

Search Algorithms

LECTURE 2 - part I

Uninformed Search Algorithms

Informed Search Algorithms

Solving problems by searching

Basic search algorithms

Breadth First Search • Uniform Cost Search
Depth First Search • Depth Limited Search
Iterative Deepening Search • Bidirectional Search

Real World Problems

- Route finding — robot navigation, airline travel planning, computer/phone networks
- Travelling salesman problem — planning movement of automatic circuit board drills
- LSI layout — design silicon chips
- Assembly sequencing — scheduling assembly of complex objects,
- Mixed/constrained problems — courier delivery, product distribution, fault service and repair
- manufacturing process control

- These are optimisation problems but mathematical (operations research) techniques are not always effective.

Solving problems by searching

- Search as a “weak method” of problem solving with wide applicability
- Uninformed search methods (use no problem-specific information)
- Informed search methods (use heuristics to improve efficiency)

Solving problems by searching

- Search as a “weak method” of problem solving with wide applicability
- Uninformed search methods (use no problem-specific information)
 - Uninformed (or “blind”) search strategies use only the information available in the problem definition (can only distinguish a goal from a non-goal state)
- Informed search methods (use heuristics to improve efficiency)
 - Informed (or “heuristic”) search strategies use task-specific knowledge.

State Space Search Problems

- State space — set of all states reachable from initial state(s) by any action sequence
- Initial state(s) — element(s) of the state space
- Transitions
 - Operators — set of possible actions at agent's disposal; describe state reached after performing action in current state, or
 - Successor function — $s(x)$ = set of states reachable from state x by performing a single action
- Goal state(s) — element(s) of the state space
- Path cost — cost of a sequence of transitions used to evaluate solutions (apply to optimization problems)

Romania Map

Task: go from Arad to Bucharest

What is the state?

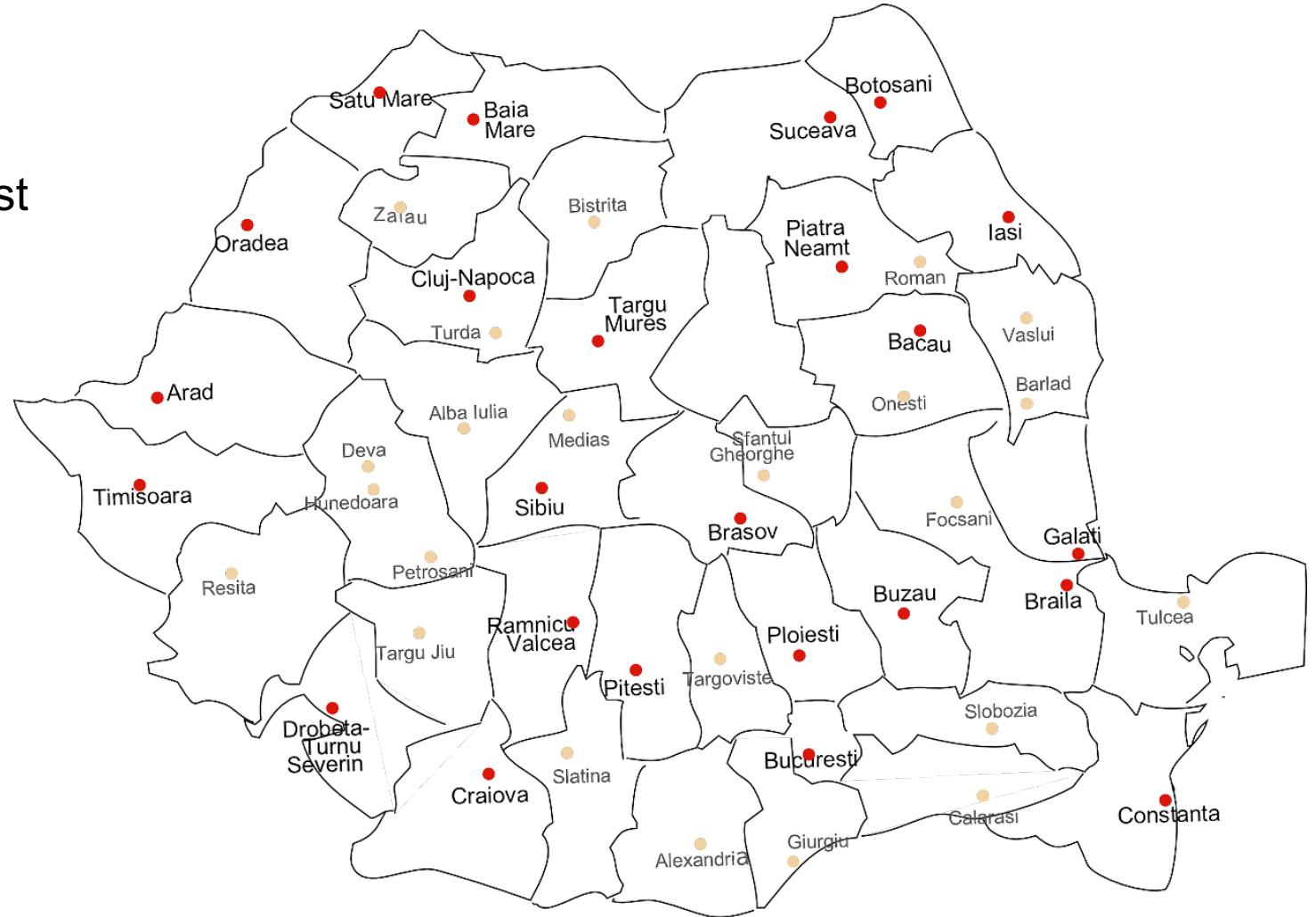
Start state?

Goal state?

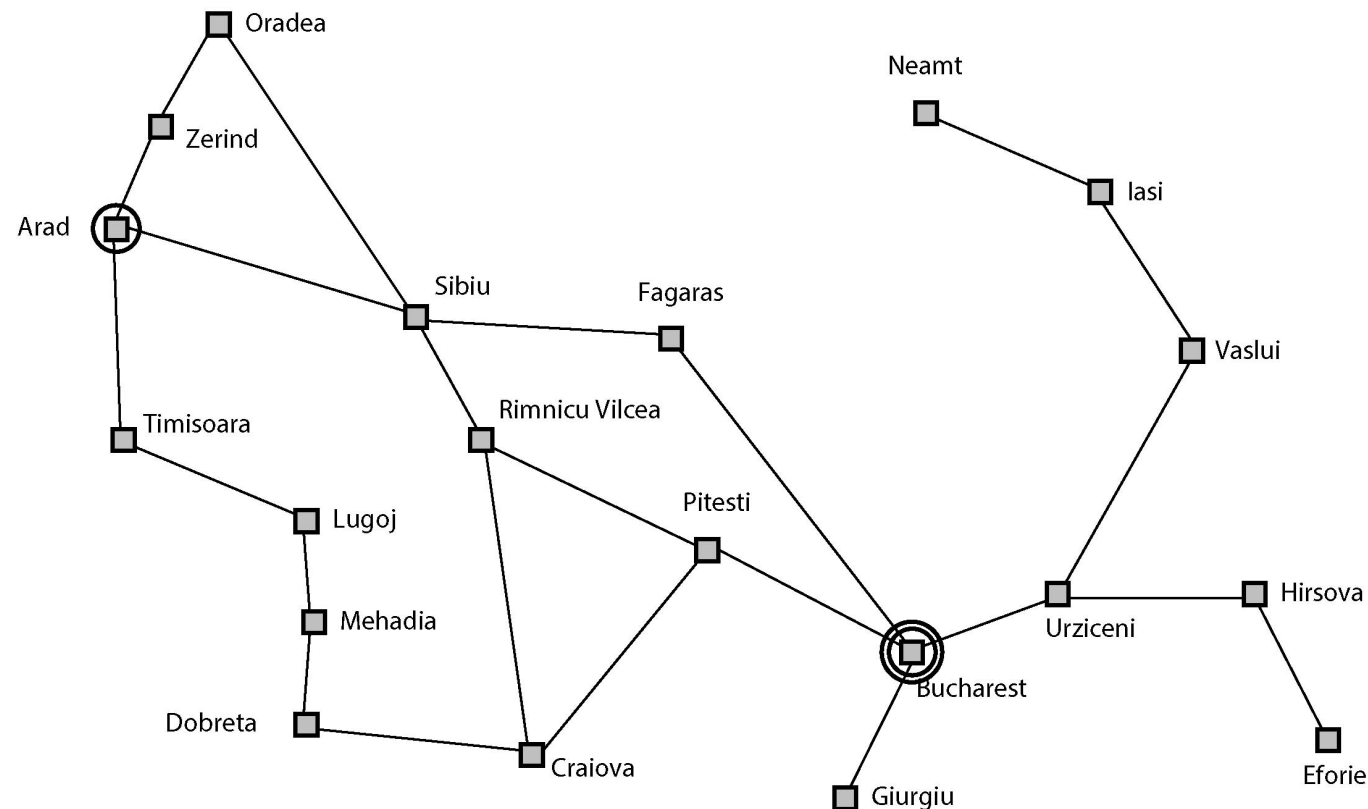
What is the domain?

