

# CS161

## Discussion 2

# Uninformed Search Algorithms

Jinghao Zhao

10/8/2021

# Search Problem

**A search problem** consists of

- Initial State
- A state space  $S = \{s_1, s_2, \dots, s_d\}$
- Actions: a set of possible actions
- Successor function (transition model):  $F(s_t, a_t) = s_{t+1}$ , sometimes with a path cost function
- Goal test : determine if solution is achieved.

**A solution:**

a sequence of actions that transform the initial state to a goal state

**Problem formulation:**

the process of deciding what actions and states to consider, given a goal.

# Search Problem – Example - Travel from CA to MA

- **Initial State:** CA
- **State space:** 50 states in U.S.
- **Actions:** go to adjacent state. cost=distance
  - For example,  $\text{Actions}(\text{CA}) = \{\text{Go}(\text{OR}), \text{Go}(\text{AZ}), \text{Go}(\text{Nevada})\}$
- **Successor function** (transition model)
  - $\text{RESULT}(\text{In}(\text{CA}), \text{Go}(\text{AZ})) = \text{In}(\text{AZ})$
- **Goal test:**
  - $\text{state} == \text{MA}?$
- **Path cost function**



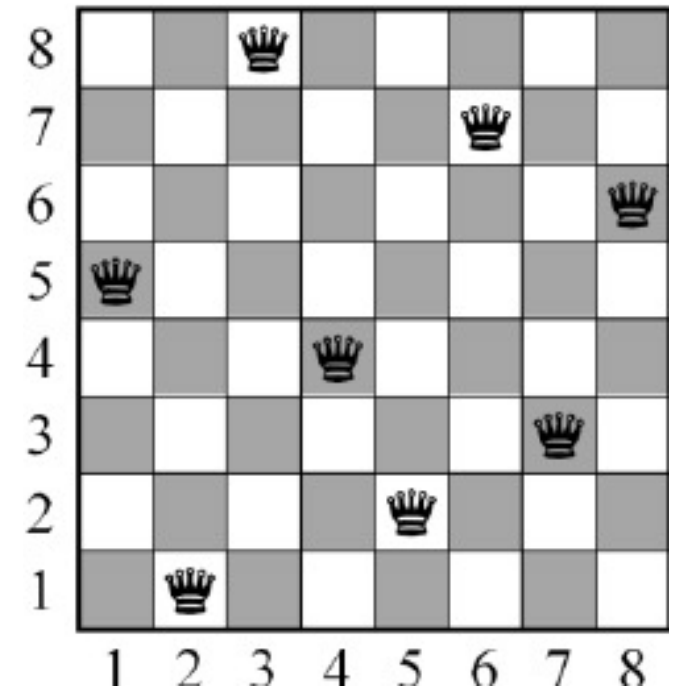
# Search Problem – 8 queens

Objective:

Place eight queens on a chessboard such that no queen attacks any other.

(No two queens on the same row, column, diagonal)

(We don't care about how or how long you find the solution)



# Search Problem – 8 queens

Formulate this problem.

- **States:** Any arrangement of 0 to 8 queens on the board is a state.
- **Initial state:** No queens on the board.
- **Actions:** Add a queen to any empty square.
- **Transition model:** Returns the board with a queen added to the specified square.
- **Goal test:** 8 queens are on the board, none attacked.

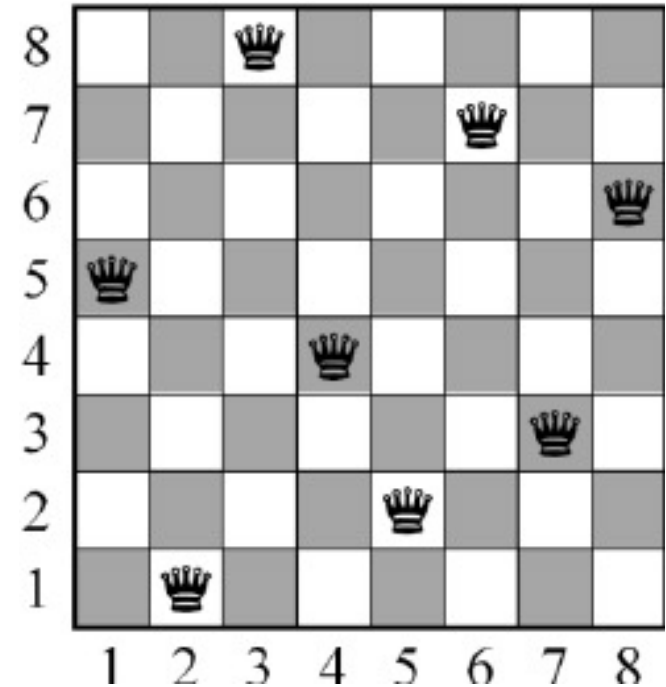
(We don't care about path cost here)

How many possible sequences?

$$64 \times 63 \times \cdots \times 57 = 1.8 \times 10^{14}$$

# Search Problem – 8 queens

- **States:** All possible arrangements of  $n$  queens ( $0 \leq n \leq 8$ ), one per column in the **leftmost  $n$  columns**, with **no queen attacking another**.
- **Actions:** Add a queen to any square in the **leftmost empty column** such that it is not attacked by any other queen.
- **Transition model:** Returns the board with a queen added to the specified square.
- **Goal test:** 8 queens are on the board, none attacked.



