Blind (Uninformed) **Search**

Russell and Norvig:
Chap. 3, Sect. 3.3 – 3.4

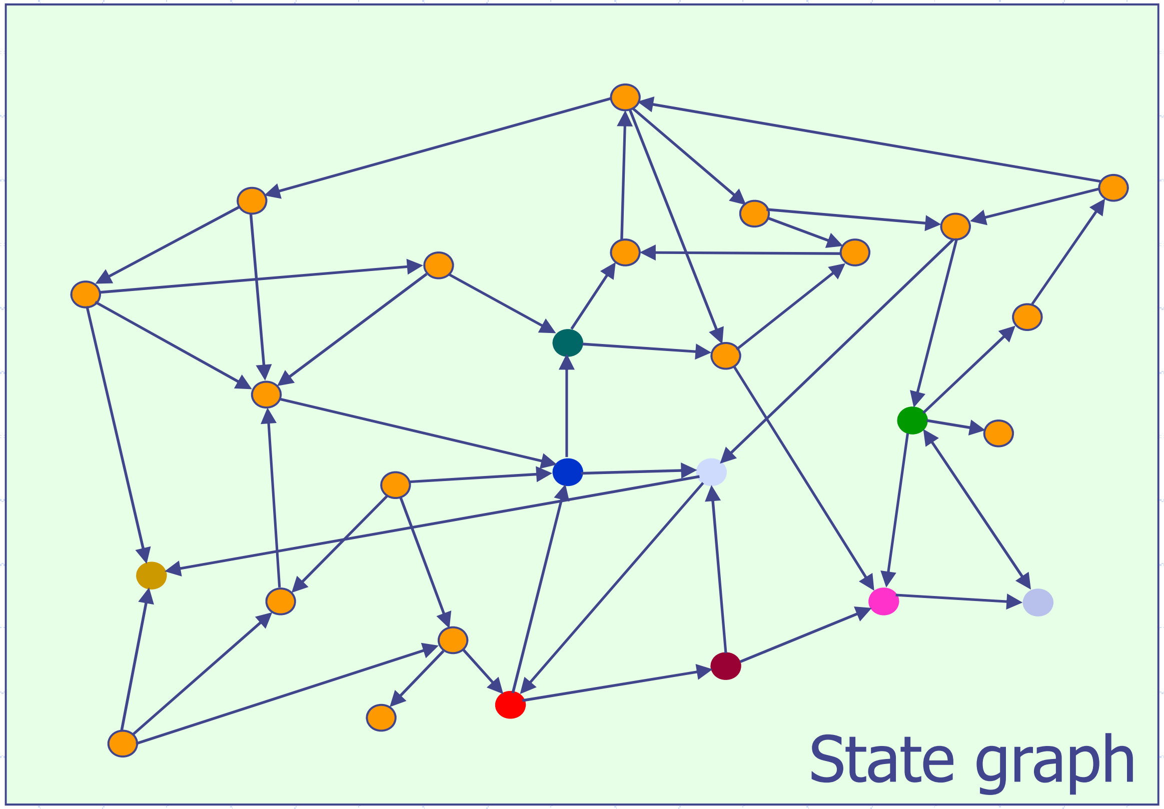
Slides from Jean-Claude Latombe at Stanford University
(used with permission)

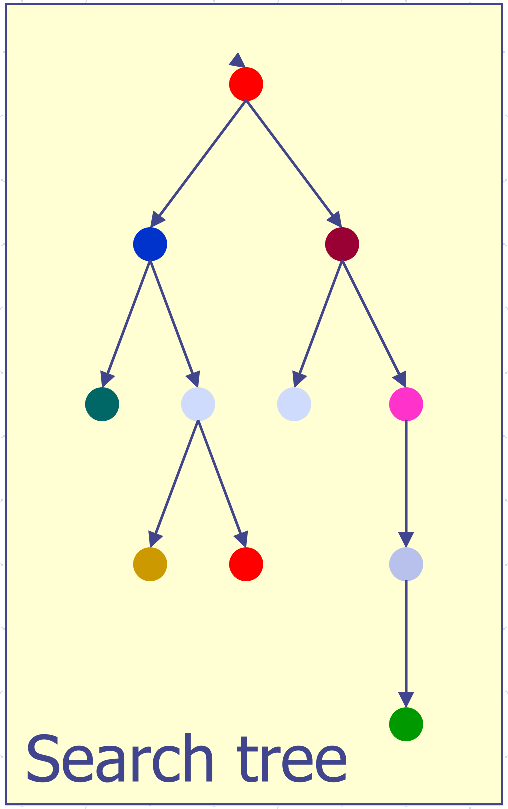
Simple Agent Algorithm

Problem-Solving-Agent

1. formulate: (abstraction!)
    1. S_0 = initial-state ← sense/read state
    2. GOAL? = goal test ← select/read goal test
    3. actions ← select/read action models
    4. transition model ← select/read model
    5. problem ← (S_0, GOAL?, actions, transition model)
2. solution ← search(problem)
3. perform(solution)

