Problem Solving-Blind Search

Instructor: Dr. Karen Hand

AI a Modern Approach. Chapter 3.

Russell and Norvig:
Chapter 3, Sections 3.1 – 3.2

Solving Problems by Searching

- AI main purpose is to solve problems.
- The first problems that AI had to solve involved searching.
- The problem-solving approach has been applied to a vast array of task environments.
- We list some of the best known here, distinguishing between toy and real-world problems.

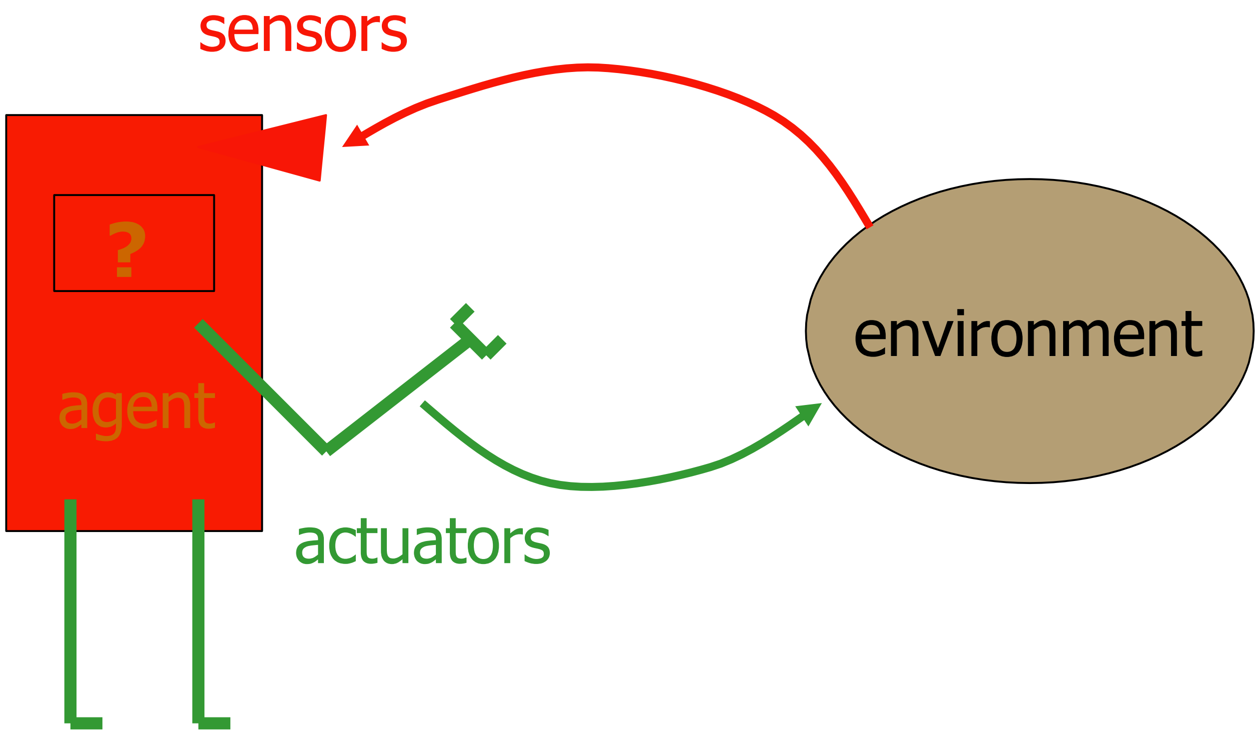
Toy problem

- Is intended to illustrate or exercise various problem-solving methods. It can be given a concise, exact description and hence is usable by different researchers to compare the performance of algorithms.

Real-world problem

- Is one whose solutions people actually care about. Such problems tend not to have a single agreed-upon description, but we can give the general flavor of their formulations.

Problem-Solving Agent



Problem Solving as Search

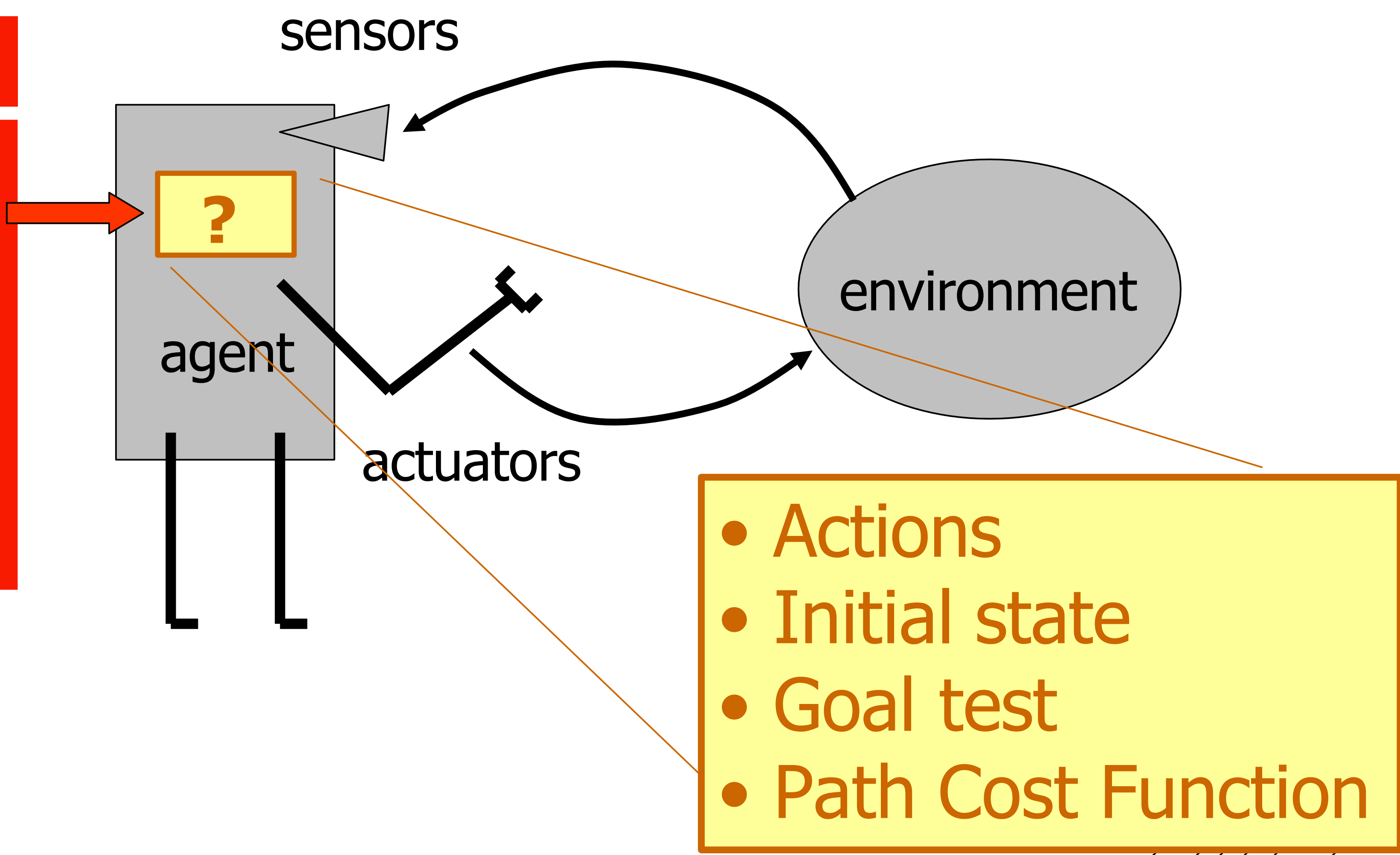
For a goal-based agent, what should it do when none of the actions it can currently perform results in a goal state?

One way to address the issues is to view goal-attainment as **problem solving**, and viewing that as a **search** through the space of possible solutions.

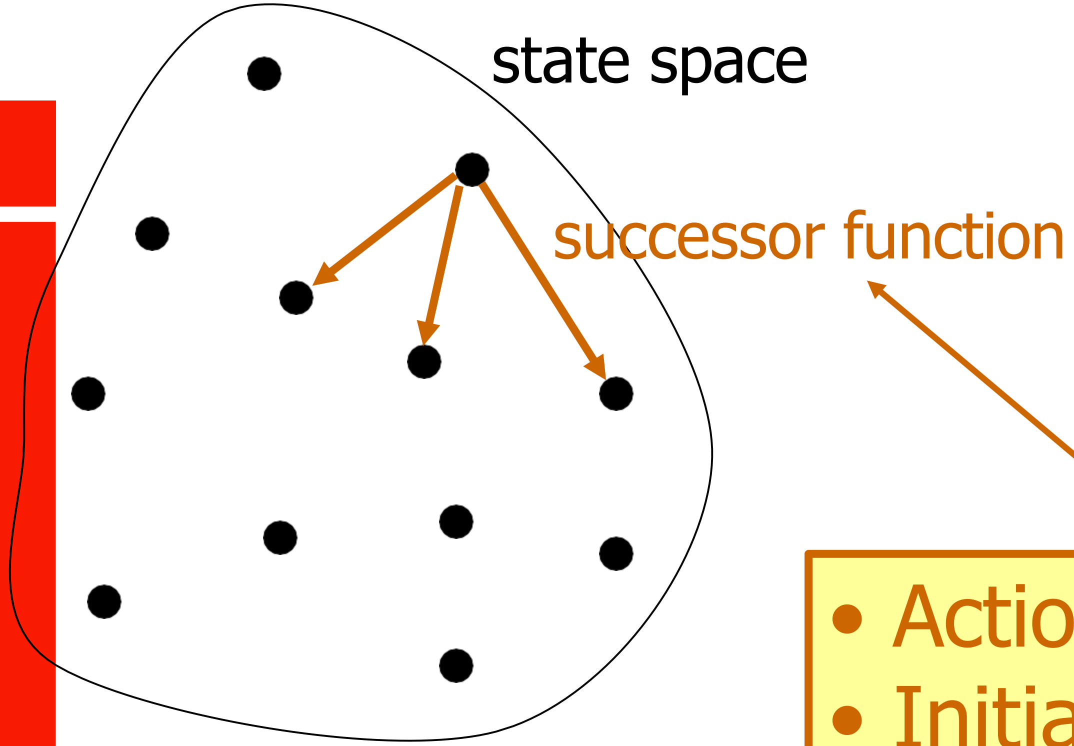
Search Problem Formulation

- Real-world environment → Abstraction
  - States within the solver are abstractions of states of the world the agent is actually in
  - Actions in the search space are abstractions of the agent's actions
  - Solutions map to sequences of real actions
  - The path cost function that assigns a numeric cost to action sequence.
- Without abstraction an agent would be swamped by the real world

