CSE 604

Artificial Intelligence

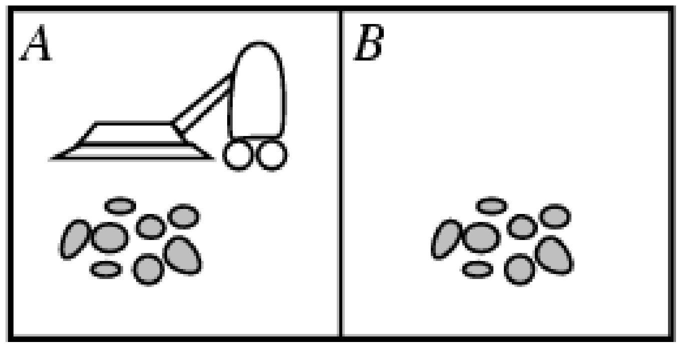
Chapter 3: Solving Problems by Searching

Adapted from slides available in Russell & Norvig's textbook webpage

Dr. Ahmedul Kabir

IIT
University of Dhaka

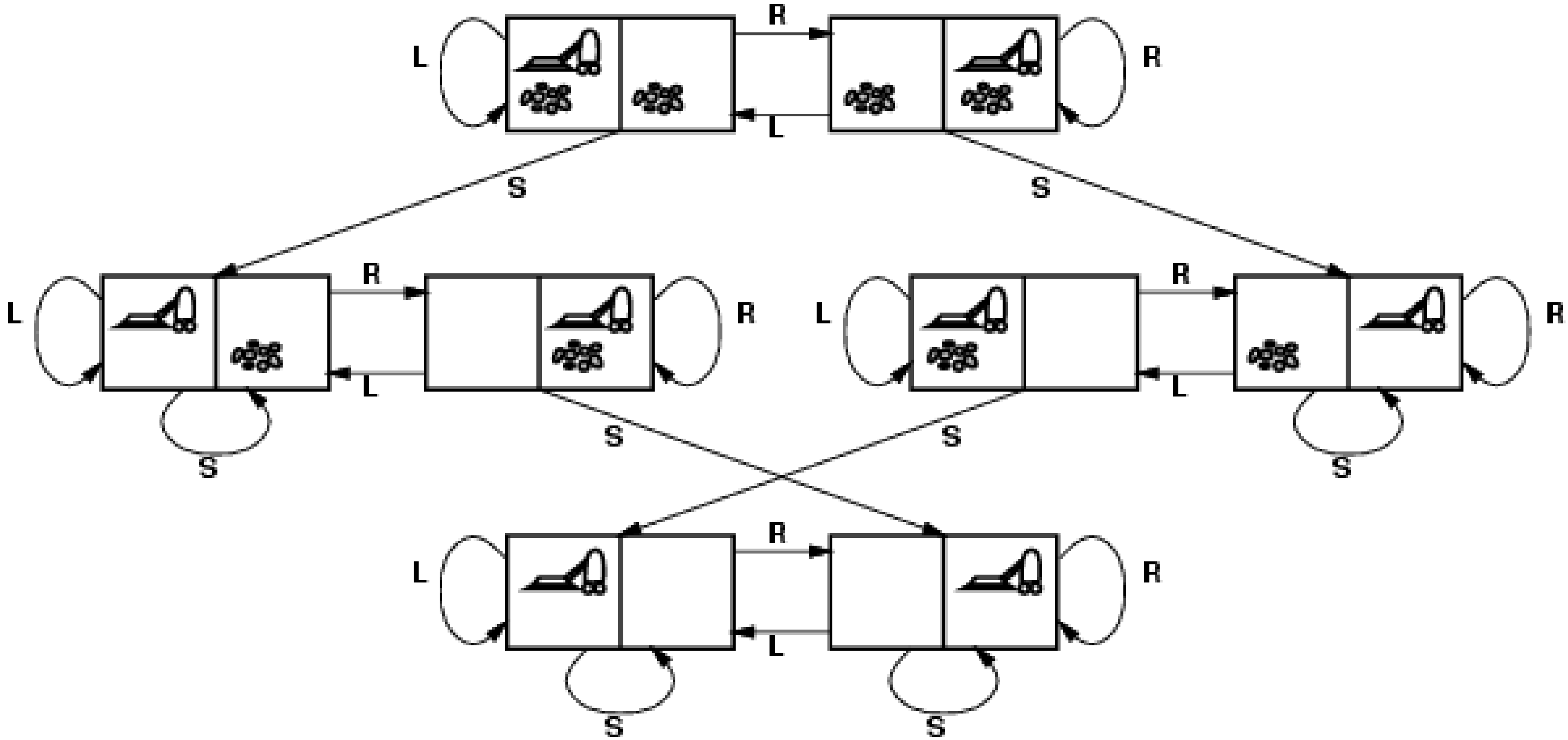
Remember the Vacuum-cleaner world?



- **Percepts**: location and contents, e.g., [A, Dirty]

- **Actions**: *Left*, *Right*, *Suck*

Vacuum world state space graph

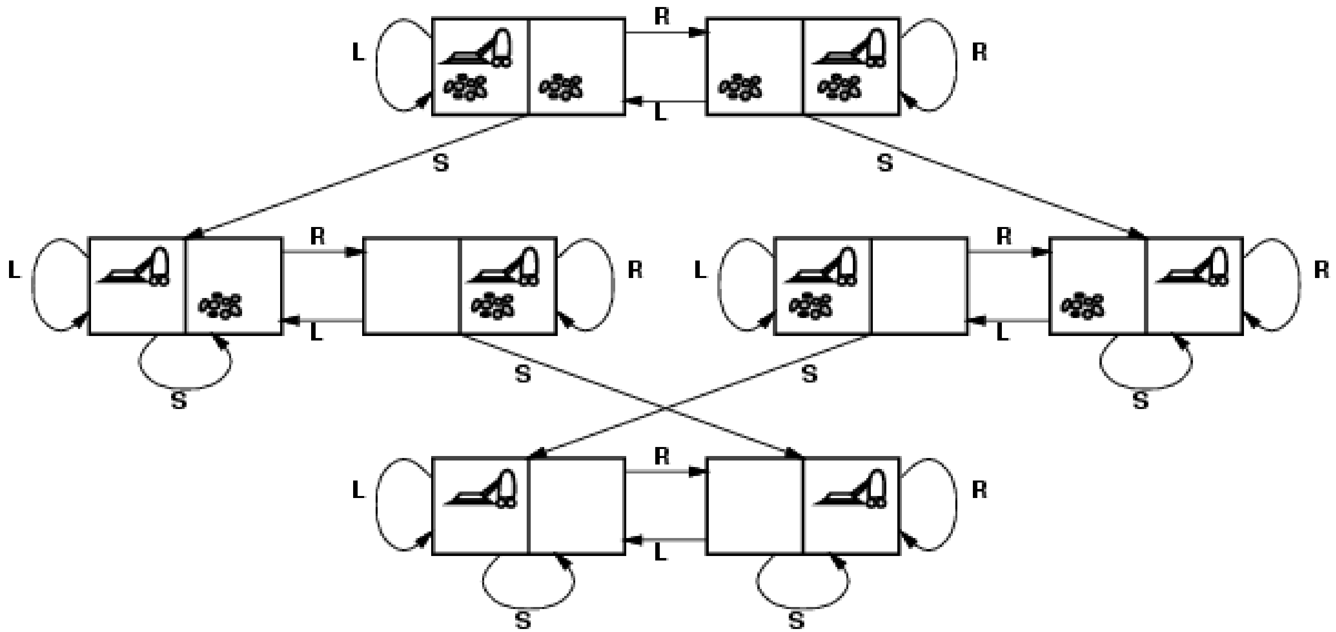


State space: Set of all reachable states. In state space graph, nodes/vertices = states, links/edges = actions