What are the actions?

What is the solution(s)?

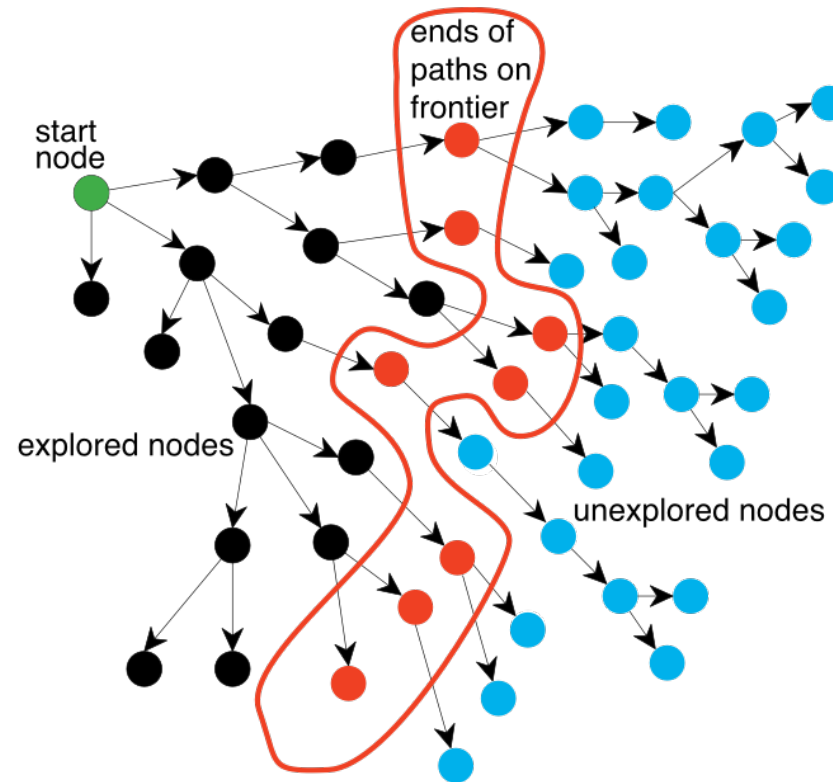


State-Space Graph for Romanian Street Path Problem



This can be modelled as a state-space search problem, where the states are locations.

Problem Solving by Graph Searching

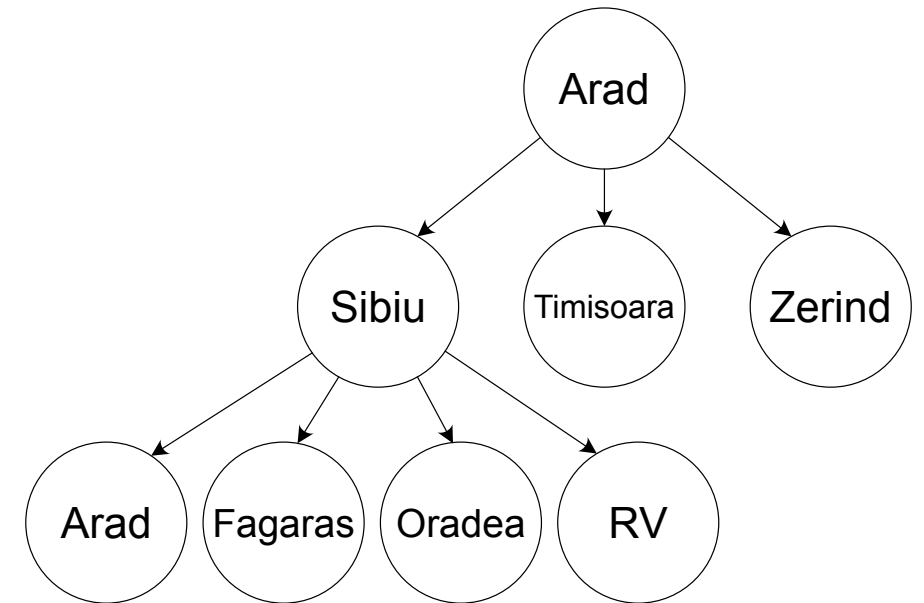
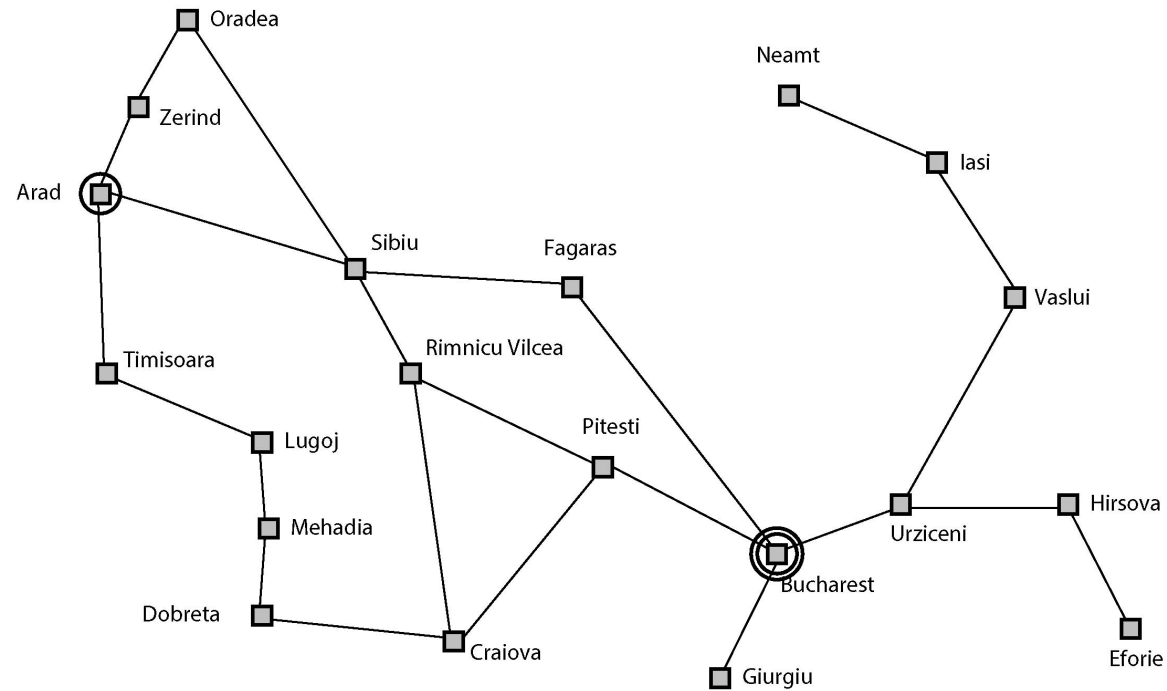


Search strategy — way in which frontier expands

Search Tree

- Search tree: superimposed over the state space.
- Root: search node corresponding to the initial state.
- Leaf nodes: correspond to states that have no successors in the tree because they were not expanded or generated no new nodes.

