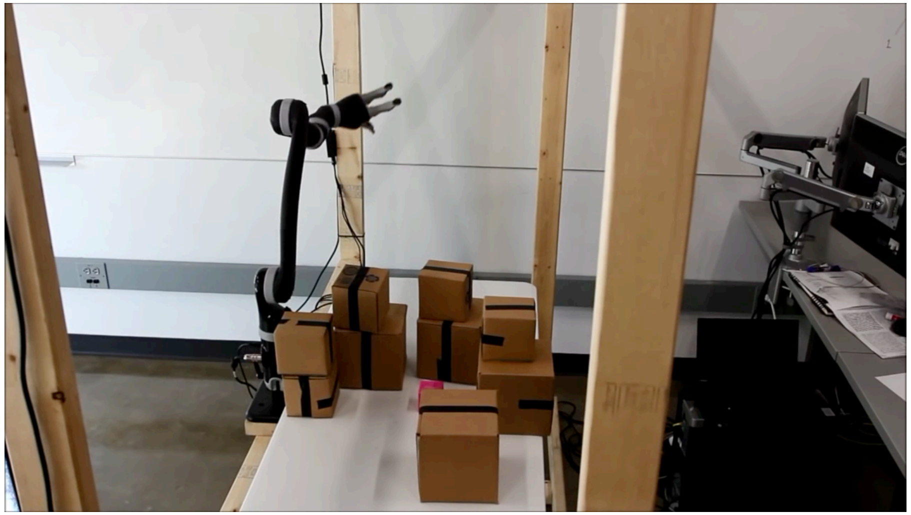
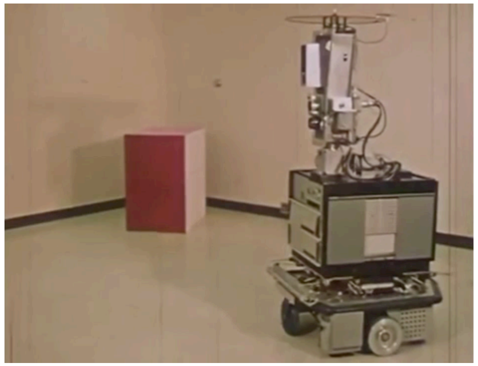
Uninformed Search

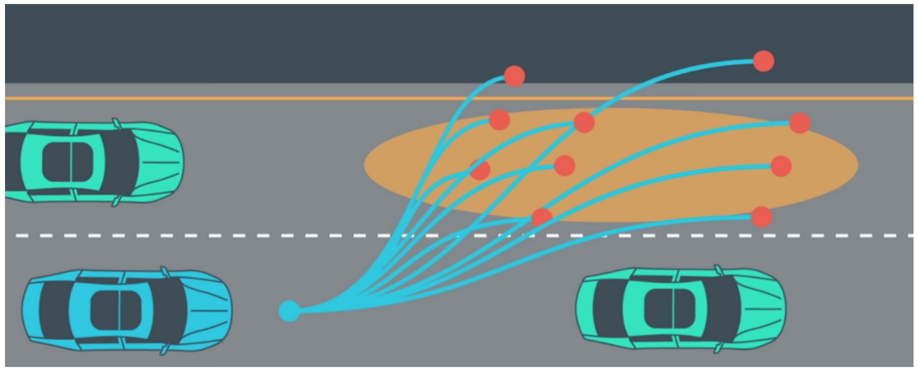
COMP3411/9814: Artificial Intelligence

When is Search Needed?

- Motion Planning
- Navigation
- Speech and Natural Language
- Task Planning
- Machine Learning
- Game Playing







Search Methods

- Uninformed search
  - 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
  - use heuristics to improve efficiency
  - Informed (or “heuristic”) search strategies use task-specific knowledge.

Overview

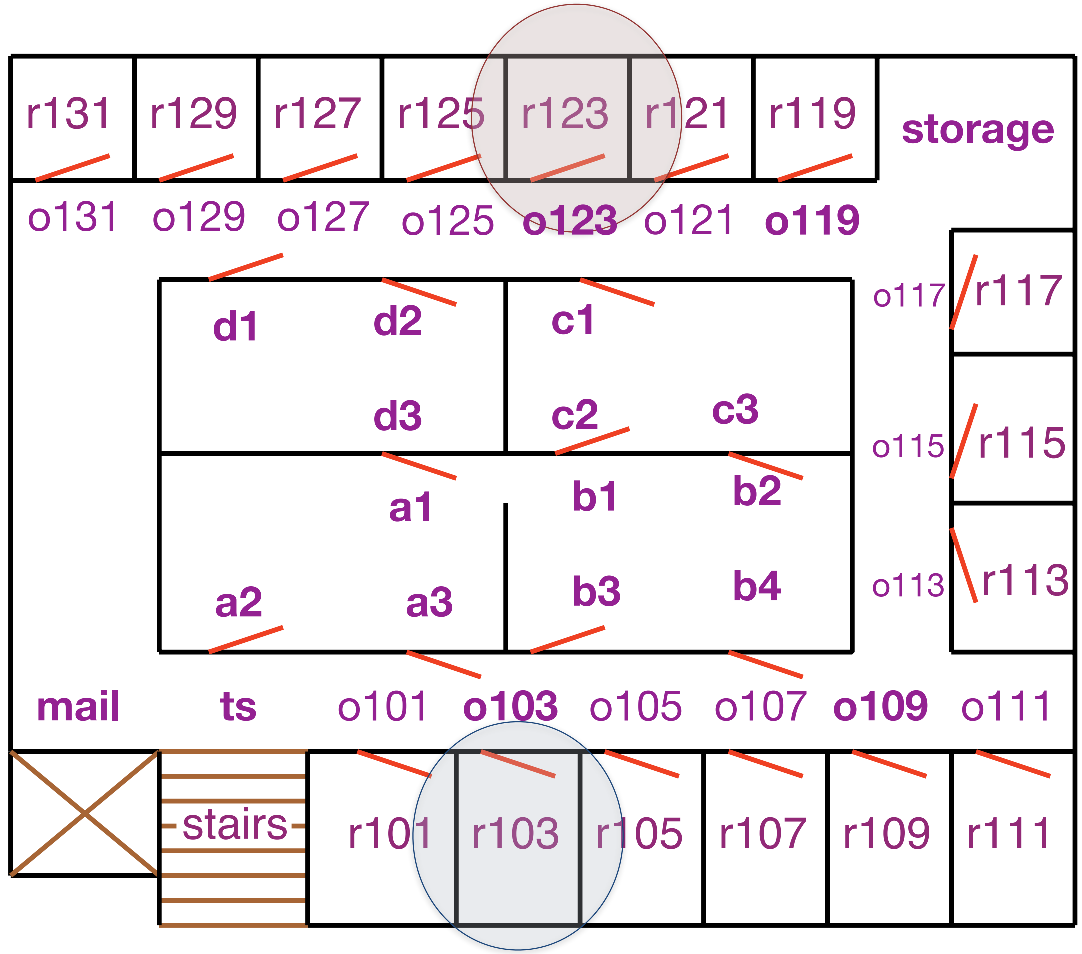
- Basic search algorithms
  - Breadth First Search
  - Depth First Search
  - Uniform Cost Search
  - Depth Limited Search
  - Iterative Deepening Search
  - Bidirectional Search