Formulation of a Problem

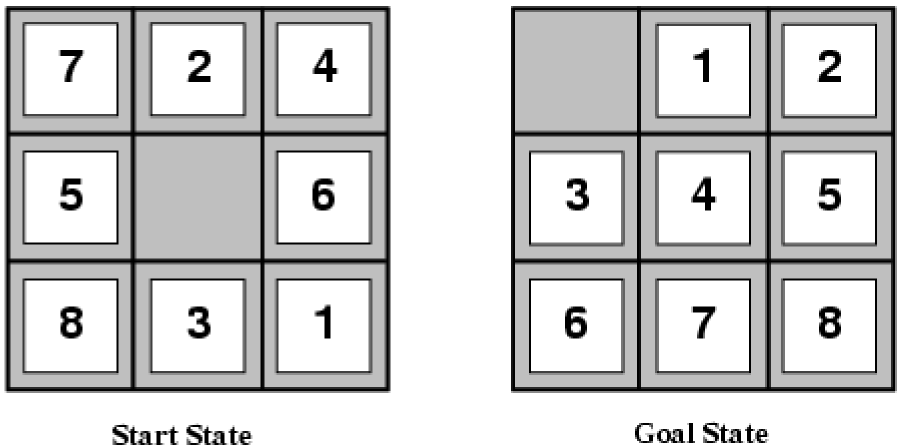
- A Problem is defined by the following items:
  - Set of **states** the agent can be in, with a designated **initial state**
  - Set of **actions** available to the agent
  - **Transition model** describing what each action does (maps a <state, action> pair to a state)
  - **Goal test** which determines if a given state is a goal state
  - A **path cost** function that assigns a numeric cost to each path

Vacuum world state space graph



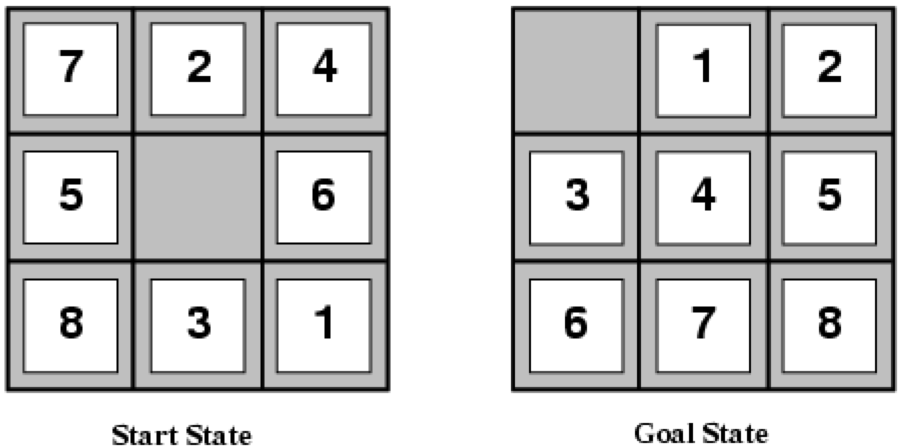
- states? binary dirt and robot location. Any state can be initial state
- actions? *Left*, *Right*, *Suck*
- Transition model? As seen in the state space graph
- goal test? no dirt at all locations
- path cost? 1 per action

Example: The 8-puzzle



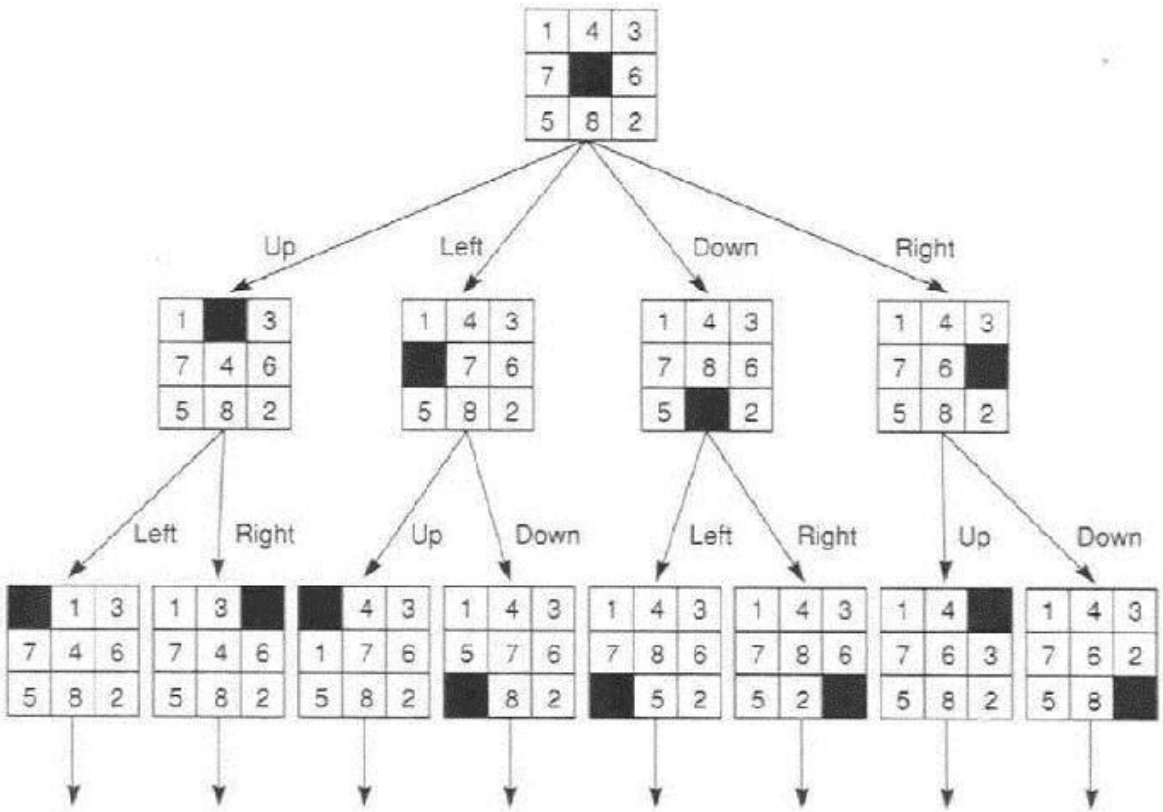
- states?
- actions?
- goal test?
- path cost?

