COMP3411/9814: Artificial Intelligence

Solving problems by searching
Basic search algorithms

Lecture Overview

- 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 optimization 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

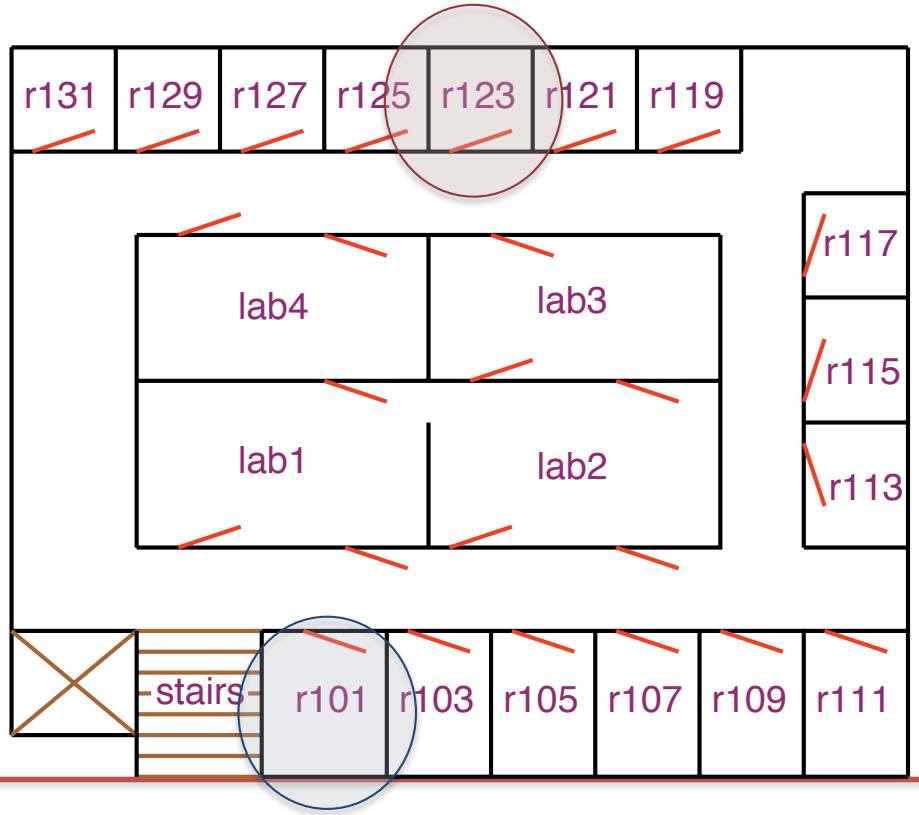
- State space — set of all states reachable from initial state(s) by any action sequence
- Initial state(s) — element(s) of the state space
- Transitions
  - Operators — set of possible actions at agent's disposal; describe state reached after performing action in current state, or
  - Successor function — $s(x)$= 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)

Domain for Delivery Robot

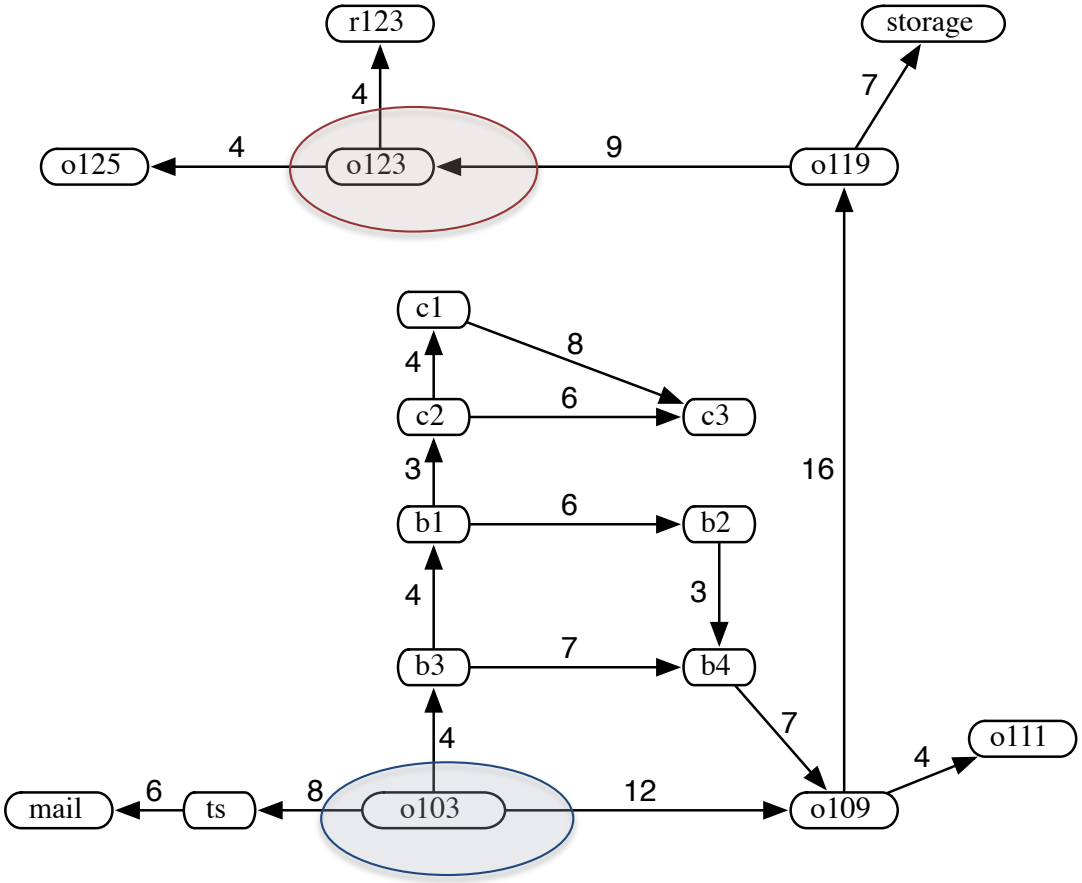
The robot wants to get from outside room 103 to the inside of room 123.

- In the robot delivery domain, the only way a robot can get through a doorway is to push the door open in the direction shown.

The task is to find a path from one location o103 to another to room r123

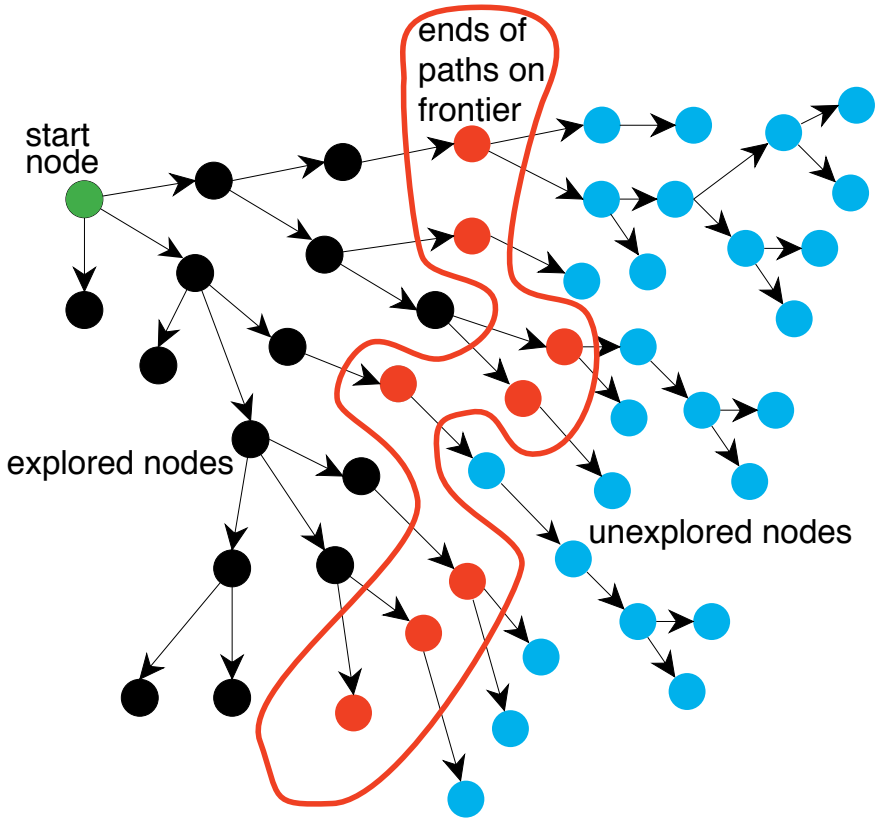


State-Space Graph for the Delivery Robot



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

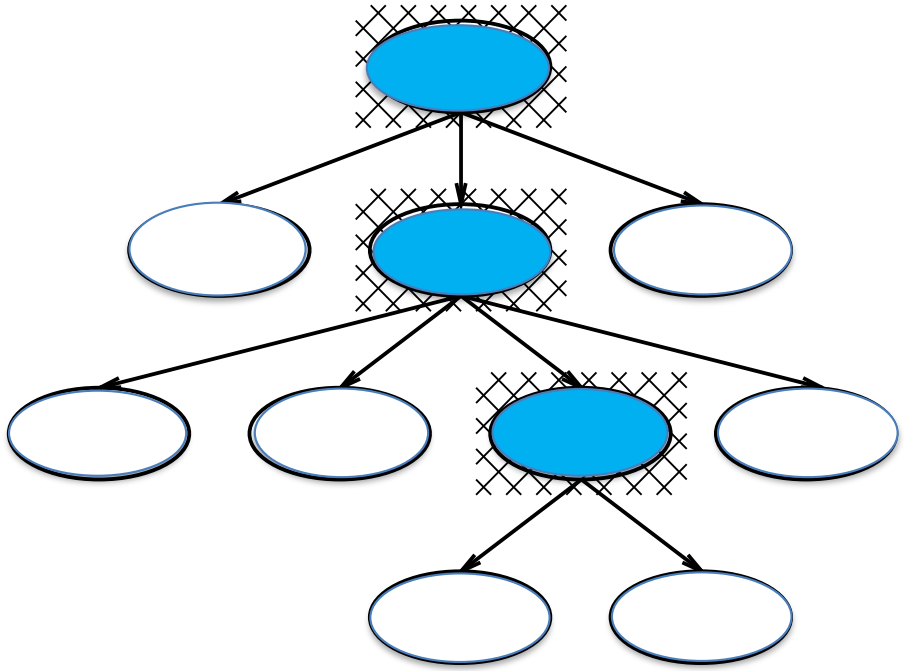
Problem Solving by Graph Searching



Search strategy — way in which frontier expands

Search Tree

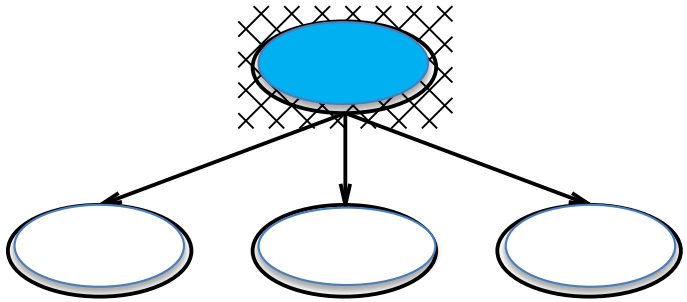
- Search tree: superimposed over the state space.
- Root: search node corresponding to the initial state.
- Leaf nodes: correspond to states that have no successors in the tree because they were not expanded or generated no new nodes.



