Artificial Intelligence- Problem Solving By Search

Amir Aavani
Amir@Aavani.net
http://Amir.Aavani.net/CS/AI-86

19. Oktober 2007

Outline

- What is Problem-Solving Agent?
- Problem Types
- Problem Formulation
- Basic Search Algorithms

Problem-Solving Agent

- It is a kind of Goal-based Agents
- 4 general steps in problem-solving:
  1. Goal Formulation
  2. Problem Formulation
  3. Search
  4. Execute
- Driving from Arad to Bucharest ...

Problem-Solving Agent- Goal Formulation

- Declaring the Goal.
- Ignoring the some actions.
- Limits the objective that agent is trying to achieve.
- Goal can be defined as set of world states.

Problem-Solving Agent- Problem Formulation

- What actions and states to consider given the goal
- It can change the world states size, enormously.
- It depends on how the agent is connected to its environment.

Problem-Solving Agent- Search

- Now, our agent knows the goal and acion(s).
- Which action to choose (with map or with-out map)?
- What action should be chosen in a state with unknown value?
- Process of looking for such a sequence is called **Search**.
- **Search Alg** takes problem as input and returns a solution in form of an action sequence.

Problem-Solving Agent- Execute

- After finding a suitable action sequence, the actions can be carried out.
- To solve a problem: "Formulate, Search, Execute".

Structure of Intelligent Agent, Cont

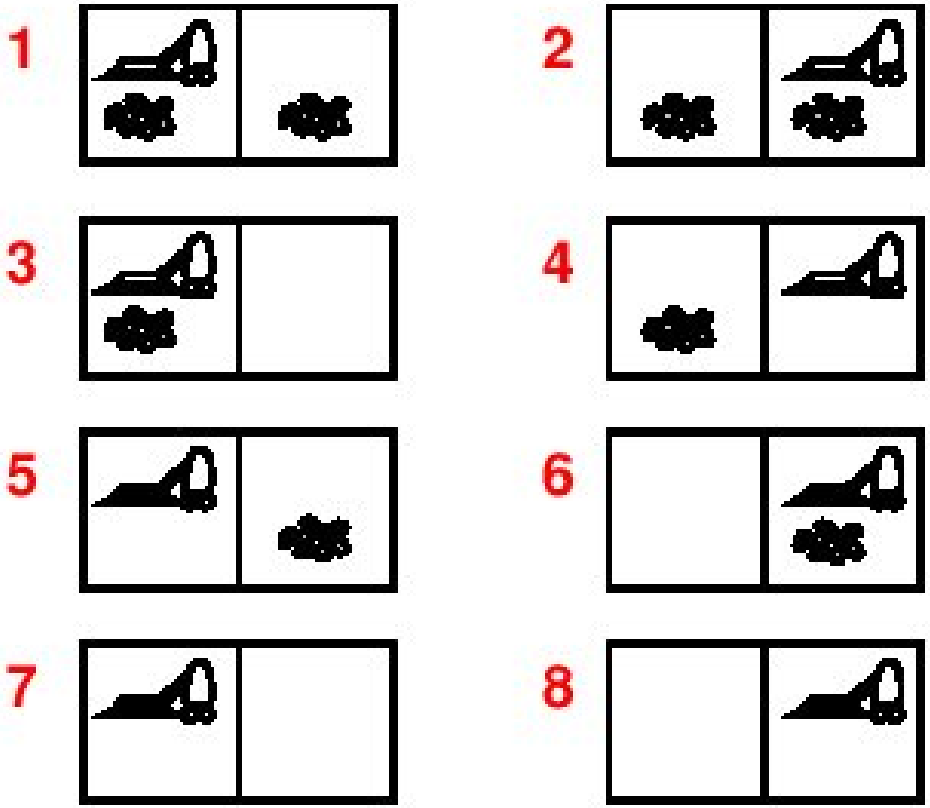
```
function SIMPLE-PROBLEM-SOLVER-AGENT (percept) return actio
  s, an action sequence, initially empty
  state, some description of the current world state
  g, a goal, initially null
  problem, a problem formulation

  state= UPDATE-STATE (state, percept)
  if s is empty then
    g= FORMULATE-GOAL (state);
    problem= FORMULATE-PROBLEM (state, g)
    s= SEARCH (problem)
  action= RECOMMENDATION (state, g)
  s= REMAINDER (s, state)
  return action
```

Formulating Problems

- There are four types of problems:
  1. Single-state problems
  2. Multiple-state problems
  3. Contigency problems
  4. Exploration problems

Formulating Problems, Cont



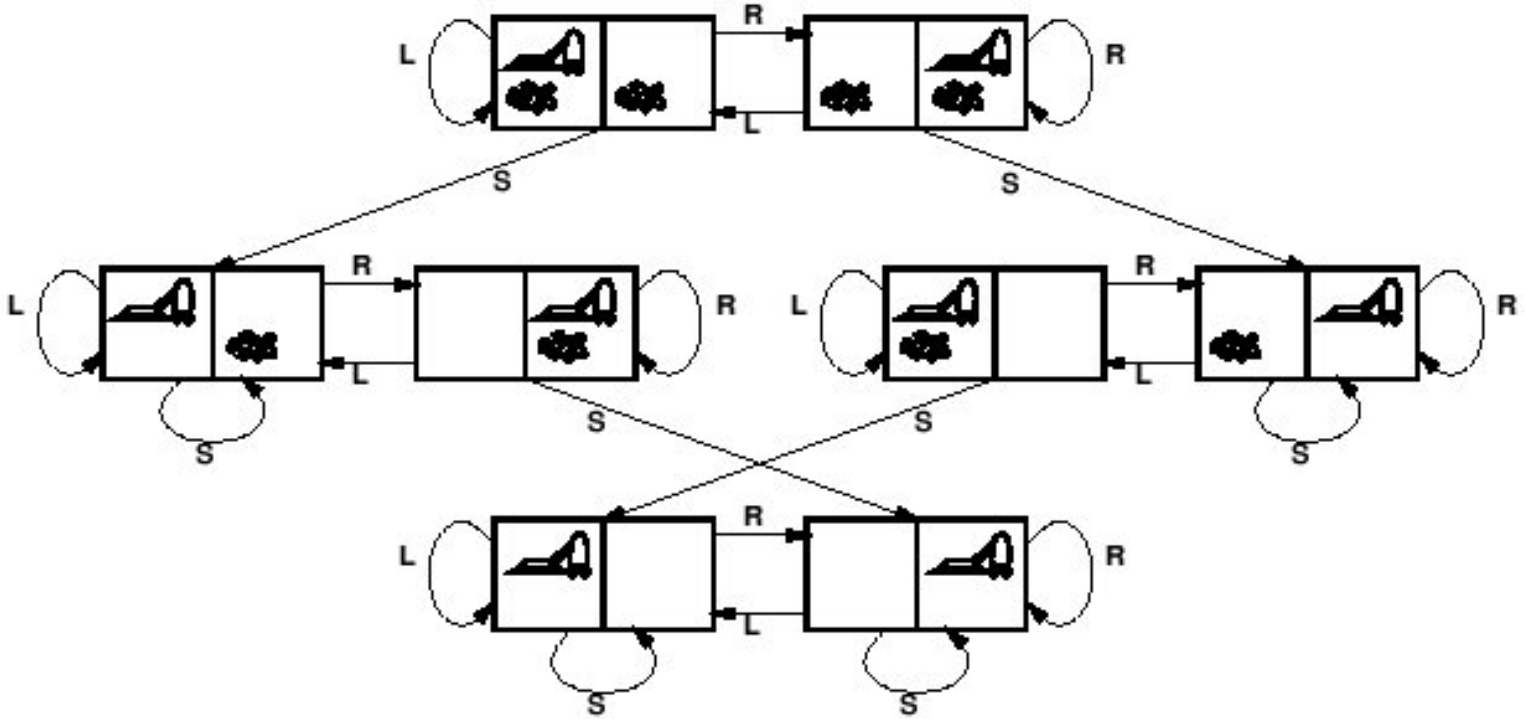
- Goal states are $\{7, 8\}$
- The world has two locations.
- Agent can be in any of them.
- Agent's actions are: Left, Right, Suck.
- Each location may be dirty or clean.

Single-State Problems

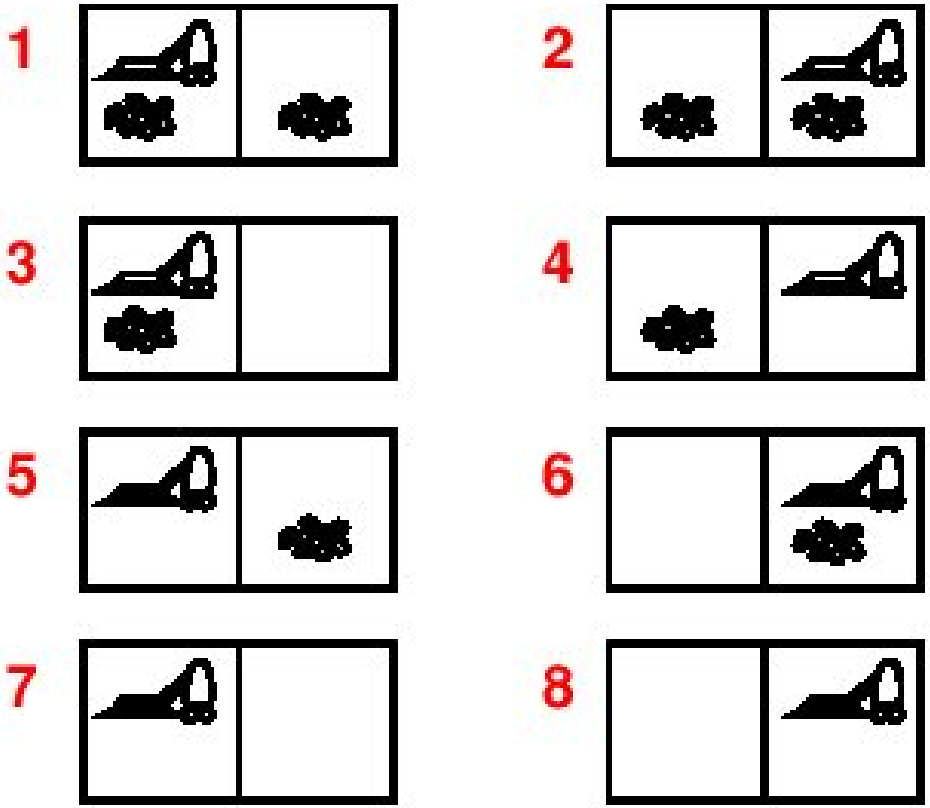
- Agents know exactly in which state it is.
- State information may be received by sensors or using a complicated process of saving track of the world.
- The initial state of the agent should be clear.
- Agent should know the exact effect of its action.

Single-State Problems, Cont

- If the vacuum cleaner agent's sensors givesw it enough information to tell
- In these problems, next state can be find using current state and action.



Single-State Problem, Cont



- Consider Initial state to be 4.
- (Left, Suck)
- Init State= 4
- After Left= 3
- After Suck= 7

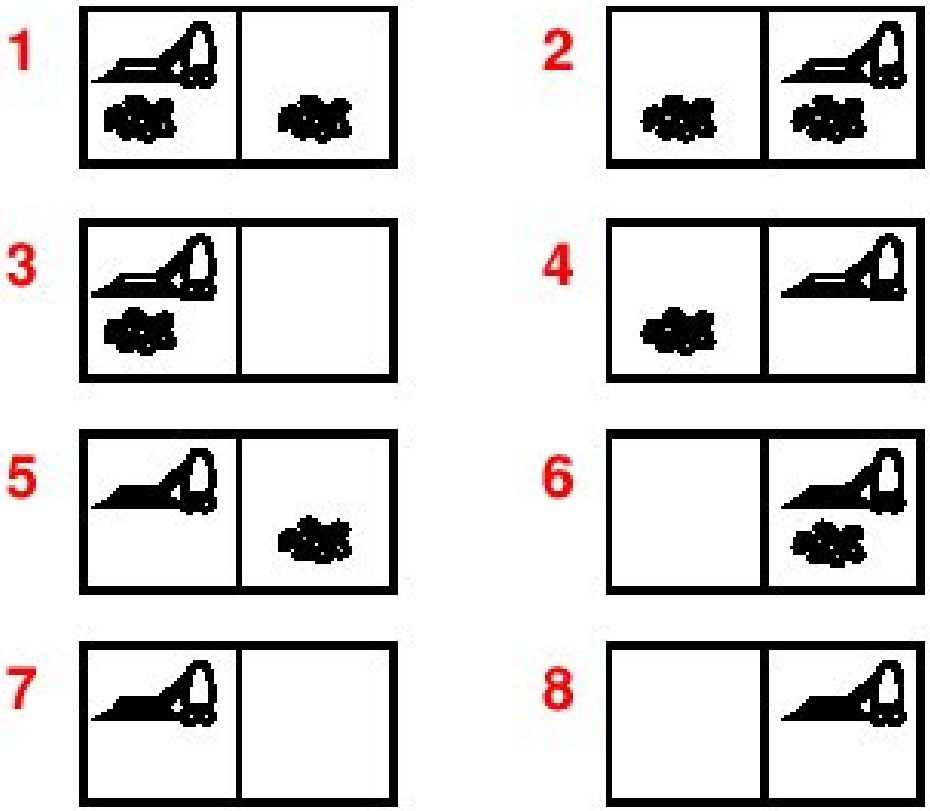
Multi-State Problems

- Agent can not find exactly in which state it is in (world is not fully accessible).
- Agent knows all the effects of its action.
- In these problesm, agent should reason about sets of states that it might go to.

