Solving problems by searching

- Problem formulation
- Example problems
- Searching for solutions
  - Breadth-first search
  - Depth-first search
  - Iterative deepening search
  - Bidirectional search
- Avoiding repeated states

Components of well-defined problems

- **Initial state**
- Available **actions** given by **successor function**
  - Initial state + successor function define **state space**
  - **Path**: sequence of states connected by actions
- **Goal test** (i.e. is current state a goal state?)
- **Path costs** (here, sum of **step costs**)

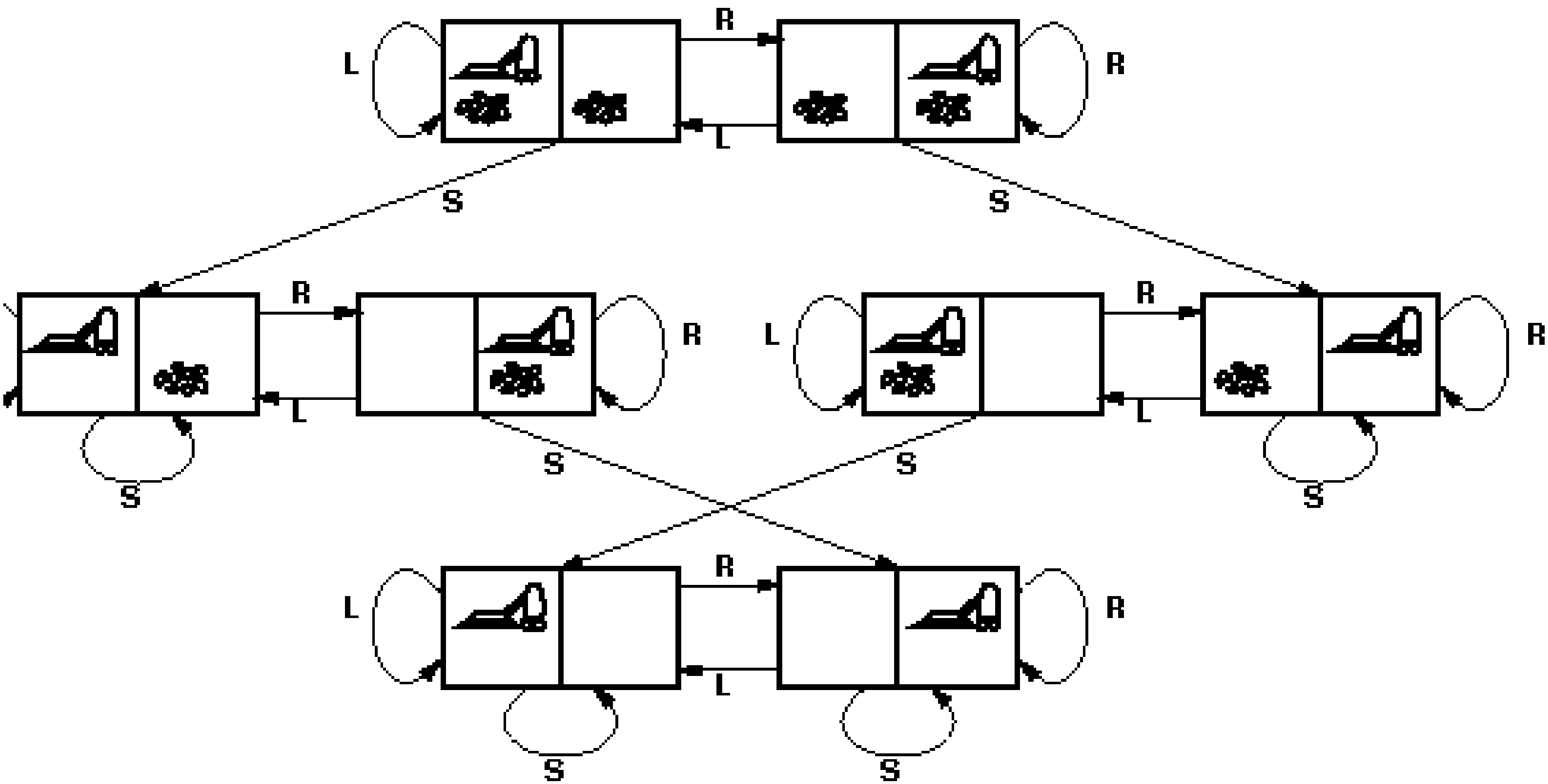
Solutions

- A **solution** to a problem is a path from the initial state to a goal state
- An **optimal solution** has the lowest path cost among all solutions
  - I.e. solution quality is measured by the path cost
- If path cost sum of step costs, and step costs equal, then optimum solution is shortest path

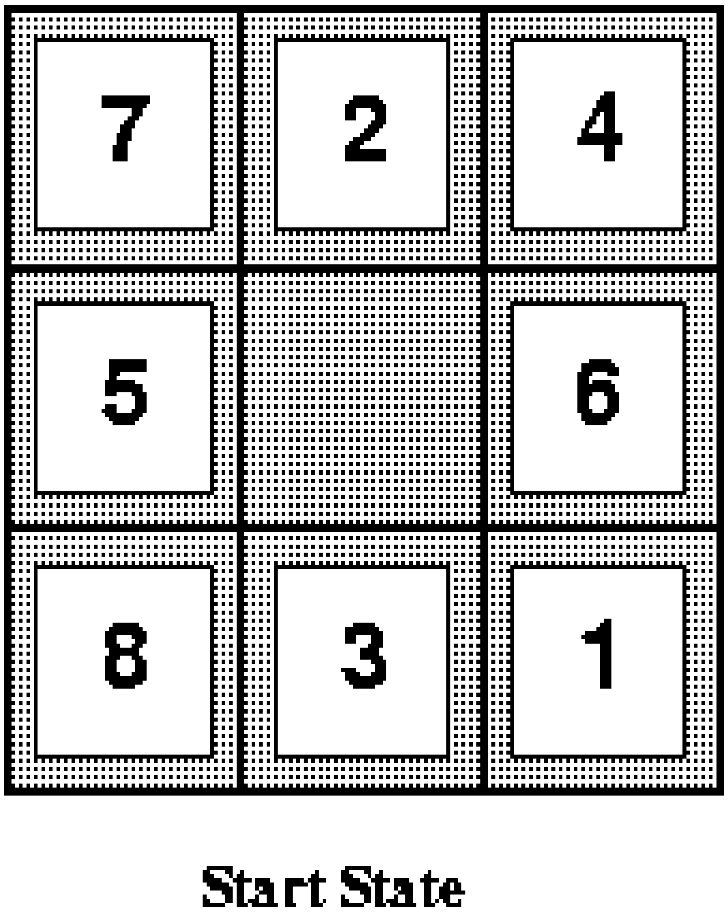
Example problem: vacuum world

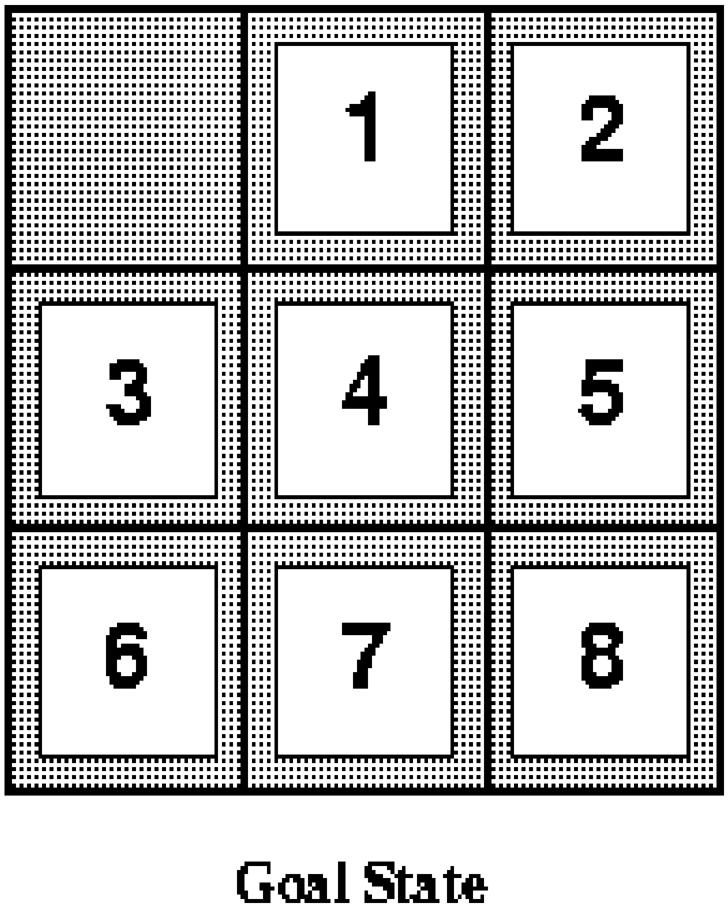
- **States**: two locations, which (a) are dirty or not and (b) contain cleaning robot or don't (8 states)
- **Initial state**: any state
- **Successor function**: generates legal states that result from actions *Left*, *Right*, and *Suck*
- **Goal test**: checks whether all squares are clean
- **Path cost**: the same cost for each step (e.g. 1), so the path cost is the number of steps it contains

