Presented by Yasin Ceran

September 17, 2024

2 / 32

Learning Objectives

At the end of the class you should be able to:

- explain how a generic searching algorithm works
- demonstrate how depth-first search will work on a graph
- demonstrate how breadth-first search will work on a graph
- predict the space and time requirements for depth-first and breadth-first searches

Graph Searching

- Generic search algorithm: given a graph, start nodes, and goal nodes, incrementally explore paths from the start nodes.
- Maintain a frontier of paths from the start node that have been explored.
- As search proceeds, the frontier expands into the unexplored nodes until a goal node is encountered.
- The way in which the frontier is expanded defines the search strategy.

Graph Searching

- Generic search algorithm: given a graph, start nodes, and goal nodes, incrementally explore paths from the start nodes.
- Maintain a frontier of paths from the start node that have been explored.
- As search proceeds, the frontier expands into the unexplored nodes until a goal node is encountered.
- The way in which the frontier is expanded defines the search strategy.

4 / 32

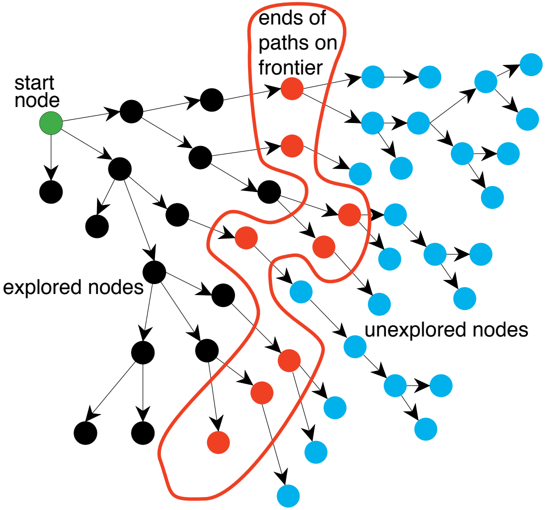
Graph Searching

- Generic search algorithm: given a graph, start nodes, and goal nodes, incrementally explore paths from the start nodes.
- Maintain a frontier of paths from the start node that have been explored.
- As search proceeds, the frontier expands into the unexplored nodes until a goal node is encountered.
- The way in which the frontier is expanded defines the search strategy.

Graph Searching

- Generic search algorithm: given a graph, start nodes, and goal nodes, incrementally explore paths from the start nodes.
- Maintain a frontier of paths from the start node that have been explored.
- As search proceeds, the frontier expands into the unexplored nodes until a goal node is encountered.
- The way in which the frontier is expanded defines the search strategy.

Problem Solving by Graph Searching



Graph Search Algorithm

Input: a graph,
a set of start nodes,
Boolean procedure $goal(n)$ that tests if n is a goal node.
$frontier := \{\langle s \rangle : s \text{ is a start node}\}$
while *frontier* is not empty:
  select and **remove** path $\langle n_0, \ldots, n_k \rangle$ from *frontier*
  if $goal(n_k)$
    return $\langle n_0, \ldots, n_k \rangle$
  for every neighbor n of n_k
    add $\langle n_0, \ldots, n_k, n \rangle$ to *frontier*
end while

6 / 32

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.

7 / 32

Optimality Criteria

- Often we don't want any solution, but the best solution or optimal solution.
- Costs on arcs give costs on paths. We want the least-cost path to a goal.

Uninformed Search Strategies

- A problem determines the graph, the start node, and the goal but not which path to select from the frontier.
- A search strategy defines the order in which paths are selected from the frontier.
- Uninformed search strategies that do not take into account the location of the goal.
- Uninformed strategies use only the information available in the problem definition
- Intuitively, these algorithms ignore where they are going until they find a goal and report success.

9 / 32

Measuring Problem-Solving Performance

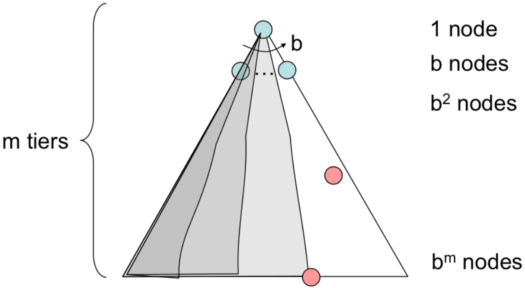
A strategy is defined by picking the **order of node expansion**

Strategies are evaluated along the following dimensions:
completeness—does it always find a solution if one exists?
time complexity—number of nodes generated/expanded
space complexity—maximum number of nodes in memory
optimality—does it always find a least-cost solution?