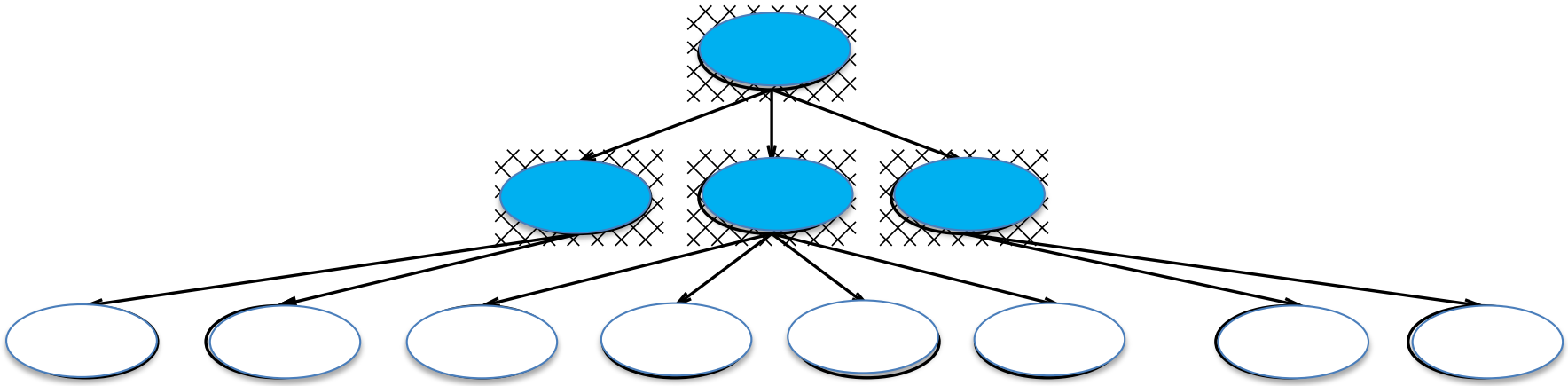
Breadth-first Search

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

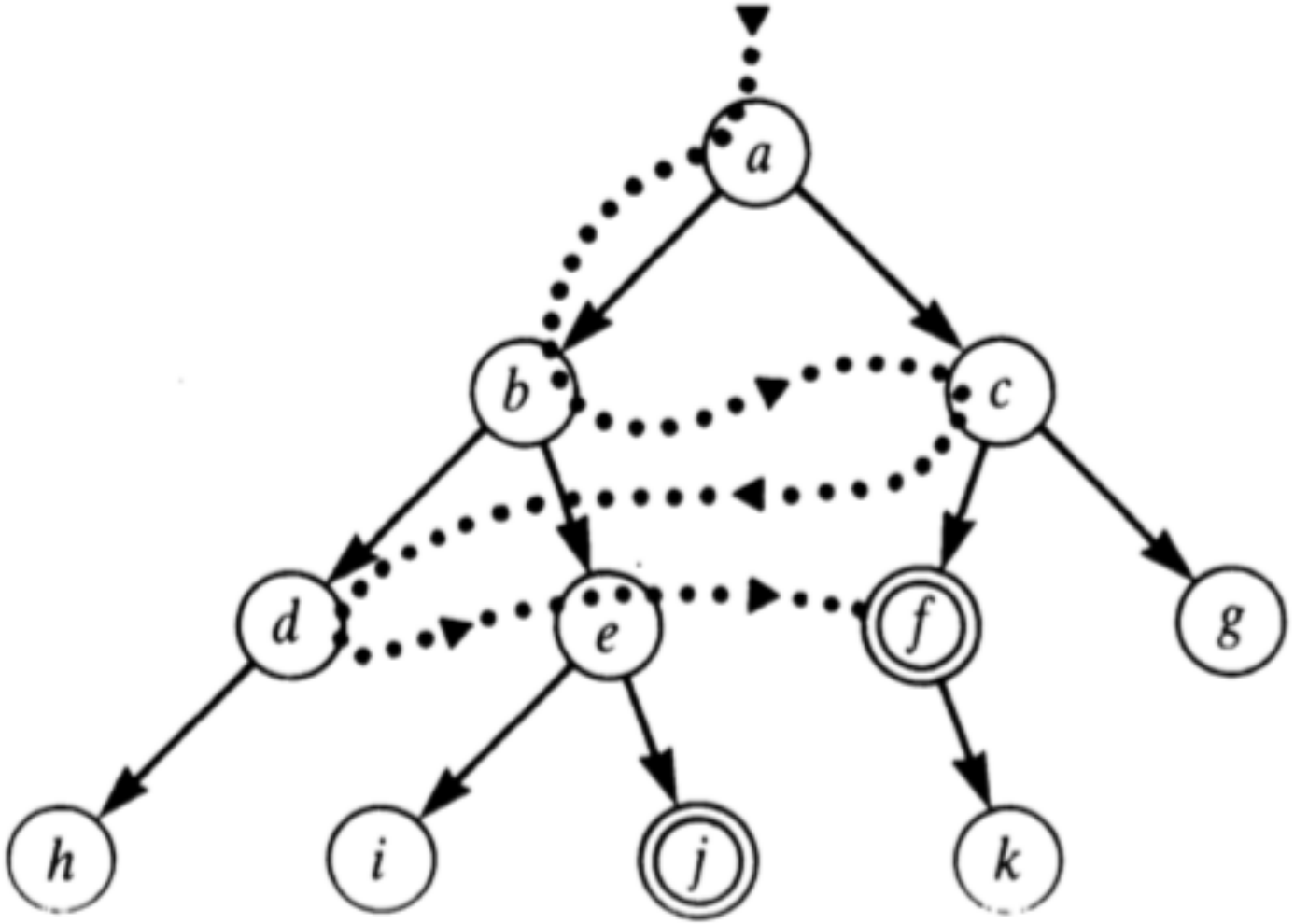
Breadth-First Search



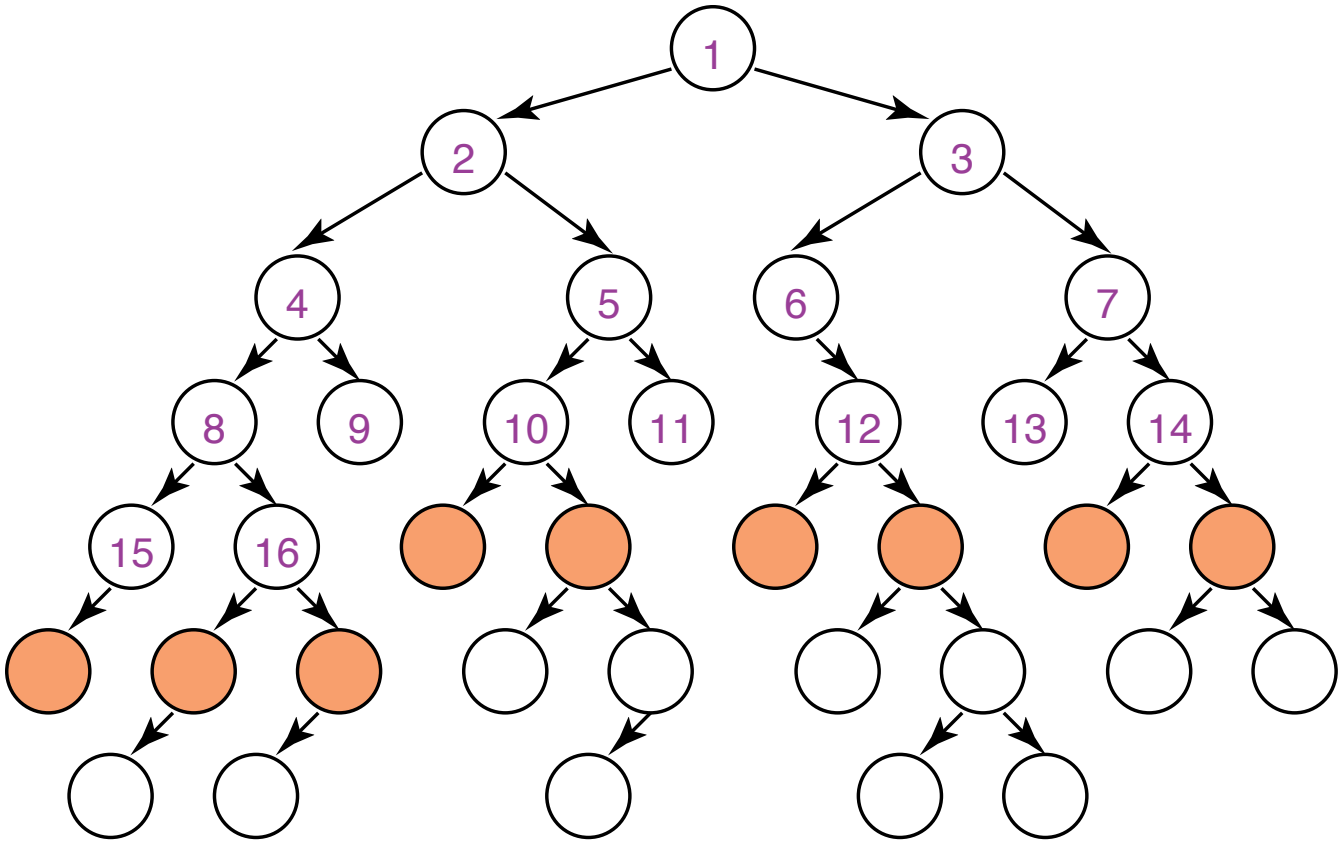
Breadth-First Search



Breadth-first Search



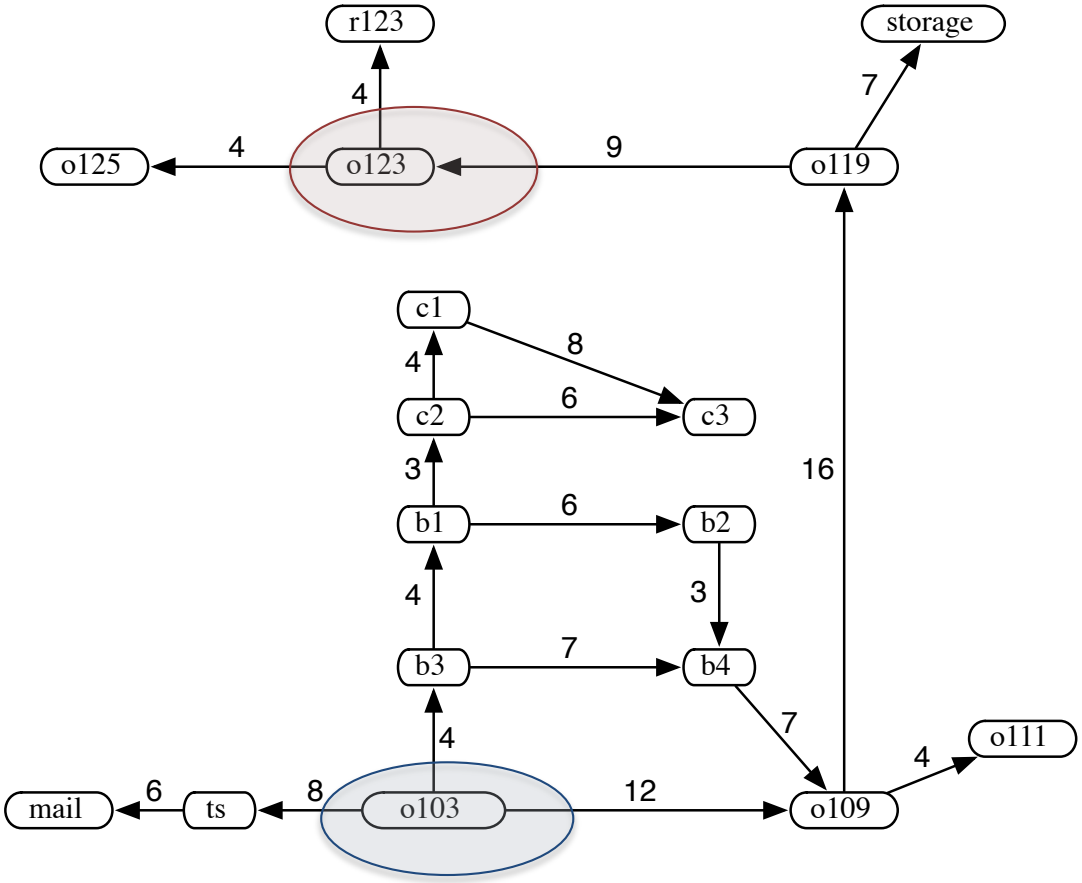
Breadth-first Search



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

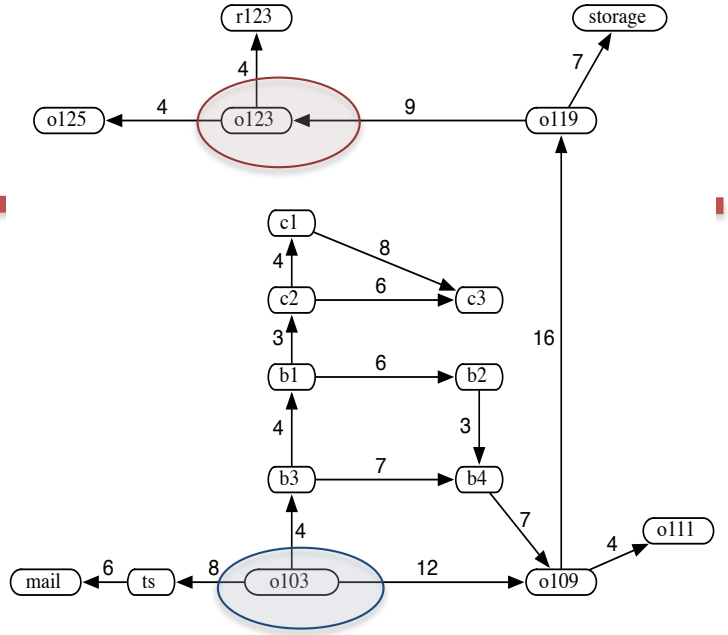
State-Space Graph for the Delivery Robot



A state-space search problem, where the states are locations.

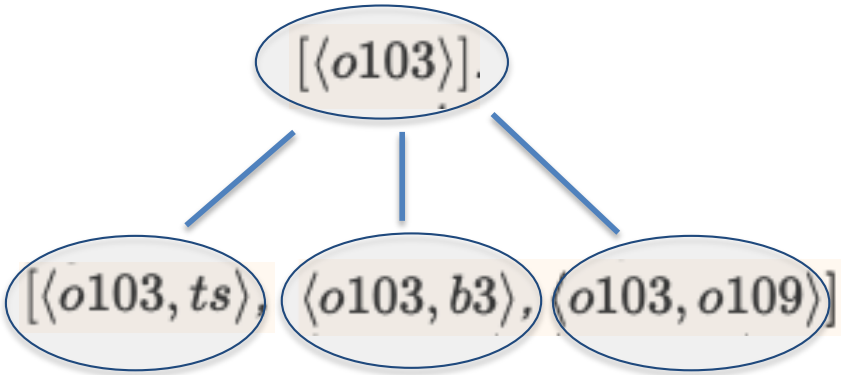
Breadth-First Search

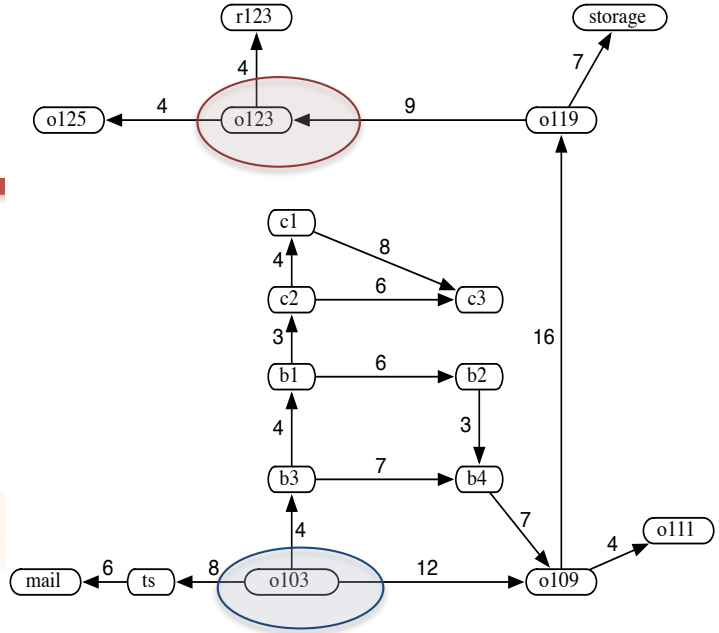
$[\langle o103 \rangle]$.



$[\langle o103 \rangle]$.