# Search Problem – 8 queens

- Incremental formulation
  - Start with an empty state
  - Each action adds a queen
- Complete-state formulation
  - Start with all 8 queens on the board
  - Moves queens around

In either case, path cost doesn't matter because only final state counts.

# Search Problem Formulation - Exercise 1

- Two friends live in different cities on a map
- On every turn, we can simultaneously move each friend to a neighboring city
- $\text{Time}(\text{city } i \rightarrow \text{neighbor } j) = \text{distance } d(i, j)$
- On each turn **the friend that arrives first must wait until the other one arrives** (and calls the first on his/her cell phone) before the next turn can begin.
- We want the two friends to **meet as quickly as possible**.

**□ Write a formulation for this search problem**



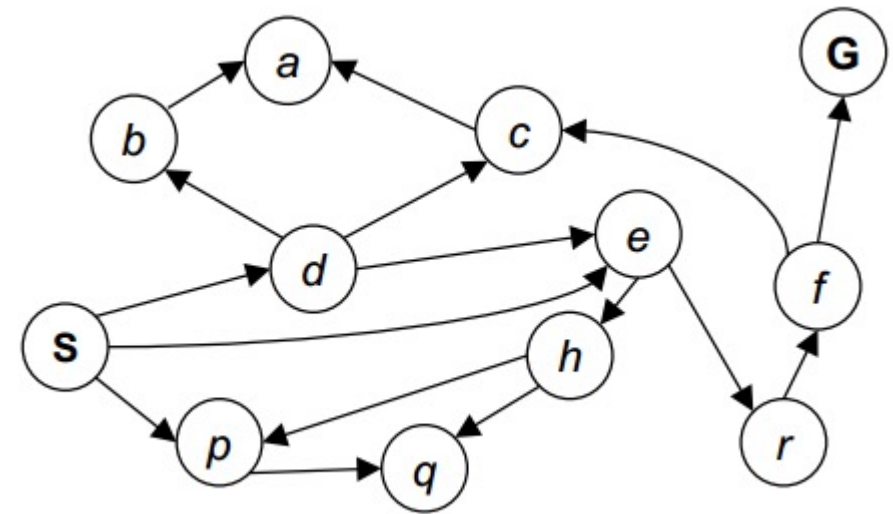
# Search Problem Formulation – Exercise 2

- 3 missionaries and 3 cannibals are on one side of a river
- A boat can hold one or two people.
- **NOT** allowed (on either side): # of cannibals > # of missionaries
- Find a way to get everyone to the other side

