Lecture 3

Agents and Problem Solving

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 Agent
  - Problem Formulation
- This Class
  - Implementing a Problem-Solving Agent
- Next Class
  - Uninformed Algorithms

Problem-Solving Agents

- **Goal formulation**
  - **A set of one or more desirable world states**
- **Problem formulation**
  - **What actions and states to consider given a goal and an initial state**
    - **States**
    - **Initial State**
    - **Actions**
    - **Transition model**
    - **Goal test**
    - **Path cost**
- **Search for solution**
  - **Given the problem, search for a solution --- a sequence of actions to achieve the goal starting from the initial state**
- **Execution of the solution**

Example Toy Problems – 8-puzzle problems

- **A 3×3 grid board with eight numbered tiles and a blank space.**
- **A tile adjacent to the blank space can slide into the space.**
- **The goal is to reach a specified goal state.**

8	2	
3	4	7
5	1	6

Initial state

→

1	2	3
4	5	6
7	8	

Goal state example

Example Toy Problems – 8-puzzle problems

- **states**
  - location of each of the tiles
- **initial state**
  - any state in state space
- **actions**
  - move blank title *Left, Right, Up,* or *Down*.
  - alternatively: move a numbered tile
- **transition model**
  - given a state and an action, this returns the resulting state
- **goal test**
  - any legitimate configuration of tiles (given; tiles in order)
  - The current state matches the goal configuration
- **path cost**
  - Each move costs 1, so the path cost is the length of the path

Example Toy Problems – 8-puzzle problems

- **State space: the set of all states reachable from the initial state**

- **If states describe the location of each of the tiles, then the total number of states = 9!, where only half states can reach goal state.**

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

15-puzzle → .65 x 10^{12}

24-puzzle → .5 x 10^{25}

- **The state space for some problems is infinite!**

Real-World Problems

- **Route-Finding Problems**
  - Objective: finding shortest path with lowest path cost to the goal state
  - States: current location
  - Operators: move from location to location
  - Goal test: “are we there yet?”; “did we get there in time?”; “found target?”
- **Route-finding algorithms are used in a variety of applications**
  - **Traveling Salesperson Problems (TSP)**
    - Objective: finding shortest tour that visits all cities in a map exactly once
    - States: current location and the set of cities the agent has visited
    - Operators: visit a neighbor (constraint: previously unvisited)
    - Goal test: “Is the tour the minimum?”
  - **Other Problems**
    - Very Large-Scale Integrated (VLSI) circuit layout
    - Robot navigation
    - Assembly sequencing