Time and space complexity are measured in terms of
b—maximum branching factor of the search tree
d—depth of the least-cost solution
m—maximum depth of the state space (may be ∞)

Depth-first search

Expand deepest unexpanded node



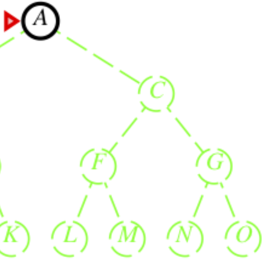
Depth-first Search

- Depth-first search treats the frontier as a LIFO (last-in, first-out) stack of paths
- It always selects one of the last elements added to the frontier.
- If the list of paths on the frontier is $[p_1, p_2, \ldots]$
  - p_1 is selected. Paths that extend p_1 are added to the front of the stack (in front of p_2).
  - p_2 is only selected when all paths from p_1 have been explored.

Depth-first search

Implementation:

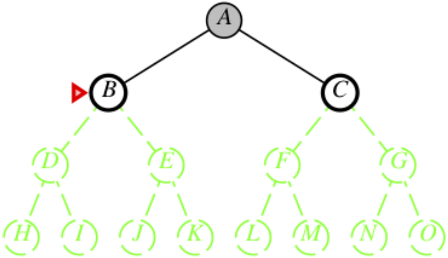
fringe = LIFO queue, i.e., put successors at front



Depth-first search

Implementation:

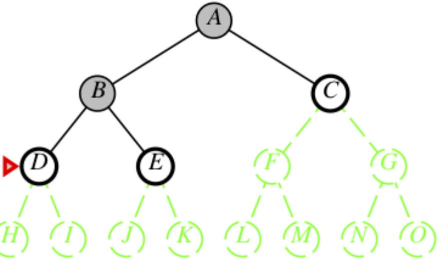
fringe = LIFO queue, i.e., put successors at front



Depth-first search

Implementation:

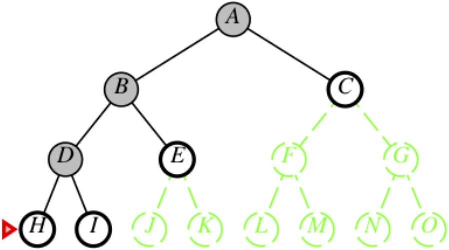
fringe = LIFO queue, i.e., put successors at front



Depth-first search

Implementation:

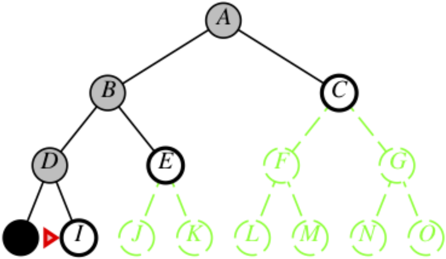
fringe = LIFO queue, i.e., put successors at front



Depth-first search

Implementation:

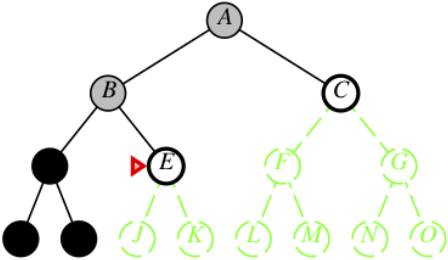
fringe = LIFO queue, i.e., put successors at front



Depth-first search

Implementation:

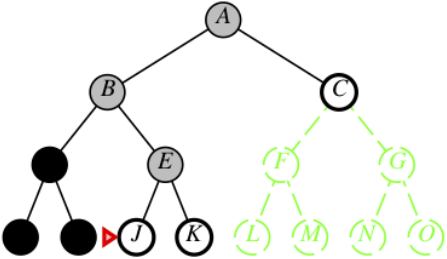
fringe = LIFO queue, i.e., put successors at front



Depth-first search

Implementation:

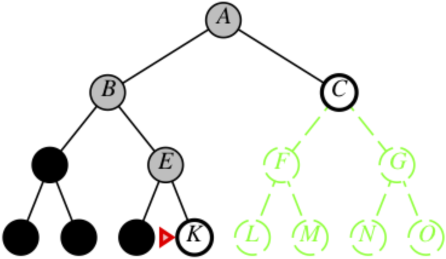
fringe = LIFO queue, i.e., put successors at front



Depth-first search

Implementation:

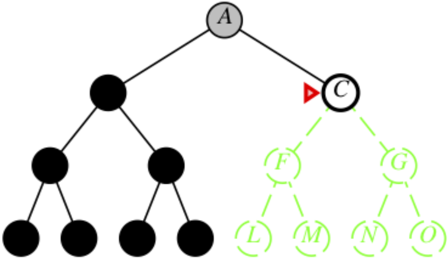
fringe = LIFO queue, i.e., put successors at front



Depth-first search

Implementation:

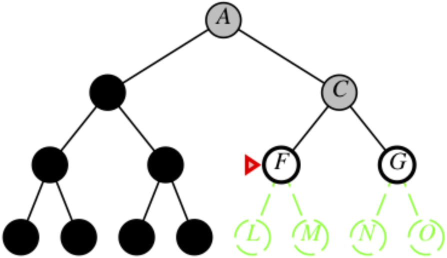
fringe = LIFO queue, i.e., put successors at front



Depth-first search

Implementation:

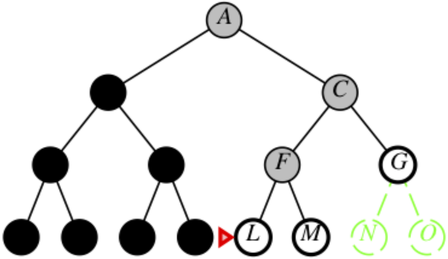
fringe = LIFO queue, i.e., put successors at front



Depth-first search

Implementation:

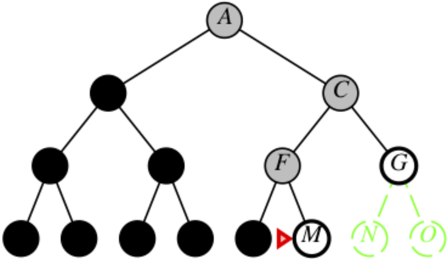
fringe = LIFO queue, i.e., put successors at front



Depth-first search

Implementation:

fringe = LIFO queue, i.e., put successors at front



Properties of depth-first search

- Complete?? No: fails in infinite-depth spaces, spaces with loops

  Modify to avoid repeated states along path
  $\Rightarrow$ complete in finite spaces

- Time?? $O(b^m)$: 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

Properties of depth-first search

- Complete?? No: fails in infinite-depth spaces, spaces with loops

  Modify to avoid repeated states along path
  $\Rightarrow$ complete in finite spaces

- Time?? $O(b^m)$: 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

14 / 32

Properties of depth-first search

- Complete?? No: fails in infinite-depth spaces, spaces with loops

  Modify to avoid repeated states along path
  $\Rightarrow$ complete in finite spaces
- Time?? $O(b^m)$: 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

14 / 32

Properties of depth-first search

- Complete?? No: fails in infinite-depth spaces, spaces with loops

  Modify to avoid repeated states along path
  $\Rightarrow$ complete in finite spaces

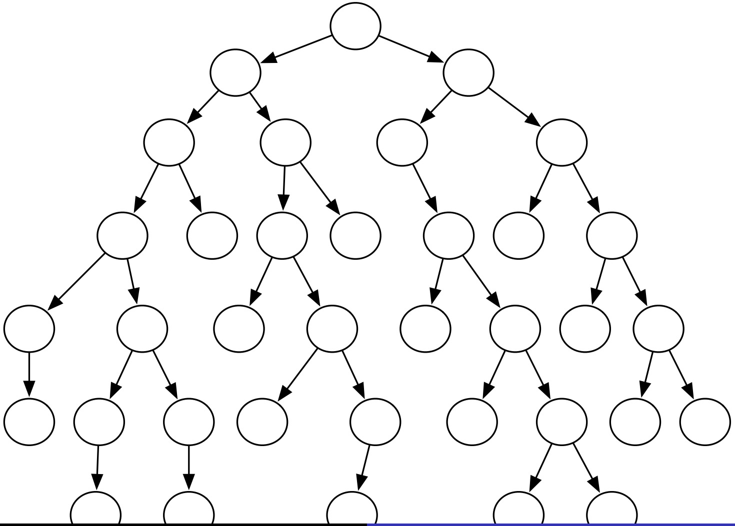
- Time?? $O(b^m)$: terrible if m is much larger than d
  but if solutions are dense, may be much faster than breadth-first

- Space?? $O(bm)$, i.e., linear space!

