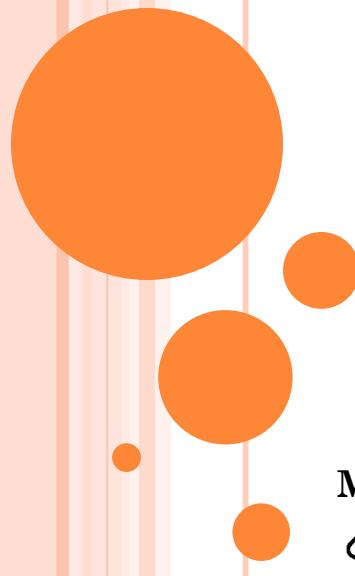


# CS480 – INTRODUCTION TO ARTIFICIAL INTELLIGENCE

TOPIC: SEARCH  
CHAPTER: 3



**Mustafa Bilgic**



<http://www.cs.iit.edu/~mbilgic>



<https://twitter.com/bilgicm>

# OUTLINE

- Problem solving agent
- The search process
- Tree search
- Graph search
- Complexity analysis

# MOTIVATION

- We have discussed several types of agents
  - Reflex agents, goal-based agents, utility-based agents, etc.
- Reflex agents act purely based on current percept, which is limited
- We now discuss, in detail, the goal-based agents, where a sequence of actions are necessary to reach a desired world state
- Actions have associated costs with them and we'd like to minimize the total cost to reach the goal state

# REPRESENTATION

- The world is represented as an atomic state
- An action takes us from one world state to another
- The environment is
  - Observable
  - Discrete
  - Deterministic
  - Known

## AN EXAMPLE

- We are now in Chicago
- We'd like to drive to Pittsburgh
- We represent the world as “In X” where X is a member of a predefined set of major cities in US
- An action takes us from being in one major city (a world state) to being in a neighboring major city (another world state)
- The cost of the action is the distance traveled
- We would like to travel from Chicago to Pittsburgh while minimizing the distance traveled

# THE SEARCH PROCESS

- We will start at an initial state
- At each state, there is a set of actions we can perform
  - The action that can be performed can be different depending on the current world state
- We have a test that can specify whether a given state is a goal state
- Each action has a cost
- We want to find the optimal solution, i.e., the minimum-cost sequence to reach the goal state

# MORE FORMALLY

- The **initial state**
  - $In(Chicago)$
- A set of **actions**
  - $Go(Milwaukee)$ ,  $Go(Madison)$ ,  $Go(Bloomington)$ , etc.
- A **transition model**:  $Result(s, a)$ :  $S \rightarrow S$ 
  - $Result(In(Chicago), Go(Madison)) = In(Madison)$
- The **goal test**
  - $\{In(Pittsburgh)\}$
- A **path cost**: Sum of step costs  $c(s, a, s')$ 
  - $c(In(Chicago), Go(Madison), In(Madison)) = 150$

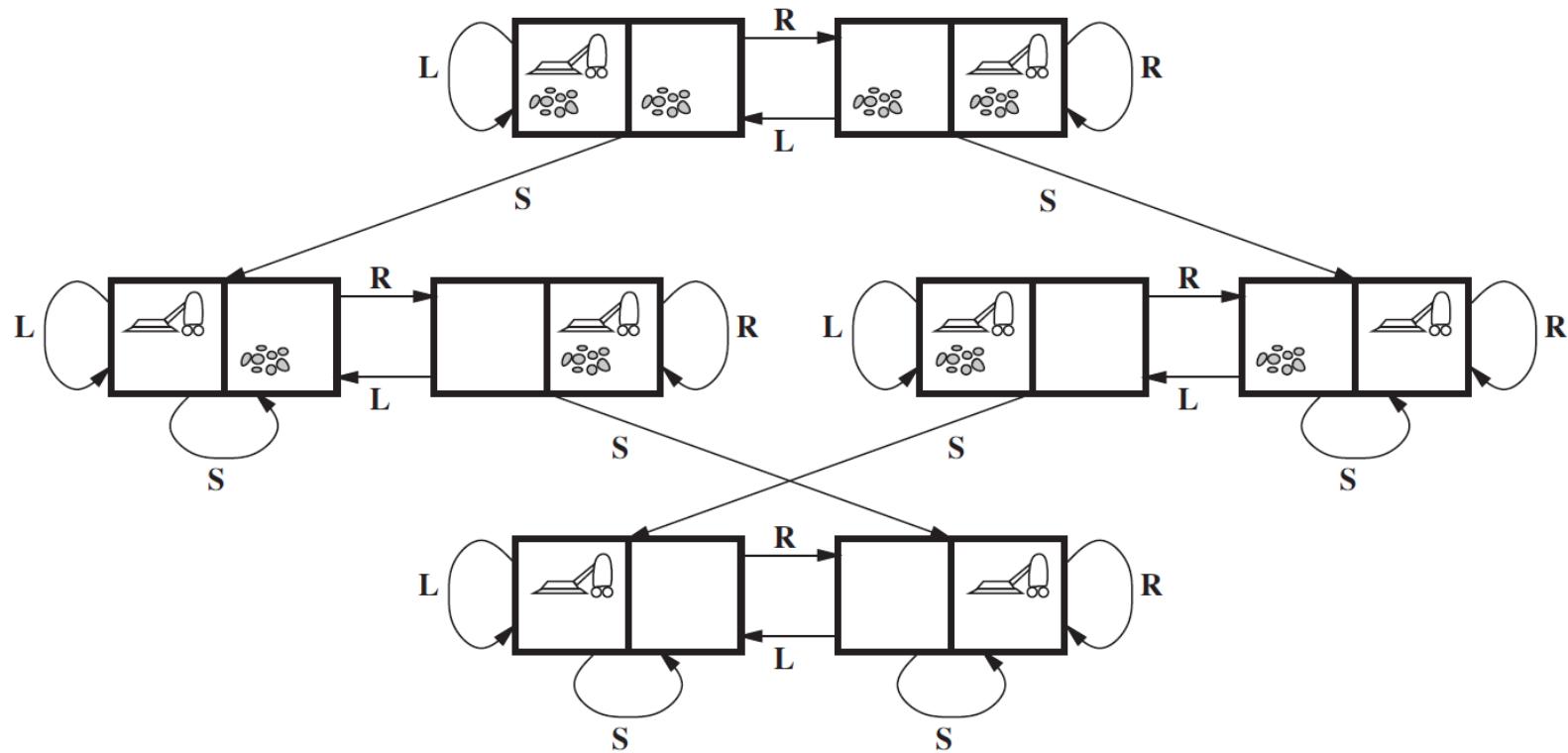
# ASSUMPTIONS AND SIMPLIFICATIONS

- For the Chicago to Pittsburgh problem, we made several assumptions and simplifications
- **What are some of these?**

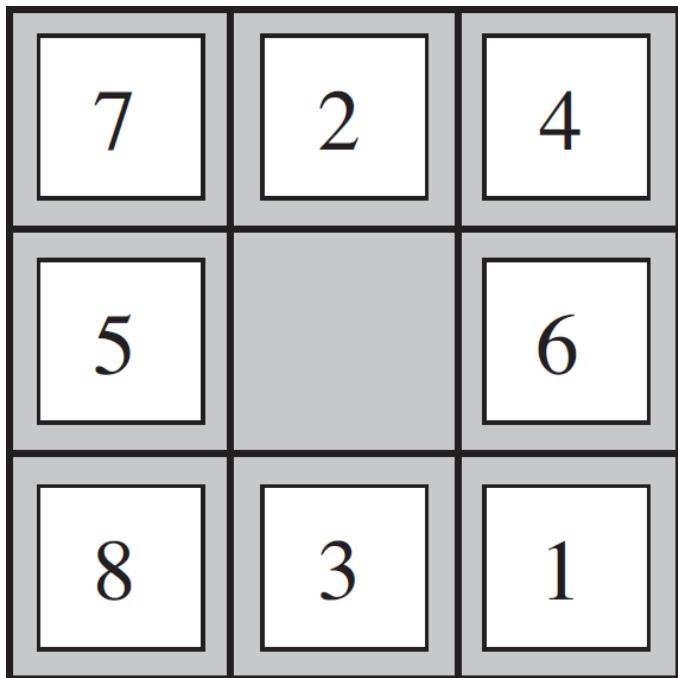
# THE VACUUM WORLD

- **States:** The cross-product of the agent's location and dirt locations
  - How many possible states?
- **Initial state:** Any state
- **Actions:** Left, Right, and Suck
- **Transition model:** Expected effects. Except Left on the left square, Right on the right square, and Suck on a clean square has no effect
- **Goal test:** Test whether all squares are clean
- **Path cost:** Each step costs 1; the path cost is then the total number of steps