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 (applies to optimisation problems)

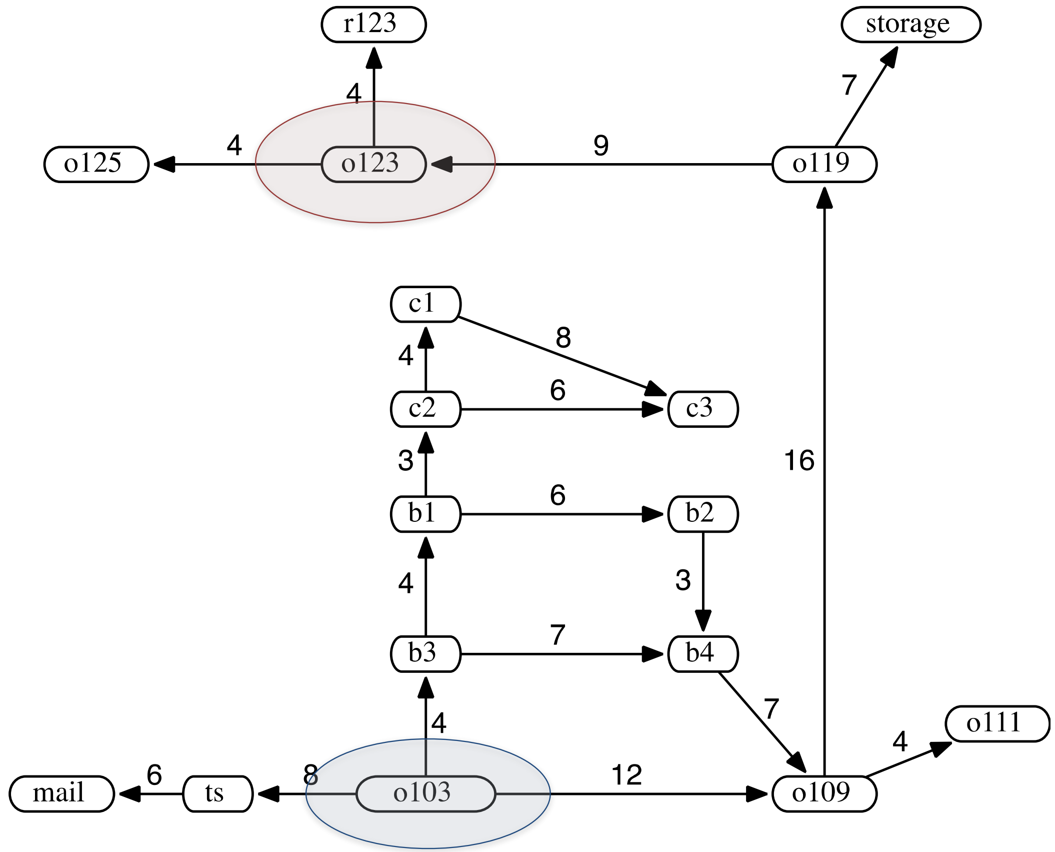
Delivery Robot

- The robot wants to get from outside room 103 to the inside of room 123.
  - 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 o103 to to r123



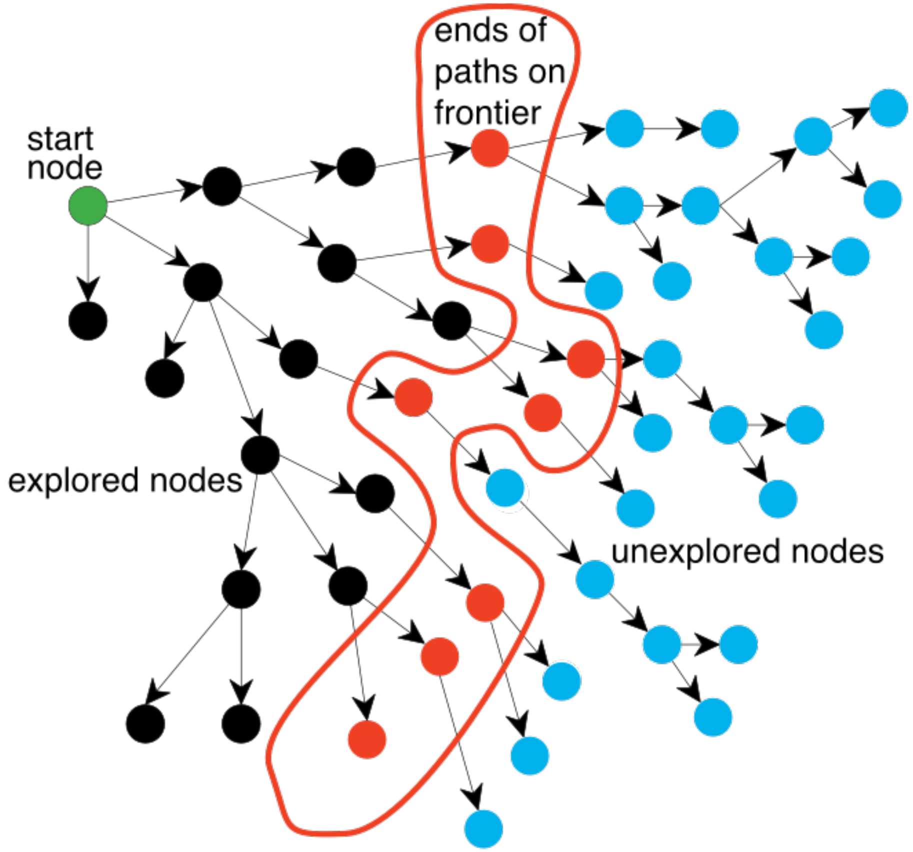
State-Space Graph for Delivery Robot

- Modelled as a state-space search problem
- States are locations.



