Uninformed Search (Blind Search)

Boris Velichkov

Exercises by Week

1. Introduction
2. **Uninformed *(Blind)* Search**
3. **Informed *(Heuristic)* Search**
4. **Constraint Satisfaction Problems**
5. **Genetic Algorithms**
6. **Games**
7. Introduction to Machine Learning
8. ***k*-Nearest Neighbors**
9. **Naïve Bayes Classifier**
10. **Decision Tree**
11. ***k*Means**
12. **Neural Networks**
13. *Additional Topics, Questions and Homeworks' Presentations*
14. *Additional Topics, Questions and Homeworks' Presentations*
15. *Additional Topics, Questions and Homeworks' Presentations*

Problem Solving and Search

- Basic Concepts
  - State Space
  - Representation of the State Space
  - Search Strategies Evaluation
  - Global Search vs Local Search
  - Uninformed Search vs Informed Search
  - Graph and Tree Traversal

State Space

- **State**: a representation (formulation) of the task in the process of its solution.
  - *Initial State* (presented with ***S***)
  - *Intermediate State* (presented with any capital letter except ***S*** and ***G***)
  - *Goal State* (presented with ***G***, if there are more: ***G_1***, ***G_2***, etc.)
- **Successor Function *(Operator)***: obtaining one state from another
- **Path Cost**: additive; e.g., sum of distances, number of actions executed, etc.
- **State Space**: the totality of all possible states that can be obtained from a given initial state.

** A **solution** is a sequence of actions leading from the initial state to a goal state*

Representation of the State Space

- **Graph** or **Tree** where each *state* is represented with **Node** and the *successor function (operator)* is represented with **Edge**.
- When the *state space* can be represented as a *tree*, it is called **Search Tree**.
  - The *initial state* is the *root* of the tree.
  - The terminal states and the *goal state* are represented with *leaves*.

Search Strategies Evaluation

Strategies are evaluated along the following dimensions:

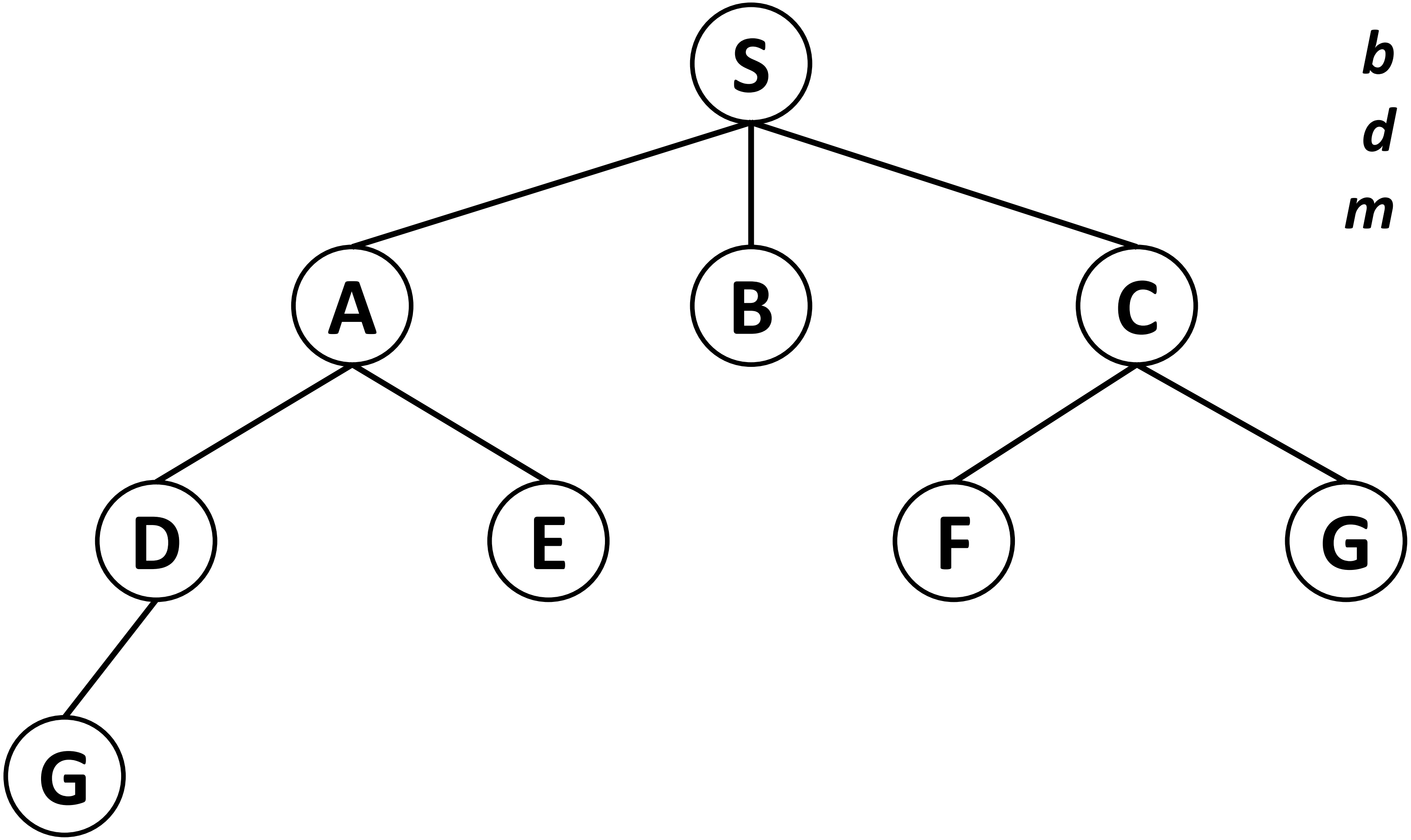
- **Completeness**: does it always find a solution if one exists?
- **Optimality**: does it always find a least-cost solution?
- **Time Complexity**: number of nodes generated/expanded
- **Space Complexity**: maximum number of nodes in memory

** We look at the* ***worst-case*** *complexity (Big O notation).*

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 ∞)

Search Strategies Evaluation



b = 3

d = 2

m = 3

Global Search vs Local Search

- **Global Search**: They look all over the state space. If necessary, all states will be traversed.
- **Local Search**: They only look at the local area, so they can only look at states in this area. If the solution is outside of it, they will not be able to find it.

** Generally Local Search is not used for finding a path from state A to state B.*

Local Search

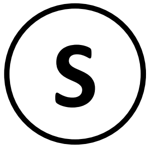
- **Local Search** in *Artificial Intelligence* is an optimizing algorithm to find the optimal solution more quickly.
- **Local search algorithms** are used when we care only about a solution but not the path to a solution.
- **Local search** is a heuristic method for solving computationally hard optimization problems.
- **Local search** can be used on problems that can be formulated as finding a solution maximizing a criterion among a number of candidate solutions.
- **Local search algorithms** move from solution to solution in the space of candidate solutions (the search space) by applying local changes, until a solution deemed optimal is found or a time bound is elapsed.

Uninformed *(Blind)* Search vs Informed *(Heuristic)* Search

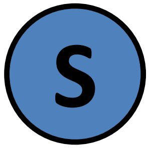
- **Uninformed Search**: Uninformed strategies use only the information available in the problem definition.
  - *Examples: DFS, BFS, UCS, DLS, IDS.*
- **Informed Search**: Informed strategies have information on the *goal state* which helps in more efficient searching. This information is obtained by a function *(heuristic)* that estimates how close a state is to the *goal state*.
  - *Examples: Greedy Best-First Search, A*, Beam Search, Hill Climbing.*

Graph and Tree Traversal

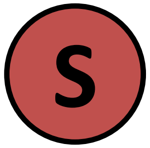
- **Node**: The *graphs* and *trees* consist of *nodes* and *edges*. When a *node* is neither *“visited”* nor *“expanded”*, we do not mark it in any way (sits uncolored).
  - *Example:*



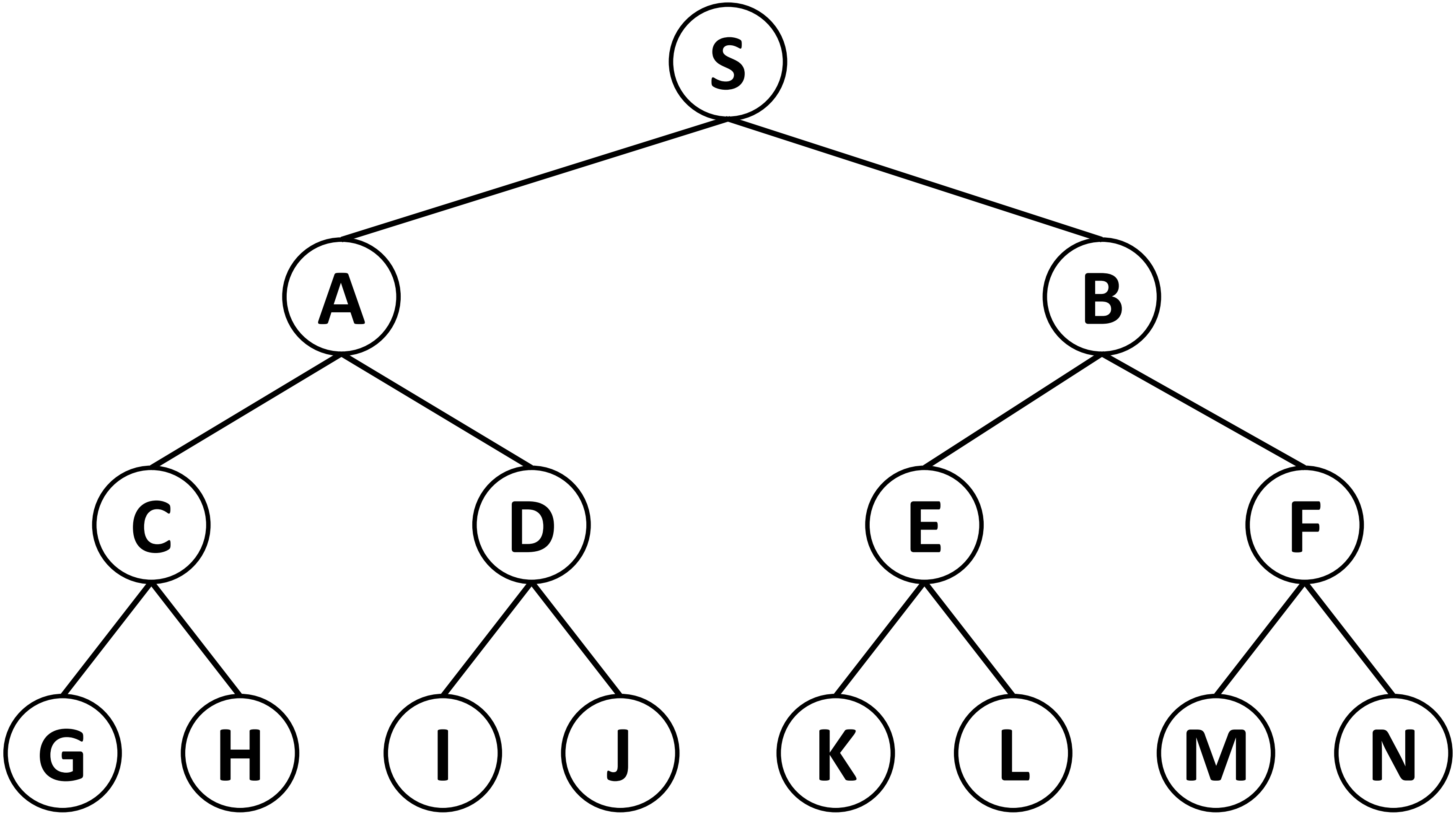
- **Visited Node**: When we visit a *node* or in other words add it to the data structure we use we call it *“visited node”*. We mark it in blue.
  - *Example:*



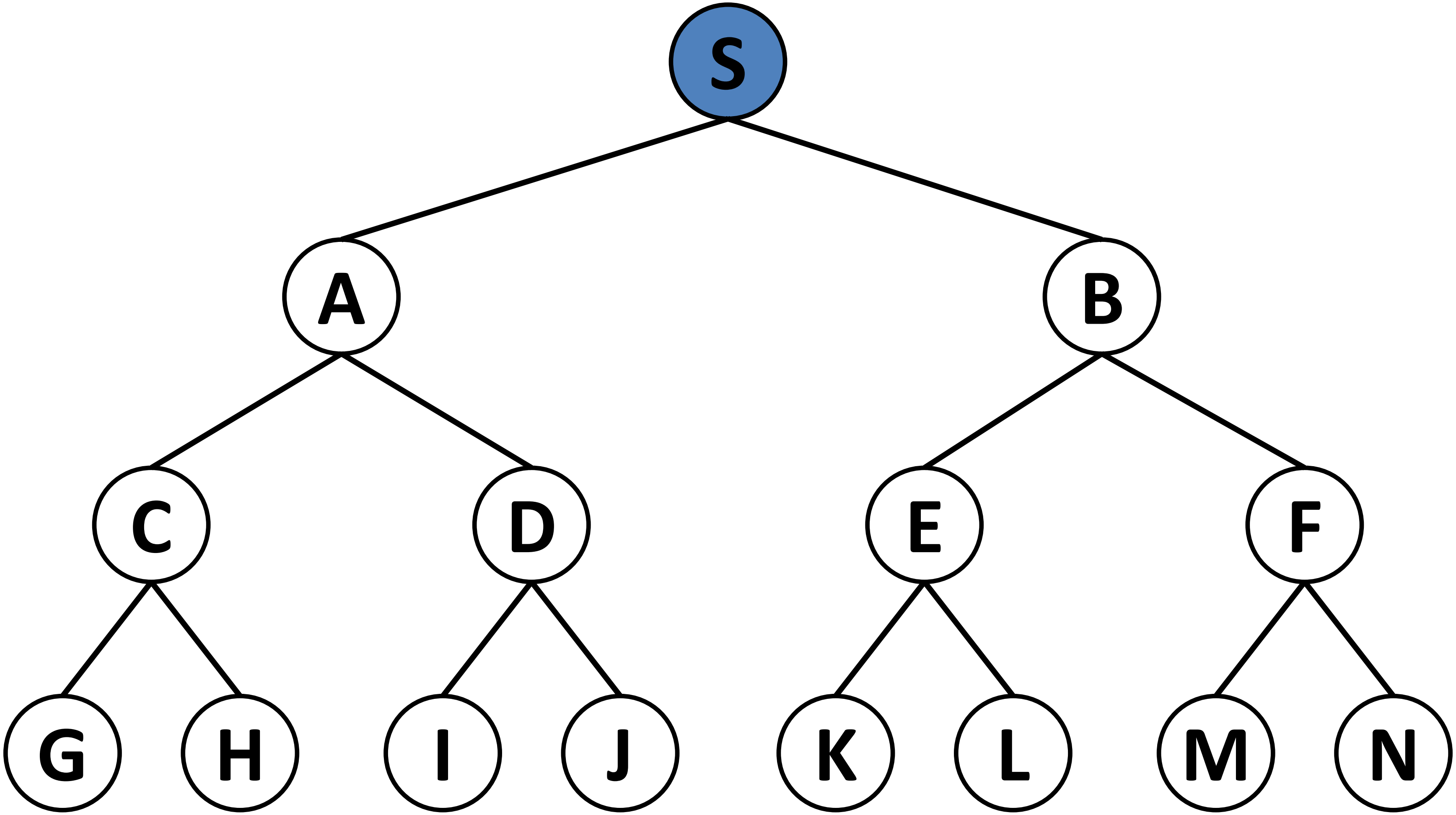
- **Expanded Node**: When we expand a *node* or in other words remove it from the data structure we use and add its children in this data structure (of course if the node has children) we call it *“expanded node”*. We mark it in red.
  - *Example:*



