# Lecture 4

## Uninformed Search

**Lusi Li**
**Department of Computer Science**
**ODU**

Reading for This Class:
Chapter 3, Russell and Norvig



Special thanks to Dr. Yaohang Li for sharing the slides

# Review

- **Last Class**
  - Problem-Solving Algorithms
    - Tree Search (Problem, Strategy)
    - Graph Search (Problem, Strategy)

- **This Class**
  - Uninformed Search Strategies
    - Breadth-first Search
    - Depth-first Search
    - Uniform-cost Search
  - Performance Measure
  - Start Assignment 1

- **Next Class**
  - Informed Search Strategies

# Uninformed Search Strategies

- **Uninformed: you have no clue whether one non-goal state is better than any other. Your search is blind.**
  - **E.g., agent in Arad with no knowledge of Romania road map, has no clue whether Go(Z) or Go(S) or Go(T) is better**
  - **All they can do is to generate successors and distinguish a goal state from a non-goal state.**

- **Various blind strategies:**
  - **Breadth-first search**
  - **Depth-first search**
  - **Uniform-cost search**
  - **Iterative deepening search (optional)**
  - **Bidirectional search (optional)**

# Search Strategy Evaluation

- **Performance measure**
  - **Complete?** Guaranteed to find a solution when there is one?
  - **Optimal?** Find the cheapest solution?
  - **Time complexity?** How long does it take to find a solution?
  - **Space complexity?** How much memory is needed to perform the search?

- **Time and space complexity are measured in terms of**
  - ***b:* maximum branching factor of the search tree**
  - ***d:* depth of the least-cost solution**
  - ***m*: maximum depth of the state space (may be ∞)**

# Uninformed Search Strategy Choices

- **Queue for Frontier:**
  - FIFO? LIFO? Priority?
- **Goal-Test:**
  - Do goal-test when node is generated?
  - Do goal-test when node is expanded?
- **Tree Search or Graph Search:**
  - Forget Expanded (or Explored, Closed) nodes?
    - Tree Search: Smaller memory cost, but larger search time
  - Or remember them?
    - Graph Search: Smaller search time, but larger memory cost
  - Classic space/time computational tradeoff

# Queue for Frontier

- FIFO (First In, First Out)
  - Results in Breadth-First Search
- LIFO (Last In, First Out)
  - Results in Depth-First Search
- Priority Queue sorted by path cost so far
  - Results in Uniform Cost Search
- Iterative Deepening Search uses Depth-First
- Bidirectional Search can use either Breadth-First or Uniform Cost Search

# When to do Goal-Test?

- Expand: a node is expanded when all its child nodes are generated
- Generate: a node is generated when its parent node is expanded

- Do Goal-Test when node is selected for expansion

  IF you care about finding the optimal path

  AND your search space may have both short expensive and long cheap paths to a goal.
  - Guard against a short expensive goal.
  - E.g., Uniform Cost search with variable step costs.

- Do Goal-Test when node is generated.
  - E.g., Breadth-first Search, Depth-first Search, or Uniform Cost search when cost is a non-decreasing function of depth only (which is equivalent to Breadth-first Search).

# Implementation of Breadth-first Search

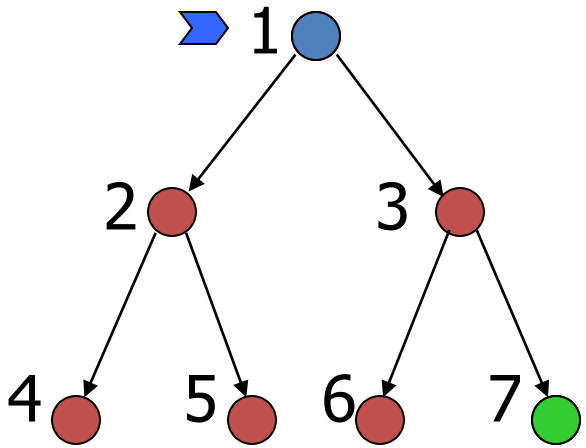
- Strategy: expand shallowest unexpanded node in Frontier
- Frontier: a set of nodes in queue to be expanded
  - Queue ("first in first out")
- E.g., Goal-test applied when node is generated
- Insert all generated successors at the end of the queue
    - Shallow nodes are expanded before deeper nodes

# Breadth-First Strategy in Tree Search

**The root node = Node 1**

**If problem. IS-GOAL(Node 1.STATE) then return node**

**Insert Node 1 at the end of FRONTIER**




FRONTIER = (1)

FRONTIER = ()

**Future = red nodes**

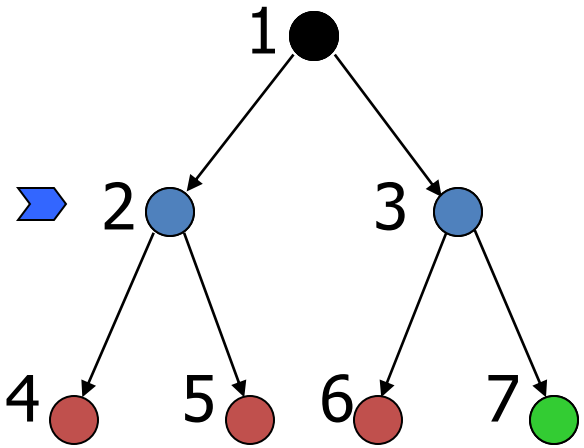
**Goal = green nodes**

**Frontier = blue nodes**

**Expanded = black nodes**

# Breadth-First Strategy in Tree Search

**Expand Node 1 to generate Node 2 and Node 3**
**If problem. IS-GOAL(Node 2.STATE) then return node**
**Insert Node 2 at the end of FRONTIER**
**If problem. IS-GOAL(Node 3.STATE) then return node**
**Insert Node 3 at the end of FRONTIER**



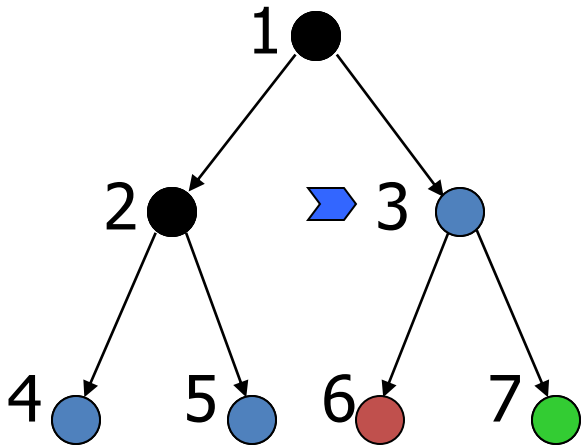

FRONTIER = (2, 3)

FRONTIER = (3)

**Future = red nodes**
**Goal = green nodes**
**Frontier = blue nodes**
**Expanded = black nodes**

# Breadth-First Strategy in Tree Search

**Expand Node 2 to generate Node 4 and Node 5**
**If problem. IS-GOAL(Node 4.STATE) then return node**
**Insert Node 4 at the end of FRONTIER**
**If problem. IS-GOAL(Node 5.STATE) then return node**
**Insert Node 5 at the end of FRONTIER**




FRONTIER = (3, 4, 5)

FRONTIER = (4, 5)

**Future = red nodes**
**Goal = green nodes**
**Frontier = blue nodes**
**Expanded = black nodes**

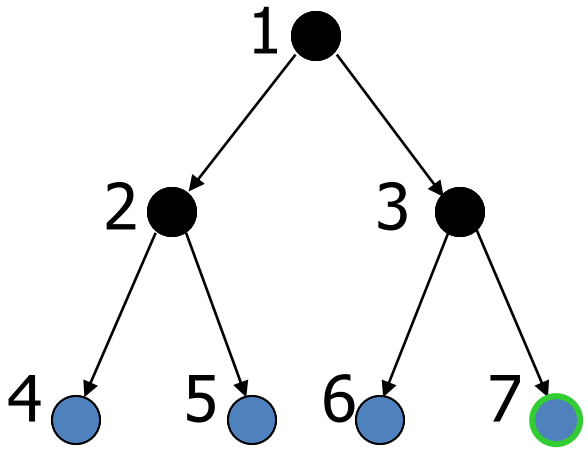
# Breadth-First Strategy in Tree Search

**Expand Node 3 to generate Node 6 and Node 7**

**If problem. IS-GOAL(Node 6.STATE) then return node**

**Insert Node 6 at the end of FRONTIER**

**If problem. IS-GOAL(Node 7.STATE) then return node**




FRONTIER = (4, 5, 6)

