Announcements

- Assignment 0 on Canvas. Please add (at top)
  - "presented" if you present it in class
  - "posted first" if you posted your application first.
- Assignment 1 available (see schedule).
- If you are on waiting list you need to do assignments (no guarantees).

Today

- AI Applications (assignment 0)
- Graph searching as an abstraction of problems

Assignment 0

- What is the application?
- What does the applications do?
- Give a goal, prior knowledge, past experiences, stimuli, actions.
- What AI technology does it us?
- Why is it intelligent?
- How well does it perform?

Searching

- Often we are not given an algorithm to solve a problem, but only a specification of what is a solution — we have to search for a solution.

Searching

- Often we are not given an algorithm to solve a problem, but only a specification of what is a solution — we have to search for a solution.
- A typical problem is when the agent knows its current state, it has a set of deterministic actions it can carry out, and wants to get to a goal state.

Searching

- Often we are not given an algorithm to solve a problem, but only a specification of what is a solution — we have to search for a solution.
- A typical problem is when the agent knows its current state, it has a set of deterministic actions it can carry out, and wants to get to a goal state.
- Many AI problems can be abstracted into the problem of finding a path in a directed graph.

Searching

- Often we are not given an algorithm to solve a problem, but only a specification of what is a solution — we have to search for a solution.
- A typical problem is when the agent knows its current state, it has a set of deterministic actions it can carry out, and wants to get to a goal state.
- Many AI problems can be abstracted into the problem of finding a path in a directed graph.
- Often there is more than one way to represent a problem as a graph.

State-space Problem

A state-space problem consists of

- a set of states (perhaps infinite)
- a start state
- a set of actions
- an action function: given a state and an action, returns a new state
- a way to recognize goal states, specified as function, $goal(s)$ that is true if s is a goal state
- a criterion that specifies the quality of an acceptable solution.

Directed Graphs

- A (directed) graph consists of a set N of nodes and a set A of ordered pairs of nodes, called arcs.

Directed Graphs

- A (directed) graph consists of a set N of nodes and a set A of ordered pairs of nodes, called arcs.
- Node n_2 is a neighbor of n_1 if there is an arc from n_1 to n_2. That is, if $\langle n_1, n_2 \rangle \in A$.

Directed Graphs

- A (directed) graph consists of a set N of nodes and a set A of ordered pairs of nodes, called arcs.
- Node n_2 is a neighbor of n_1 if there is an arc from n_1 to n_2. That is, if $\langle n_1, n_2 \rangle \in A$.
- A path is a sequence of nodes $\langle n_0, n_1, \ldots, n_k \rangle$ such that $\langle n_{i-1}, n_i \rangle \in A$.

Directed Graphs

- A (directed) graph consists of a set N of nodes and a set A of ordered pairs of nodes, called arcs.
- Node n_2 is a neighbor of n_1 if there is an arc from n_1 to n_2. That is, if $\langle n_1, n_2 \rangle \in A$.
- A path is a sequence of nodes $\langle n_0, n_1, \ldots, n_k \rangle$ such that $\langle n_{i-1}, n_i \rangle \in A$.
- Given start nodes and goal nodes, a solution is a path from a start node to a goal node.

Directed Graphs

- A (directed) graph consists of a set N of nodes and a set A of ordered pairs of nodes, called arcs.
- Node n_2 is a neighbor of n_1 if there is an arc from n_1 to n_2. That is, if $\langle n_1, n_2 \rangle \in A$.
- A path is a sequence of nodes $\langle n_0, n_1, \ldots, n_k \rangle$ such that $\langle n_{i-1}, n_i \rangle \in A$.
- Given start nodes and goal nodes, a solution is a path from a start node to a goal node.
- When there is a cost associated with arcs, the cost of a path is the sum of the costs of the arcs in the path:

$$cost(\langle n_0, n_1, \ldots, n_k \rangle) = \sum_{i=1}^{k} cost(\langle n_{i-1}, n_i \rangle)$$

An optimal solution is one with minimum cost.