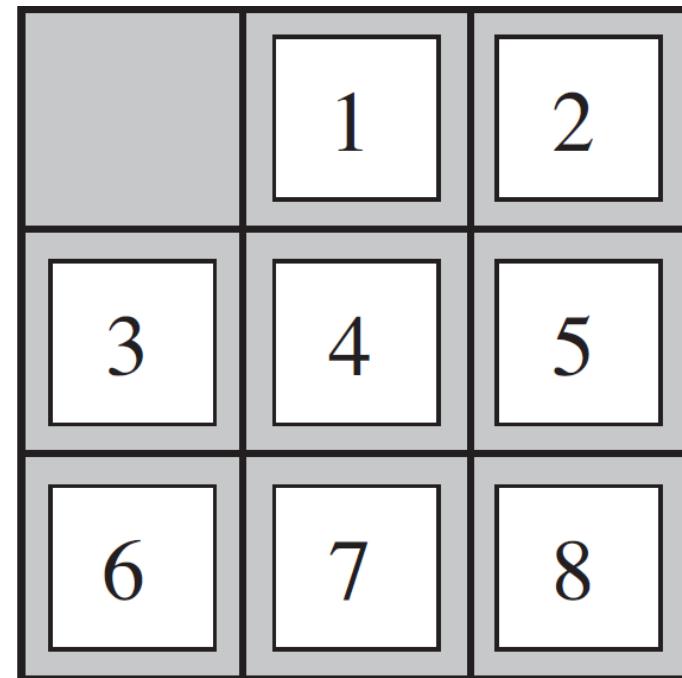
# THE VACUUM STATE SPACE



# THE 8-PUZZLE



Start State

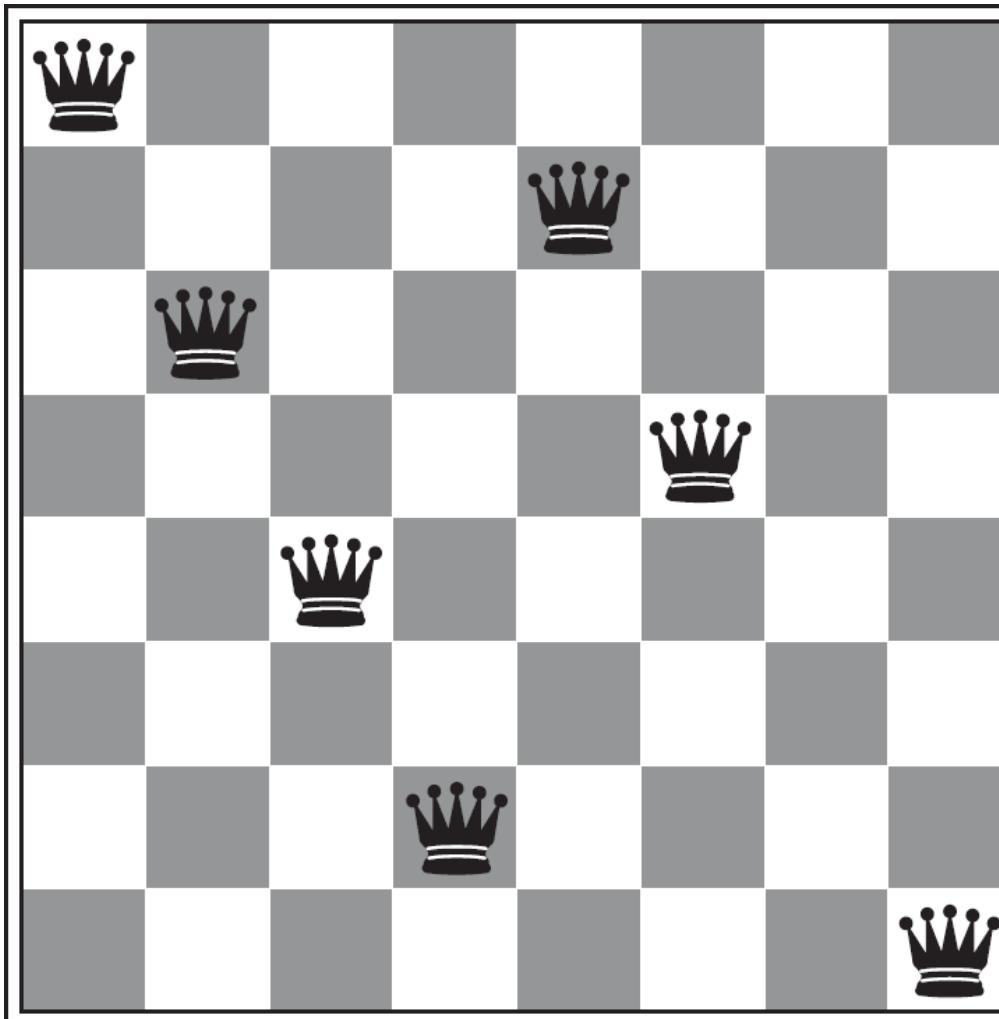


Goal State

# THE 8-PUZZLE

- **States:** All possible locations of the eight tiles and the blank tile
  - How many?
- **Initial state:** Any state
- **Actions:** Left, Right, Up, Down, except on the edges
- **Transition model:** The state with the blank the numbered tile switched
- **Goal test:** Blank on the top left, others are ordered in increasing order
- **Path cost:** Each step costs 1; the path cost is then the total number of steps

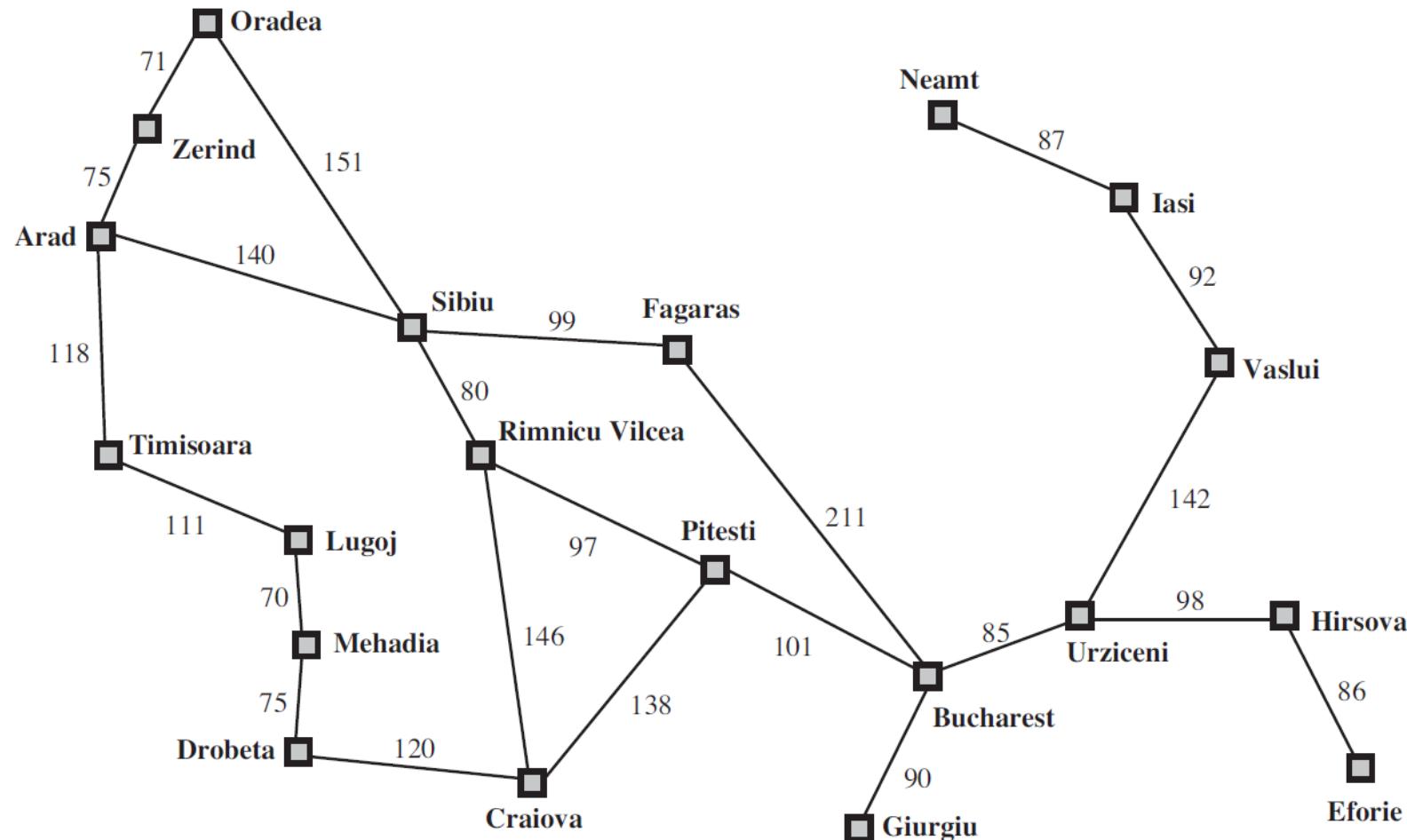
# THE 8-QUEENS PROBLEM



# THE 8-QUEENS PROBLEM

- **States:** Any arrangement of 0 to 8 queens on the board
  - How many?
- **Initial state:** No queens on the board
- **Actions:** Add a queen to an empty square
- **Transition model:** The board with the added queen
- **Goal test:** 8 queens on the board, none attacked
- **Path cost:** Each step costs 1; the path cost is then the total number of steps

# ROMANIA



# ROMANIA

- **States:** Any city
- **Initial state:** Arad
- **Actions:** Go to a neighboring city
- **Transition model:** The city that was just traveled
- **Goal test:** Bucharest
- **Path cost:** Step costs are distances between the neighboring cities; path cost is the total distance traveled

# LET'S SOLVE TOY VERSIONS

- The 4-Queens problem
  - Initial state: Empty board
  - Action: Place a queen on the left most empty column
- The 3-Puzzle problem
  - Initial state: First row: 2 – Blank, Second row: 3 – 1
  - Action: Move the blank tile

# SEARCH

- The key algorithm of the goal-based agent is *search*
  1. Start with the initial state
  2. Expand that state through possible actions
  3. Pick an unexplored state
  4. If goal, return solution; otherwise, go to step 2

# TREE SEARCH

```
function TREE-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier
```

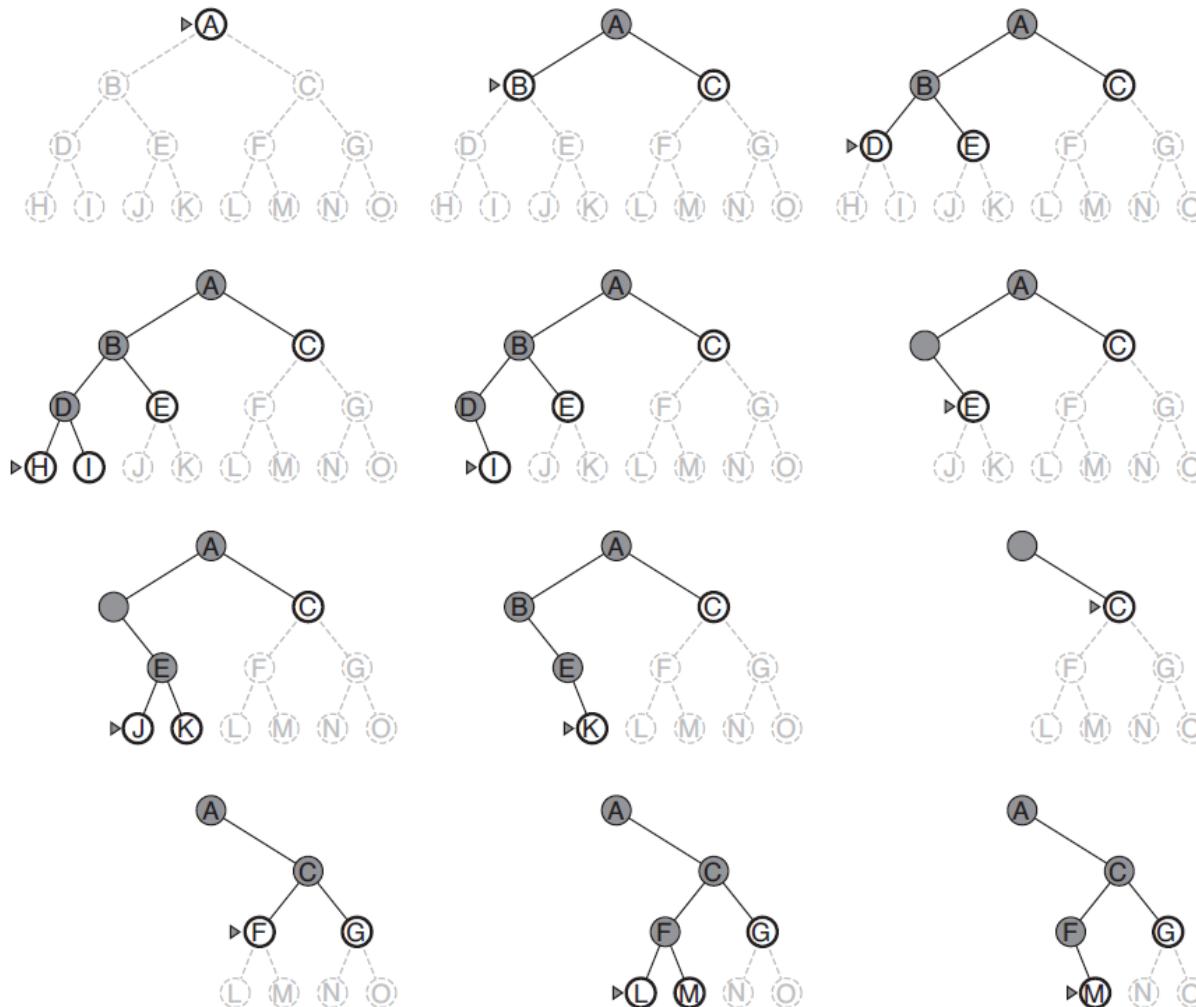
The key question is which leaf node to pick for expansion.