Multi-State Problem, Cont

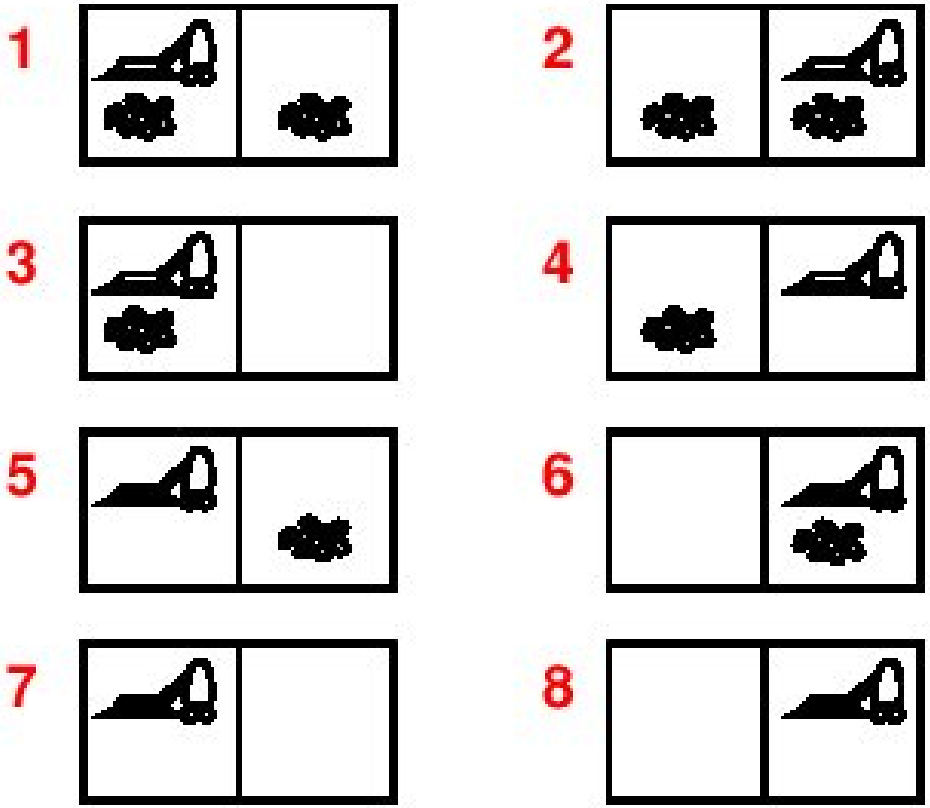
- Agent still knows the effect of his actions.
- The sensor-less vacuum cleaner agent.
- So, agent doesn't know the initial state.
- Is there a sequence of actions that leads to goal?

Multi-State Problem, Cont



- YES.
- (Left, Suck, Right, Suck)
- Init States=$\{1, 2, 3, 4, 5, 6, 7, 8\}$
- After Left= $\{1, 3, 5, 7\}$
- After Suck= $\{5, 7\}$
- After Right= $\{6, 8\}$
- After Suck= $\{8\}$

Contigency Problems



- The effect of an action is not clear.
- Usually in non-deterministic environment.
- Example: The suck action obeys Murphy Law.
- 4=¿ $\{2, 4\}$

Contigency Problem, Cont

- Does an action sequence exist for a contigency problem?
- Sometimes, No.
- Example: Murphy suck with limited dirt sensor.
- We need to Interleaving between search and execute.

Exploring Problem

- The agent has no information about effect of his action.
- Like new born baby.
- Agent should experiment to discover what its actions do.
- Usually, agent can not be simulated.
- Example: Being lost in a new country without map.

Well-defined problems and solutions

- A problem is a collection of information that agent will use to decide what to do.
- Basic elements of problem definition are STATEs and ACTIONs.

Single-state problem

- A problem is defined by four items:
- Initial state
- Successor function S: STATE $\rightarrow$ <ACTION, STATE>
  S (1)= $\{(Right, 1), (Left, 2), (Suck, 5)\}$
- Goal test can be:
  A set of state, e.g. x= {Bucharest}
  A function: e.g., checkmate test in chess

Single-state problem, Cont

- Path cost function assigns a cost to a path.
  Sum of costs of individual actions.
- A Solution is a path from initial state to (a) goal state with minimun Path cost.

Multi-state problem

-
- Try to find the answers by yourself!!
- Initial state set: ?
- Successor function: ?
- Goal test: ?
- When we are at the Goal?

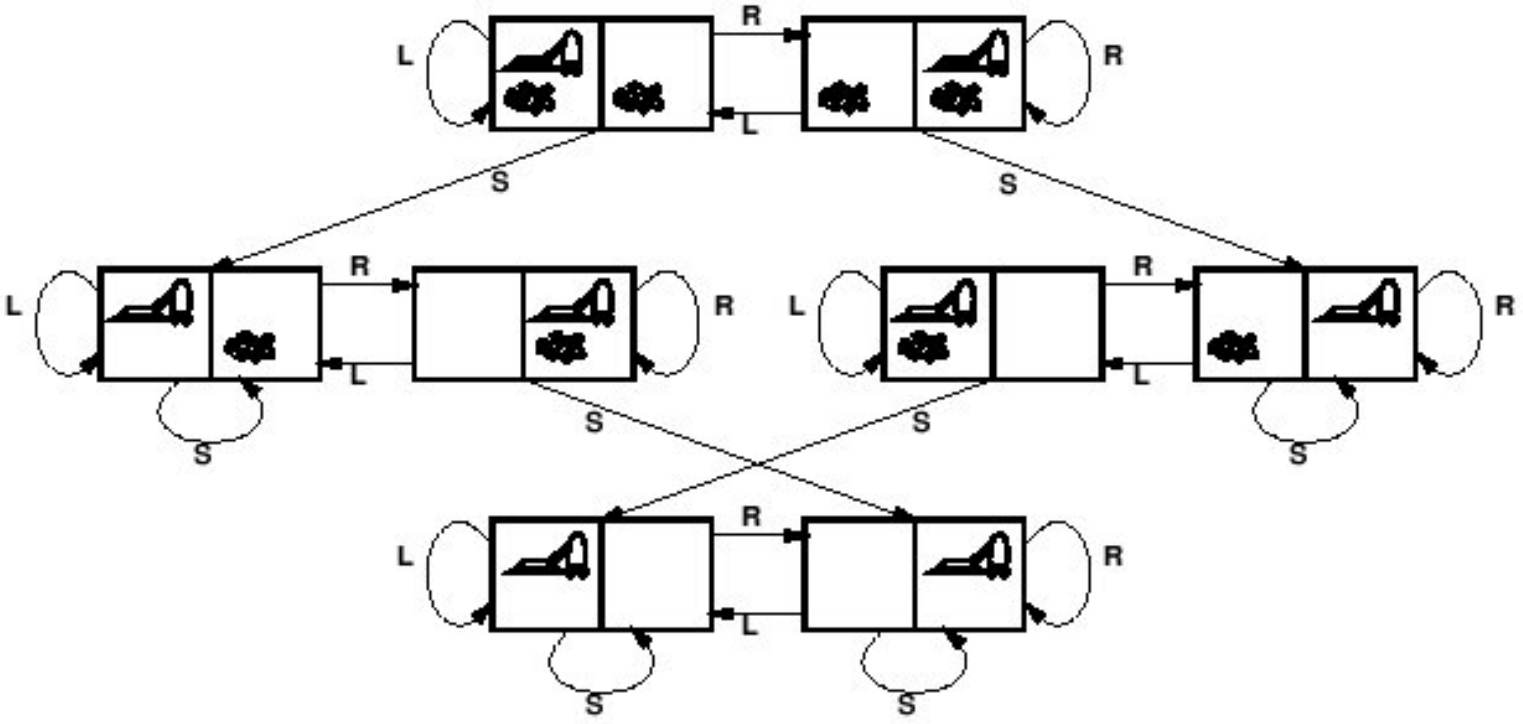
Search Alg. performance

- There are three ways of measuring:
  1. Can it find a solution?
  2. Is it a good solution?
  3. What is the search cost?
- Total cost (of the search): path cost+ search cost

Choosing states and actions

- Abstraction: process of removing details from a representation.
- State: set of real states
  e.g., In Arad
- Action: complex combination of real action.
  e.g., Arad $\rightarrow$ Silbiu
- Solution: Set of real paths which are solutions in real world.
  e.g., Arad, Sibiu, Fagaras, Bucharest

Single-State Problems, Cont

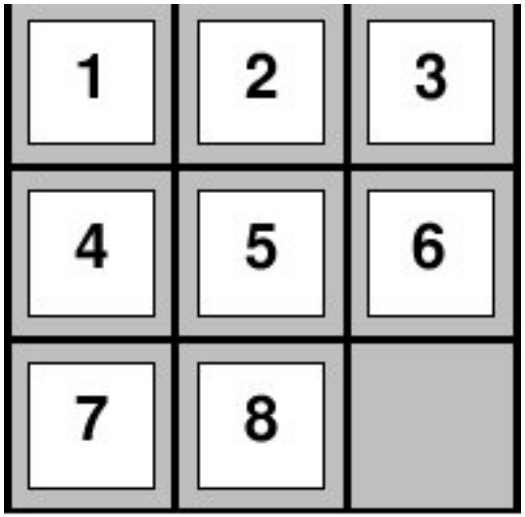


- States: <Robot Location, Cell1 is dirty, Cell2 is dirty>
- Actions: Left, Right, Suck
- Goal: Both cells are cleaned
- Path cost: 1 per action

Problems in AI

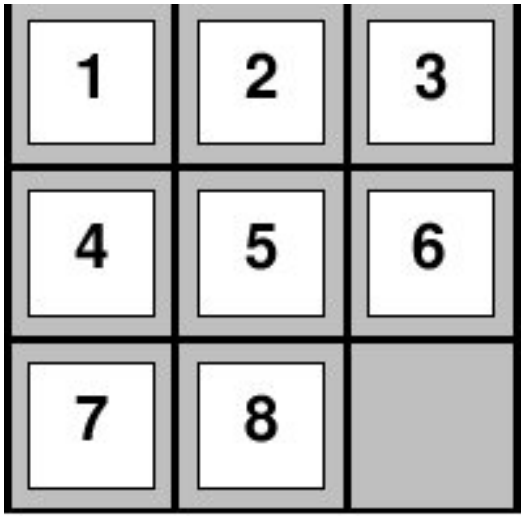
- Toy problems
  - have a concise description.
  - can be used for comparing search alg.
- Real-world problems
  - have no concise description.
  - May have several problem formulations.

8-Puzzle problem



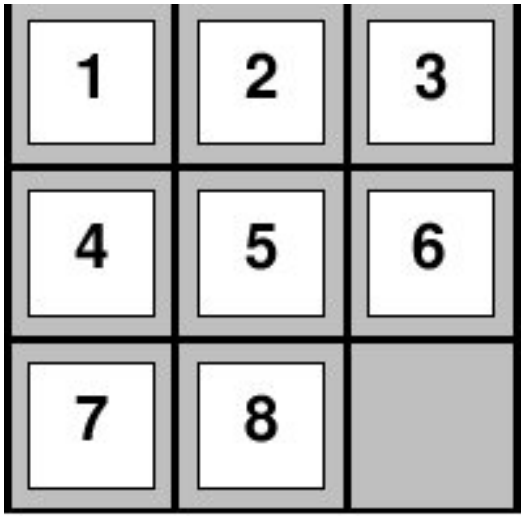
- States:
- Actions:
- Goal test:
- Path cost:

8-Puzzle problem



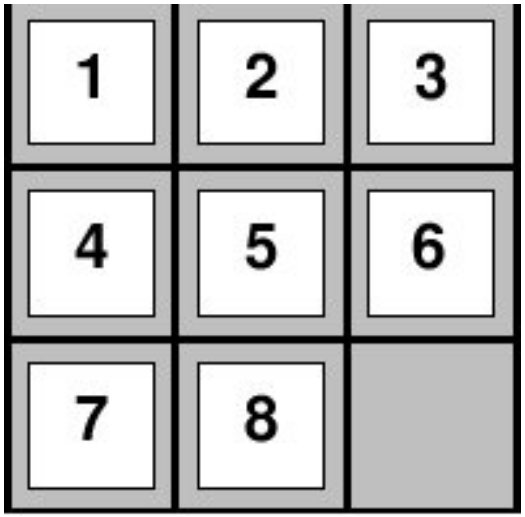
- States: Location of tiles as a sequence of integers.
- Actions:
- Goal test:
- Path cost:

8-Puzzle problem



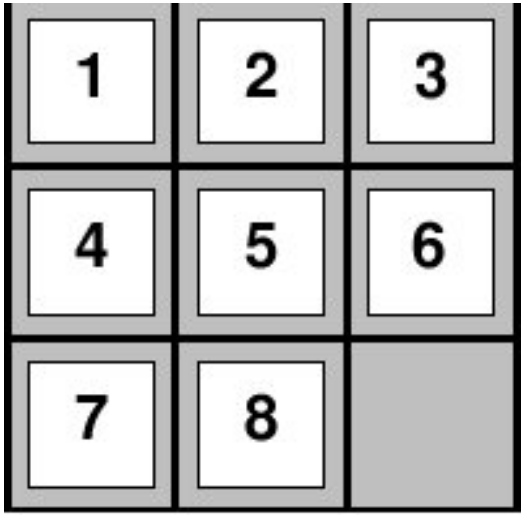
- States: Location of tiles as a sequence of integers.
- Actions: Blank moves left, right, up or down.
- Goal test:
- Path cost:

8-Puzzle problem



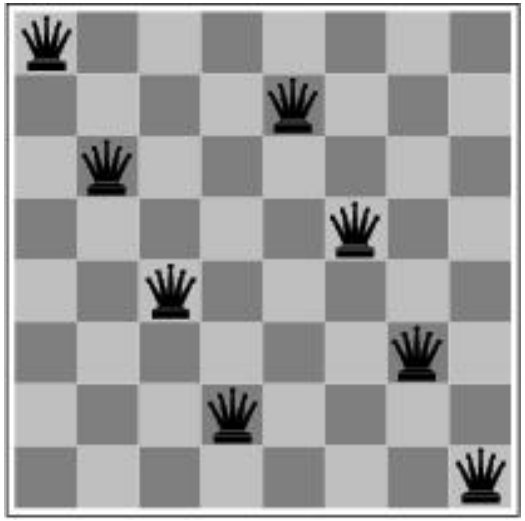
- States: Location of tiles as a sequence of integers.
- Actions: Blank moves left, right, up or down.
- Goal test: Goal state (123456780).
- Path cost:

8-Puzzle problem



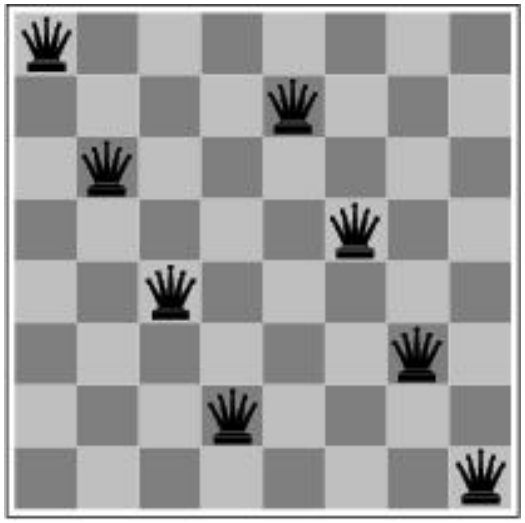
- States: Location of tiles as a sequence of integers.
- Actions: Blank moves left, right, up or down.
- Goal test: Goal state (123456780).
- Path cost: one per move.