Problem-Solving Agent

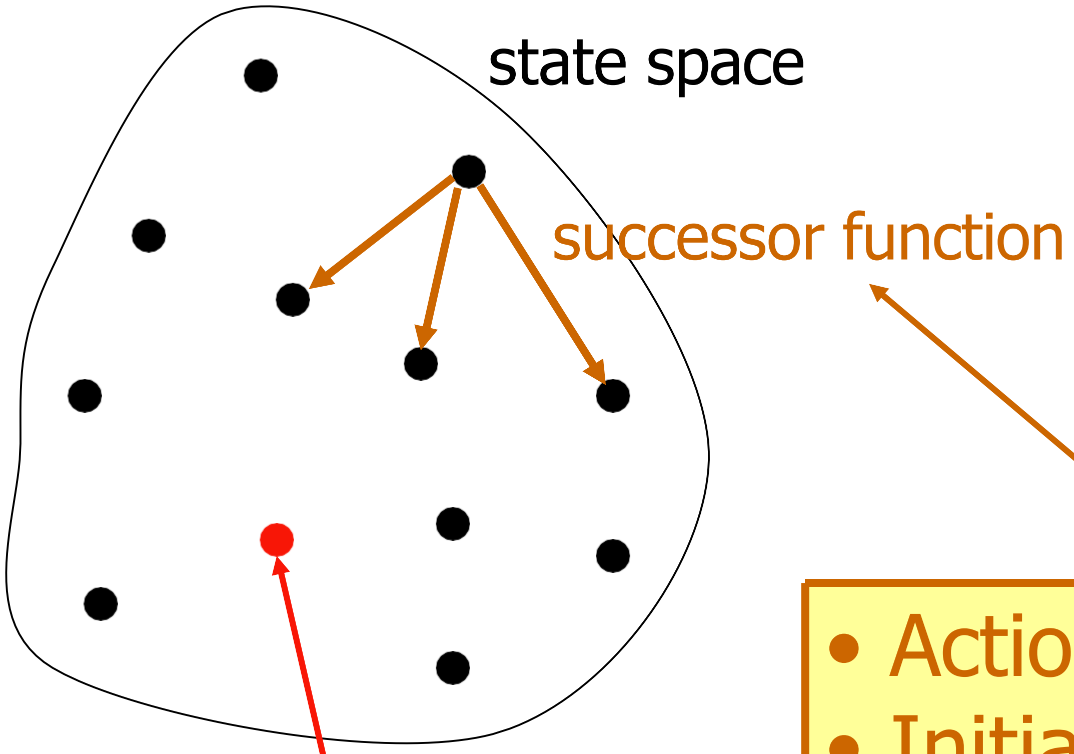


State Space and Successor Function



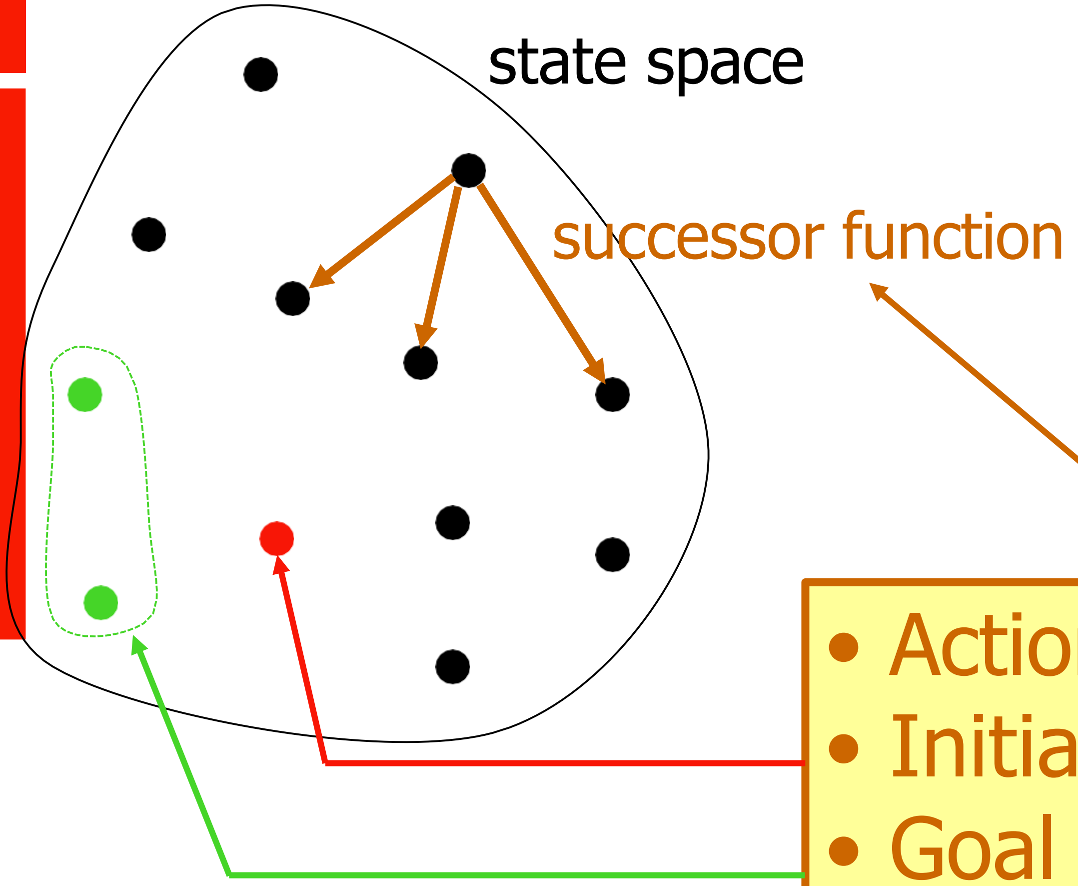
- Actions
- Initial state
- Goal test
- Path Cost Function

Initial State



- Actions
- Initial state
- Goal test
- Path Cost Function

Goal Test



- Actions
- Initial state
- Goal test
- Path Cost Function

Example: 8-puzzle

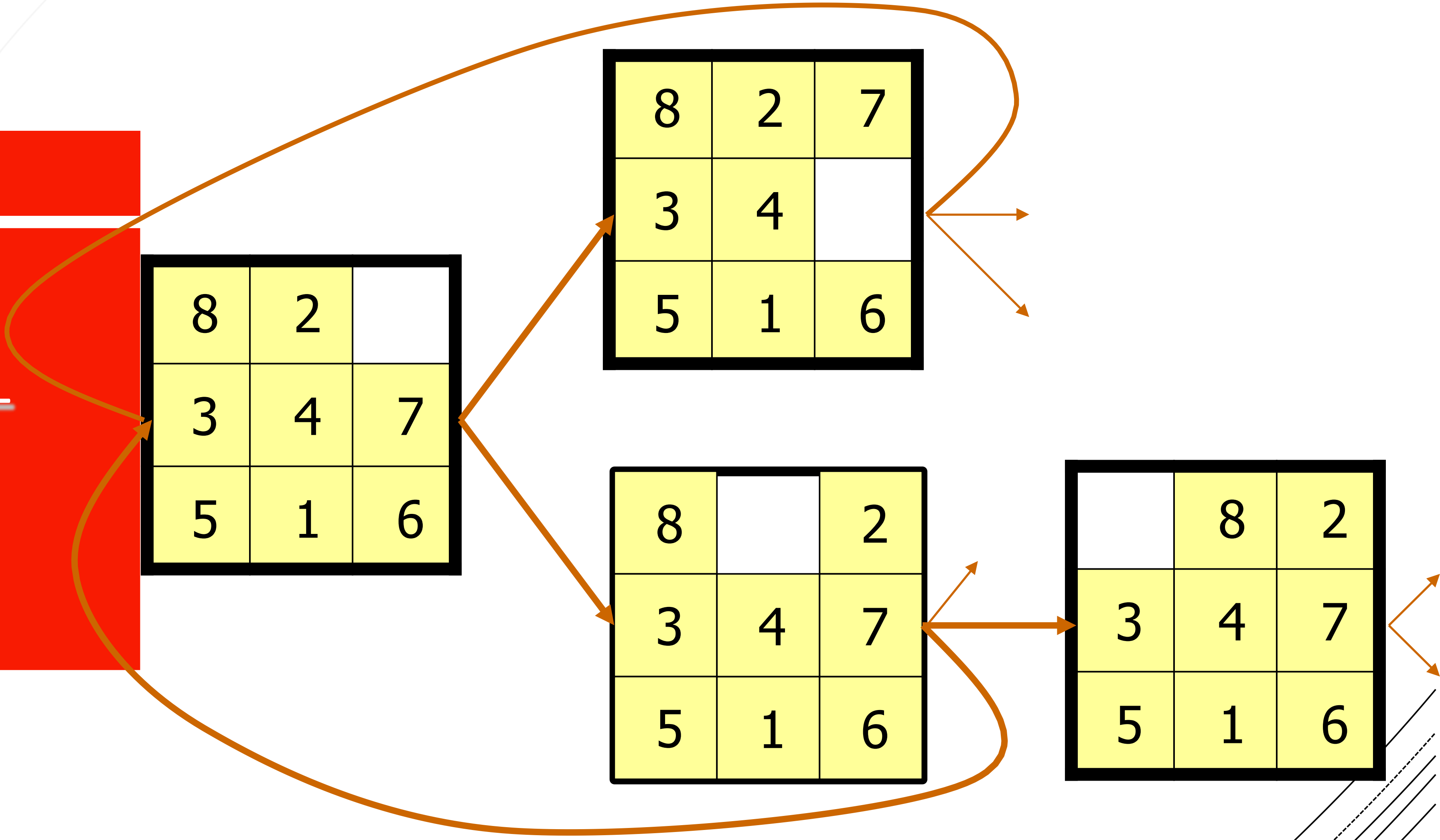
8	2	
3	4	7
5	1	6

Initial state

1	2	3
4	5	6
7	8	

Goal state

Example: 8-puzzle



Example: 8-puzzle

Size of the state space = 9!/2 = 181,440

15-puzzle → 6.5×10^{11}

24-puzzle → 5×10^{24}

0.018 sec

18 hours

16 billion years

10 millions states/sec

Search Problem

- State space
- Initial state
- Successor function
- Goal test
- Path cost

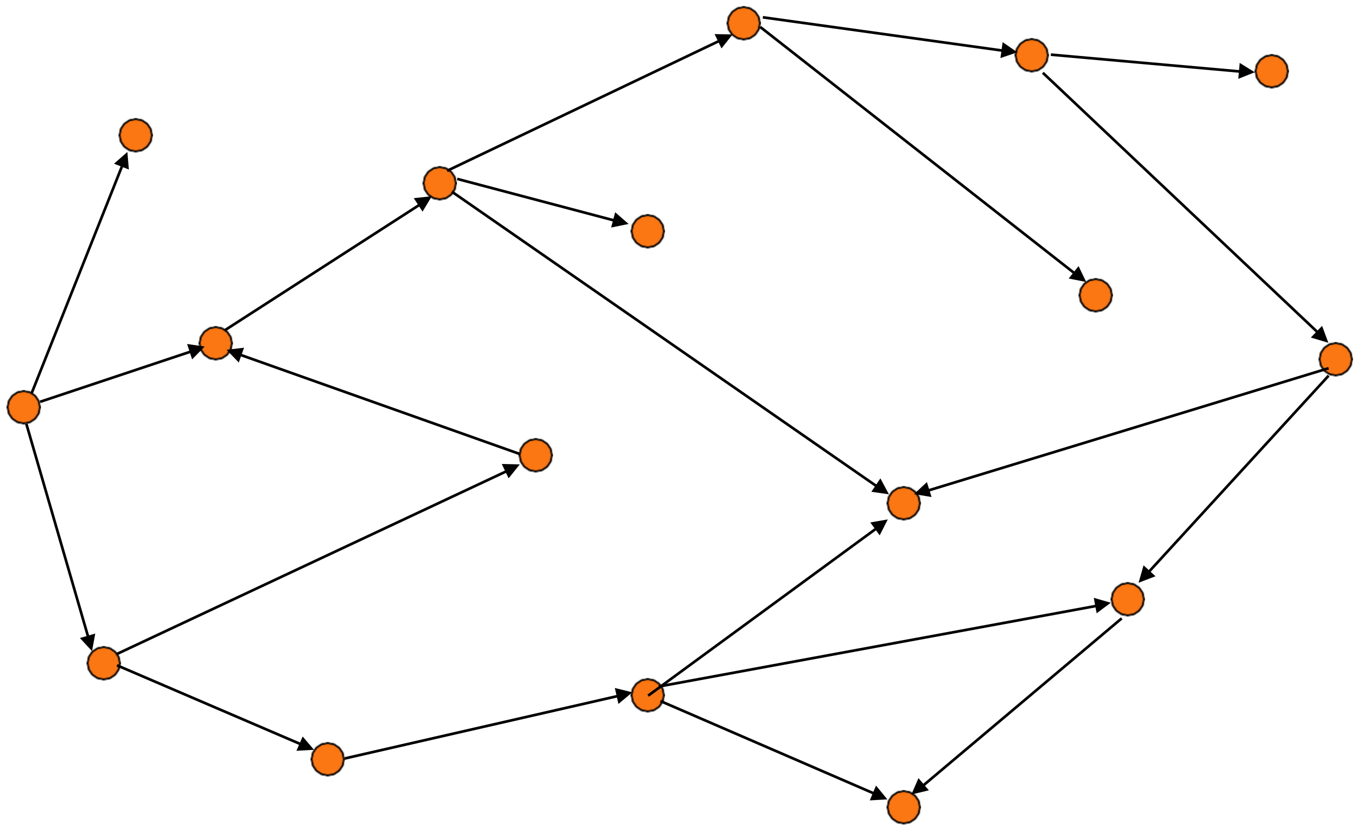
What is a Solution?

- A sequence of actions that when performed will transform the initial state into a goal state
- Or sometimes just the goal state itself, when getting there is trivial

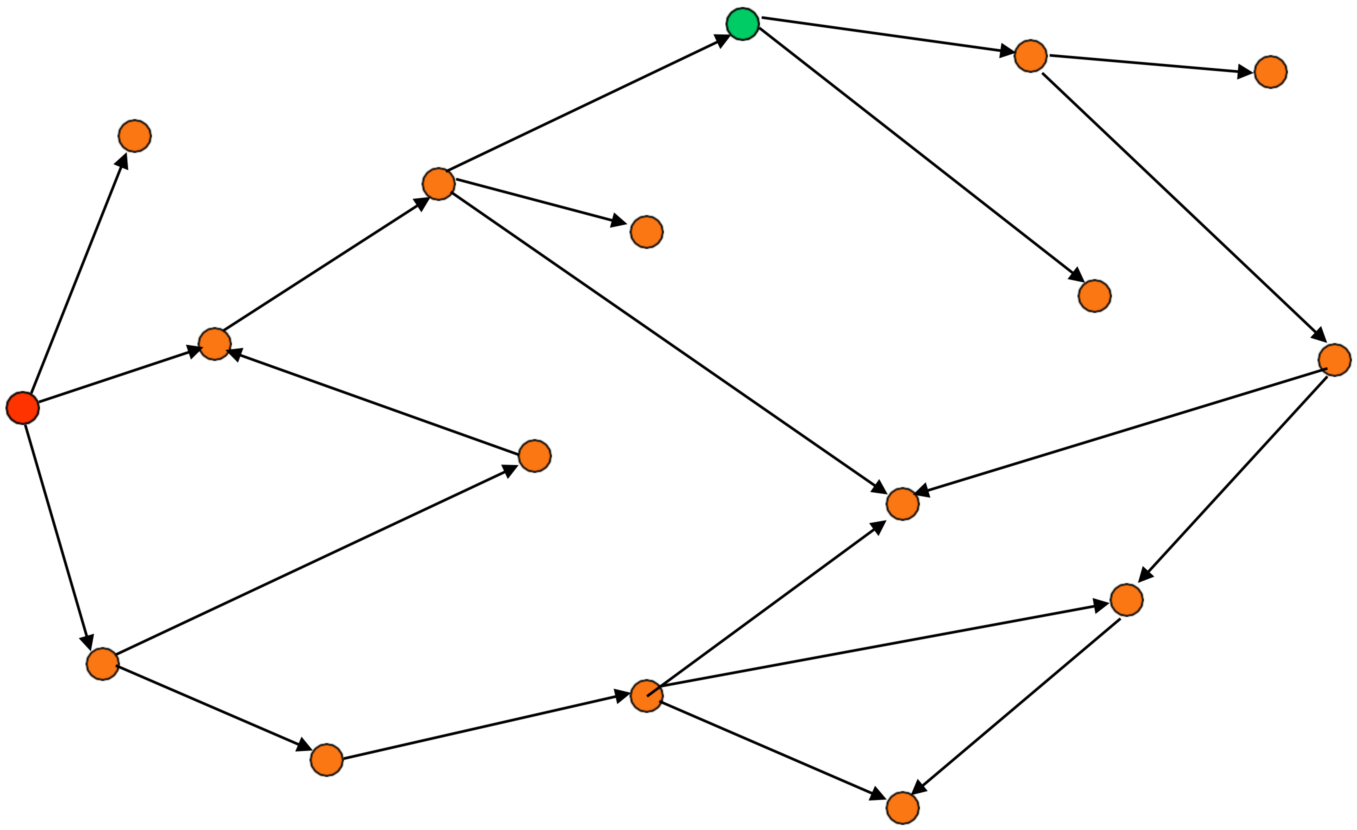
Assumptions in Basic Search

- The environment is static
- The environment is discretizable
- The environment is observable
- The actions are deterministic

