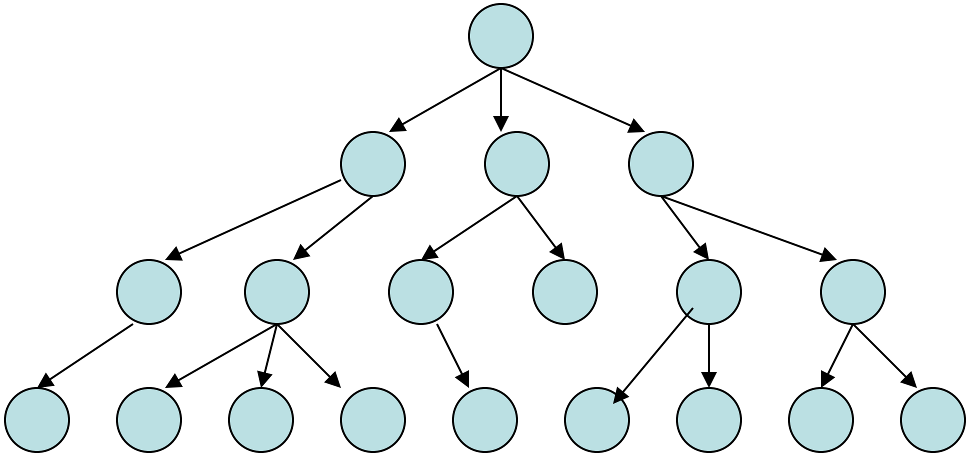
Solving Problems by Searching



Terminology

- State
- State Space
- Initial State
- Goal Test
- Action
- Step Cost
- Path Cost
- State Change Function
- State-Space Search

Formal State-Space Model

Problem = (S, s, A, f, g, c)

S = state space
s = initial state
A = set of actions
f = state change function
g = goal test function
c = cost function

$$x \xrightarrow[c(a)]{a} y$$

State-Space Model (cont)

Problem = (S, s, A, f, g, c)

- How do we define a solution?

- How about an optimal solution?

3 Coins Problem
A Very Small State Space Problem

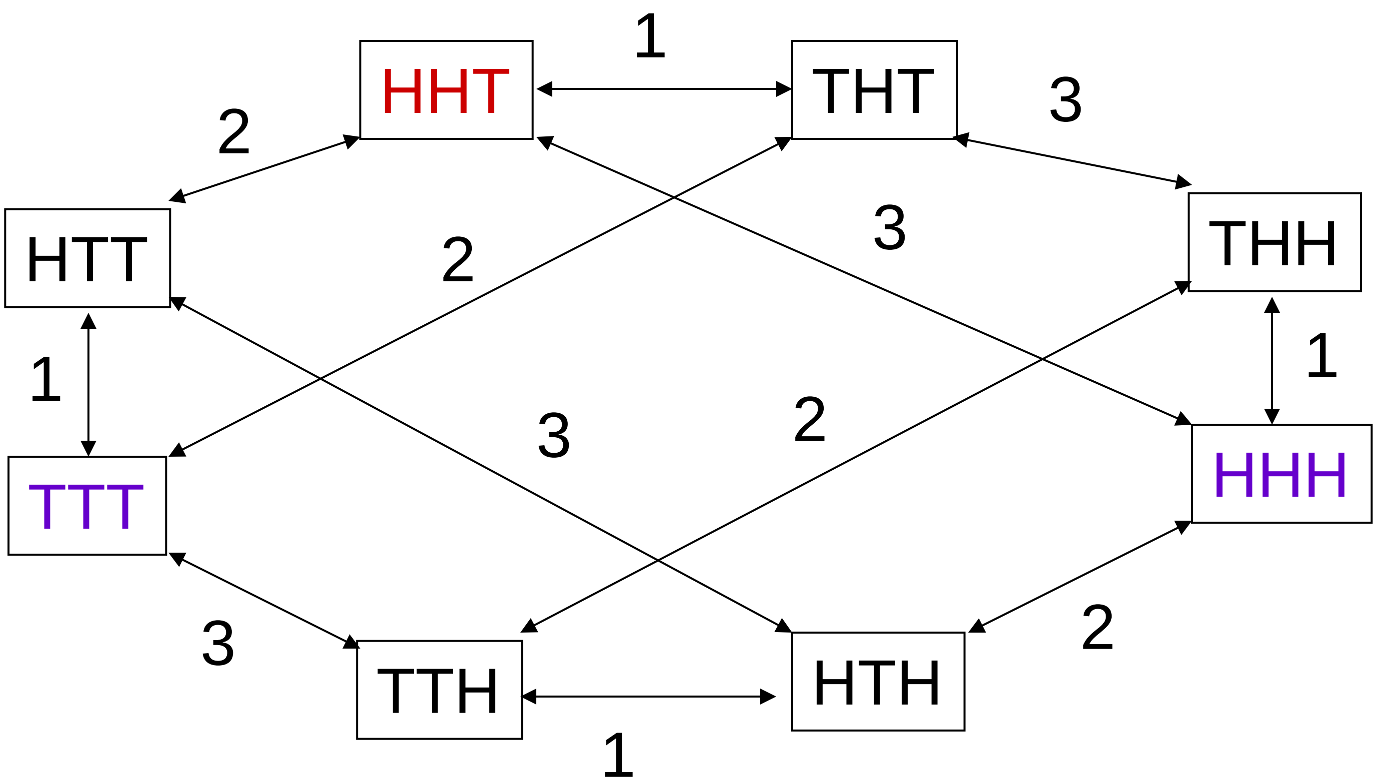
- There are 3 (distinct) coins: coin1, coin2, coin3.
- The initial state is H H T
- The legal operations are to turn over exactly one coin.
  - 1 (flip coin1), 2 (flip coin2), 3 (flip coin3)
- There are two goal states: H H H
  T T T

What are S, s, A, f, g, c ?