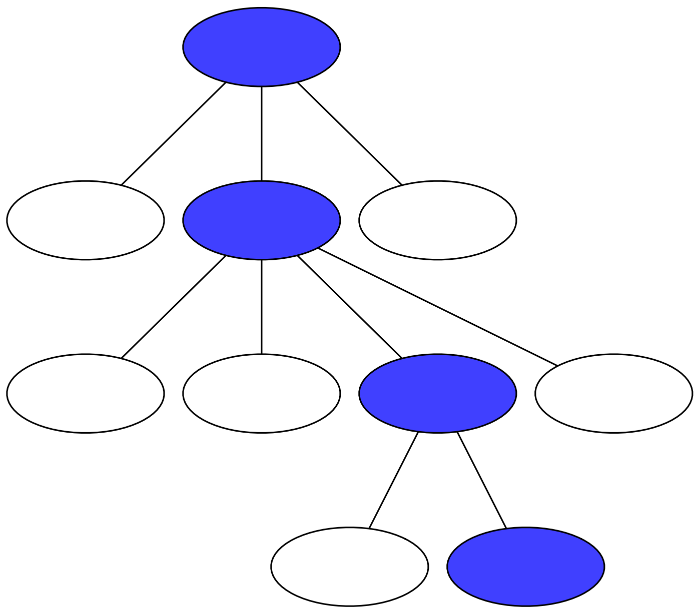
Problem Solving by Graph Searching

Search strategy differ in the way they expand the frontier

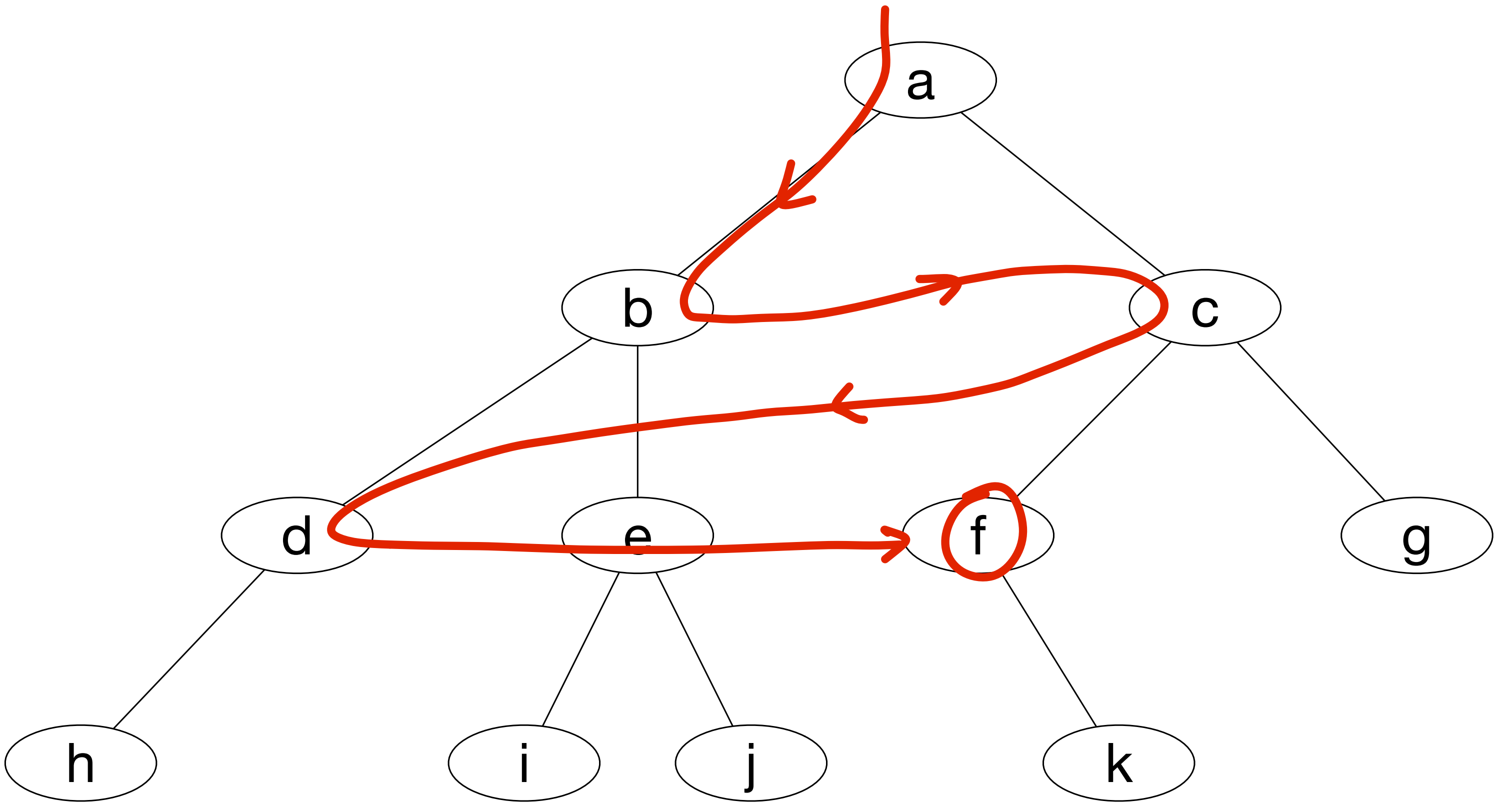


Search Tree

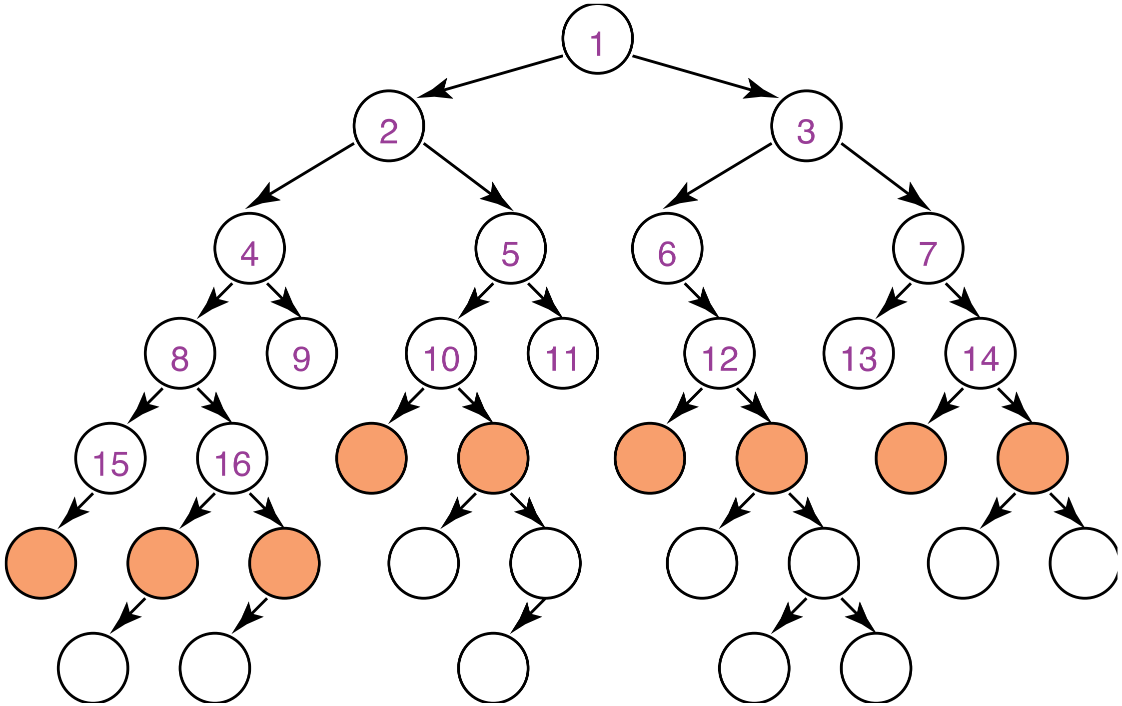
- Search tree: superimposed over the state space.
- Root: search node corresponding to the initial state.
- Leaf nodes: correspond to states that have no successors in the tree because they were not expanded or generated no new nodes.