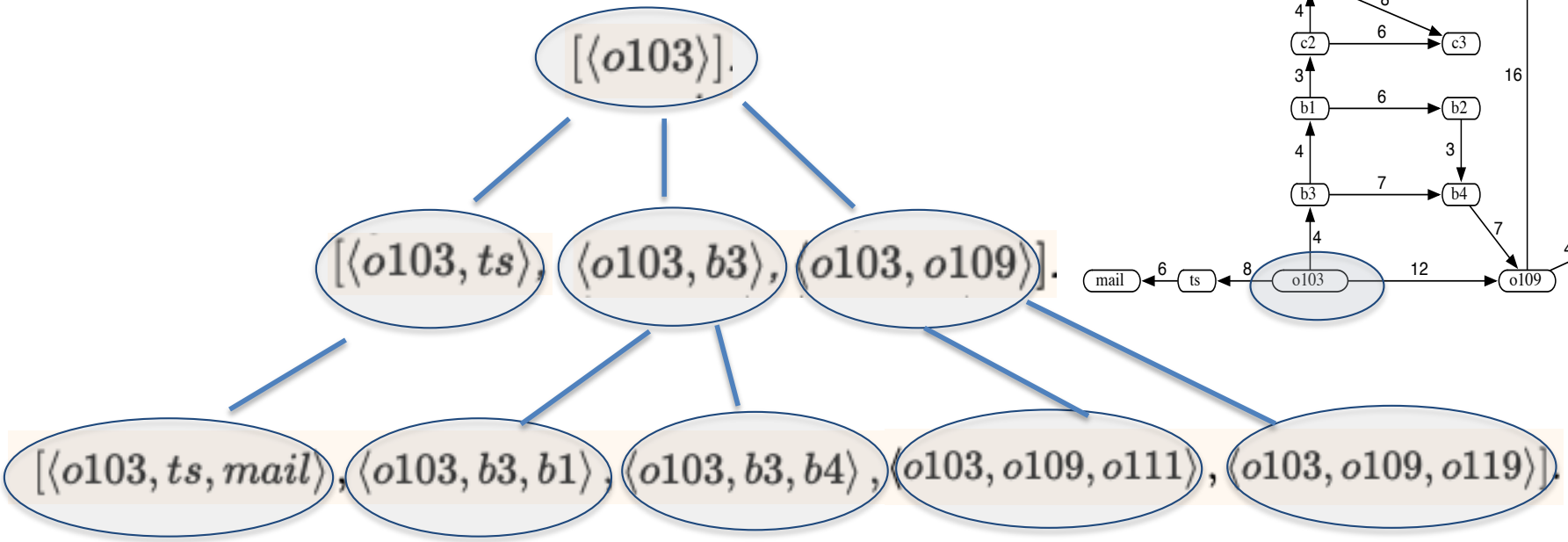
Breadth-First Search





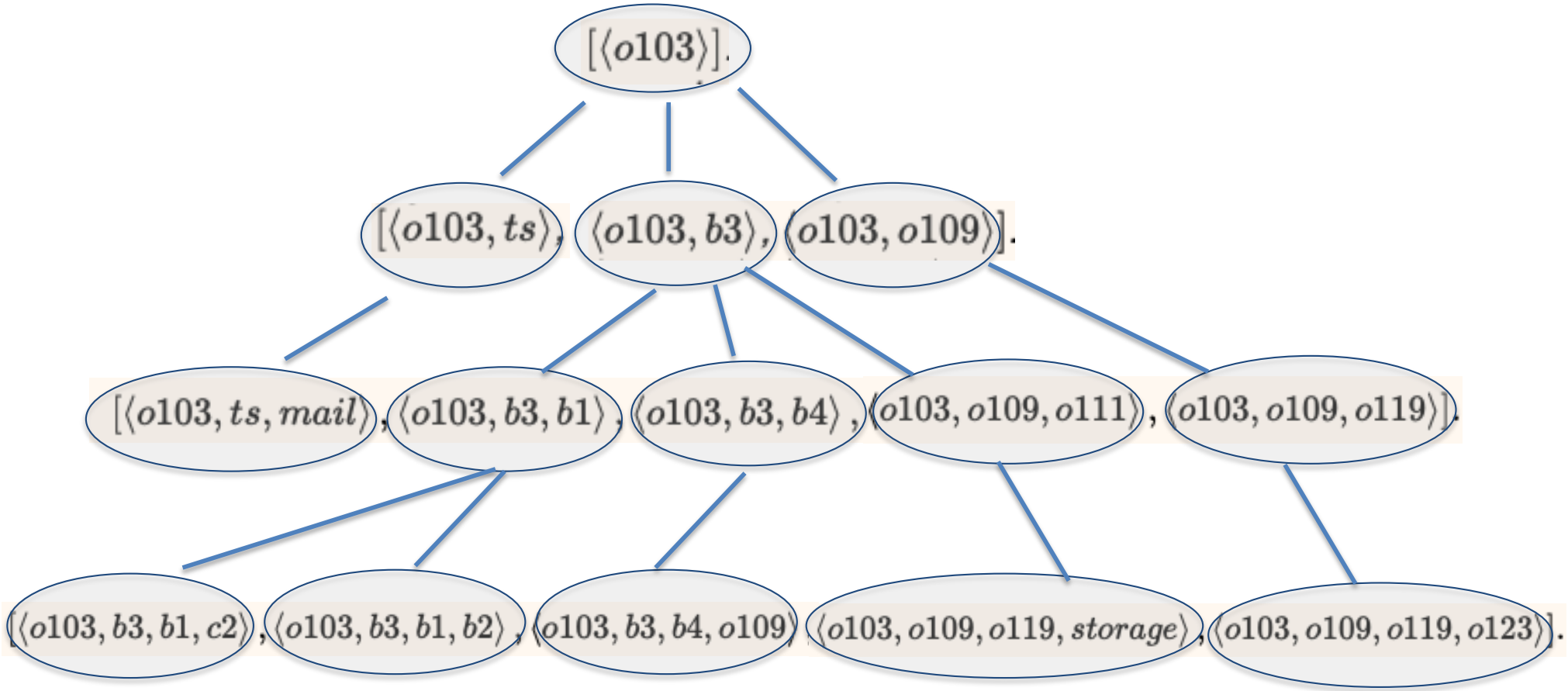
$[\langle o103, ts\rangle, \langle o103, b3\rangle, \langle o103, o109\rangle].$

Breadth-First Search

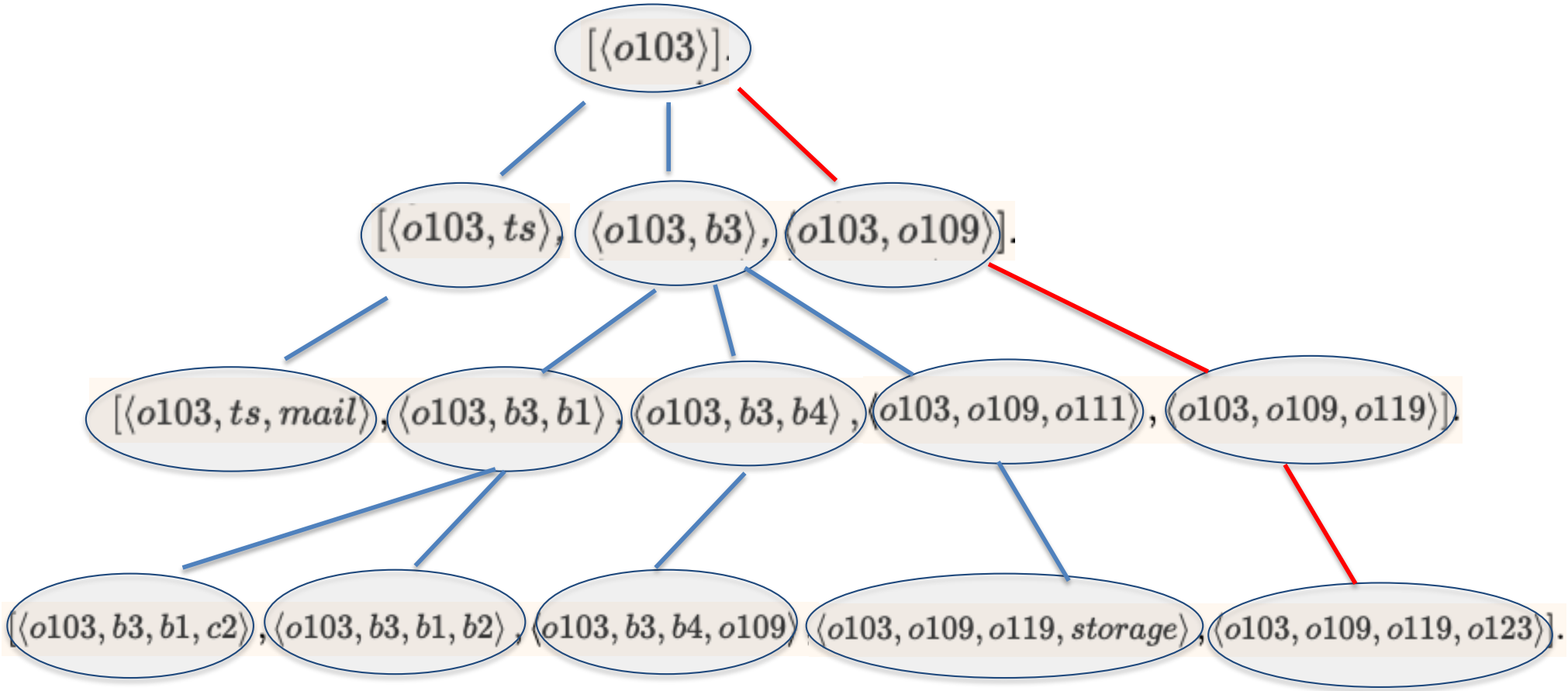


$[\langle o103, ts, mail \rangle, \langle o103, b3, b1 \rangle, \langle o103, b3, b4 \rangle, \langle o103, o109, o111 \rangle, \langle o103, o109, o119 \rangle]$.