State space for vacuum world



Example problem: 8-puzzle

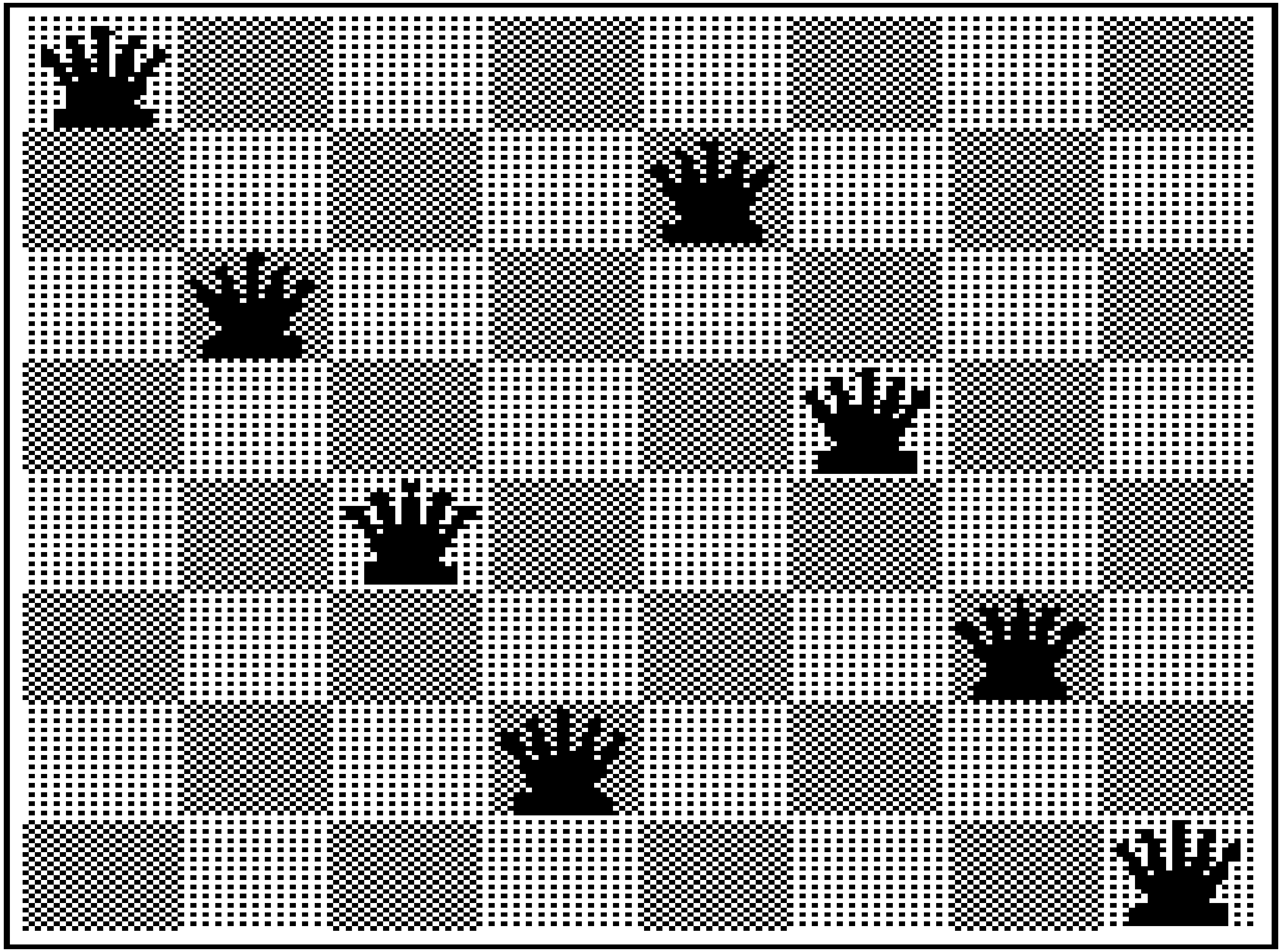




Example problem: 8-puzzle

- **States**: specify location of each tile and the blank
- **Initial state**: any state
- **Successor function**: generates legal states that result from actions *Left*, *Right*, *Up* and *Down*
- **Goal test**: checks state matches goal configuration shown on previous slide
- **Path cost**: the same for each step (e.g. 1), so the path cost is the number of steps it contains

Example problem: 8-queens



Example problem: 8-queens

- **States**: any arrangement of 0 to 8 queens
- **Initial state**: no queens on the board
- **Successor function**: add queen to any empty position
- **Goal test**: 8 queens on the board, none attacked
- **Path cost**: path cost is of no interest because only the final state counts

Example problem: 8-queens

- Number of states in previous formulation:

$$64 \times 63 \times \ldots \times 57 \approx 3 \times 10^{14}$$

- Better formulation:
  - **States**: n queens ($0 \leqq n \leq 8$), one per column in the leftmost n columns, with no queen attacking another
  - **Successor function**: add queen in any square in leftmost empty column such that it is not attacked
- Result: only 2,057 states!

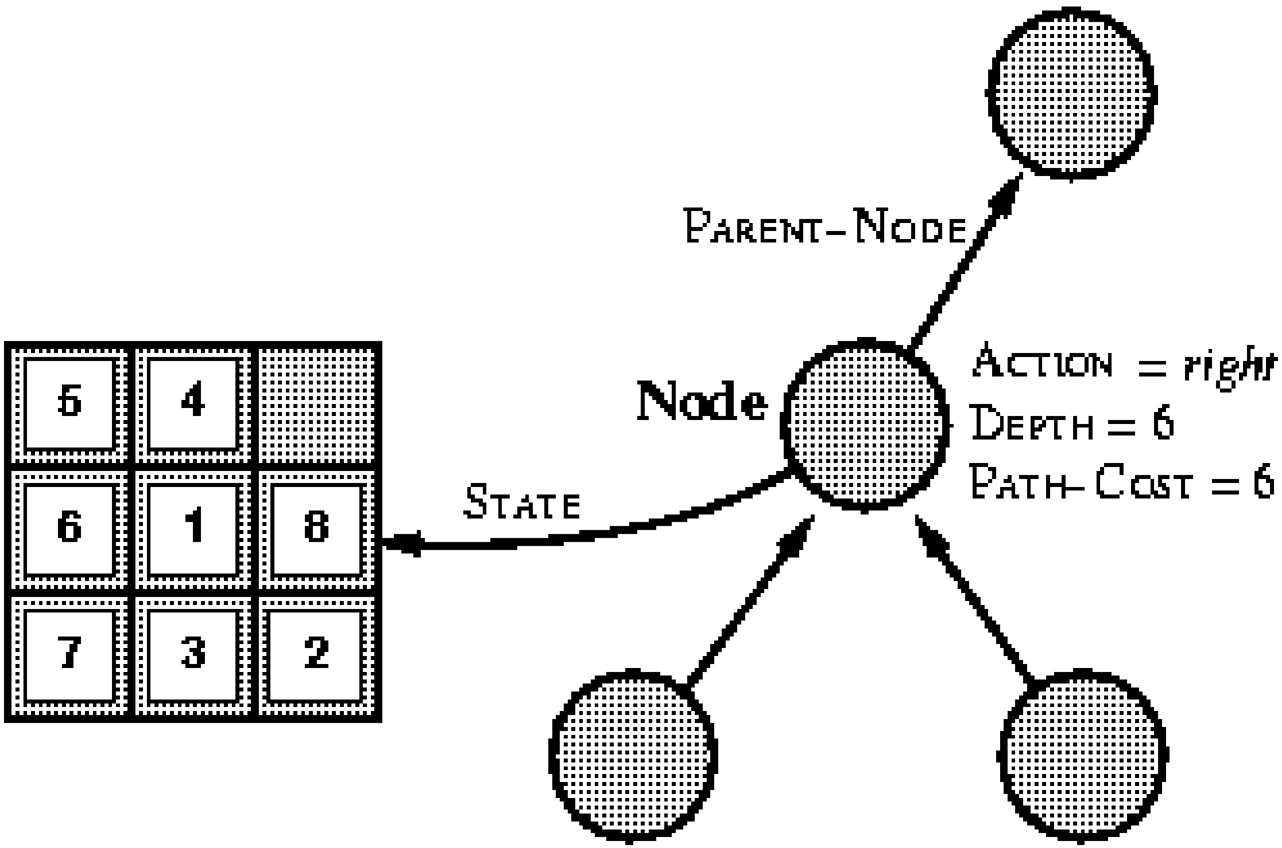
Tree-search algorithms

- Generate **search tree** by **expanding search nodes** according to a **search strategy**

function GENERAL-SEARCH(*problem, strategy*) **returns** a solution, or failure
    initialize the search tree using the initial state of *problem*
    loop do
        if there are no candidates for expansion **then return** failure
        choose a leaf node for expansion according to *strategy*
        if the node contains a goal state **then return** the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end

