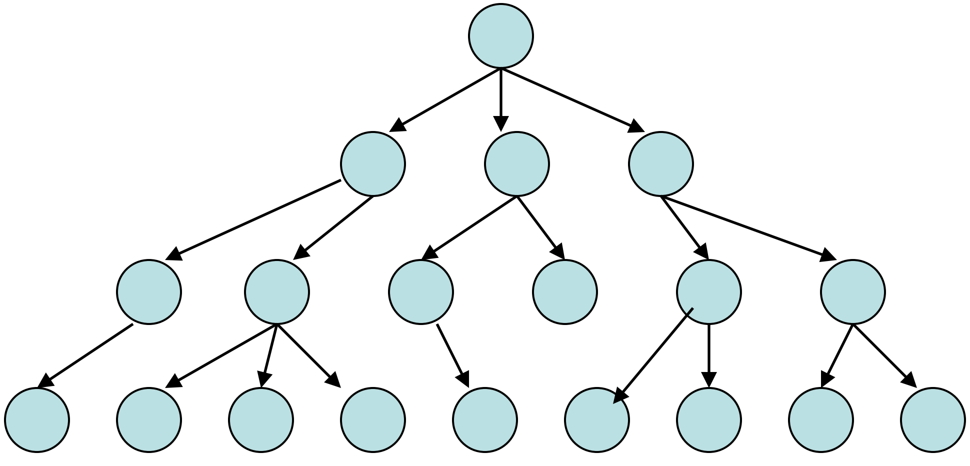
Solving Problems by Searching



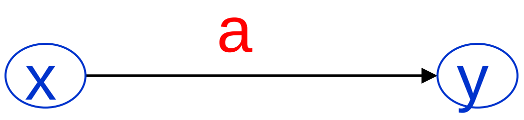
Terminology

- 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	f: S x A -> S
g = goal test function	g: S -> {true,false}
c = cost function	c: S x A x S -> R



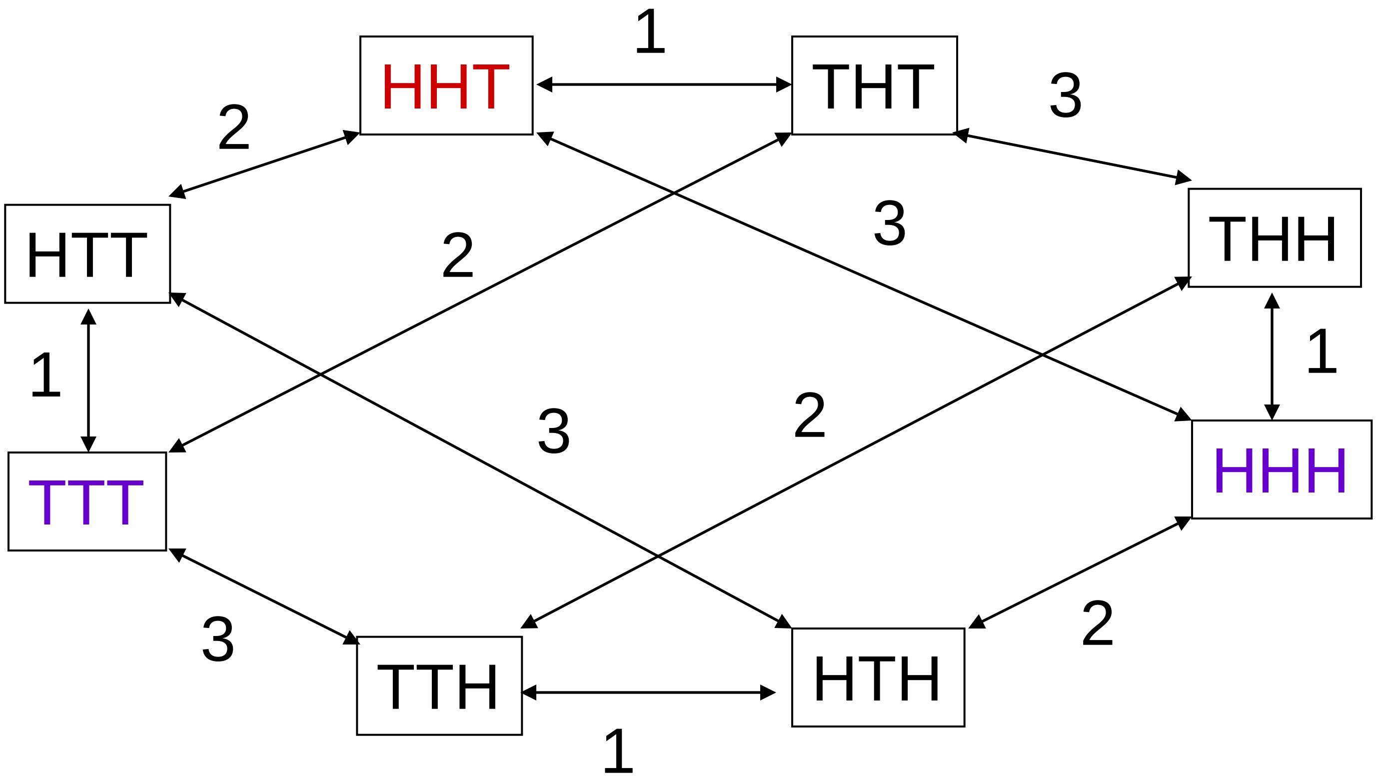
- How do we define a solution?
- How about an optimal solution?