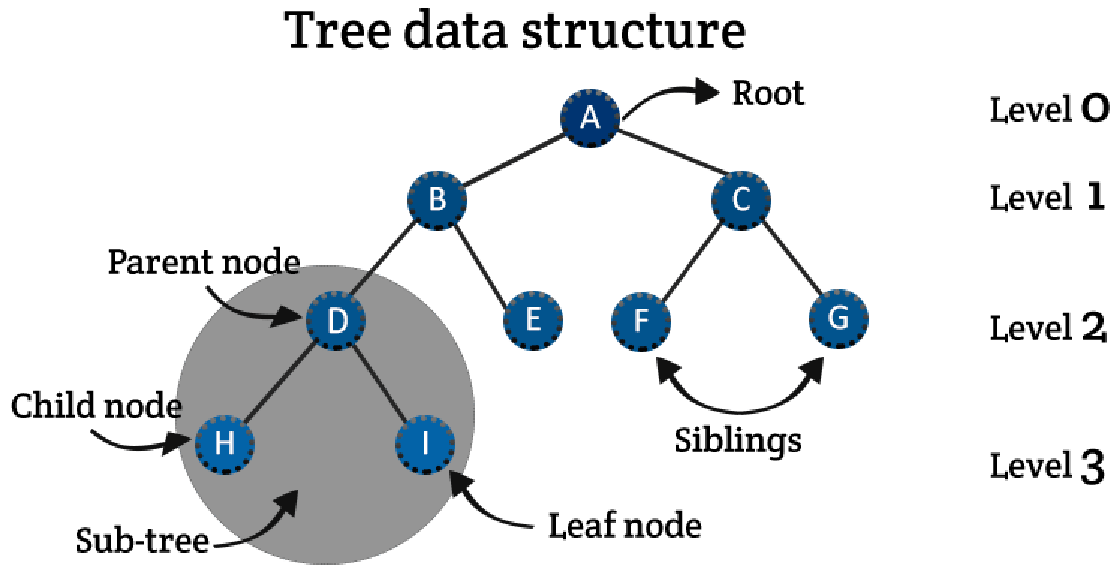
A Simple Problem-Solving Agent

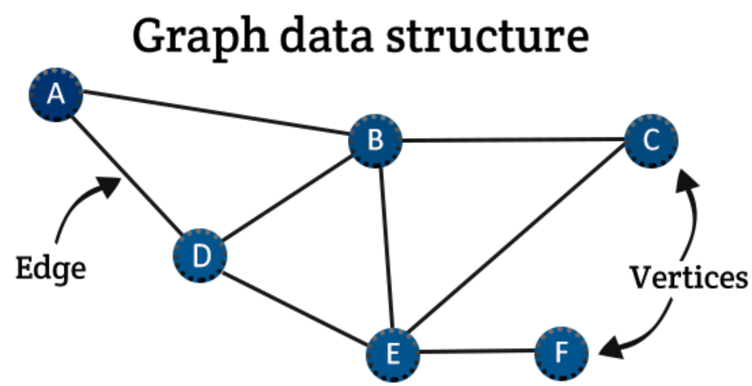
```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
   persistent: seq, an action sequence, initially empty
               state, some description of the current world state
               goal, a goal, initially null
               problem, a problem formulation

   state ← UPDATE-STATE(state, percept)
   if seq is empty then
       goal ← FORMULATE-GOAL(state)
       problem ← FORMULATE-PROBLEM(state, goal)
       seq ← SEARCH(problem)
       if seq = failure then return a null action
   action ← FIRST(seq)
   seq ← REST(seq)
   return action
```

Concepts in Data Structure and Algorithm

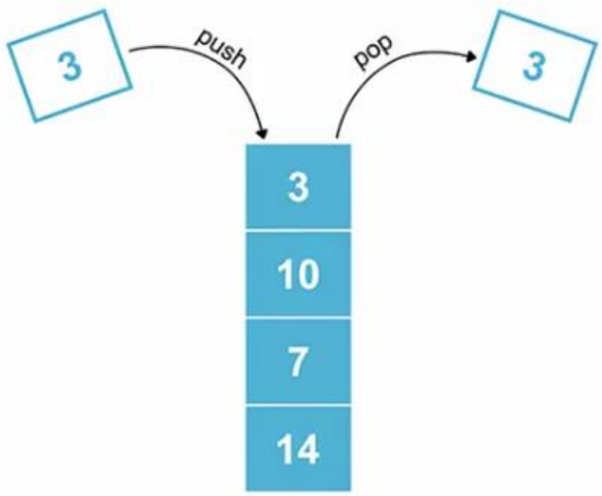
- **Data structures**
  - **Tree**
    - **Nodes**
      - root node
      - parent node
      - child node
      - leaf node
    - **Edges**
  - **Graph**
    - **Nodes**
    - **Edges**



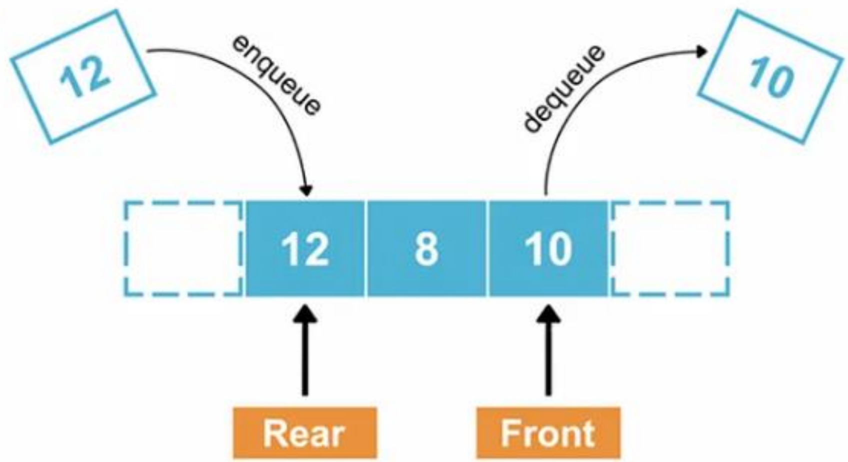


Concepts in Data Structure and Algorithm

- **Data structures**
  - **Stack**
    - **Last-In-First-Out**
  - **Queue**
    - **First-in-First-Out**



Stack



Queue

Search Terminology

- **Search**
  - The process of looking for a sequence of actions that can start from the initial state and arrive in the goal state