## TWO FAMOUS AND IMPORTANT SEARCH STRATEGIES

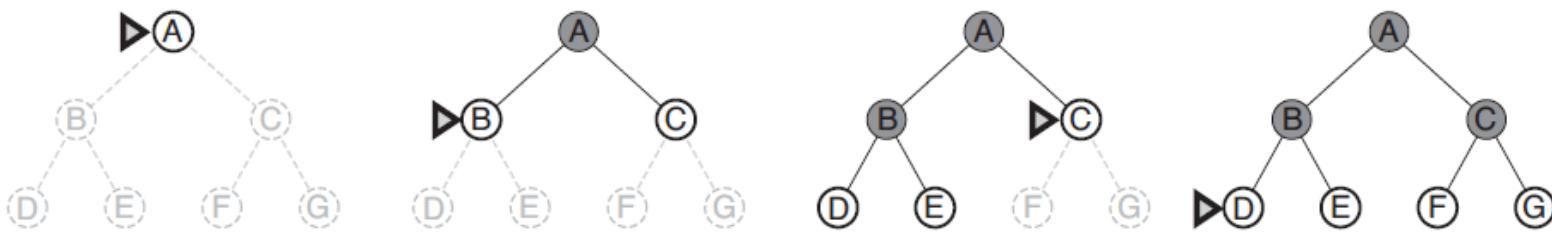
- Depth-first search
  - i.e., go as deep as possible
  - **frontier** = LIFO queue (stack)
- Breadth-first search
  - i.e., go broad; expand all the nodes at level  $i$  before expanding the nodes at level  $i+1$
  - **frontier** = FIFO queue

This is the textbook version; we do it a bit differently. Why?

# DEPTH-FIRST SEARCH



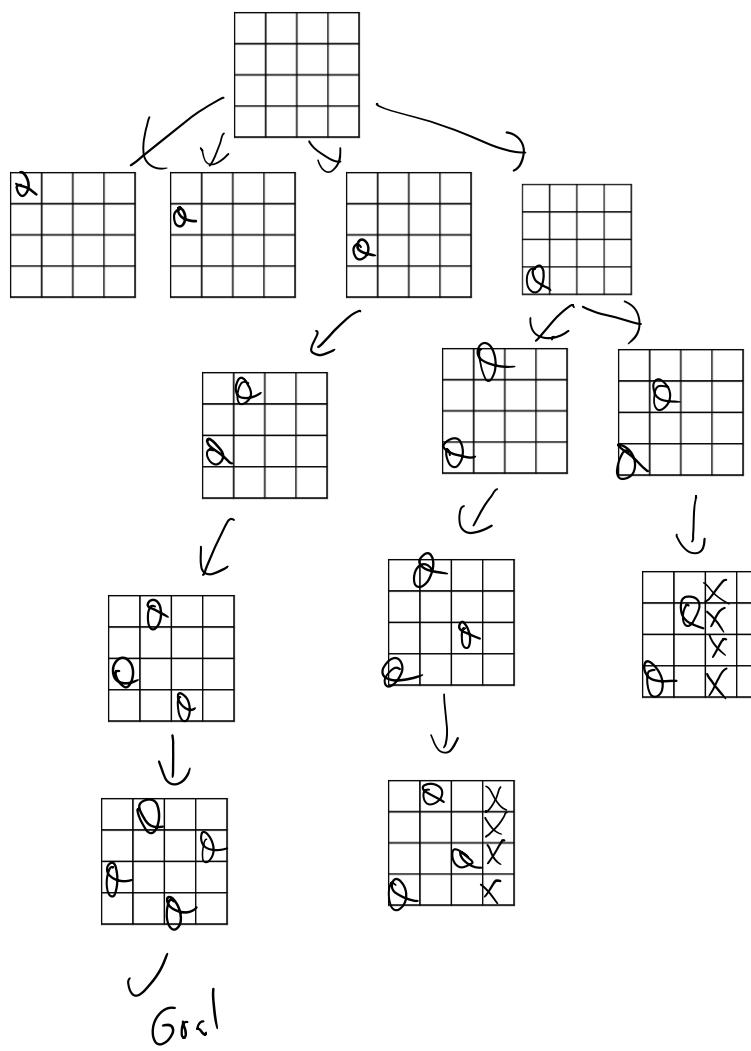
# BREADTH-FIRST SEARCH



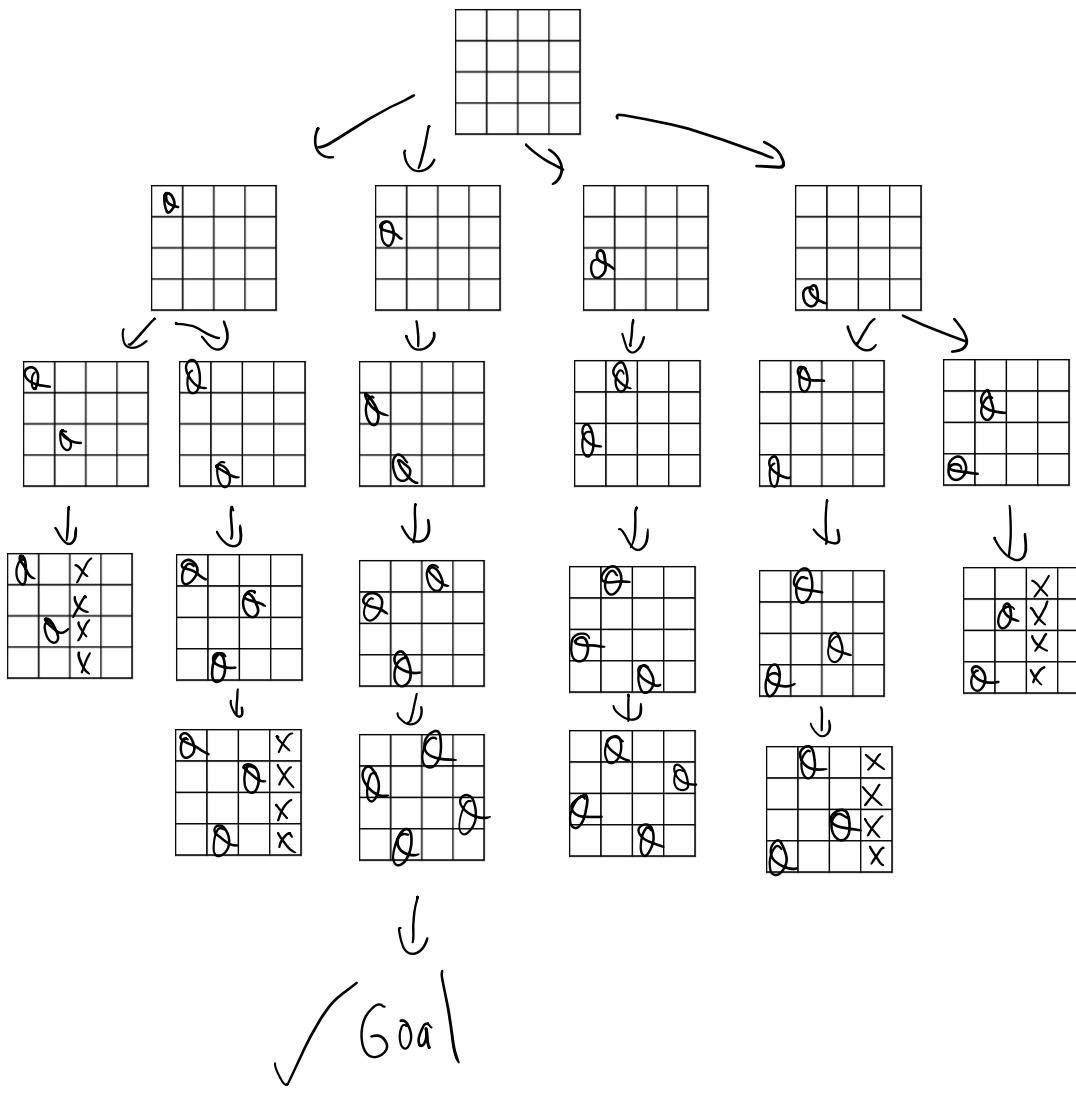
# LET'S SEE THEM IN ACTION

- 4-Queens
- 3-Puzzle
- Travel
  - A → B, C
  - B → D, E
  - C → F, G
  - D → H, I
  - E → J, K
  - F → L, M
  - G → N, O
  - H, I, J, K, L, M, N, O are terminal states
  - Start: A, Goal: O