**Write a formulation for this search problem**

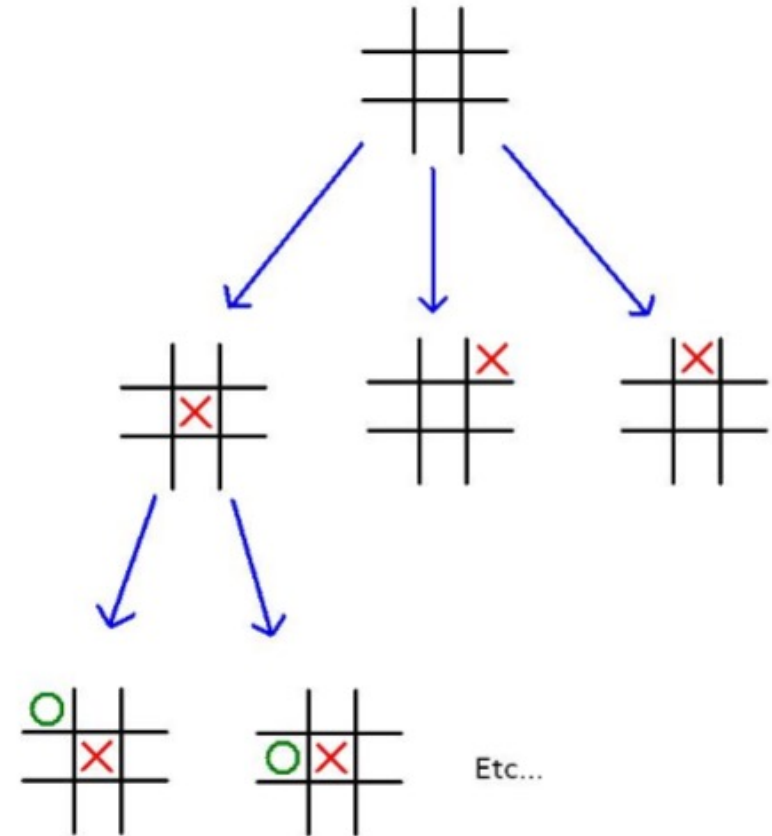
# State space graph

- Nodes are (abstracted) world configurations
- Arcs represent successors (action results)
- Goal test: one or a set of goal nodes
- Each state occurs only once!



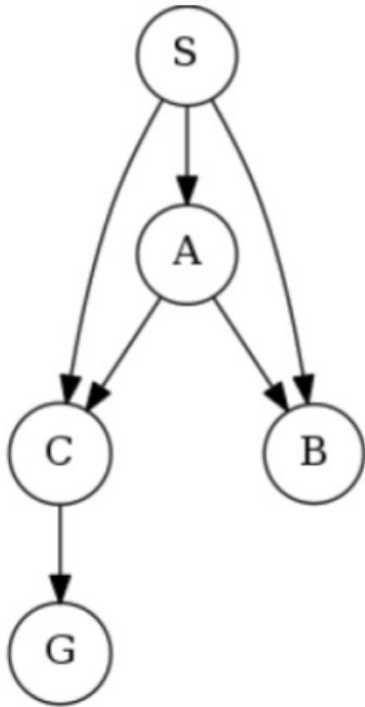
# Search Tree

- A "what if" tree of plans and their outcomes
- Root: initial state
- Children: correspond to successors



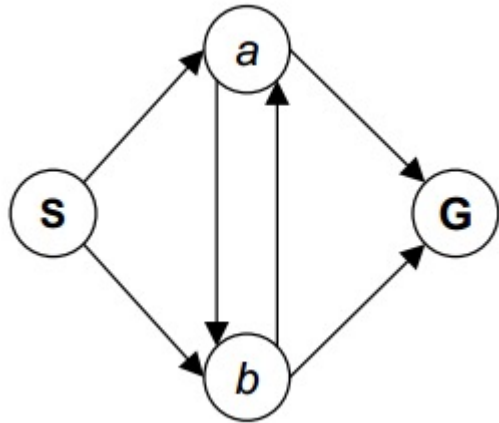
# Quiz

- How many nodes are there in this search tree?



# State Space Graph vs Search Tree

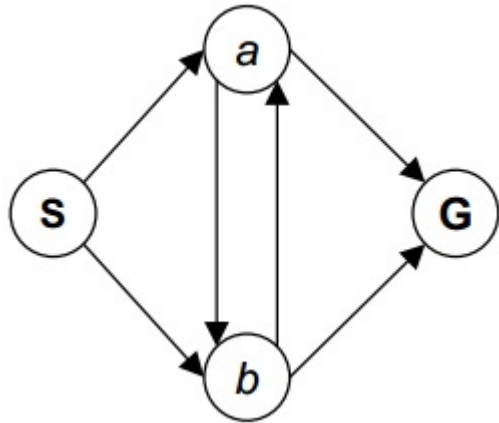
4-state graph



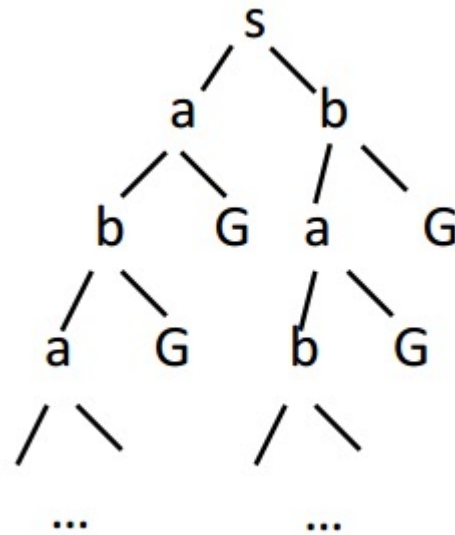
How big is the search tree?

