CS 235: Artificial Intelligence

Week 2

Blind (Uninformed) Search

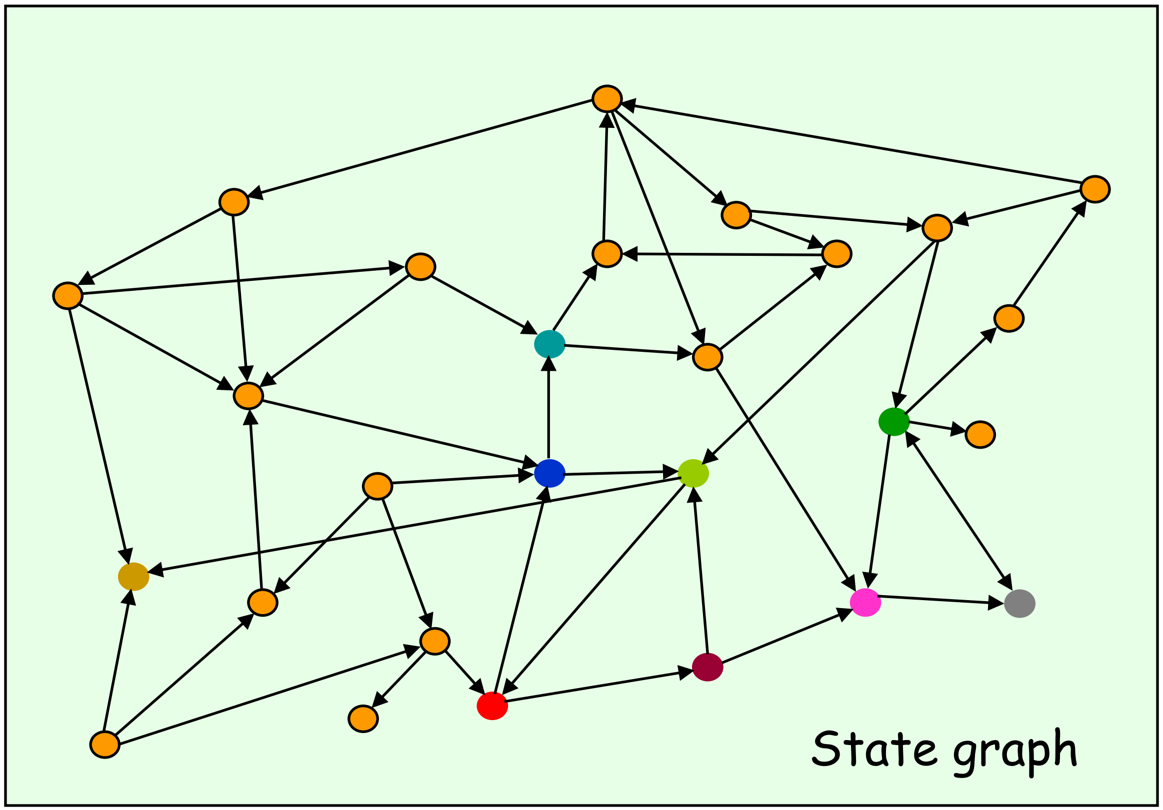
Dr. Moumita Roy
CSE Dept., IIITG

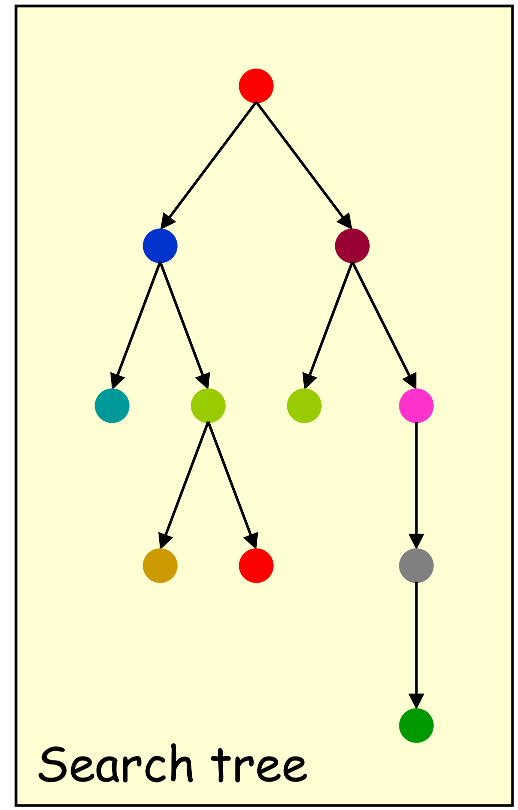
Reference: http://ai.stanford.edu/~latombe/cs121/2011/schedule.htm

Simple Problem-Solving-Agent Agent Algorithm

1. $s_0 \leftarrow$ sense/read initial state
2. GOAL? $\leftarrow$ select/read goal test
3. Succ $\leftarrow$ read successor function
4. solution $\leftarrow$ **search**(s_0, GOAL?, Succ)
5. perform(solution)