What is a state (clicker)?

What is a state (clicker)?

A state needs to include enough information to

- determine what is the next state

What is a state (clicker)?

A state needs to include enough information to

- determine what is the next state
- determine whether the goal is achieved

What is a state (clicker)?

A state needs to include enough information to

- determine what is the next state
- determine whether the goal is achieved
- determine the cost.

What is a state (clicker)?

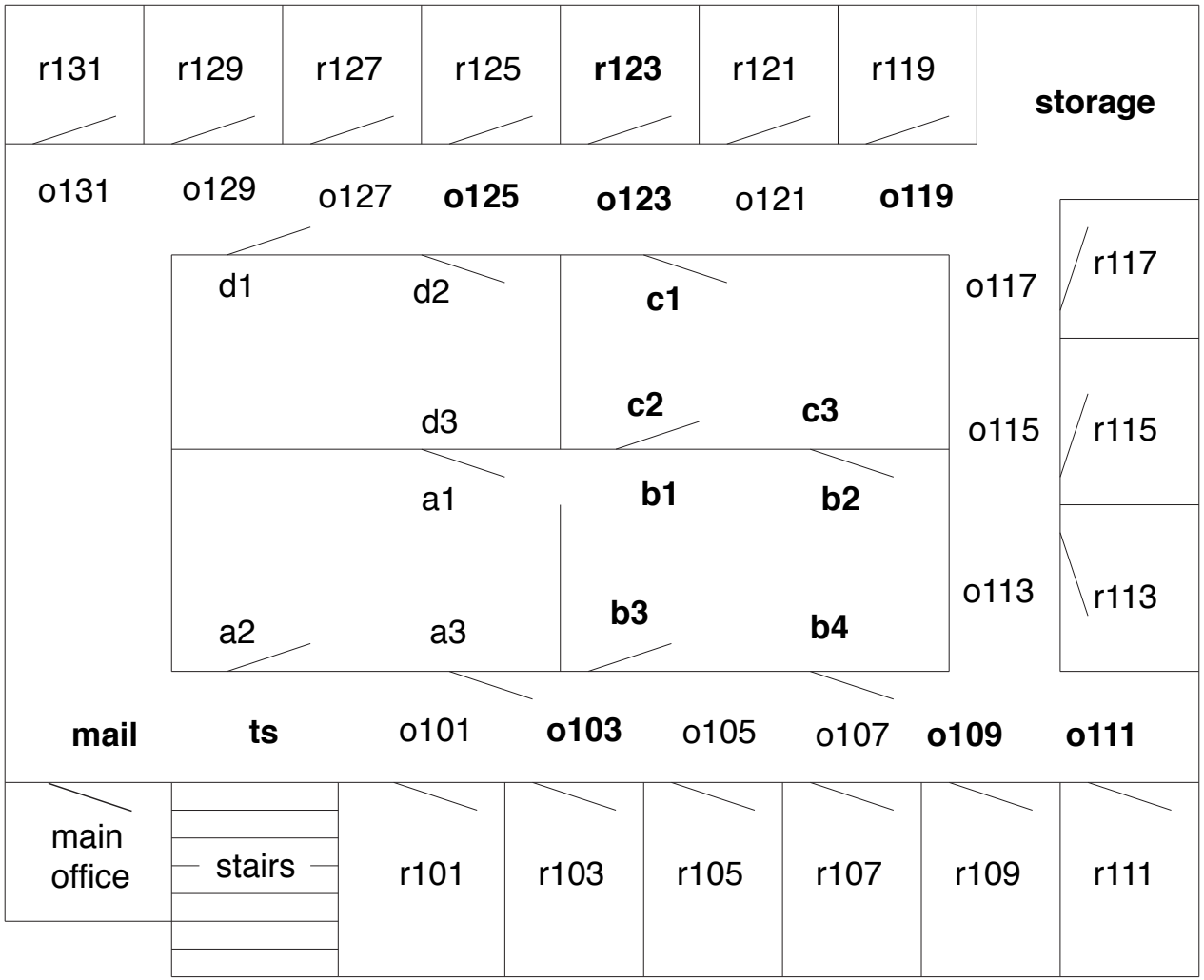
A state needs to include enough information to

- determine what is the next state
- determine whether the goal is achieved
- determine the cost.