- **Solution**
  - This sequence of actions (path) is called a solution

- **Optimal Solution**
  - Has the lowest cost among all the solutions

- **Execution**
  - Carry out the actions specified by the solution

Basic Search Concepts

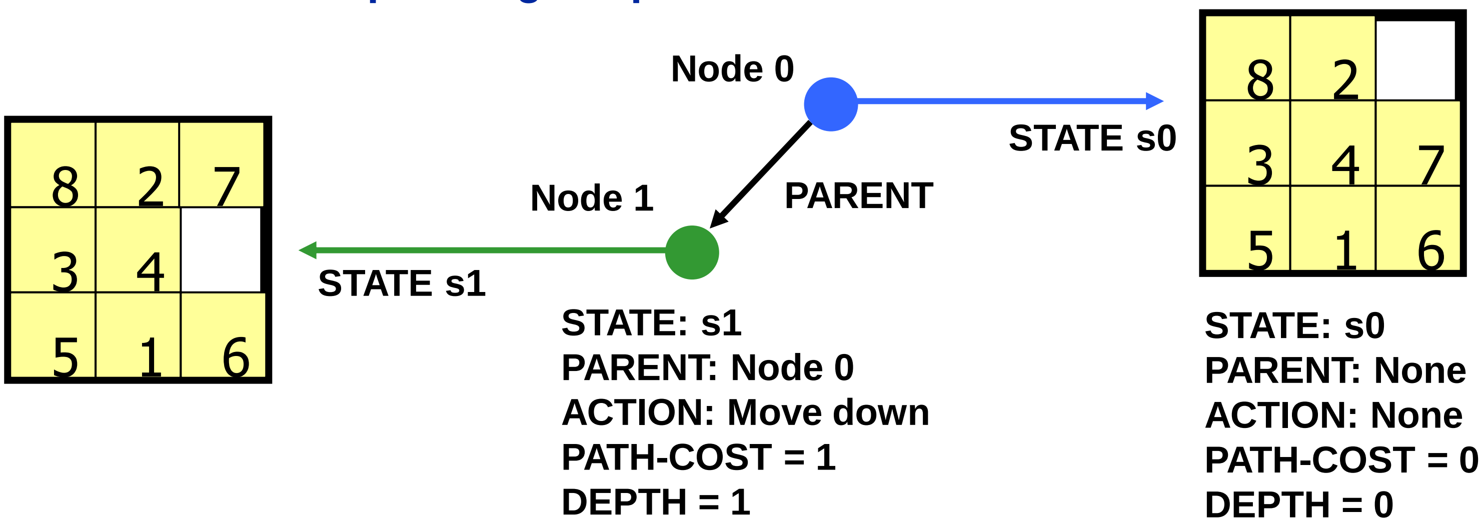
- Finding out a solution is done by
  - searching through the state space using a search tree

- Search tree
  - generated by the initial state and successor function
  - Initial state: the root is a search node
  - Expanding: applying successor function to the current state, thereby generating a new set of states, i.e., (child) nodes
    - newly generated nodes are added to the frontier
    - Frontier (fringe): the set of nodes not yet expanded
  - Leaf nodes: the states having no successors

  - Search strategy: determines the next node to be expanded
    - can be achieved by ordering the nodes in the frontier
    - good choice → fewer work → faster
  - Important: state space ≠ search tree

Search Nodes ≠ States

- **State space has unique states while a search tree may have cyclic paths**
- **State *s*: an admissible configuration of the world**
- **Node *n:* a data structure used to represent the search tree, that contains:**
  - ***n*.STATE: the state *s* to which *n* corresponds to**
  - ***n*.PARENT: the node in the search tree that generated node *n***
  - ***n*.ACTION: the action that was applied to the parent to generate *n***
  - ***n*.PATH-COST: the cost, *g(n),* of the path from the initial node to *n***
  - **n.DEPTH: number of steps along the path from the initial state**