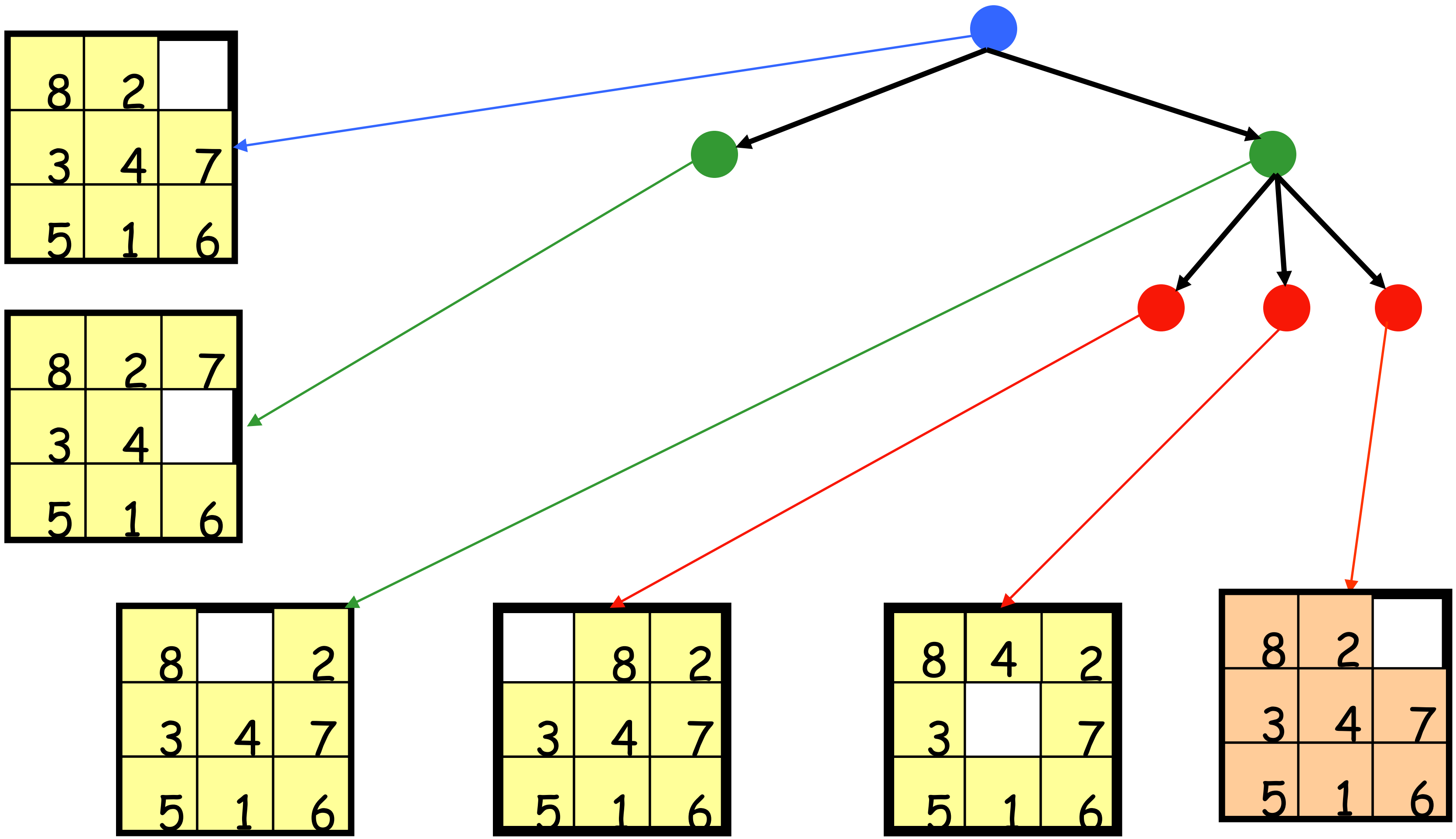
Search Tree



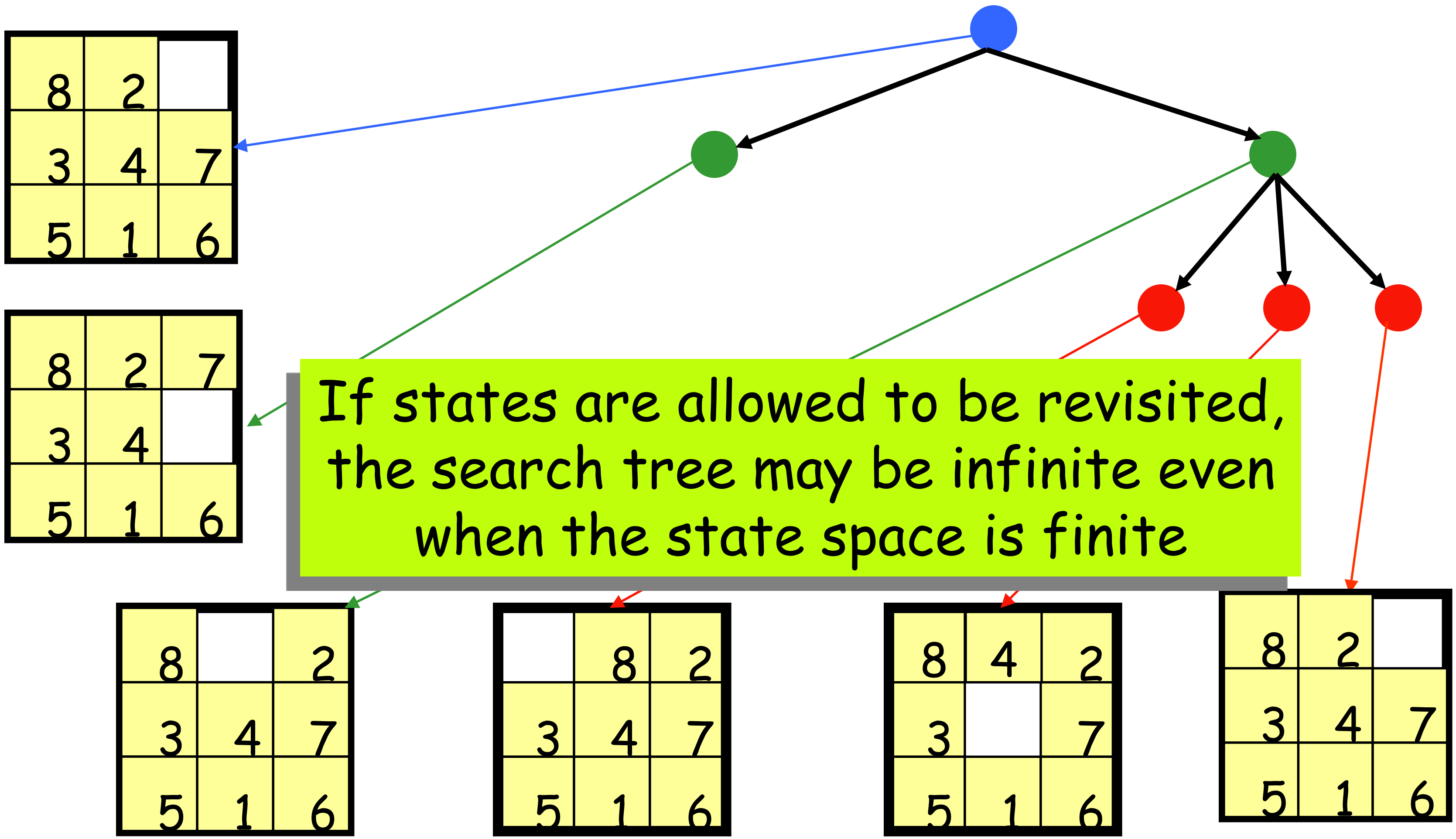


Note that some states may be visited multiple times

Search Nodes and States

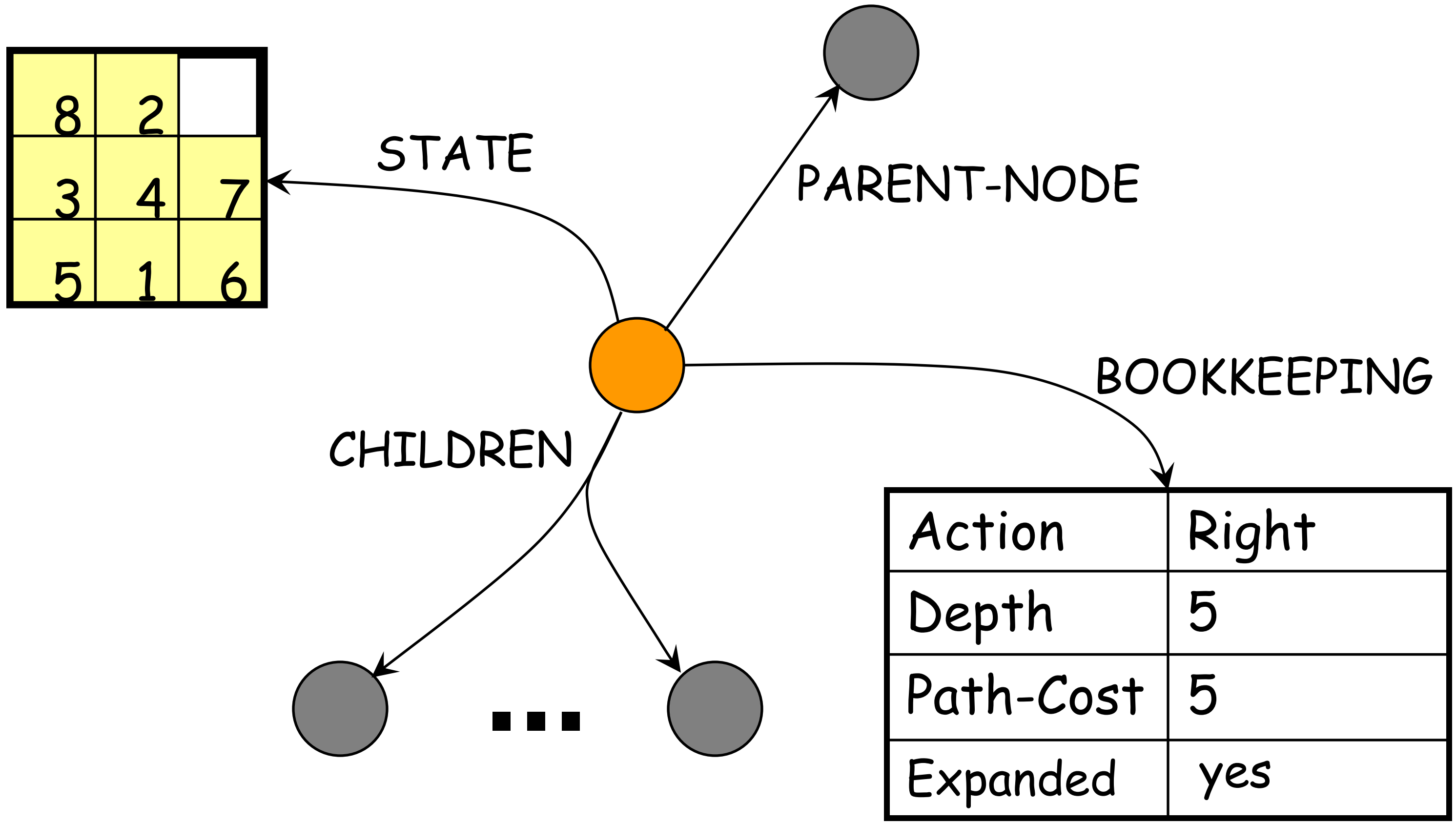


Search Nodes and States



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



Depth of a node N
= length of path from root to N

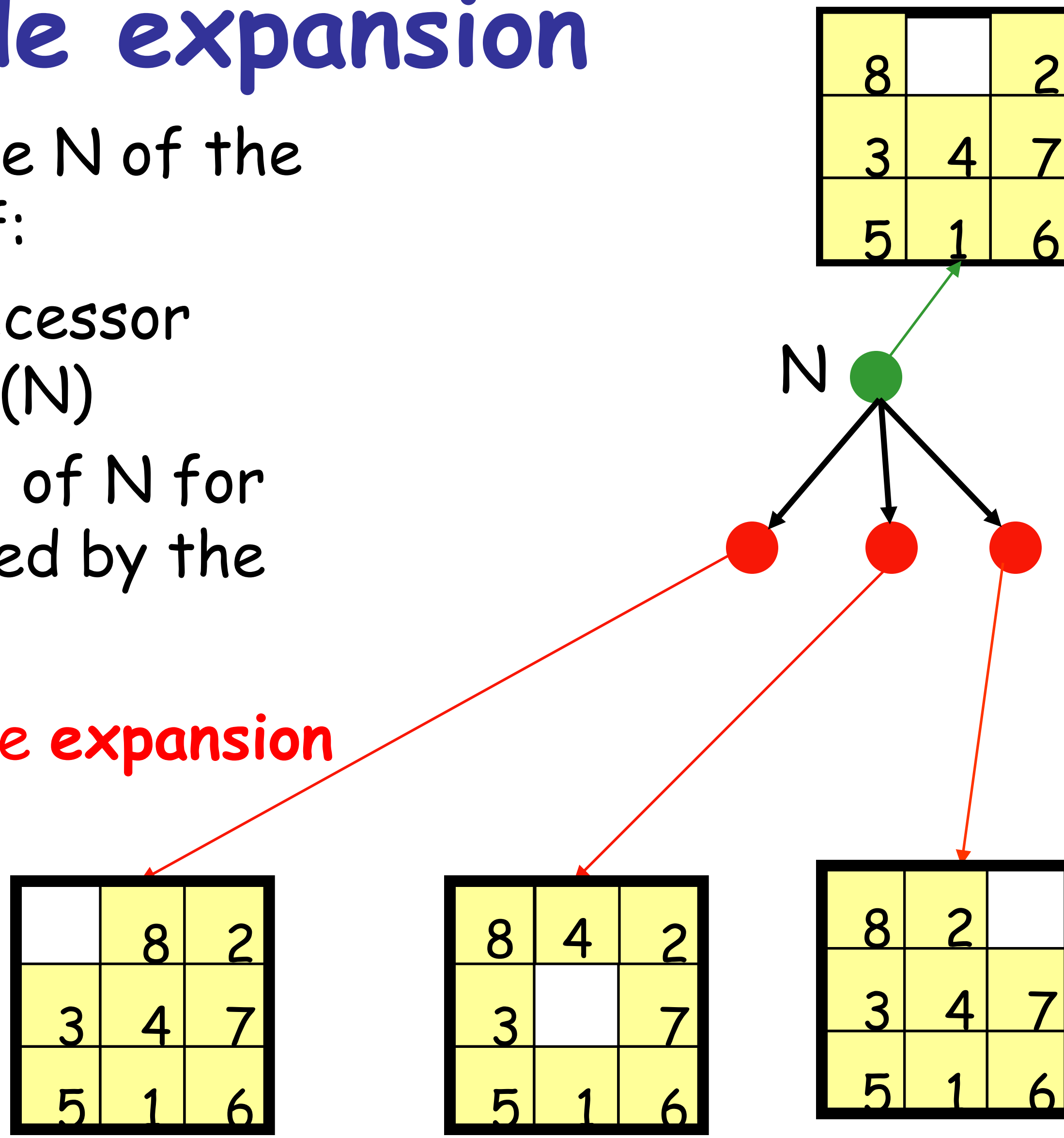
(depth of the root = 0)

Node expansion

The expansion of a node N of the search tree consists of:

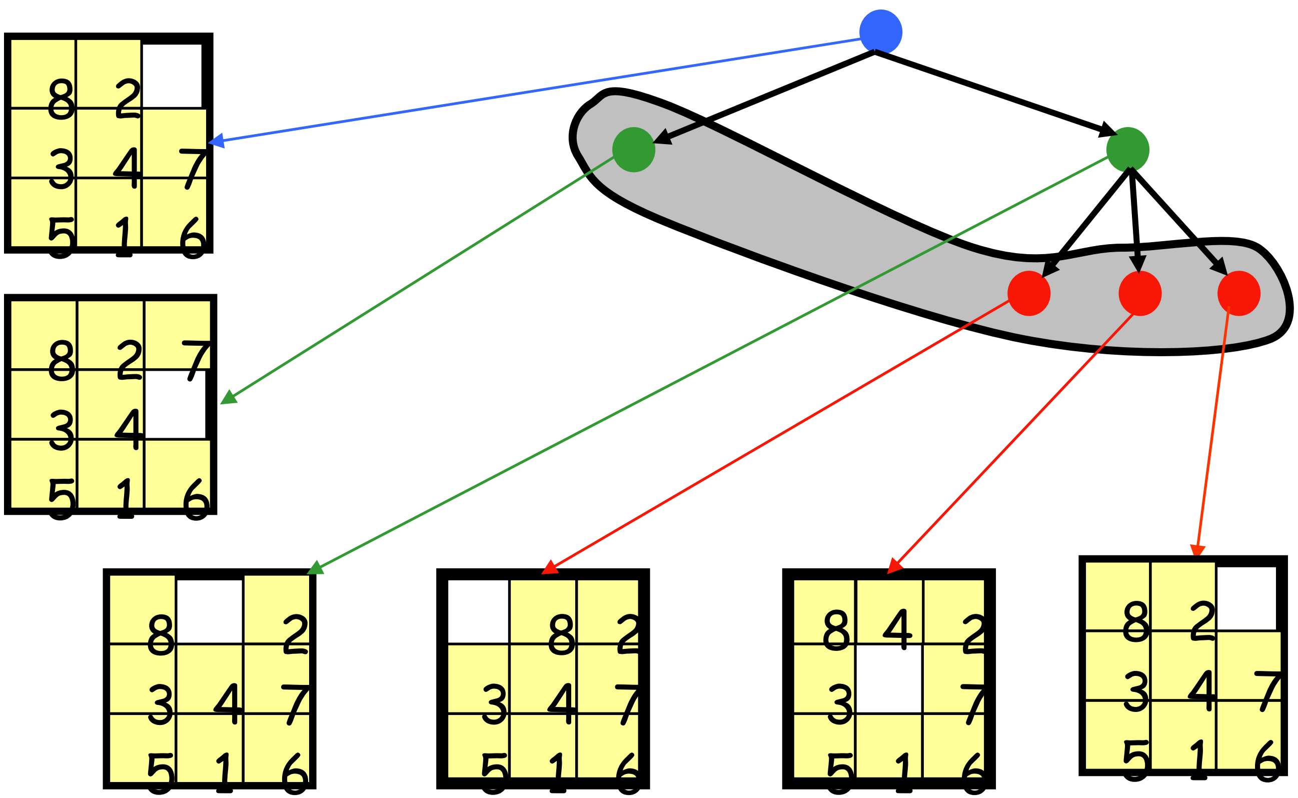
1) Evaluating the successor function on STATE(N)
2) Generating a child of N for each state returned by the function

node **generation** ≠ node **expansion**



Fringe of Search Tree

- The fringe is the set of all search nodes that haven't been expanded yet



Search Strategy

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

Search Algorithm #1

SEARCH#1

1. If GOAL?(initial-state) then return initial-state
2. INSERT(initial-node,FRINGE)
3. Repeat:
   a. If empty(FRINGE) then return failure
   b. N ← REMOVE(FRINGE)
   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',FRINGE)