# State Space Graph vs Search Tree

## 4-state graph

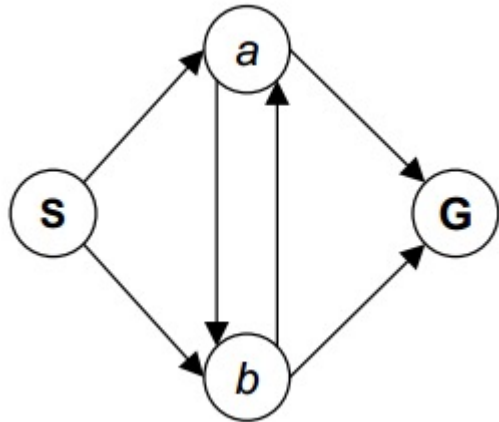


## How big is the search tree?

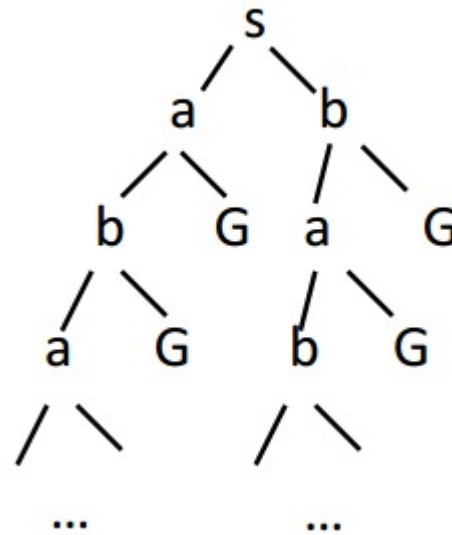
$$\infty$$


# State Space Graph vs Search Tree

## 4-state graph



## How big is the search tree?

 $\infty$ 

- Expand out potential tree nodes
- Maintain a fringe of partial plans under consideration
- Try to expand as few tree nodes as possible

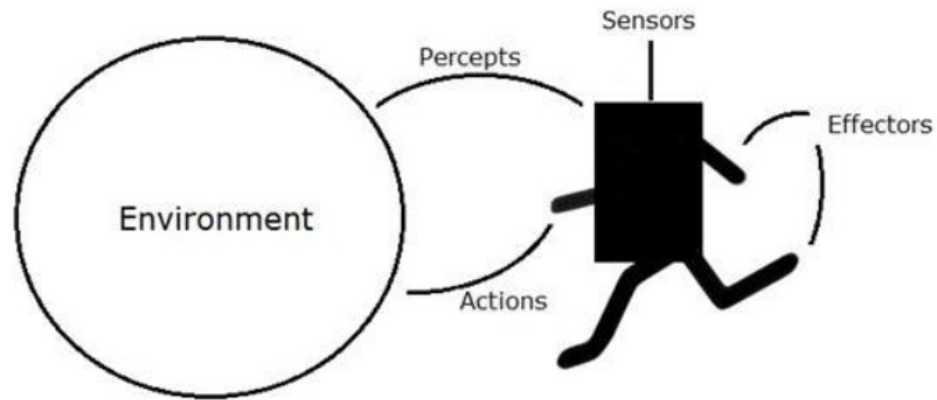
# Search Problem

- Before an agent can start searching for solutions, a **goal** must be identified and a well-defined **problem** must be formulated
- Search problem formulation
  - **Initial State**
  - **A state space**
  - **Actions:** a set of possible actions
  - **Successor function** (transition model):  $F(s_t, a_t) = s_{t+1}$  sometimes with a **path cost function**
  - **Goal test:** determine if solution is achieved.
- A **solution**: a sequence of actions (a **path**) that transform the initial state to a goal state



# Agents

An agent *perceives* its *environment* through *sensors* and *acts* upon it through *actuators*



# Agents

- **Rational agents:**
  - Choose actions that maximize the expected utility
  - Example 1: have a goal and a cost
    - Reach the goal with the lowest cost
  - Example 2: have numerical utilities, rewards, etc.
    - Take actions that maximize total reward over time
      - Reinforcement learning

# How to evaluate search algorithms

- **completeness**
- **optimality**
- **time complexity**
- **space complexity**

# Key concepts

- Frontier (nodes to expand)
- Generation: A node  $n$  is generated when a data structure representing its contents is created, i.e., when it is allocated in main memory.
- Expansion: A node  $n$  is expanded when all its children are generated.