- **Nodes are on specific paths, while states are not**

Frontier

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

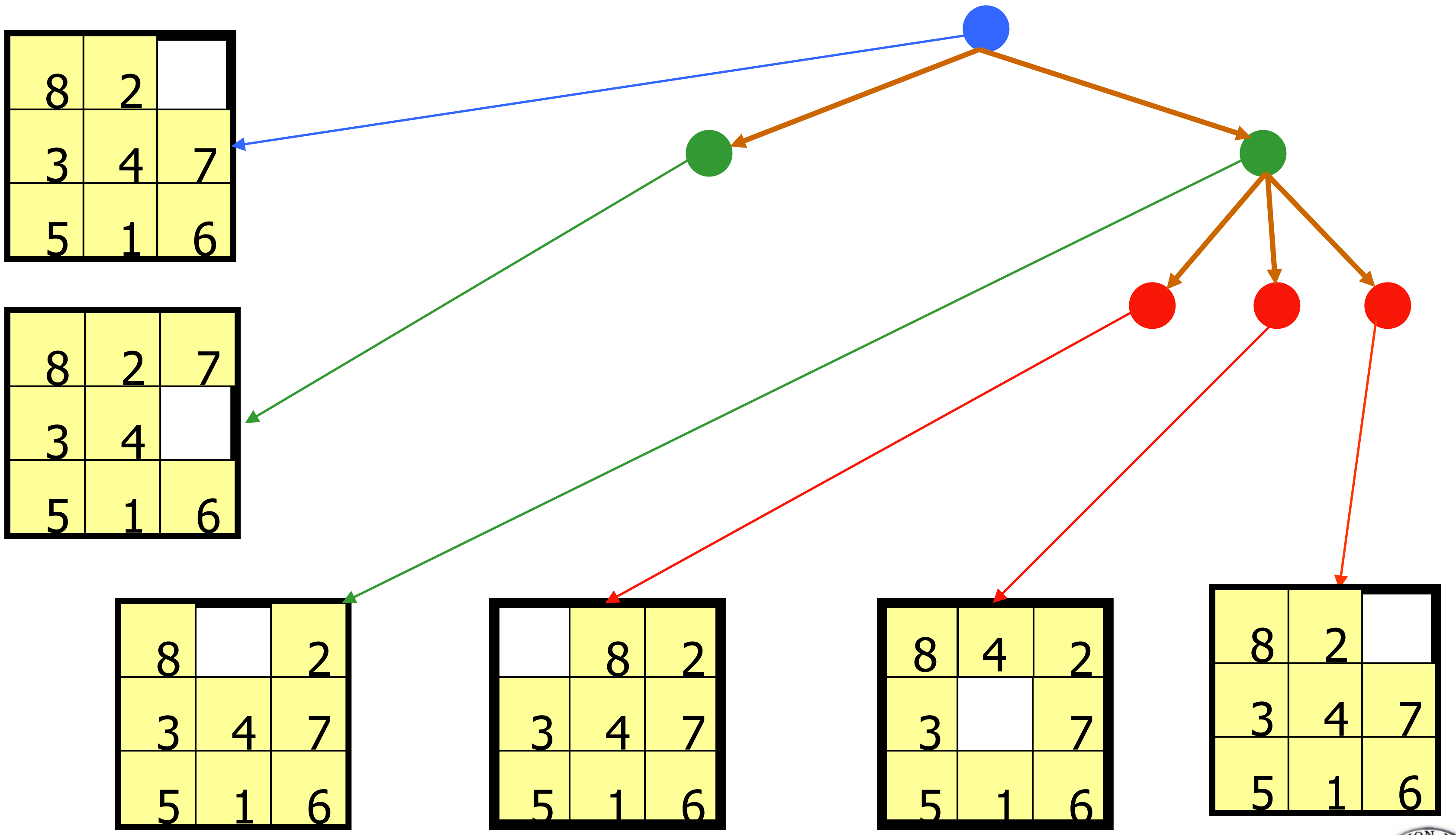
Searching for Solutions

- **A solution is found by two kinds of search tree algorithms:**
  - **Tree Search (problem, search strategy)**
    - **maintains a frontier**
  - **Graph Search (problem, search strategy)**
    - **maintains a frontier and an explored set**