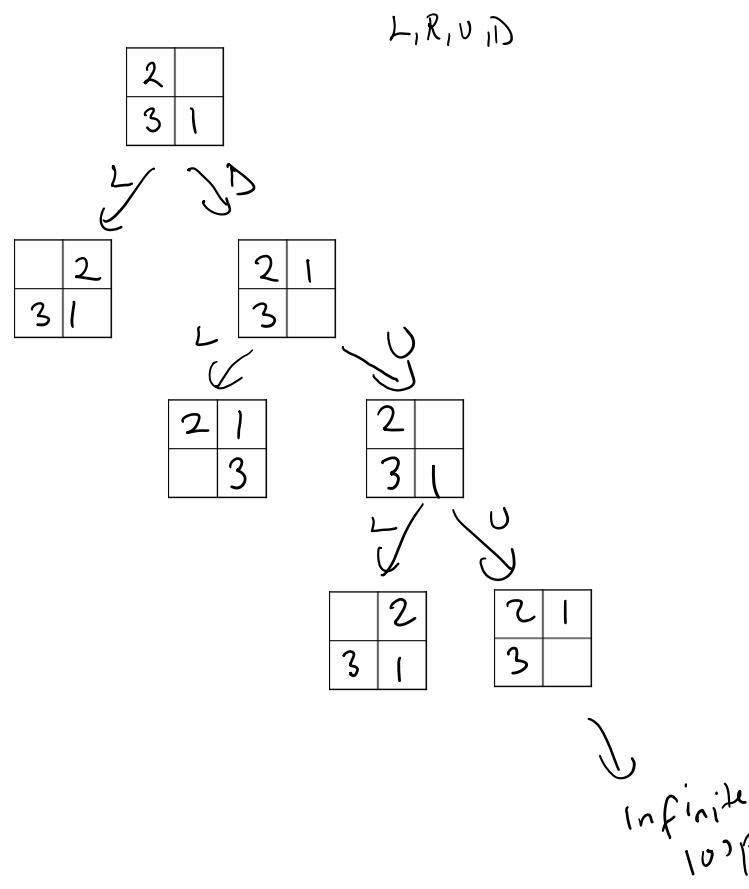
# 4 Queens - DFS



## 4 Queens - BFS

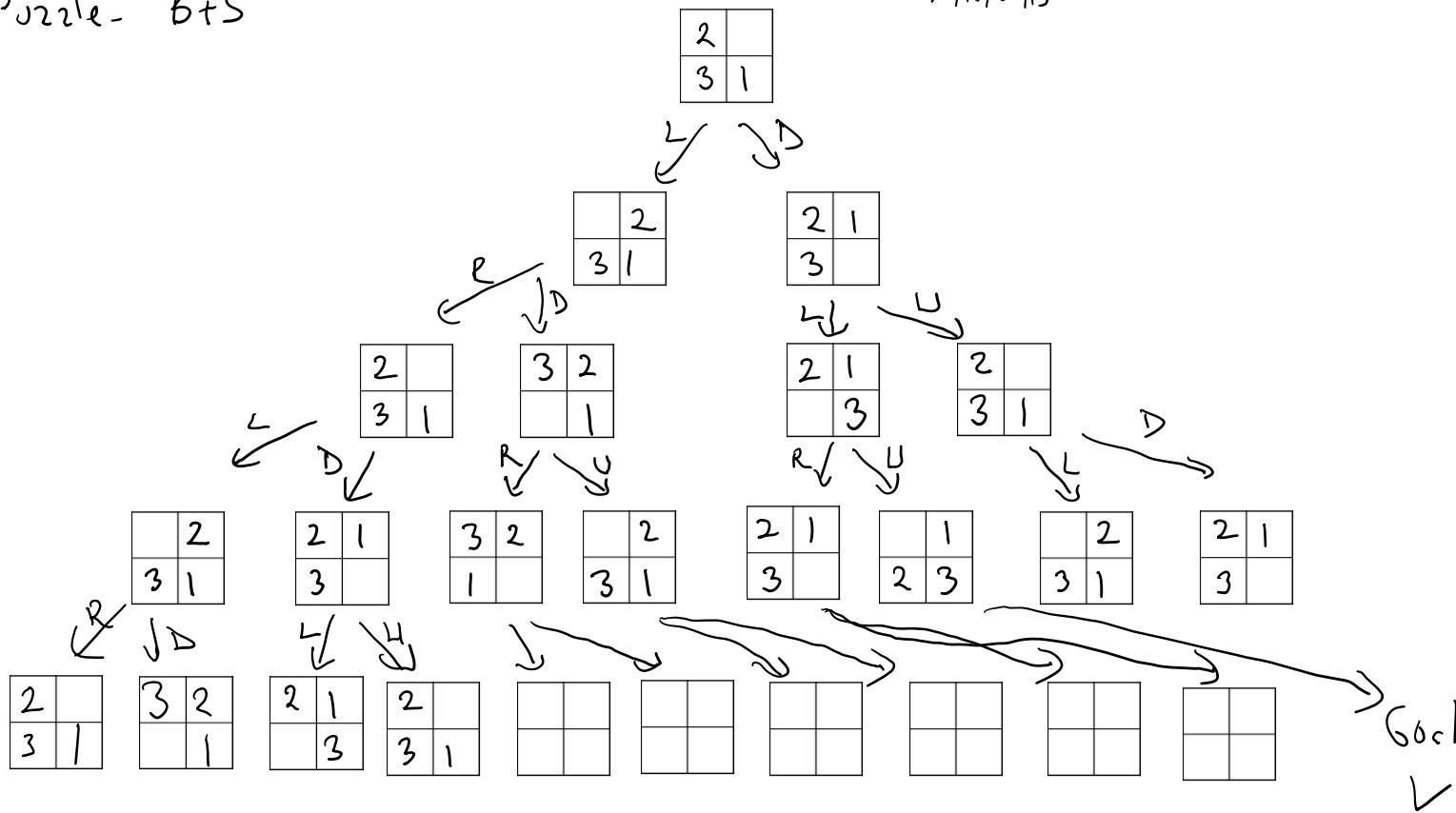


### 3 puzzle - DFS



3 puzzle - BFS

L, R, U, D

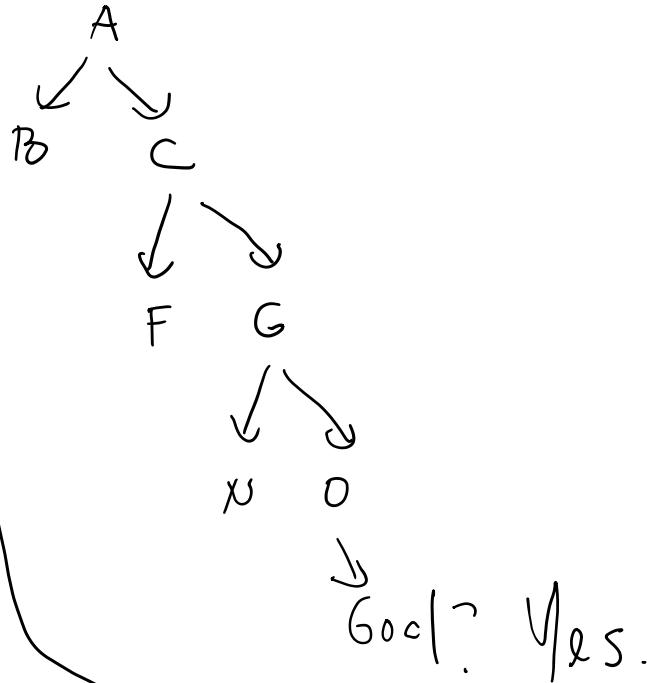
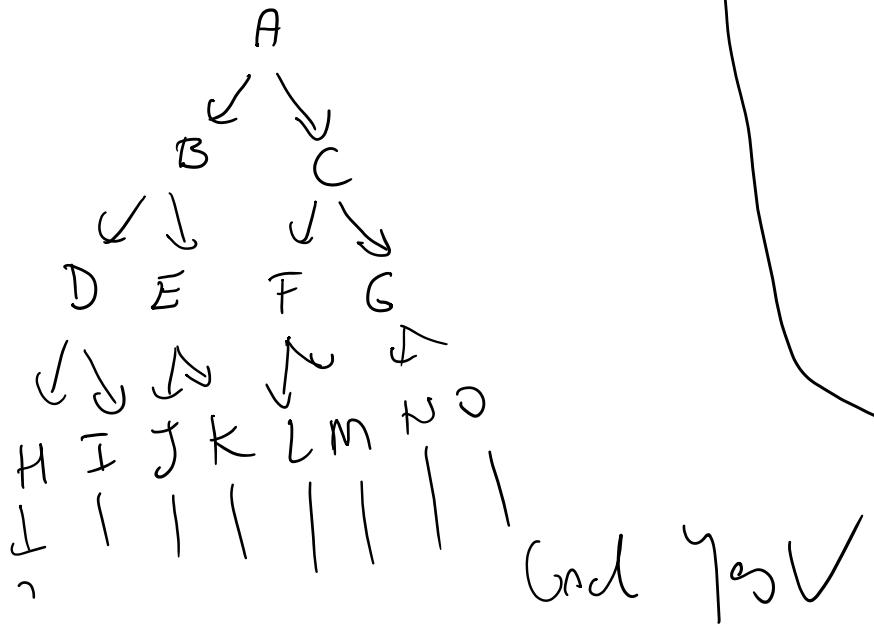


DFS

Frontier: ~~A B C F G N O~~

Frontier: ~~A B C D E F G H I J K~~

BFS



# COMPLEXITY

- We're concerned with
  - Time complexity
  - Space complexity
  - Completeness (if there is a solution, can it find it?)
  - Optimality (is the found solution the optimal one?)
- Three relevant quantities
  - The number of nodes expanded
  - The maximum size of the **frontier**
  - The cost of the solution

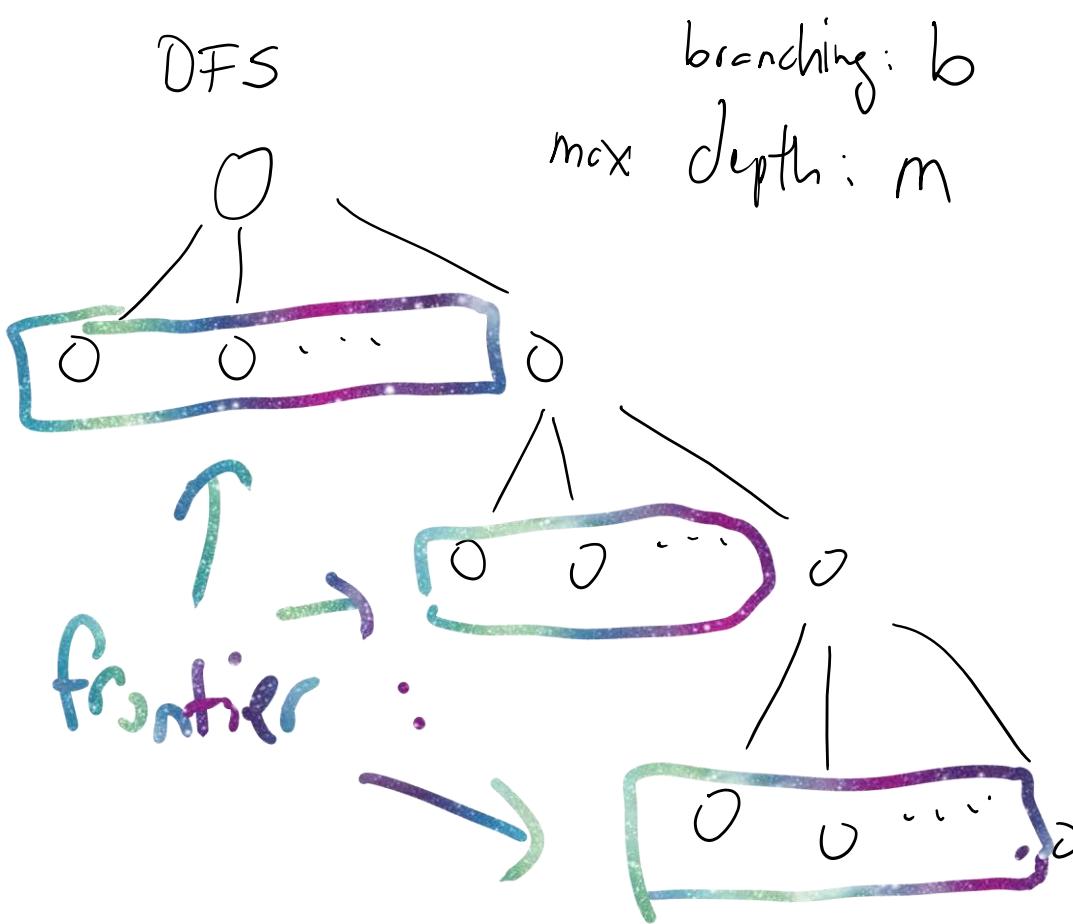
# ANALYSIS – TREE SEARCH

Criterion	Depth-first	Breadth-first
Complete?		
Time		
Space		
Optimal?		

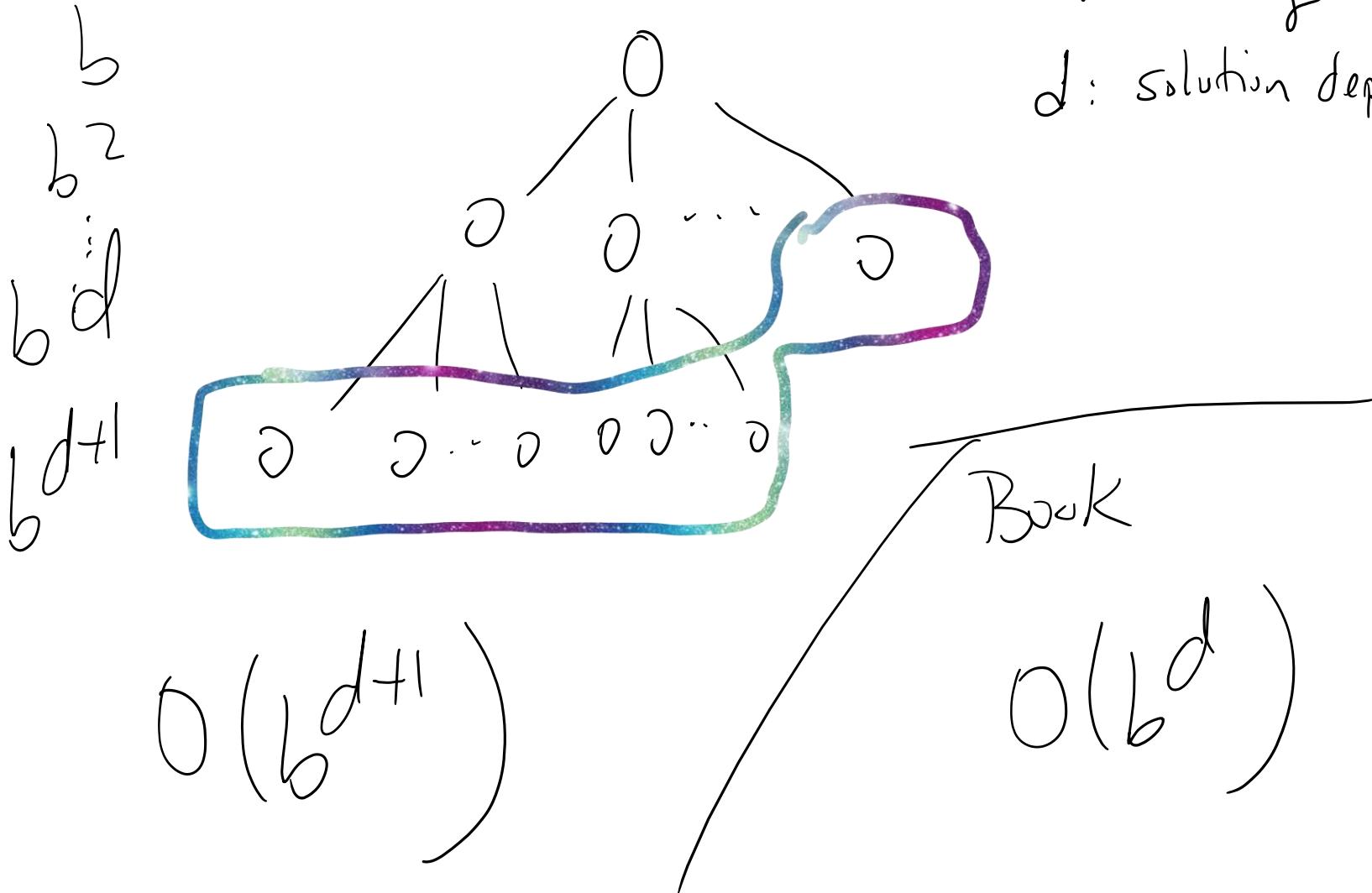
# ANALYSIS – TREE SEARCH

Criterion	Depth-first	Breadth-first
Complete?	No	Yes
Time	$O(b^{m+1})$	$O(b^{d+1})$
Space	$O(b^m)$	$O(b^{d+1})$ or $O(b^d)$
Optimal?	No	Yes, if all steps have equal cost