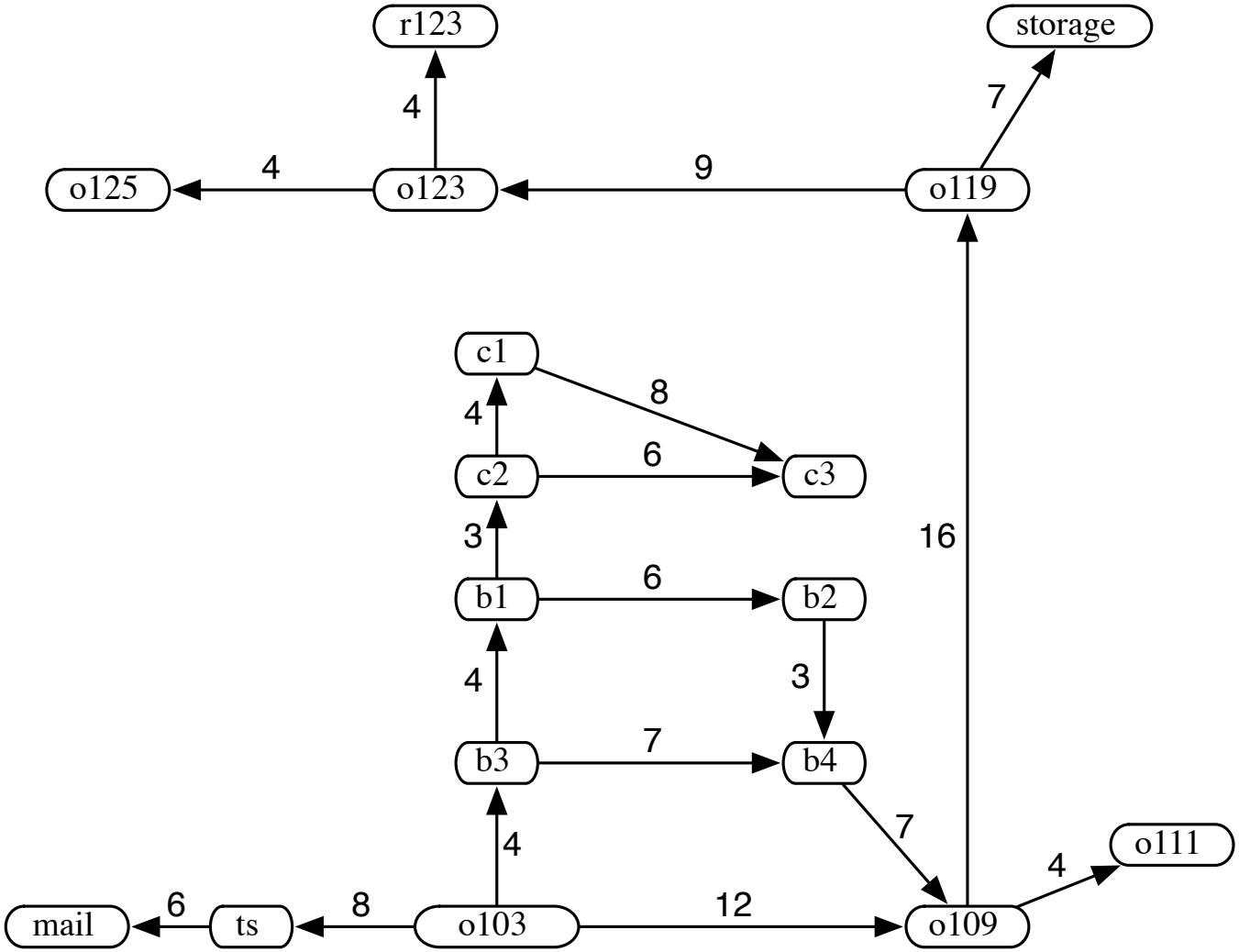
Often there are many options for what to include in the state.
Keep the states as simple as possible but no simpler.