Example: The 8-puzzle



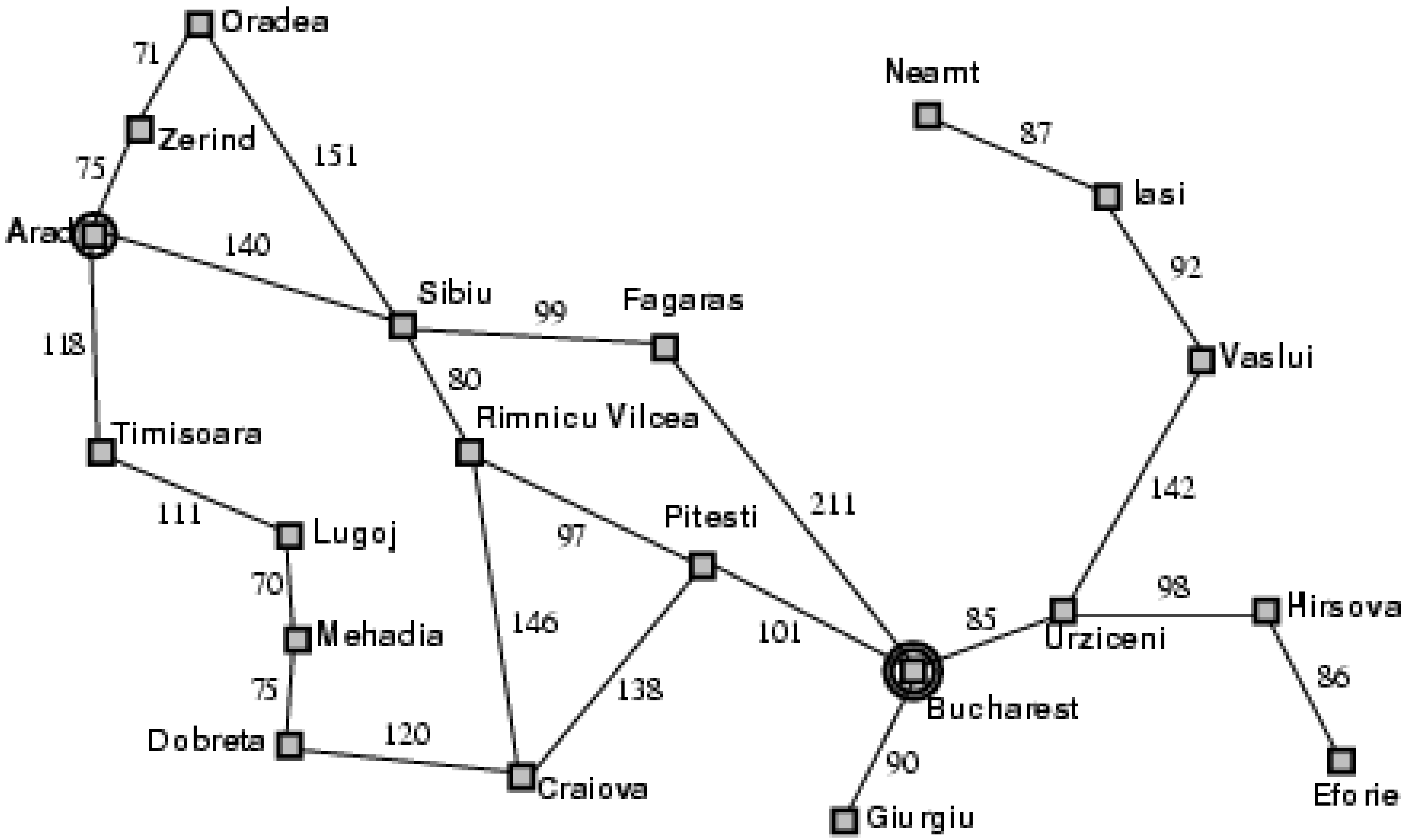
- states? locations of tiles
- actions? move blank left, right, up, down
- goal test? = goal state (given)
- path cost? 1 per move

Example: The 8-puzzle



Partial state space graph

Example: Romania



Search strategies

- A search 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
  - 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 ∞)

Uninformed search strategies

- Uninformed search strategies use only the information available in the problem definition
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search

Basic concept

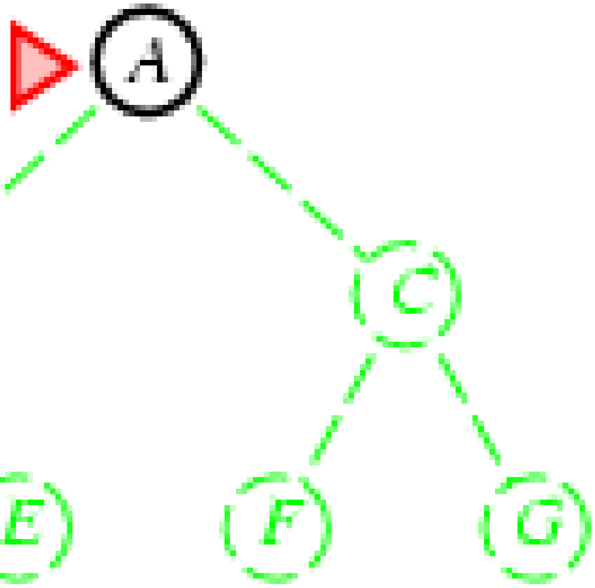
- **Frontier** (or fringe): The set of all leaf nodes available for expansion at any given point

- The basics of each algorithm:
  - Start from initial node
  - Expand adjacent nodes and put them in the frontier
  - Choose the next node from the frontier for expansion
  - Repeat until goal is found, or some ending criteria is met

- The algorithms differ in the way they choose the next node from the frontier

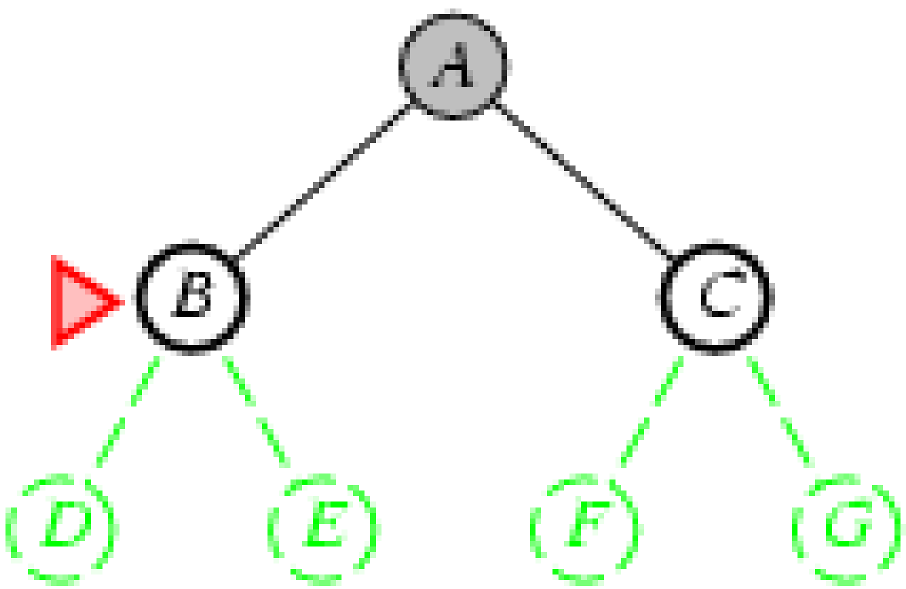
Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *frontier* is a FIFO queue, i.e., new successors go at end



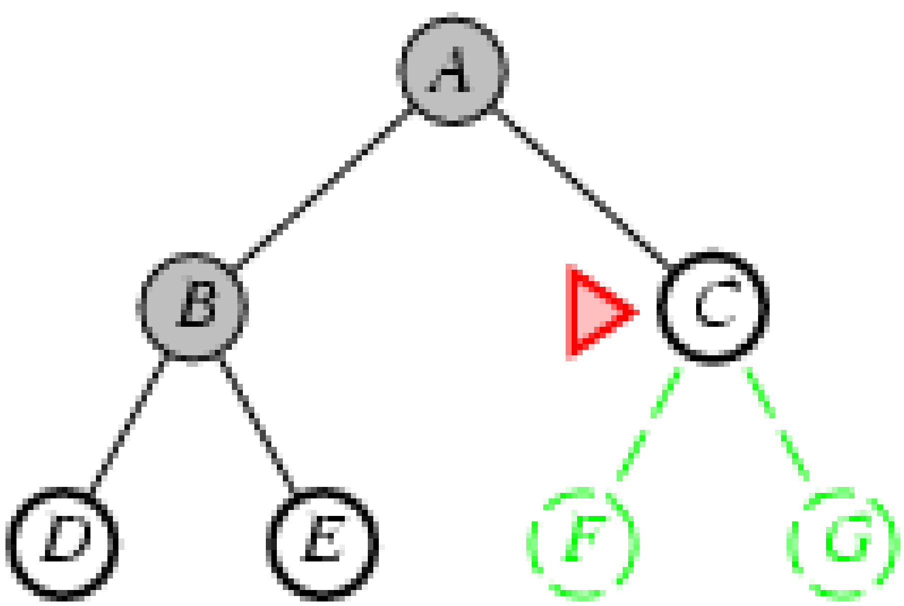
Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *frontier* is a FIFO queue, i.e., new successors go at end