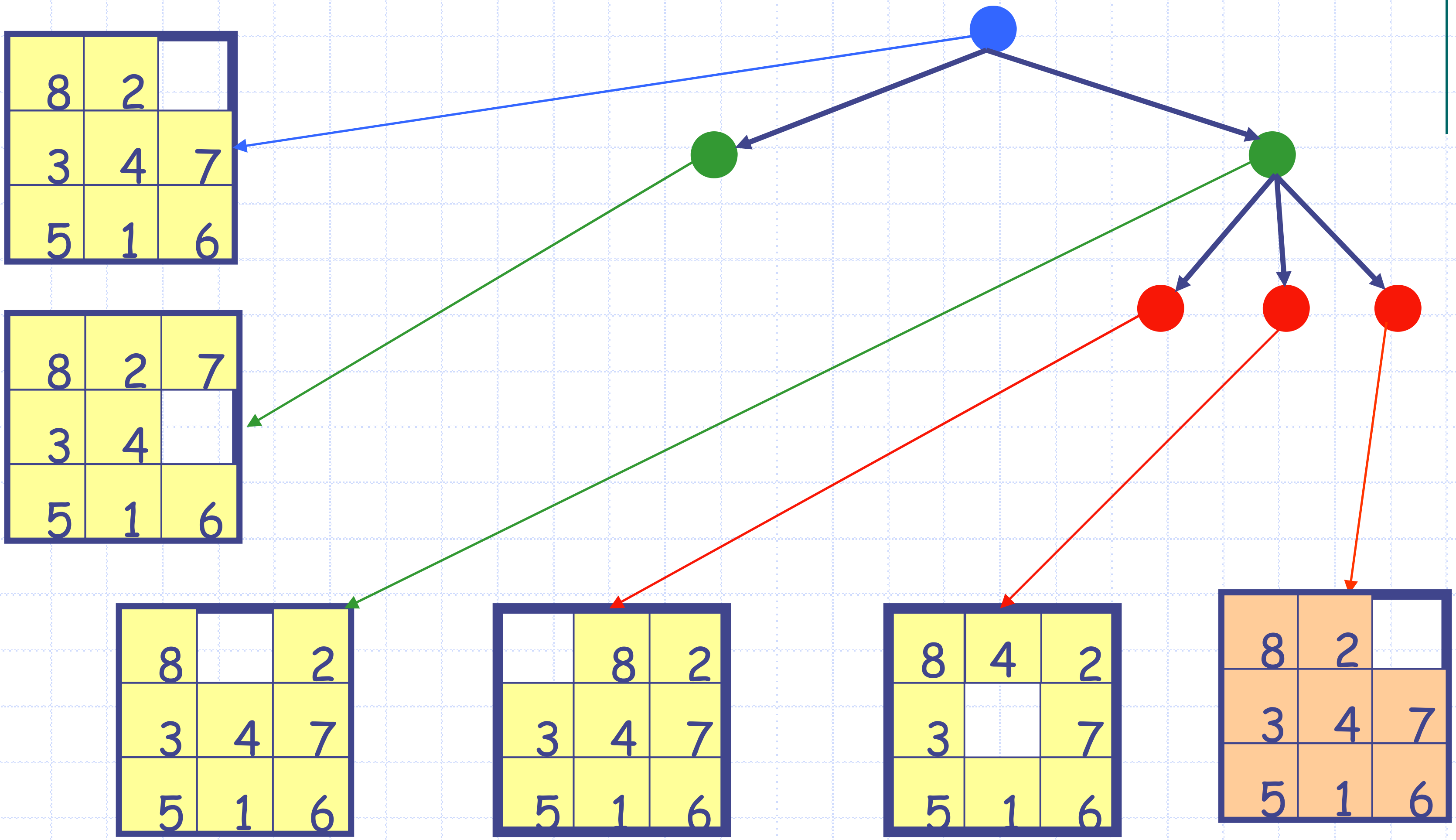
Search Tree



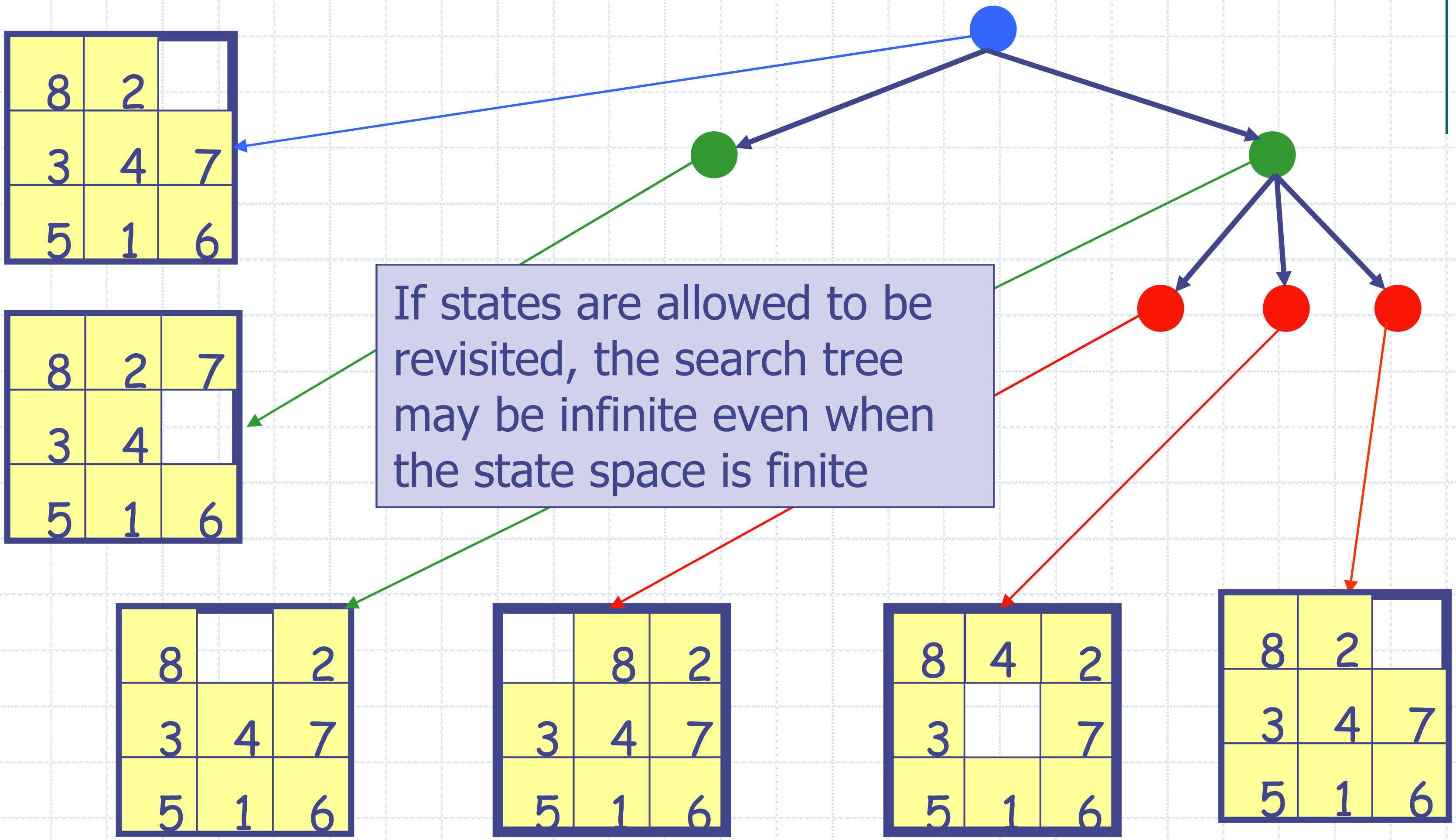


Note that some states may be visited multiple times

Search Nodes and States

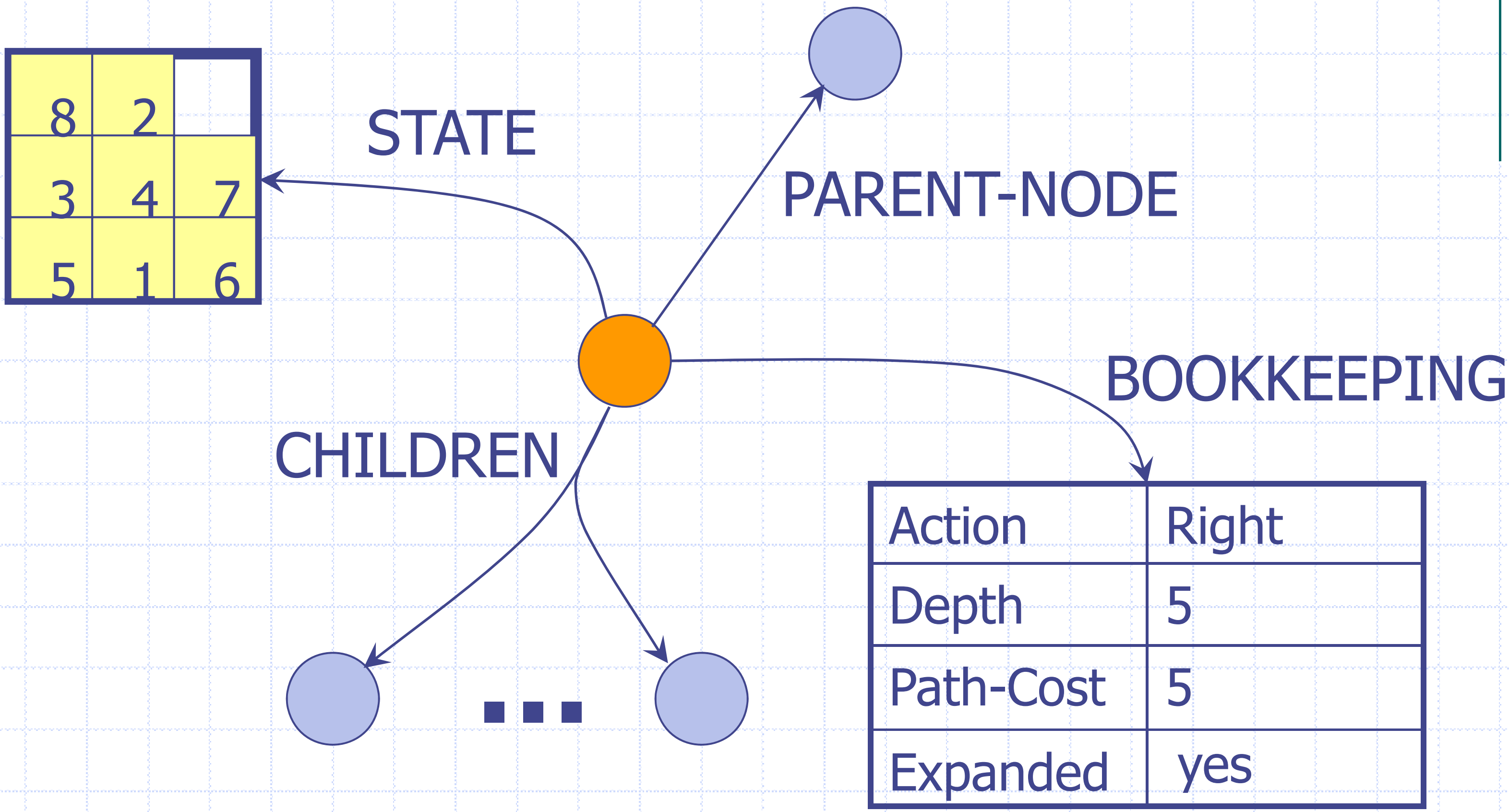


Search Nodes and States



If states are allowed to be revisited, the search tree may be infinite even when the state space is finite

Data Structure of a Node

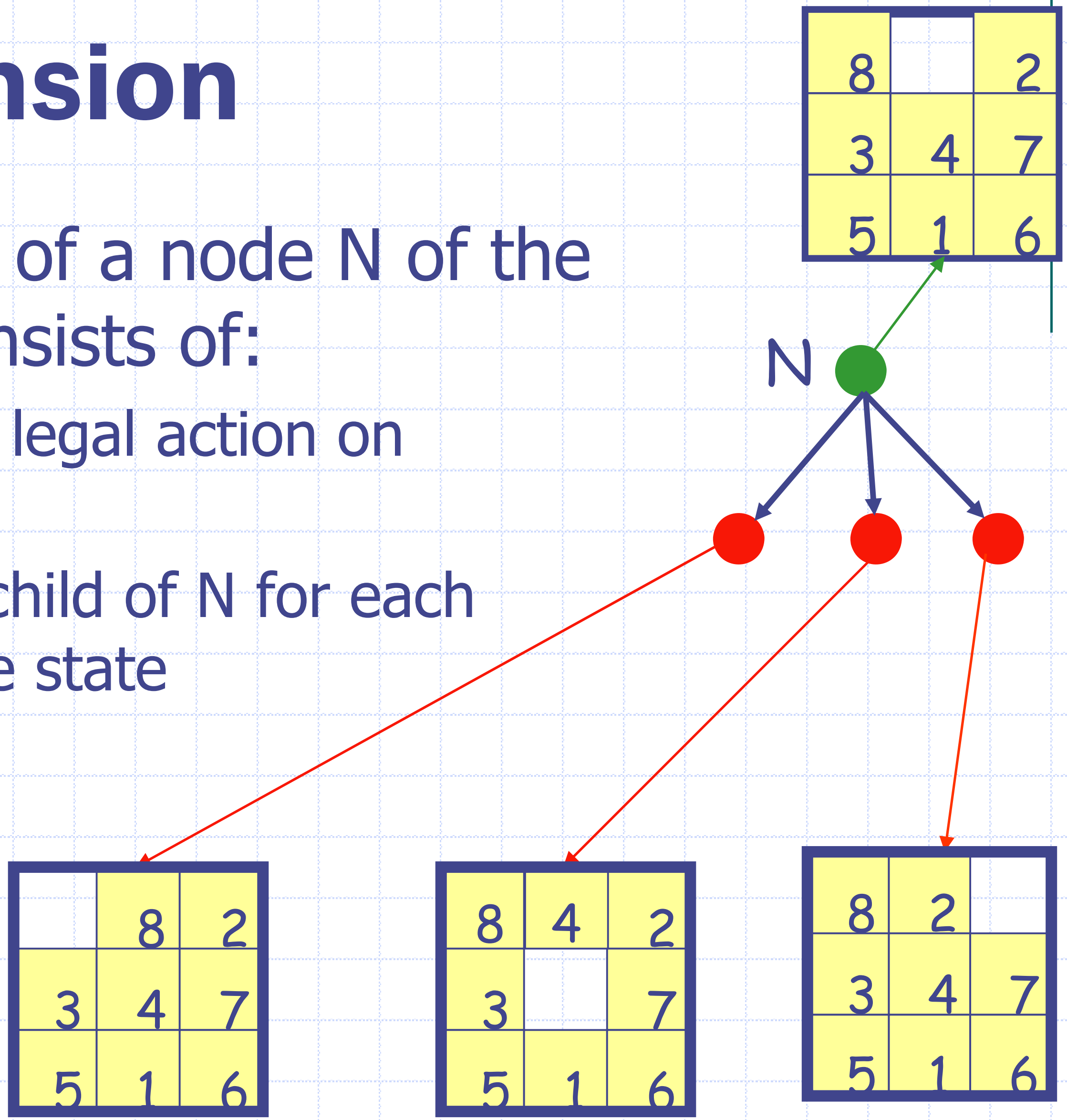


Action	Right
Depth	5
Path-Cost	5
Expanded	yes

Depth of a node N = length of path from root to N
(depth of the root = 0)

Node expansion

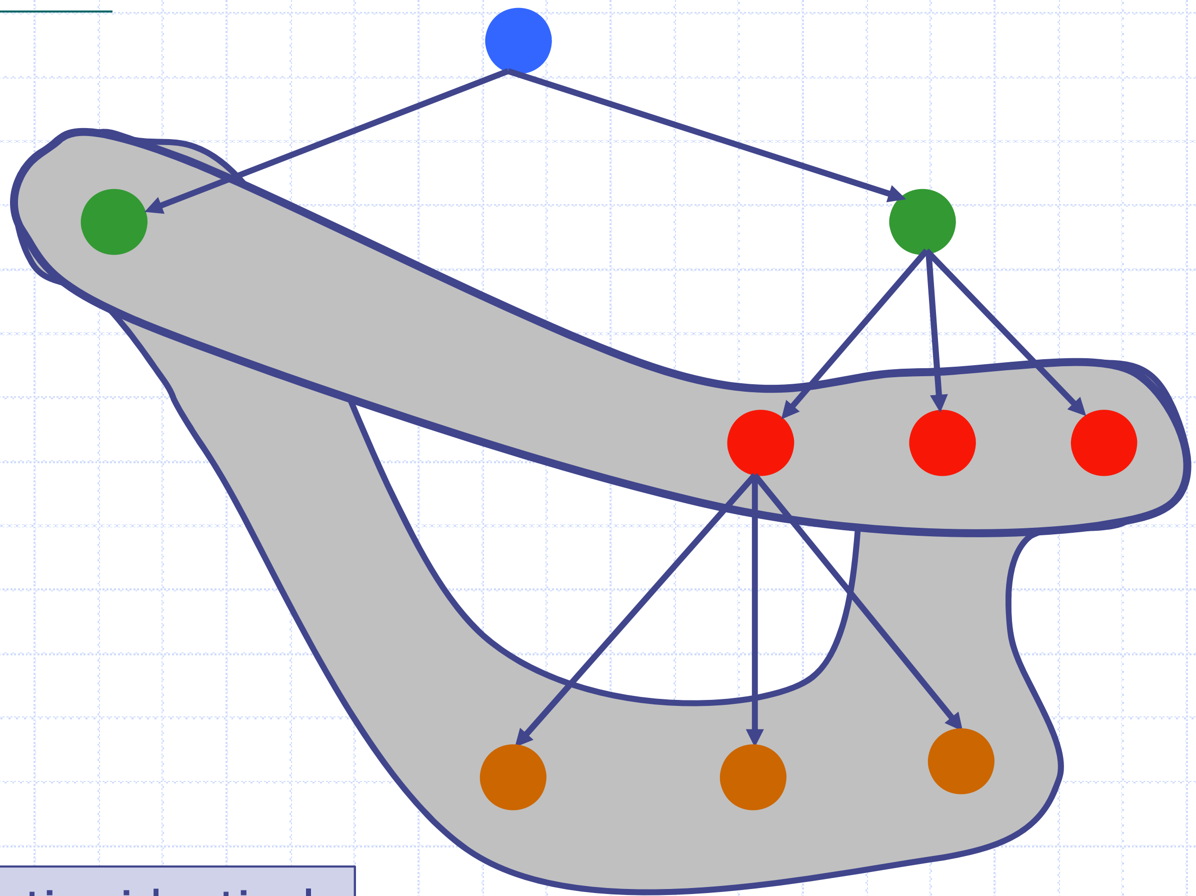
- The expansion of a node N of the search tree consists of:
  - Applying each legal action on STATE(N)
  - Generating a child of N for each new accessible state



node generation
≠
node expansion!