Example Problem for Delivery Robot

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



State-Space Graph for the Delivery Robot (acyclic_delivery_problem)



Robot Cleaner

- 2 rooms, one cleaning robot
- rooms can be clean or dirty
- robot actions:
  suck: makes the room that the robot is in clean
  move: move to other room
- Goal: have both rooms clean

Robot Cleaner

- 2 rooms, one cleaning robot
- rooms can be clean or dirty
- robot actions:
  suck: makes the room that the robot is in clean
  move: move to other room
- Goal: have both rooms clean
- How many states are there? What are they?

Graph Search Algorithm

Input: a graph
a start node s
Boolean procedure $goal(n)$ that tests if n is a goal node
$frontier := \{\langle s \rangle\}$
while *frontier* is not empty:
select and **remove** path $\langle n_0, \ldots, n_k \rangle$ from *frontier*
if $goal(n_k)$
return $\langle n_0, \ldots, n_k \rangle$
$Frontier := Frontier \cup \{\langle n_0, \ldots, n_k, n \rangle : \langle n_k, n \rangle \in A\}$
end while

Graph Search Algorithm

- Which value is selected from the frontier at each stage defines the search strategy.
- The neighbors define the graph.
- *goal* defines what is a solution.
- If more than one answer is required, the search can continue from the return.