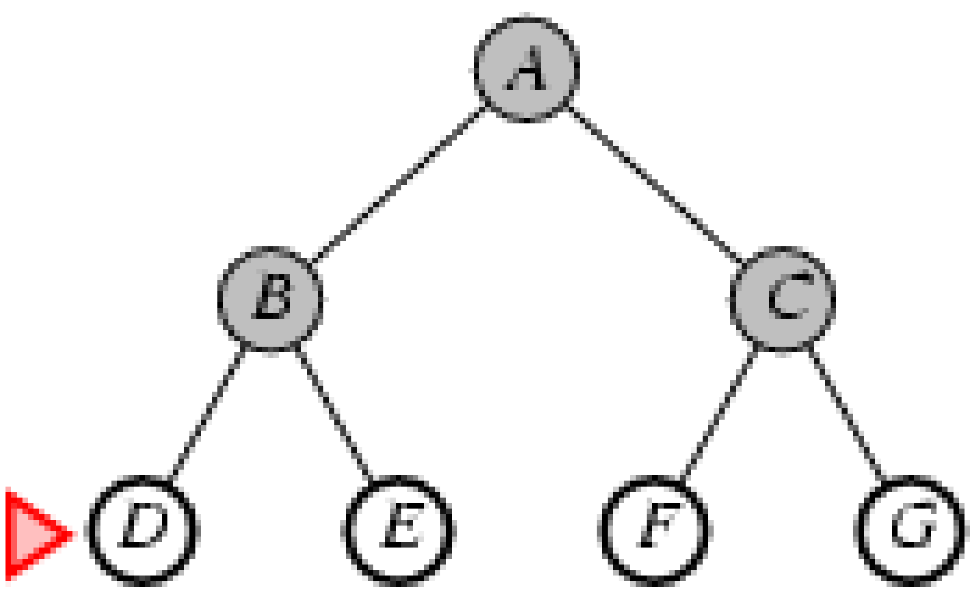
Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *frontier* is a FIFO queue, i.e., new successors go at end



Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *frontier* is a FIFO queue, i.e., new successors go at end



Properties of breadth-first search

- Complete? Yes (if b is finite)
- Time? $1+b+b^2+b^3+\ldots+b^d = O(b^d)$
- Space? $O(b^d)$ (keeps every node in memory)
- Optimal? Yes (if cost = 1 per step)
- Space is the bigger problem (more than time)

Uniform-cost search

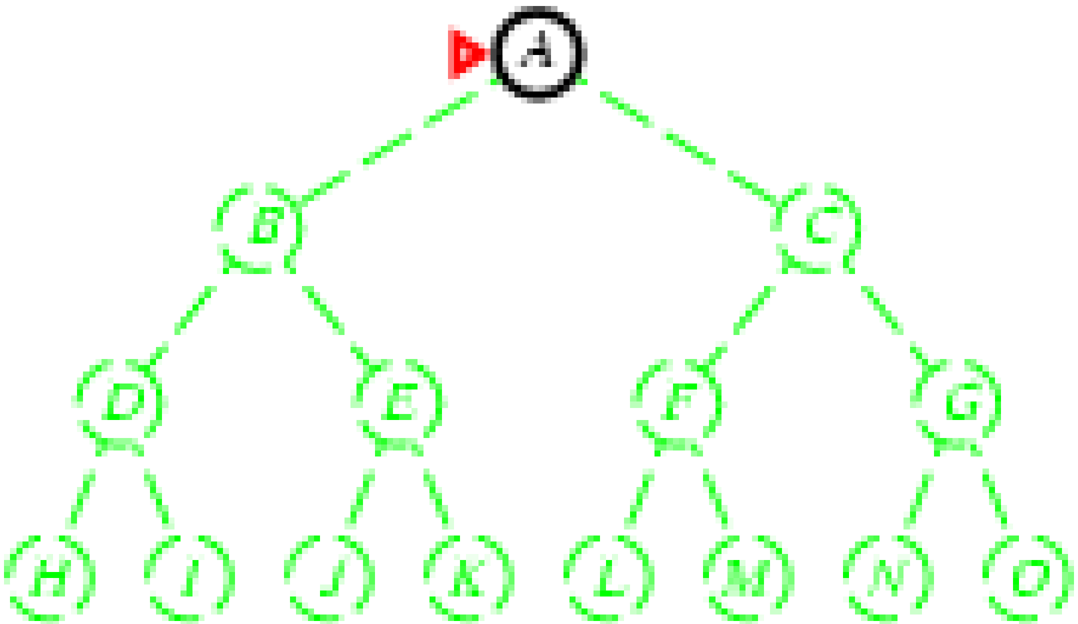
- Expand least-cost unexpanded node
- Implementation:
  - *frontier* = queue ordered by path cost

- Equivalent to breadth-first if step costs all equal

- Complete? Yes, if step cost ≥ ε
- Time? # of nodes with $g \leq$ cost of optimal solution, $O(b^{ceiling(C^*/\varepsilon)})$ where C^* is the cost of the optimal solution
- Space? # of nodes with $g \leq$ cost of optimal solution, $O(b^{ceiling(C^*/\varepsilon)})$
- Optimal? Yes – nodes expanded in increasing order of $g(n)$

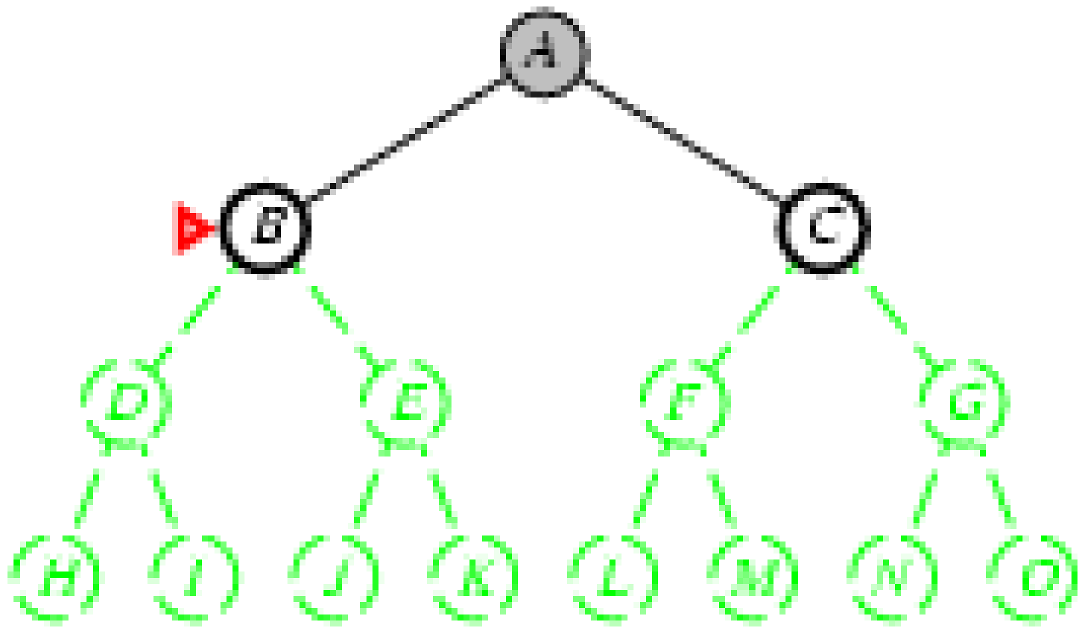
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO stack, i.e., put successors at front