3 Coins Problem: Get from HHT to either HHH or TTT via operators: flip coin 1, flip coin 2, and flip coin 3 (flip = turn over)

- S
- s
- A
- f
- g
- c

State-Space Graph



- What are some solutions?
- What if the problem is changed to allow only 3 actions?

Modified State-Space Problem

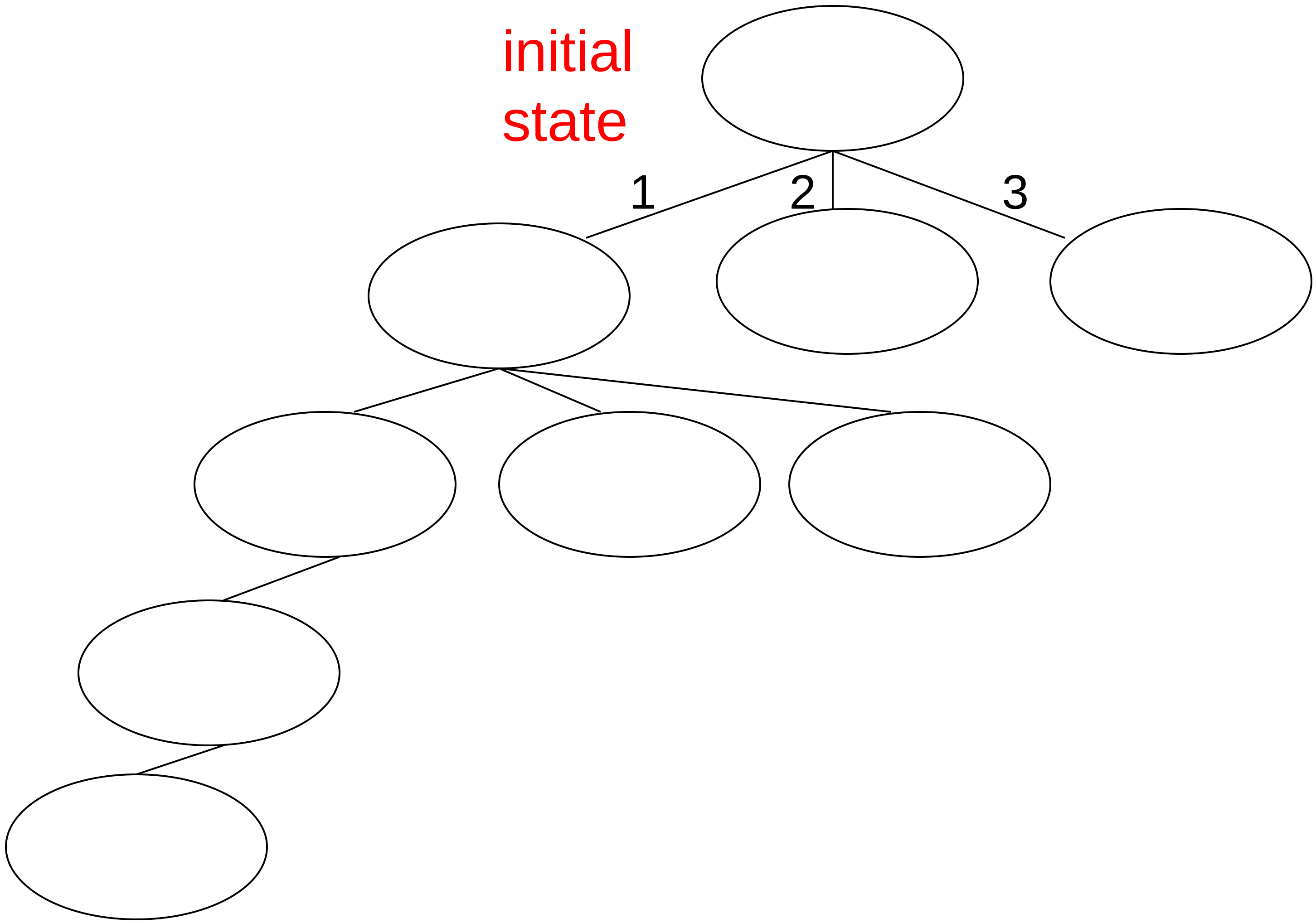
- How would you define a state for the new problem requiring exactly 3 actions?

- How do you define the operations (1, 2, 3) with this new state definition?

Modified State-Space Problem

- What do the paths to the goal states look like now?
- (H,H,T,0) ->

How do we build a search tree for the modified 3 coins problem?