Breadth-First Search



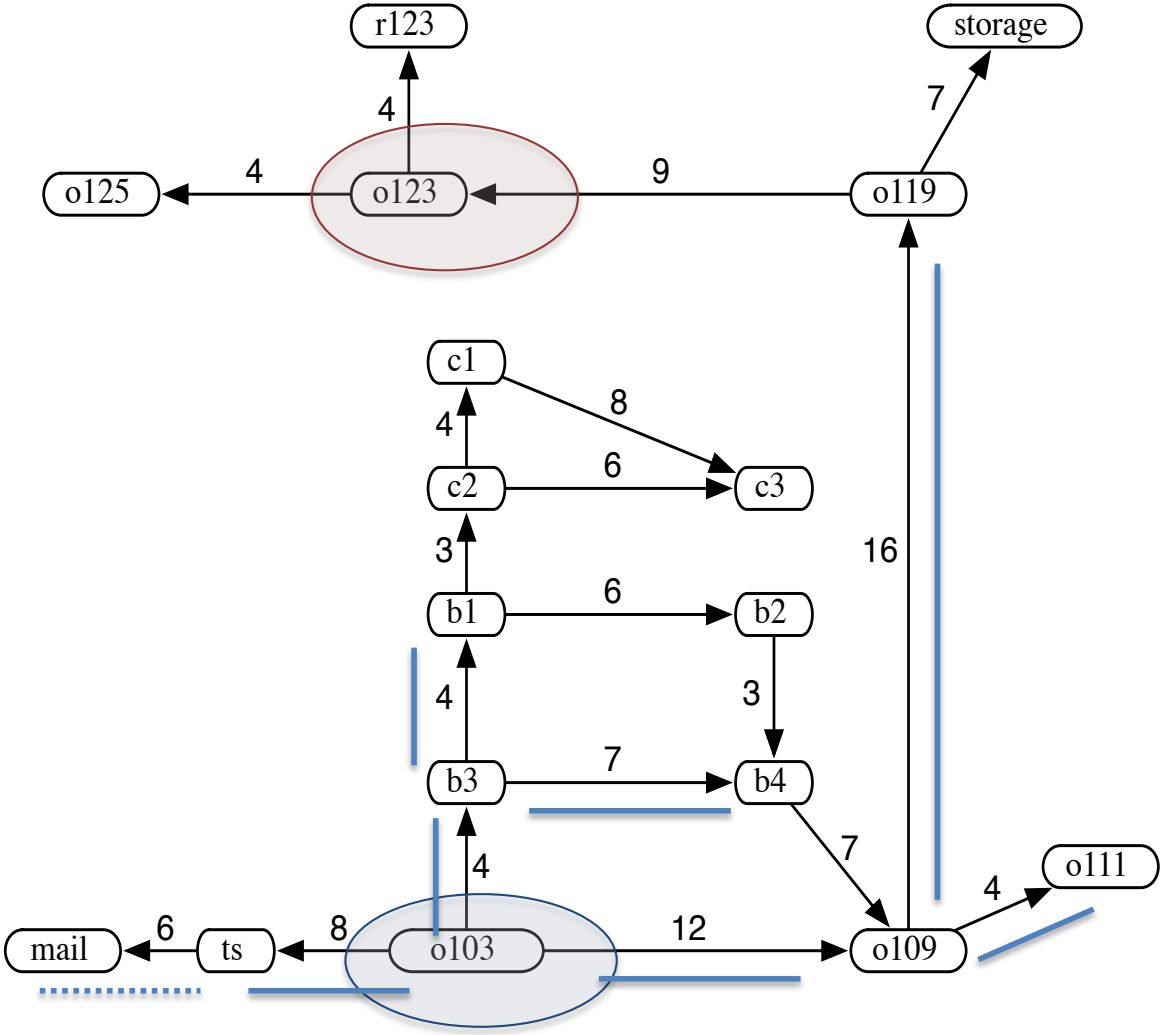
Breadth-First Search



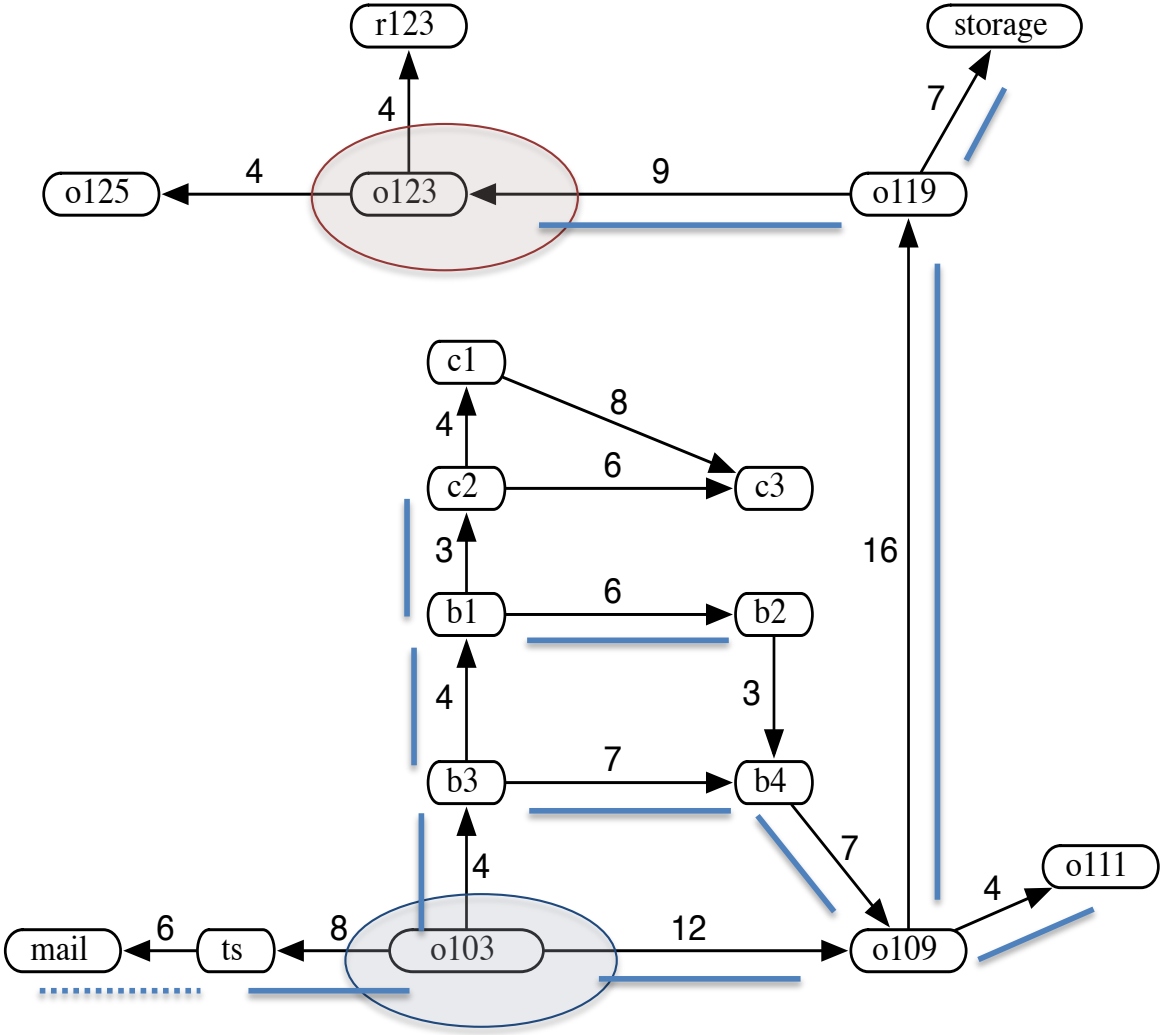
$[\langle o103, b3, b1, c2\rangle, \langle o103, b3, b1, b2\rangle, \langle o103, b3, b4, o109\rangle, \langle o103, o109, o119, storage\rangle, \langle o103, o109, o119, o123\rangle].$

Note how in breadth-first search each path on the frontier has either the same number of arcs or one more arc than the next element of the frontier that will be selected.

State-Space Graph for the Delivery Robot



State-Space Graph for the Delivery Robot



Complexity of Breadth-first Search

- Does breadth-first search guarantee to find the path with fewest arcs?

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?

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?

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?

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?

Properties of breadth-first search

- Complete? Yes (if *b* is finite the shallowest goal is at a fixed depth *d* and will be found before any deeper nodes are generated)

- Time? $1 + b + b^2 + b^3 + \ldots + b^d = \frac{b^{d+1}-1}{b-1} = O(b^d)$

- 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.

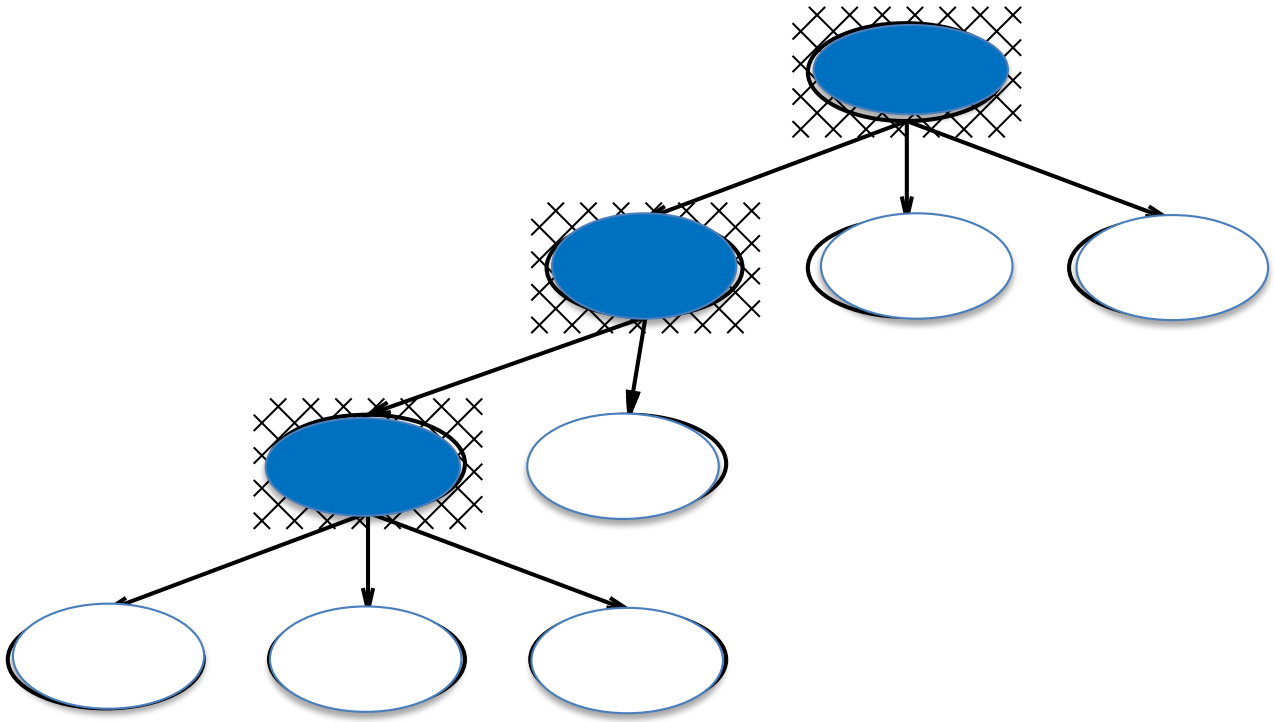
Depth First Search

- 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

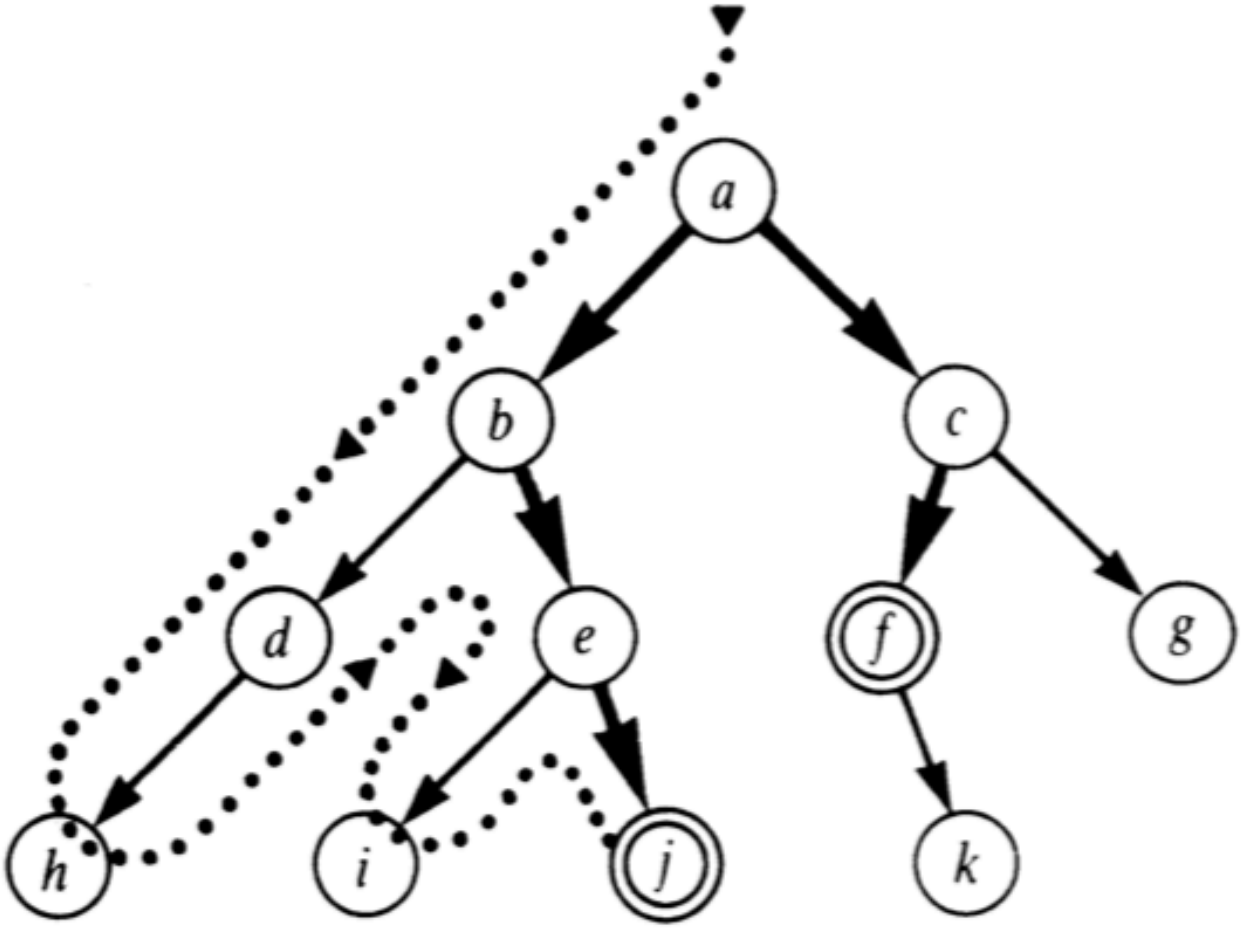
together with any remaining unexpanded siblings of nodes along path

- Stop when node with goal state is expanded
- Include check that generated state has not already been explored along a path – cycle checking