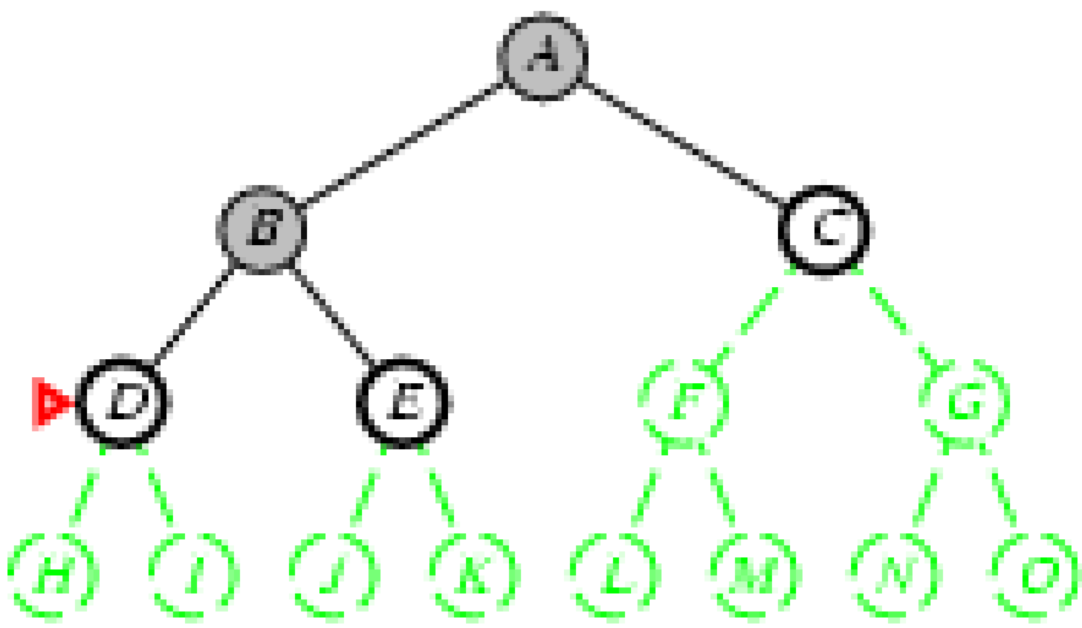
Depth-first search

- Expand deepest unexpanded node
- Implementation:
    - *frontier* = LIFO stack, i.e., put successors at front



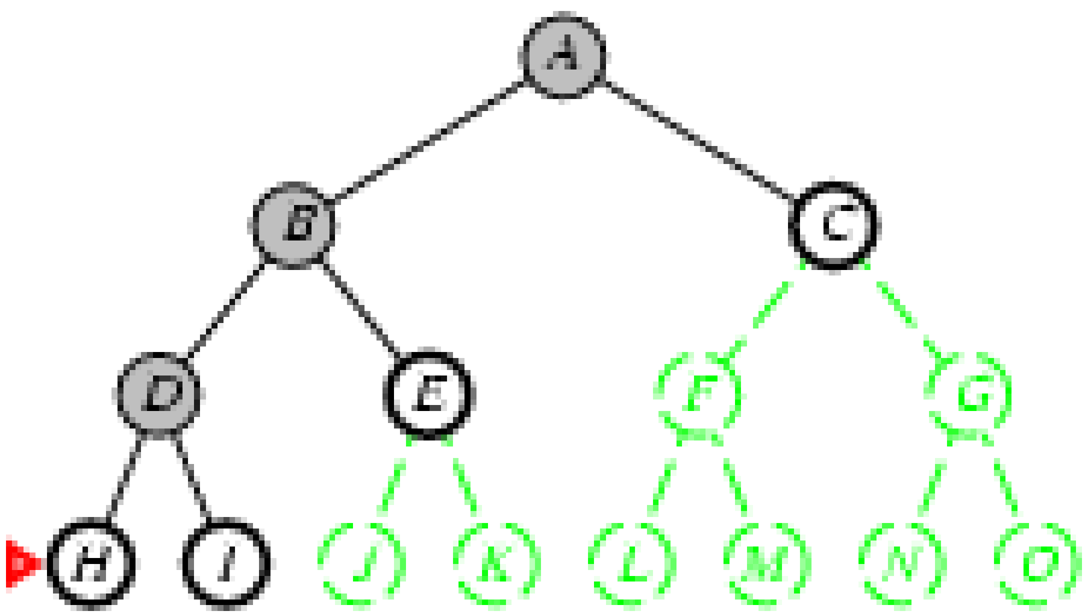
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO stack, i.e., put successors at front



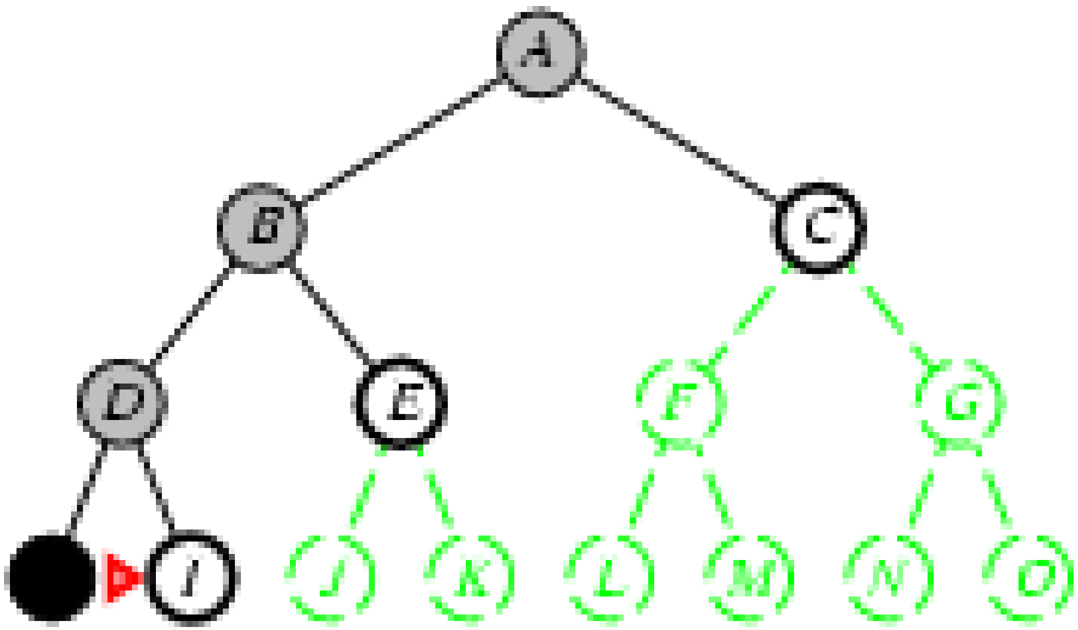
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO stack, i.e., put successors at front