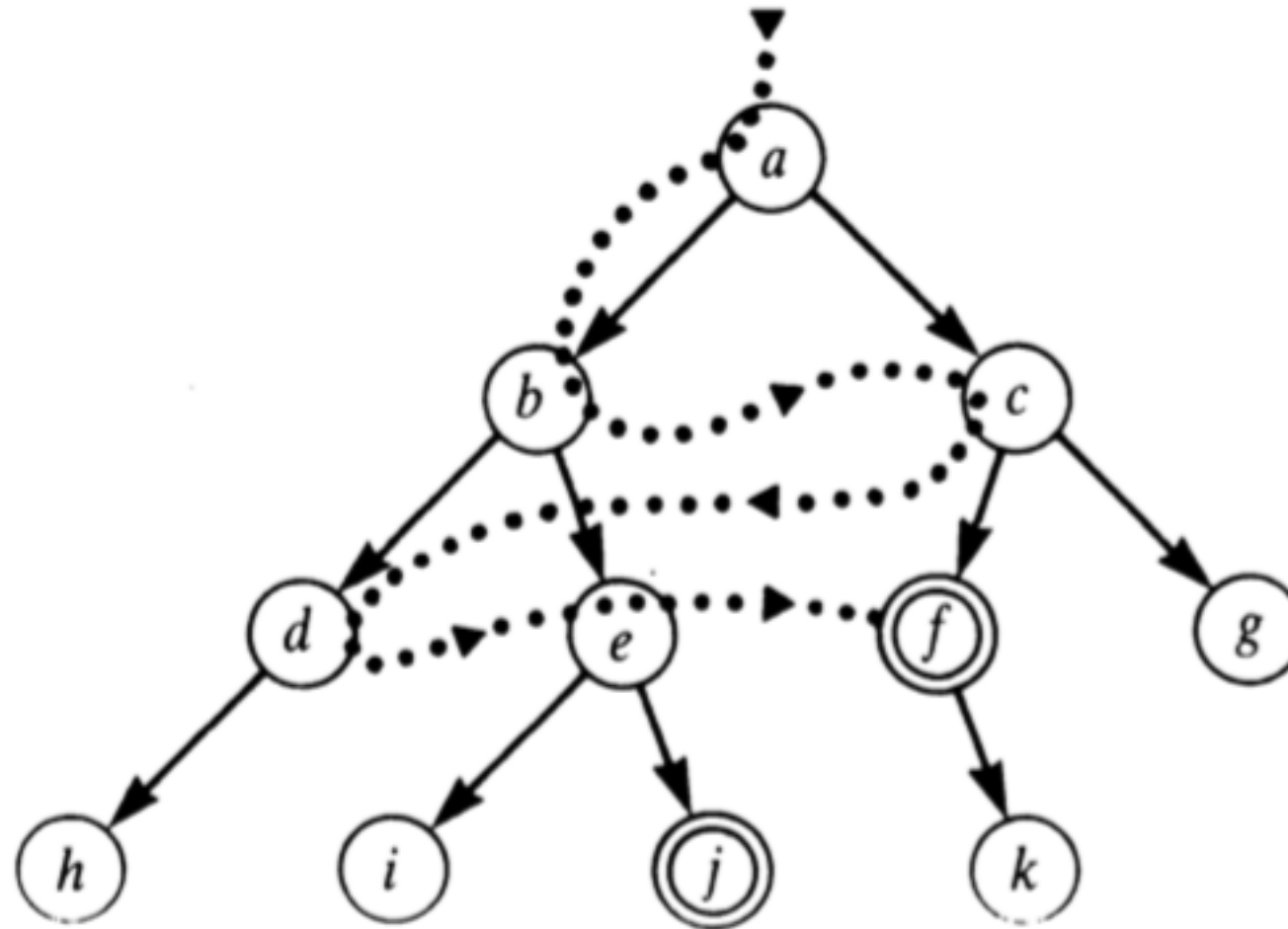
Romania Map - Graph Searching



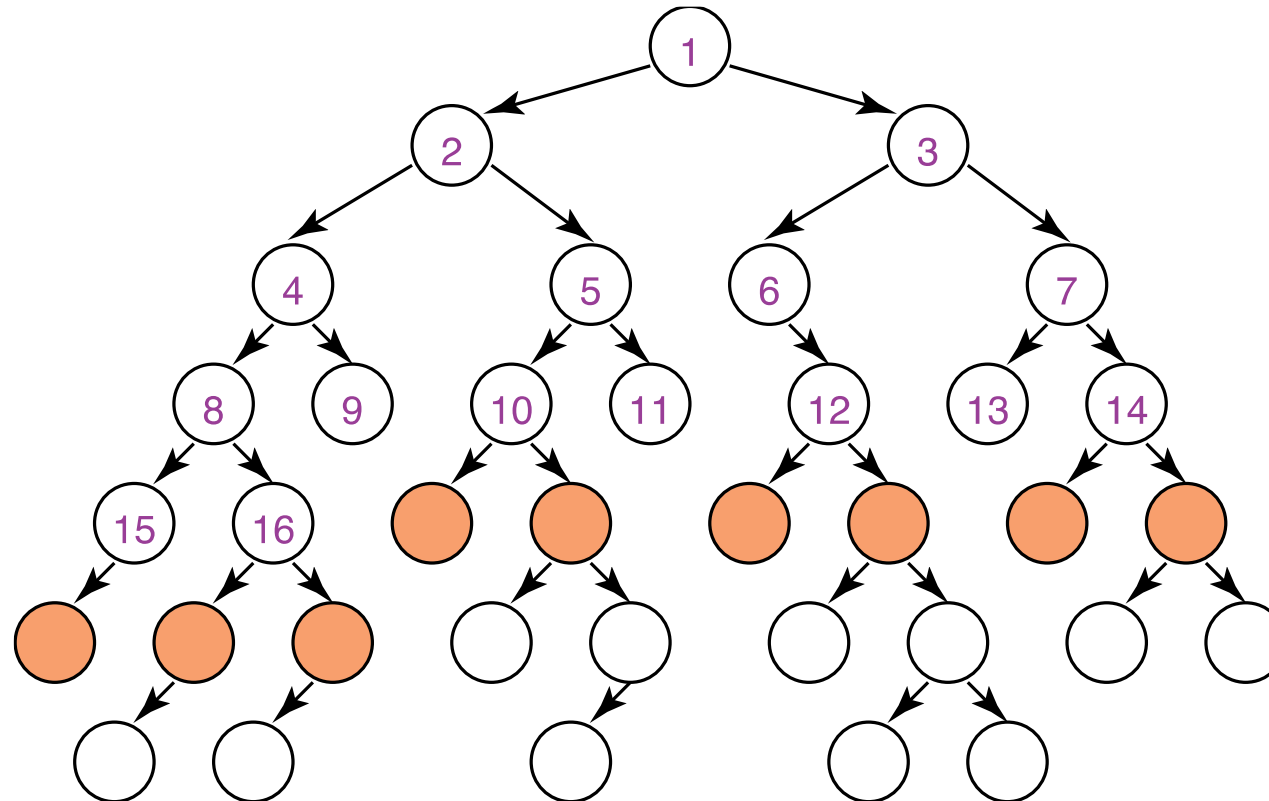
Breadth-first Search

- Breadth-first search treats the frontier as a queue.
- It always selects the earliest element added to the frontier.
- If the list of paths on the frontier is $[p_1, p_2, \dots, p_r]$:
 - p_1 is selected. Its neighbours are added to the end of the queue, after p_r .
 - p_2 is selected next.