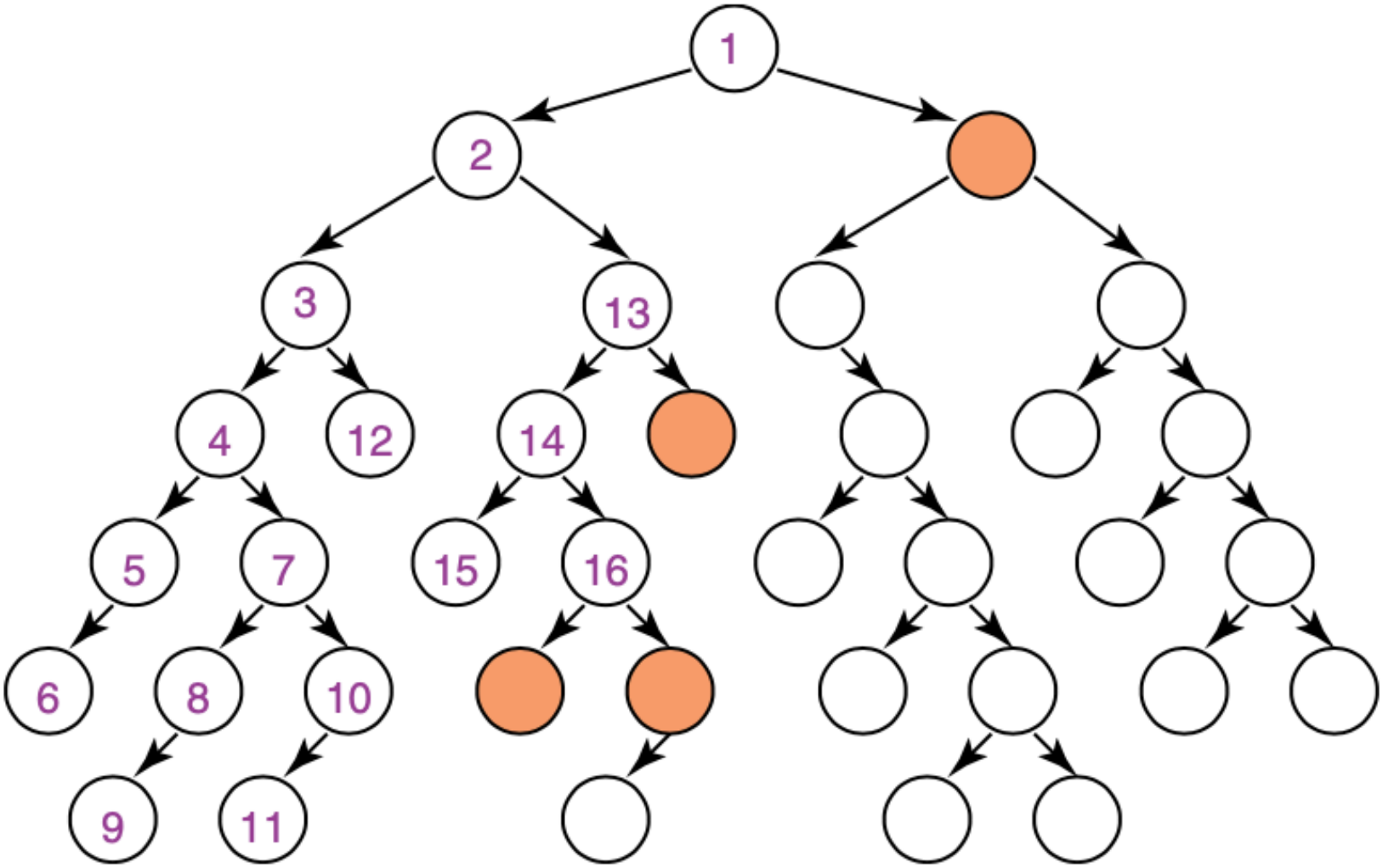
Depth First Search



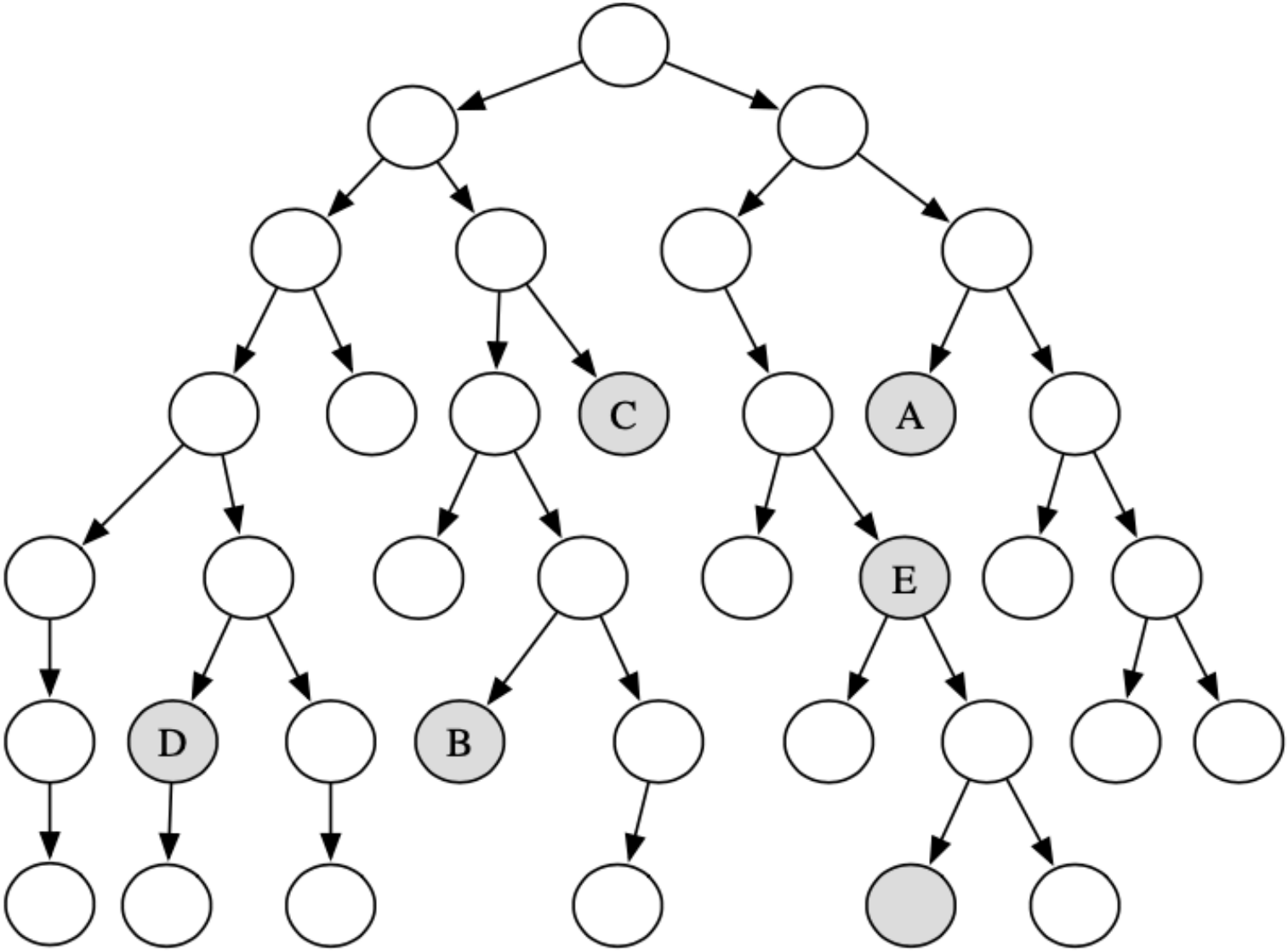
Depth-first Search - DFS



Illustrative Graph — Depth-first Search



Which shaded goal will depth-first search find first?



Properties of depth-first search

- Complete? No! fails in infinite-depth spaces, spaces with loops
  - Modify to avoid repeated states along path → complete in finite spaces
- Time? $O(b^m)$, m = maximum depth of search tree
  - terrible if m is much larger than d
  - but if solutions are dense, may be much faster than breadth-first
- 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.

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

  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

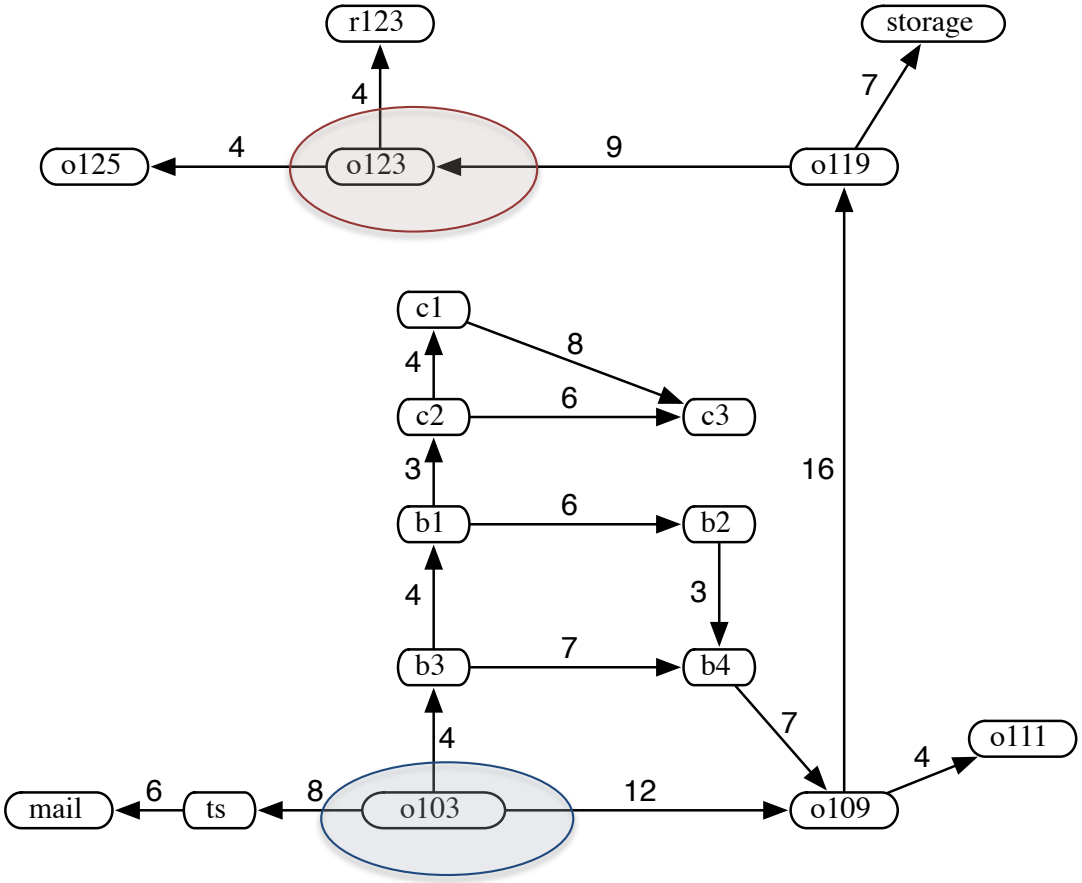
- The simplest search method that is guaranteed to find a minimum cost path is **lowest-cost-first search** or Uniform-Cost Search
  - is similar to breadth-first search, but instead of expanding a path with the fewest number of arcs, it selects a path with the lowest cost.
  - is implemented by treating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

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

Uniform-Cost Search

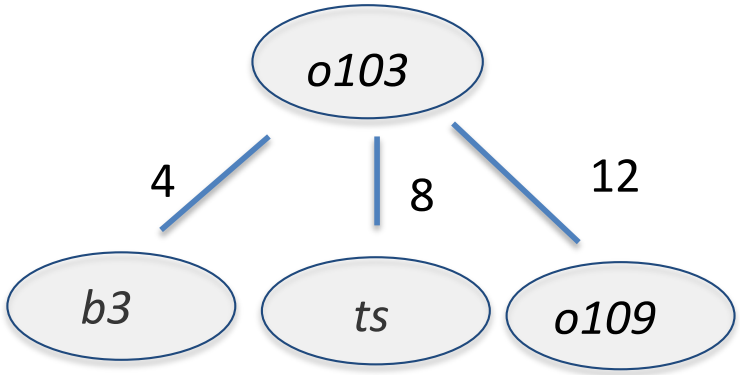
- Expand root first, then expand least-cost unexpanded node
- Implementation: priority queue = insert nodes in order of increasing path cost - (*lowest path cost* g(n)).
- Reduces to Breadth First Search when all actions have same cost
- Finds the cheapest goal provided path cost is monotonically increasing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