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 ?

State-Space Graph



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

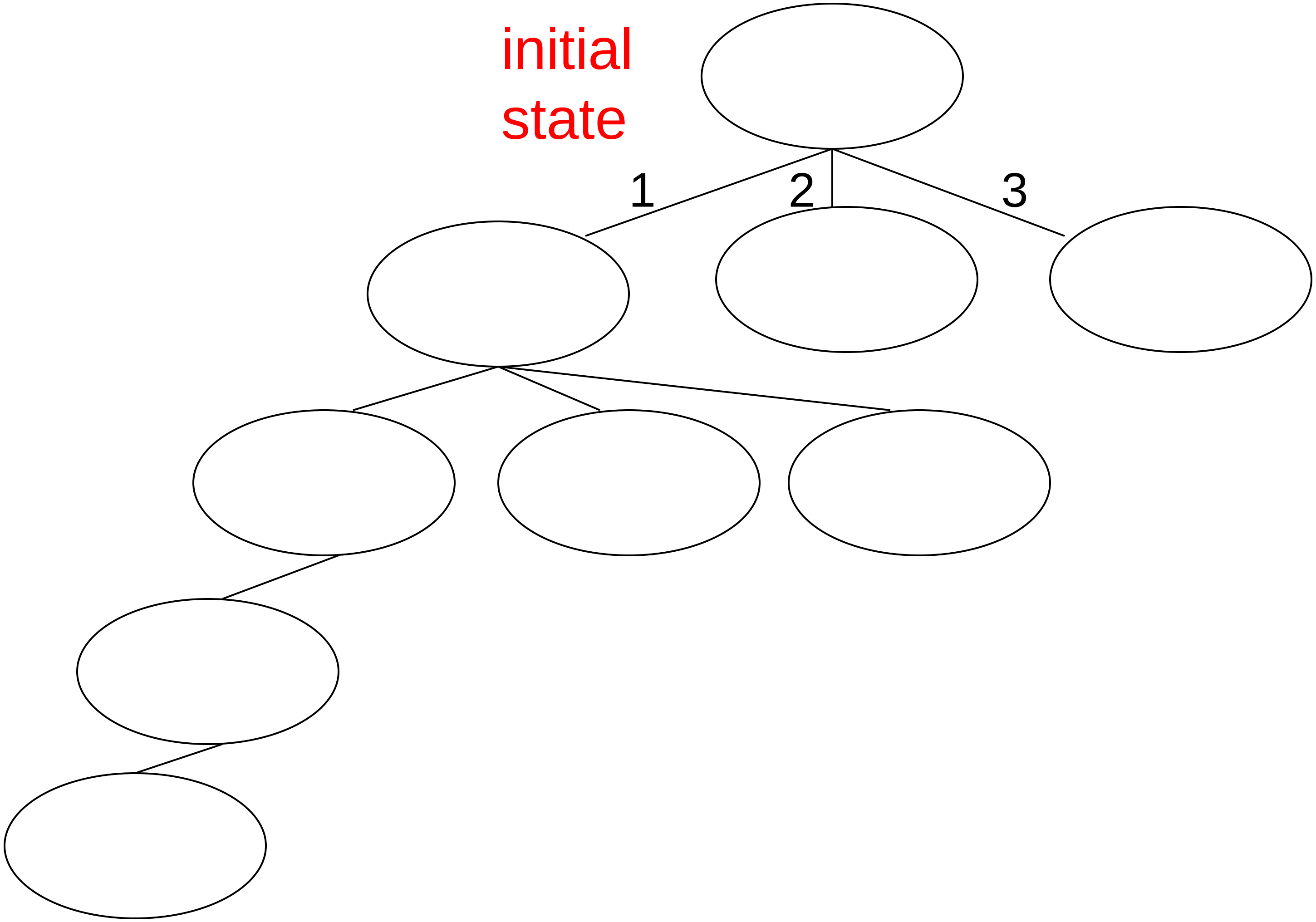
Modified State-Space Problem

- 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?

- What do the paths to the goal states look like now?