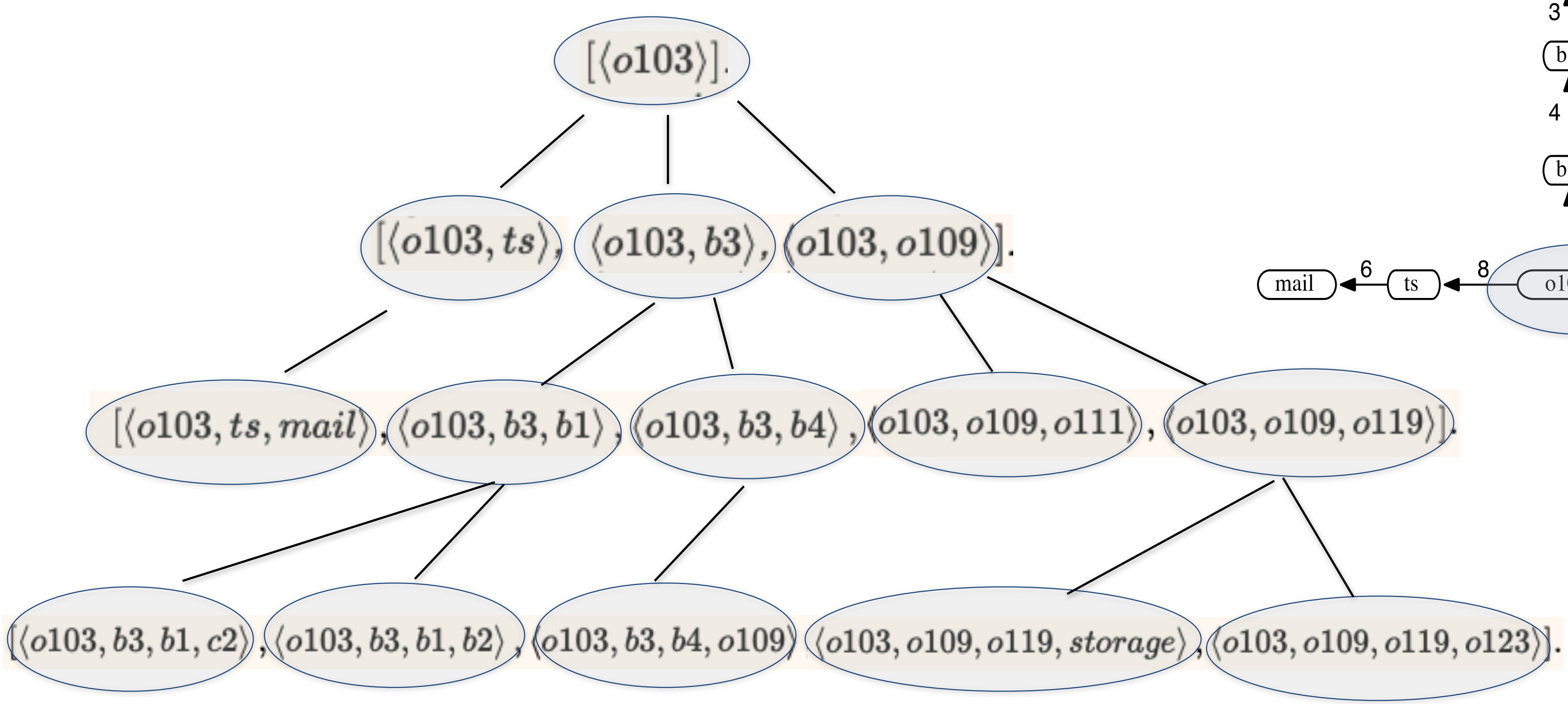
Breadth-First Search

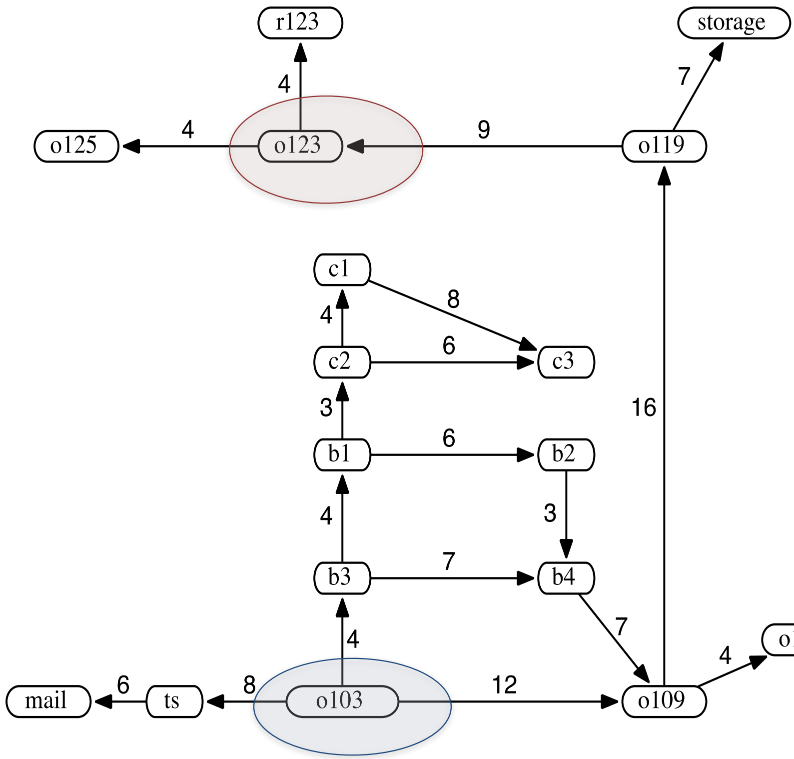


Breadth-first Search Frontier



Breadth-First Search





After each iteration, each path on the frontier has either the same number of arcs