Implementation: nodes

- Node: data structure constituting part of a search tree



Implementation: general tree search

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
   fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
   loop do
      if fringe is empty then return failure
      node ← REMOVE-FRONT(fringe)
      if GOAL-TEST[problem] applied to STATE(node) succeeds return node
      fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND( node, problem) returns a set of nodes
   successors ← the empty set
   for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
      s ← a new NODE
      PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
      PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
      DEPTH[s] ← DEPTH[node] + 1
      add s to successors
   return successors
```

Search strategy (determines order of node expansion)

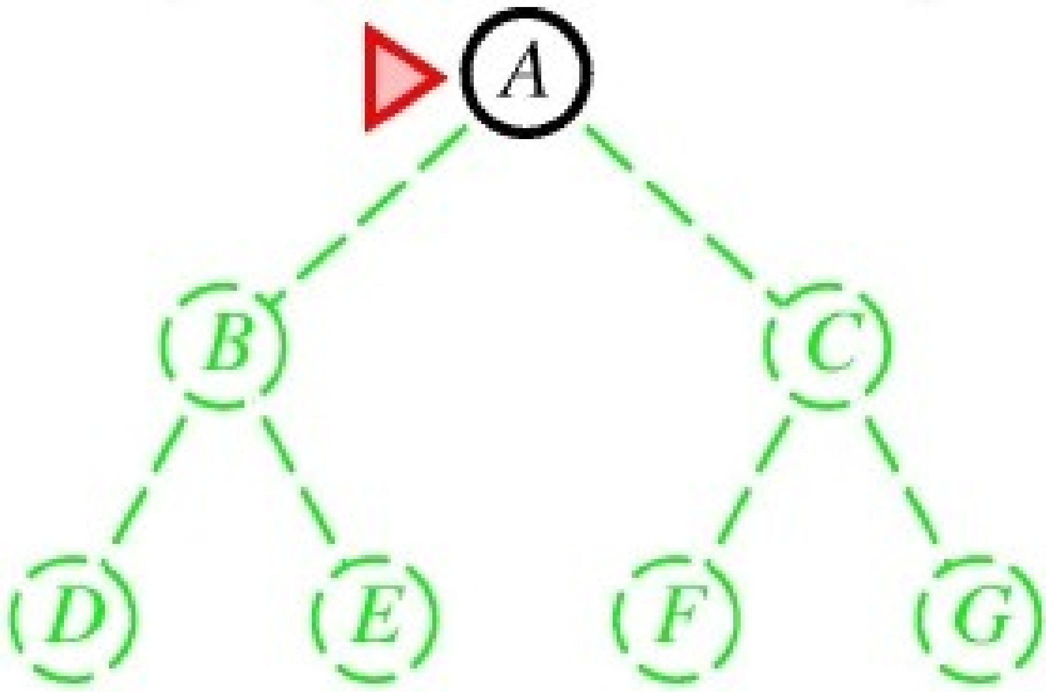
- Properties of search strategies:
  - **Completeness**: does it find a solution if one exists?
  - **Time complexity**: how long does it take?
  - **Space complexity**: how much memory does it take?
  - **Optimality**: does it find the optimal solution?
- Complexity is measured in terms of
  - **branching factor** *b*
  - depth of shallowest goal node *d*
  - maximum length of any path *m*

Uninformed search strategies

- Can only use the information available in the problem definition (also called **blind search**)
- Popular algorithms:
  - Breadth-first search
  - Depth-first search