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

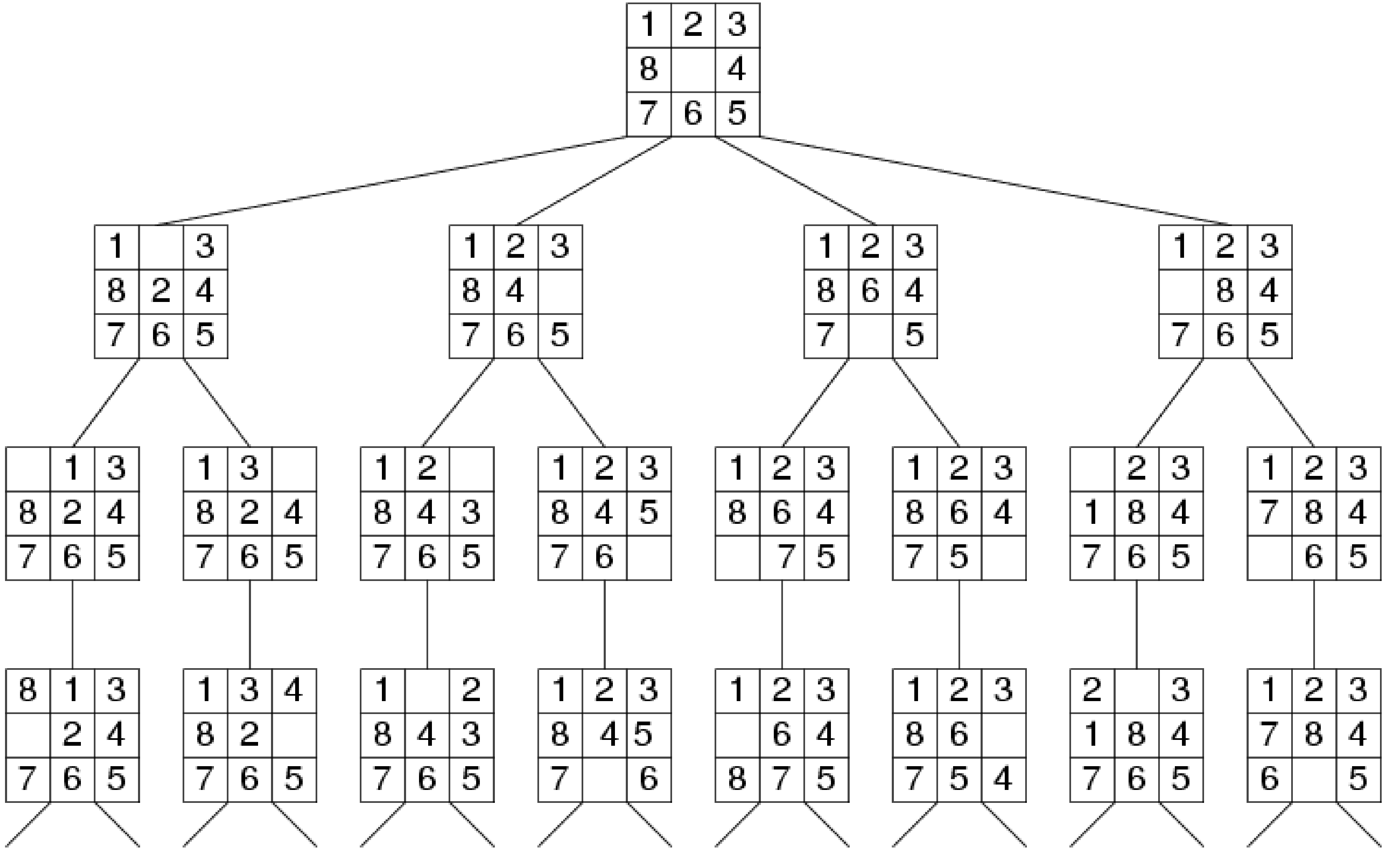
goal state

B	1	2
3	4	5
6	7	8

B=blank

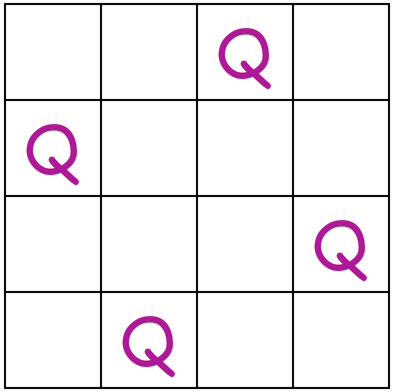
1. Formalize a state as a data structure
2. Show how start and goal states are represented.
3. How many possible states are there?
4. How would you specify the state-change function?
5. What is the goal test?
6. What is the path cost function?
7. What is the complexity of the search?

Search Tree Example: Fragment of 8-Puzzle Problem Space

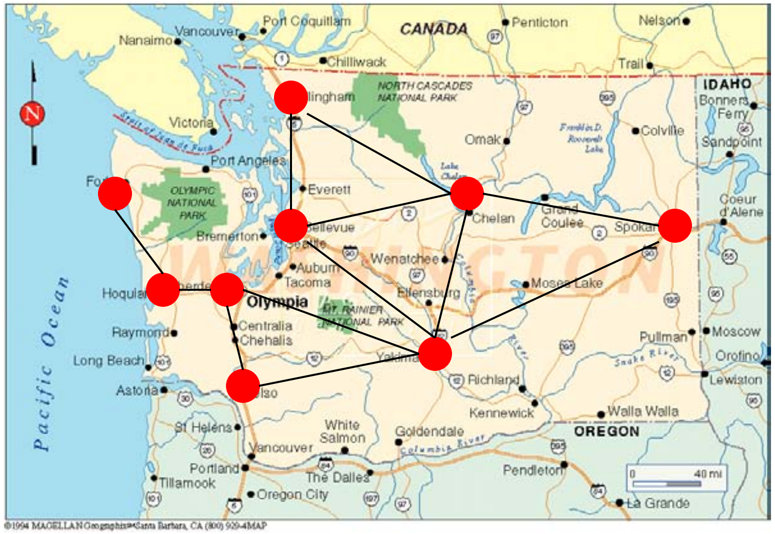


Another Example: N Queens

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



Example: Route Planning



- 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.

General Search Paradigm