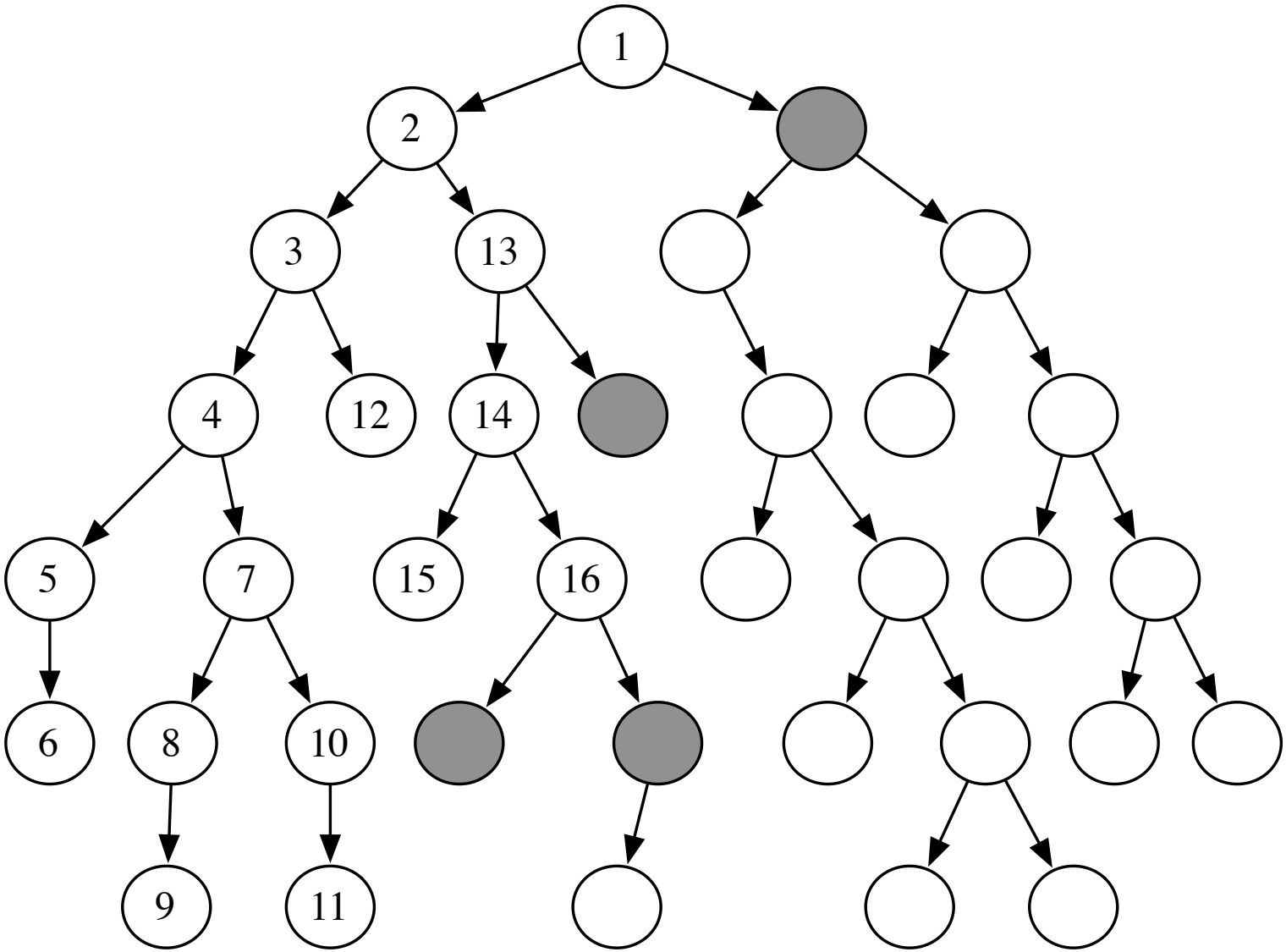
Depth-first Search

- Depth-first search treats the frontier as a stack. (First-in last-out)