Breadth-first Search



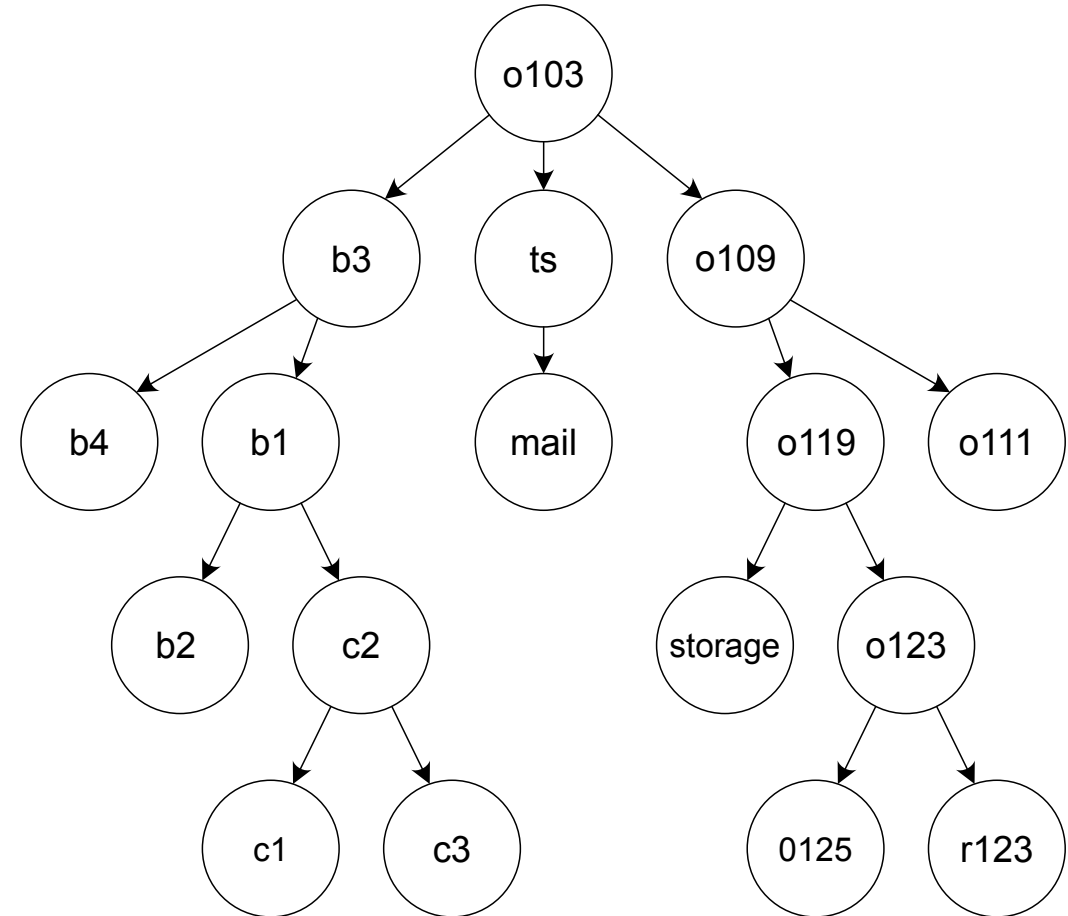
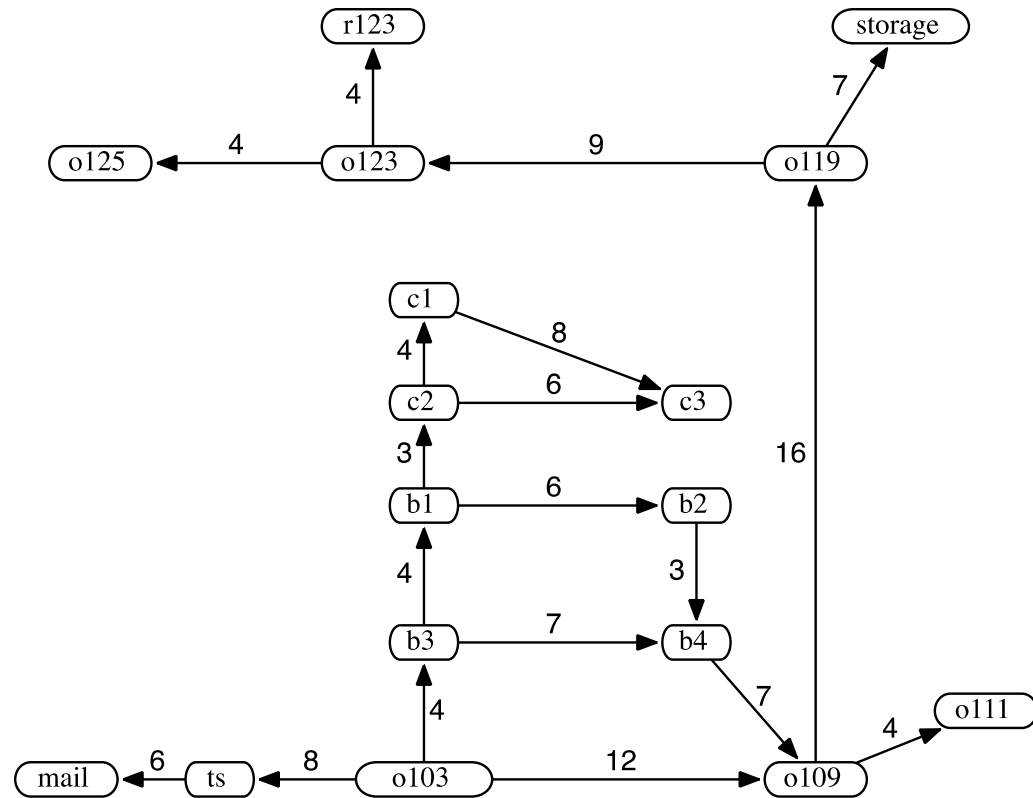
Breadth-first Search



Breadth-First Search

- All nodes are expanded at a given depth in the tree before any nodes at the next level are expanded
- Can be implemented by using a queue to store frontier nodes
 - put newly generated successors at end of queue
- Stop when node with goal state is generated
- Include check that generated state has not already been explored
 - Needs a new data structure for set of explored states
- Very systematic
- Finds the shallowest goal first

Breadth-First Search



Complexity of Breadth-first Search

- Does breadth-first search guarantee to find the path with fewest arcs?
- What happens on infinite graphs or on graphs with cycles if there is a solution?
- What is the time complexity as a function of the length of the path selected?
- What is the space complexity as a function of the length of the path selected?
- How does the goal affect the search?

Demonstration — Breadth-first Search

Properties of breadth-first search

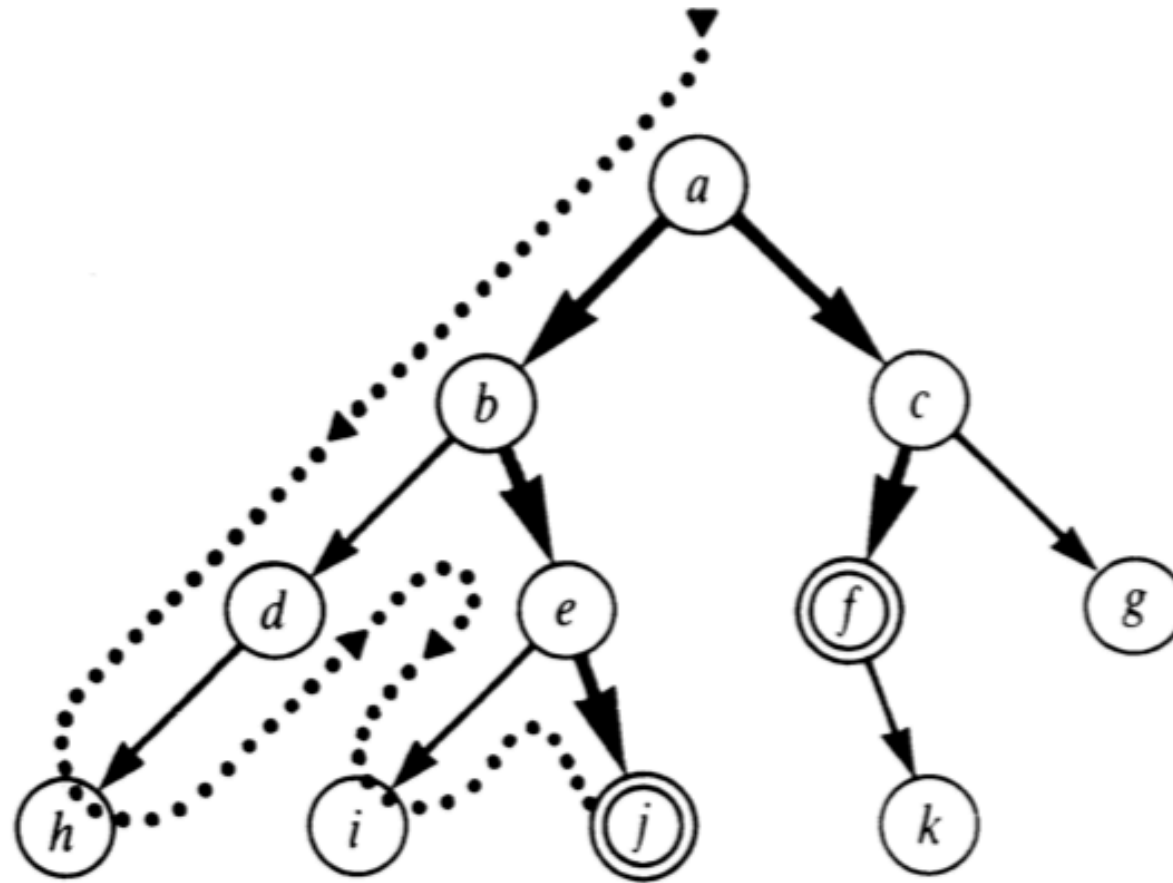
Algorithm	Completeness	Admissibility	Space	Time
Breadth-First Search	guaranteed, if b is finite	guaranteed, if arcs have the same cost	$O(b^d)$	$O(b^d)$