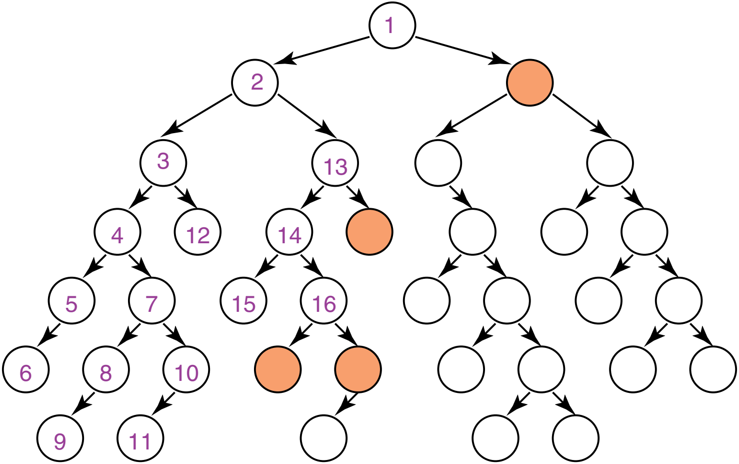
- Optimal?? No

Illustrative Graph - Depth-first search

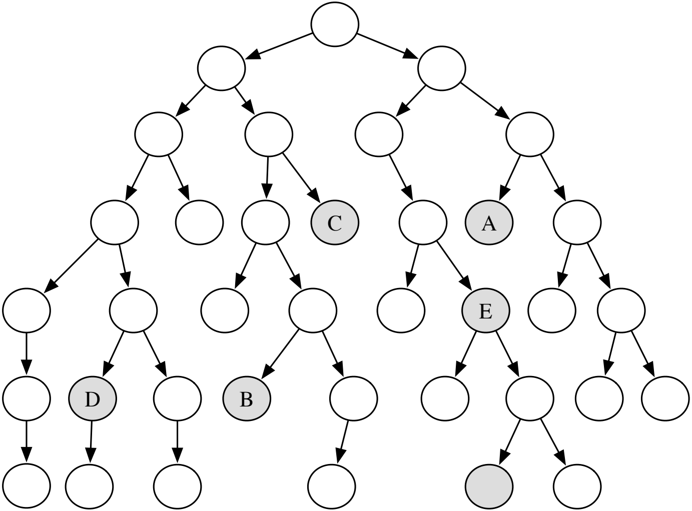
Start node is at the top.



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?
- 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?

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?

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?

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?

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?

Complexity of Depth-first Search

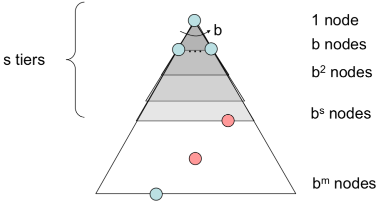
- Depth-first search isn't guaranteed to halt on infinite graphs or on graphs with cycles.
- The space complexity is linear in the size of the path being explored.
- Search is unconstrained by the goal until it happens to stumble on the goal.