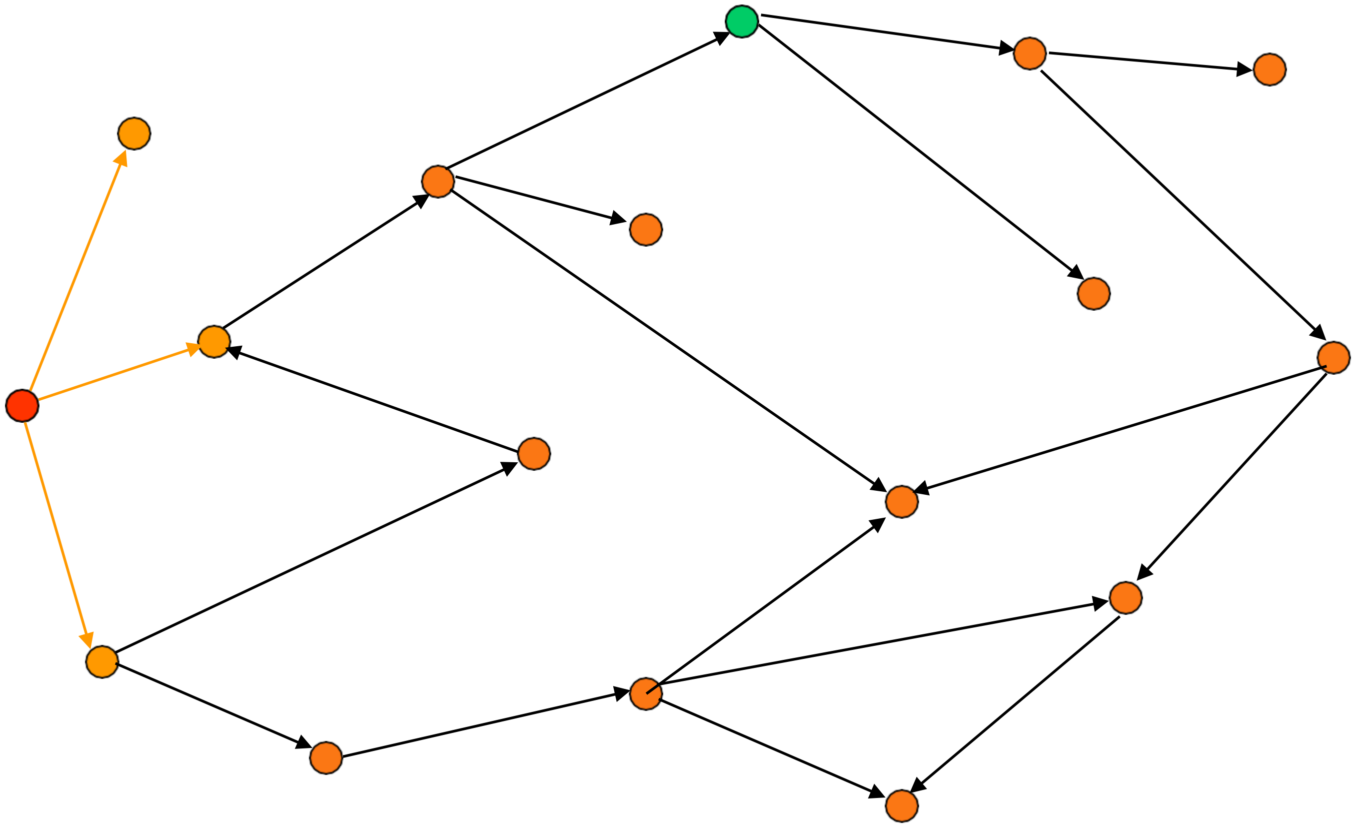
Search of State Space



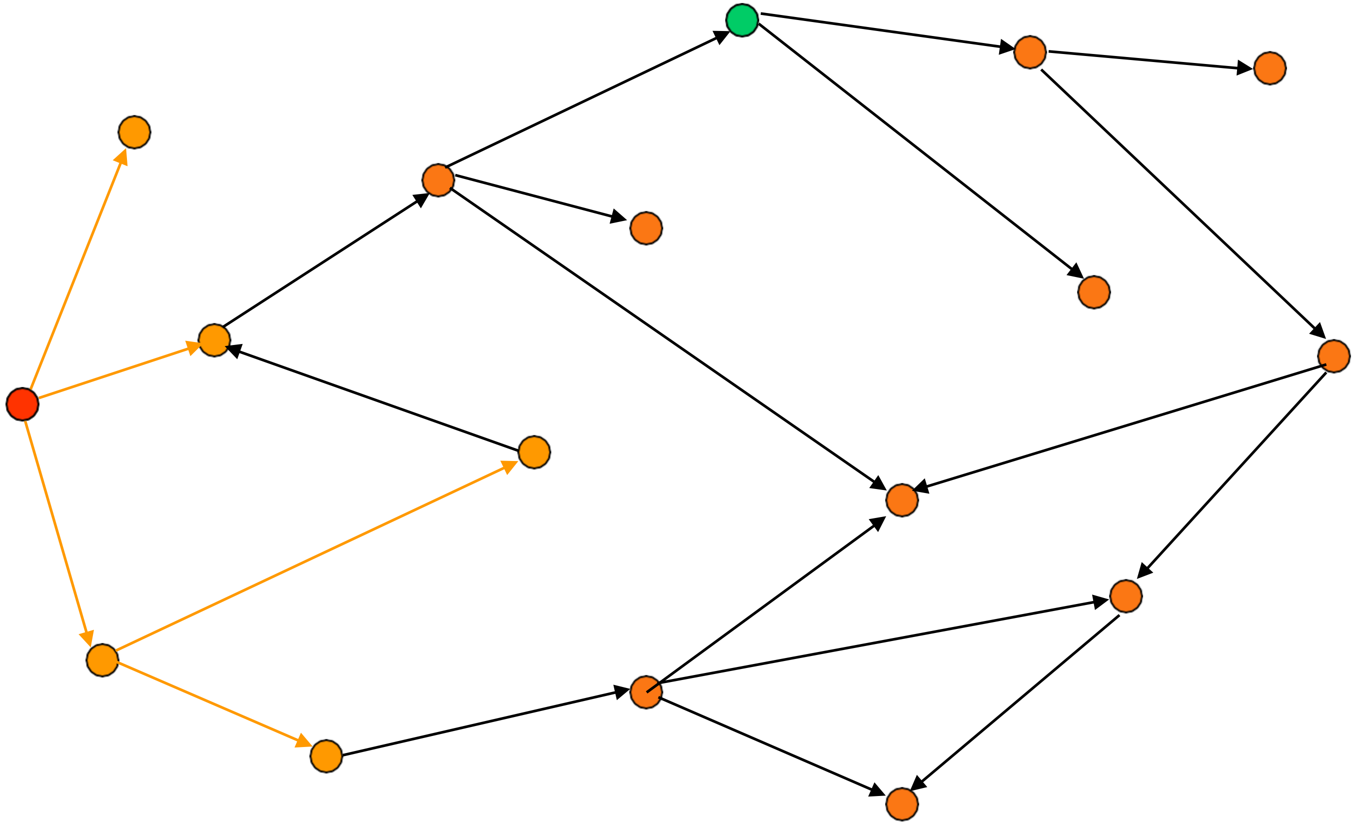
Search of State Space



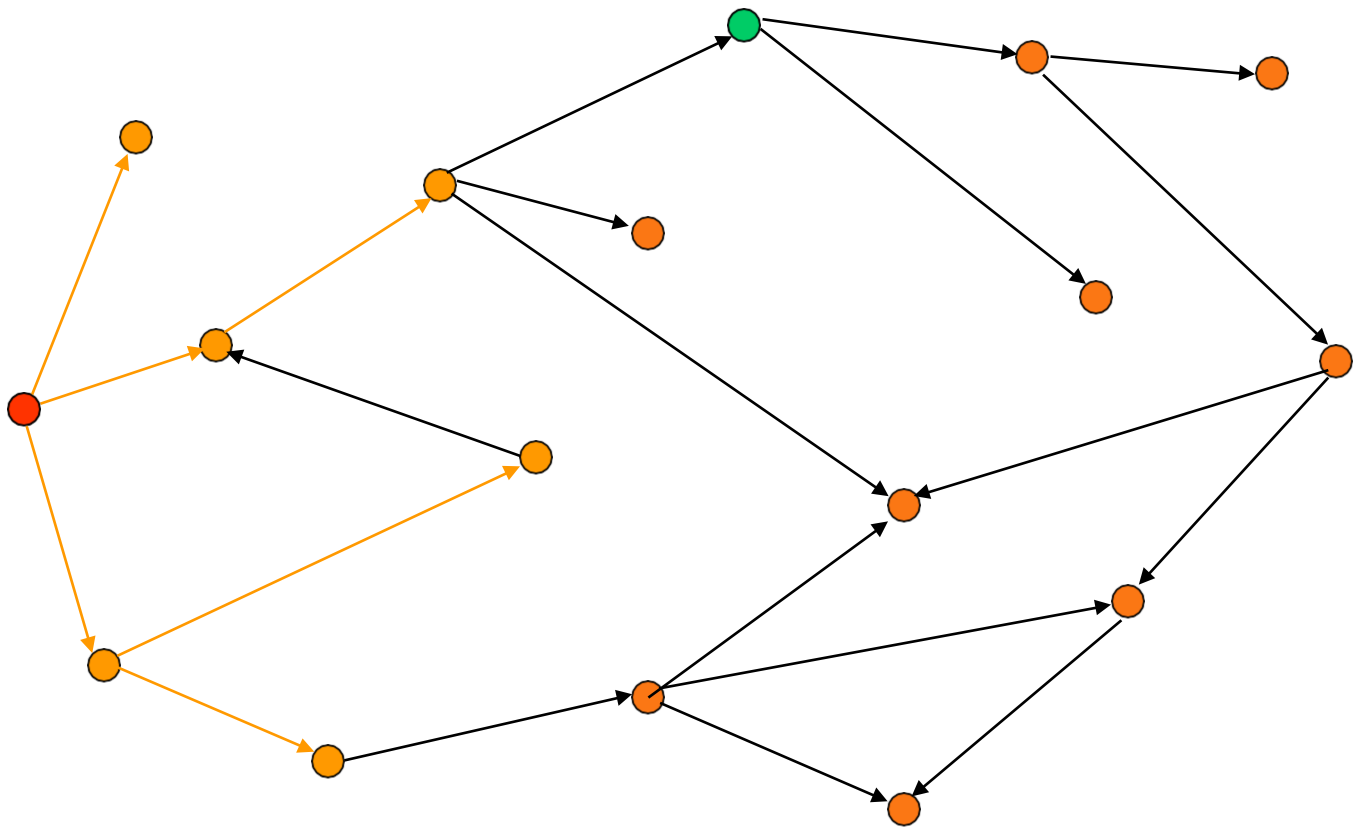
Search of State Space



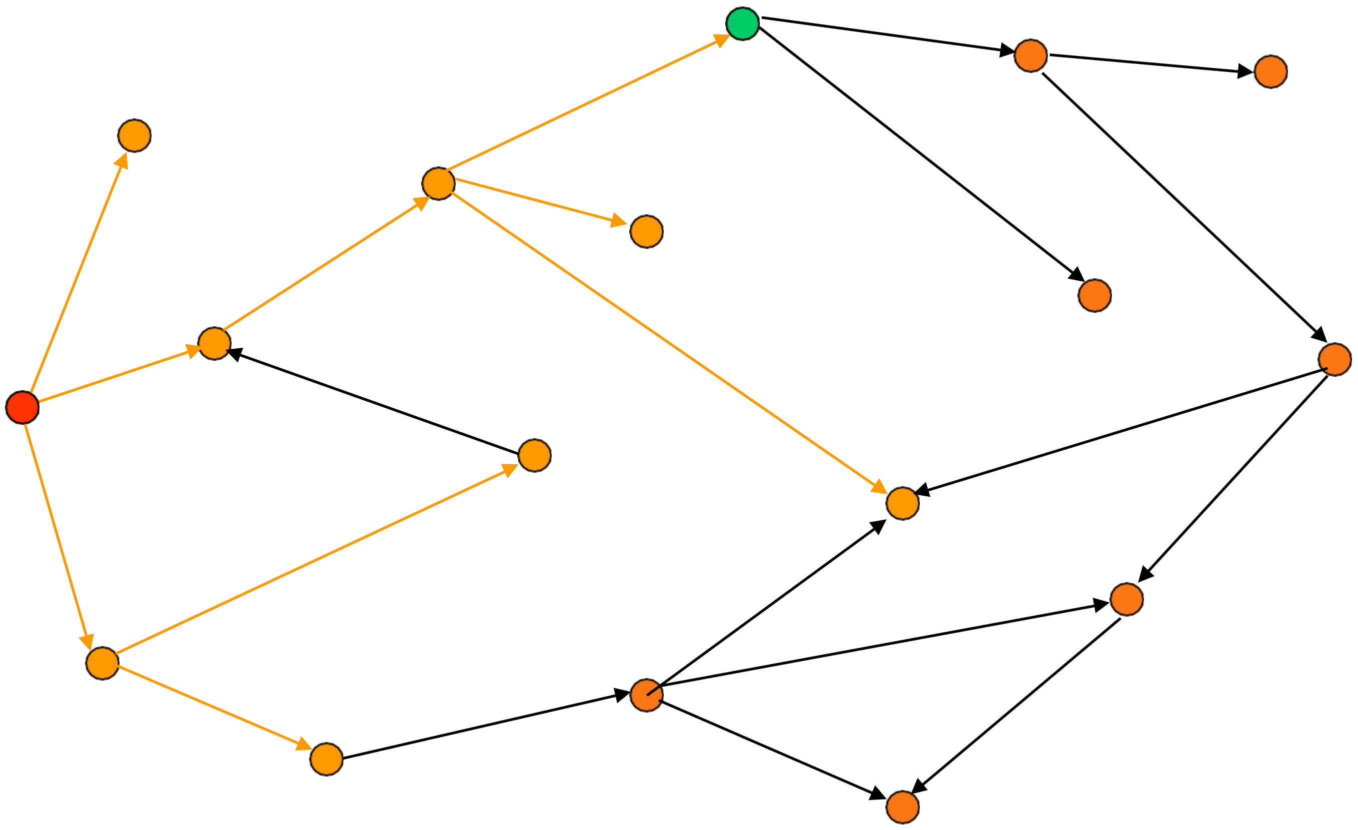
Search of State Space



Search of State Space



Search of State Space



→ search tree

Implementation : states vs. Nodes

- A state is (representation of) a physical configuration.
- A node is a data structure constituting part of a search tree includes **parent, children, depth, path cost**.
- States do not have parents, children, depth, or path cost.
- The expanding process creates new nodes, filling in the various fields and using the successor function of the problem to create the corresponding states.