The 8-Puzzle Problem

one initial state

1	2	3
8	B	4
7	6	5

goal state

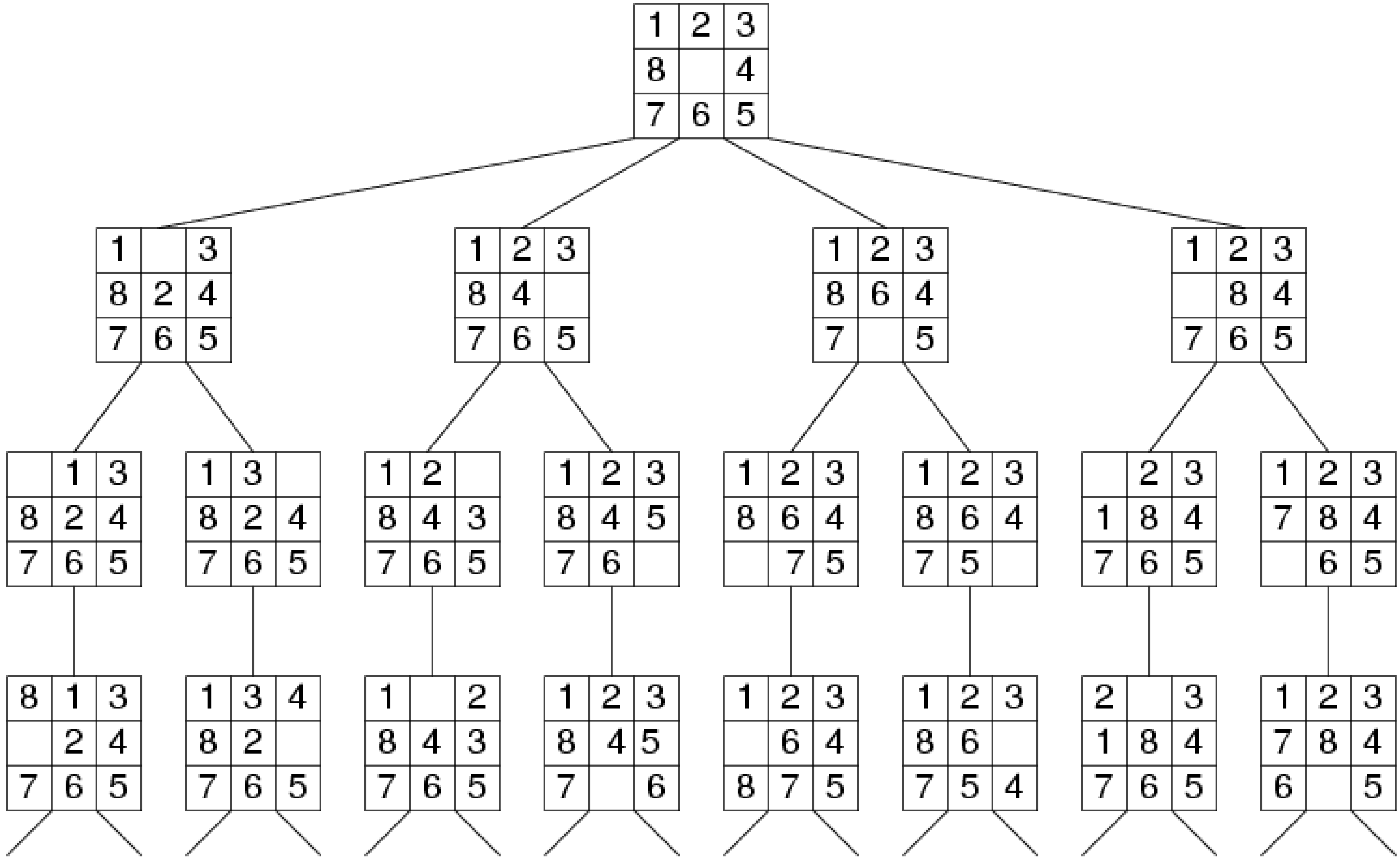
B	1	2
3	4	5
6	7	8

B=blank

1. What data structure easily represents a state?
2. How many possible states are there?
3. How would you specify the state-change function?

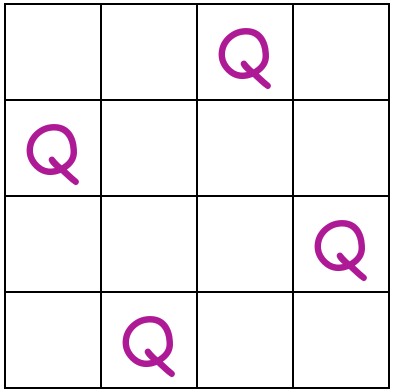
4. What is the path cost function?
   uniform cost (=1)
5. What is the complexity of the search?