  - Iterative deepening search
  - Bidirectional search

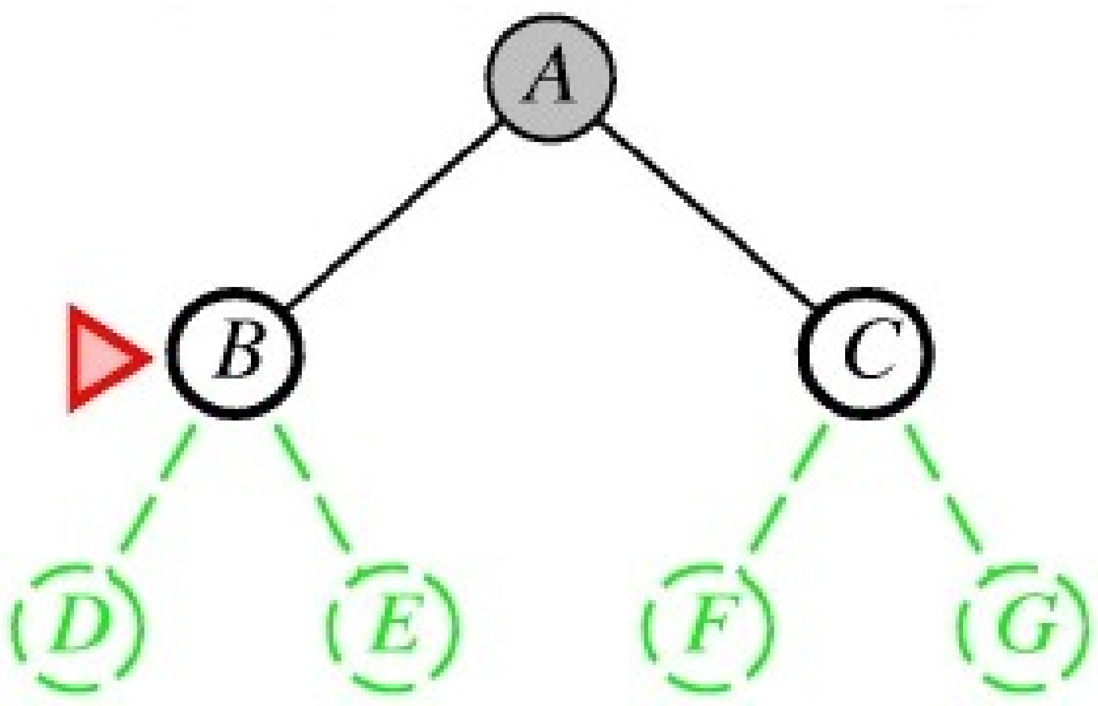
Breadth-first search

- Expand shallowest unexpanded node
- *fringe* is a FIFO queue, i.e. new nodes go at end



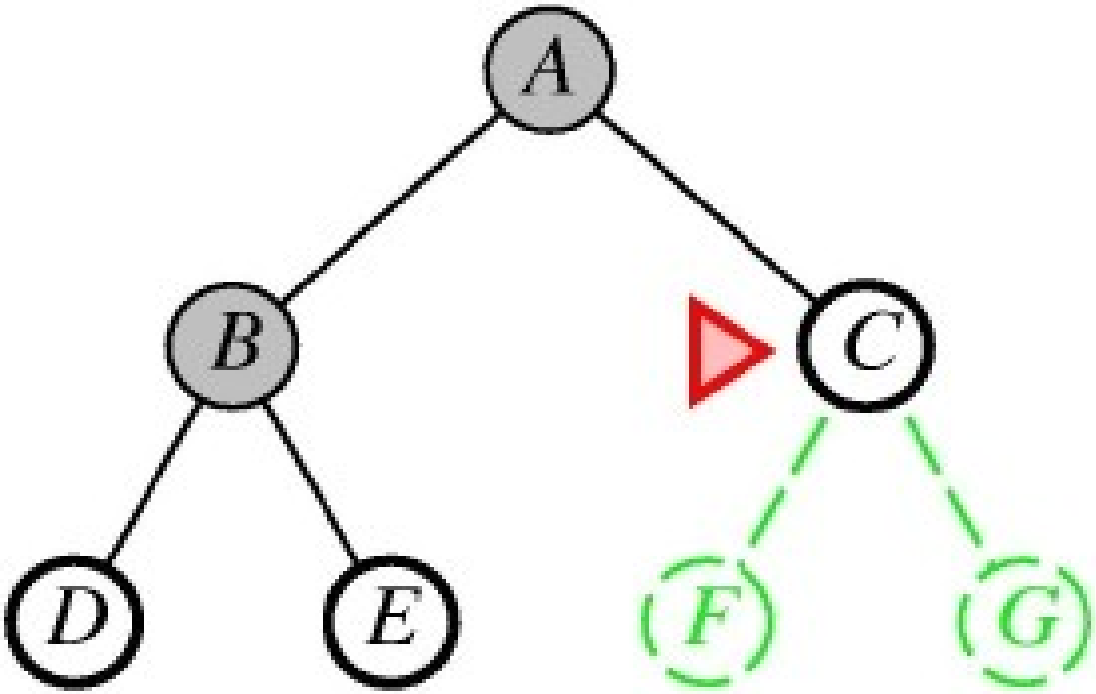
Breadth-first search

- Expand shallowest unexpanded node
- *fringe* is a FIFO queue, i.e. new nodes go at end



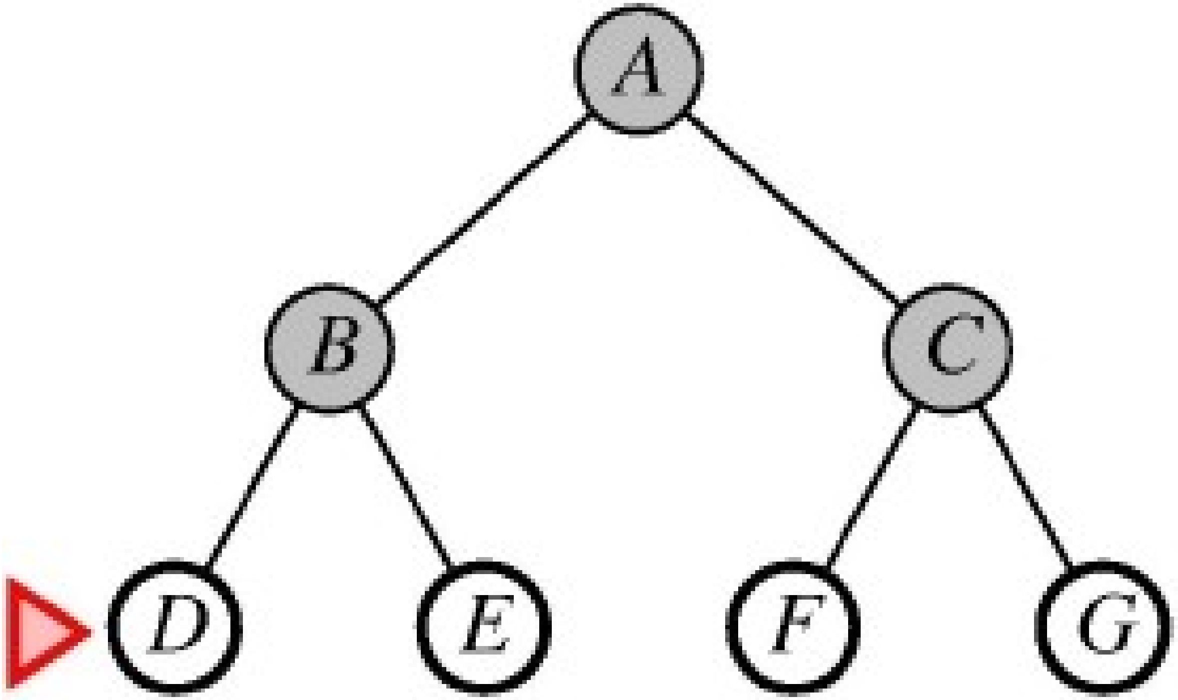
Breadth-first search

- Expand shallowest unexpanded node
- *fringe* is a FIFO queue, i.e. new nodes go at end



Breadth-first search

- Expand shallowest unexpanded node
- *fringe* is a FIFO queue, i.e. new nodes go at end



Properties of breadth-first search

- Complete?
- Time?
- Space?
- Optimal?

Properties of breadth-first search

- Complete?
  - Yes! (if *b* is finite)
- Time?
- Space?
- Optimal?

Properties of breadth-first search

- Complete?
  - Yes! (if b is finite)
- Time? (assuming solution is at level d)
  - $b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$
- Space?
- Optimal?

Properties of breadth-first search

- Complete?
  - Yes! (if b is finite)
- Time? (assuming solution is at level d)
  - $b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$
- Space?
  - $O(b^{d+1})$ (node either fringe node or ancestor of one)
- Optimal?

Properties of breadth-first search

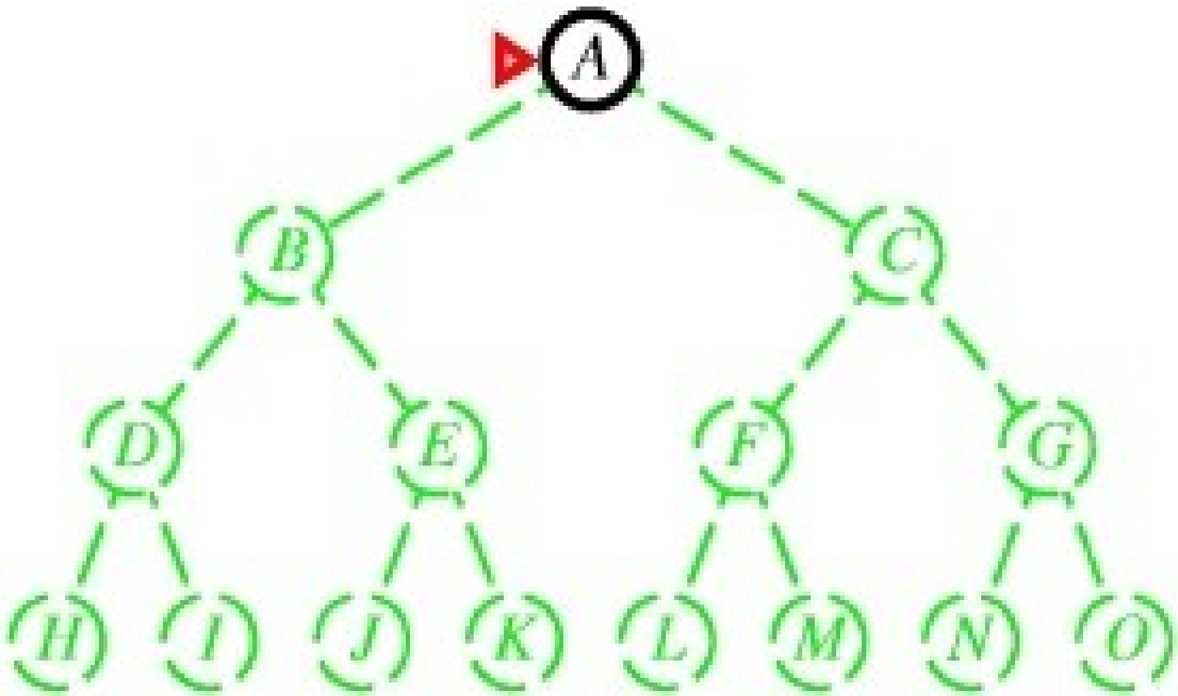
- Complete?
  - Yes! (if b is finite)
- Time? (assuming solution is at level d)
  - $b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$
- Space?