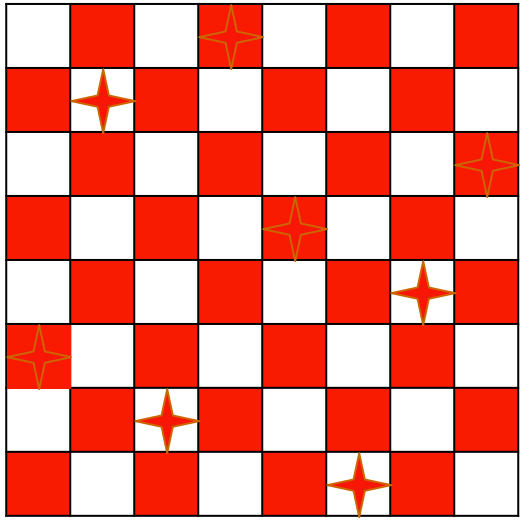
Simple Agent Algorithm

Problem-Solving-Agent

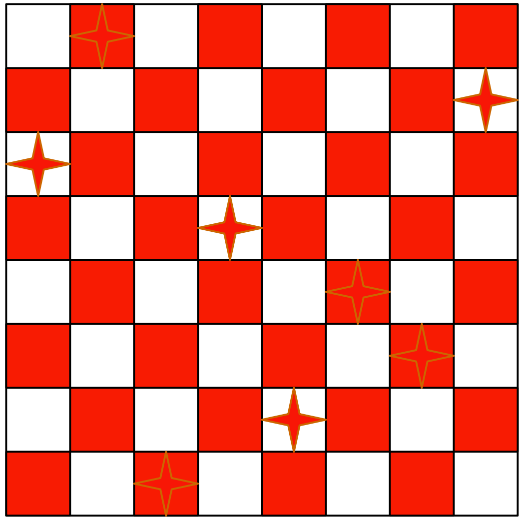
1. initial-state ← sense/read state
2. goal ← select/read goal
3. successor ← select/read action models
4. problem ← (initial-state, goal, successor)
5. solution ← search(problem)
6. perform(solution)

Example: 8-queens

Place 8 queens in a chessboard so that no two queens are in the same row, column, or diagonal.

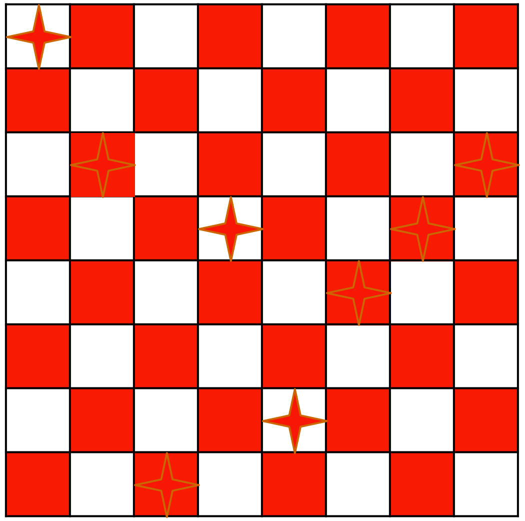


A solution



Not a solution

Example: 8-queens

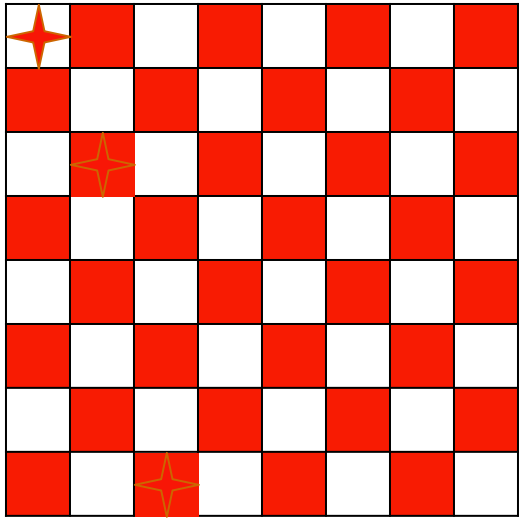


Formulation #1:

- States: any arrangement of 0 to 8 queens on the board
- Initial state: 0 queens on the board
- Successor function: add a queen in any square
- Goal test: 8 queens on the board, none attacked

→ 64^8 states with 8 queens

Example: 8-queens



→ 2,067 states

Formulation #2:

- States: any arrangement of k = 0 to 8 queens in the k leftmost columns with none attacked
- Initial state: 0 queens on the board
- Successor function: add a queen to any square in the leftmost empty column such that it is not attacked by any other queen
- Goal test: 8 queens on the board

Search Problem Variants

- One or several initial states
- One or several goal states
- The solution is the path or a goal node
  - In the 8-puzzle problem, it is the path to a goal node
  - In the 8-queen problem, it is a goal node
- Any, or the best, or all solutions

Important Parameters

8-puzzle → 181,440
15-puzzle → .65 x 10^{12}
24-puzzle → .5 x 10^{25}

- Number of states in state space

8-queens → 2,057
100-queens → 10^{52}

There exist techniques to solve N-queens problems efficiently!

Stating a problem as a search problem is not always a good idea!

Important Parameters

- Number of states in state space
- Size of memory needed to store a state
- Running time of the successor function

Applications

- Route finding: airline travel, telephone/computer networks
- Touring problems
- VLSI layout
- Automatic assembly sequencing
- Internet searching

Summary

- Problem-solving agent
- State space, successor function, search
- Examples: 8-puzzle, 8-queens, route finding, robot navigation, assembly planning
- Assumptions of basic search
- Important parameters

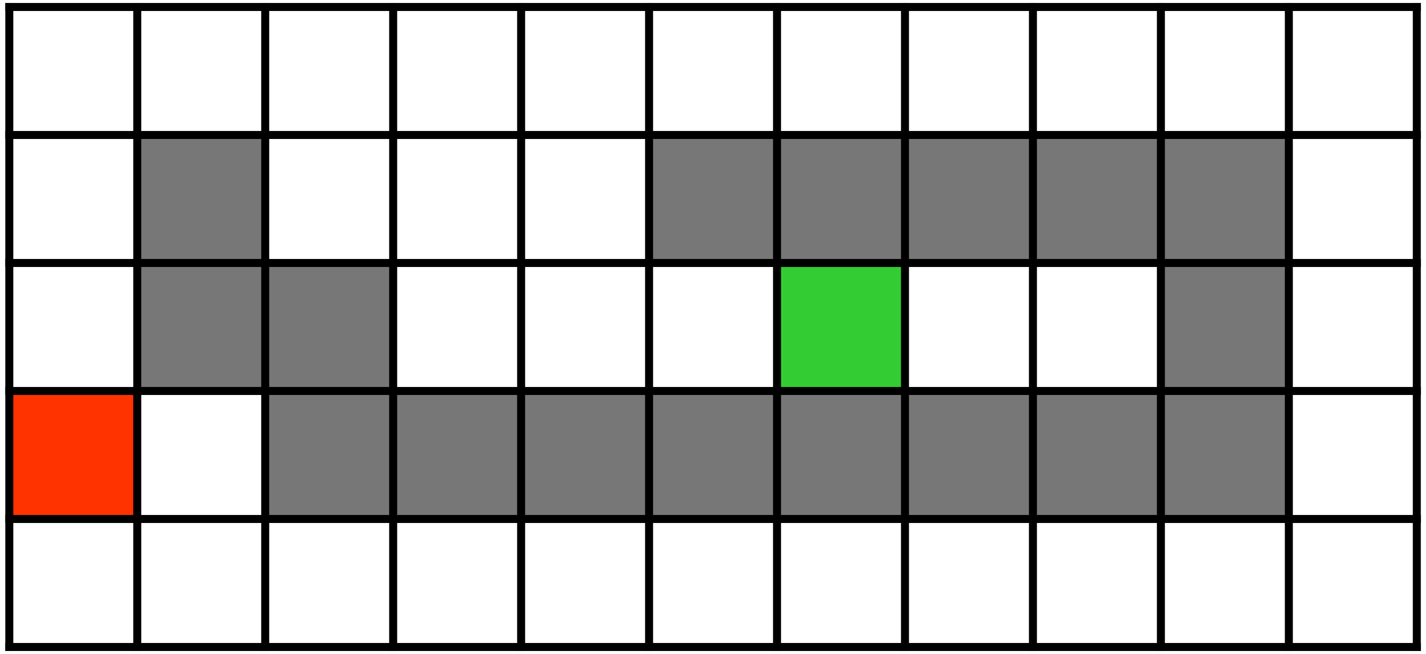
Acknowledgement

- Krishnaram Kenthapadi , CS121: Introduction to Artificial Intelligence, 2003.
- Jim Martin, CSCI 5582: Introduction to Artificial Intelligence, 2003.
- AI Instructor's Resource Page, http://aima.cs.berkeley.edu/instructors.html

Blind Search

Russell and Norvig:
Chapter 3, Sections 3.3 – 3.4

Robot Navigation

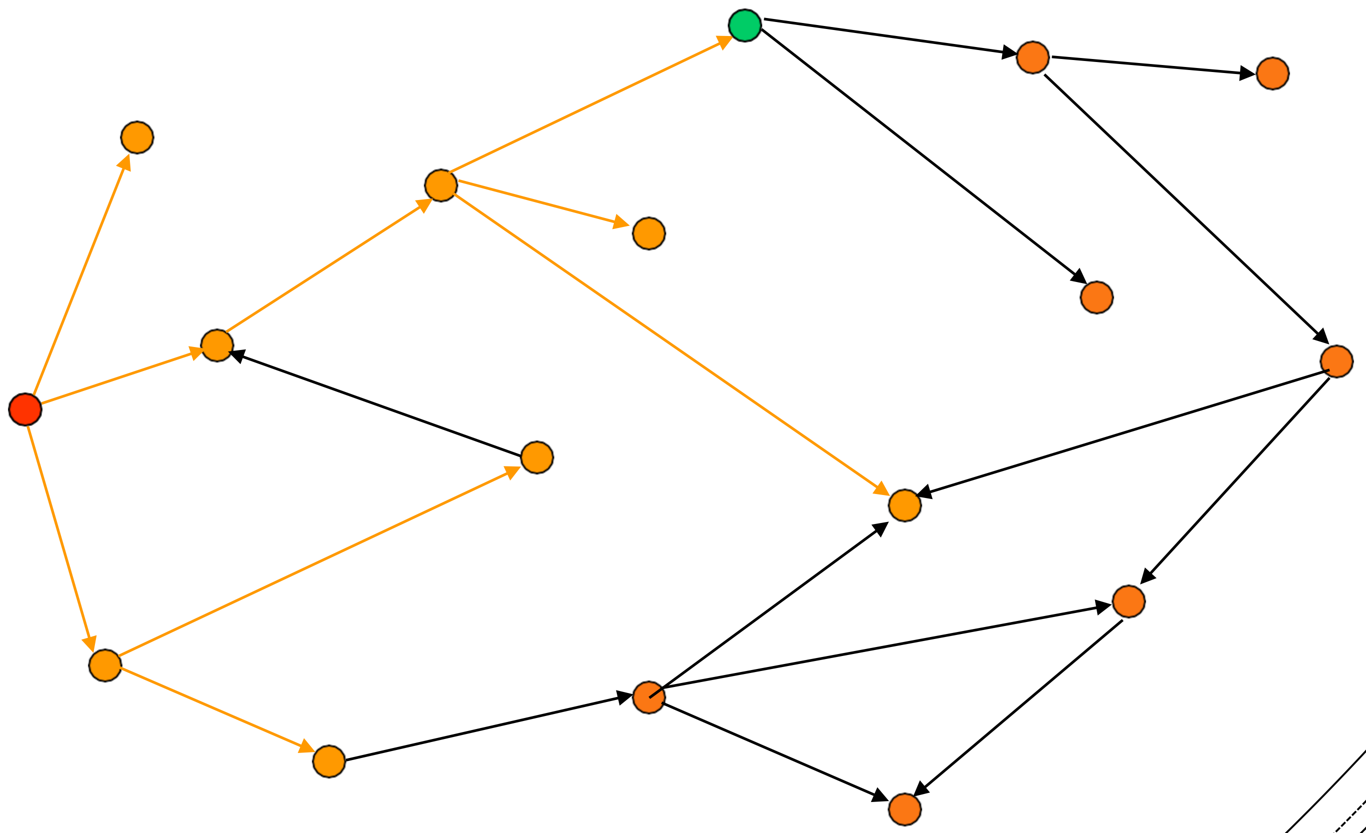


Simple Agent Algorithm

Problem-Solving-Agent

1. initial-state ← sense/read state
2. goal ← select/read goal
3. successor ← select/read action models
4. problem ← (initial-state, goal, successor)
5. solution ← search(problem)
6. perform(solution)