Search Tree Example: Fragment of 8-Puzzle Problem Space



Another Example: N Queens

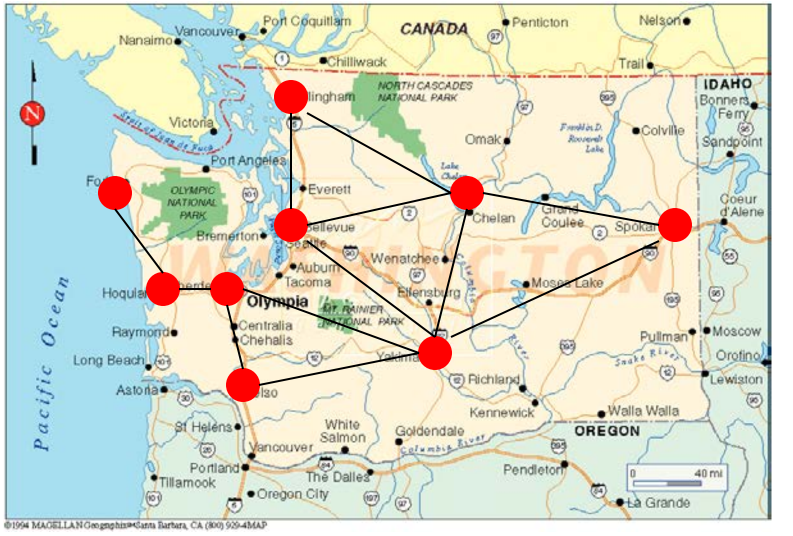
Place exactly one Q in each column so that no two Q's are in the same row or diagonal



- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)
- Output

Example: Route Planning

Find the shortest route from the starting city to the goal city given roads and distances.



- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)
- Output:

Search in AI

- Search in Data Structures
  - You're given an existent tree.
  - You search it in different orders.
  - It resides in memory.
- Search in Artificial Intelligence
  - The tree does not exist.
  - You have to generate it as you go.
  - For realistic problems, it does not fit in memory.

Search Strategies (Ch 3)

- Uninformed Search

  The search is blind, only the order of search is important.

- Informed Search

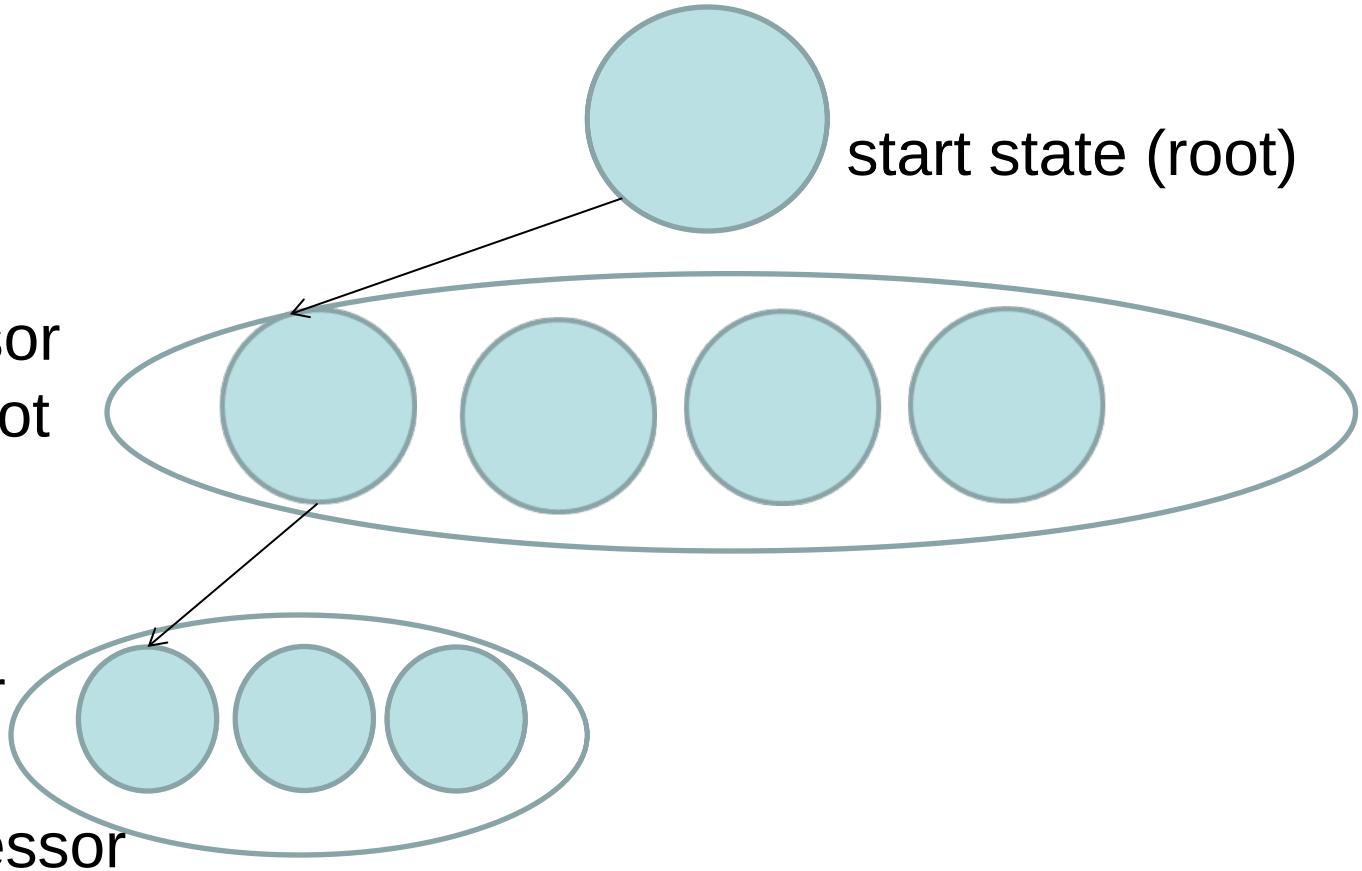
  The search uses a heuristic function to estimate the goodness of each state.