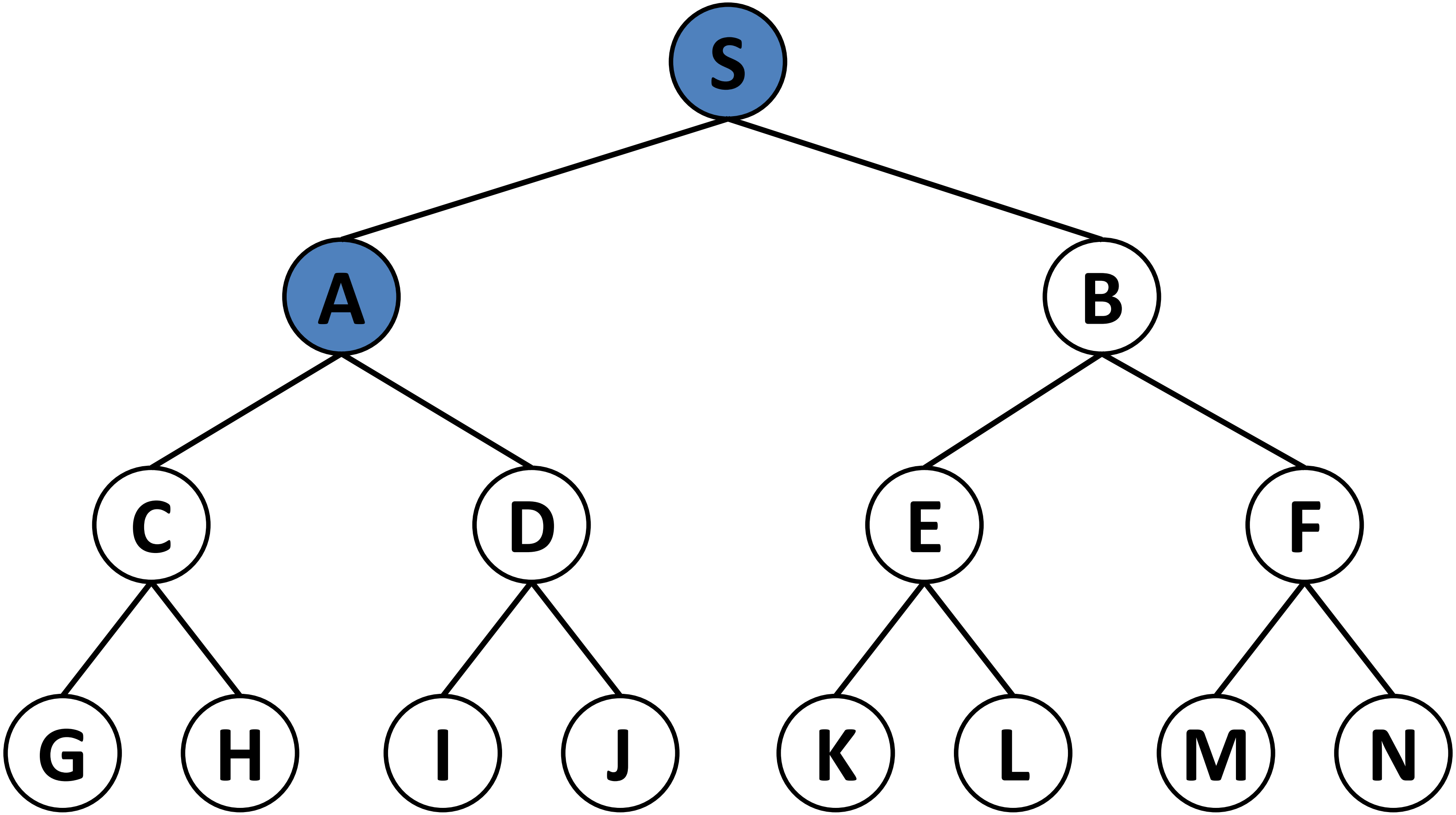
Graph and Tree Traversal: DFS



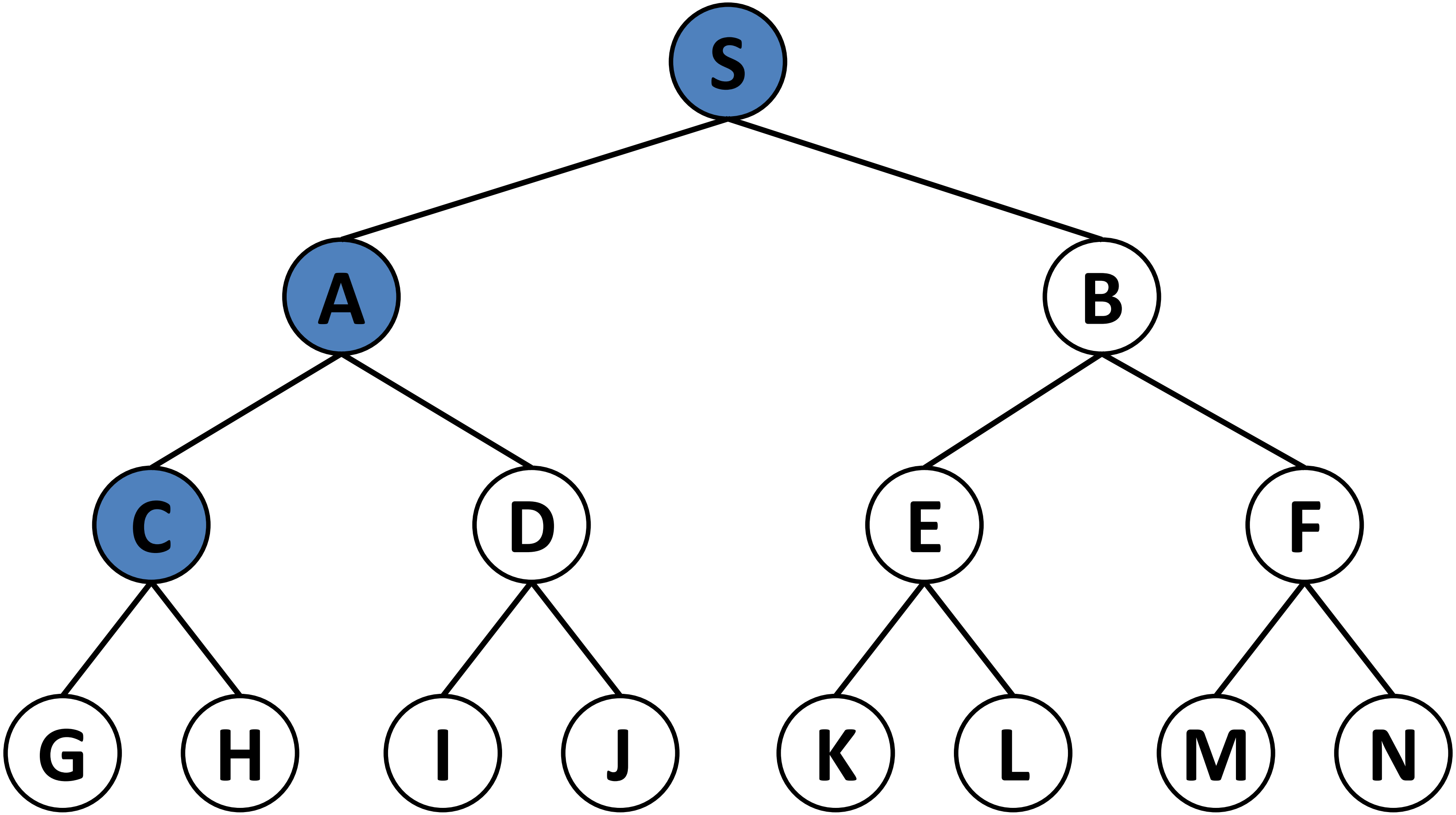
Graph and Tree Traversal: DFS



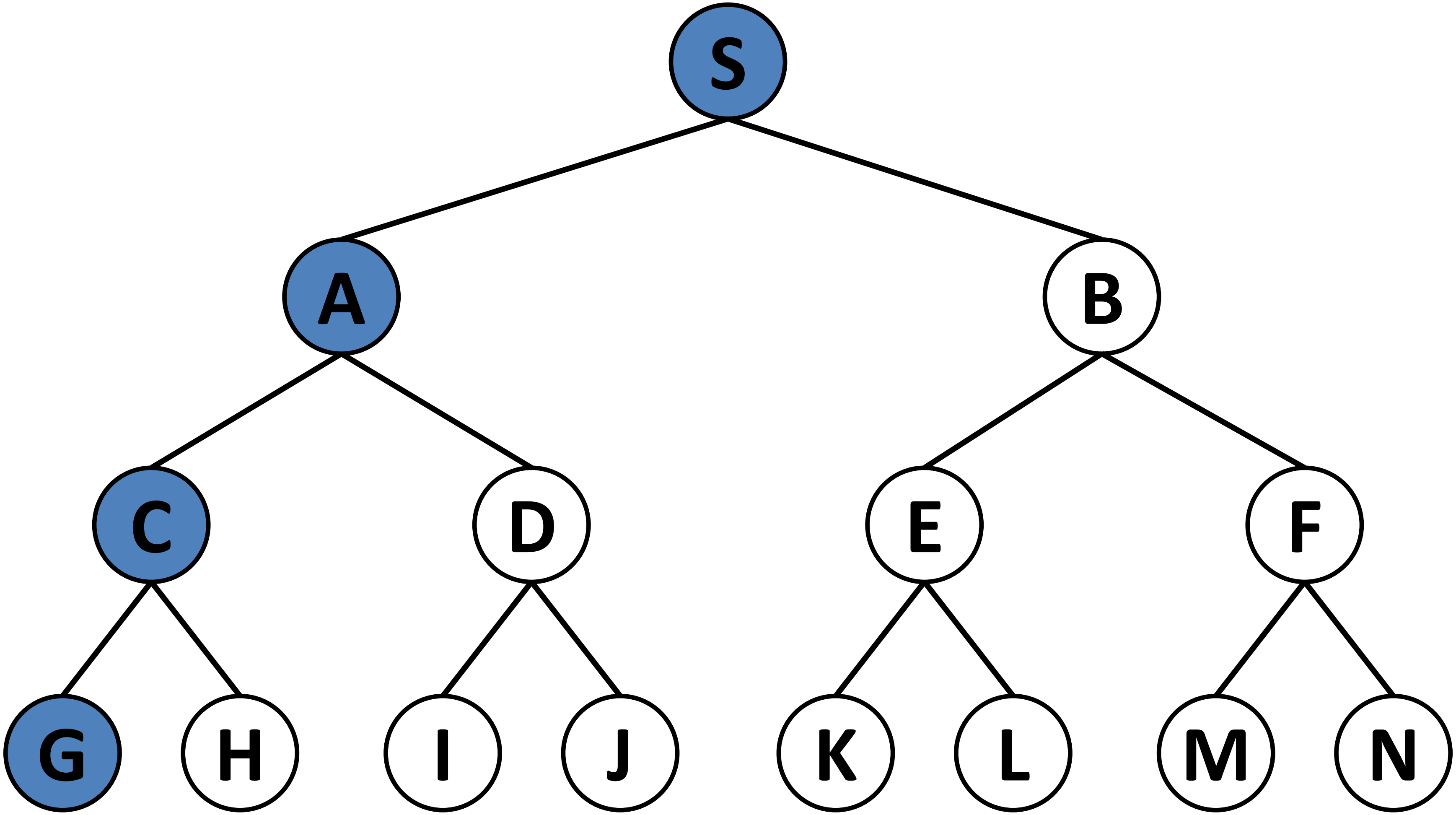
Graph and Tree Traversal: DFS



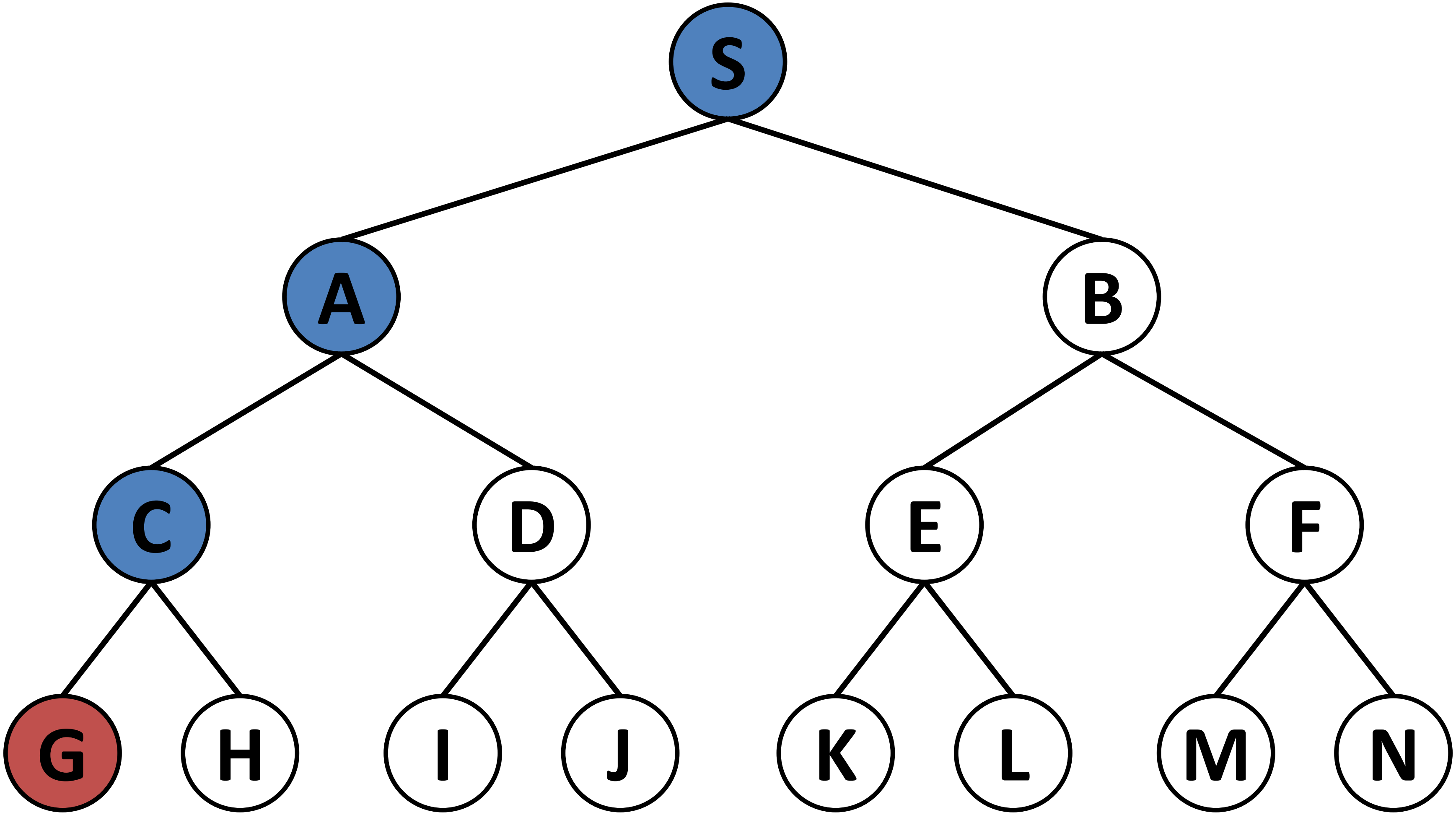
Graph and Tree Traversal: DFS



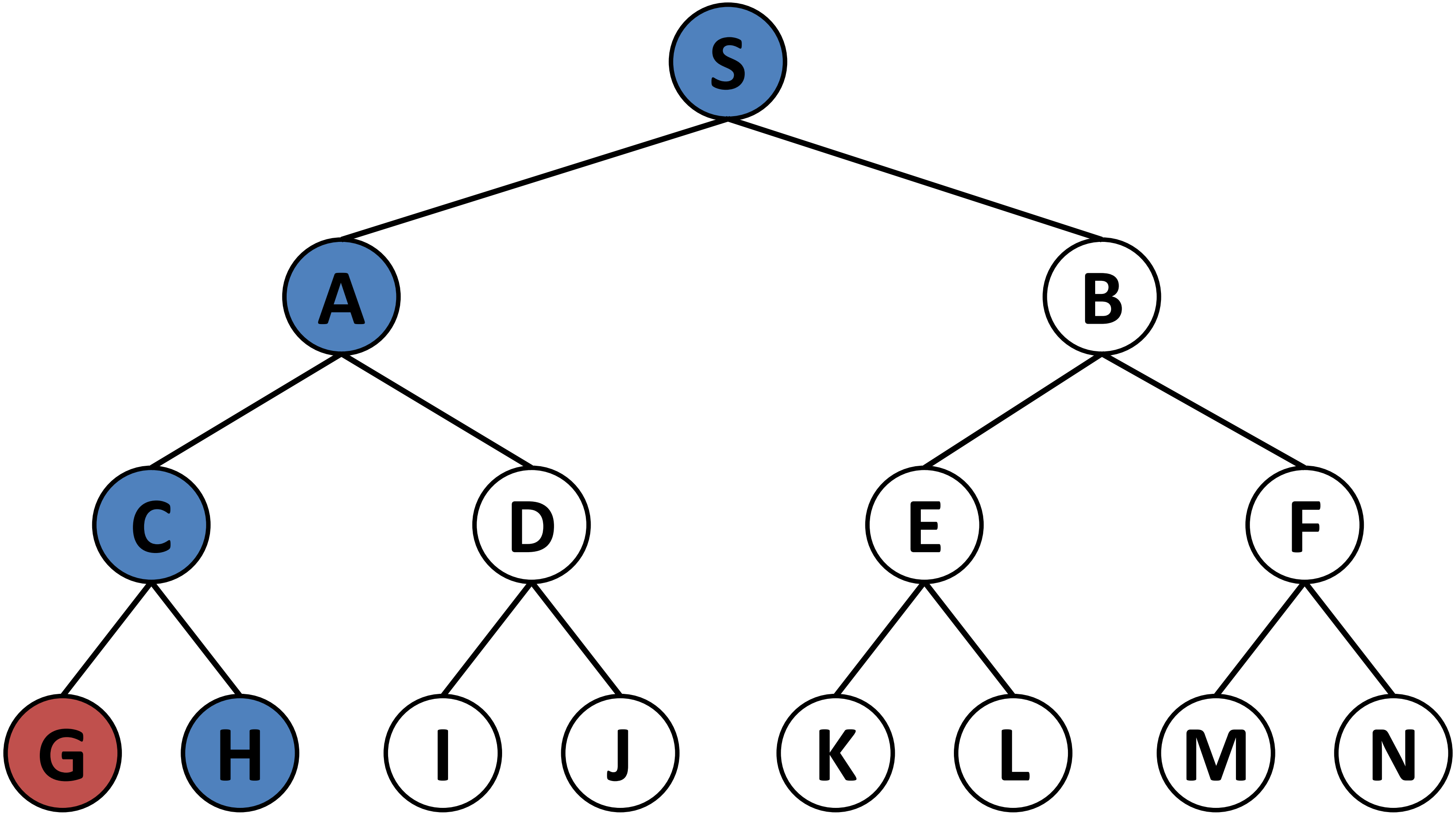
Graph and Tree Traversal: DFS



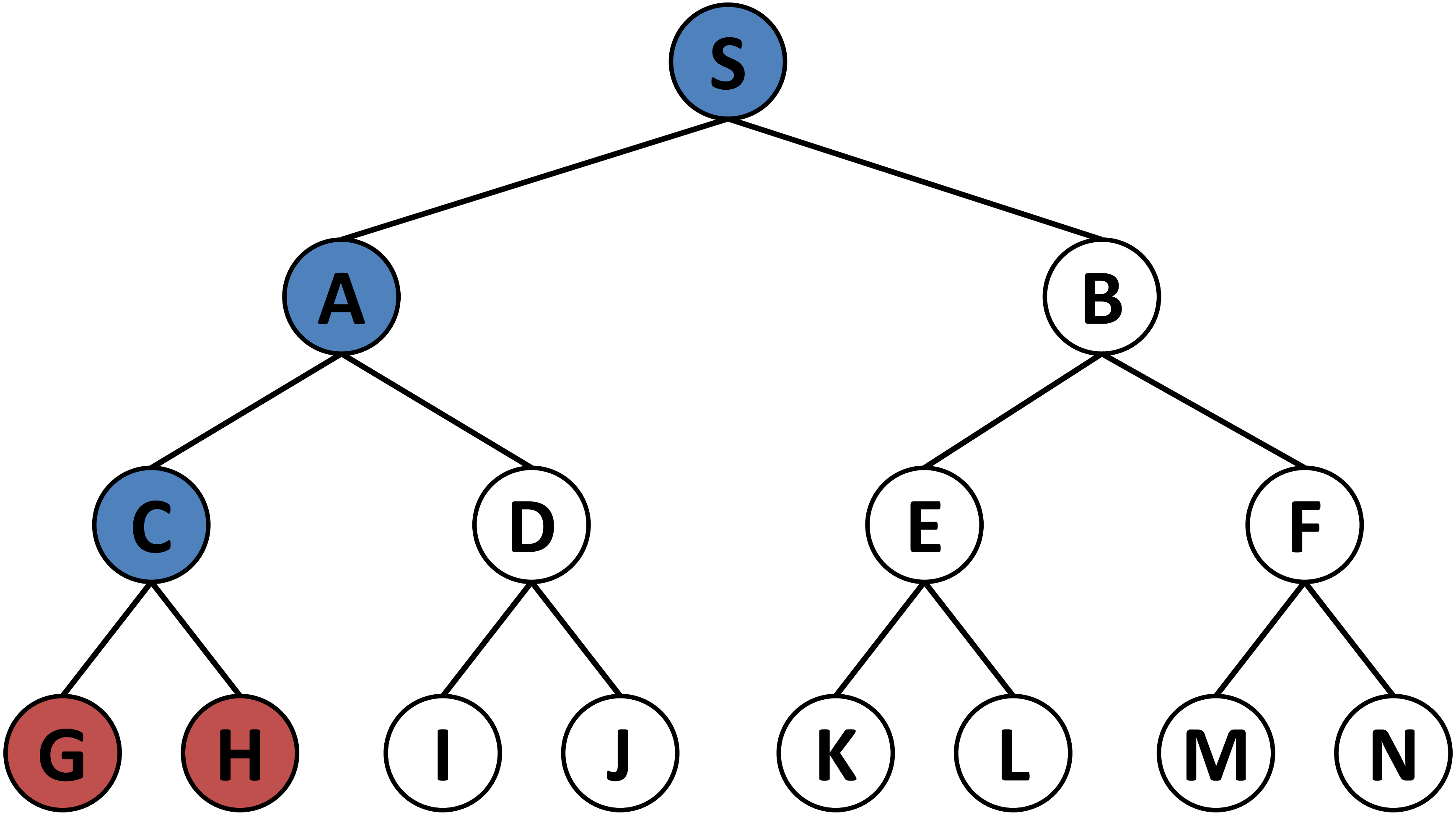
Graph and Tree Traversal: DFS



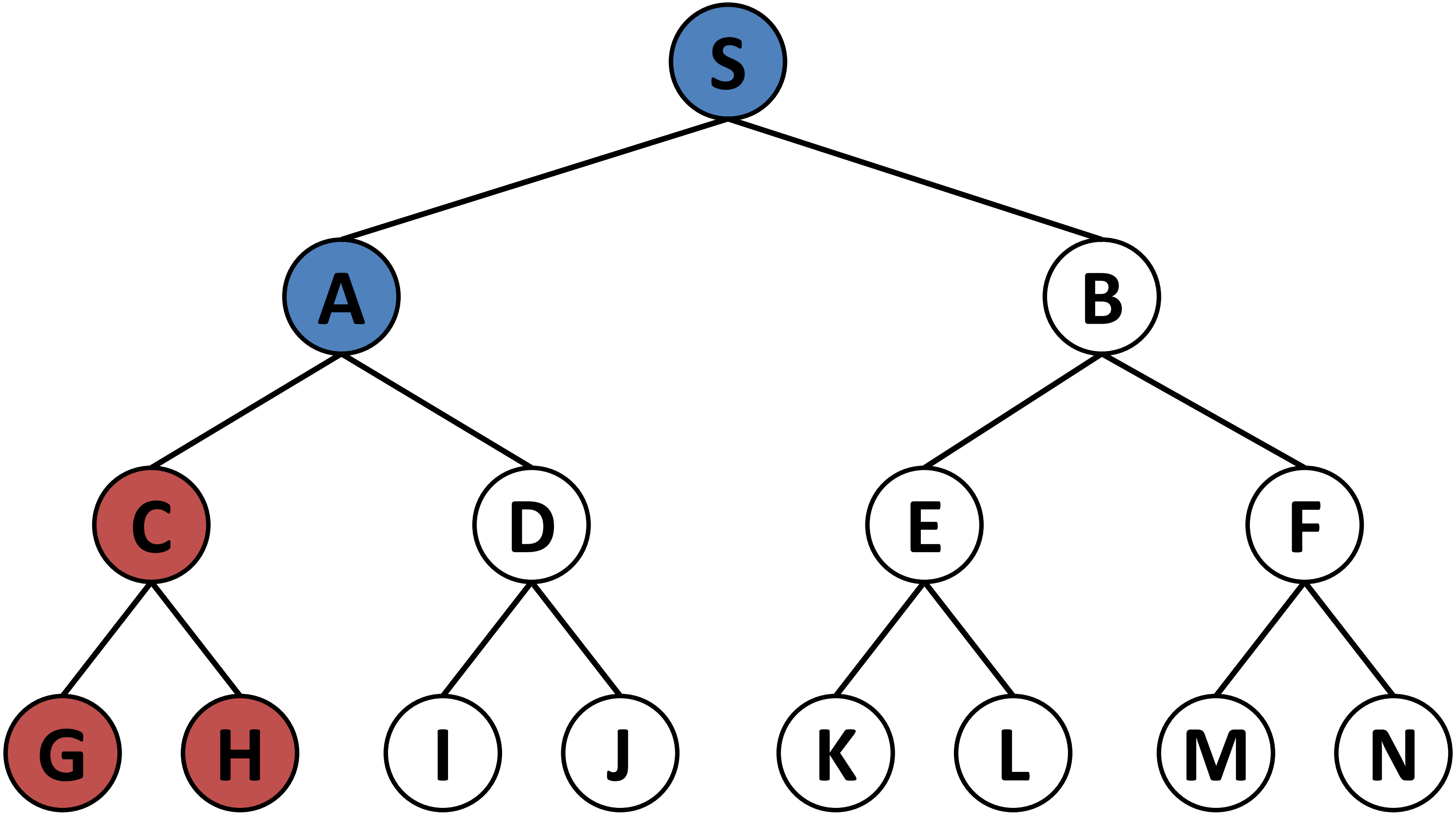
Graph and Tree Traversal: DFS



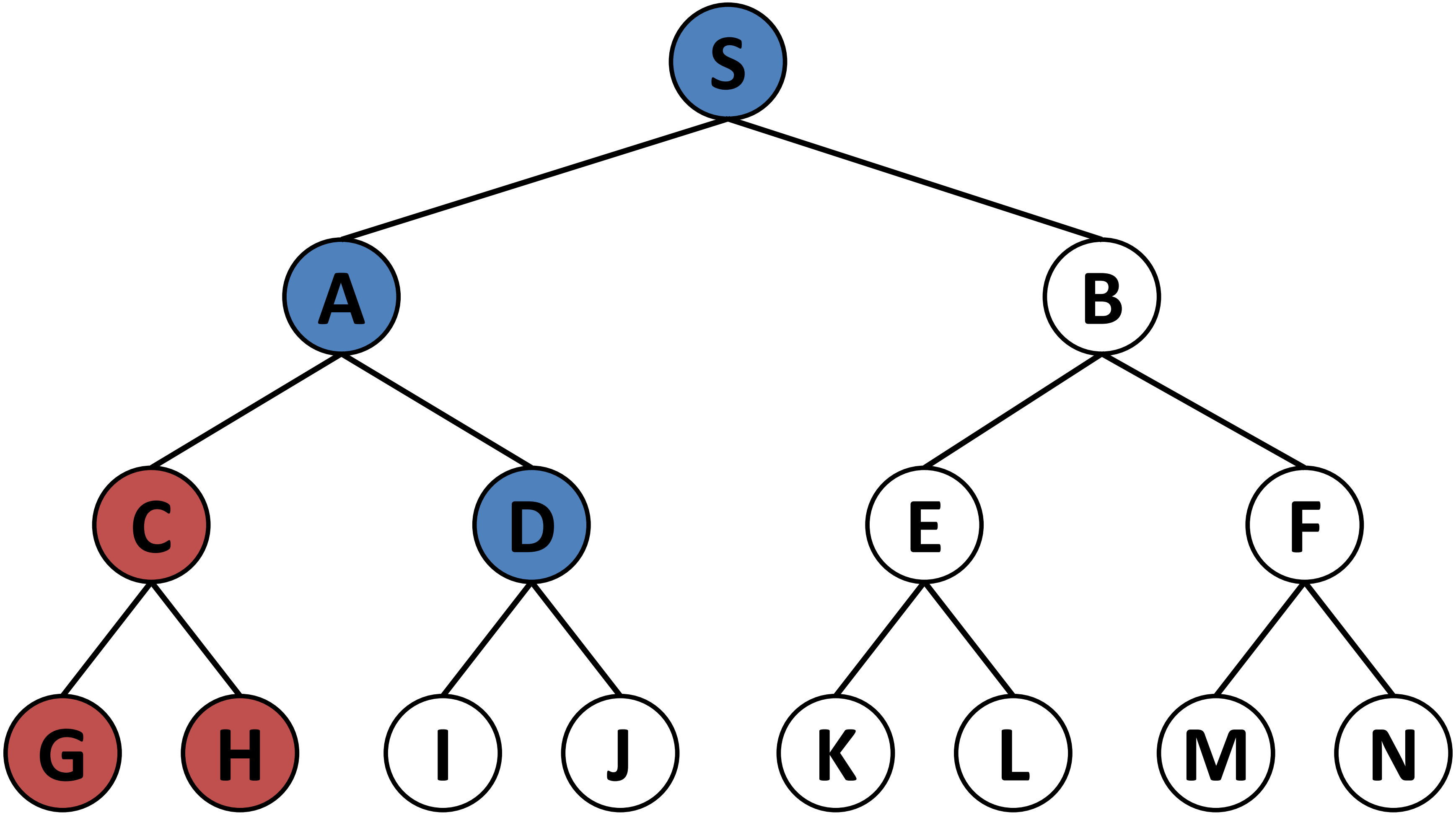
Graph and Tree Traversal: DFS



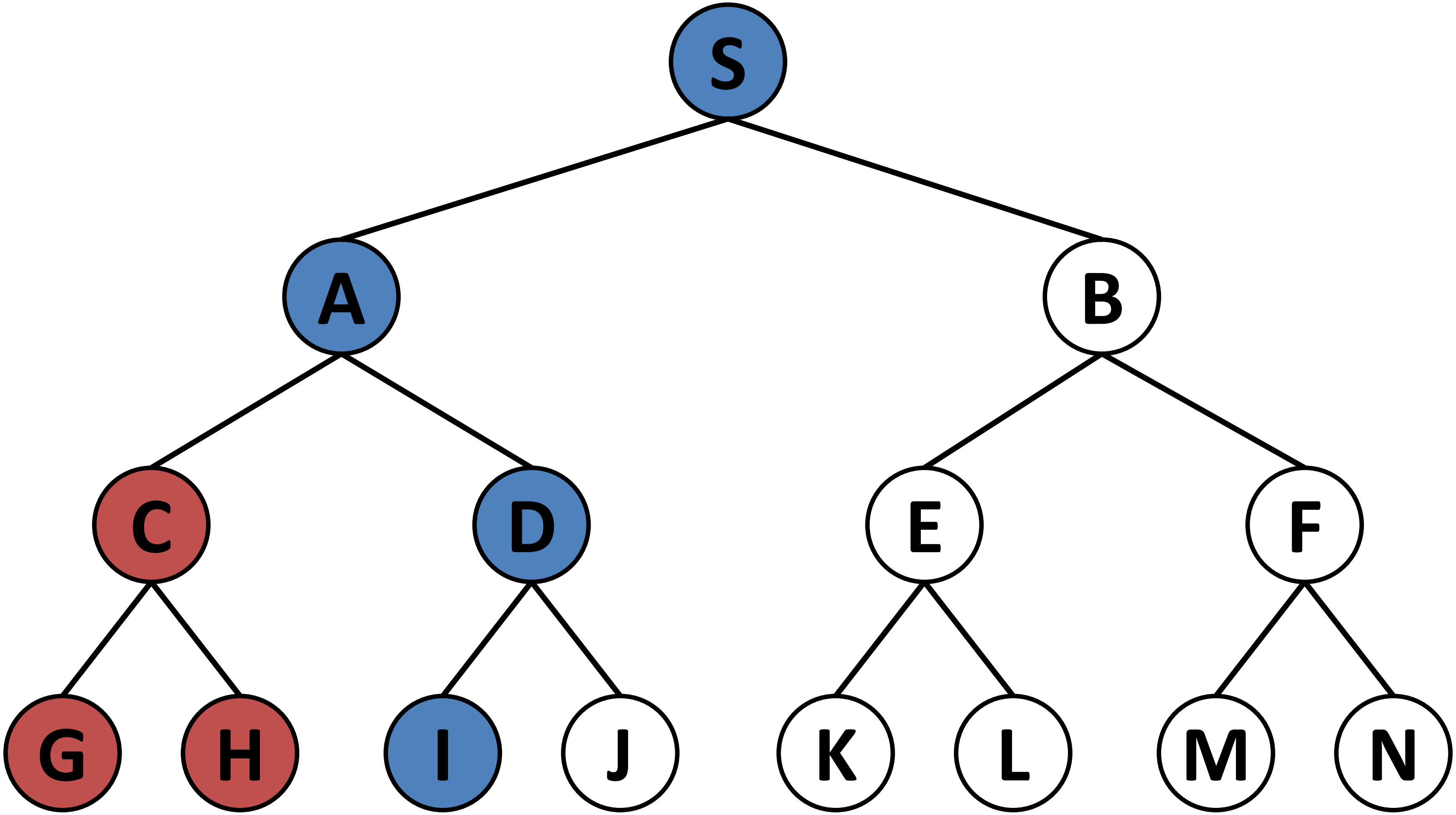
Graph and Tree Traversal: DFS



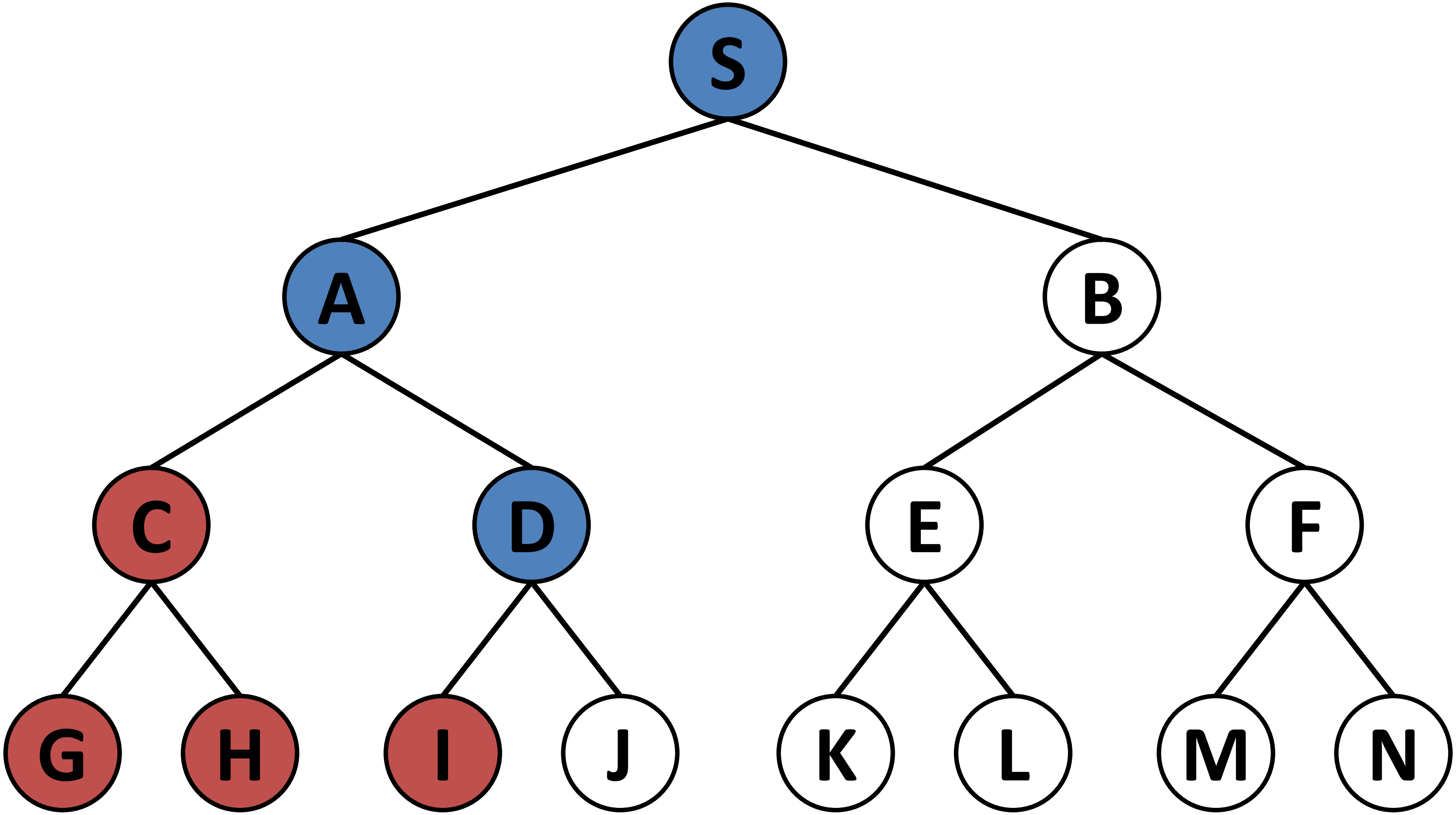
Graph and Tree Traversal: DFS



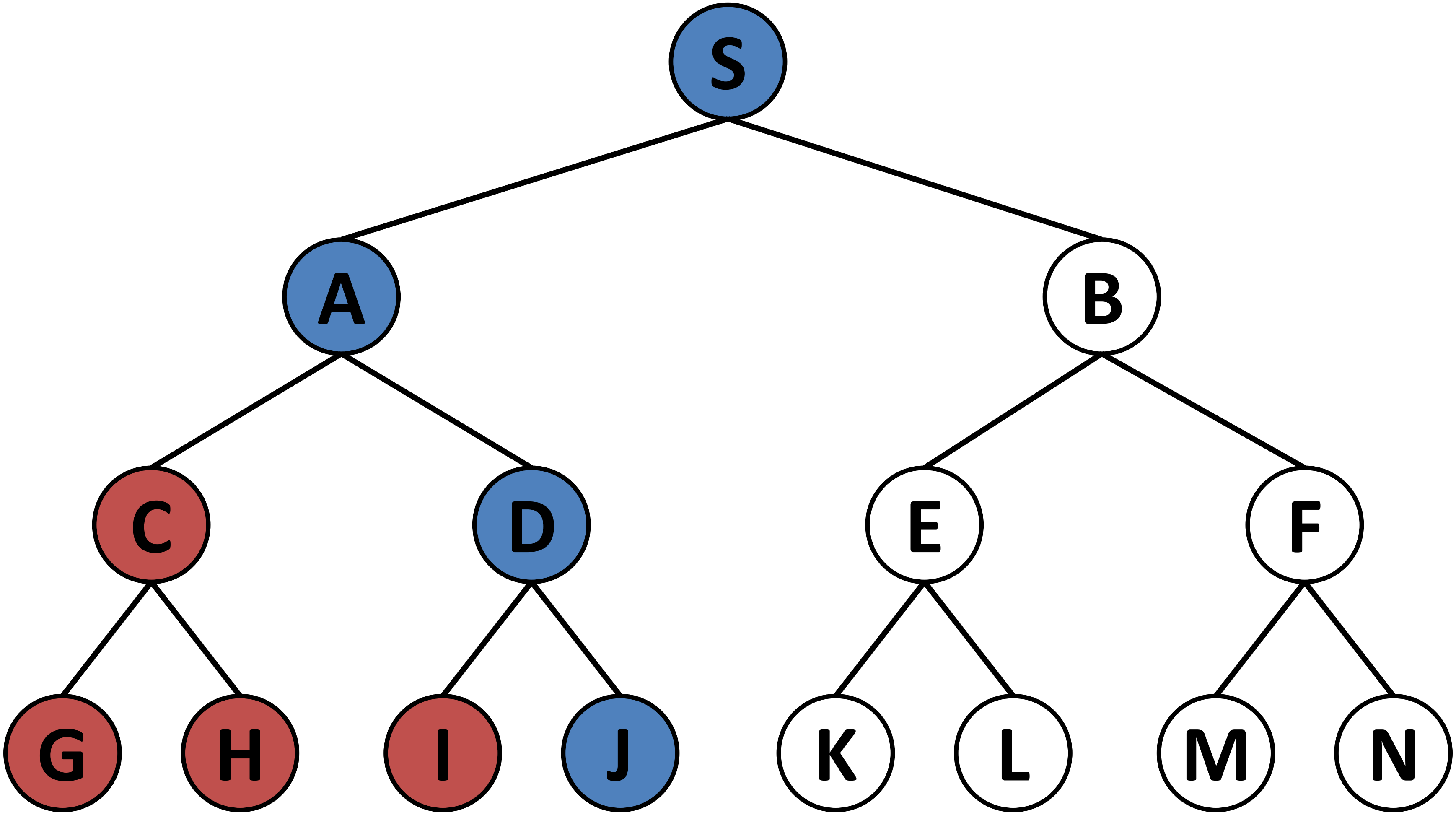
Graph and Tree Traversal: DFS



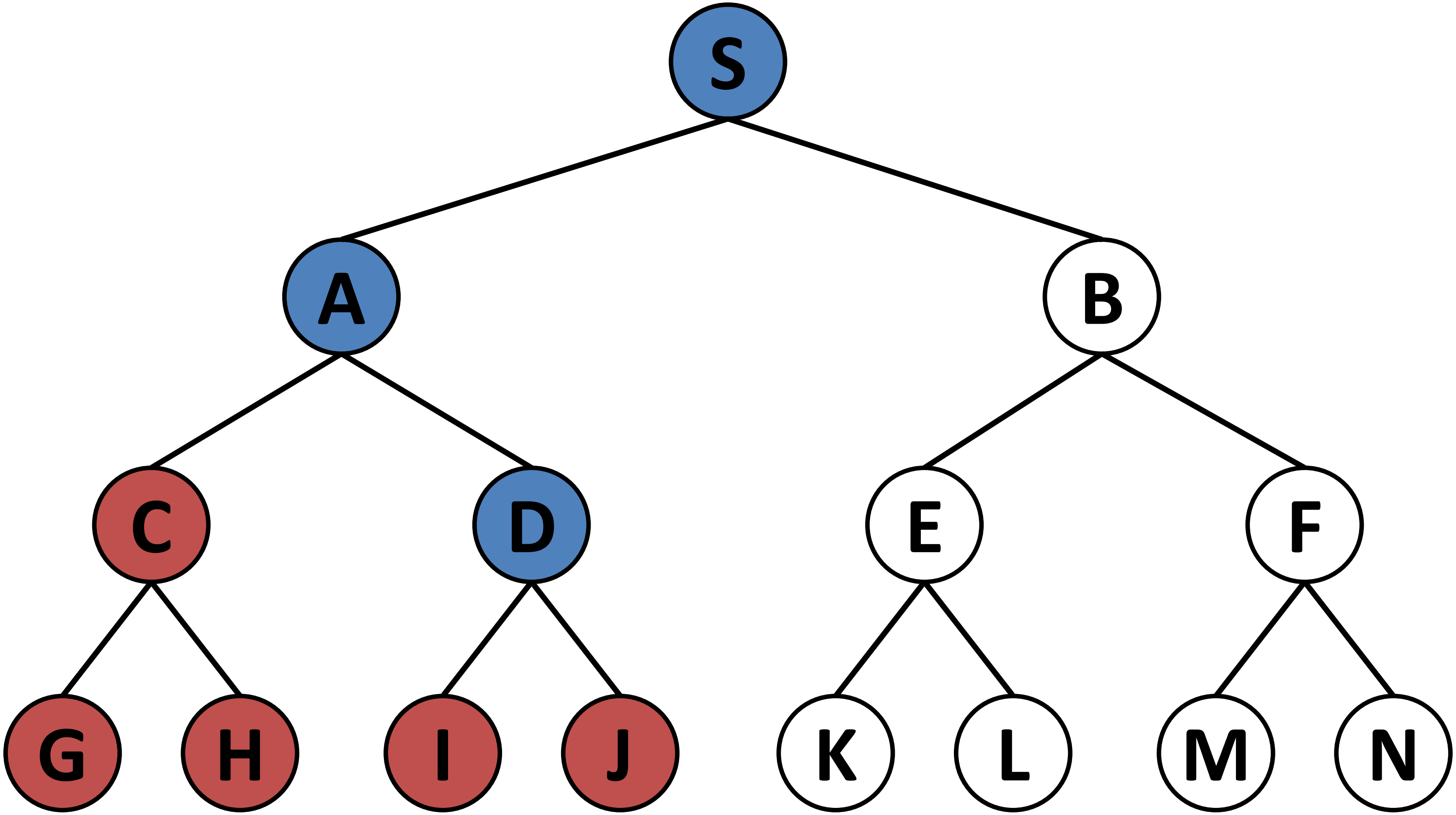
Graph and Tree Traversal: DFS



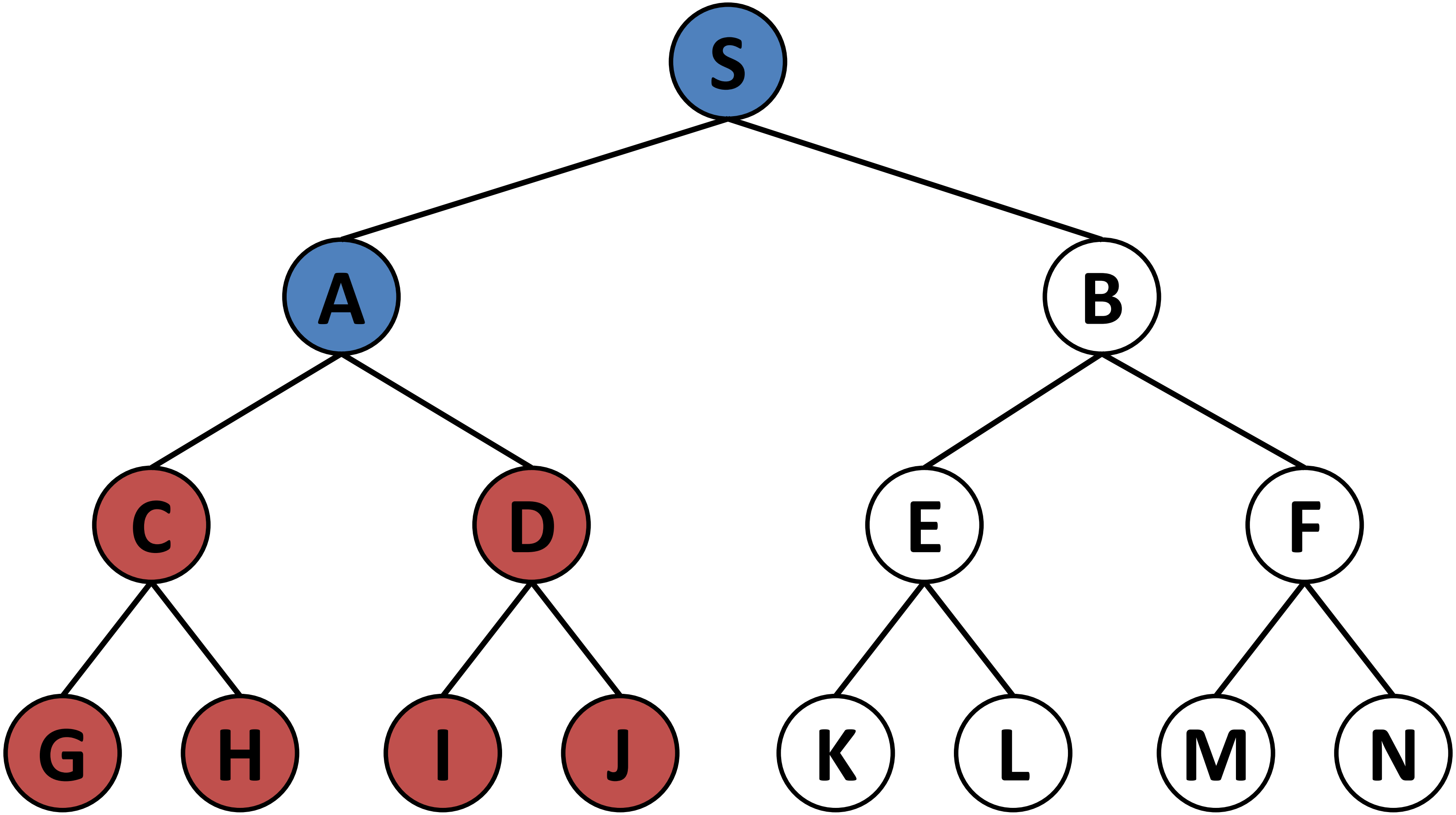
Graph and Tree Traversal: DFS



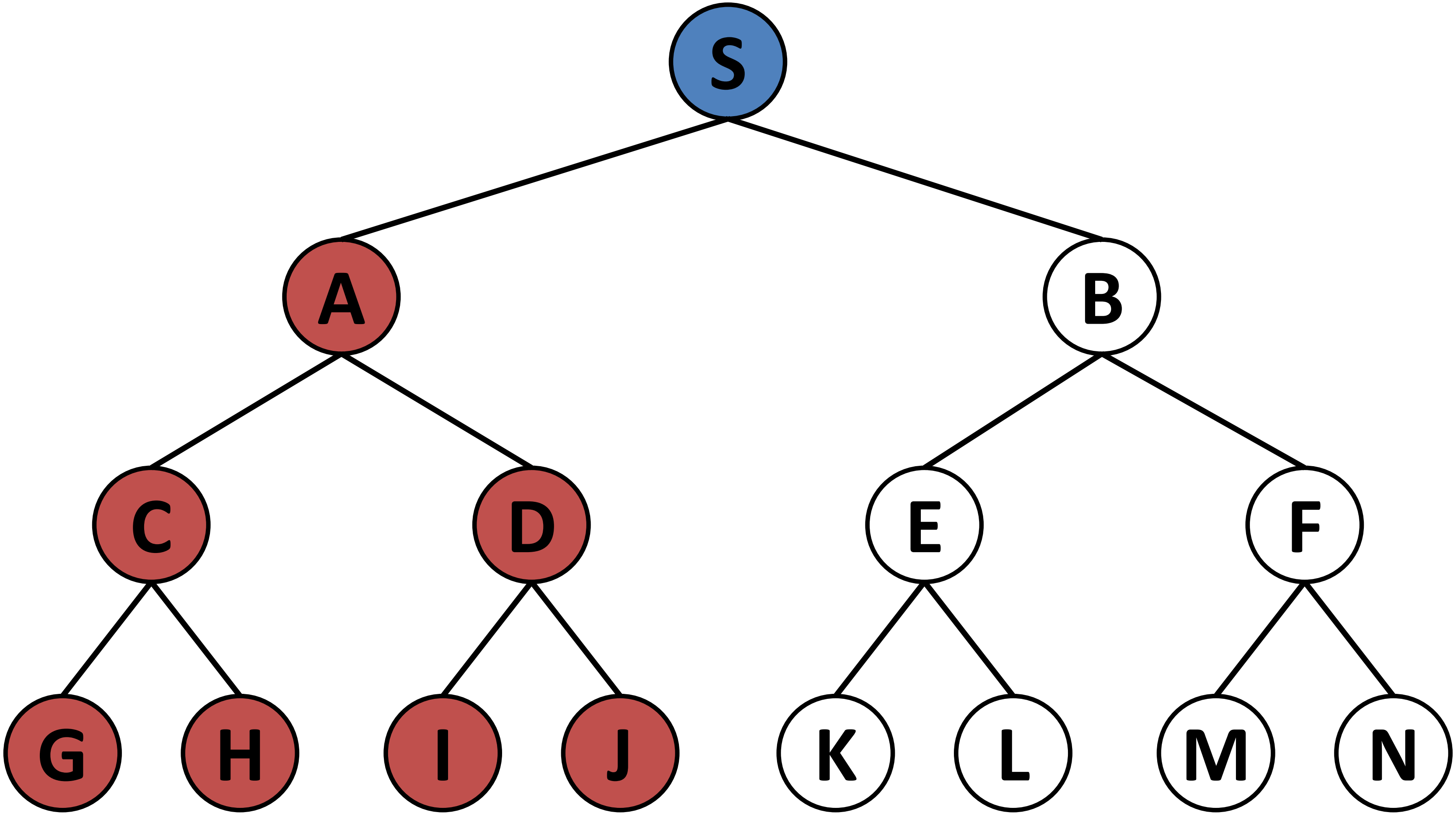
Graph and Tree Traversal: DFS



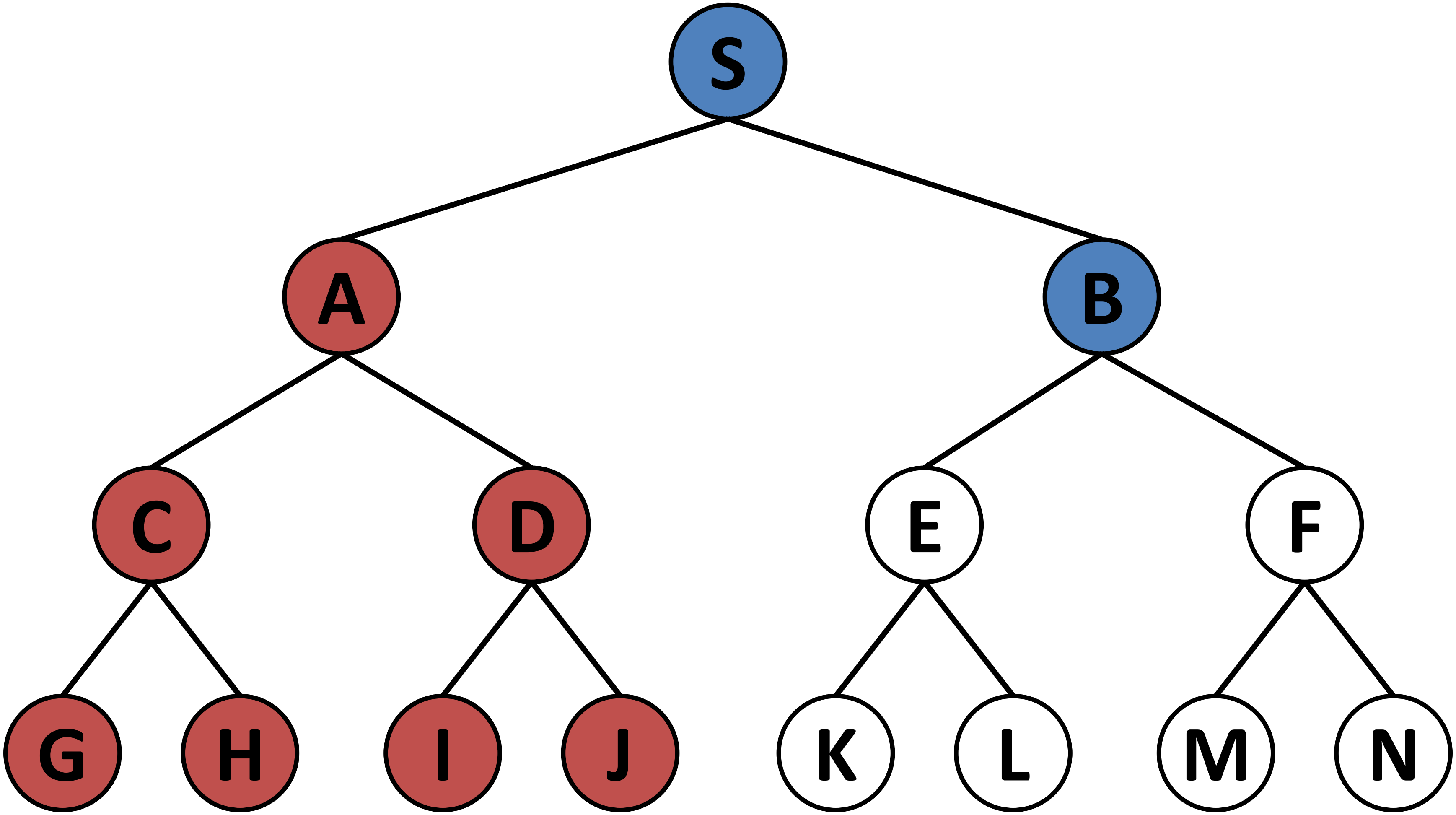
Graph and Tree Traversal: DFS



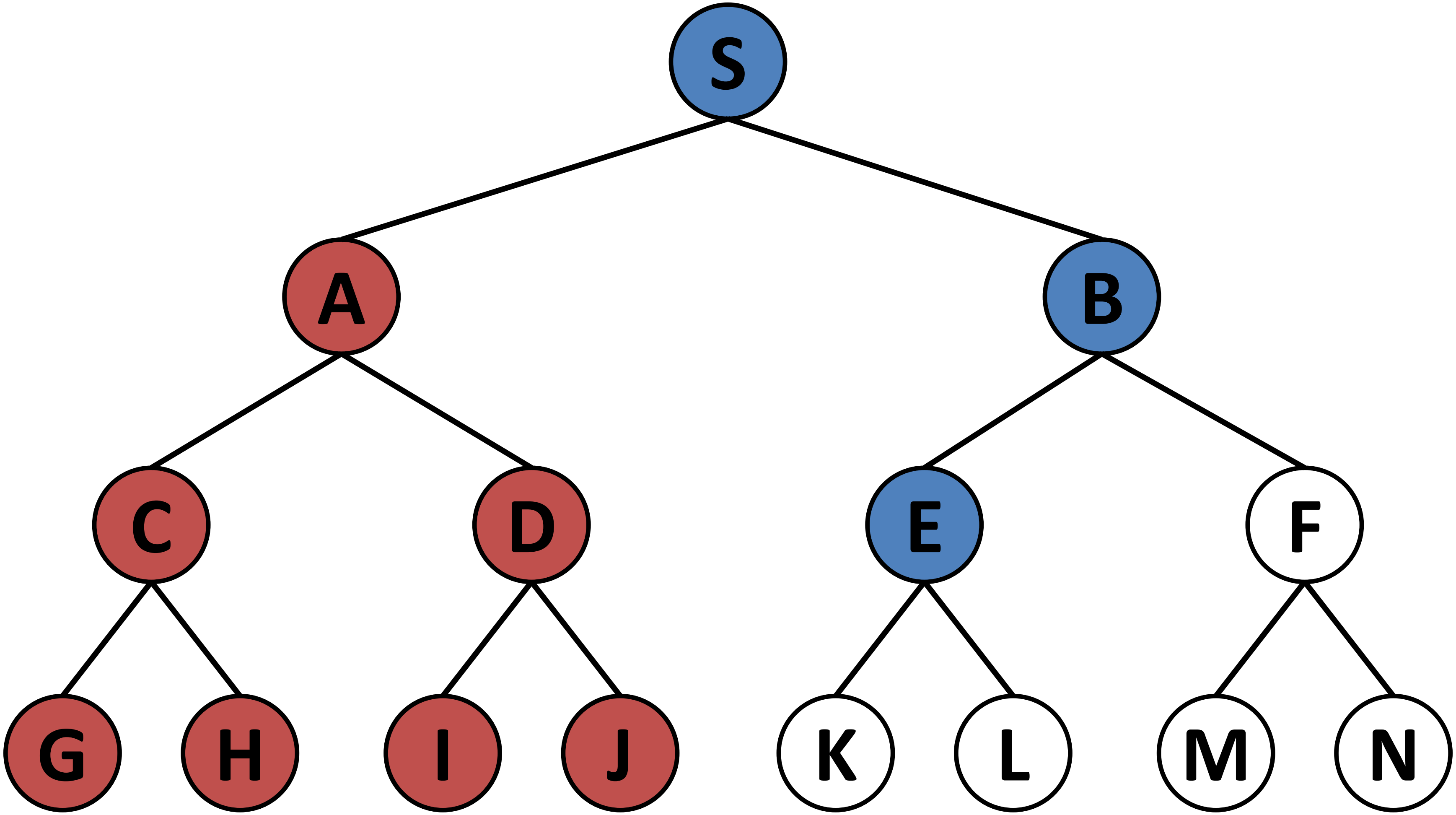
Graph and Tree Traversal: DFS



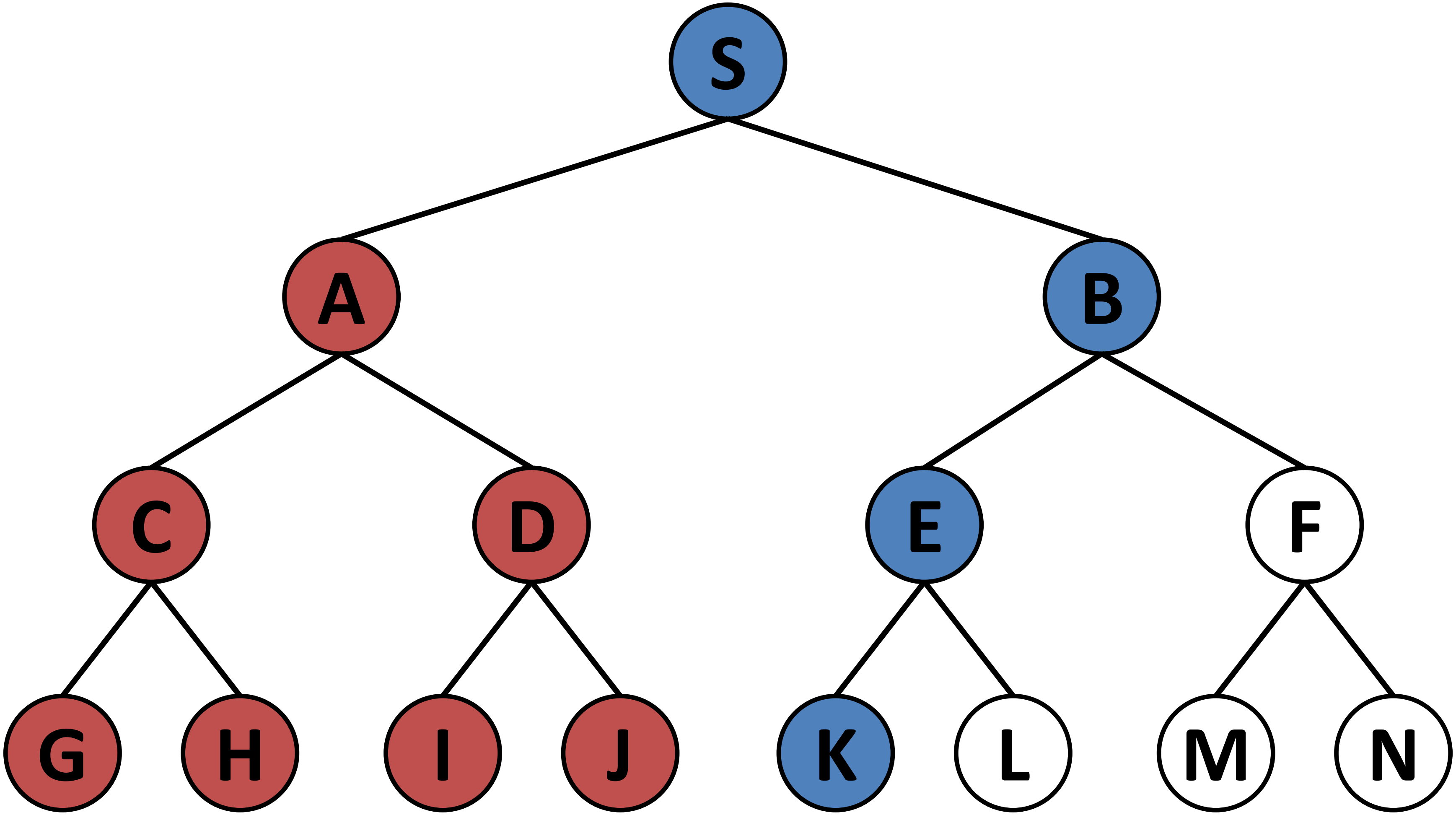
Graph and Tree Traversal: DFS



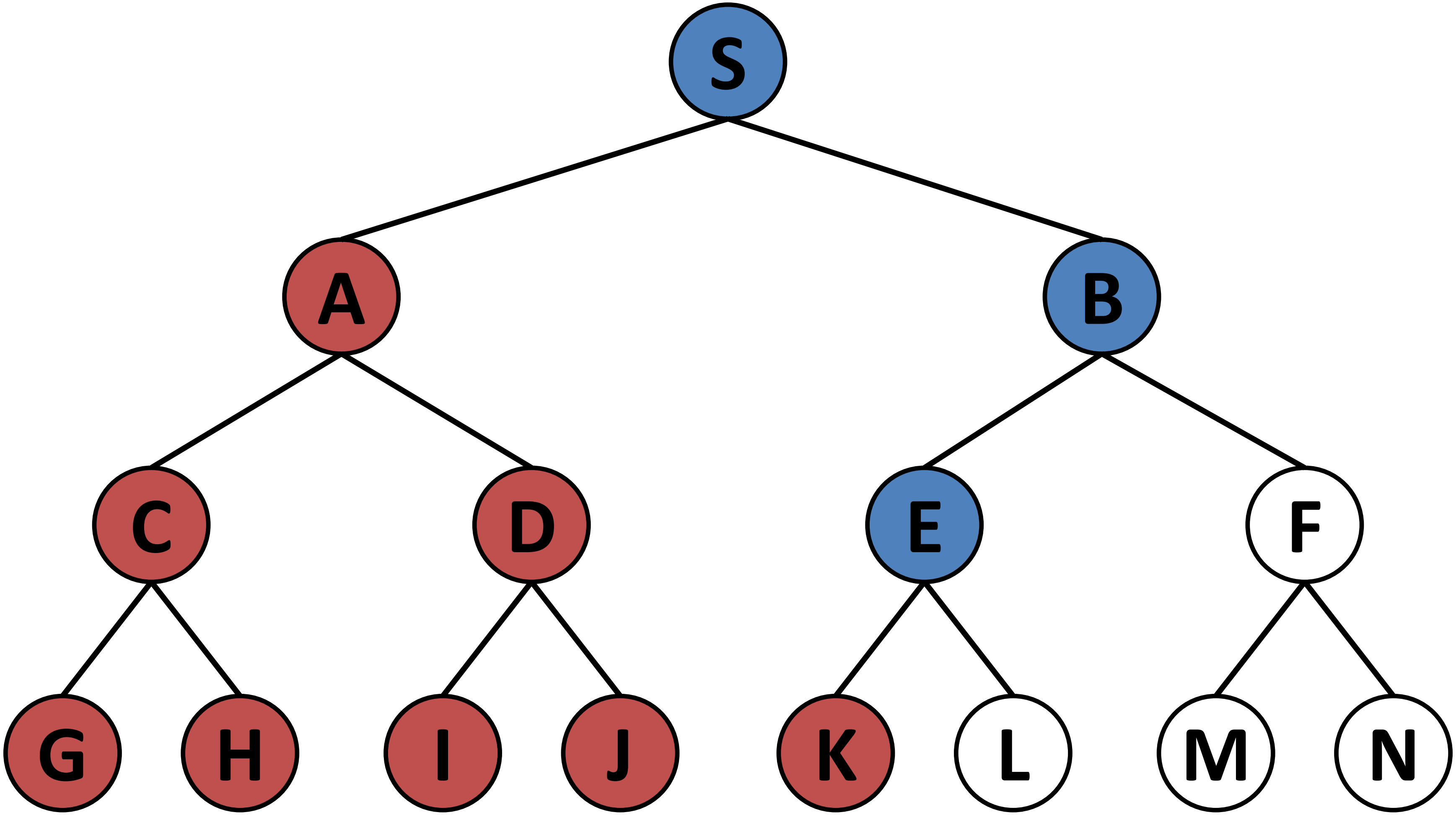
Graph and Tree Traversal: DFS



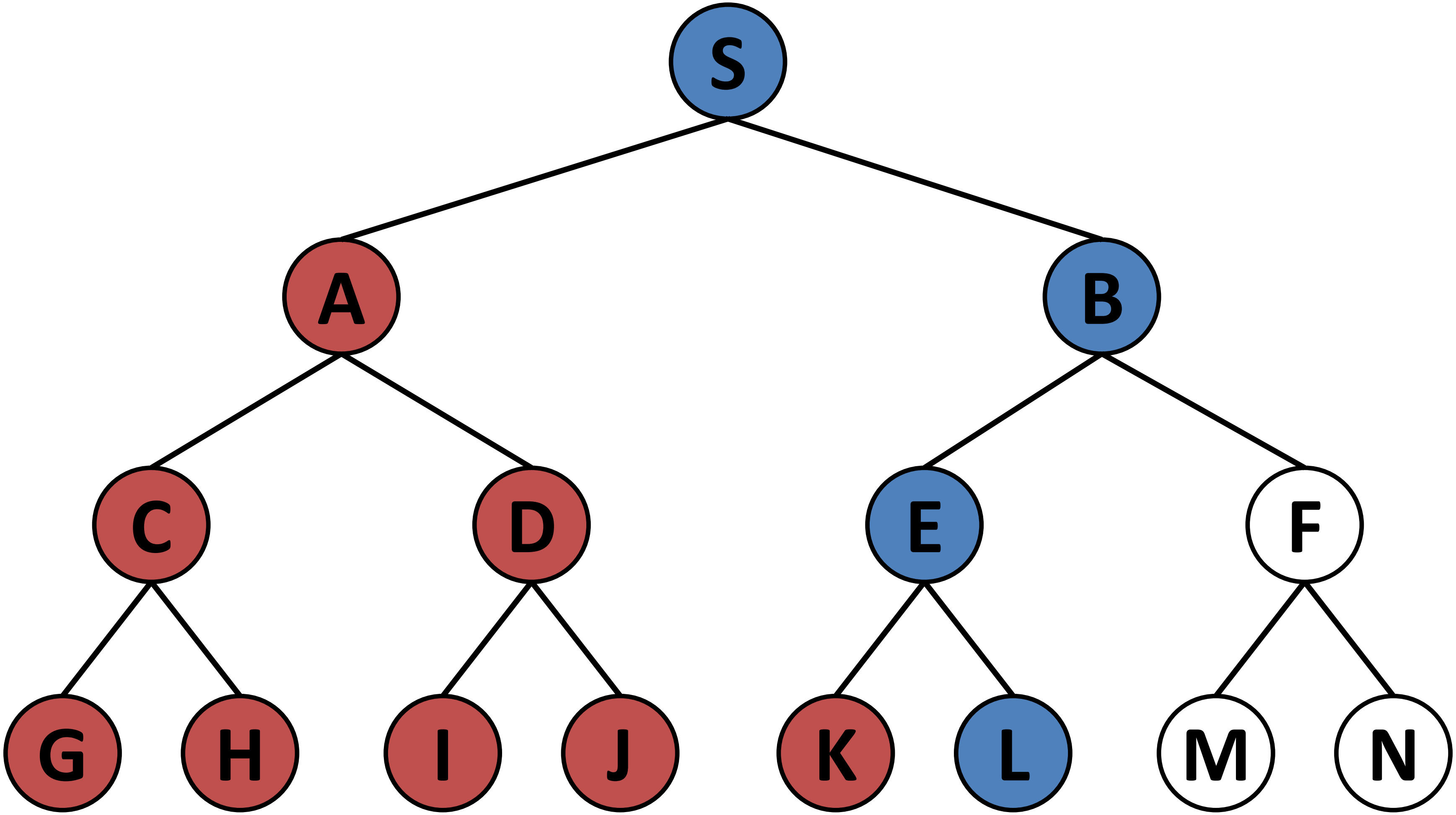
Graph and Tree Traversal: DFS



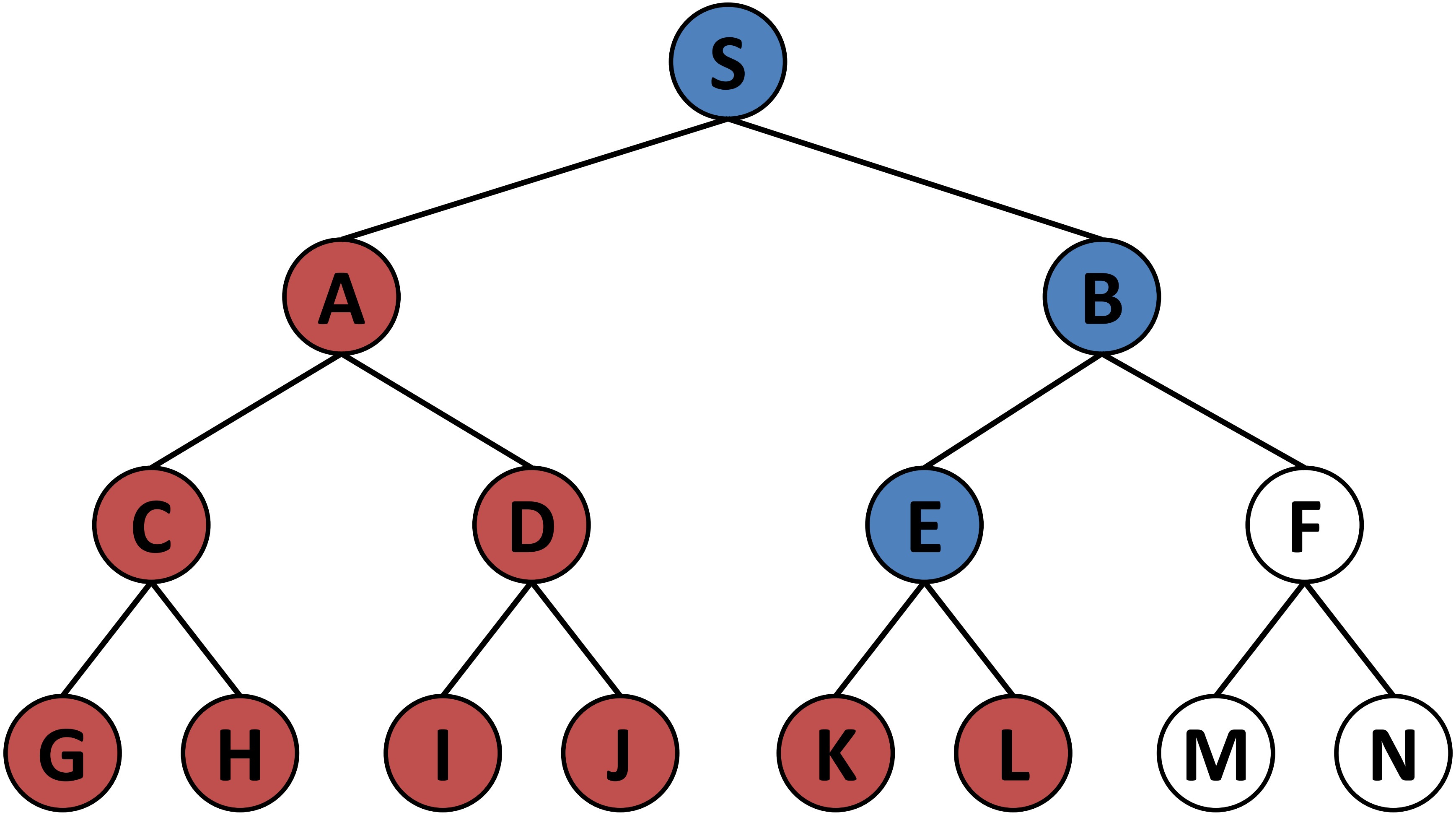
Graph and Tree Traversal: DFS



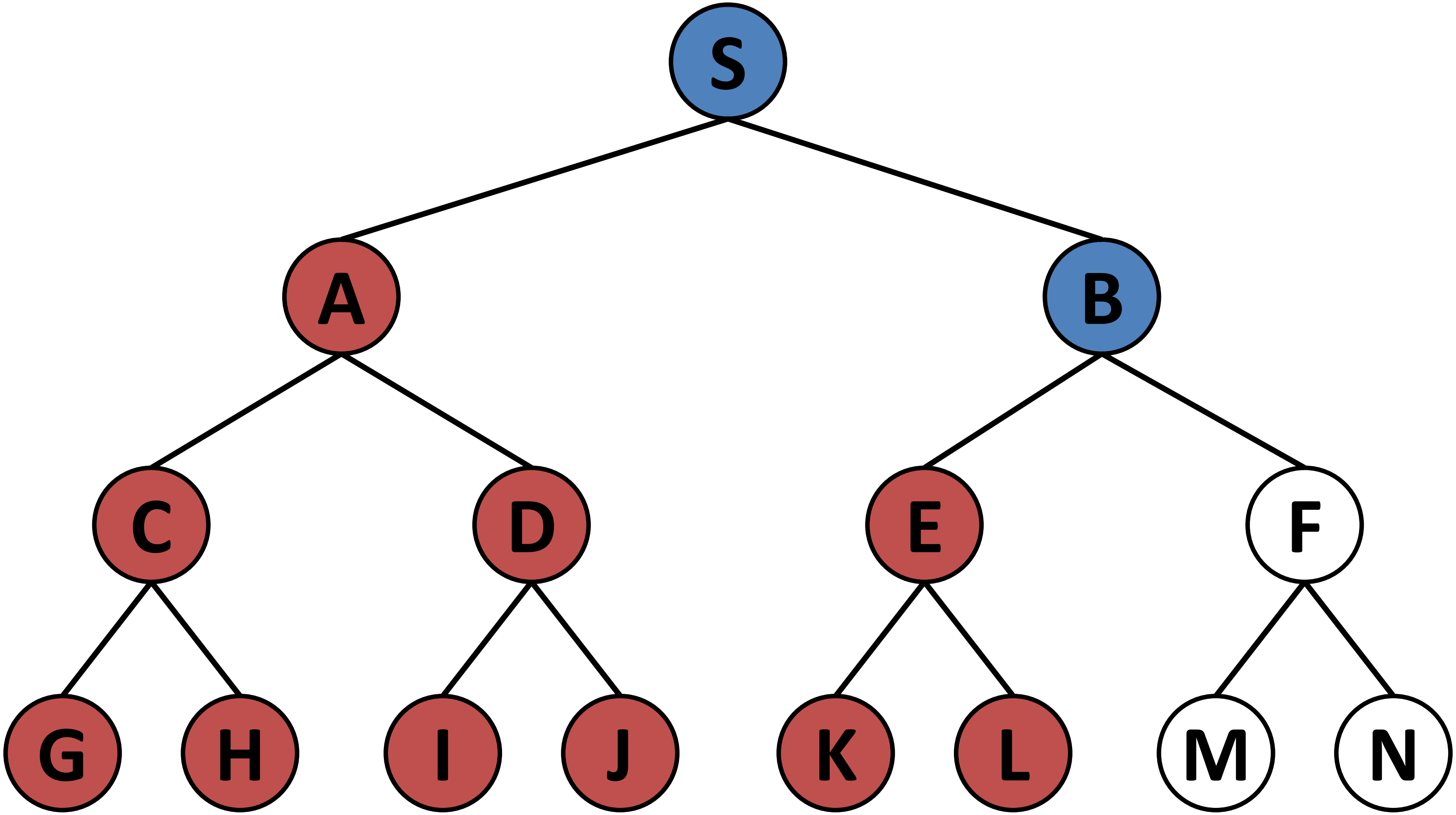
Graph and Tree Traversal: DFS



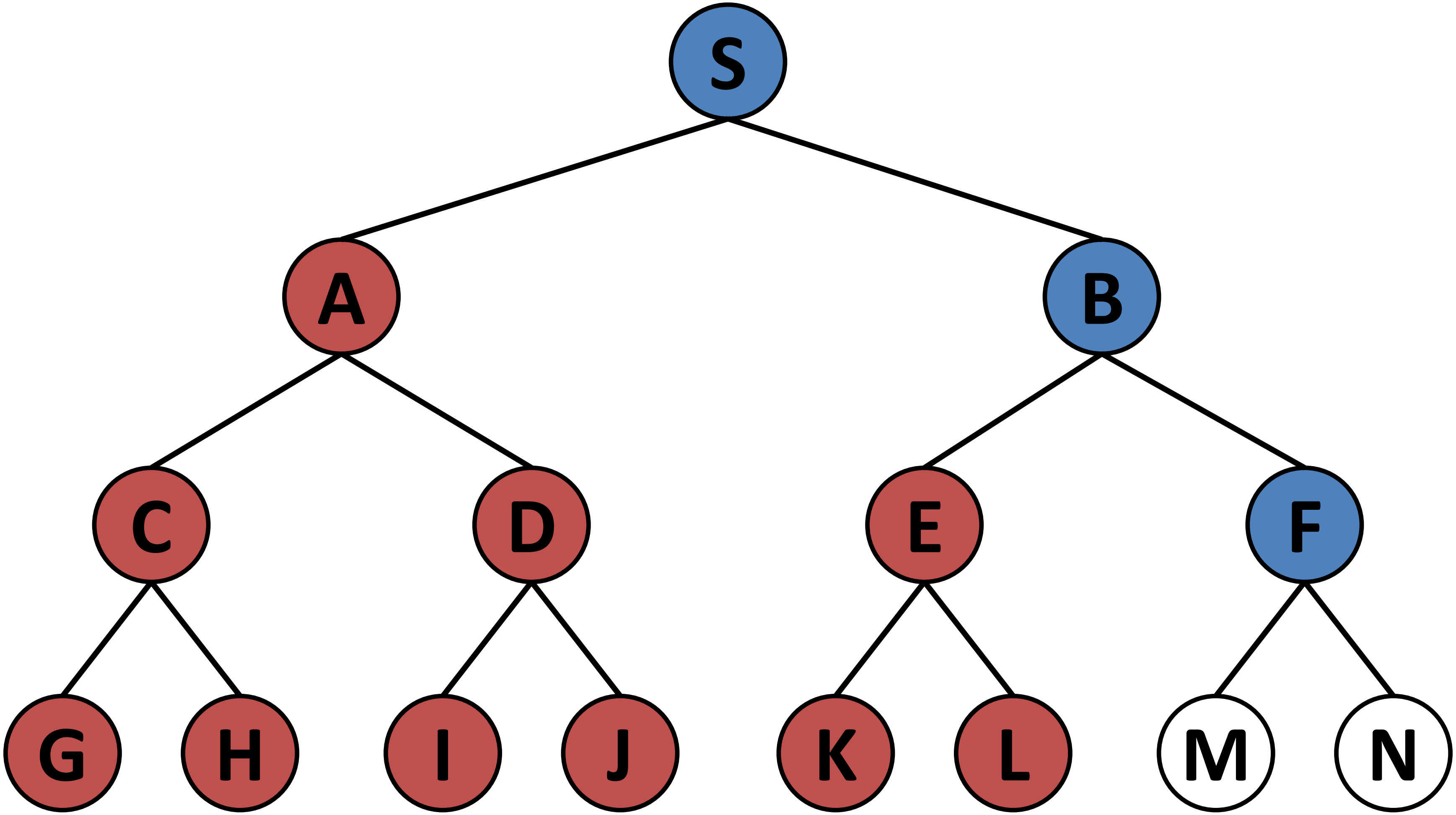
Graph and Tree Traversal: DFS



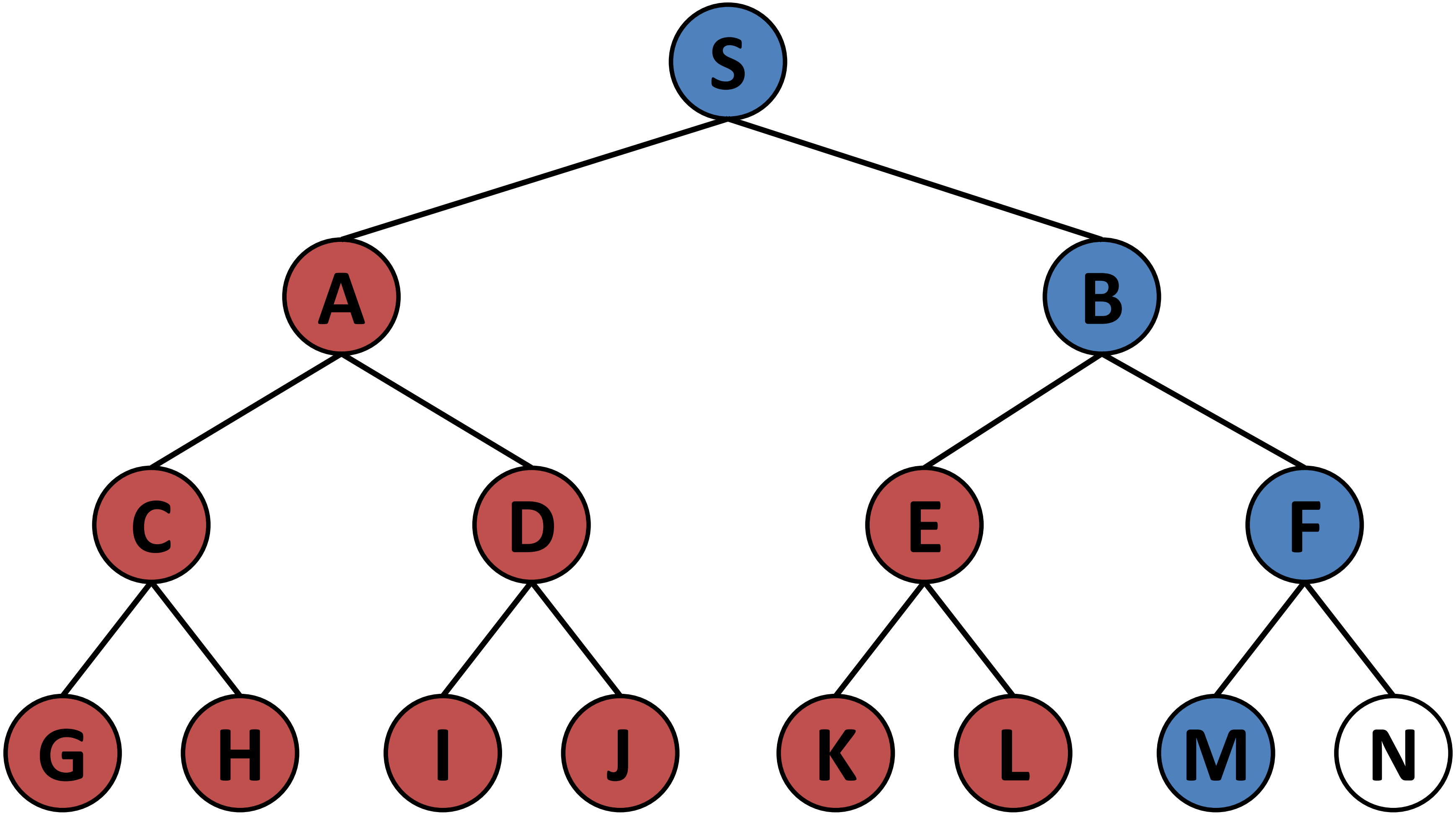
Graph and Tree Traversal: DFS



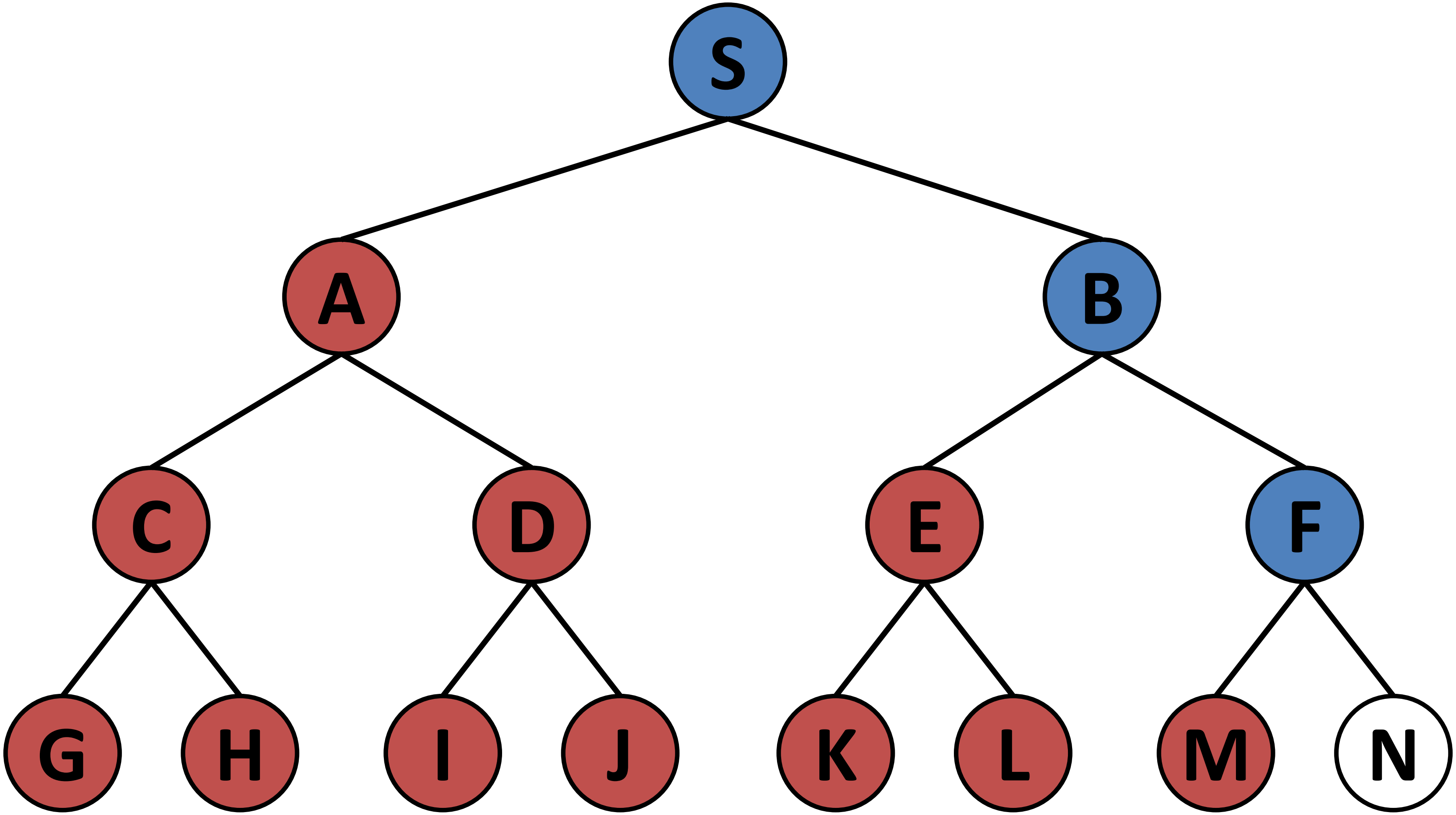
Graph and Tree Traversal: DFS



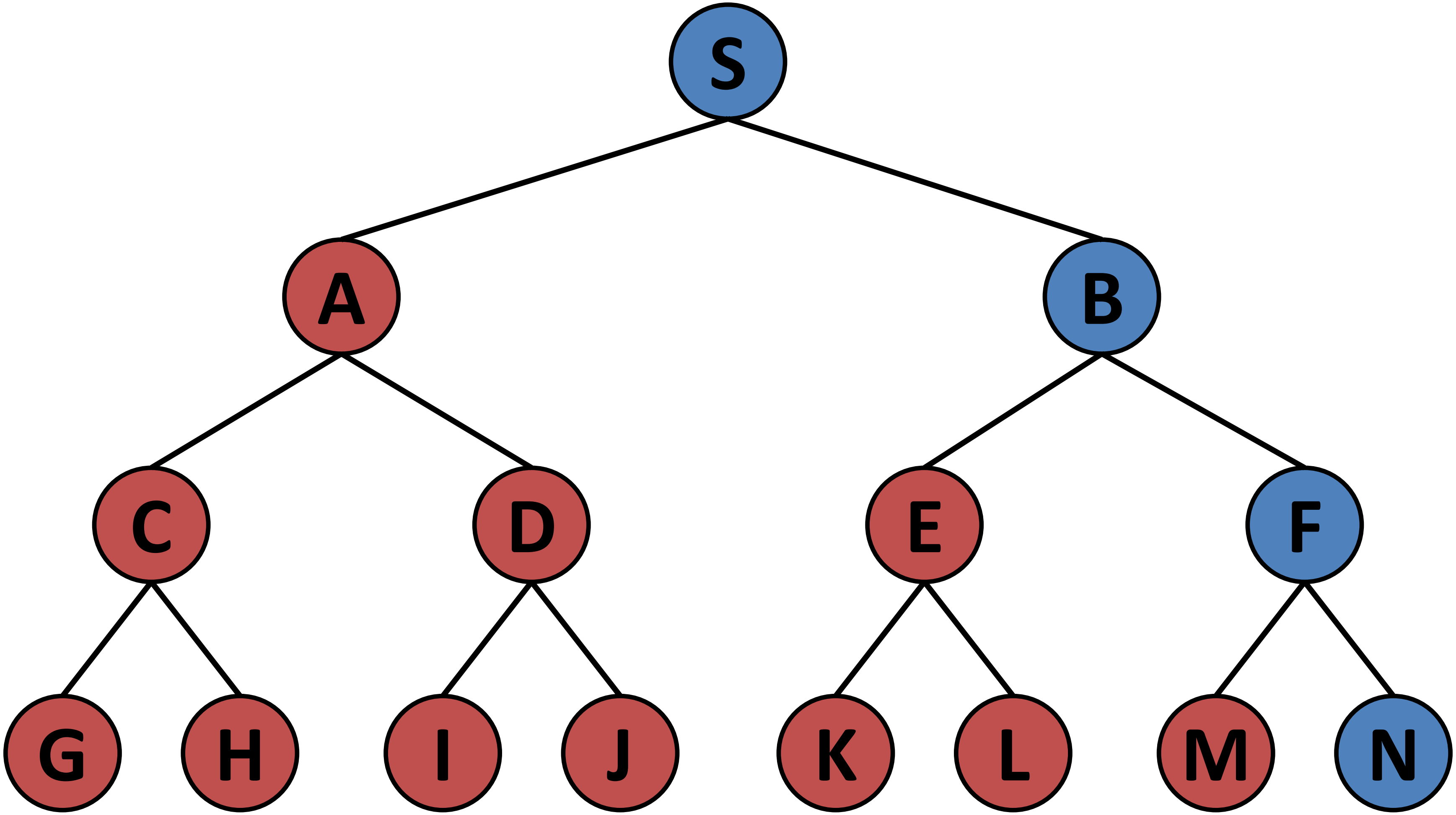
Graph and Tree Traversal: DFS



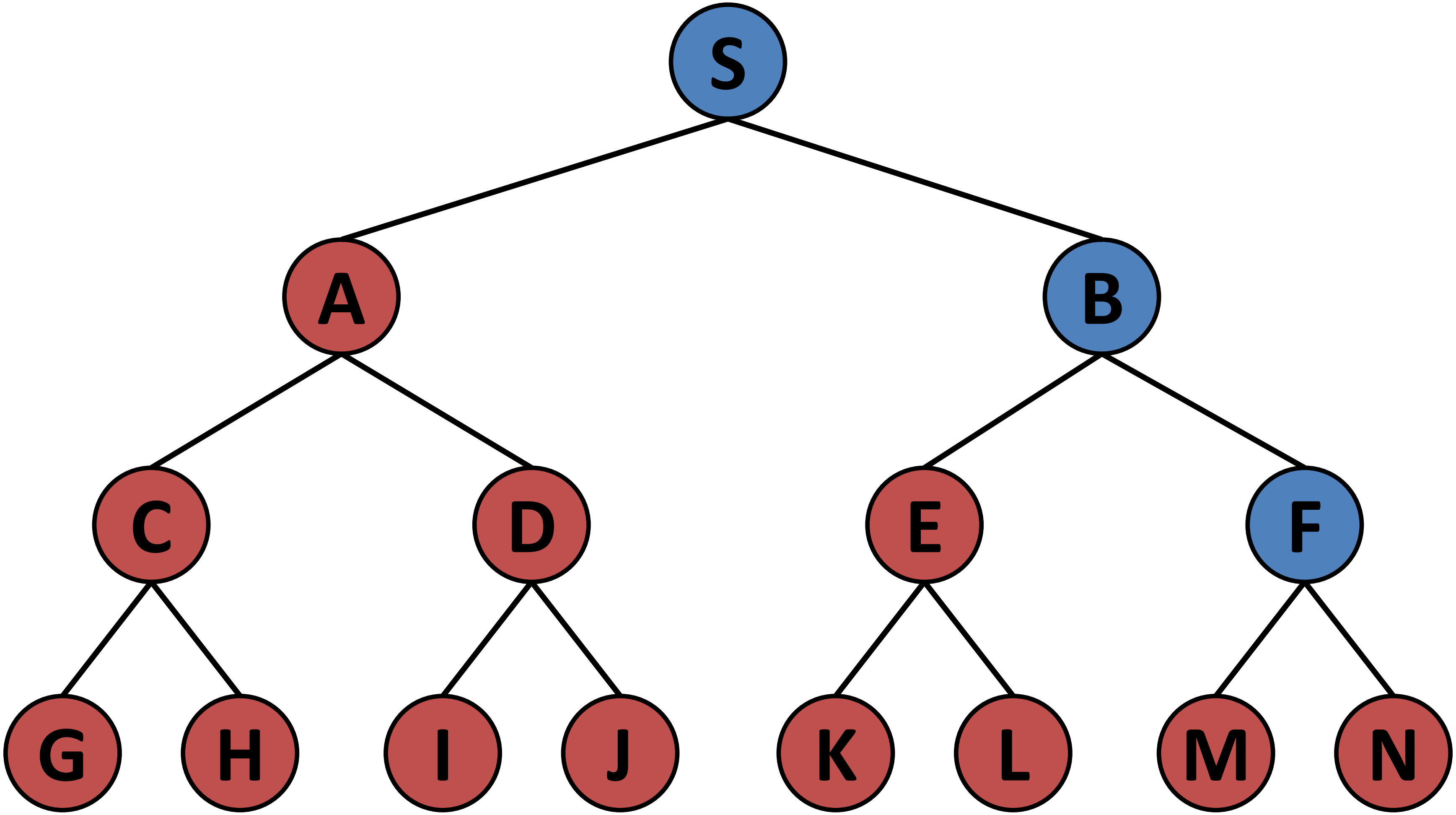
Graph and Tree Traversal: DFS



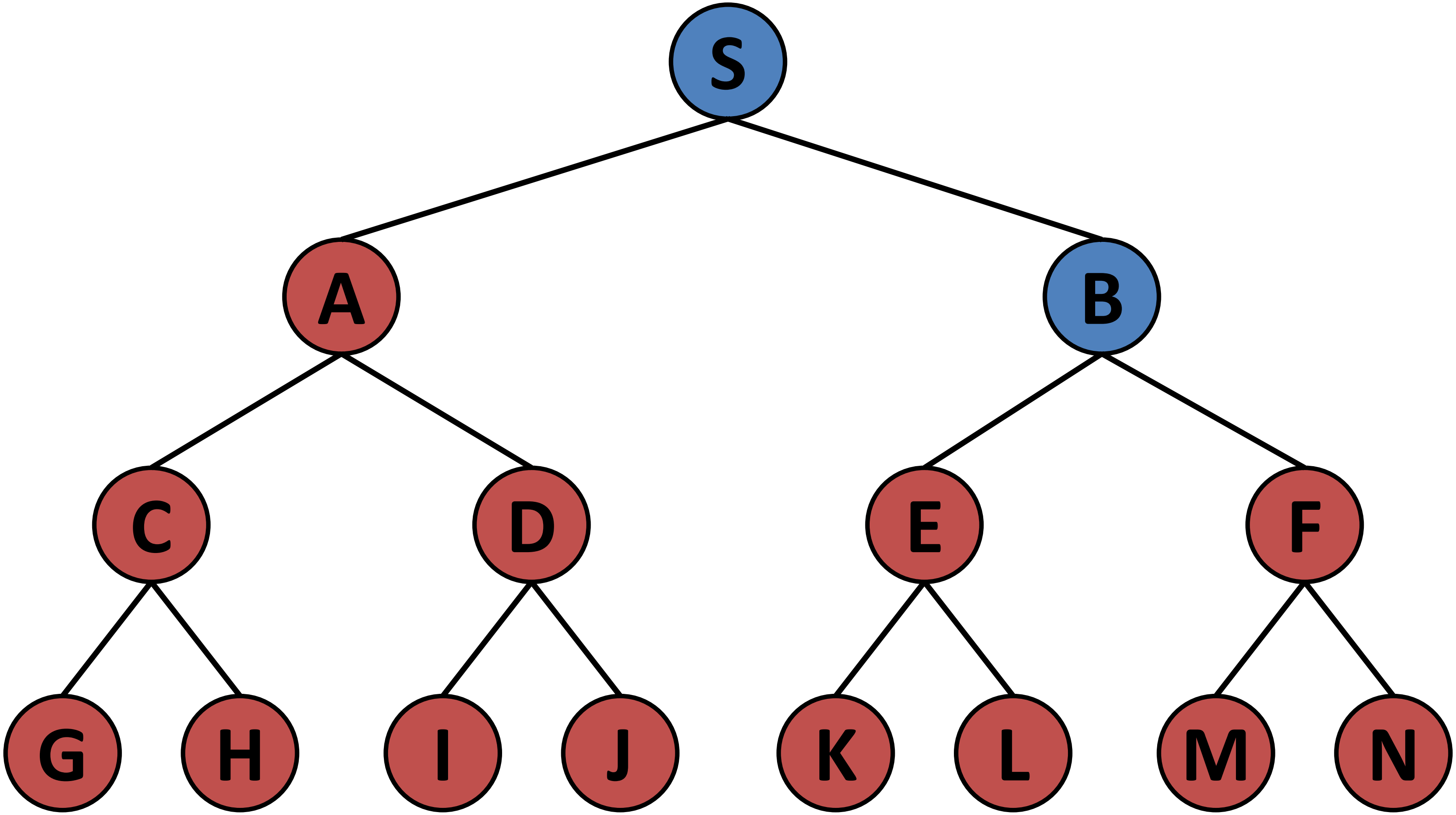
Graph and Tree Traversal: DFS



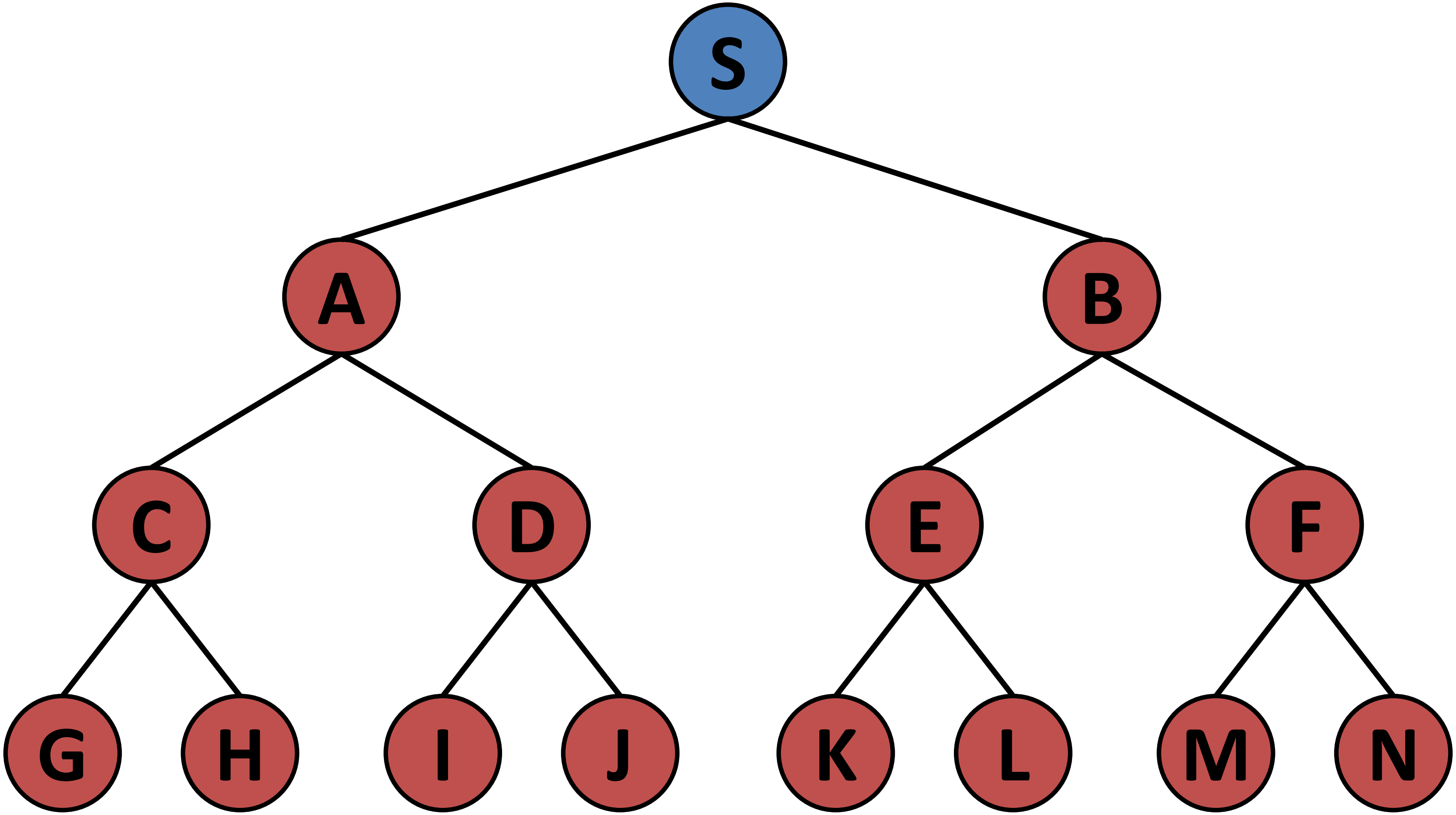
Graph and Tree Traversal: DFS



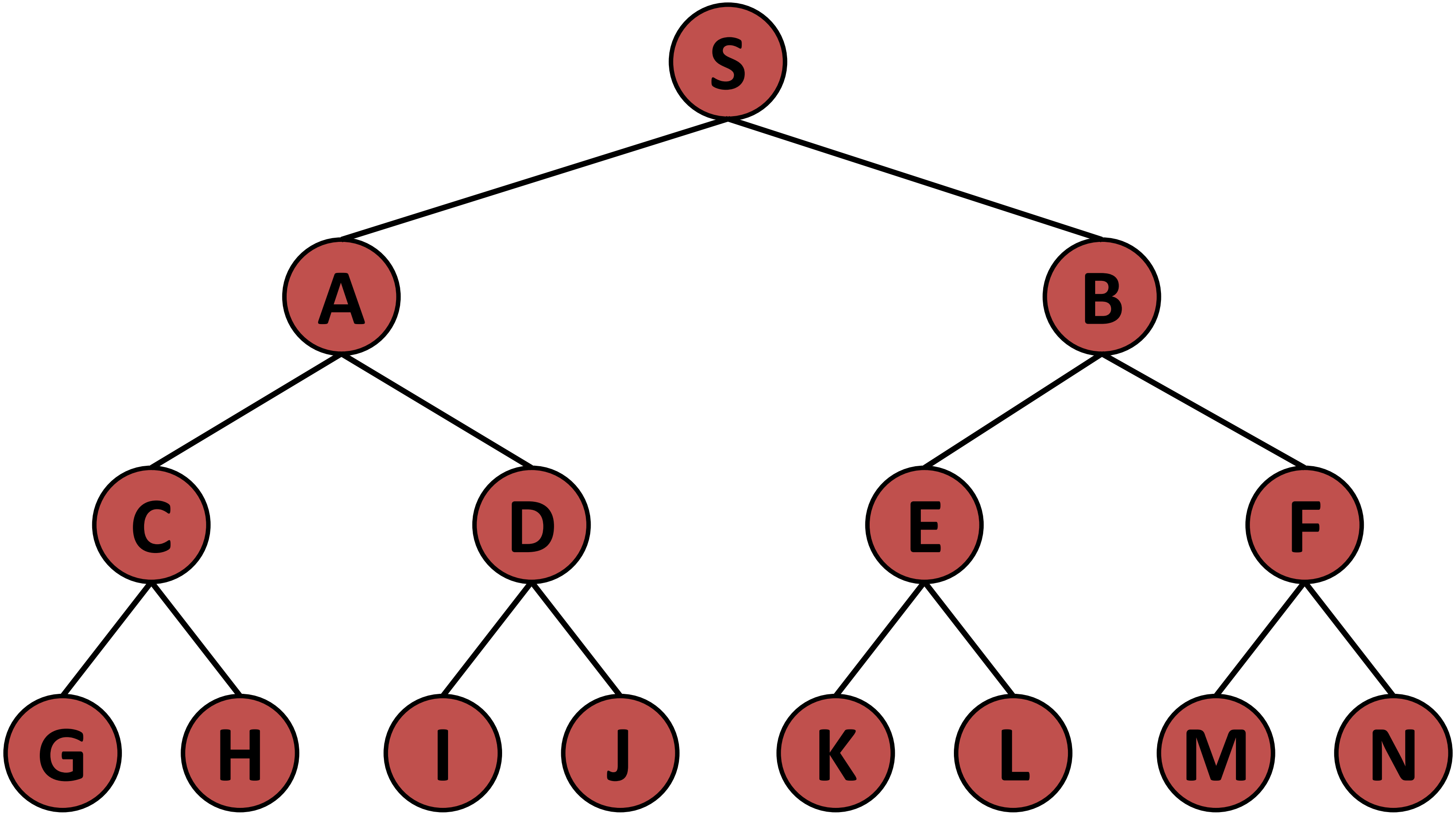
Graph and Tree Traversal: DFS



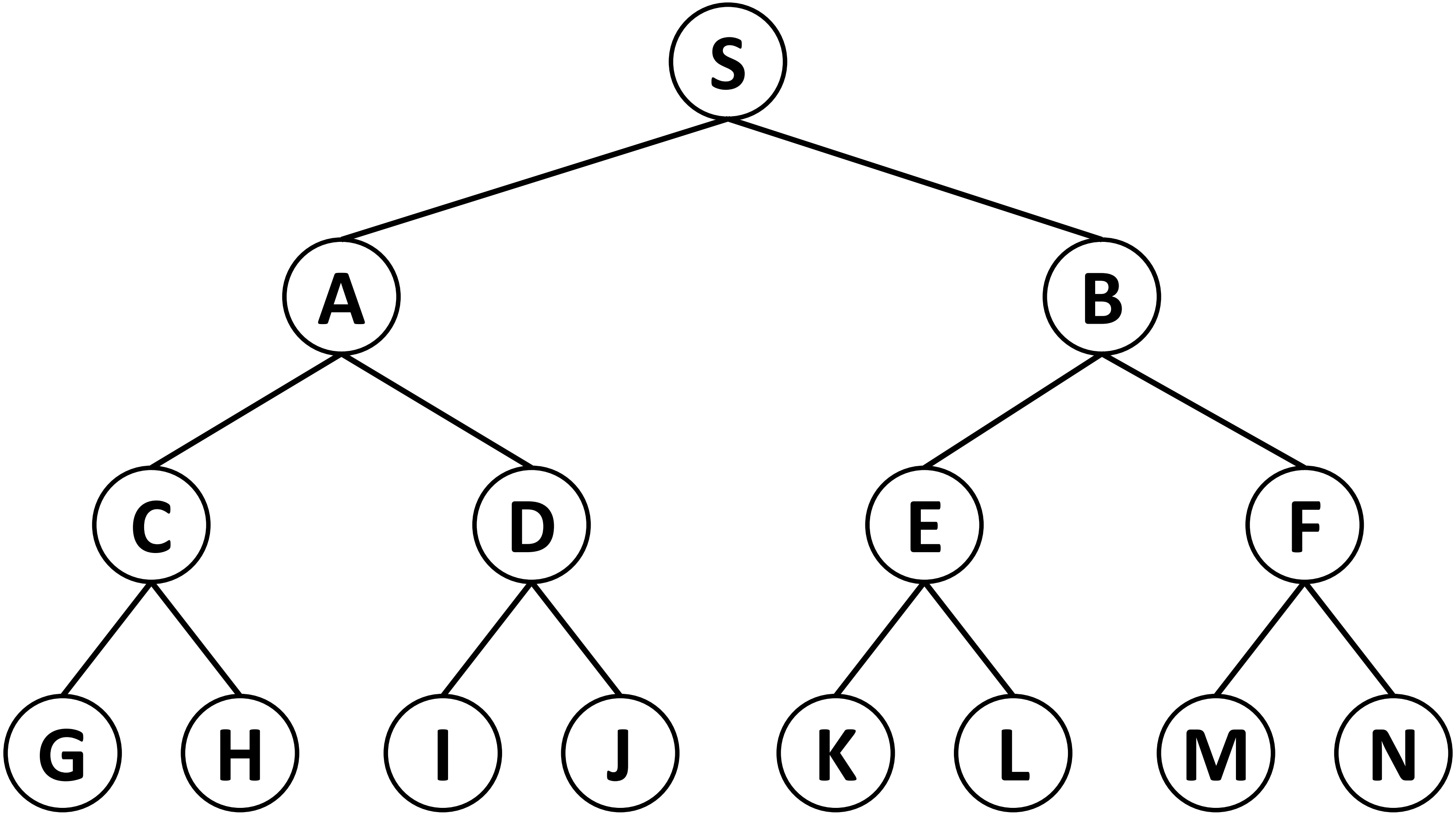
Graph and Tree Traversal: DFS



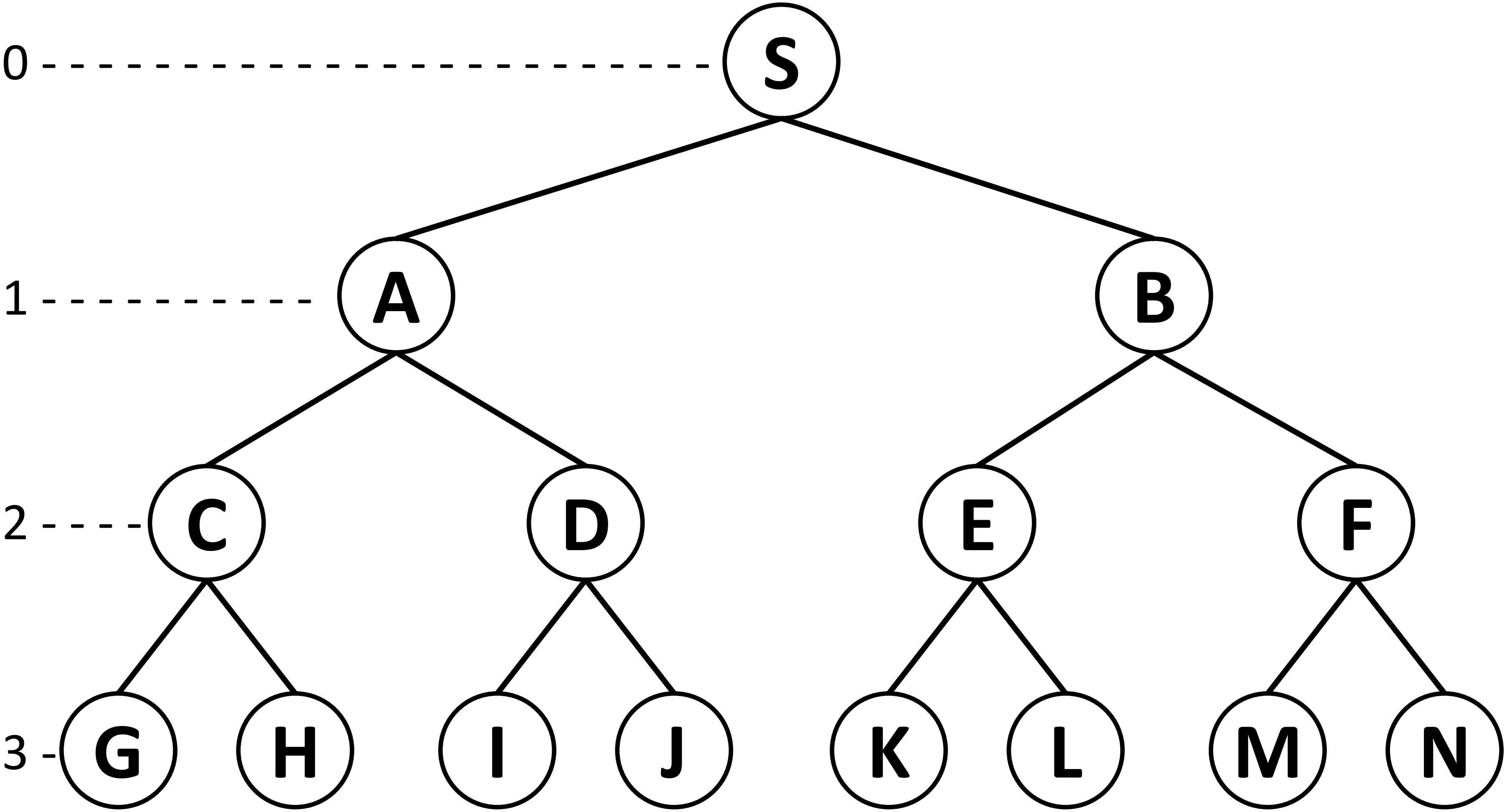
Graph and Tree Traversal: DFS



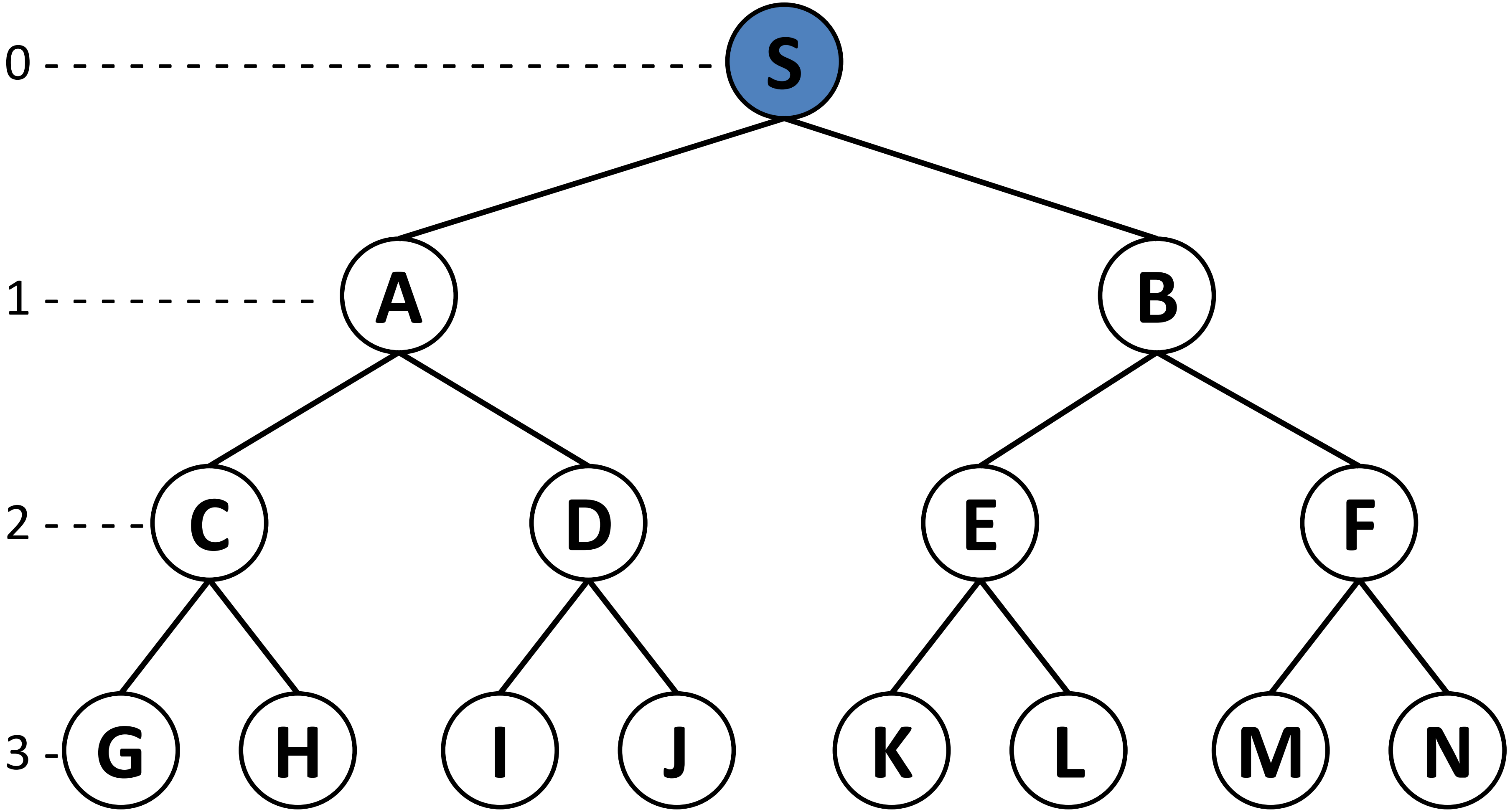
Graph and Tree Traversal: BFS



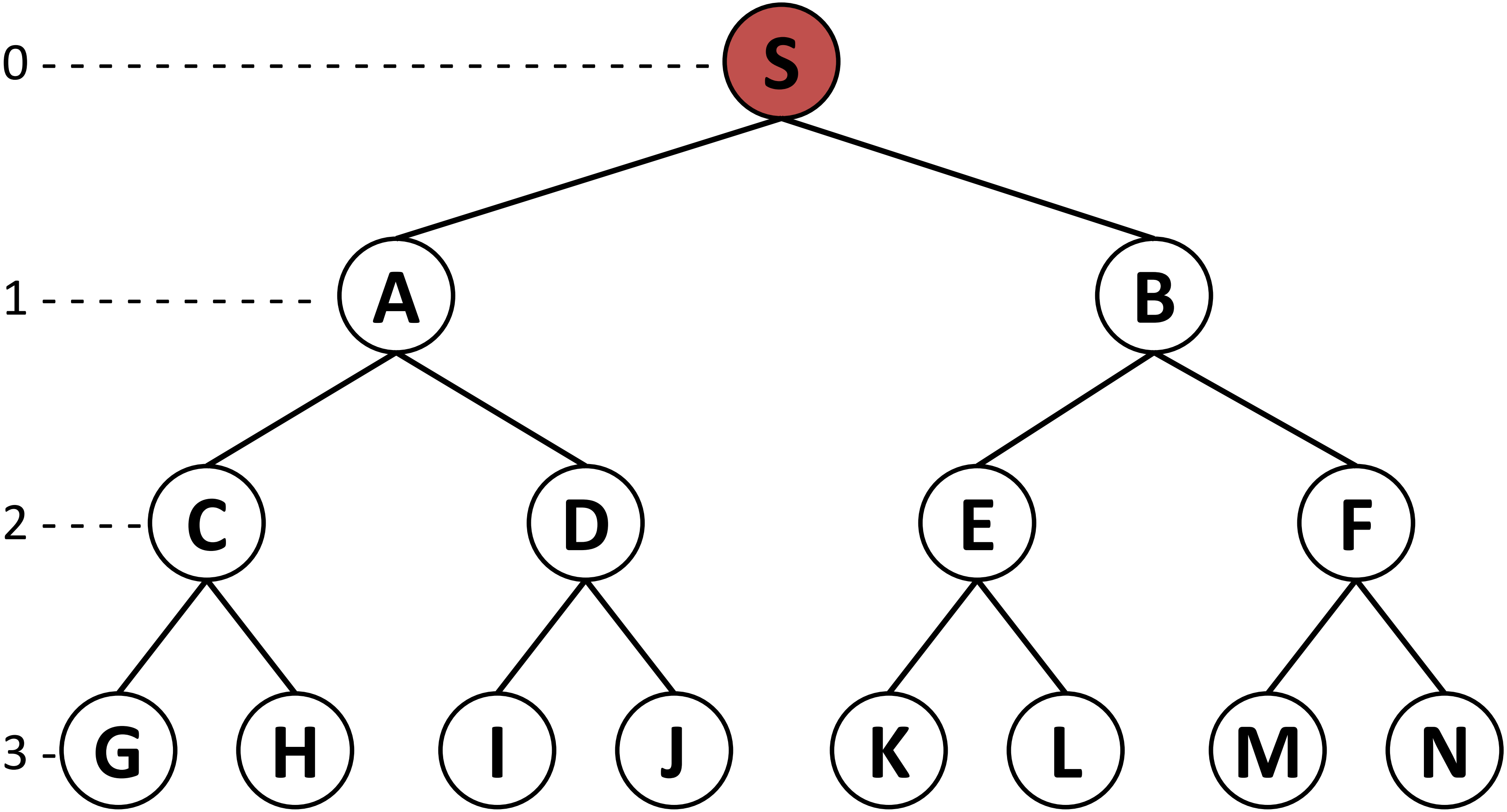
Graph and Tree Traversal: BFS



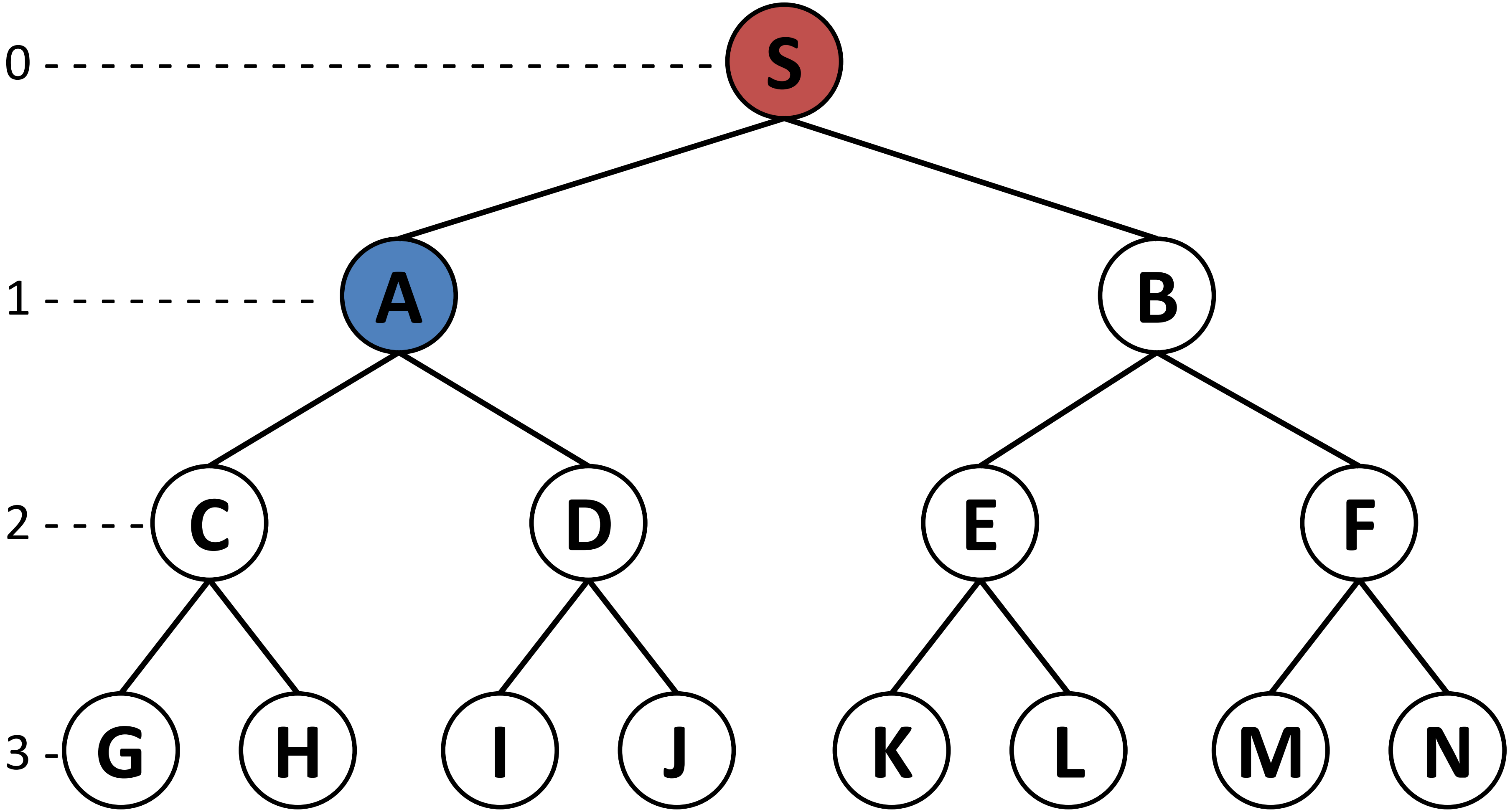
Graph and Tree Traversal: BFS



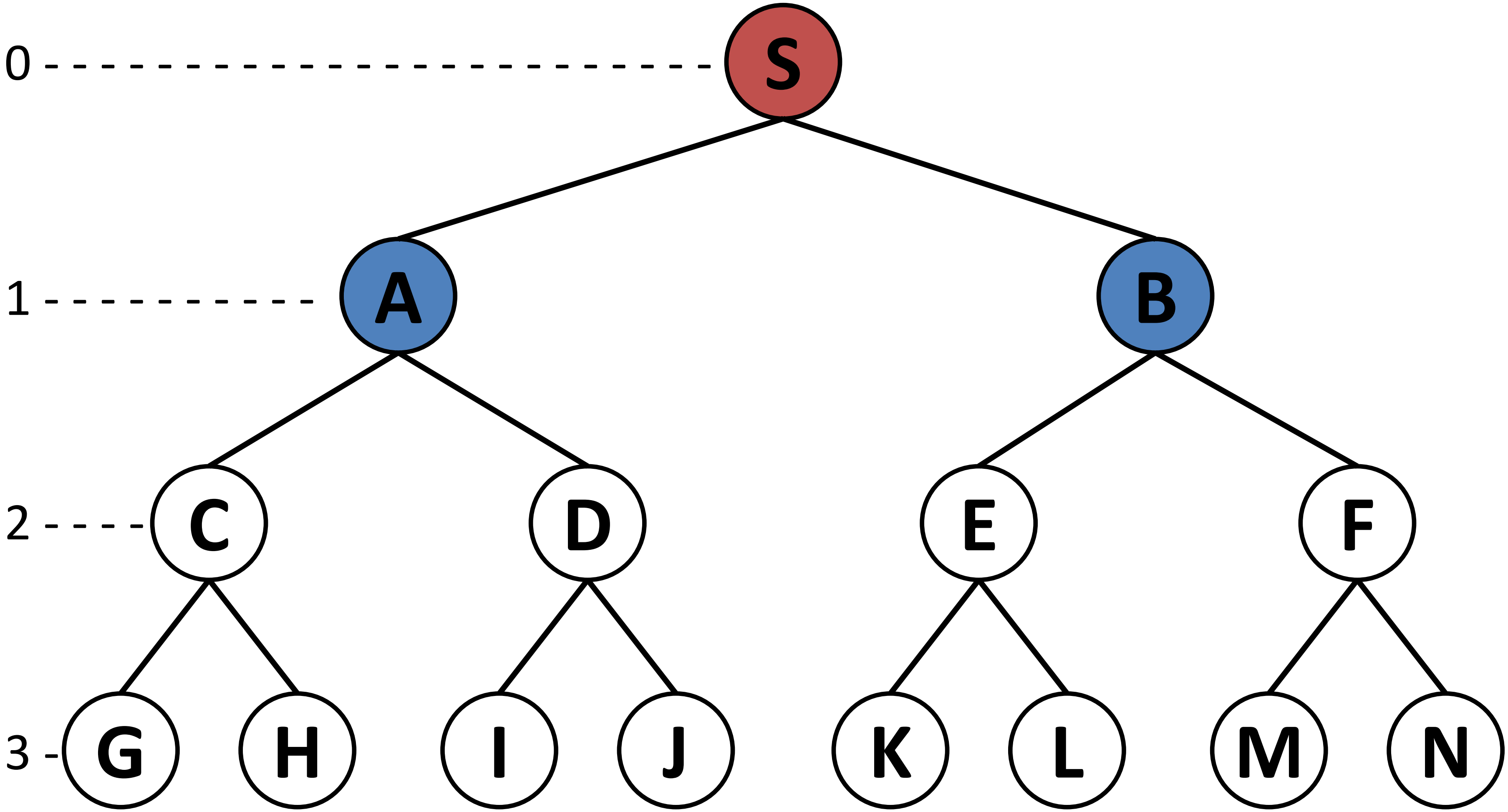
Graph and Tree Traversal: BFS



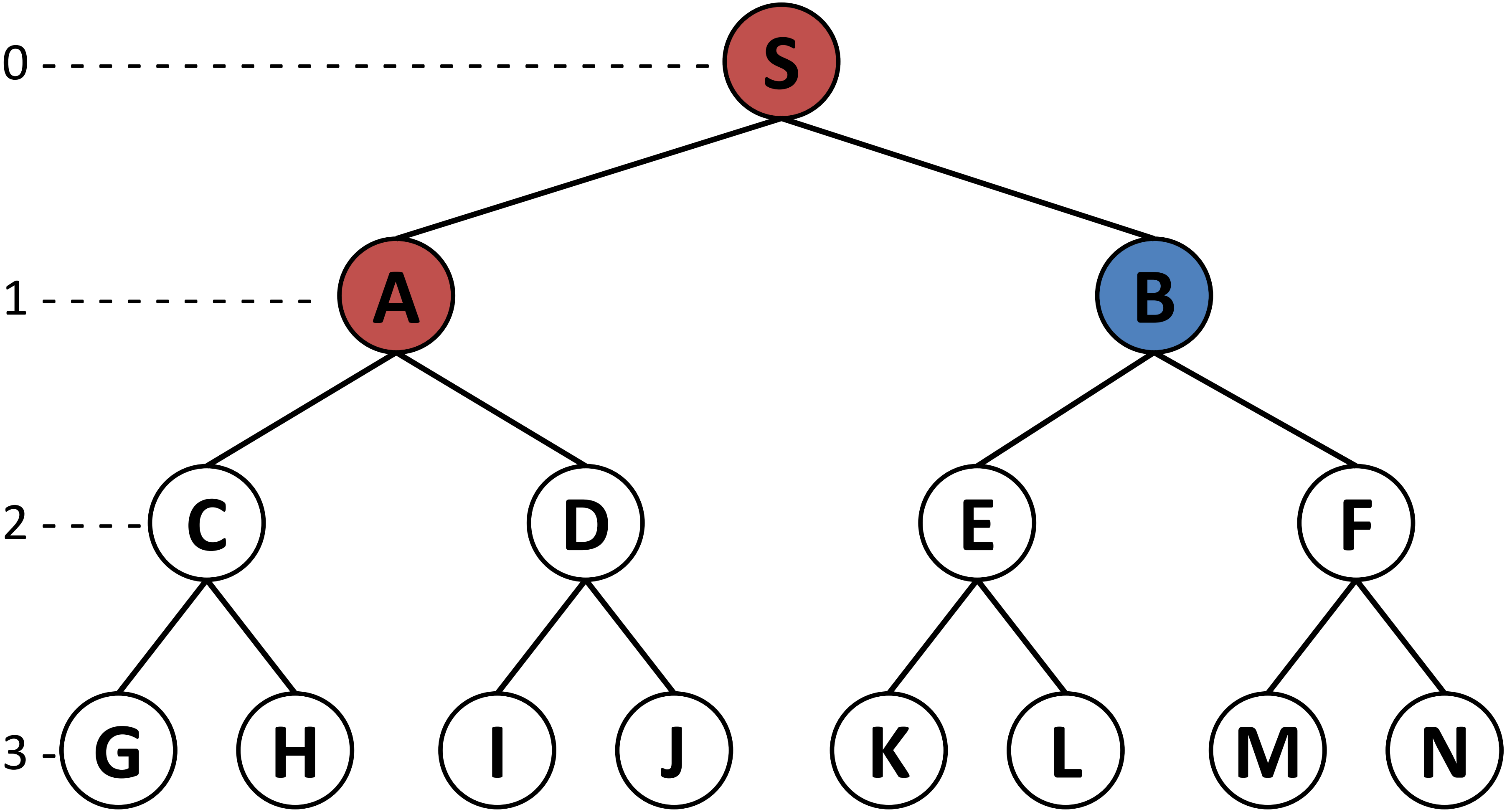
Graph and Tree Traversal: BFS



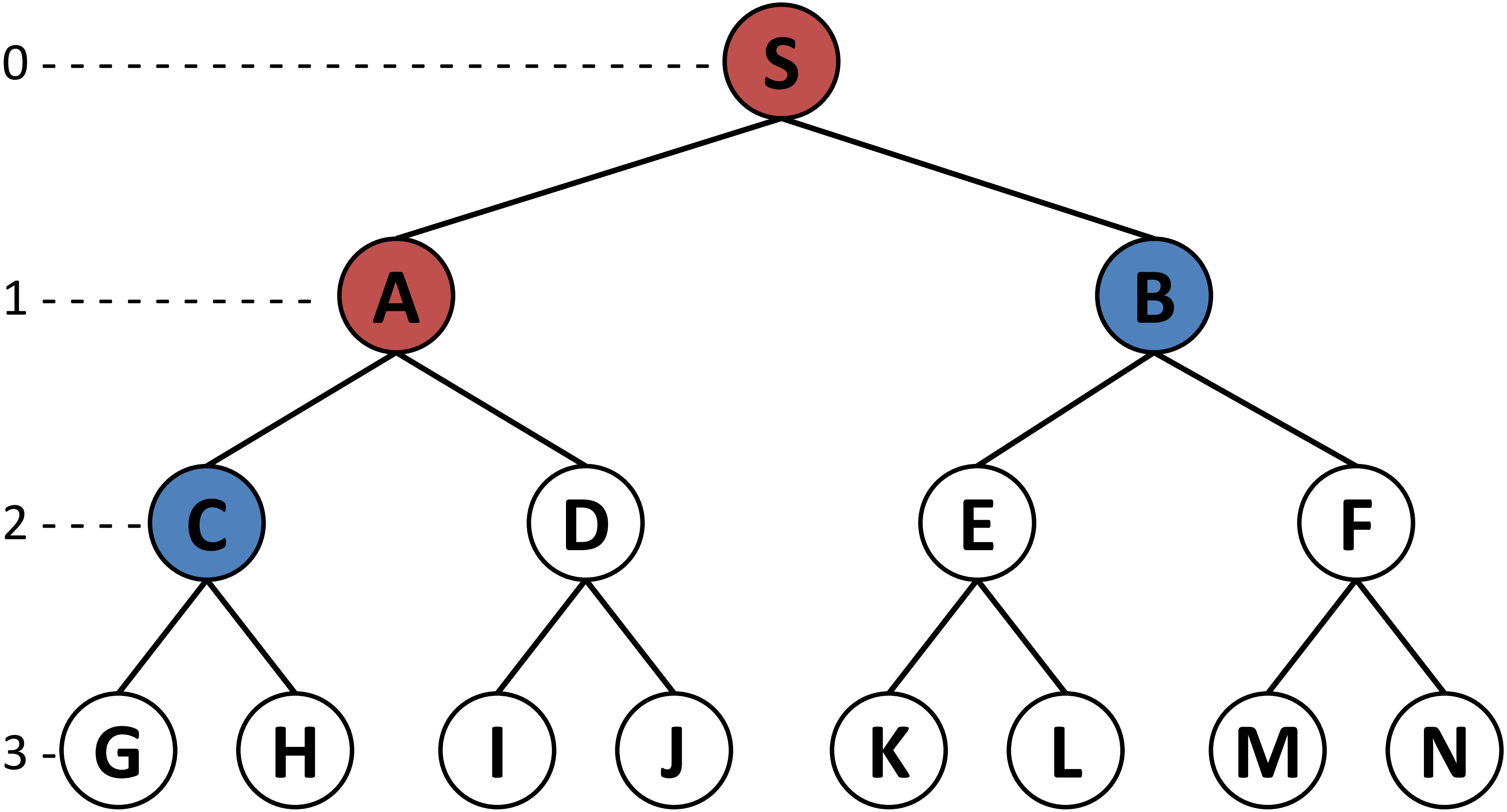