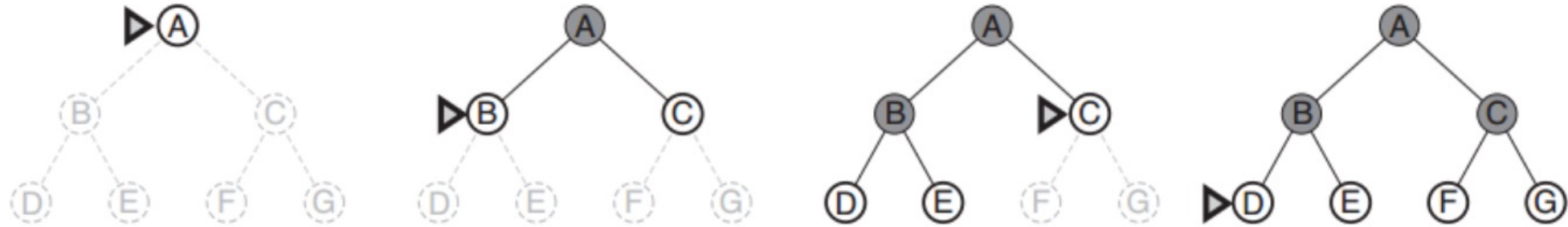
# Uninformed Search

- BFS
- DFS, Depth-Limited Search
- Iterative Deepening
- Uniform-cost search

# BFS

- BFS
  - Expands shallowest nodes first
  - Complete
  - Optimal for unit step costs
  - Time complexity (*exponential*): # of generated nodes
    - $b + b^2 + \dots + b^d = O(b^d)$  (*Goal test on generation* by default)
    - Goal test on expansion:  $O(b^{d+1})$
  - Space complexity (*exponential*):  $O(b^d)$ 