**Future = red nodes**

**Goal = green nodes**

**Frontier = blue nodes**

**Expanded = black nodes**

# Breadth-First Strategy in Graph Search Example

- **8-puzzle problem example**

| | | |
|---|---|---|
| 2 | 8 | 3 |
| 1 | 6 | 4 |
| 7 | | 5 |

**Initial State**

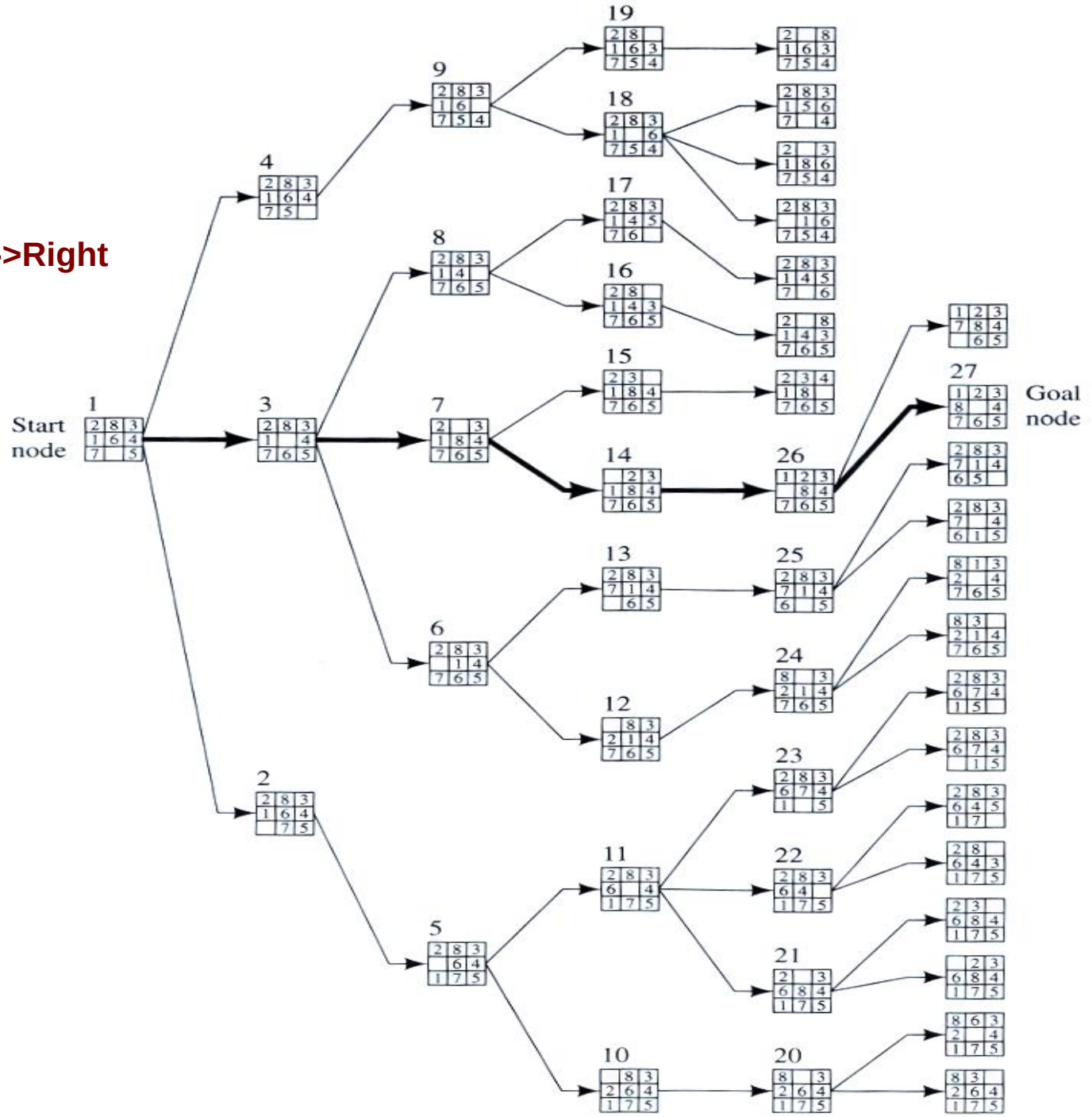
| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | | 4 |
| 7 | 6 | 5 |

**Goal State**

# BFS Example

- **Solution**

**Up -> Up -> Left -> Down ->Right**



# Evaluation of Breadth-First Search

- **Complete?**
  - **Yes, it always reaches a goal (if *b* and *d* are finite)**
- **Optimal?**
  - **No, for general cost functions.**
  - **Yes, only if cost is a non-decreasing function only of depth.**
- **Time?**
  - **$1 + b + b^2 + \ldots + b^d = (b^{d+1}-1)/(b-1) = O(b^d)$**
  - **This is the number of nodes we generate**
- **Space?**
  - **$O(b^d)$ for both Tree Search and Graph Search**
  - **All nodes at a given level need to be stored in the memory until the goal node is found or the entire tree is explored**
- **Usually space is the bigger problem (more than time)**

**b: branching factor (**maximum number of successors of any node**)**
**d: depth of shallowest goal node (**the number of steps along the path from the root**)**

# Bidirectional Search

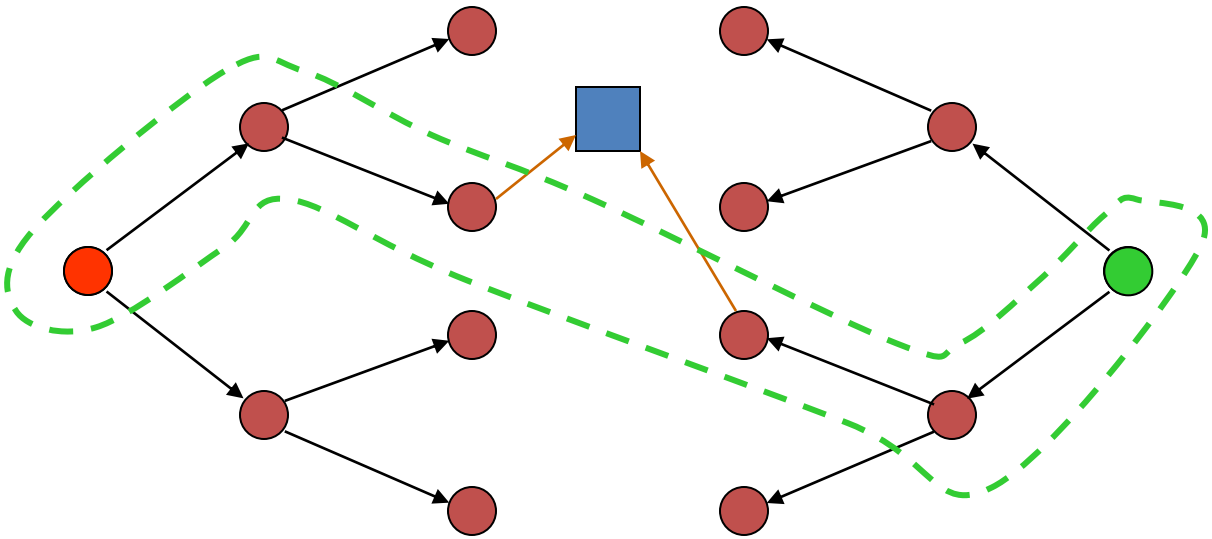
- Improve the complexity of BFS
- Run two simultaneous searches
  - One forward from the initial state
  - The other backward from the goal
  - Stopping when the two frontiers intersect

- Implementation
  - Keep two frontiers
  - Compare two frontiers at every step (instead of goal-test)
  - When the intersection is not empty, the search stops

- Analysis
  - Optimality: Yes with uniform cost
  - Complete: Yes
  - Time and space complexity: $O(b^{d/2})$

# Bidirectional Strategy

2 frontier queues: FRONTIER1 and FRONTIER2



Time and space complexity = $O(b^{d/2}) << O(b^d)$

# Implementation of Depth-first Search

- Strategy: Expand deepest unexpanded node
- Frontiers
  - Stack (“last in first out”)
  - Push all newly generated successors into the stack

- Now, we try Goal-Test when node is selected for expansion

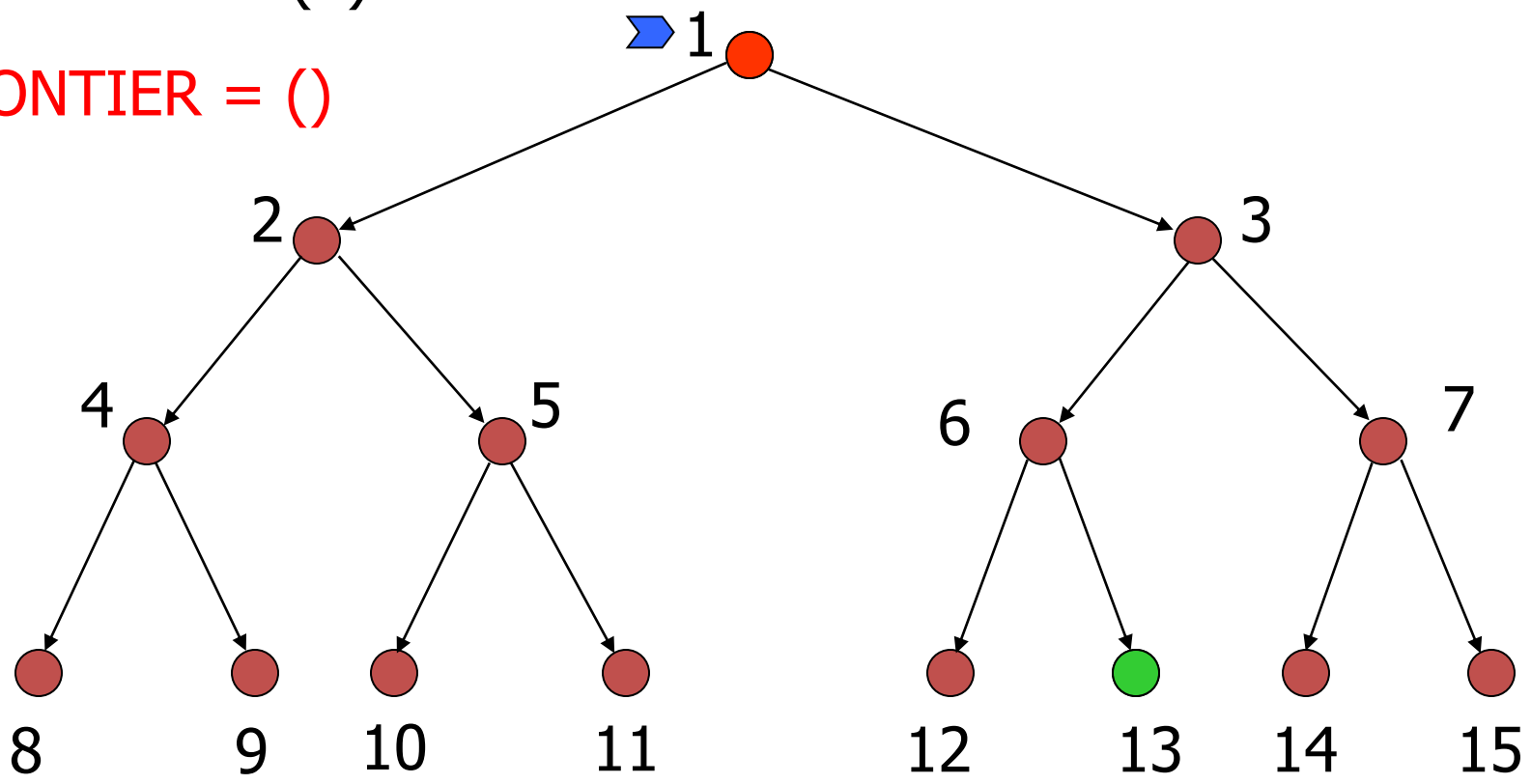
# Depth-First Strategy

**New nodes are inserted at the front of FRONTIER**

**Future = red nodes**
**Goal = green nodes**
**Frontier = blue nodes**
**Expanded = black nodes**
**Forgotten = white nodes**