Frontier of Search Tree

- The **frontier** is the set of all search nodes that haven’t been expanded yet



Is the frontier identical to the set of leaves?

Search Strategy

- The frontier is the set of all search nodes that haven't been expanded yet
- The frontier is implemented as a priority queue FRONTIER
  - INSERT(node, FRONTIER)
  - POP(FRONTIER)
- The ordering of the nodes in FRONTIER defines the search strategy

Search Algorithm #1

SUCCESSORS(s) returns set of states reachable by single legal actions from s

1. If GOAL?(S_0) then return S_0
2. INSERT(N_0,FRONTIER)
3. Repeat:
   a. If EMPTY?(FRONTIER) then return failure
   b. N = POP(FRONTIER)
   c. s = STATE(N)
   d. For every state s' in SUCCESSORS(s)
      i. Create a new node N' as a child of N
      ii. If GOAL?(s') then return path or goal state
      iii. INSERT(N',FRONTIER)

Expansion of N

Performance Measures

- Completeness
  A search algorithm is complete if it finds a solution whenever one exists
- Optimality
  A search algorithm is optimal if it returns a minimum-cost path whenever a solution exists
- Complexity
  It measures the time and amount of memory required by the algorithm

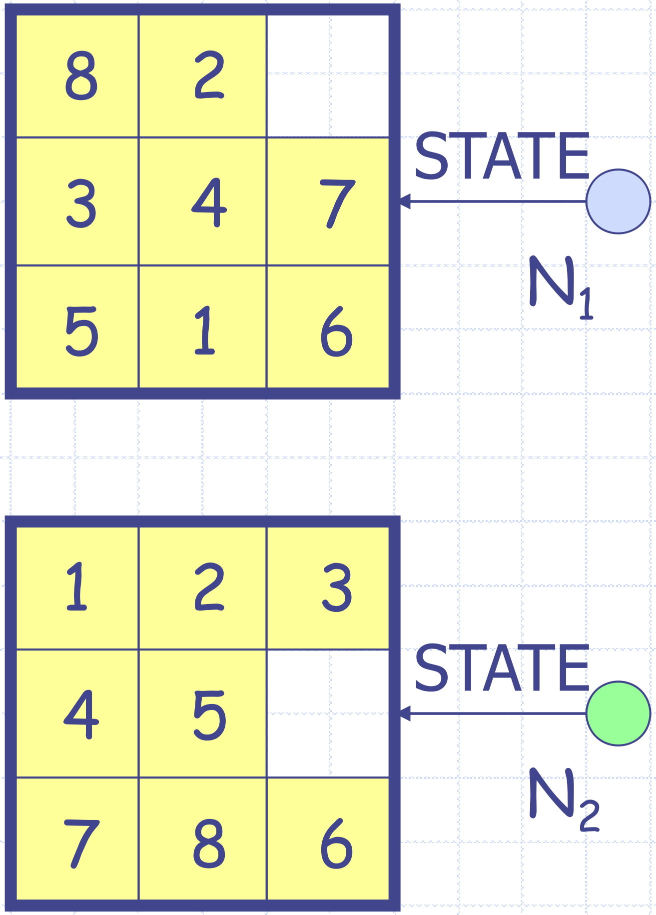
Blind vs. Heuristic Strategies

- Blind (or un-informed) strategies do not exploit state descriptions to order FRONTIER. They only exploit the positions of the nodes in the search tree

- Heuristic (or informed) strategies exploit state descriptions to order FRONTIER (the most "promising" nodes are placed at the beginning of FRONTIER)

Example

For a blind strategy, N_1 and N_2 are just two nodes (at some position in the search tree)

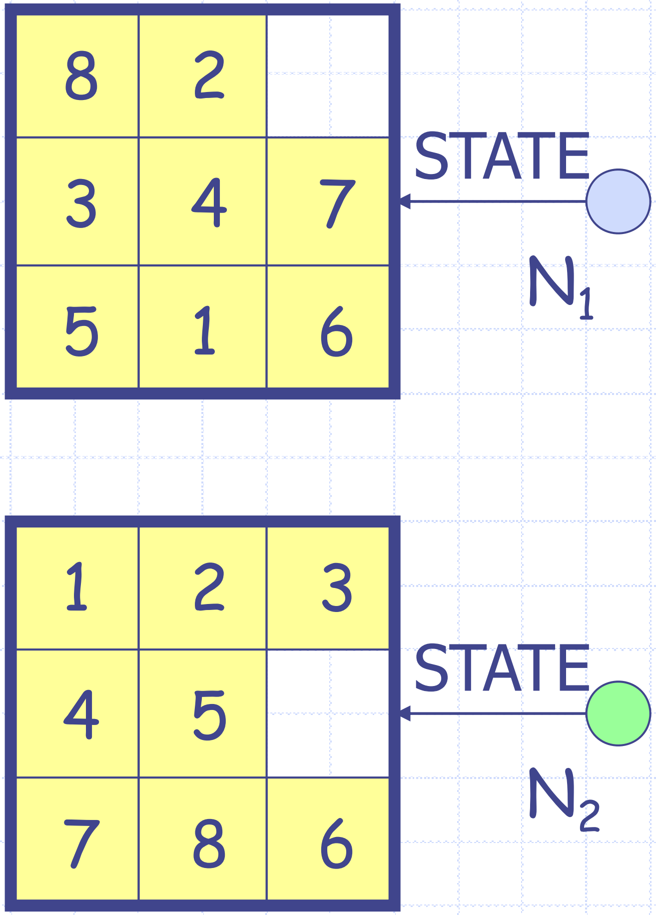


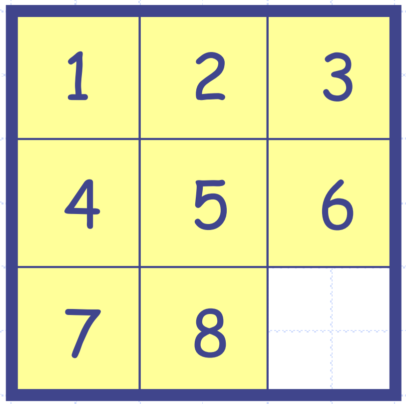
1	2	3
4	5	6
7	8	

Goal state

Example

For a heuristic strategy counting the number of misplaced tiles, N_2 is more promising than N_1





Goal state

Remark

- Some search problems, such as the (n^2-1)-puzzle, are NP-hard
- One can't expect to solve all instances of such problems in less than exponential time (in n)
- One may still strive to solve each instance as efficiently as possible

This is the purpose of the search strategy

Blind Strategies

- Breadth-first
  - Bidirectional
- Depth-first
  - Depth-limited
  - Iterative deepening

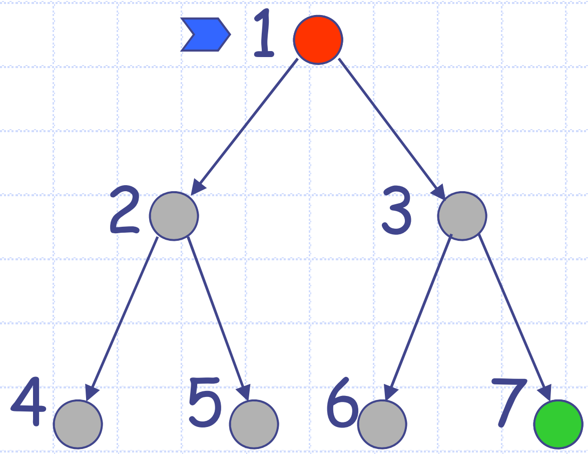
Arc cost = 1

- Uniform-Cost (variant of breadth-first)

Arc cost = c(action) $\geq \varepsilon > 0$

Breadth-First Strategy

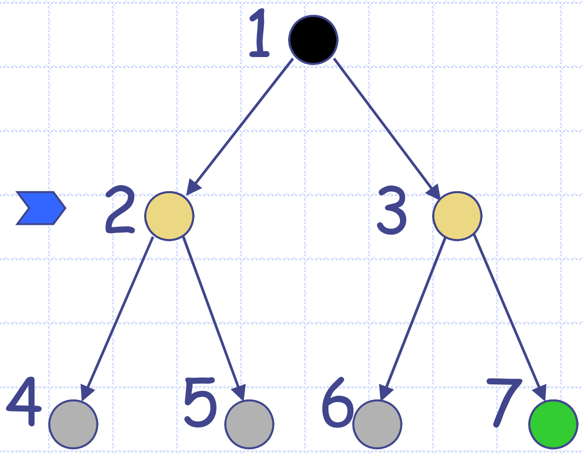
- New nodes are inserted at the end of FRONTIER



FRONTIER= (1)

Breadth-First Strategy

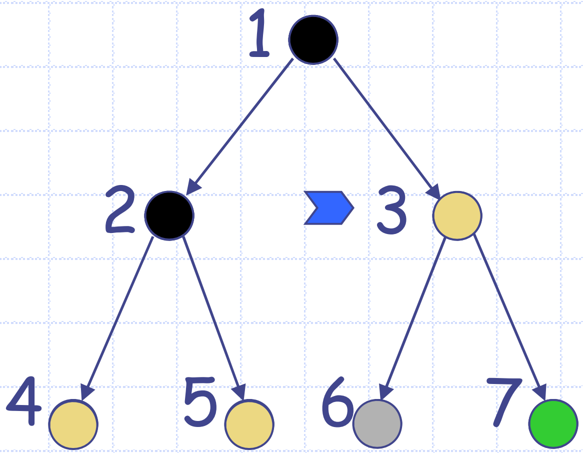
- New nodes are inserted at the end of FRONTIER



FRONTIER = (2, 3)

Breadth-First Strategy

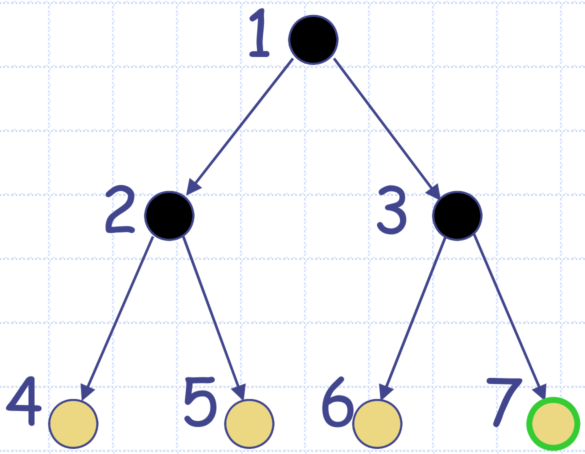
- New nodes are inserted at the end of FRONTIER



FRONTIER = (3, 4, 5)

Breadth-First Strategy

- New nodes are inserted at the end of FRONTIER



FRONTIER = (4, 5, 6, 7)

Important Parameters

- Maximum number of successors of any state

  → **branching factor b** of the search tree

- Minimal length (≠ cost) of a path between the initial and a goal state

  → **depth d** of the shallowest goal node in the search tree

BF Evaluation

- b: branching factor
- d: depth of shallowest goal node
- Breadth-first search is:
  - Complete? Not complete?
  - Optimal? Not optimal?

