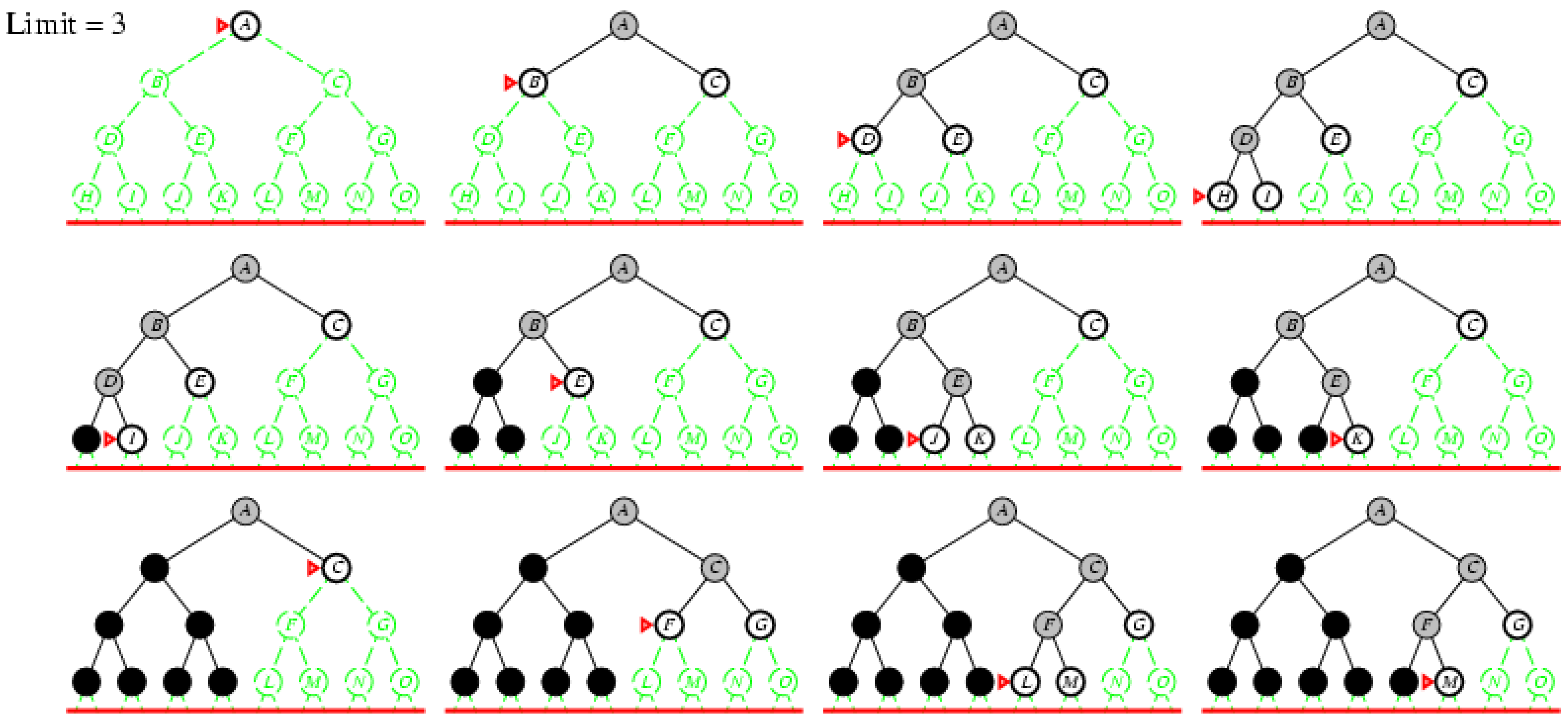
Iterative deepening search L=1



Iterative deepening search *L*=2



Iterative deepening search *L*=3



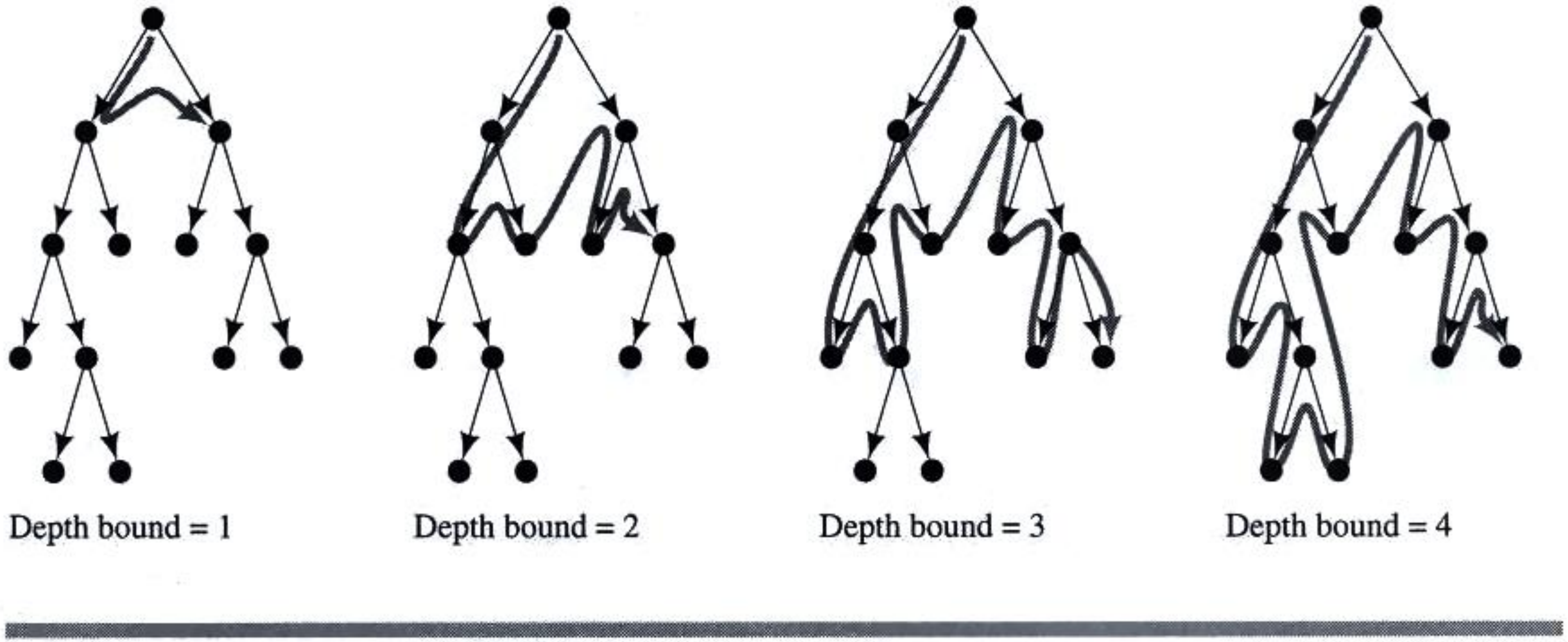
Properties of iterative deepening search

- Complete: Yes
- Time Complexity: $(d+1)b^0 + d\ b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space Complexity: $O(bd)$
- Optimal: Yes, if step cost = 1 or increasing function of depth.

Iterative deepening search

- In general, iterative deepening is the preferred uninformed search method when the search space is large and the depth of the solution is not known.

Example IDS



Stages in Iterative–Deepening Search

Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

End of Slides