  - $O(b^{d+1})$ (node either fringe node or ancestor of one)
- Optimal?
  - Only if all step costs are equal

Uniform-cost search

- Expand unexpanded node with lowest path cost
- *fringe* is queue ordered by path costs
- Complete? Yes, if step costs $\geq \epsilon$
- Time? #nodes with path cost $\leq$ cost of optimal solution $C \Rightarrow O(b^{\lceil C/\epsilon \rceil})$
- Space? Same
- Optimal? Yes

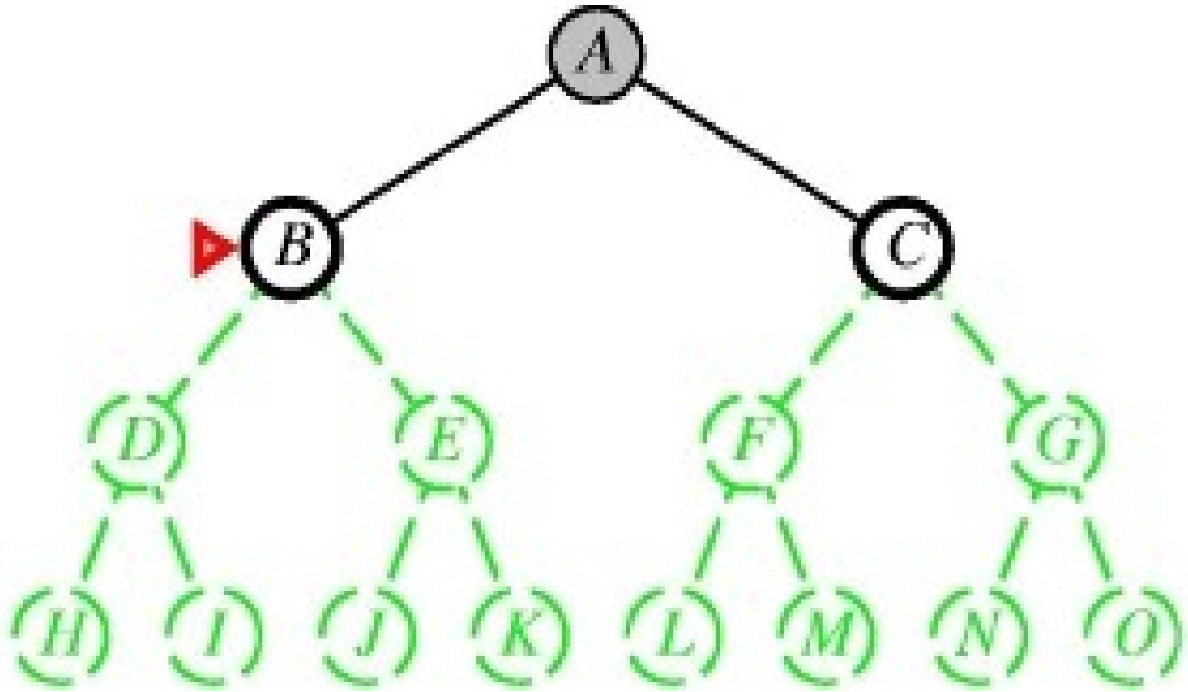
Depth-first search

- Expand deepest unexpanded node
- *fringe* is a LIFO queue, i.e. new nodes go at front



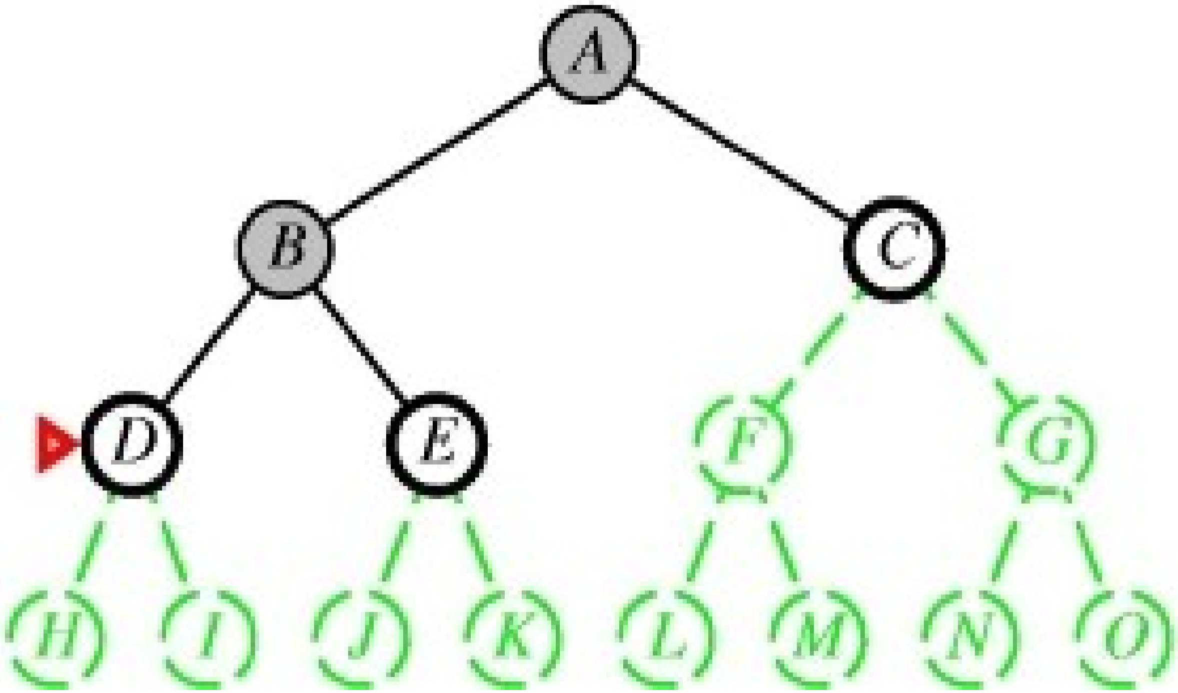
Depth-first search

- Expand deepest unexpanded node
- *fringe* is a LIFO queue, i.e. new nodes go at front



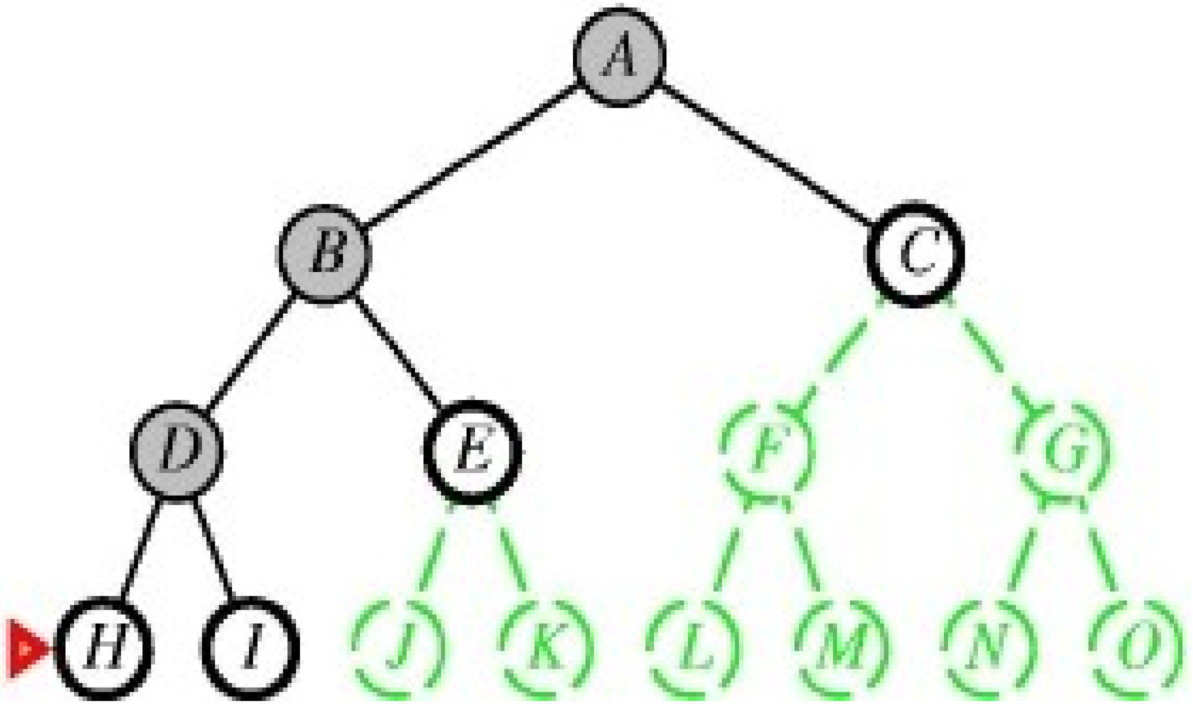
Depth-first search

- Expand deepest unexpanded node
- *fringe* is a LIFO queue, i.e. new nodes go at front



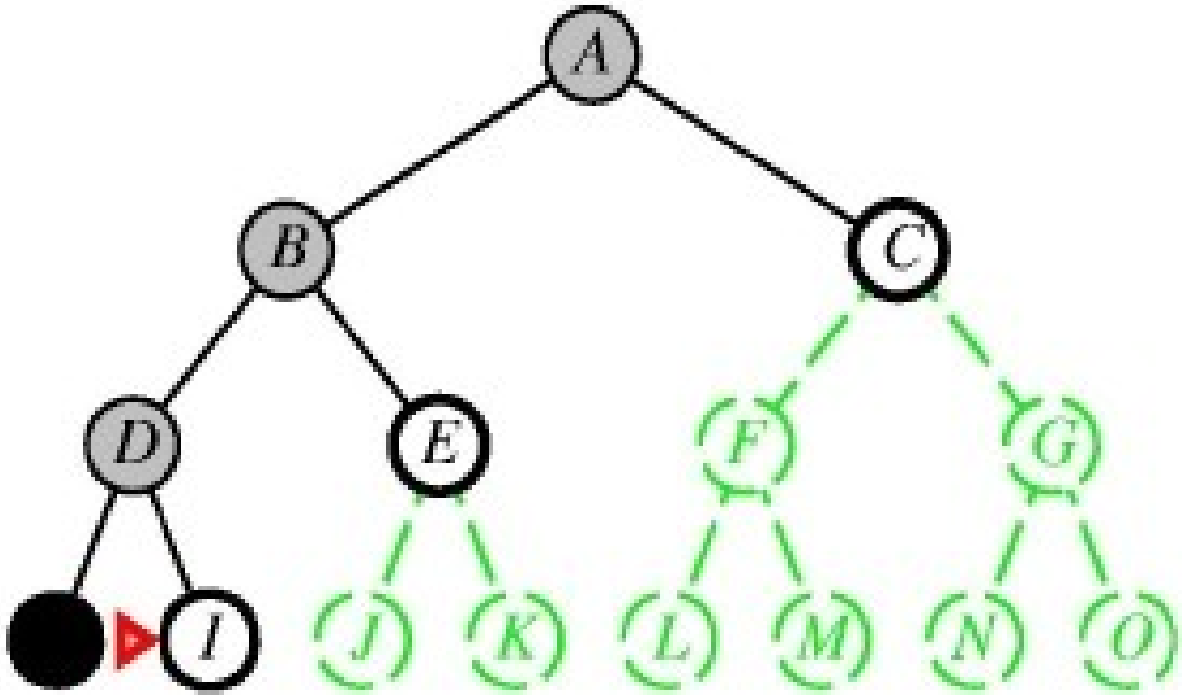
Depth-first search

- Expand deepest unexpanded node
- *fringe* is a LIFO queue, i.e. new nodes go at front



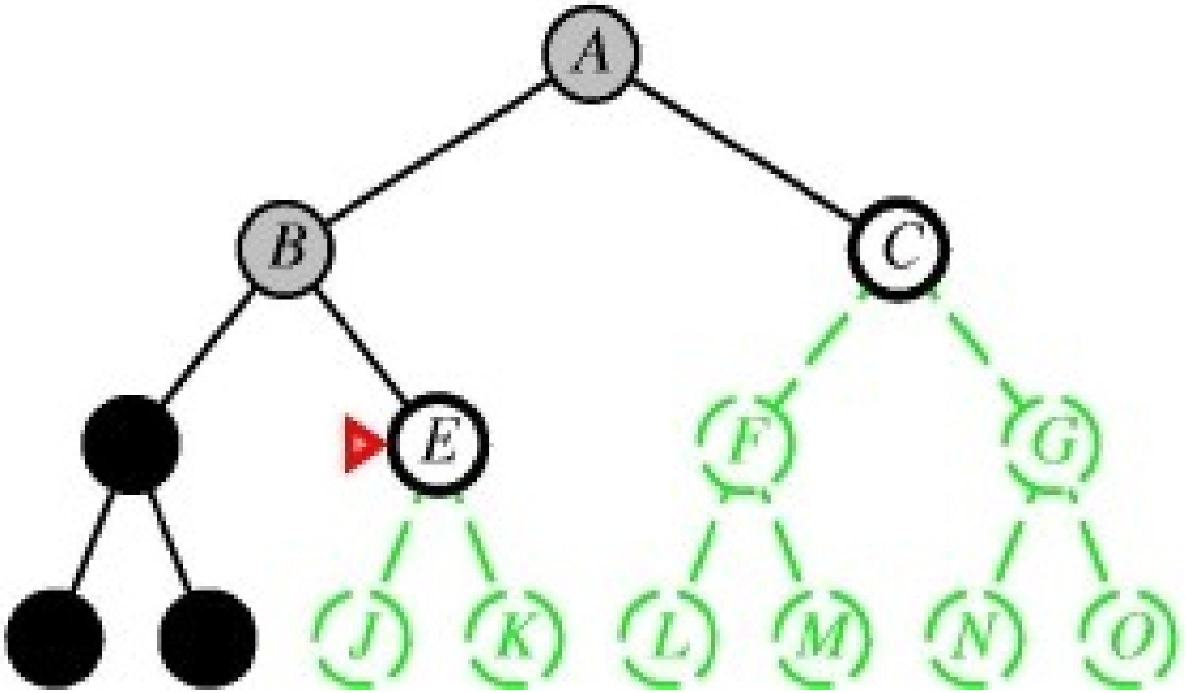
Depth-first search

- Expand deepest unexpanded node
- *fringe* is a LIFO queue, i.e. new nodes go at front



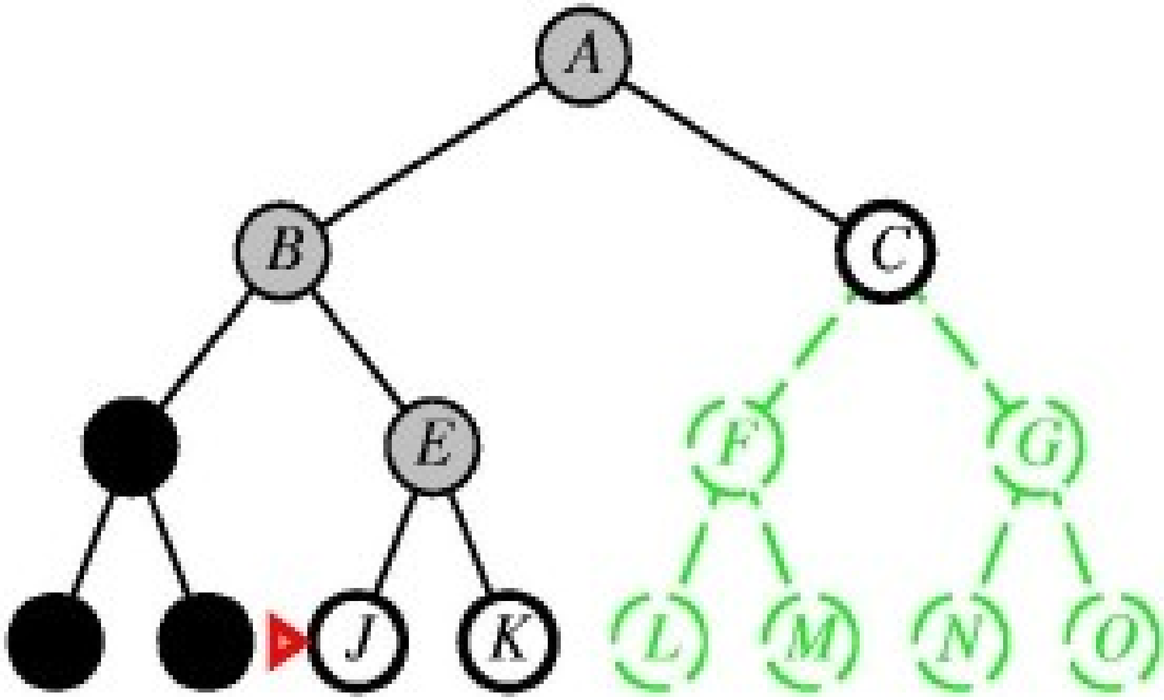
Depth-first search

- Expand deepest unexpanded node
- *fringe* is a LIFO queue, i.e. new nodes go at front



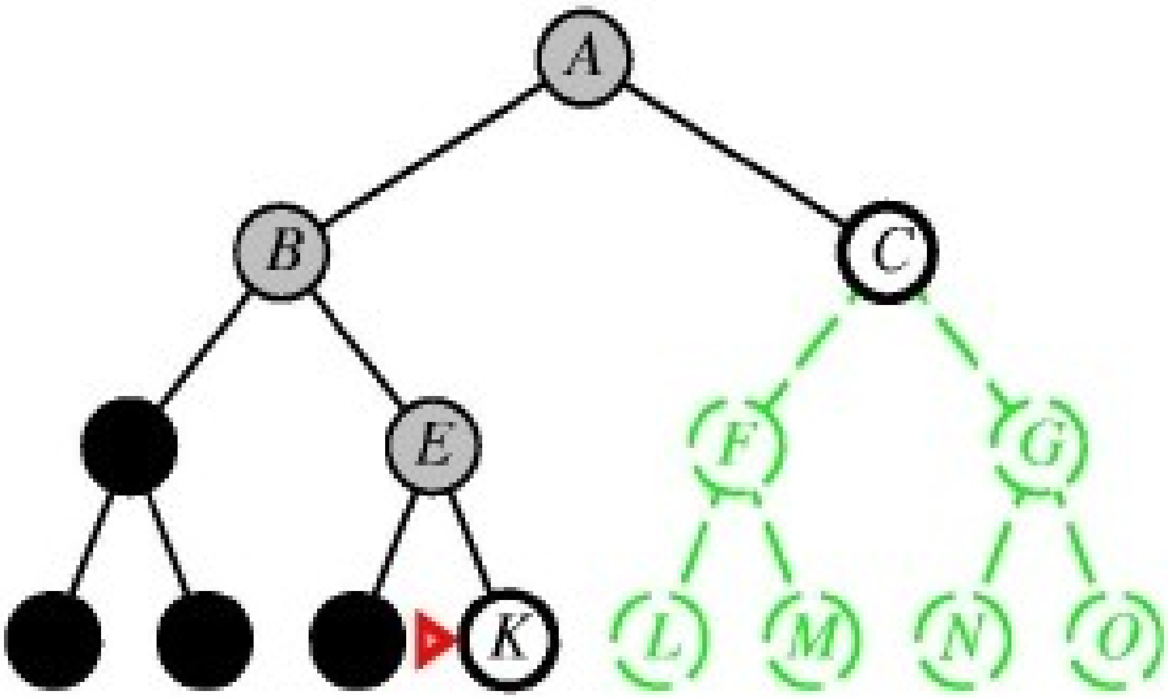
Depth-first search

- Expand deepest unexpanded node
- *fringe* is a LIFO queue, i.e. new nodes go at front



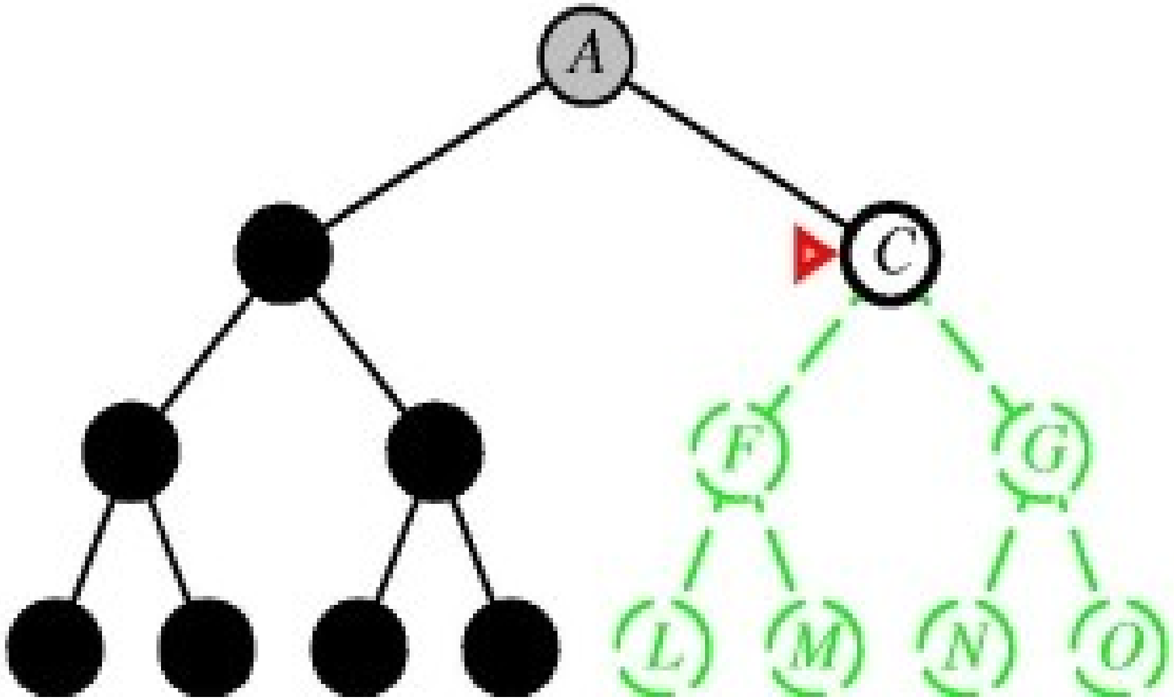