BF Evaluation

- b: branching factor
- d: depth of shallowest goal node
- Breadth-first search is:
  - Complete
  - Optimal if step cost is 1
- Number of nodes generated:

BF Evaluation

- b: branching factor
- d: depth of shallowest goal node
- Breadth-first search is:
  - Complete
  - Optimal if step cost is 1
- Number of nodes generated:
  $1 + b + b^2 + \ldots + b^d \quad = \quad O(b^d)$

$\rightarrow$ Time and space complexity is $O(b^d)$

Time and Memory Use

d	# Nodes	Time	Memory
2	110	.11 msec	107 kilobytes
4	11,110	11 msec	10.6 megabytes
6	$\sim 10^{6}$	1.1 sec	1 gigabyte
8	$\sim 10^{8}$	2 min	103 gigabytes
10	$\sim 10^{10}$	3 hours	10 terabytes
12	$\sim 10^{12}$	13 days	1 petabyte
14	$\sim 10^{14}$	3.5 years	99 petabytes

Assumptions: b = 10; 1 million nodes/sec; 1000 bytes/node

Remark

- If a problem has no solution, breadth-first may run for ever (if the state space is infinite or states can be revisited arbitrary many times)

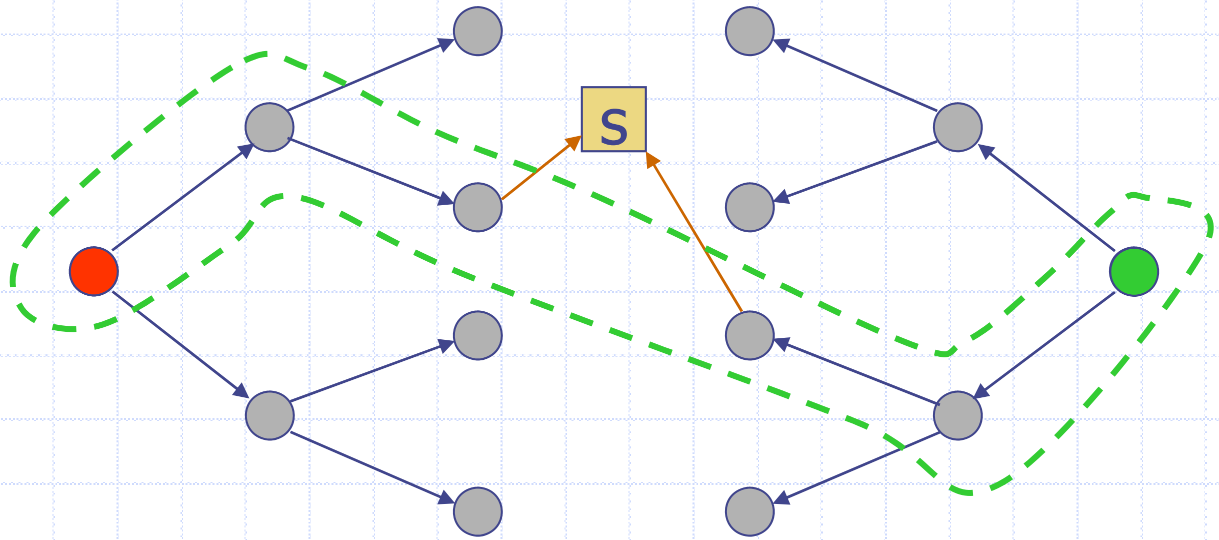
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

?

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

Bidirectional Strategy

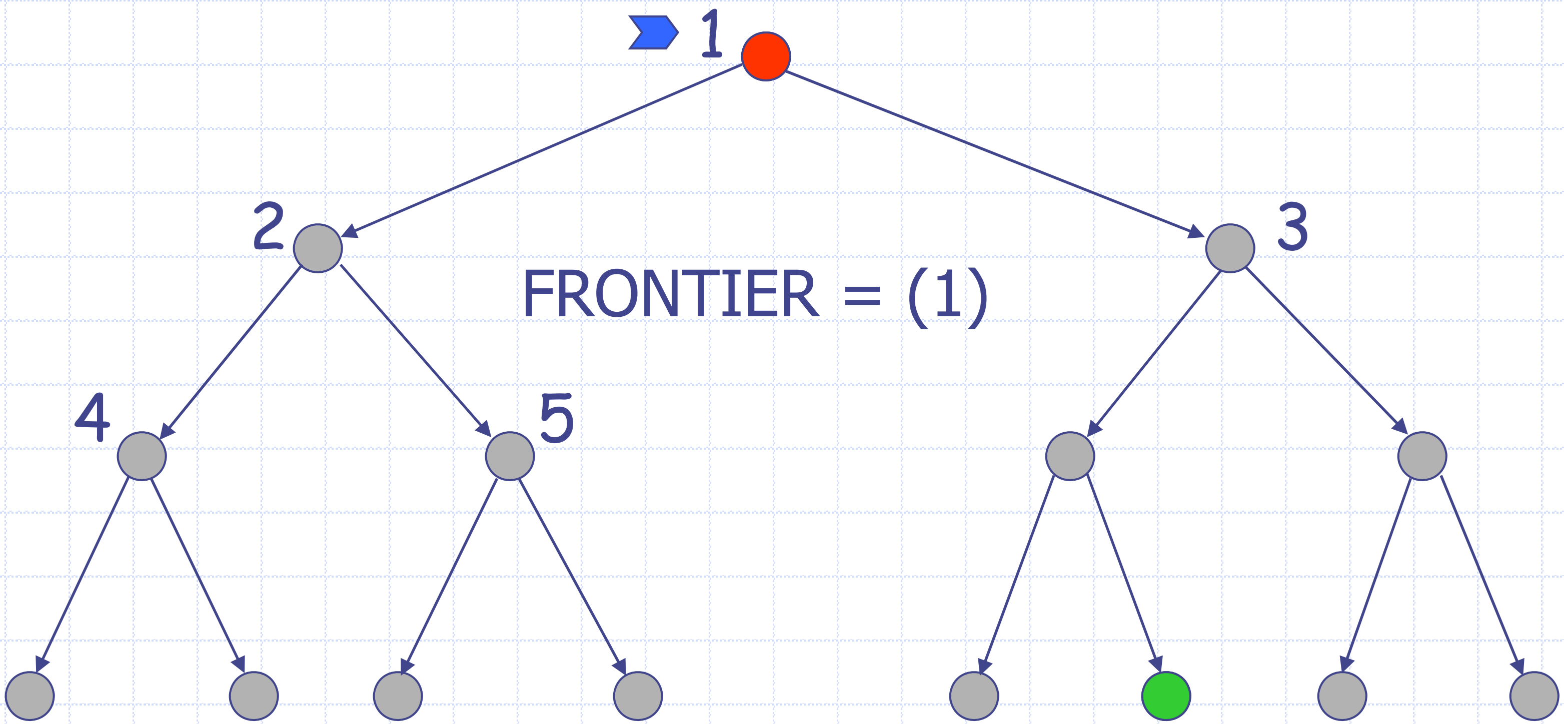
2 frontier queues: FRONTIER1 and FRONTIER2



Time and space complexity is $O(b^{d/2}) \ll O(b^d)$
if both trees have the same branching factor b