8-Queens problem



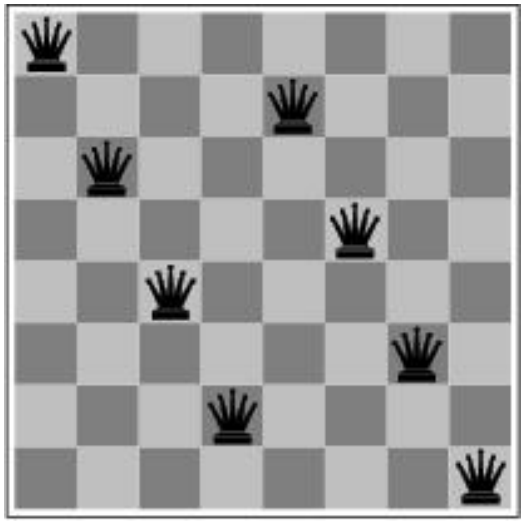
- Placing 8 queens on board while no one attacks another.
- There are two formulations:
  - Incremental formulation
  - Complete-state formulation

8-Queens problem- Incremental formulation



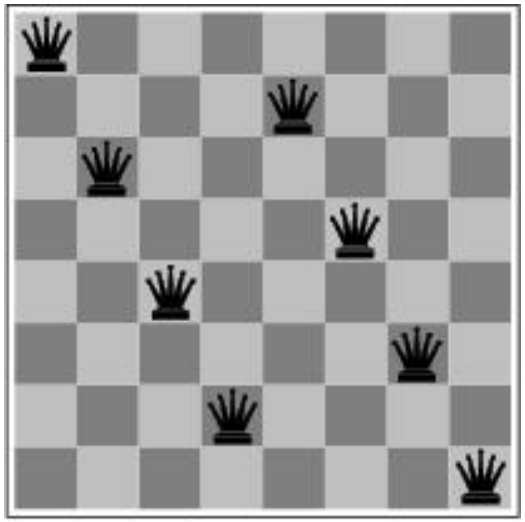
- States:
- Actions:

8-Queens problem- Incremental formulation



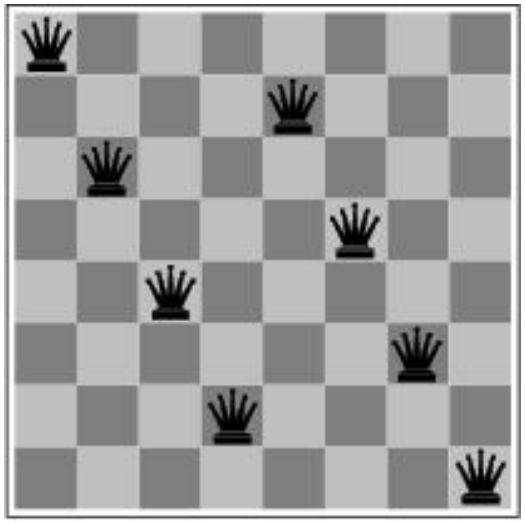
- States: Arrangement of 0 to 8 queens.
- Actions:

8-Queens problem- Incremental formulation



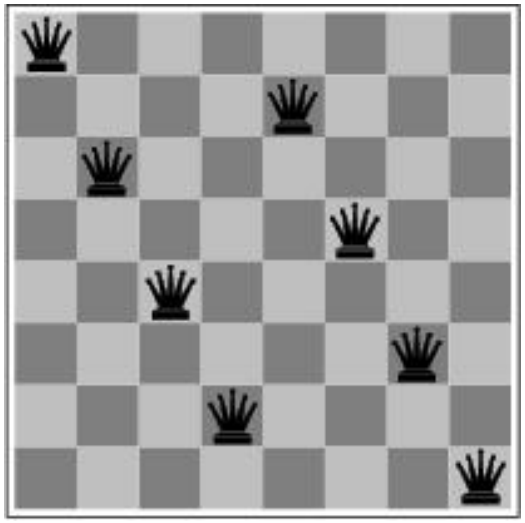
- States: Arrangement of 0 to 8 queens.
- Actions: Adding a queen.

8-Queens problem- Complete-state formulation



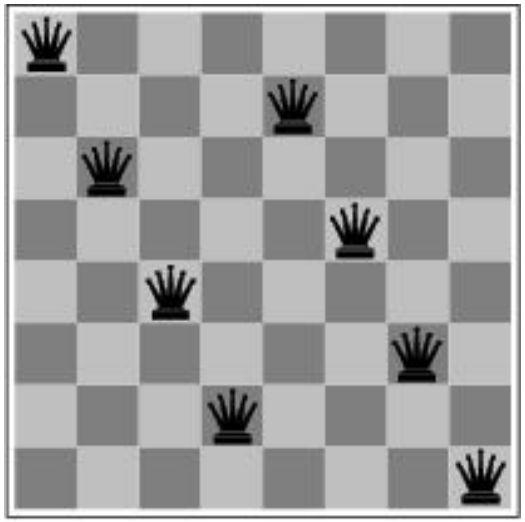
- States:
- Actions:

8-Queens problem- Complete-state formulation



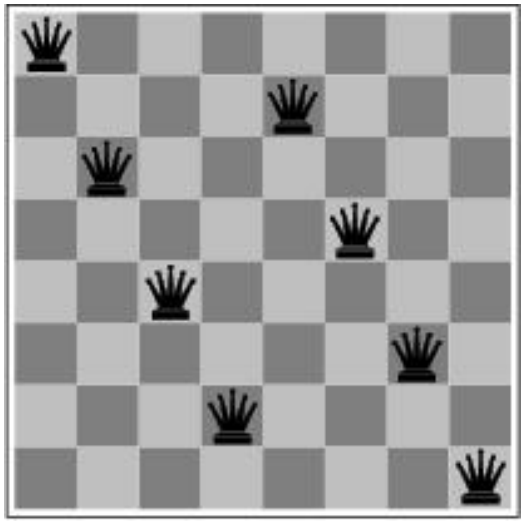
- States: arrangement of 8 queens, one in each column
- Actions:

8-Queens problem- Complete-state formulation



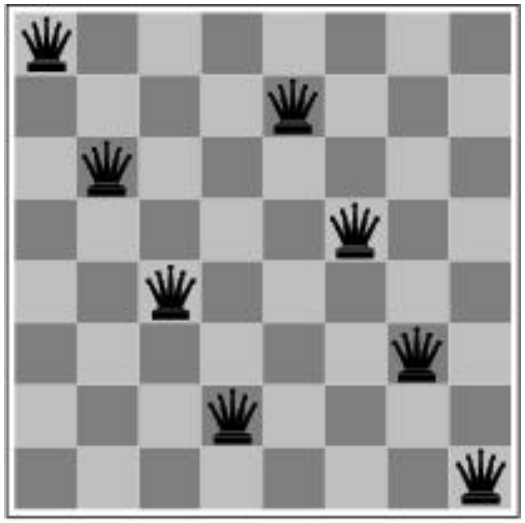
- States: arrangement of 8 queens, one in each column
- Actions: move any attacked queens

8-Queens problem



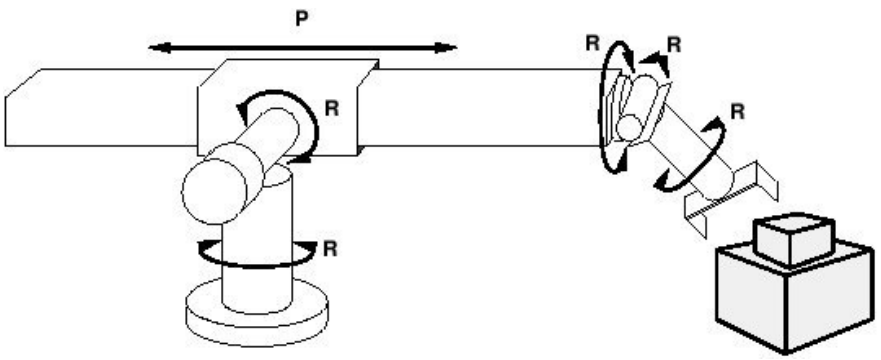
- Goal test:
- Path cost:

8-Queens problem



- Goal test: No queen attack another one.
- Path cost: 0.

Real-world problems- Robotic Assembly



- States: coordinates of joint angles, arms, etc
- Action: Move one or more joints
- Goal test: complete assembly
- Path cost: Time, Power.

Generating action sequences

- Search function: returns a solution or failure.
- Expanding: Applying operators(actions) on the current state.
- Search Strategy: The choice of which state ot expand first is determined by it.

General Search

- Search Alg build a search tree whose root is correspond to initial state.
- Nodes in tree are expanded using operators.
- Some nodes may be visited more than once.

General Search

```
function GENERAL-SEARCH (problem, strategy)
  return a solution or failure

  search-tree= INITIALIZE-TREE (init-state-of-problem)
  node= initial-state-of-problem

  while (true)
    node= STRATEGY (tree)
    if node is empty then return failure
    if node contains a goal state then
      return solution
    else
      EXPAND-NODE (tree, node)
  end
```

Search Strategies

- Search strategy defines the order of nodes expansion.
- Important factors in choosing search strategies:
  - Completeness
  - Time complexity
  - Space complexity
  - Optimality

Search Strategies, Cont

- Time and Space complexities are measured in terms of:
  - b: Maximum branching factor of search tree
  - d: Depth of the least-cost solution
  - m: Maximum depth of state space

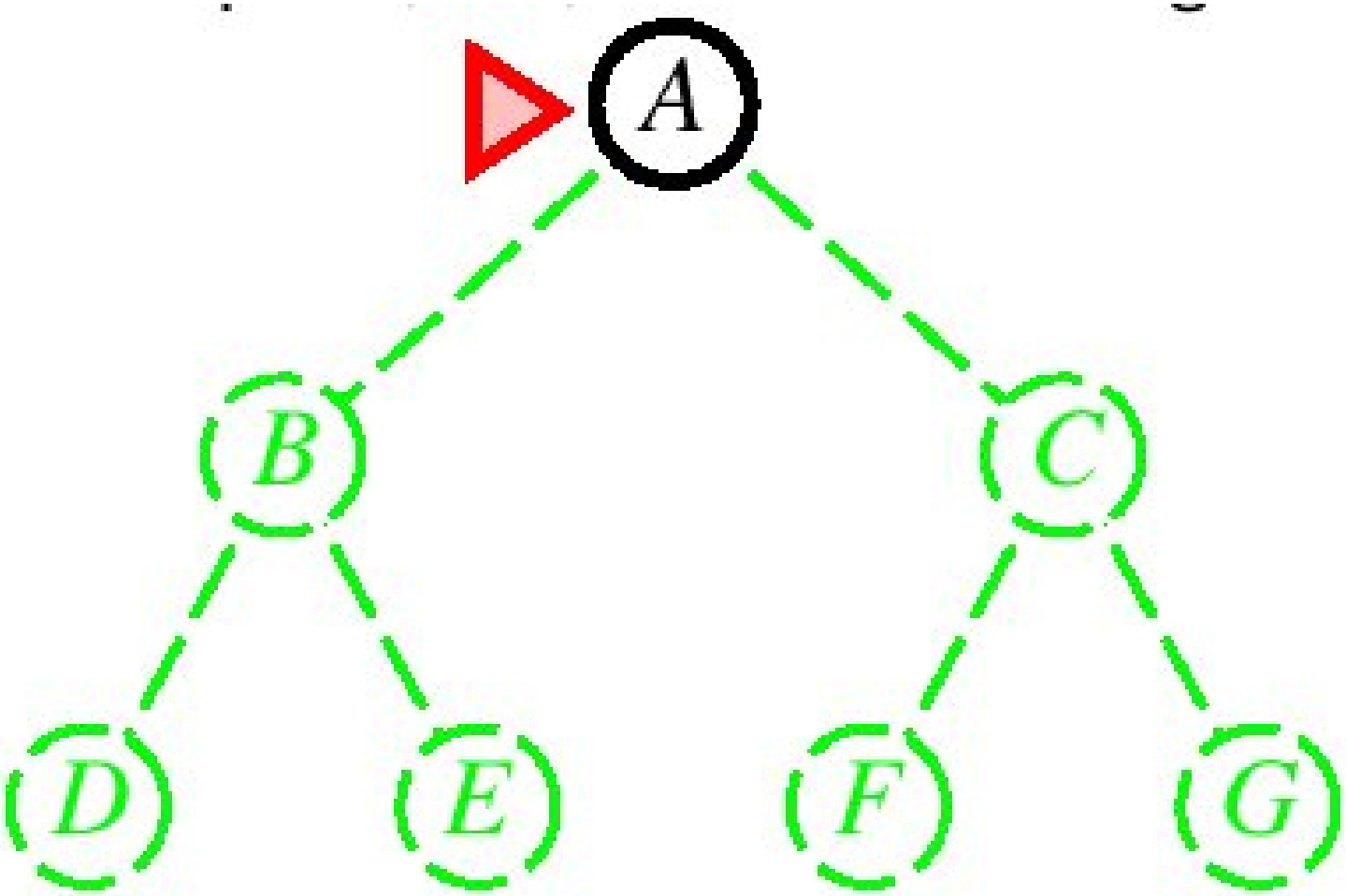
Uninformed Search

- Uninformed search: Have no information about number of steps or path cost.
- Breadth-first search
- Uniform search
- Space complexity
- Optimality