Expansion of N

Performance Measures

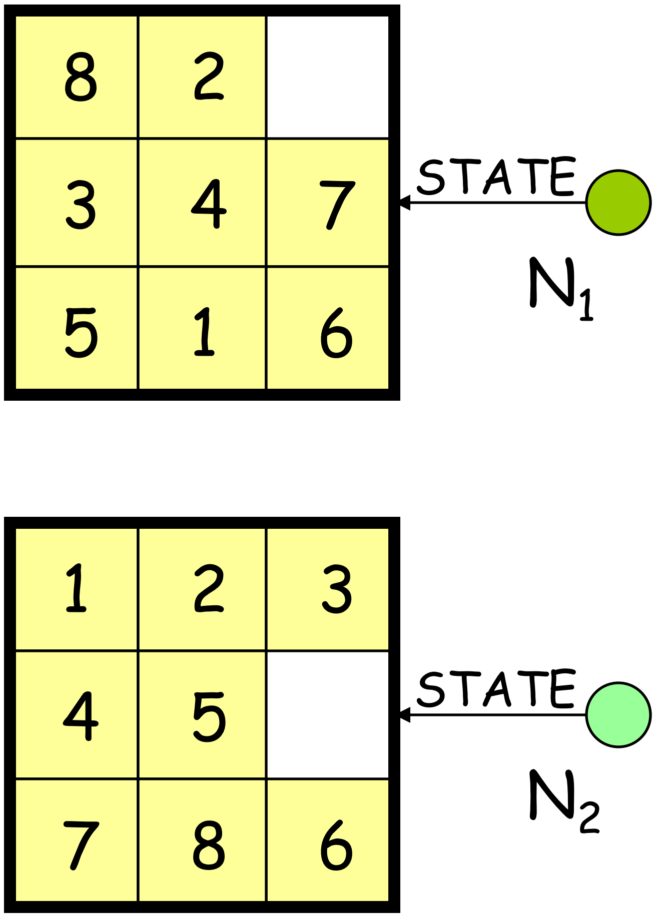
- Completeness
  A search algorithm is complete if it finds a solution whenever one exists
  [What about the case when no solution exists?]
- Optimality
  A search algorithm is optimal if it returns a minimum-cost path whenever a solution exists
- Complexity
  It measures the time and amount of memory required by the algorithm

Blind vs. Heuristic Strategies

- Blind (or un-informed) strategies do not exploit state descriptions to order FRINGE. They only exploit the positions of the nodes in the search tree
- Heuristic (or informed) strategies exploit state descriptions to order FRINGE (the most "promising" nodes are placed at the beginning of FRINGE)

Example

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

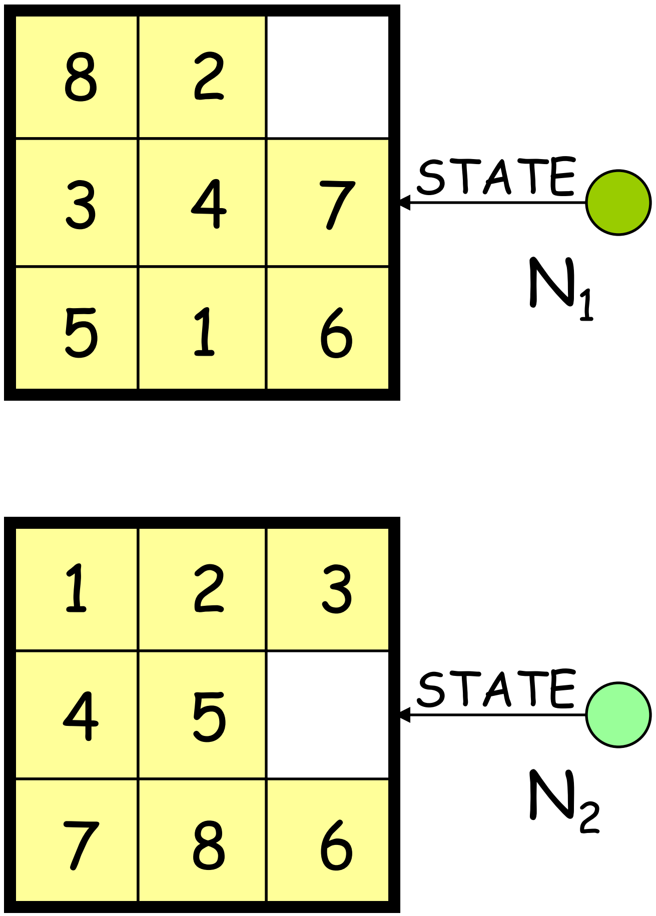


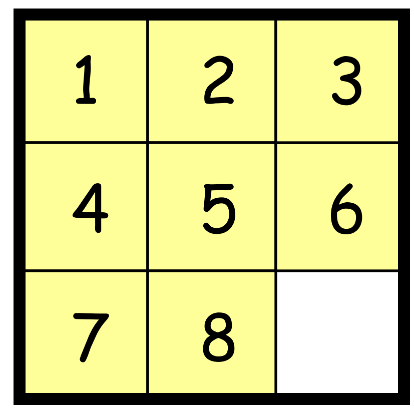
1	2	3
4	5	6
7	8	

Goal state

Example

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





Goal state

Blind Strategies

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

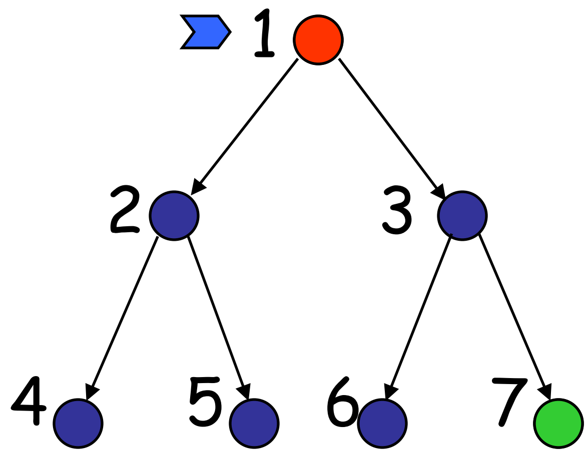
Arc cost = 1

- Uniform-Cost
  (variant of breadth-first)

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

Breadth-First Strategy

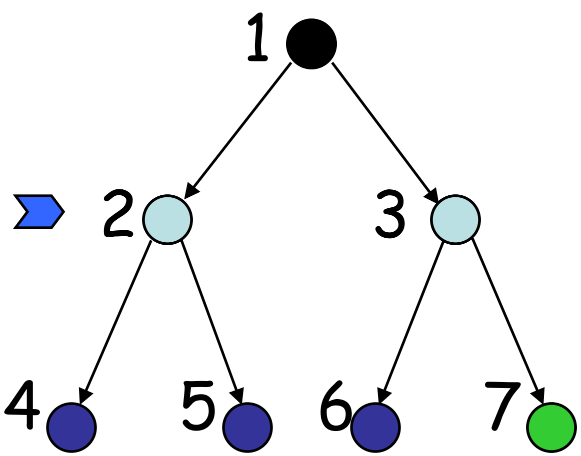
New nodes are inserted at the end of FRINGE



FRINGE = (1)

Breadth-First Strategy

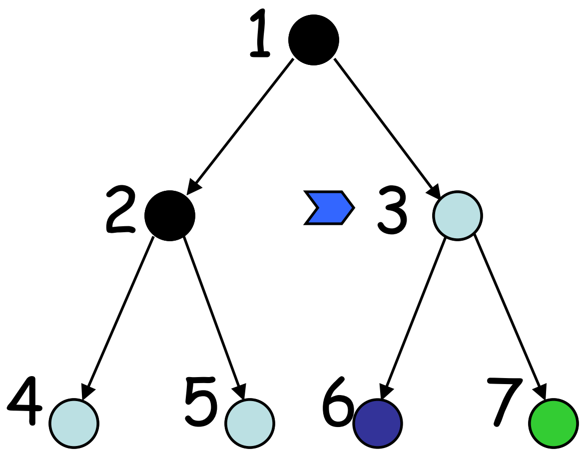
New nodes are inserted at the end of FRINGE



FRINGE = (2, 3)

Breadth-First Strategy

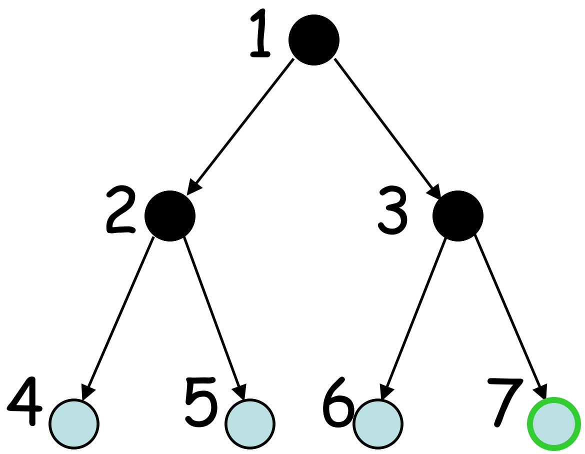
New nodes are inserted at the end of FRINGE



FRINGE = (3, 4, 5)

Breadth-First Strategy

New nodes are inserted at the end of FRINGE



FRINGE = (4, 5, 6, 7)

Important Parameters

1) Maximum number of successors of any state

   → branching factor b of the search tree

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

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

Evaluation

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

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:

  ???

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:
  $1 + b + b^2 + \ldots + b^d \quad = \;$???

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:
  $1 + b + b^2 + \ldots + b^d = (b^{d+1}-1)/(b-1) = O(b^d)$
- → Time and space complexity is $O(b^d)$
- When we can apply goal test? Node generation/Node expansion?