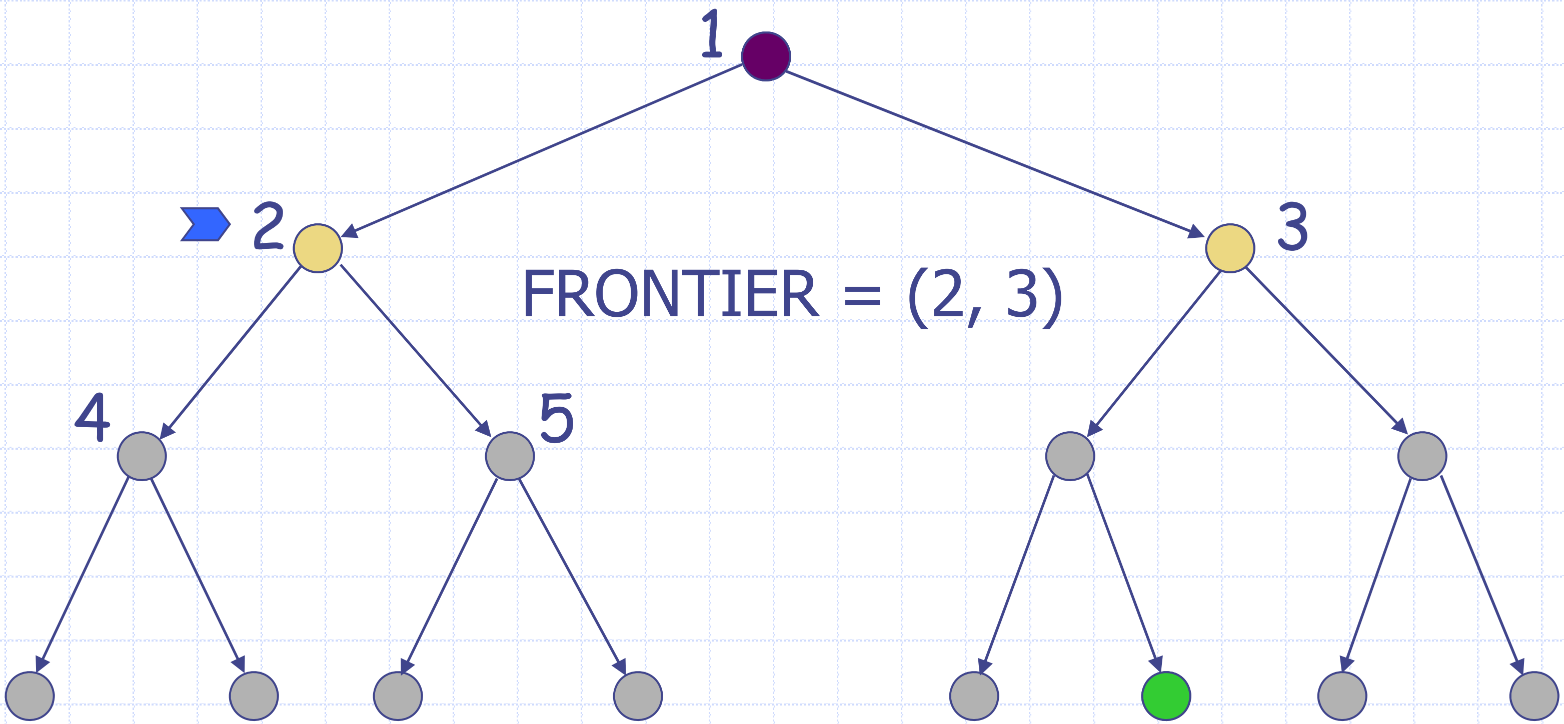
Depth-First Strategy

- New nodes are inserted at the front of FRONTIER



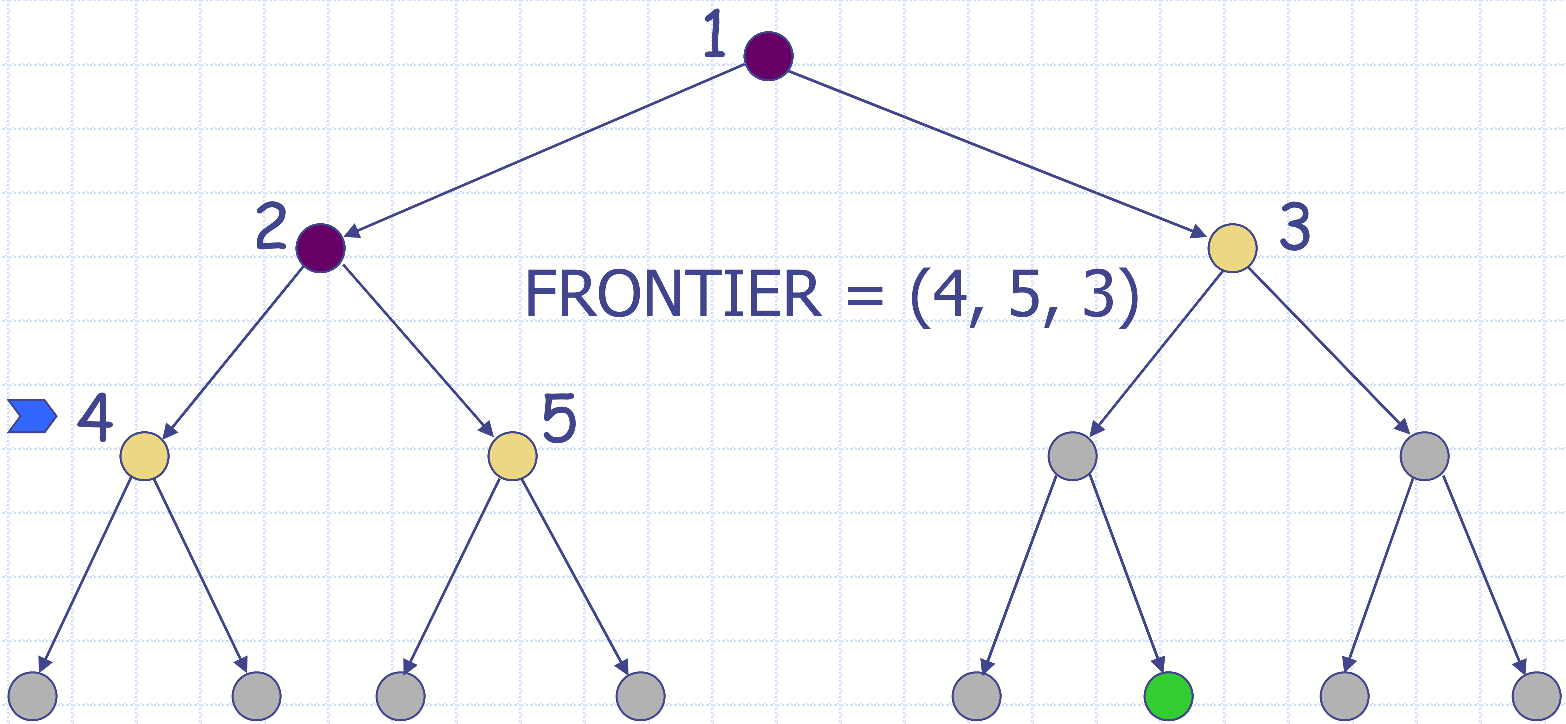
Depth-First Strategy

- New nodes are inserted at the front of FRONTIER



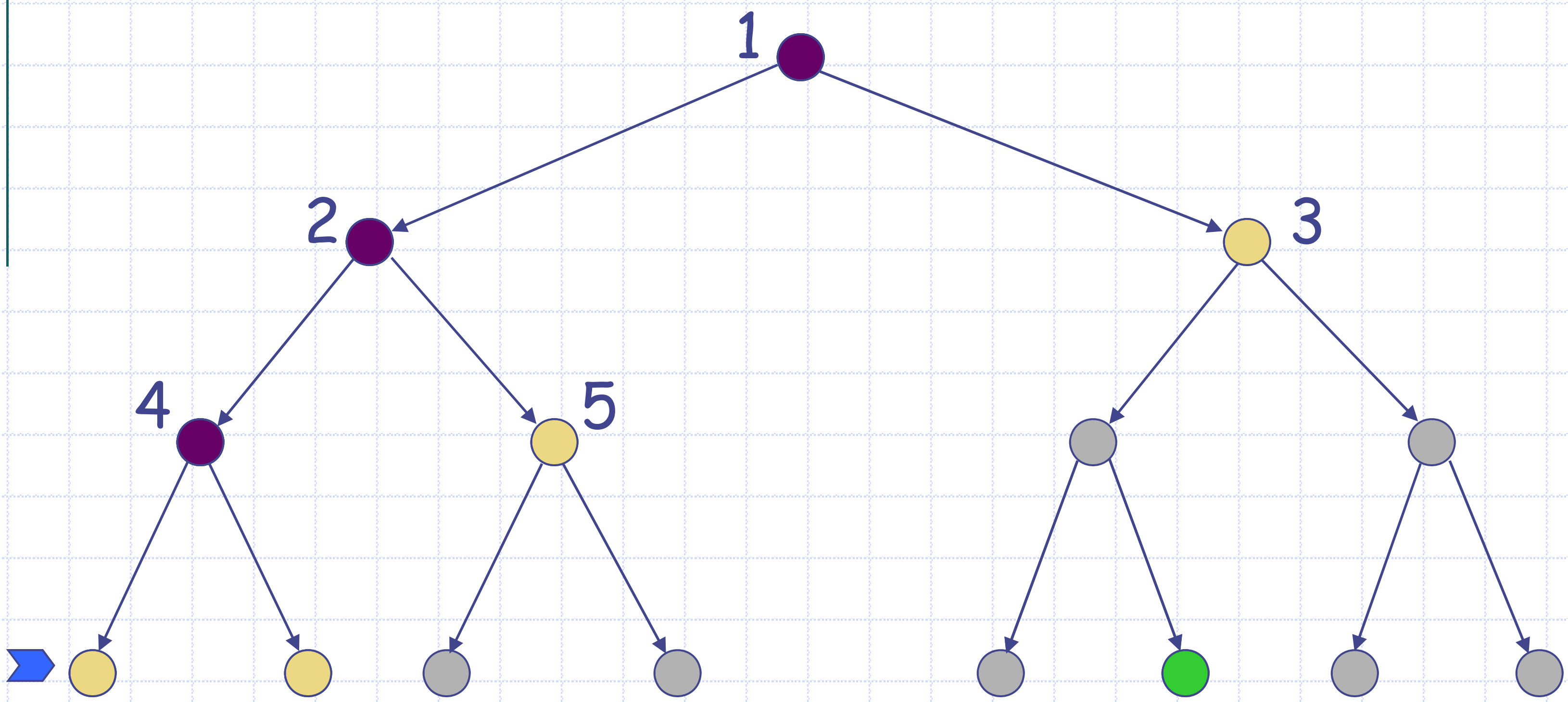
Depth-First Strategy

- New nodes are inserted at the front of FRONTIER



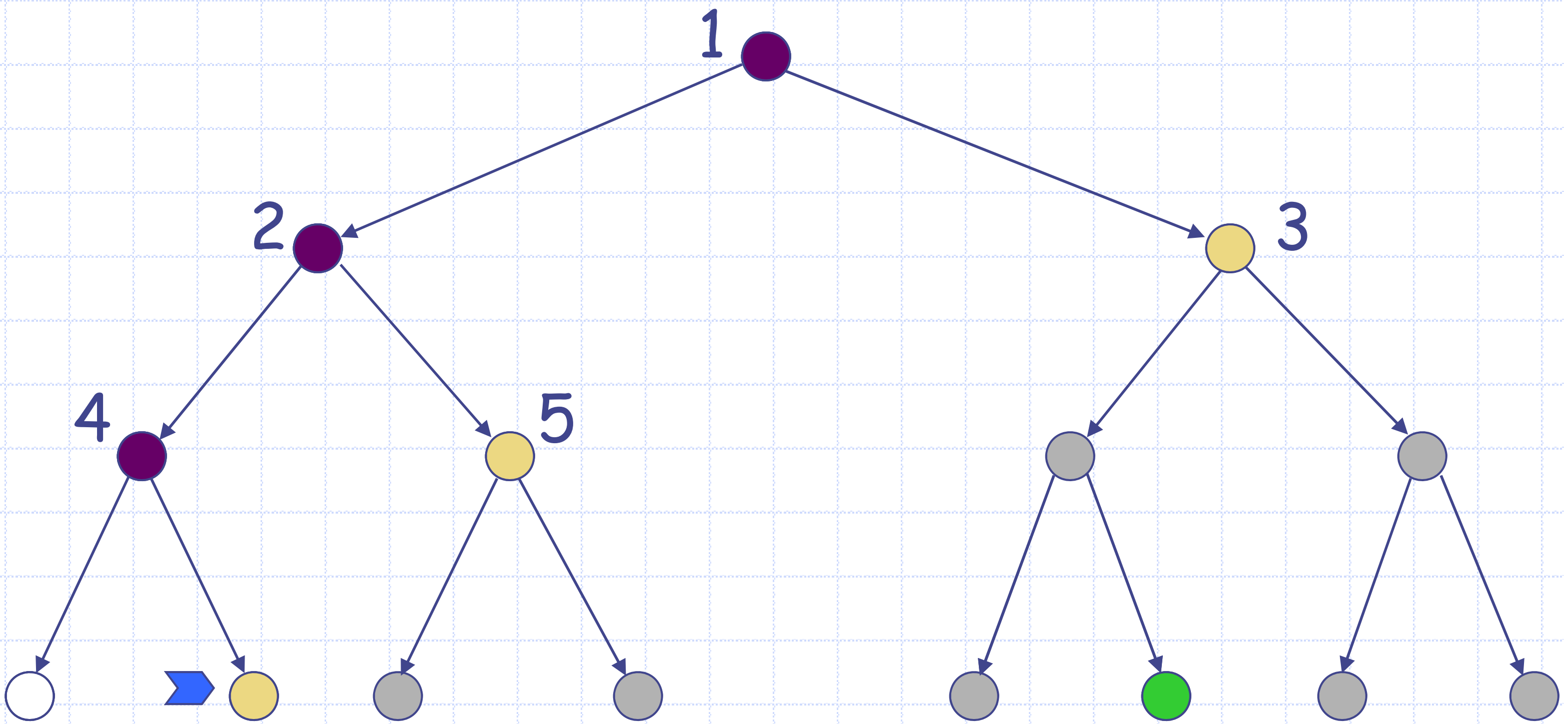
Depth-First Strategy

- New nodes are inserted at the front of FRONTIER



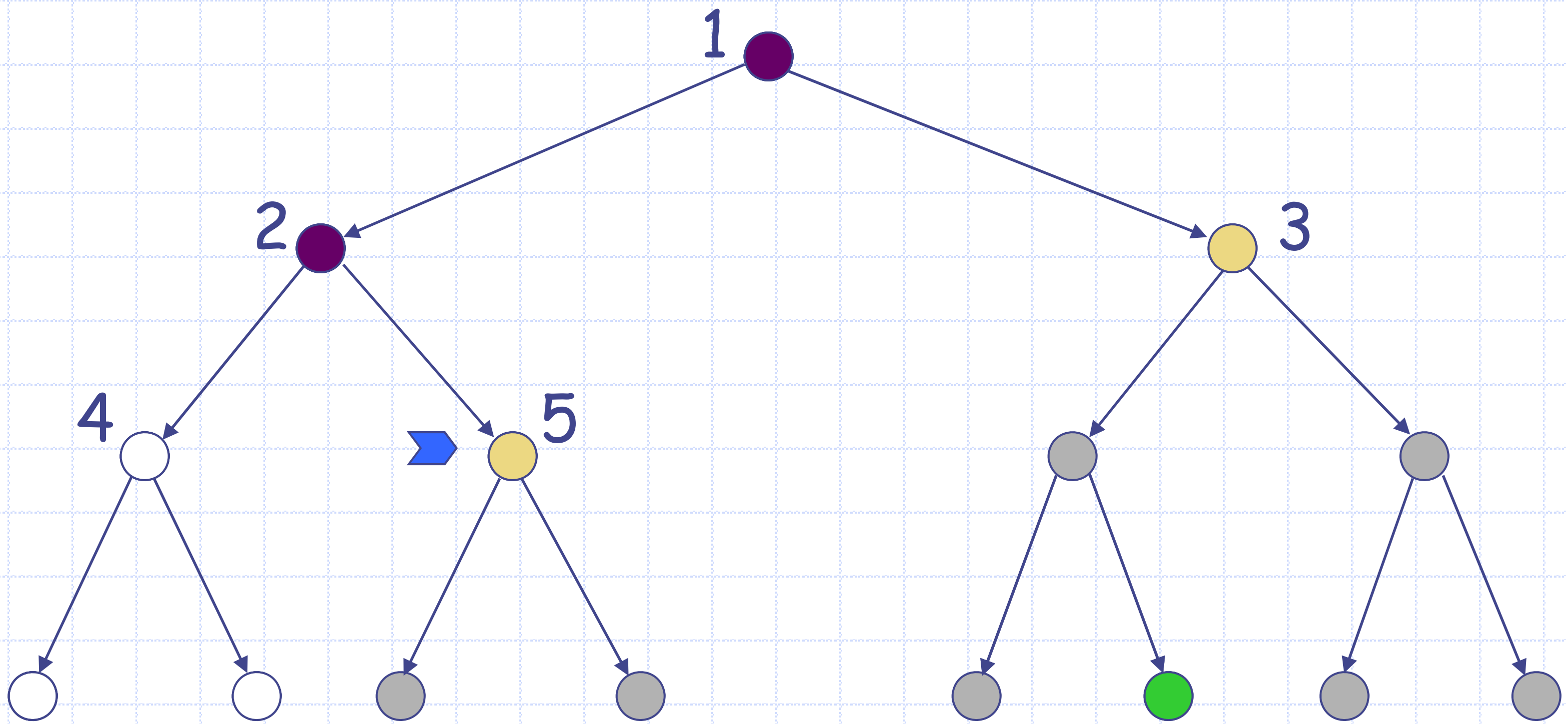
Depth-First Strategy

- New nodes are inserted at the front of FRONTIER



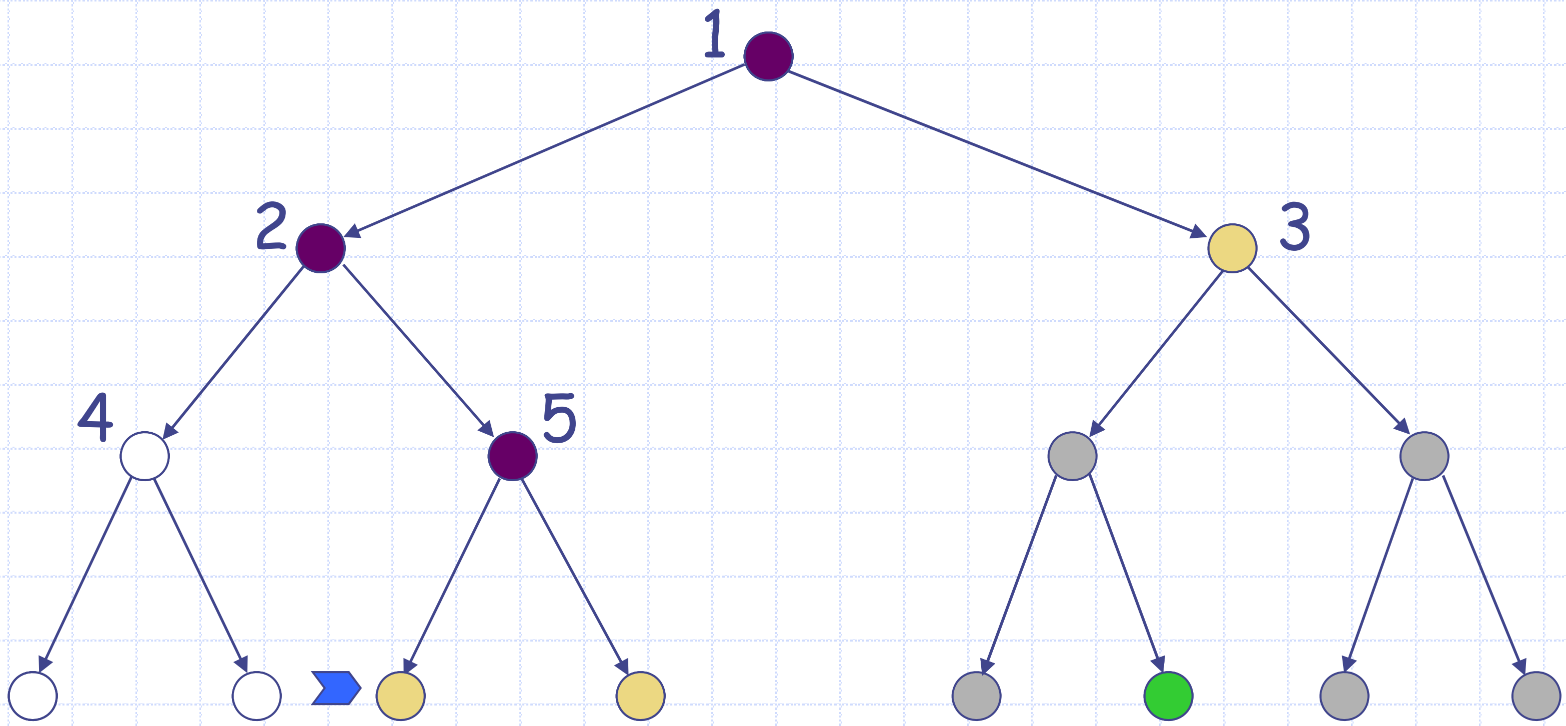
Depth-First Strategy

- New nodes are inserted at the front of FRONTIER



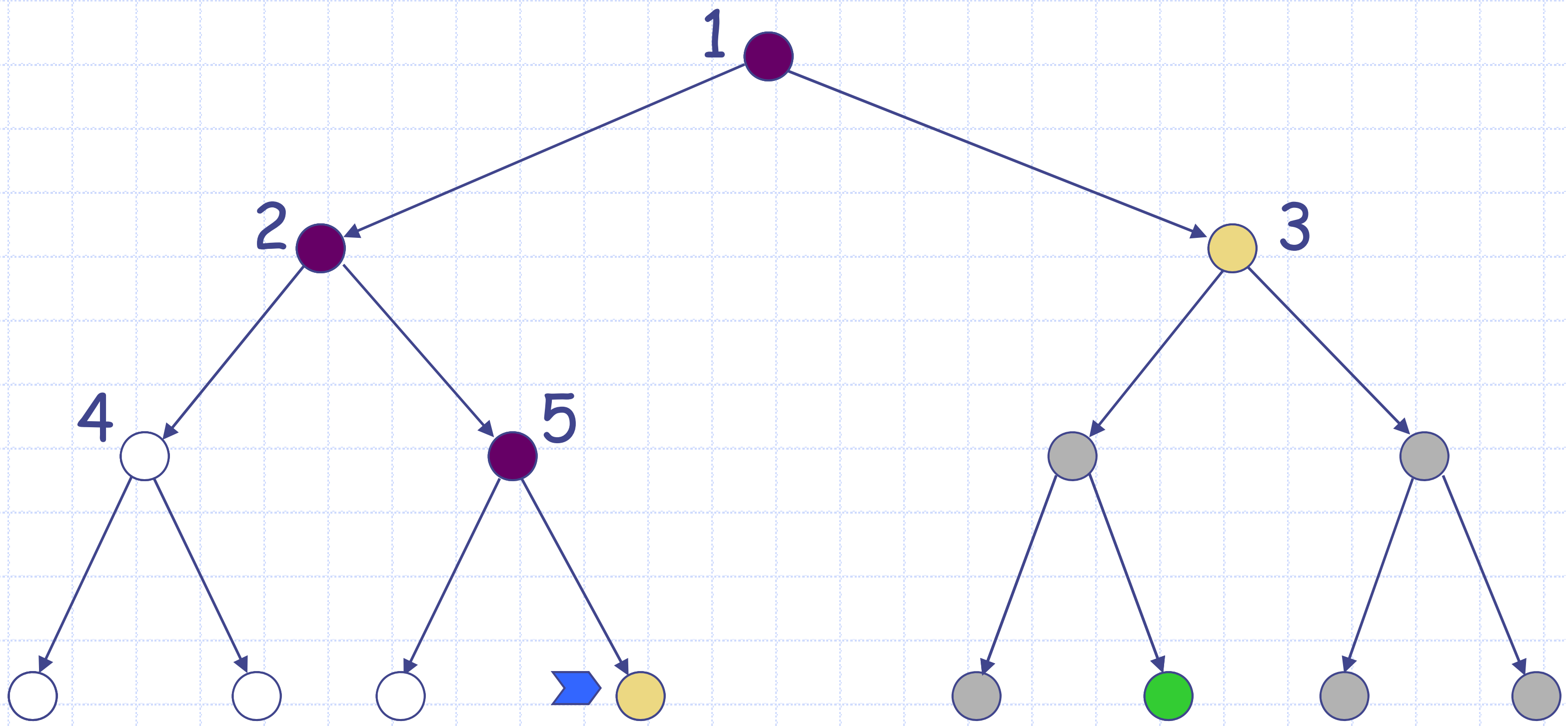