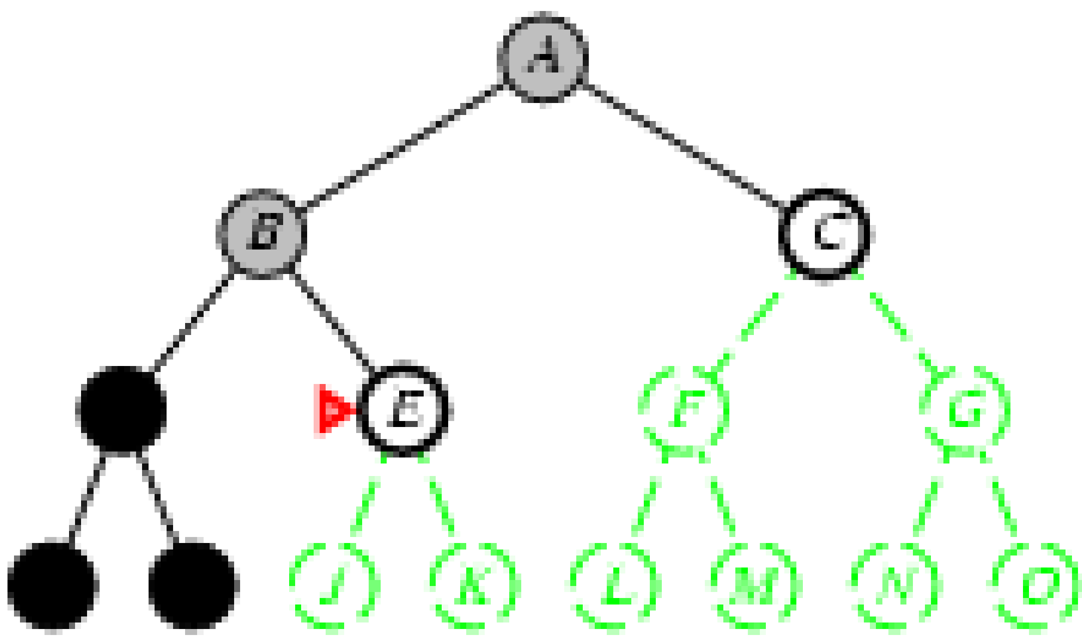
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO stack, i.e., put successors at front



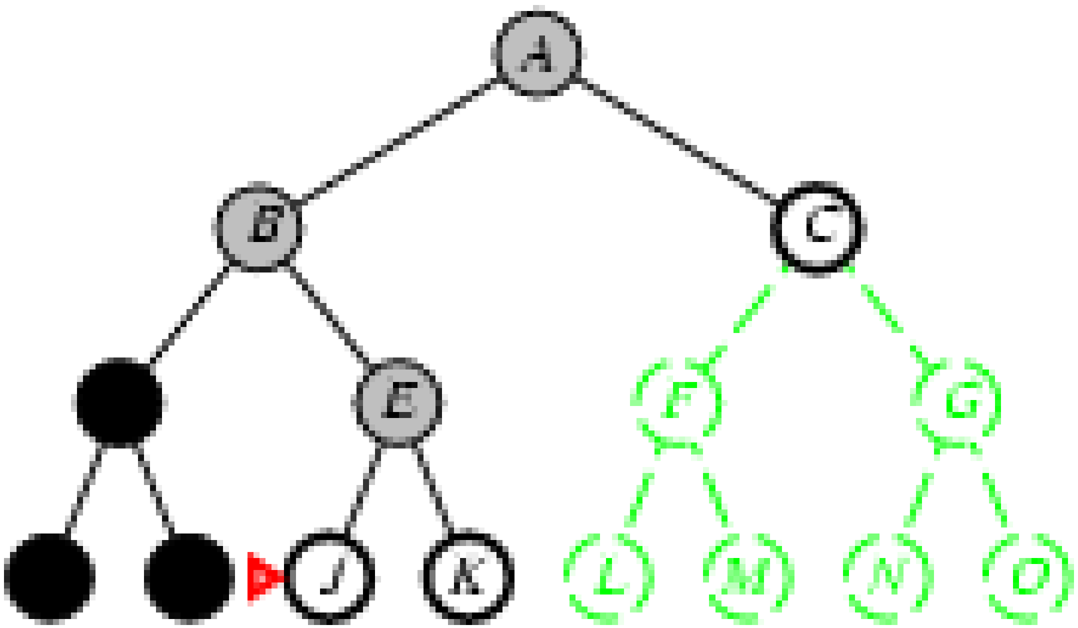
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO stack, i.e., put successors at front



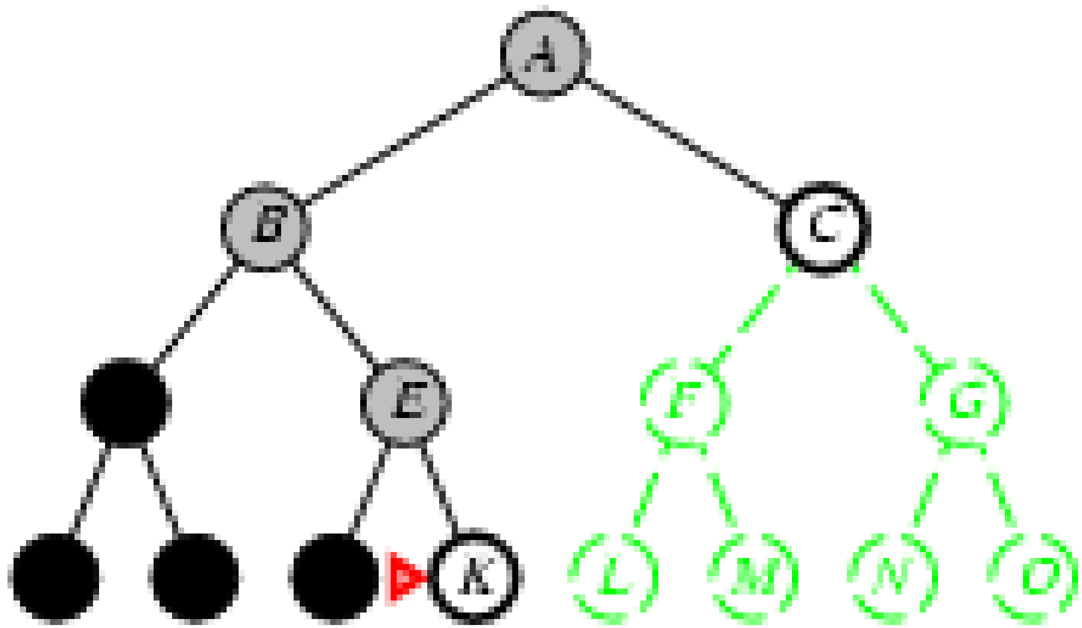
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO stack, i.e., put successors at front