Read this in the text addendum.

78 Chapter 3. Solving Problems by Searching

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

Figure 3.7 An informal description of the general tree-search and graph-search algorithms. The parts of GRAPH-SEARCH marked in bold italic are the additions needed to handle repeated states.

Basic Idea

- Start with the initial state
- Maintain a queue of states to visit
  - Breadth-First search: the queue is FIFO
  - Uniform-Cost search: the queue is ordered by lowest path cost
  - Depth-First search: the queue is LIFO (a stack)
  - 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?

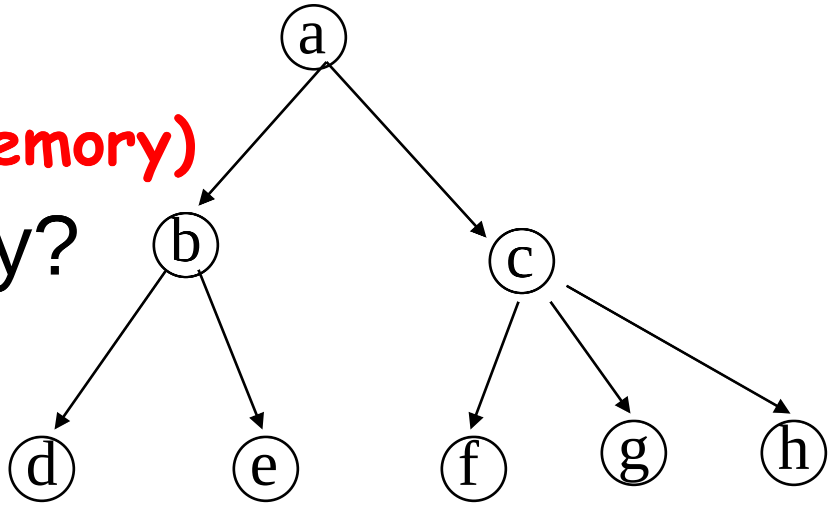
Breadth-First Search

- Maintain LIFO 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)