State-Space Graph for the Delivery Robot



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

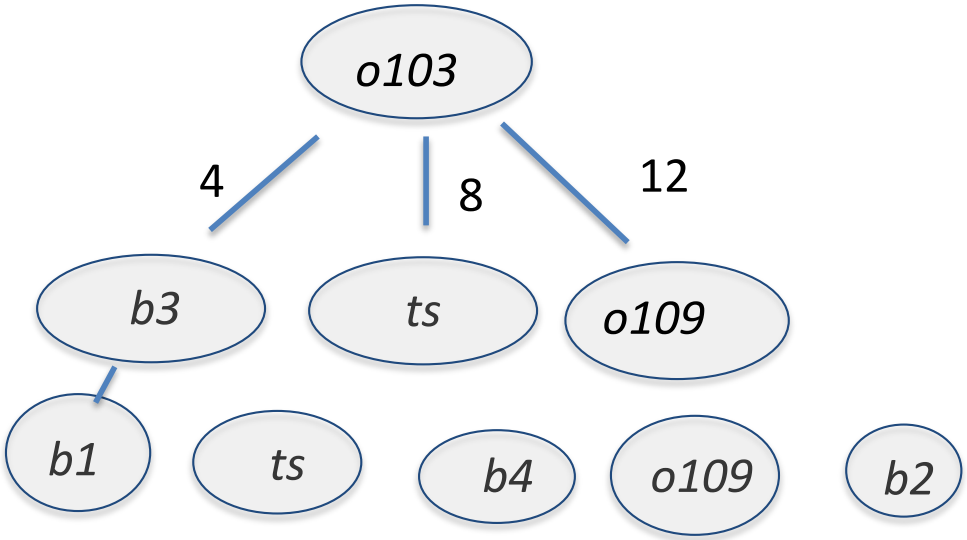
Uniform-Cost Search



$[o103_0]$

$[b3_4, ts_8, o109_{12}]$

Uniform-Cost Search



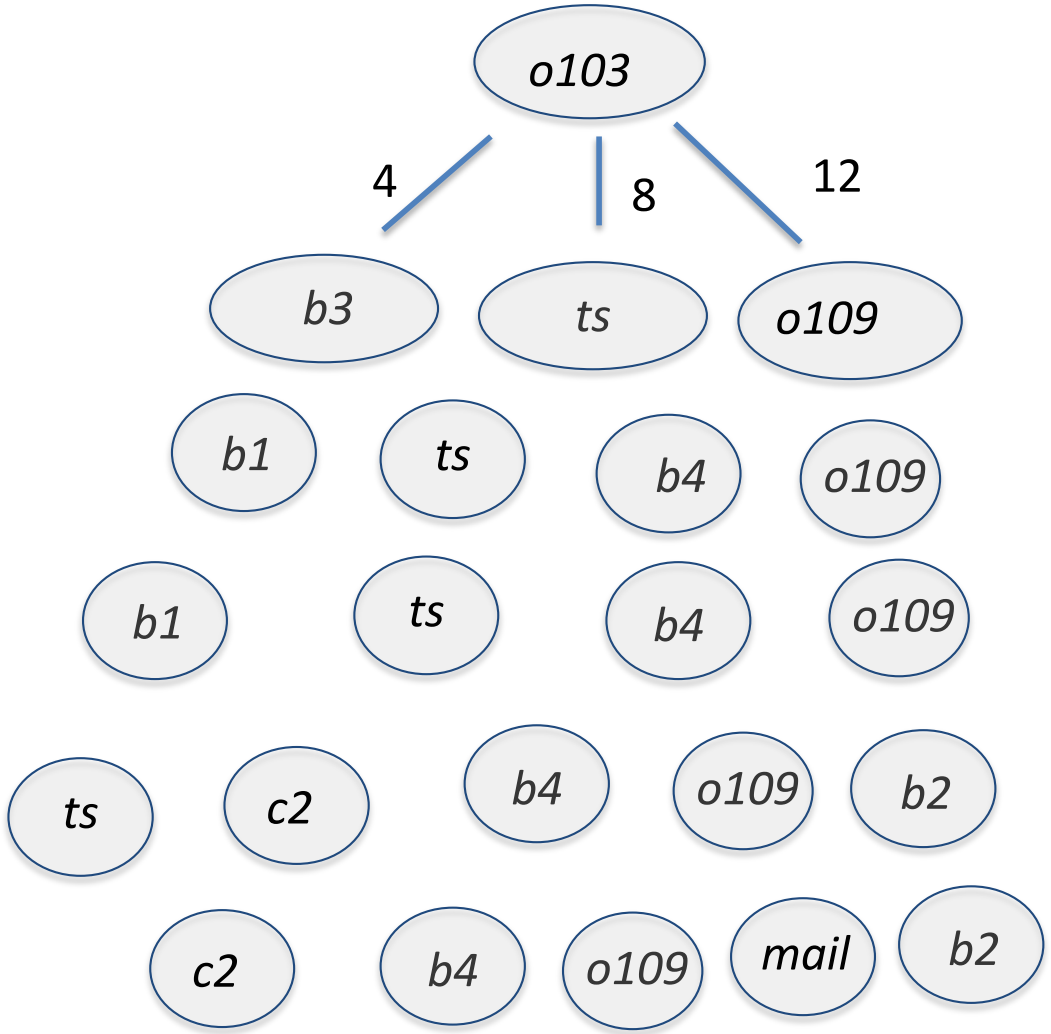
$[o103_0]$

$[b3_4, ts_8, o109_{12}]$

$[b1_8, ts_8, b4_{11}, o109_{12}]$

$[ts_8, c2_{11}, b4_{11}, o109_{12}, b2_{14}]$

Uniform-Cost Search



$[o103_0]$

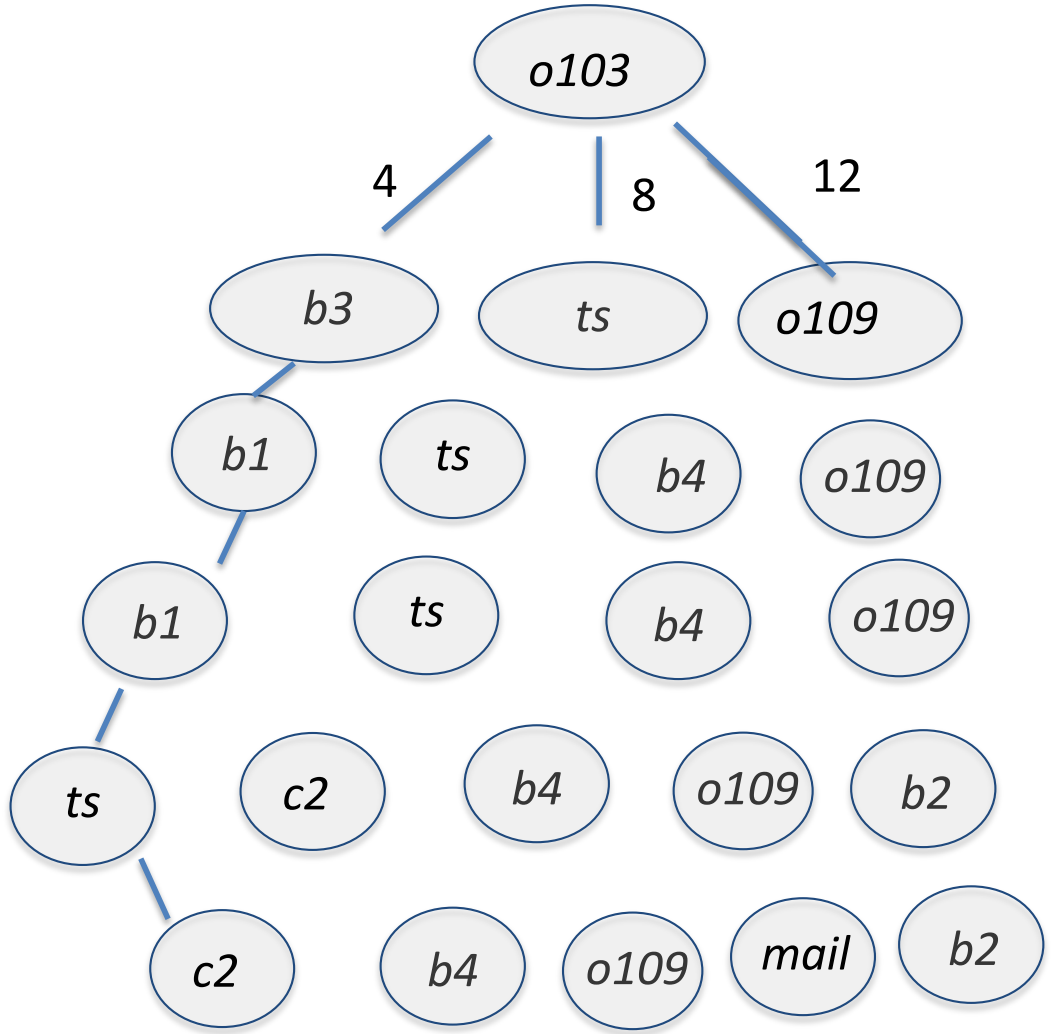
$[b3_4, ts_8, o109_{12}]$

$[b1_8, ts_8, b4_{11}, o109_{12}]$

$[ts_8, c2_{11}, b4_{11}, o109_{12}, b2_{14}]$

$[c2_{11}, b4_{11}, o109_{12}, mail_{14}, b2_{14}]$

Uniform-Cost Search



$[o103_0]$

$[b3_4, ts_8, o109_{12}]$

$[b1_8, ts_8, b4_{11}, o109_{12}]$

$[ts_8, c2_{11}, b4_{11}, o109_{12}, b2_{14}]$

$[c2_{11}, b4_{11}, o109_{12}, mail_{14}, b2_{14}]$

Uniform-Cost Search

- The bounded arc cost is used to guarantee the lowest-cost search will find a solution, when one exists, in graphs with finite branching factor.

Properties of Uniform-Cost Search

- Complete? Yes, if *b* is finite and if step cost ≥ ε with ε > 0
- Time? $O(b^{\lceil C^*/\varepsilon \rceil})$ where C^* = cost of the optimal solution and assume every action costs at least ε
- Space? $O(b^{\lceil C^*/\varepsilon \rceil})$ ($b^{\lceil C^*/\varepsilon \rceil} = b^d$ if all step costs are equal
- Optimal? Yes – nodes expanded in increasing order of *g(n)*

Summary of Search Strategies

Strategy	Frontier Selection	Complete	Halts	Space
Depth-first	Last node added			
Breadth-first	First node added			
Lowest-cost-first	Minimal $cost(p)$			

Complete — guaranteed to find a solution if there is one (for graphs with finite number of neighbours, even on infinite graphs)

Halts — on finite graph (perhaps with cycles).

Space — as a function of the length of current path

Summary of Search Strategies

Strategy	Frontier Selection	Complete	Halts	Space
Depth-first	Last node added	No	No	Linear
Breadth-first	First node added	Yes	No	Exp
Lowest-cost-first	Minimal $cost(p)$	Yes	No	Exp

Complete — guaranteed to find a solution if there is one (for graphs with finite number of neighbours, even on infinite graphs)

Halts — on finite graph (perhaps with cycles).

Space — as a function of the length of current path

Depth Limited Search

Expands nodes like Depth First Search but imposes a cutoff on the maximum depth of path.

- Complete? Yes (no infinite loops anymore)
- Time? $O(b^k)$ 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 ?

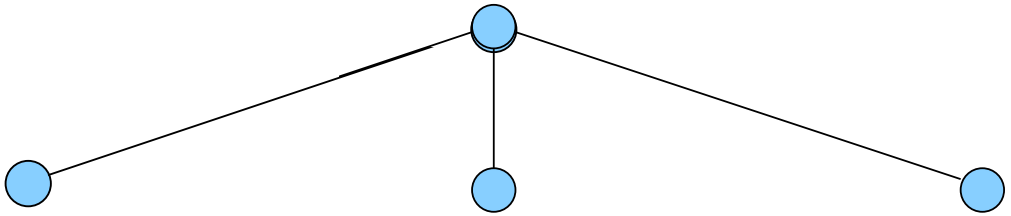
Iterative Deepening Search

- 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.

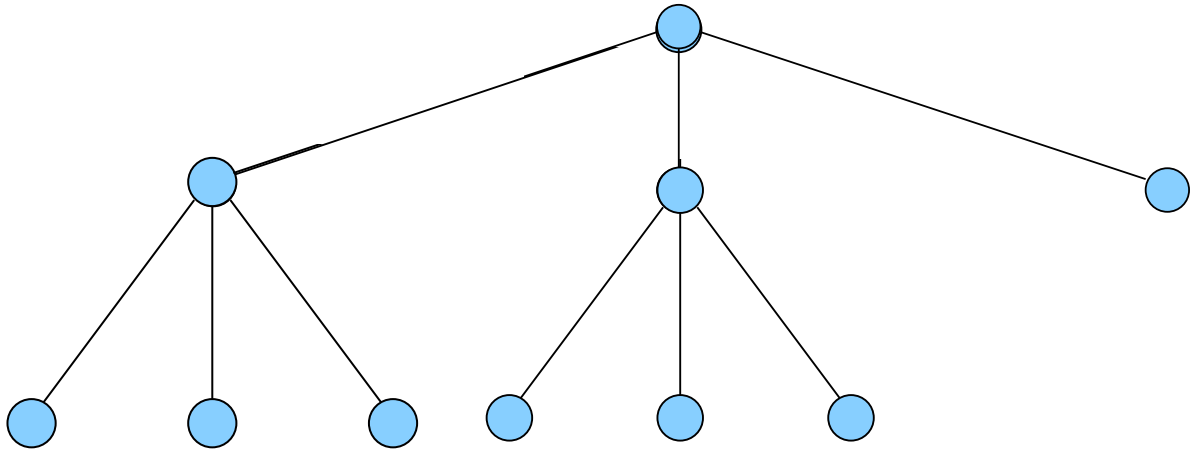
- Early states will be expanded multiple times, but that might not matter too much because most of the nodes are near the leaves.