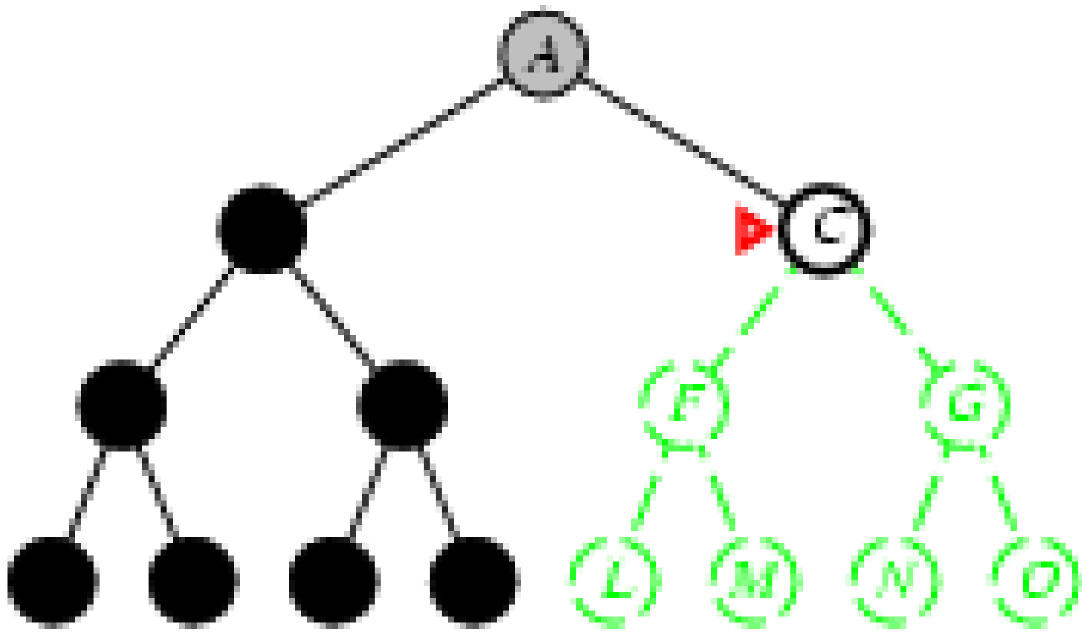
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO stack, i.e., put successors at front



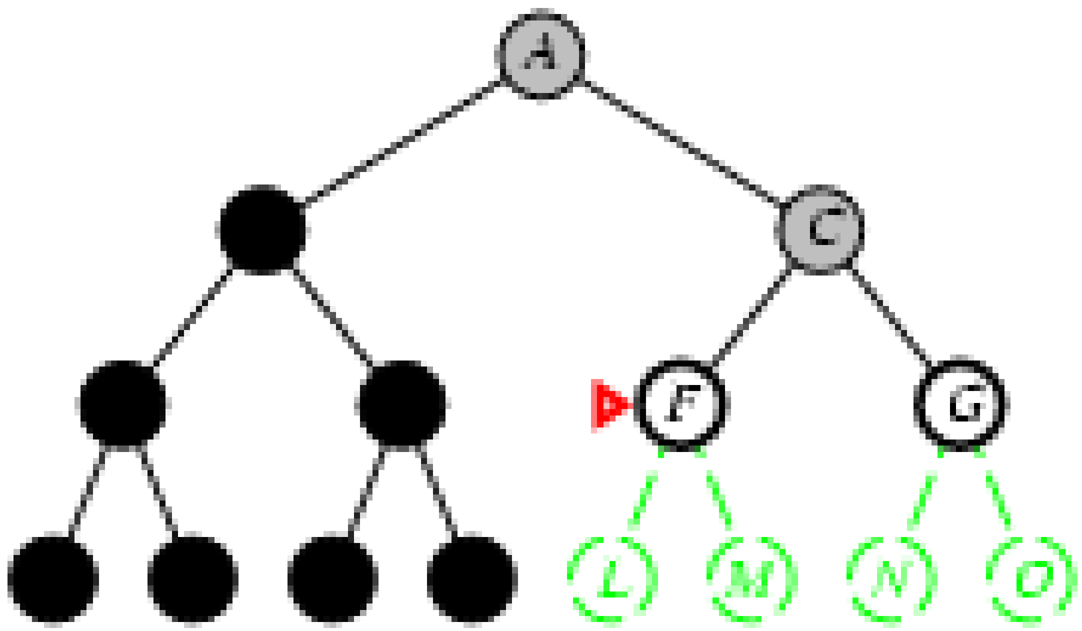
Depth-first search

- Expand deepest unexpanded node
- Implementation:
    - *frontier* = LIFO stack, i.e., put successors at front



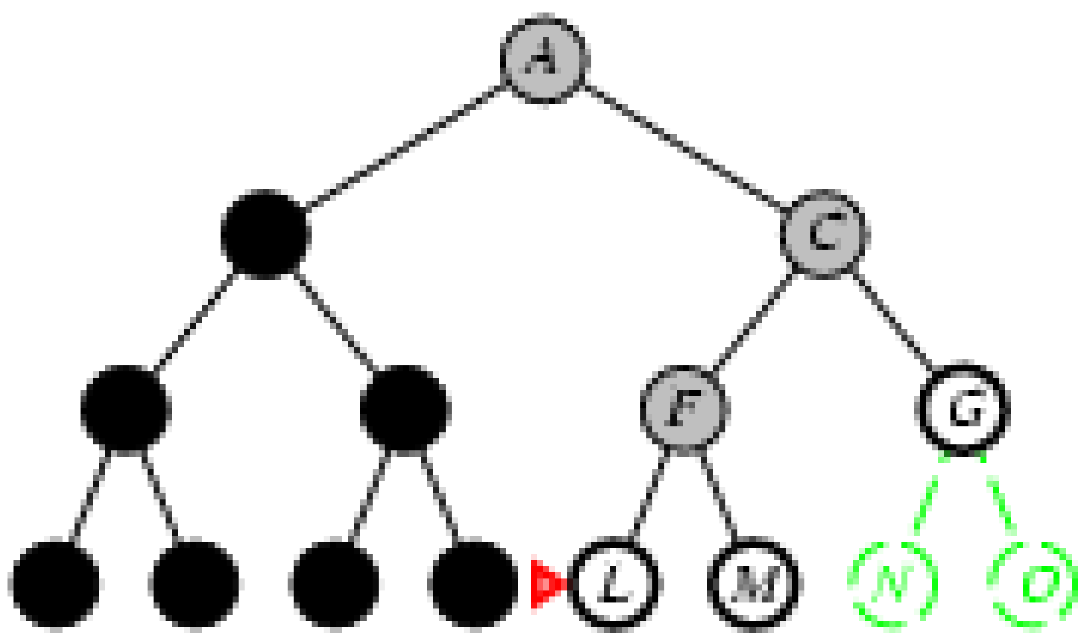
Depth-first search

- Expand deepest unexpanded node
- Implementation:
    - *frontier* = LIFO stack, i.e., put successors at front



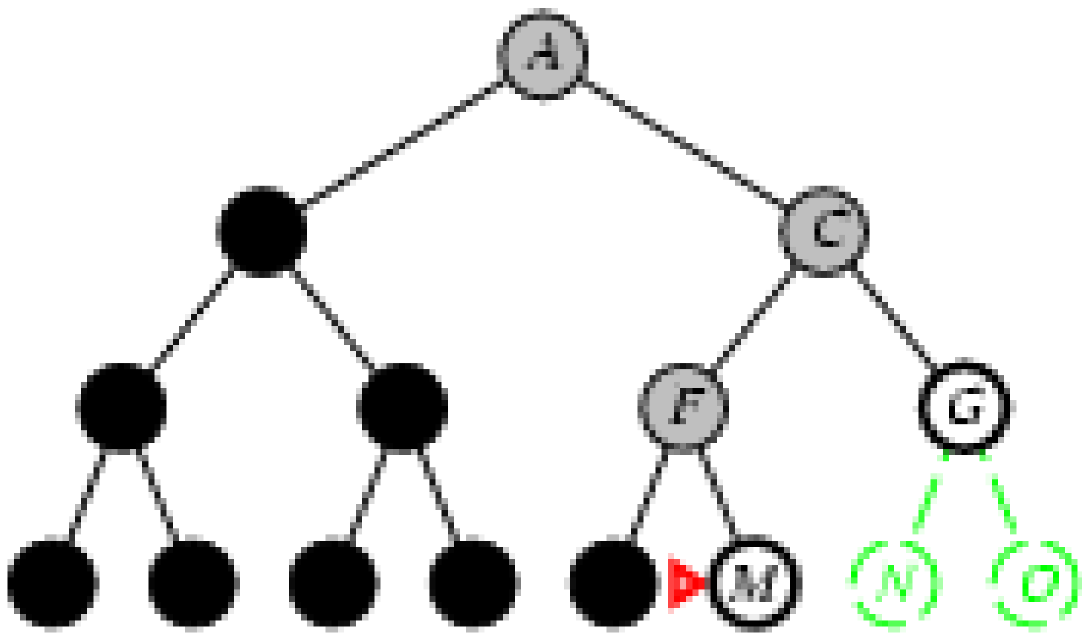
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO stack, i.e., put successors at front



Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO stack, 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
    → 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

Depth-limited search

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

- Complete? No
- Time? $O(b^l)$
- Space? $O(bl)$
- Optimal? No

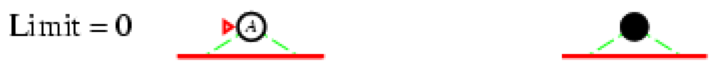
Iterative deepening search

= depth-limited search on repeat!