Search of State Space



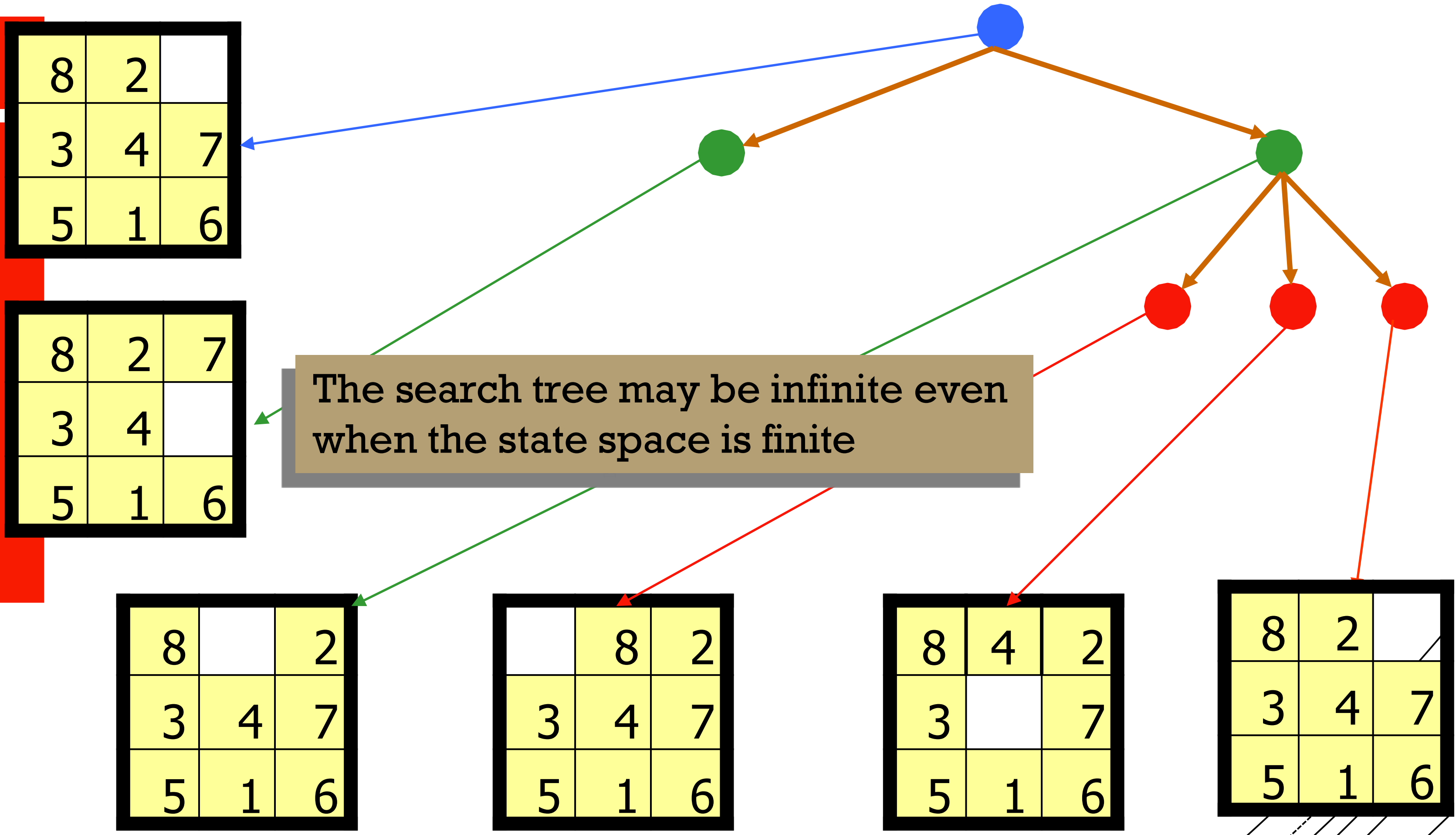
→ search tree

Basic Search Concepts

- Search tree
- Search node
- Node expansion

- Search strategy: At each stage it determines which node to expand

Search Nodes ≠ States



Node Data Structure

- STATE
- PARENT
- ACTION
- COST

If a state is too large, it may be preferable to only represent the initial state and (re-)generate the other states when needed

Fringe

- Set of search nodes that have not been expanded yet
- Implemented as a queue FRINGE
  - INSERT(node,FRINGE)
  - REMOVE(FRINGE)
- The ordering of the nodes in FRINGE defines the search strategy

Search Algorithm

1. If GOAL?(initial-state) then return initial-state
2. INSERT(initial-node,FRINGE)
3. Repeat:
   - If FRINGE is empty then return failure
   - n ← REMOVE(FRINGE)
   - s ← STATE(n)
   - For every state s’ in SUCCESSORS(s)
     - Create a node n’ as a successor of n
     - If GOAL?(s’) then return path or goal state
     - INSERT(n’,FRINGE)

Performance Measures

- Completeness
  Is the algorithm guaranteed to find a solution when there is one?
- Optimality
  Is this solution optimal?
- Time complexity
  How long does it take?
- Space complexity
  How much memory does it require?

Important Parameters

- Maximum number of successors of any state
  → branching factor b of the search tree

- Minimal length of a path in the state space between the initial and a goal state
  → depth d of the shallowest goal node in the search tree

- maximum length of a path in the state space
  → m

Important Remark

- Some problems formulated as <u>search problems</u> are NP-hard problems (e.g., $(n^2\text{-}1)$-puzzle).
- We cannot expect to solve such a problem in less than exponential time in the worst-case.
- But we can nevertheless strive to solve as many instances of the problem as possible.

Blind Strategies

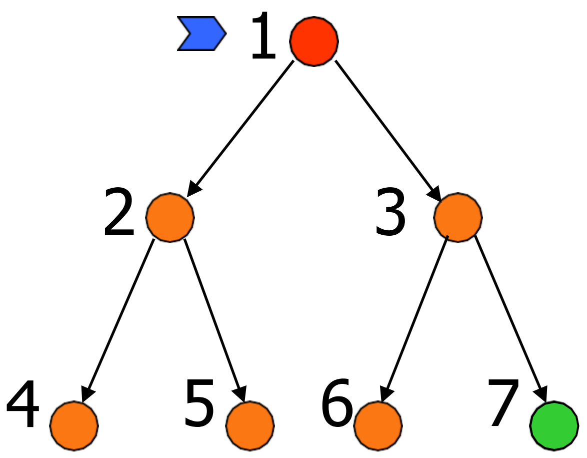
- Breadth-first
  - Bidirectional
- Depth-first
  - Depth-limited
  - Iterative deepening
- Uniform-Cost

Step cost = 1

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

Breadth-First Strategy

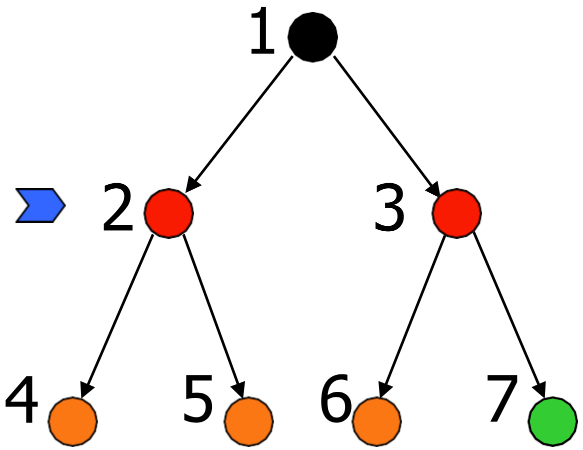
New nodes are inserted at the end of FRINGE



FRINGE = (1)

Breadth-First Strategy

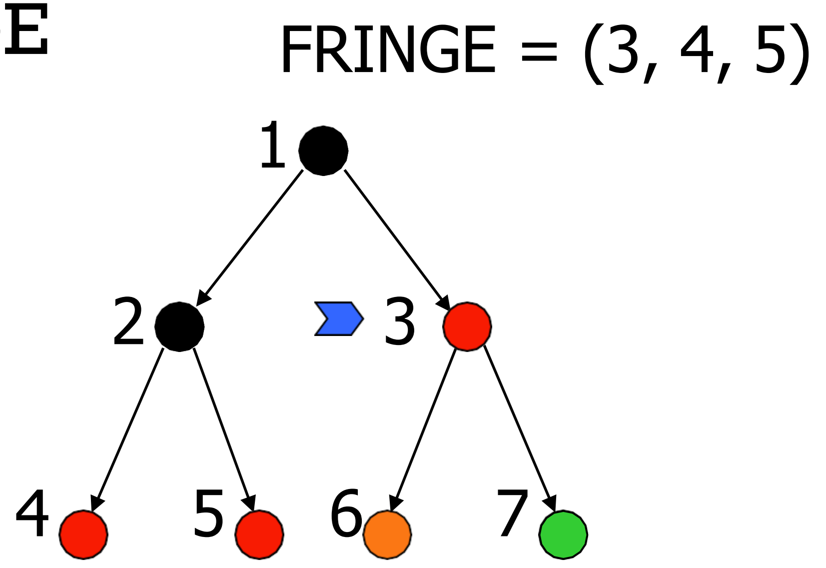
New nodes are inserted at the end of FRINGE



FRINGE = (2, 3)

Breadth-First Strategy

New nodes are inserted at the end of FRINGE

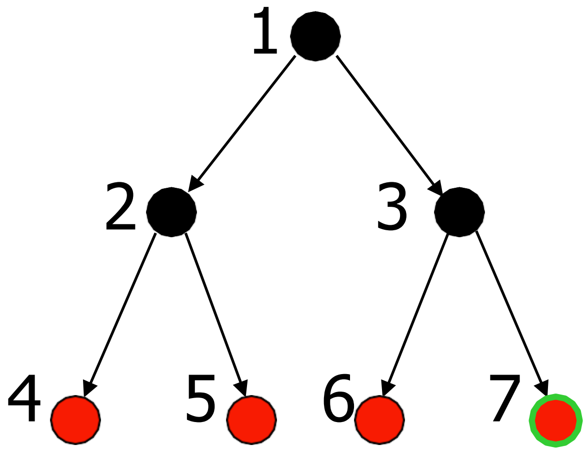


Breadth-First Strategy

New nodes are inserted at the end of FRINGE

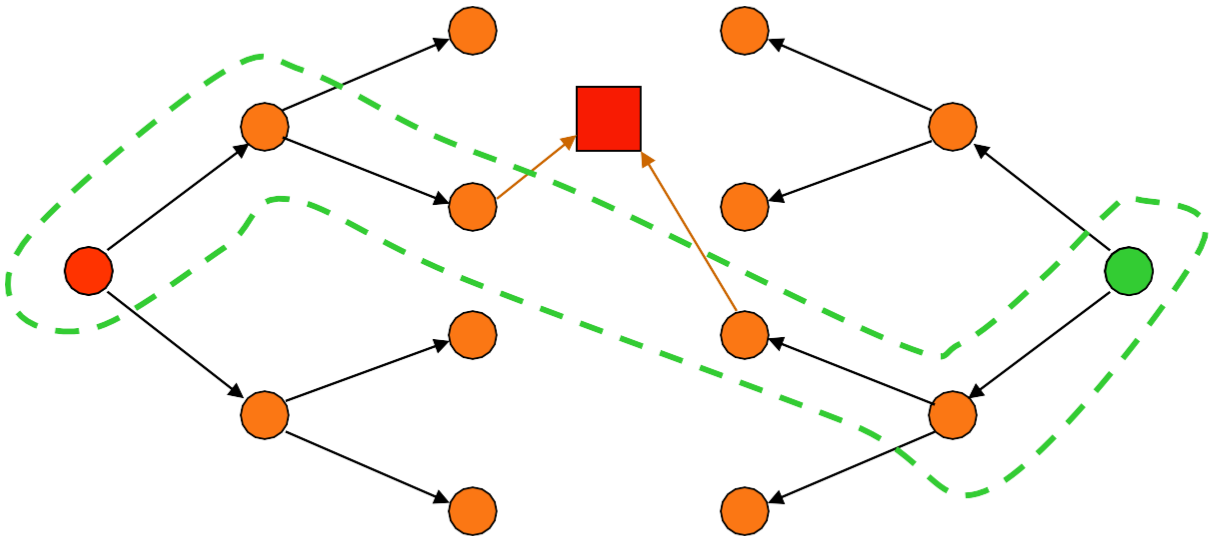
FRINGE = (4, 5, 6, 7)

[Node 7 (being the goal) could not actually inserted into the fringe; shown here for clarity]