Tree Search

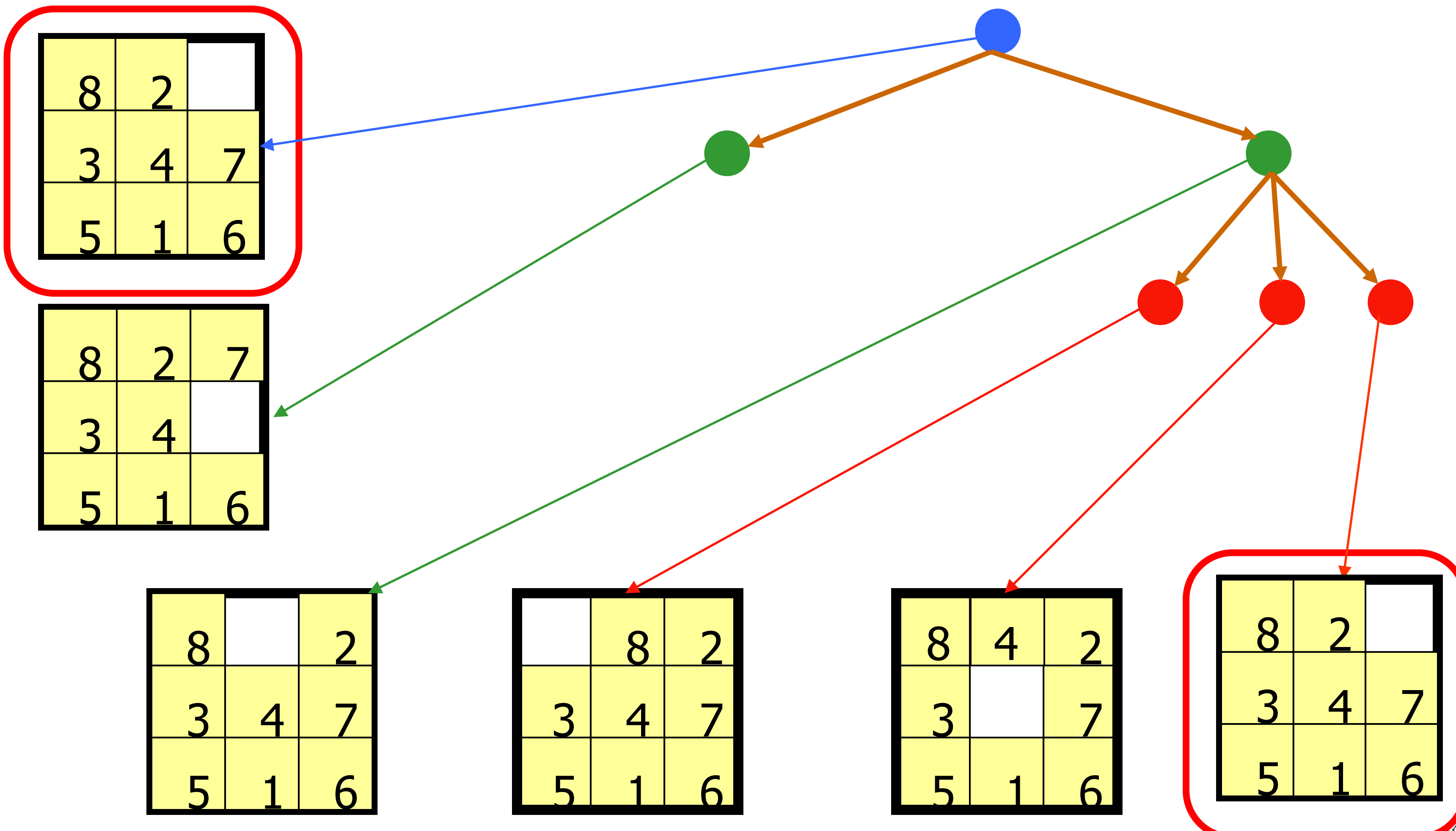
- **function Tree-Search (problem, strategy)**
- **initialize the *frontier using* the initial state of problem**
- **loop**
  - **if there are no frontier nodes then return failure**
  - **choose a frontier node for expansion using strategy and remove it from frontier**
  - **if the node contains a goal then return the corresponding solution**
  - **else expand the node and add the resulting nodes to the set of frontier nodes**

Tree Search Example



Tree Search Example

Same state, different nodes! Cyclic path!