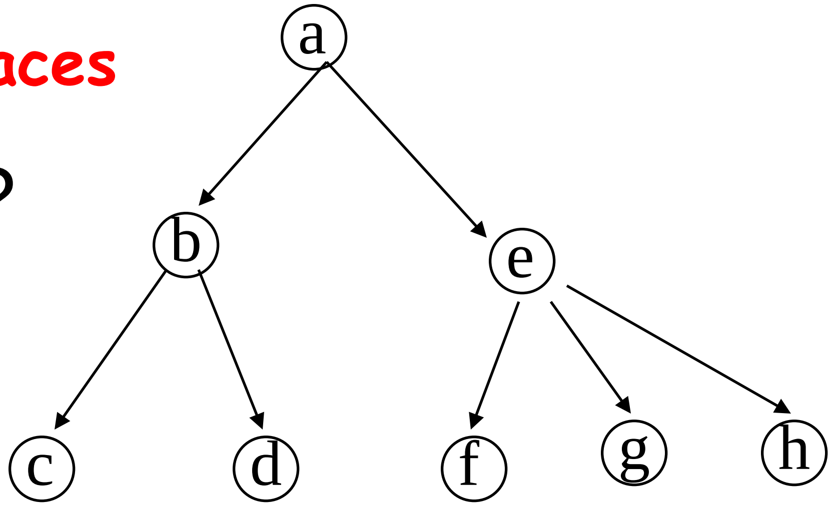
Depth-First Search

- 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)