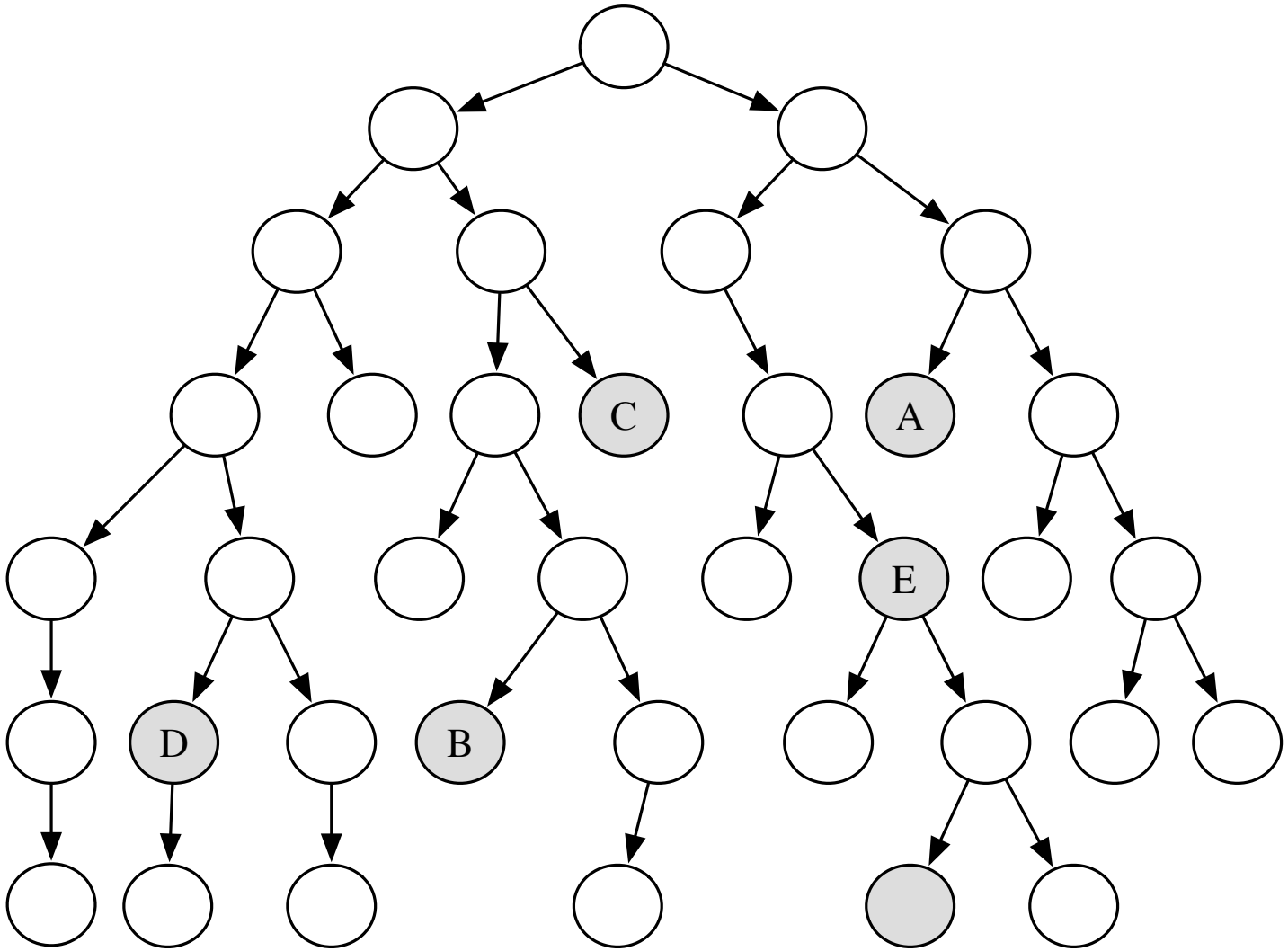
Depth-first Search

- Depth-first search treats the frontier as a stack. (First-in last-out)
- It always selects one of the last elements added to the frontier.

Illustrative Graph — Depth-first Search



Which shaded goal will depth-first search find first?



Complexity of Depth-first Search

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

Complexity of Depth-first Search

- Does depth-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 Depth-first Search

- Does depth-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 length of the path selected?

Complexity of Depth-first Search

- Does depth-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 length of the path selected?
- What is the space complexity as a function of length of the path selected?