Bidirectional Strategy

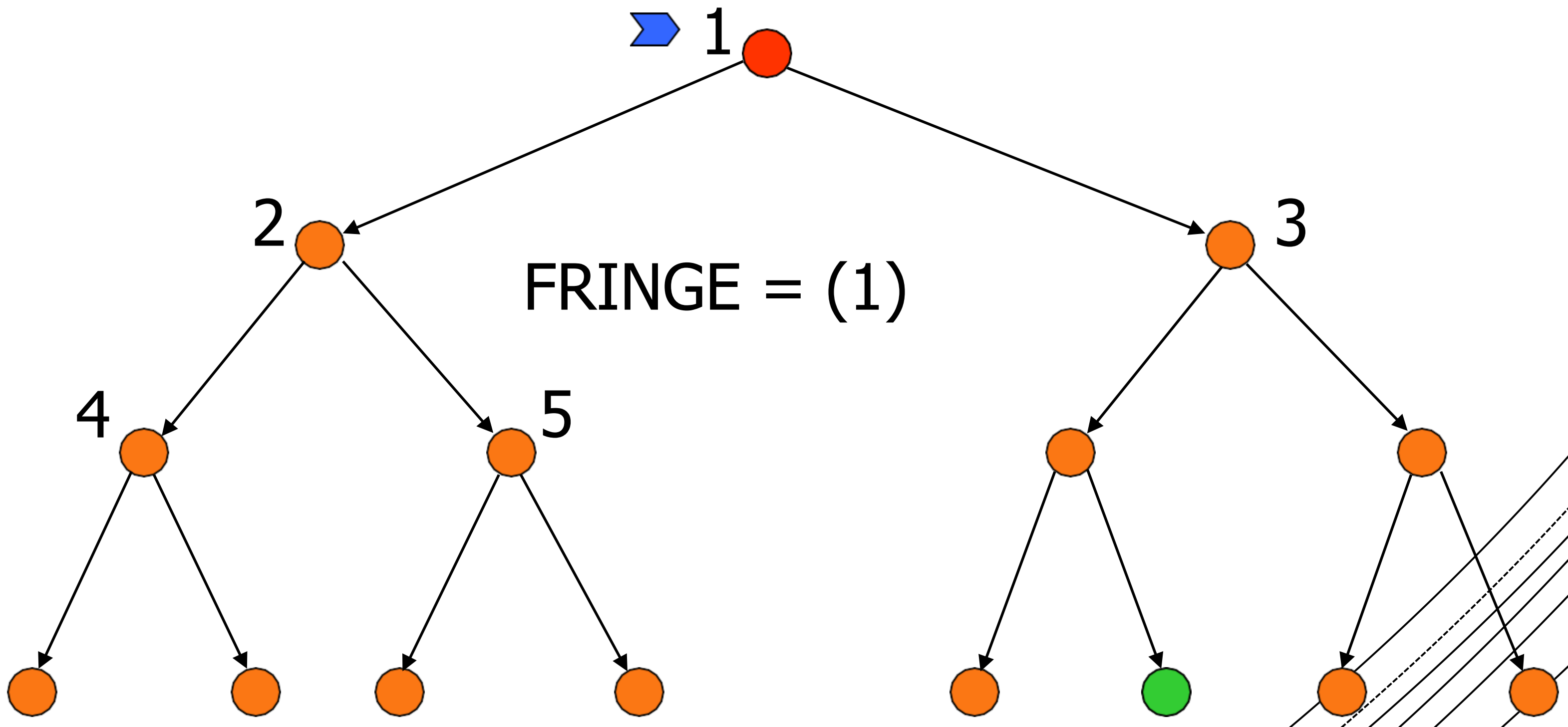
2 fringe queues: FRINGE1 and FRINGE2



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

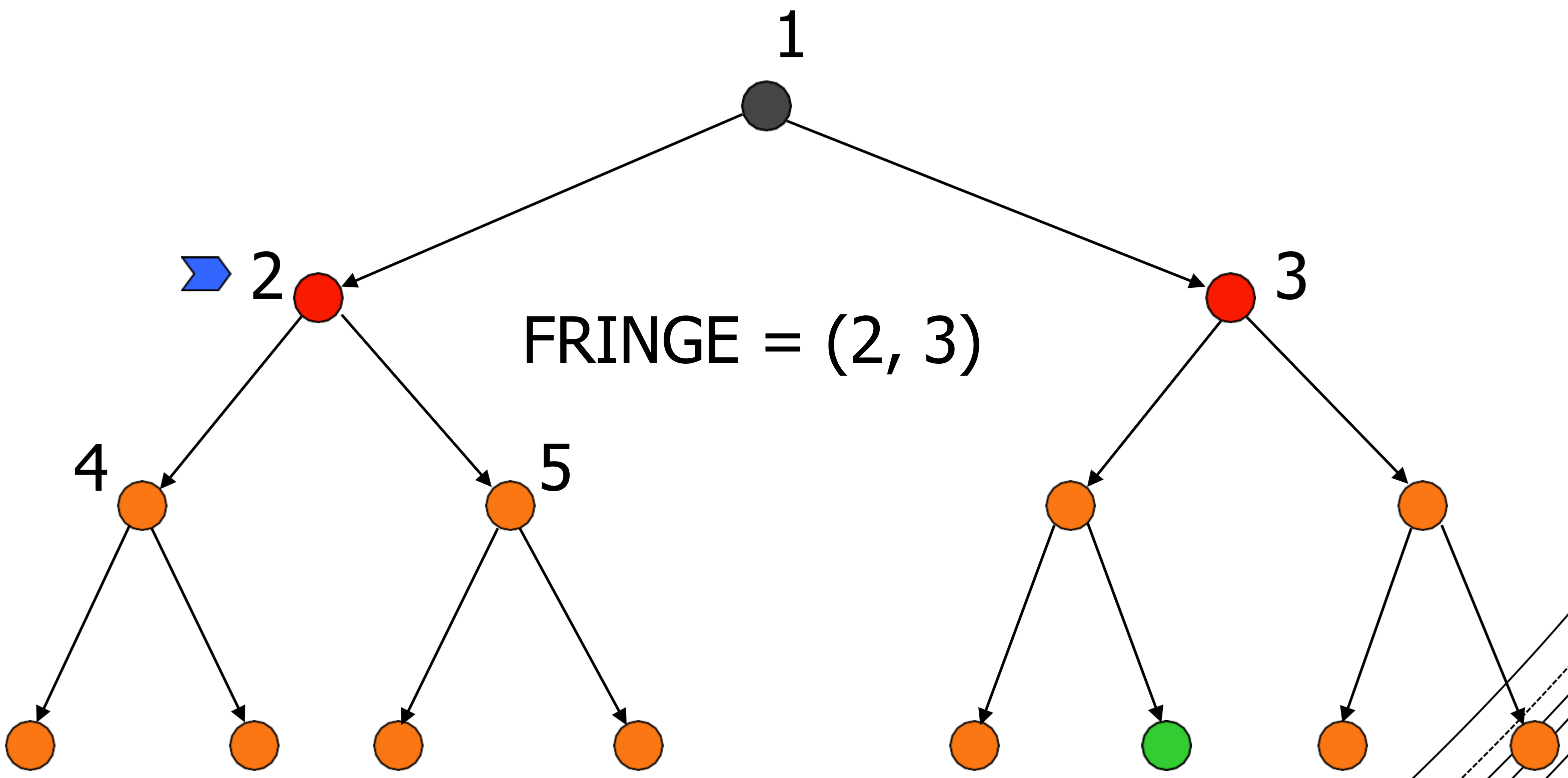
Depth-First Strategy

New nodes are inserted at the front of FRINGE



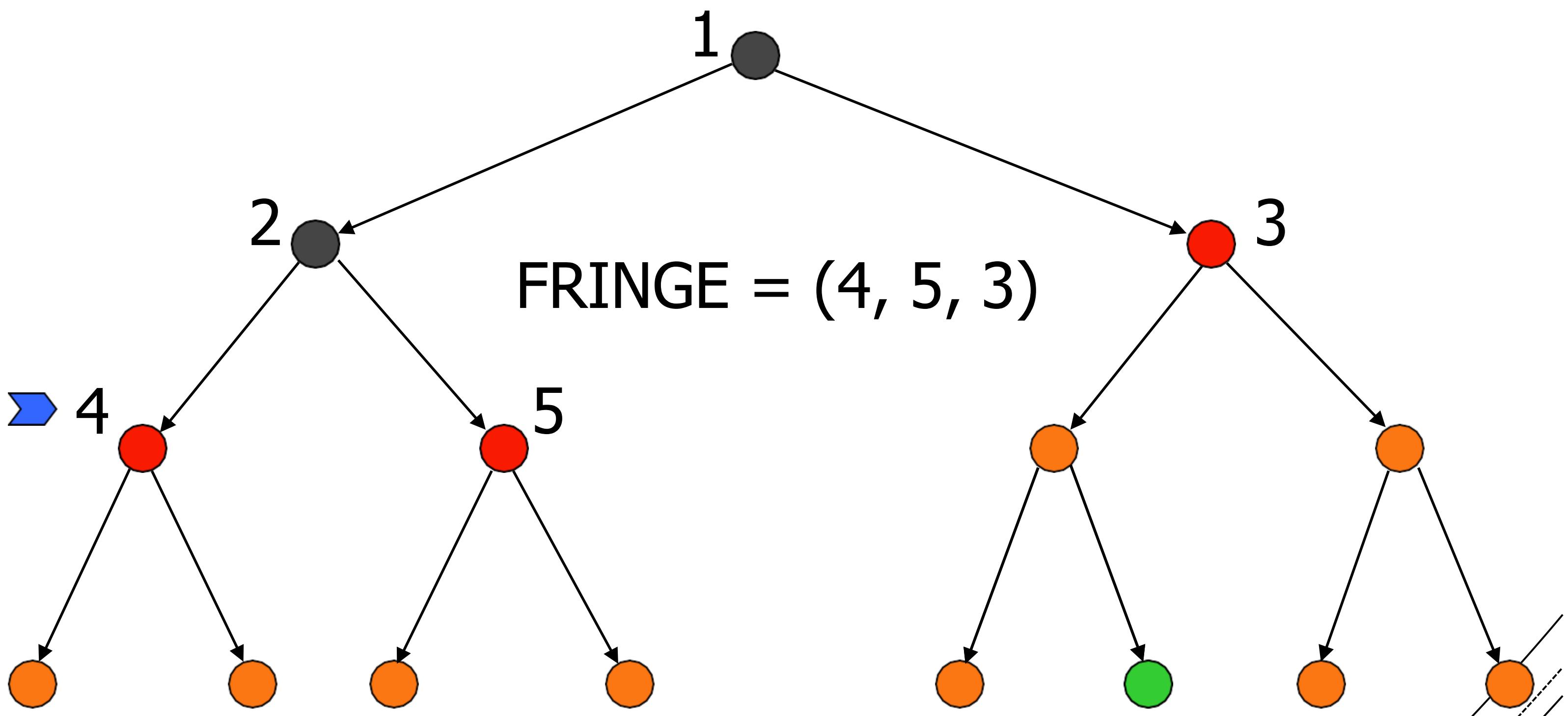
Depth-First Strategy

New nodes are inserted at the front of FRINGE



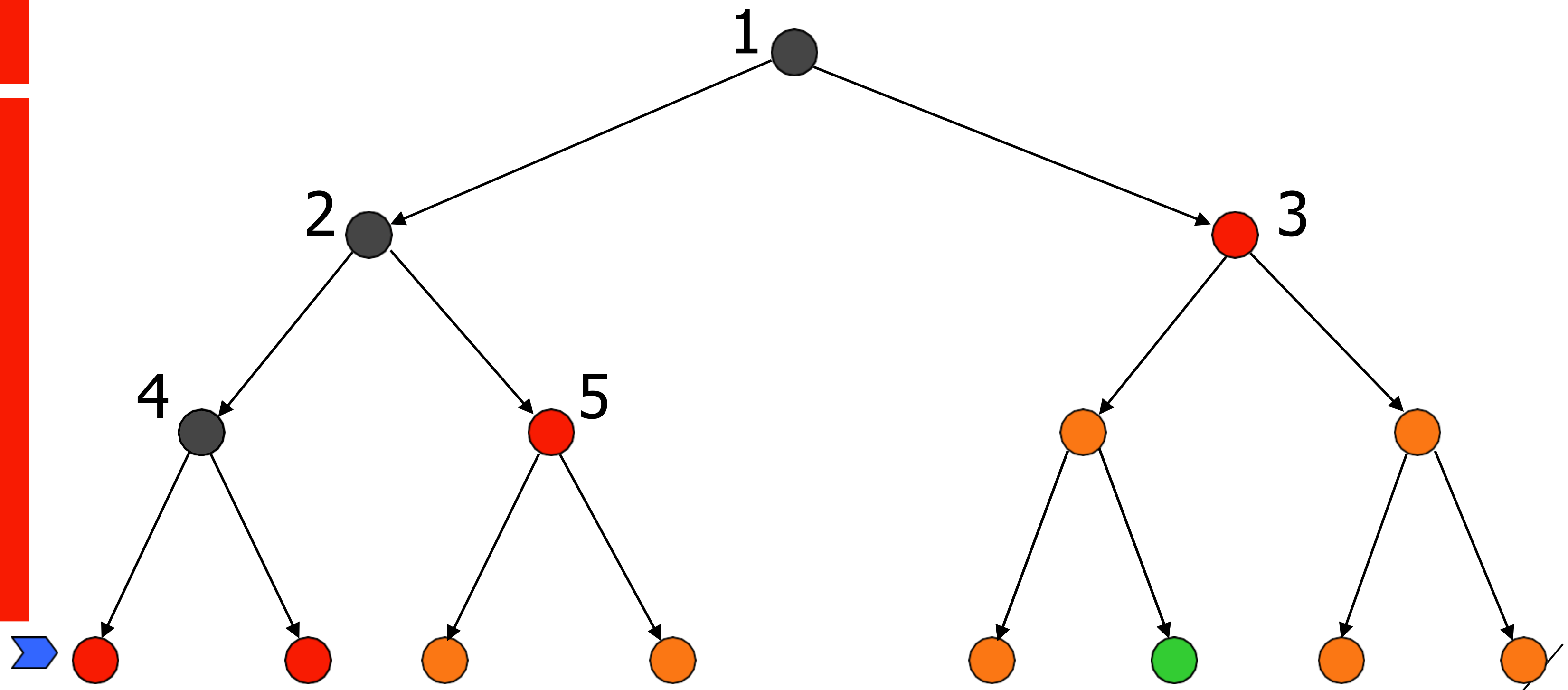
New nodes are inserted at the front of FRINGE