$$b^{-1} + b^{-1} + \dots + b^{-1} = O(b^m)$$



BFS



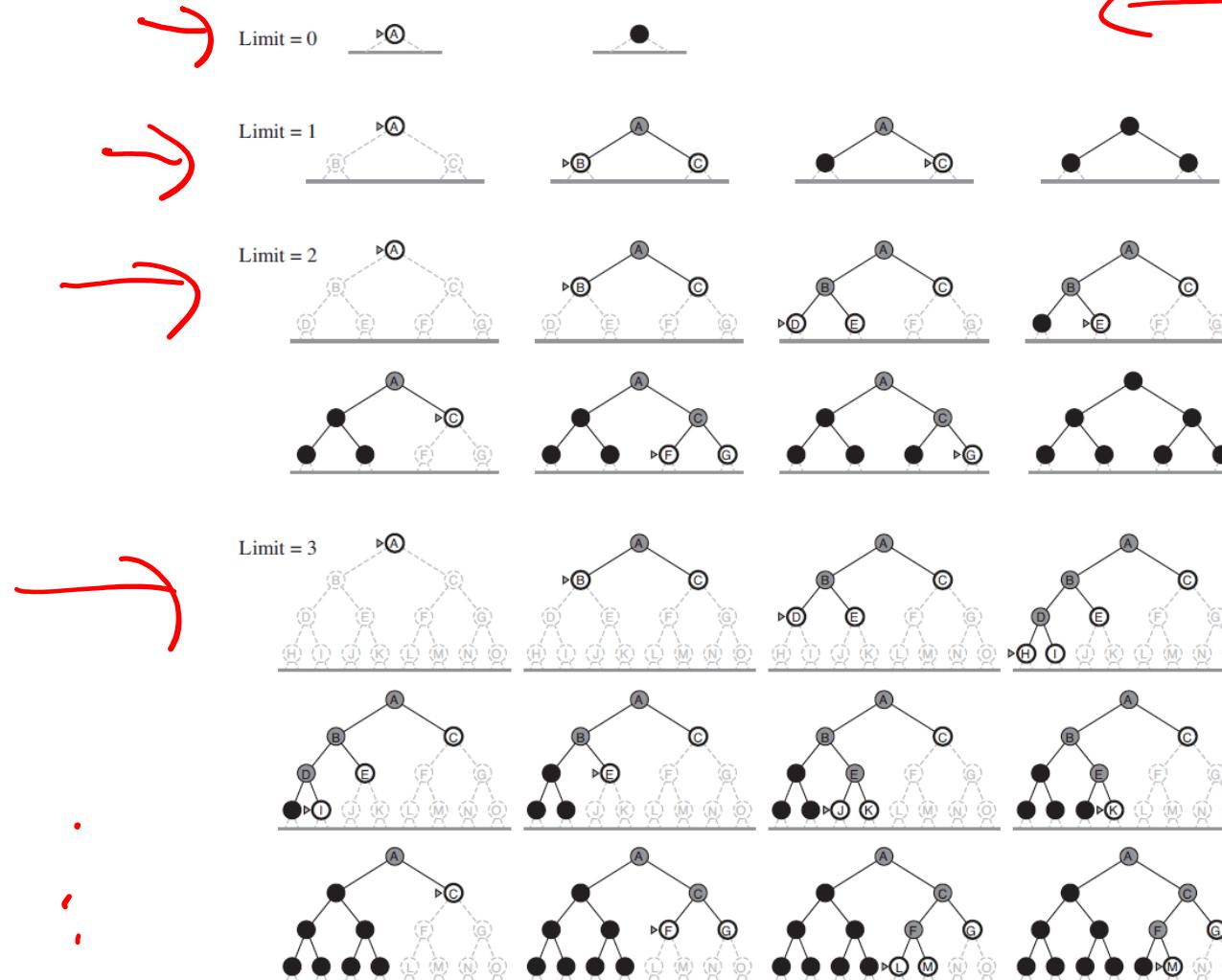
# MEMORY IS A BIG PROBLEM

- Breadth-first search has an exponential memory requirement
  - Depth-first search requires polynomial size memory, but it is not complete and it is not optimal
  - What can we do?
    - Depth-limited search
      - Impose a limit on the depth
      - Is it complete? Optimal?
    - Iterative-deepening search
      - Start with a depth limit of 0, and run depth-limited search; if no goal found, increase the depth limit and repeat the process
      - Is it complete? Optimal?
- Yes, Yes, if step costs are identical*

This is the textbook version; we do the DFS part differently.

What are the time and space complexities for IDS?

## ITERATIVE-DEEPENING SEARCH



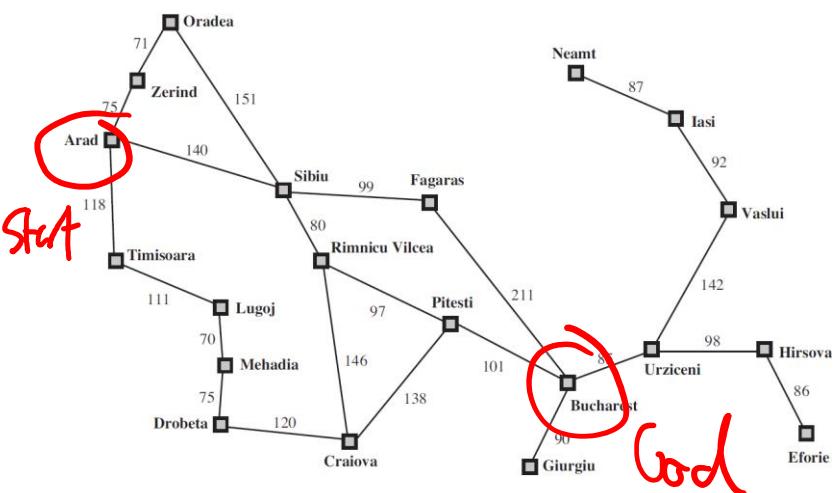
# WHAT IF THE STEP COSTS ARE NOT IDENTICAL?

- BFS and Iterative Deepening are not guaranteed to be optimal anymore
  - Define  $f(n) = h(n) + g(n)$ 
    - $h(n)$  : Cost estimate from  $n$  to the goal
    - $g(n)$  : Cost from the initial state to  $n$
1. Uniform-cost search
    - Expand using  $g(n)$
  2. Greedy best-first search
    - Expand using  $h(n)$
  3. A\* search
    - Expand using  $f(n)$

# THE RUNNING EXAMPLE

Edge costs

$h(n)$

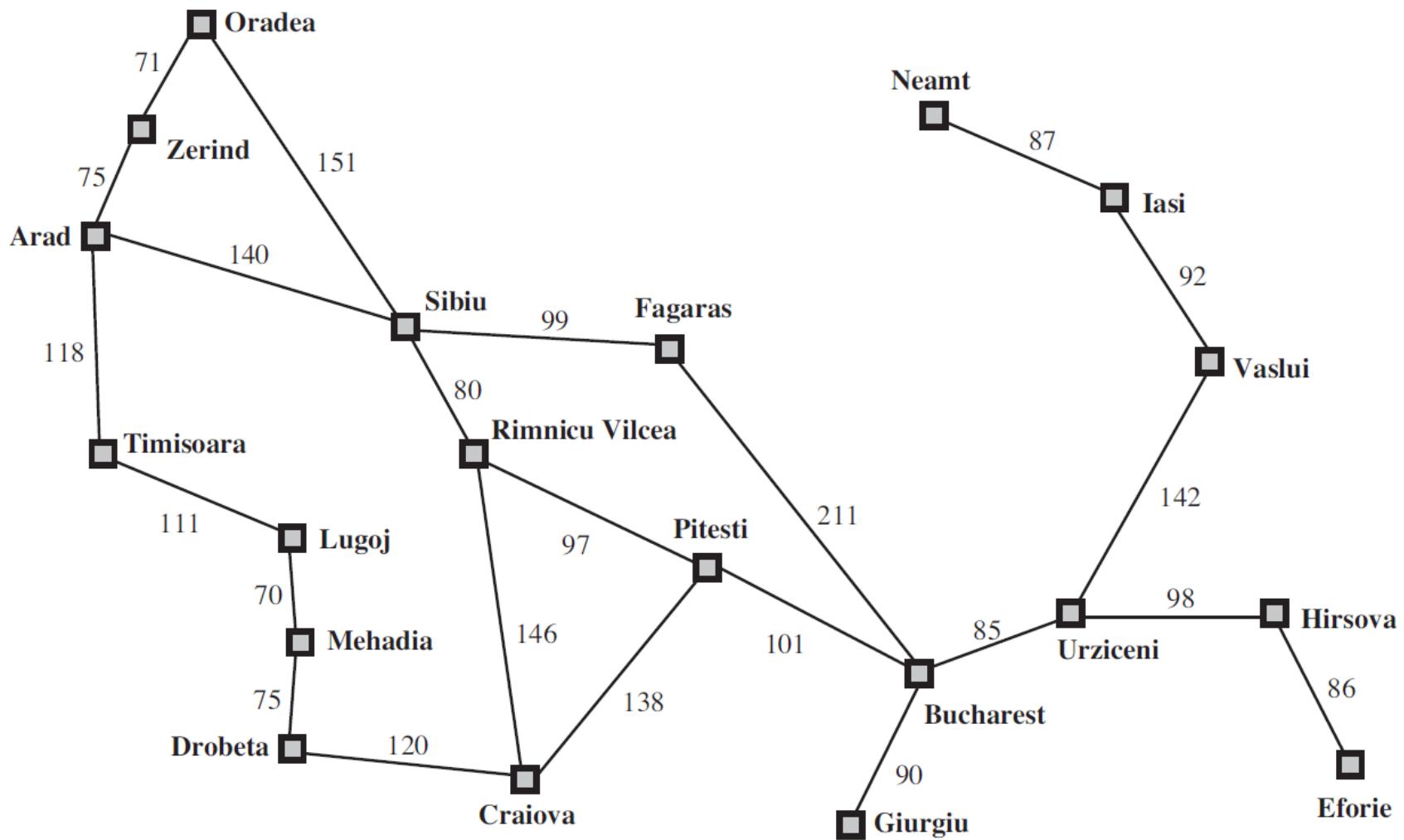


<b>Arad</b>	366
<b>Bucharest</b>	0
<b>Craiova</b>	160
<b>Drobeta</b>	242
<b>Eforie</b>	161
<b>Fagaras</b>	176
<b>Giurgiu</b>	77
<b>Hirsova</b>	151
<b>Iasi</b>	226
<b>Lugoj</b>	244