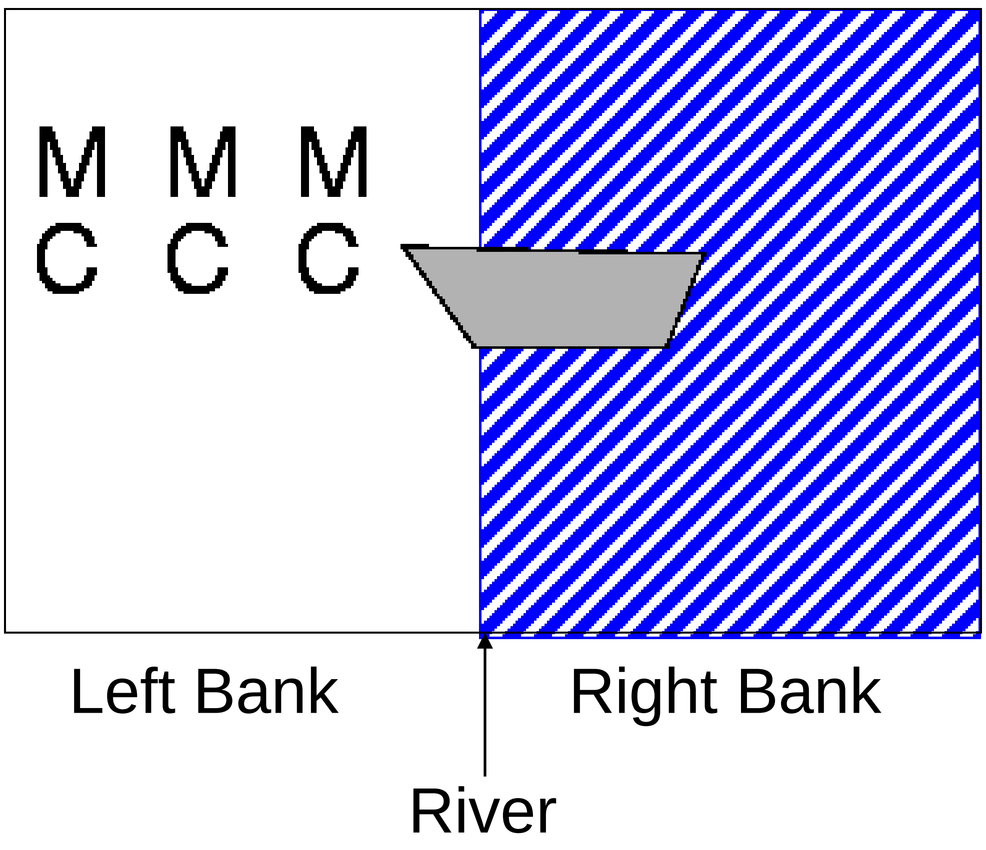


The Missionaries and Cannibals Problem (from text problem 3.22)

- Three missionaries and three cannibals are on one side 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, 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.

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?)

Assignment (problem 3.9b, which is part of the first homework set)

- Implement and solve the problem
  - You may use breadth-first or depth-first blind search.
  - Definitely avoid repeated states along a path.
  - Keep track of how many states are searched.
- Use the computer language of your choice for this assignment.
  - Java (default for solutions)
  - C++
  - Lisp or Lisp variant

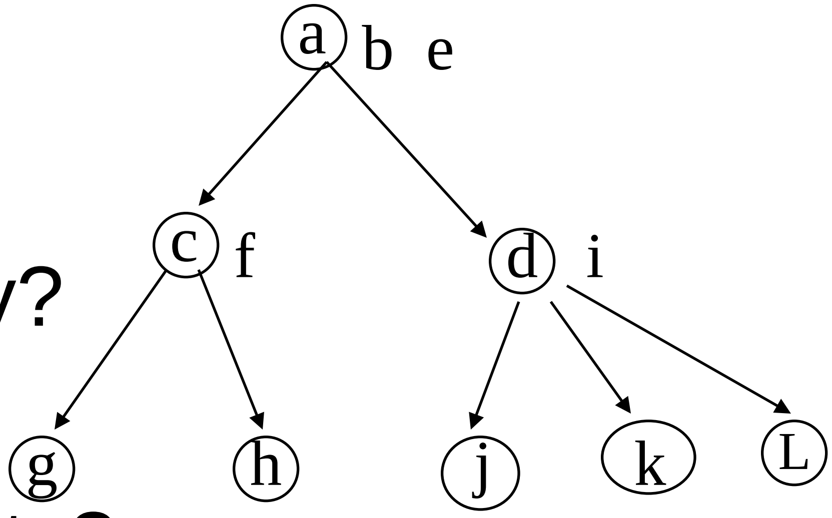
Iterative Deepening Search

- DFS with limit; incrementally grow limit ℓ
- Evaluation (for solution at depth d)
  - Complete?

    Yes, if $\ell >= d$
  - Time Complexity?

    $O(b^d)$
  - Space Complexity?