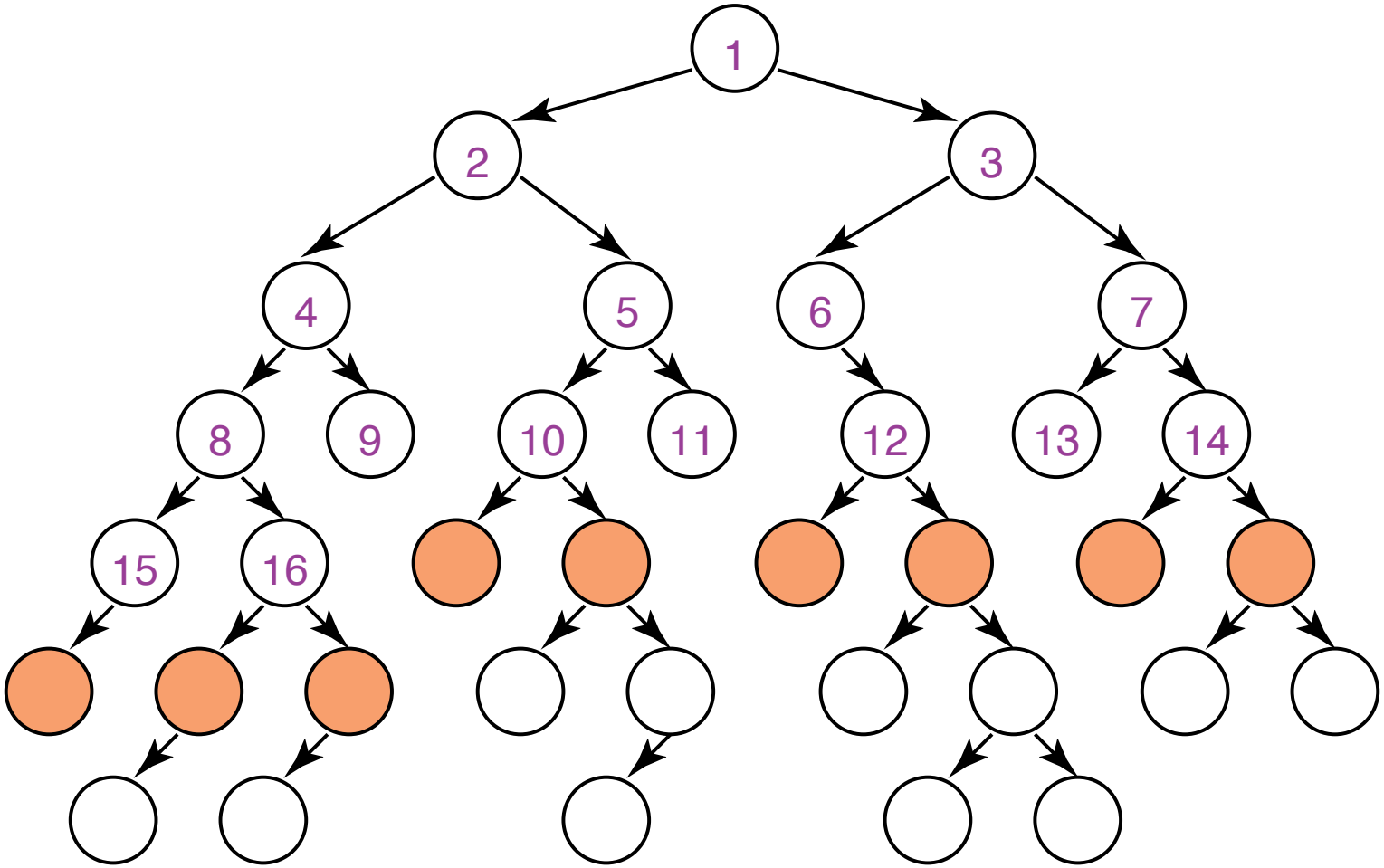
Complexity of Depth-first Search

- Does depth-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 length of the path selected?
- What is the space complexity as a function of length of the path selected?
- How does the goal affect the search?