FRONTIER = (1)

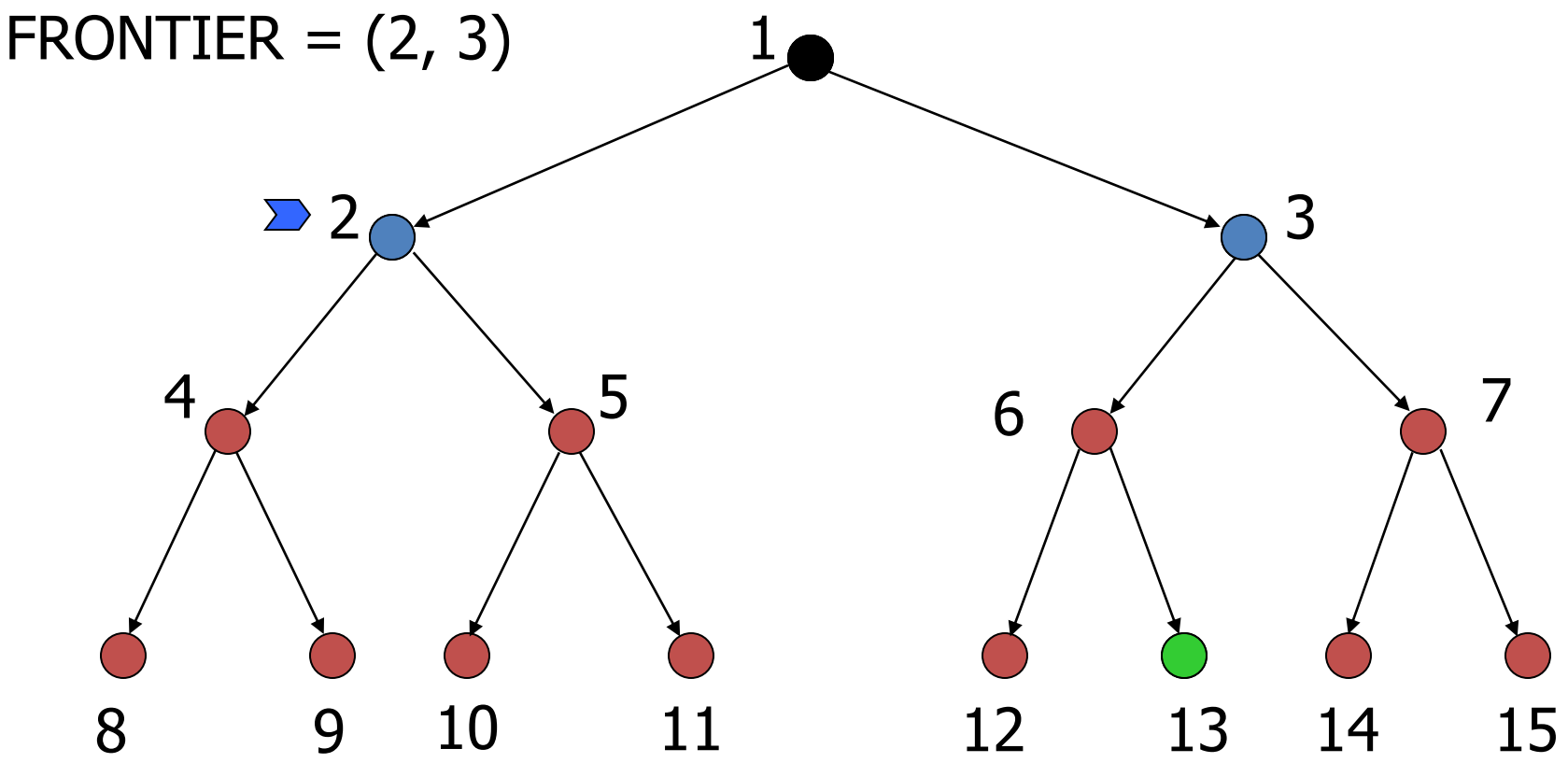
FRONTIER = ()



# Depth-First Strategy

- Remove Node 1 from frontier
- Do Goal-Test
- Generate Node 2, Node 3 and push them into FRONTIER

Future = red nodes
Goal = green nodes
Frontier = blue nodes
Expanded = black nodes
Forgotten = white nodes

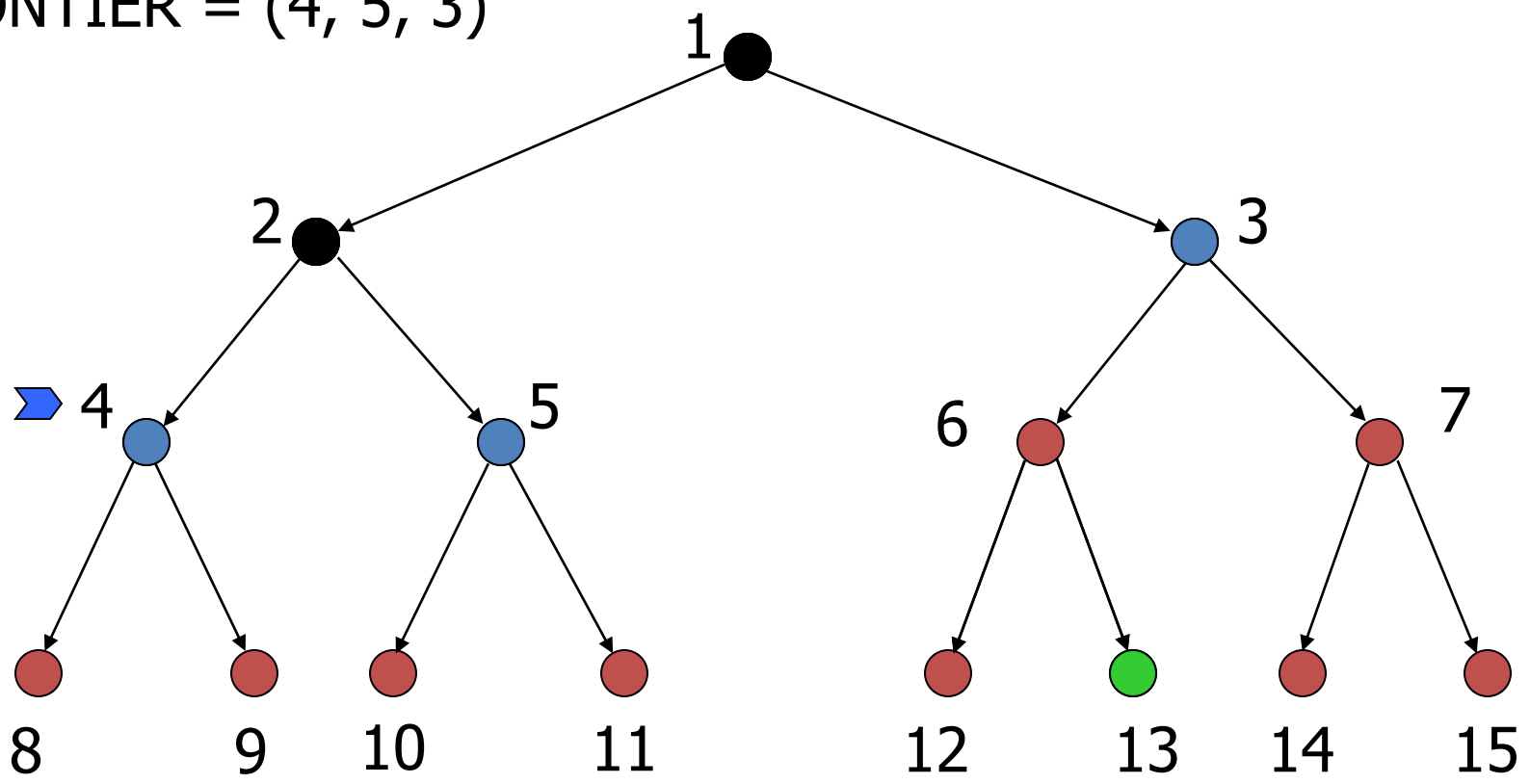


# Depth-First Strategy

- Remove Node 2 from frontier
- Do Goal-Test
- Generate Node 4, Node 5 and push them into FRONTIER

Future = red nodes
Goal = green nodes
Frontier = blue nodes
Expanded = black nodes
Forgotten = white nodes

FRONTIER = (4, 5, 3)