- Complete? Yes, *it terminates with a solution when one exists*
- Time? $b^d + b^{d-1} + b^{d-2} + \dots + b^0 = O(b^d)$
- Space? $O(b^d)$ *keeps every node in memory; generate all nodes up to level d*
- Optimal? Yes, but only if all actions have the same cost
- Space is the big problem for BFS; it grows exponentially with depth!

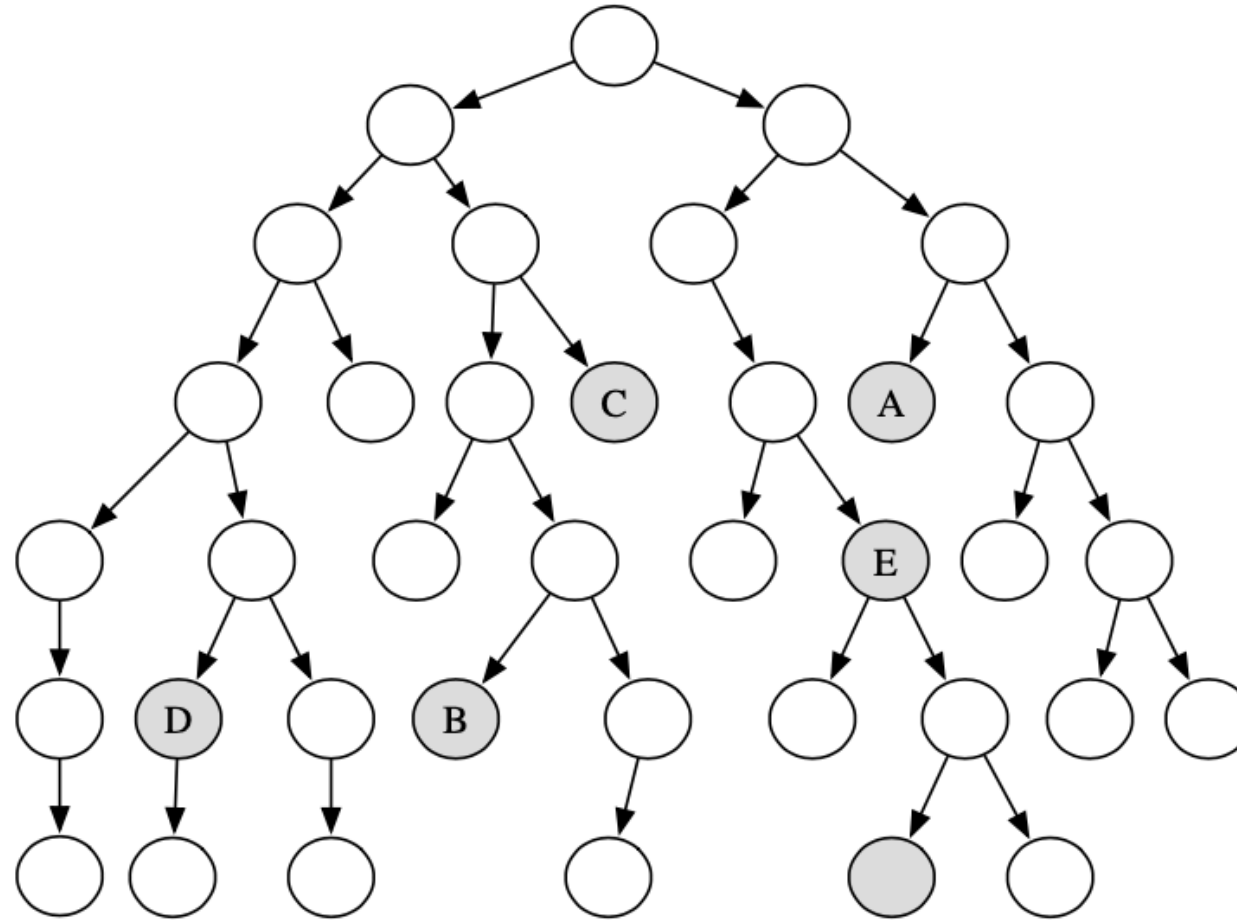
Depth First Search

- Expands one of the nodes at the deepest level of the tree
- Implementation:
 - Implementing the frontier as a stack = insert newly generated states at the front of the queue (thus making it a stack)
 - can alternatively be implemented by recursive function calls
- In depth-first search, like breadth-first search, the order in which the paths are expanded does not depend on the goal.