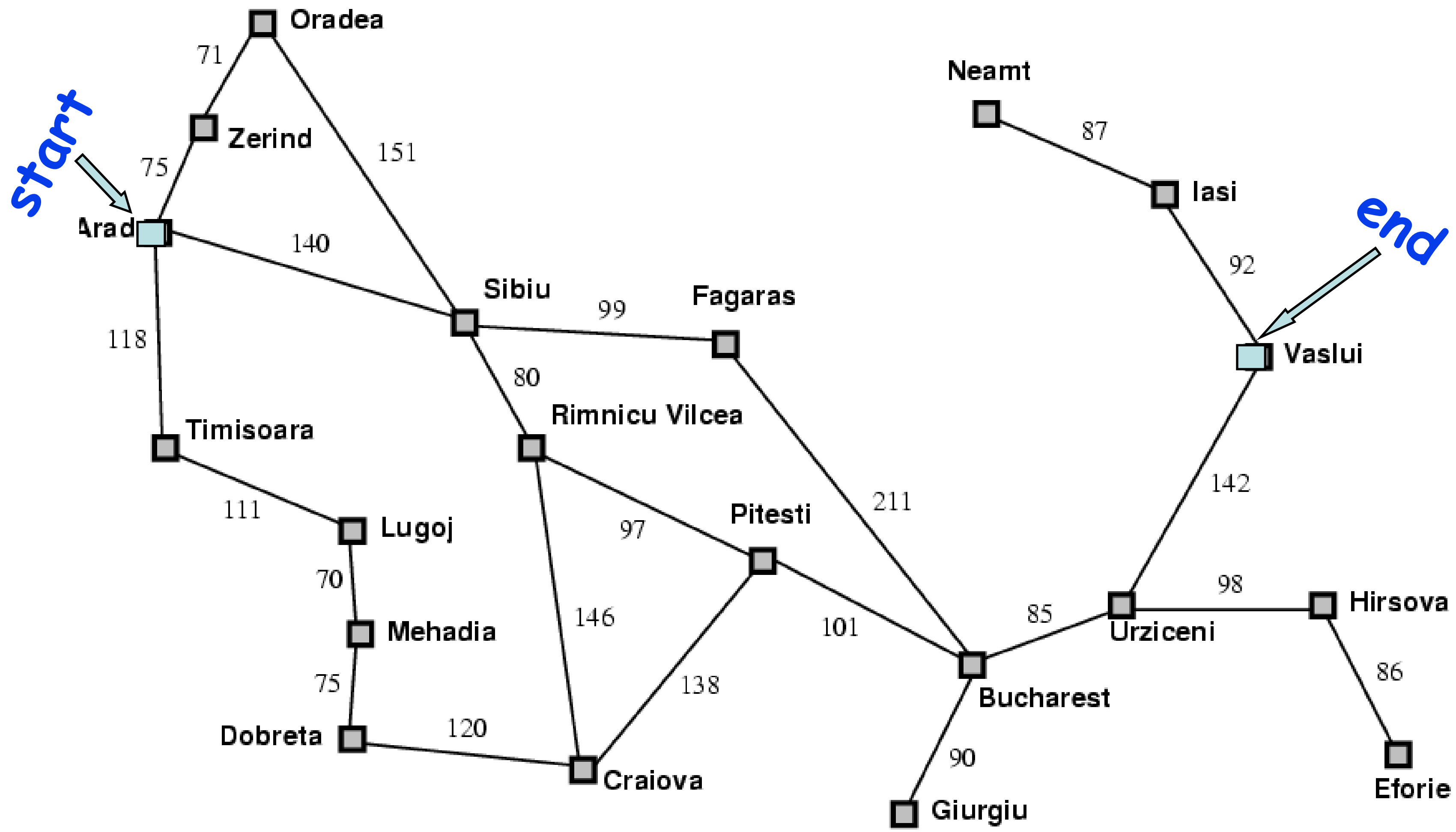
    $O(d)$



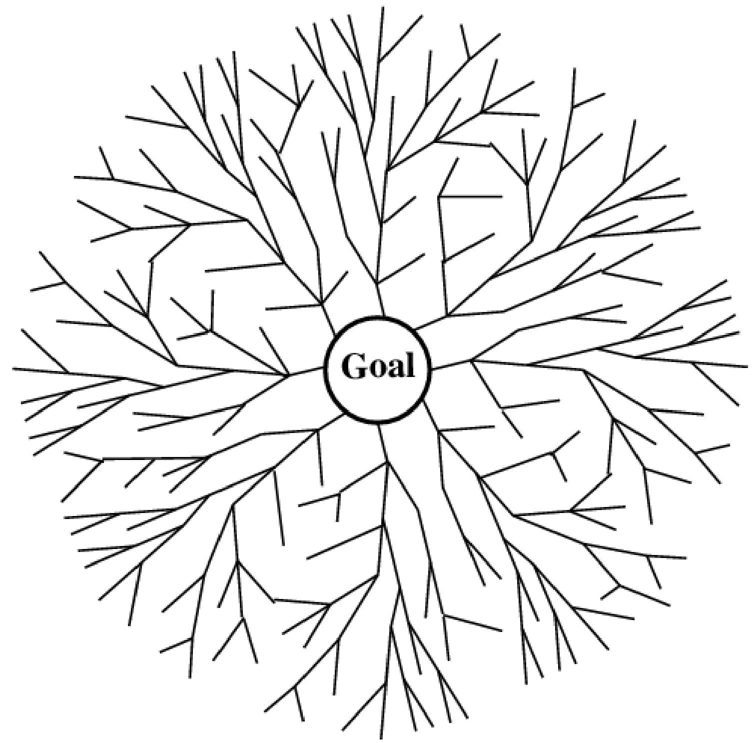
Cost of Iterative Deepening

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 can 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"