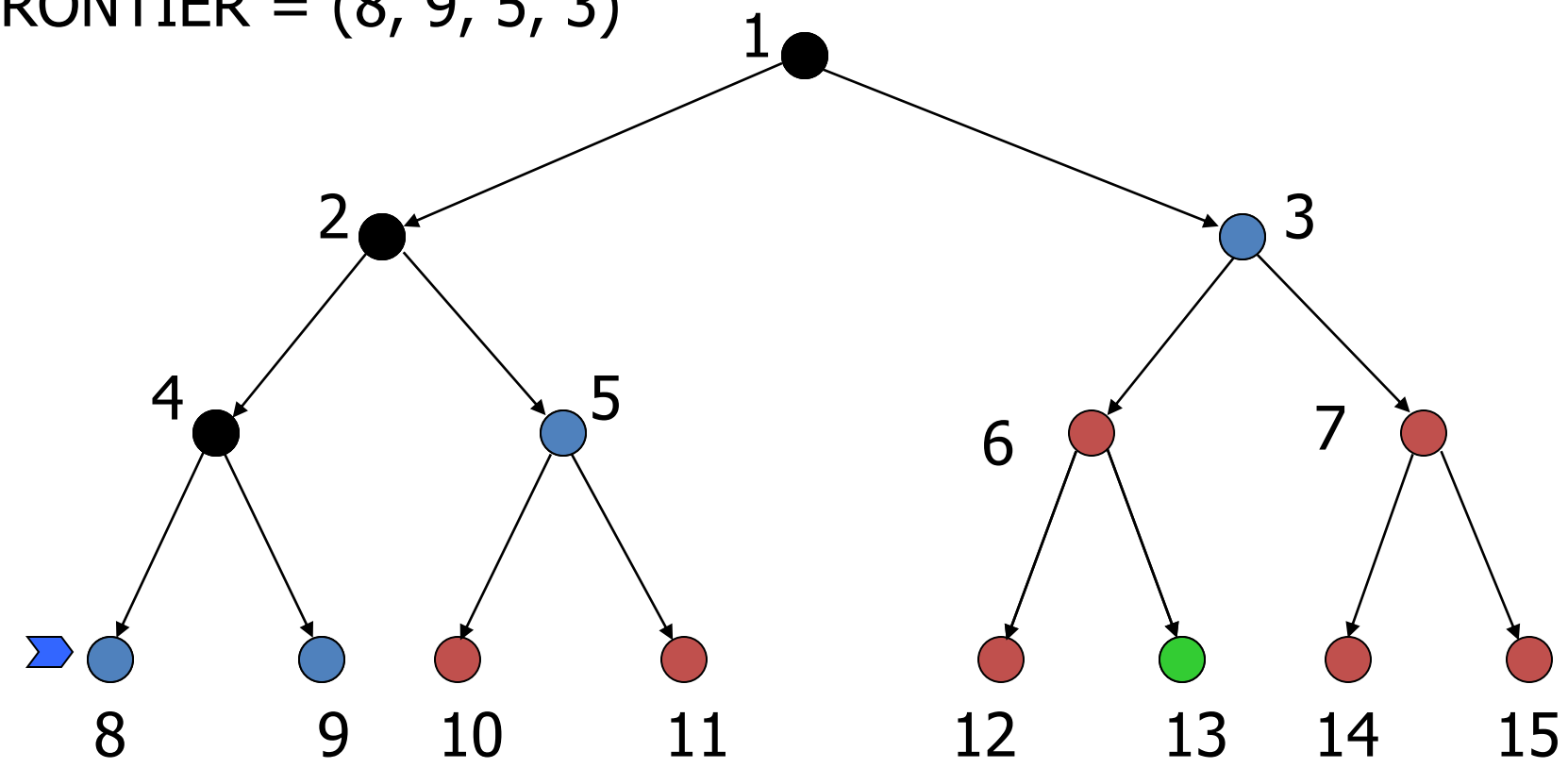


# Depth-First Strategy

- Remove Node 4 from frontier
- Do Goal-Test
- Generate Node 8, Node 9 and push them into FRONTIER

Future = red nodes
Goal = green nodes
Frontier = blue nodes
Expanded = black nodes
Forgotten = white nodes

FRONTIER = (8, 9, 5, 3)

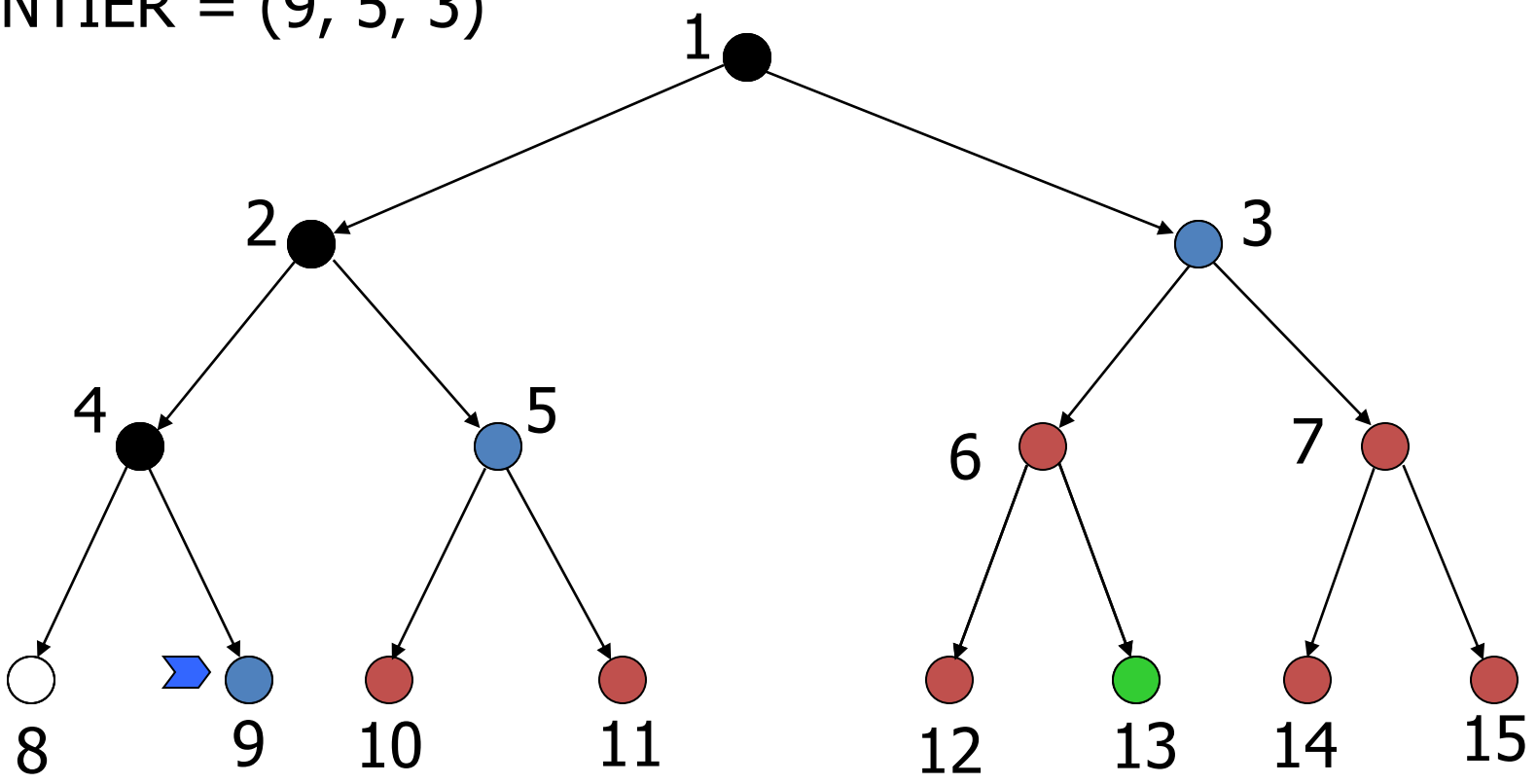


# Depth-First Strategy

- Remove Node 8 from frontier
- Do Goal-Test
- Node 8 has no successors and is removed from memory

Future = red nodes
Goal = green nodes
Frontier = blue nodes
Expanded = black nodes
Forgotten = white nodes

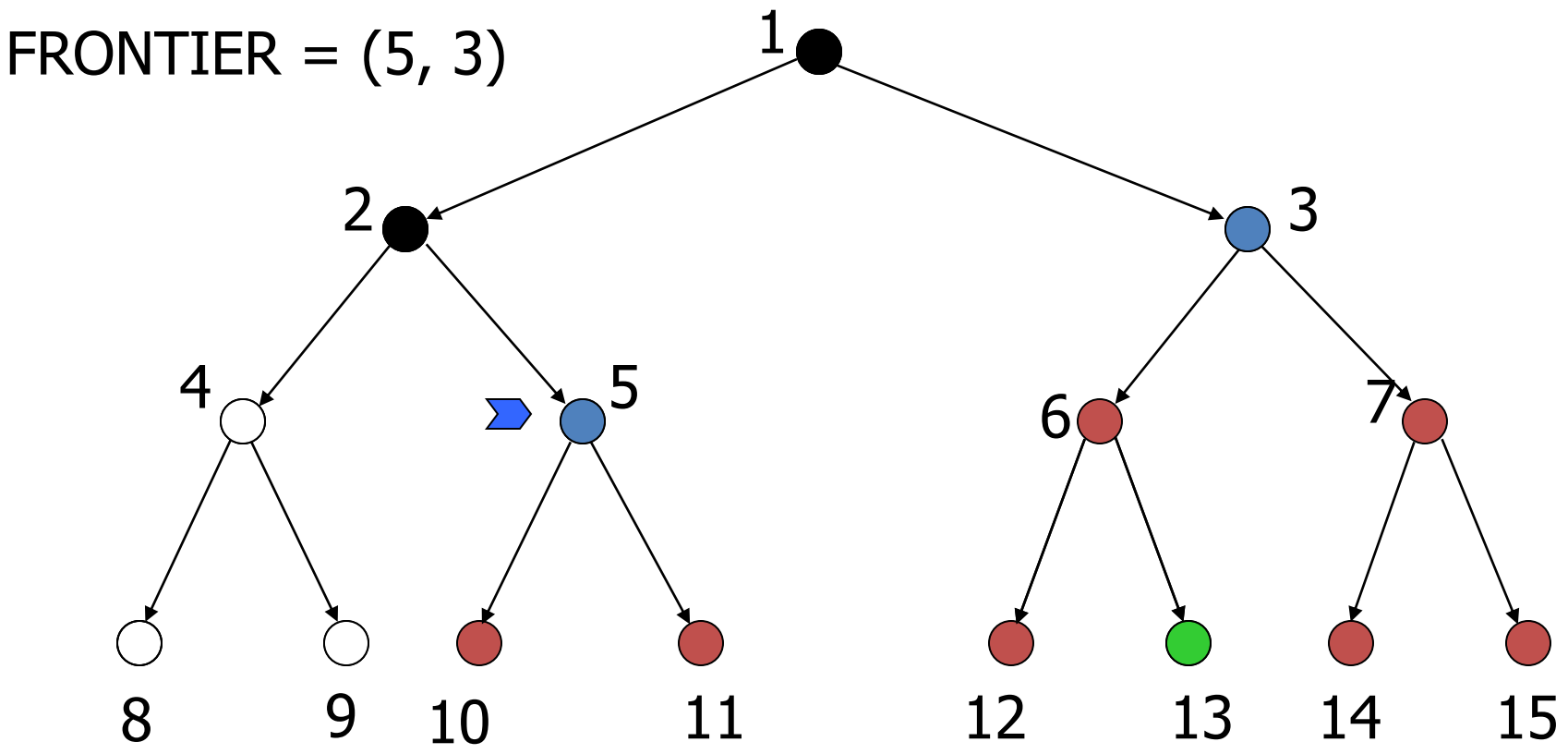
FRONTIER = (9, 5, 3)