    - Frontier  $O(b^d)$  Nodes in last generated layer

# BFS

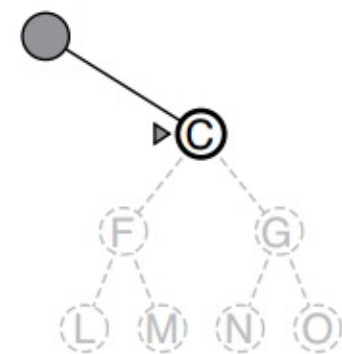
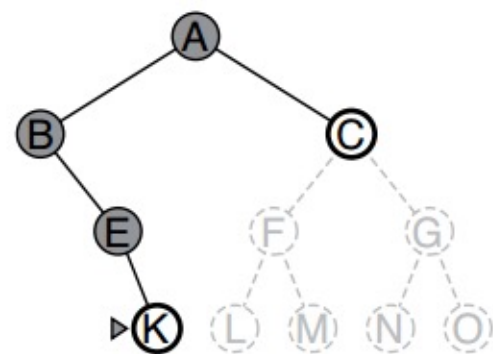
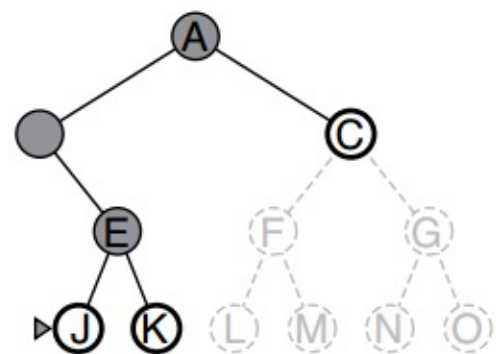
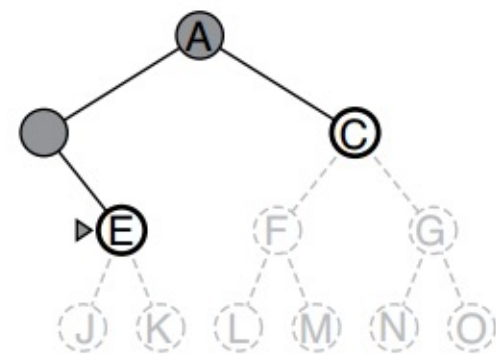
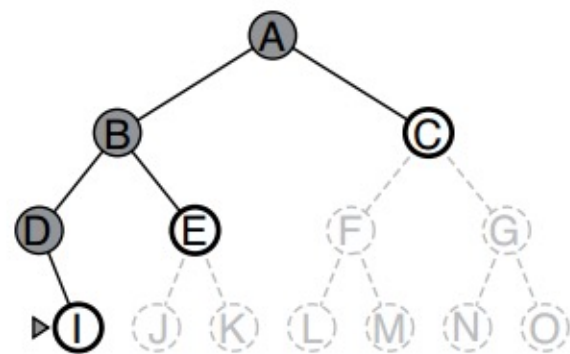
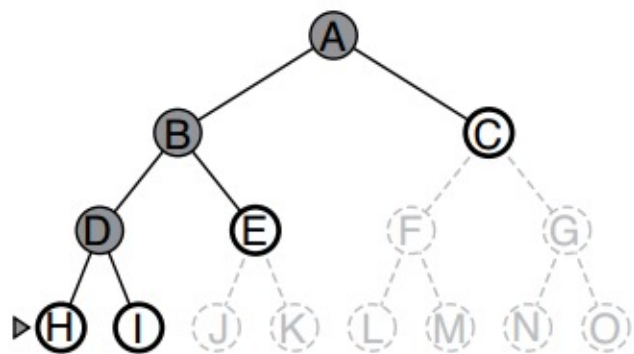
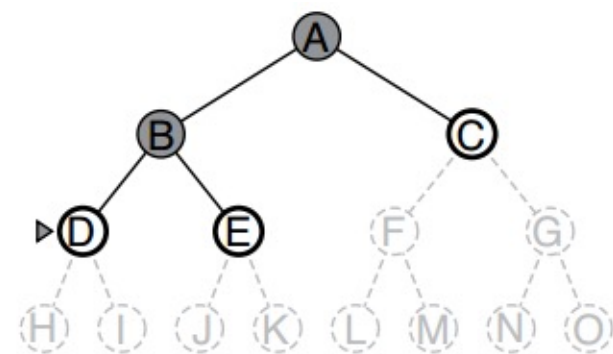
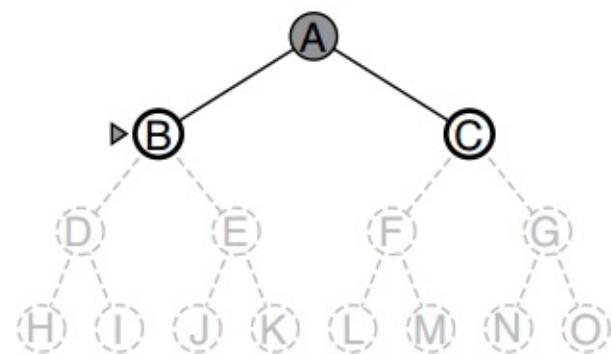
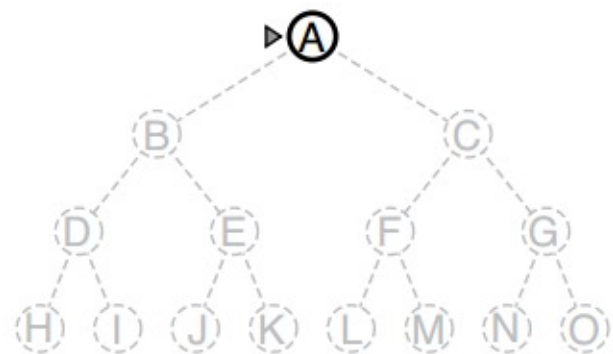