Evaluation

- If goal test is applied during node expansion (rather than node generation)
- In this case, the whole layer at depth d may be expanded before the goal was detected
- Time complexity: $O(b^{d+1})$

Time and Memory Requirements

d	# Nodes	Time	Memory
2	111	.01 msec	11 Kbytes
4	11,111	1 msec	1 Mbyte
6	~10^6	1 sec	100 Mb
8	~10^8	100 sec	10 Gbytes
10	~10^{10}	2.8 hours	1 Tbyte
12	~10^{12}	11.6 days	100 Tbytes
14	~10^{14}	3.2 years	10,000 Tbytes

Assumptions: b = 10; 1,000,000 nodes/sec; 100bytes/node

Time and Memory Requirements

d	# Nodes	Time	Memory
2	111	.01 msec	11 Kbytes
4	11,111	1 msec	1 Mbyte
6	$\sim 10^6$	1 sec	100 Mb
8	$\sim 10^8$	100 sec	10 Gbytes
10	$\sim 10^{10}$	2.8 hours	1 Tbyte
12	$\sim 10^{12}$	11.6 days	100 Tbytes
14	$\sim 10^{14}$	3.2 years	10,000 Tbytes

Assumptions: b = 10; 1,000,000 nodes/sec; 100bytes/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)

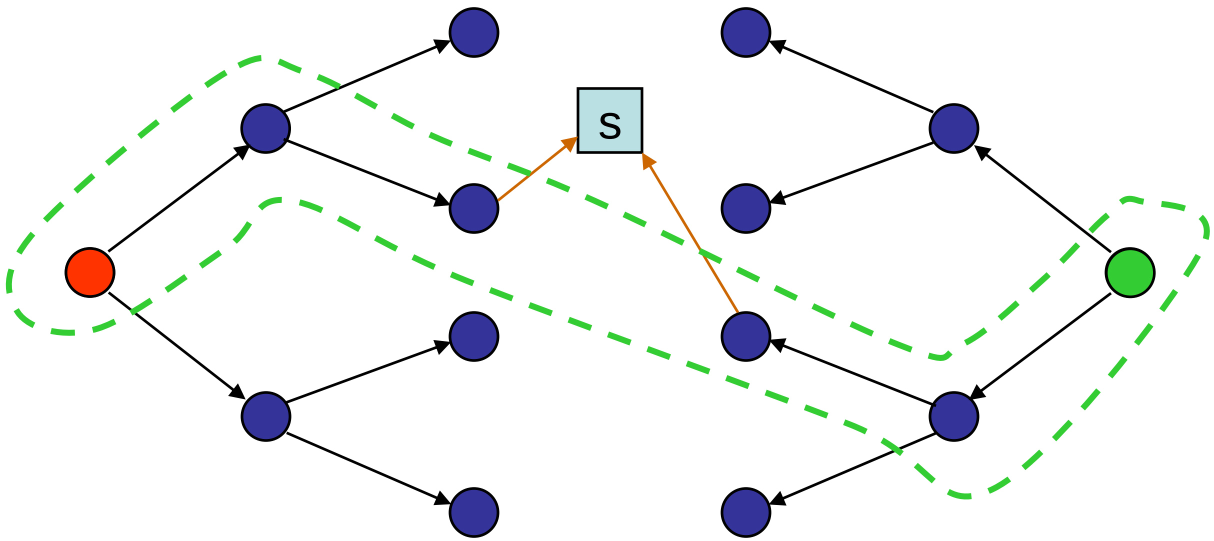
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

?

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

Bidirectional Strategy

2 fringe queues: FRINGE1 and FRINGE2

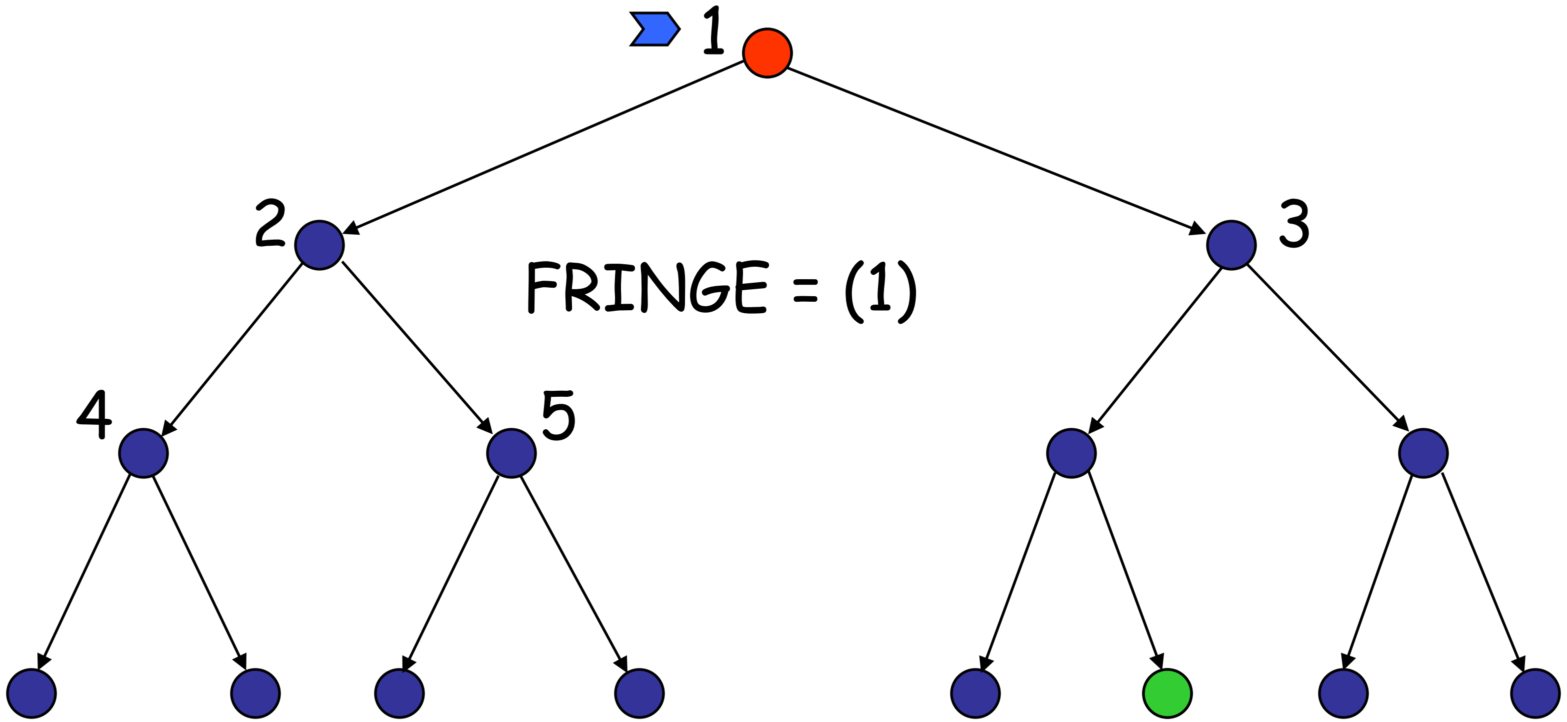


Time and space complexity is $O(b^{d/2}) << O(b^d)$
if both trees have the same branching factor b

Question: How we can search backward?
reverse action is not possible always.

Depth-First Strategy

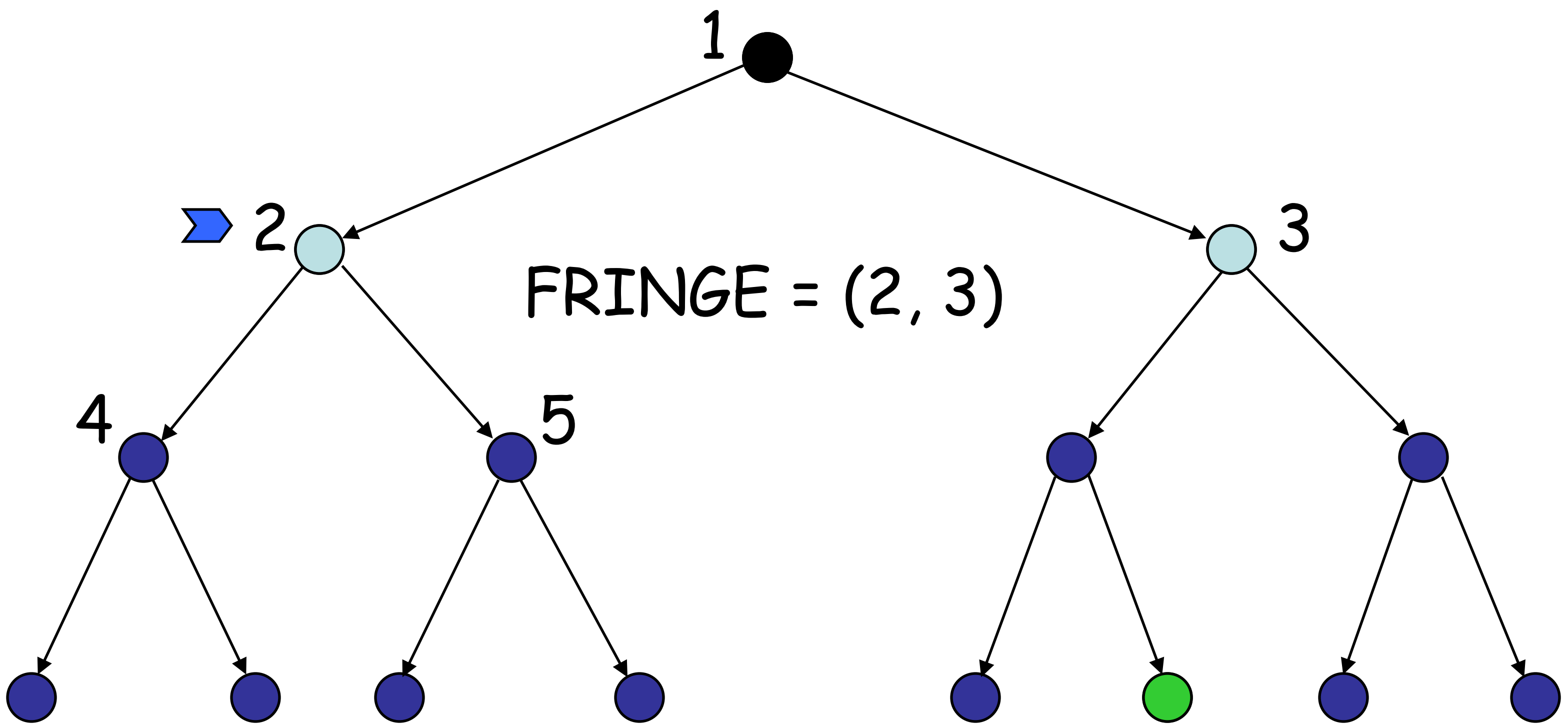
New nodes are inserted at the front of FRINGE



FRINGE = (1)