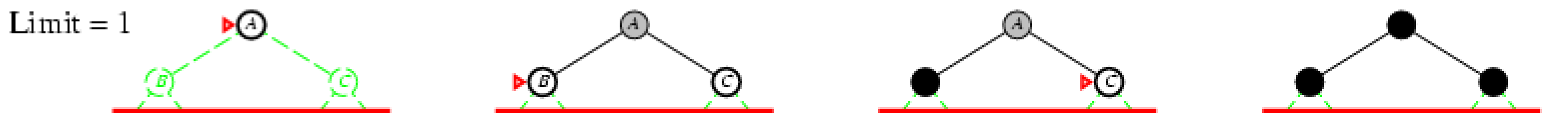
Limit *l* is increased at each iteration until goal is found

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution, or fail-
ure
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
```

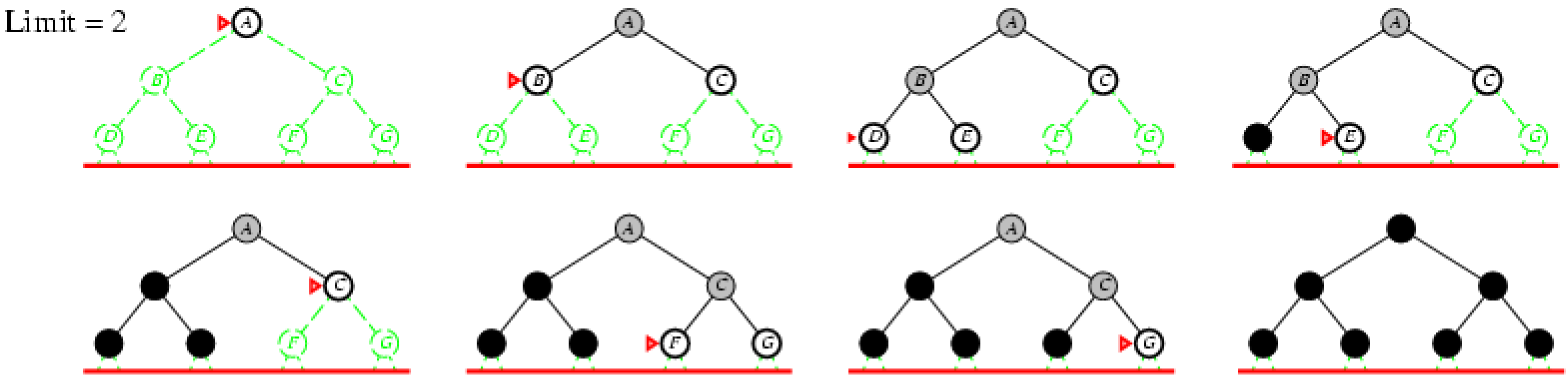
Iterative deepening search $l = 0$



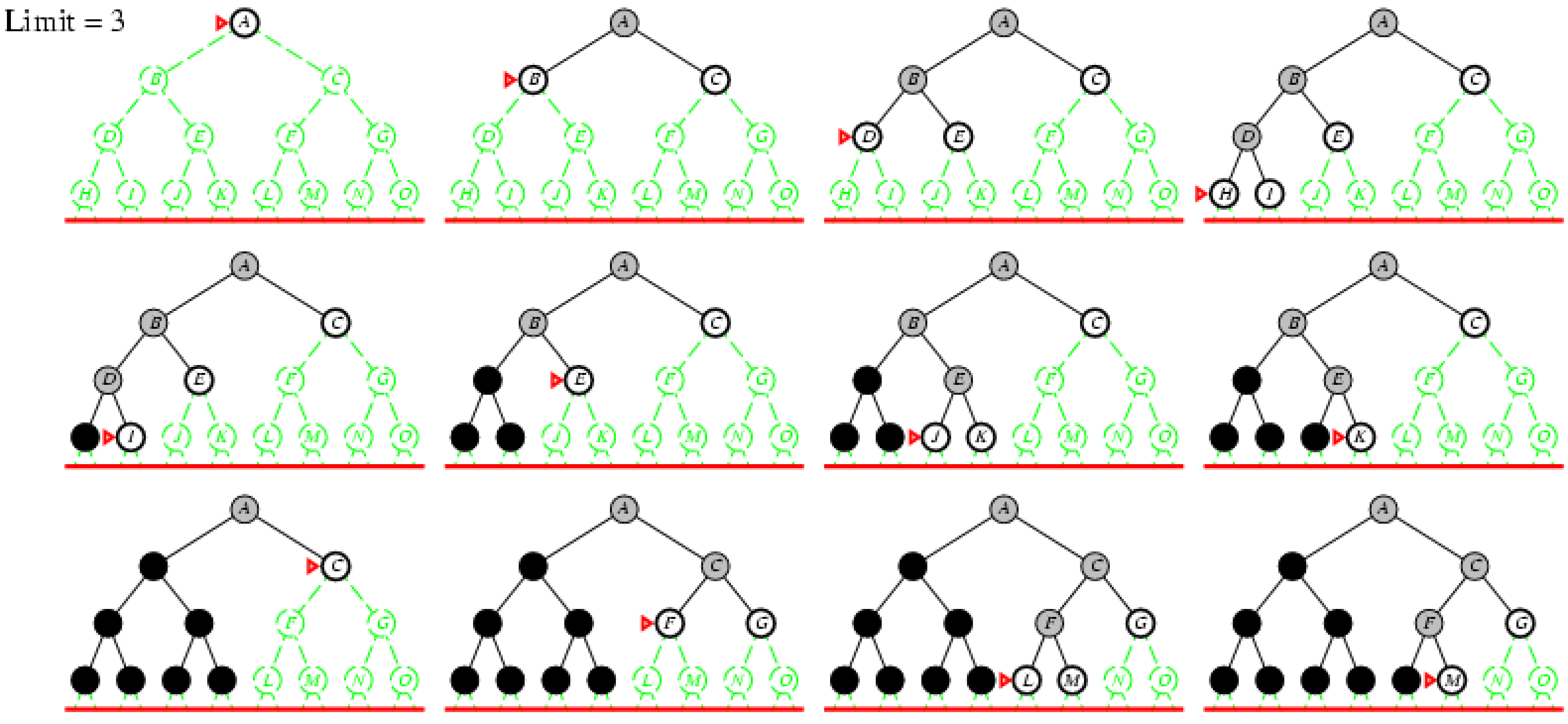
Iterative deepening search $l=1$



Iterative deepening search l =2



Iterative deepening search l =3



Properties of iterative deepening search

- Complete? Yes
- Time? $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1

Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$
Optimal?	Yes[c]	Yes	No	No	Yes[c]