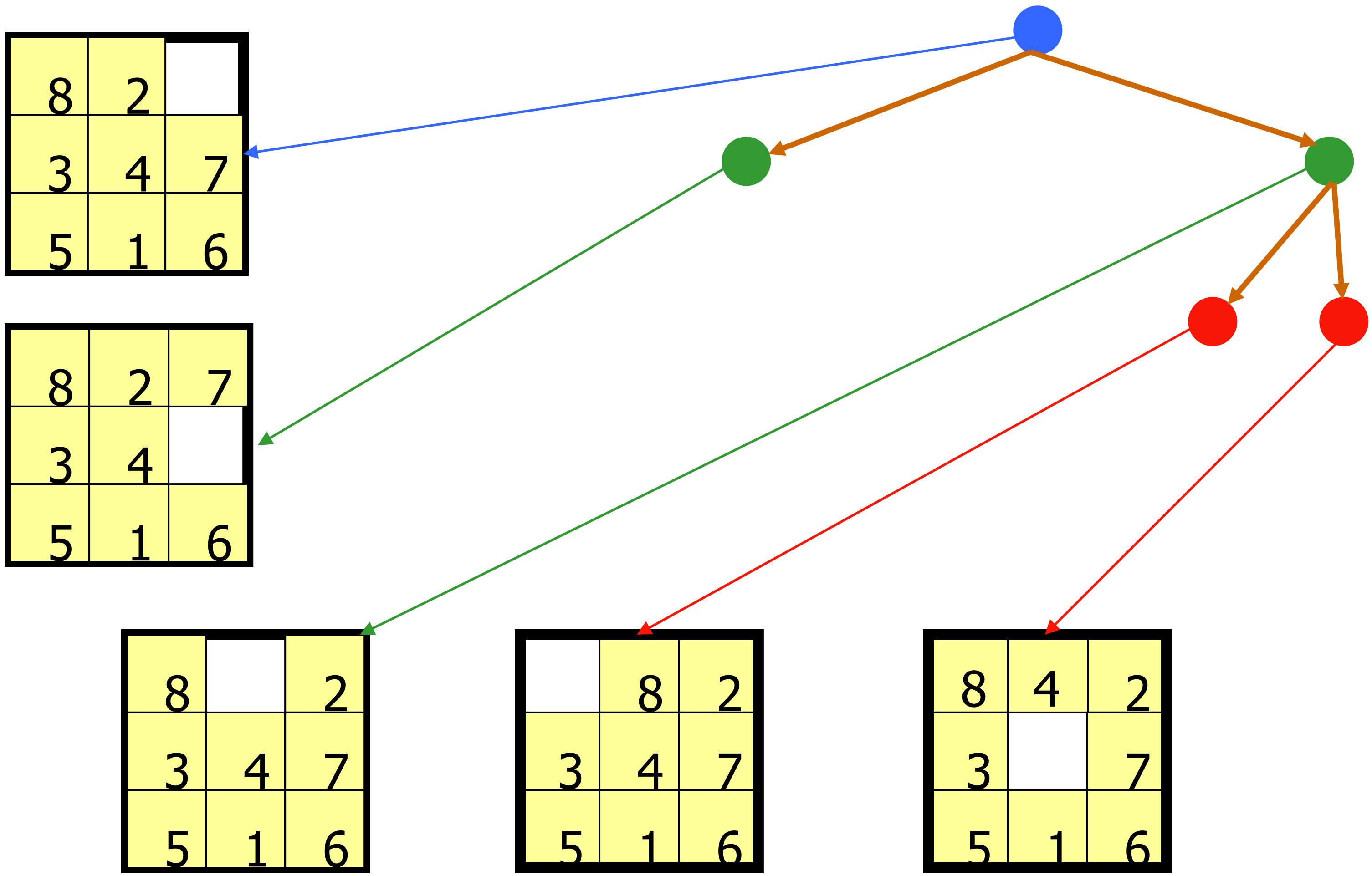


Graph Search

- function Graph-Search (problem, strategy)
- initialize the *frontier using* the initial state of problem
- initialize the explored set to be empty
- loop
  - if there are no frontier nodes then return failure
  - choose a frontier node for expansion using strategy, remove it from frontier, and add it to the explored set
  - if the node contains a goal then return the corresponding solution
  - else expand the node and add the resulting nodes to the set of frontier nodes, only if not in the frontier or explored set

How to avoid cyclic path? Introducing an explored set

Graph Search Example



Graph Search

- **function Graph-Search (problem, strategy)**
- **initialize the *frontier using* the initial state of problem**
- **initialize the explored set to be empty**
- **loop**
  - **if there are no frontier nodes then return failure**
  - **choose a frontier node for expansion using strategy, remove it from frontier, and add it to the explored set**
  - **if the node contains a goal then return the corresponding solution**
  - **else expand the node and add the resulting nodes to the set of frontier nodes, only if not in the frontier or explored set**