Depth-First Strategy



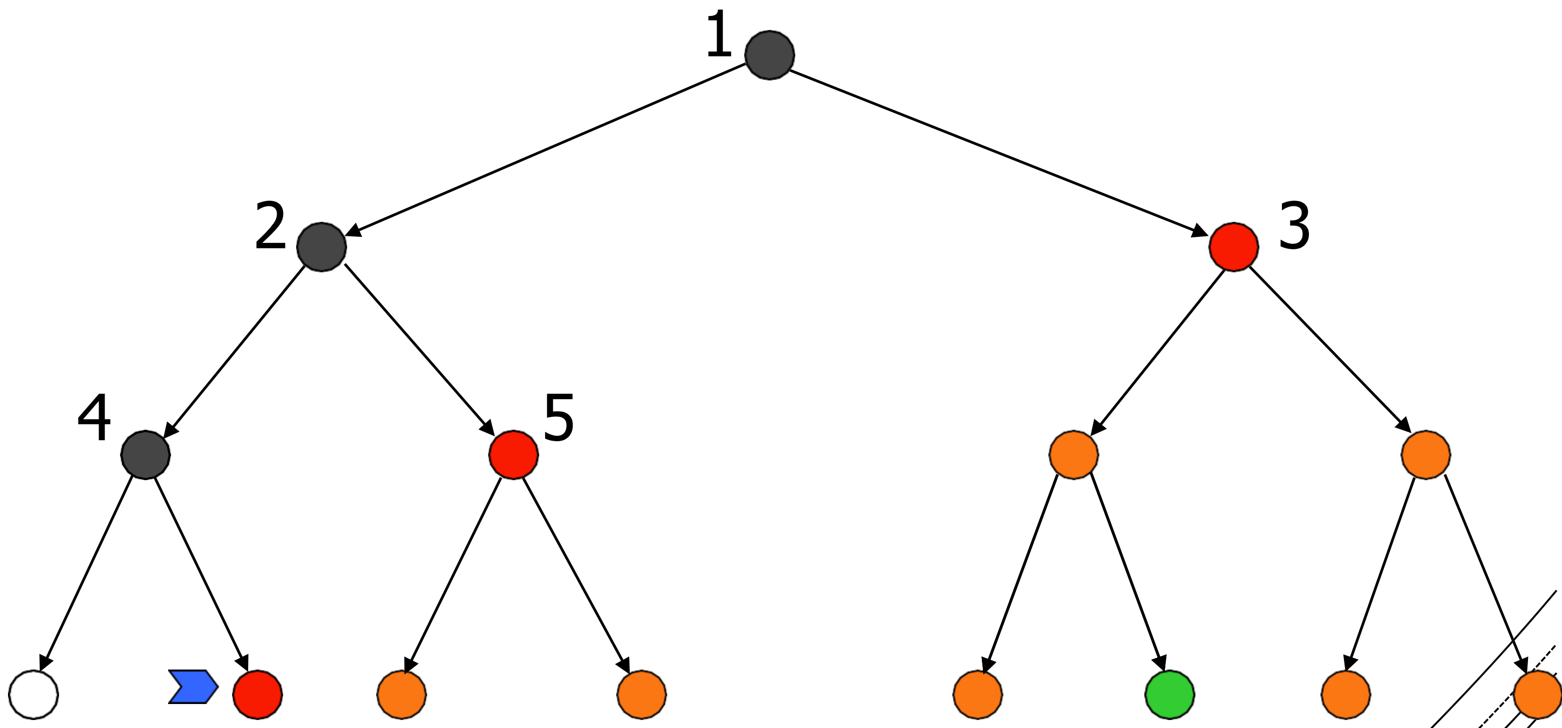
New nodes are inserted at the front of FRINGE

Depth-First Strategy



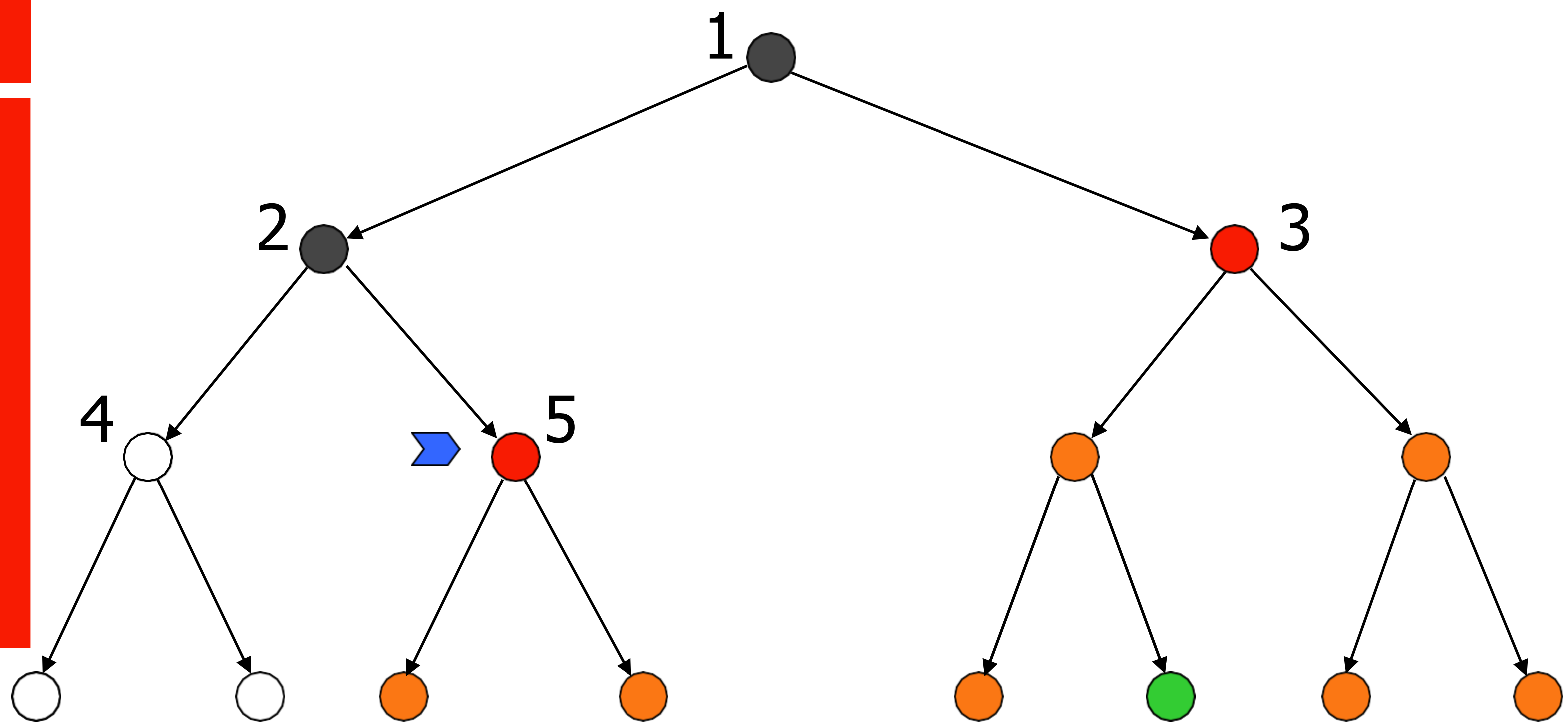
Depth-First Strategy

New nodes are inserted at the front of FRINGE



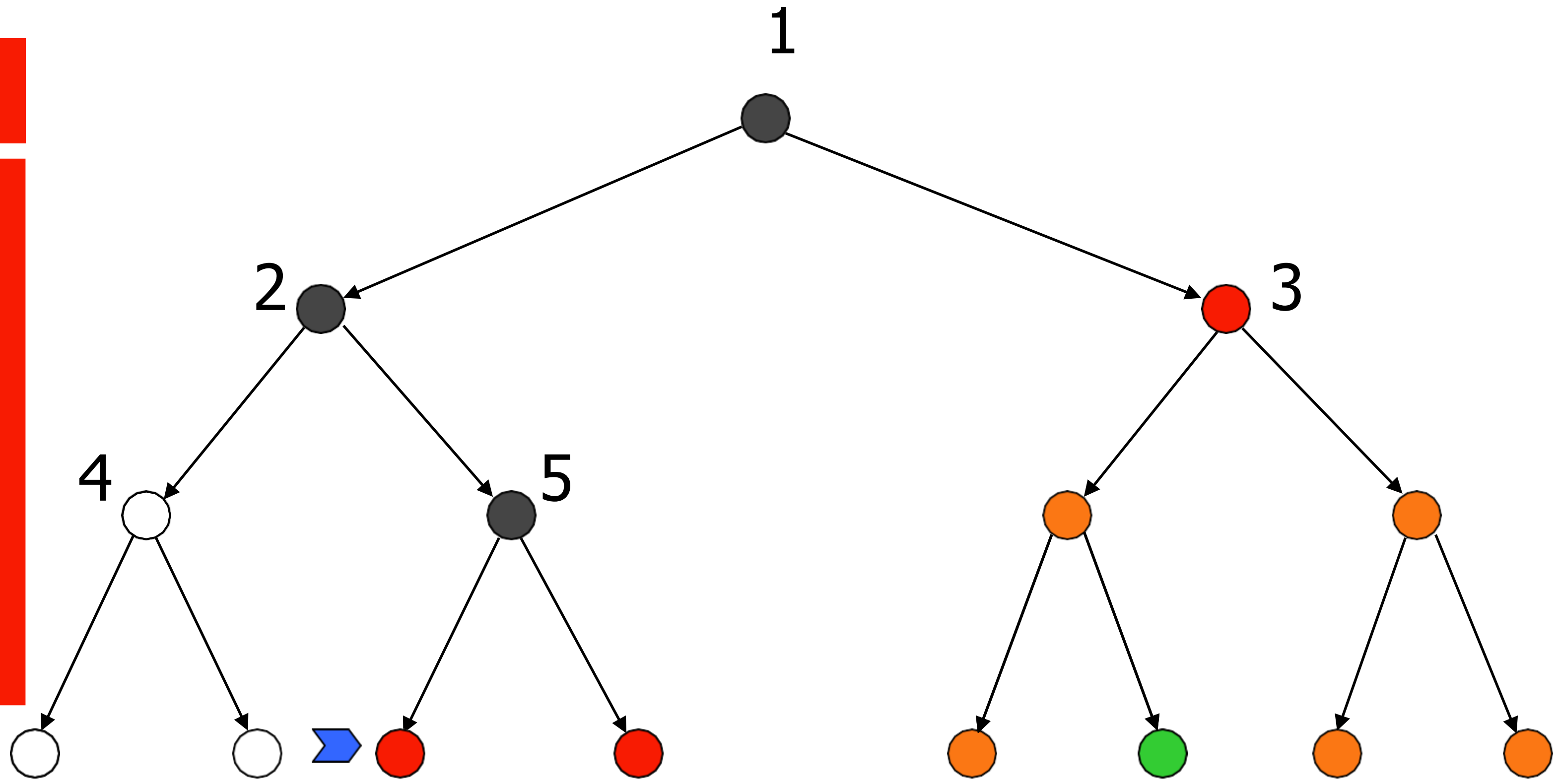
New nodes are inserted at the front of FRINGE