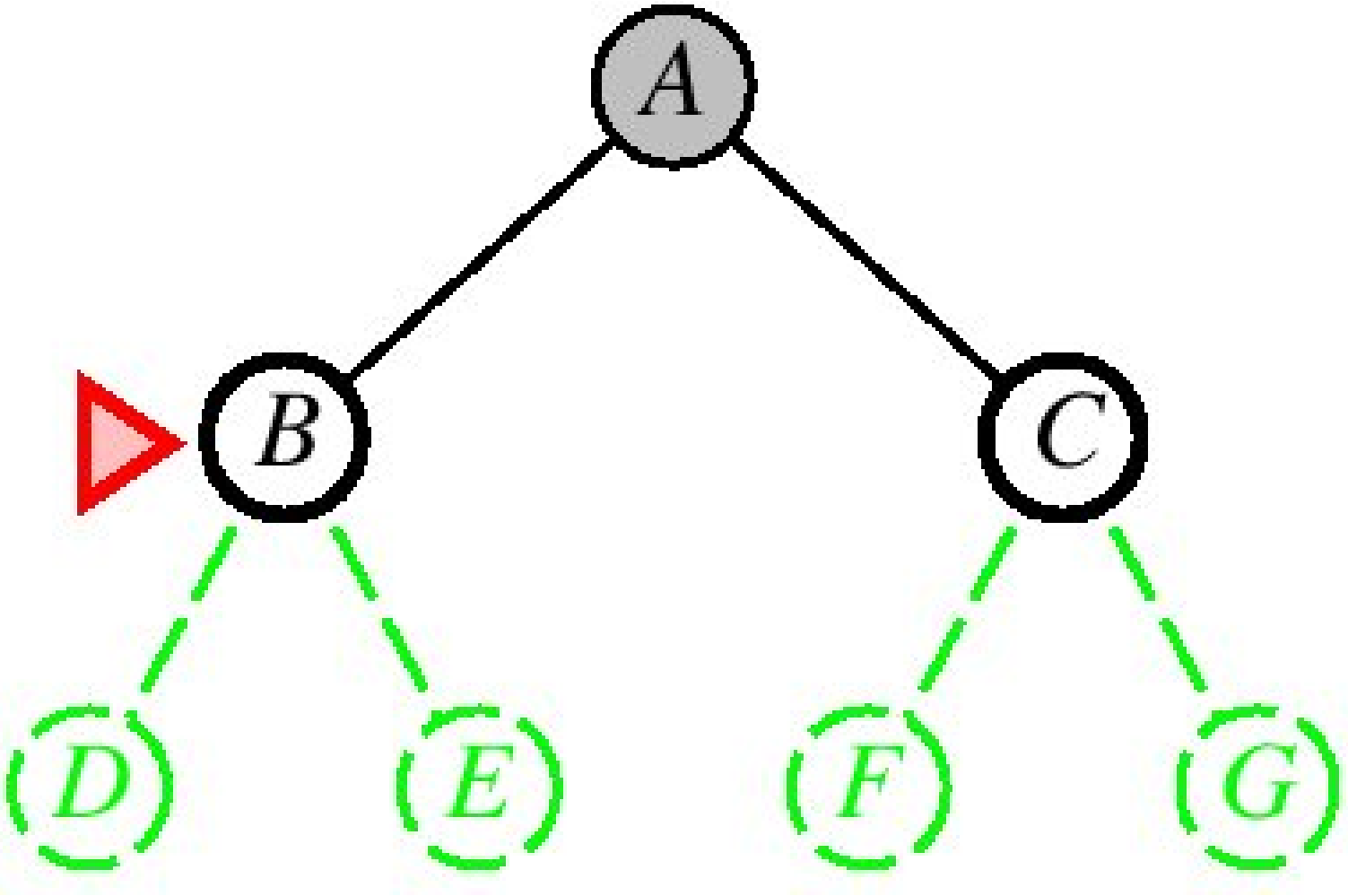
Breadth-first search

- Expand the earliest un-expanded node.
- FIFO: First In, First Out
- Can be implemented using a Queue.

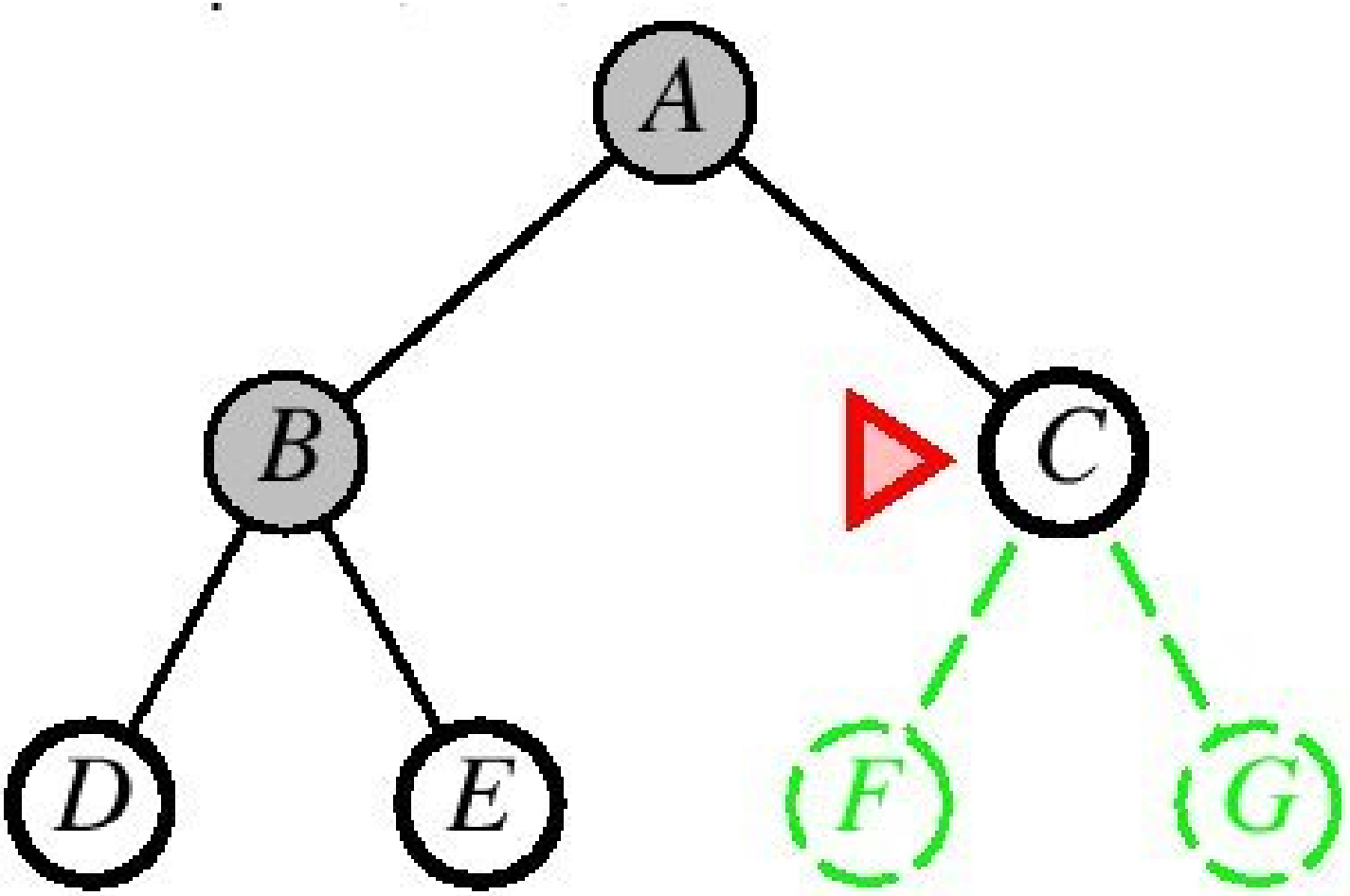
Breadth-first search, Cont



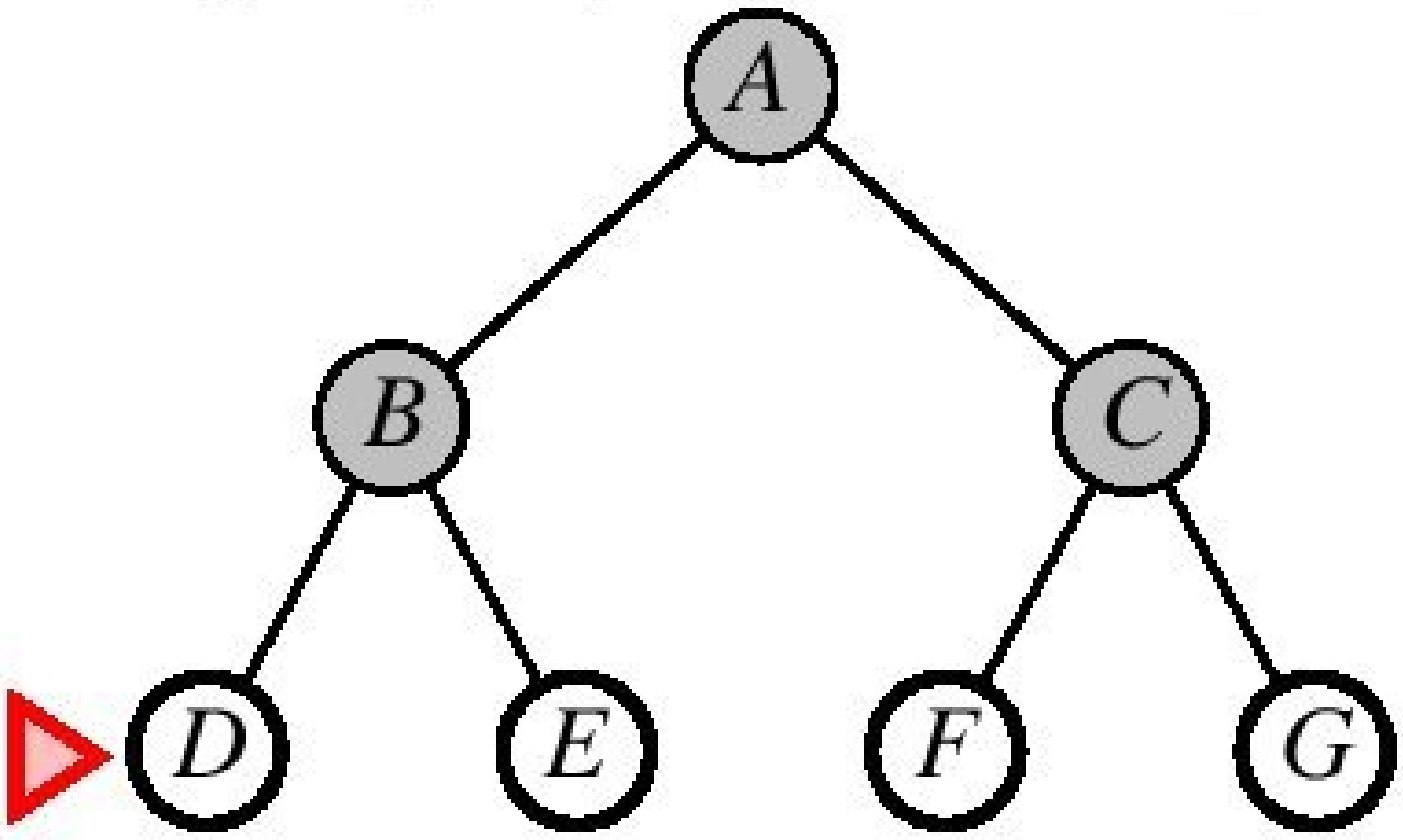
Breadth-first search, Cont



Breadth-first search, Cont



Breadth-first search, Cont



Breadth-first search, Cont

- Complete:
- Time complexity:
- Space complexity:
- Optimality:

Breadth-first search, Cont

- Complete: Yes (if b is not infinite)
- Time complexity:
- Space complexity:
- Optimality:

Breadth-first search, Cont

- Complete: Yes (if b is not infinite)
- Time complexity: $1 + b + b^2 + \cdots + b^d = O(b^d)$
- Space complexity:
- Optimality:

Breadth-first search, Cont

- Complete: Yes (if b is not infinite)
- Time complexity: $1 + b + b^2 + \cdots + b^d = O(b^d)$
- Space complexity: Your third assignment
- Optimality:

Breadth-first search, Cont

- Complete: Yes (if b is not infinite)
- Time complexity: $1 + b + b^2 + \cdots + b^d = O(b^d)$
- Space complexity: Your third assignment
- Optimality: No (in general problems)

Breadth-first search, Cont

- The main issue with Breadth-first search is space.
- The space needed is exponential with respect of d.

Uniform-cost search

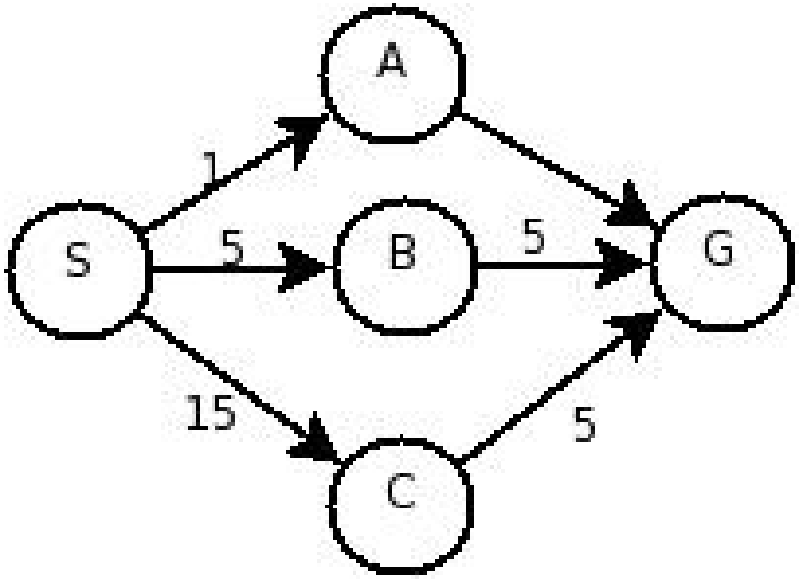
- BFS finds the goal which needs the minimum no. of actions.
- Let $g(v)$ denotes the cost of reaching to node v.
- BFS is optimal iff $g(v) = Depth(v)$
- Uniform-cost search is an optimal search strategy.

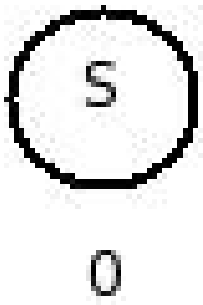
Uniform-cost search, Cont

- Expands the least-cost unexpanded node.
- It doesn't look forward, but just look backward!!
- It can be implemented using a priority queue.
- To be optimal, it may continue searching even if found a goal.