# Depth-First Strategy

- Remove Node 9 from frontier
- Do Goal-Test
- Node 9 has no successors and is removed from memory
- Node 4 is removed from memory

Future = red nodes
Goal = green nodes
Frontier = blue nodes
Expanded = black nodes
Forgotten = white nodes

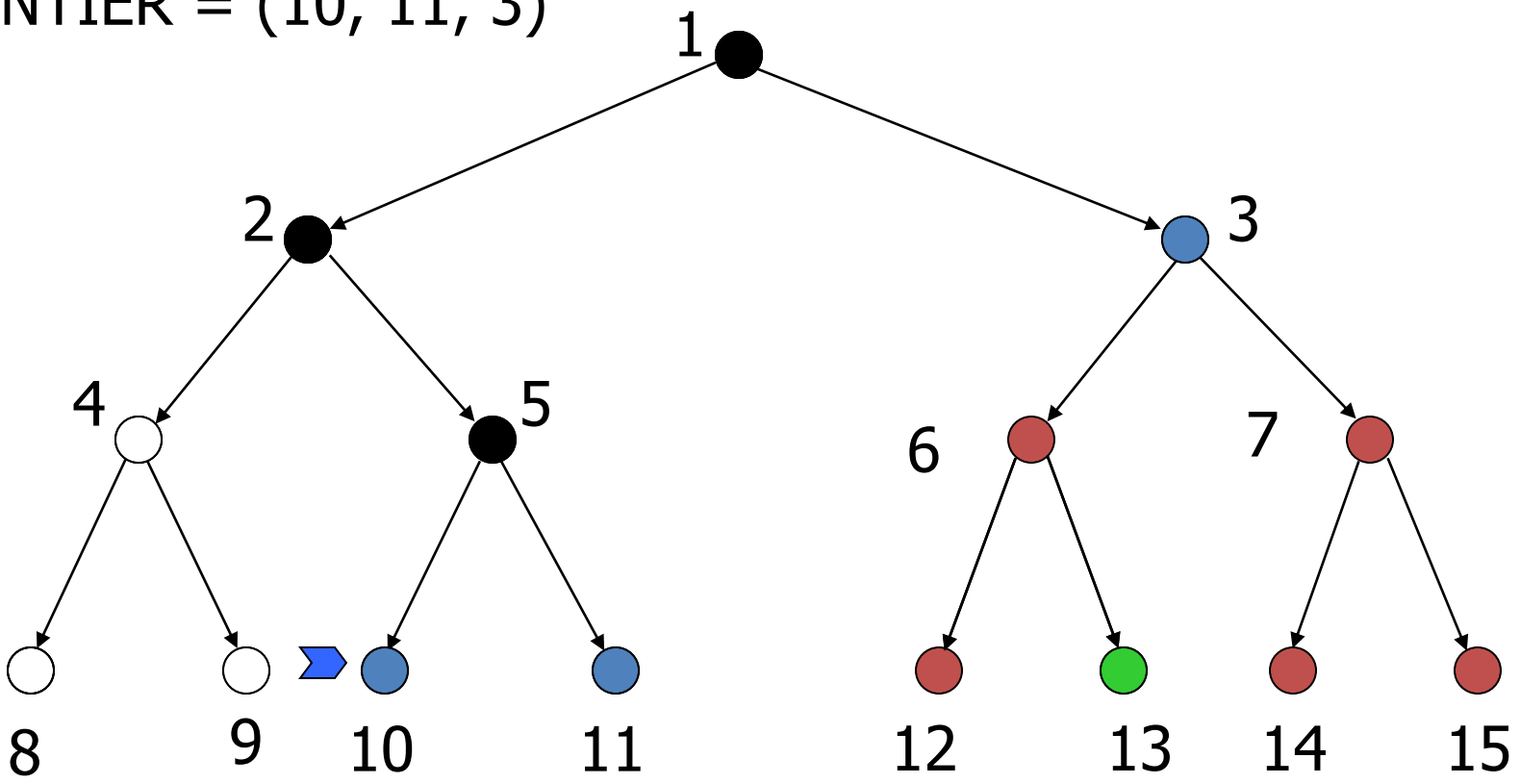


# Depth-First Strategy

- Remove Node 5 from frontier
- Do Goal-Test
- Generate Node 10, Node 11 and push them into FRONTIER

Future = red nodes
Goal = green nodes
Frontier = blue nodes
Expanded = black nodes
Forgotten = white nodes

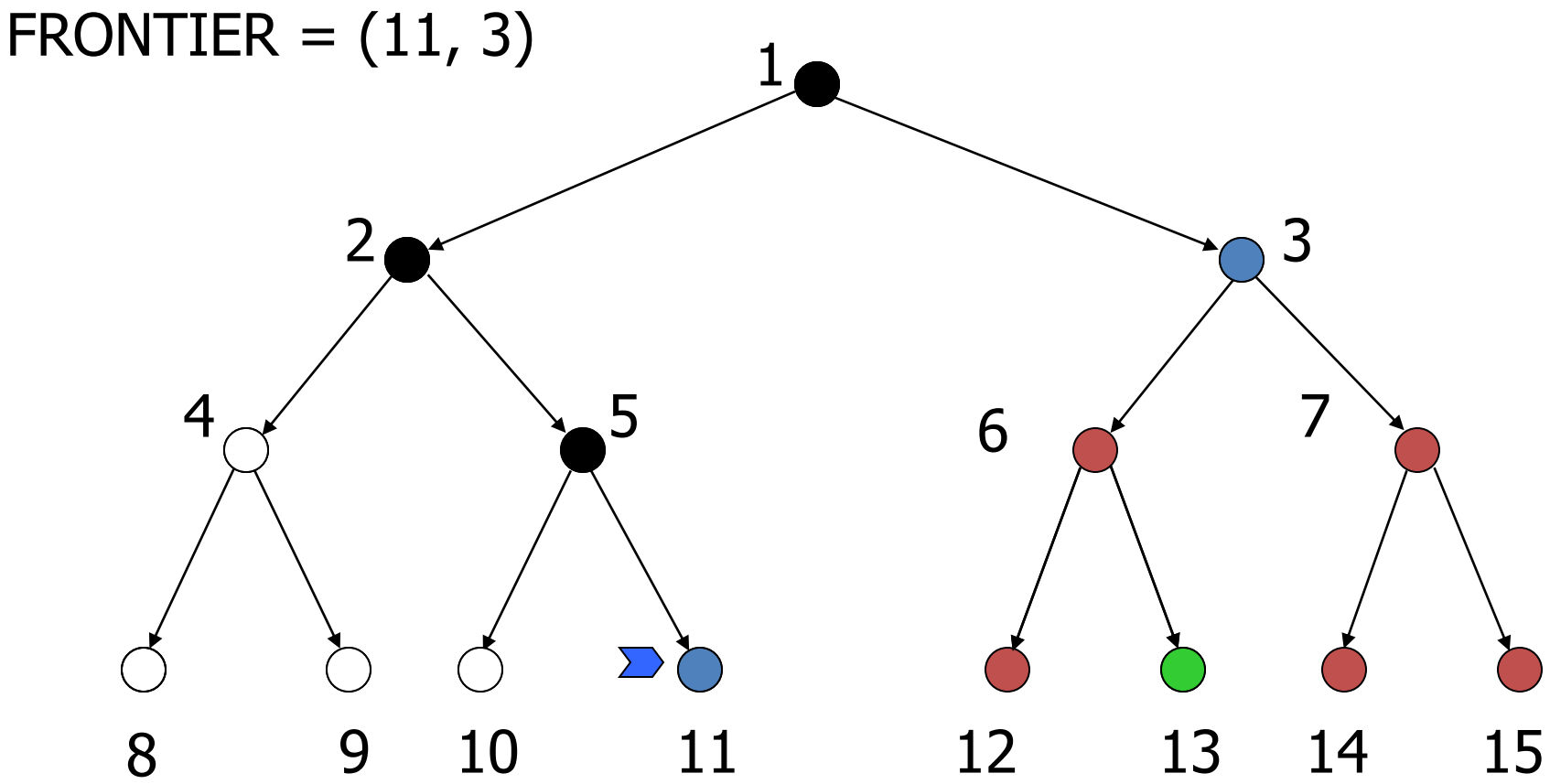
FRONTIER = (10, 11, 3)