Graph and Tree Traversal: BFS



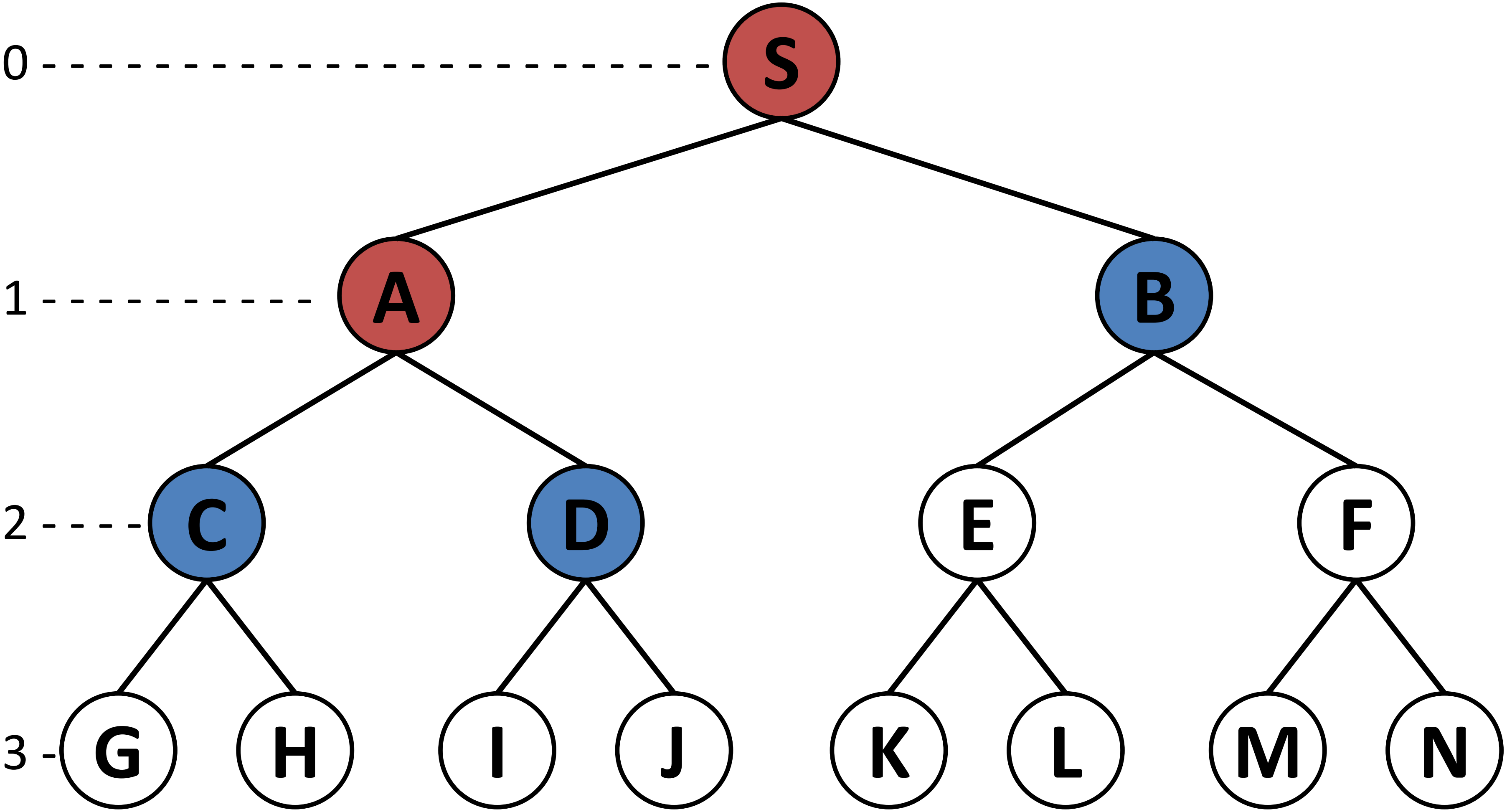
Graph and Tree Traversal: BFS



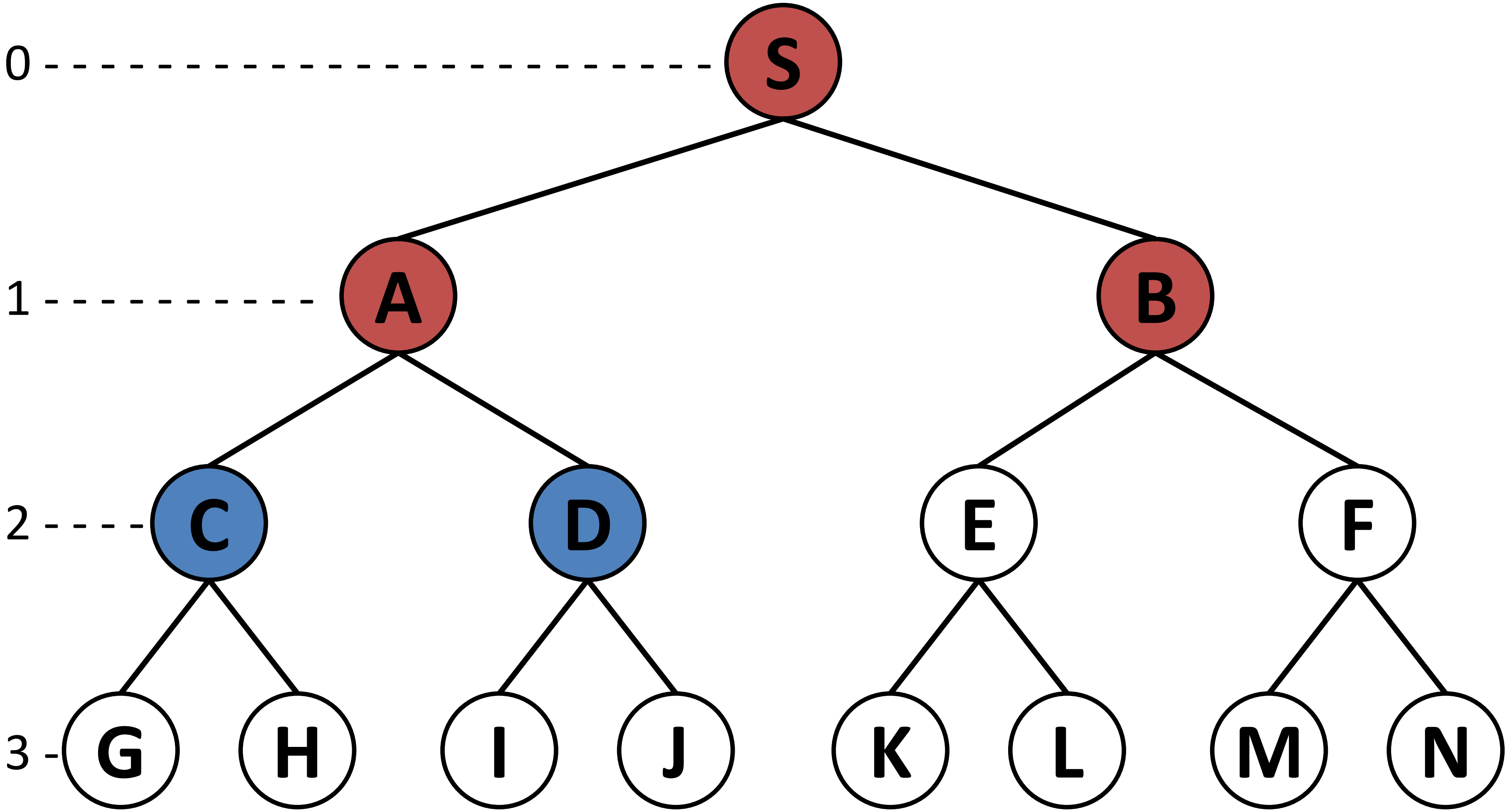
Graph and Tree Traversal: BFS



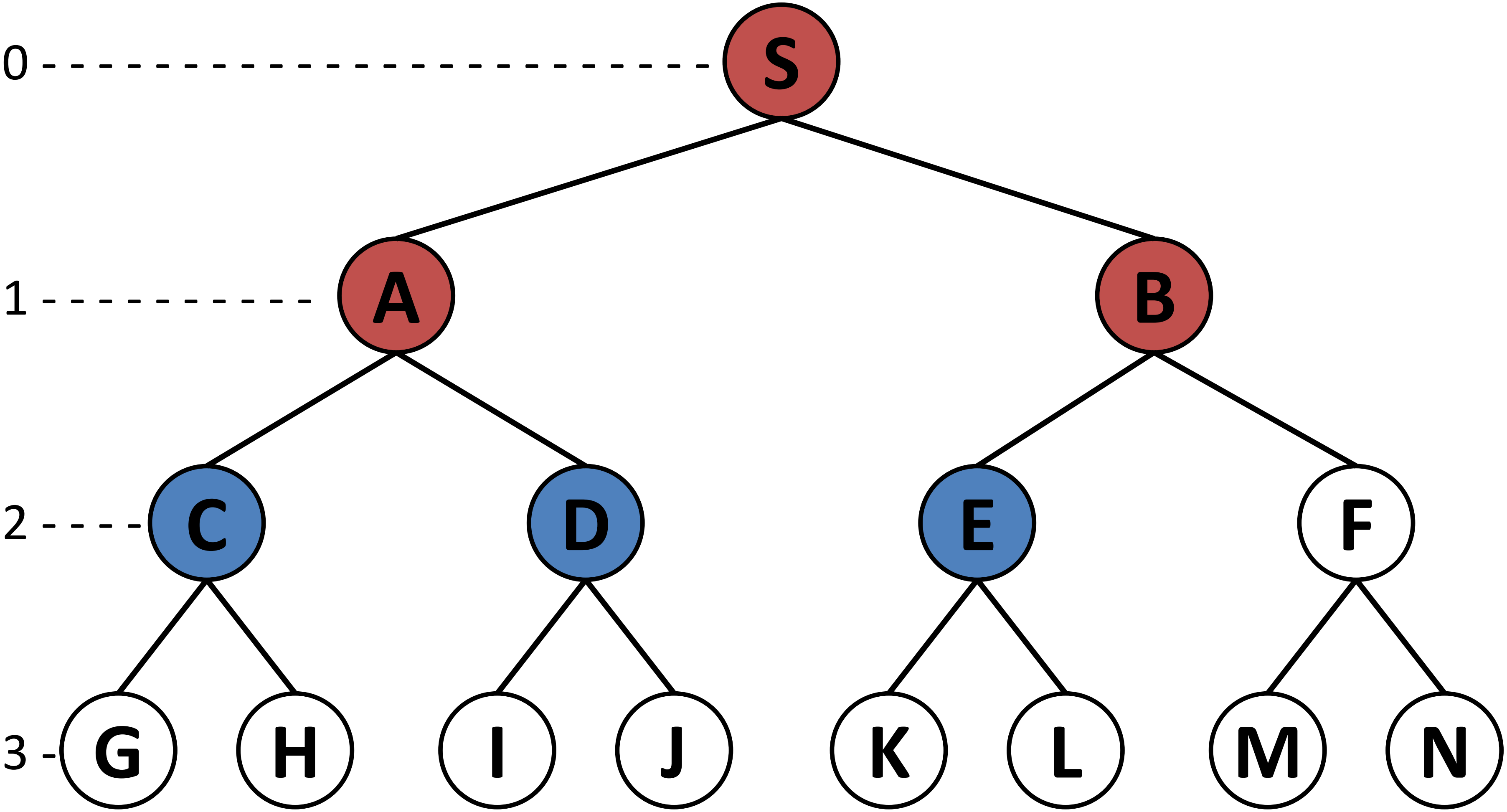
Graph and Tree Traversal: BFS



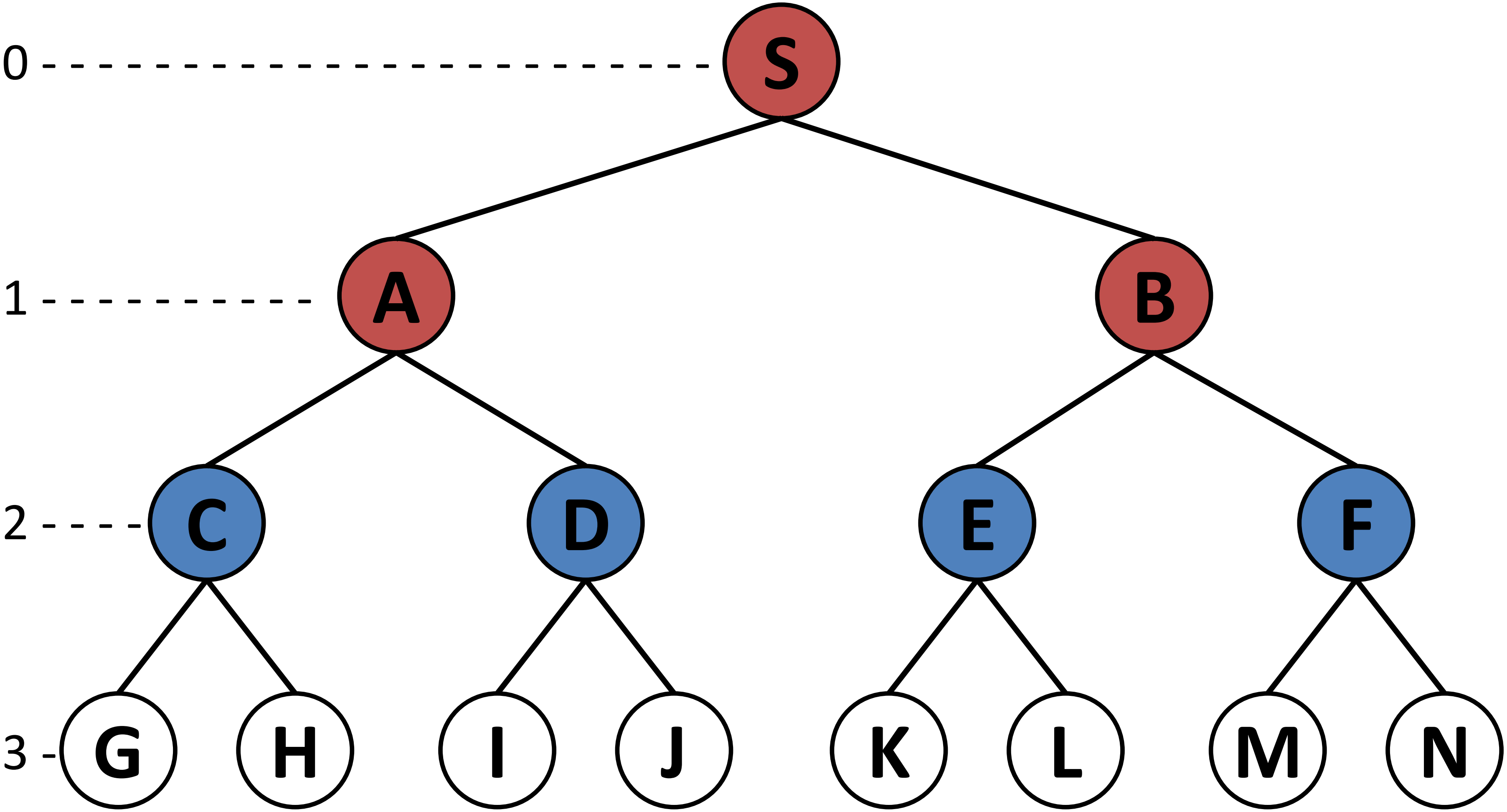
Graph and Tree Traversal: BFS



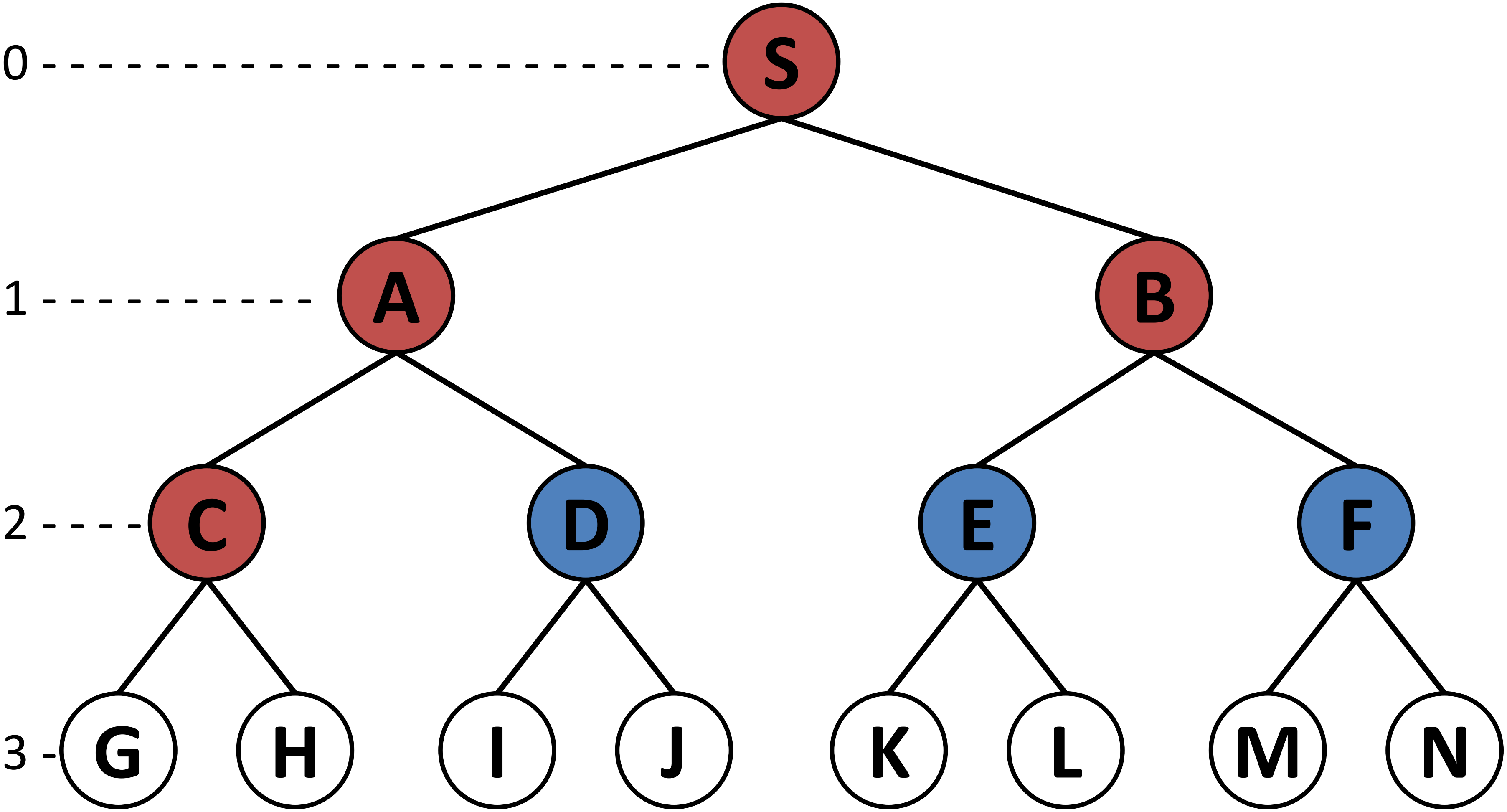
Graph and Tree Traversal: BFS



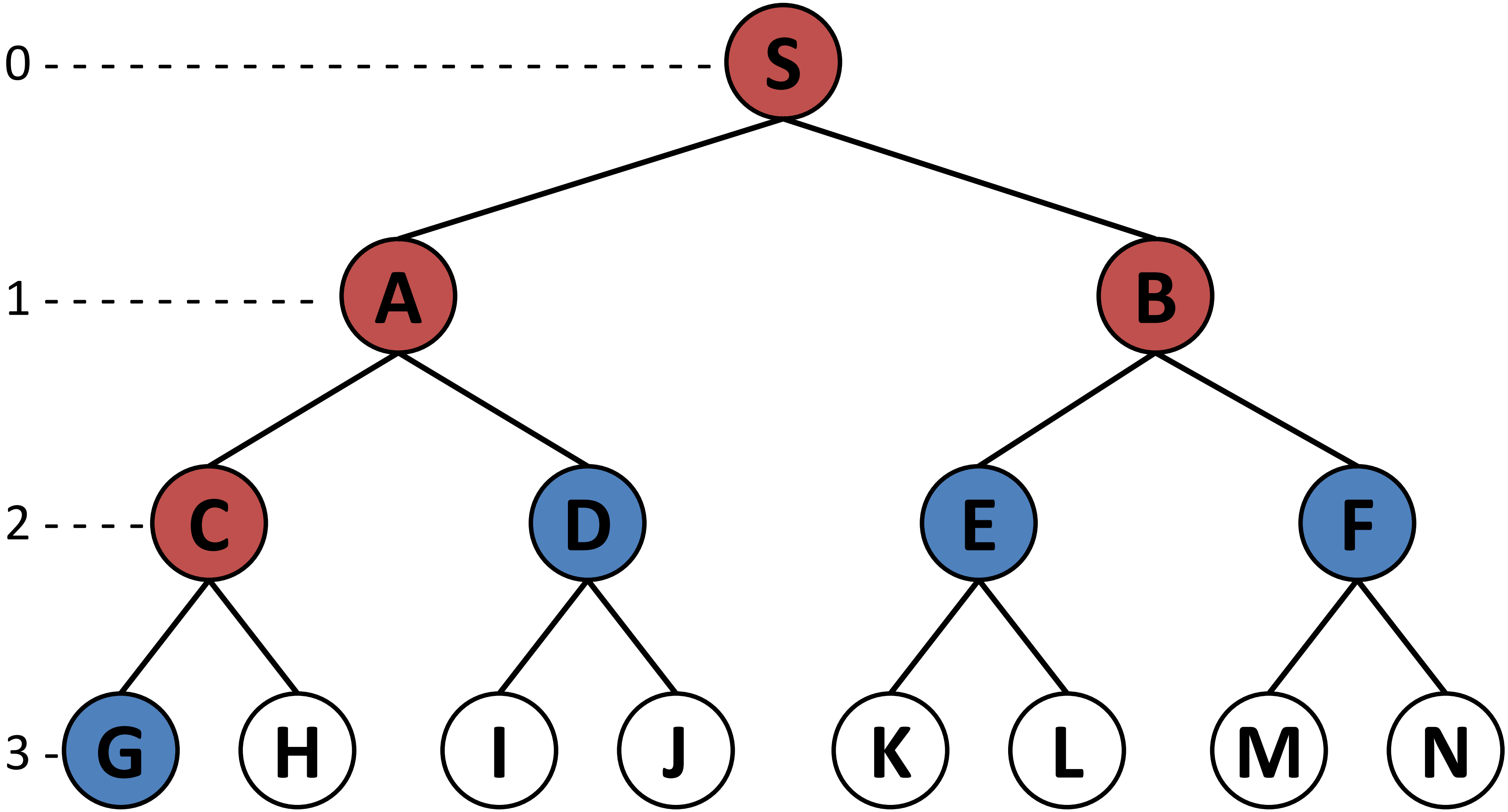
Graph and Tree Traversal: BFS



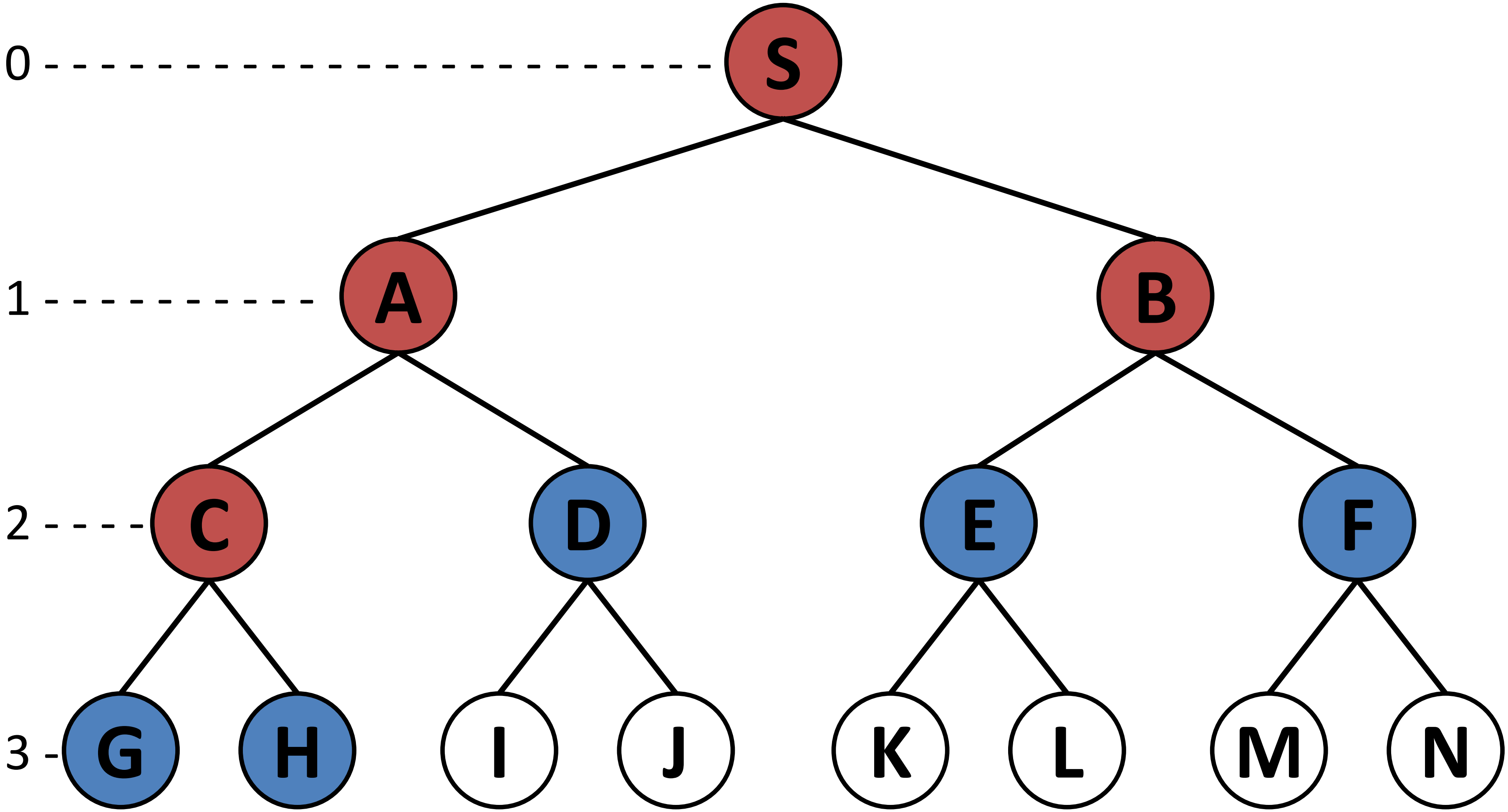
Graph and Tree Traversal: BFS



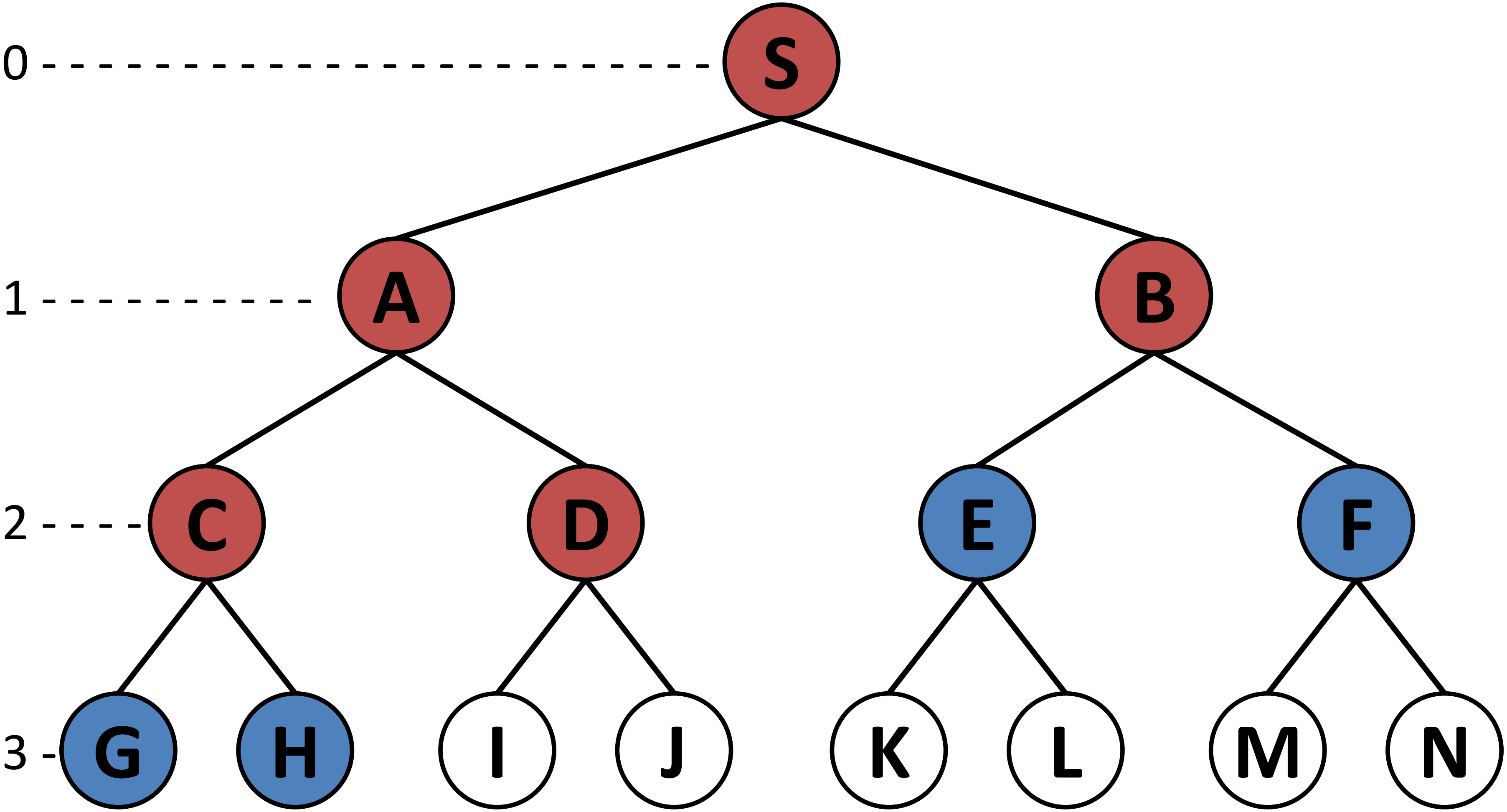
Graph and Tree Traversal: BFS



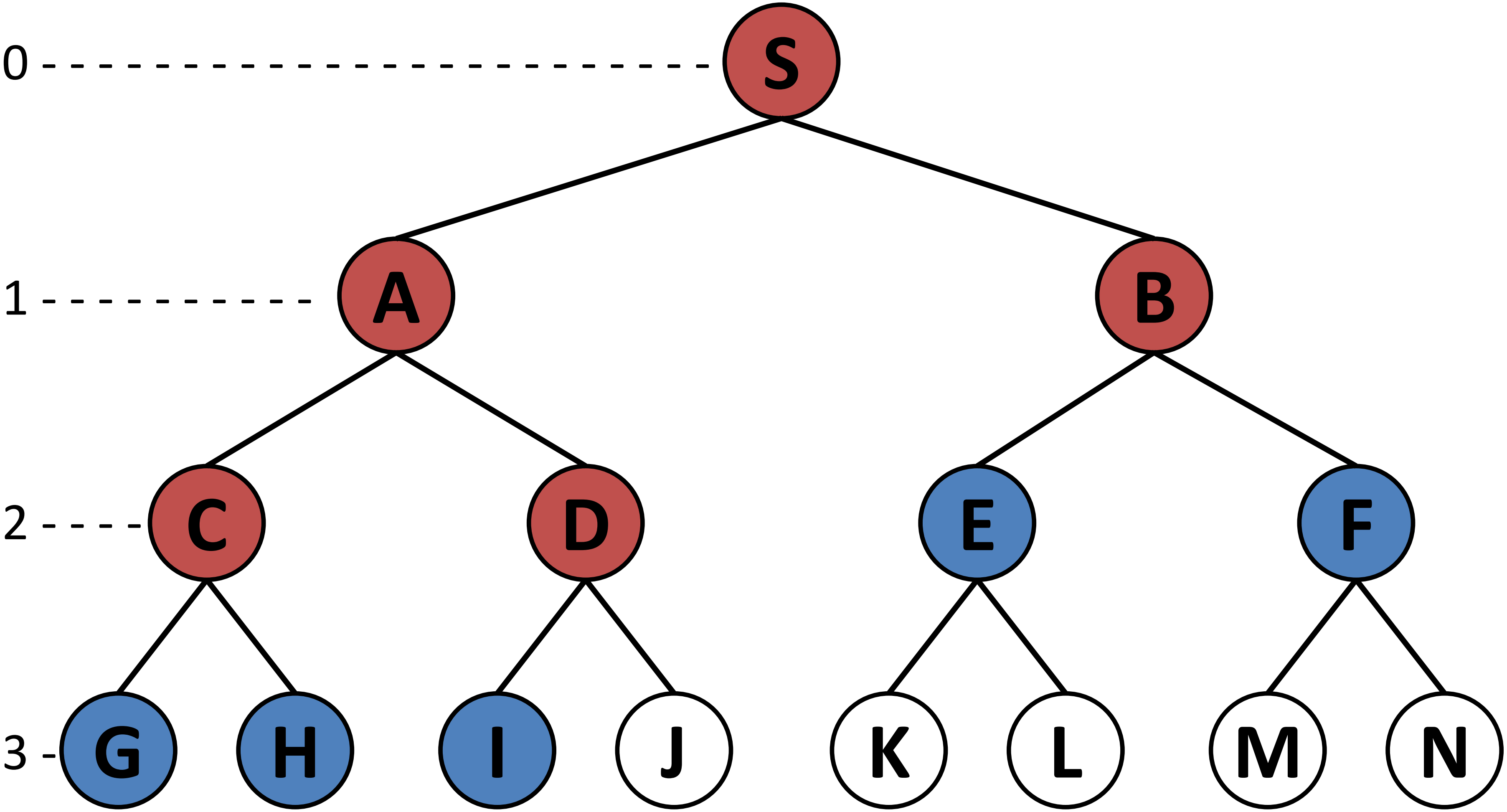
Graph and Tree Traversal: BFS



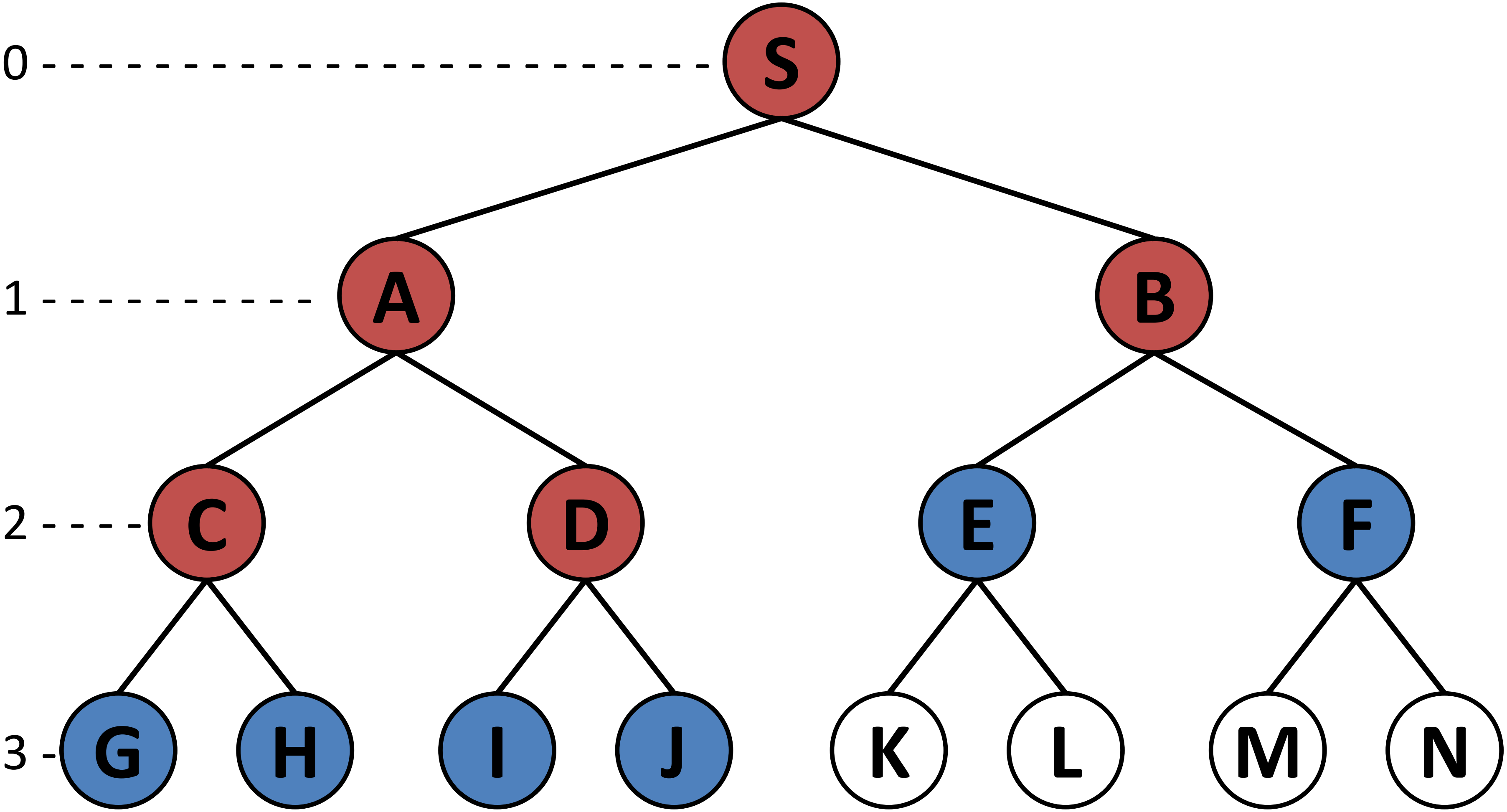
Graph and Tree Traversal: BFS



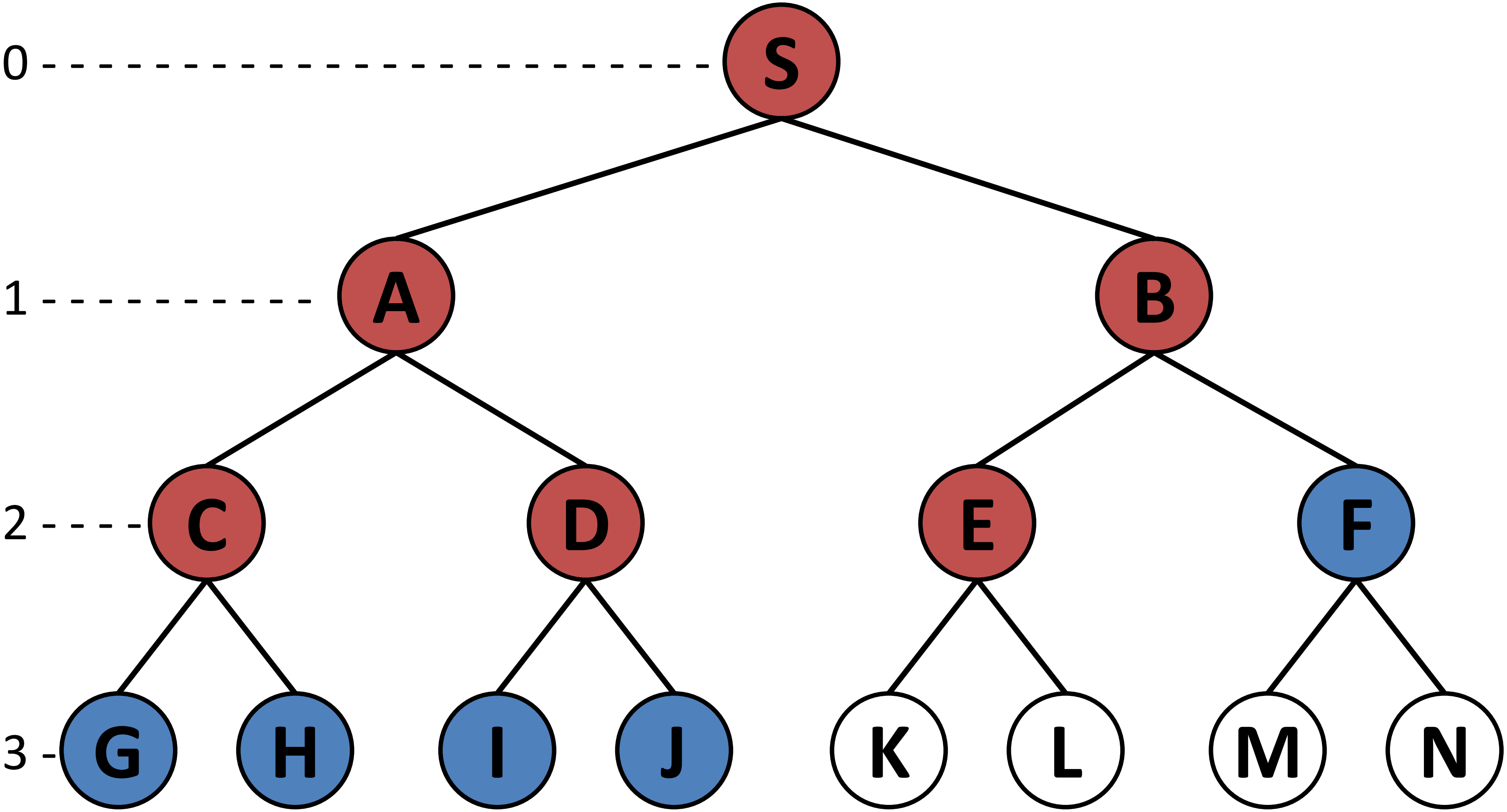
Graph and Tree Traversal: BFS



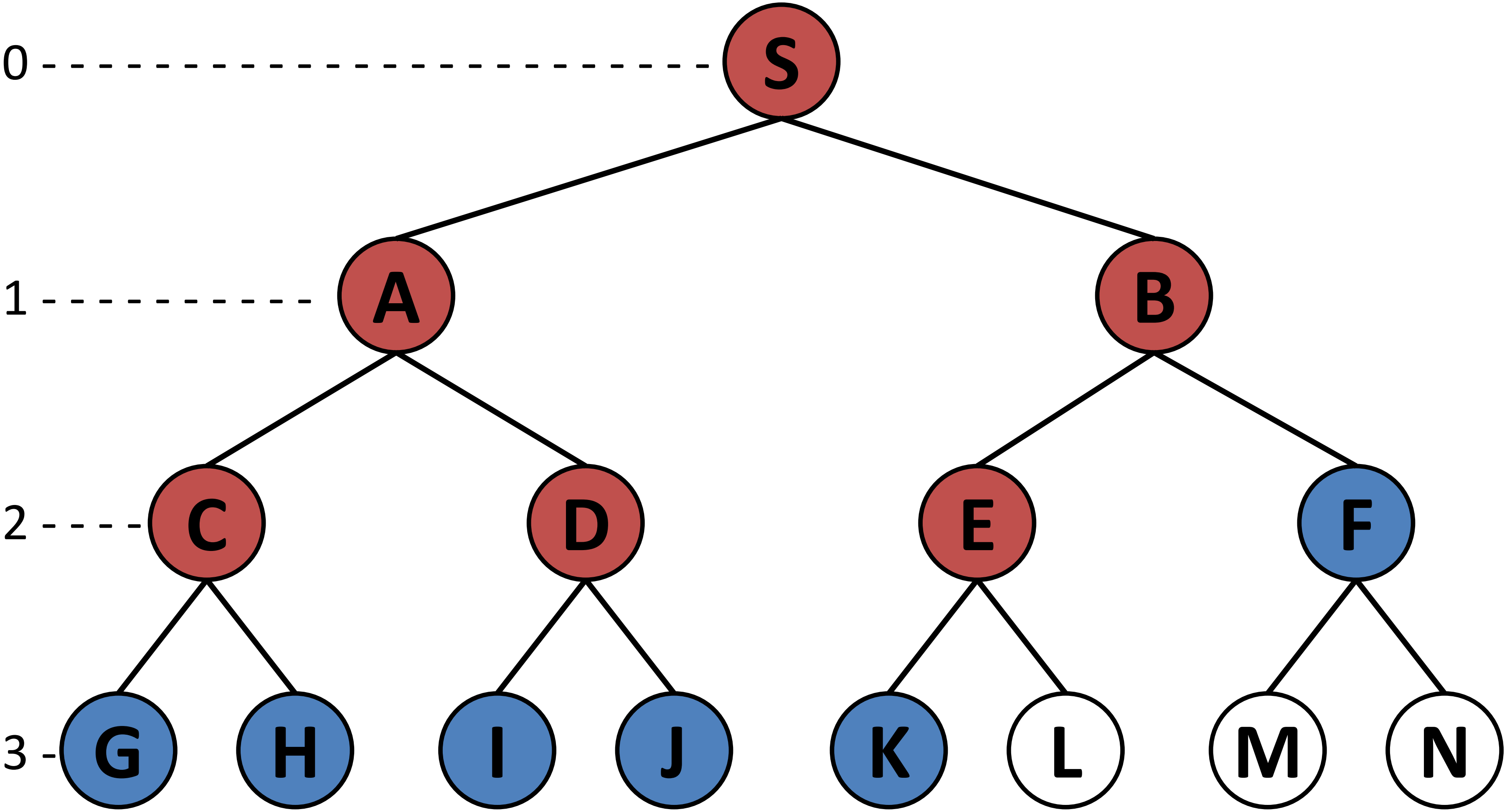
Graph and Tree Traversal: BFS



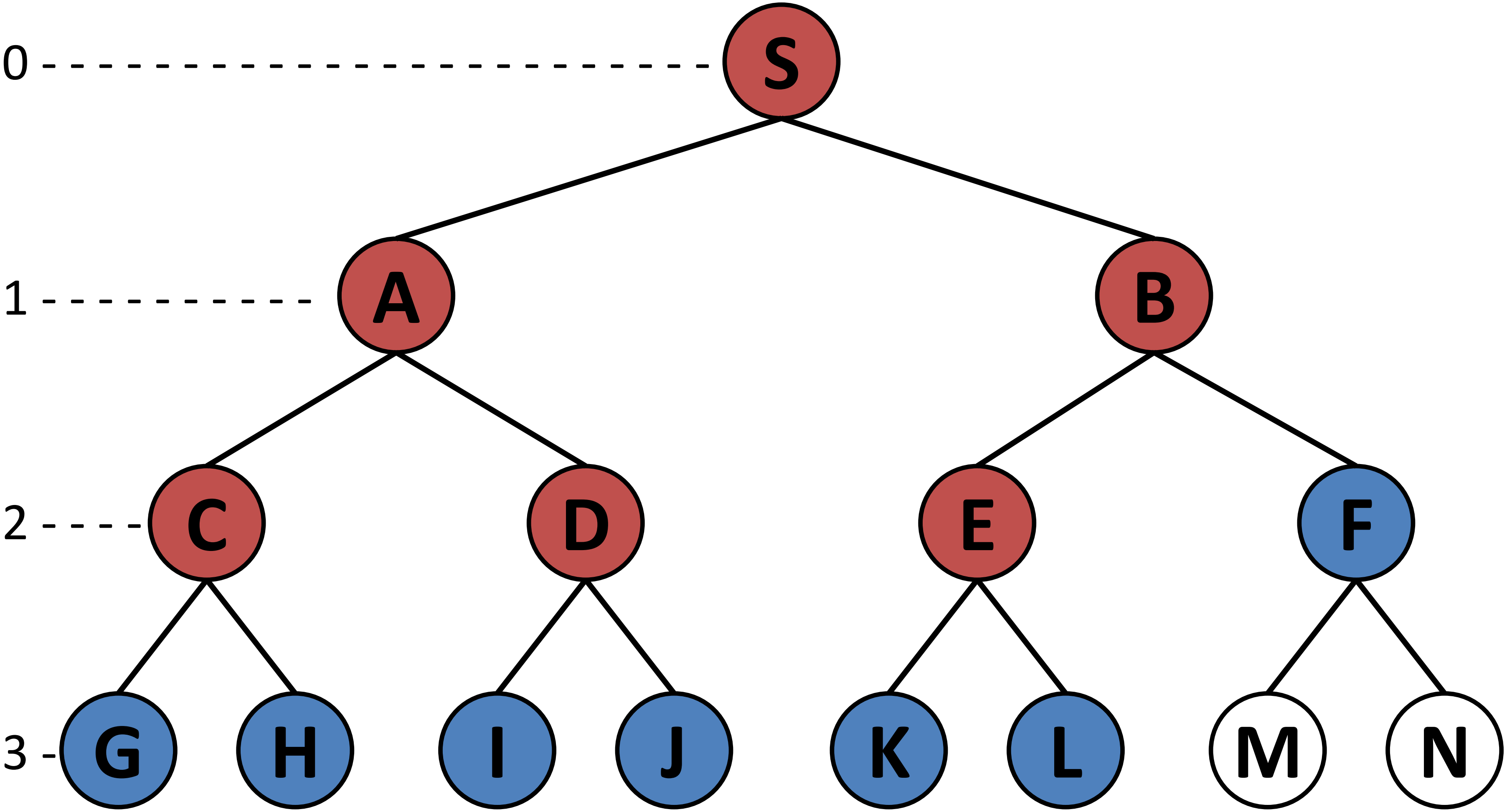
Graph and Tree Traversal: BFS



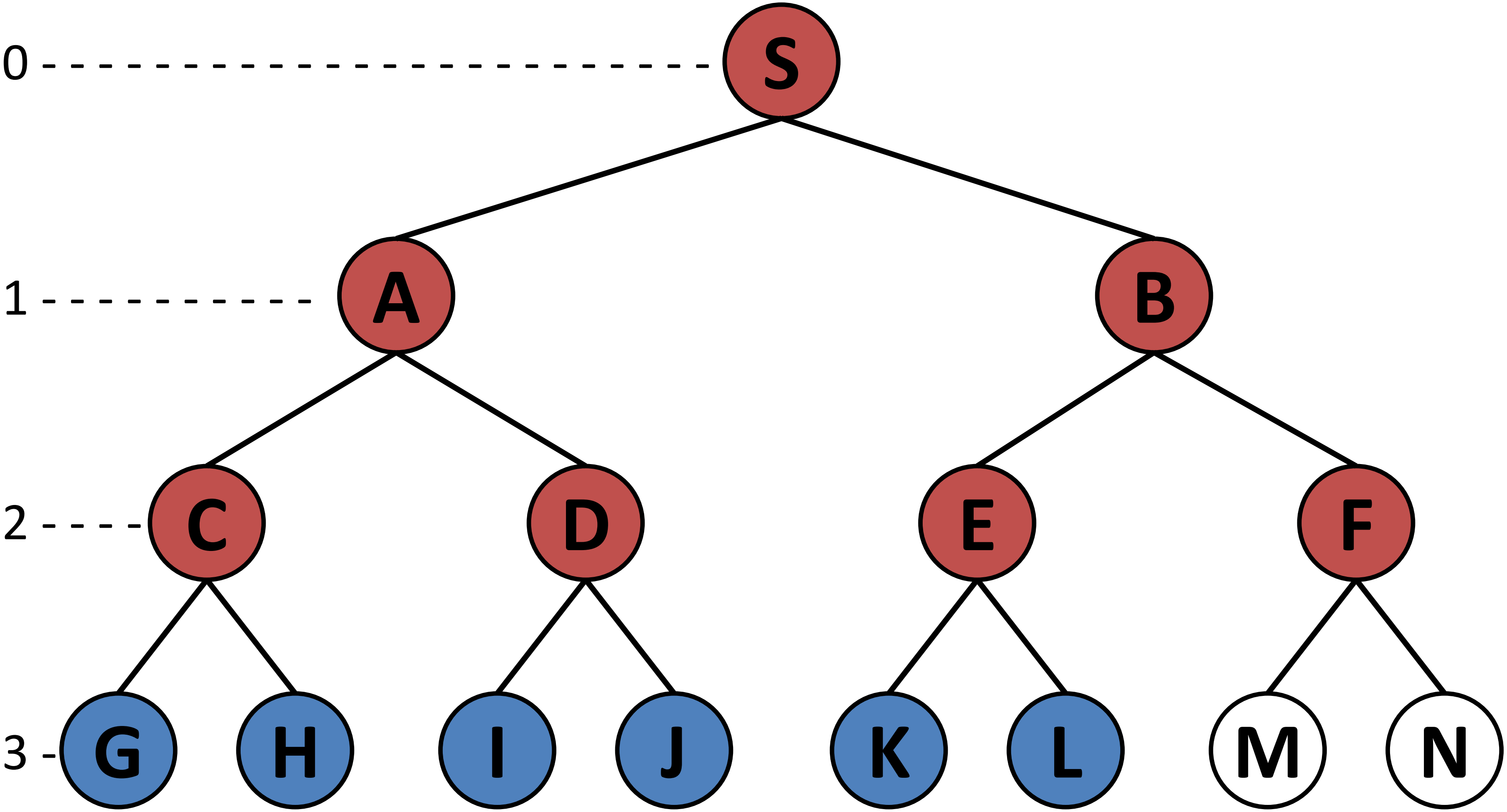
Graph and Tree Traversal: BFS



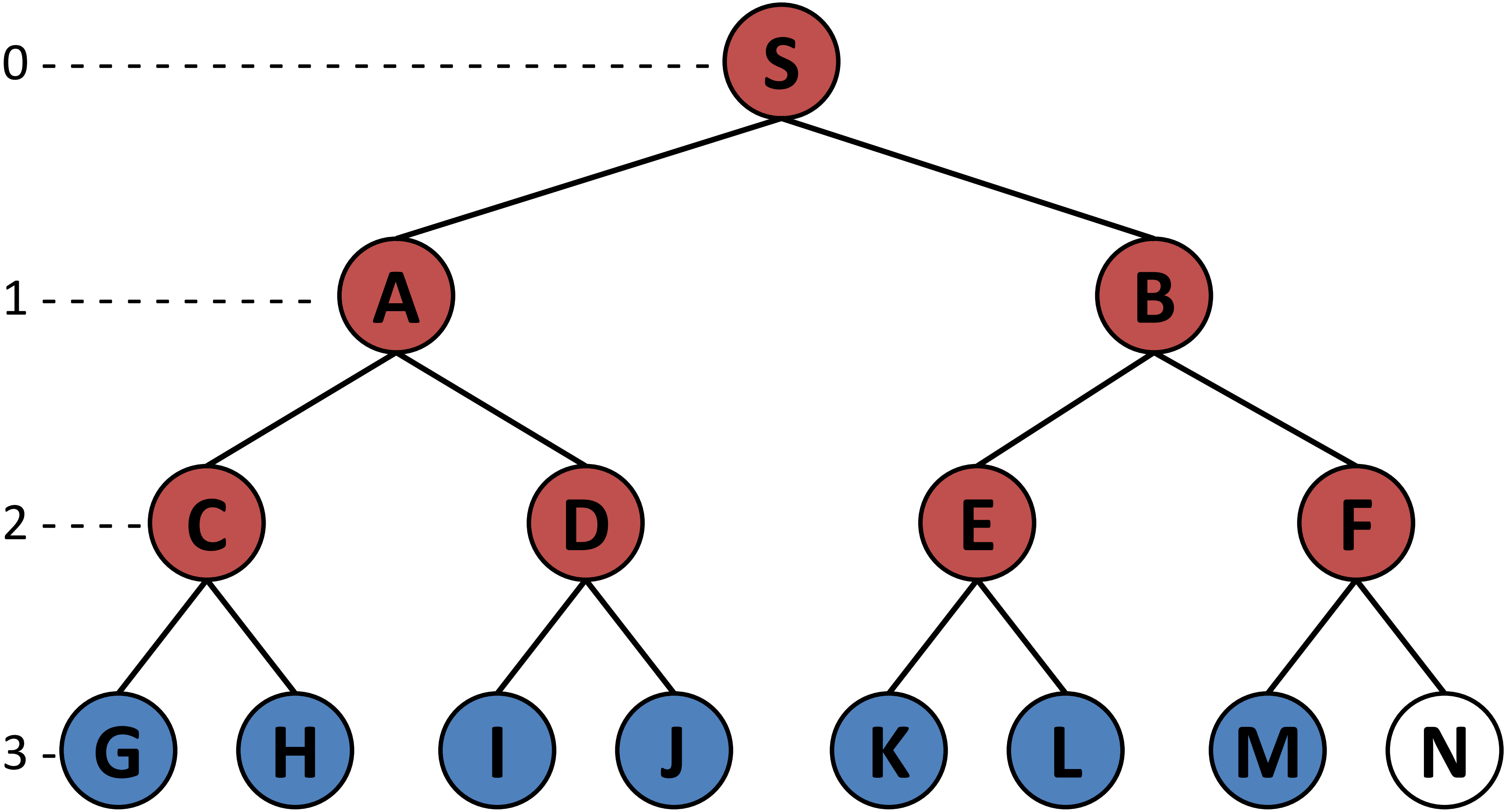
Graph and Tree Traversal: BFS



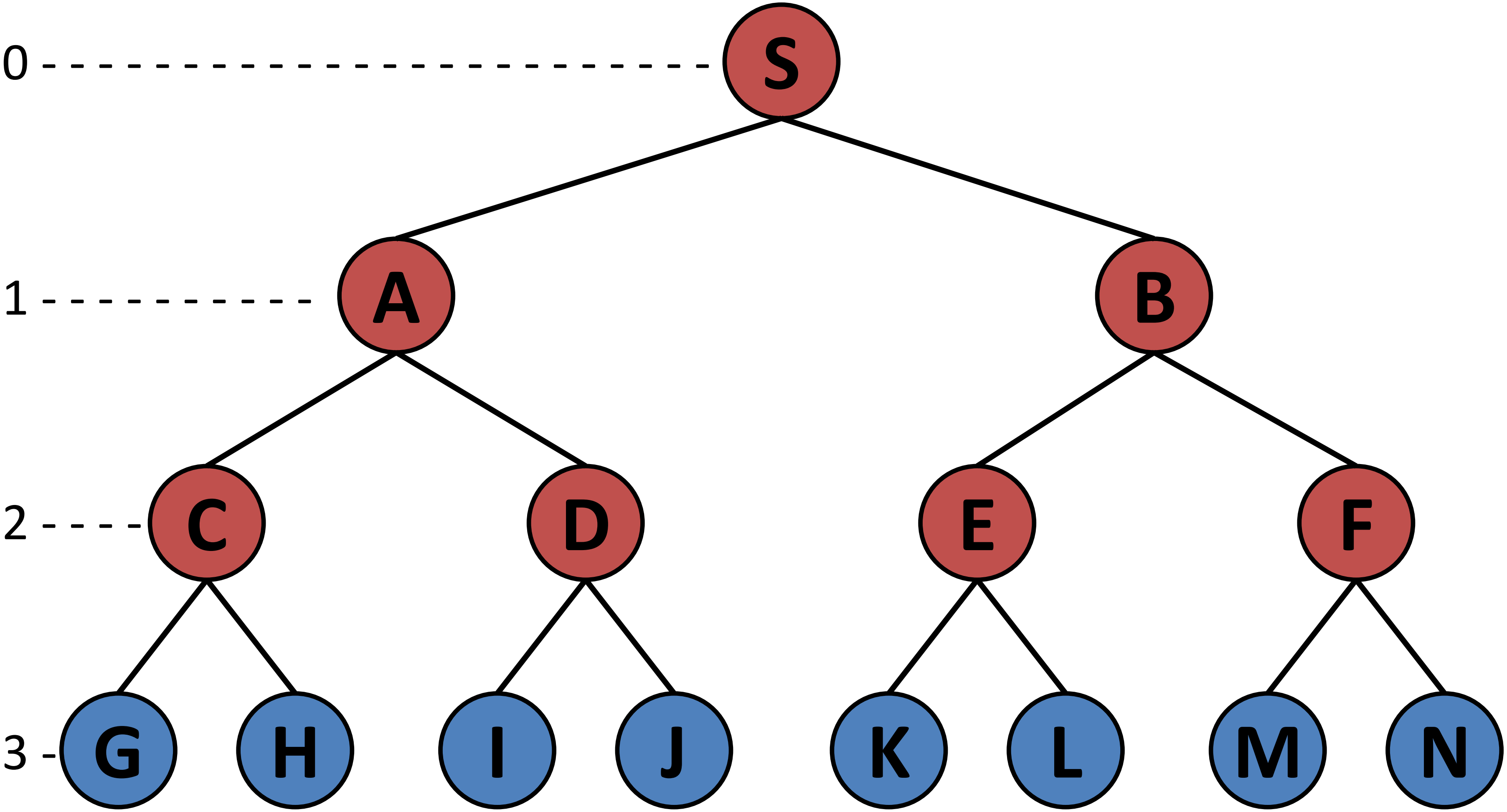
Graph and Tree Traversal: BFS



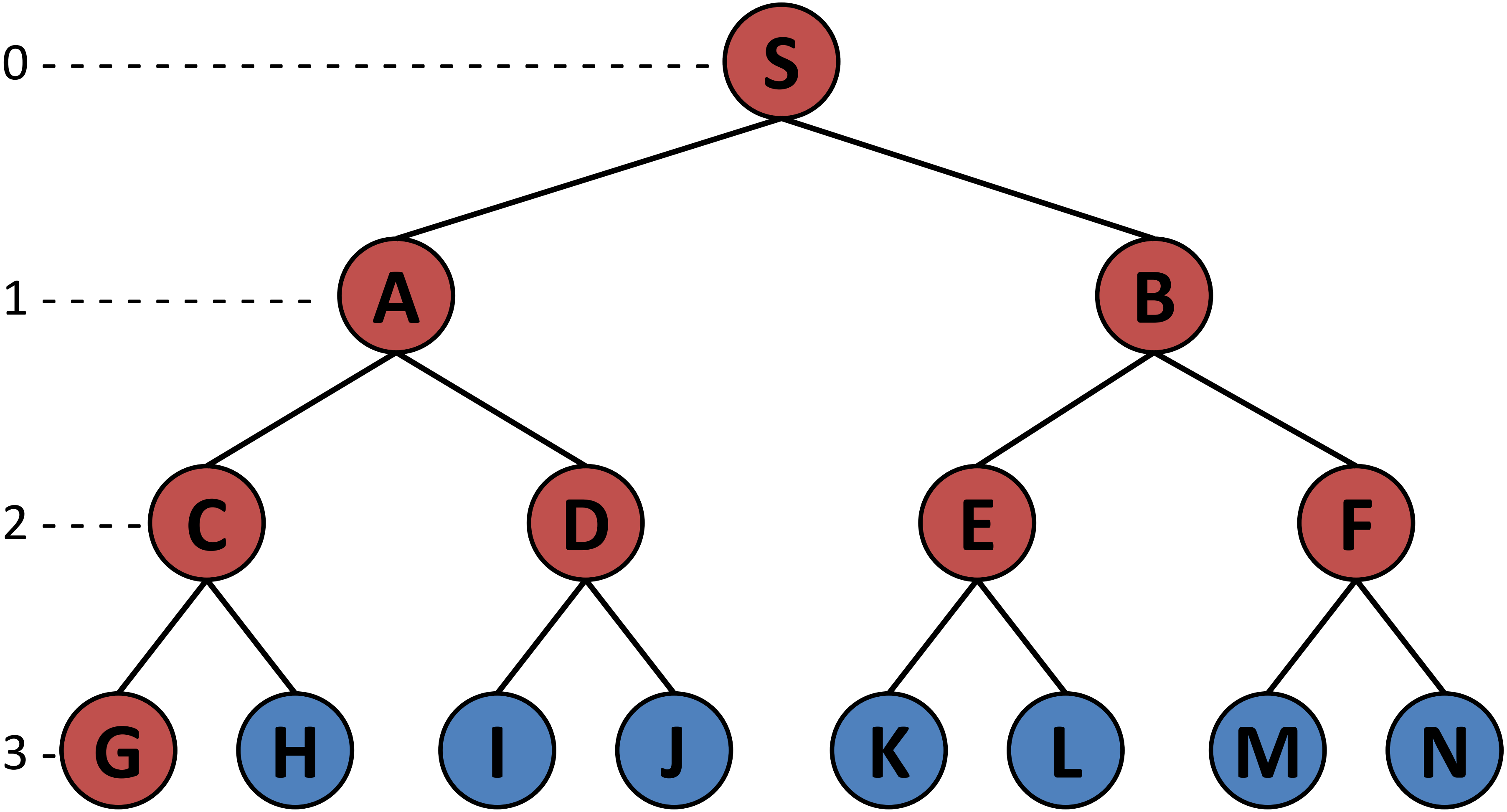
Graph and Tree Traversal: BFS



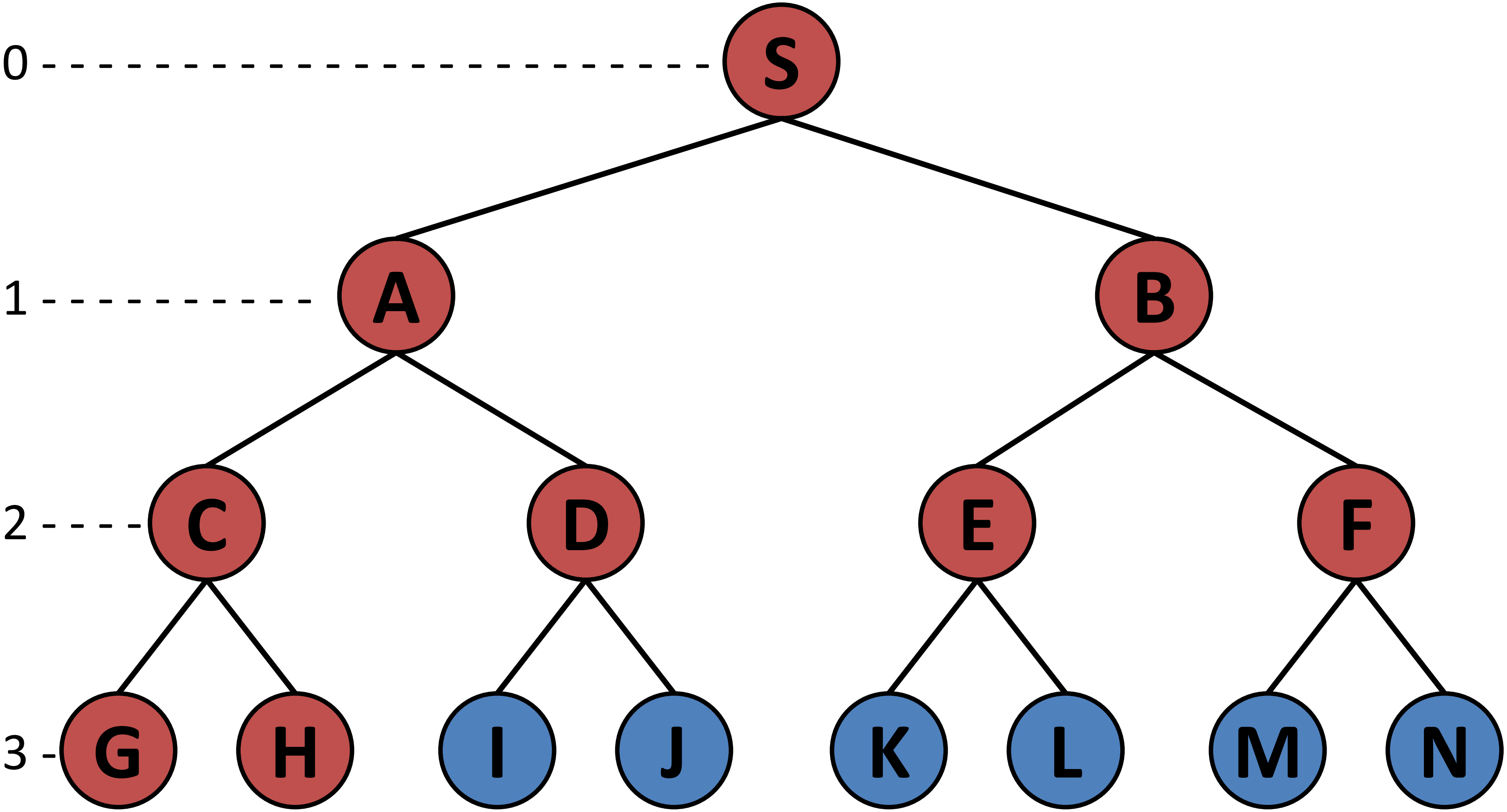
Graph and Tree Traversal: BFS



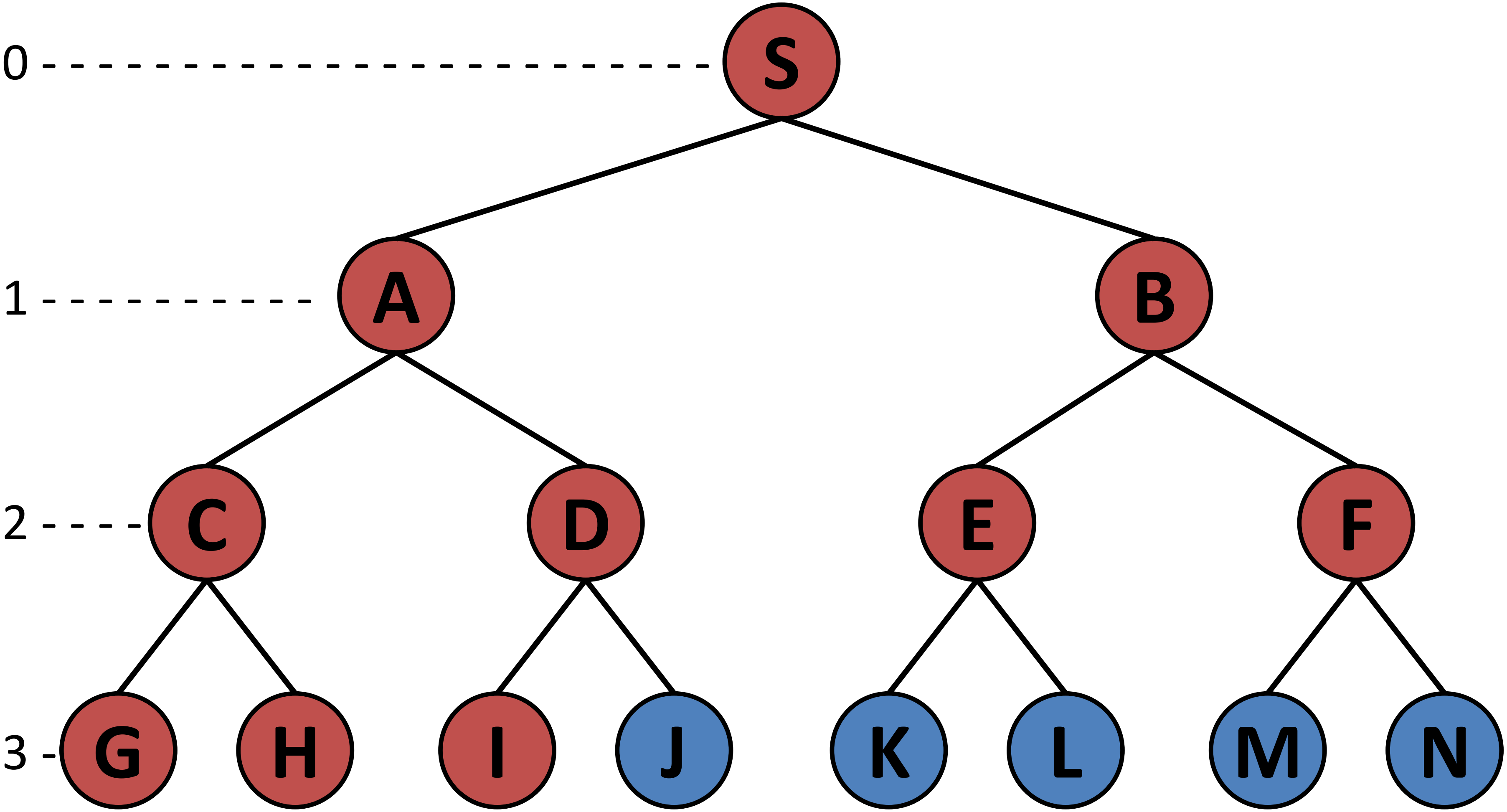
Graph and Tree Traversal: BFS



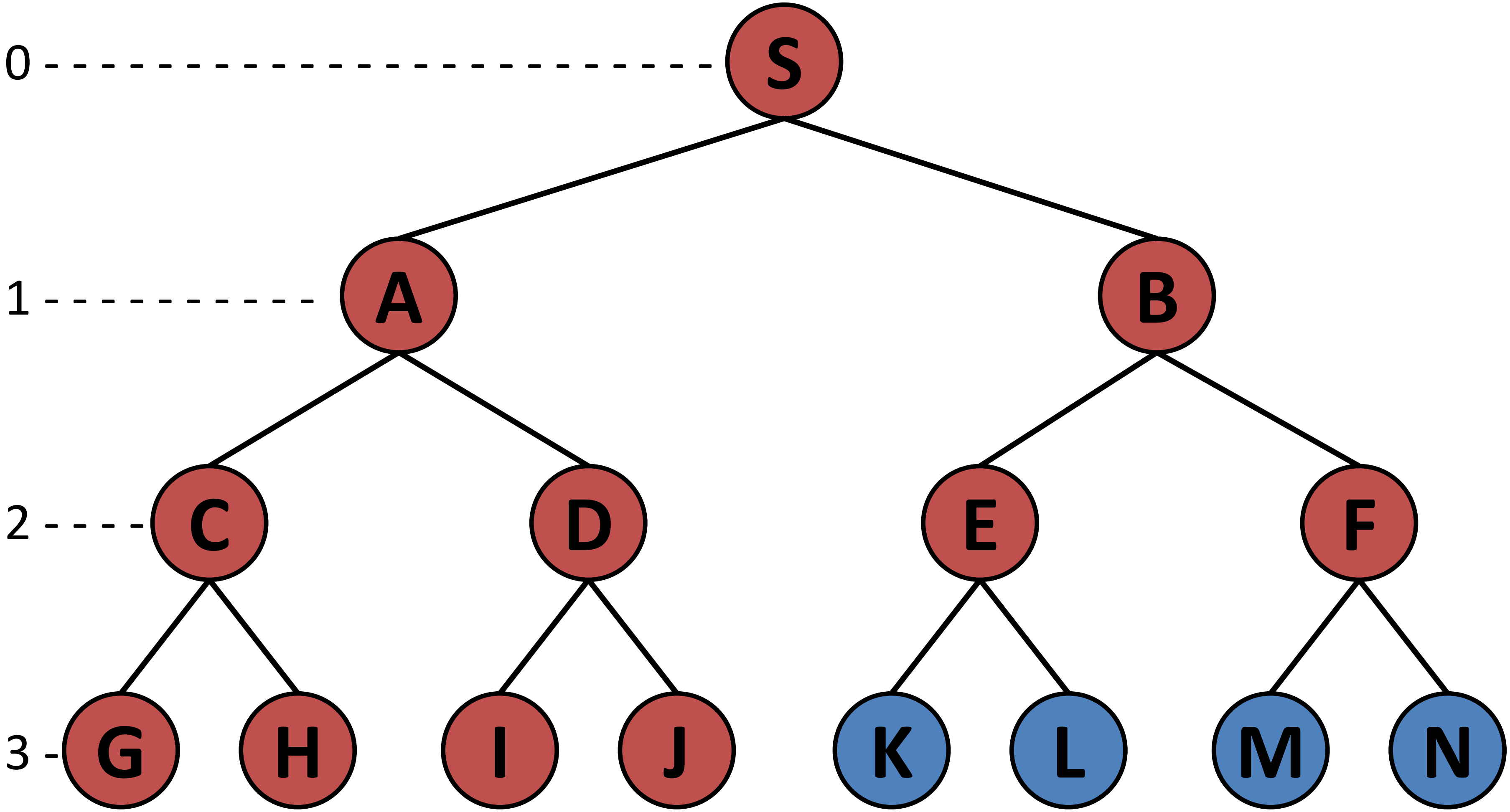
Graph and Tree Traversal: BFS



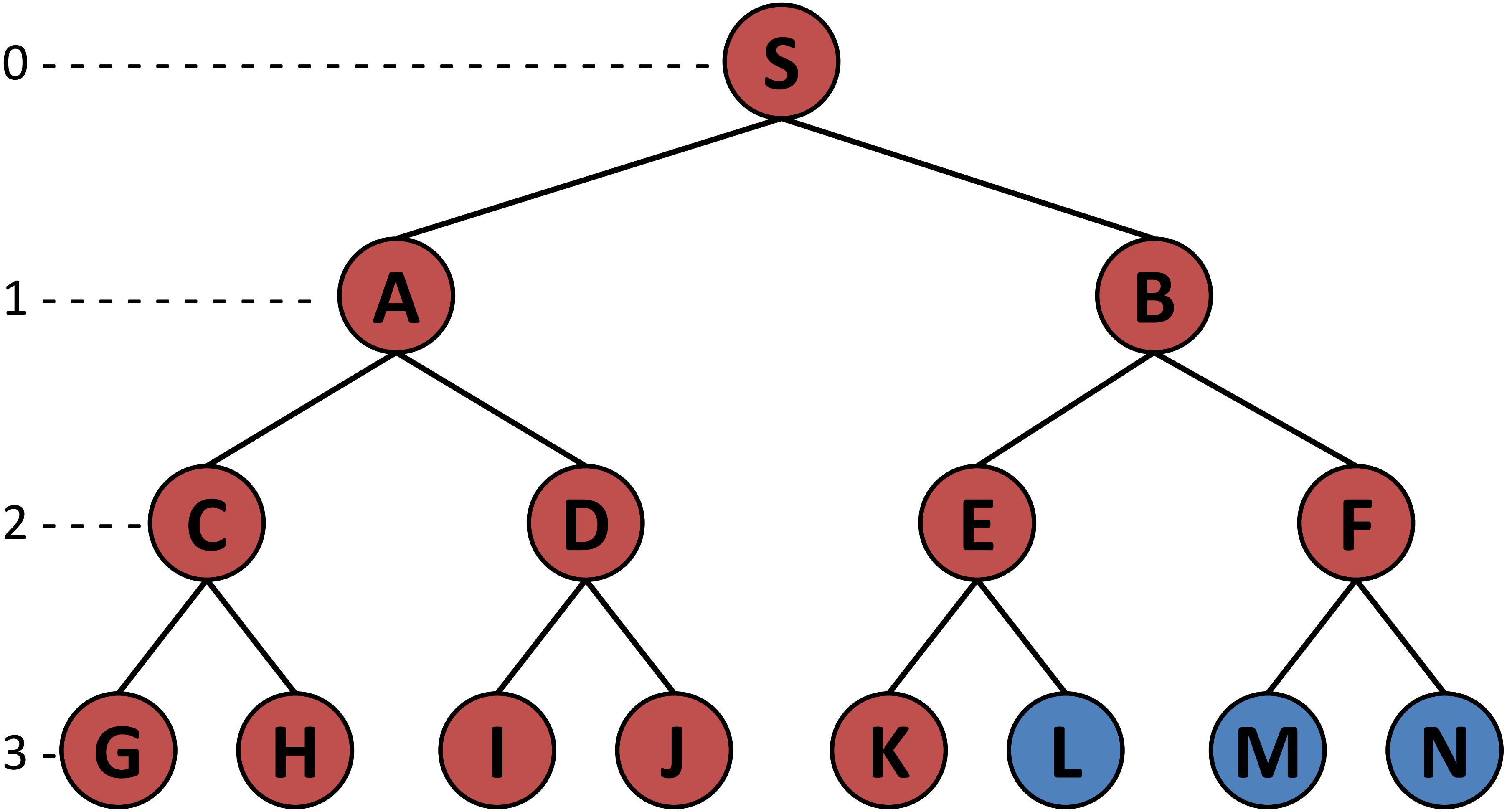
Graph and Tree Traversal: BFS



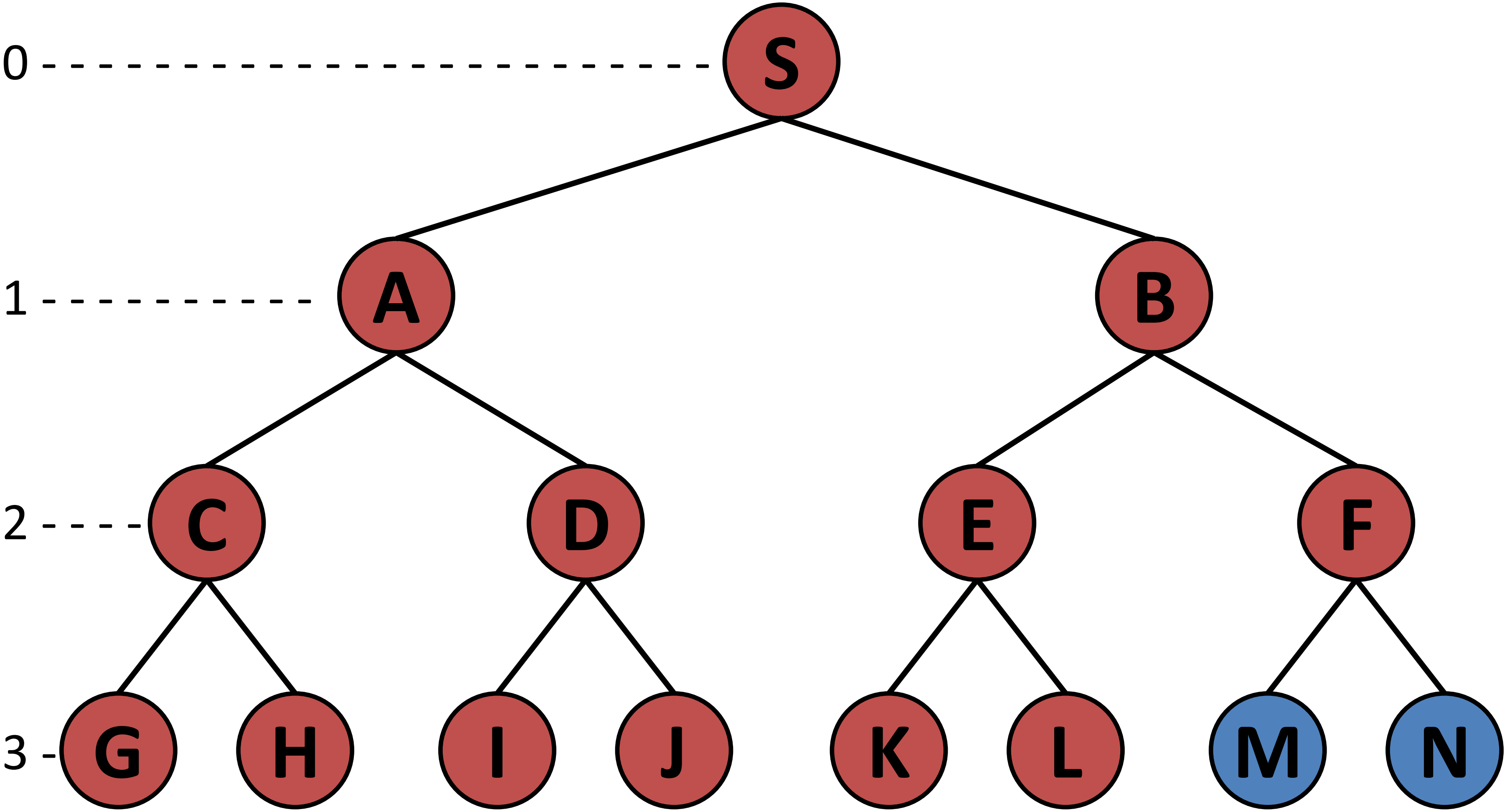
Graph and Tree Traversal: BFS



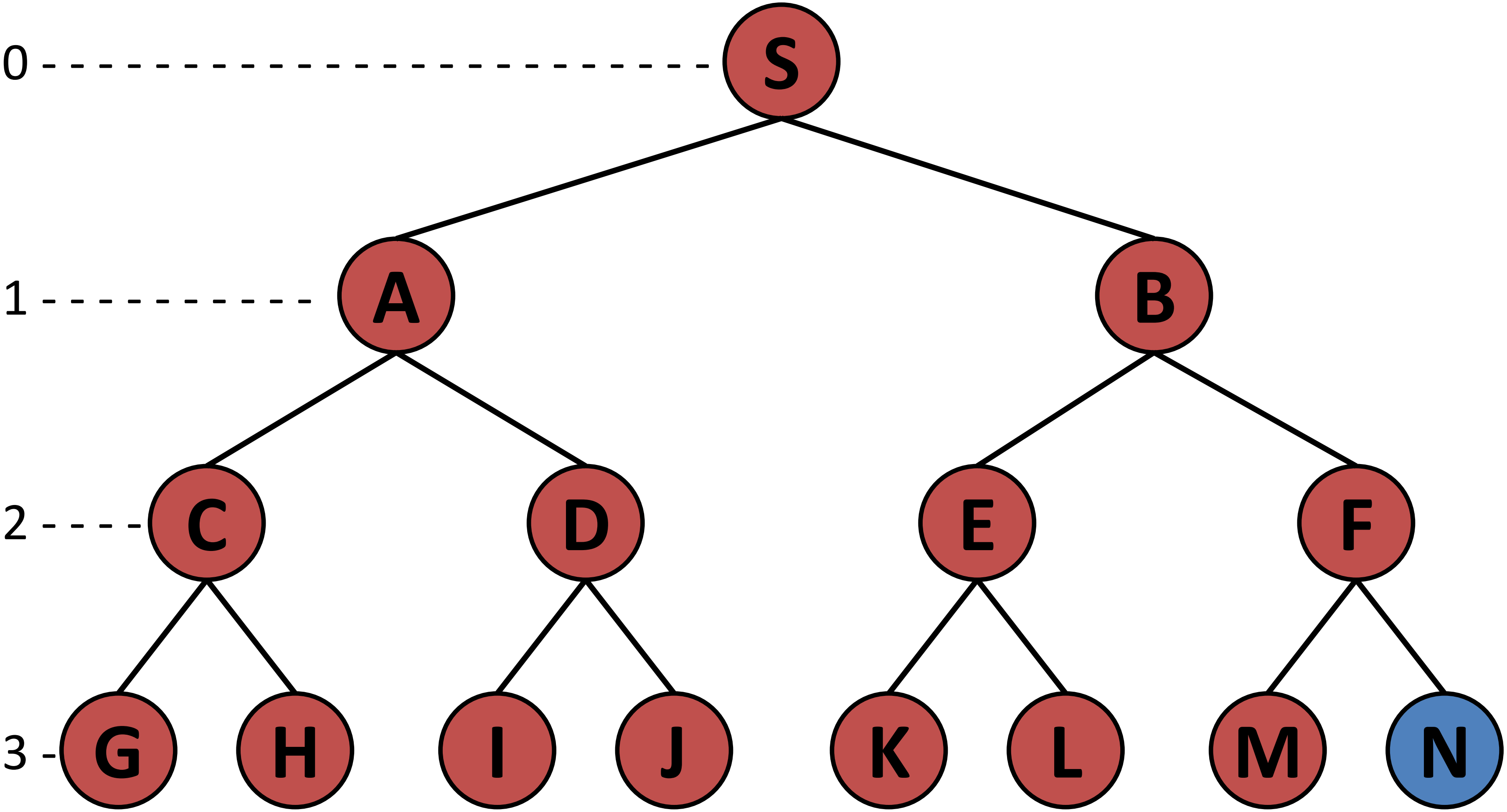
Graph and Tree Traversal: BFS



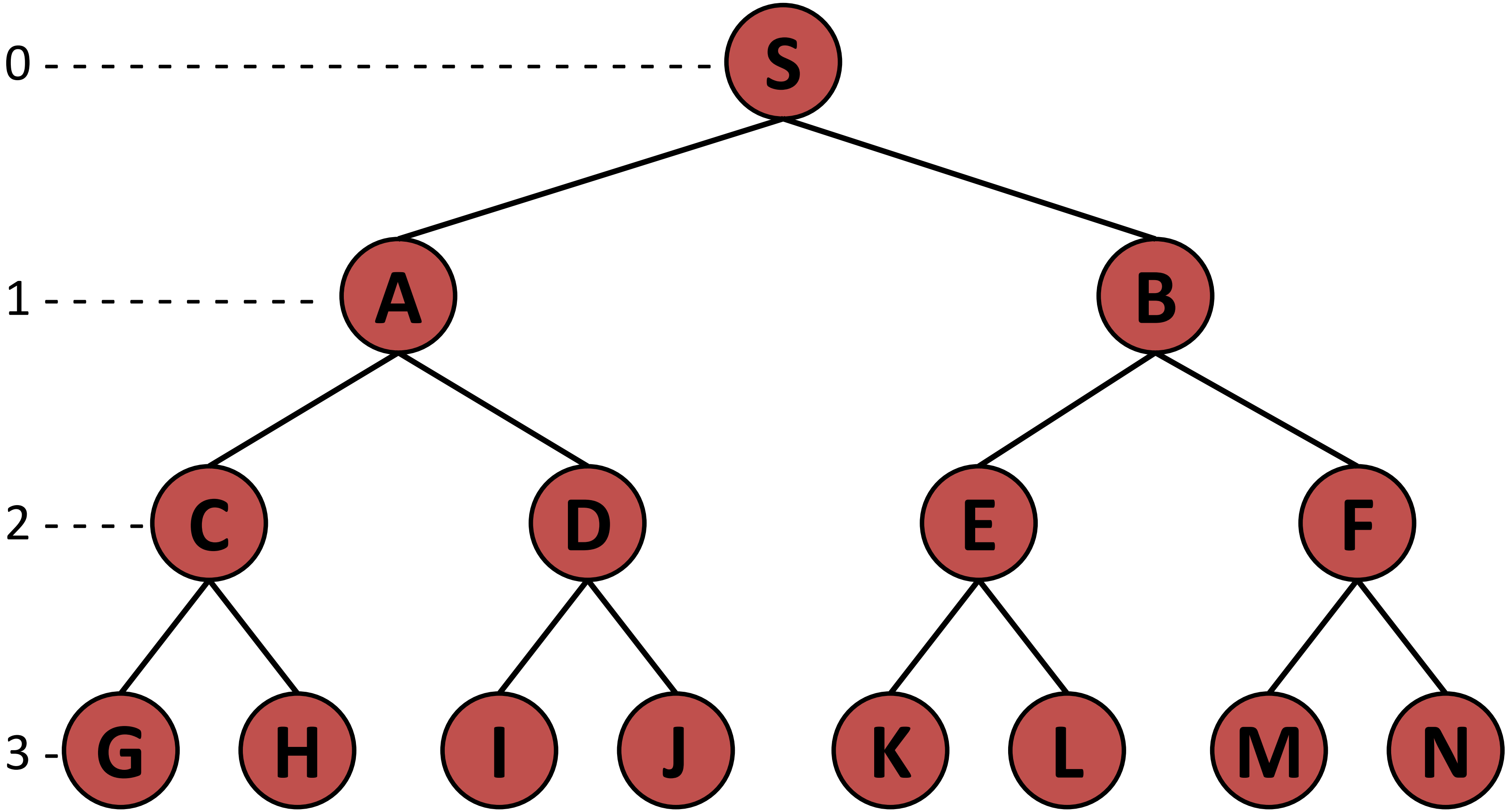
Graph and Tree Traversal: BFS



Graph and Tree Traversal: BFS



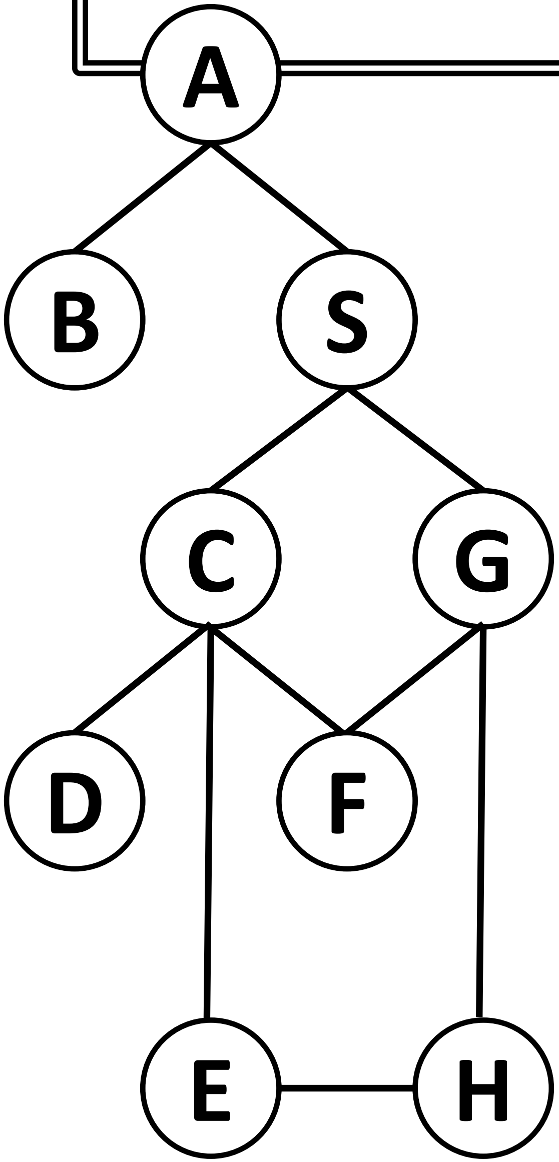
Graph and Tree Traversal: BFS

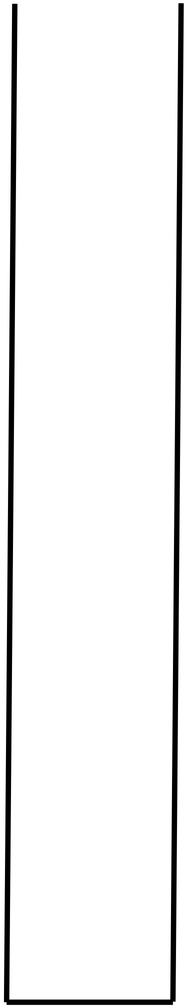


Uninformed *(Blind)* Search

- *“Uninformed strategies use only the information available in the problem definition.”*
- Uninformed Search Algorithms:
  - Depth-First Search (DFS)
  - Breadth-First Search (BFS)
  - Uniform-Cost Search (UCS)
  - Depth-Limited Search (DLS)
  - Iterative Deepening Search (IDS)

Depth-First Search





Stack (FILO)

Visited Nodes:
{}

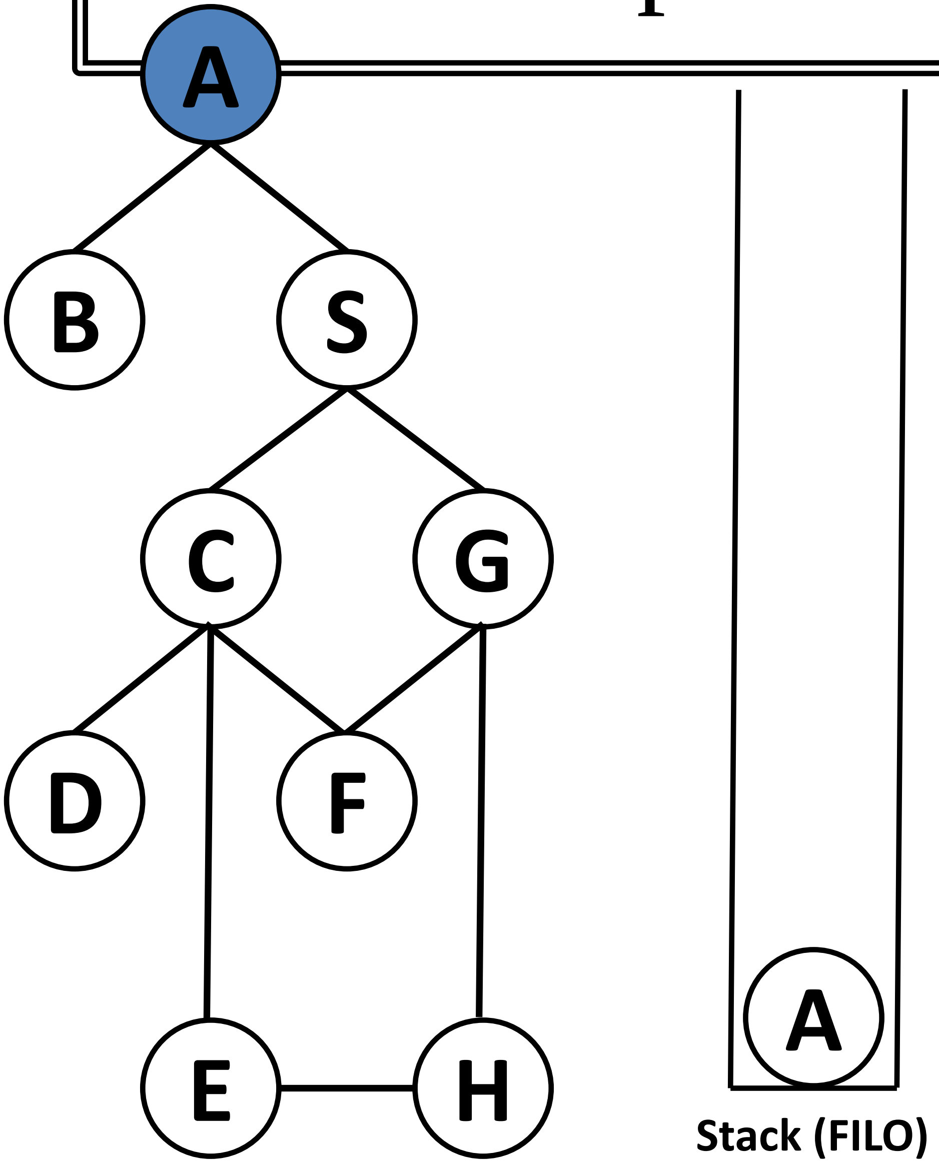
Legend:

○ S Node

🔵 S Visited Node

🔴 S Expanded Node

Depth-First Search



Visited Nodes:
{A}

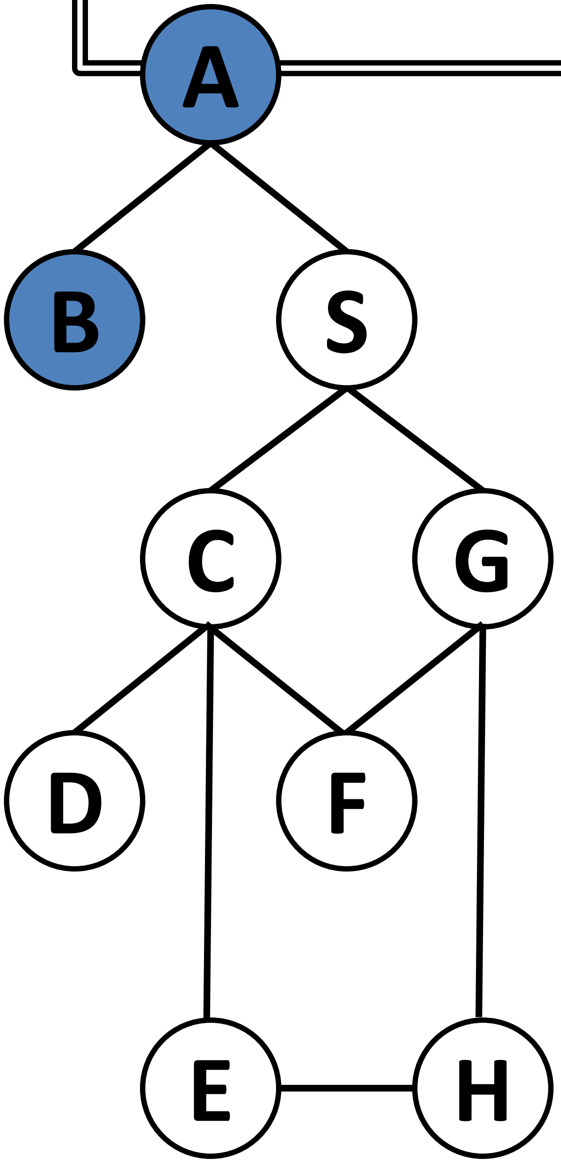
Legend:

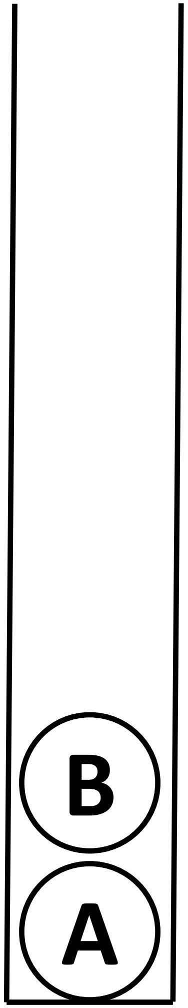
○ **Node**

🔵 **Visited Node**

🔴 **Expanded Node**

Depth-First Search





Stack (FILO)

Visited Nodes:
{A,B}