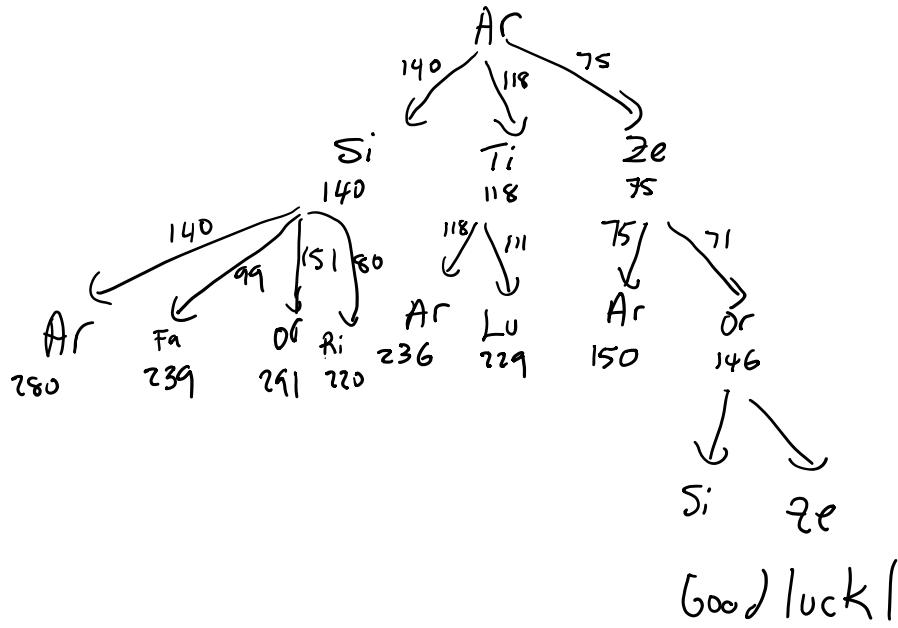
<b>Mehadia</b>	241
<b>Neamt</b>	234
<b>Oradea</b>	380
<b>Pitesti</b>	100
<b>Rimnicu Vilcea</b>	193
<b>Sibiu</b>	253
<b>Timisoara</b>	329
<b>Urziceni</b>	80
<b>Vaslui</b>	199
<b>Zerind</b>	374