Breadth-first Search

- Breadth-first search treats the frontier as a queue
- It selects the first element in the queue to explore next
- If the list of paths on the frontier is $[p_1, p_2, \ldots, p_r]$:
    - 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

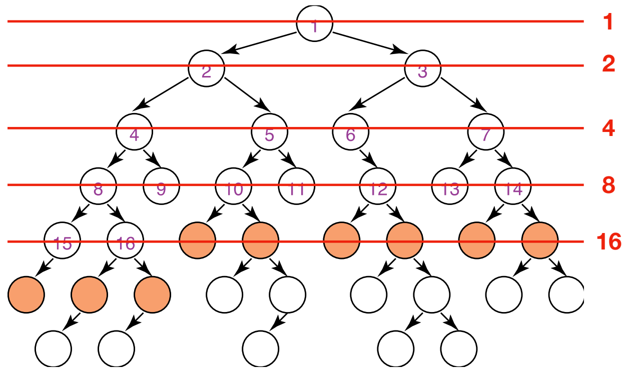
- All nodes are expanded at a same depth in the tree before any nodes at the next level are expanded
- Can be implemented by using a queue to store frontier nodes
  - put newly generated successors at end of queue
- Stop when node with goal state is reached
- Include check that state has not already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

Complexity of Breadth-first Search

- Does breadth-first search guarantee finding the shortest path?
- 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 breadth, b, is finite, the shallowest goal is at a fixed depth, d, and will be found before any deeper nodes are generated)



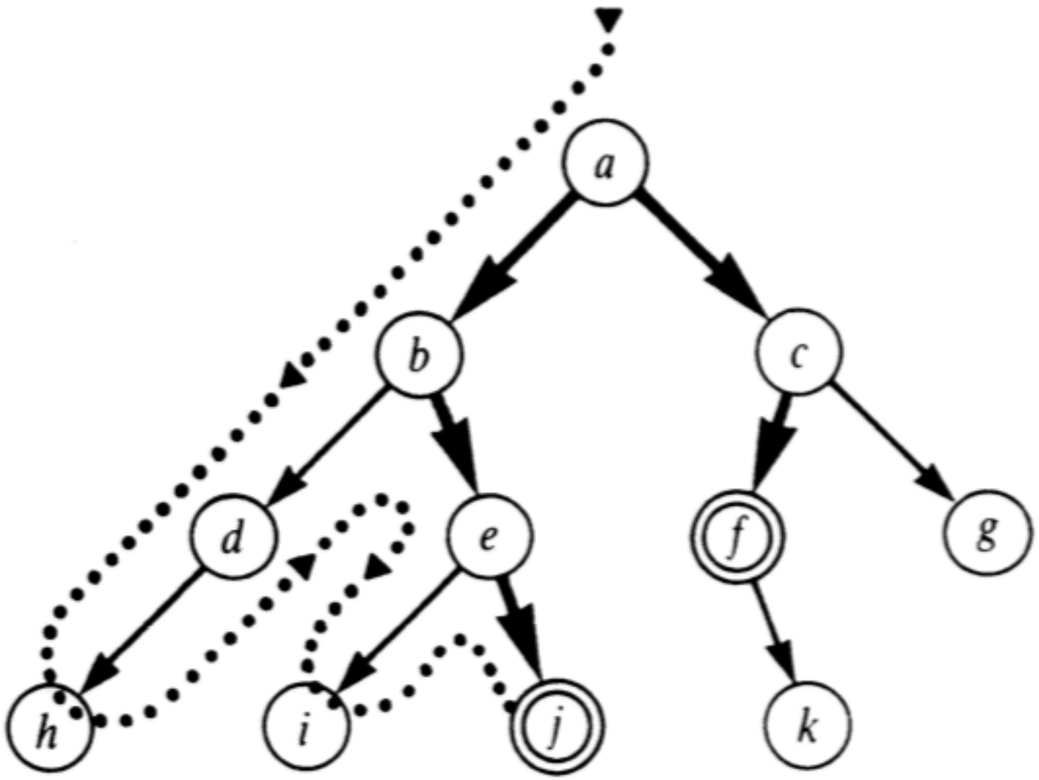
Time? $$1 + b^2 + b^3 + \ldots + b^d = \frac{b^{d+1} - 1}{b - 1} = O(b^d)$$