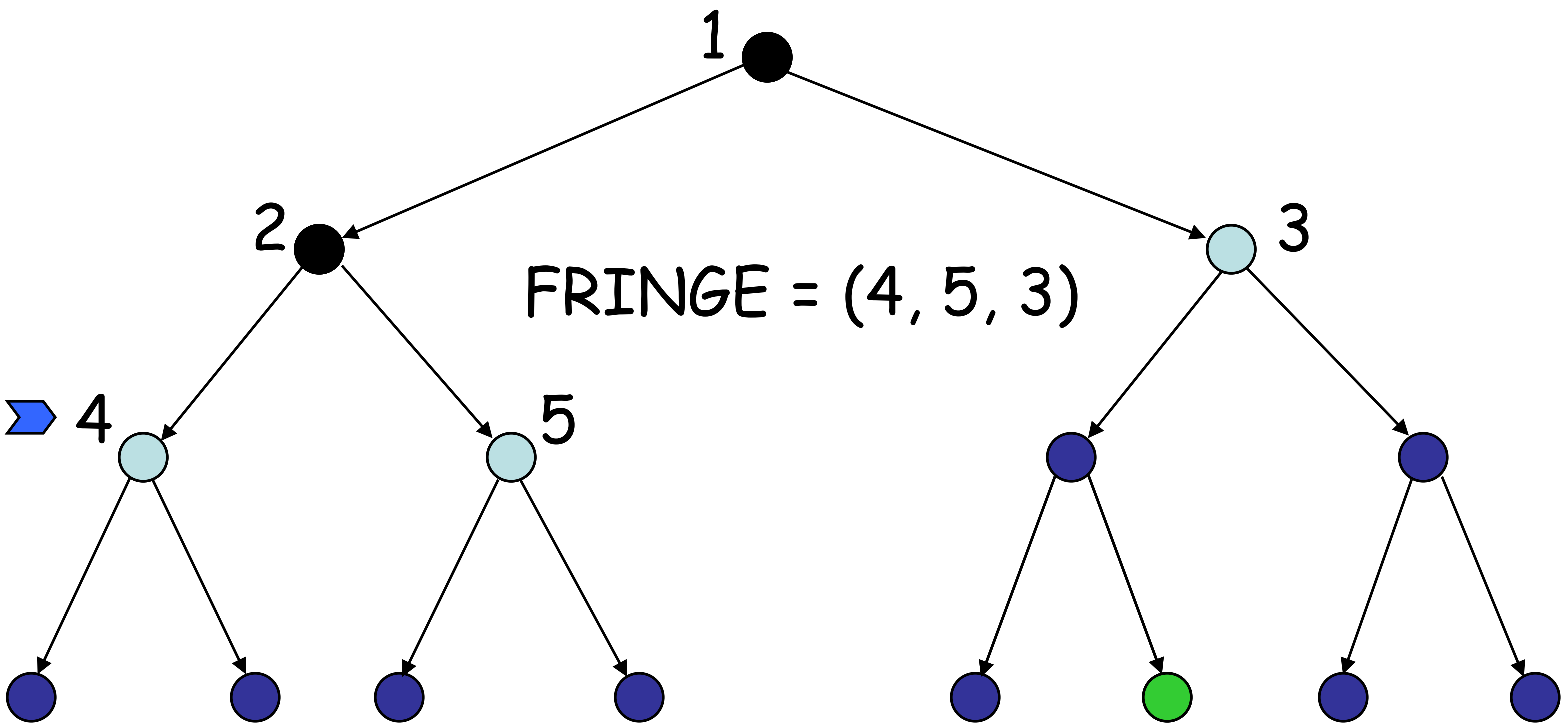
Depth-First Strategy

New nodes are inserted at the front of FRINGE



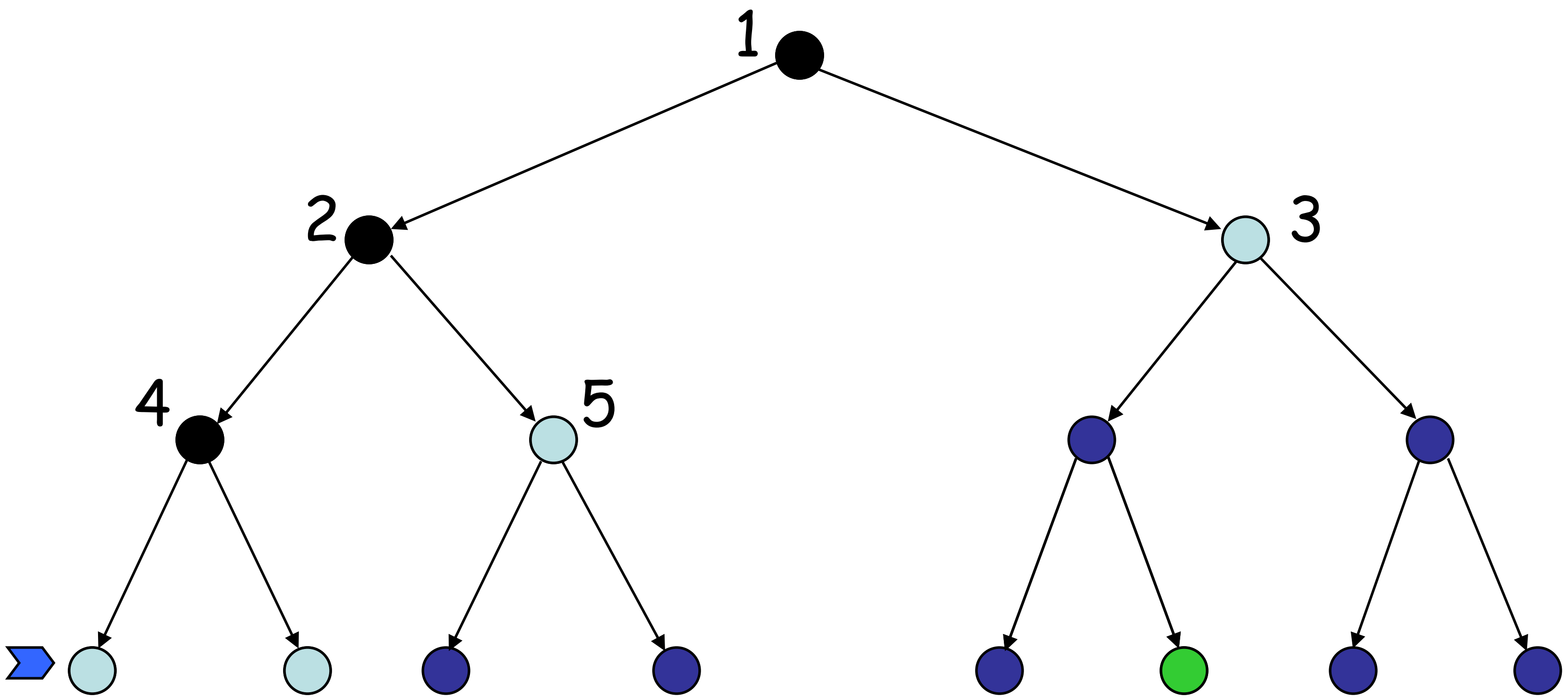
Depth-First Strategy

New nodes are inserted at the front of FRINGE



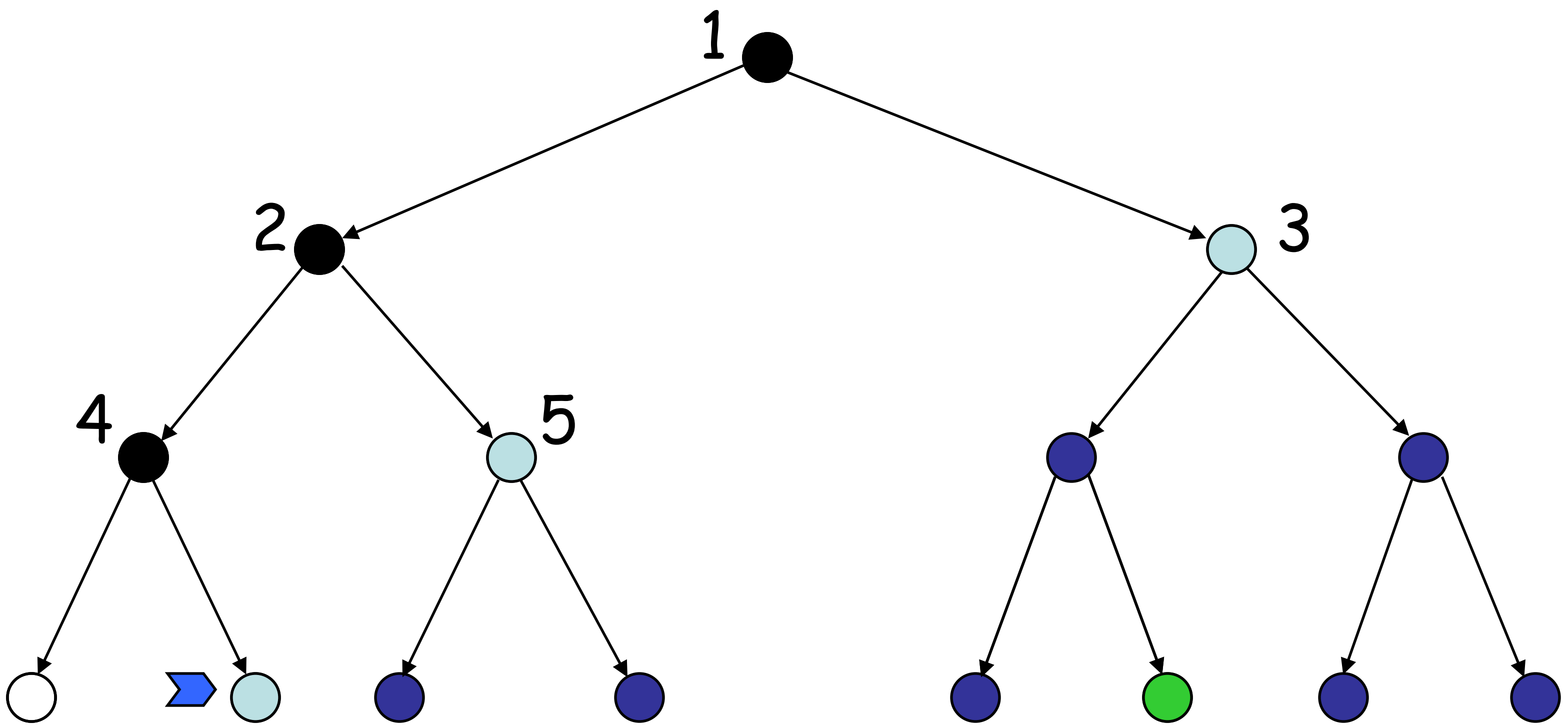
Depth-First Strategy

New nodes are inserted at the front of FRINGE



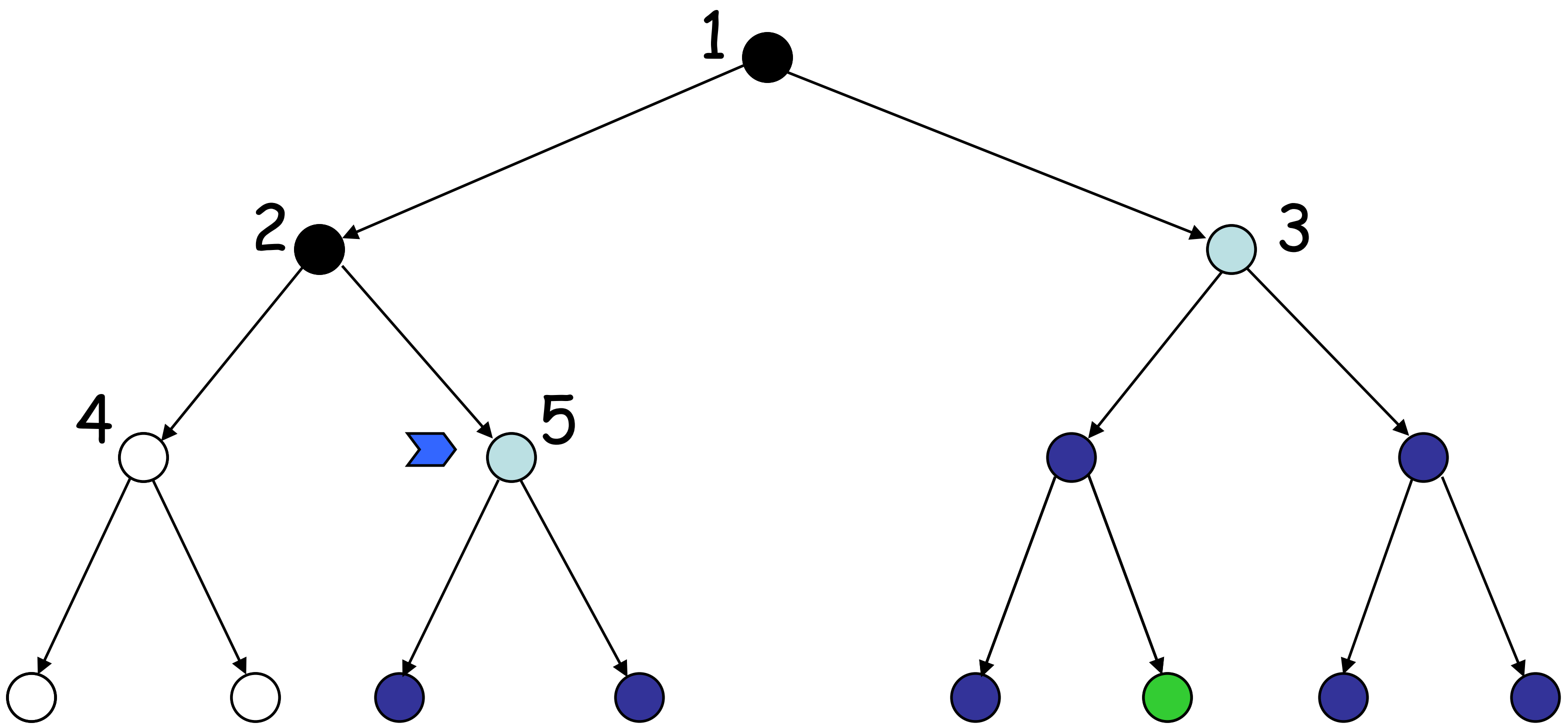
Depth-First Strategy

New nodes are inserted at the front of FRINGE



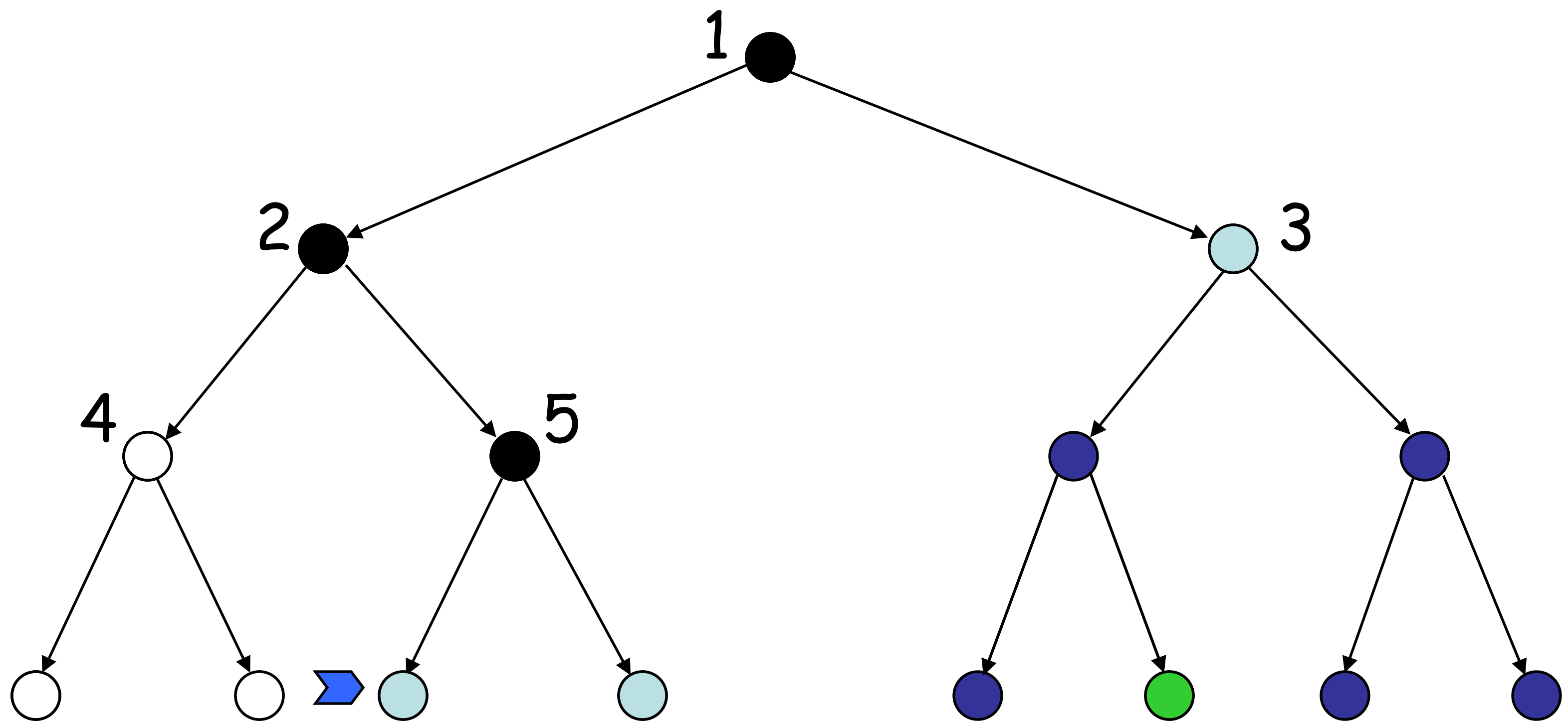
Depth-First Strategy

New nodes are inserted at the front of FRINGE



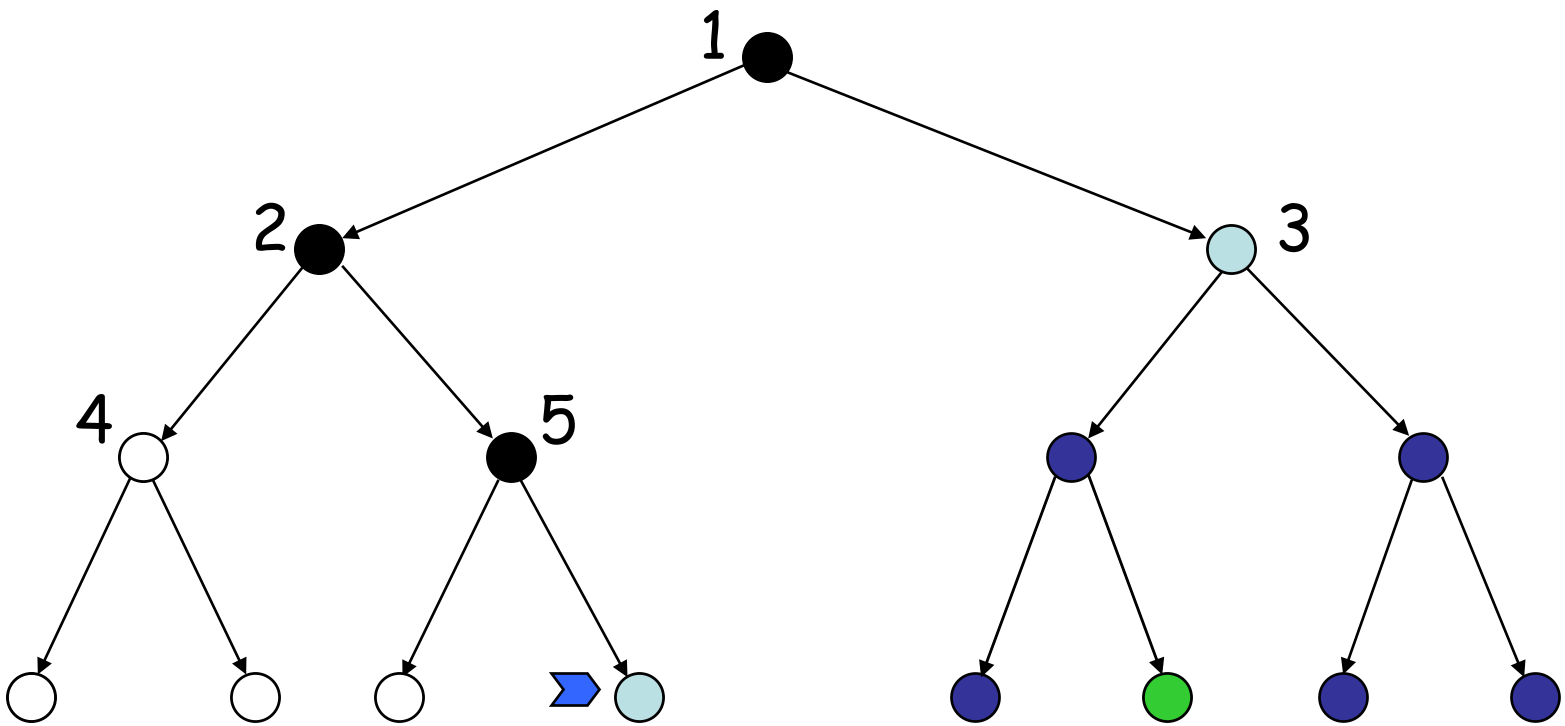
Depth-First Strategy

New nodes are inserted at the front of FRINGE



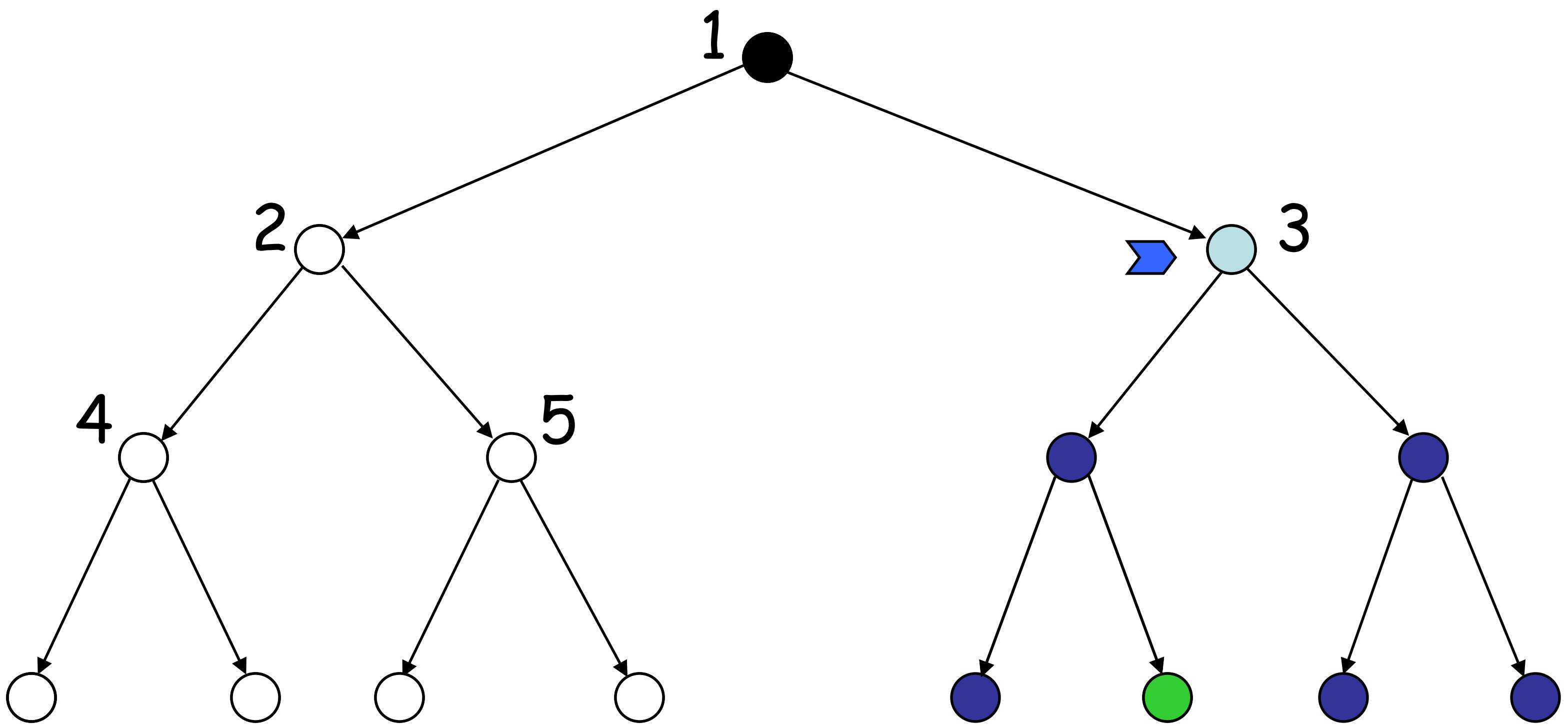
Depth-First Strategy

New nodes are inserted at the front of FRINGE



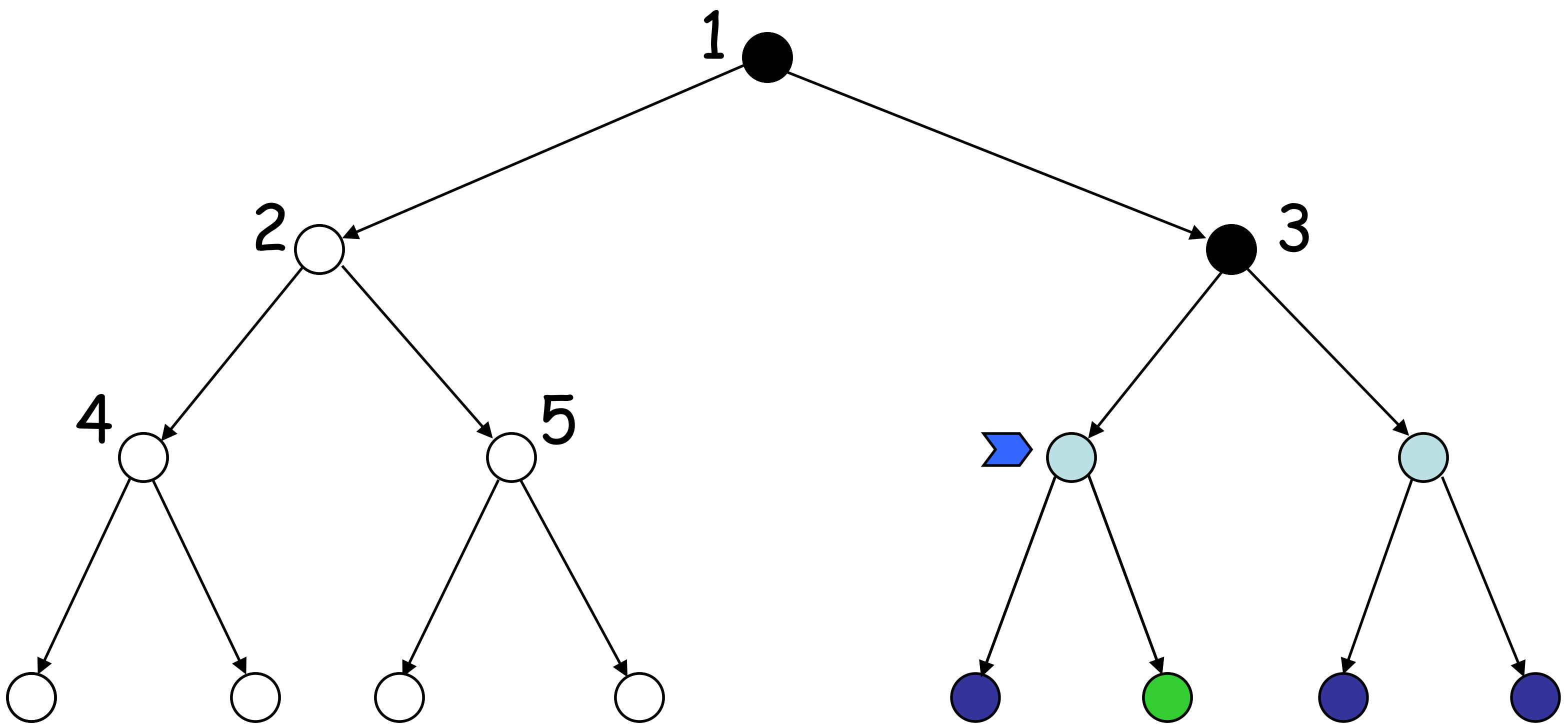