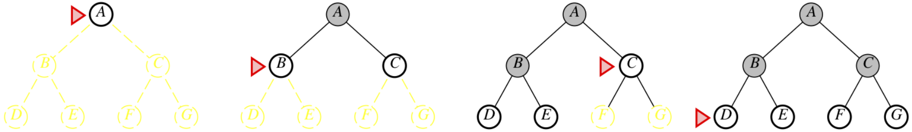
19 / 32

Breadth-first search

Expand shallowest unexpanded node

Implementation:
fringe is a FIFO queue, i.e., new successors go at end



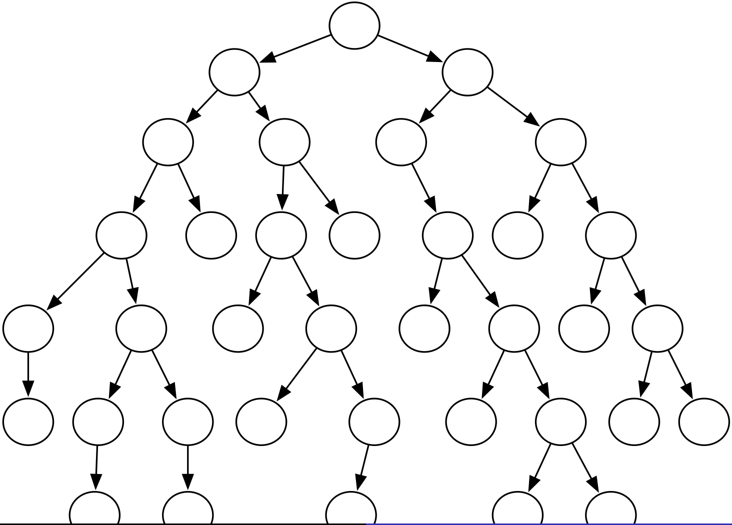


Breadth-first Search

- 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 $[p_1, p_2, \ldots, p_r]$:
    - p_1 is selected. Its neighbors are added to the end of the queue, after p_r.
    - p_2 is selected next.

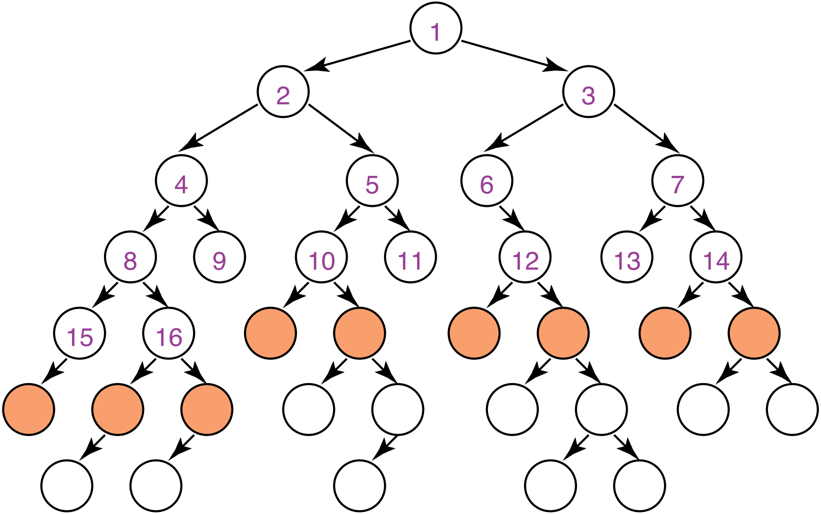
Illustrative Graph - Breadth-first search

Start node is at the top.

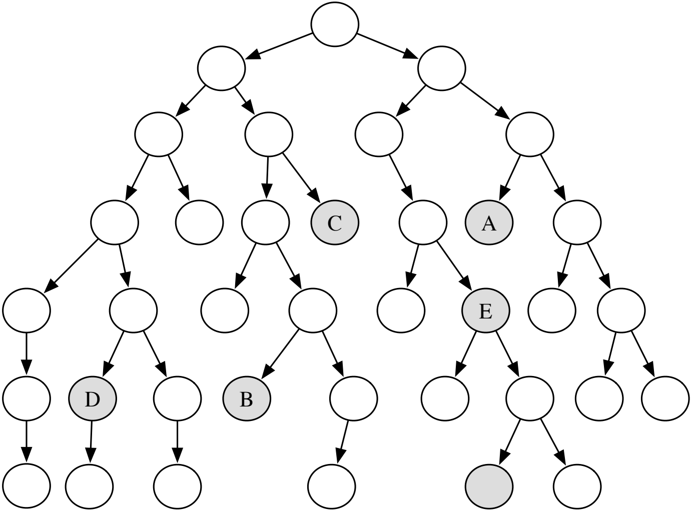


22 / 32

Illustrative Graph — Breadth-first Search



Which shaded goal will breadth-first search find first?



24 / 32

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?

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?

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?

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?

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?

Complexity of Breadth-first Search

- The branching factor of a node is the number of its neighbors.
- If the branching factor for all nodes is finite, breadth-first search is guaranteed to find a solution if one exists.
  It is guaranteed to find the path with fewest arcs.
- Time complexity is exponential in the path length:
  b^n, where b is branching factor, n is path length.
- The space complexity is exponential in path length: b^n.
- Search is unconstrained by the goal.