Optimal but memory inefficient

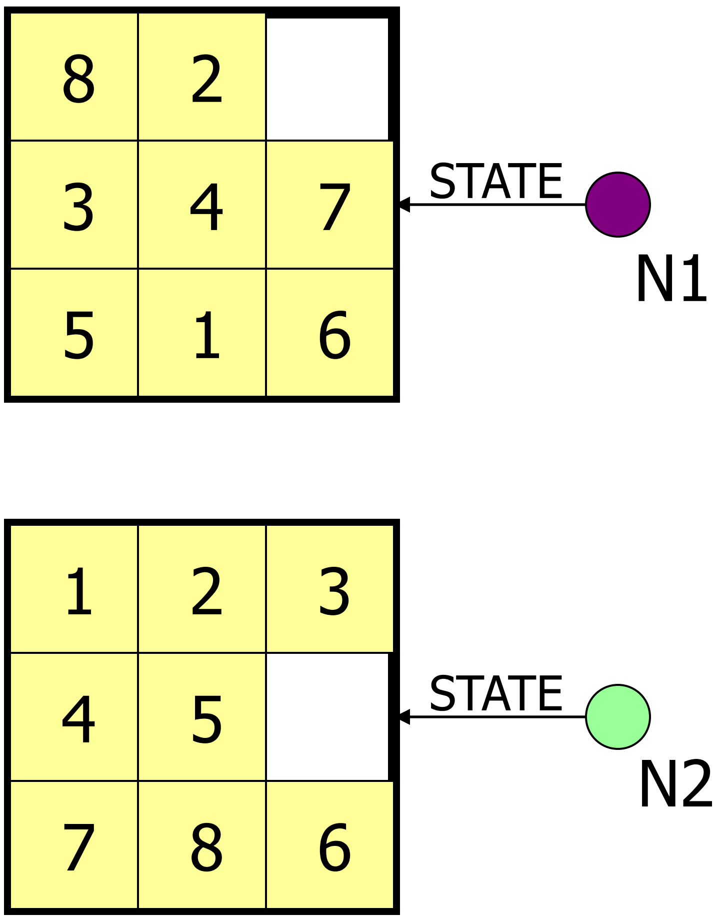
Differences from Tree-Search

- **Each node is associated to a different state**
- **Every path from initial state to an unexplored state has to pass through the frontier.**

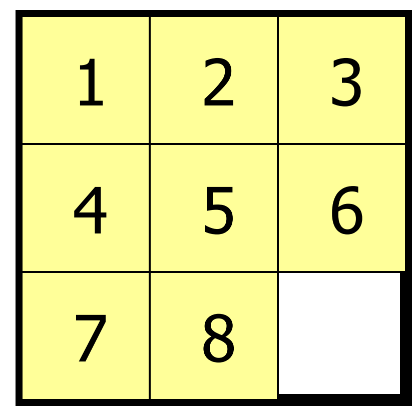
Search Strategies

- **All search strategies are distinguished by *the order* in which nodes are expanded.**
  - **Uninformed search (Blind Search)**
    - **no information about the path cost from the current state to the goal**
    - **search the state space blindly by generating successors and distinguishing goals from non-goals**
  - **Informed search (Heuristic Search)**
    - **know whether one non-goal state is “more promising” than another**
    - **a cleverer strategy that searches toward the goal**

8-puzzle Example for “More Promising”



For a heuristic strategy counting the number of misplaced tiles, N2 is more promising than N1



Goal state

Measuring problem-solving performance

- **Completeness**
  - Is the algorithm guaranteed to find a solution when there is one?
- **Optimality**
  - Does the strategy find the optimal solution?
- **Time Complexity**
  - How long does it take to find a solution?
- **Space Complexity**
  - How much memory is needed to perform the search?

Measuring problem-solving performance

- In AI, complexity is expressed in
  - b, branching factor, maximum number of successors of any node
  - d, the depth of the shallowest goal node.
    (depth of the least-cost solution)
  - m, the maximum length of any path in the state space

- Time and space is measured in
  - The number of nodes generated or expanded during the search
  - The maximum number of nodes stored in memory

- For effectiveness of a search algorithm
  - The total cost = path cost (g) of the solution found + search cost
  - search cost = time necessary to find the solution
  - Tradeoff

Uninformed Search

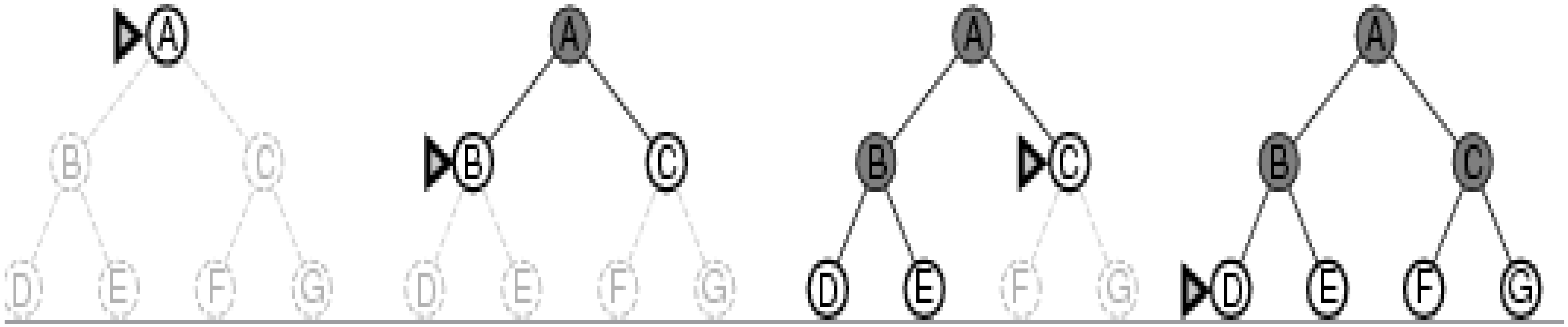
- **Breadth-first Search (BFS)**
- **Depth-first Search (DFS)**
- **Uniform-Cost Search (UCS)**