# Depth-First Strategy

- Remove Node 10 from frontier
- Do Goal-Test
- Node 10 has no successors and is removed from memory

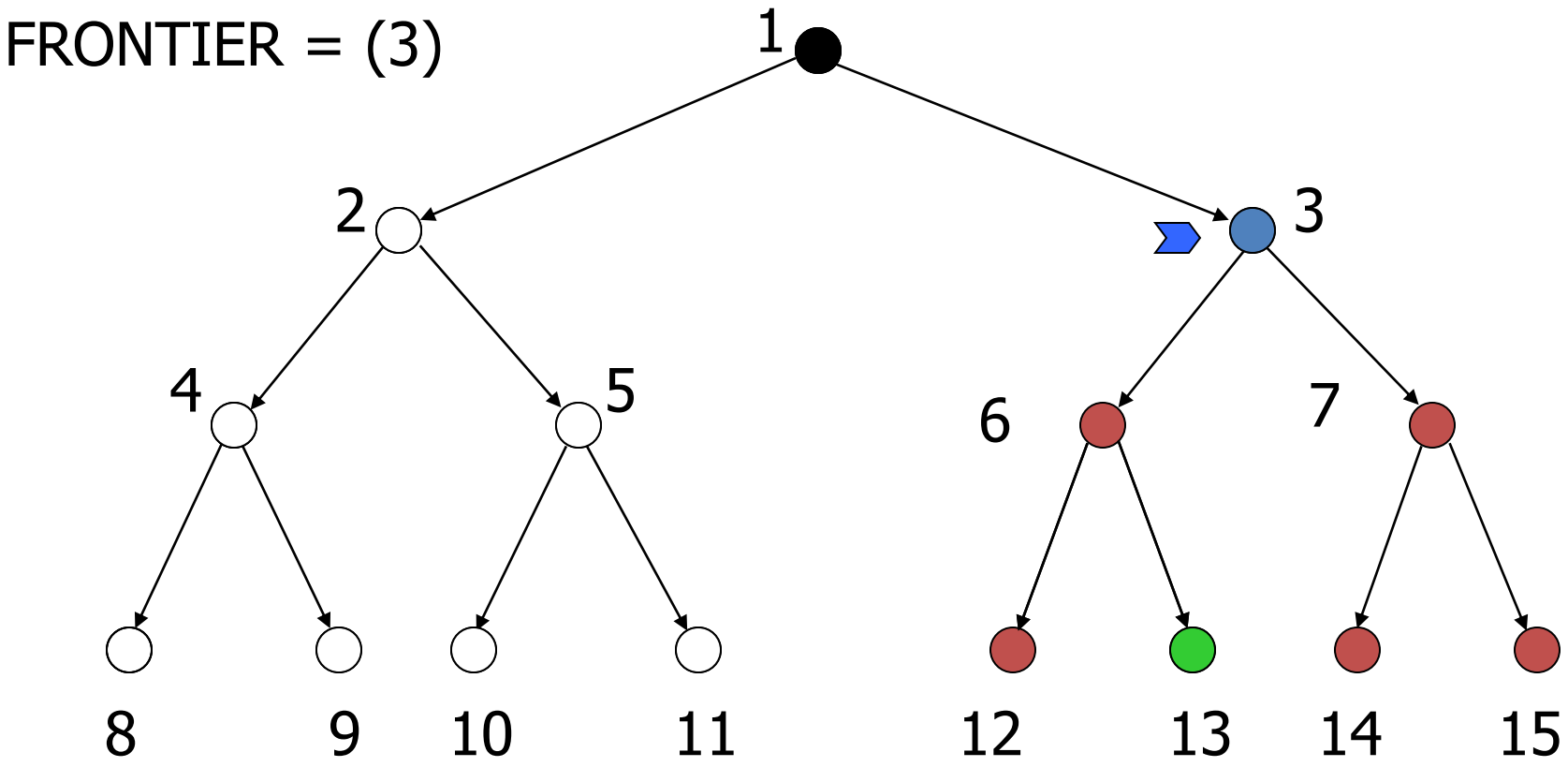
Future = red nodes
Goal = green nodes
Frontier = blue nodes
Expanded = black nodes
Forgotten = white nodes



# Depth-First Strategy

- Remove Node 11 from frontier
- Do Goal-Test
- Node 11 has no successors and is removed from memory
- Node 5 and Node 2 are removed

Future = red nodes
Goal = green nodes
Frontier = blue nodes
Expanded = black nodes
Forgotten = white nodes

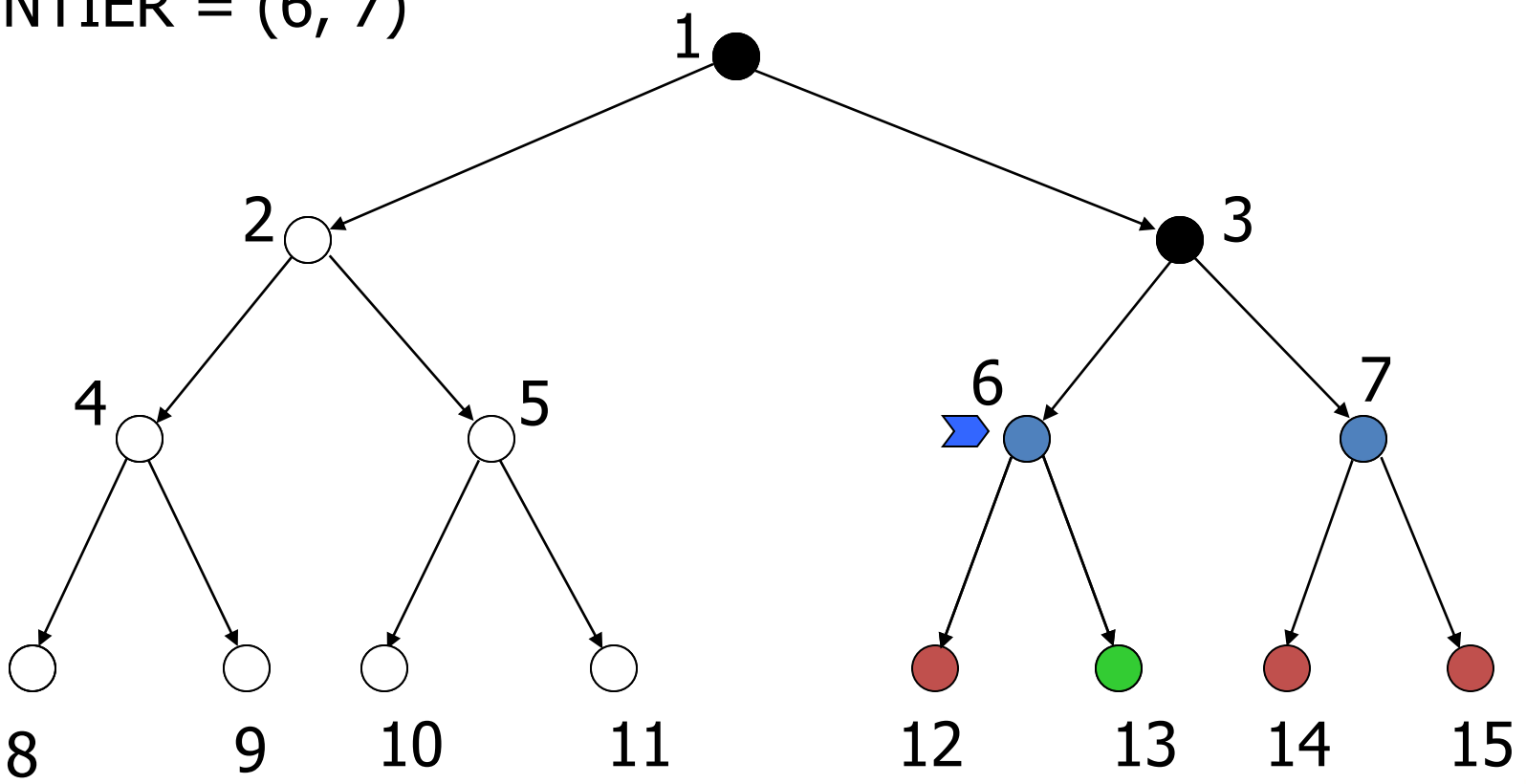


# Depth-First Strategy

- Remove Node 3 from frontier
- Do Goal-Test
- Generate Node 6, Node 7 and push them into FRONTIER

Future = red nodes
Goal = green nodes
Frontier = blue nodes
Expanded = black nodes
Forgotten = white nodes

FRONTIER = (6, 7)