Depth-first search

- Expand deepest unexpanded node
- *fringe* is a LIFO queue, i.e. new nodes go at front



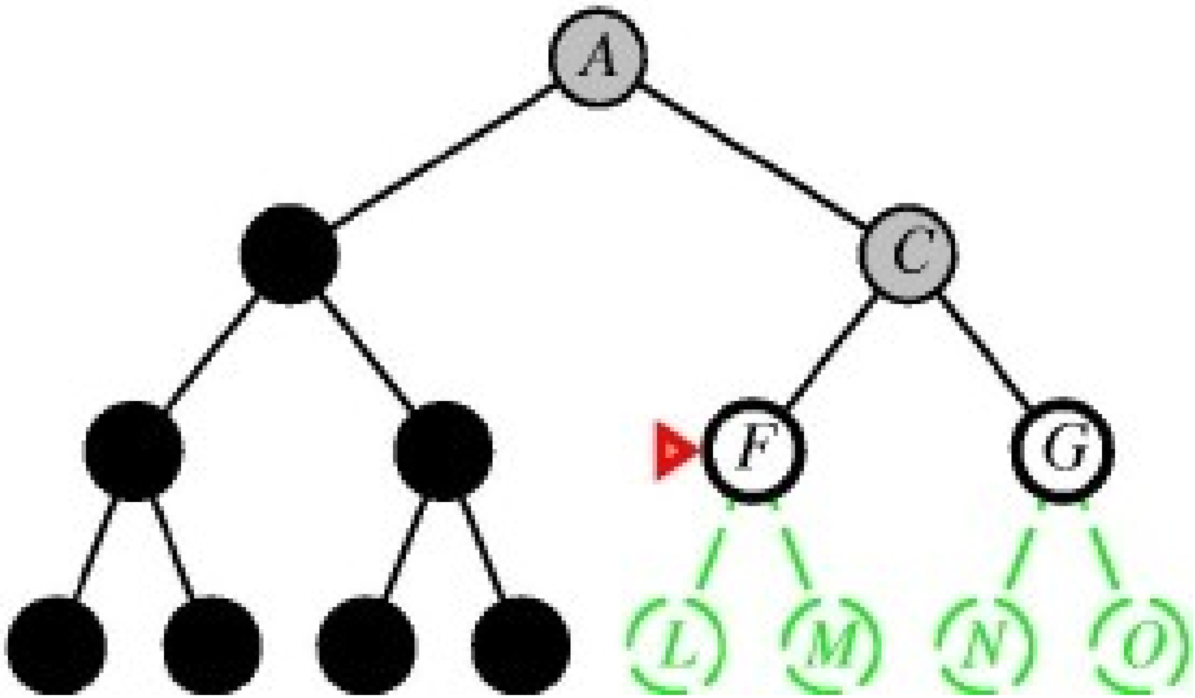
Depth-first search

- Expand deepest unexpanded node
- *fringe* is a LIFO queue, i.e. new nodes go at front



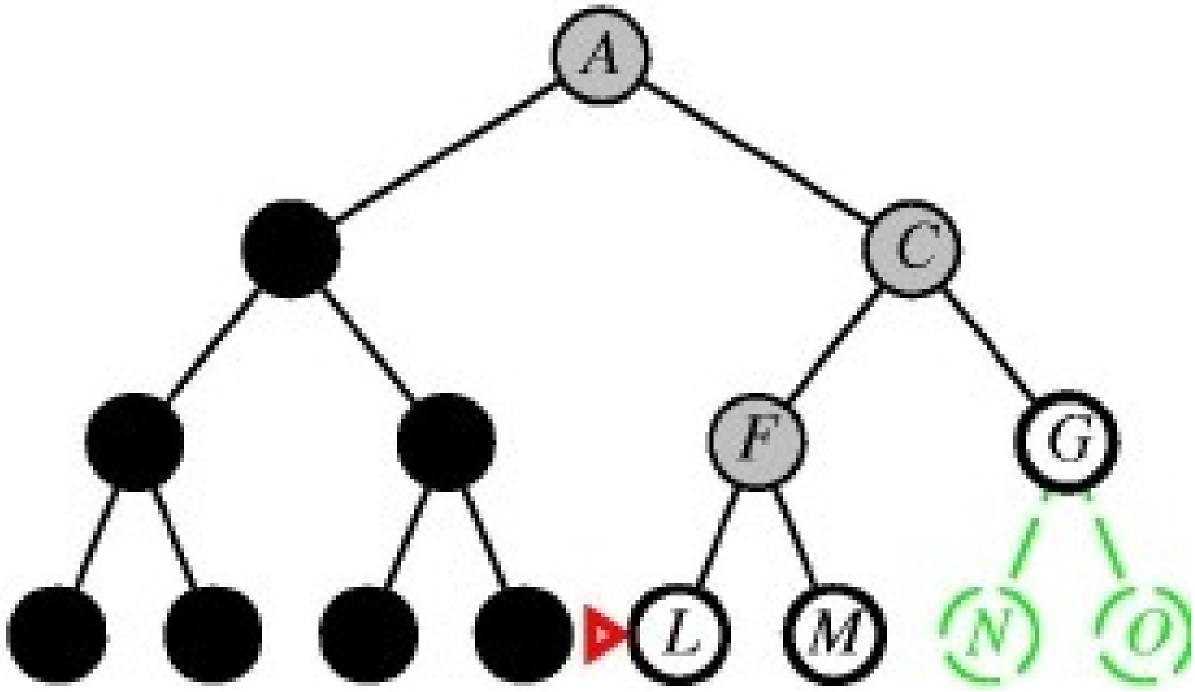
Depth-first search

- Expand deepest unexpanded node
- *fringe* is a LIFO queue, i.e. new nodes go at front



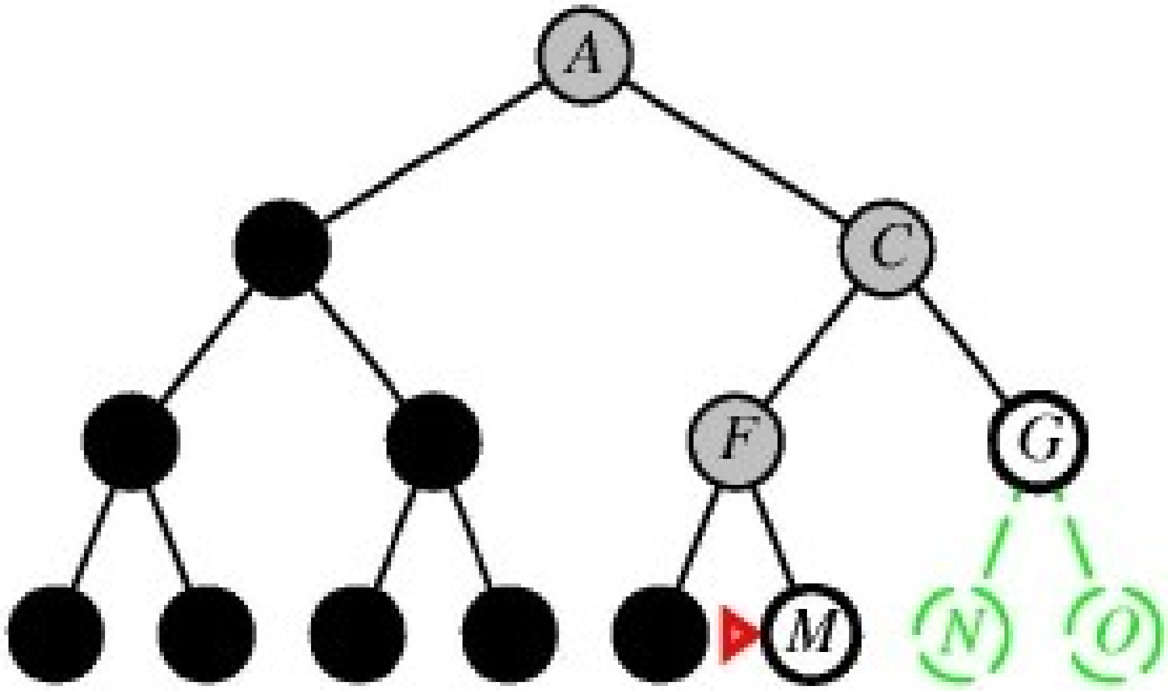
Depth-first search

- Expand deepest unexpanded node
- *fringe* is a LIFO queue, i.e. new nodes go at front



Depth-first search

- Expand deepest unexpanded node
- *fringe* is a LIFO queue, i.e. new nodes go at front



Properties of depth-first search

- Complete?
- Time?
- Space?
- Optimal?

Properties of depth-first search

- Complete?
  - No, fails in infinite spaces, spaces with loops
- Time?
- Space?
- Optimal?

Properties of depth-first search

- Complete?
  - No, fails in infinite spaces, spaces with loops
- Time?
  - $O(b^m)$: terrible if m is much larger than d
- Space?
- Optimal?

Properties of depth-first search

- Complete?
  - No, fails in infinite spaces, spaces with loops
- Time?
  - $O(b^m)$: terrible if m is much larger than d
- Space?
  - $O(bm)$ (single path and unexpanded siblings)
- Optimal?
  - No

Depth-limited search

- Depth-first search with depth limit

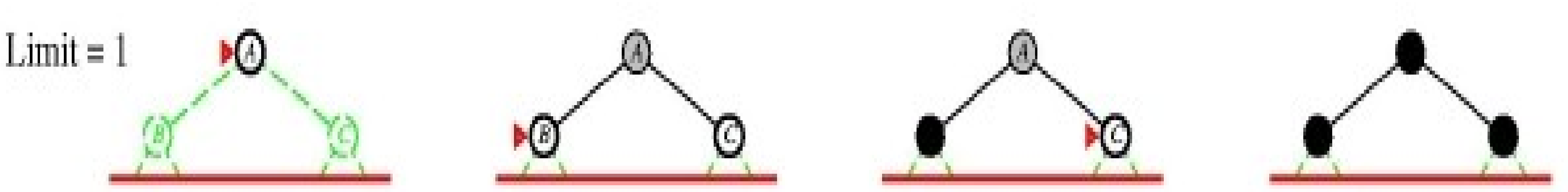
```
function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
   RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