# DFS and Depth-Limited Search

- DFS
  - Not optimal
  - Not complete
  - Time complexity  $O(b^m)$  ( $m$ : maximum depth)
  - Space complexity  $O(bm)$  Fringe: path and siblings along the path
- Depth-Limited search: add a depth bound  $l$ 
  - Complete
  - Not optimal
  - Time complexity  $O(b^l)$
  - Space complexity  $O(bl)$



# DFS



# Iterative Deepening Search

- Depth-first search. Increase depth limits until a goal is found
- Complete; Optimal for unit step costs
- Space complexity of DFS:  $O(bd)$  (d: depth of the shallowest solution)
- Time complexity comparable to BFS
  - (Analysis: see lecture slides)

# Iterative Deepening

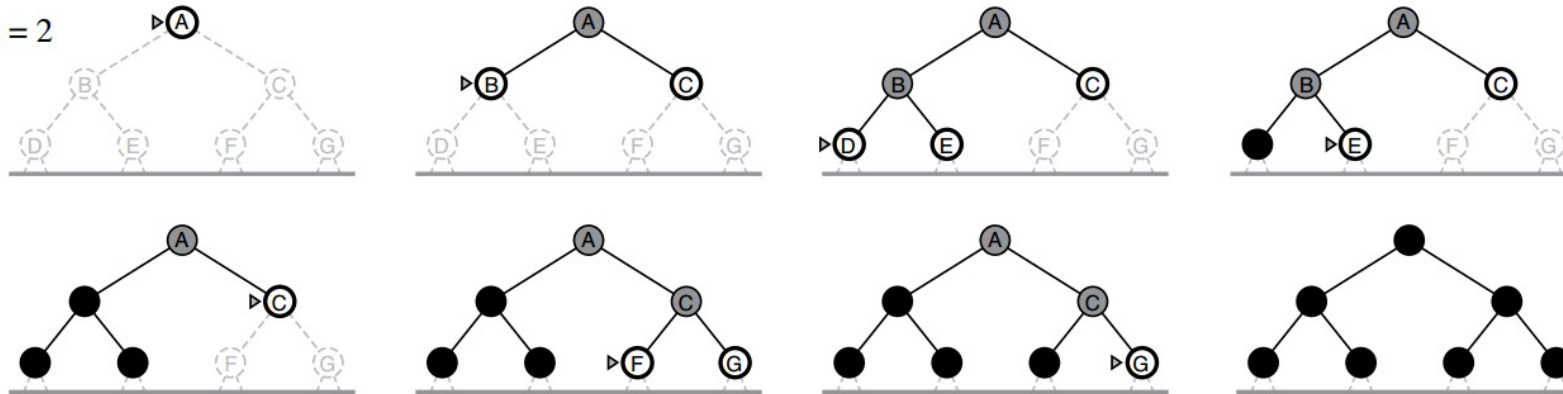
Limit = 0



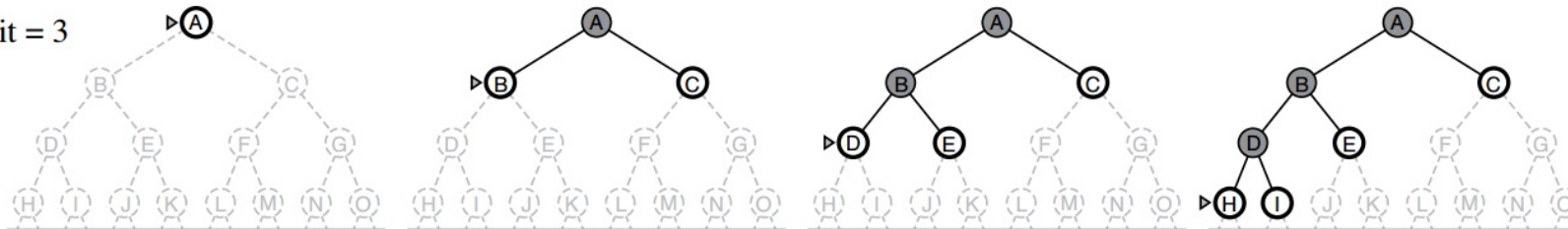
Limit = 1



Limit = 2



Limit = 3



# Uniform-Cost Search

- Expands the node with lowest path cost
- Uniform-cost search **keeps going** after a goal node has been generated.
  - Terminate after a goal node is expanded
- Optimal in general
  - Infinite loop if there is a path with an infinite sequence of zero-cost actions
- Complete
  - (If all step cost  $>$  a small positive constant  $\epsilon$ )