Depth-First Search by Recursion*

You will use this for Missionary-Cannibal Problem.

- Search is a recursive procedure that is called with the start node and has arg s.
- It checks first if s is the goal.
- It also checks if s is illegal or too deep.
- If neither, it generates the list L of successors of its argument s.
- It iterates through list L, calling itself recursively for each state in L.

Depth-First Search by Recursion



start state (root)

successor
list of root

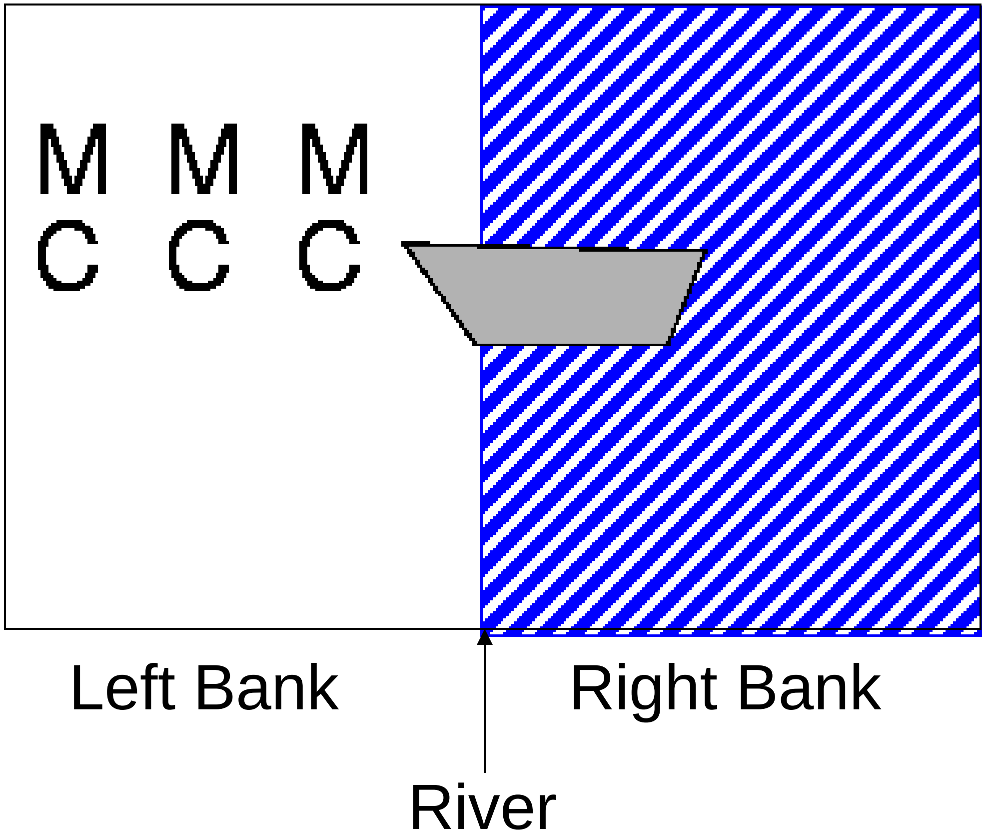
successor
list of
first successor
of root

The Missionaries and Cannibals Problem (from text problem 3.9)

- Three missionaries and three cannibals are on one side (left) of a river, along with a boat that can hold one or two people.
- If there are ever more cannibals than missionaries on one side of the river, the cannibals will eat the missionaries. (We call this a “dead” state.)
- Find a way to get everyone to the other side (right), without ever leaving a group of missionaries in one place (left or right) outnumbered by the cannibals in that place, ie. without anyone getting eaten.

Missionaries and Cannibals Problem

Missionaries and Cannibals Problem



Missionary and Cannibals Notes

- Define your state as (M,C,S)
  - M: number of missionaries on left bank
  - C: number of cannibals on left bank
  - S: side of the river that the boat is on

- When the boat is moving, we are in between states. When it arrives, everyone gets out.

(3,3,L) → (3,1,R)   What action did I apply?

What are all the actions?

- Left to right

1. MCR
2. MMR
3. ?
4. ?
5. ?

- Right to left

1. MCL
2. MML
3. ?
4. ?
5. ?

When is a state considered “DEAD”?

1. There are more cannibals than missionaries on the left bank. (Bunga-Bunga)

2. There are more cannibals than missionaries on the right bank. (Bunga-Bunga)

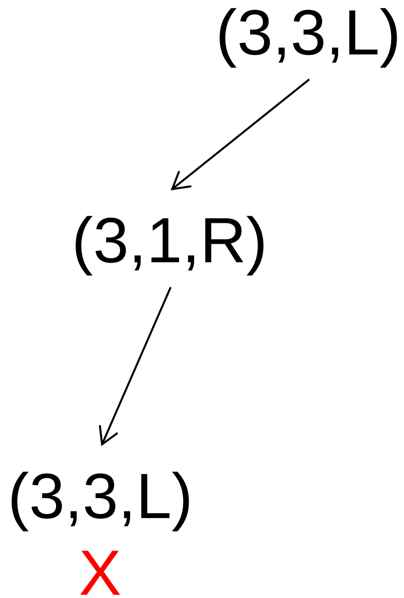
3. There is an ancestor state of this state that is exactly the same as this state. (Why?)