Depth-first Search - DFS



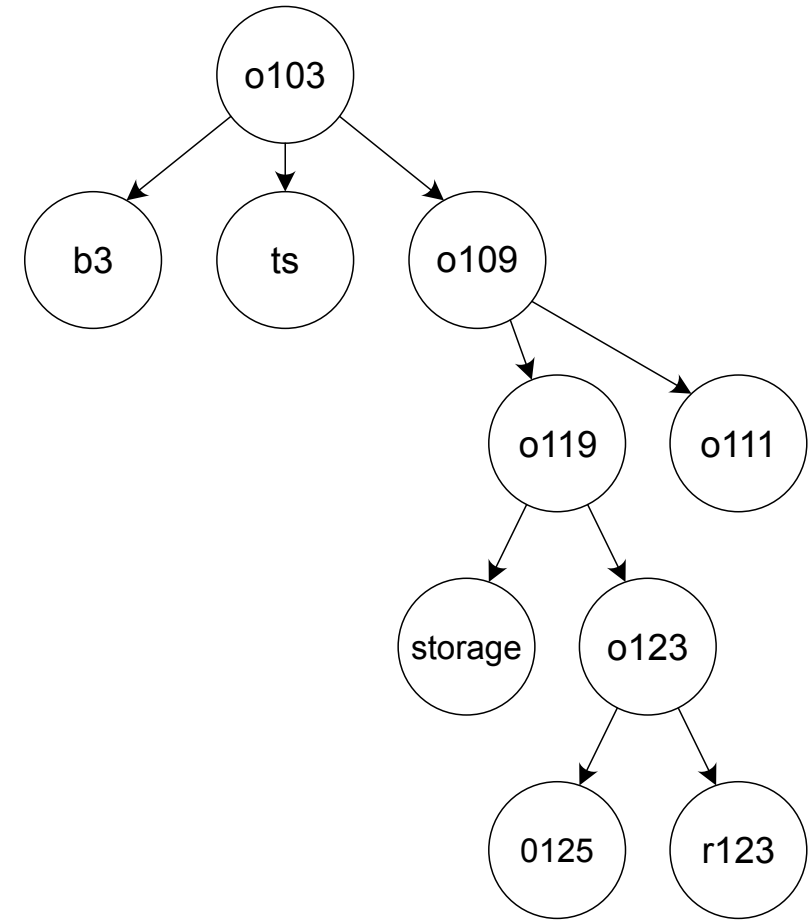
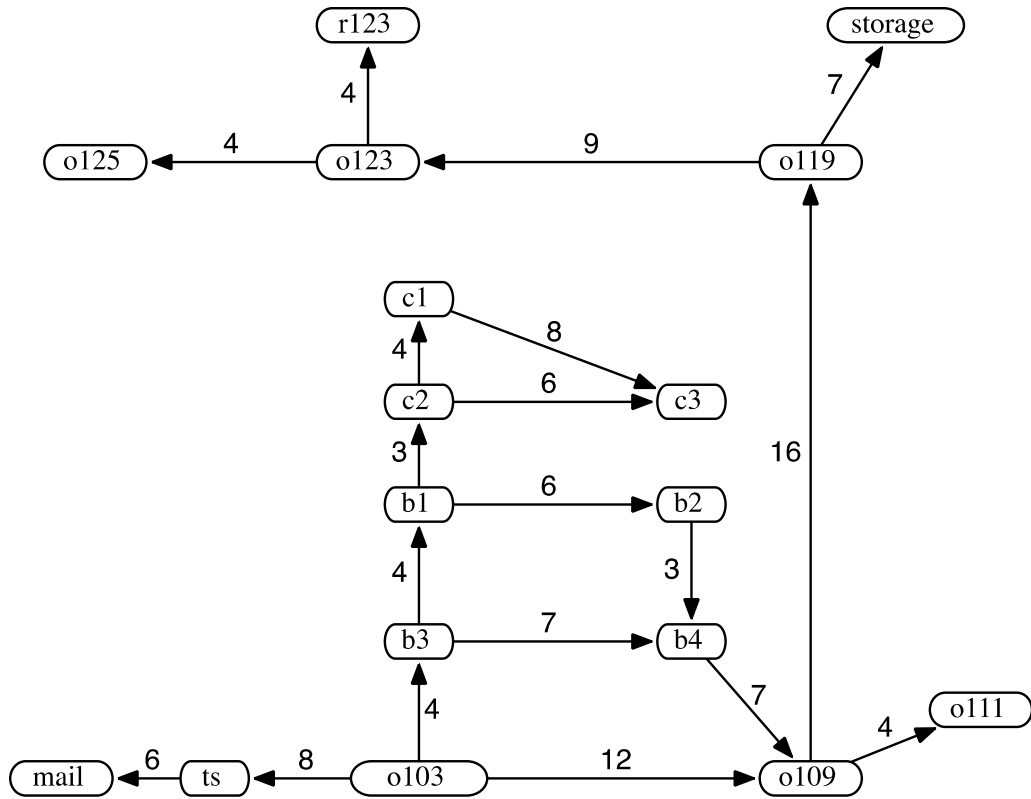
Which shaded goal will depth-first search find first?



Depth First Search

- Idea: Always expand node at deepest level of tree and when search hits a dead-end return back to expand nodes at a shallower level
- Can be implemented using a stack of explored + frontier nodes
- At any point depth-first search stores single path from root to leaf
 - together with any remaining unexpanded siblings of nodes along path
- Stop when node with goal state is expanded
- Include check that generated state has not already been explored along a path – cycle checking

Depth-First Search



Demonstration — Depth-first Search

Properties of depth-first search

Algorithm	Completeness	Admissibility	Space	Time
Depth-First Search	not guaranteed	not guaranteed	$O(bm)$	$O(b^m)$

- Complete? No! fails in infinite-depth spaces, spaces with loops
 - Modify to avoid repeated states along path in finite spaces
- Time? $O(bm)$, m = maximum depth of search tree
 - terrible if m is much larger than d
 - but if solutions are dense, can be much faster than breadth-first search
- Space? $O(bm)$, i.e., linear space!
- Optimal? No, can find suboptimal solutions first.

Depth-First Search — Analysis

- In cases where problem has many solutions, depth-first search may outperform breadth-first search because there is a good chance it will find a solution after exploring only a small part of the space
- However, depth-first search may get stuck following a deep or infinite path even when a solution exists at a relatively shallow level
- Therefore, depth-first search is not complete and not optimal
 - Avoid depth-first search for problems with deep or infinite path

Lowest-cost-first Search - Uniform-Cost Search

- Sometimes there are costs associated with arcs. The cost of a path is the sum of the costs of its arcs.

$$\text{cost}(\langle n_0, \dots, n_k \rangle) = \sum_{i=1}^k \text{cost}(\langle n_{i-1}, n_i \rangle)$$

- An optimal solution is one with minimum cost.
- For many domains, arcs have non-unit costs, the aim is to find an optimal solution, a solution such that no other solution has a lower total cost.
 - For example, for a delivery robot, the cost of an arc may be resources (e.g., time, energy) required by the robot to carry out the action represented by the arc, and the aim is for the robot to solve a given goal using fewest resources.

Lowest-cost-first Search - Uniform-Cost Search

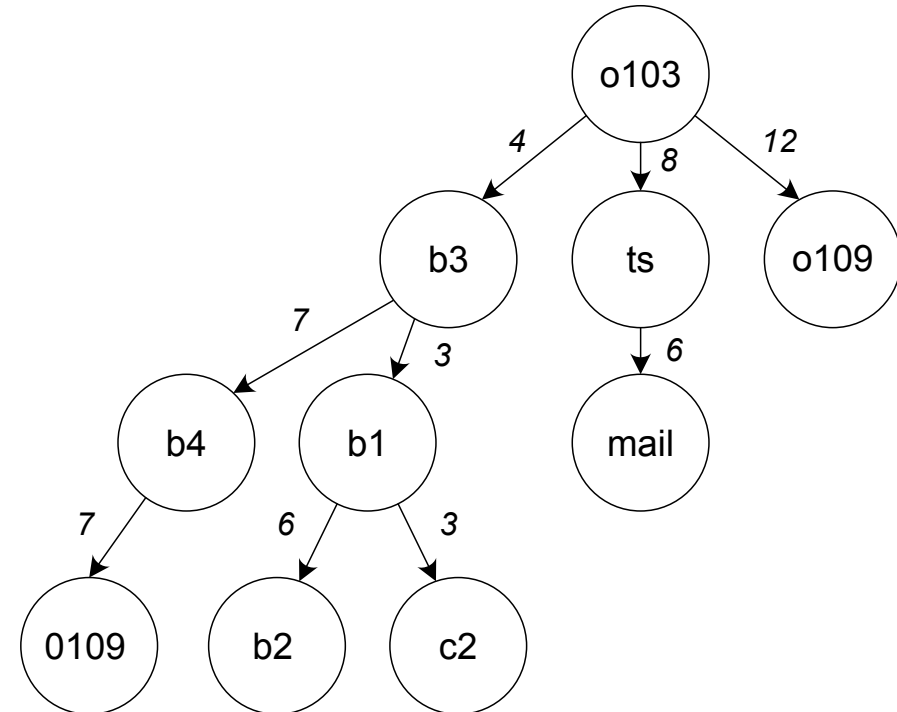
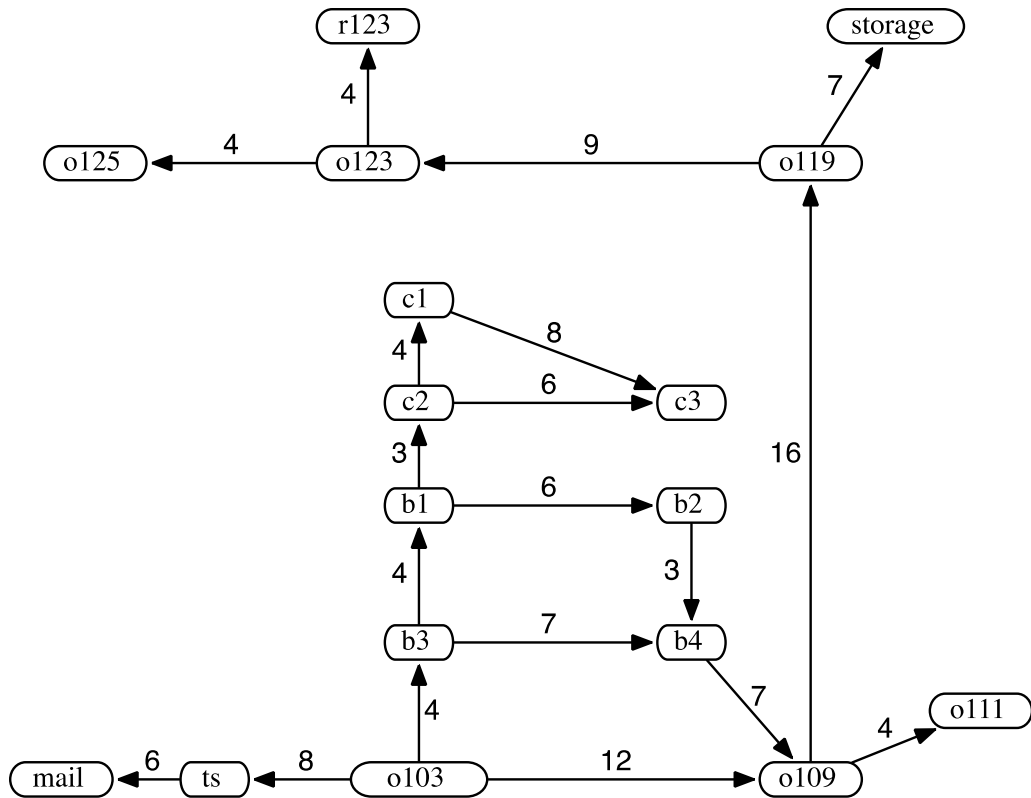
- The simplest search method that is guaranteed to find a minimum cost path is Lowest-Cost-First Search or Uniform-Cost Search
 - is similar to breadth-first search, but it selects a path with the lowest cost.
 - the frontier is implemented as a priority queue ordered by the **cost function**.