Depth-First Strategy

- New nodes are inserted at the front of FRONTIER



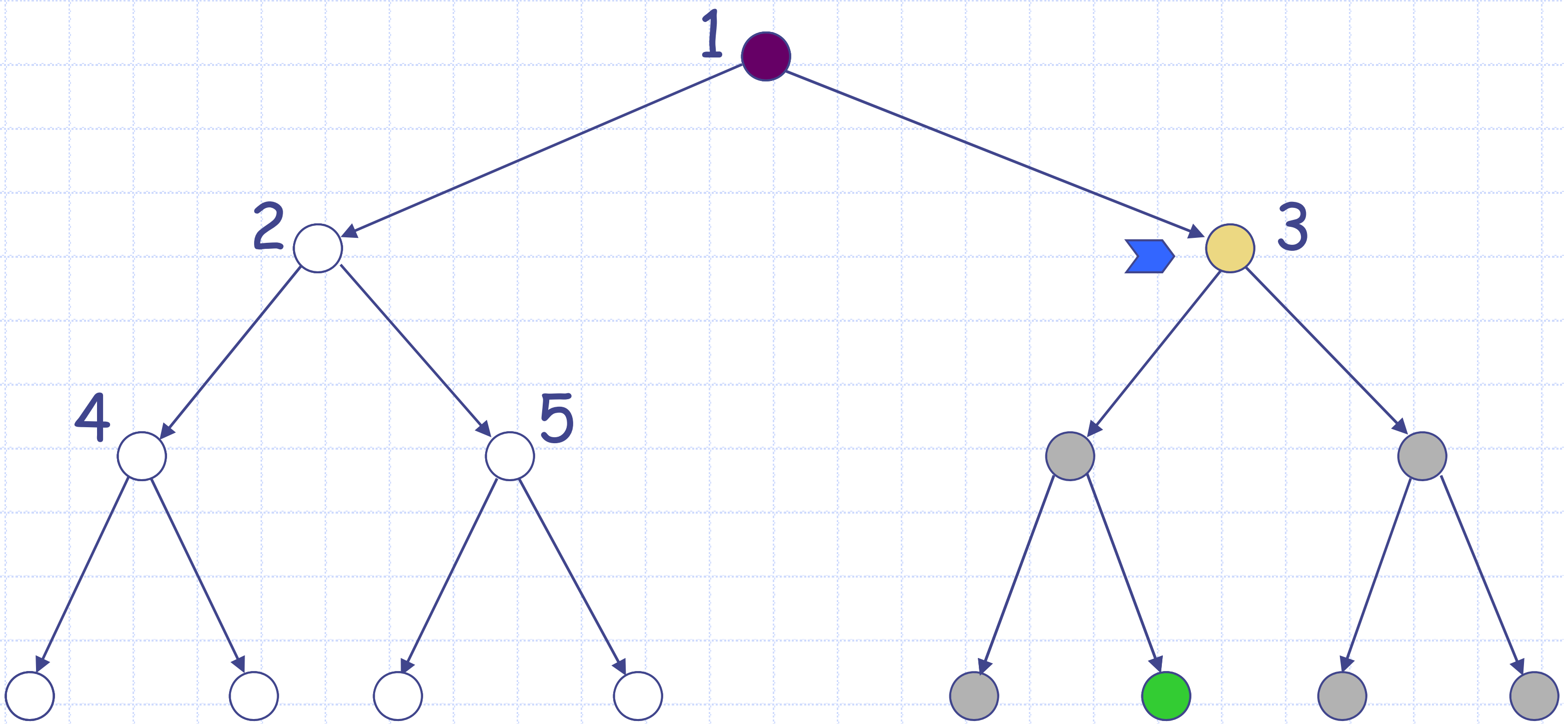
Depth-First Strategy

- New nodes are inserted at the front of FRONTIER



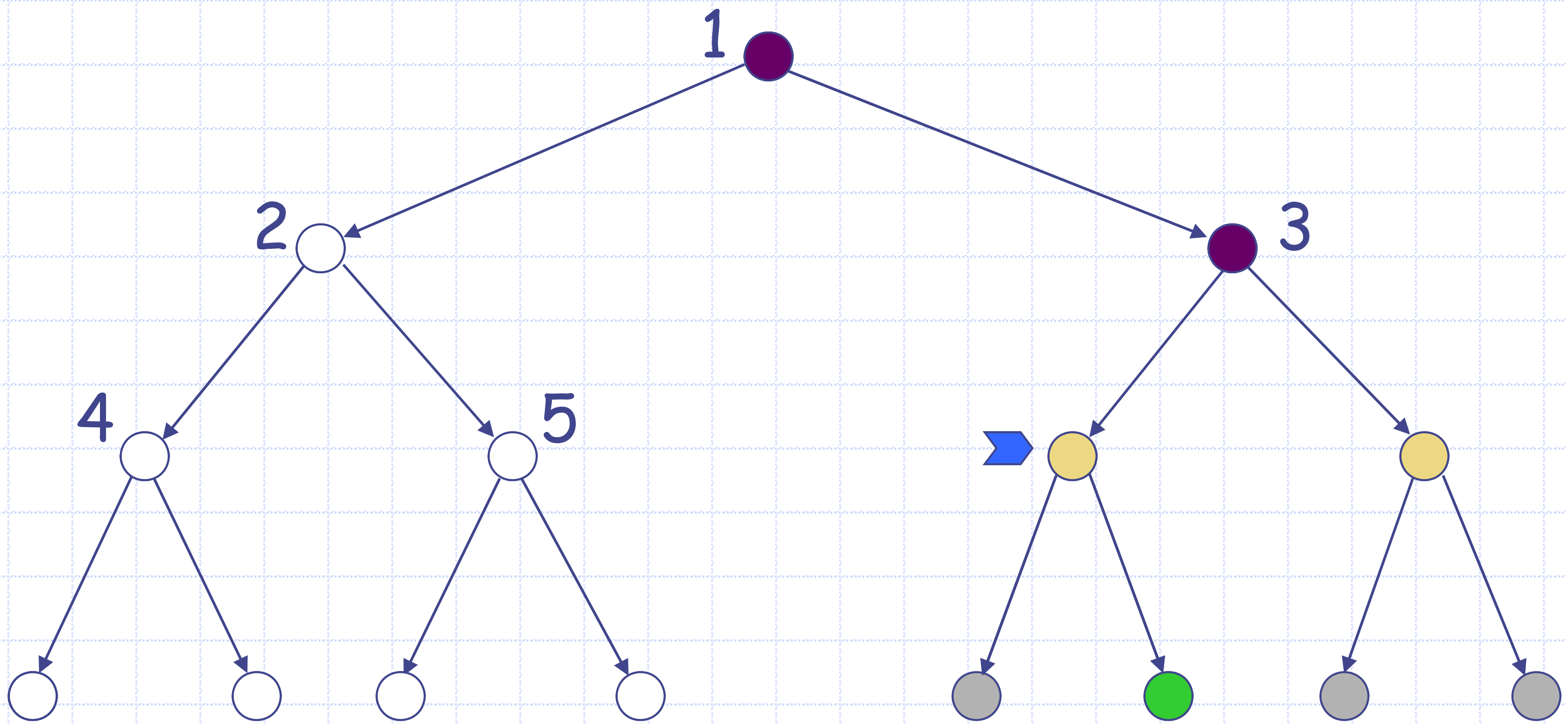
Depth-First Strategy

- New nodes are inserted at the front of FRONTIER



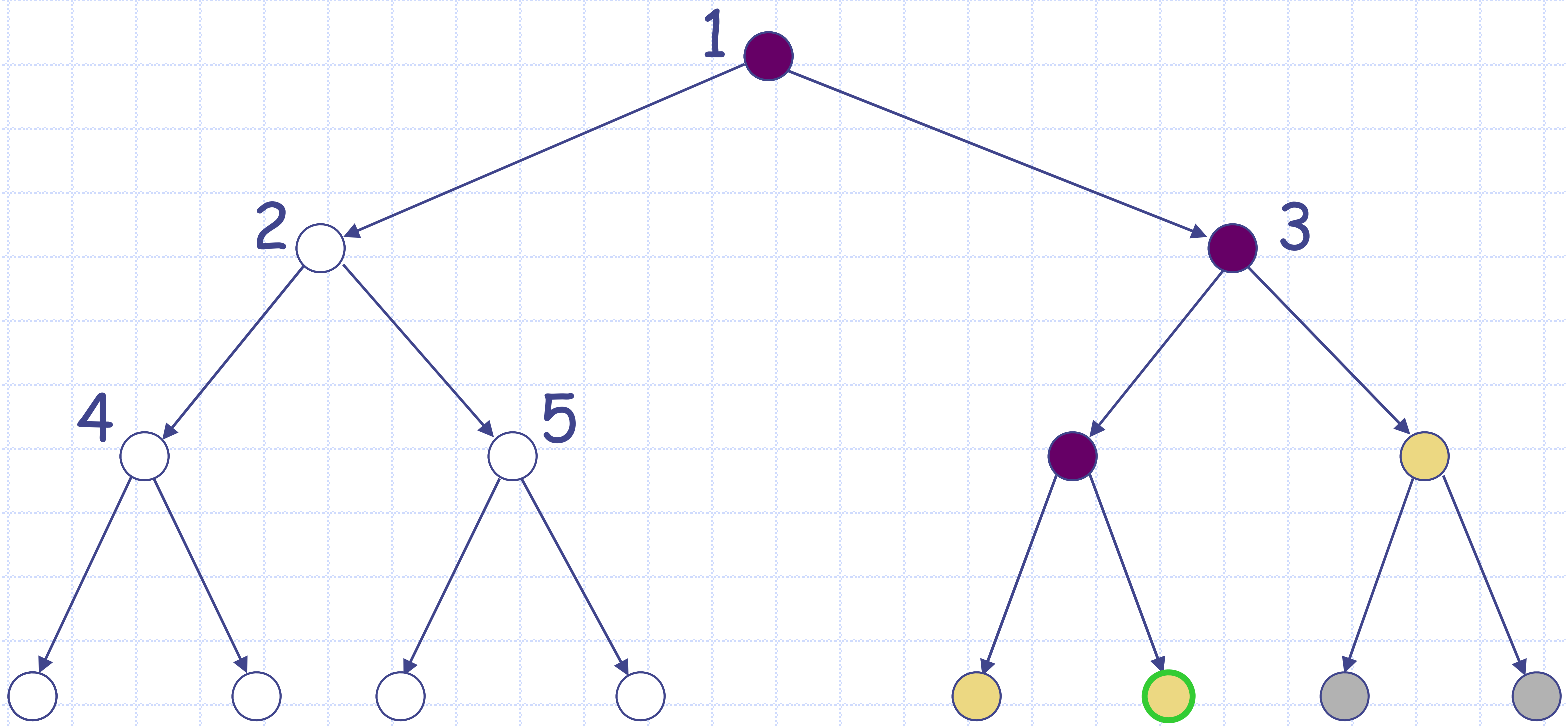
Depth-First Strategy

- New nodes are inserted at the front of FRONTIER



Depth-First Strategy

- New nodes are inserted at the front of FRONTIER



DF Evaluation

- b: branching factor
- d: depth of shallowest goal node
- m: maximal depth of a leaf node
- Depth-first search is:
  - Complete?
  - Optimal?

DF Evaluation

Reminder:
Breadth-first requires $O(b^d)$ time and space

- b: branching factor
- d: depth of shallowest goal node
- m: maximal depth of a leaf node
- Depth-first search is:
  - Complete only for finite search tree
  - Not optimal
- Number of nodes generated (worst case):
  $1 + b + b^2 + \ldots + b^m = O(b^m)$

→Time complexity is $O(b^m)$

→Space complexity is $O(bm)$

Depth-Limited Search

- Depth-first with depth cutoff k
  (depth at which nodes are not expanded)

- Three possible outcomes:
  - Solution
  - Failure (no solution)
  - Cutoff (no solution within cutoff)

Iterative Deepening Search

- Provides the best of both breadth-first and depth-first search

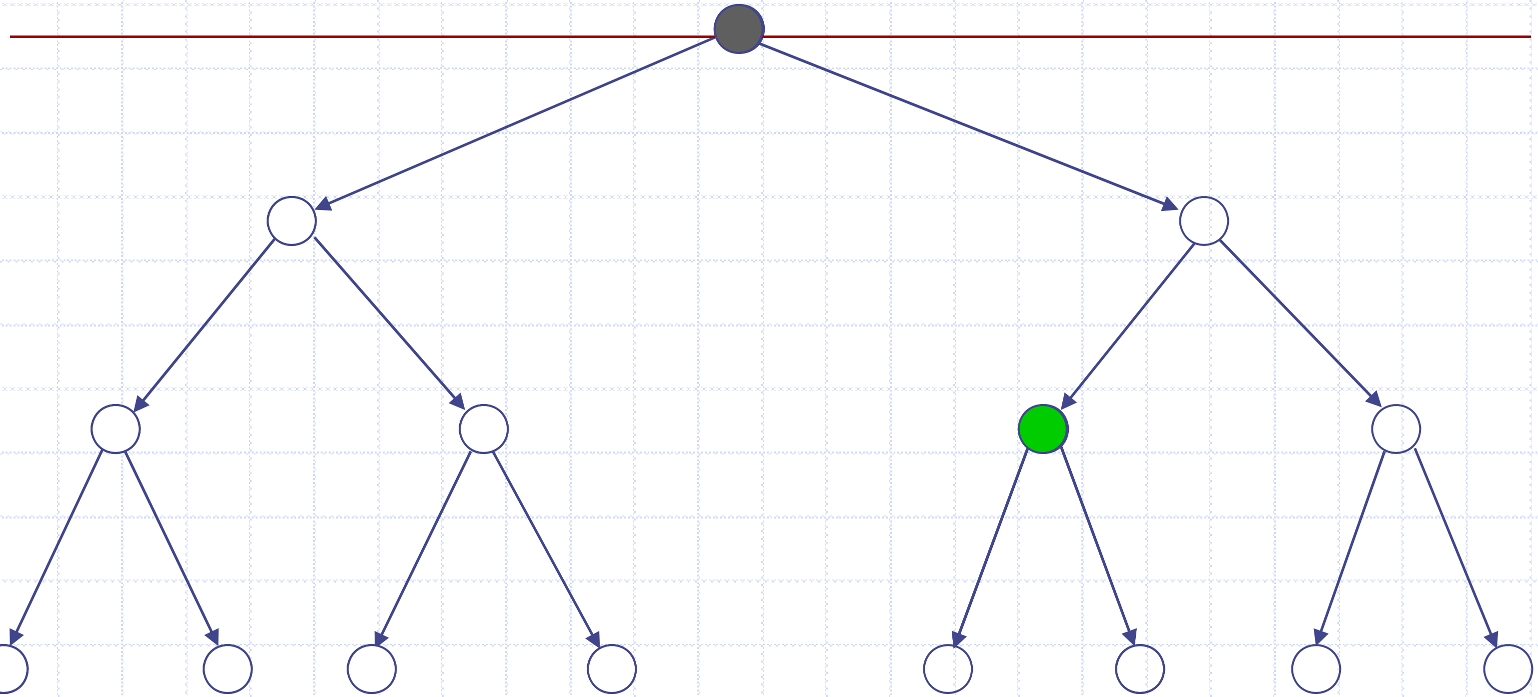
IDS

For k = 0, 1, 2, … do:

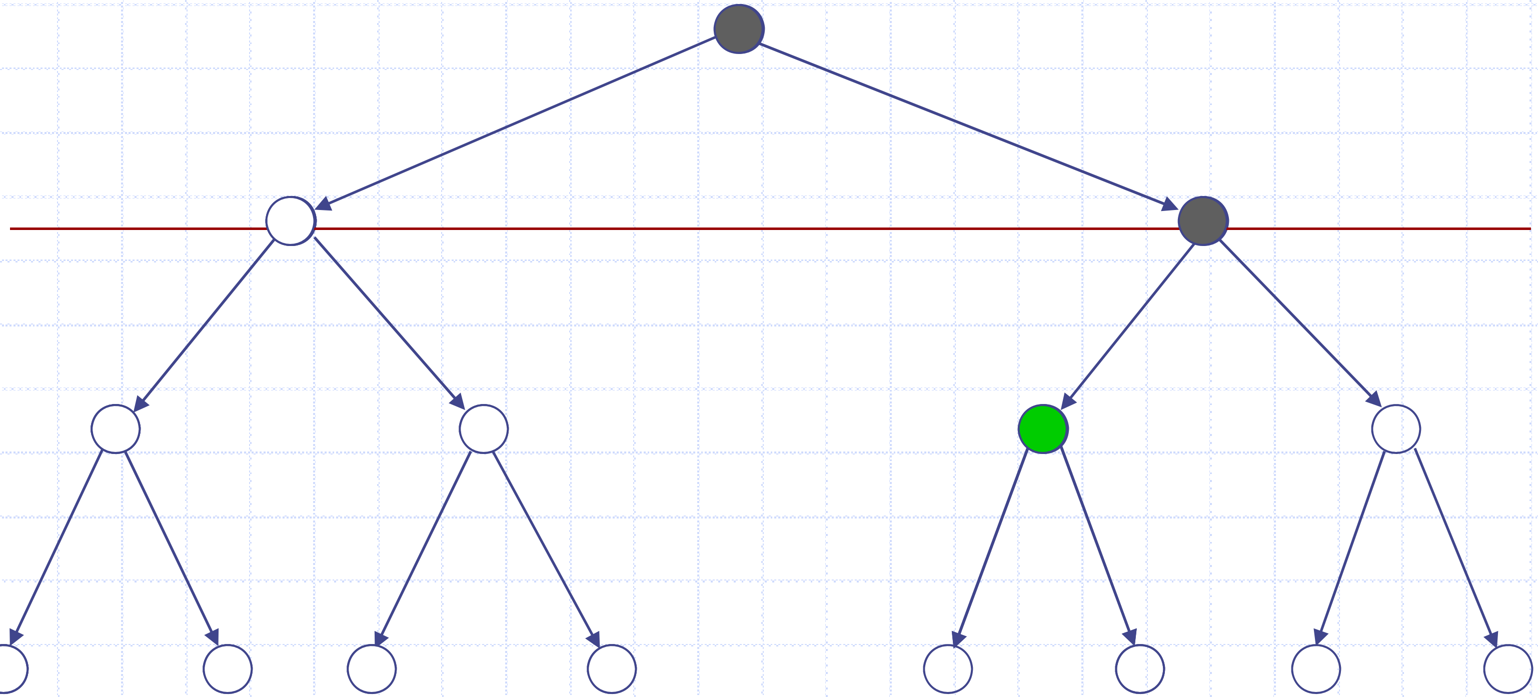
  Perform depth-first search with depth cutoff k

  (i.e., only generate nodes with depth $\leq$ k)

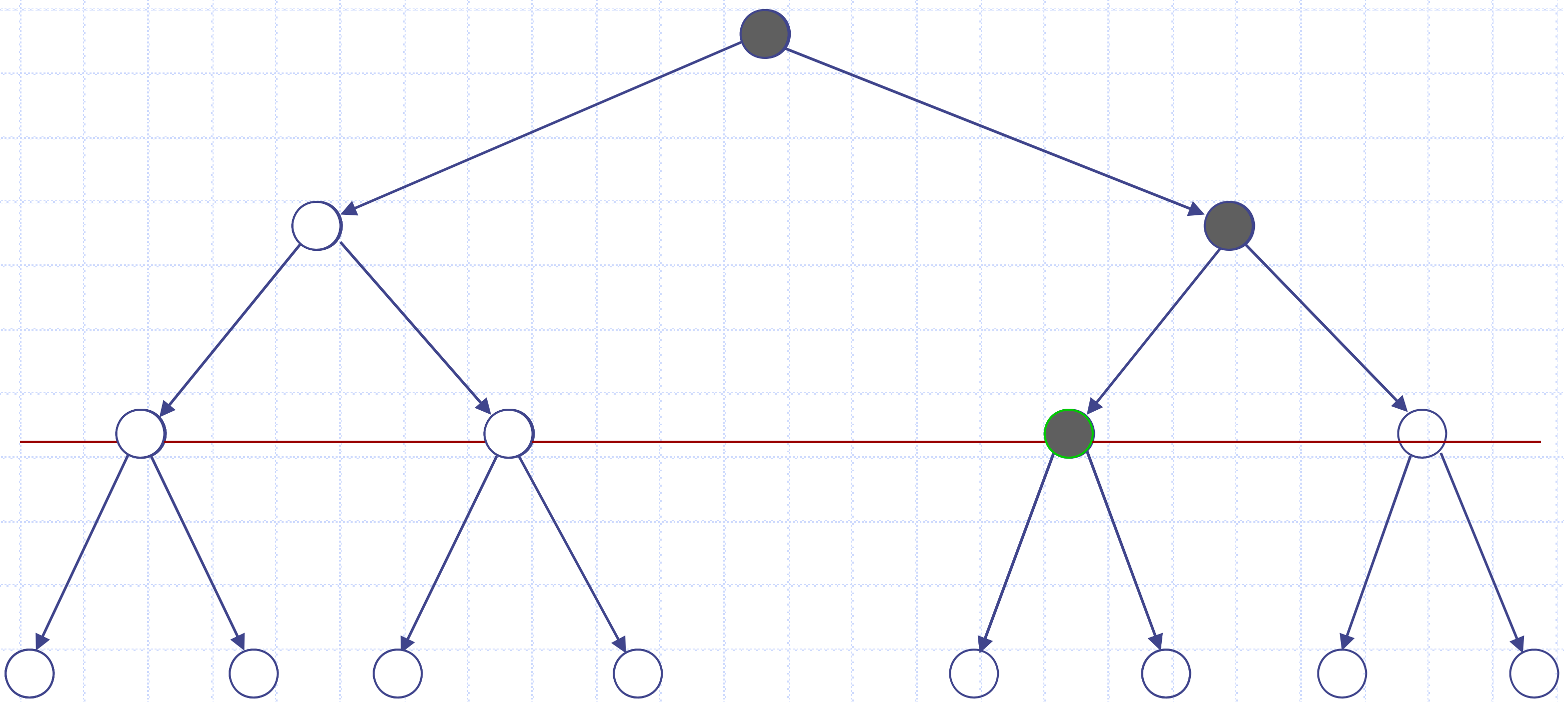
Iterative Deepening



Iterative Deepening



Iterative Deepening



ID Evaluation

- Iterative deepening search is:
  - Complete
  - Optimal if step cost =1
- Time complexity is:

  $db + (d-1)b^2 + \ldots + (1)\, b^d = O(b^d)$
- Space complexity is: $O(bd)$

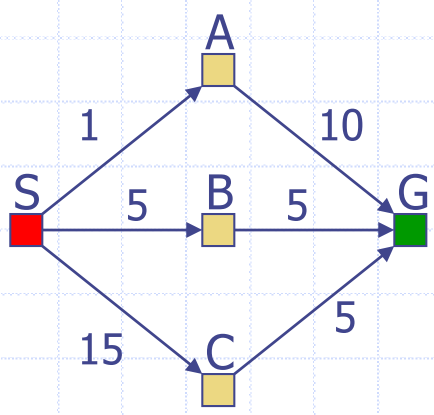
Comparison of Strategies

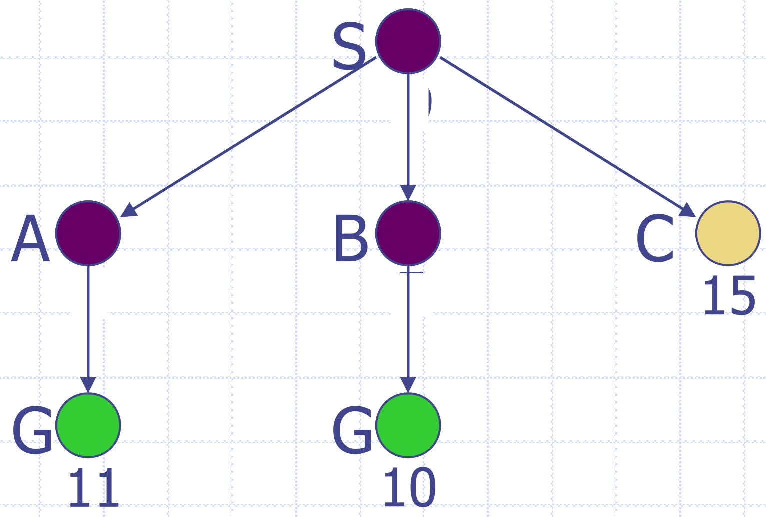
- Breadth-first is complete and optimal, but has high space complexity
- Depth-first is space efficient, but is neither complete, nor optimal
- Iterative deepening is complete and optimal, with the same space complexity as depth-first and almost the same time complexity as breadth-first

Uniform-Cost Search

- Each arc has some cost $c \geq \varepsilon > 0$
- The cost of the path to each node N is

  $g(N) = \Sigma$ costs of arcs
- The goal is to generate a solution path of minimal cost
- The nodes N in the queue FRONTIER are sorted in increasing g(N)





- Need to modify search algorithm

Search Algorithm #2

The goal test is applied to a node when this node is expanded, not when it is generated.

1. INSERT(N_0,FRONTIER)
2. Repeat:
   a. If EMPTY?(FRONTIER) then return failure
   b. N = POP(FRONTIER)
   c. s = STATE(N)
   d. If GOAL?(s) then return path or goal state
   e. For every state s' in SUCCESSORS(s)
      i. Create a new node N' as a child of N
      ii. INSERT(N',FRONTIER)

Expansion of N