Same Ancestor State

Stack

(3,3,L)

(3,1,R)



Assignment

- Implement and solve the problem
  - You MUST use recursive depth-first blind search.
  - You must detect illegal states (cannibals can eat missionaries) and repeated states along a path.
  - You must keep going and print out all four solutions.
- You must use Python
- Full instructions will be on the assignment page.

General Search Paradigm
(Figure 3.7 in text)

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

1. What is the *frontier*?
2. How do we choose?
3. What does expand mean?

General Search Paradigm
(Figure 3.7 in text)

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

OPEN CLOSED

Basic Idea

- Start with the initial state
- Maintain a (general) queue of states to visit
  - Depth-First search: the queue is LIFO (stack)
  - Breadth-First search: the queue is FIFO (queue)
  - Uniform-Cost search: the queue is ordered by lowest path cost g (path from start to node)
  - Depth-Limited search: DFS with a depth limit
  - Iterative-Deepening search: DFS with depth limit sequence 1, 2, 3, …. till memory runs out
  - Bidirectional Search

Performance Criteria

- Completeness: Does it find a solution when there is one?
- Optimality: Does it find the optimal solution in terms of cost?
- Time complexity: How long does it take to find a solution
- Space Complexity: How much memory is needed?

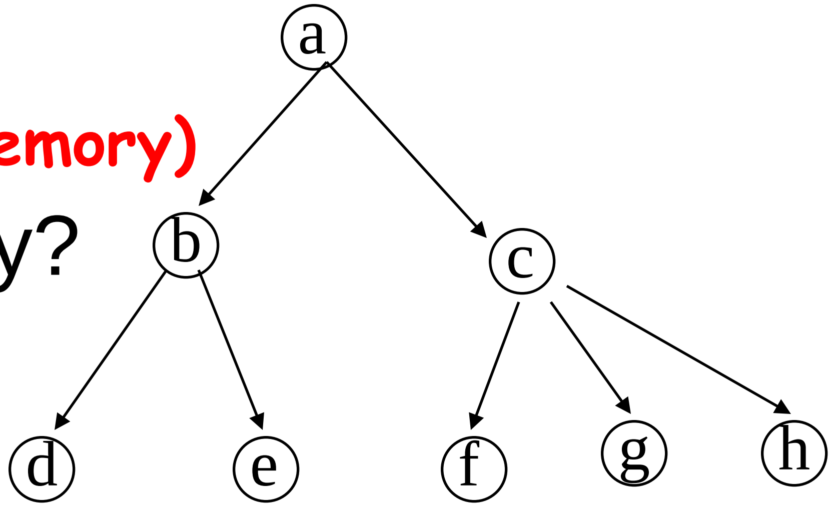
Breadth-First Search

- Maintain FIFO queue of nodes to visit
- Evaluation (branching factor b; solution at depth d)
  - Complete?

    Yes (if enough memory)
  - Time Complexity?

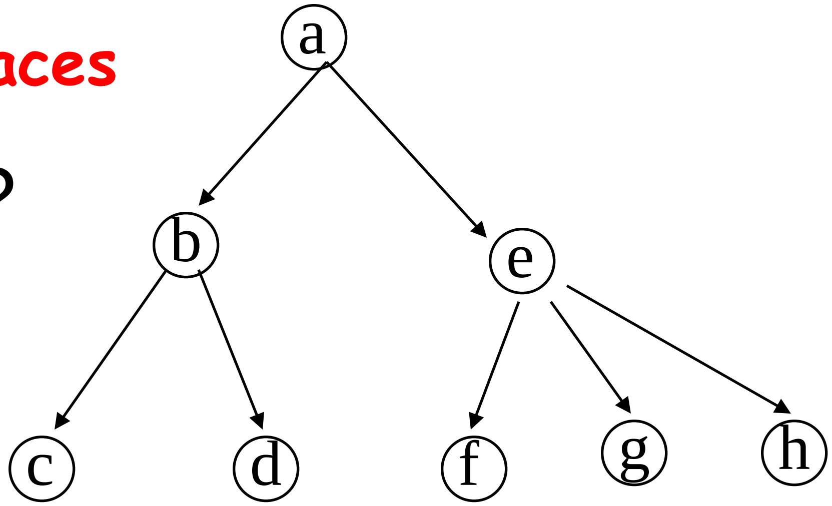
    O(b^d)
  - Space?