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 - DFS

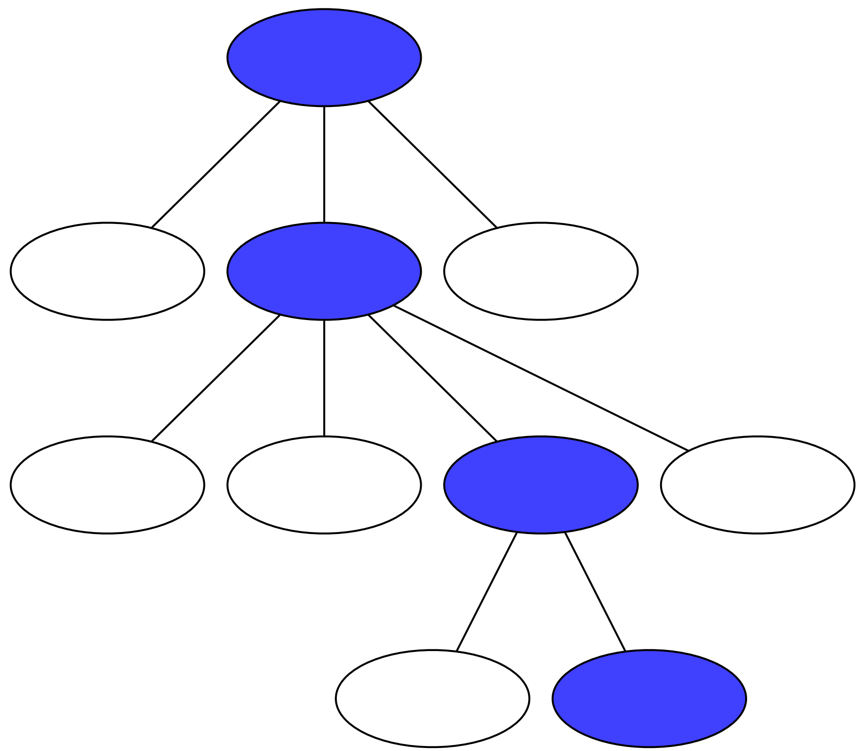


Depth First Search

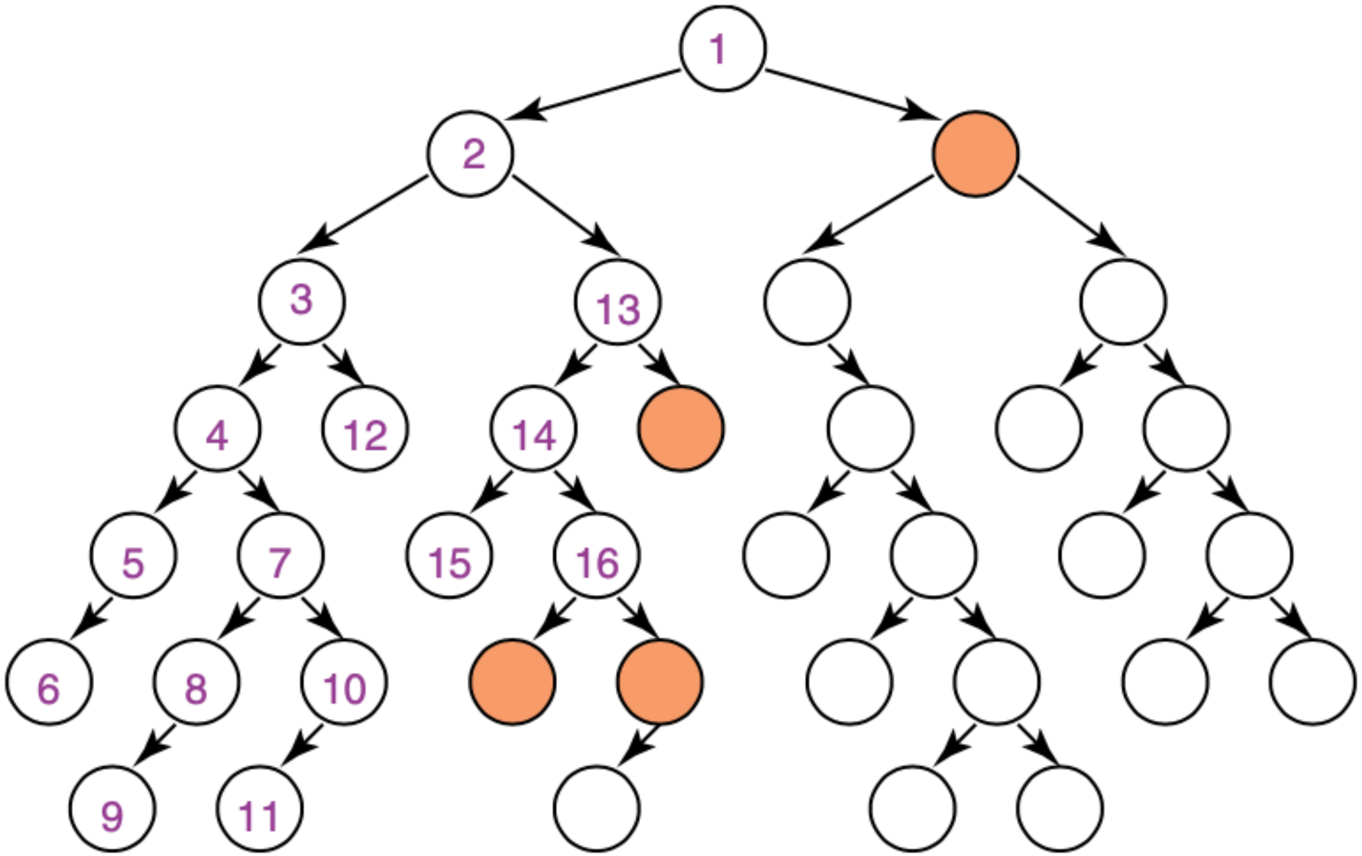
- Expand one node at the deepest level reached so far
- Implementation:
  - Implement the frontier as a stack, i.e. insert newly generated states at the front of the open list (frontier)
  - Can be implemented by recursive function calls, where call stack maintains open list
- In depth-first search, like breadth-first, the order in which the paths are expanded does not depend on the goal.

Depth First Search

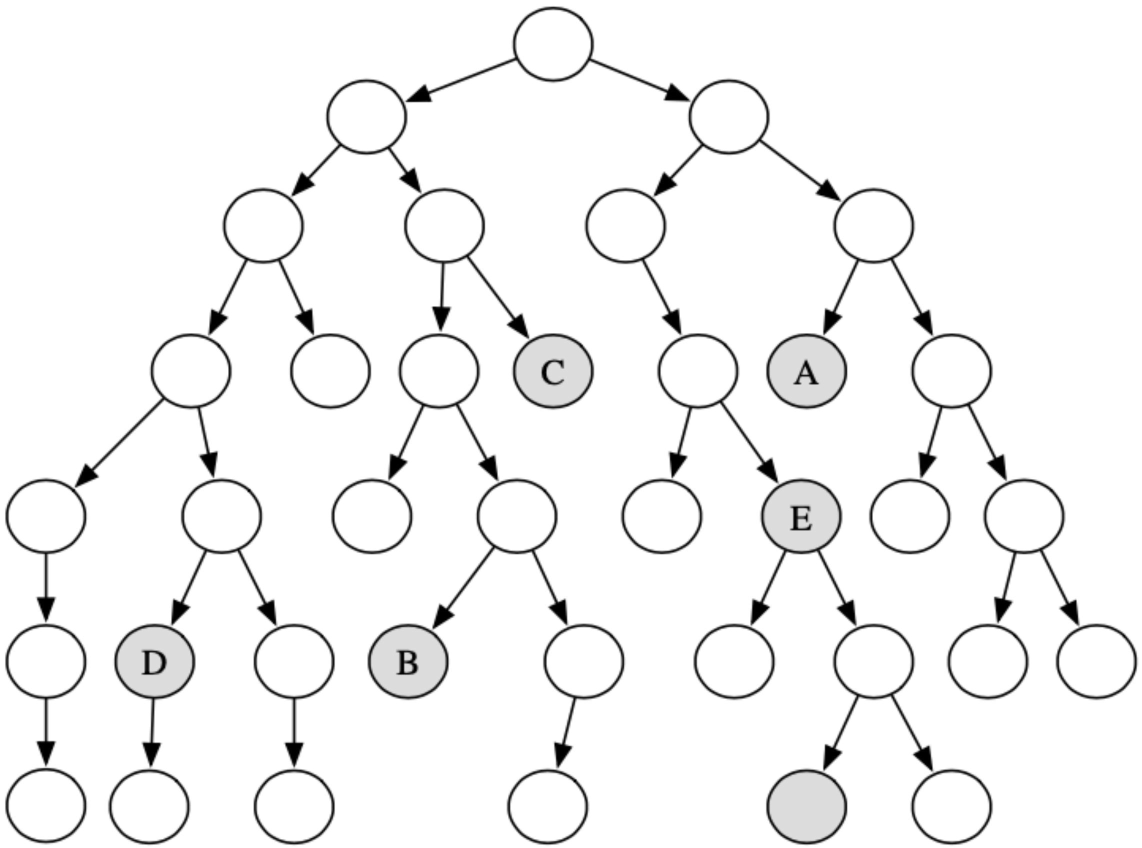
- At any point depth-first search stores single path from root to leaf, together with any remaining unexpanded siblings of nodes along path
- Stop when node with goal state is expanded
- Include check that state has not already been explored along a path – cycle checking



Depth-first Search Example



Which goal (shaded) will depth-first search find first?