# UNIFORM-COST SEARCH

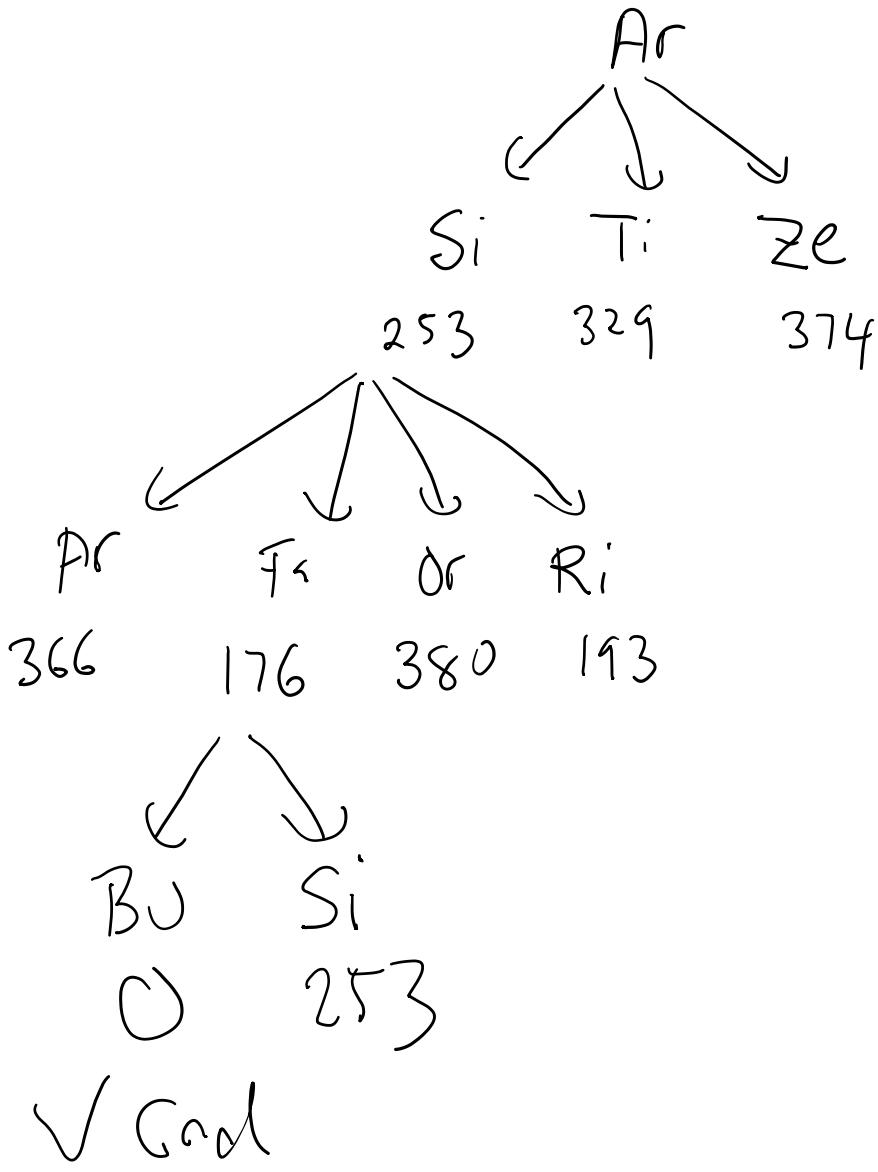
- Arad to Bucharest



VCS  
 $g(n)$

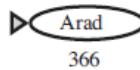


GBFS  
 $h(n)$

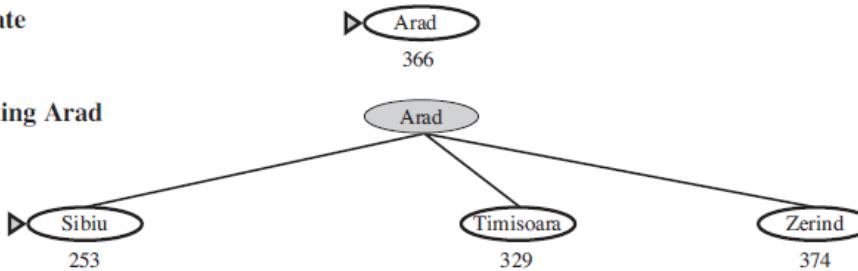


# GREEDY BEST-FIRST SEARCH

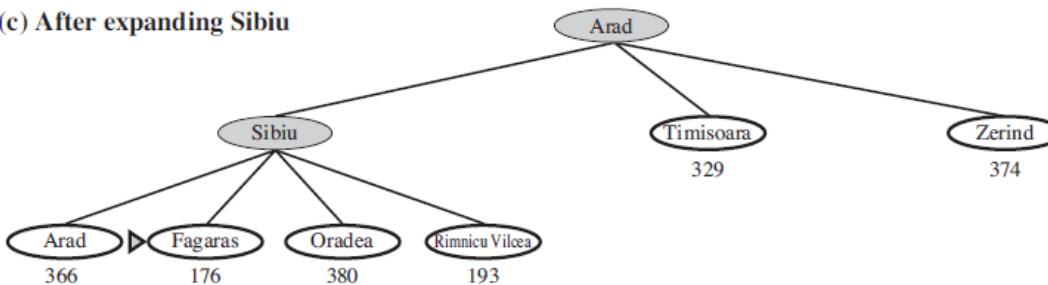
(a) The initial state



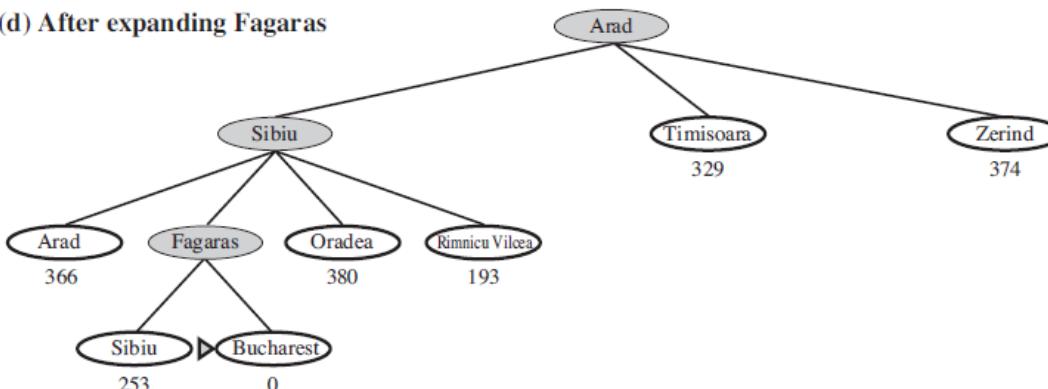
(b) After expanding Arad

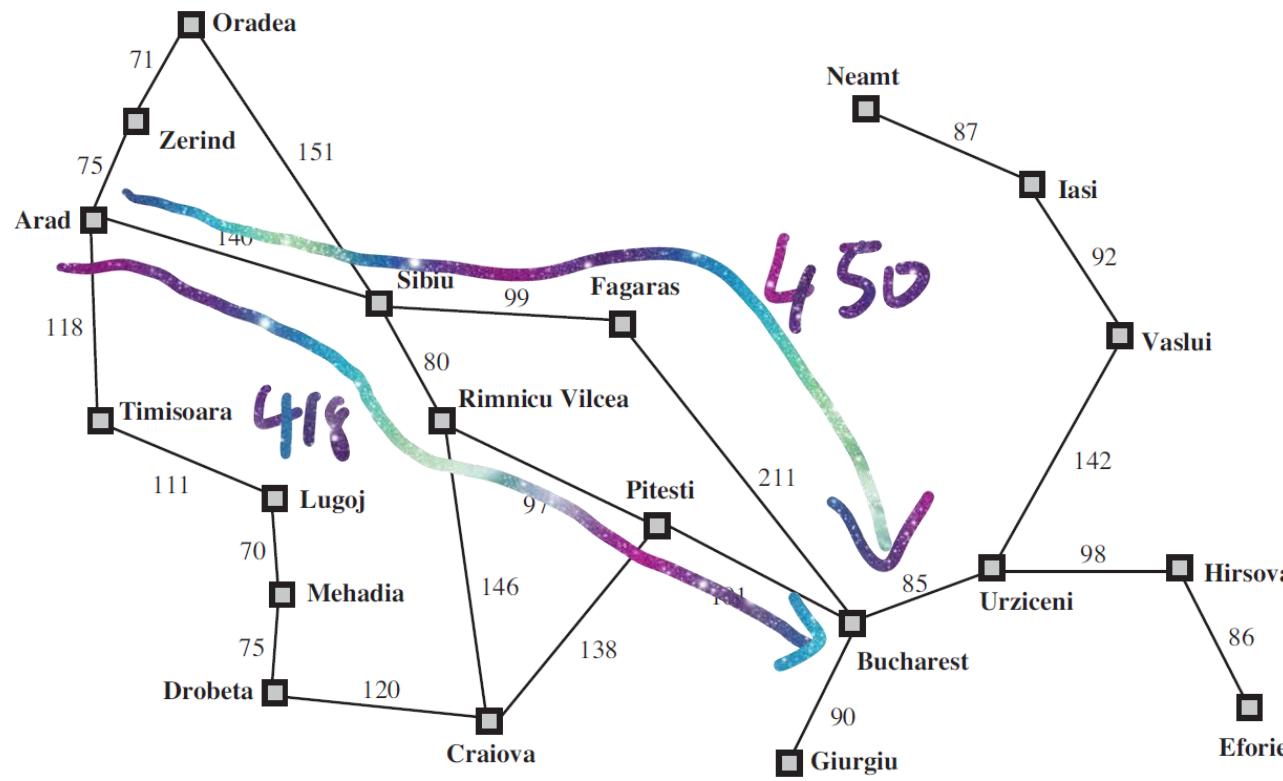


(c) After expanding Sibiu



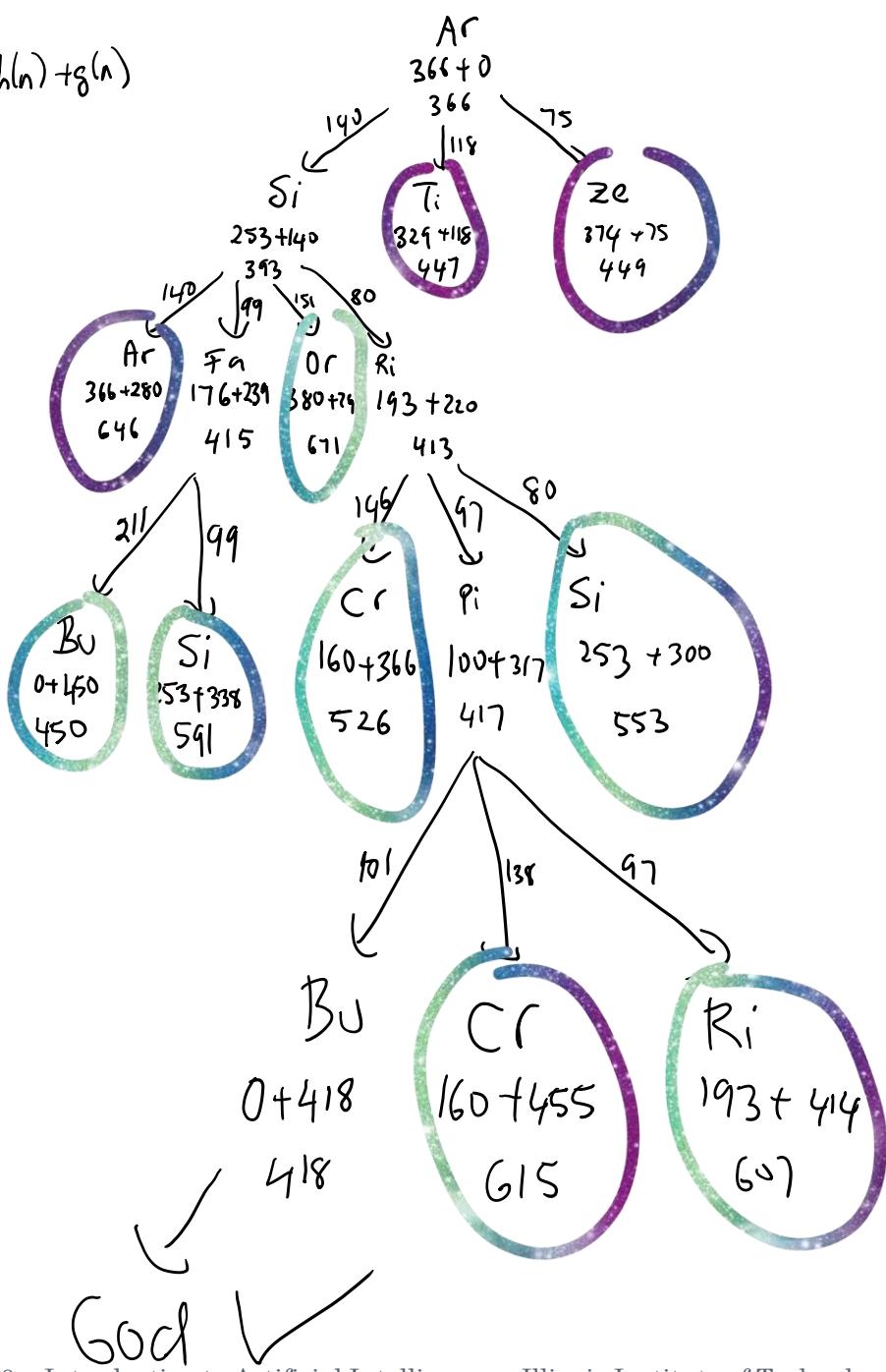
(d) After expanding Fagaras





<b>Arad</b>	<u>366</u>	<b>Mehadia</b>	241
<b>Bucharest</b>	<u>0</u>	<b>Neamt</b>	234
<b>Craiova</b>	160	<b>Oradea</b>	380
<b>Drobeta</b>	242	<b>Pitesti</b>	100
<b>Eforie</b>	161	<b>Rimnicu Vilcea</b>	193
<b>Fagaras</b>	176	<b>Sibiu</b>	<u>253</u>
<b>Giurgiu</b>	77	<b>Timisoara</b>	<u>329</u>
<b>Hirsova</b>	151	<b>Urziceni</b>	<u>80</u>
<b>Iasi</b>	226	<b>Vaslui</b>	199
<b>Lugoj</b>	244	<b>Zerind</b>	<u>374</u>

$$A^* \\ f(n) = h(n) + g(n)$$

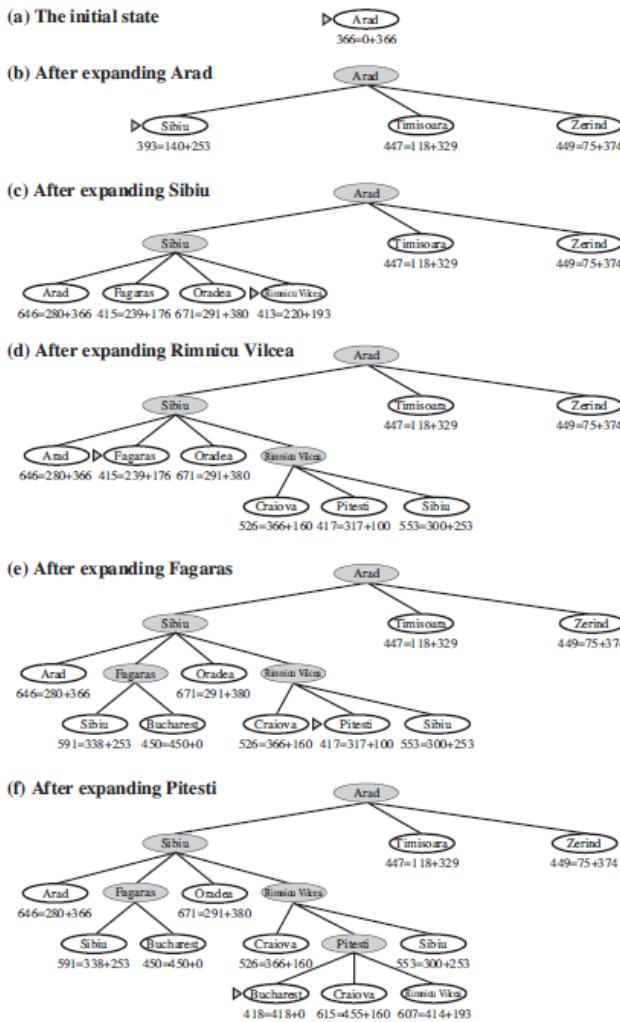


## Solution

Ar → Si → P → Pi → Bu

418

# A\* SEARCH



# REPEATED STATES

- Repeated states can be a major problem
- Repeated states are especially common for problems where actions are reversible
  - But, they also occur for problems where there are multiple distinct paths between two states
- One way to avoid this problem is to remember the set of explored and generated states

# TREE SEARCH VS. GRAPH SEARCH

**function** TREE-SEARCH(*problem*) **returns** a solution, or failure  
  initialize the frontier using the initial state of *problem*  
**loop do**  
    **if** the frontier is empty **then return** failure  
    choose a leaf node and remove it from the frontier  
    **if** the node contains a goal state **then return** the corresponding solution  
    expand the chosen node, adding the resulting nodes to the frontier

**function** GRAPH-SEARCH(*problem*) **returns** a solution, or failure  
  initialize the frontier using the initial state of *problem*  
  **initialize the explored set to be empty**  
**loop do**  
    **if** the frontier is empty **then return** failure  
    choose a leaf node and remove it from the frontier  
    **if** the node contains a goal state **then return** the corresponding solution  
    **add the node to the explored set**  
    expand the chosen node, adding the resulting nodes to the frontier  
    **only if not in the frontier or explored set**

Warning: this is Figure 3.7 in the book. The last line needs to change to guarantee optimality.

# MODIFIED GRAPH SEARCH

**function** GRAPH-SEARCH(*problem*) **returns** a solution, or failure

  initialize the frontier using the initial state of *problem*

*initialize the explored set to be empty*

**loop do**

**if** the frontier is empty **then return** failure

    choose a leaf node and remove it from the frontier

**if** the node contains a goal state **then return** the corresponding solution

*add the node to the explored set*

    expand the chosen node

      for child in children of the chosen node:

**if** child is not in explored:

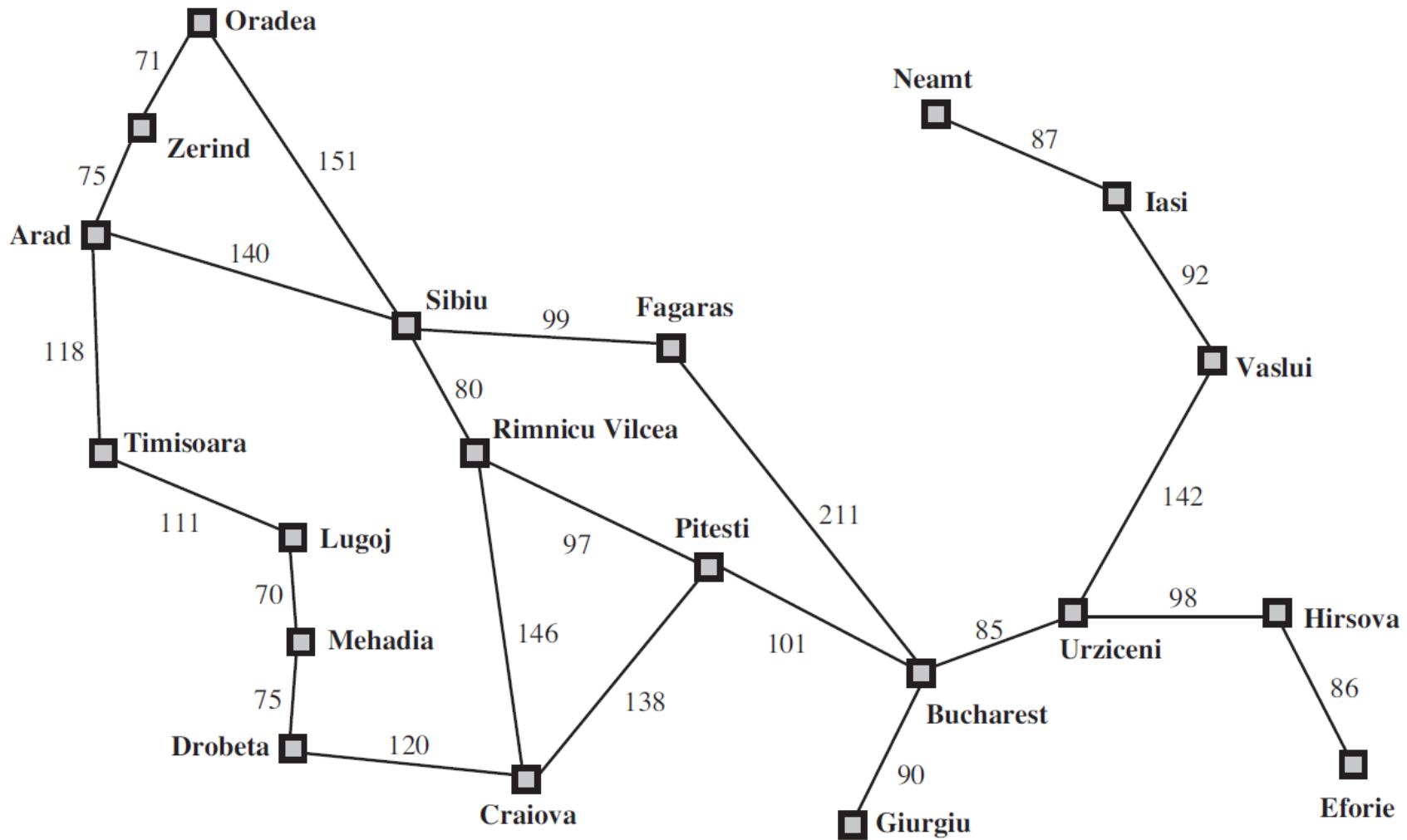
          add child to frontier only if

          child is not in frontier or

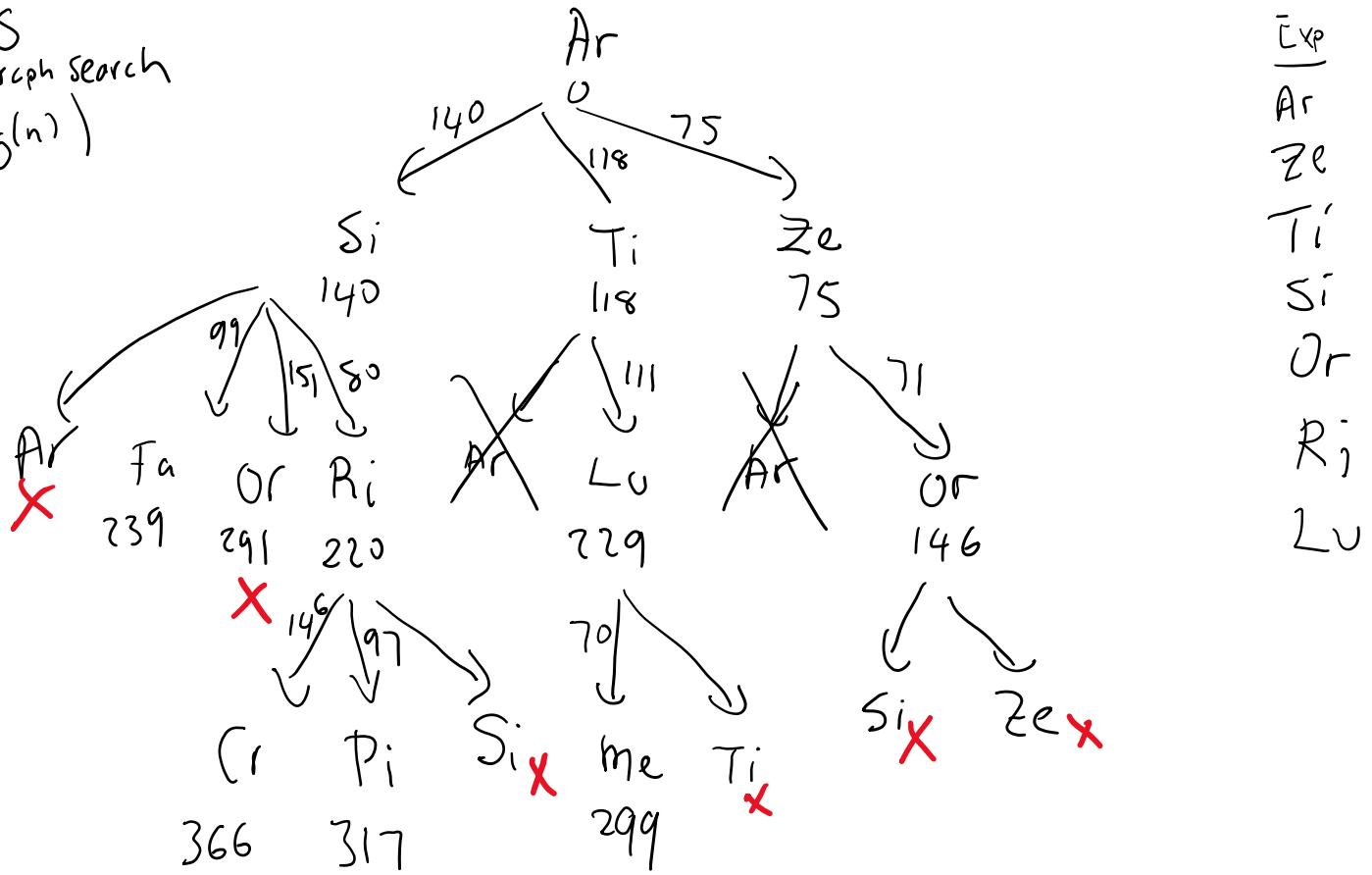
          child has ~~a lower cost~~ than the one in frontier

          is better

# GRAPH SEARCH EXAMPLES



VCS  
graph search  
 $(\mathcal{G}(n))$



Exp

Ar

Ze

Ti

Si

Or

Ri

Lu

Left as an exercise! Ü

# QUESTIONS TO THINK ABOUT

How does graph search affect

- Completeness,
- Time complexity,
- Space complexity, and
- Optimality

of various search algorithms?

# A\* OPTIMAL?

- Tree version is optimal if
  - $h(n)$  is admissible
- Graph version is optimal if
  - $h(n)$  is consistent

# ADMISSIBLE? CONSISTENT?

$$f(n) = g(n) + h(n)$$

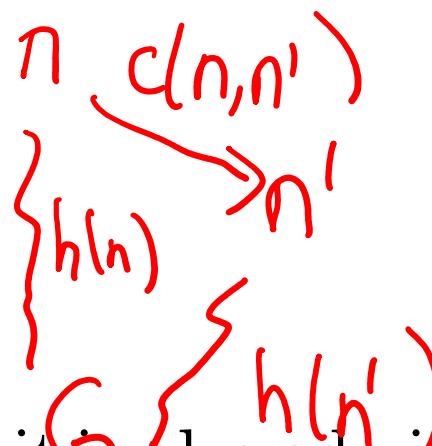
- **Admissible** if

- $h(n)$  ~~does not~~ overestimate the optimal cost
- That is  $h(n)$  is always optimistic
- E.g., straight line distance between two cities

- **Consistent** if

- If  $h(n) \leq c(n, n') + h(n')$
- $n'$  is the successor of  $n$
- Triangle inequality

- If a heuristic is consistent, it ~~is~~ also admissible  
(the other way around is not guaranteed)



# A\* OPTIMALITY – GRAPH SEARCH – PROOF

1. If  $h(n)$  is consistent, then  $f(n)$  along any path is non-decreasing
  2. When  $n$  is expanded, the optimal path to it has been found
- Proof is given on page 95

## A\* VS UNIFORM-COST SEARCH

- Compare and contrast A\* and UCS
  - Which one is better and why?
- Given the same heuristic function, can you design a complete and optimal algorithm that expands fewer nodes than A\*?

$A^* = UCS$  When?

$$h(n) = 0 \quad \forall n$$

S: start state

G: goal state

Actions:

Costs:

DFS - LIFO      — not guaranteed optm  
BFS - FIFO      — optimal steps left then

Tree  
Graph

IDS

UCS — optimal

GBFS — not opt

A\* — optimal

$A^*$  — UCS  
 $\forall n \ h(n) = 0$   
 $\forall n$

---

$h(n)$  — admissible

does not overestimate the cost

---

$$h_2 \geq h_1$$

## 8-PUZZLE HEURISTICS

- $h_1$  = The number of misplaced tiles
- $h_2$  = Manhattan distance

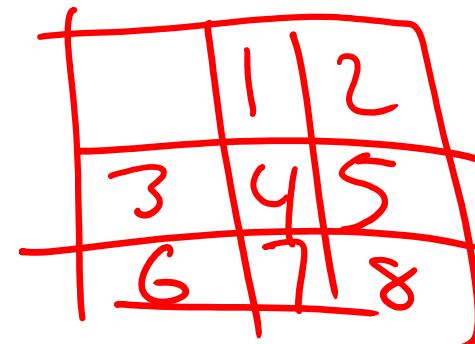
← 6

← 12

1: 1  
2 3  
3: 1  
4 0  
5: 3  
6 3  
7: 0 / 12  
8: 1

Let's see an example

5	6	1
2	4	8
3	7	



# EFFECTIVE BRANCHING FACTOR

- If the total number of nodes generated is  $N$  and the solution depth is  $d$ , then
  - $b^*$  is the branching factor that a uniform tree of depth  $d$  would need to have in order to contain  $N+1$  nodes
- $N+1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$
- If A\* finds a solution at depth 4 using 40 nodes, what is  $b^*$ ?
  - $\approx 2.182$
- A good heuristic function achieves  $b^* \approx 1$

$$1 + b + b^2 + b^3 + \dots + b^d = N + 1$$

$$b = ?$$

# IDS vs A\*

f-puzzle  
 $h_1$ : misplace  
 $h_2$ : MD

d	Search Cost (nodes generated)			Effective Branching Factor		
	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	-	539	113	-	1.44	1.23
16	-	1301	211	-	1.45	1.25
18	-	3056	363	-	1.46	1.26
20	-	7276	676	-	1.47	1.27
22	-	18094	1219	-	1.48	1.28
24	-	39135	1641	-	1.48	1.26

**Figure 3.29** Comparison of the search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and A\* algorithms with  $h_1$ ,  $h_2$ . Data are averaged over 100 instances of the 8-puzzle for each of various solution lengths d.

# HEURISTIC FUNCTIONS

- Let  $h(n)=0$  for all  $n$ 
  - Is it admissible? Yes
  - Is it consistent? Yes
  - Is it any good? No
- Can we say an admissible heuristic function  $h_i$  is always better than another admissible heuristic function  $h_j$ ?
- How can we find good heuristics?
- A heuristic: run uniform-cost search, find the solution, return the cost of the solution as the heuristic value. Can you beat it?
- Let  $h(n) = \max(h_1(n), h_2(n), \dots, h_m(n))$ , where  $h_i(n)$ , are admissible
  - Is  $h(n)$  admissible? Yes
  - Can we say  $h(n)$  is better than any of  $h_i(n)$ ? Yes
  - What's the catch? Time

# SUMMARY

- Depth-first search
- Breadth-first search
- Depth-limited depth-first search
- Iterative deepening
- Uniform cost search:  $g(n)$
- Greedy best-first search:  $h(n)$
- A\* search:  $g(n) + h(n)$

Which one is better under what conditions?

# A BIT HISTORY

- [https://en.wikipedia.org/wiki/A\\* search algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)
  - “A\* was created as part of the Shakey project, which had the aim of building a mobile robot that could plan its own actions.”
  - Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics SSC4*. 4 (2): 100–107
    - <https://doi.org/10.1109%2FTSSC.1968.300136>
- Shakey
  - <https://www.youtube.com/watch?v=7bsEN8mwUB8>
  - Minute: 3:50

# NEXT

- Skip Chapter 4
- Chapter 5: Adversarial search – game playing