Depth-First Strategy

New nodes are inserted at the front of FRINGE



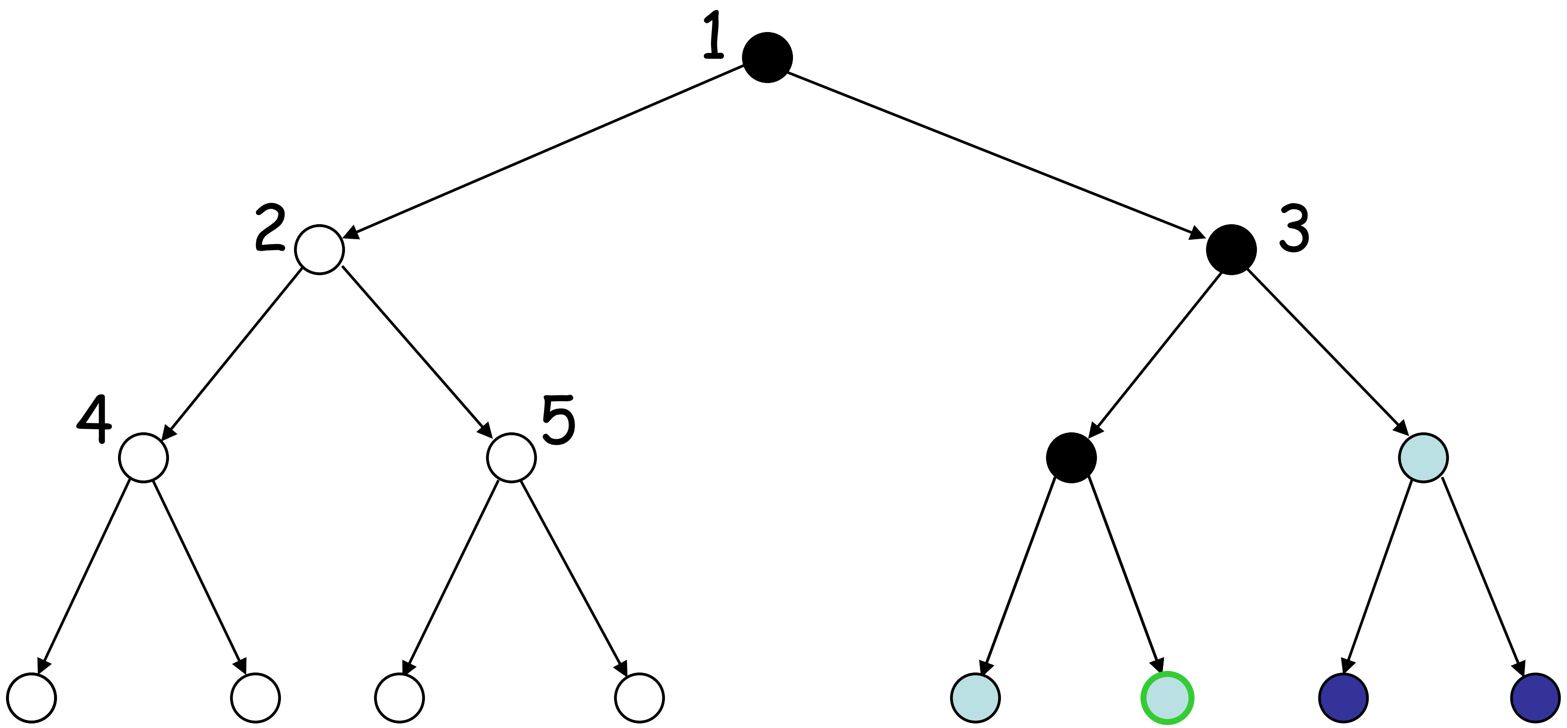
Depth-First Strategy

New nodes are inserted at the front of FRINGE



Depth-First Strategy

New nodes are inserted at the front of FRINGE



Evaluation

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

Evaluation

- 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 $O(bm)$ [or $O(m)$]

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

Evaluation

- Space complexity: it needs to store a single path along with the remaining unexpanded sibling nodes for each node on the path
- DFS requires a storage for only O(bm) nodes
- Can we further reduced it to O(m)?

Evaluation

- A variant of DFS: **backtracking search**

- One successor is generated at a time rather than all successors