Uniform-cost search, Cont

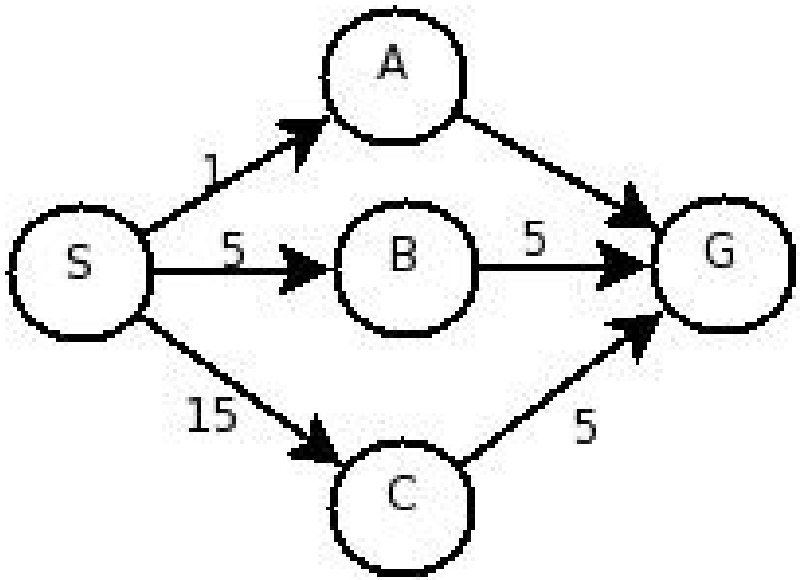
- To be optimal, it may continue searching even if found a goal.
- It continues expanding while the least-cost is bigger than the min goal-cost

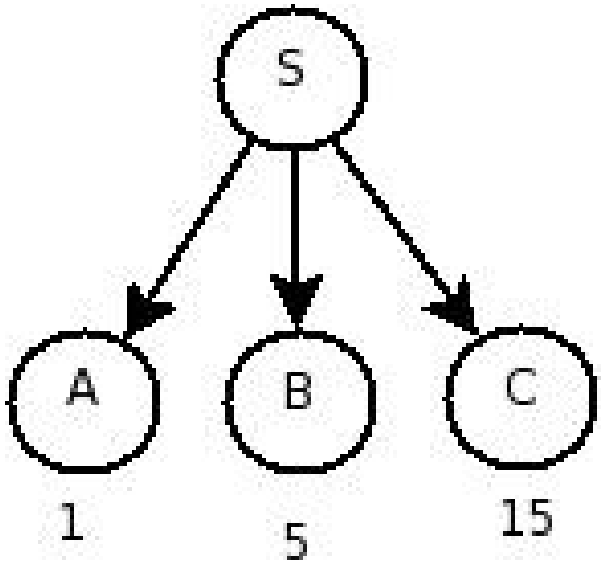




Uniform-cost search, Cont

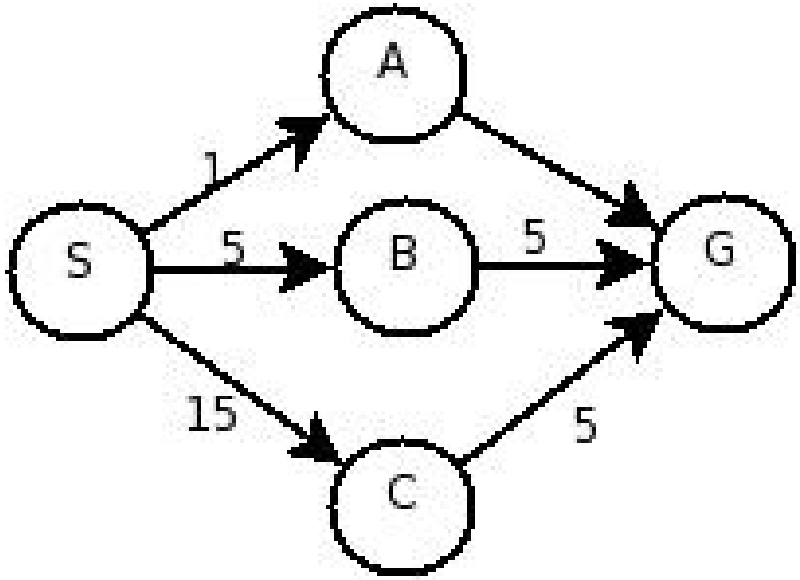
- To be optimal, it may continue searching even if found a goal.
- It continues expanding while the least-cost is bigger than the min goal-cost

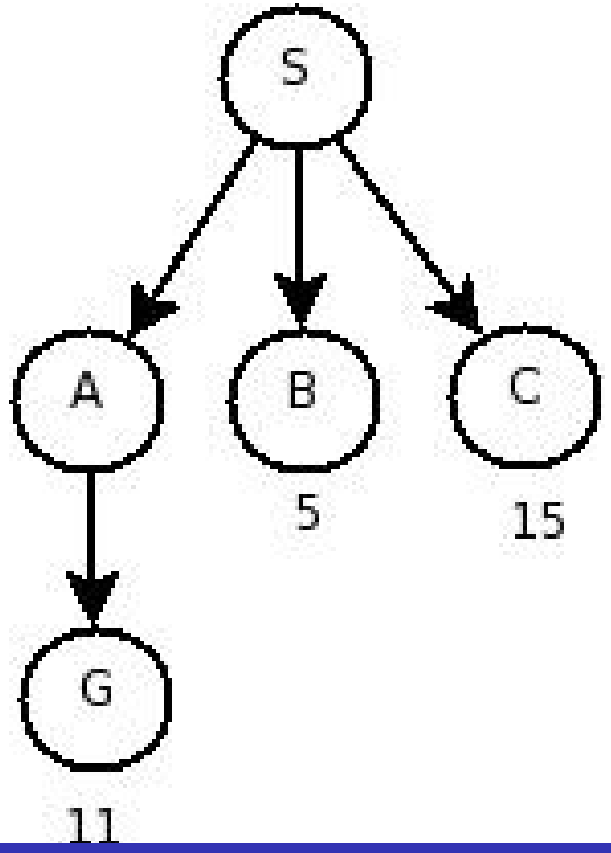




Uniform-cost search, Cont

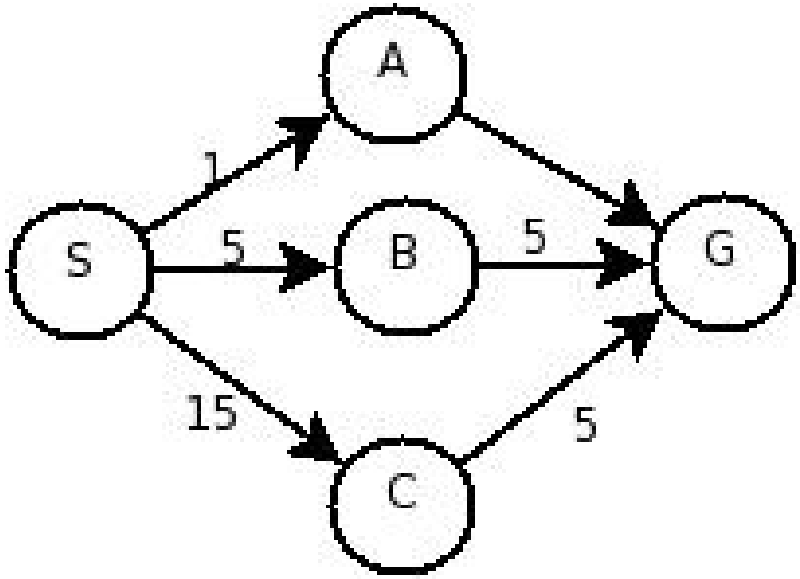
- To be optimal, it may continue searching even if found a goal.
- It continues expanding while the least-cost is bigger than the min goal-cost

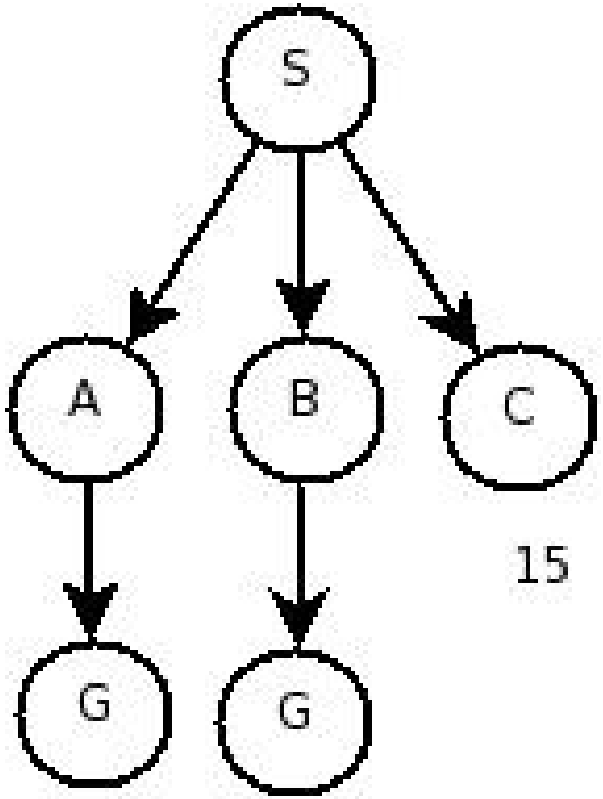




Uniform-cost search, Cont

- To be optimal, it may continue searching even if found a goal.
- It continues expanding while the least-cost is bigger than the min goal-cost





Uniform-cost search, Cont

- Uniform-cost search finds the cheapest solution(goal) if:
- Cost a path never decreases along the path:
- i.e. $g(v) \leq g(SUCCESSOR(v))$
- i.e. all of actions have non-negative cost
- Why?

Uniform-cost search, Cont

- Complete:
- Time complexity:
- Space complexity:
- Optimality:

Uniform-cost search, Cont

- Let ϵ be the least of actions' cost
- Complete:
- Time complexity:
- Space complexity:
- Optimality:

Uniform-cost search, Cont

- Let ϵ be the least of actions' cost
- Complete: YES, if $\epsilon \geq 0$.
- Time complexity:
- Space complexity:
- Optimality:

Uniform-cost search, Cont

- Let C^* be the cost of optimal solution.
- Complete: YES, if $\epsilon \geq 0$.
- Time complexity: $O(b^{\frac{C^*}{\epsilon}})$
- Space complexity:
- Optimality:

Uniform-cost search, Cont

- Let C^* be the cost of optimal solution.
- Complete: YES, if $\epsilon \geq 0$.
- Time complexity: $O(b^{\frac{C^*}{\epsilon}})$
- Space complexity: $O(b^{\frac{C^*}{\epsilon}})$
- Optimality:

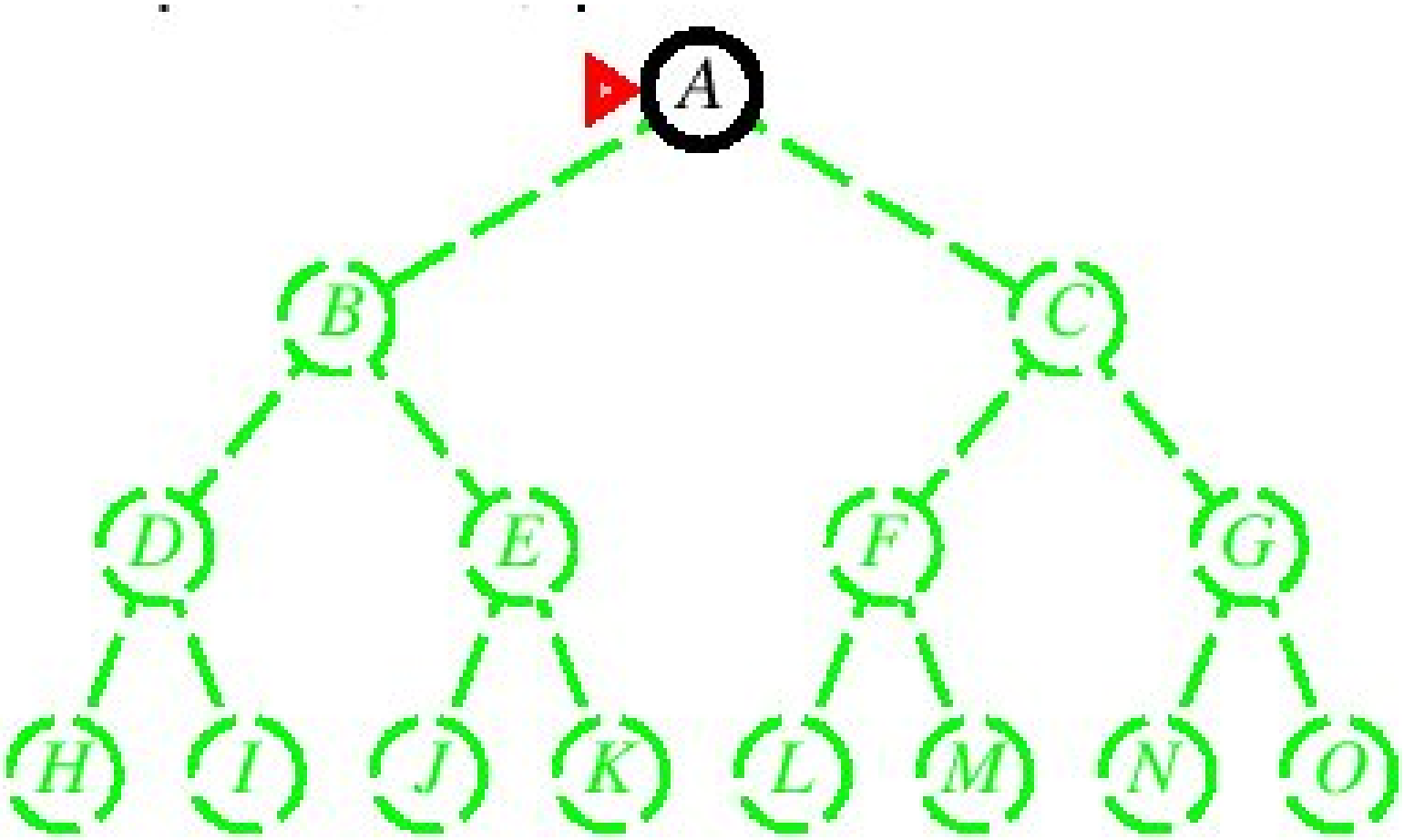
Uniform-cost search, Cont

- Let C^* be the cost of optimal solution.
- Complete: YES, if $\epsilon \geq 0$.
- Time complexity: $O(b^{\frac{C^*}{\epsilon}})$
- Space complexity: $O(b^{\frac{C^*}{\epsilon}})$
- Optimality: YES

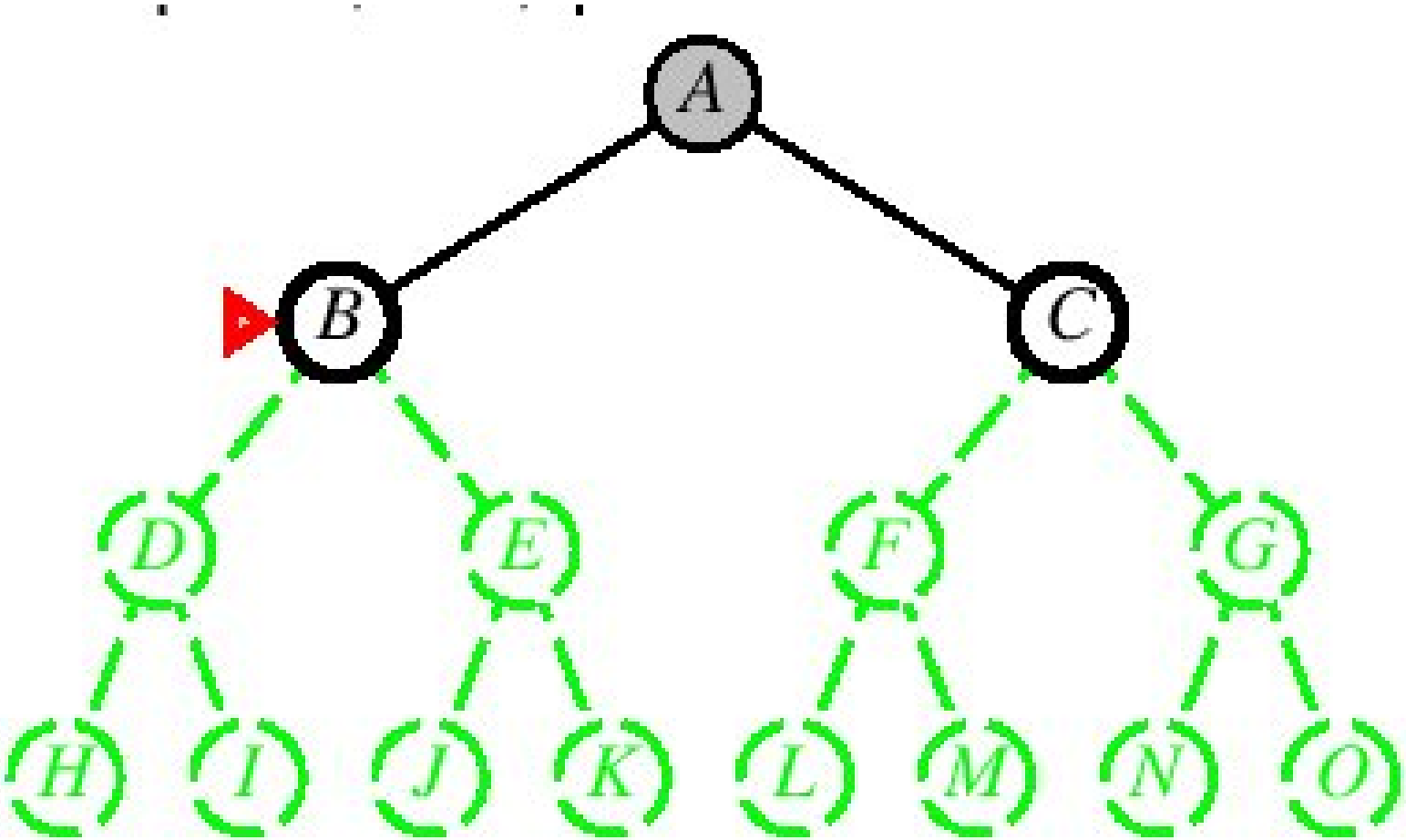
Depth-first search

- Expand the deepest un-expanded node.
- LIFO: Last In, First Out
- Can be implemented using a stack.