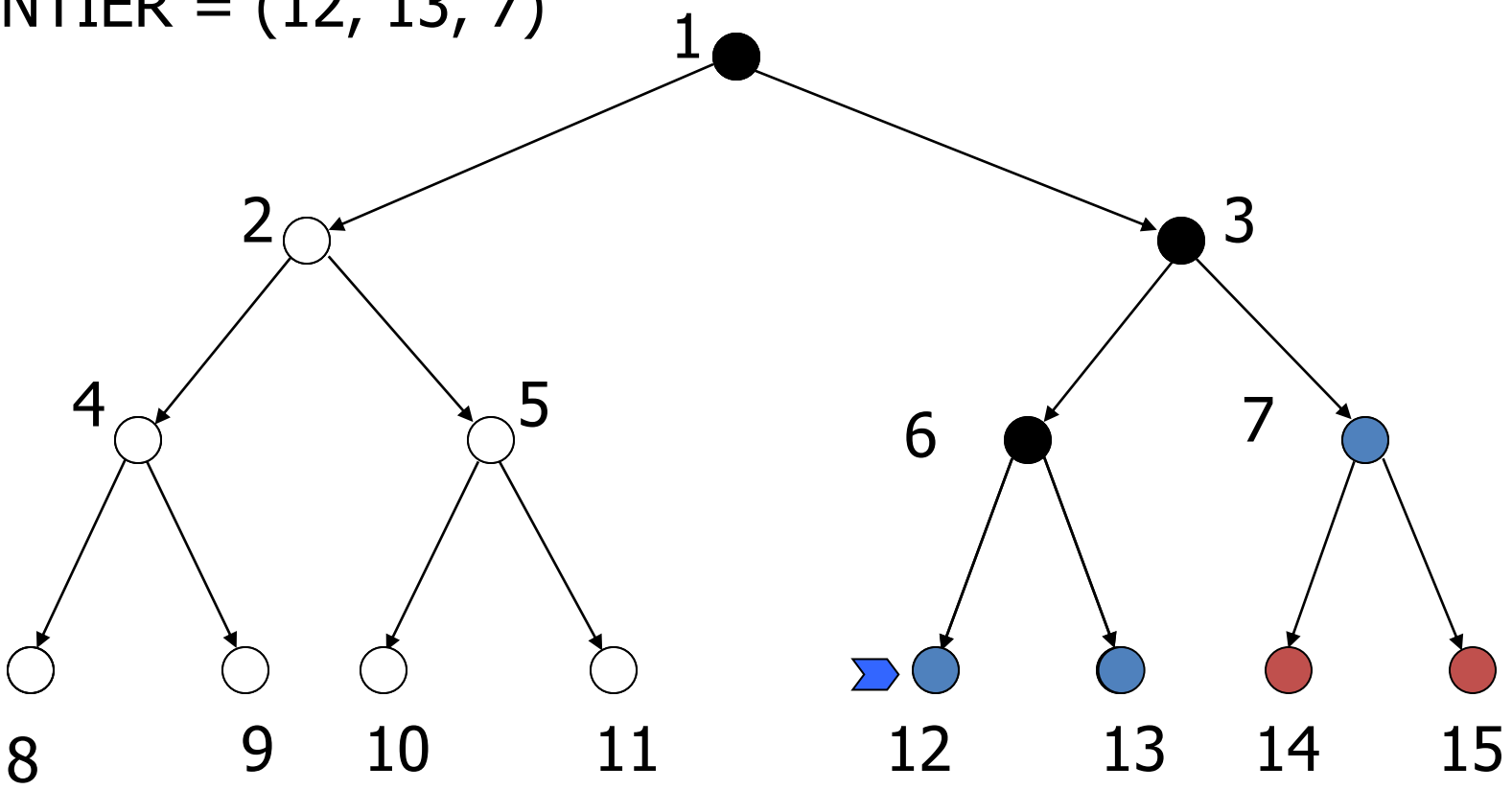


# Depth-First Strategy

- Remove Node 6 from frontier
- Do Goal-Test
- Generate Node 12, Node 13 and push them into FRONTIER

Future = red nodes
Goal = green nodes
Frontier = blue nodes
Expanded = black nodes
Forgotten = white nodes

FRONTIER = (12, 13, 7)

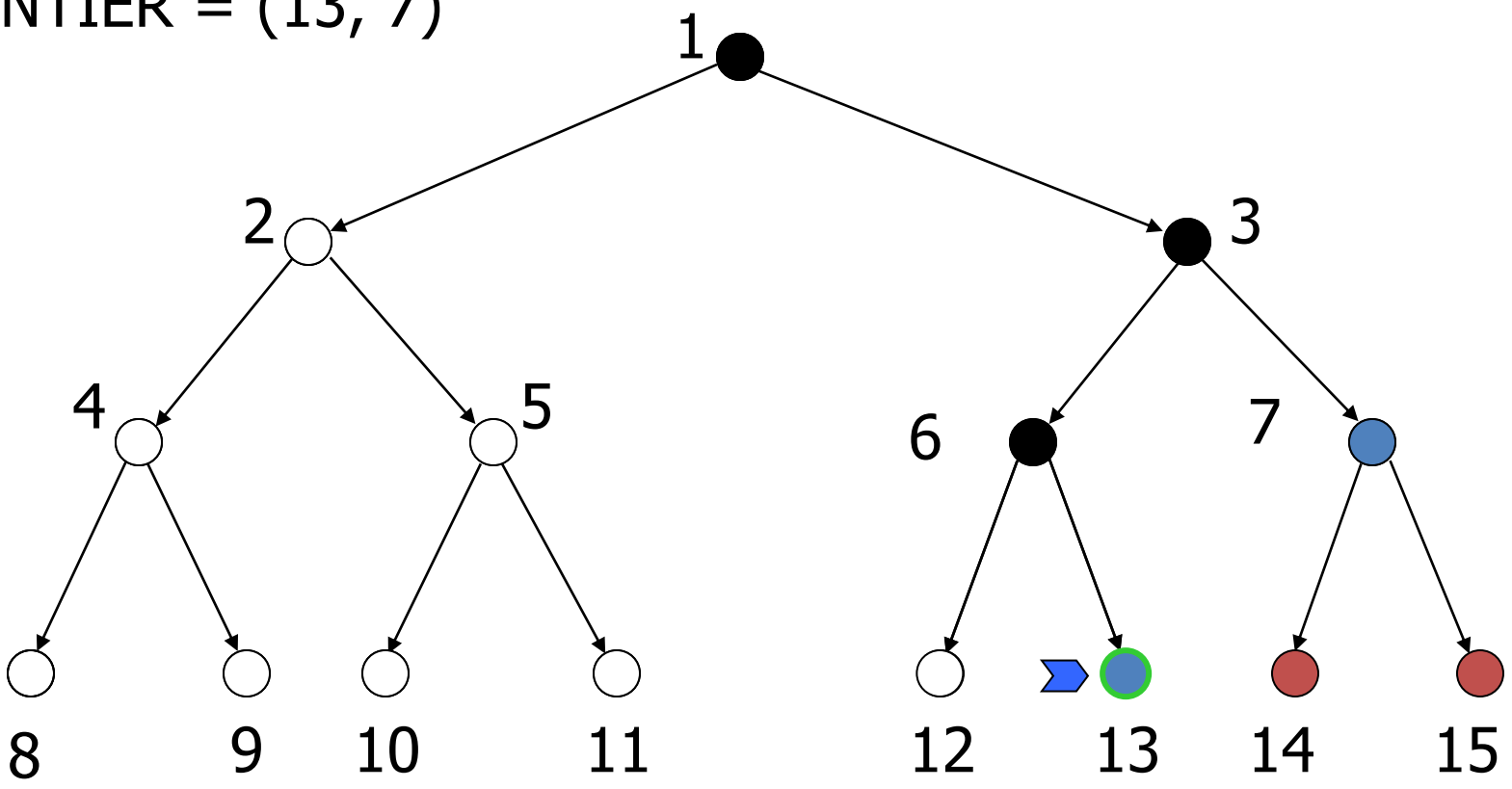


# Depth-First Strategy

- Remove Node 12 from frontier
- Do Goal-Test
- Node 12 has no successors and is removed

Future = red nodes
Goal = green nodes
Frontier = blue nodes
Expanded = black nodes
Forgotten = white nodes

FRONTIER = (13, 7)

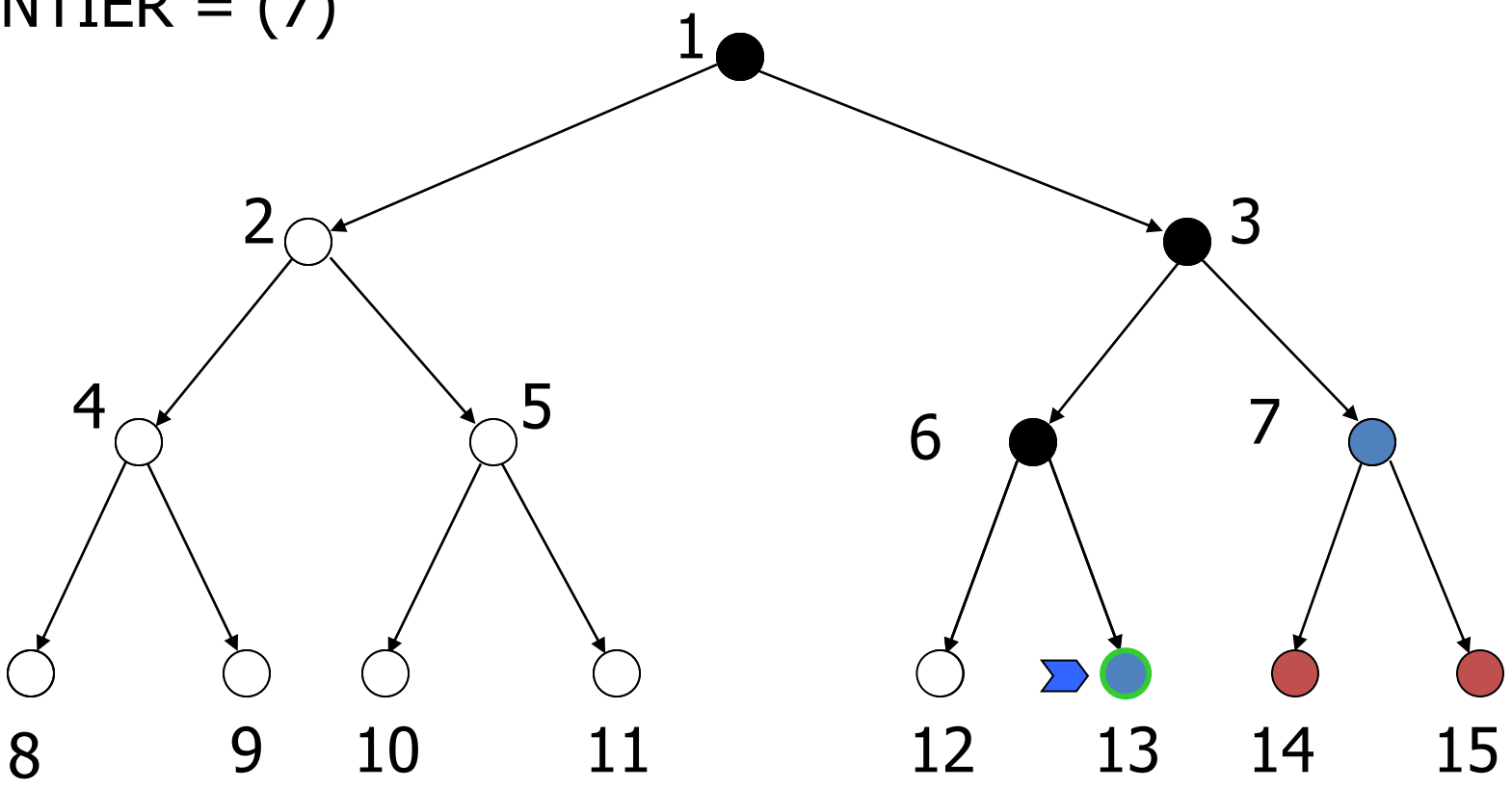


# Depth-First Strategy

- Remove Node 13 from frontier
- Do Goal-Test return Node 13

Future = red nodes
Goal = green nodes
Frontier = blue nodes
Expanded = black nodes
Forgotten = white nodes