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
   if cutoff-occurred? then return cutoff else return failure
```

Iterative deepening search

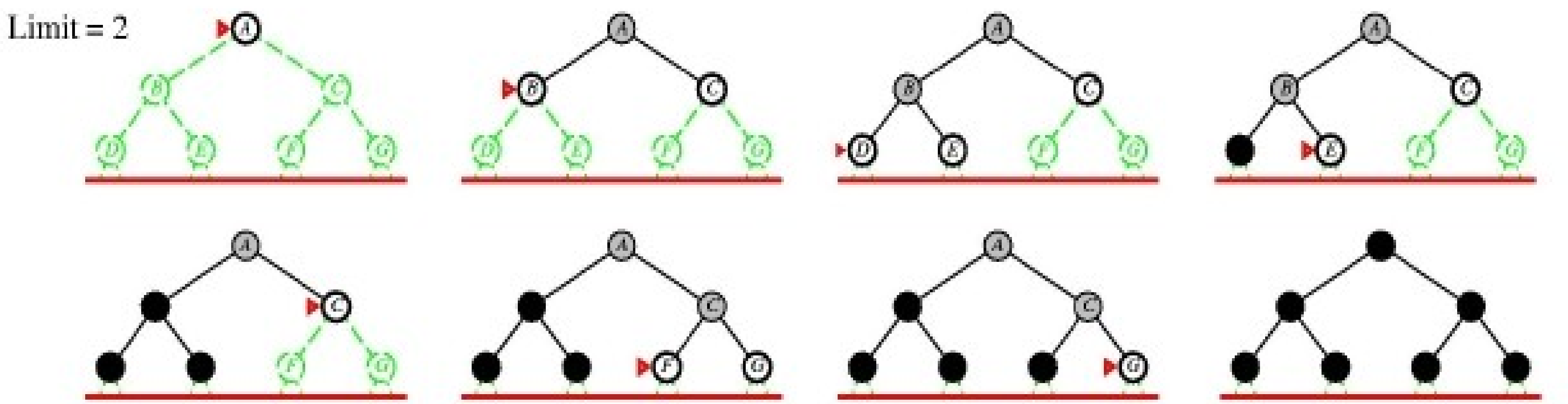
```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution
    inputs: problem, a problem