Breadth-first Search

- **Search Strategy**
  - Always expands the shallowest node in the current frontier of the search tree

- **Procedure**
  1. The root node is expanded first (FIFO)
  2. All the nodes generated by the root node are then expanded
  3. And then their successors and so on

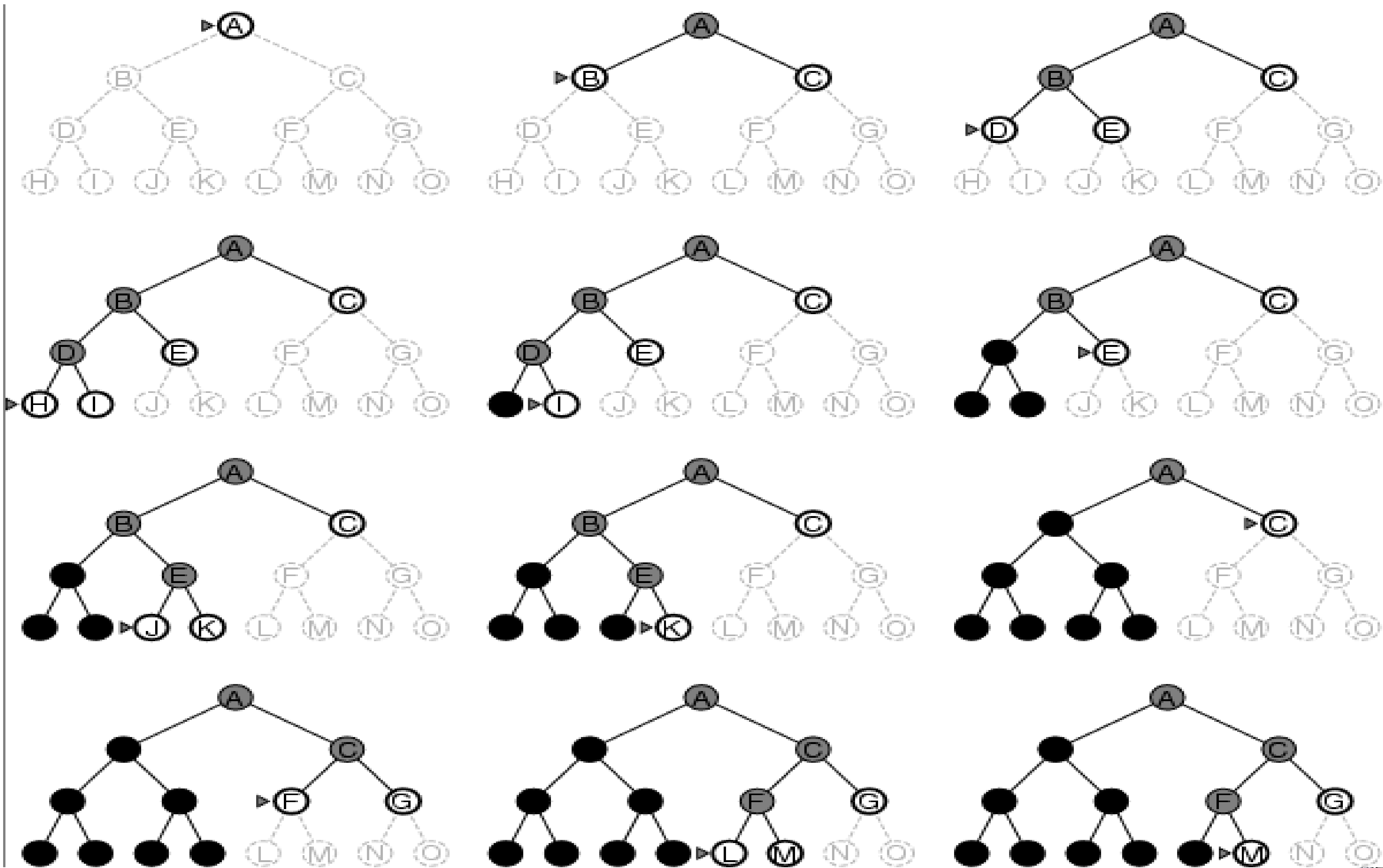
Bread-First Search



Depth-first Search

- **Strategy**
  - Always expands the deepest node in the current frontier of the search tree
- **Procedure**
  - Start from the root node
  - Expand a successor node at the next level
  - Until reach the leaf, then "back up" to the next "shallower" node that still has unexplored successors

Depth-First Search Example



Summary

- Problem-Solving Agent
- Important AI problems
- Search Terminology
- Measuring Search Algorithms
- Uninformed Search
  - Breadth-first search
  - Depth-first search

What I want you to do

- **Review Chapter 3**