    O(b^d)



Depth-First Search

- Maintain stack of nodes to visit
- Evaluation (branching factor b; solution at depth d)
  - Complete?
    Not for infinite spaces
  - Time Complexity?
    O(b^d)
  - Space ?
    O(d)



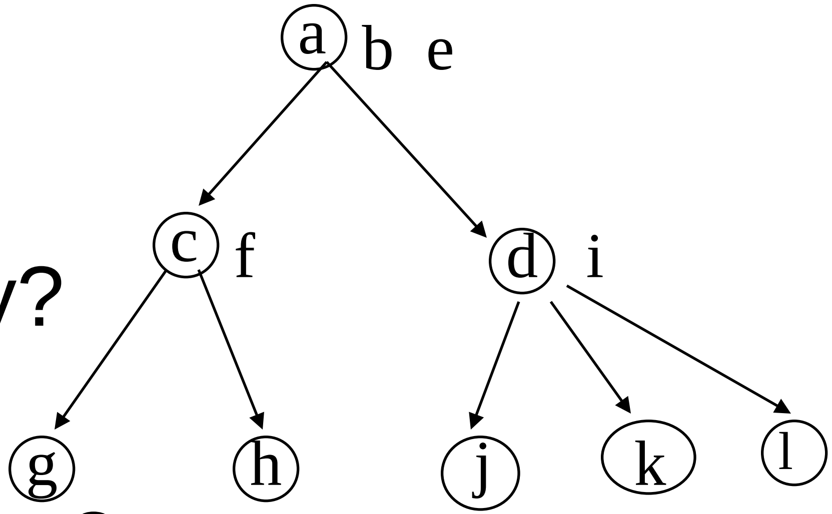
Iterative Deepening Search

- DFS with depth limit; incrementally grow limit $\ell = 0, 1, 2, \ldots$
- Evaluation (for solution at depth d)
  - Complete?

    Yes, if ℓ >= d
  - Time Complexity?

    O(b^d)
  - Space Complexity?

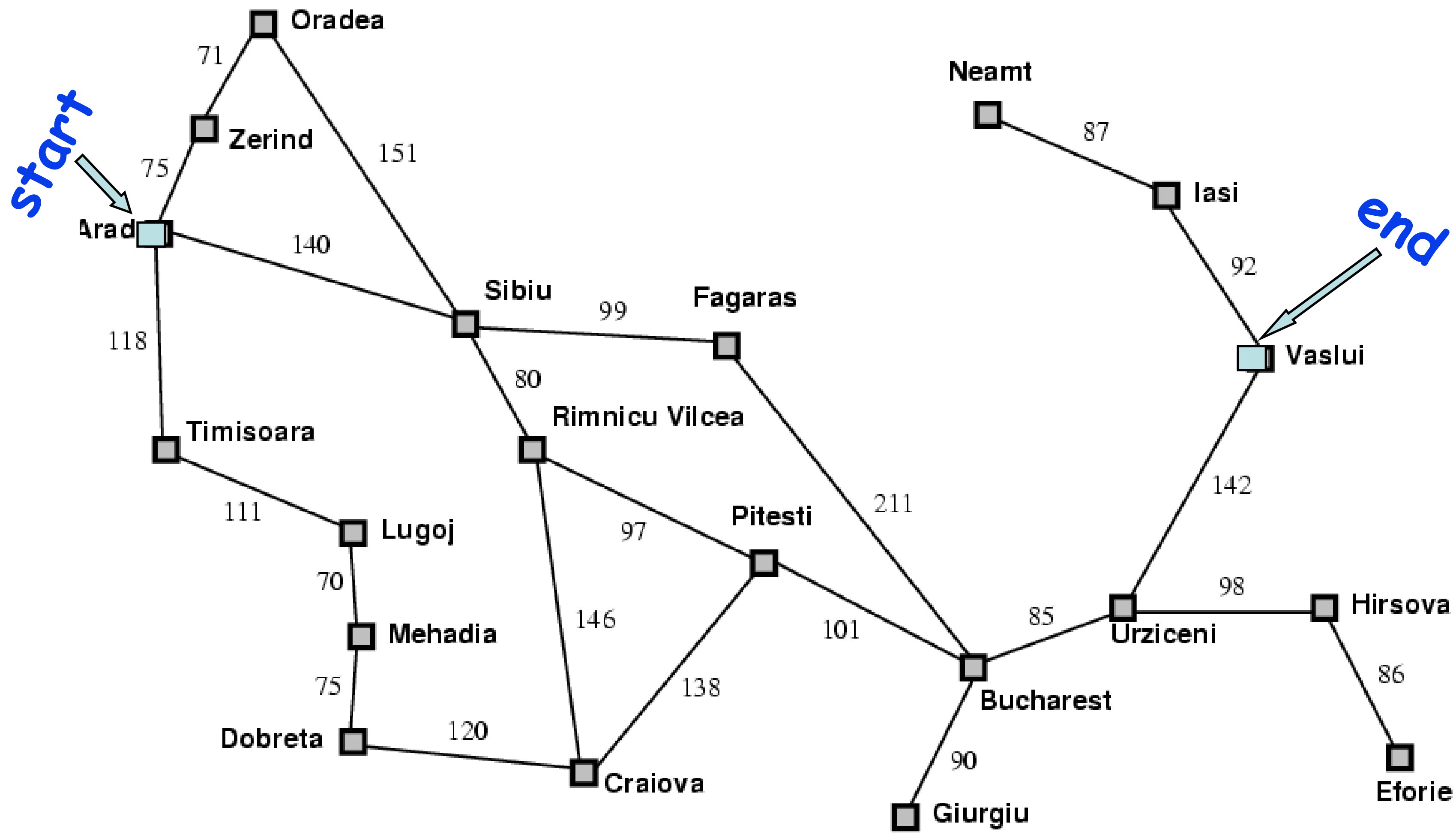
    O(d)