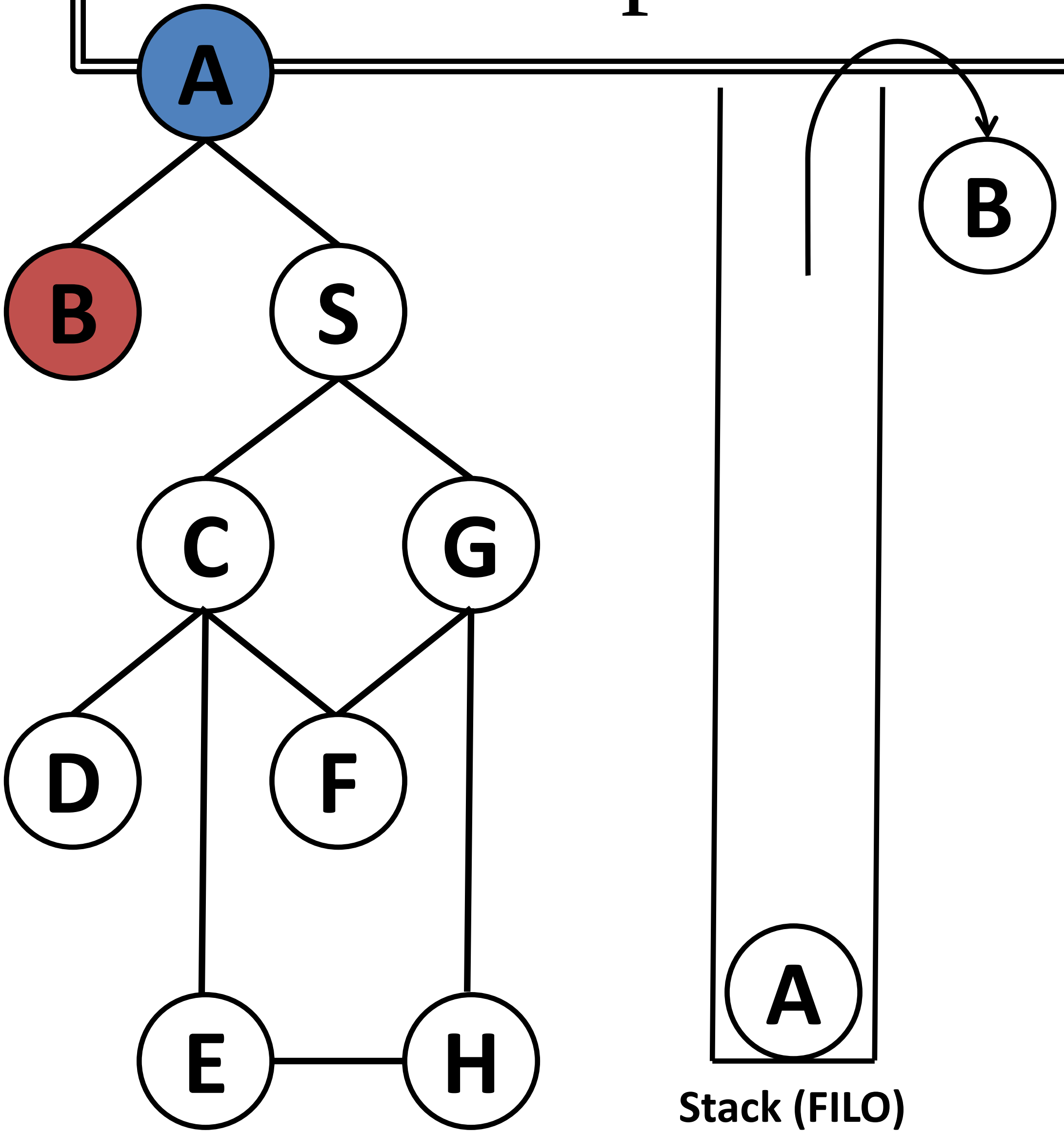
Legend:

○ S **Node**

🔵 S **Visited Node**

🔴 S **Expanded Node**

Depth-First Search



Visited Nodes:
{A,B}

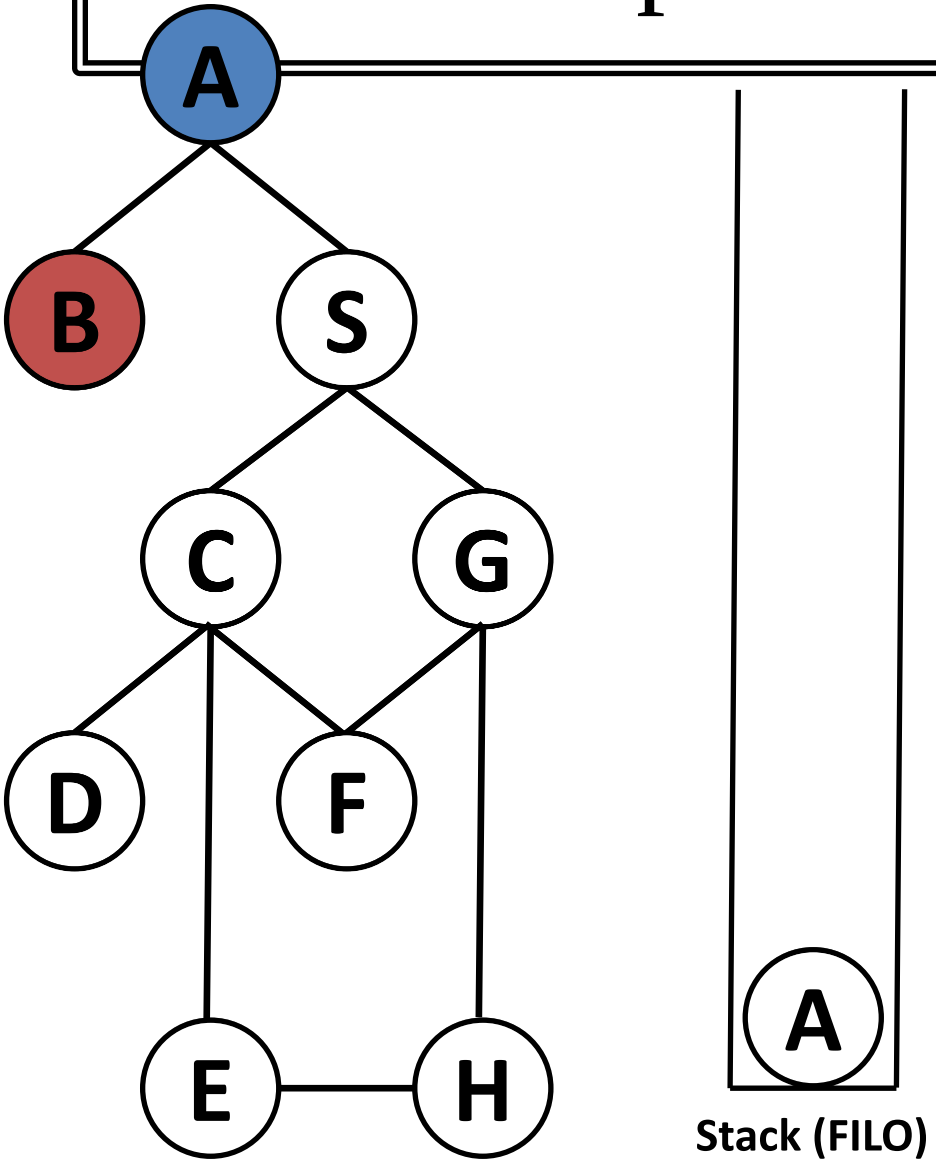
Legend:

S **Node**

S **Visited Node**

S **Expanded Node**

Depth-First Search



Visited Nodes:
{A,B}

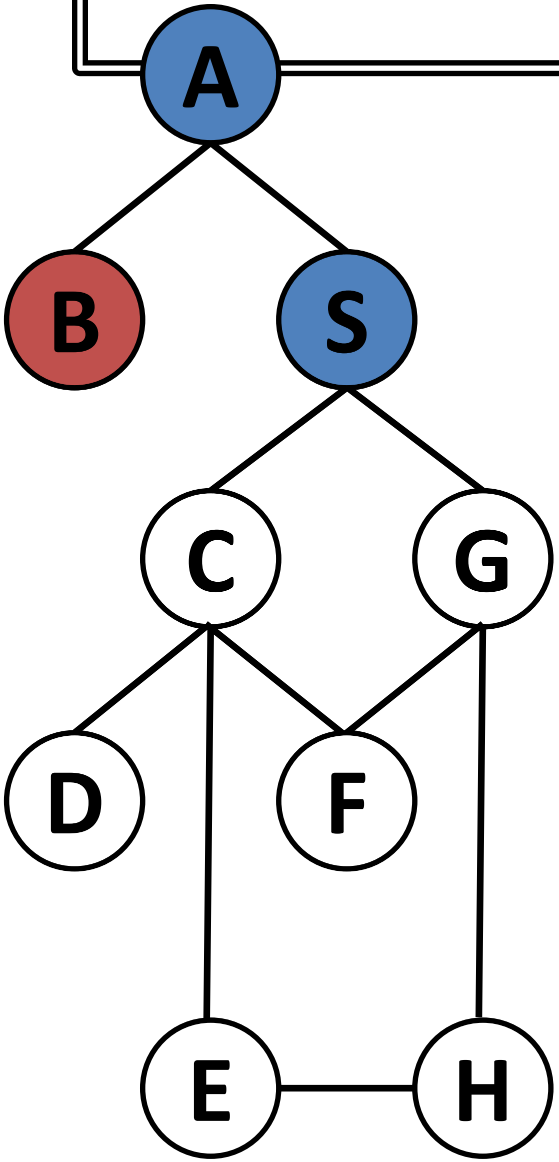
Legend:

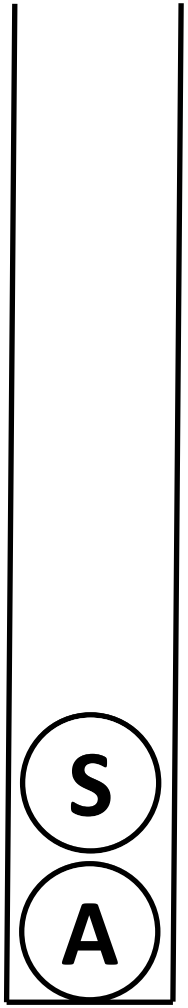
S **Node**

S **Visited Node**

S **Expanded Node**

Depth-First Search





Stack (FILO)

Visited Nodes:
{A,B,S}

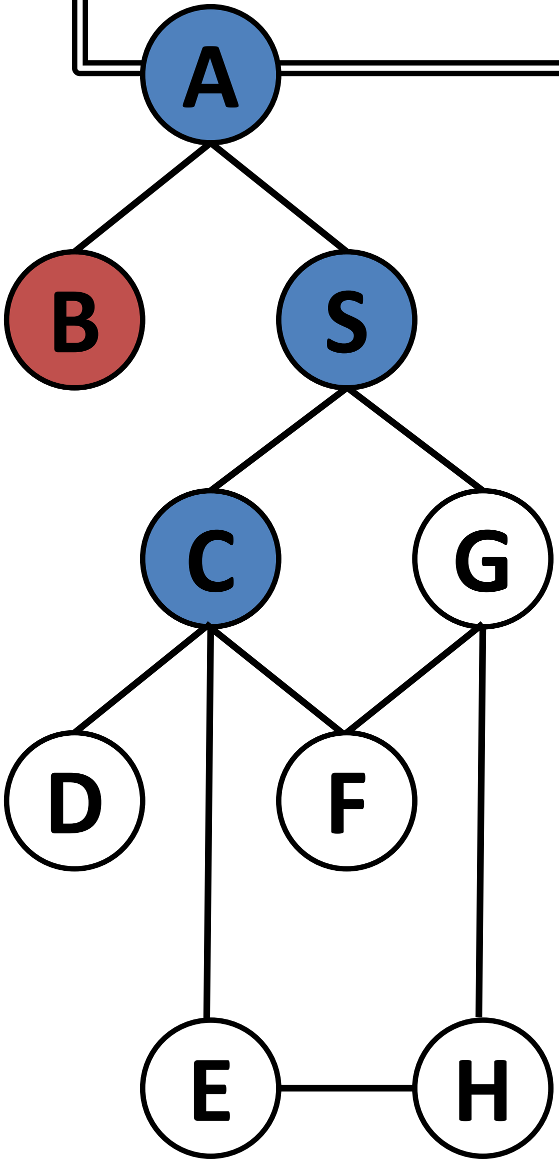
Legend:

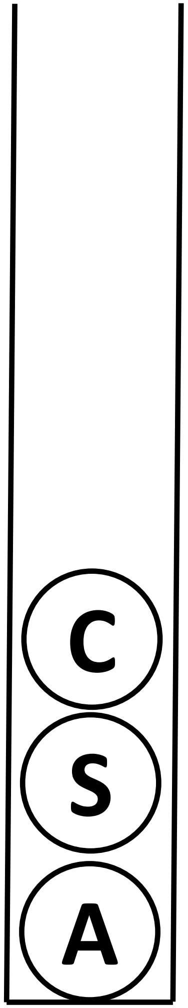
S **Node**

S **Visited Node**

S **Expanded Node**

Depth-First Search





Stack (FILO)

Visited Nodes:
{A,B,S,C}

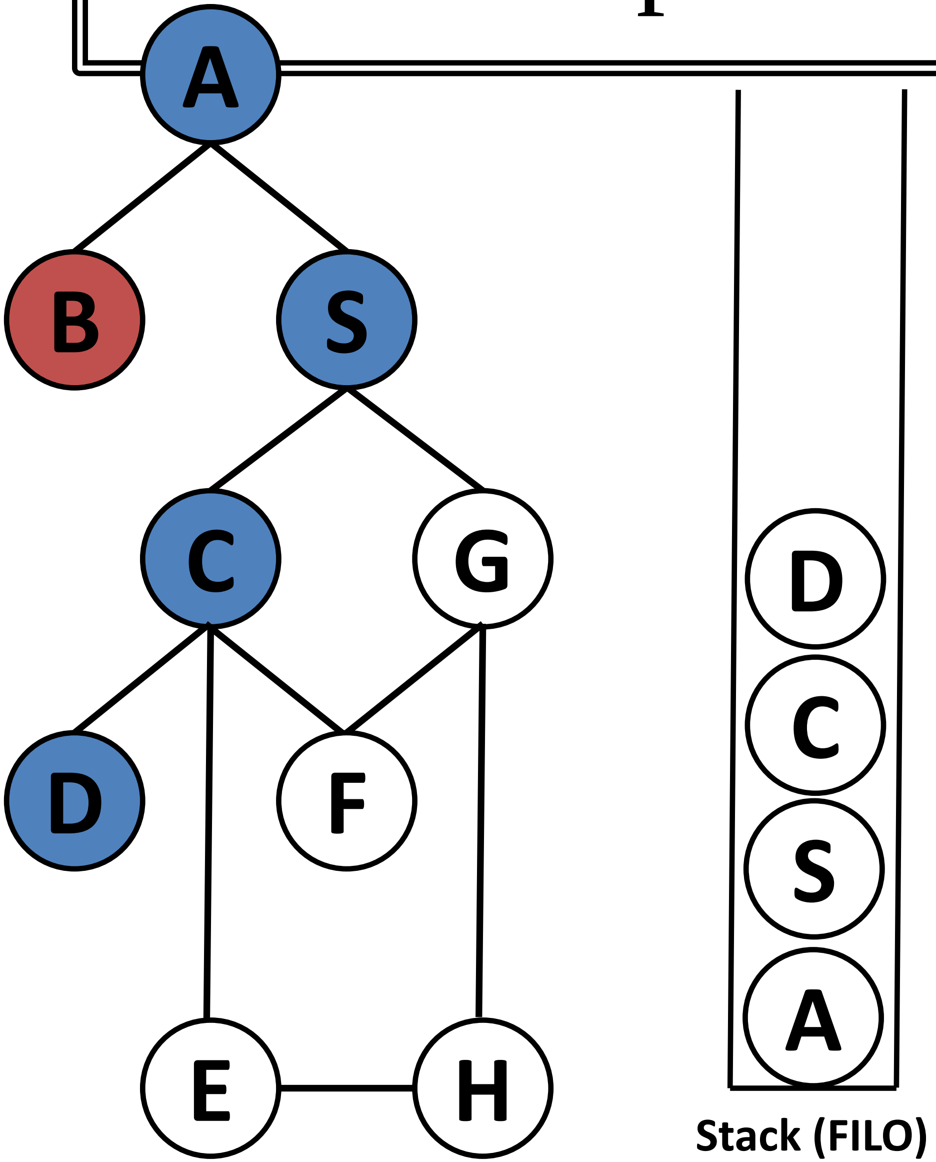
Legend:

S **Node**

S **Visited Node**

S **Expanded Node**

Depth-First Search



Visited Nodes:
{A,B,S,C,D}

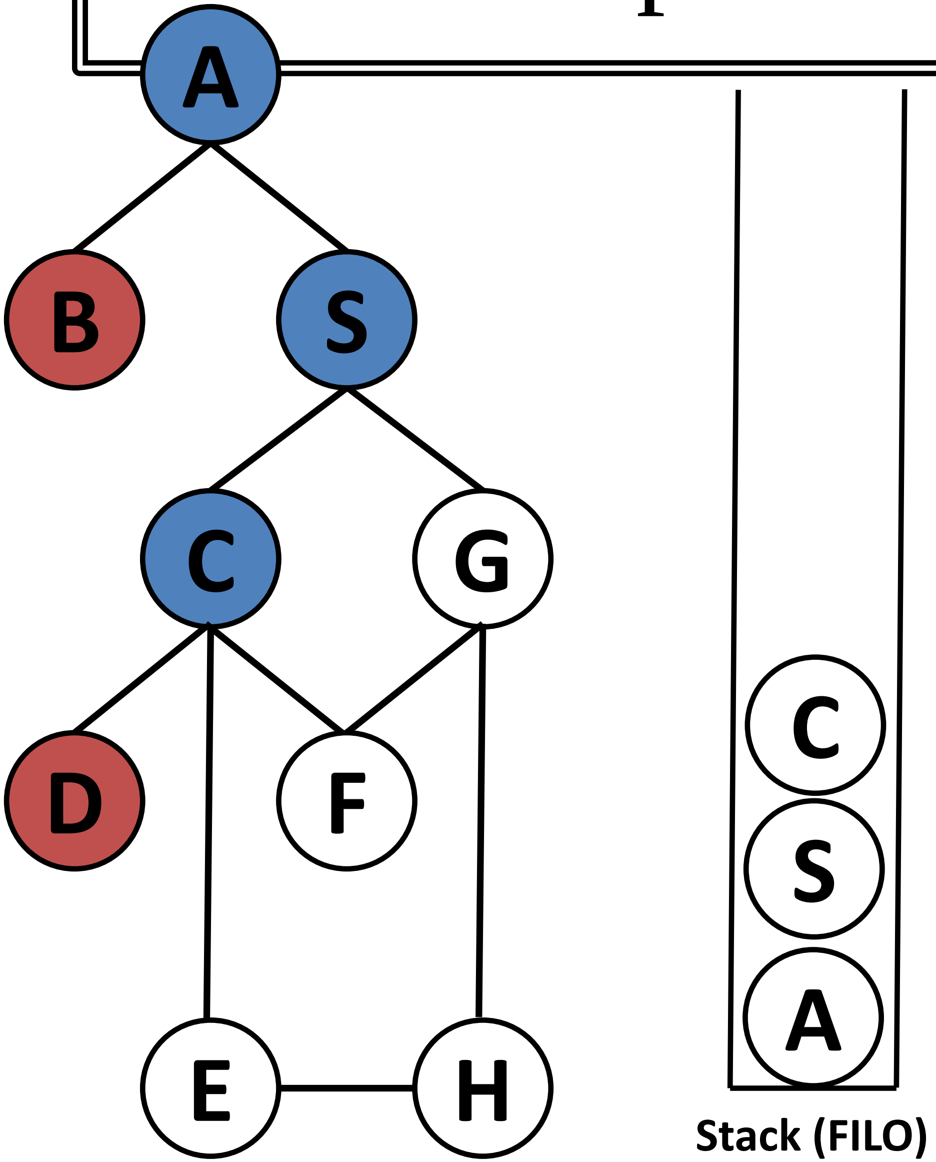
Legend:

○ S **Node**

🟦 S **Visited Node**

🟥 S **Expanded Node**

Depth-First Search



Visited Nodes:
{A,B,S,C,D}

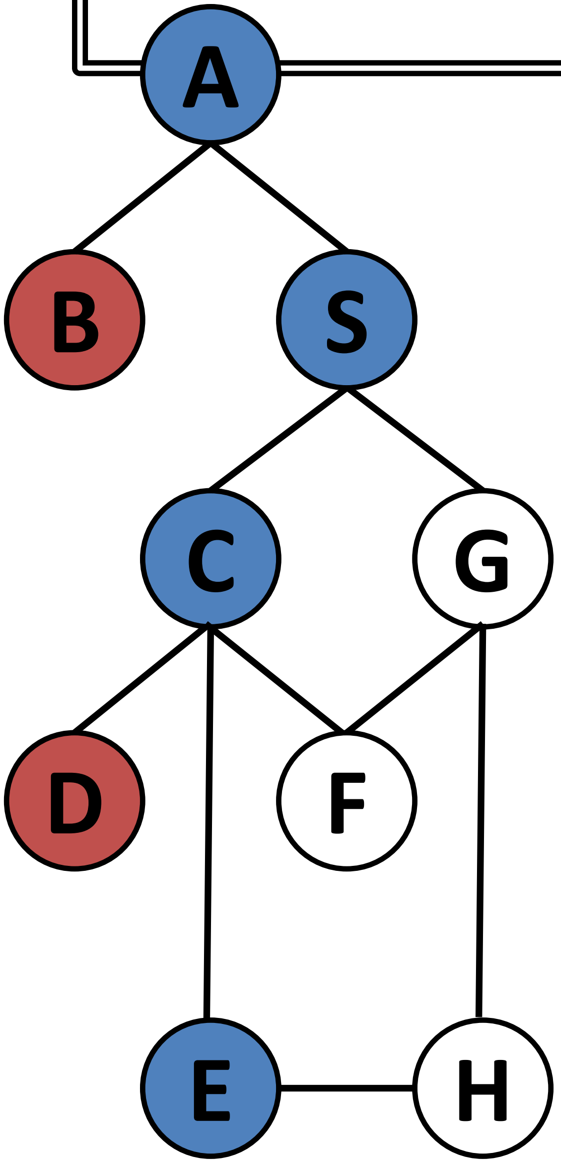
Legend:

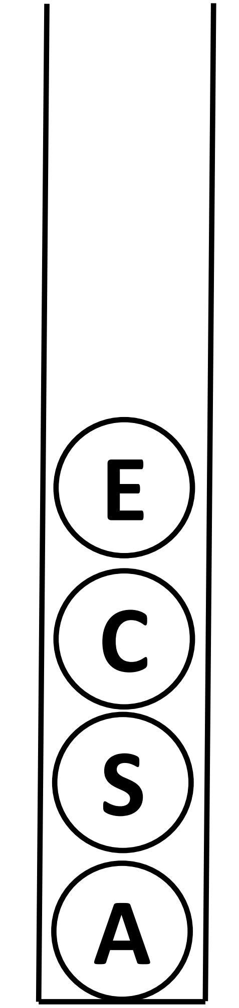
S **Node**

S **Visited Node**

S **Expanded Node**

Depth-First Search





Stack (FILO)

Visited Nodes:
{A,B,S,C,D,E}

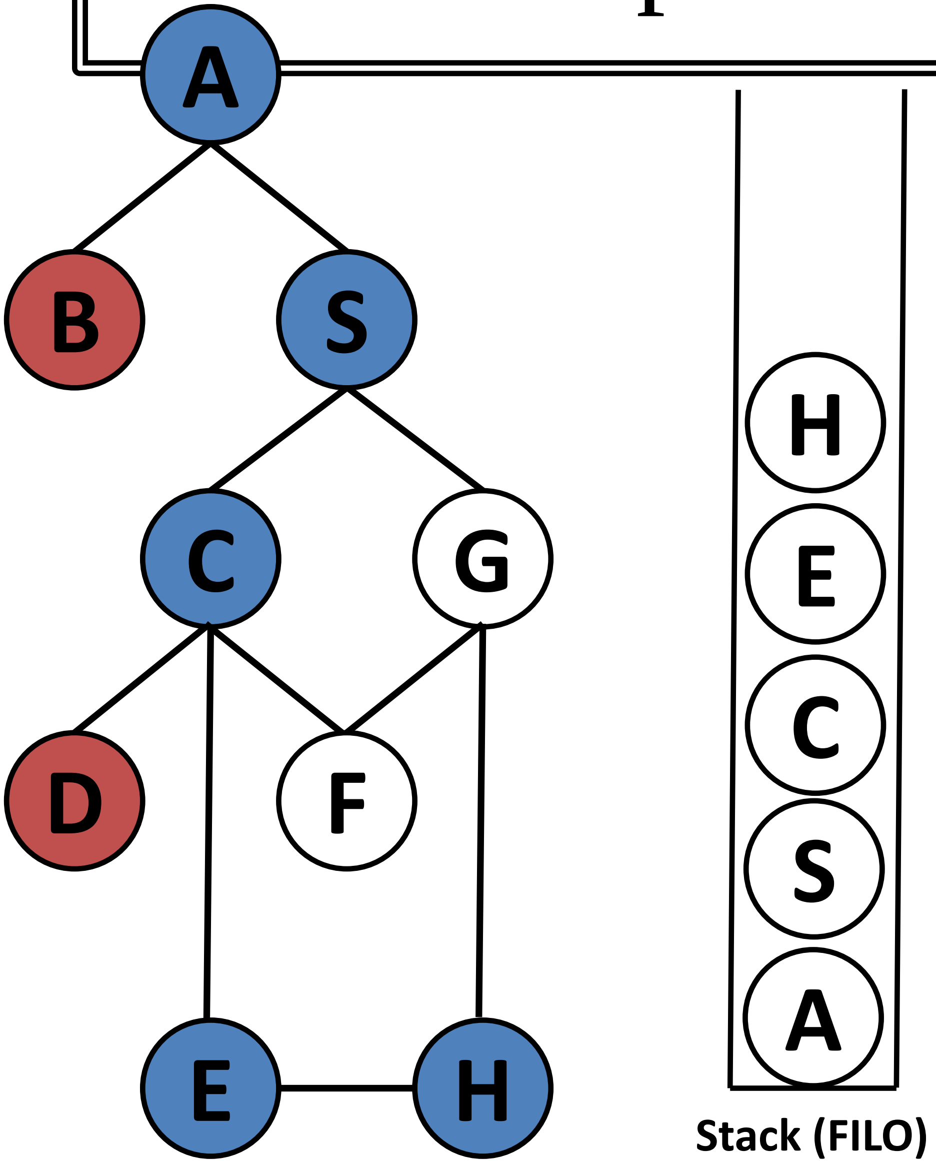
Legend:

○ S **Node**

● S **Visited Node**

● S **Expanded Node**

Depth-First Search

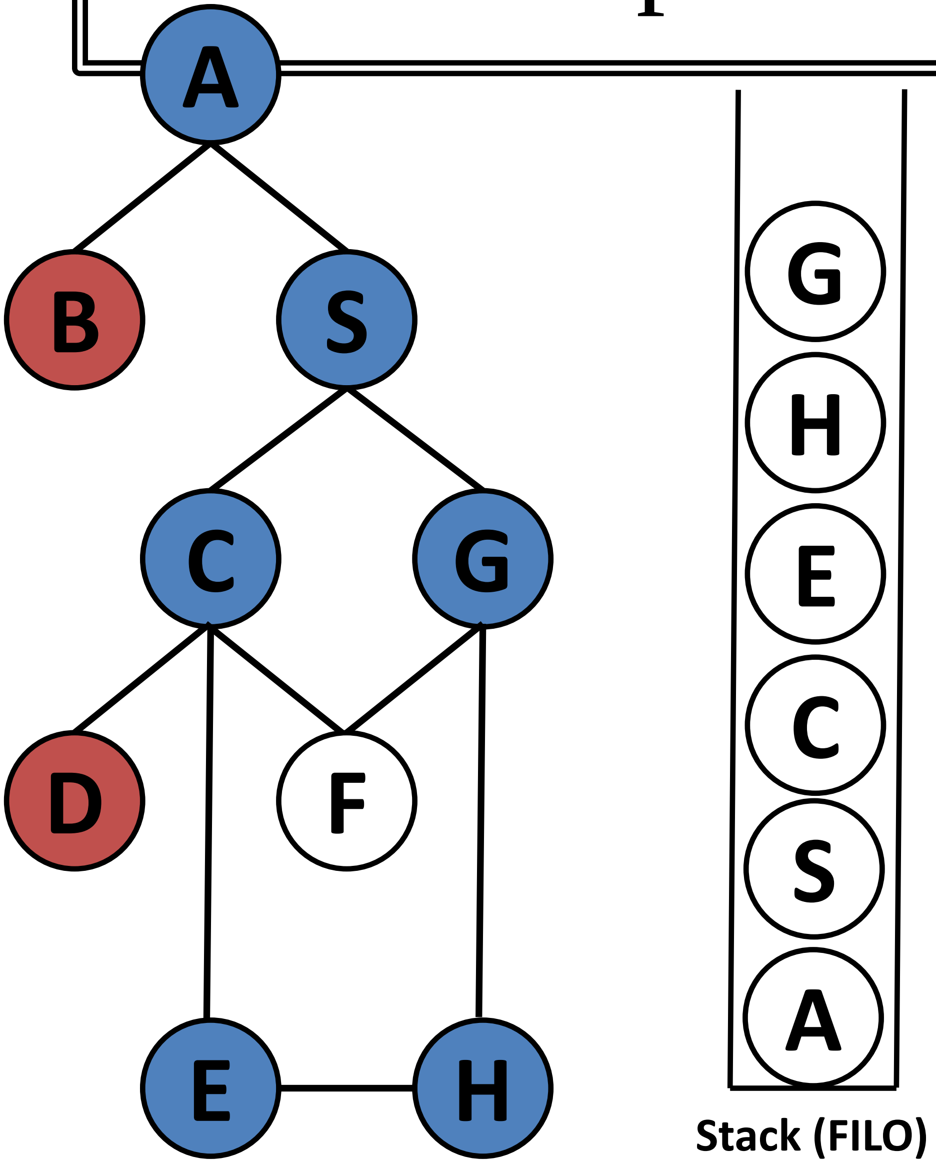


Visited Nodes:
{A,B,S,C,D,E,H}

Legend:

- ○ S **Node**
- 🟦 S **Visited Node**
- 🟥 S **Expanded Node**

Depth-First Search

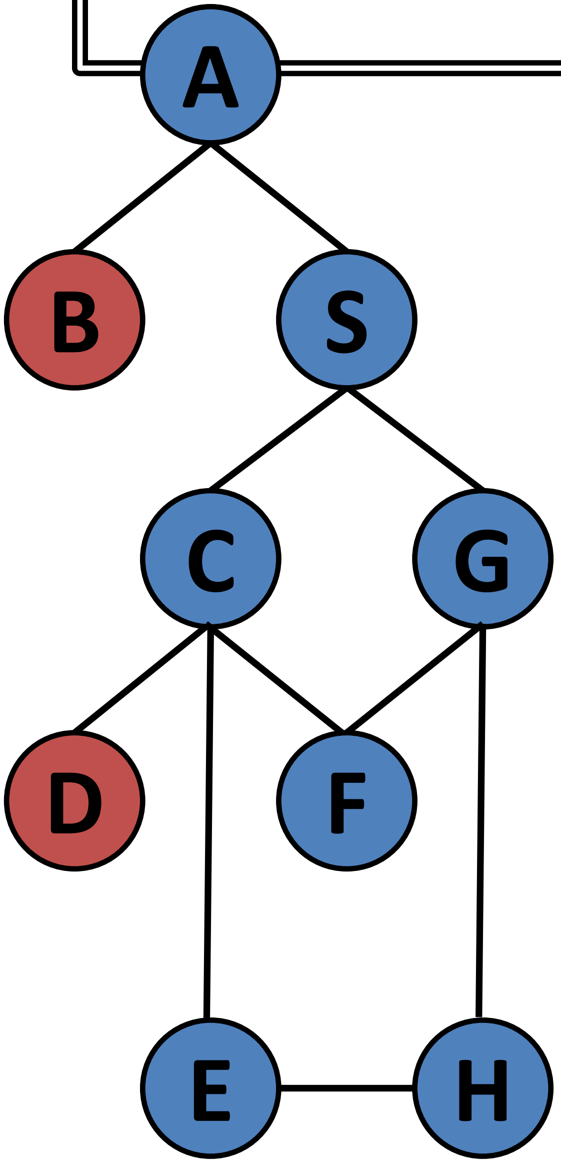


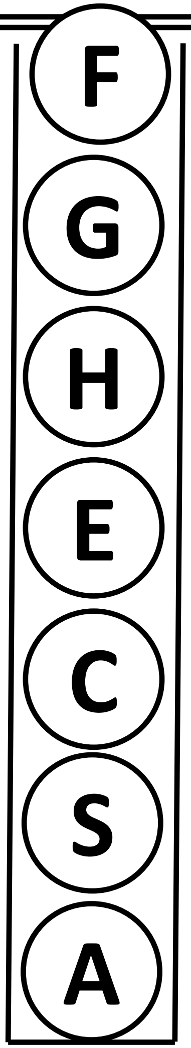
Visited Nodes:
{A,B,S,C,D,E,H,G}

Legend:

- ○ S **Node**
- 🔵 S **Visited Node**
- 🔴 S **Expanded Node**

Depth-First Search





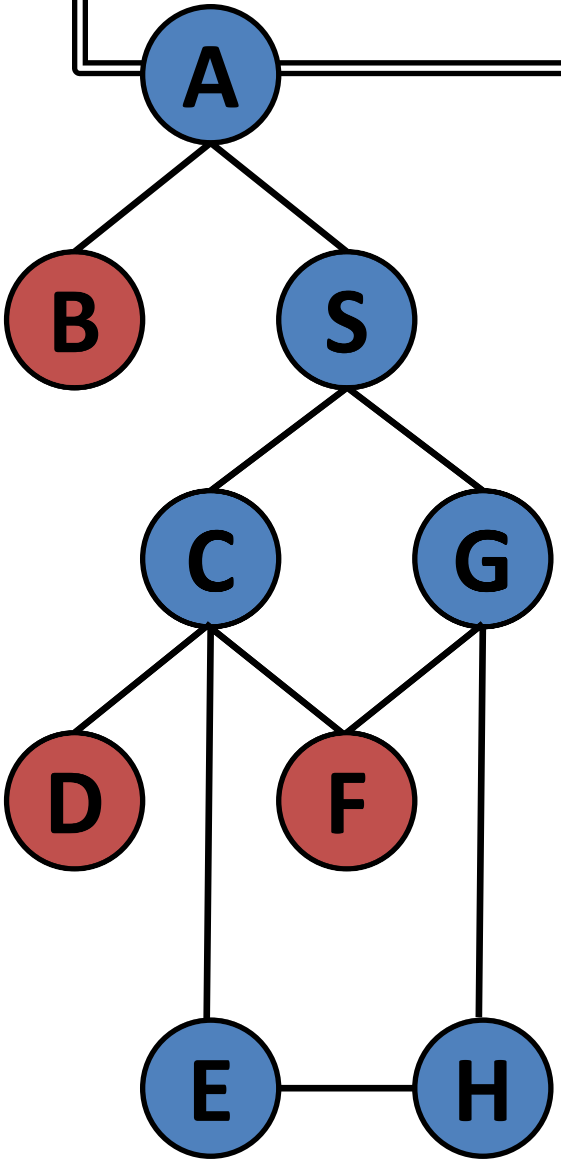
Stack (FILO)

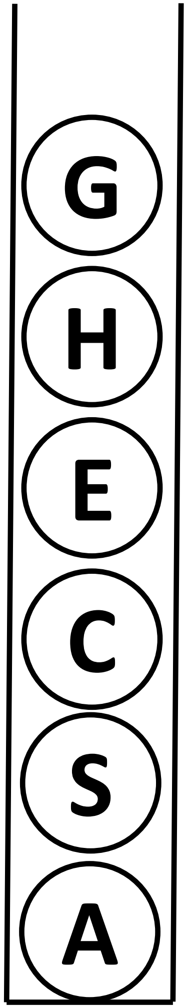
Visited Nodes:
{A,B,S,C,D,E,H,G,F}

Legend:

- ○ **Node**
- 🔵 **Visited Node**
- 🔴 **Expanded Node**

Depth-First Search





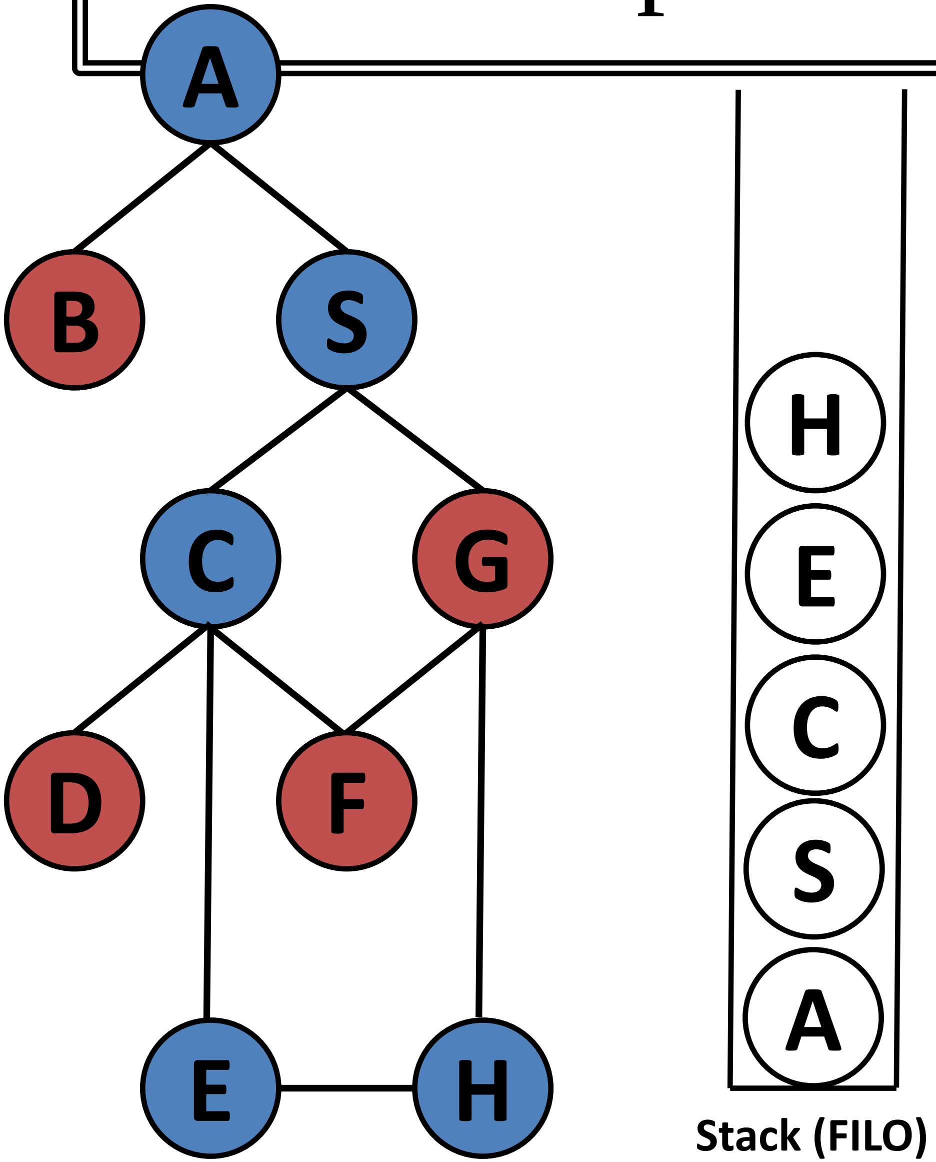
Stack (FILO)

Visited Nodes:
{A,B,S,C,D,E,H,G,F}

Legend:

- ⚪ S — **Node**
- 🔵 S — **Visited Node**
- 🔴 S — **Expanded Node**

Depth-First Search



Visited Nodes:
{A,B,S,C,D,E,H,G,F}

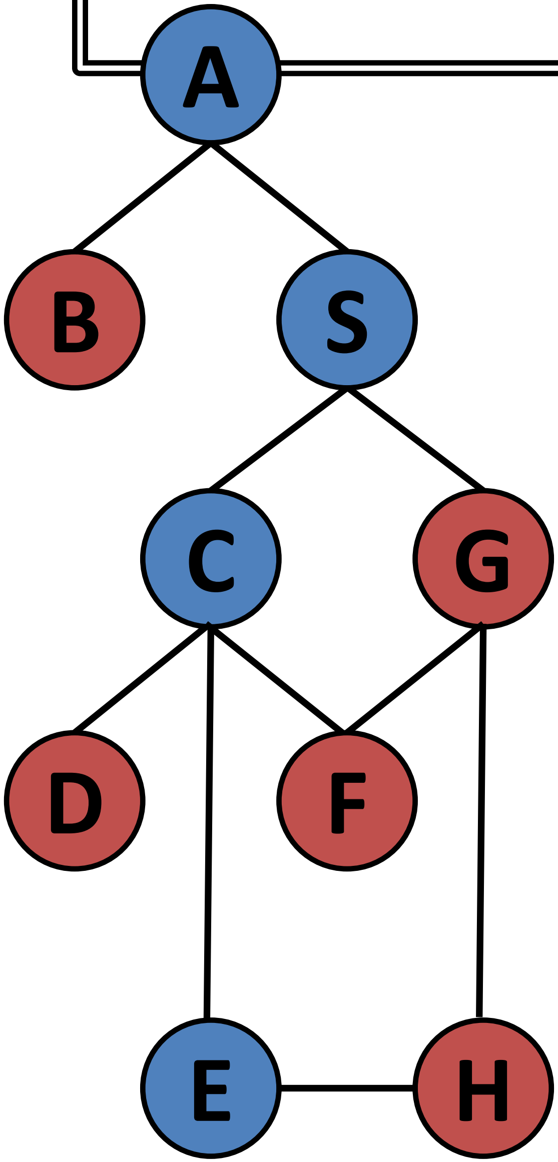
Legend:

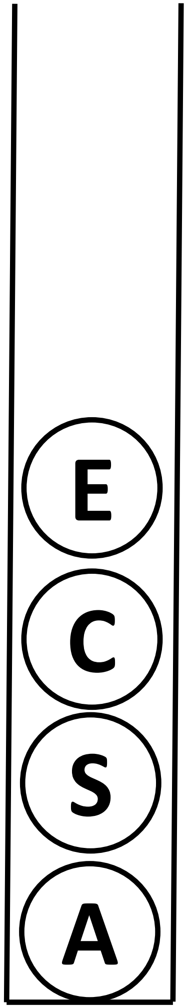
○ **S** **Node**

🔵 **S** **Visited Node**

🔴 **S** **Expanded Node**

Depth-First Search





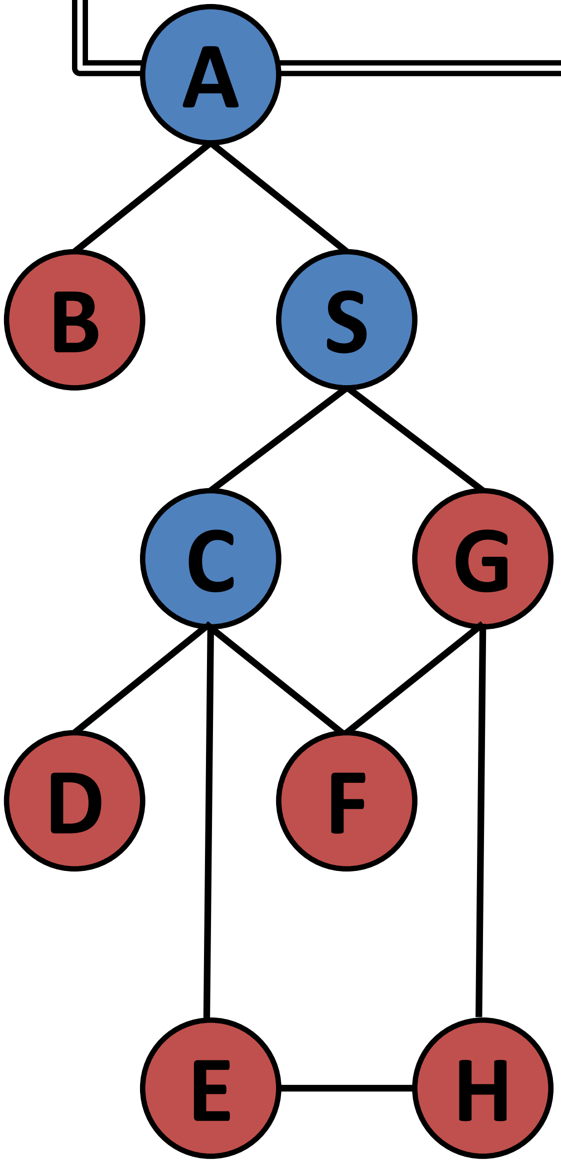
Stack (FILO)

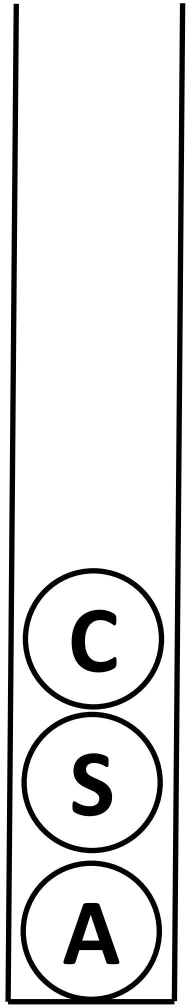
Visited Nodes:
{A,B,S,C,D,E,H,G,F}

Legend:

- ○ **Node**
- 🔵 **Visited Node**
- 🔴 **Expanded Node**

Depth-First Search





Stack (FILO)

Visited Nodes:
{A,B,S,C,D,E,H,G,F}

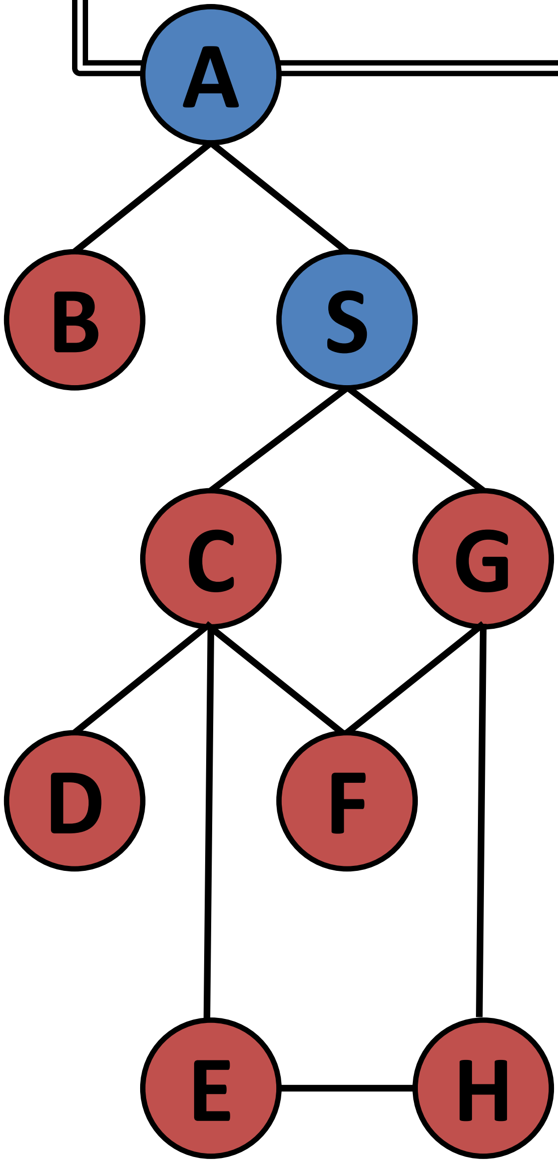
Legend:

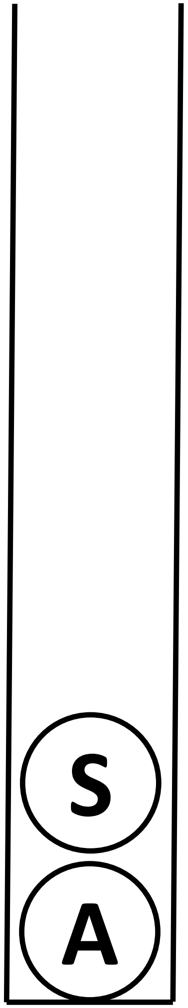
○ S **Node**

🔵 S **Visited Node**

🔴 S **Expanded Node**

Depth-First Search





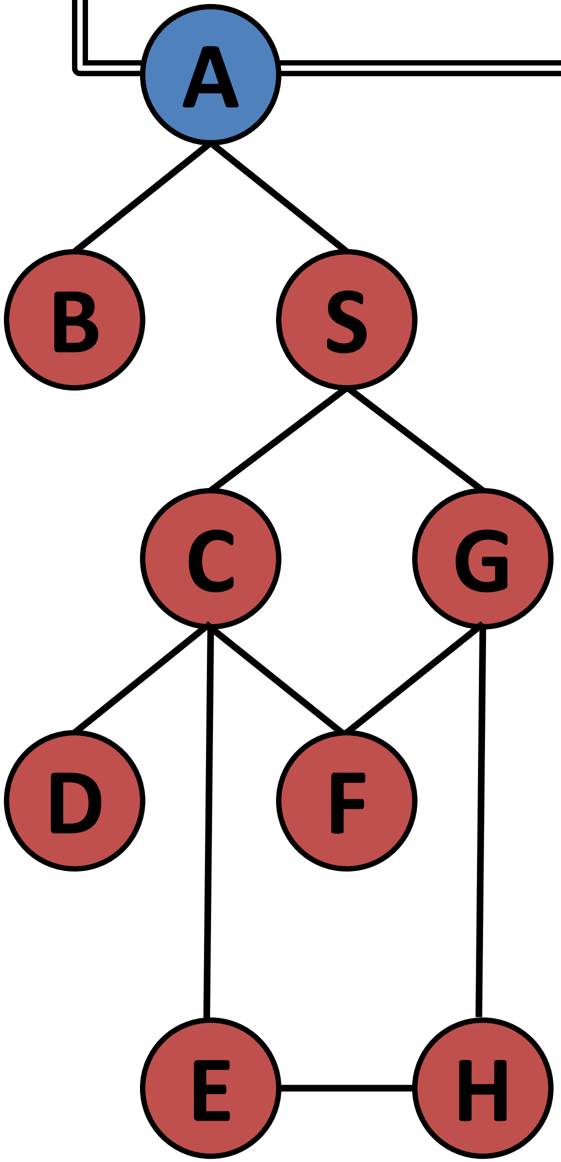
Stack (FILO)

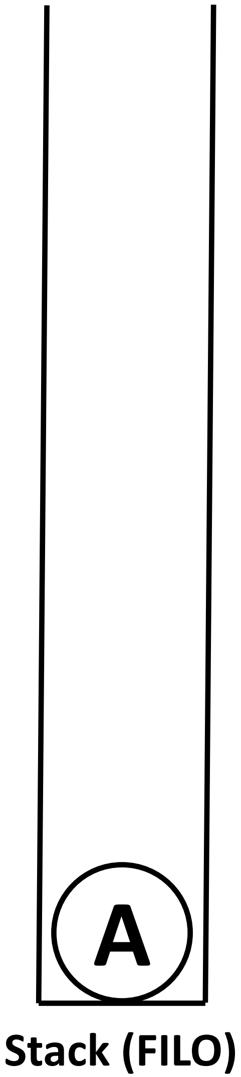
Visited Nodes:
{A,B,S,C,D,E,H,G,F}

Legend:

- ○ **Node**
- 🔵 **Visited Node**
- 🔴 **Expanded Node**

Depth-First Search





Visited Nodes:
{A,B,S,C,D,E,H,G,F}

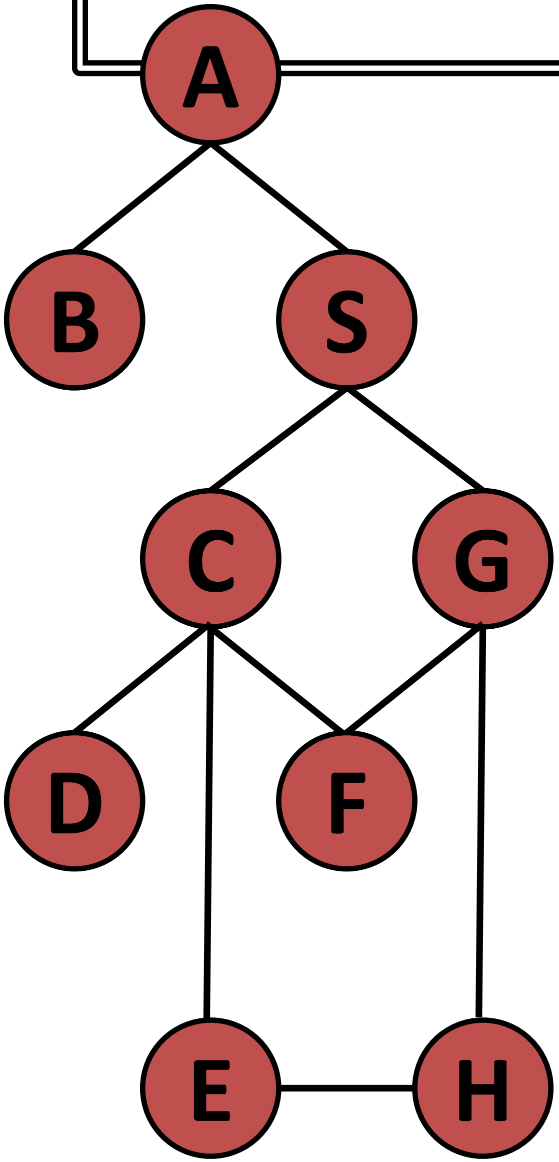
Legend:

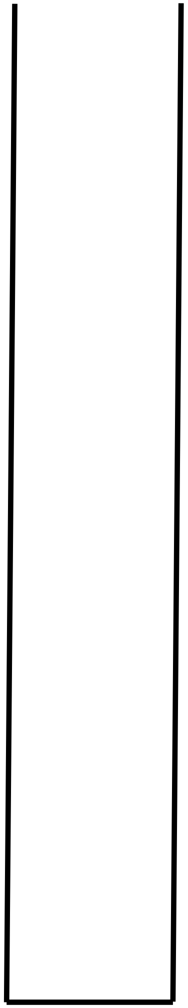
S **Node**

S **Visited Node**

S **Expanded Node**

Depth-First Search





Stack (FILO)

Visited Nodes:
{A,B,S,C,D,E,H,G,F}

Legend:

○ **Node**

🔵 **Visited Node**

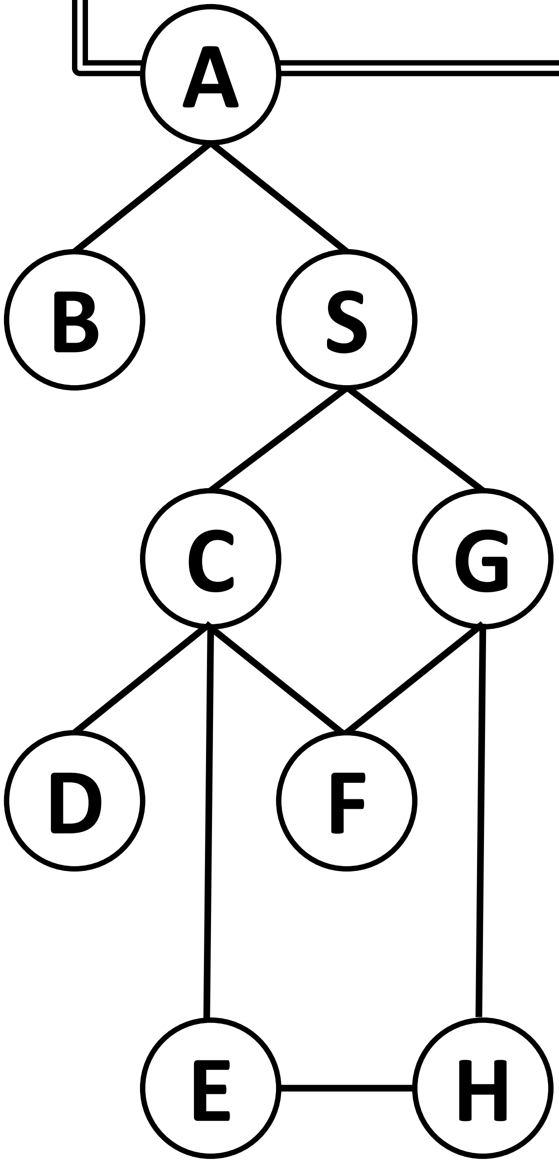
🔴 **Expanded Node**

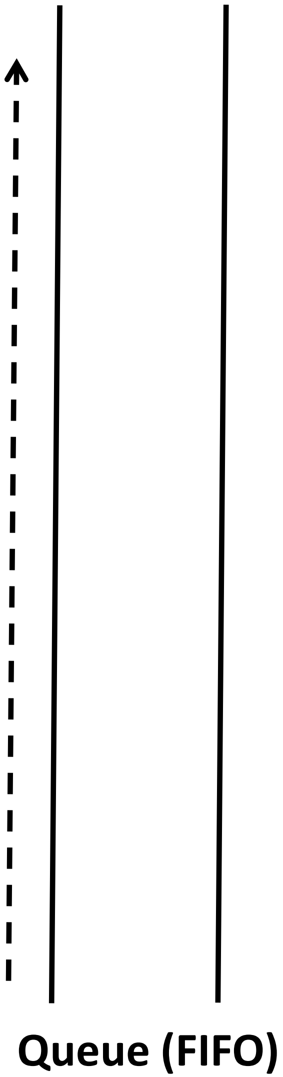
Depth-First Search (DFS)

- ***Complete?***
  - ***No***: fails in infinite-depth spaces, spaces with loops
    Modify to avoid repeated states along path
    ⇒ *complete in finite spaces*
- ***Optimal?***
  - ***No***
- ***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!*

** A **dense graph** is a graph in which the number of edges is close to the maximal number of edges.*

Breadth-First Search





Visited Nodes:
{}

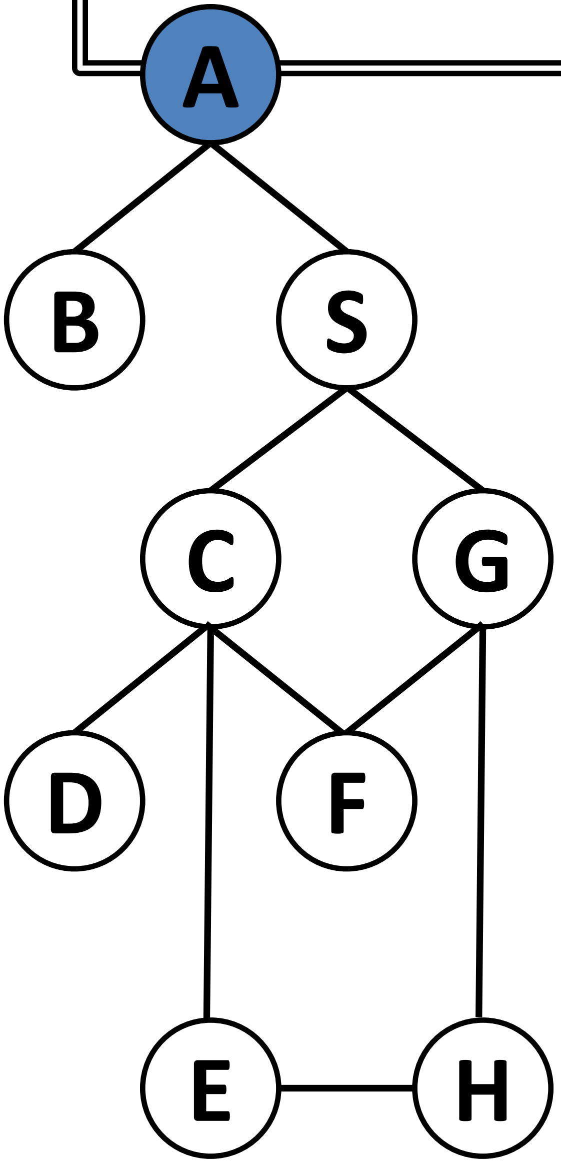
Legend:

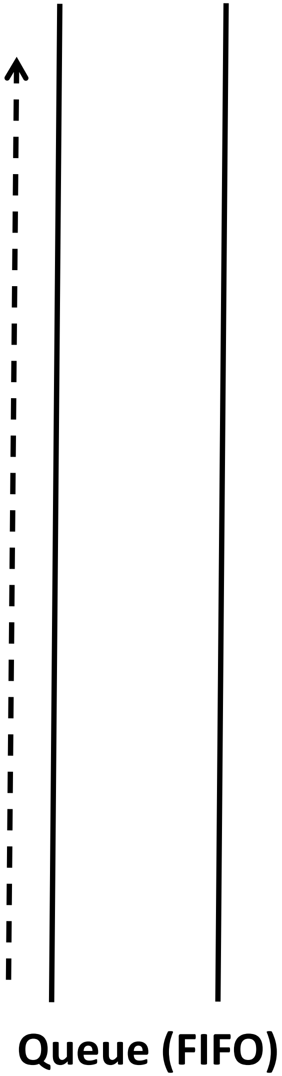
○ **Node**

🔵 **Visited Node**

🔴 **Expanded Node**

Breadth-First Search



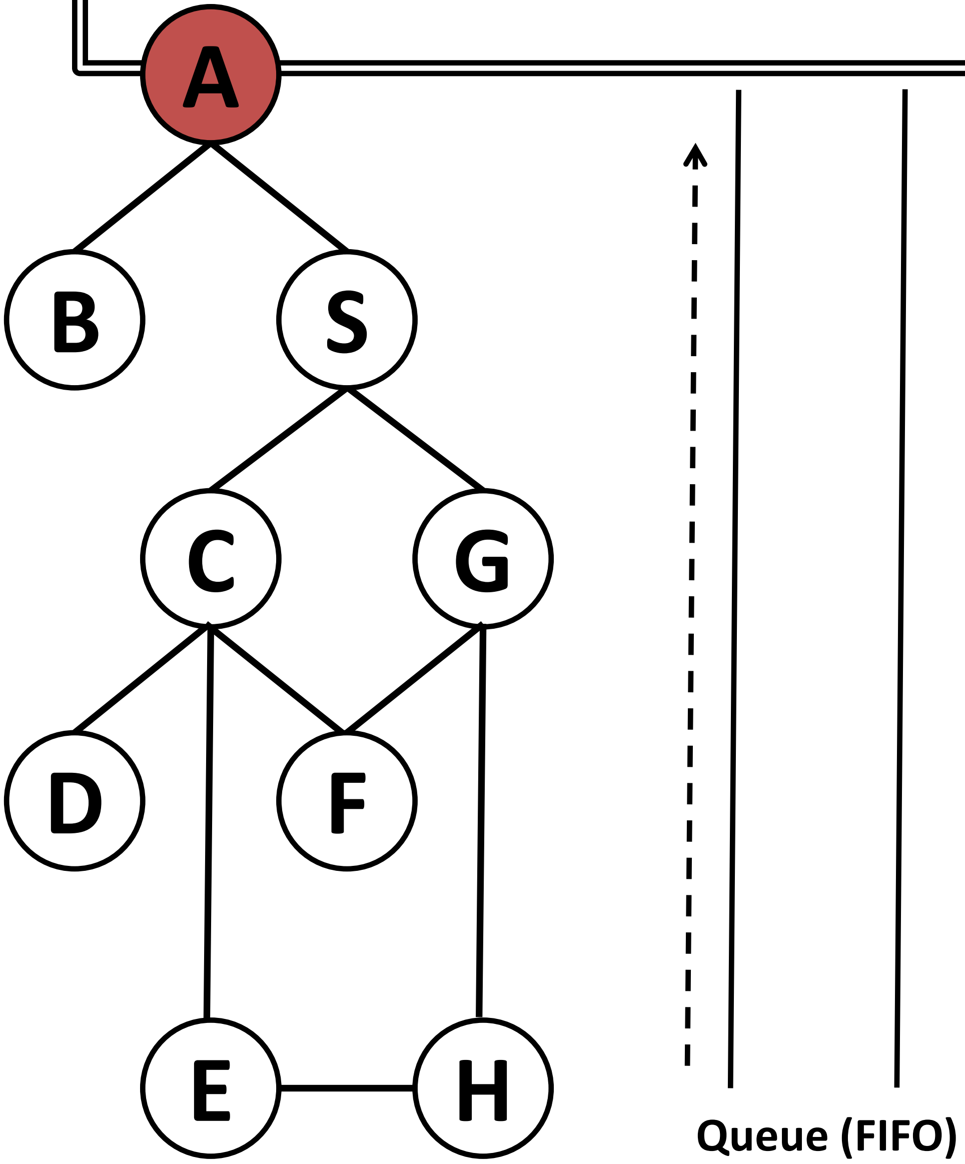


Visited Nodes:
{A}

Legend:

- S — **Node**
- S — **Visited Node**
- S — **Expanded Node**

Breadth-First Search

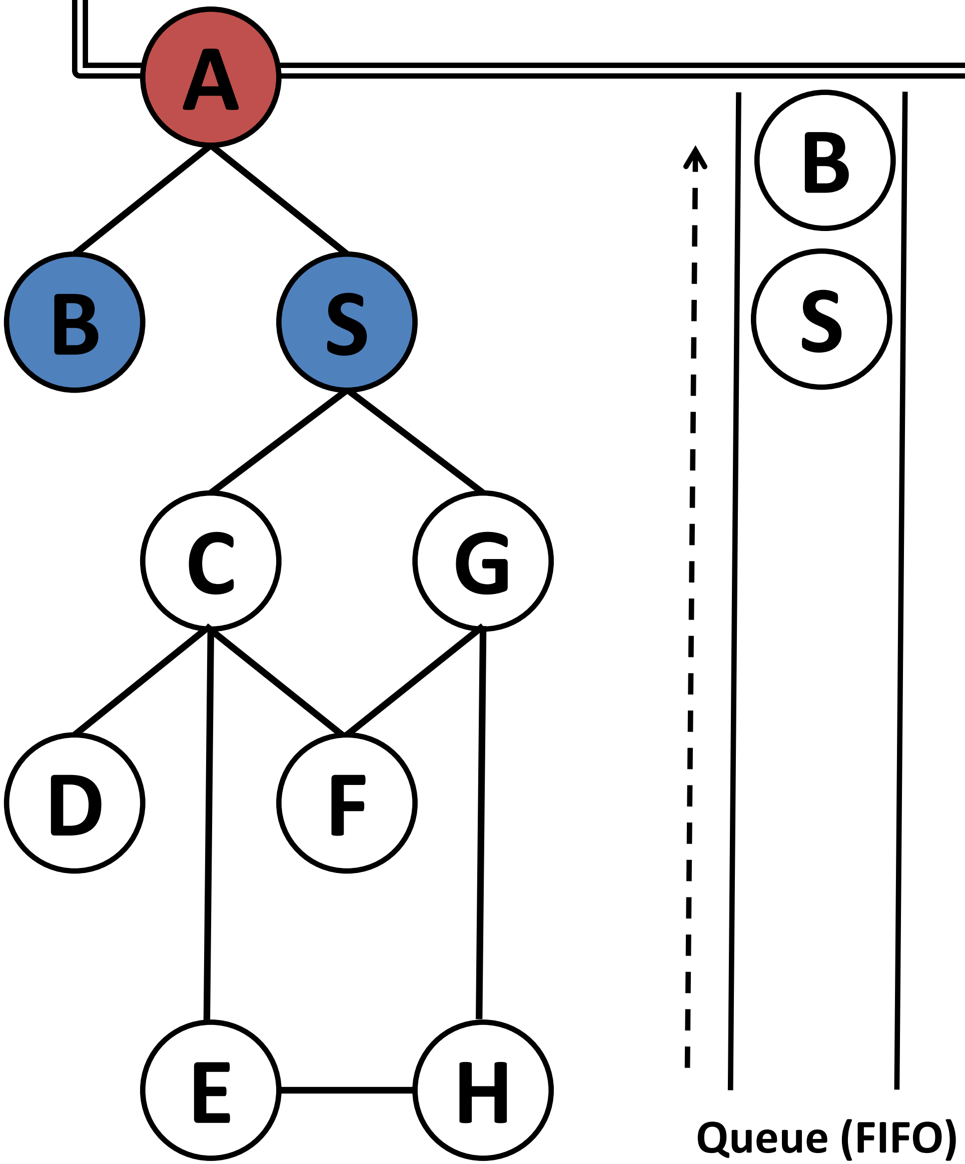


Visited Nodes:
{A}

Legend:

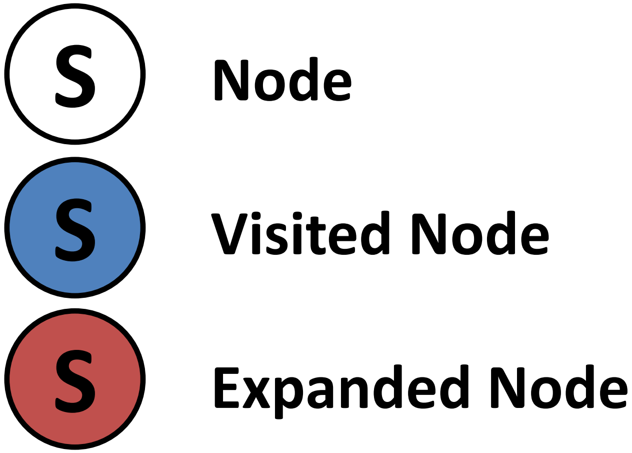
- **S** Node
- **S** Visited Node
- **S** Expanded Node

Breadth-First Search

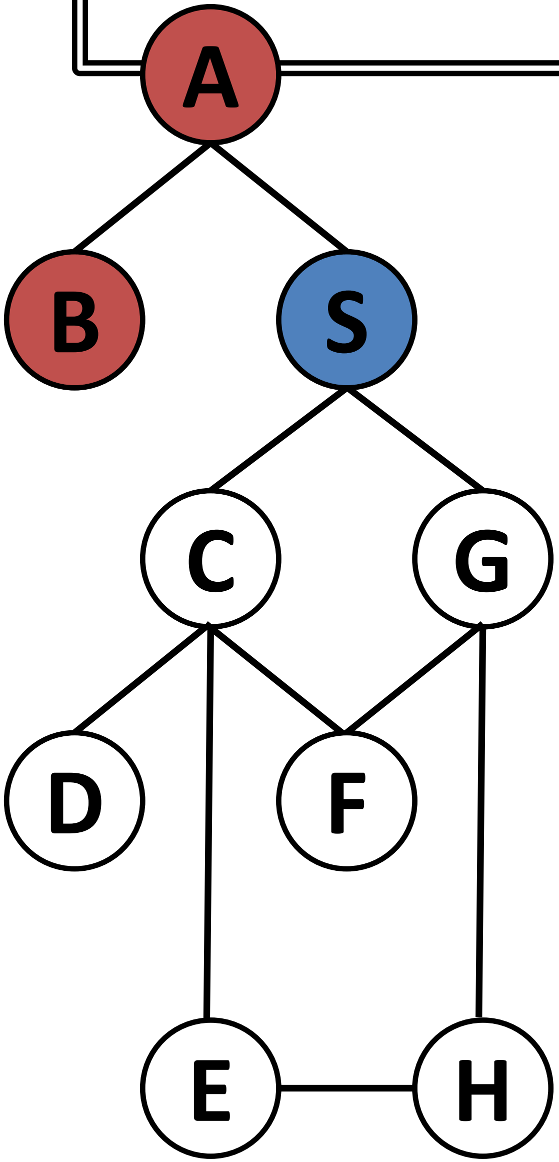


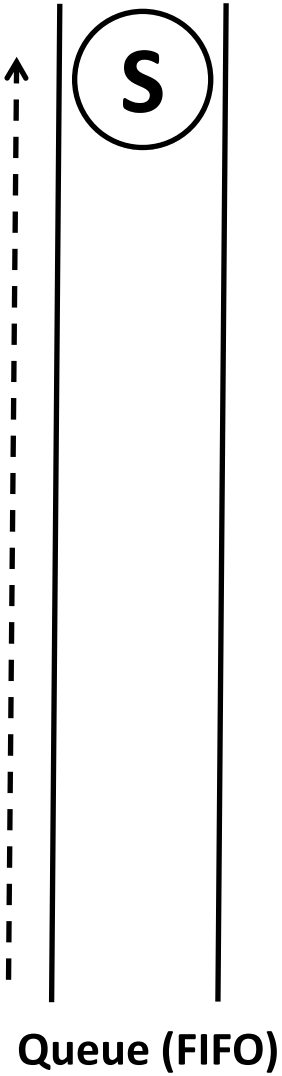
Visited Nodes:
{A,B,S}

Legend:



Breadth-First Search





Visited Nodes:
{A,B,S}

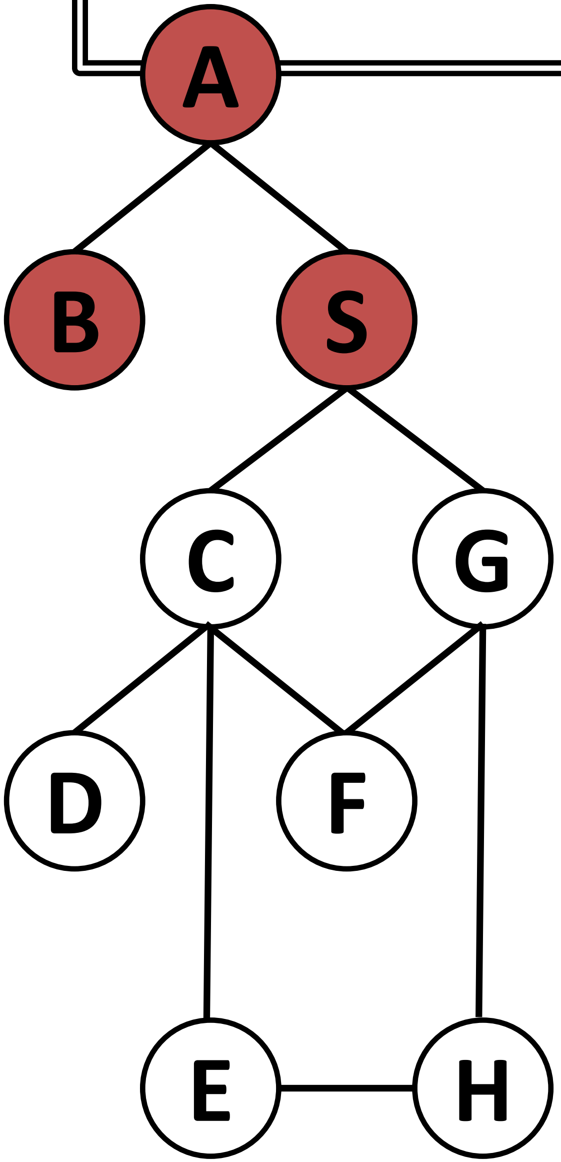
Legend:

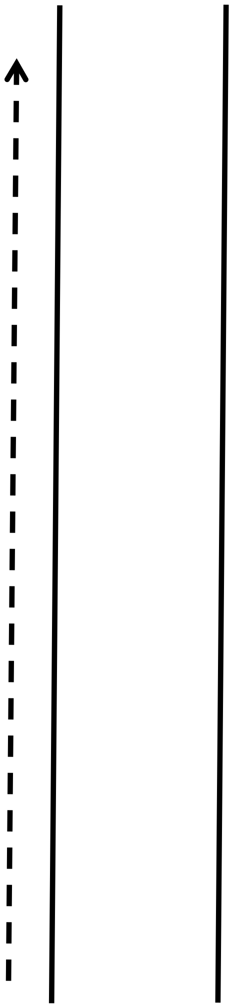
⚪ S **Node**

🔵 S **Visited Node**

🔴 S **Expanded Node**

Breadth-First Search





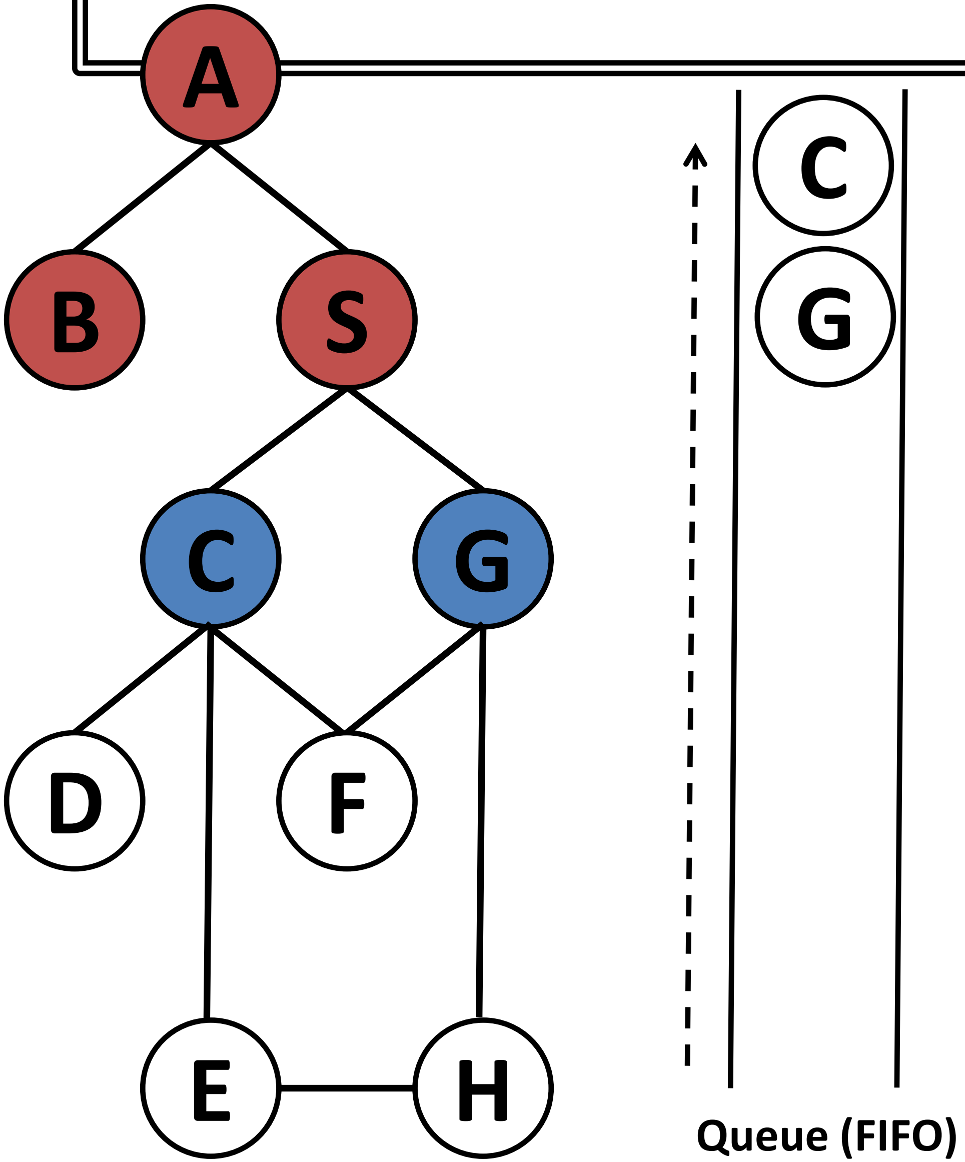
Queue (FIFO)

Visited Nodes:
{A,B,S}

Legend:

- S **Node**
- S **Visited Node**
- S **Expanded Node**

Breadth-First Search



Visited Nodes:
{A,B,S,C,G}

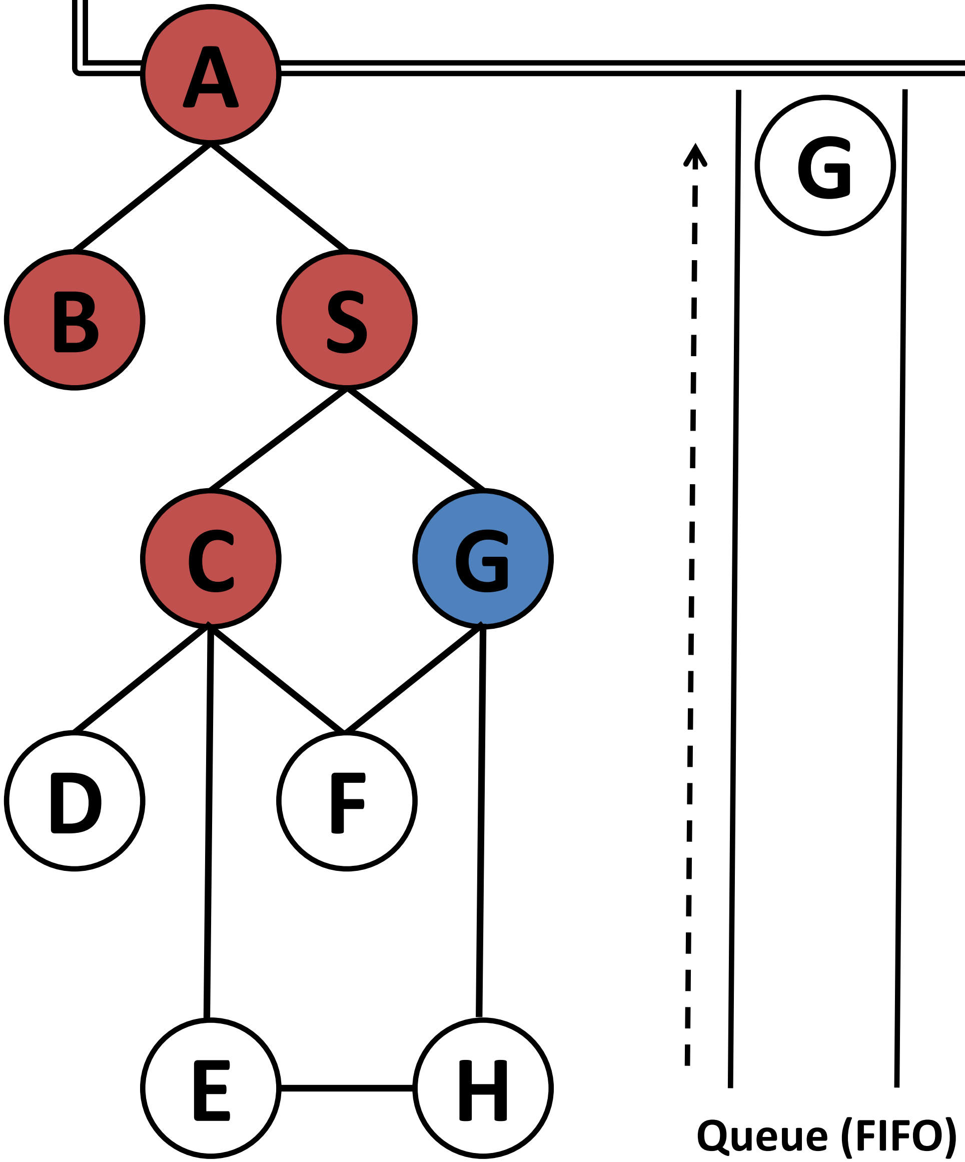
Legend:

S **Node**

S **Visited Node**

S **Expanded Node**

Breadth-First Search



Visited Nodes:
{A,B,S,C,G}

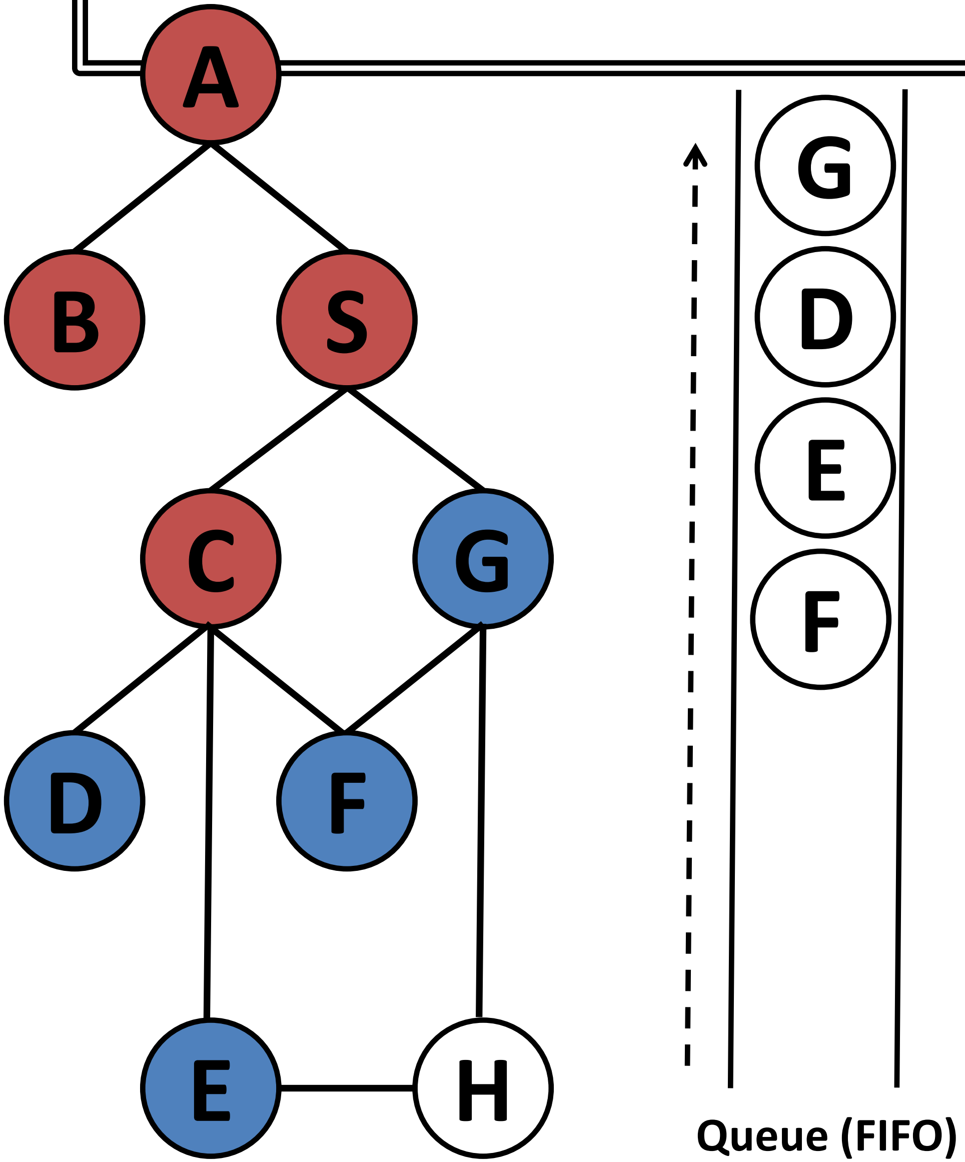
Legend:

S **Node**

S **Visited Node**

S **Expanded Node**

Breadth-First Search



Visited Nodes:
{A,B,S,C,G,D,E,F}

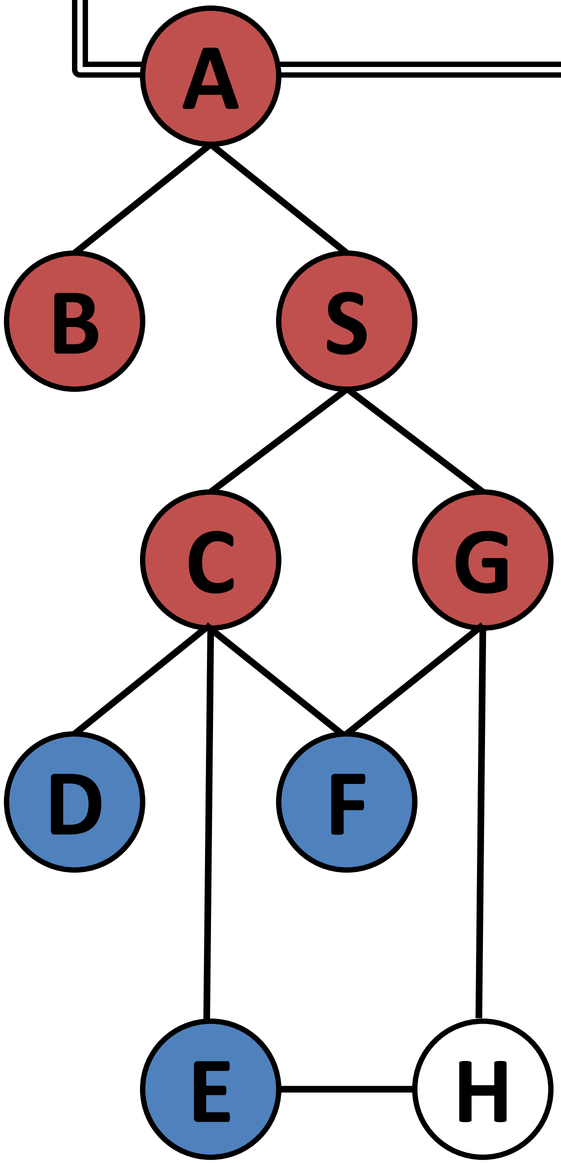
Legend:

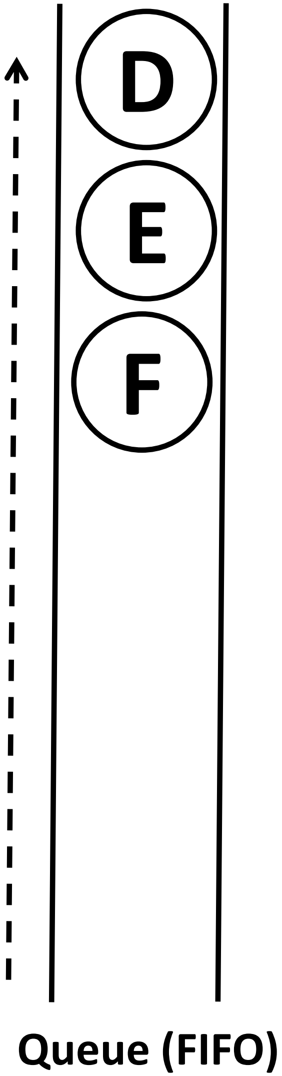
○ S Node

🔵 S Visited Node

🔴 S Expanded Node

Breadth-First Search





Visited Nodes:
{A,B,S,C,G,D,E,F}

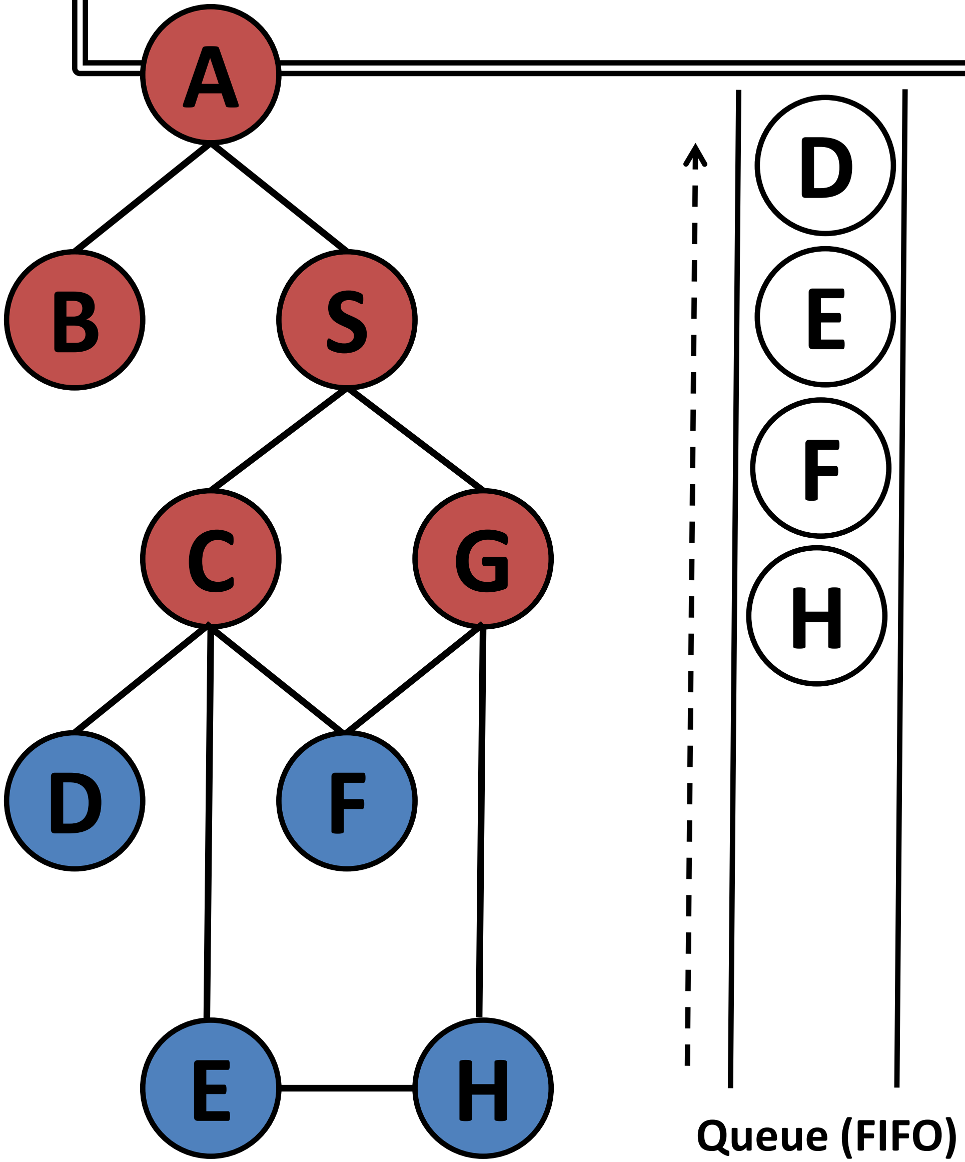
Legend:

S **Node**

S **Visited Node**

S **Expanded Node**

Breadth-First Search

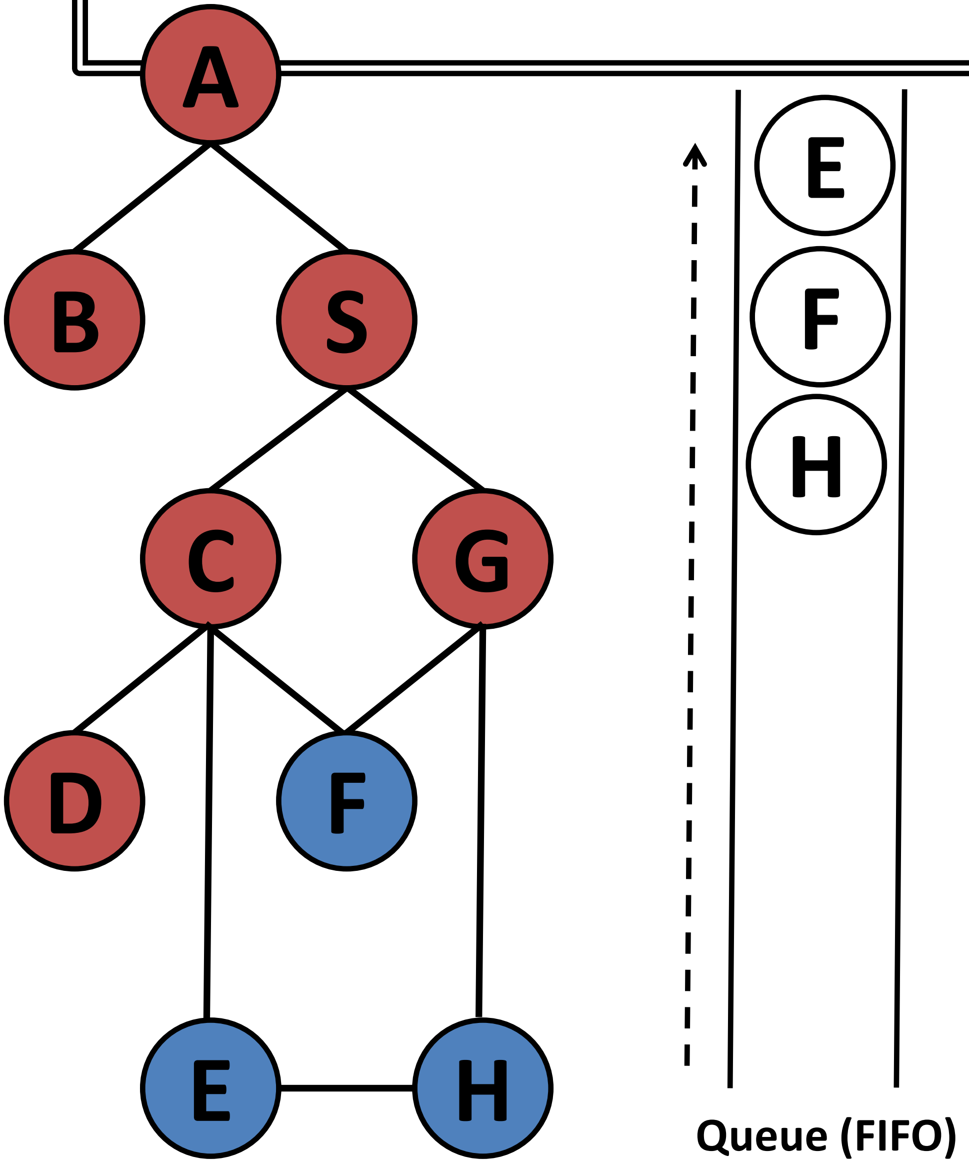


Visited Nodes:
{A,B,S,C,G,D,E,F,H}

Legend:

- ○ **Node**
- 🔵 **Visited Node**
- 🔴 **Expanded Node**

Breadth-First Search



Queue (FIFO)

Visited Nodes:
{A,B,S,C,G,D,E,F,H}

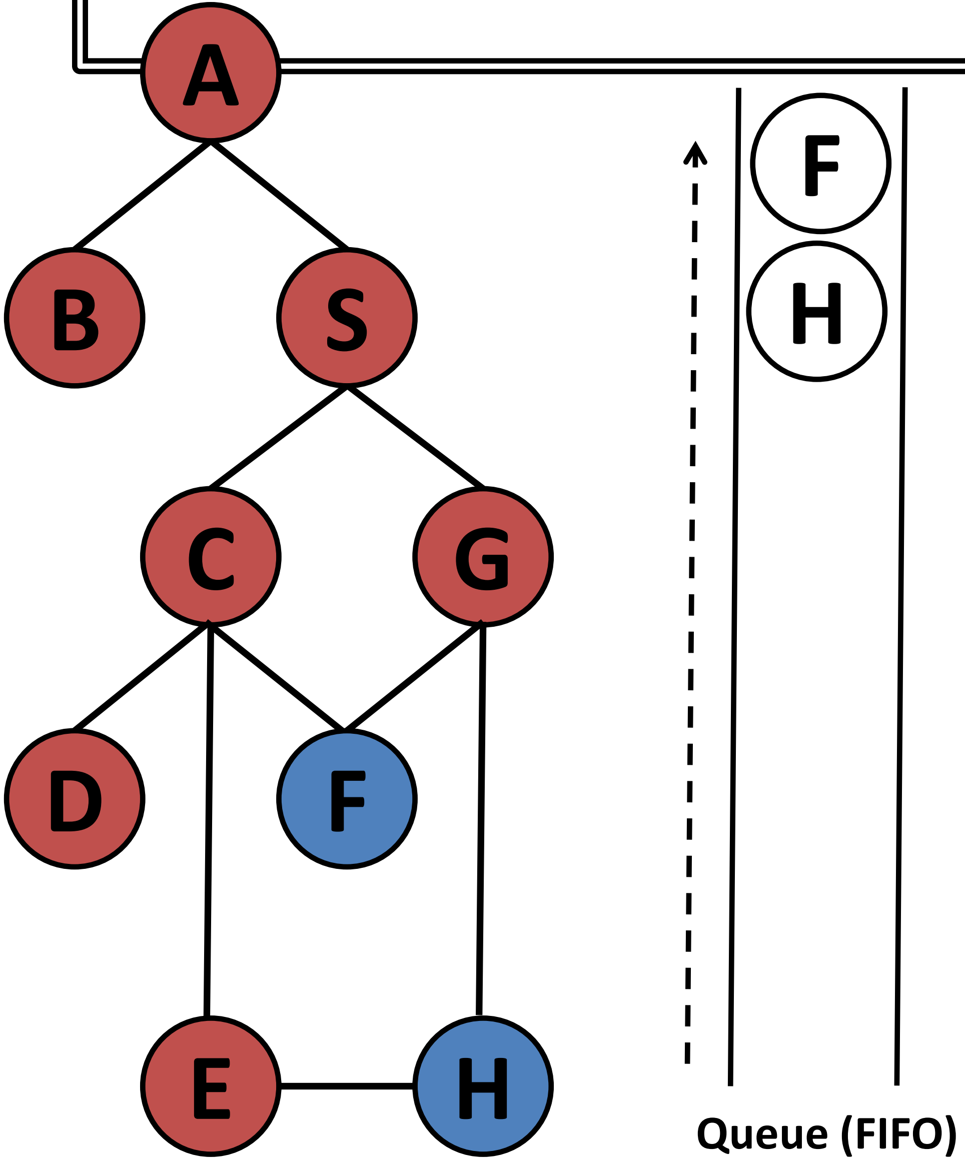
Legend:

S **Node**

S **Visited Node**

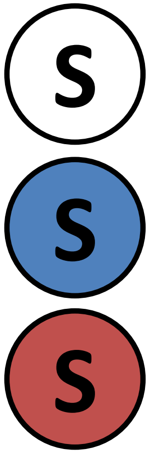
S **Expanded Node**

Breadth-First Search



Visited Nodes:
{A,B,S,C,G,D,E,F,H}

Legend:

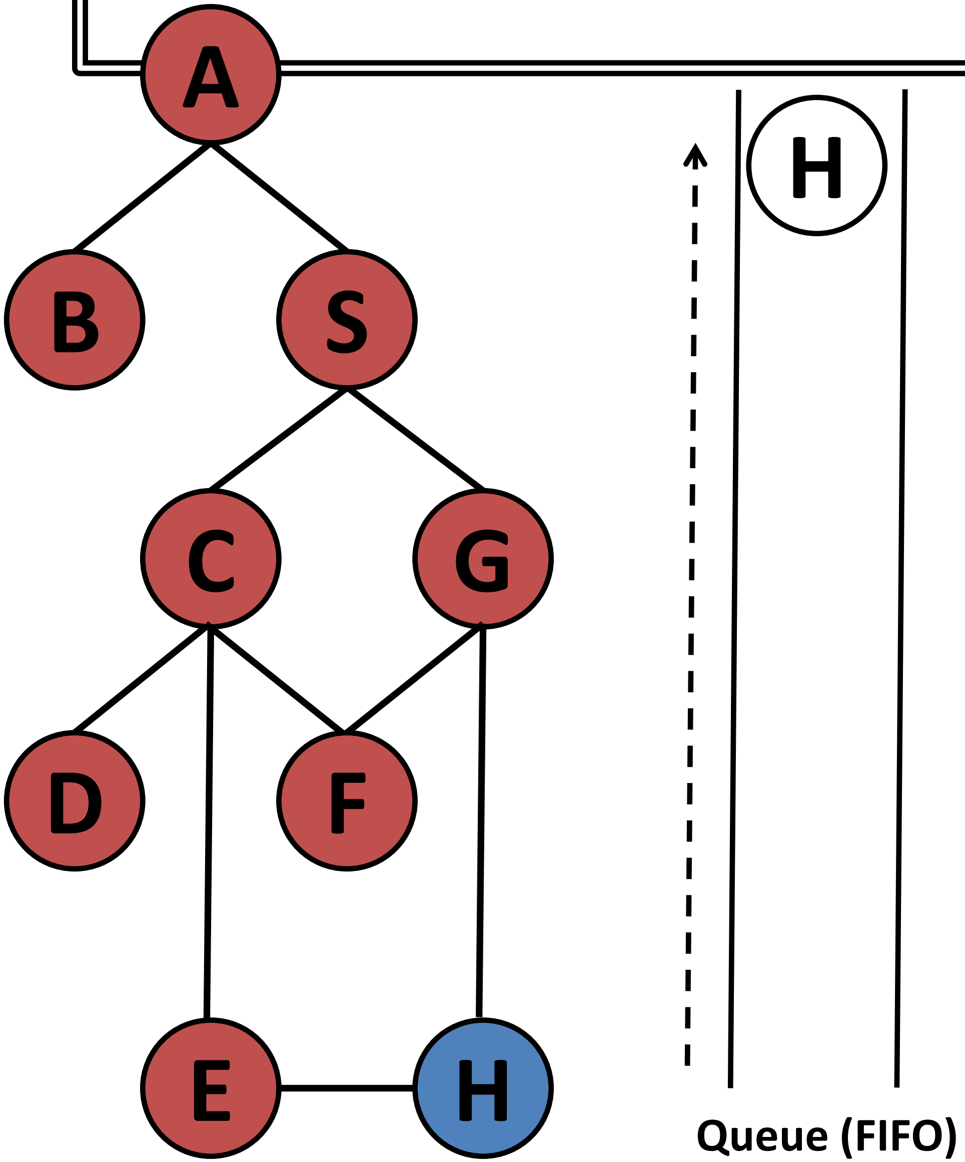


Node

Visited Node

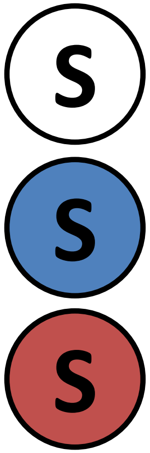
Expanded Node

Breadth-First Search



Visited Nodes:
{A,B,S,C,G,D,E,F,H}

Legend:

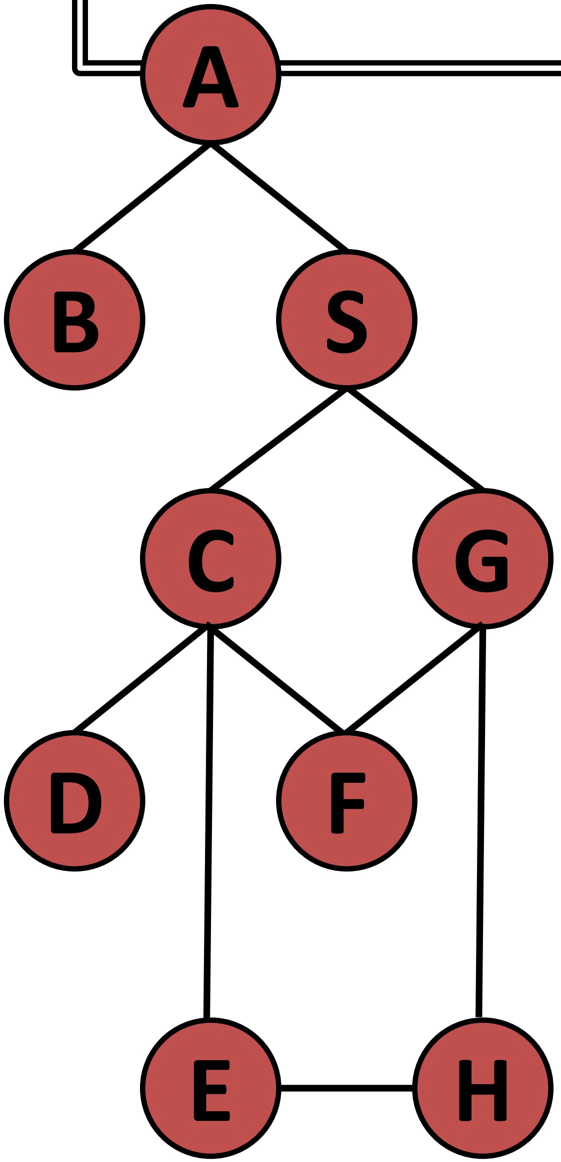


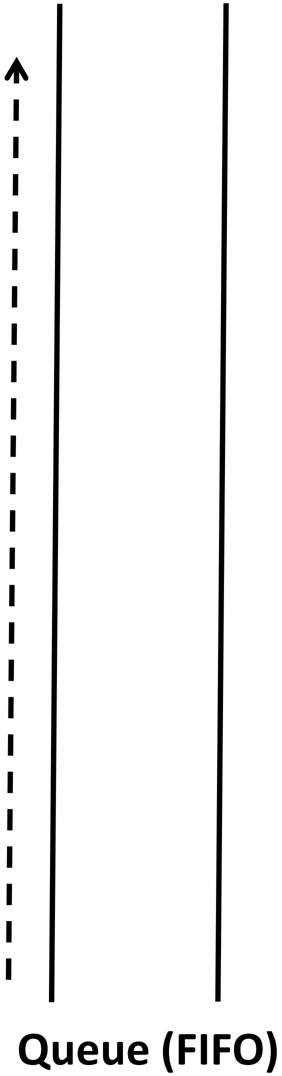
Node

Visited Node

Expanded Node

Breadth-First Search





Visited Nodes:
{A,B,S,C,G,D,E,F,H}

Legend:

○ S **Node**

🟦 S **Visited Node**

🟥 S **Expanded Node**

Breadth-First Search

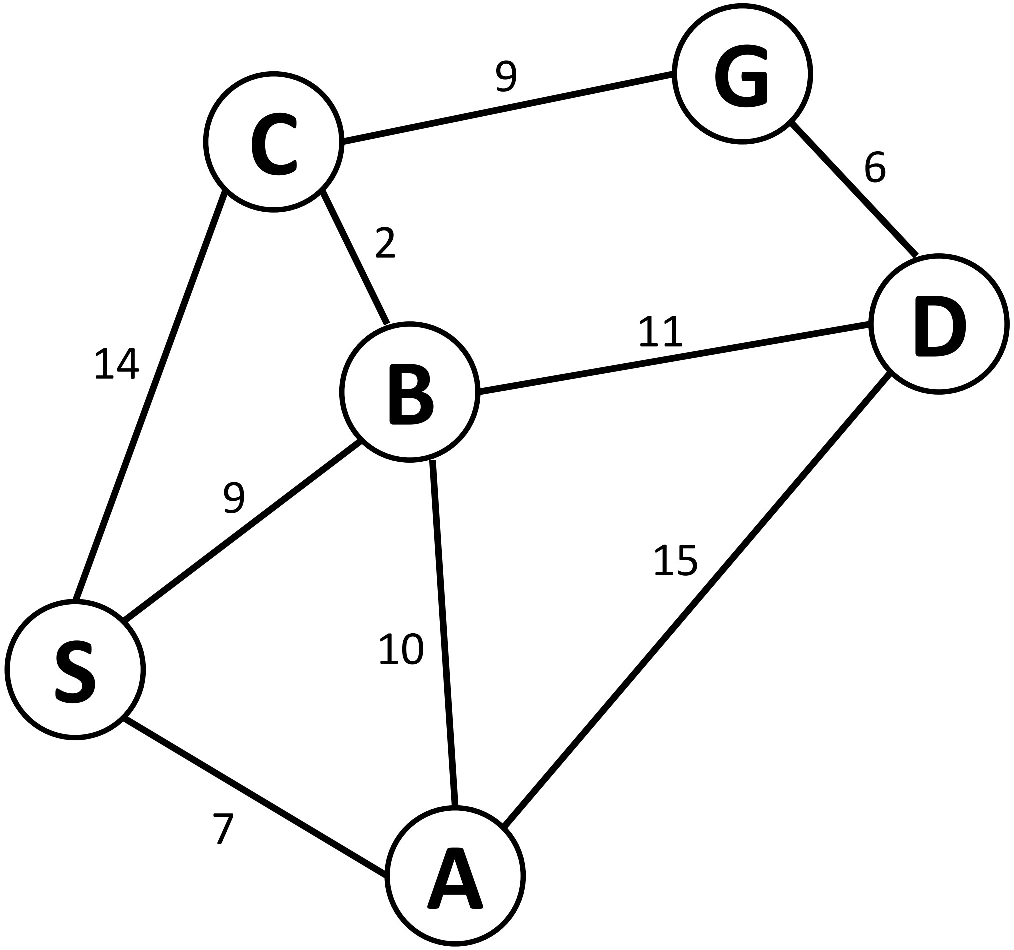
```
Node {
    Node parent
    T value
    List<Node> children
}
```

Breadth-First Search (BFS)

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

** **Space** is the big problem; can easily generate nodes at 100MB/sec so 24hrs = 8640GB.*

Uniform-Cost Search



Visited Nodes:
{}

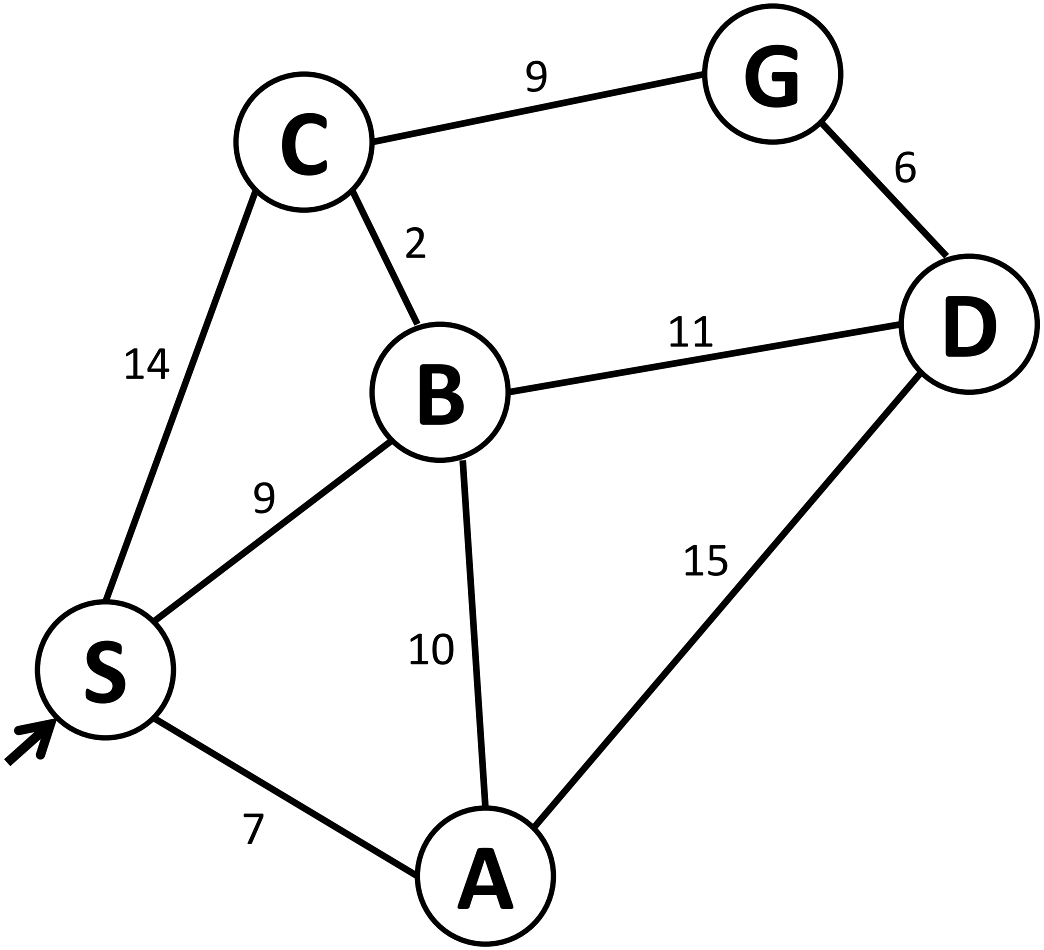
Legend:

S **Node**

S **Visited Node**

S **Expanded Node**

Uniform-Cost Search



Visited Nodes:
{}

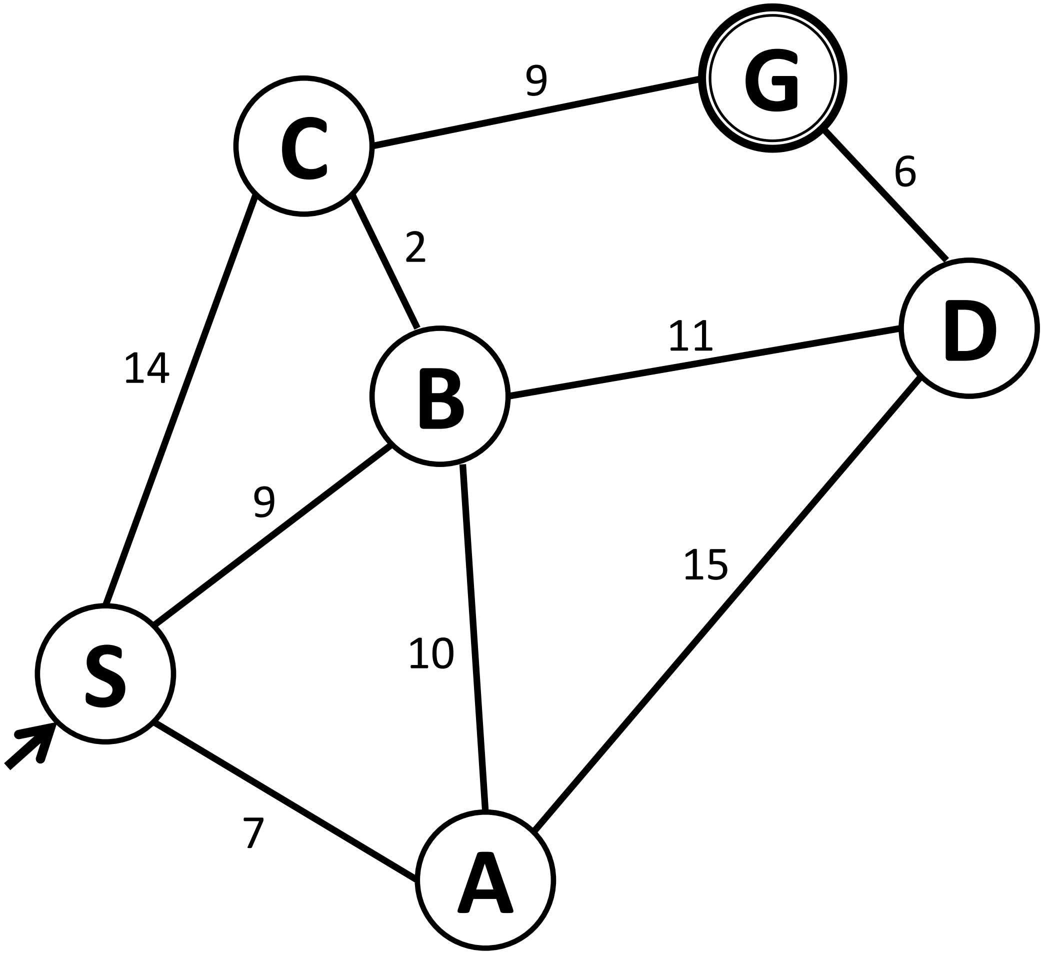
Legend:

○ **Node**

🔵 **Visited Node**

🔴 **Expanded Node**

Uniform-Cost Search



Visited Nodes:
{}

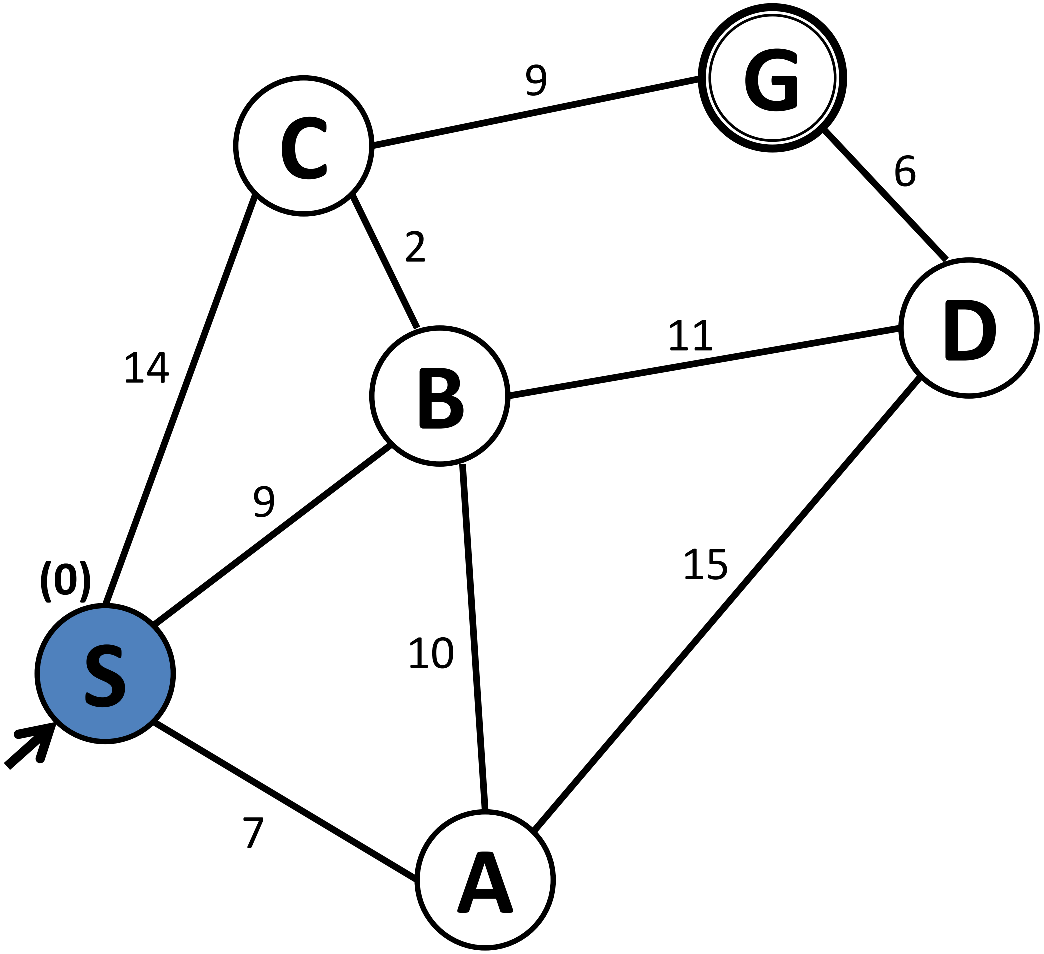
Legend:

S **Node**

S **Visited Node**

S **Expanded Node**

Uniform-Cost Search



Visited Nodes:
{S}

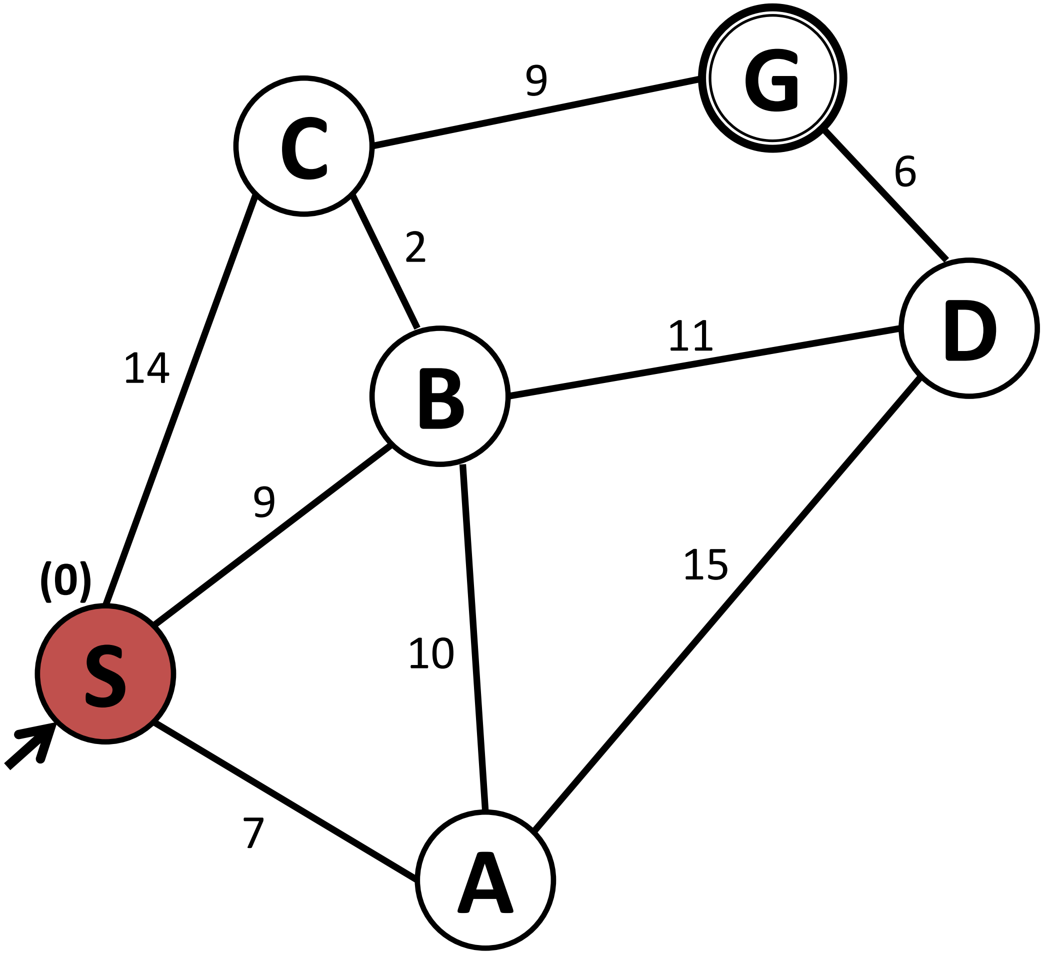
Legend:

○ **S** **Node**

🔵 **S** **Visited Node**

🔴 **S** **Expanded Node**

Uniform-Cost Search



Visited Nodes:
{S}

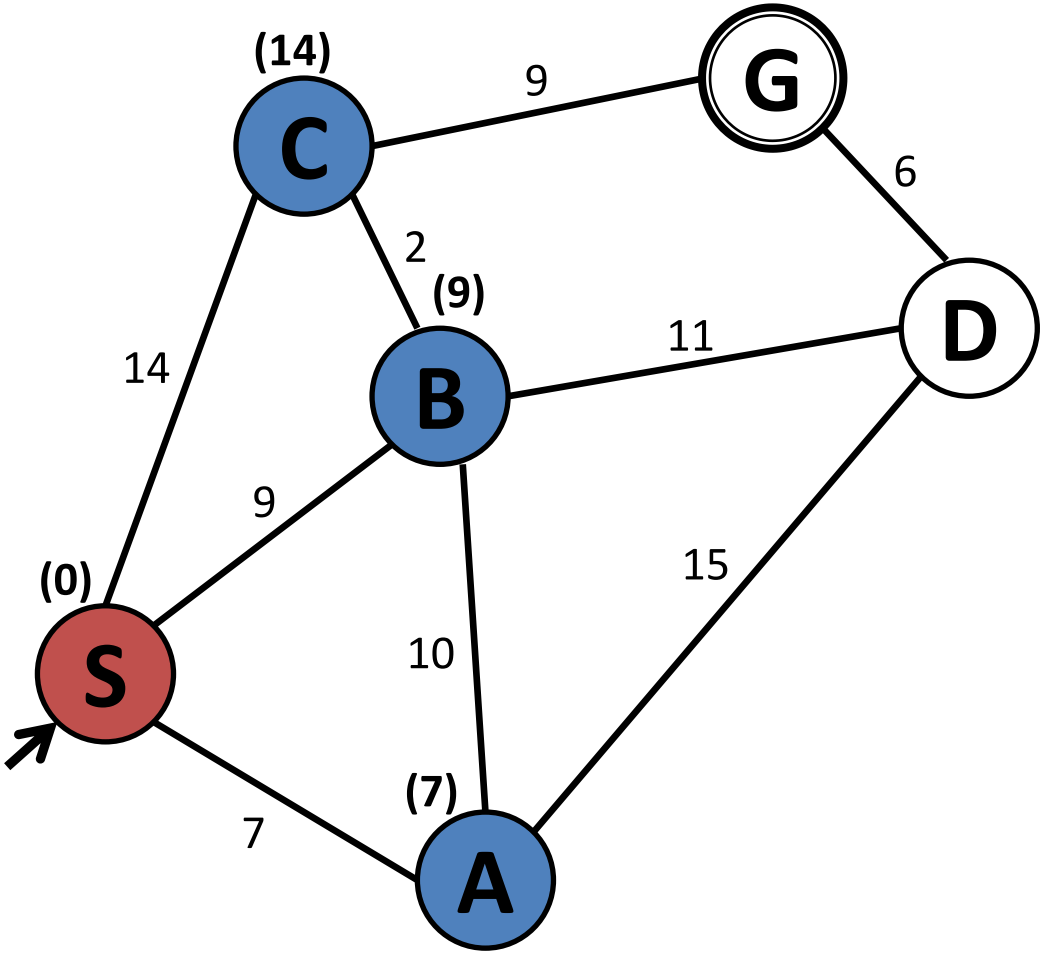
Legend:

S **Node**

S **Visited Node**

S **Expanded Node**

Uniform-Cost Search

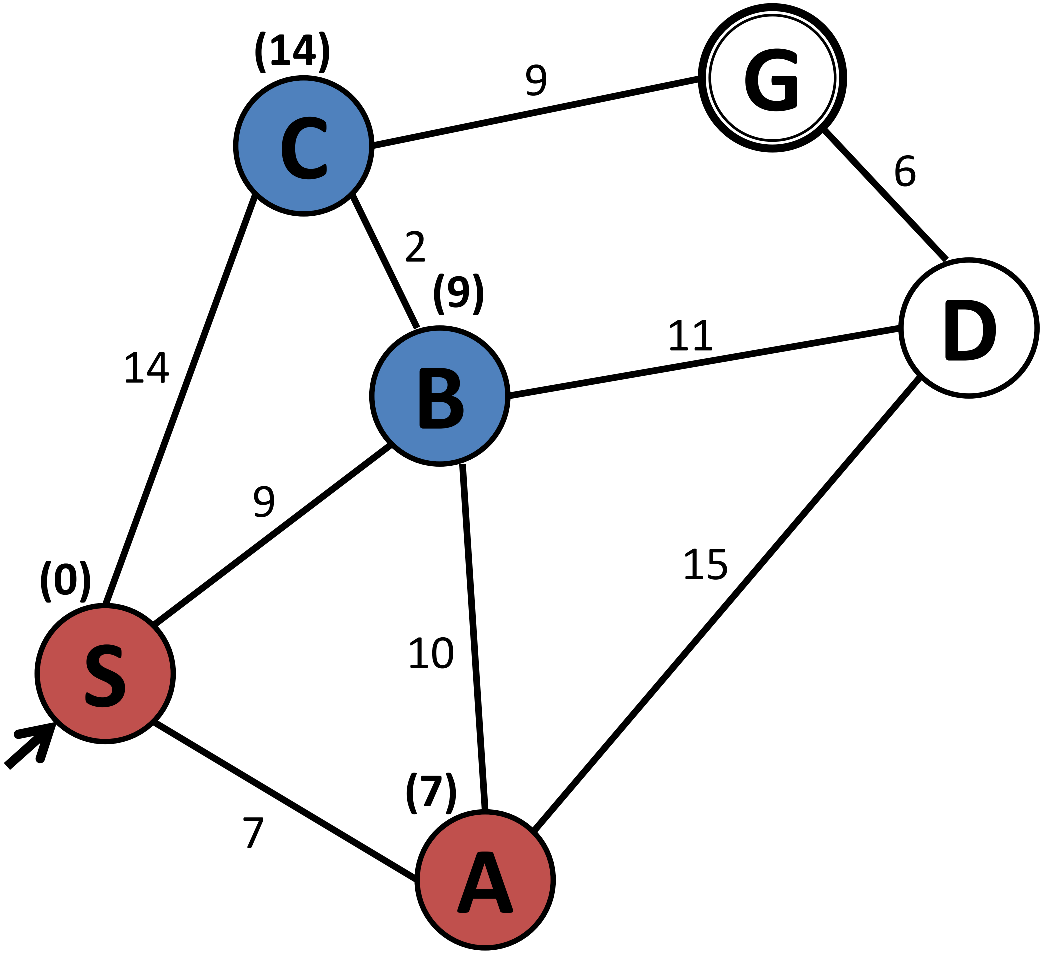


Visited Nodes:
{S,A,B,C}

Legend:

S	**Node**
S	**Visited Node**
S	**Expanded Node**

Uniform-Cost Search



Visited Nodes:
{S,A,B,C}

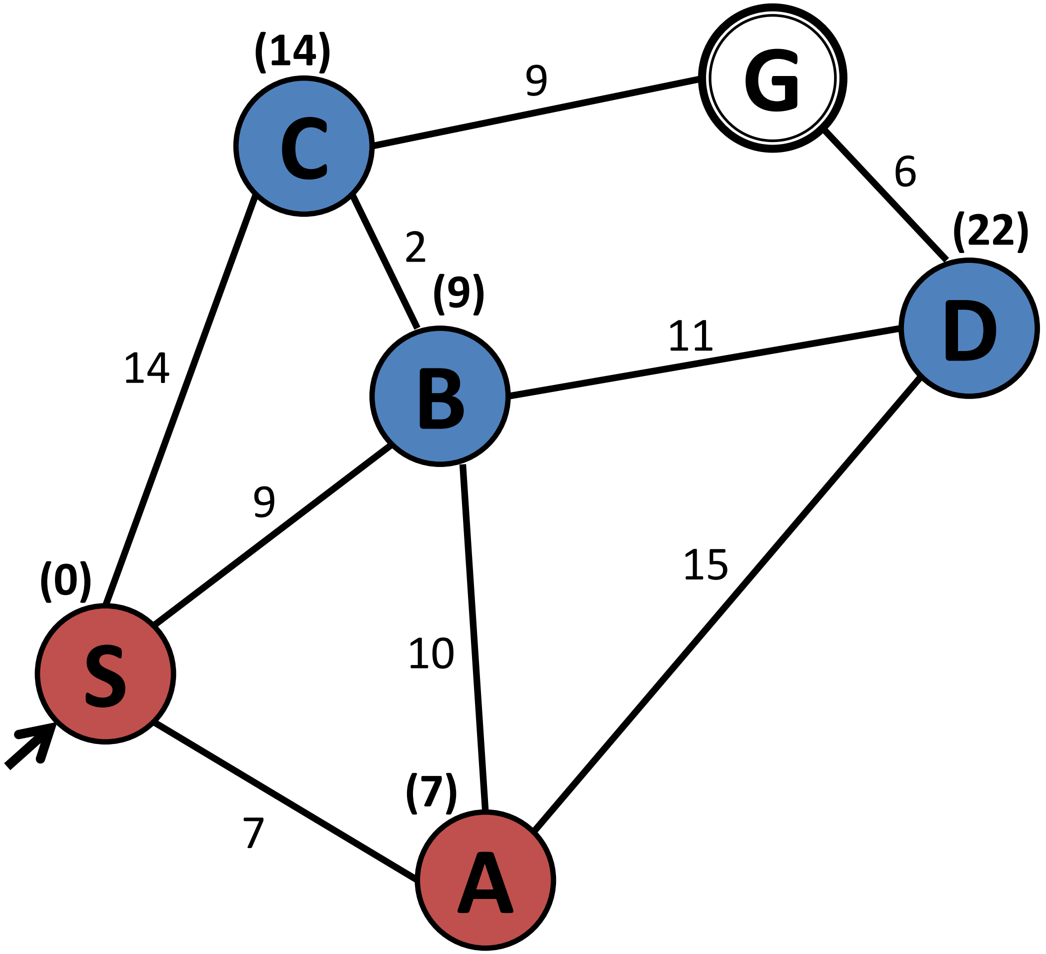
Legend:

S **Node**

S **Visited Node**

S **Expanded Node**

Uniform-Cost Search

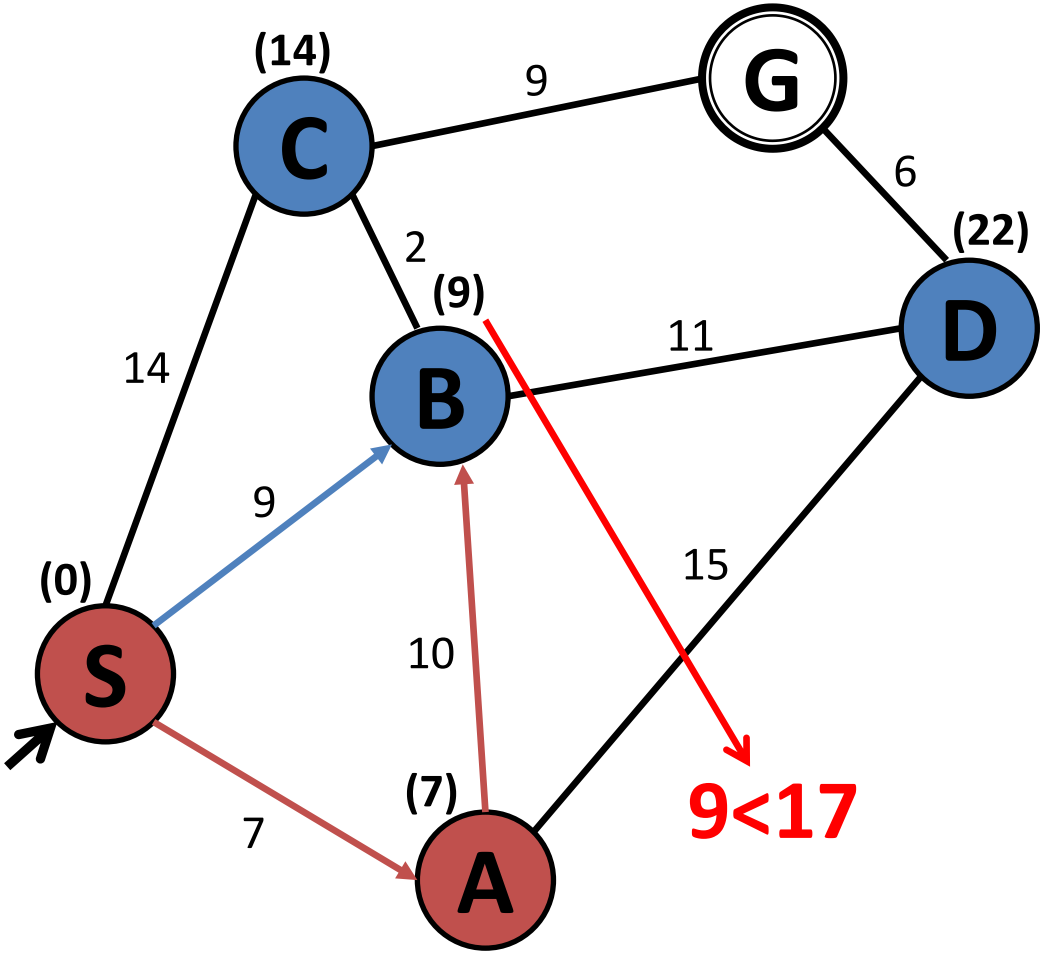


Visited Nodes:
{S,A,B,C,D}

Legend:

- ○ S **Node**
- 🔵 S **Visited Node**
- 🔴 S **Expanded Node**

Uniform-Cost Search



Visited Nodes:
{S,A,B,C,D}

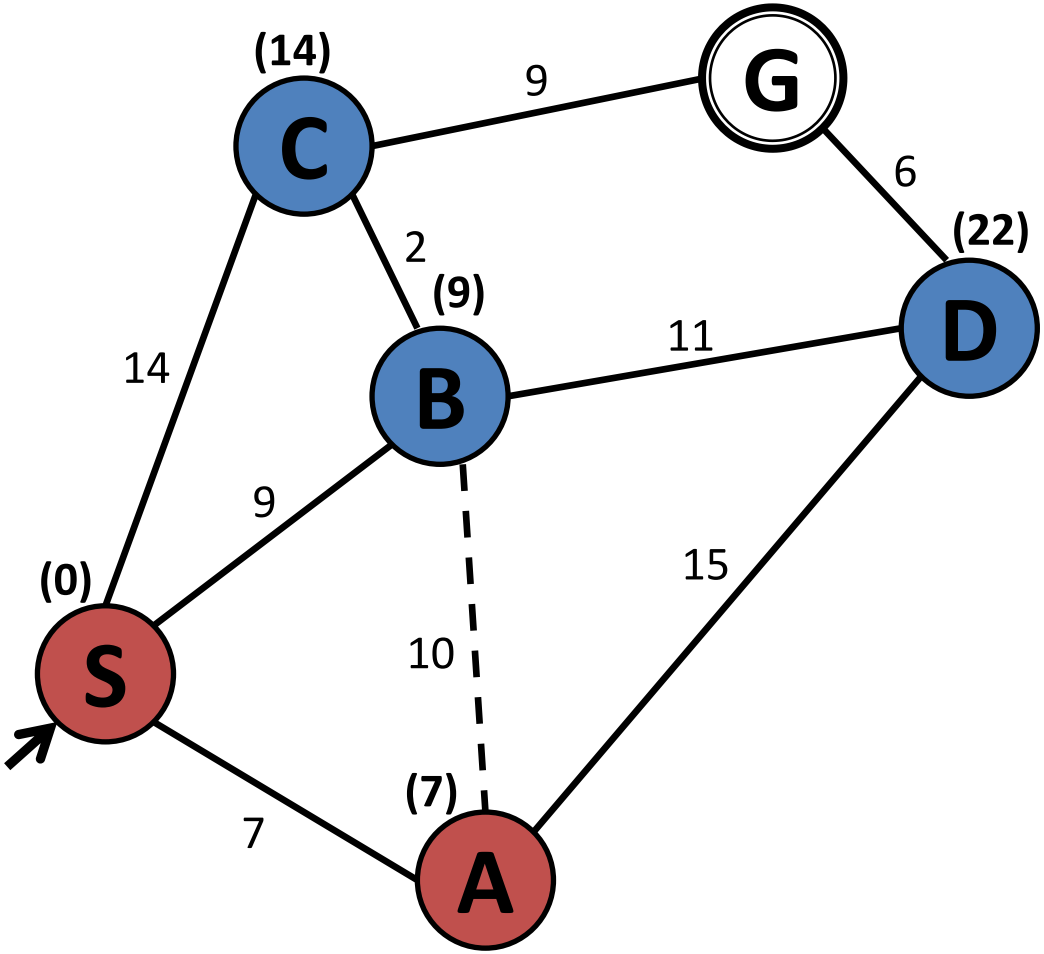
Legend:

S **Node**

S **Visited Node**

S **Expanded Node**

Uniform-Cost Search



Visited Nodes:
{S,A,B,C,D}

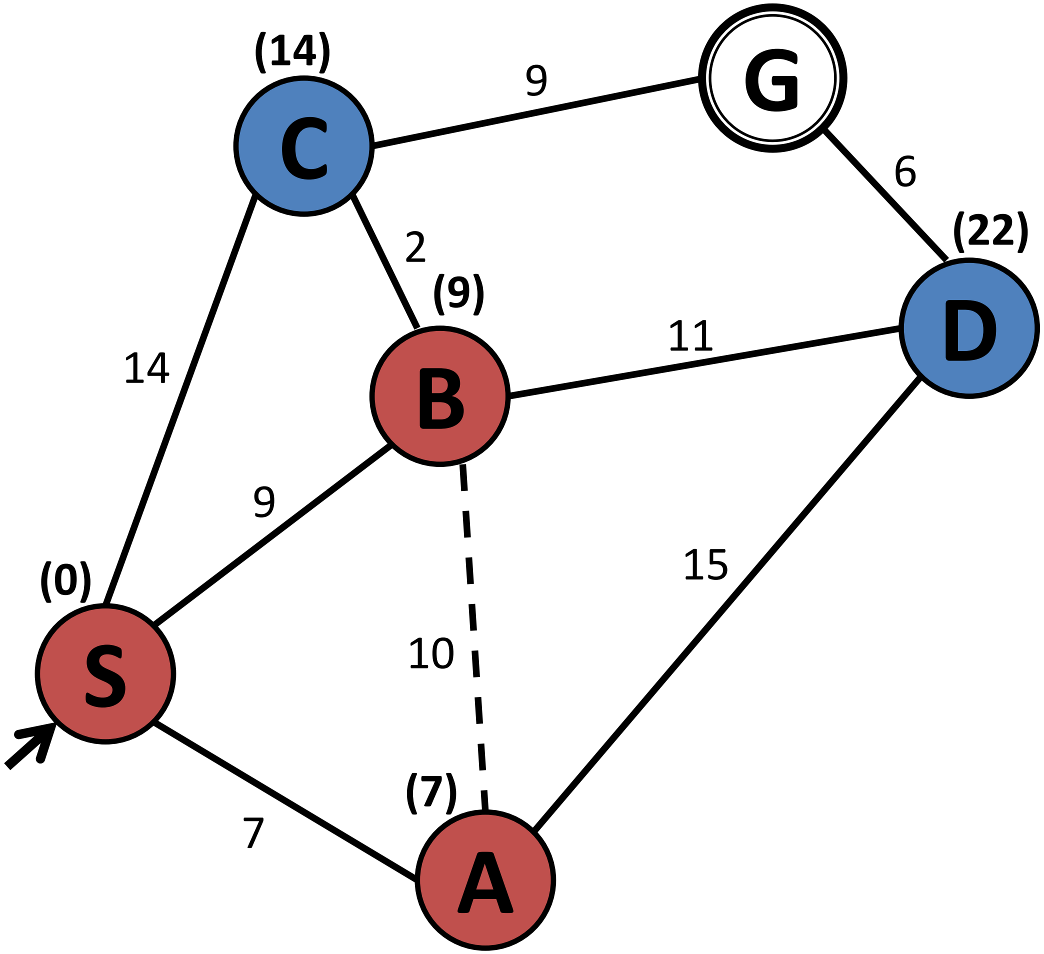
Legend:

○ S **Node**

🔵 S **Visited Node**

🔴 S **Expanded Node**

Uniform-Cost Search



Visited Nodes:
{S,A,B,C,D}

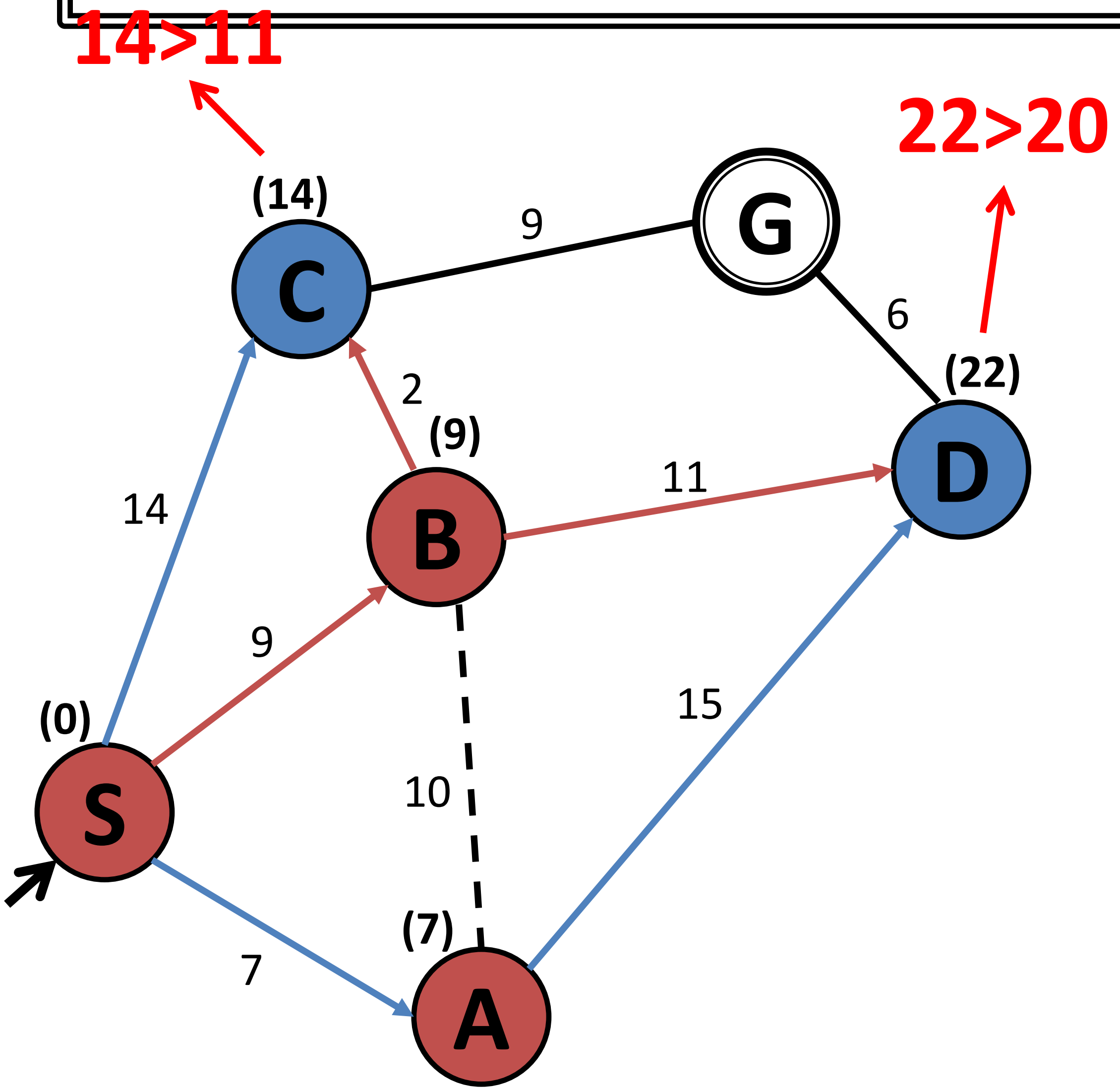
Legend:

S **Node**

S **Visited Node**

S **Expanded Node**

Uniform-Cost Search



Visited Nodes:
{S,A,B,C,D}

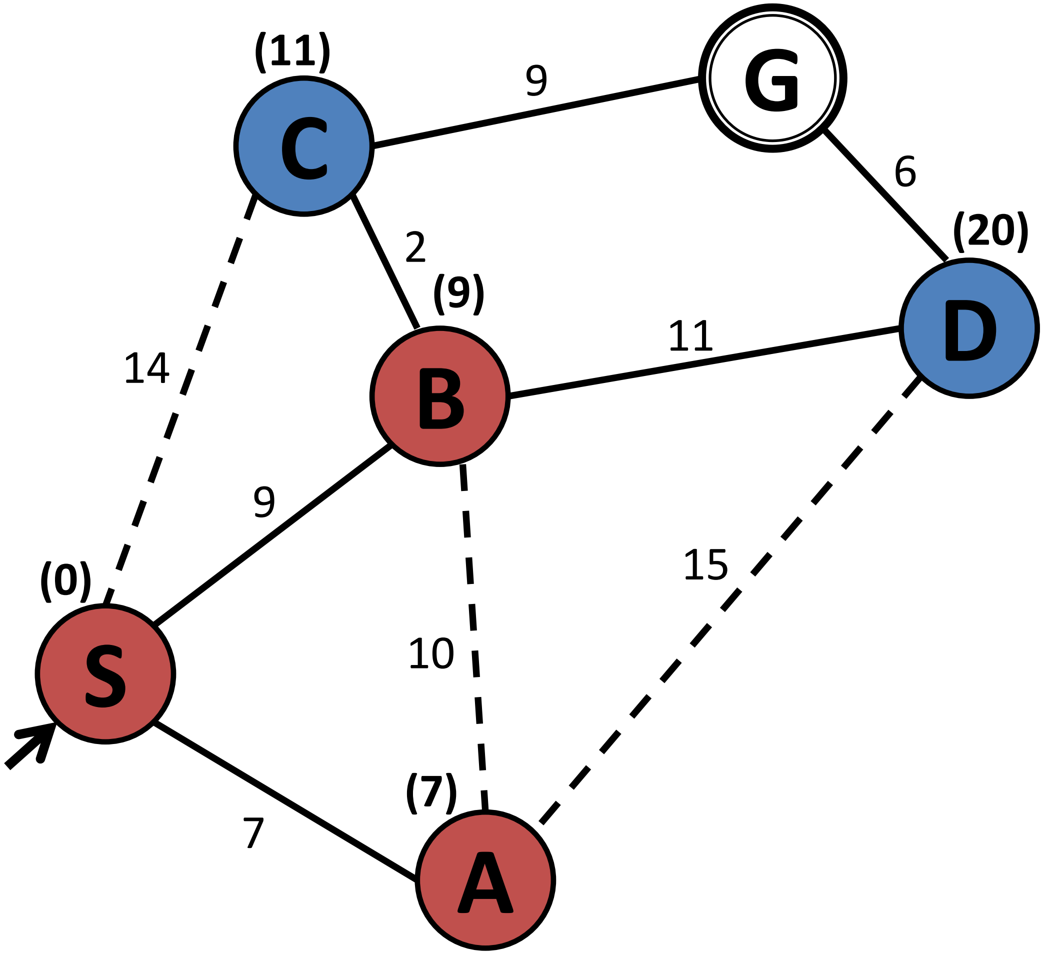
Legend:

S **Node**

S **Visited Node**

S **Expanded Node**

Uniform-Cost Search

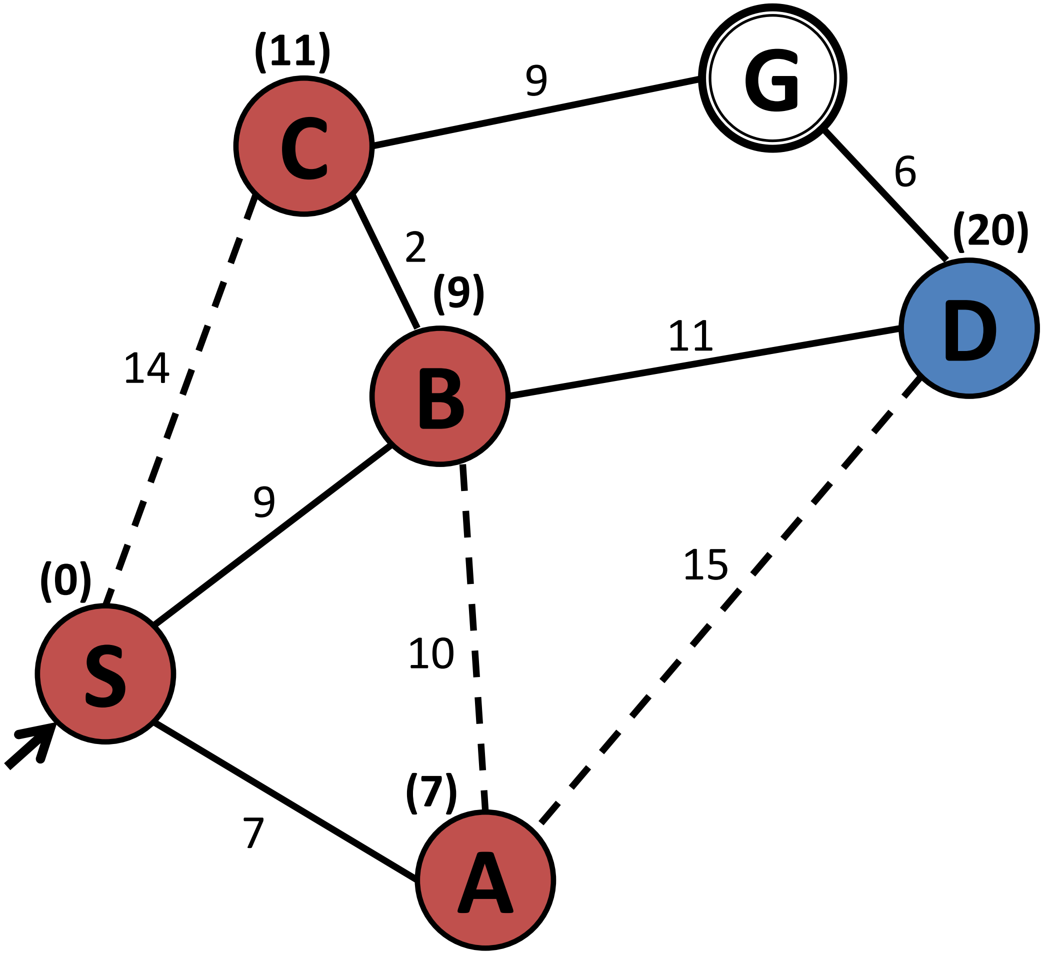


Visited Nodes:
{S,A,B,C,D}

Legend:

○ S	**Node**
🟦 S	**Visited Node**
🟥 S	**Expanded Node**

Uniform-Cost Search



Visited Nodes:
{S,A,B,C,D}

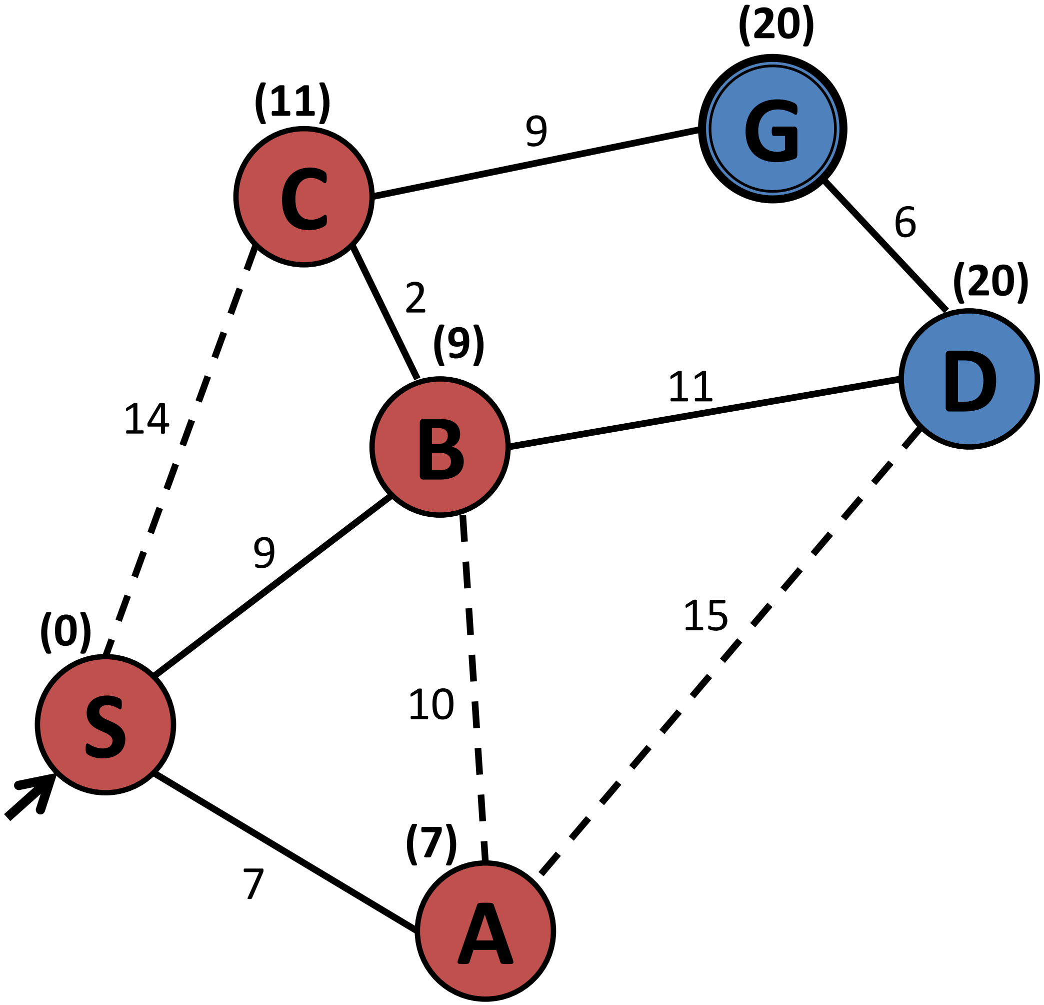
Legend:

S **Node**

S **Visited Node**

S **Expanded Node**

Uniform-Cost Search



Visited Nodes:
{S,A,B,C,D,*G*}

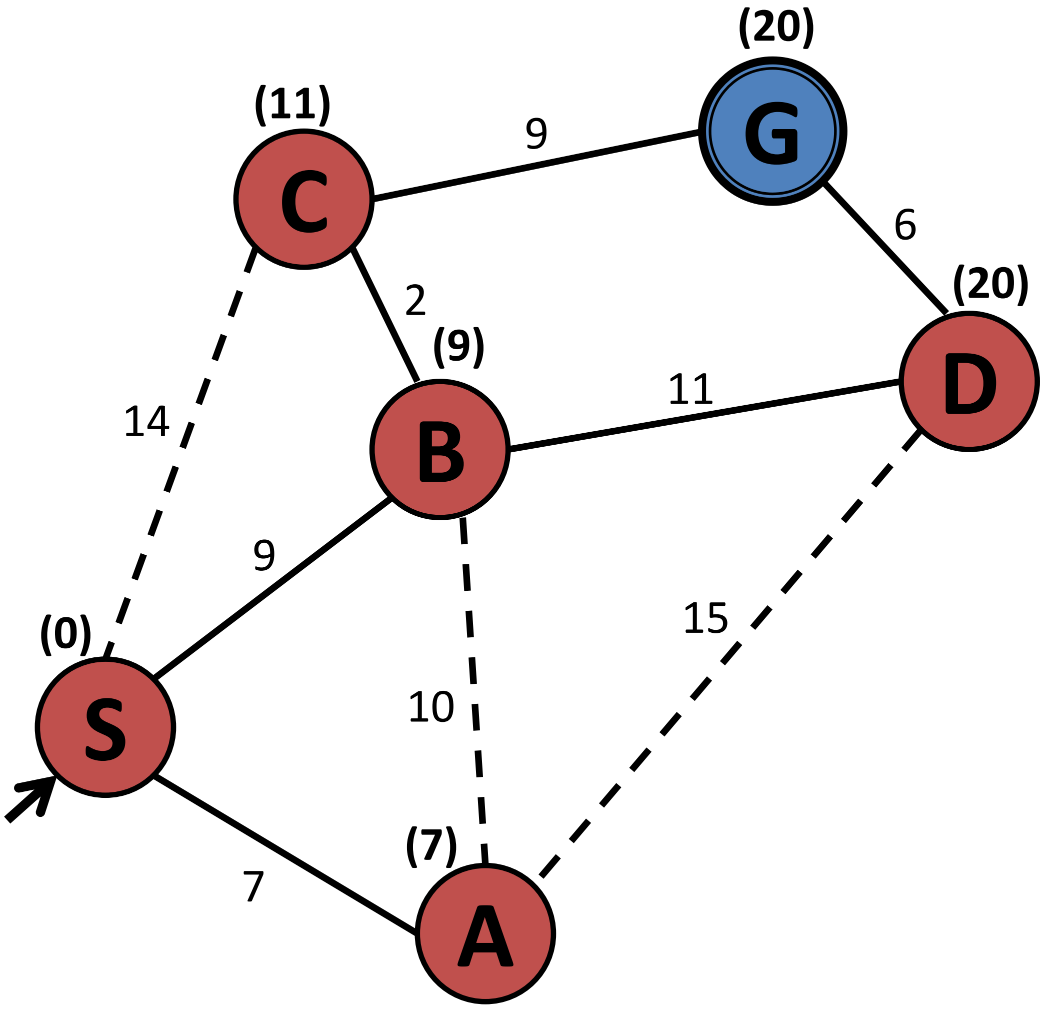
Legend:

S **Node**

S **Visited Node**

S **Expanded Node**

Uniform-Cost Search

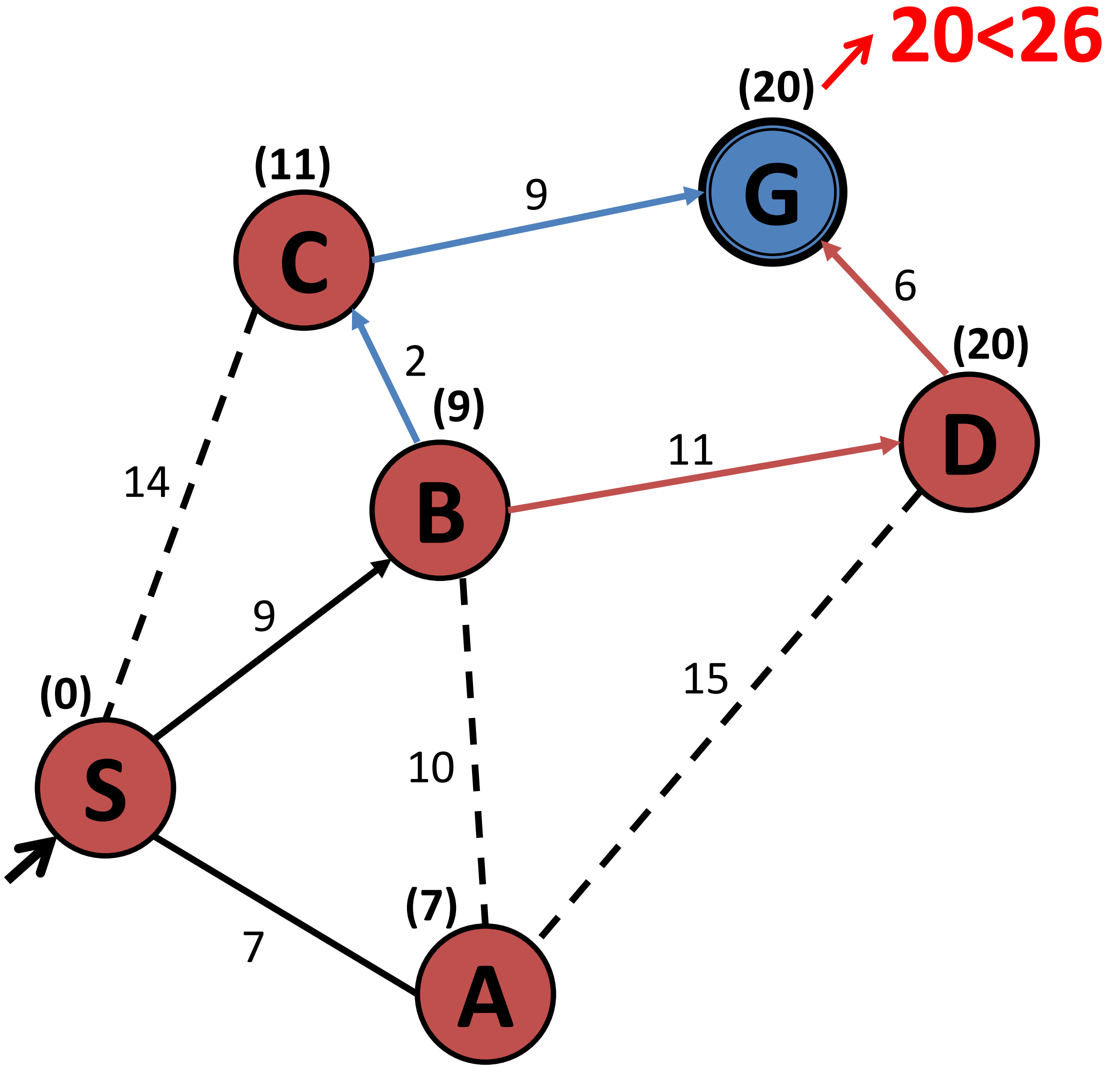


Visited Nodes:
{S,A,B,C,D,*G*}

Legend:

- ○ S **Node**
- 🟦 S **Visited Node**
- 🟥 S **Expanded Node**

Uniform-Cost Search



Visited Nodes:
{S,A,B,C,D,*G*}

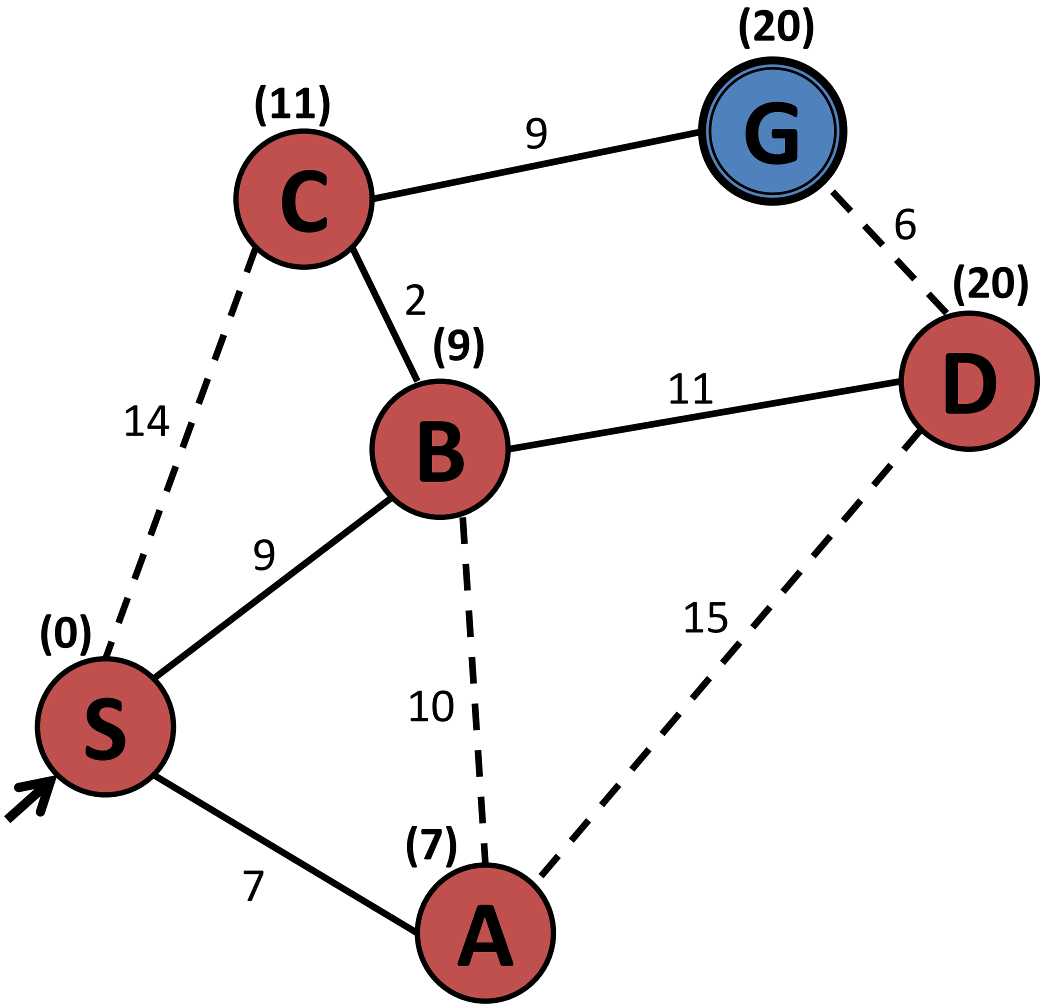
Legend:

S **Node**

S **Visited Node**

S **Expanded Node**

Uniform-Cost Search



Visited Nodes:
{S,A,B,C,D,*G*}

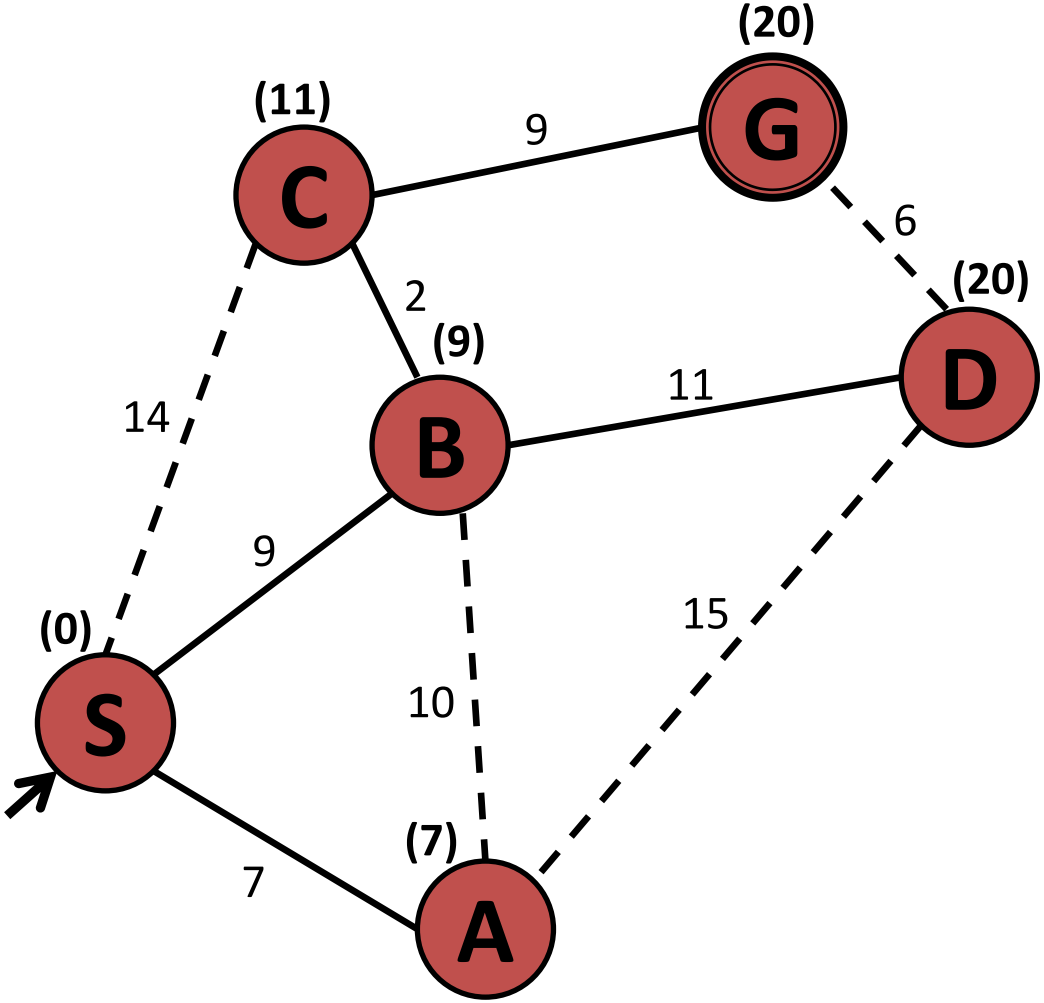
Legend:

S **Node**

S **Visited Node**

S **Expanded Node**

Uniform-Cost Search



Visited Nodes:
{S,A,B,C,D,*<u>G</u>*}

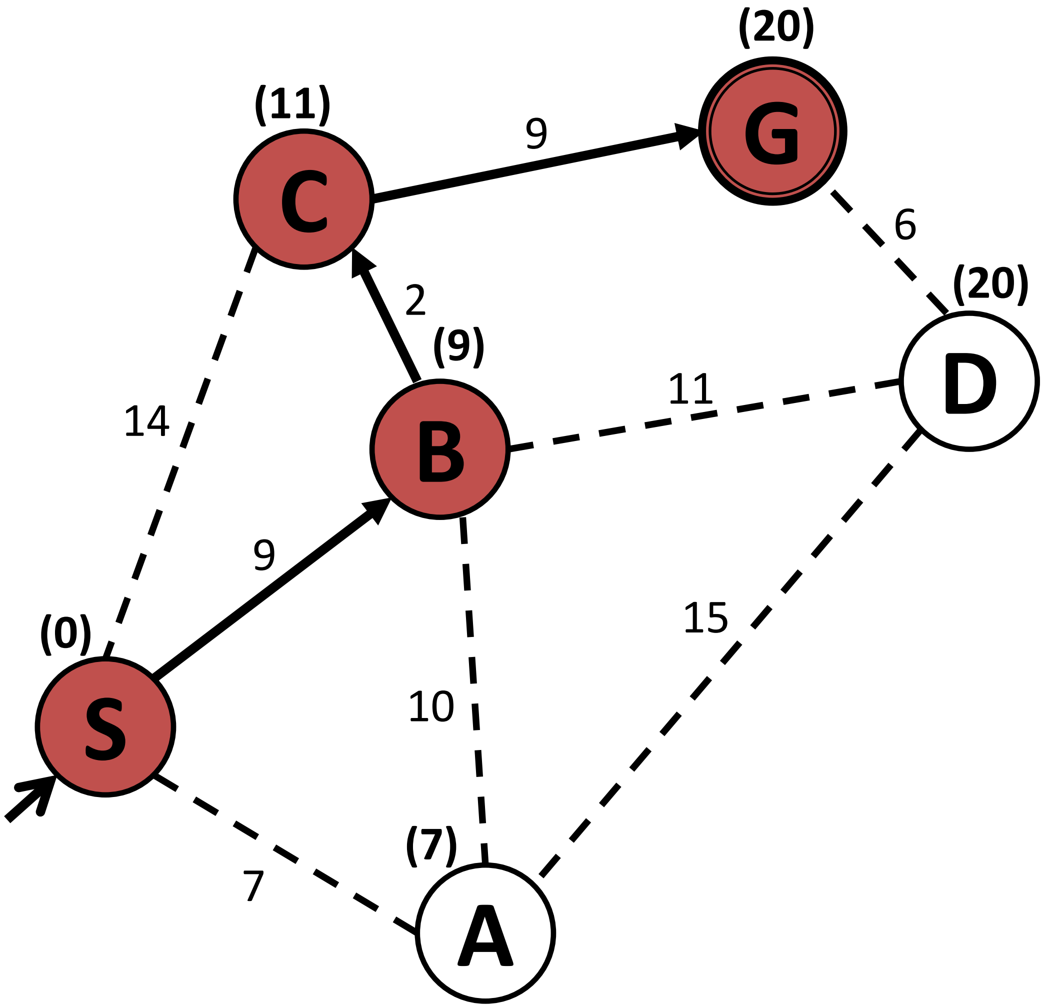
Legend:

○ **Node**

🔵 **Visited Node**

🔴 **Expanded Node**

Uniform-Cost Search



Visited Nodes:
{S,A,B,C,D,*G*}

Legend:

○ **Node**

🔵 **Visited Node**

🔴 **Expanded Node**

Uniform-Cost Search (UCS)

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

** Expand least-cost unexpanded node using queue ordered by path cost, lowest first*

** Equivalent to breadth-first if step costs all equal*

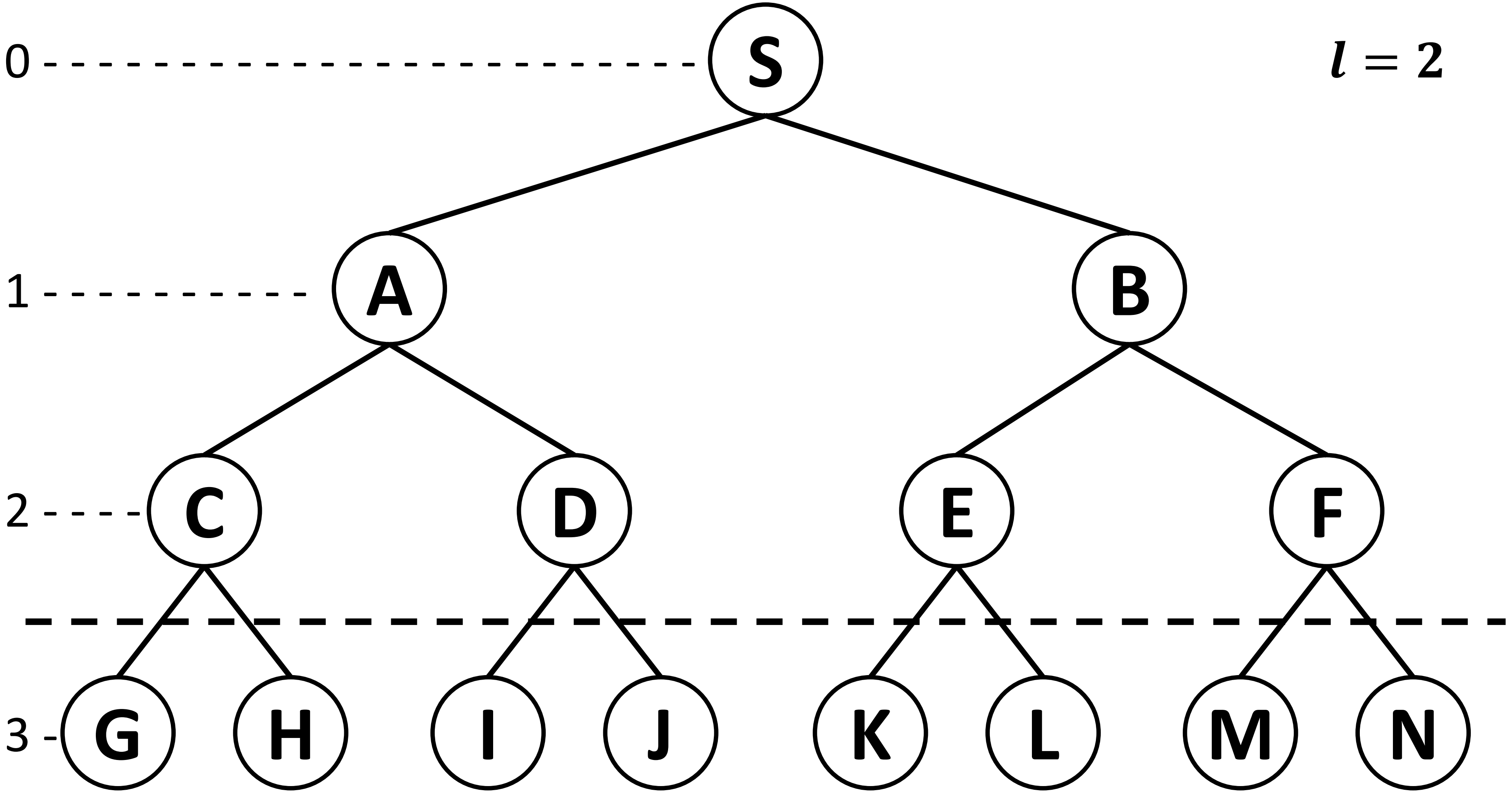
Depth-Limited Search

- *= depth-first search* with depth limit ***l***,

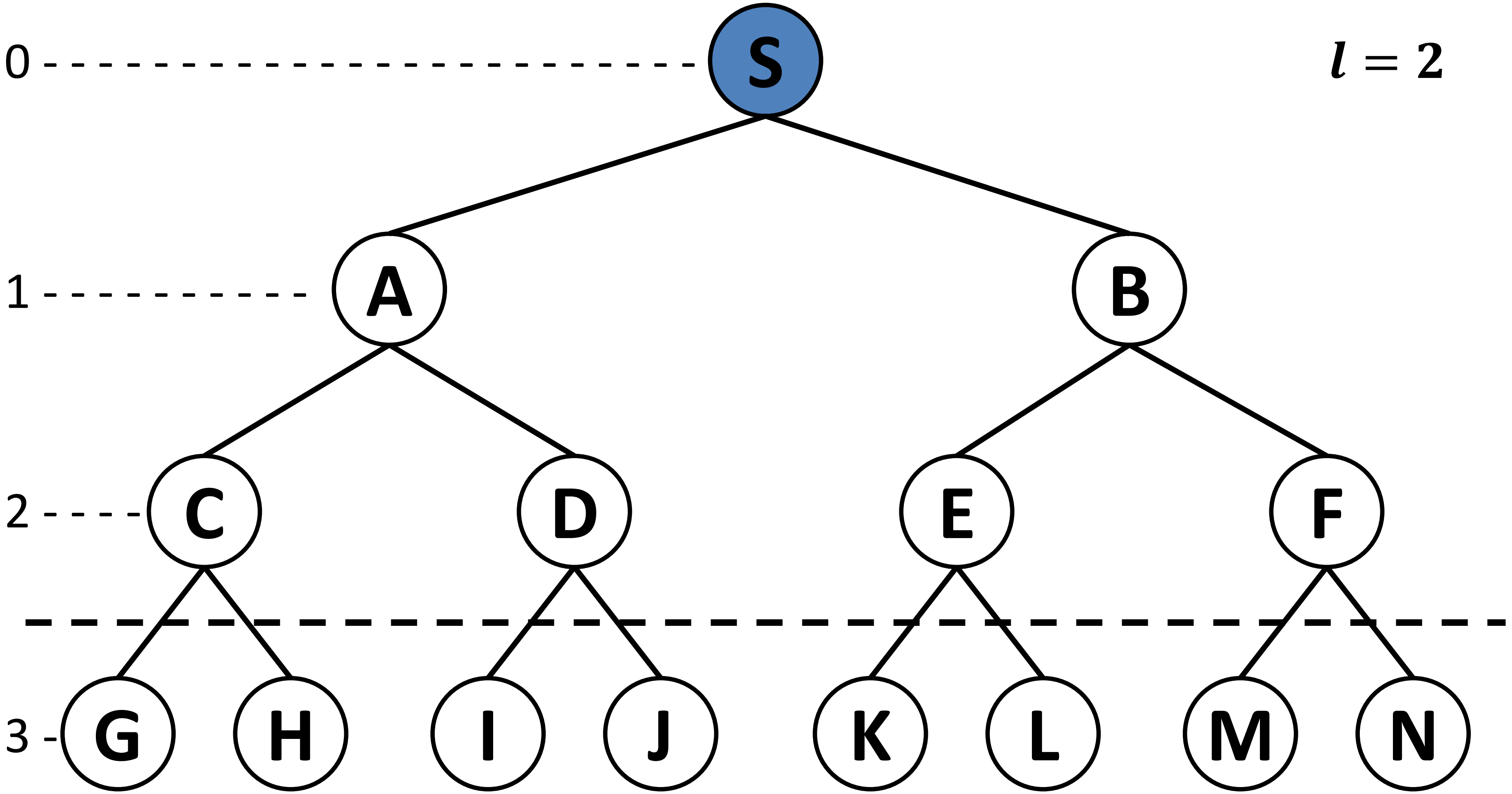
  i.e., nodes at depth ***l*** have no successors