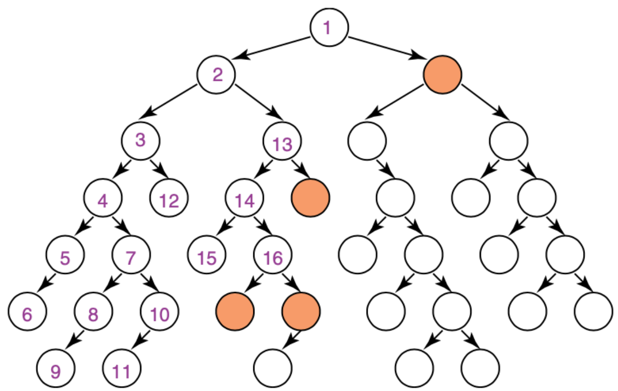


Properties of depth-first search

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 transitions have a cost
- Cost of a path is the sum of the costs of its arcs:

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

- An optimal solution has minimum cost
- **Delivery robot example:**
  - cost of arc may be resources (e.g., time, energy) required to execute action represented by the arc
  - aim is to reach goal using least 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**
  - similar to breadth-first search, but instead of expanding path with least number of arcs, select path with lowest cost
  - implemented by treating the frontier as a priority queue ordered by the cost function

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

Lowest-Cost Search for Delivery Robot

- Edges are labelled with cost
    - e.g. distance to travel
- Sort queue by increasing cost of path to the node

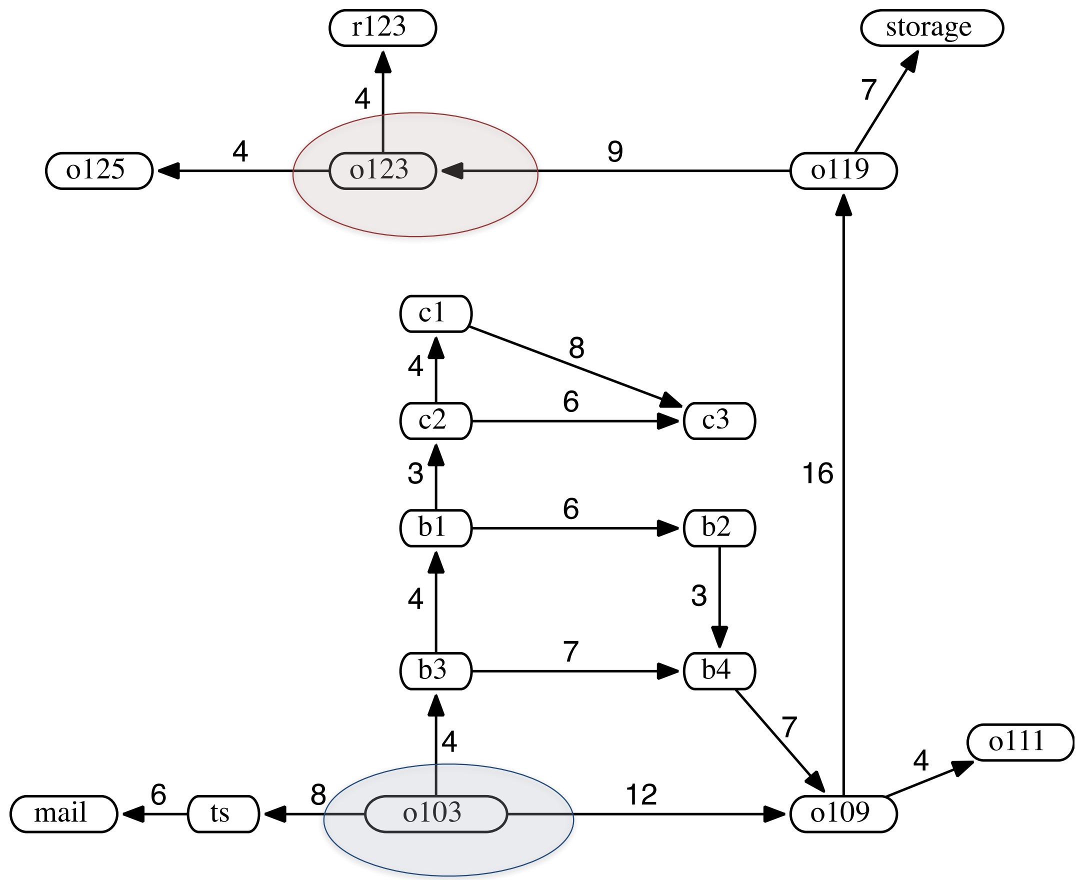
$[o103_0]$

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

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

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

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



Uniform-Cost Search

- Expand root first, then expand least-cost unexpanded node
- Implementation with priority queue
  - insert nodes in order of increasing path cost - lowest path cost is $g(n)$.
- 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)

Properties of Uniform-Cost Search

Complete? Yes, if *b* is finite and if transition $cost \geq \epsilon$ with $\epsilon > 0$

Time? Worst case, $O(b^{[C*/\epsilon]})$ where $C*$ = cost of the optimal solution

every transition costs at least ϵ

$\therefore$ cost per step is $\dfrac{C*}{\epsilon}$

Space? $O(b^{[C*/\epsilon]})$, $b^{[C*/\epsilon]} = b^d$ 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	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 Bounded 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 ?