Properties of breadth-first search

- Complete?? Yes (if b is finite)
- Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d
- Space?? $O(b^{d+1})$ (keeps every node in memory)
- Optimal?? Yes (if cost $= 1$ per step); not optimal in general

27 / 32

Properties of breadth-first search

- Complete?? Yes (if b is finite)
- Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d
- Space?? $O(b^{d+1})$ (keeps every node in memory)
- Optimal?? Yes (if cost $= 1$ per step); not optimal in general

27 / 32

Properties of breadth-first search

- Complete?? Yes (if b is finite)
- Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d
- Space?? $O(b^{d+1})$ (keeps every node in memory)
- Optimal?? Yes (if cost $= 1$ per step); not optimal in general

27 / 32

Properties of breadth-first search

- Complete?? Yes (if b is finite)
- Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d
- Space?? $O(b^{d+1})$ (keeps every node in memory)
- Optimal?? Yes (if cost = 1 per step); not optimal in general

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.

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.

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.

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.

Depth-limited search

= depth-first search with depth limit l,
i.e., nodes at depth l have no successors

Recursive implementation:

function Depth-Limited-Search(*problem*, *limit*) **returns** soln/fail/cutoff
Recursive-DLS(Make-Node(Initial-State[*problem*]), *problem*, *limit*)

function Recursive-DLS(*node*, *problem*, *limit*) **returns** soln/fail/cutoff
cutoff-occurred? ← false
if Goal-Test(*problem*, State[*node*]) **then return** *node*
else if Depth[*node*] = *limit* **then return** *cutoff*
else for each *successor* **in** Expand(*node*, *problem*) **do**
result ← Recursive-DLS(*successor*, *problem*, *limit*)
if *result* = *cutoff* **then** *cutoff-occurred?* ← true
else if *result* ≠ *failure* **then return** *result*

Iterative deepening search

function Iterative-Deepening-Search(*problem*) **returns** a solution
inputs: *problem*, a problem

for *depth* $\leftarrow$ 0 **to** ∞ **do**
result $\leftarrow$ Depth-Limited-Search(*problem, depth*)
if *result* $\neq$ cutoff **then return** *result*
end

Iterative deepening search

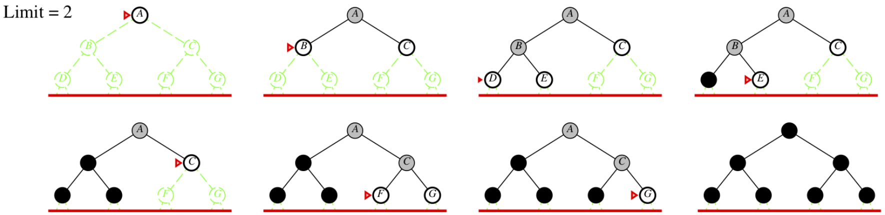
Limit = 0

Iterative deepening search

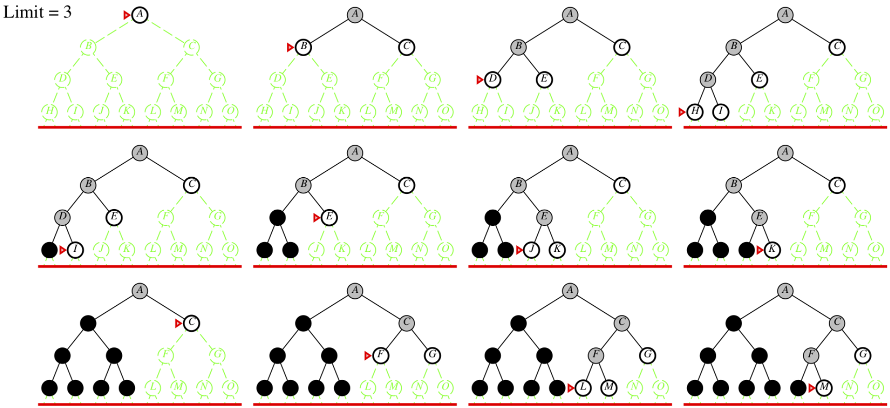
Limit = 1

31 / 32

Iterative deepening search



Iterative deepening search



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

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