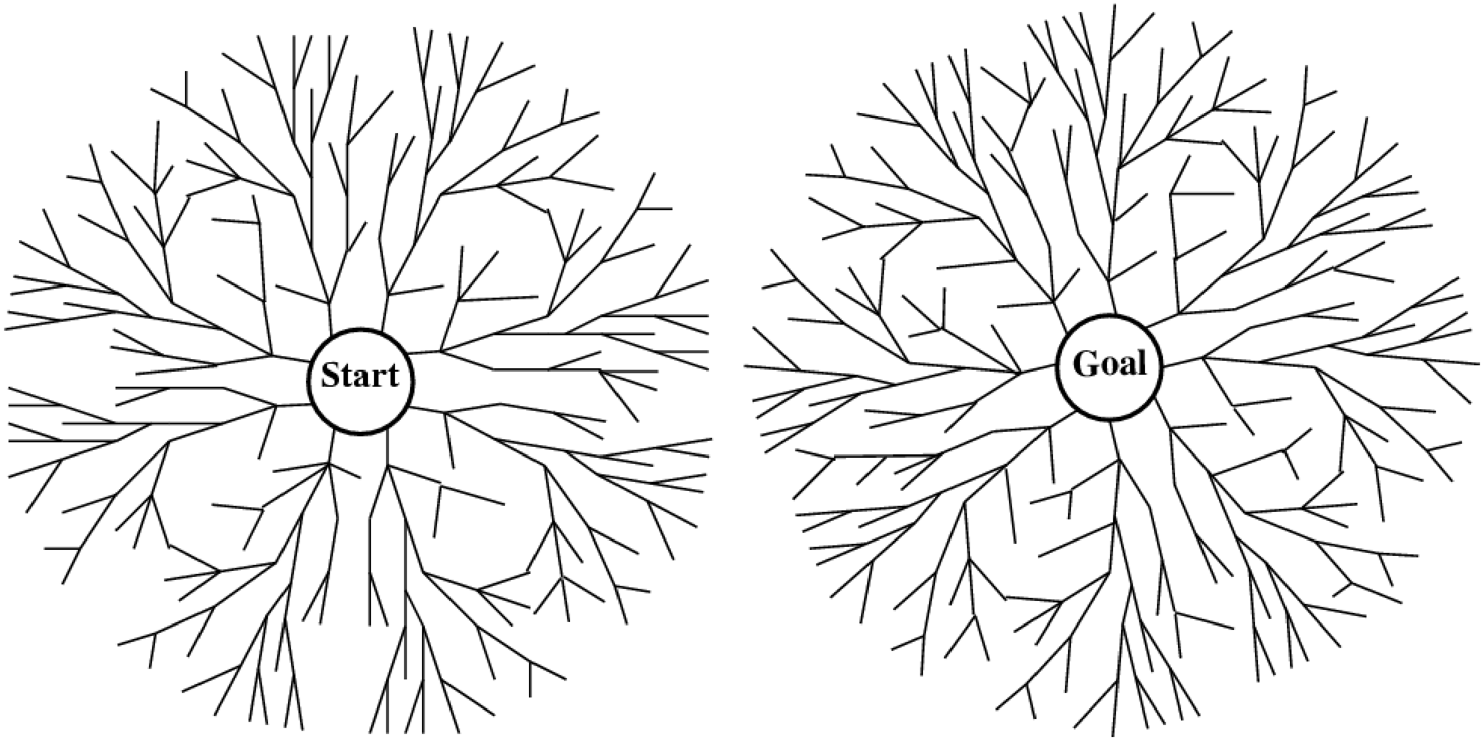
Cost of Iterative Deepening

b	ratio IDS to DFS
2	3:1
3	2:1
5	1.5:1
10	1.2:1
25	1.08:1
100	1.02:1

Forwards *vs.* Backwards



vs. Bidirectional



- Replace the goal test with a check to see if the frontiers of the two searches intersect.
- How might this be done efficiently?

Uniform-Cost Search

- Expand the node n with the lowest path cost g(n)

- Implement by storing the frontier as a priority queue ordered by g(n).

- Apply the goal test when the node is selected for expansion

- If a newly generated node n is already on the frontier as node n´ and if pathcost(n) < pathcost(n´), then replace n’ with n.

Comparison of Blind Methods

3.4.7 Comparing uninformed search strategies

Figure 3.21 compares search strategies in terms of the four evaluation criteria set forth in Section 3.4. This comparison is for tree-search versions. For graph searches, the main differences are that depth-first search is complete for finite state spaces, and that the space and time complexities are bounded by the size of the state space.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^a	Yes^{a,b}	No	No	Yes^a	Yes^{a,d}
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^c	Yes	No	No	Yes^c	Yes^{c,d}

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; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

Problem

- All these blind methods are too slow for real applications
- Solution → add guidance
- → "informed search"