$$cost(\langle n_0, \dots, n_k \rangle) = \sum_{i=1}^k cost(\langle n_{i-1}, n_i \rangle)$$

Uniform-Cost Search

- Expand root first, then expand least-cost unexpanded node
- Implementation: priority queue = insert nodes in order of increasing path cost - (lowest path cost $g(n)$).
- Reduces to Breadth First Search when all actions have same cost
- Finds the cheapest goal provided path cost is monotonically increasing along each path (i.e. no negative-cost steps)

Uniform-Cost Search



Properties of Uniform-Cost Search

Algorithm	Completeness	Admissibility	Space	Time
Uniform-Cost Search	guaranteed, if b is finite and cost $\geq \epsilon$	guaranteed	$O(b^{[1 + C / \epsilon]})$	$O(b^{[1 + C / \epsilon]})$

- Complete? Yes, if b is finite and if step cost $\geq \epsilon$ with $\epsilon > 0$
- Time? $O(b^{[1 + C / \epsilon]})$ where C^* = cost of the optimal solution and assume every action costs at least ϵ
- Space? $O(b^{[1 + C / \epsilon]})$
- Optimal? Yes – nodes expanded in increasing order of $g(n)$

Depth Limited Search

Algorithm	Completeness	Admissibility	Space	Time
Depth Limited Search	guaranteed, if b is finite	not guaranteed	$O(bk)$	$O(bk)$

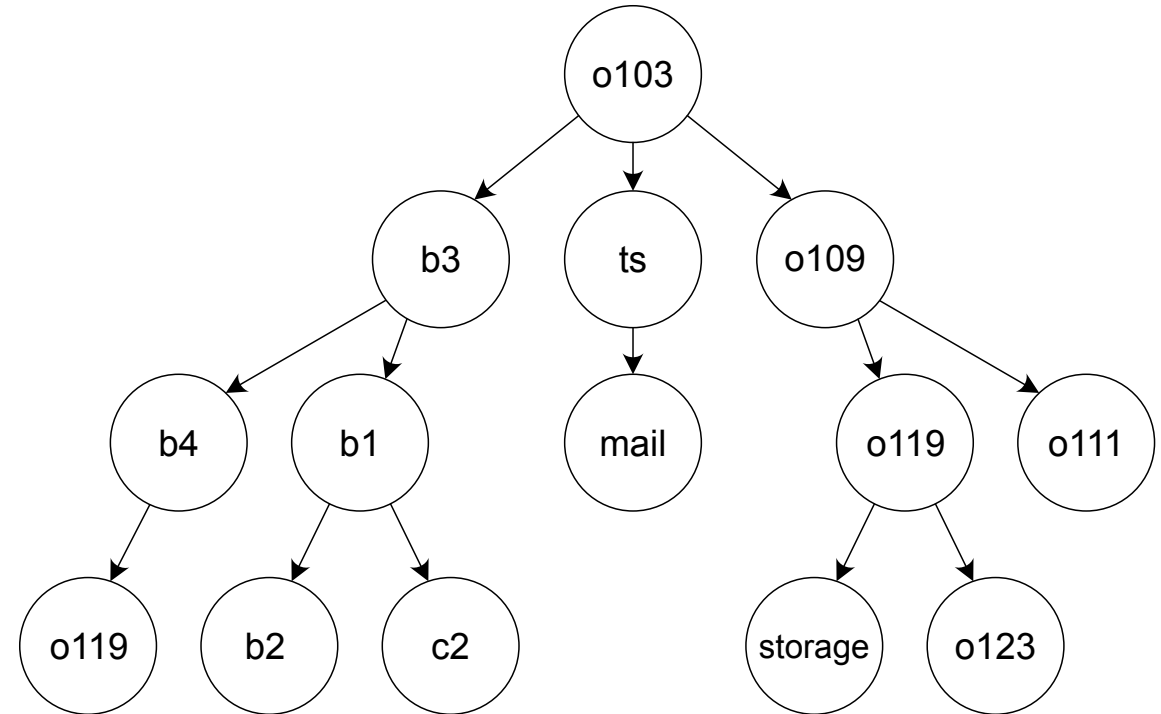
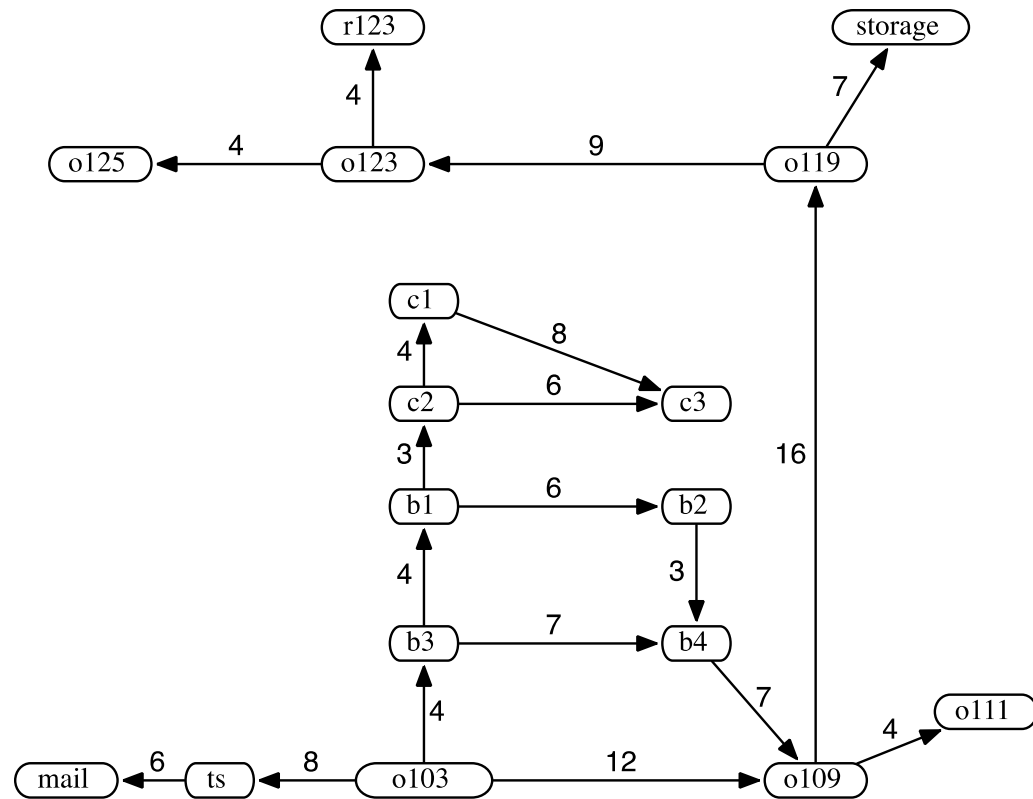
- Expands nodes like Depth First Search but imposes a cutoff on the maximum depth of path.
- Complete? Yes (no infinite loops anymore)
- Time? $O(bk)$ where k is the depth limit
- Space? $O(bk)$, i.e., linear space similar to DFS!
- Optimal? No, can find suboptimal solutions first.