- Also known as *Depth-Bounded Search*

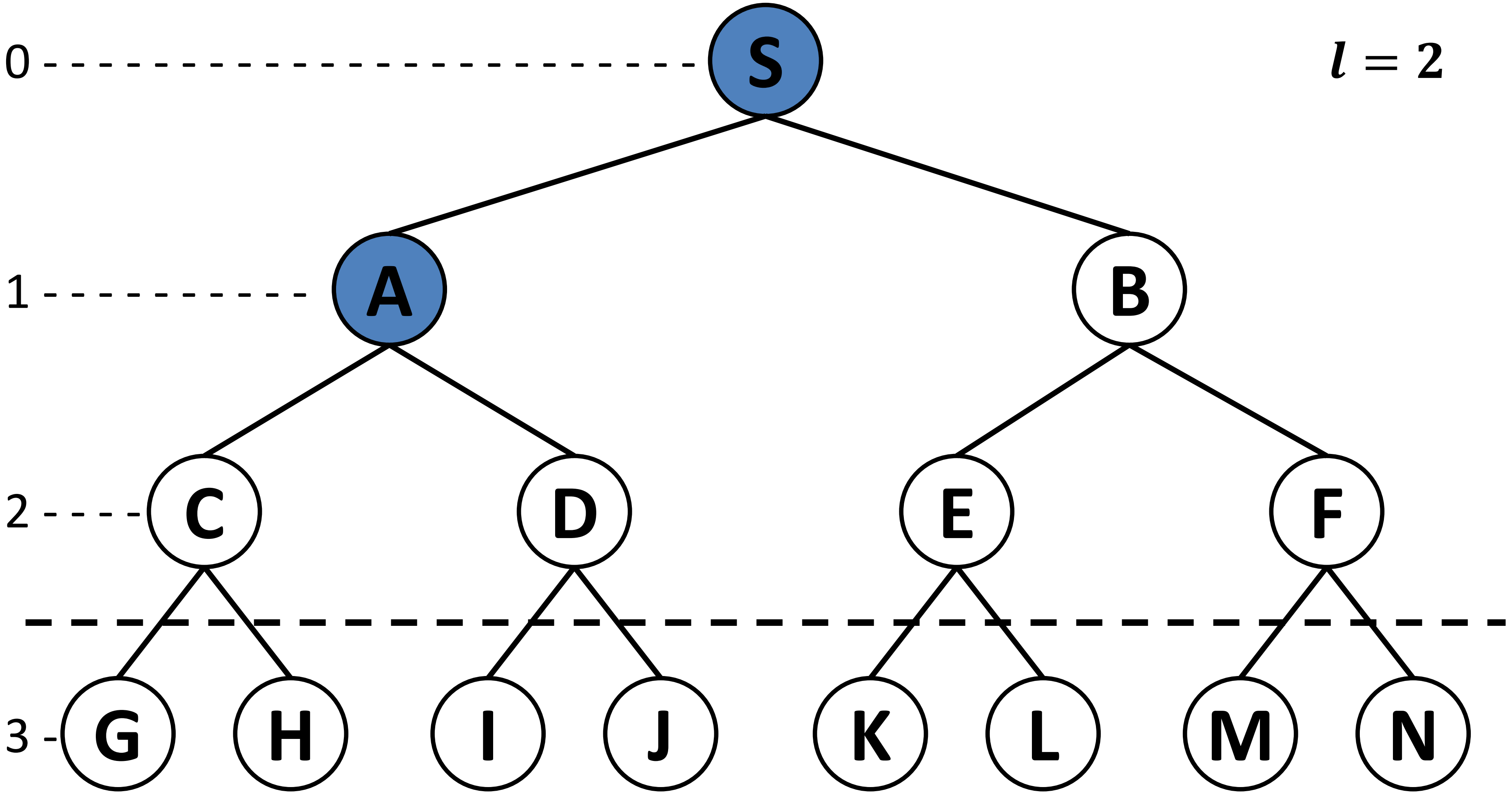
Depth-Limited Search



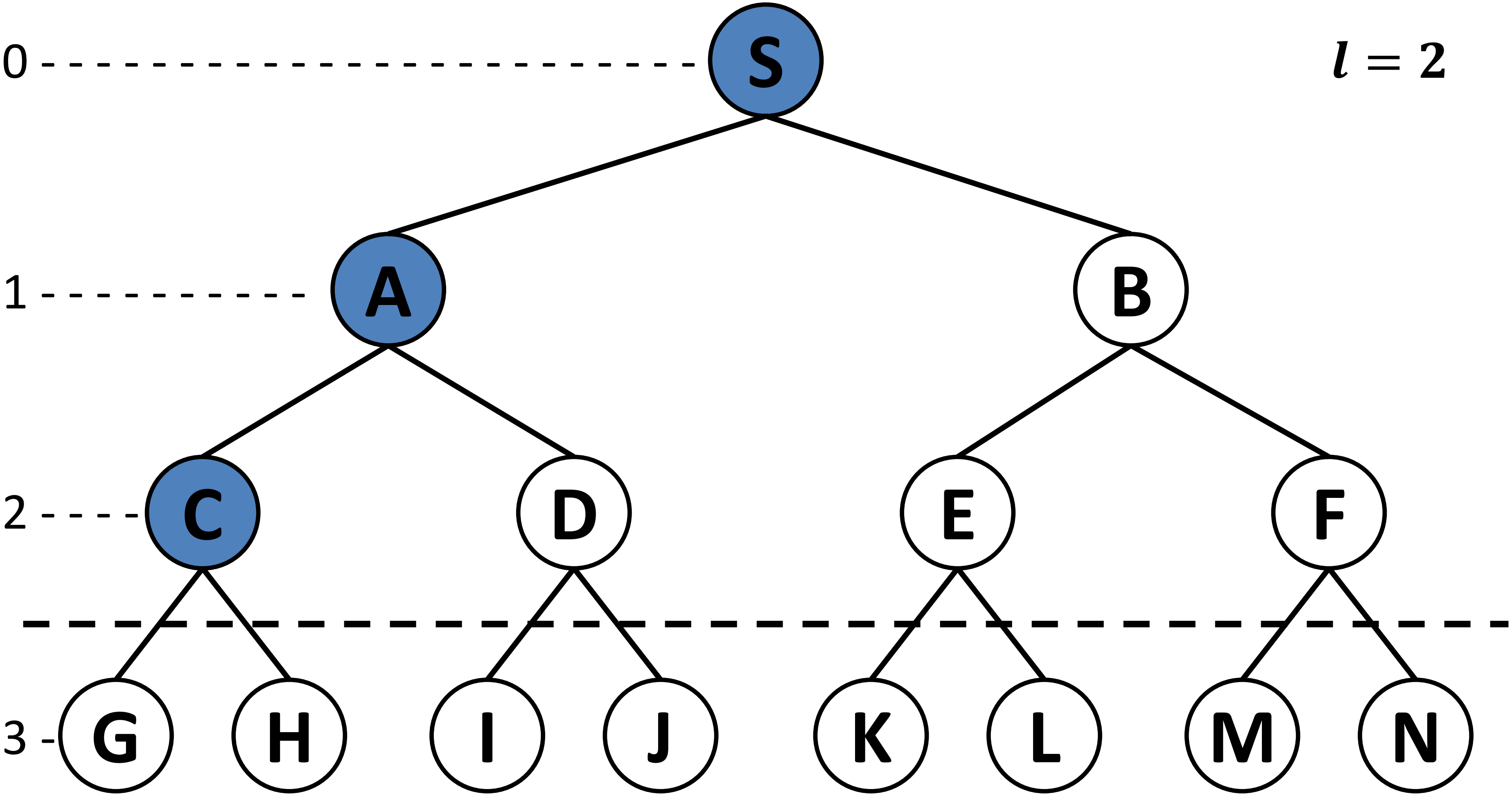
Depth-Limited Search



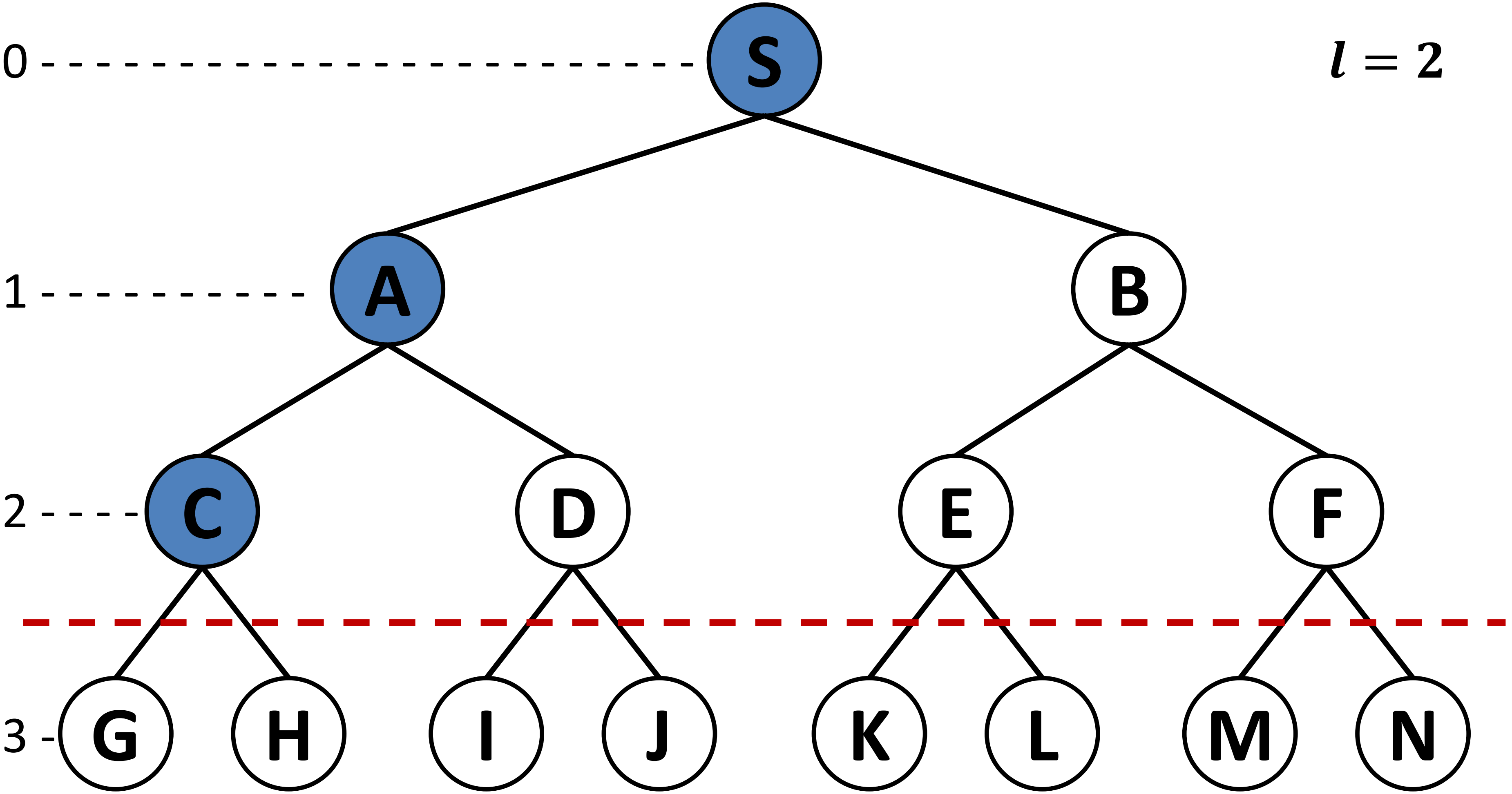
Depth-Limited Search



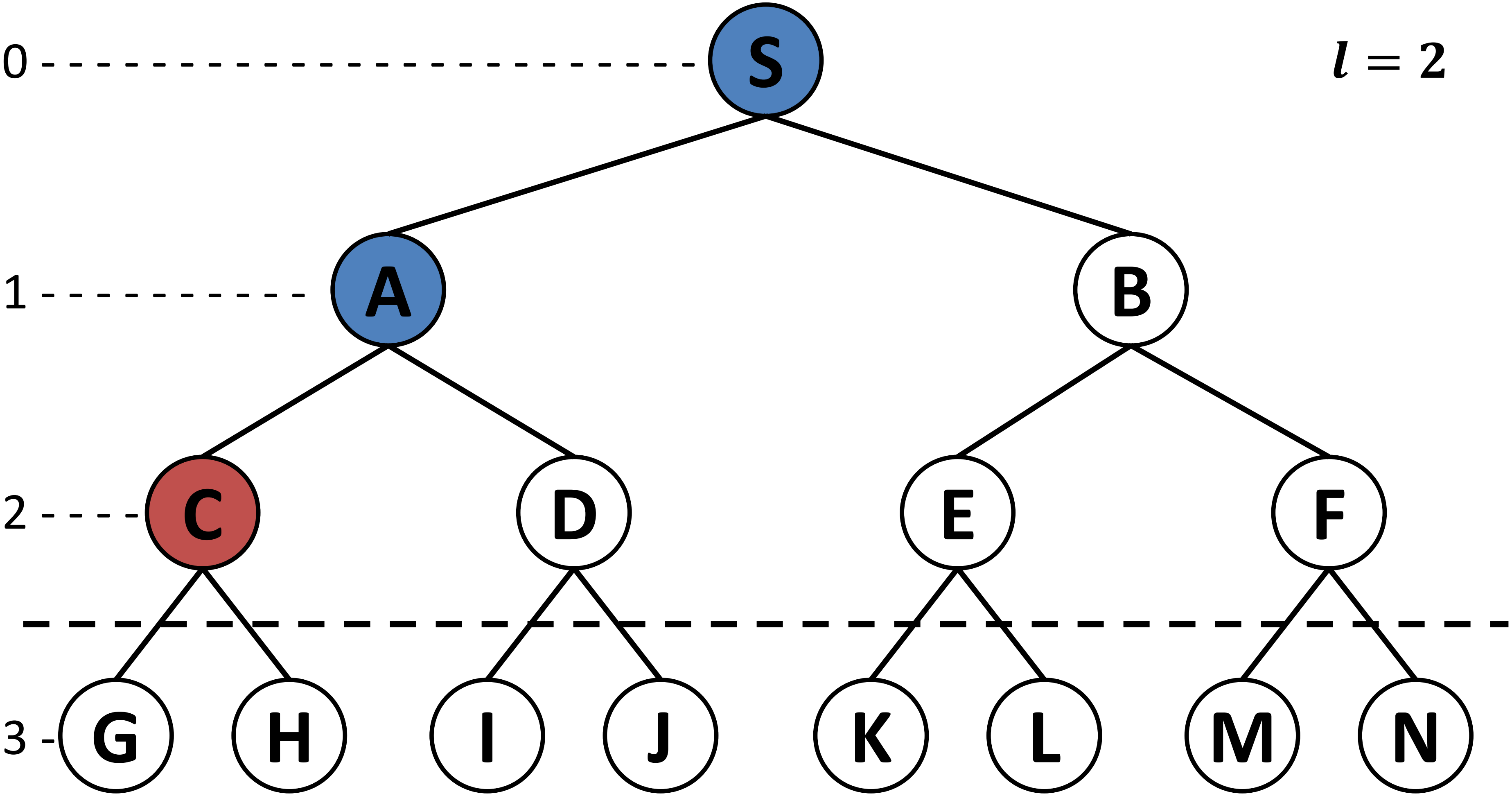
Depth-Limited Search



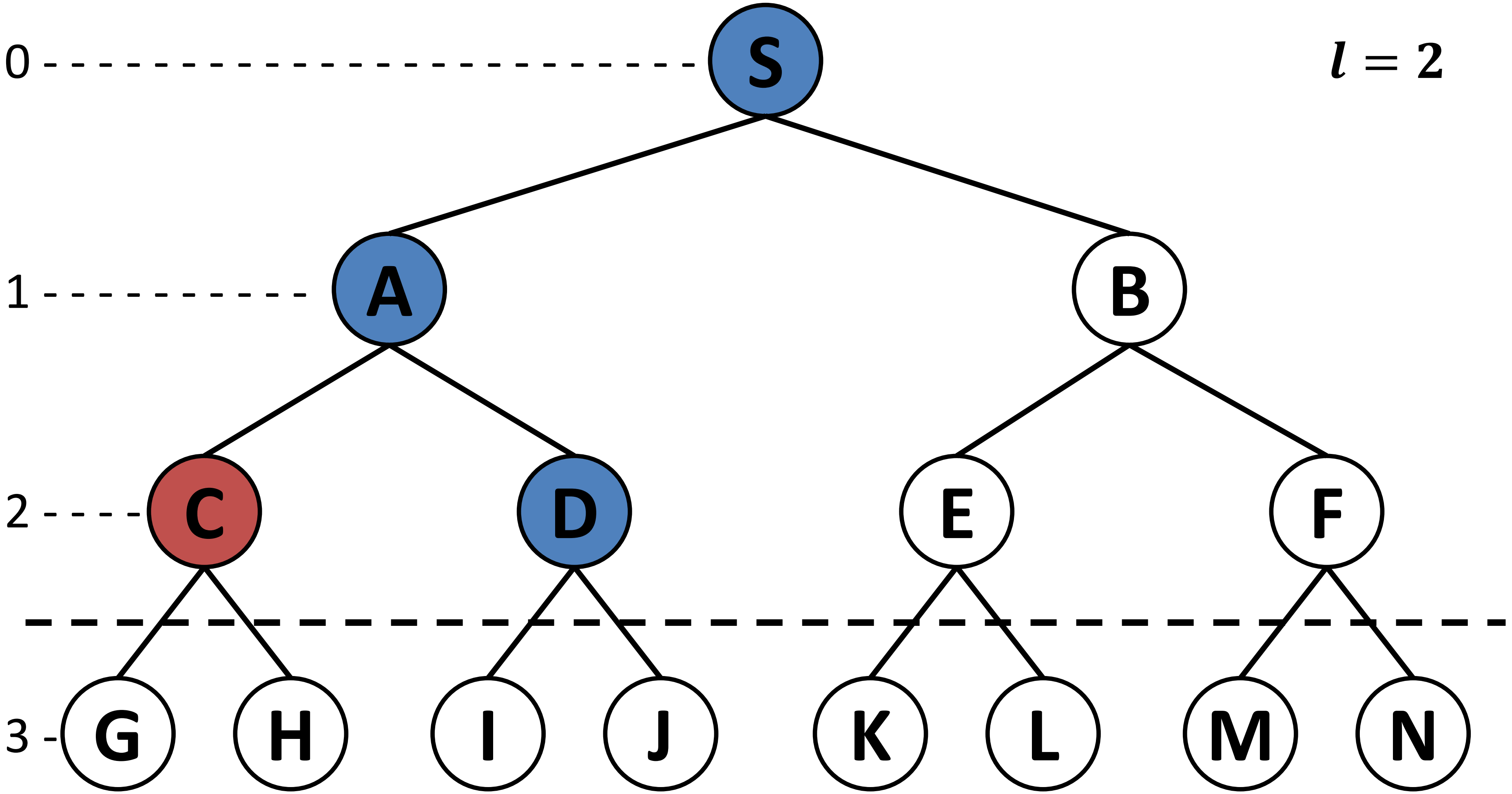
Depth-Limited Search



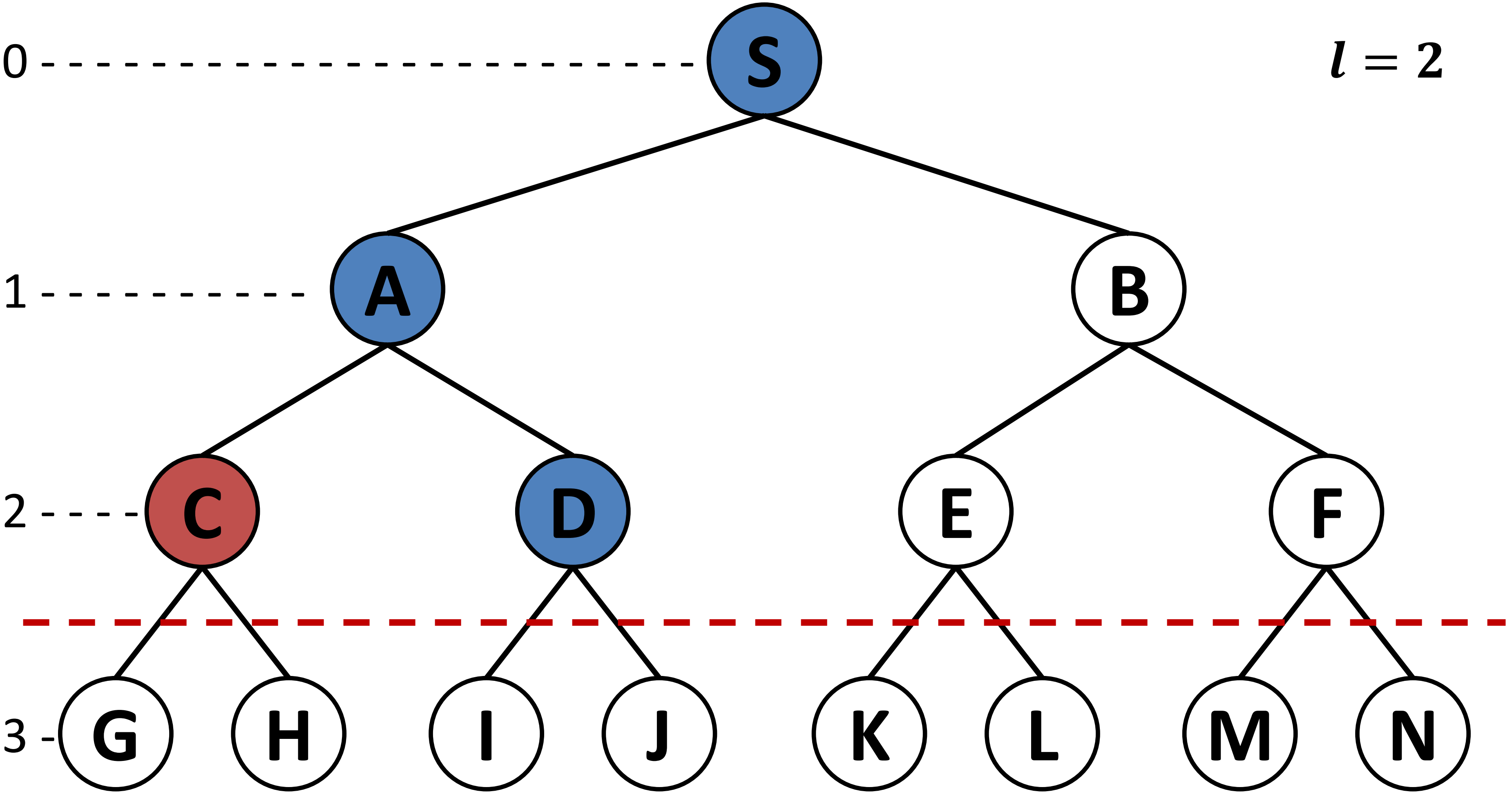
Depth-Limited Search



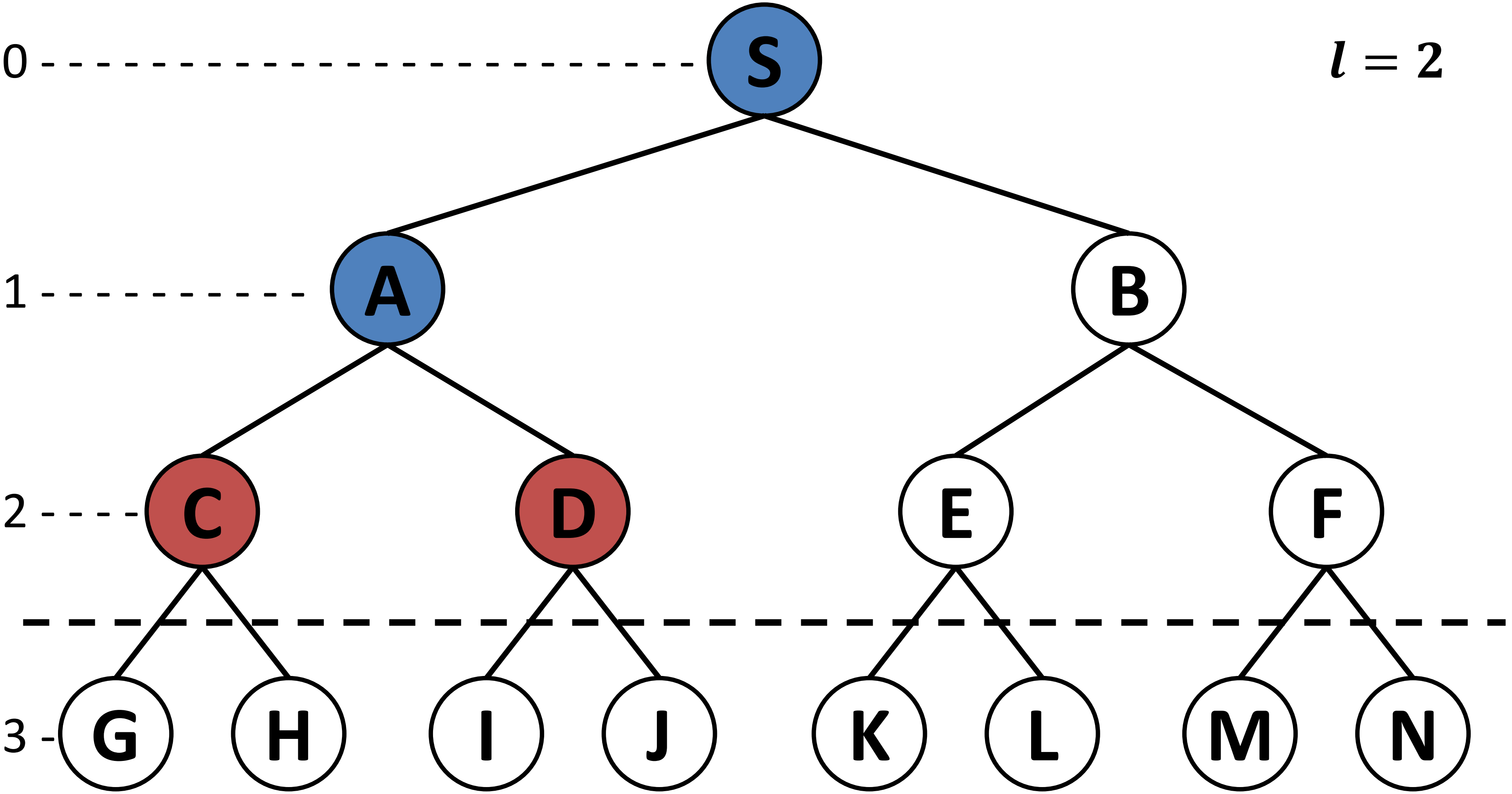
Depth-Limited Search



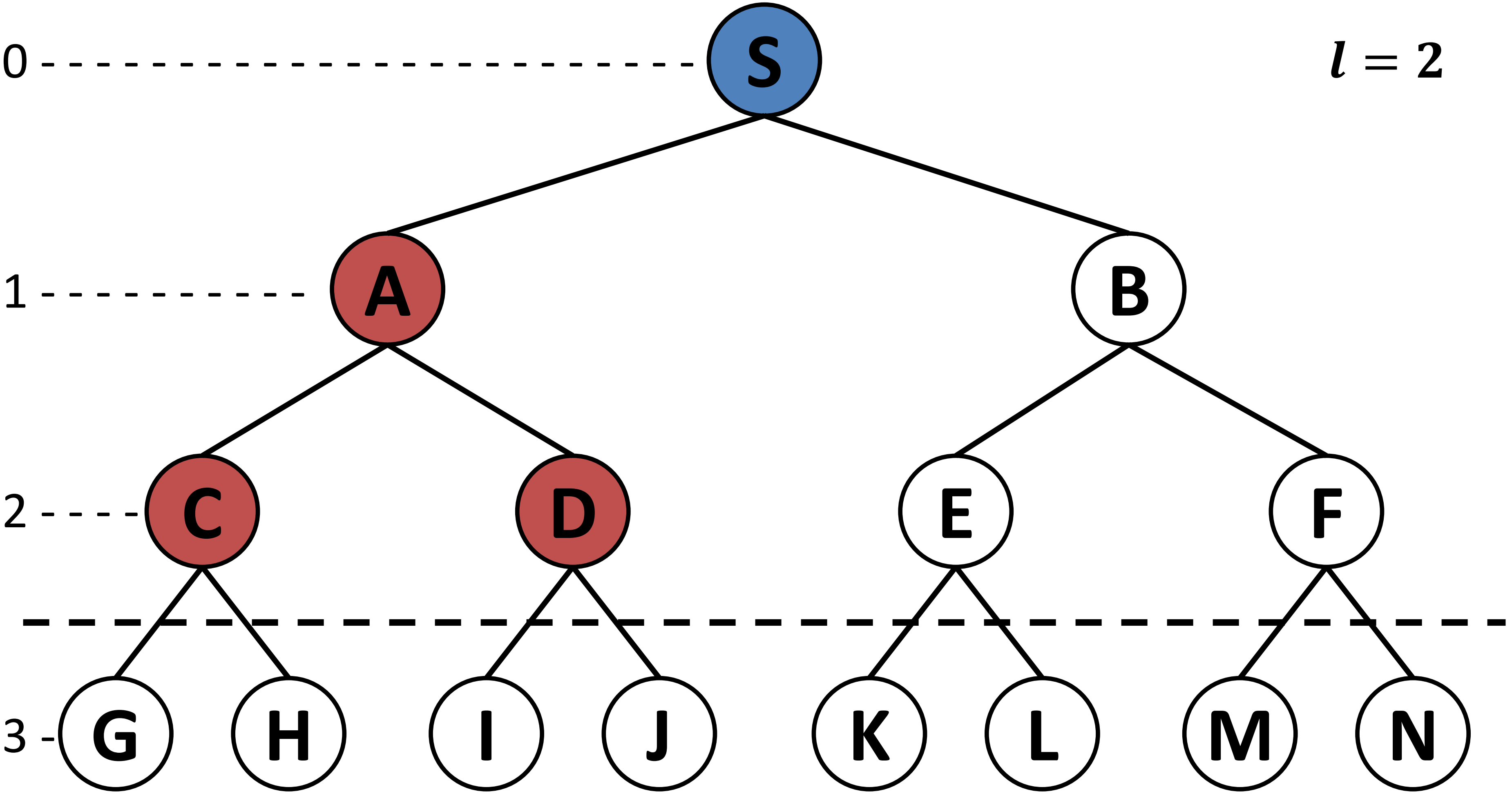
Depth-Limited Search



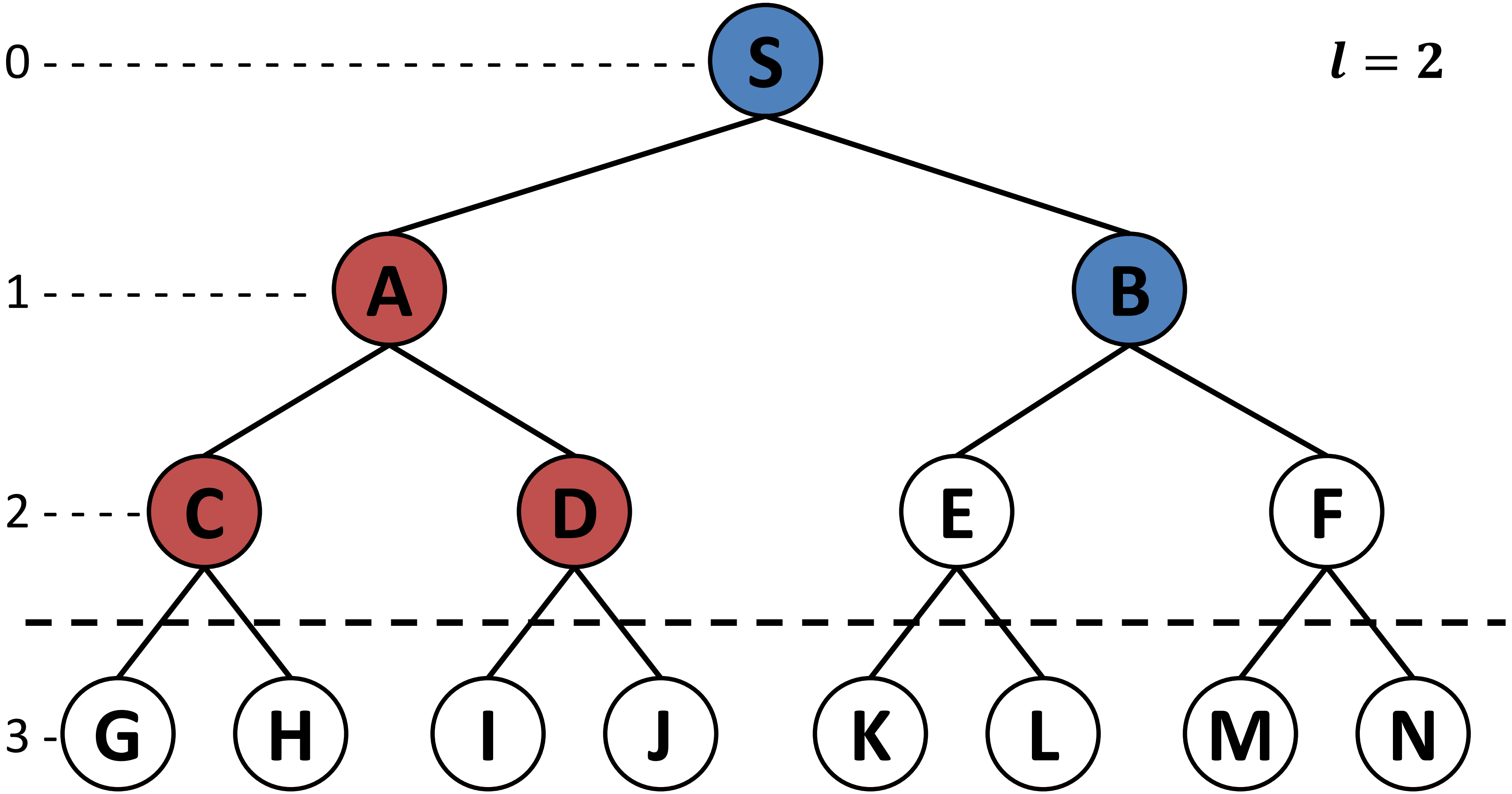
Depth-Limited Search



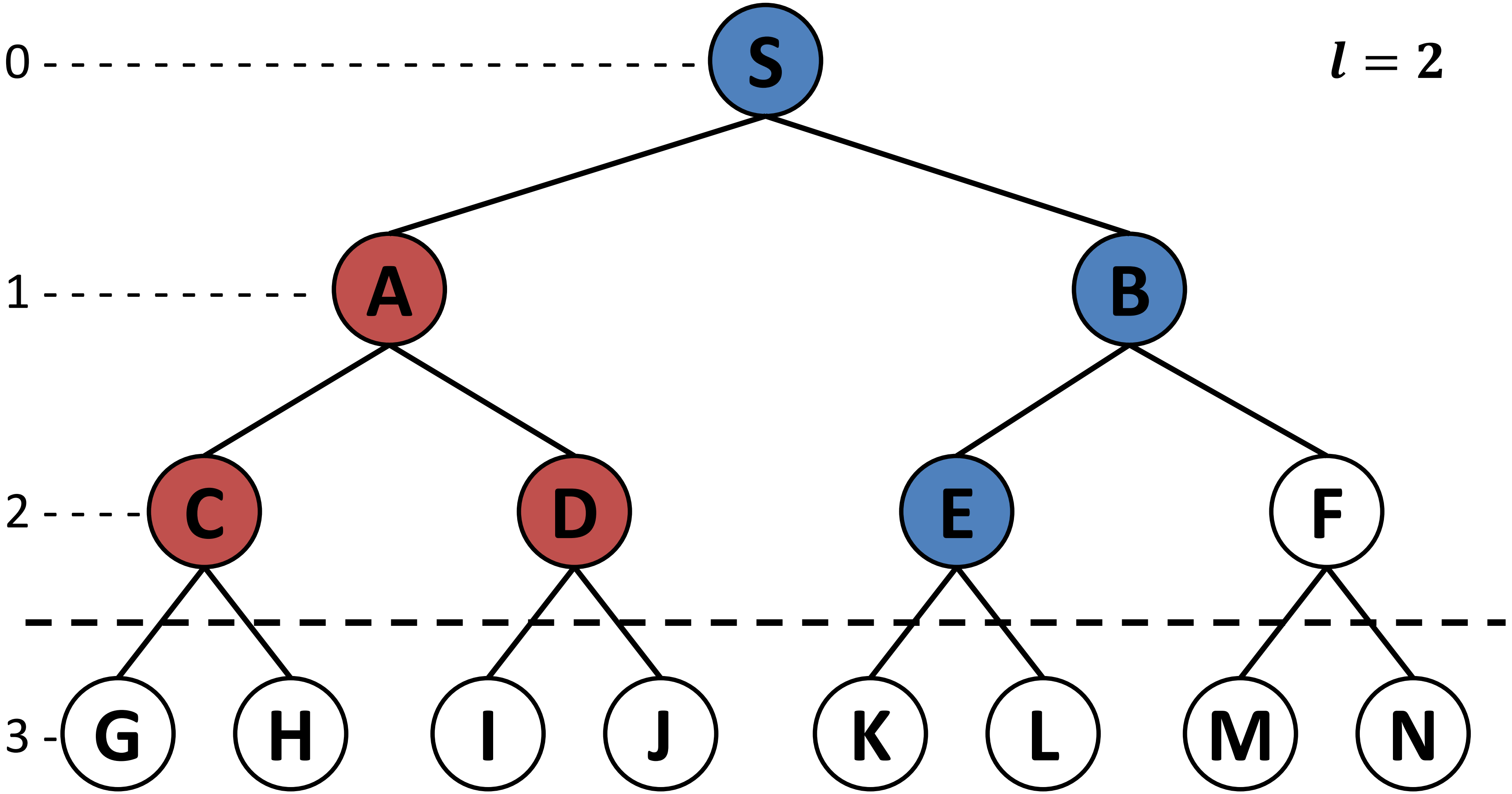
Depth-Limited Search



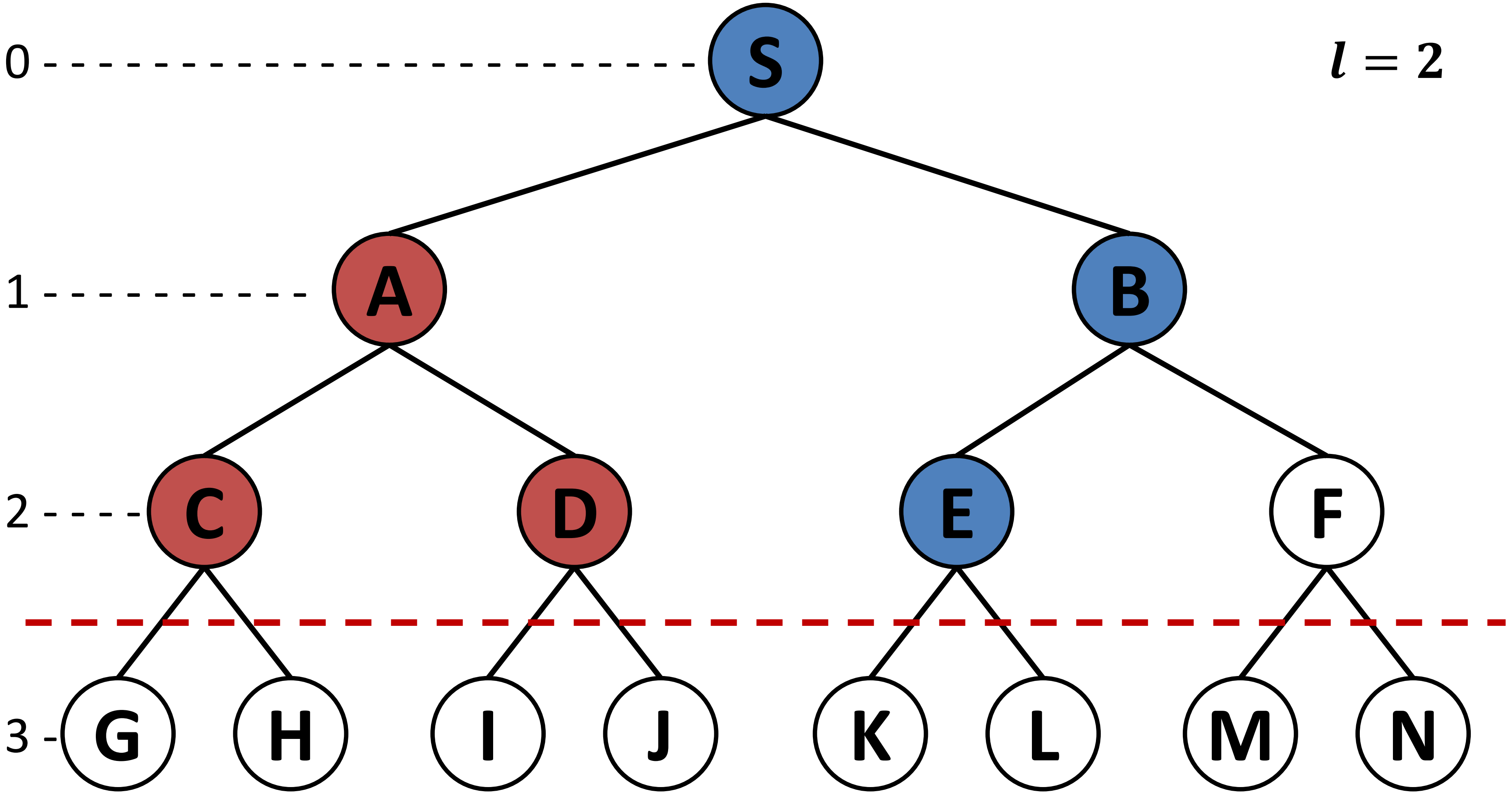
Depth-Limited Search



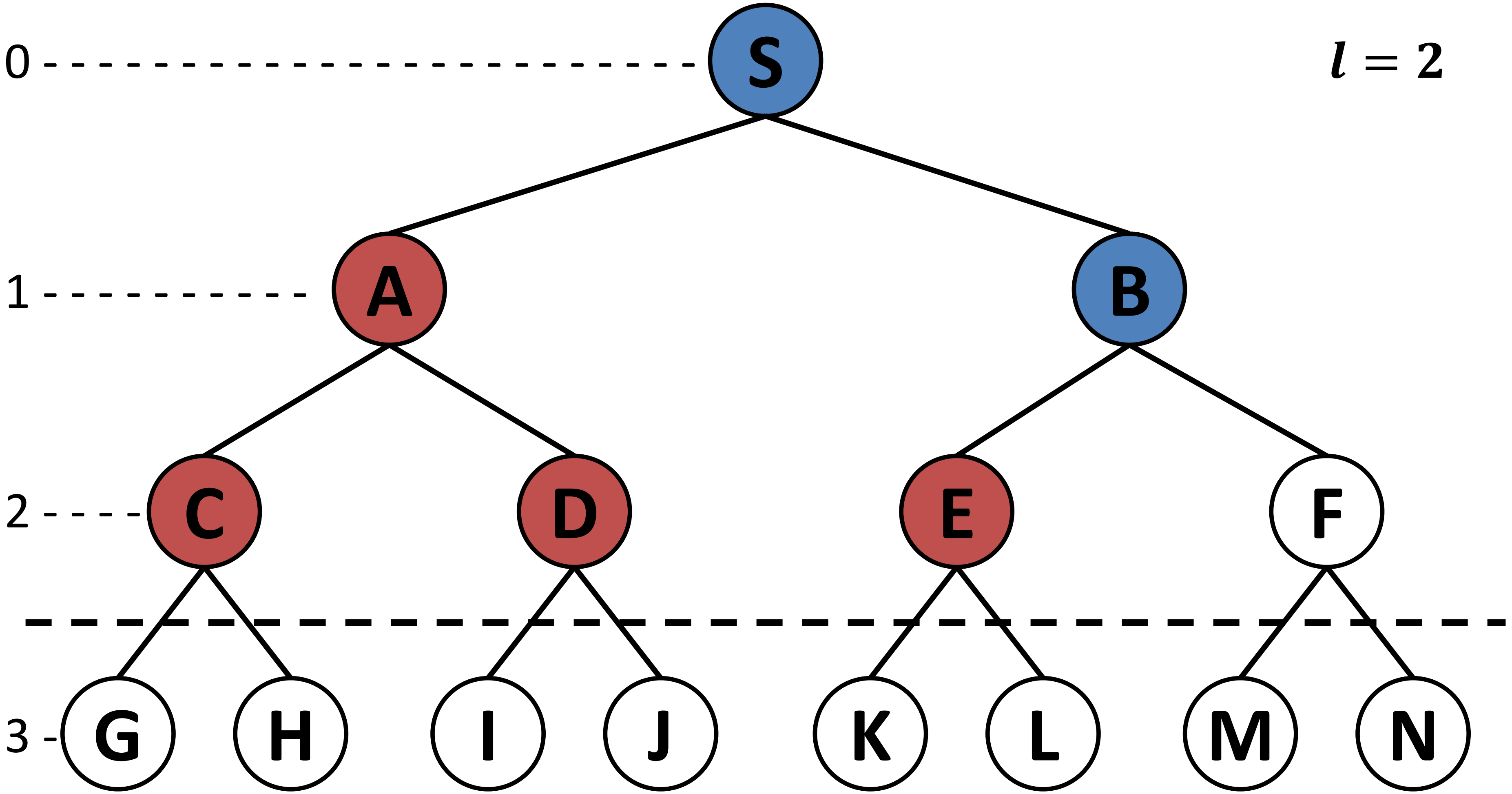
Depth-Limited Search



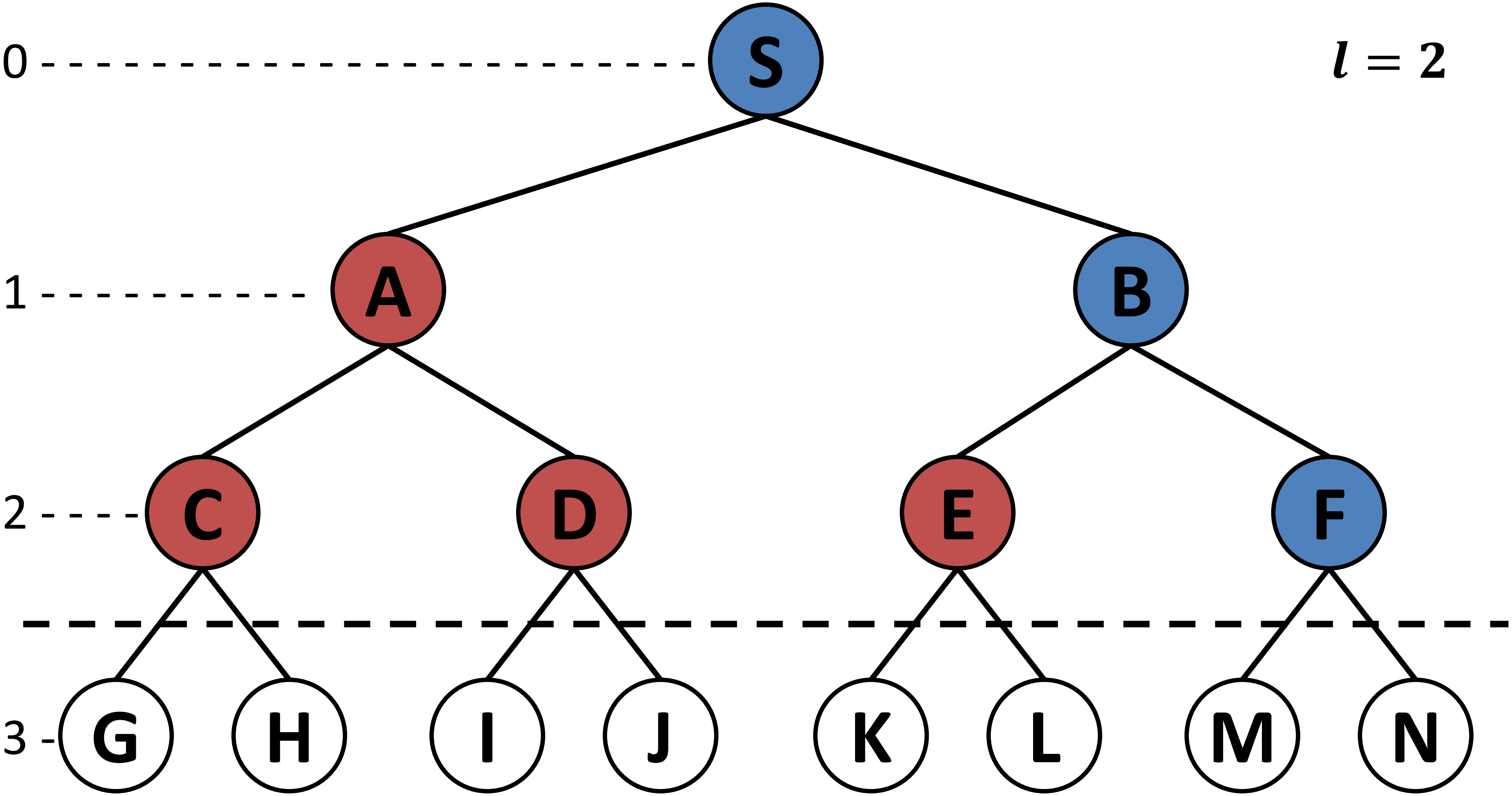
Depth-Limited Search



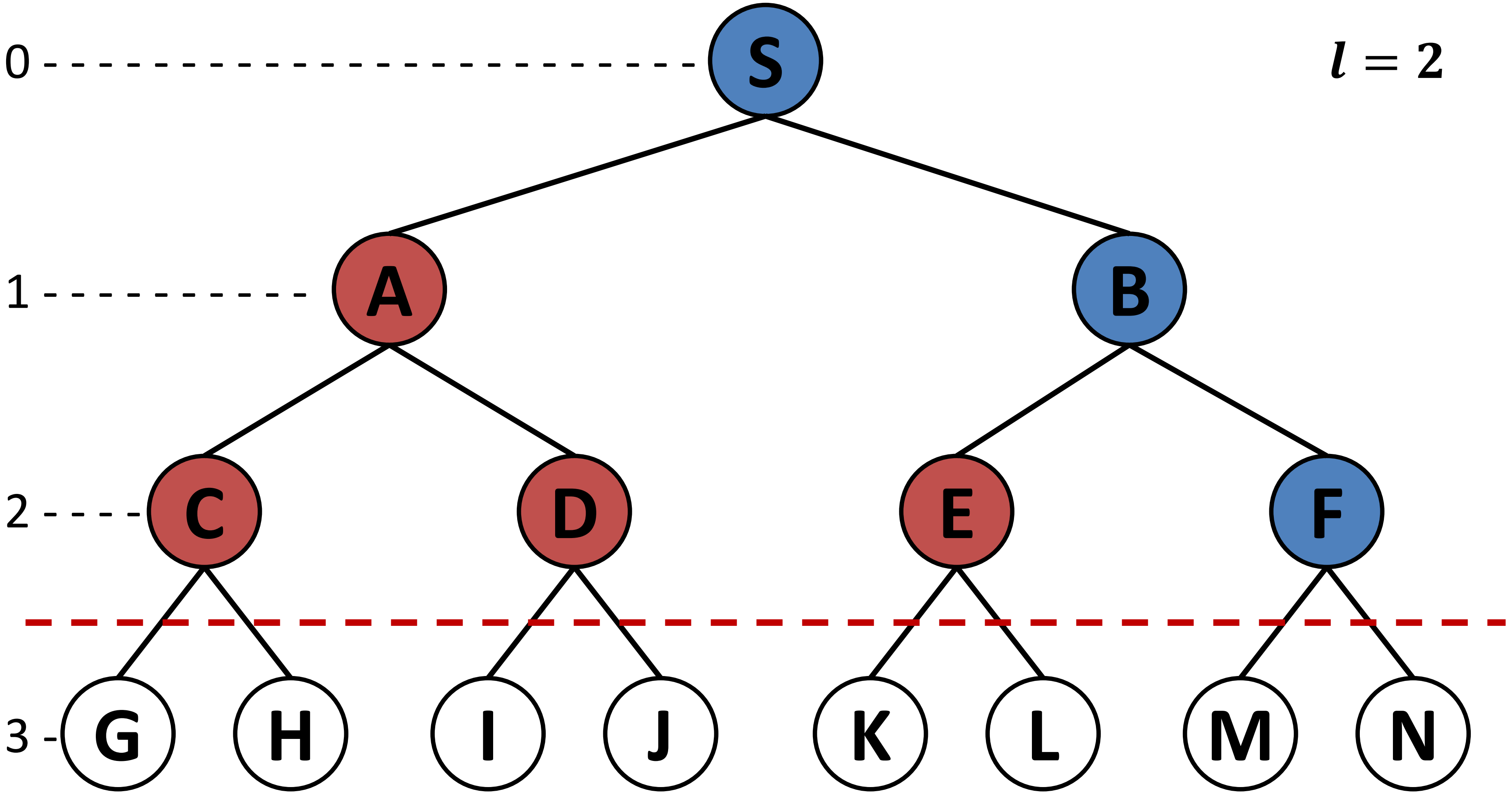
Depth-Limited Search



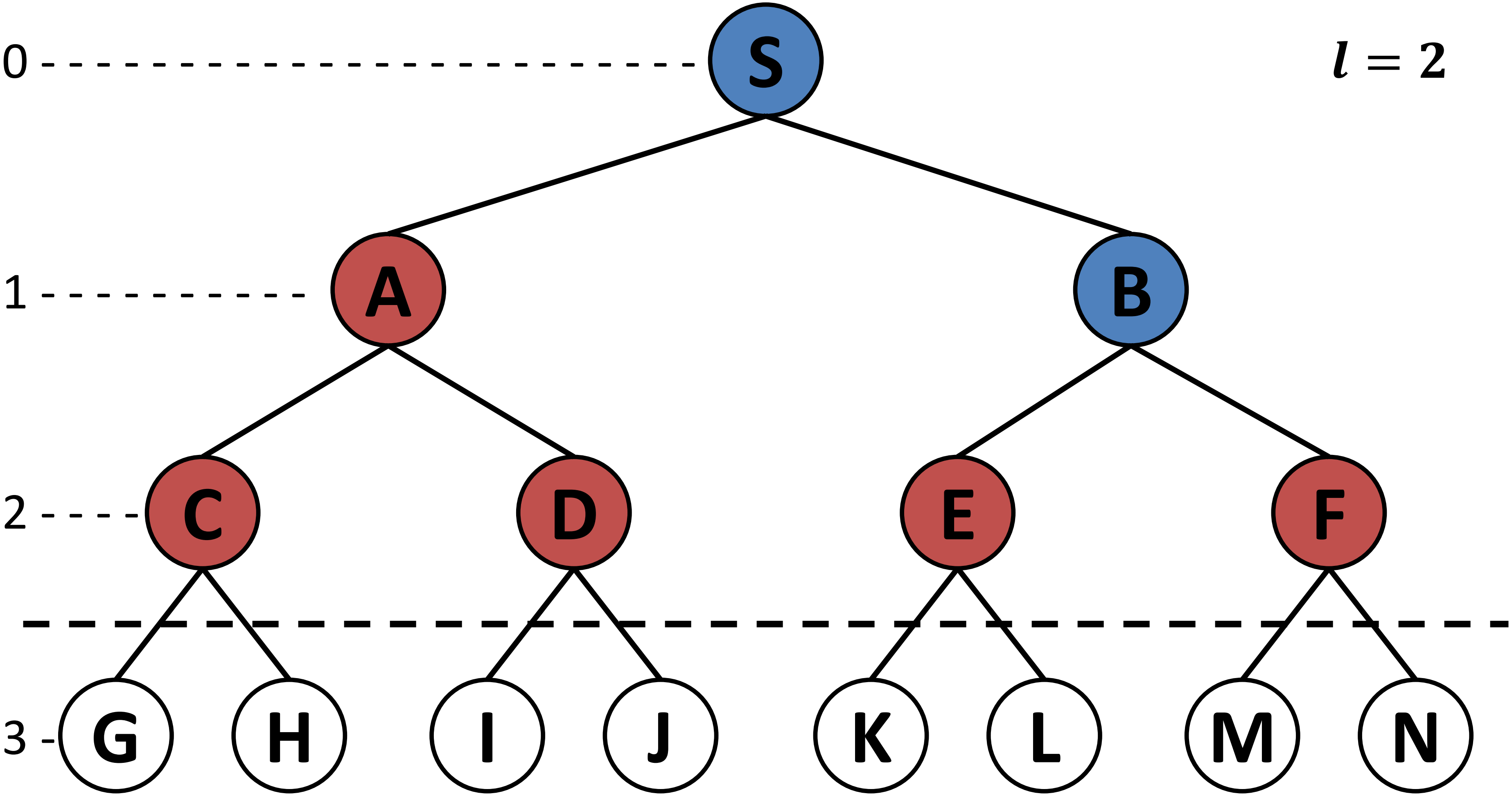
Depth-Limited Search



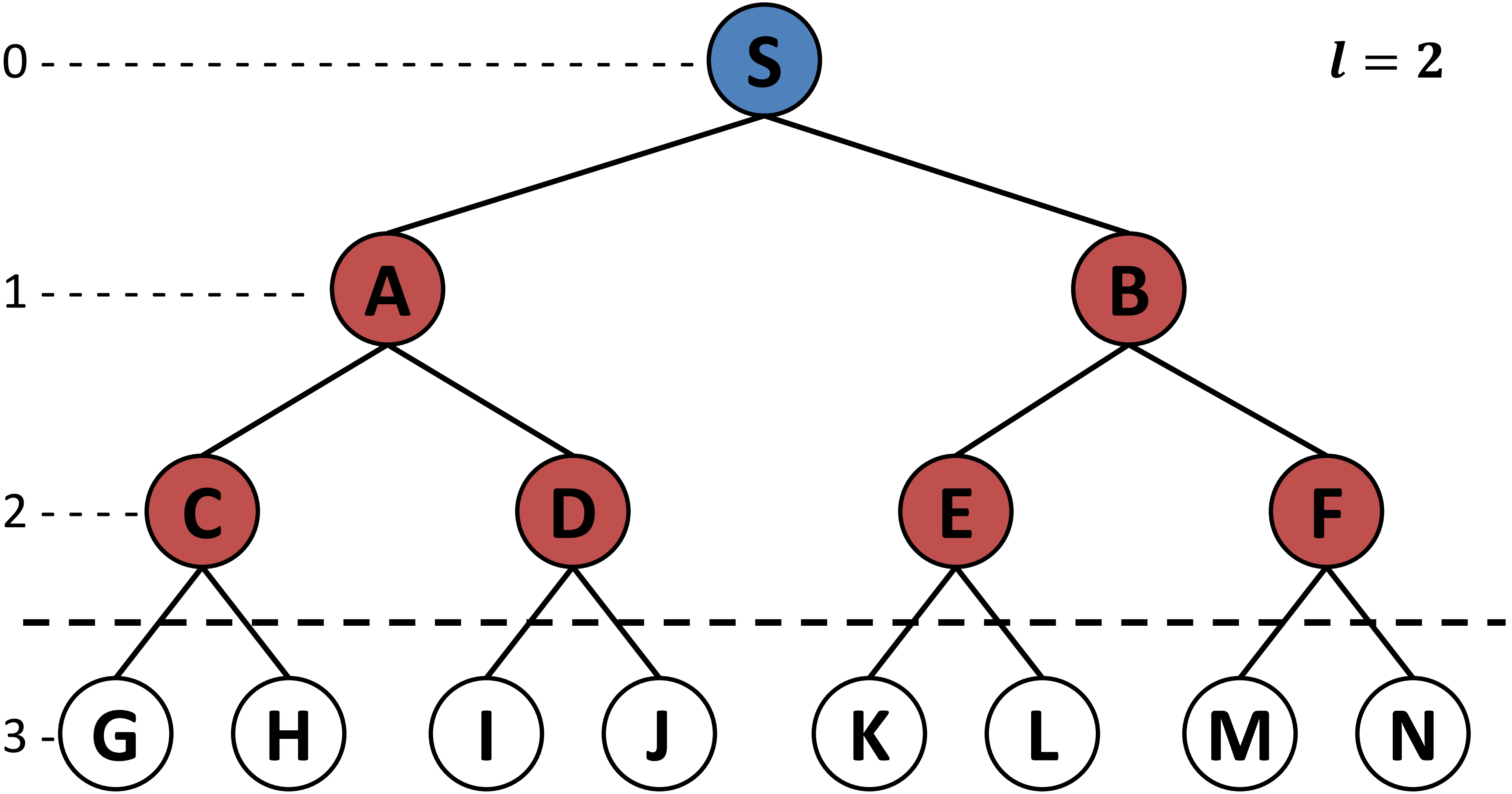
Depth-Limited Search



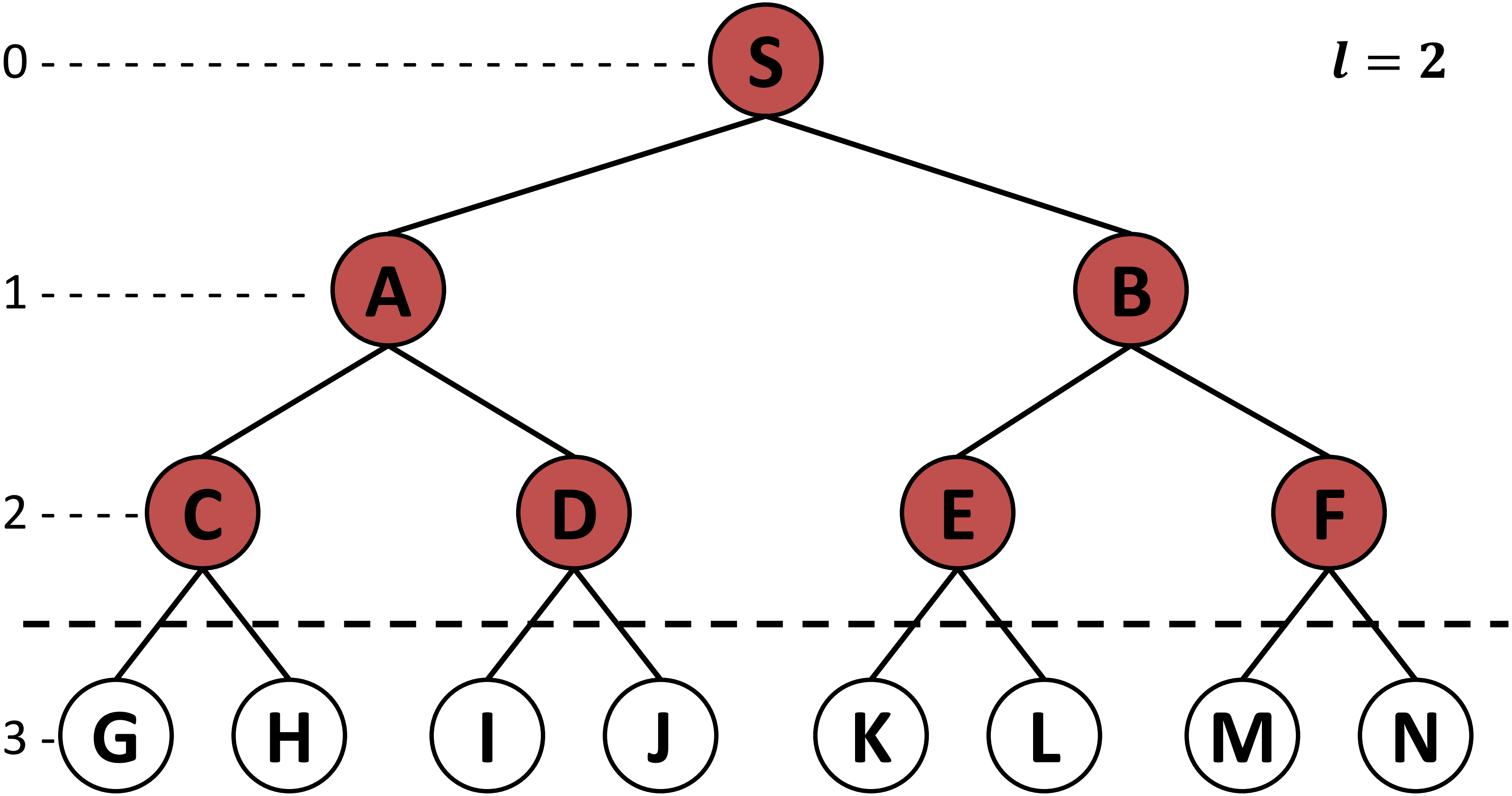
Depth-Limited Search



Depth-Limited Search



Depth-Limited Search



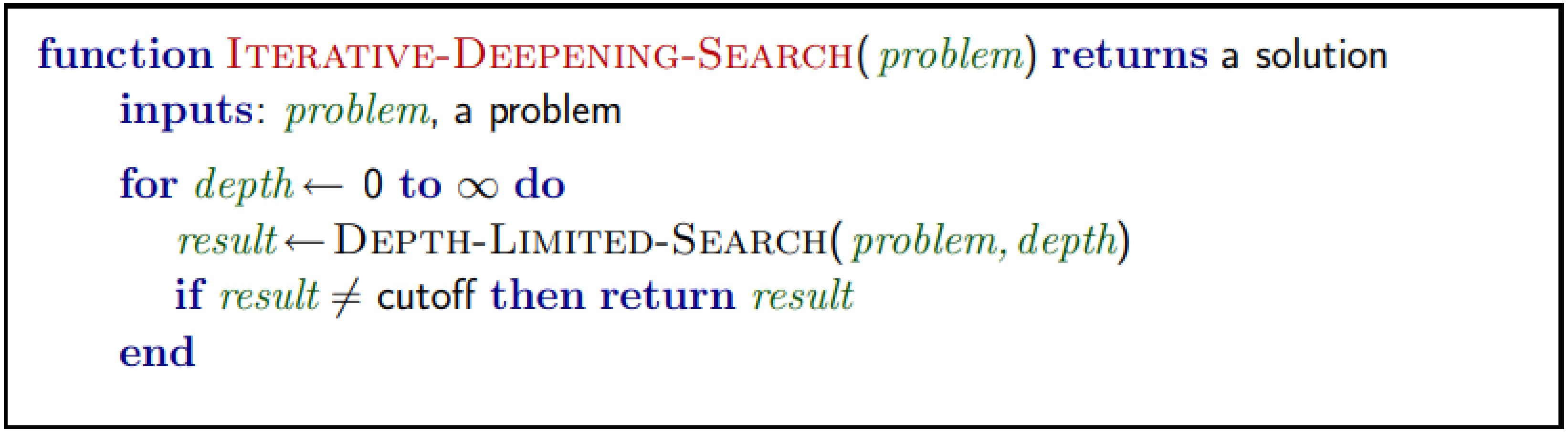
Depth-Limited Search (DLS)

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

- ***Complete?***
  - ***Yes***, if $l \geq d$
- ***Optimal?***
  - ***No***
- ***Time?***
  - ***$O(b^l)$***
- ***Space?***
  - ***$O(bl)$***

** DLS searches in a subspace (local area), but it is NOT a local search algorithm!*

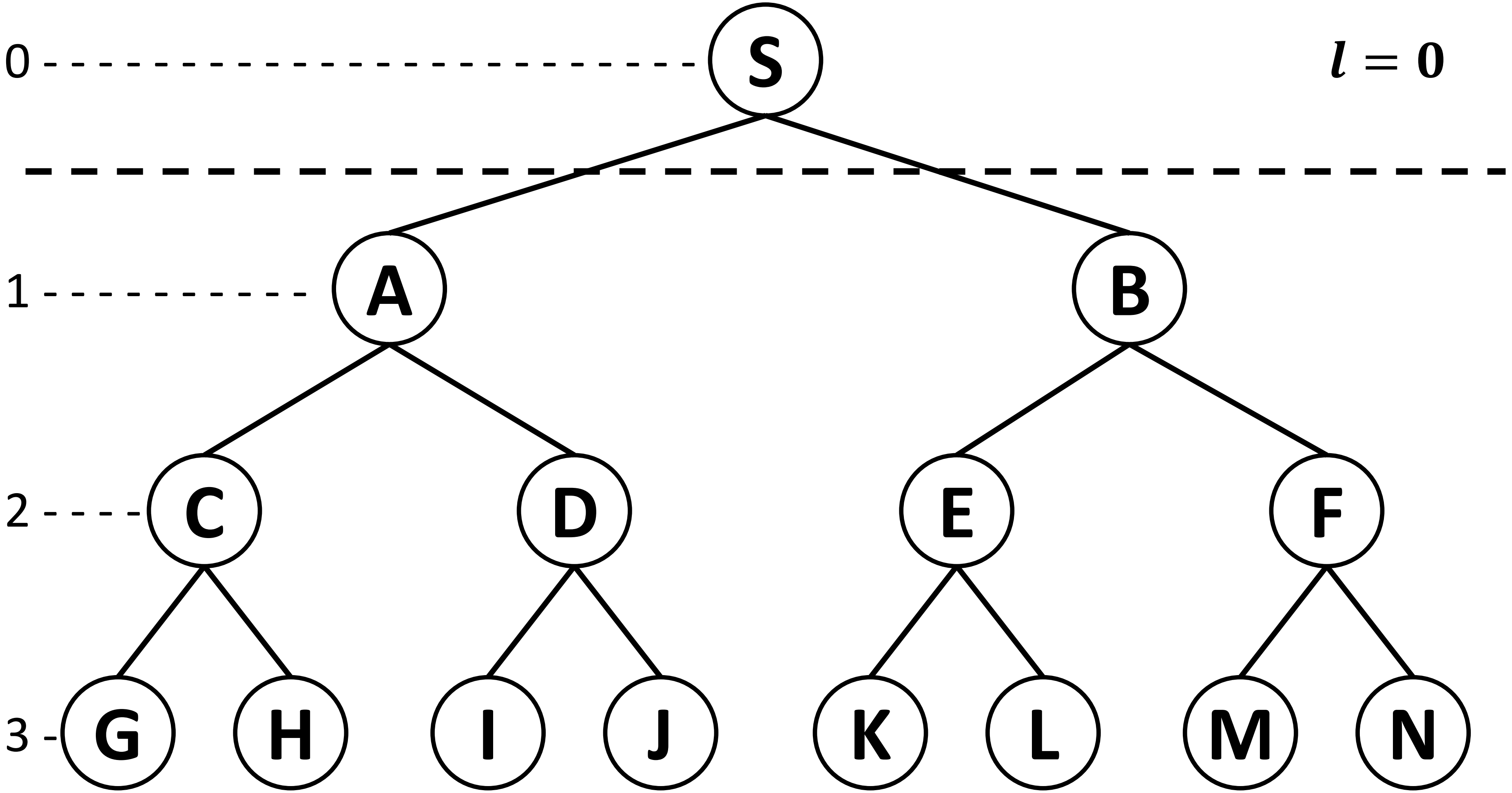
Iterative Deepening Search (IDS)