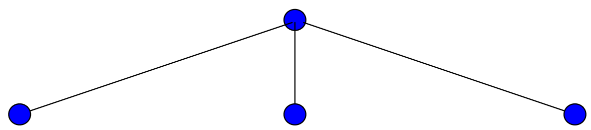
Iterative Deepening Search

- Depth-bounded search: hard to decide on a depth bound
- Iterative deepening: Try all possible depth bounds in turn
- Combines benefits of depth-first and breadth-first search

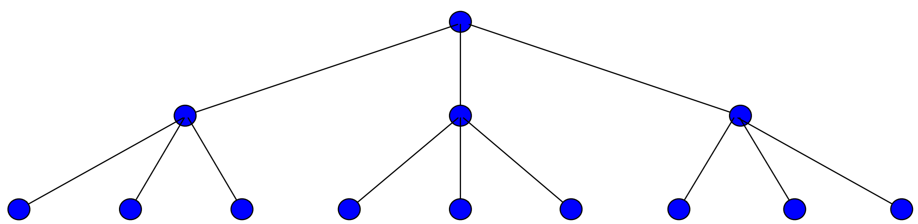
Iterative Deepening Search

- Tries to combine the benefits of depth-first (low memory) and breadth-first (optimal and complete)
- Does a series of depth-limited depth-first 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.

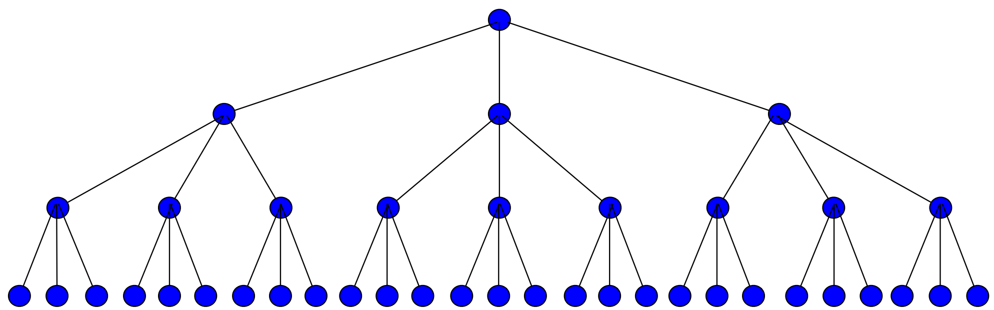
Iterative Deepening Search



Iterative Deepening Search



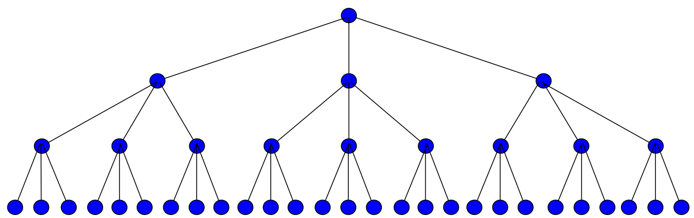
Iterative Deepening Search



Properties of Iterative Deepening Search

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

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

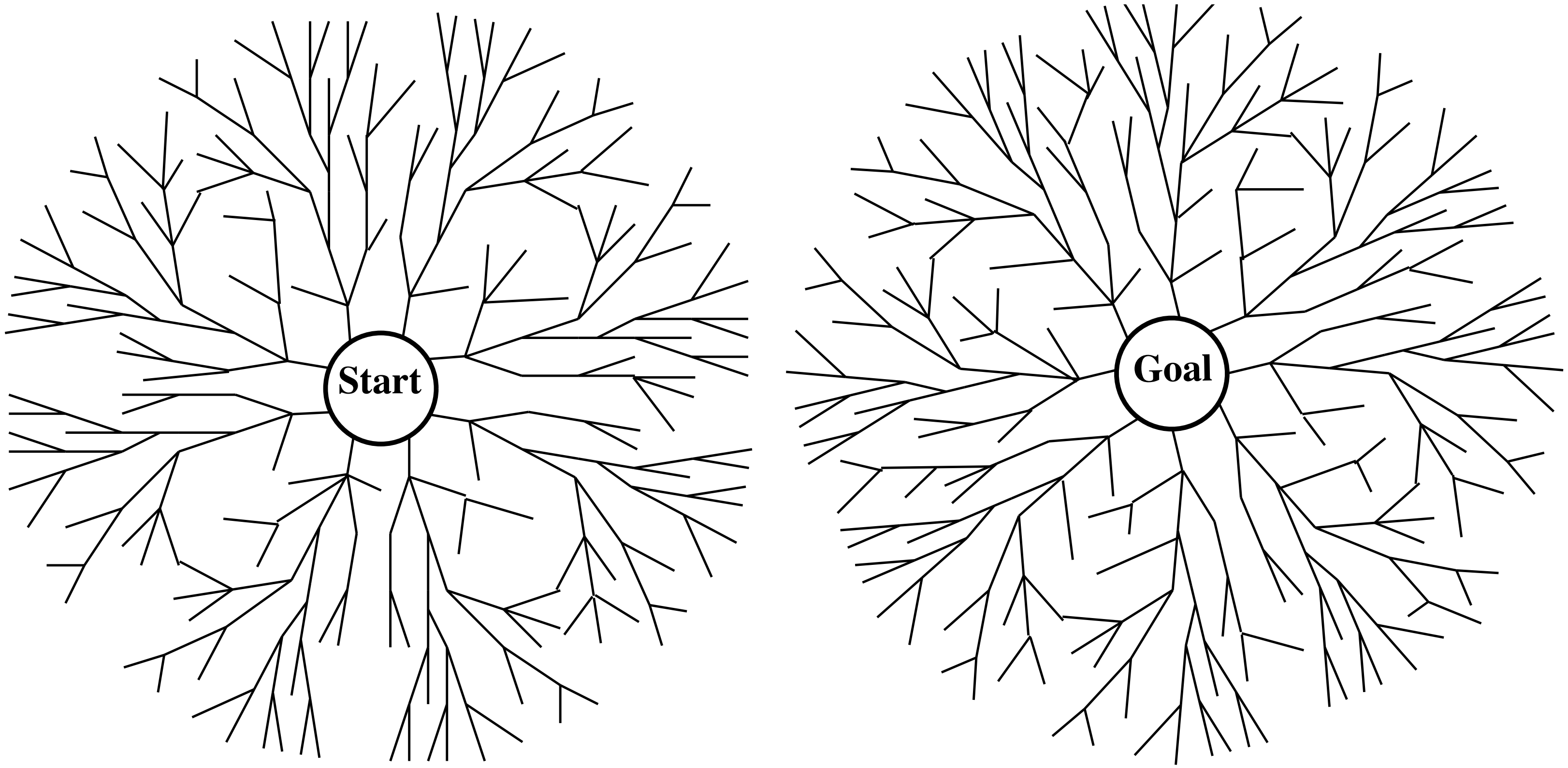


  - Example b=10, d=5:
    - depth-bounded: 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111
    - iterative-deepening: 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456
    - only about 11% more nodes (for b = 10).

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

- Search both forward from the initial state and backward from the goal
  - stop when the two searches meet in the middle.
- Need efficient way to check if a new node appears in the other half of the search.
  - 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.