- Each partially expanded node remembers which successor to generate next.
- Here, only O(m) memory is needed.
- We must able to go back to undo each modification when we go back to generate the next successor

Depth-Limited Search

- Depth-first with depth cutoff k (depth at which nodes are not expanded)
- Failure of DFS in infinite state space can be avoided by fixing a predetermined depth limit k
- Three possible outcomes:
  - Solution
  - Failure (no solution)
  - Cutoff (no solution within cutoff)

Evaluation

- Time complexity: $O(b^k)$ and space complexity: $O(bk)$
- Introduce incompleteness if $k<d$ and the shallowest goal node is not within depth limit
- Non optimal if we choose $k>d$

Iterative Deepening Search

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

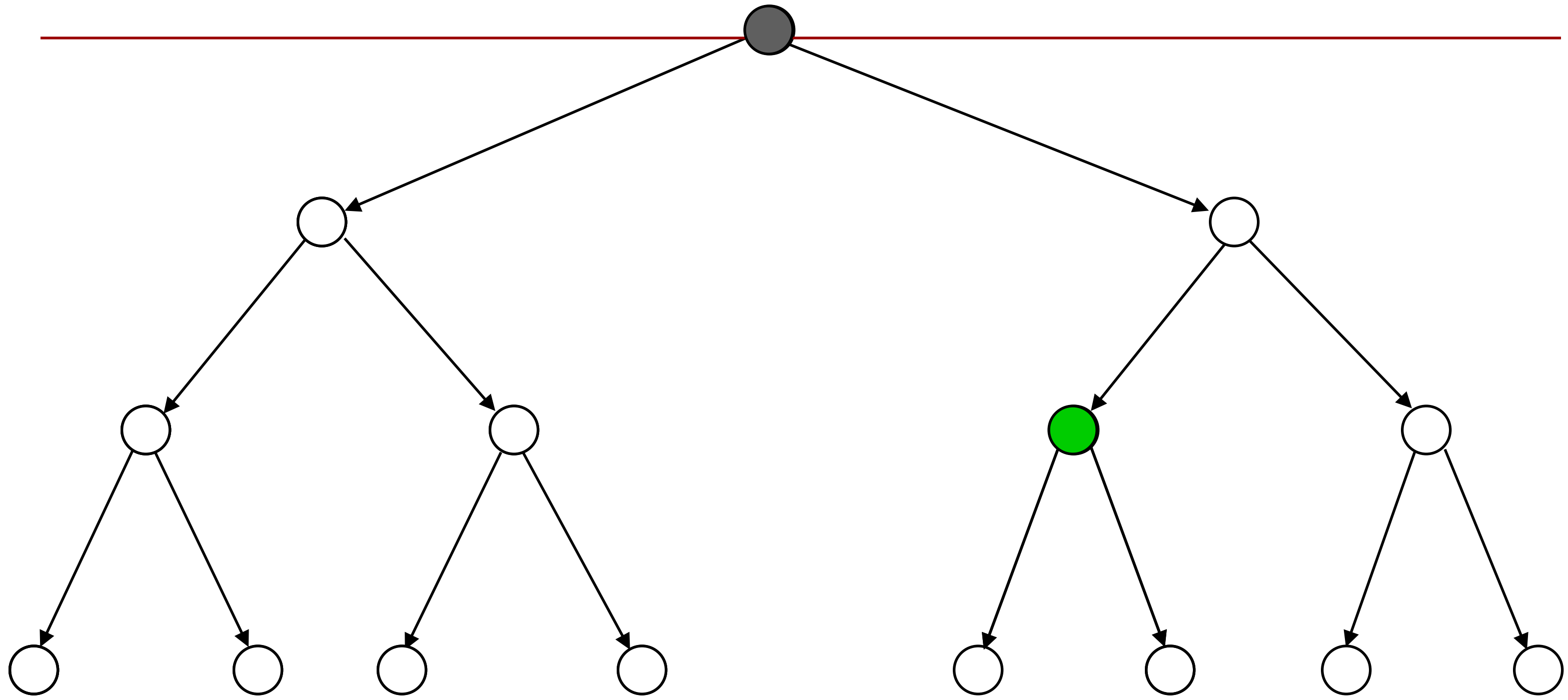
IDS

For k = 0, 1, 2, … do:

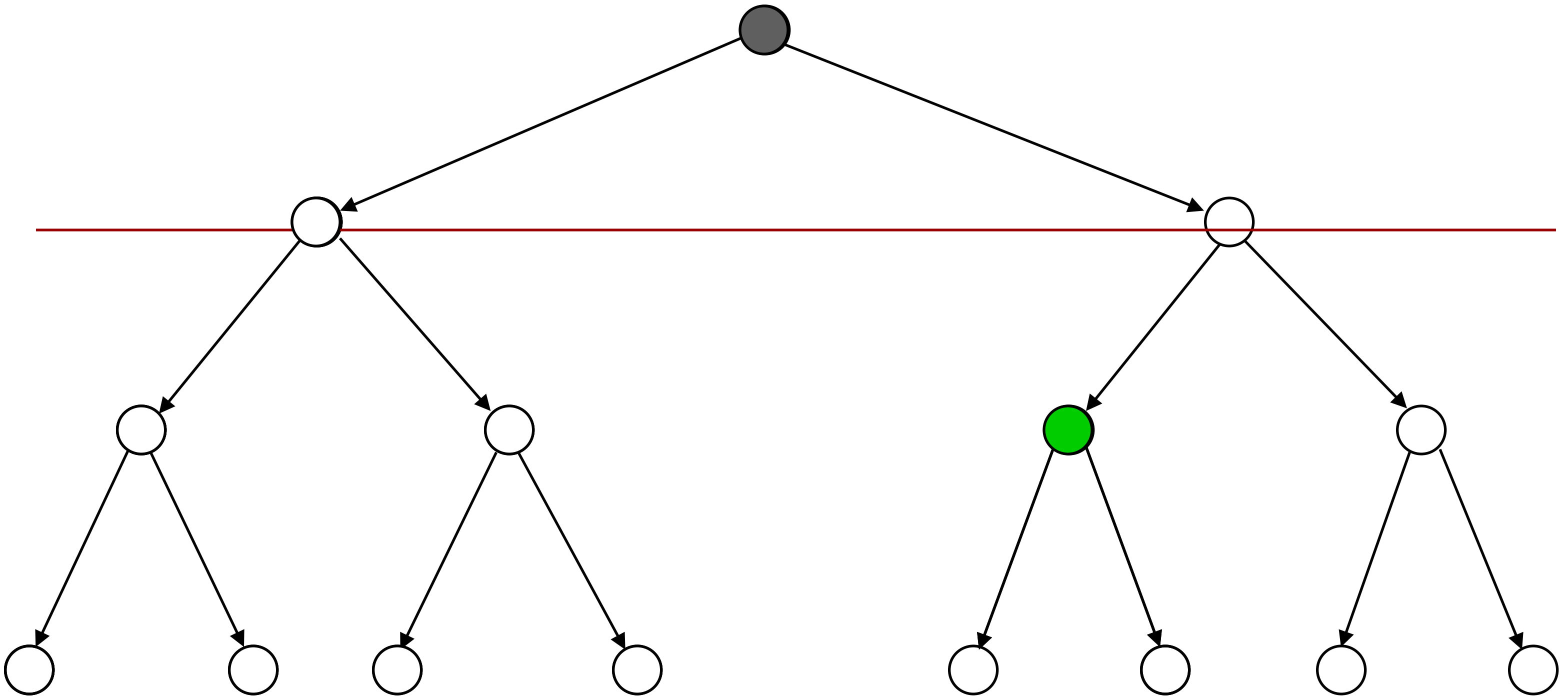
  Perform depth-first search with depth cutoff k

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

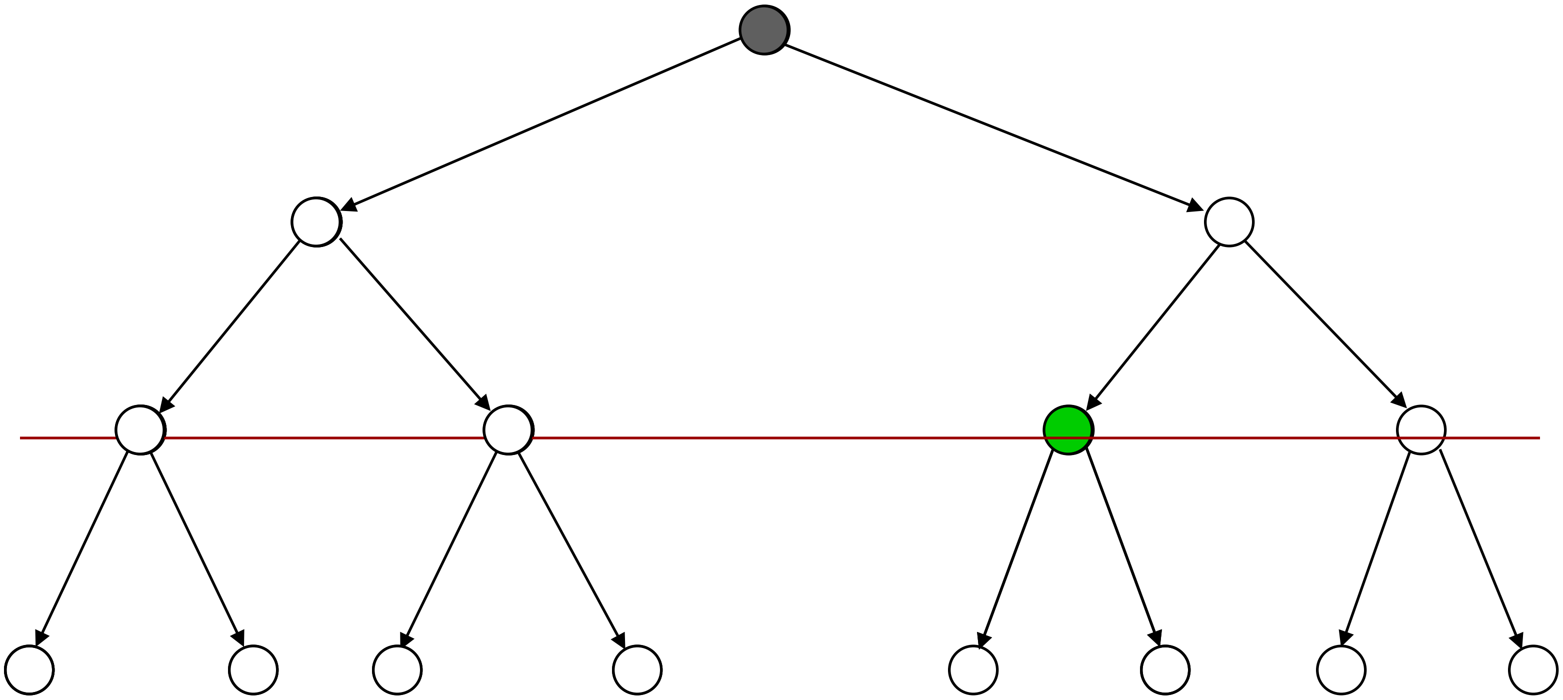
Iterative Deepening



Iterative Deepening



Iterative Deepening



Performance

- Iterative deepening search is:
  - Complete
  - Optimal if step cost =1
- Time complexity is:
  $(d+1)(1) + db + (d-1)b^2 + \ldots + (1)\, b^d = O(b^d)$
- Space complexity is: $O(bd)$ or $O(d)$

Number of Generated Nodes (Breadth-First & Iterative Deepening)

d = 5 and b = 2

BF	ID
1	1 x 6 = 6
2	2 x 5 = 10
4	4 x 4 = 16
8	8 x 3 = 24
16	16 x 2 = 32
32	32 x 1 = 32
63	120

120/63 ~ 2

Number of Generated Nodes (Breadth-First & Iterative Deepening)

d = 5 and b = 10

BF	ID
1	6
10	50
100	400
1,000	3,000
10,000	20,000
100,000	100,000
111,111	123,456

123,456/111,111 ~ 1.111

Comparison of Strategies

- Infinite state space: number of states is potentially high. Branching factor (no of actions) may also be infinite (but not common). Here, the search tree is also infinite.
- Finite state space may also lead to the infinite search tree.
- BFS is complete and optimal (same arc cost) for both the cases (if there is a shallowest goal node at depth d, it will eventually find it after expanding all the nodes at the depth shallower than d)
- DFS is likely to stuck into a wrong path. It is not complete and optimal even if the state space is infinite but the shallowest goal sate is in a depth which is much lower than infinity.
- DFS is complete only if the search tree is finite.

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

Outline of Search Algorithm(Ver-2)

1. Initialize: Set OPEN/FRINGE = $\{s_0\}$
2. Fail: If OPEN = { }, Terminate with Fail
3. Select: Select a state/node, n, from OPEN
4. Terminate: If n $\in$ G, terminate with success
5. Expand: Generate the successors of n using successor function and insert them in OPEN
6. Loop: Go To Step 2.

Revisited States

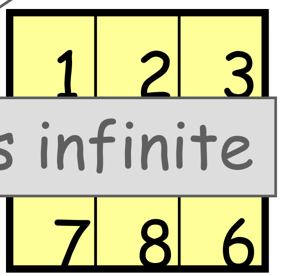
No

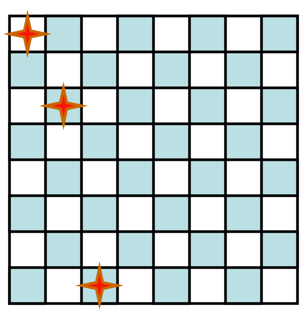
Few

Many

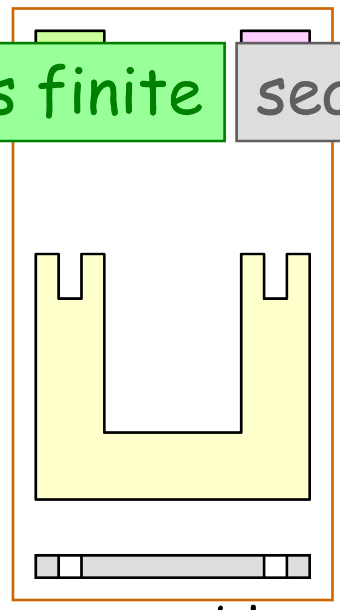
search tree is finite

search tree is infinite

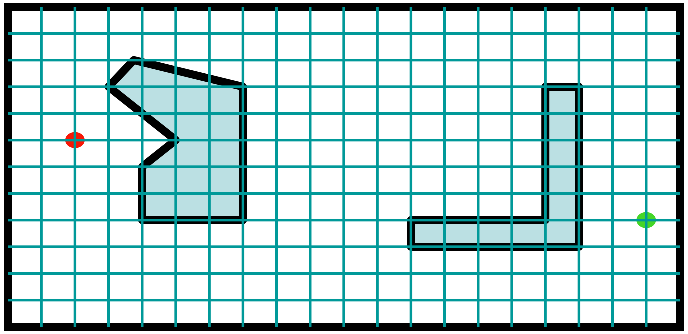




8-queens



assembly planning



8-puzzle and robot navigation

Outline of Search Algorithm(Ver-3)

1. Initialize: Set OPEN = $\{s_0\}$, **CLOSED = { }**
2. Fail: If OPEN = { }, Terminate with failure
3. Select: Select a state, n, from OPEN

   and save n in CLOSED
4. Terminate: If n ∈ G, terminate with success
5. Expand: Generate the successors of n using successor function

   For each successor, m, insert m in OPEN

   only if m $\notin$[OPEN ∪ CLOSED]
6. Loop: Go To Step 2

Uniform-Cost Search (UCS)

- BFS can generate the shallowest goal node.
- However, the shallowest goal node is not necessarily the optimal one.
- BFS is optimal when all actions have the same cost (commonly).
- **Problem:** BFS always expands the shallowest unexpanded node .
- We need a search strategy which is optimal from any cost.
- Instead of expanding the shallowest goal node, **UCS** expands a node n with lowest path cost g(n)

Uniform-Cost Search

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

  g(n) = Σ costs of arcs

  w(n,m)= arc cost between node n and m
- The goal is to generate a solution path of minimal cost
- The nodes n in the queue FRINGE are sorted in increasing g(n)