Example: <https://www.youtube.com/watch?v=XyoucHYKYSE>

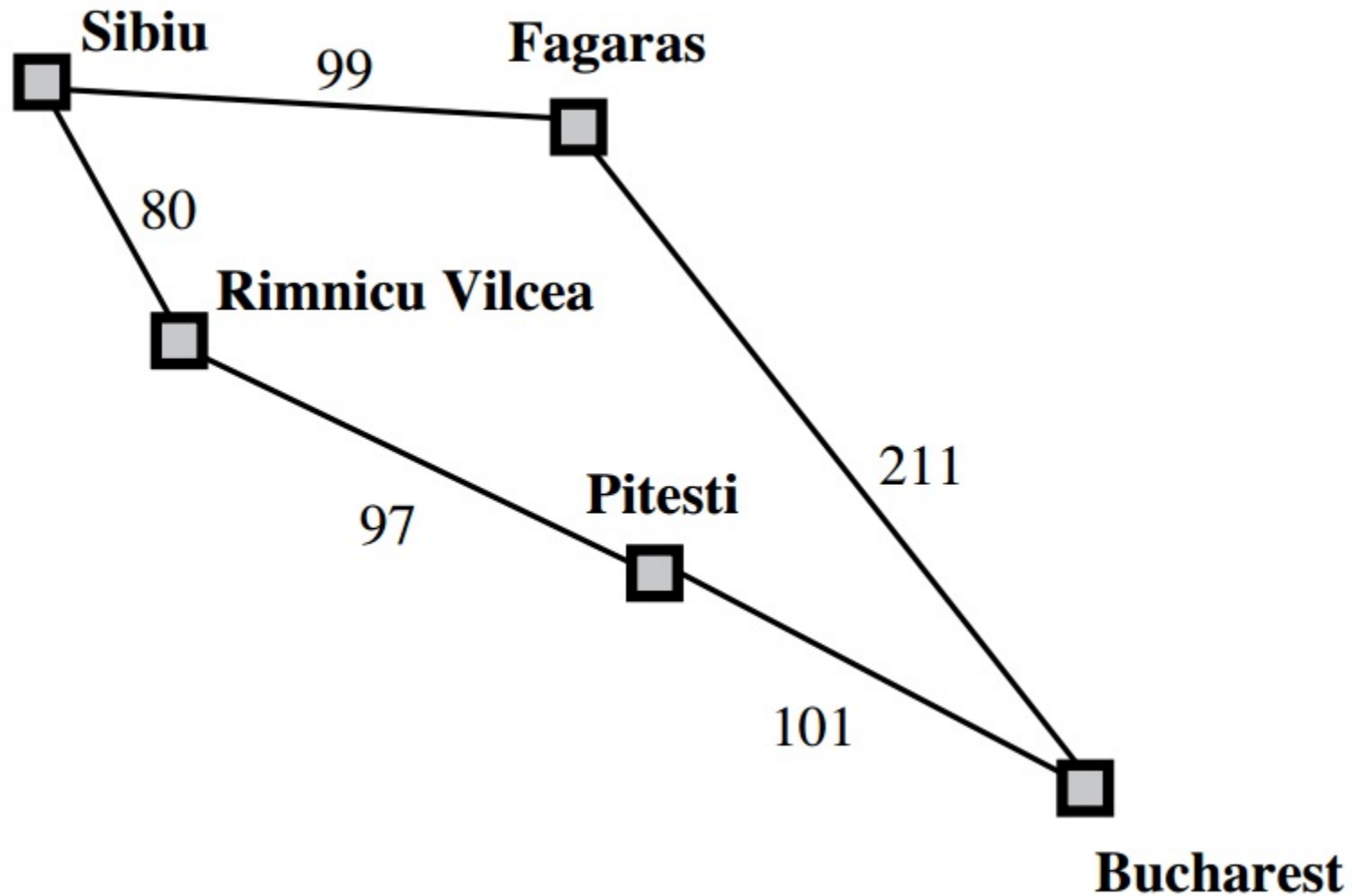
# Uniform-Cost Search

- Time Complexity (Worst Case)
  - $C$ : cost of optimal solution
  - Every action costs at least  $\epsilon$
  - Time complexity:  $O(b^{1+\lceil C/\epsilon \rceil})$ 
    - When all step costs are equal:  $O(b^{1+d})$  (test on expansion)
- Space Complexity (Worst Case)
  - Fringe: priority queue (priority: cumulative cost)
  - Worst case: roughly the last tier,  $O(b^{1+\lceil C/\epsilon \rceil})$

# Uniform-Cost Search

- Uniform-cost search and BFS
  - BFS stops after a goal node is generated (unless otherwise specified)
  - Uniform-cost search keeps going after a goal node has been generated

# Uniform-Cost Search - Example



# Bidirectional Search

- Run two simultaneous searches (BFS/Iterative Deepening)
  - One forward from the initial state
  - The other backward from the goal
- Replacing Goal test: Check whether the frontiers of the two searches intersect
  - The first found solution may not be optimal
    - Additional search is required to make sure there isn't short-cut across the gap! (Will show later)



# Bidirectional Search (cont'd)

- What if we have multiple goal states?
  - For explicitly listed goal states: construct a new dummy goal state
    - Dummy goal state's immediate predecessors are all the actual goal states
  - For abstract description (e.g. “no queen attacks another queen”)
    - Bidirectional search is difficult to use
- Time complexity & Space Complexity (Using two BFS)
  - $O(b^{d/2})$

# Exercise - Word Ladder

- Input:
  - *beginWord*
  - *endWord* (*endWord* != *beginWord*)
  - A dictionary
- Transformation:
  - wordA -> wordB (e.g. "hit"-> "hot")
  - Only one letter can be changed at a time.
  - wordB must be in dictionary
- Question:
  - Transform *beginWord* to *endWord*
  - How many transformations we need at least?
    - Return -1 if no such sequence
- **How do you solve it?**

# Exercise – Word Ladder

## Example

- *beginWord* = "hit"
- *endWord* = "cog",
- *dictionary* = ["hot","dot","dog","lot","log","cog"]
- Output: 4
  - "hit" -> "hot" -> "dot" -> "dog" -> "cog"

# Bidirectional Search (cont'd)

We mentioned

- The first found solution may not be optimal
  - Additional search is required to make sure there isn't short-cut across the gap!
- When does this happen?
  - What if "hot" and "log" are connected in the word ladder example?
    - "hit" -> "hot" -> "dot" -> "dog" -> "log"
    - "hit" -> "hot" -> "lot" -> "log"

# Comparison

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes <sup>a</sup>	Yes <sup>a,b</sup>	No	No	Yes <sup>a</sup>	Yes <sup>a,d</sup>
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes <sup>c</sup>	Yes	No	No	Yes <sup>c</sup>	Yes <sup>c,d</sup>

**Figure 3.21** Evaluation of tree-search strategies.  $b$  is the branching factor;  $d$  is the depth of the shallowest solution;  $m$  is the maximum depth of the search tree;  $\ell$  is the depth limit. Superscript caveats are as follows: <sup>a</sup> complete if  $b$  is finite; <sup>b</sup> complete if step costs  $\geq \epsilon$  for positive  $\epsilon$ ; <sup>c</sup> optimal if step costs are all identical; <sup>d</sup> if both directions use breadth-first search.