Depth-First Strategy



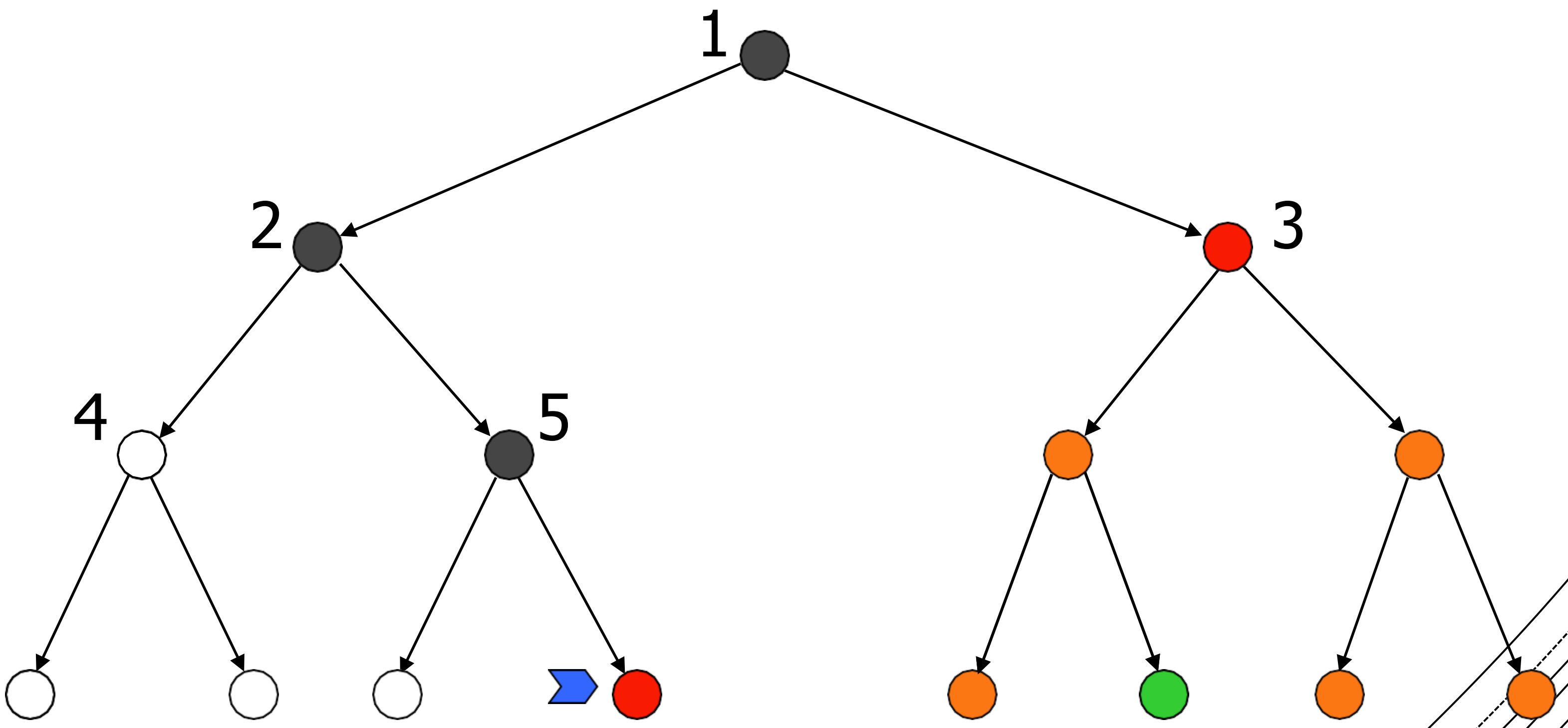
Depth-First Strategy

New nodes are inserted at the front of FRINGE



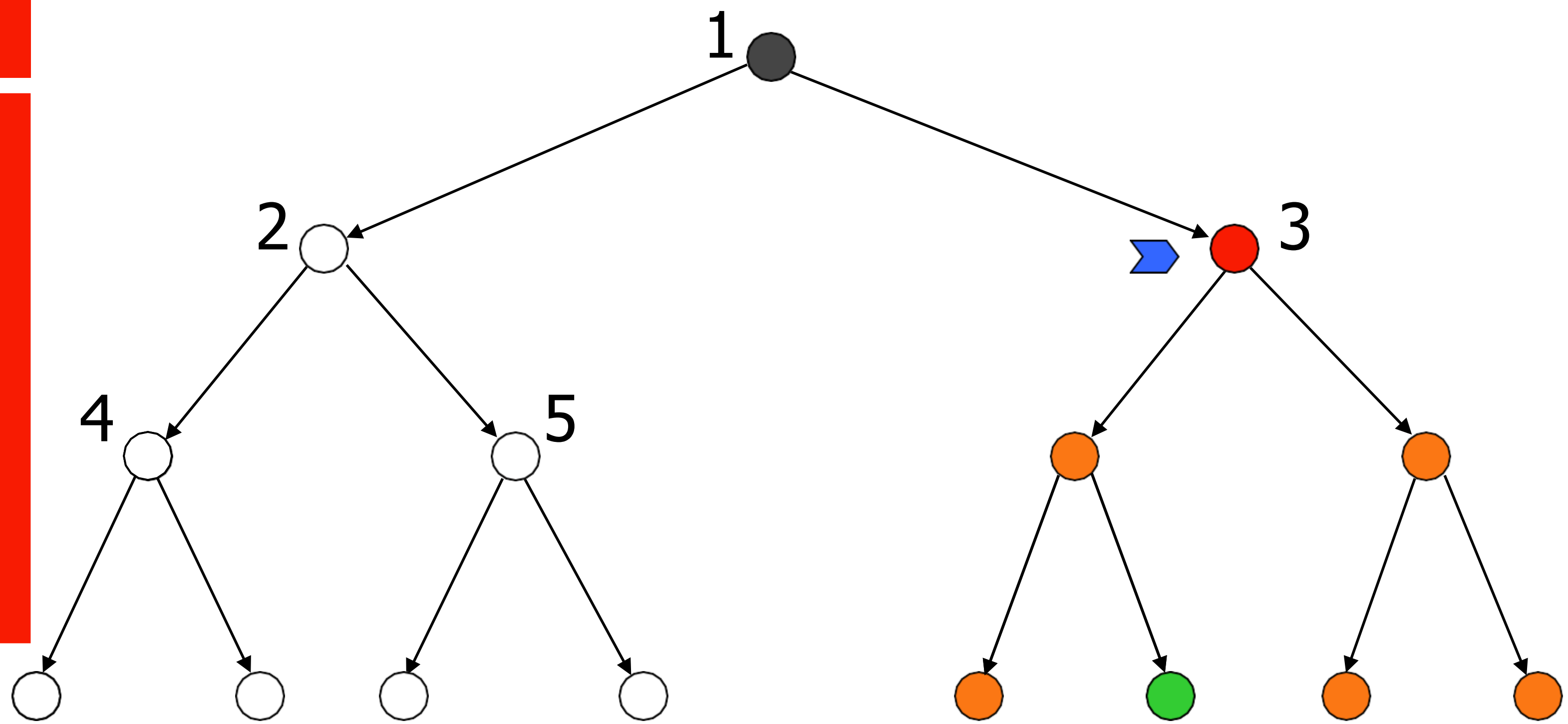
Depth-First Strategy

New nodes are inserted at the front of FRINGE



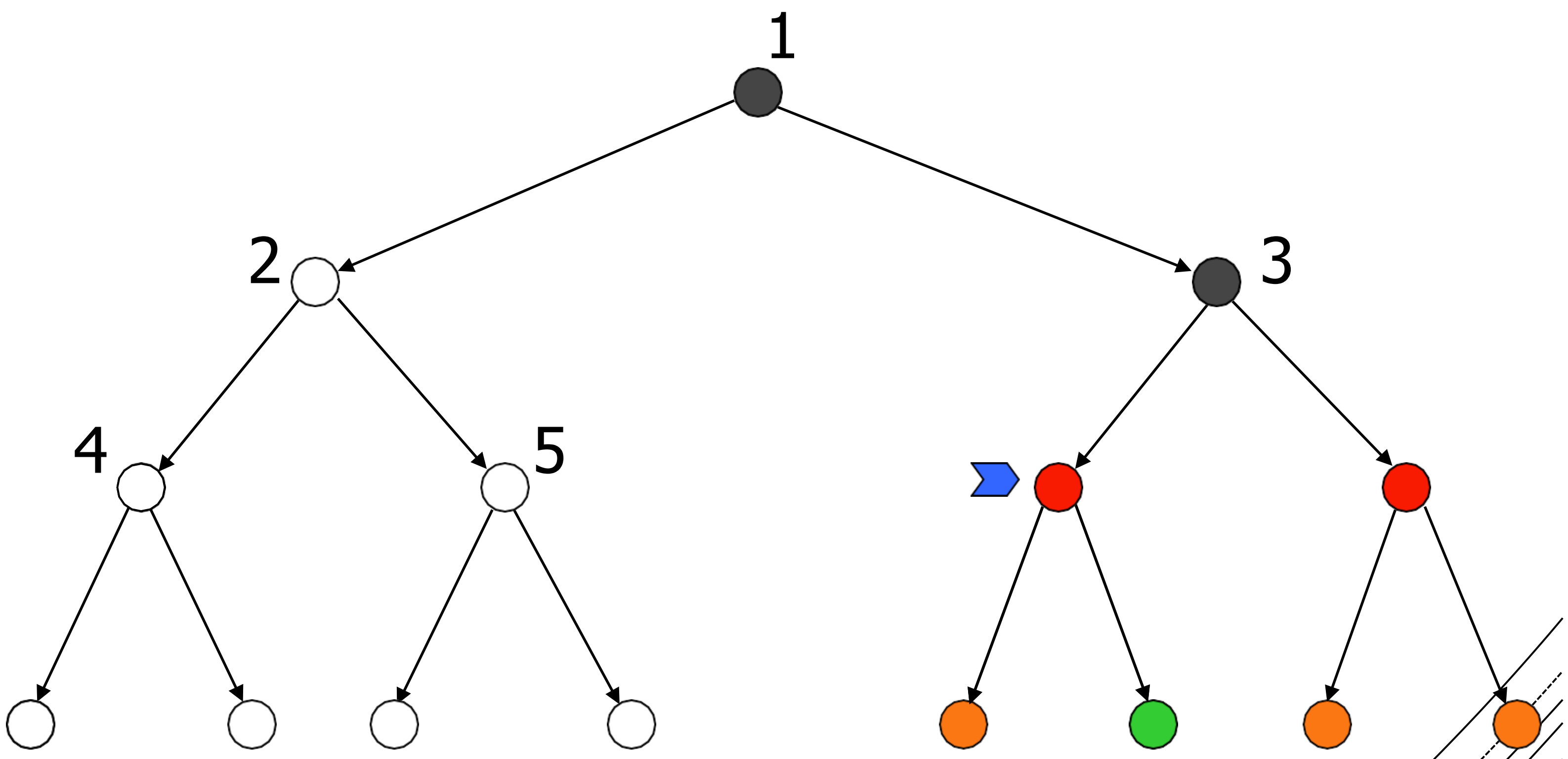
Depth-First Strategy

New nodes are inserted at the front of FRINGE



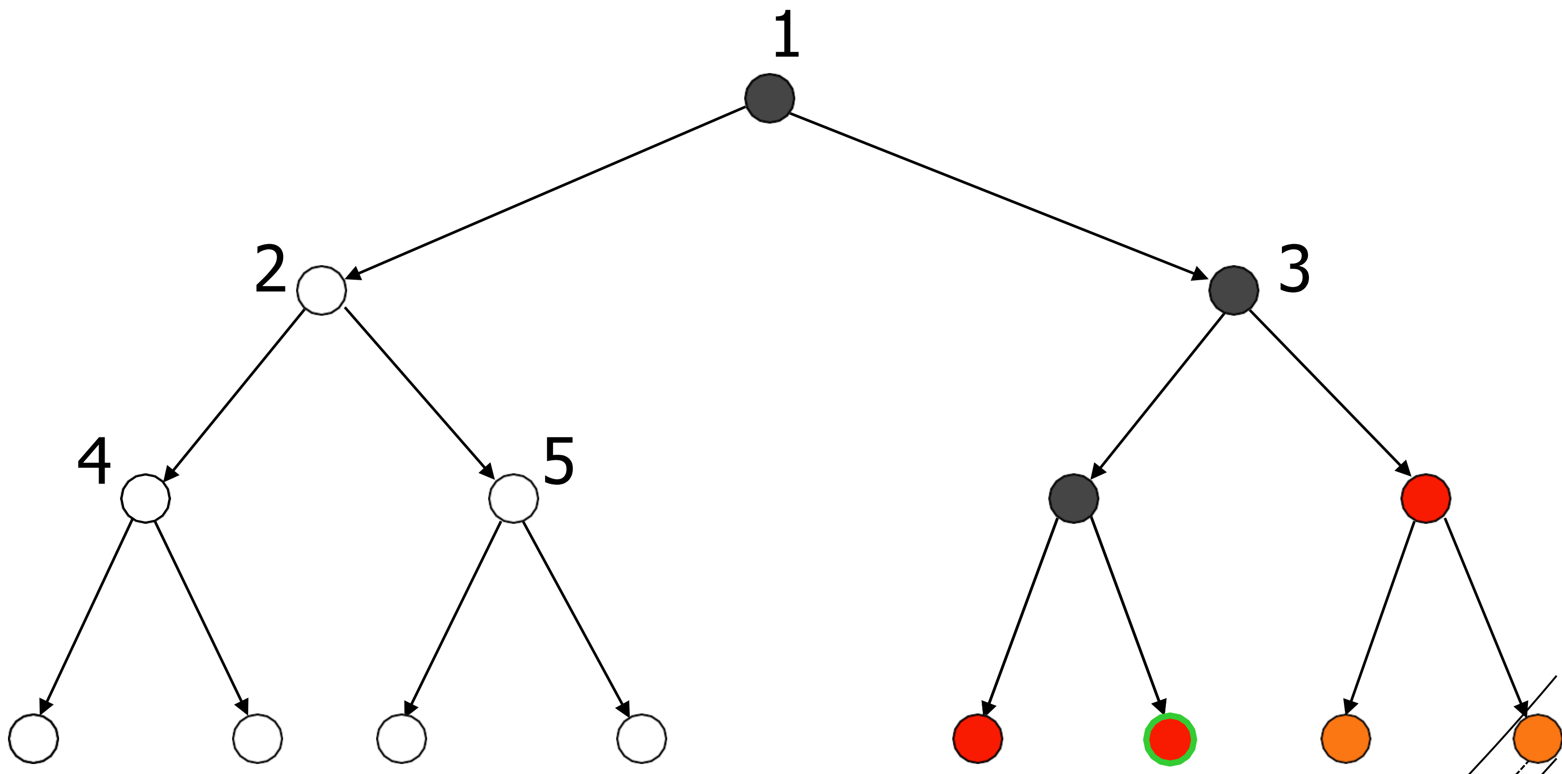
Depth-First Strategy

New nodes are inserted at the front of FRINGE



Depth-First Strategy

New nodes are inserted at the front of FRINGE



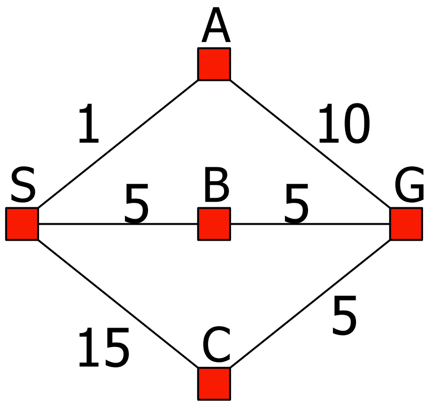
Depth-Limited Strategy

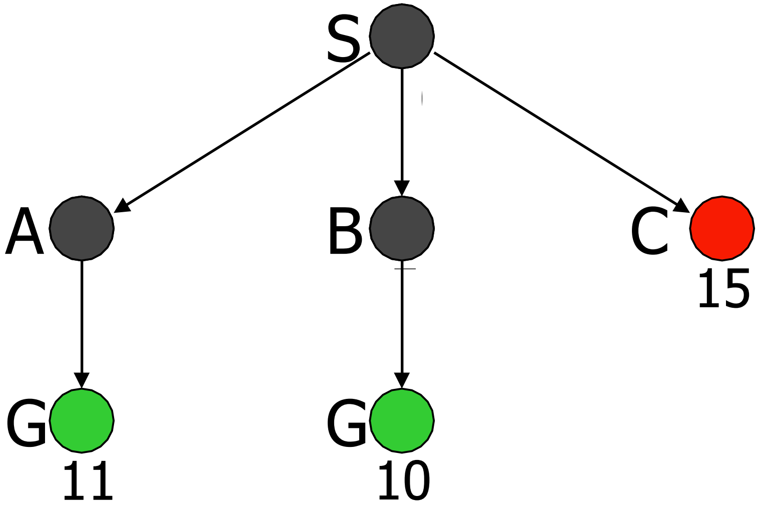
- Depth-first with depth cutoff k (maximal depth below which nodes are not expanded)
- Three possible outcomes:
  - Solution
  - Failure (no solution)
  - Cutoff (no solution within cutoff)

Uniform-Cost Strategy

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

  $g(N) = \Sigma$ costs of all steps.
- The goal is to generate a solution path of minimal cost.
- The queue FRINGE is sorted in increasing cost.





Modified Search Algorithm

1. INSERT(initial-node,FRINGE)
2. Repeat:
   - If FRINGE is empty then return failure
   - n ← REMOVE(FRINGE)
   - s ← STATE(n)
   - If GOAL?(s) then return path or goal state
   - For every state s' in SUCCESSORS(s)
     - Create a node n' as a successor of n
     - INSERT(n',FRINGE)

So far …

- Search tree ≠ state space
- Search strategies: breadth-first, depth-first, and variants
- Evaluation of strategies: completeness, optimality, time and space complexity
- Avoiding repeated states
- Optimal search with variable step costs

Summary

- Search tree ≠ state space
- Search strategies: breadth-first and depth-first
- Evaluation of strategies: completeness, optimality, time and space complexity

Acknowledgement

- Krishnaram Kenthapadi , CS121: Introduction to Artificial Intelligence, 2003.

- AI Instructor's Resource Page, http://aima.cs.berkeley.edu/instructors.html