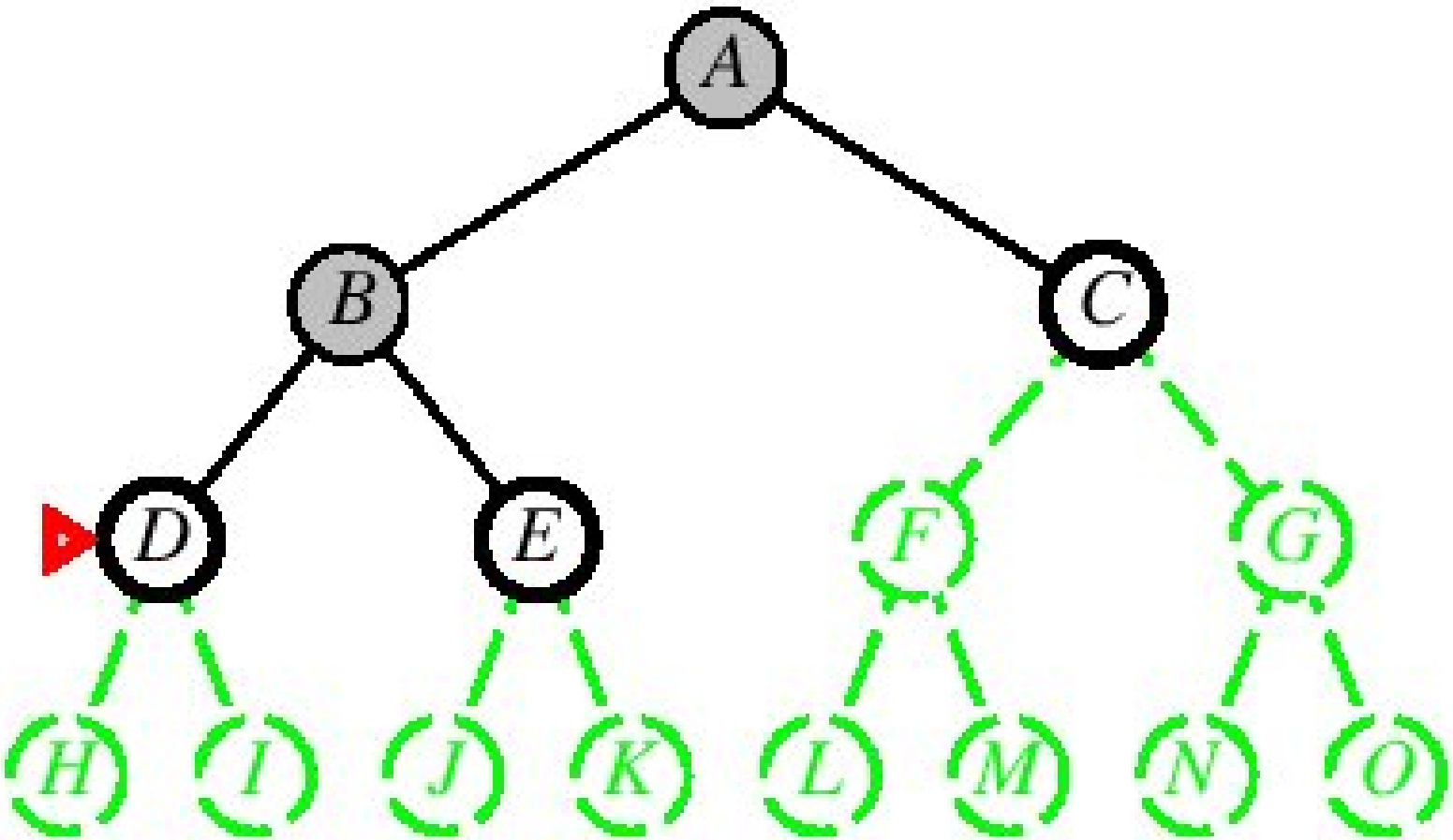
Depth-first search, Cont



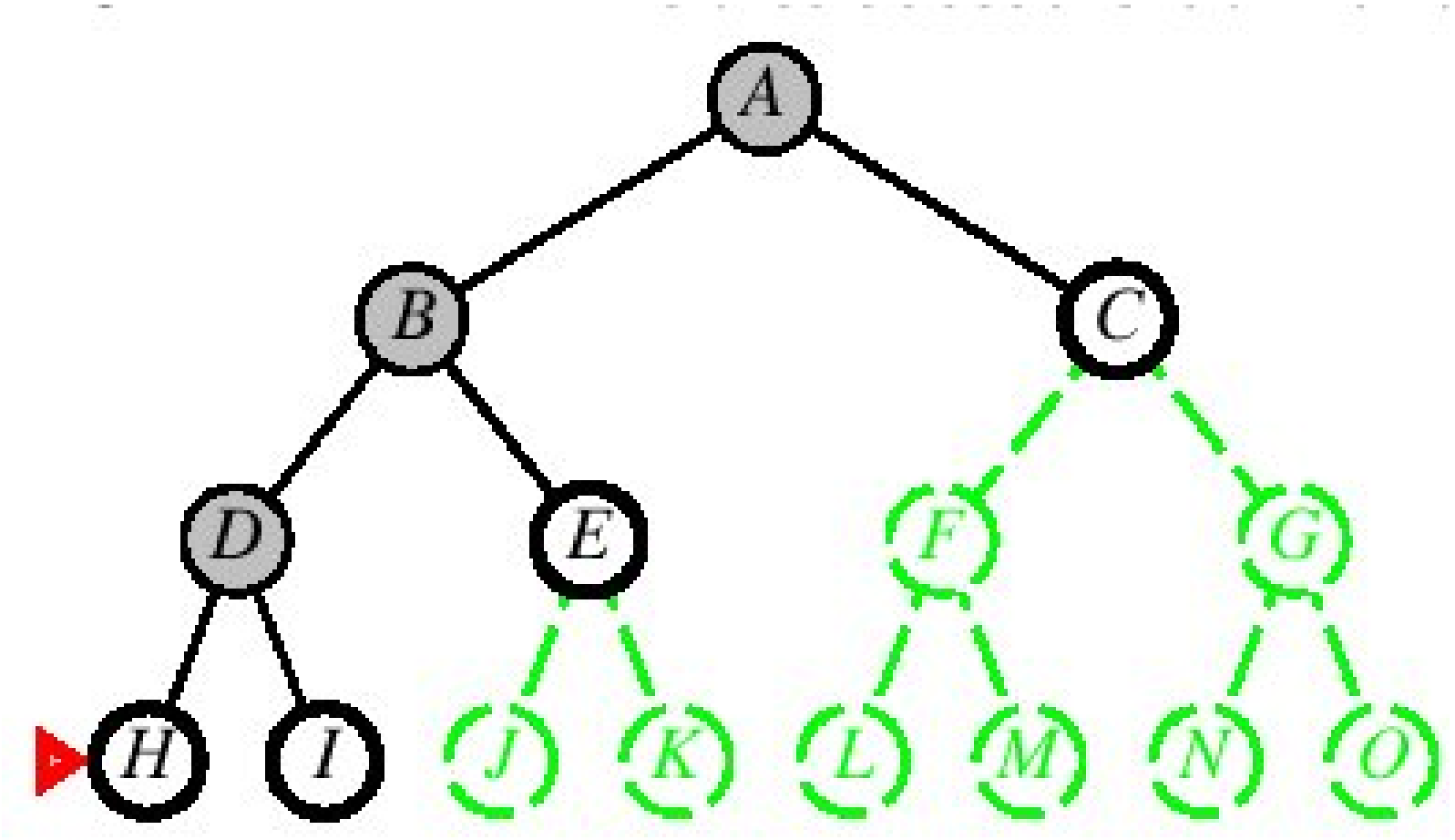
Depth-first search, Cont



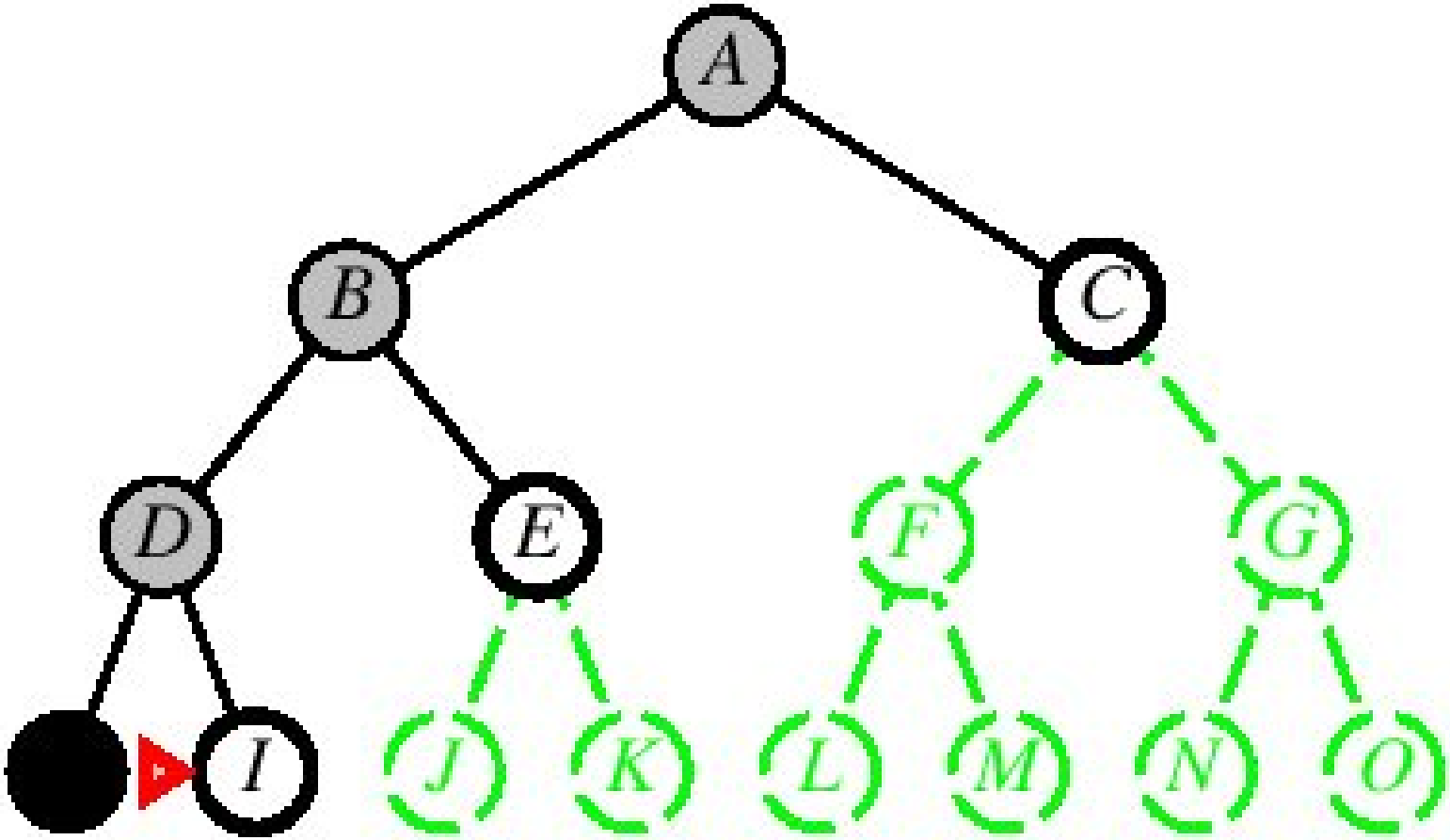
Depth-first search, Cont



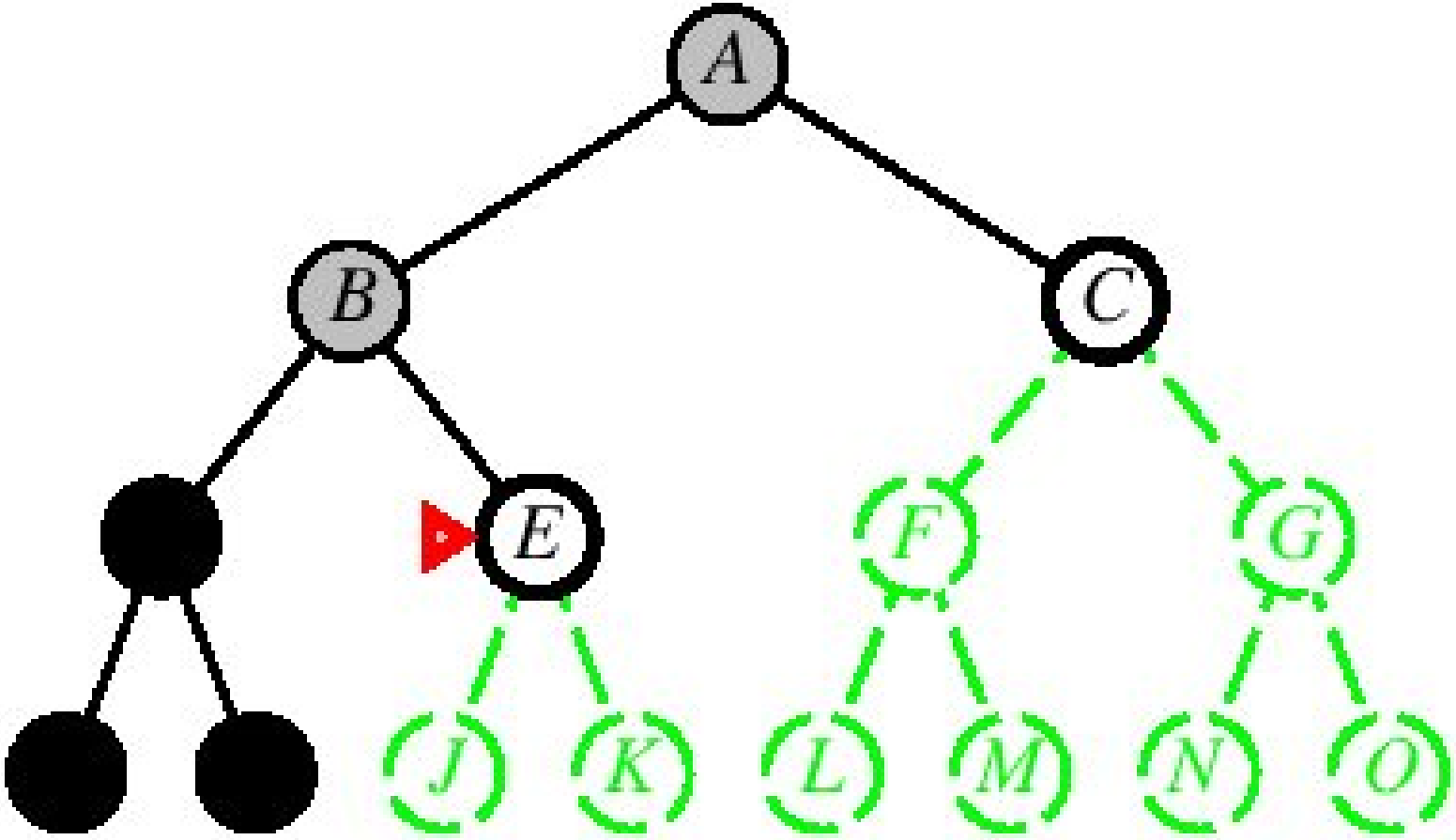
Depth-first search, Cont



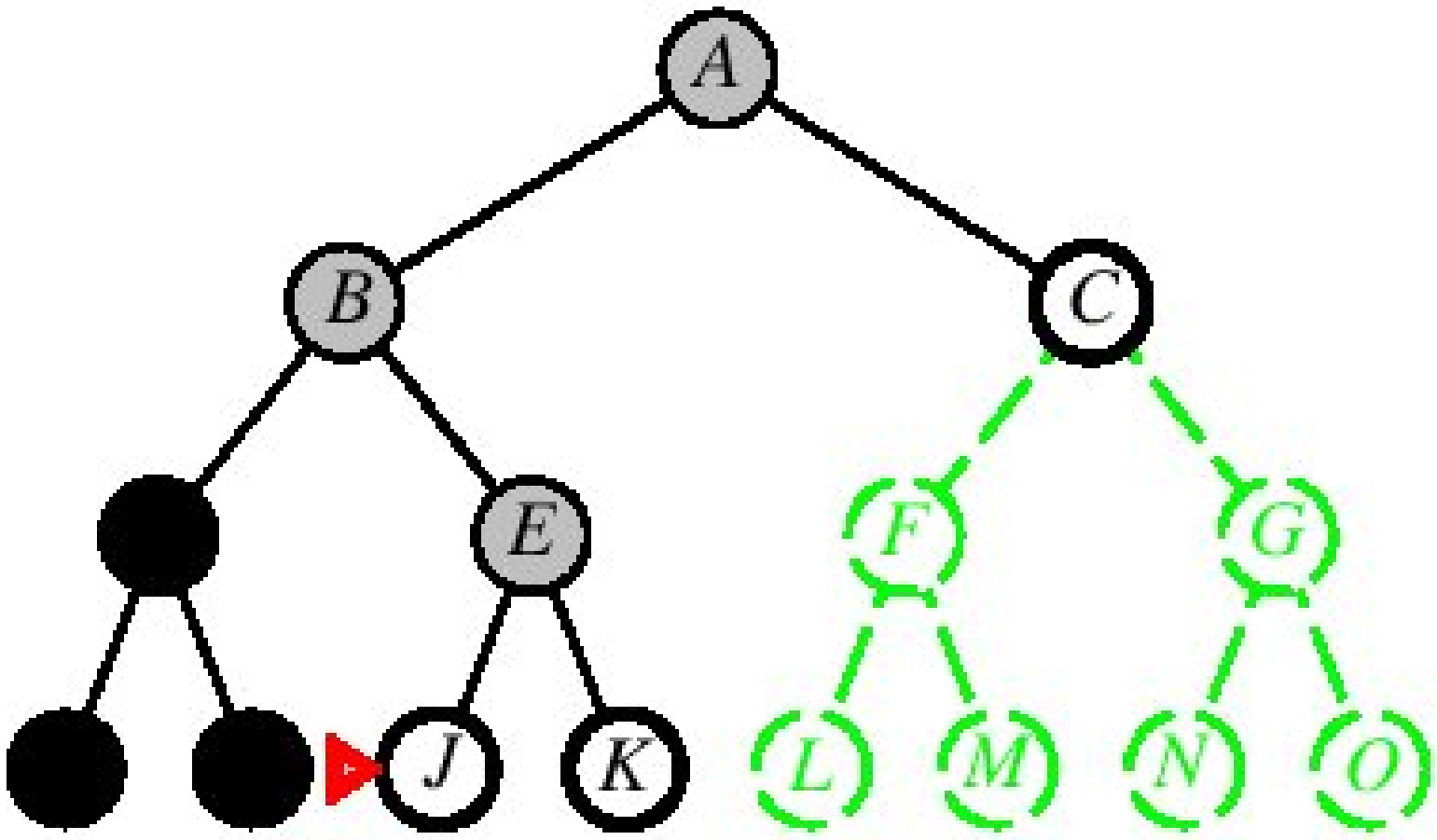
Depth-first search, Cont



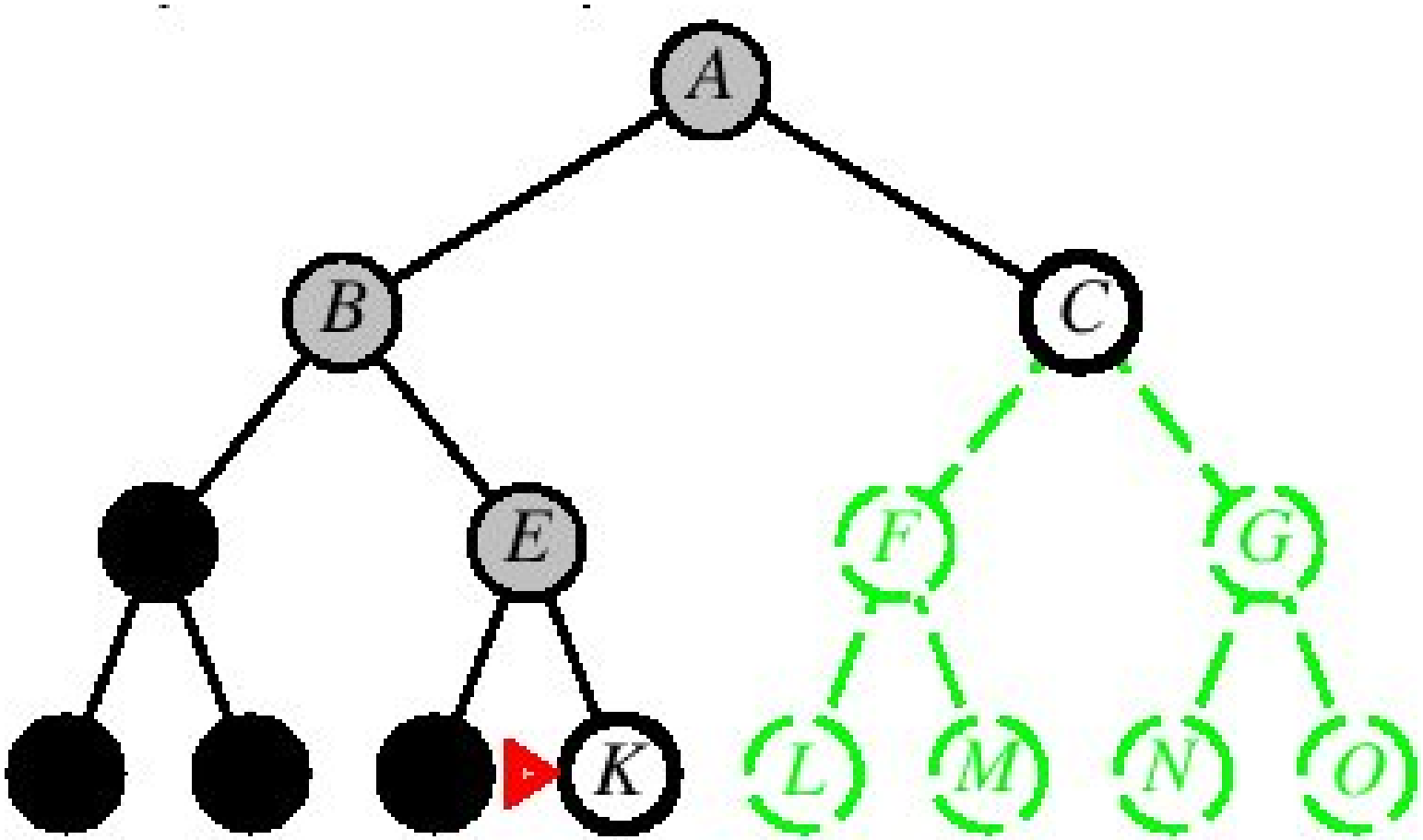
Depth-first search, Cont



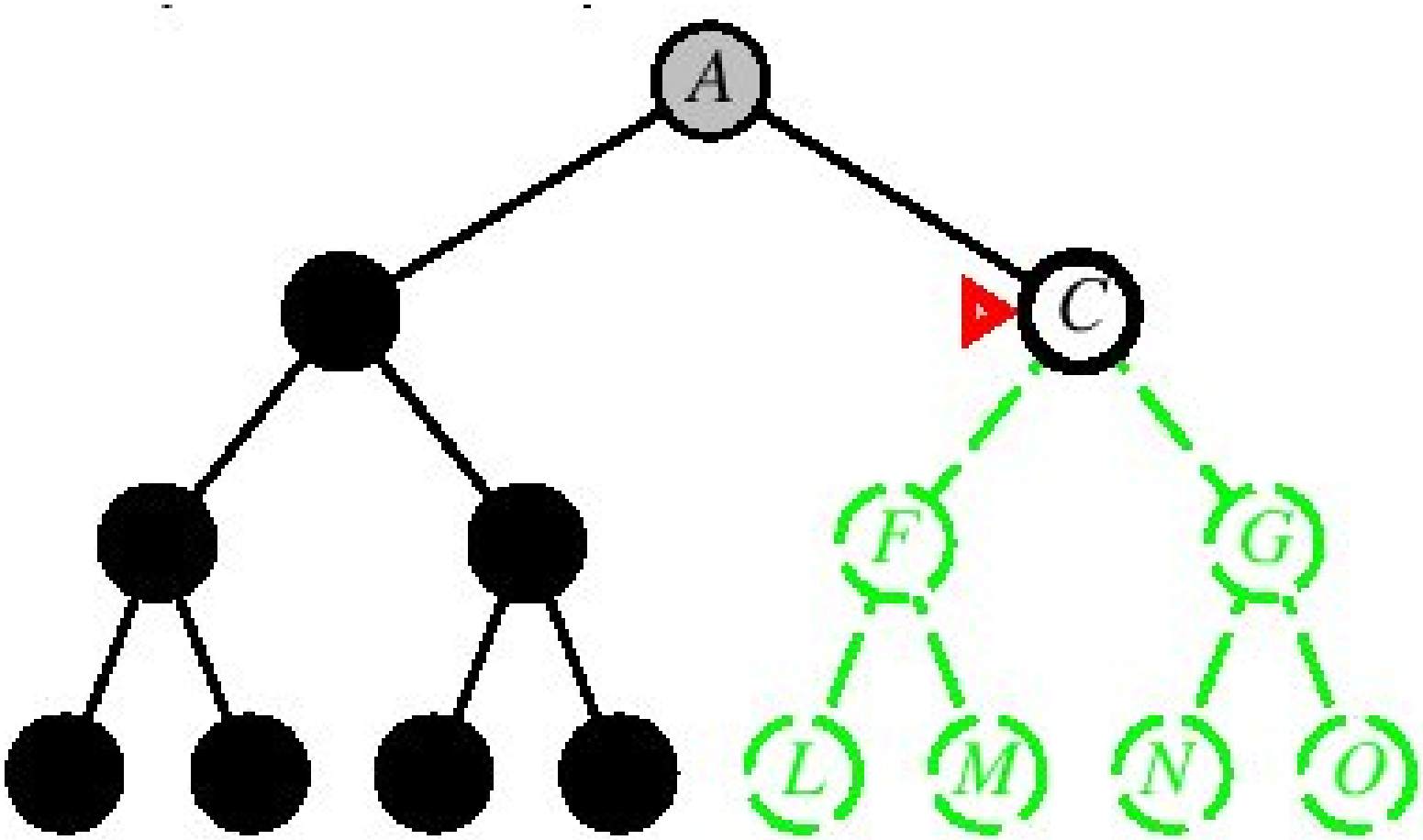
Depth-first search, Cont



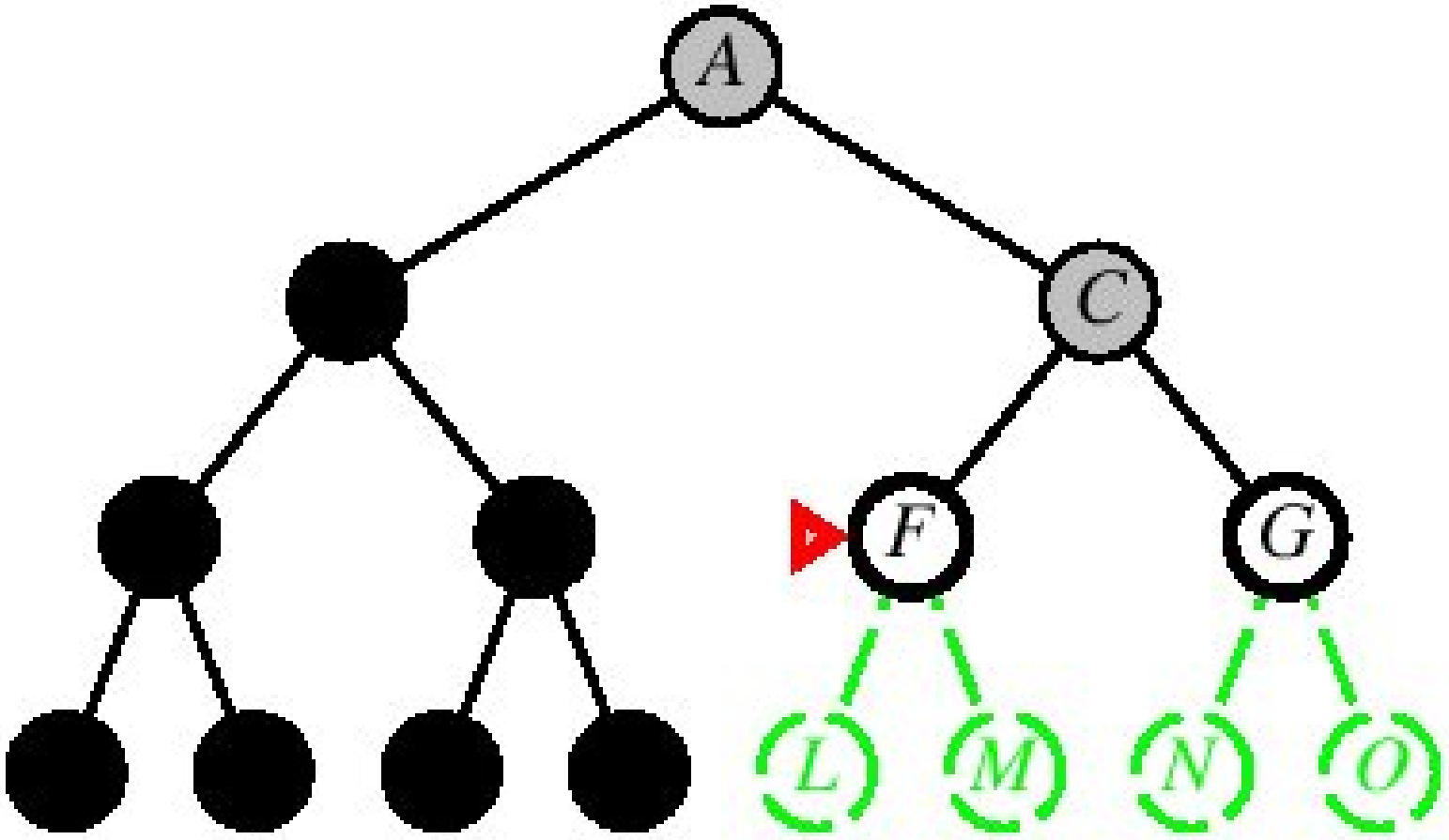
Depth-first search, Cont



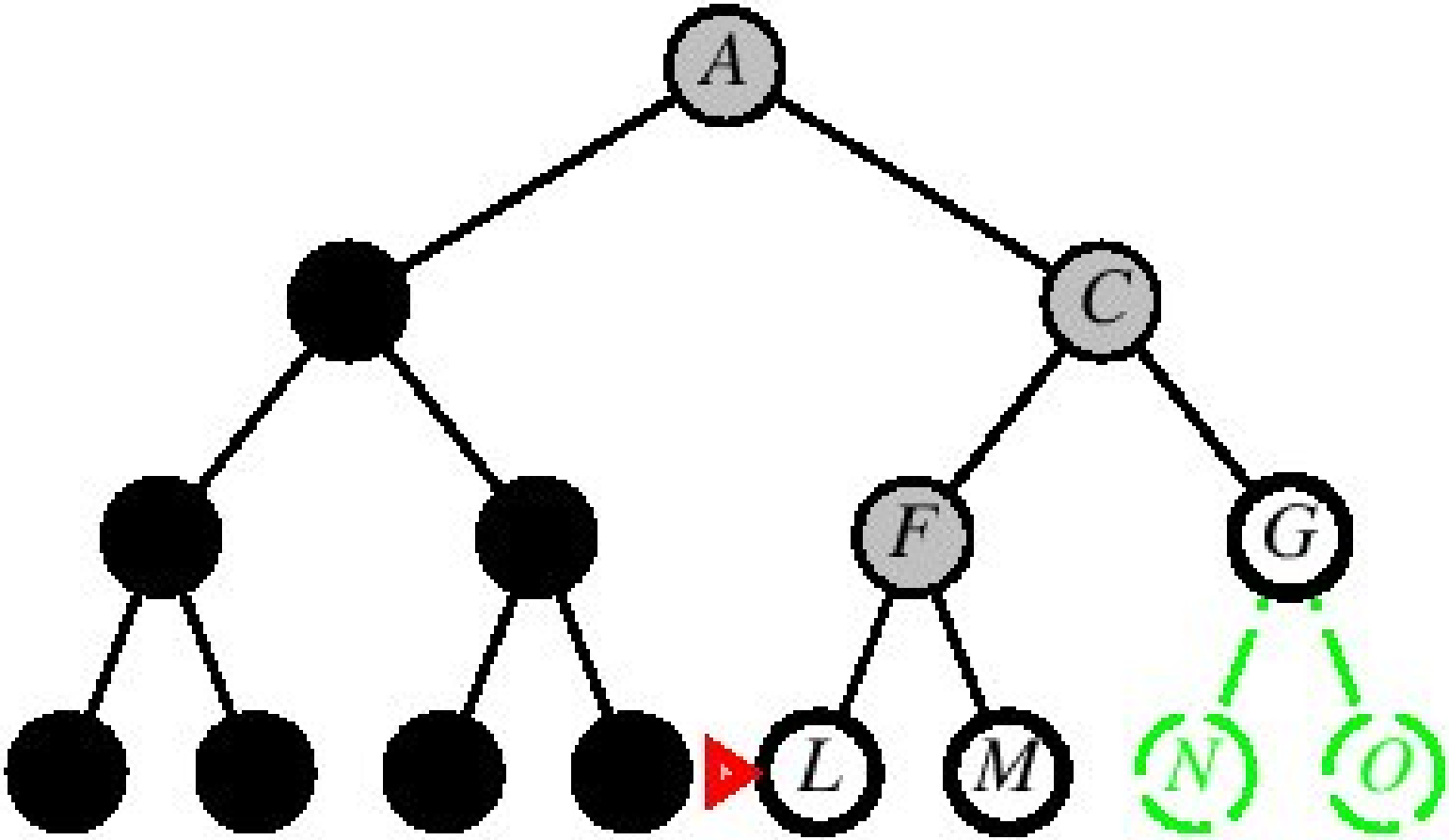
Depth-first search, Cont



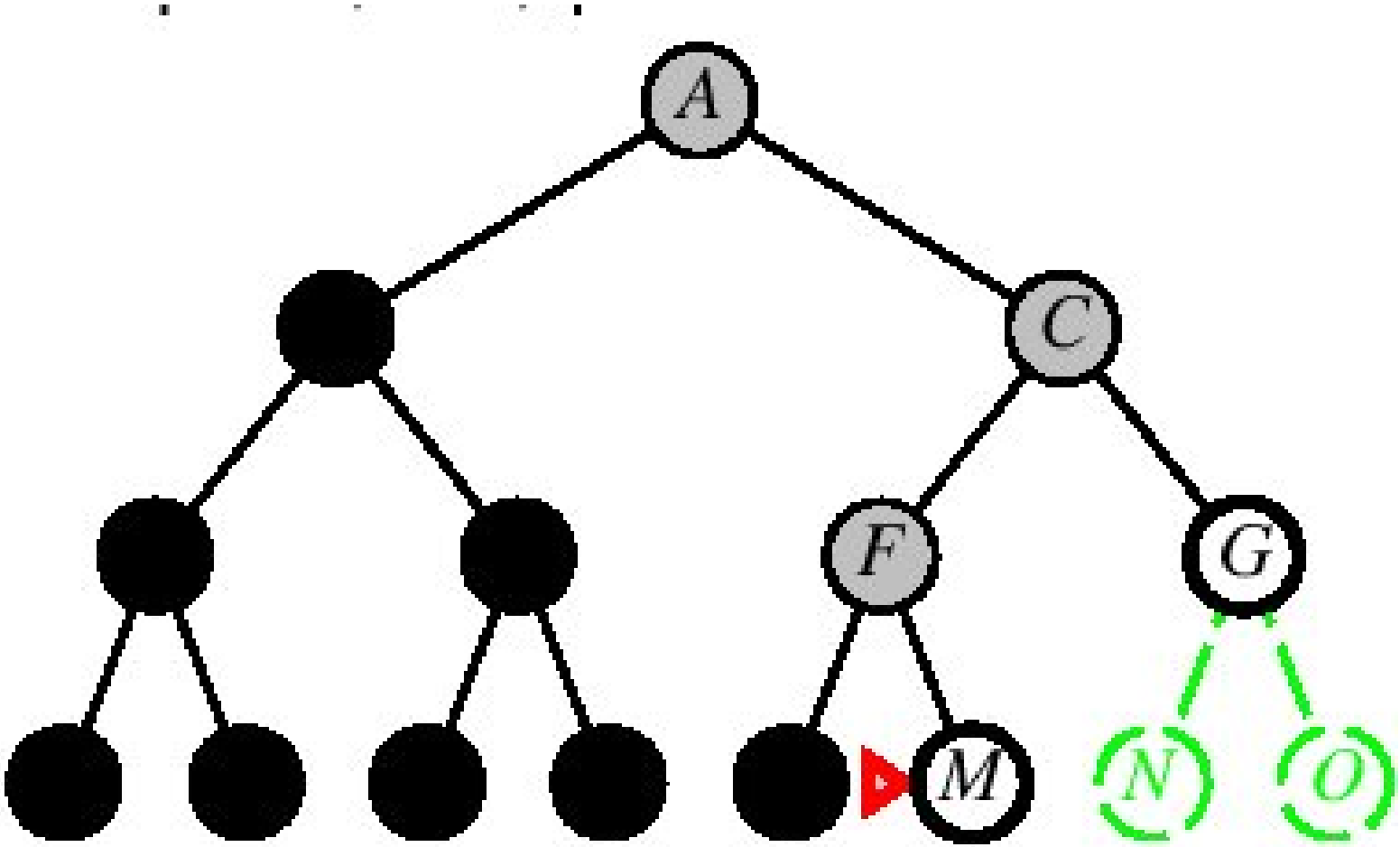
Depth-first search, Cont



Depth-first search, Cont



Depth-first search, Cont



Depth-first search, Cont

- Complete:
- Time complexity:
- Space complexity:
- Optimality:

Depth-first search, Cont

- Complete: No: Fails in infinite depth spaces and spaces with loops.
- Time complexity:
- Space complexity:
- Optimality:

Depth-first search, Cont

- Complete: No: Fails in infinite depth spaces and spaces with loops.
- Time complexity: $O(b^m)$
- Space complexity:
- Optimality:

Depth-first search, Cont

- Complete: No: Fails in infinite depth spaces and spaces with loops.
- Time complexity: $O(b^m)$
- Space complexity: $O(bm)$