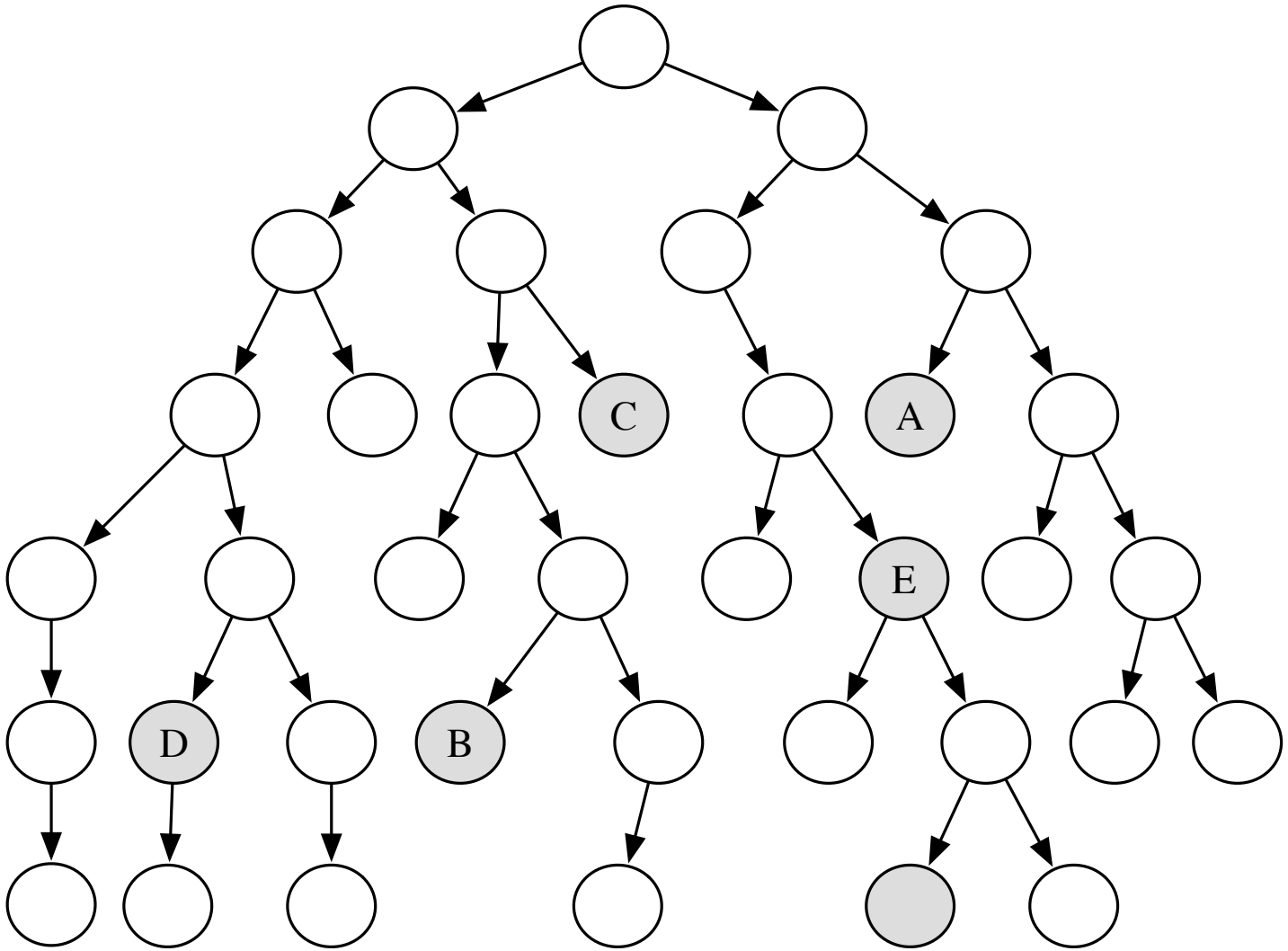
Breadth-first Search

- Breadth-first search treats the frontier as a queue (first-in, first-out).
- It always selects one of the earliest elements added to the frontier.

Illustrative Graph — Breadth-first Search



Which shaded goal will breadth-first search find first?



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?

Lowest-cost-first 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.

Lowest-cost-first 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.
- At each stage, lowest-cost-first search selects a path on the frontier with lowest cost.
- The frontier is a priority queue ordered by path cost.
- The first path to a goal is

Lowest-cost-first 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.
- At each stage, lowest-cost-first search selects a path on the frontier with lowest cost.
- The frontier is a priority queue ordered by path cost.
- The first path to a goal is a least-cost path to a goal node.
- When arc costs are equal $\Longrightarrow$

Lowest-cost-first 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.
- At each stage, lowest-cost-first search selects a path on the frontier with lowest cost.
- The frontier is a priority queue ordered by path cost.
- The first path to a goal is a least-cost path to a goal node.
- When arc costs are equal $\Longrightarrow$breadth-first search.

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