Problem: How to pick a good limit ?

Iterative Deepening Search

- Tries to combine the benefits of depth-first (low memory) and breadth-first (optimal and complete) by doing a series of depth- limited searches to depth 1, 2, 3, etc.
- Early states will be expanded multiple times, but that might not matter too much because most of the nodes are near the leaves.

Iterative Deepening Search



Iterative Deepening Search

- It can be very difficult to decide upon a depth limit for search
- The maximum path cost between any two nodes is known as the diameter of the state space
- This would be a good candidate for a depth limit but it may be difficult to determine in advance
- Idea: Try all possible depth limits in turn
- Combines benefits of depth-first and breadth-first search

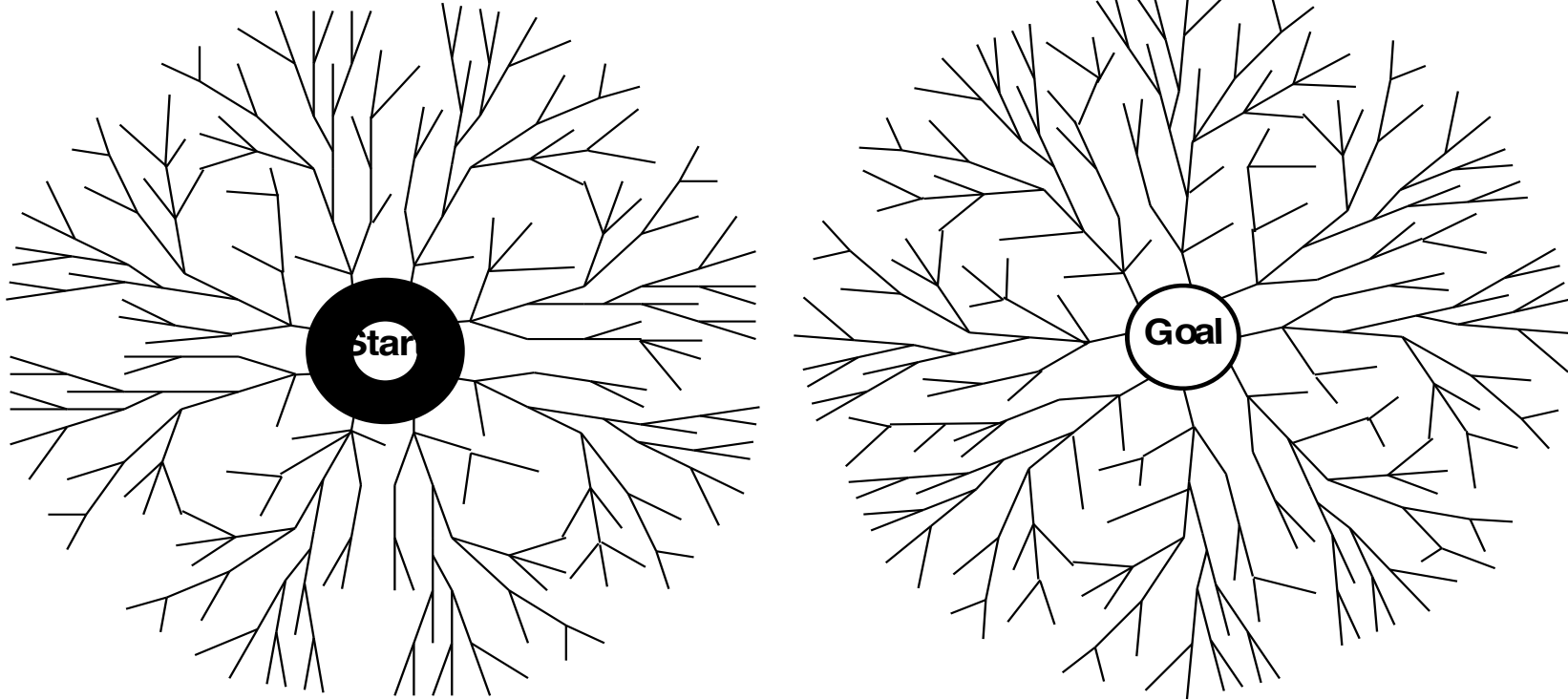
Demonstration — Iterative Deepening Search

Properties of Iterative Deepening Search

Algorithm	Completeness	Admissibility	Space	Time
IDS	guaranteed, if b is finite	guaranteed, if arcs have the same cost	$O(bd)$	$O(b^d)$

- Complete? Yes.
- Time: $O(b^d)$ nodes at the top level are expanded once, nodes at the next level twice, and so on
- Space? $O(bd)$
- Optimal? Yes, if step costs are identical.
- In general, iterative deepening is the preferred search strategy for a large search space where depth of solution is not known

Bidirectional Search



Bidirectional Search

- Idea: Search both forward from the initial state and backward from the goal, and stop when the two searches meet in the middle.
- We need an efficient way to check if a new node already appears in the other half of the search. The complexity analysis assumes this can be done in constant time, using a Hash Table.
- Assume branching factor = b in both directions and that there is a solution at depth = d . Then bidirectional search finds a solution in $O(2b^{d/2}) = O(b^{d/2})$ time steps.
- To check for intersection it must have all states from one of the searches in memory, therefore space complexity is $O(b^{d/2}) + O(b^{d/2}) = 2O(b^{d/2}) = O(b^{d/2})$

Bidirectional Search — Analysis

Algorithm	Completeness	Admissibility	Space	Time
Bidirectional Search	depends on search algorithms used	depends on search algorithms used	$O(b^{d/2})$	$O(b^{d/2})$

- Complete? Depends on the search algorithm used. If BFS, then yes.
- Time: $O(b^{d/2})$
- Space? $O(b^{d/2})$
- Optimal? Depends on the search algorithm used. If BFS, then yes.

Complexity Results for Uninformed Search

Algorithm	Completeness	Admissibility	Space	Time
Breadth-First Search	guaranteed, if b is finite	guaranteed, if arcs have the same cost	$O(b^d)$	$O(b^d)$
Depth-First Search	not guaranteed	not guaranteed	$O(bm)$	$O(b^m)$
Uniform Cost Search	guaranteed, if b is finite and cost $\geq \epsilon$	guaranteed	$O(b^{[1 + C / \epsilon]})$	$O(b^{[1 + C / \epsilon]})$
Depth Limited Search	guaranteed, if b is finite	not guaranteed	$O(bk)$	$O(bk)$
Iterative Deepening Search	guaranteed, if b is finite	guaranteed, if arcs have the same cost	$O(bd)$	$O(b^d)$
Bidirectional Search	depends on search algorithms used	depends on search algorithms used	$O(b^{d/2})$	$O(b^{d/2})$

Summary

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored.
- Variety of Uninformed search strategies
- Iterative Deepening Search uses only linear space and not much more time than other Uninformed algorithms.