    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH(problem, depth)
        if result ≠ cutoff then return result
    end
```

Iterative deepening search ($l = 0$)

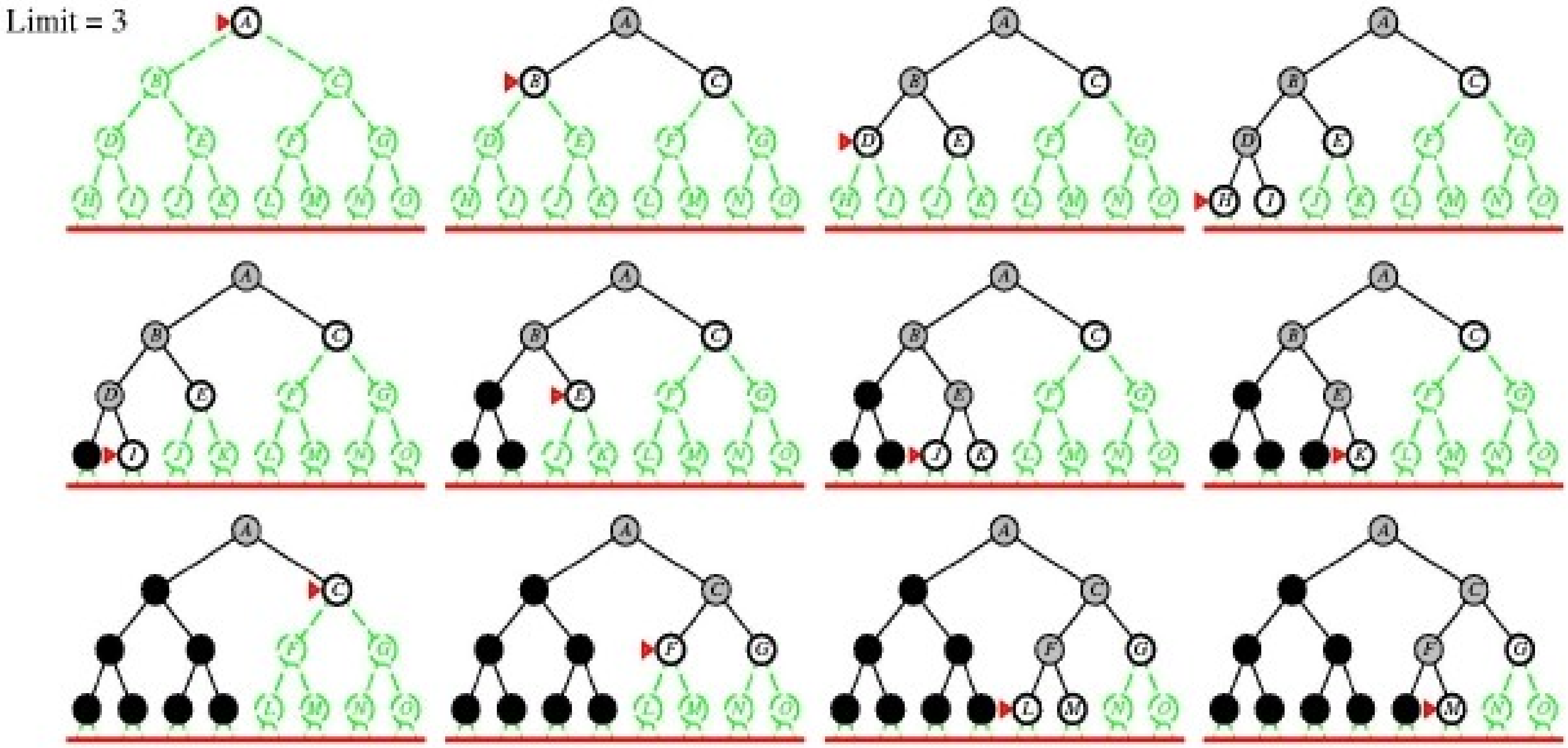
Iterative deepening search ($l = 1$)



Iterative deepening search ($l = 2$)



Iterative deepening search ($l = 3$)



Properties of iter. deepening search

- Complete?
- Time?
- Space?
- Optimal?

Properties of iter. deepening search

- Complete?
  - Yes, if b is finite
- Time?
- Space?
- Optimal?

Properties of iter. deepening search

- Complete?
  - Yes, if b is finite
- Time?
  - $db^1 + (d - 1)b^2 + \ldots + b^d = O(b^d)$
- Space?
- Optimal?

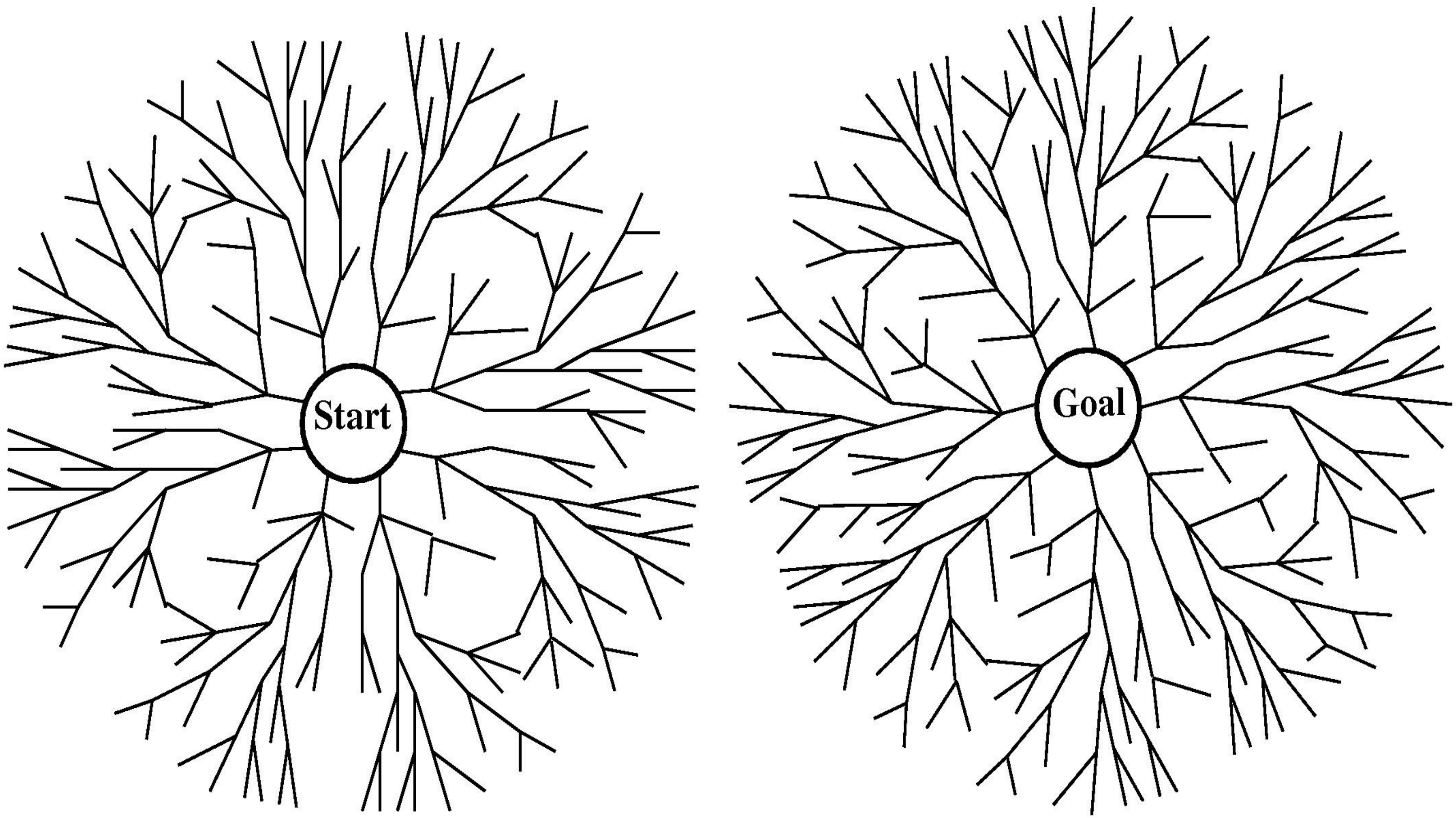
Properties of iter. deepening search

- Complete?
  - Yes, if b is finite
- Time?
  - $db^1 + (d - 1)b^2 + \ldots + b^d = O(b^d)$
- Space?
  - $O(bd)$
- Optimal?

Properties of iter. deepening search

- Complete?
  - Yes, if b is finite
- Time?
  - $db^1 + (d - 1)b^2 + \ldots + b^d = O(b^d)$
- Space?
  - $O(bd)$
- Optimal?
  - Yes, if step cost are all identical

Bidirectional search

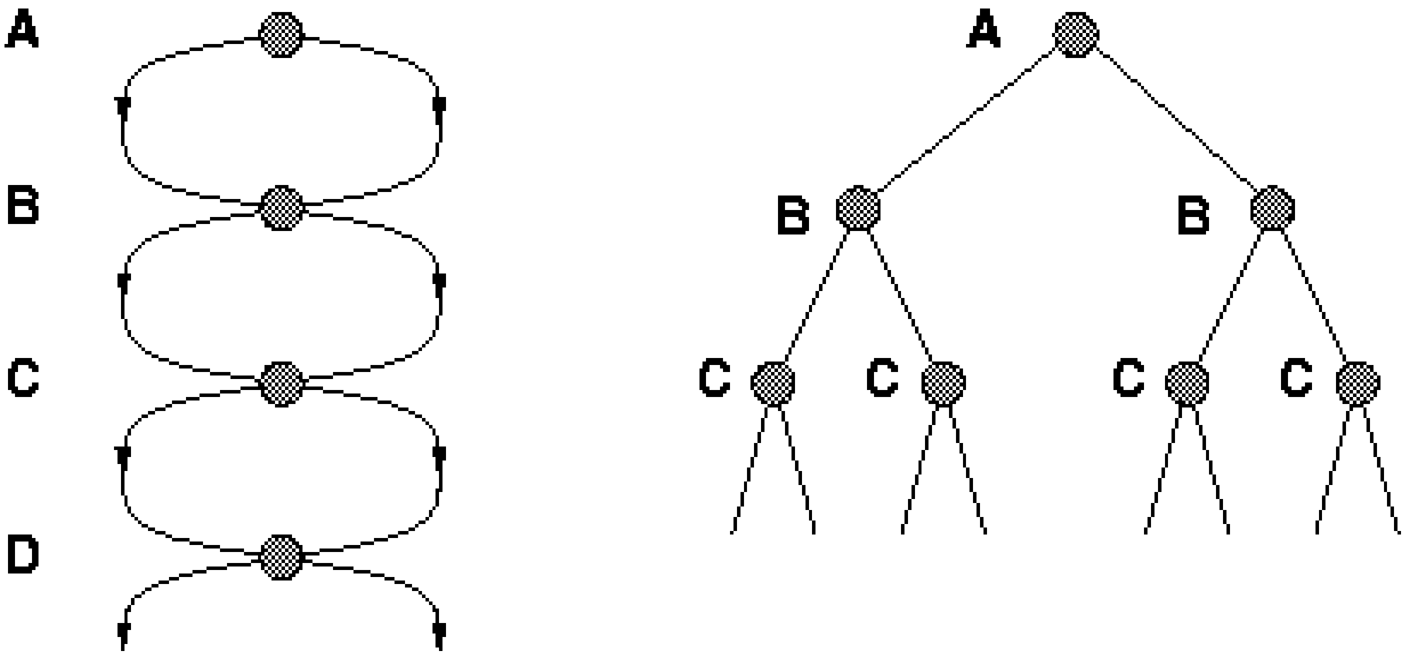


Properties of bidirectional search based on breadth-first search

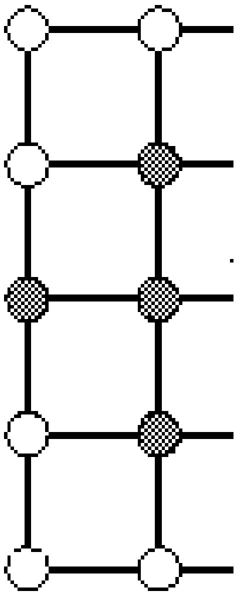
- Complete?
  - Yes, if b is finite
- Time?
  - $O(b^{d/2})$
- Space?
  - $O(b^{d/2})$
- Optimal?
  - Yes, if step costs are all identical

Repeated states

- Failure to detect repeated states can turn a linear problem into an exponential one



(a) (b)



Graph search

```
function GRAPH-SEARCH(problem, fringe) returns a solution, or failure

   closed ← an empty set
   fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
   loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return node
        if STATE[node] is not in closed then
             add STATE[node] to closed
             fringe ← INSERTALL(EXPAND(node, problem), fringe)
   end
```