Iterative Deepening Search

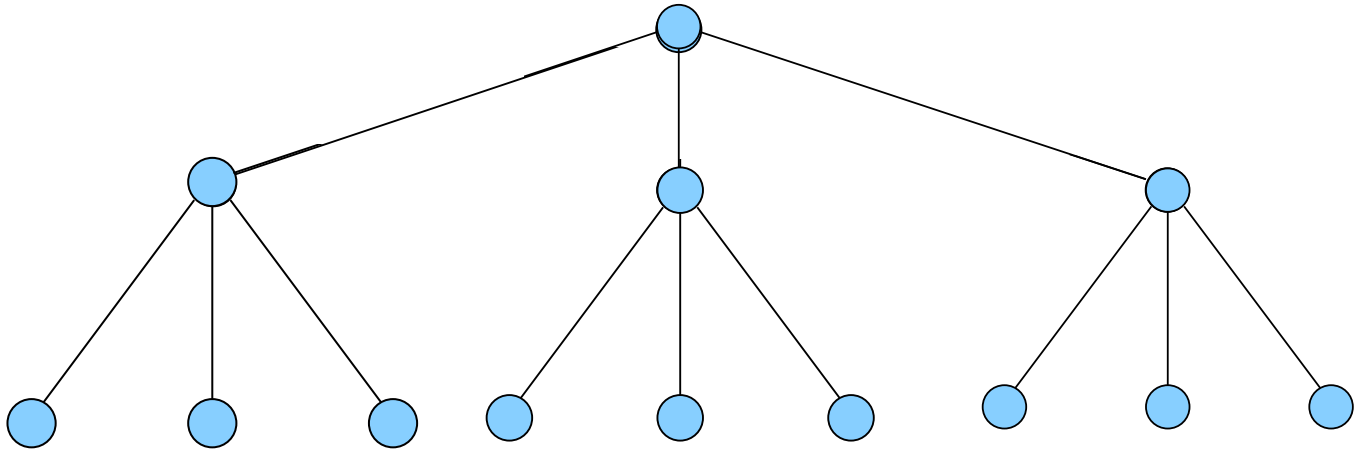
Iterative Deepening Search



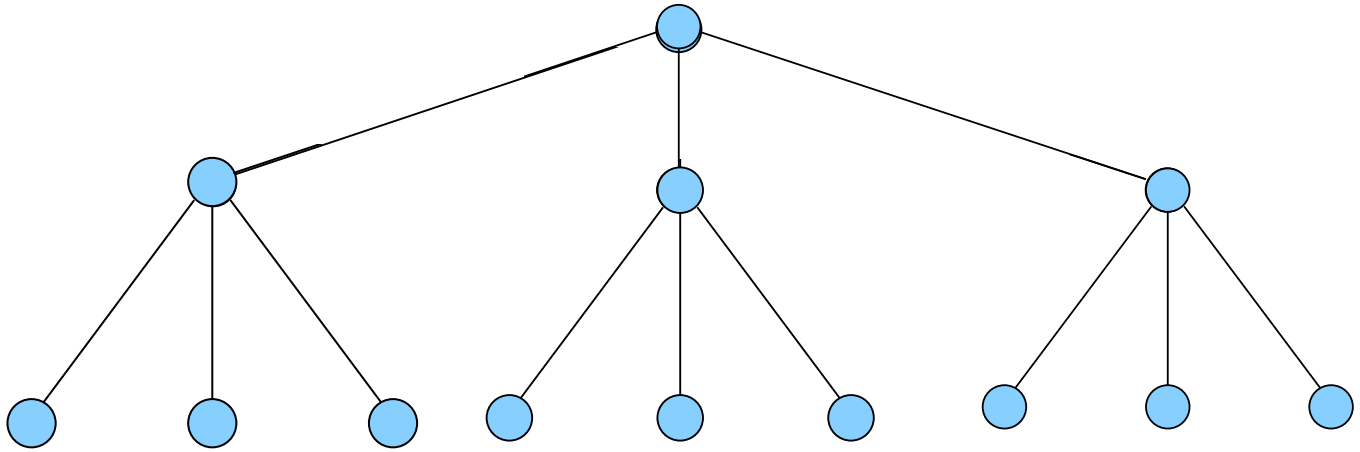
Iterative Deepening Search



Iterative Deepening Search

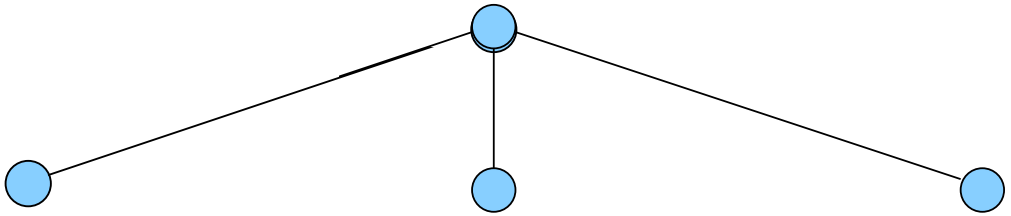


Iterative Deepening Search

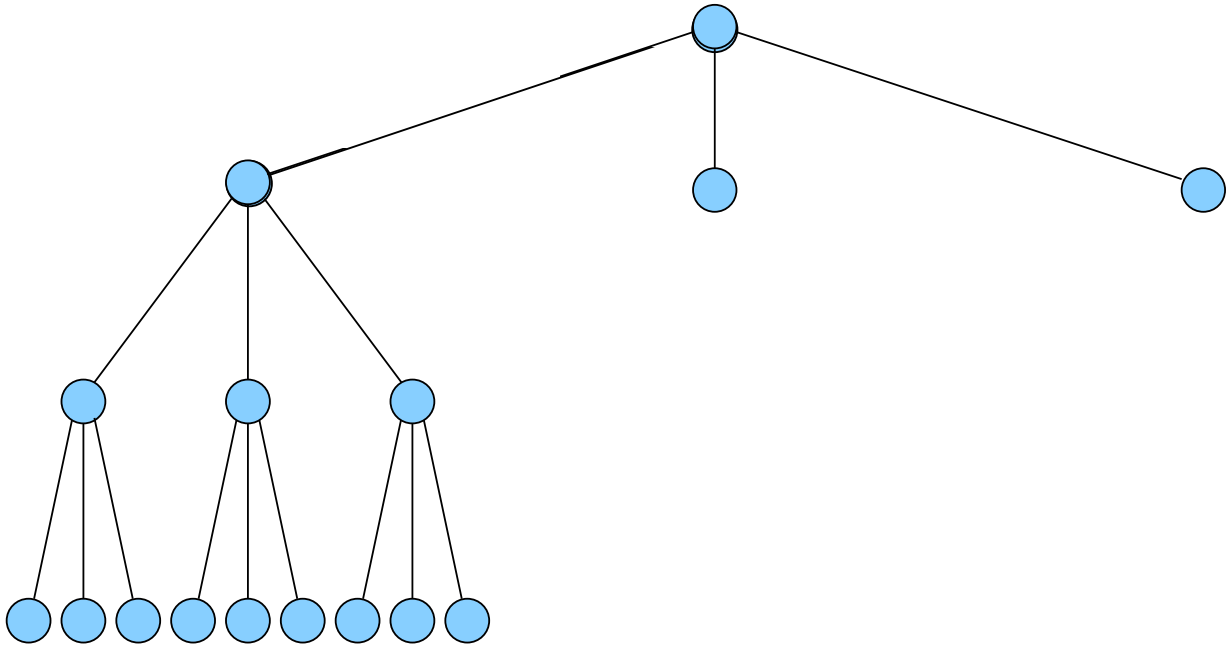


Iterative Deepening Search

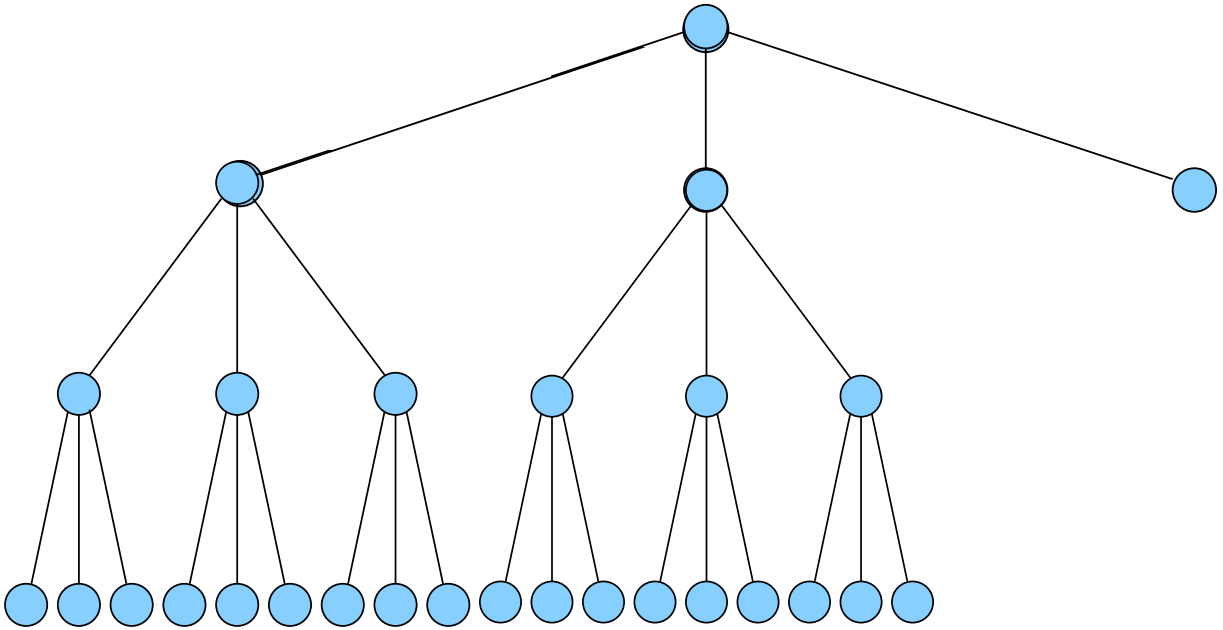
Iterative Deepening Search



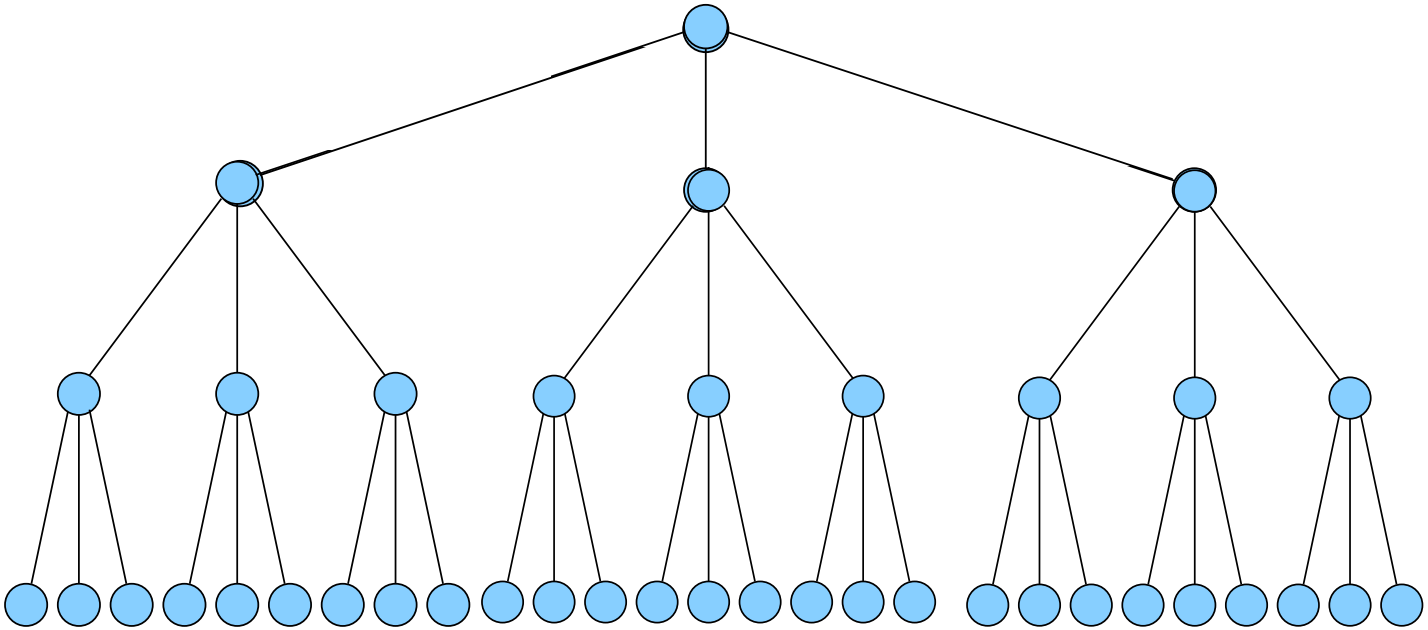
Iterative Deepening Search



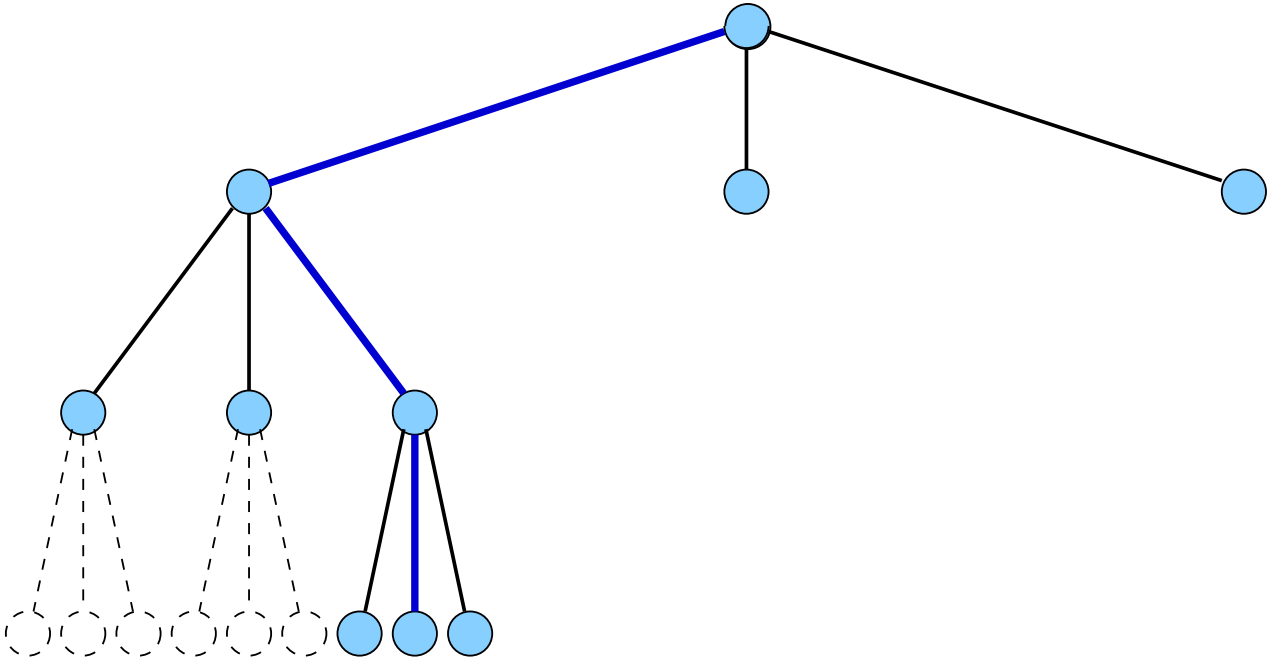
Iterative Deepening Search



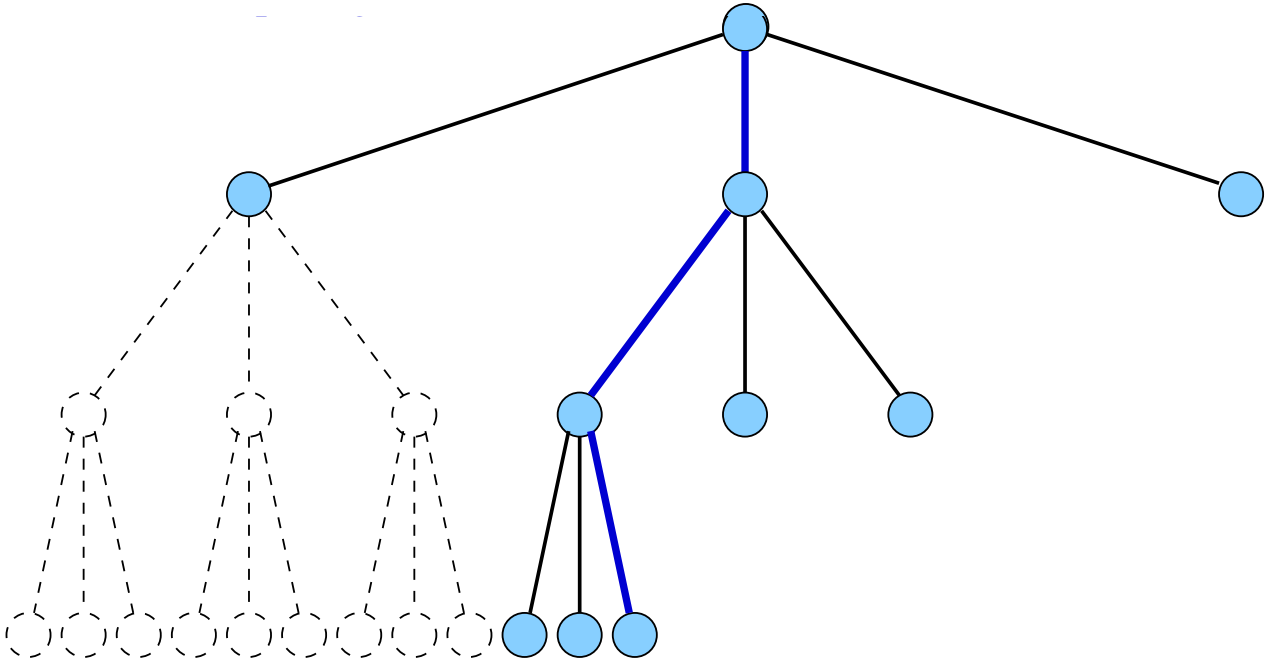
Iterative Deepening Search



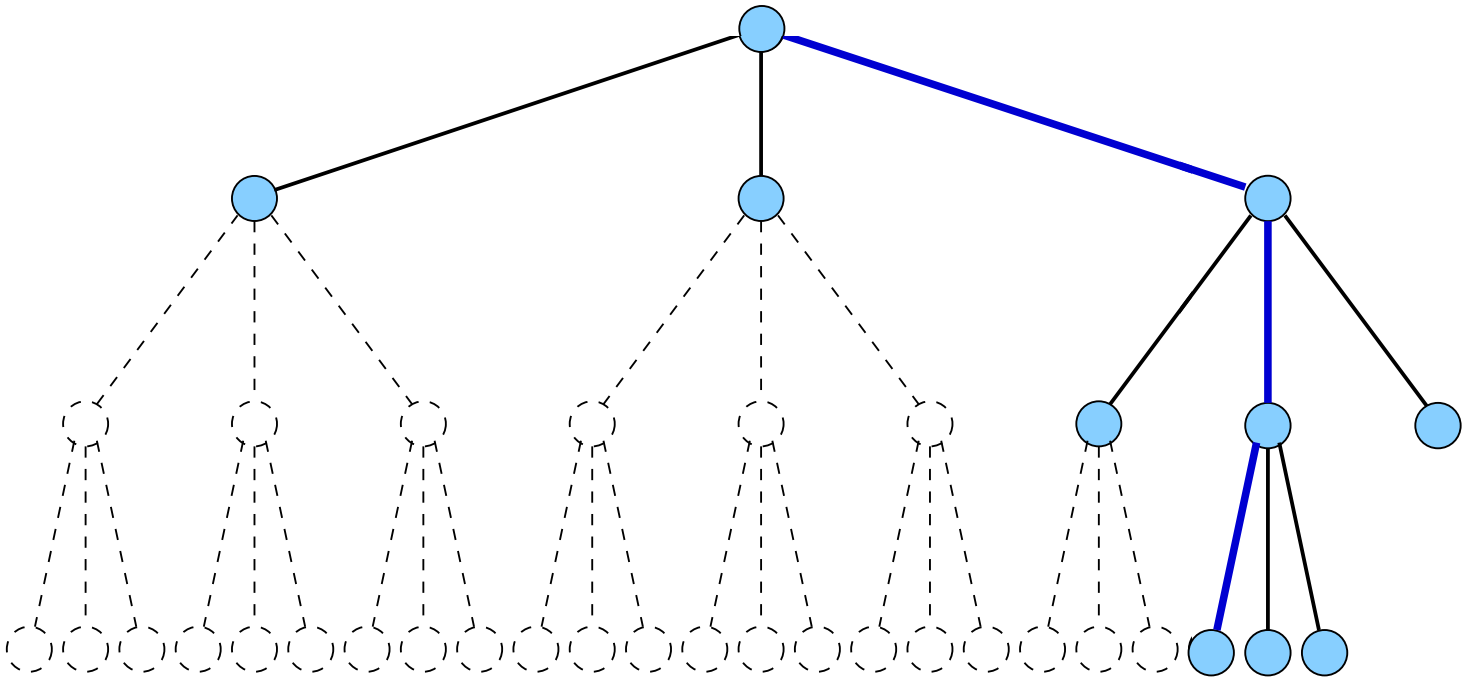
Iterative Deepening Search



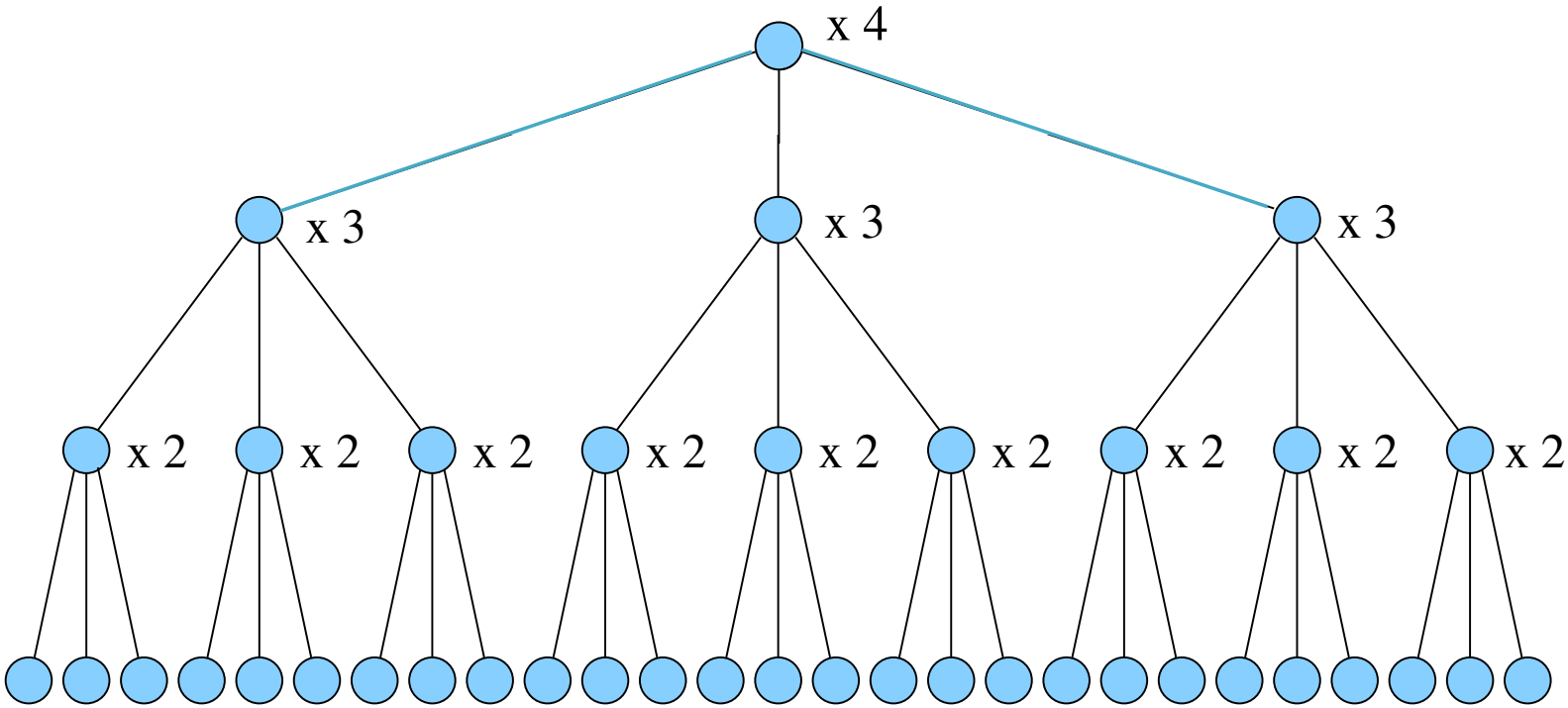
Iterative Deepening Search



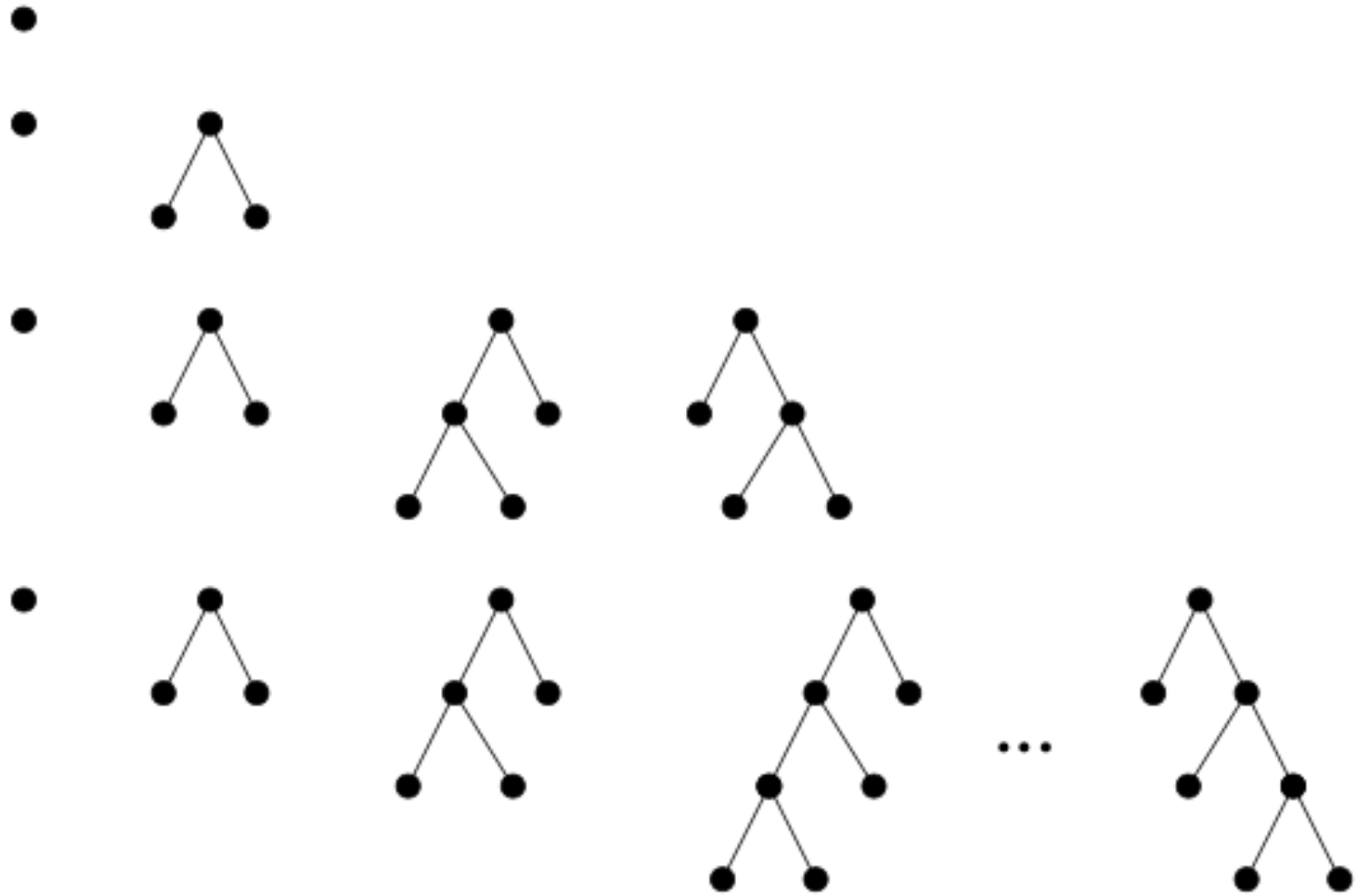
Iterative Deepening Search



Iterative Deepening Search



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

Properties of Iterative Deepening Search

- Complete? Yes.

- Time: nodes at the bottom level are expanded once, nodes at the next level twice, and so on:
  - depth-limited: $1 + b + b^2 + b^3 + \ldots + b^d = \frac{b^{d+1}-1}{b-1} = O(b^d)$
  - Iterative deepening:

$$(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + 2 \cdot b^{d-1} + 1 \cdot b^d = O(b^d)$$

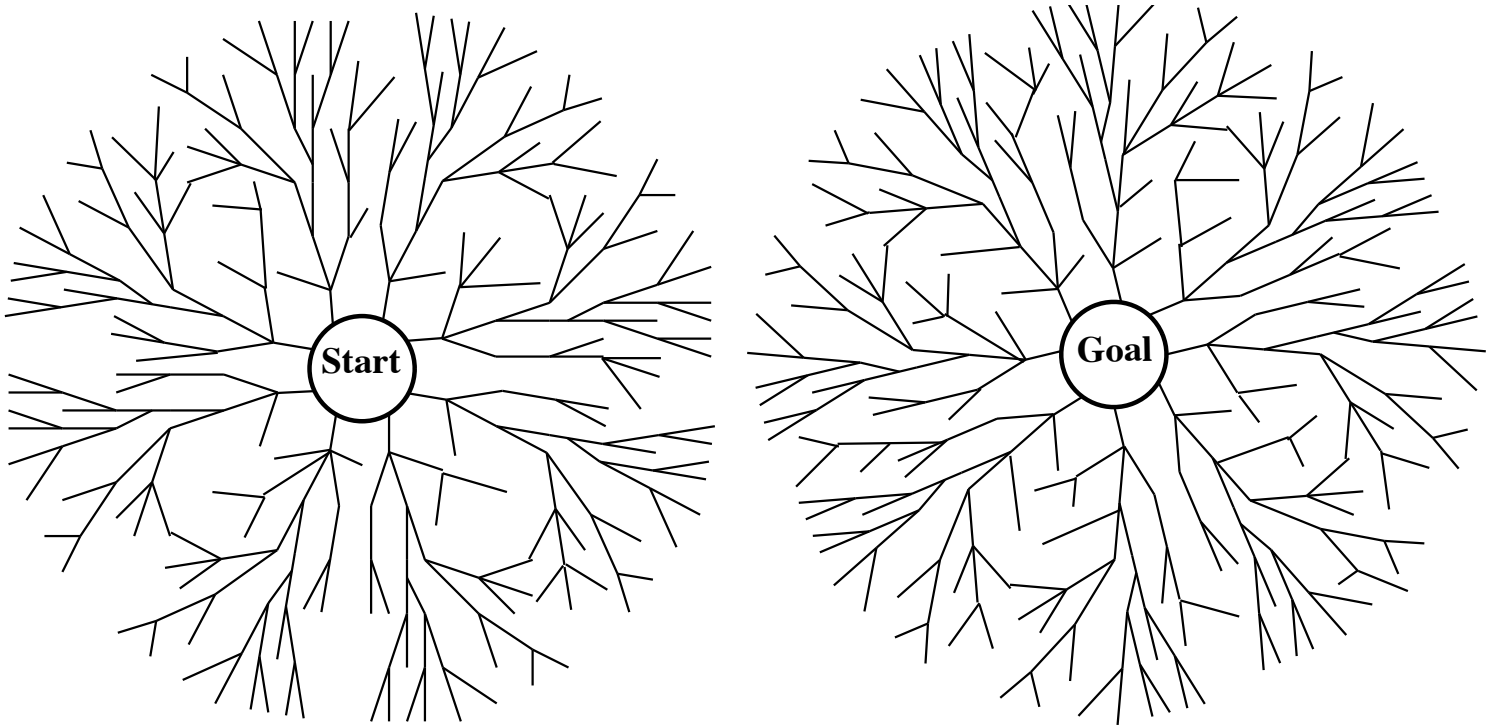
  - Example b=10, d=5:
    - depth-limited: 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111
    - iterative-deepening: 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456
    - only about 11% more nodes (for b = 10).

Properties of Iterative Deepening Search

- Complete? Yes.
- Time: $O(b^d)$
- 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.

Bidirectional Search — Analysis

- If solution exists at depth *d* then bidirectional search requires time

$$O(2b^{\frac{d}{2}}) = O(b^{\frac{d}{2}})$$

(assuming constant time checking of intersection)

- To check for intersection must have all states from one of the searches in memory, therefore space complexity is $O(b^{\frac{d}{2}})$

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.

Complexity Results for Uninformed Search

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Time	$O(b^d)$	$O(b^{\lceil C^*/\varepsilon \rceil})$	$O(b^m)$	$O(b^k)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{\lceil C^*/\varepsilon \rceil})$	$O(bm)$	$O(bk)$	$O(bd)$
Complete?	Yes[1]	Yes[2]	No	No	Yes[1]
Optimal ?	Yes[3]	Yes	No	No	Yes[3]

b = branching factor, d = depth of the shallowest solution,
m = maximum depth of the search tree, k = depth limit.

1 = complete if b is finite.

2 = complete if b is finite and step costs $\geq \varepsilon$ with $\varepsilon > 0$.

3 = optimal if actions all have the same cost.