- Optimality:

Depth-first search, Cont

- Complete: No: Fails in infinite depth spaces and spaces with loops.
- Time complexity: $O(b^m)$
- Space complexity: $O(bm)$
- Optimality: No

Depth-Limited search

- A kind of DFS with depth l (Nodes at depth l has no children),
- Avoid stucking in loop.
- Can be applied in spaces with infinite (depth)

Depth-Limited search, Cont

```
function DLS (node, depth) returns solution/fail/cutoff
  cutoff-occured= false;
  if GOAL-TEST (problem, STATE[node]) return node
  else if DEPTH [node]= limit then return cutoff
  else for each successor in EXPAND (node, problem) do
    result= DLS (successor, depth)
    if result= cutoff then cutoff-occured= true
    else if result!= failure then return result
  if cutoff-occured return cutoff else return failure
```

Iterative Deeping search

```
function IDS (problem) returns solution or failure
  for d= 0 to MaxInt do
    result= DLS (INITIAL-STATE (problem), d);
    if result<> cutoff then return result
```

Iterative Deeping search, Cont

- Complete:
- Time complexity:
- Space complexity:
- Optimality:

Iterative Deeping search, Cont

- Complete: YES
- Time complexity:
- Space complexity:
- Optimality:

Iterative Deeping search, Cont

- Complete: YES
- Time complexity: $(d+1)b^0 + db^1 + (d-1)b^2 + \cdots + b^d$