FRONTIER = (7)



# Evaluation of Depth-First Search

- **Complete?**
  - **No: it fails infinite-depth spaces**
  - **Yes: for finite search tree**
- **Optimal? No: it may find a non-optimal goal first**
- **Number of nodes generated (Worst Case):**
  **$1 + b + b^2 + \ldots + b^m = O(b^m)$**
- **Time complexity is $O(b^m)$**
  - **Terrible if m is much larger than d**
  - **If solutions are located deep, may be much faster than BFS**
- **Space complexity**
  - **O(bm) linear space for tree search**
    - **remember a single path without explored set + remaining siblings unexpanded nodes**
  - **$O(b^m)$ for graph search**

**b: branching factor**
**d: depth of shallowest goal node**
**m: maximal depth of a leaf node**

# DFS vs. BFS

- DFS goes off into one branch until it reaches a leaf node
  - not good if the goal is on another branch
  - neither complete nor optimal
  - uses much less space than BFS for tree search
    - much fewer visited nodes to keep track of
    - smaller frontier

- BFS is more careful by checking all alternatives
  - complete and optimal
  - very memory-intensive

# Depth-Limited Strategy

- To avoid the infinite depth problem of DFS, we can decide to only search until depth k, i.e. we don't expand nodes beyond depth k.

  => Depth-limited Search

- Depth-first with depth cutoff k
  - Only search until depth k
  - Do not expand nodes beyond depth k

- Two possible outcomes:
  - Solution
  - Cutoff (no solution within cutoff) because of a bad k

# Iterative Deepening Strategy

- What if solution is deeper than k?
  - Increase k iteratively

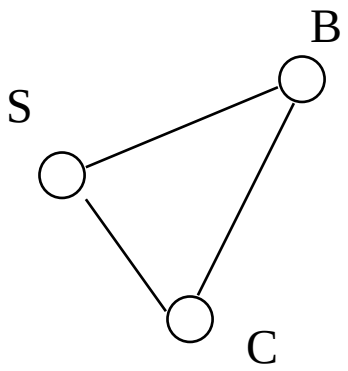
    => Iterative Deepening Search

Repeat for k = 0, 1, 2, …:

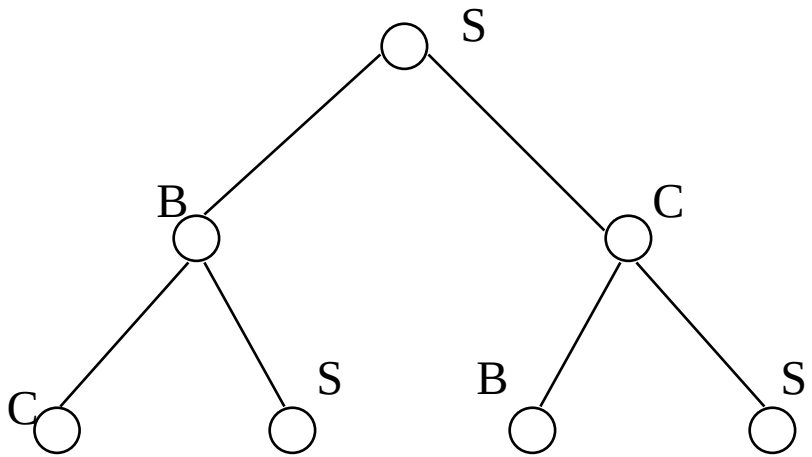
Perform DFS with depth cutoff k

- Complete?
  - Yes, for the same reason BFS is complete
- Optimal?
  - No, for general cost functions
  - Yes, if cost is a non-decreasing function only of depth
- Time complexity? Seems wasteful but most of the nodes are at the bottom level; total
  - $(d+1)(1) + db + (d-1)b^2 + \ldots + (1)\, b^d = O(b^d)$
- Space complexity?
  - O(bd)

# Repeated States




State Space




Example of a Search Tree

- **Problems of Repeated States**
  - **Waste time by expanding states that have already been encountered and expanded before**
  - **Infinite loop**

# Avoid Repeated States in DFS

- **Problems in Depth-first Search**
  - **May lead to infinite loop**
    - **Cannot find a solution**

- **Depth-first Search**
  - **Keep track of the path from the root to the current node**
  - **Compare the current node with the nodes in the path**
    - **Discard it if it is already in the path**
  - **Prevent infinite loop**

# Eliminate Repeated States

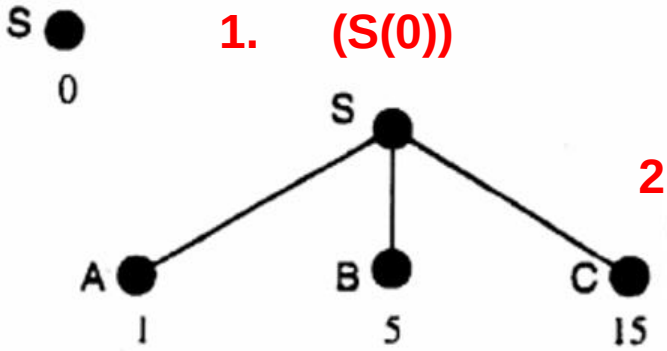
- Closed List (Explored)
  - Store every expanded nodes
- Open List (Frontier)
  - Store unexpanded nodes
- Graph-Search
  - Match current node with nodes in Explored List
  - Discard if it is in closed list
- Tree-Search
  - Match current node with nodes in the path
  - Discard if it is in the path
- Discussion
  - Time/Space Trade-off

# Uniform-Cost Search

- Strategy: expand unexpanded node with lowest path cost g(n)
- Frontier:
  - a priority queue
  - new successors are merged into the queue sorted by path cost
  - when a new node with a lower path cost for a state that is already in the frontier is encountered, the standard practice is to replace the old node with the new node to the frontier.
  - when a new node with a lower path cost for a state that is already in the explored set is encountered, the standard practice is to add the new node to the frontier for further exploration. After expansion, it will eventually replace the old node in the explored set.

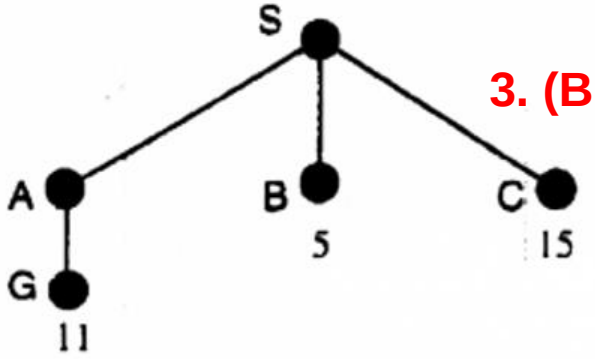
- Goal-Test applied when node is expanded with variable step costs

# Uniform-Cost Search

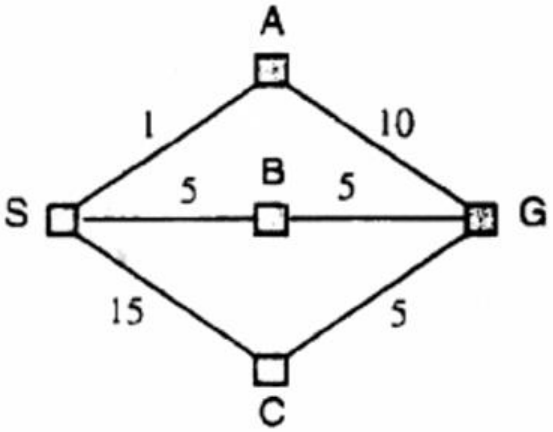



1. (S(0))

2. (A(1), B(5), C(15))



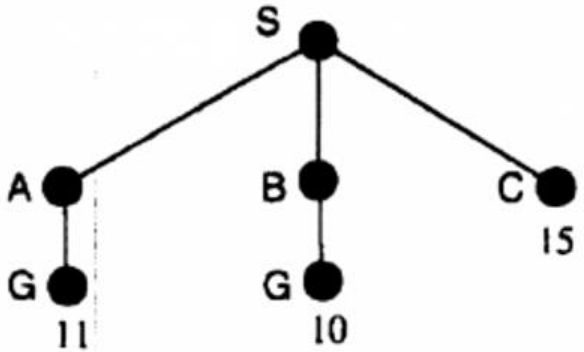

3. (B(5), G(11), C(15))




(a)

4. (G(10), C(15))

*5. Goal reached!*




(b)

# Summary

- Breadth-first search
  - Analysis
- Depth-first search
  - Analysis
- Uniform-cost search
- Depth-limited search (Optional)
- Iterative deepening search (Optional)

# What I want you to do

- Review Chapter 3
- Work on your assignment 1