- Space complexity:
- Optimality:

Iterative Deeping search, Cont

- Complete: YES
- Time complexity: $(d+1)b^0 + db^1 + (d-1)b^2 + \cdots + b^d$
- Space complexity: $O(bd)$
- Optimality:

Iterative Deeping search, Cont

- Complete: YES
- Time complexity: $(d+1)b^0 + db^1 + (d-1)b^2 + \cdots + b^d$
- Space complexity: $O(bd)$
- Optimality: YES, if step cost$=1$

Bidirectional search

- Search from both sides:
- From goal backward toward initial state.
- From initial state forward toward goal.
- $O(b^d)$ can be decreased to $O(2b^{\frac{d}{2}}) = O(b^{\frac{d}{2}})$

Bidirectional search, Cont

- It is not applicable for all problems.
- It is hard situation when we have more than one goal.
- The parts can be searched with different search methods.

Bidirectional search, Cont

- Complete: YES
- Time complexity: $O(b^{\frac{d}{2}})$
- Space complexity: $O(b^{\frac{d}{2}})$
- Optimality: YES

Summary of algorithms

	Complete?	Time	Space	Optimal
Breadth-First	YES($d < \infty$)	b^{d+1}	b^{d+1}	YES (Costs=1)
Uniform-Cost	YES($\epsilon \geq 0$)	$b^{\frac{C^*}{\epsilon}}$	$b^{\frac{C^*}{\epsilon}}$	YES ($\epsilon > 0$)
Depth-First	NO	b^m	bm	No
Depht-Limited	YES($l \geq d$)	b^l	bl	No
Iterative Deepening	YES	b^d	bd	Yes(cost=1)
Bidirectional Search	YES	$b^{\frac{d}{2}}$	$b^{\frac{d}{2}}$	Yes