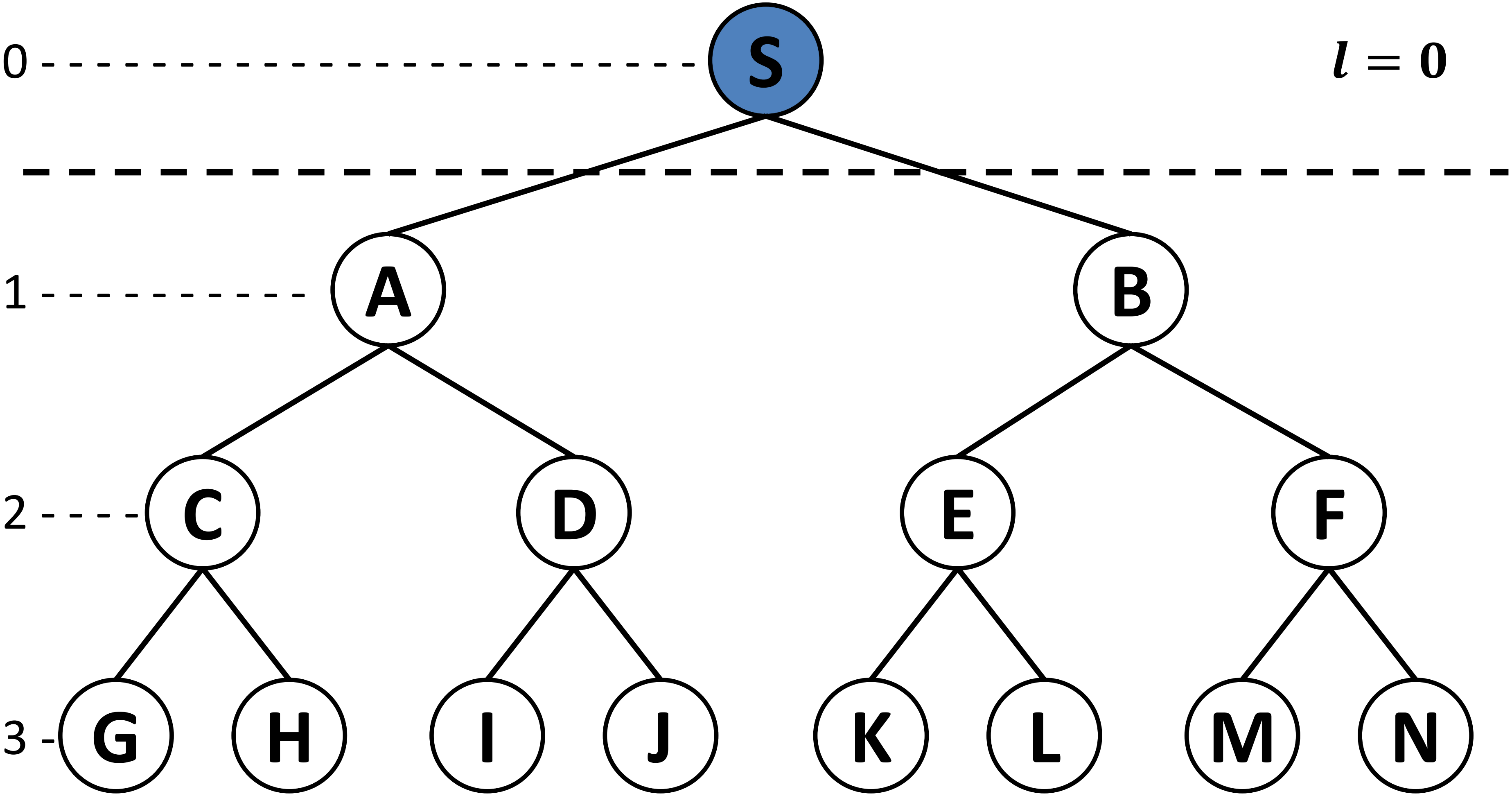
```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution
    inputs: problem, a problem

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

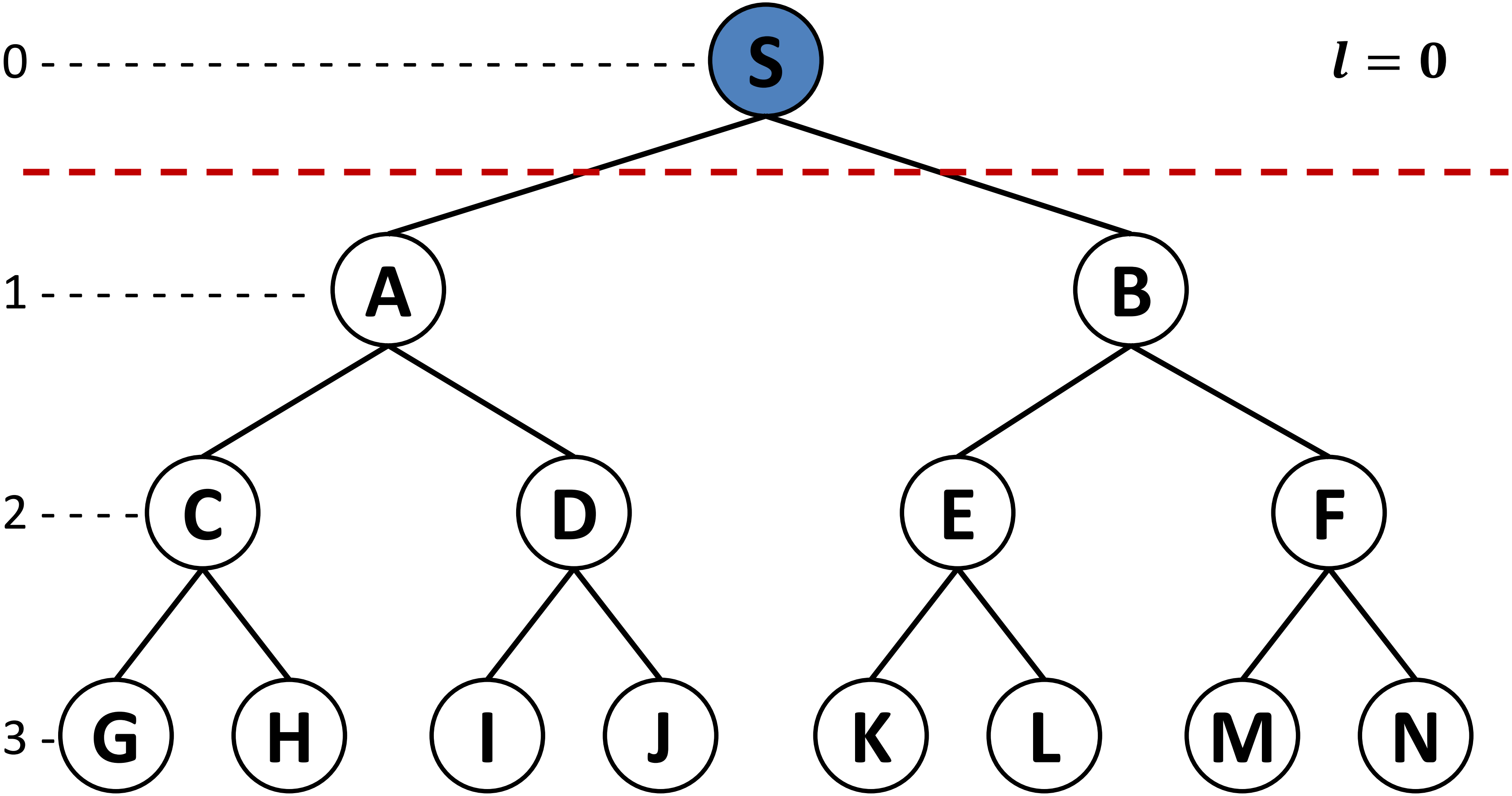
Iterative Deepening Search



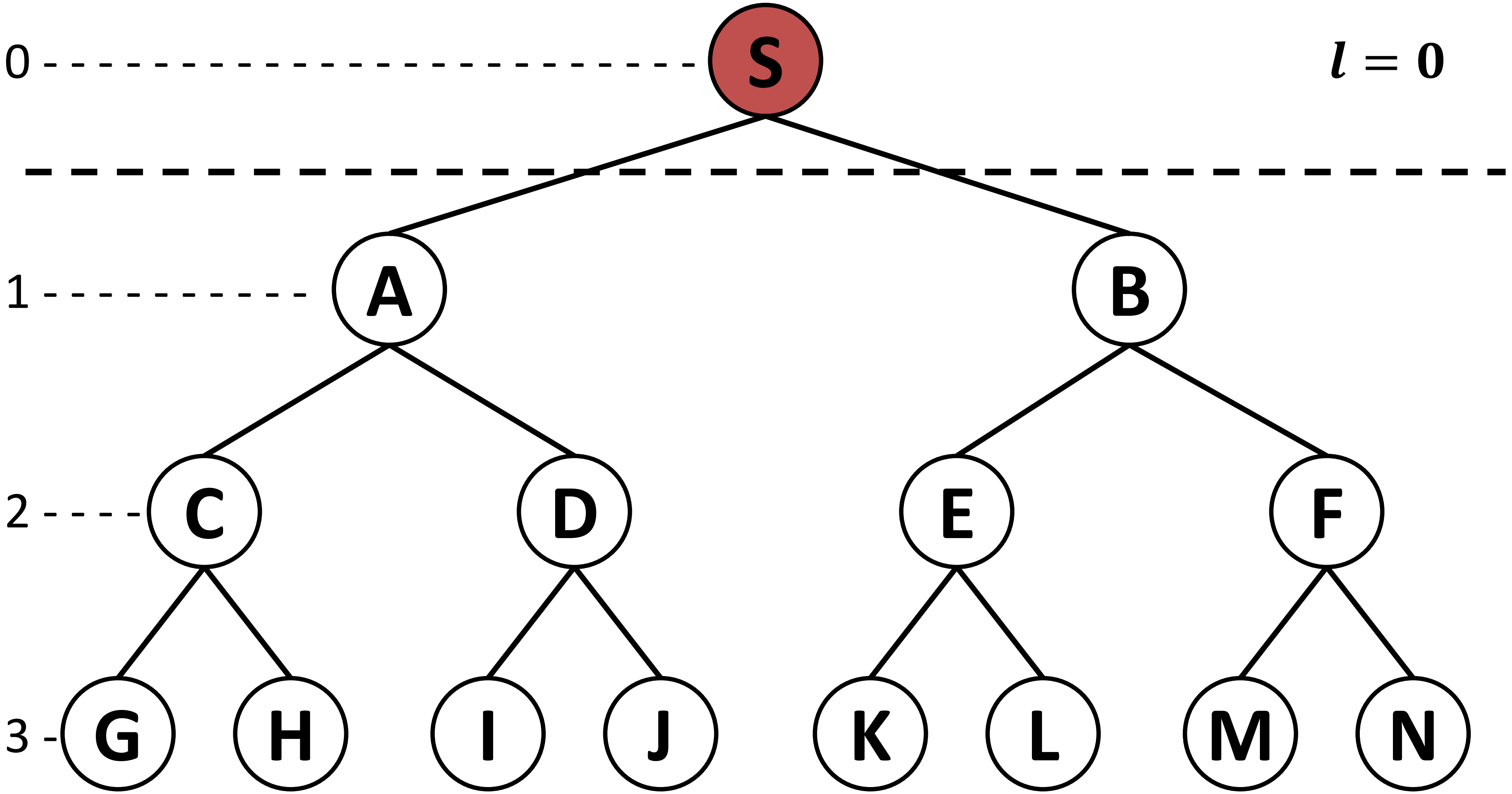
Iterative Deepening Search



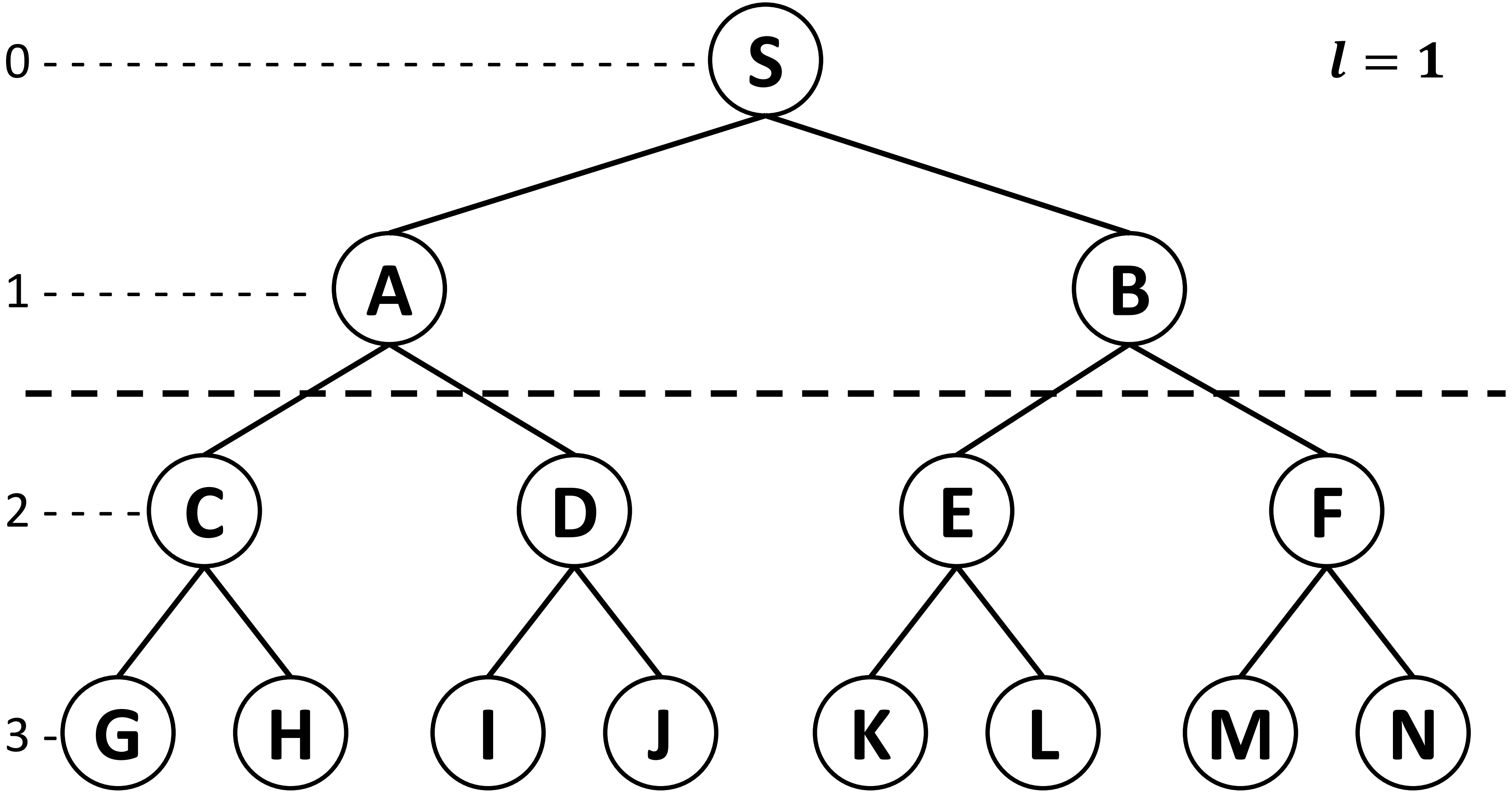
Iterative Deepening Search



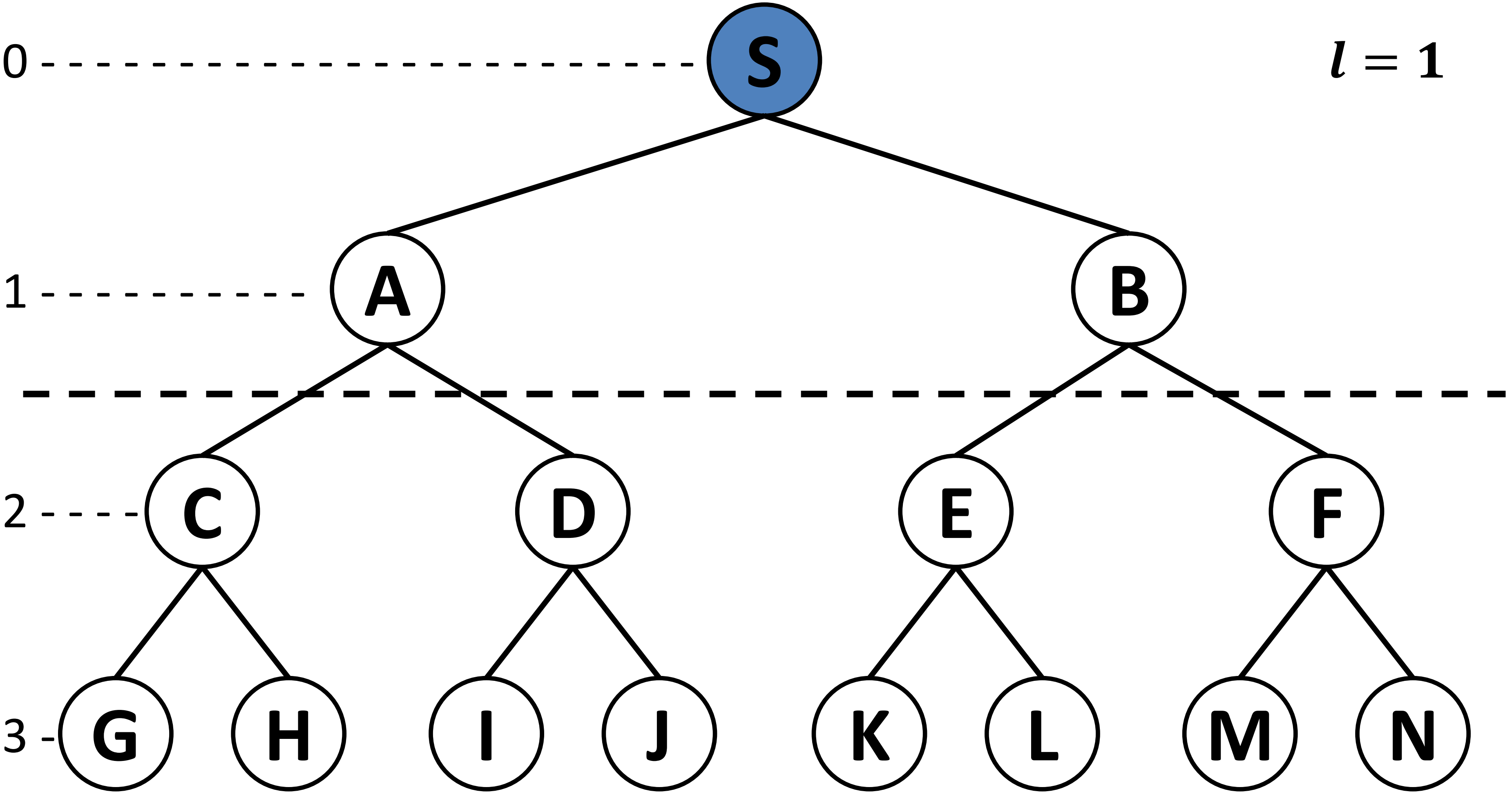
Iterative Deepening Search



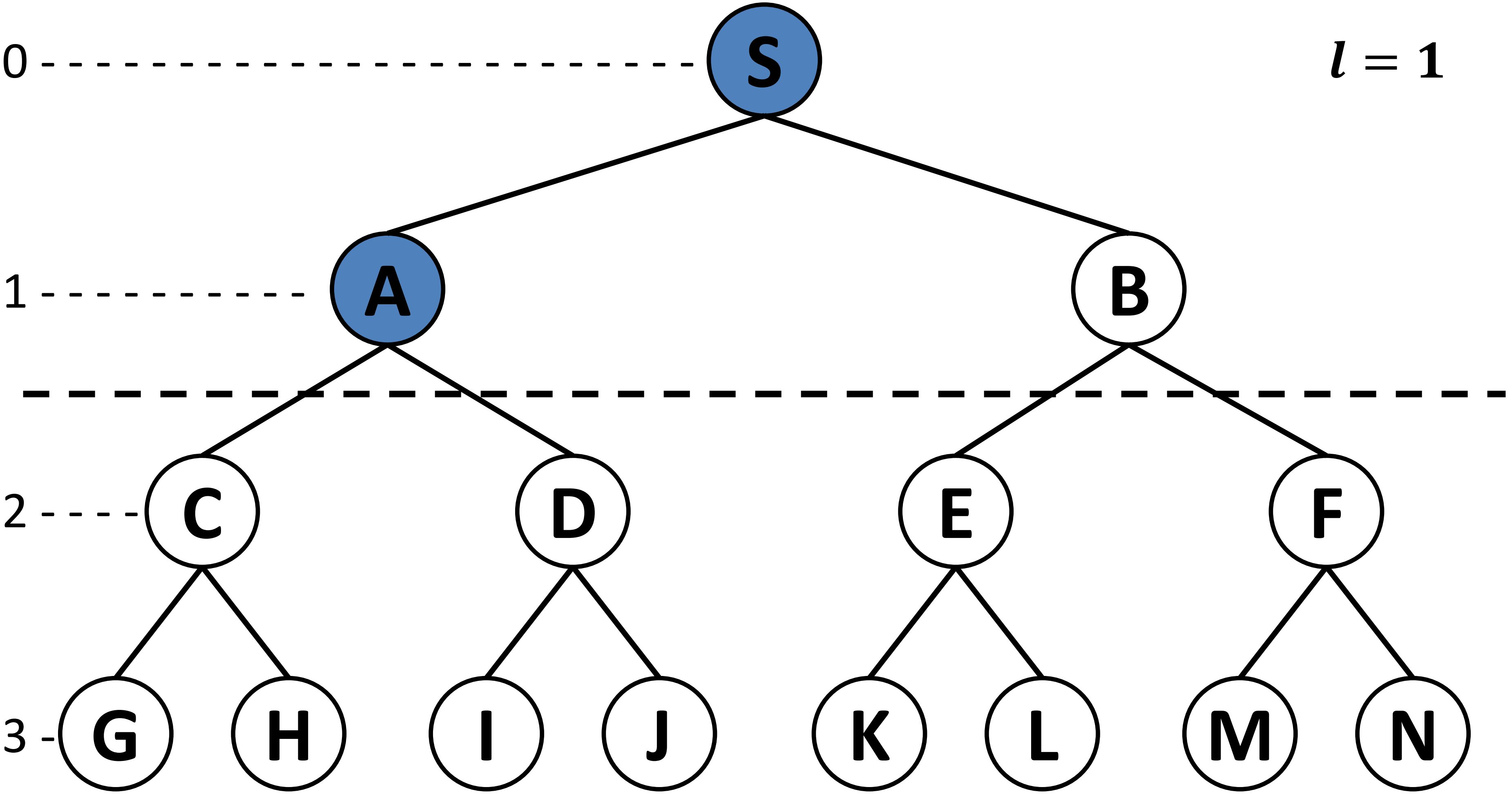
Iterative Deepening Search



Iterative Deepening Search

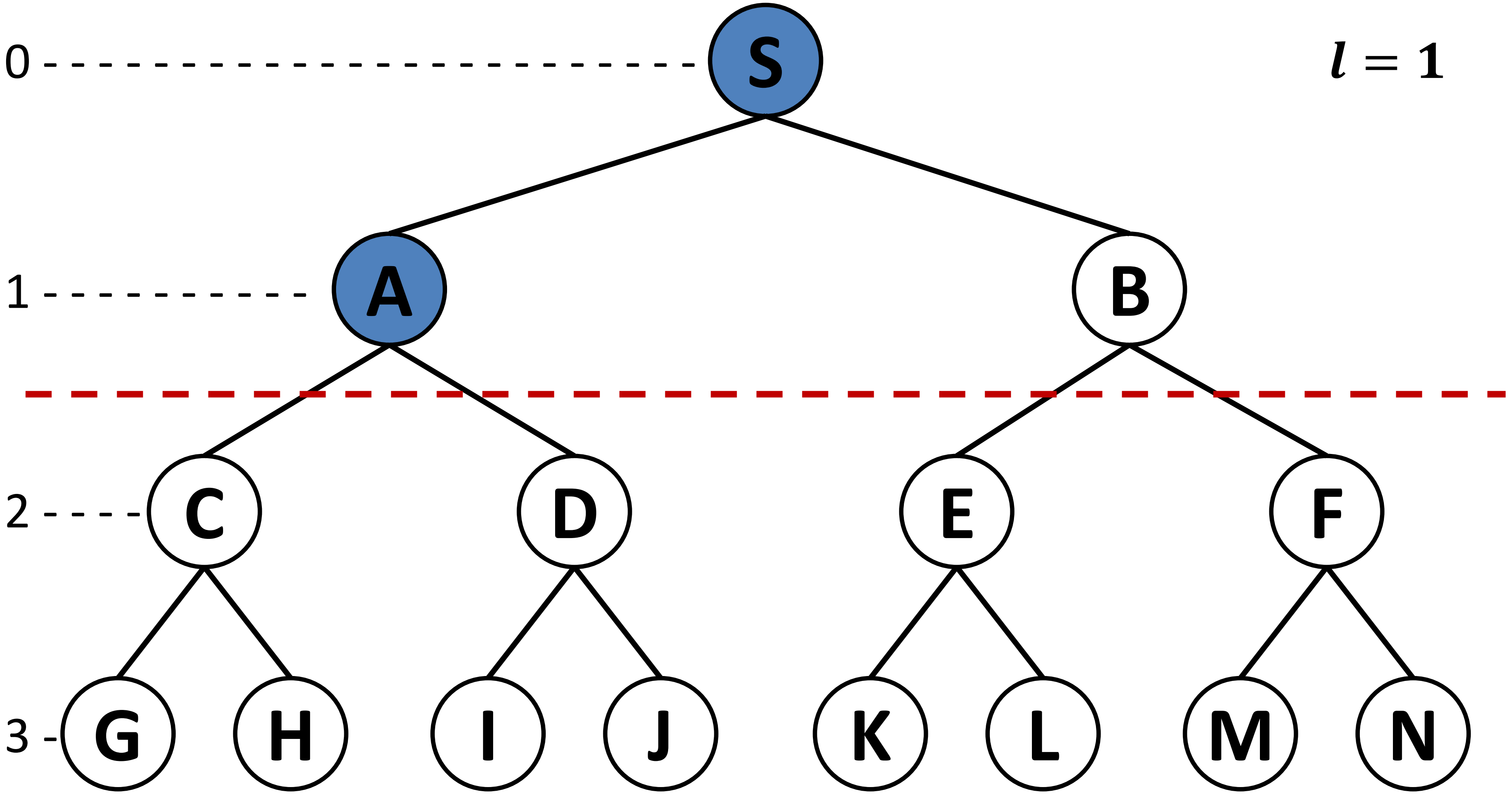


Iterative Deepening Search

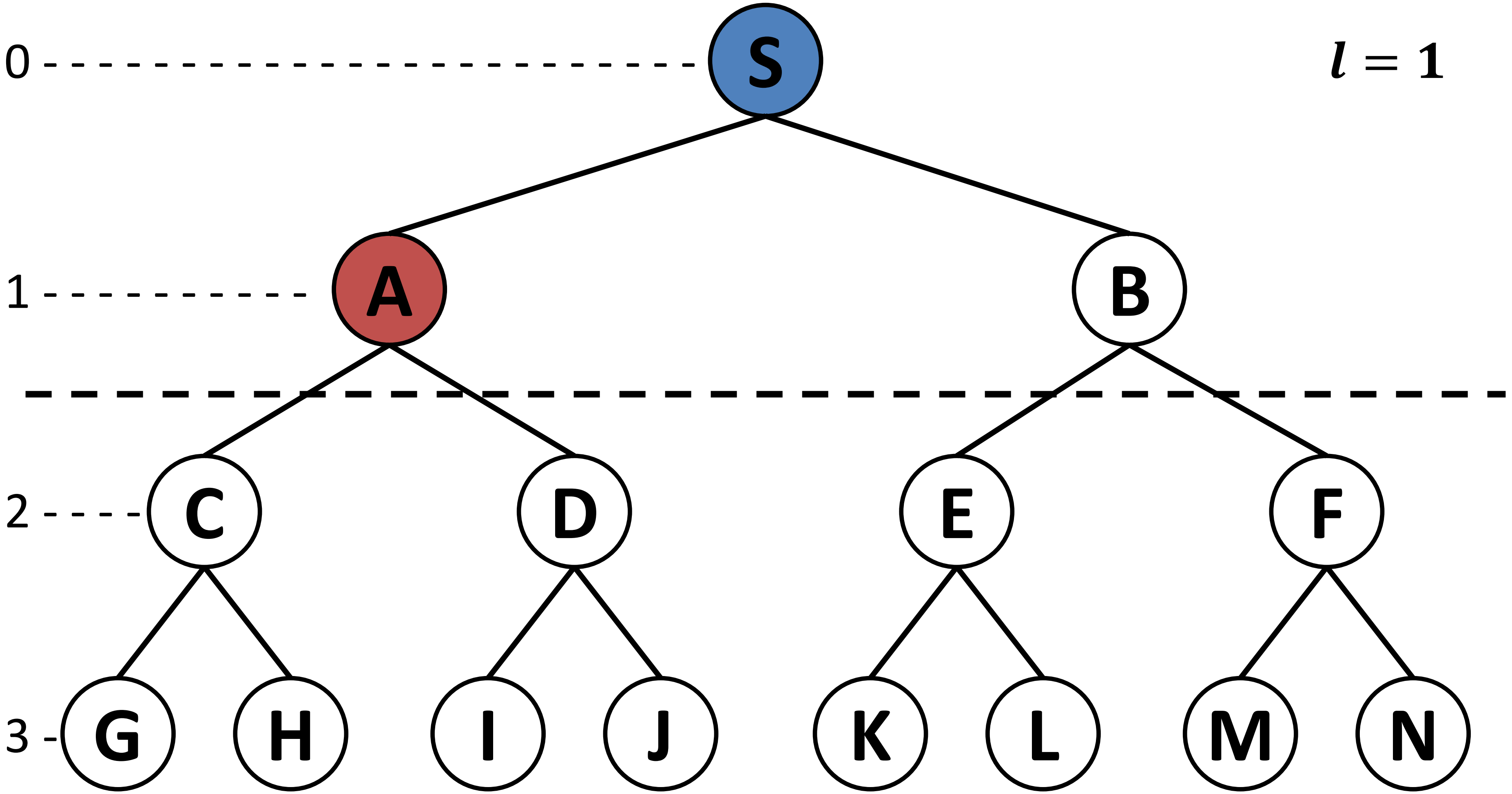


$l = 1$

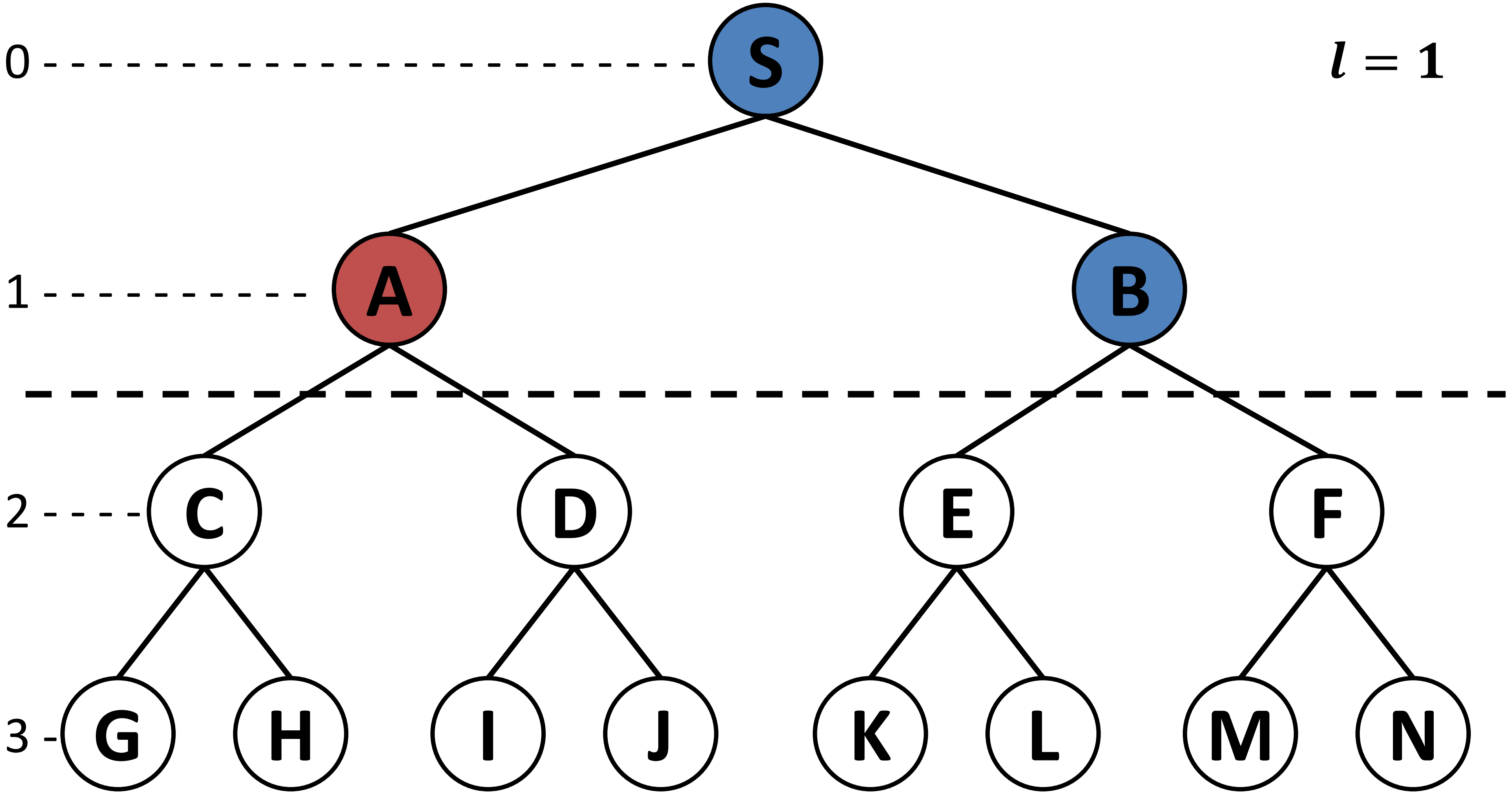
Iterative Deepening Search



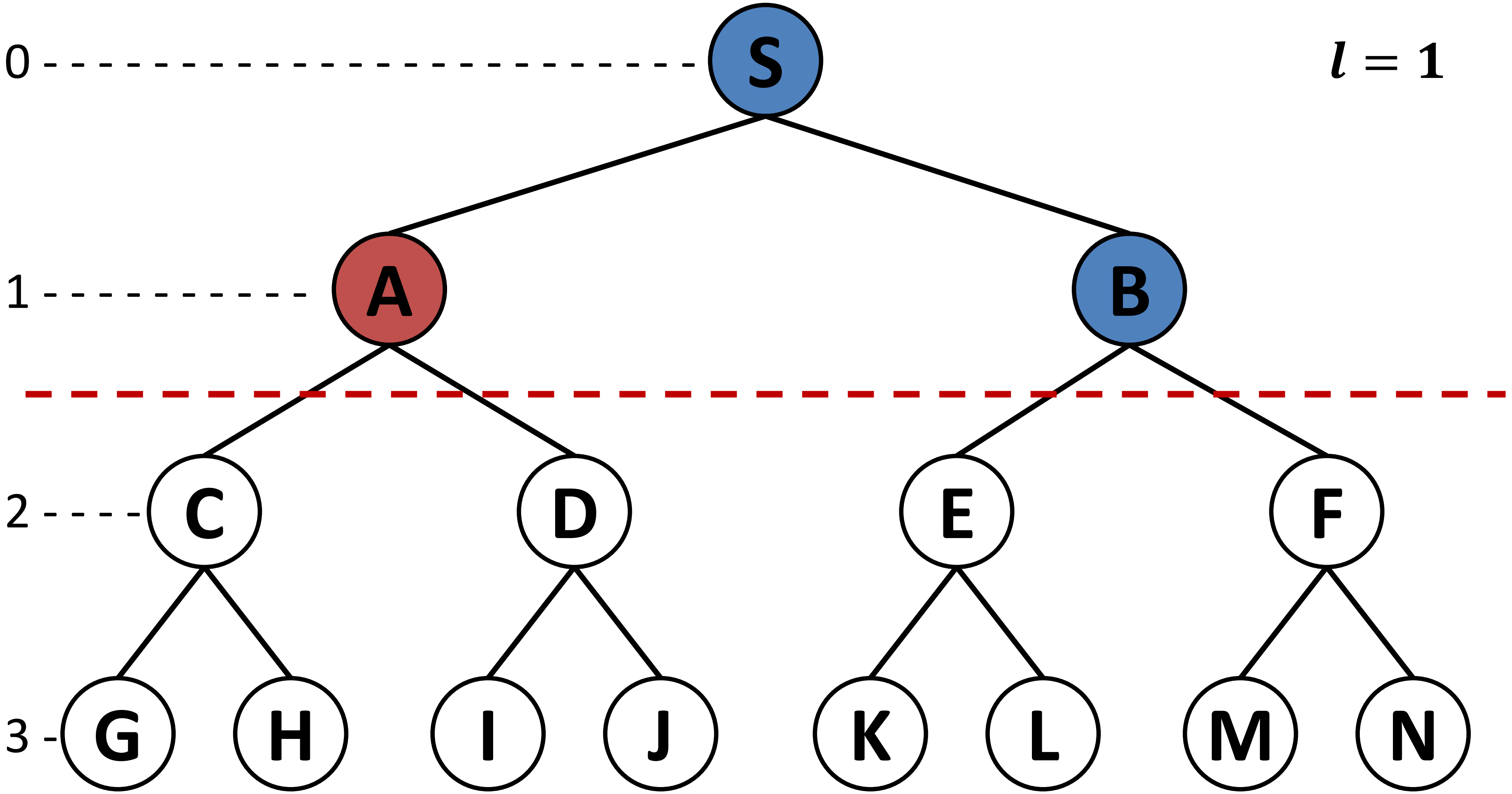
Iterative Deepening Search



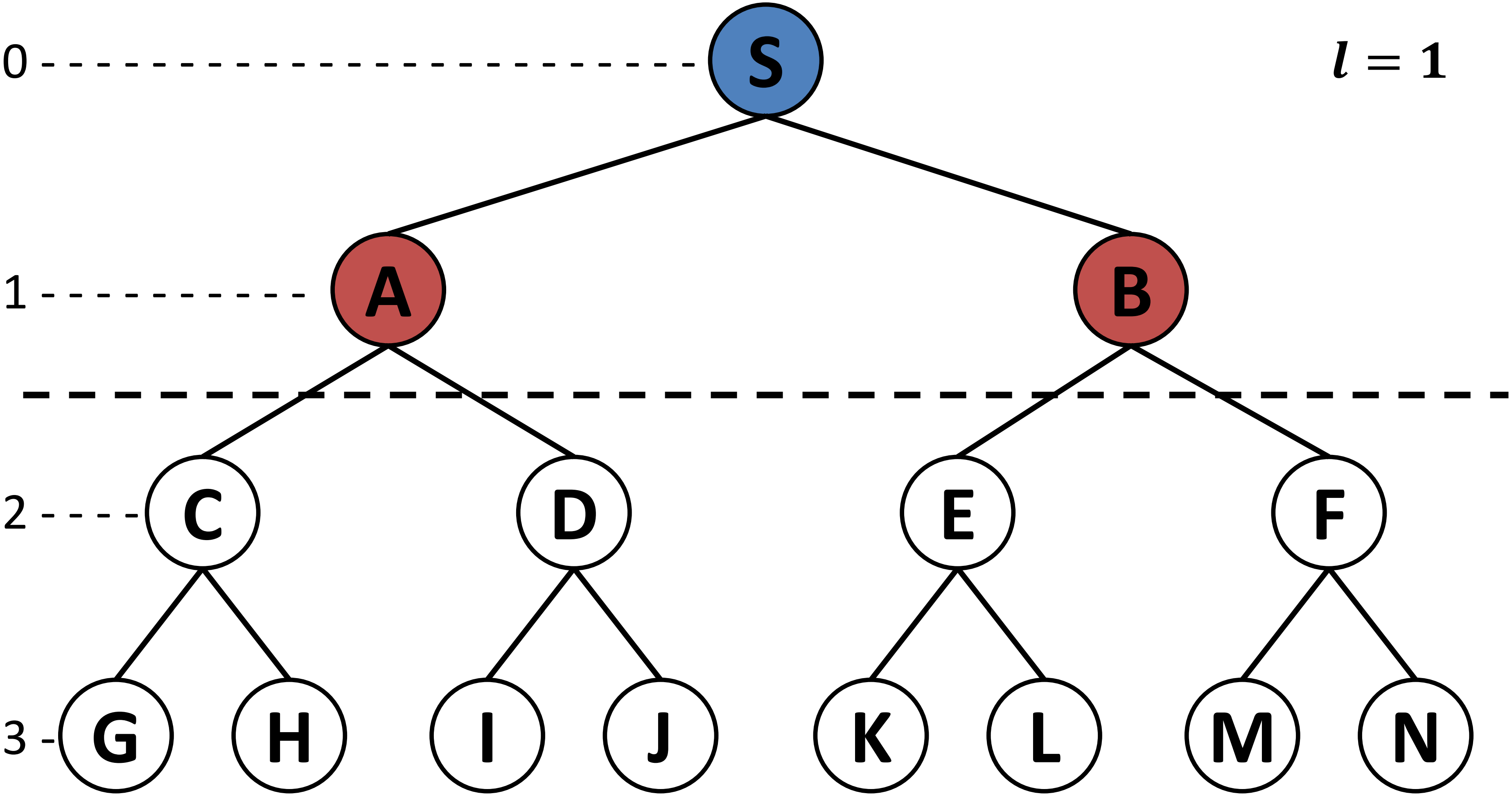
Iterative Deepening Search



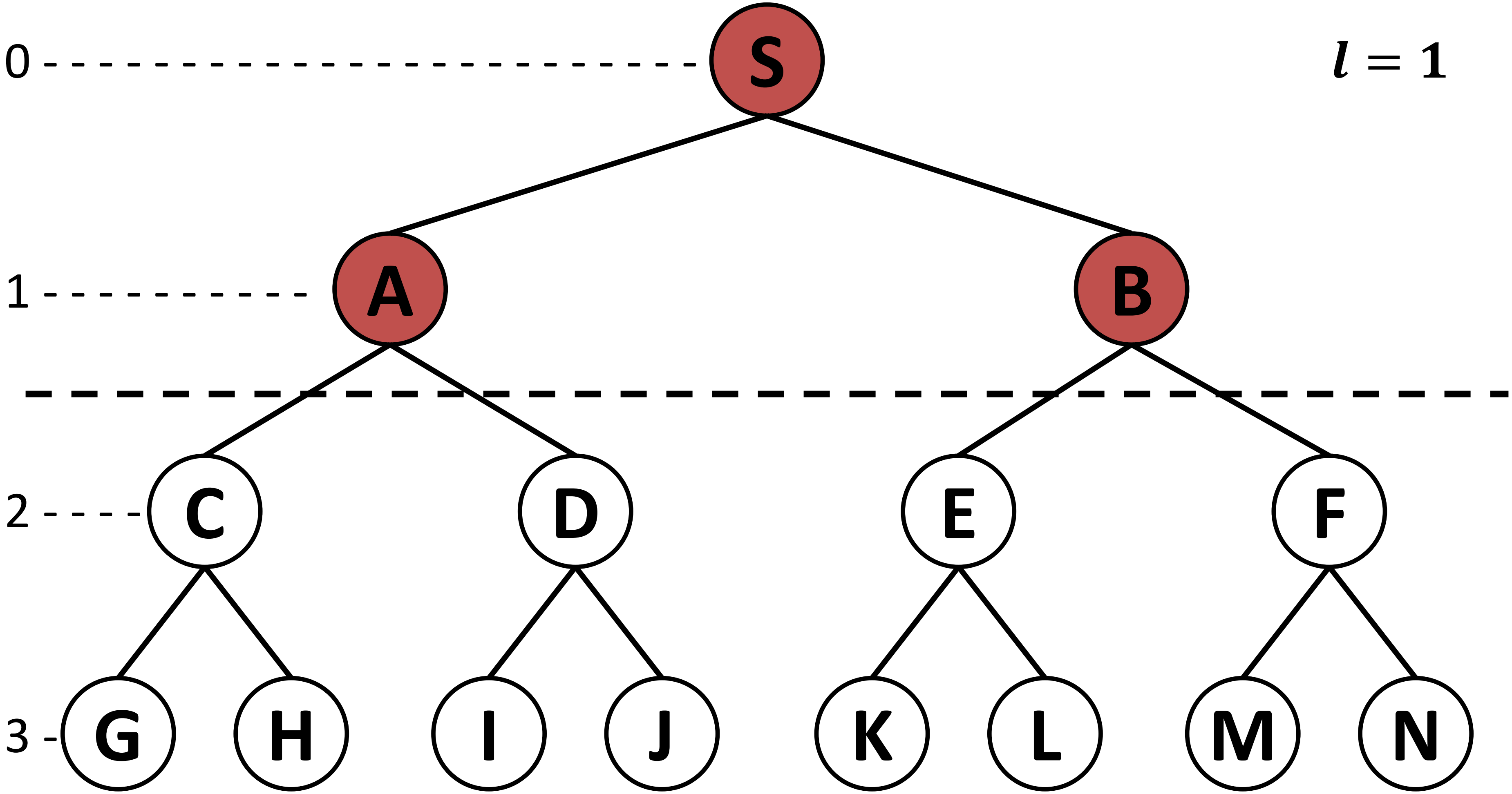
Iterative Deepening Search



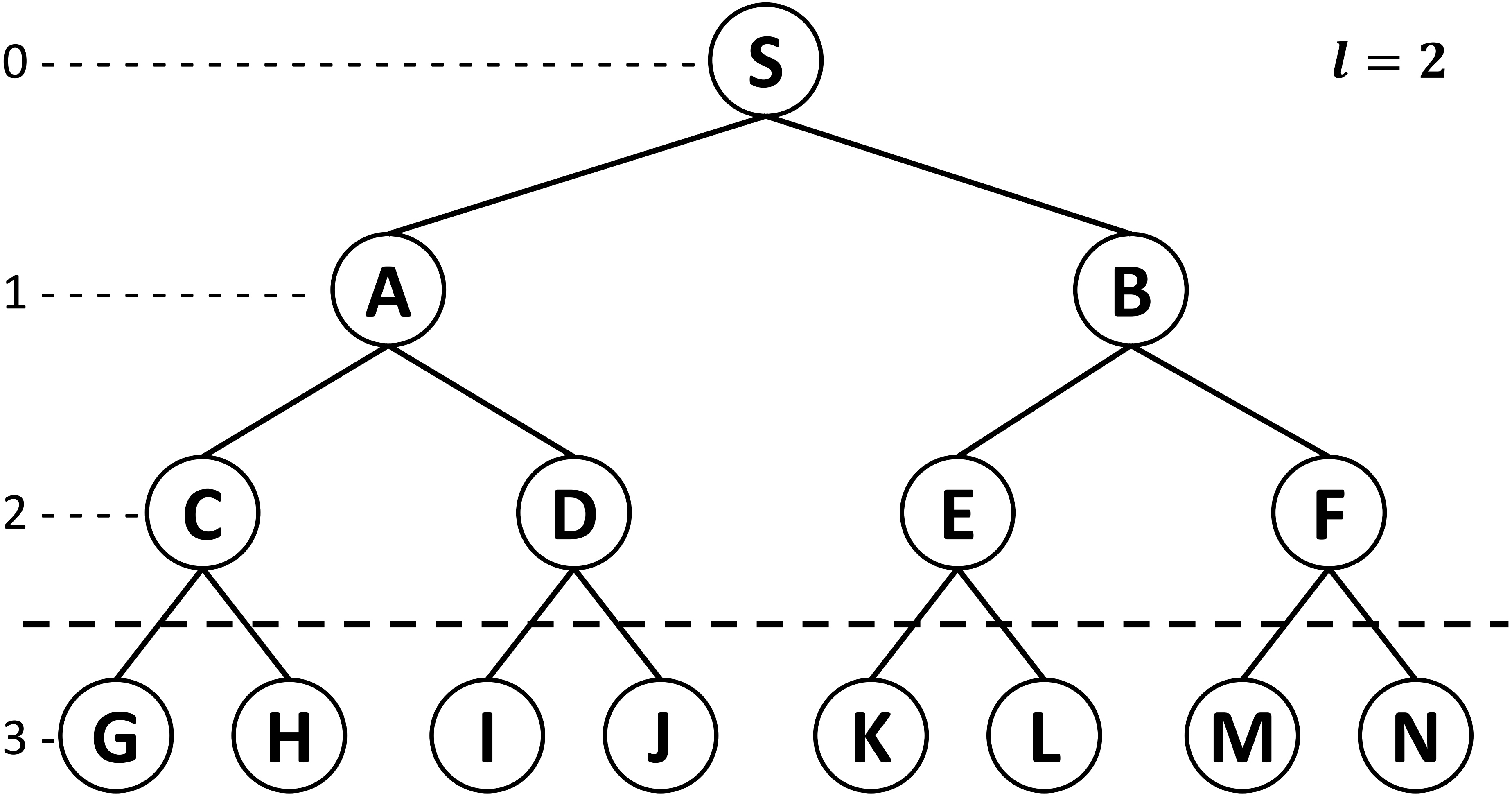
Iterative Deepening Search



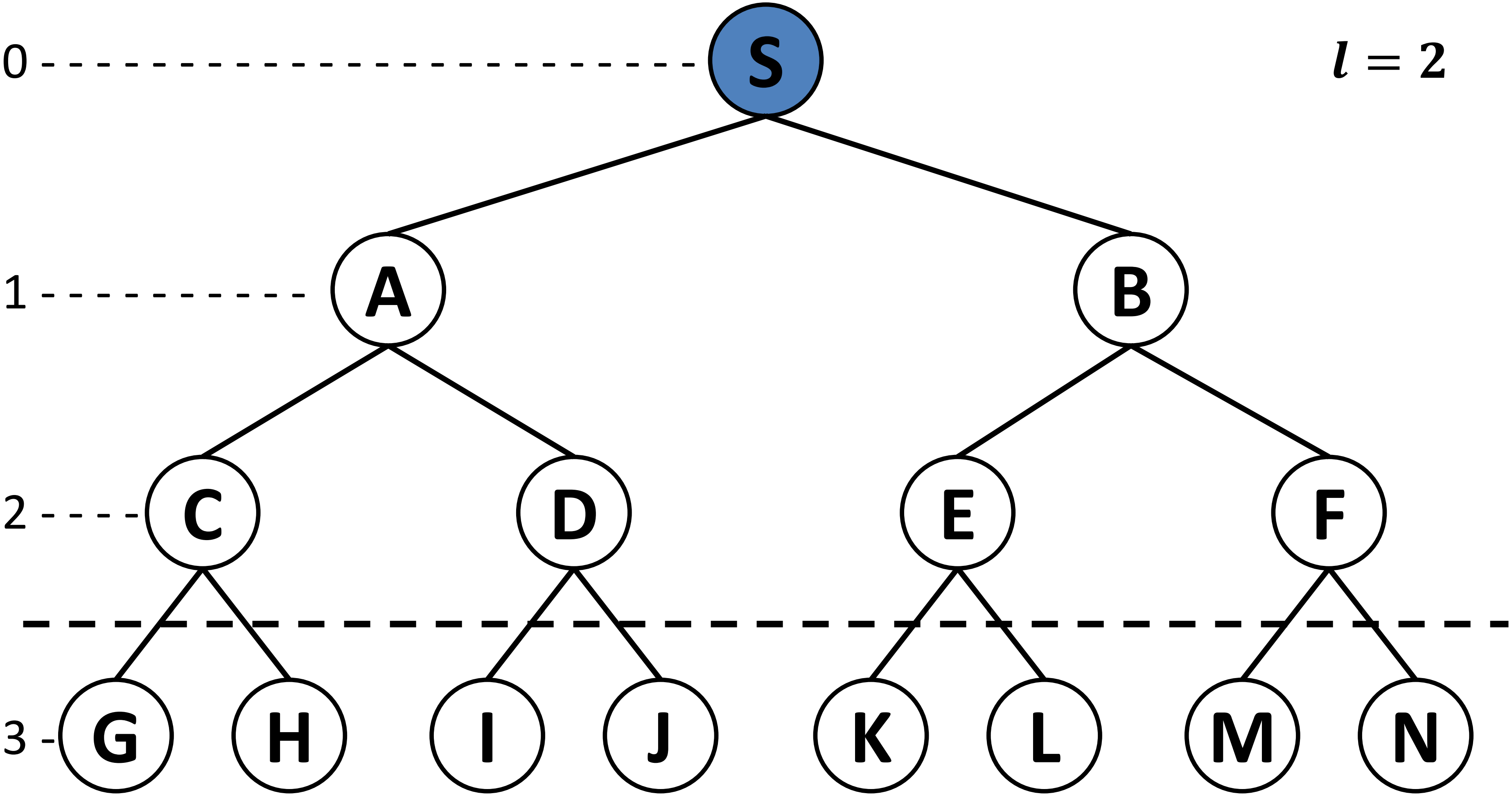
Iterative Deepening Search



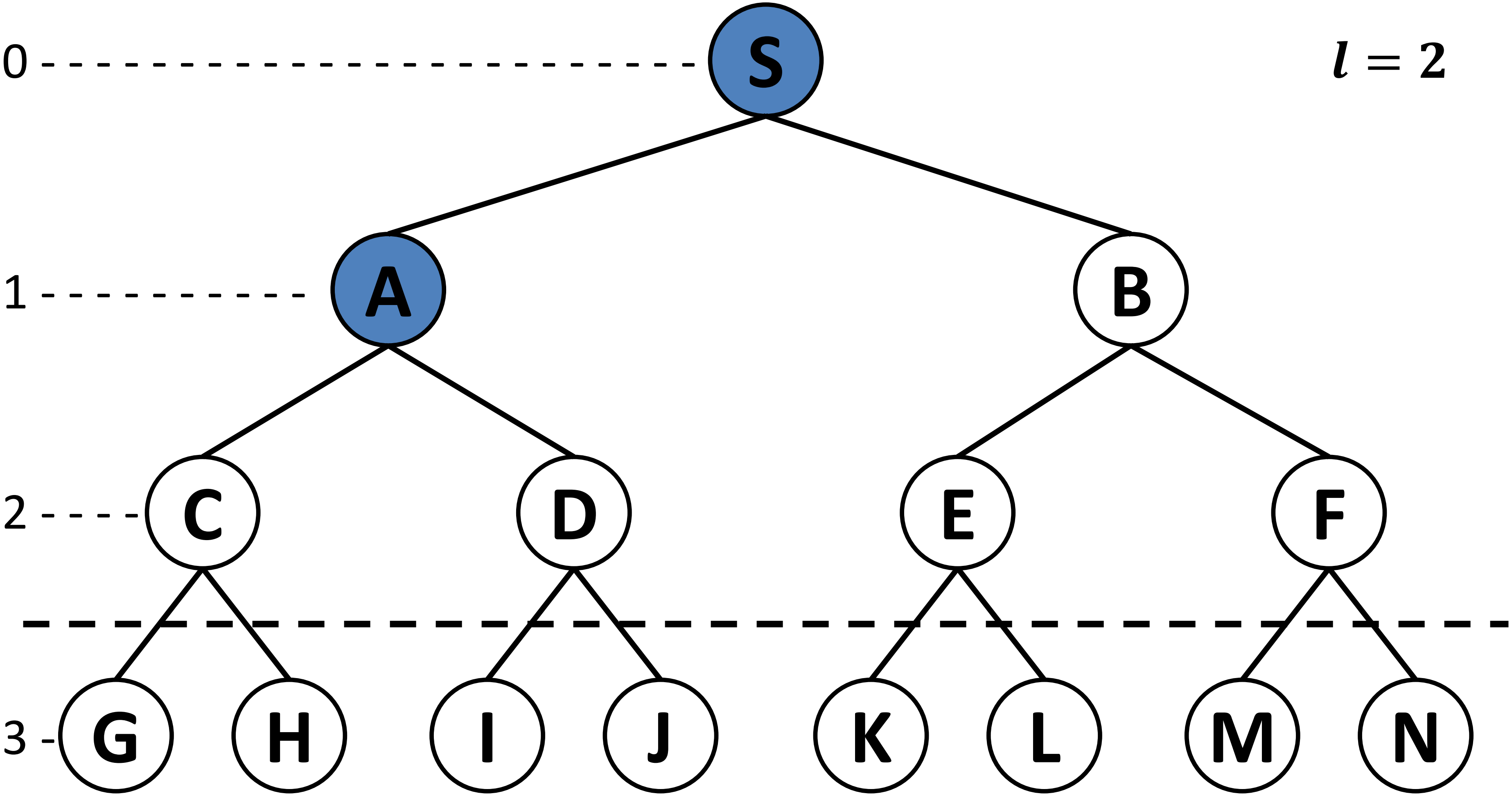
Iterative Deepening Search



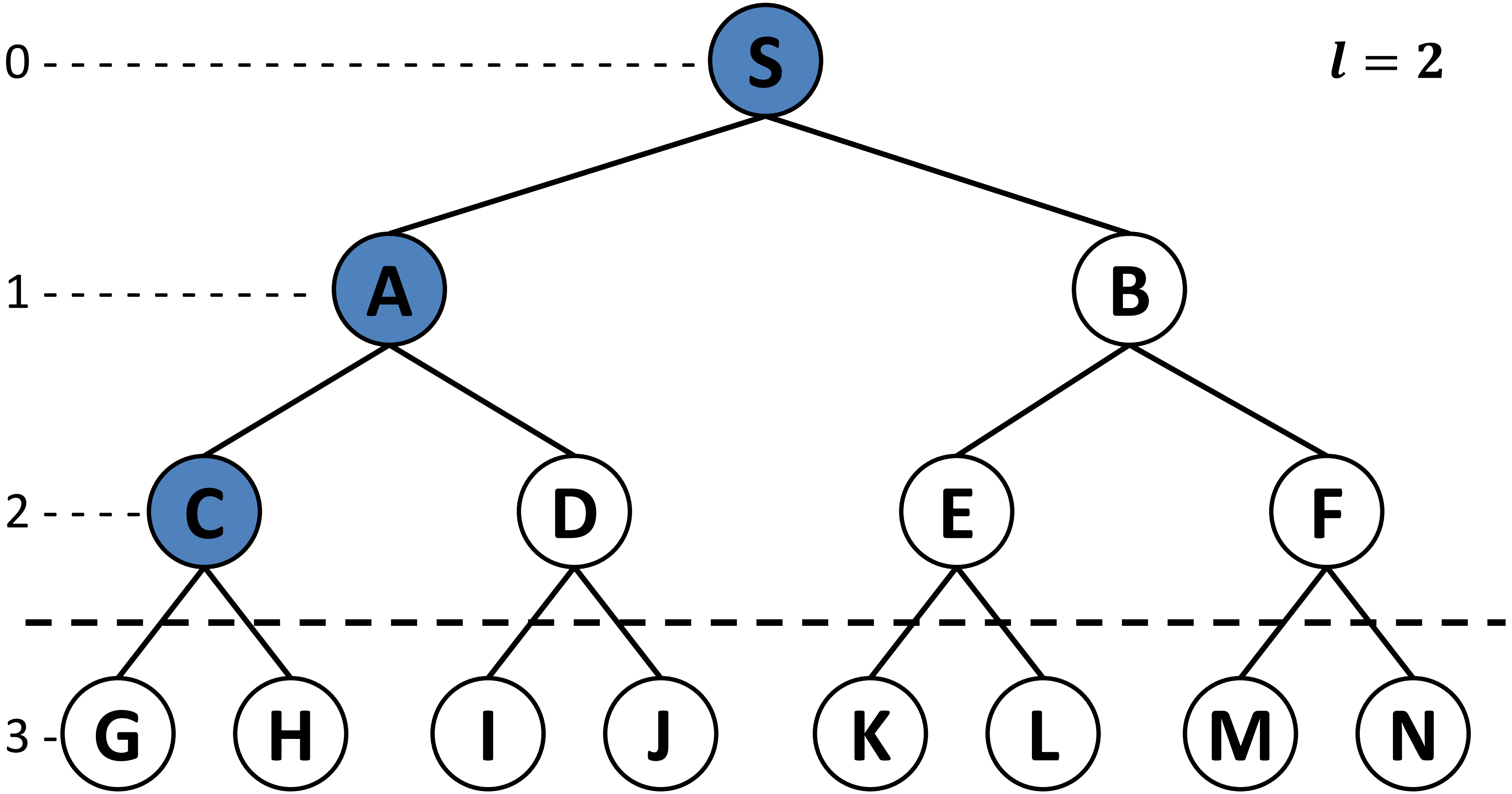
Iterative Deepening Search



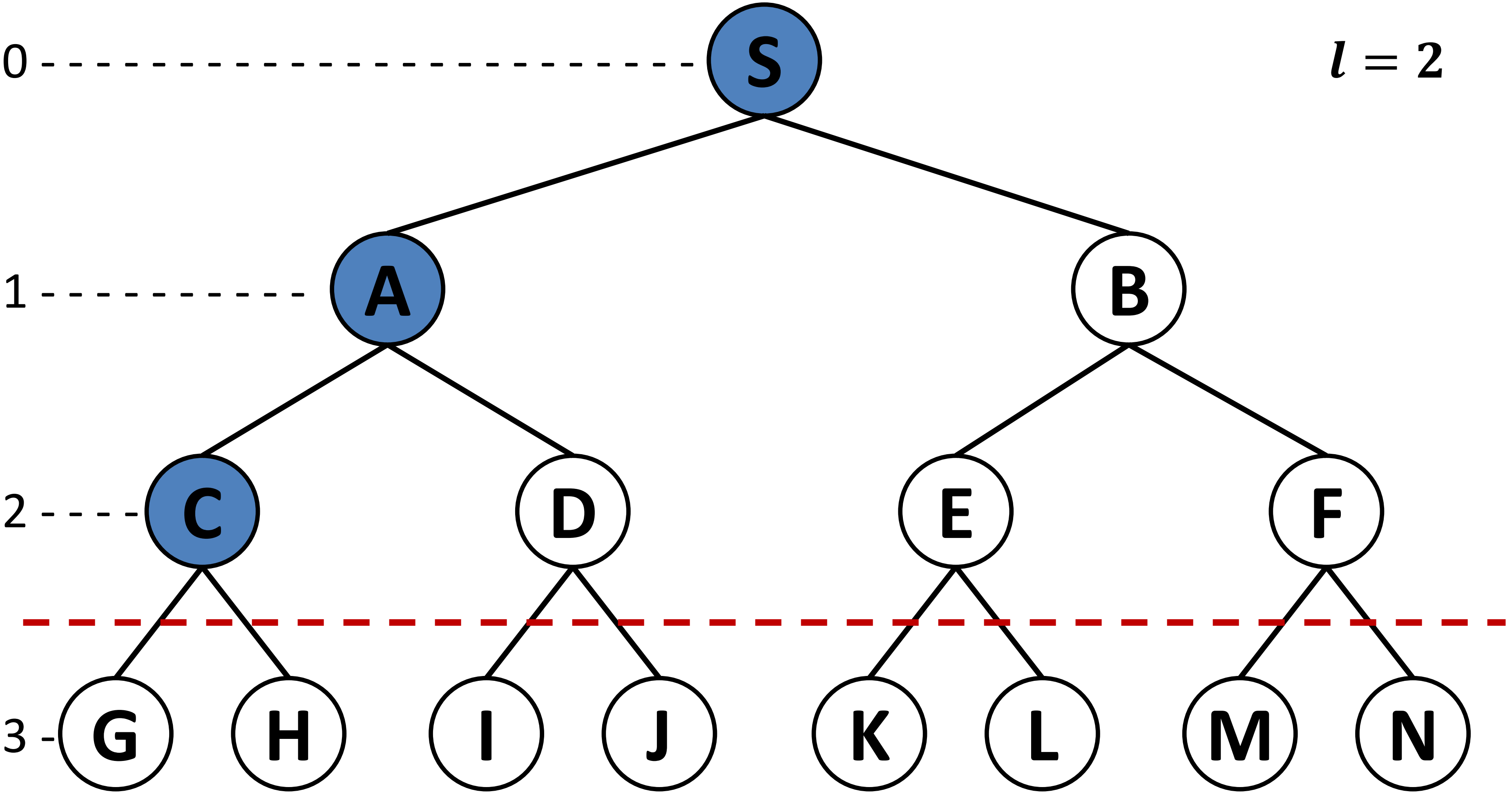
Iterative Deepening Search



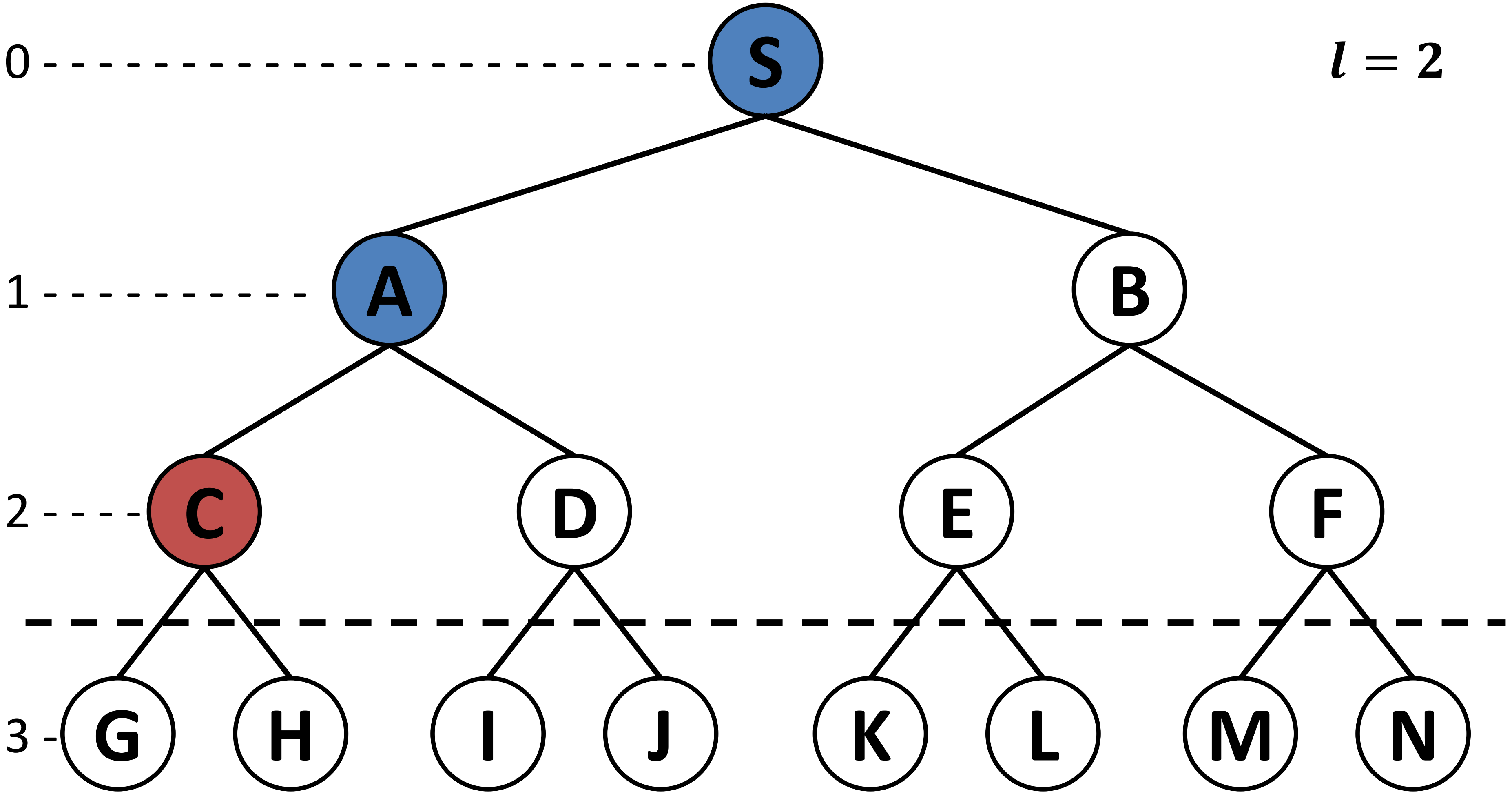
Iterative Deepening Search



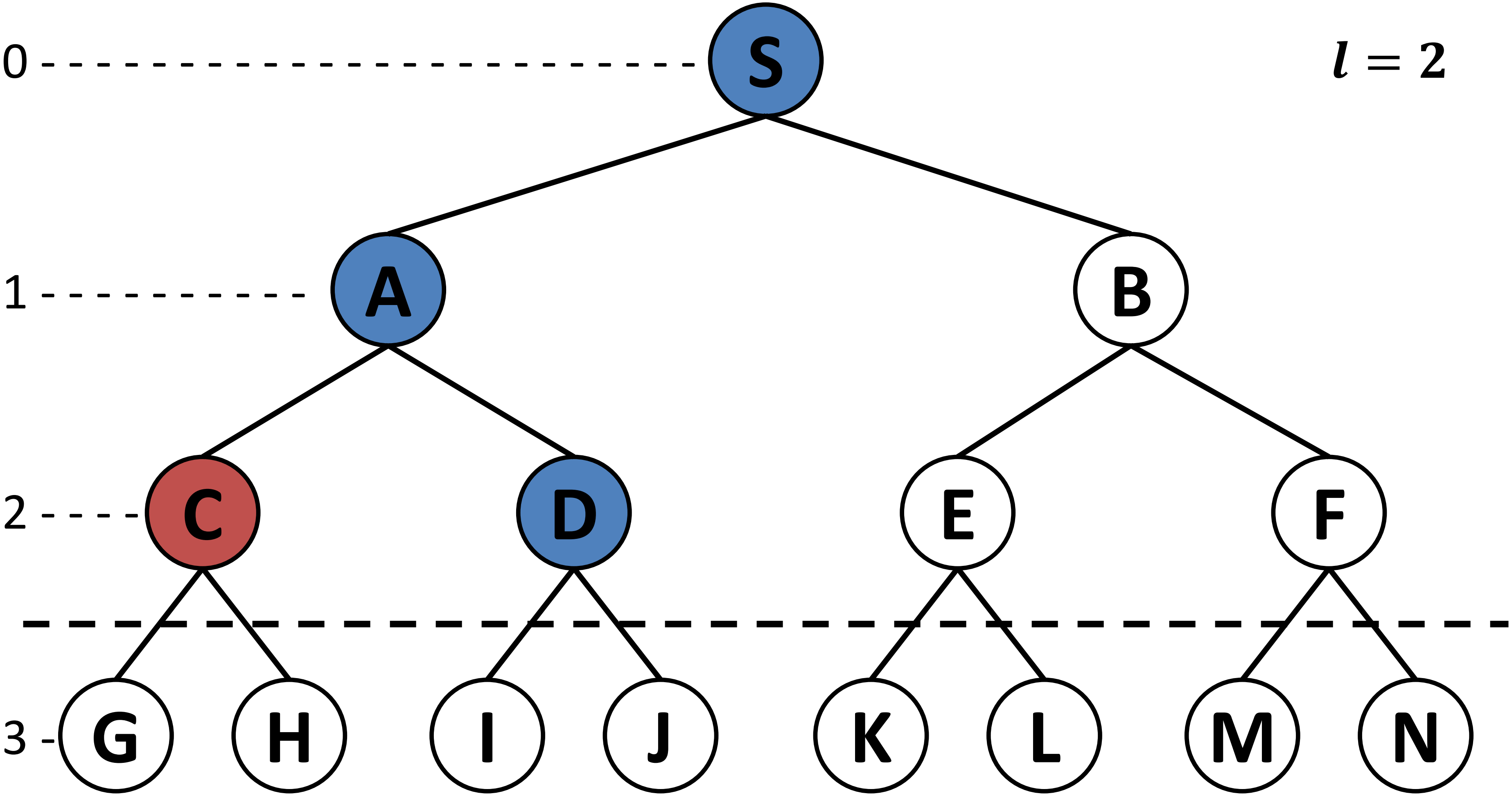
Iterative Deepening Search



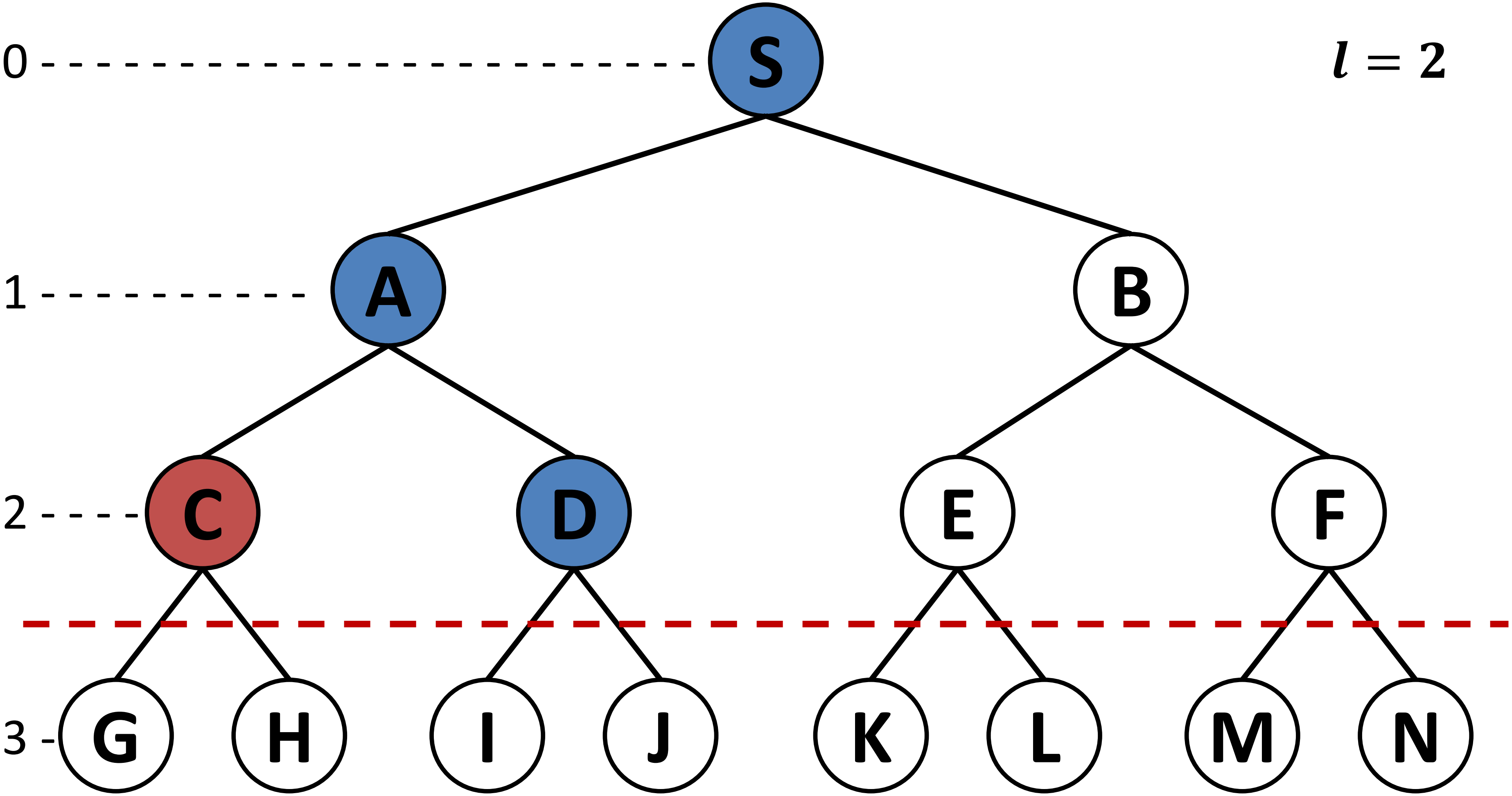
Iterative Deepening Search



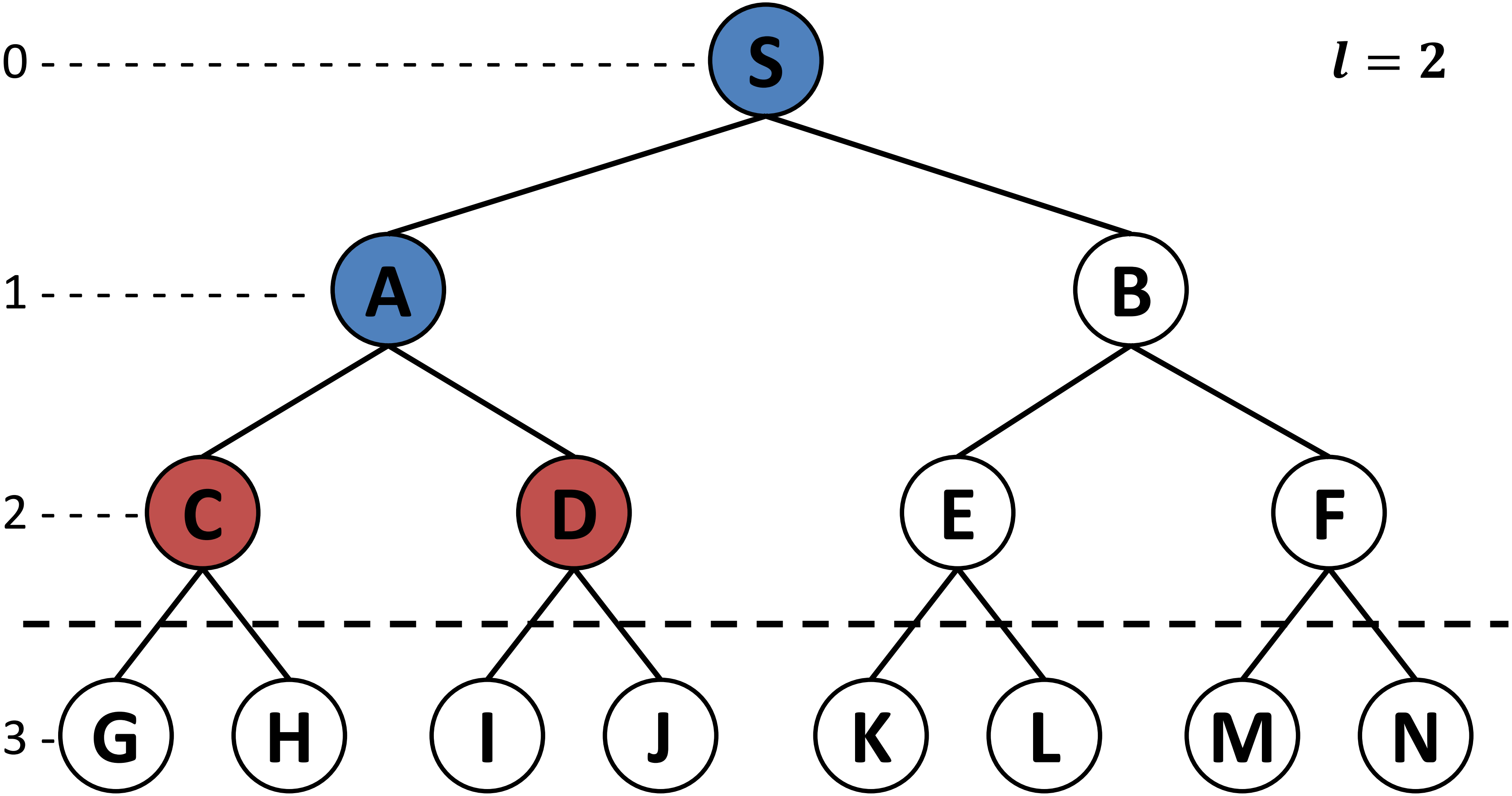
Iterative Deepening Search



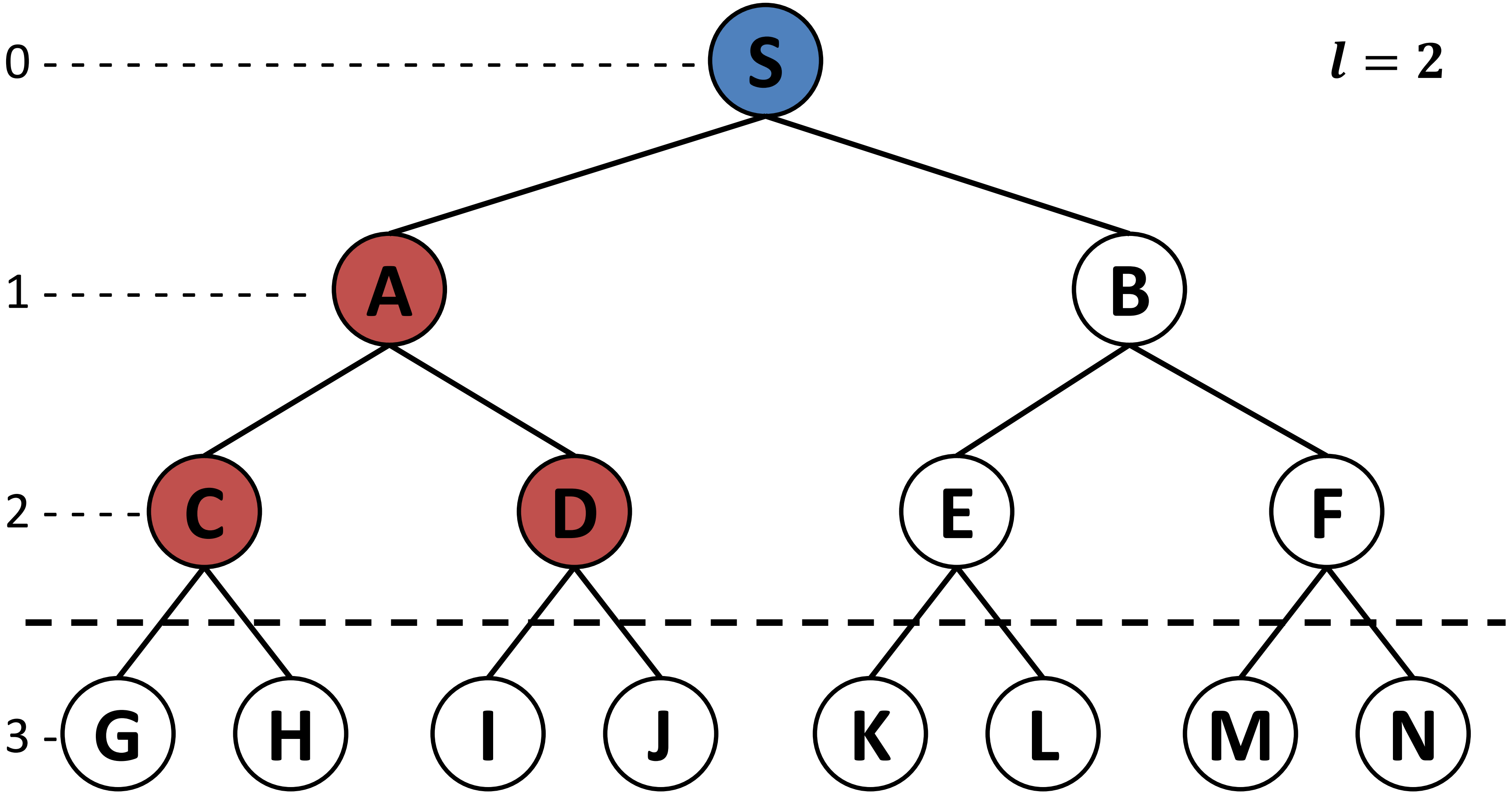
Iterative Deepening Search



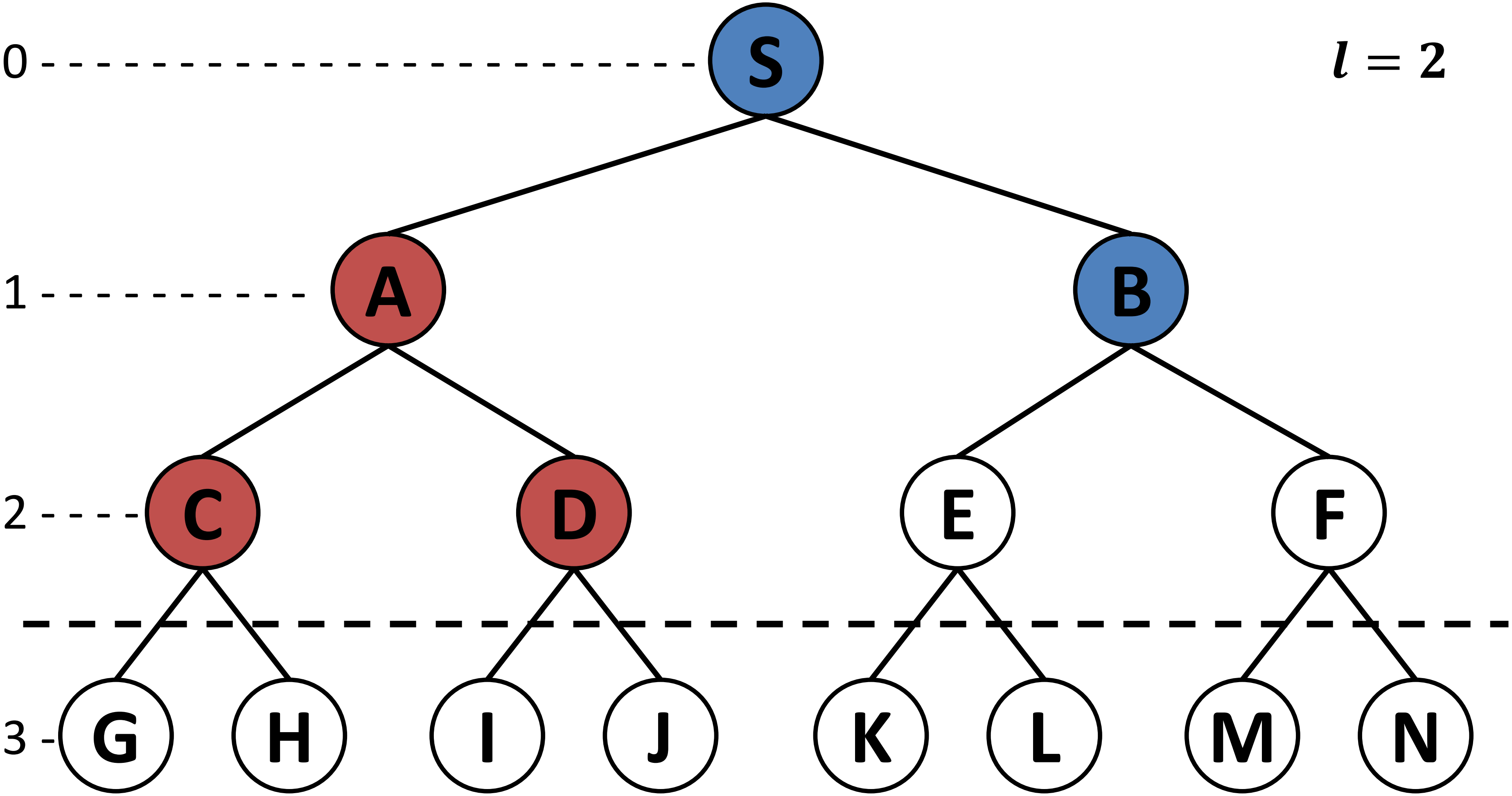
Iterative Deepening Search



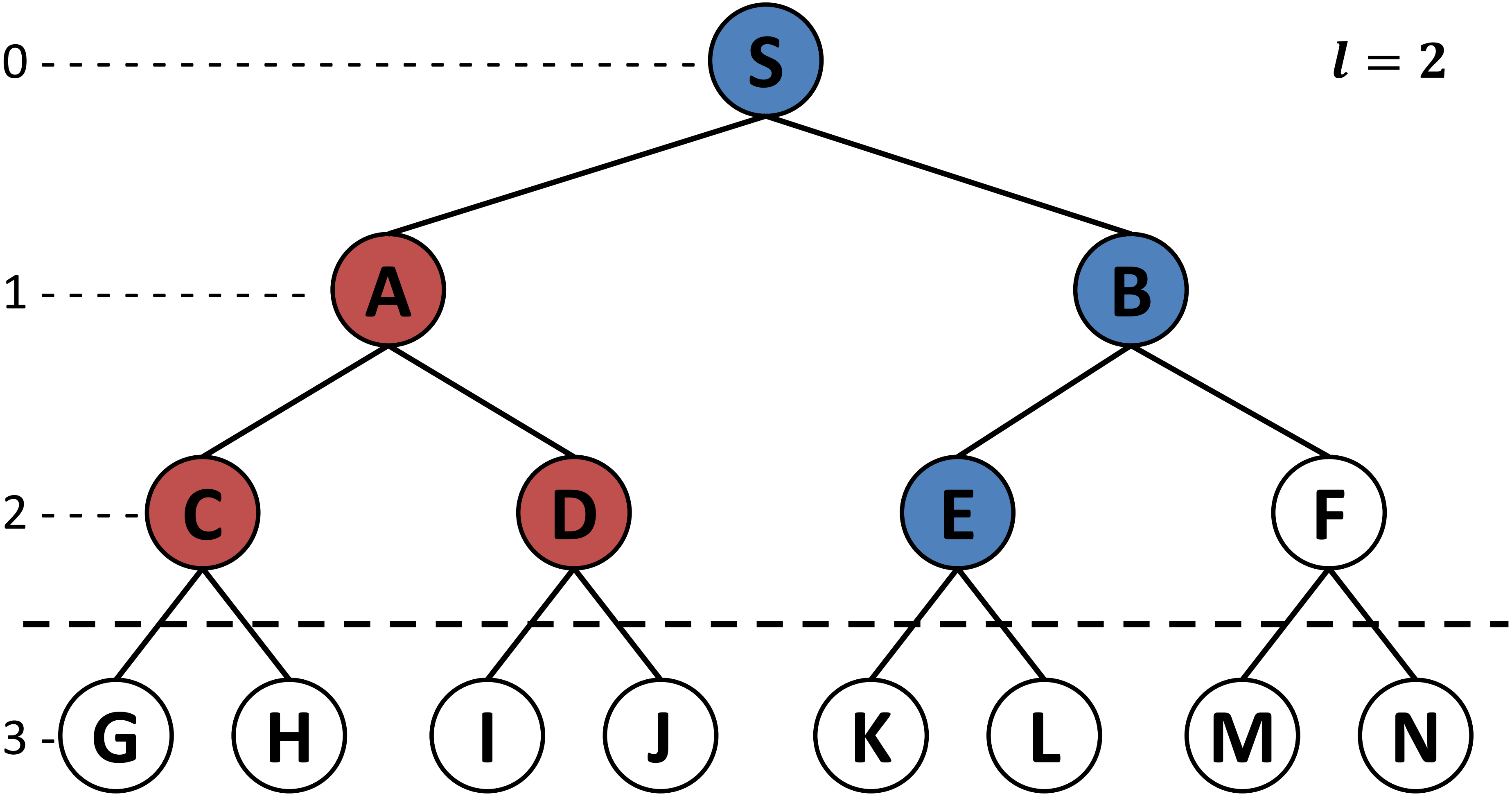
Iterative Deepening Search



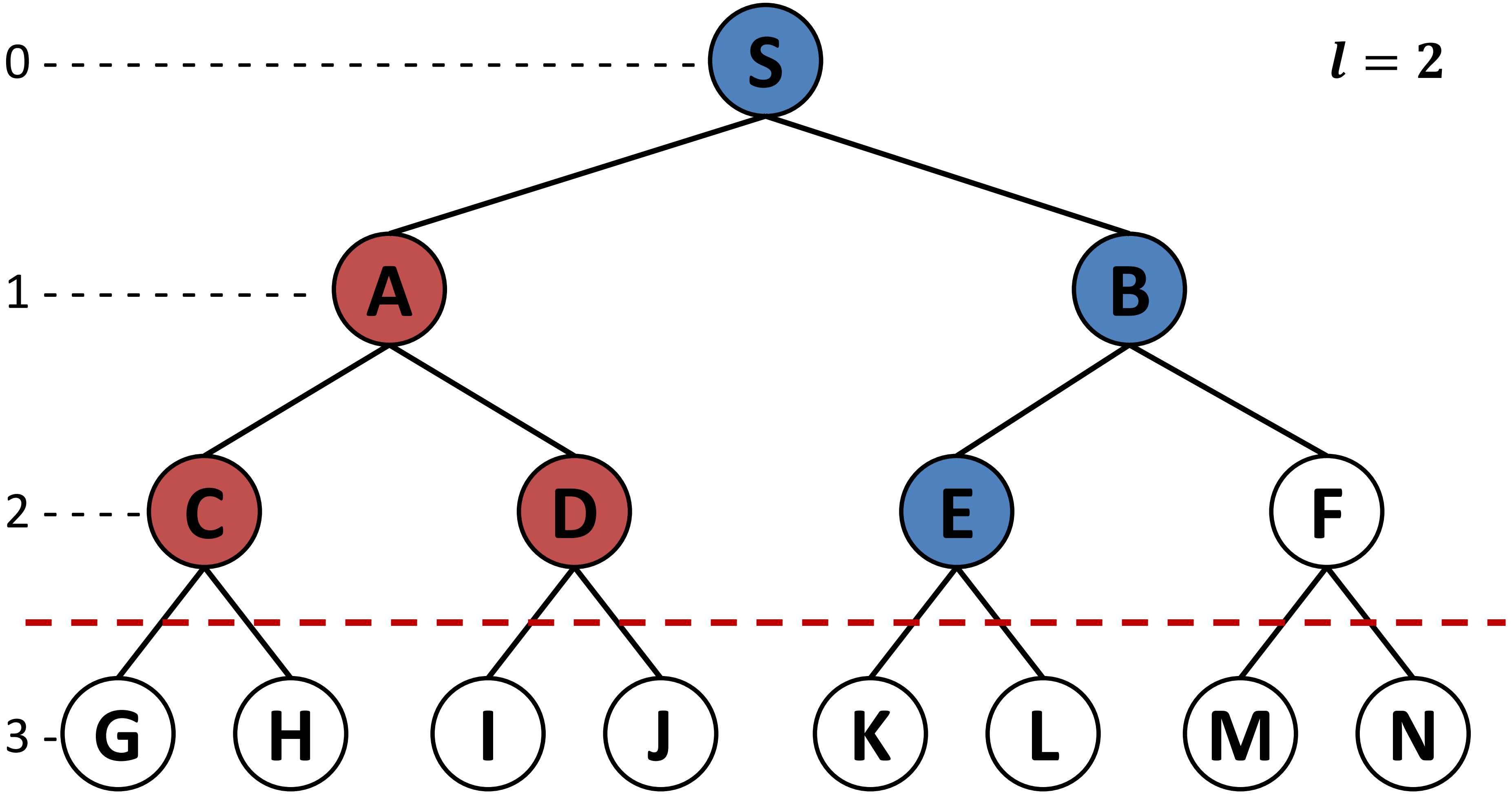
Iterative Deepening Search



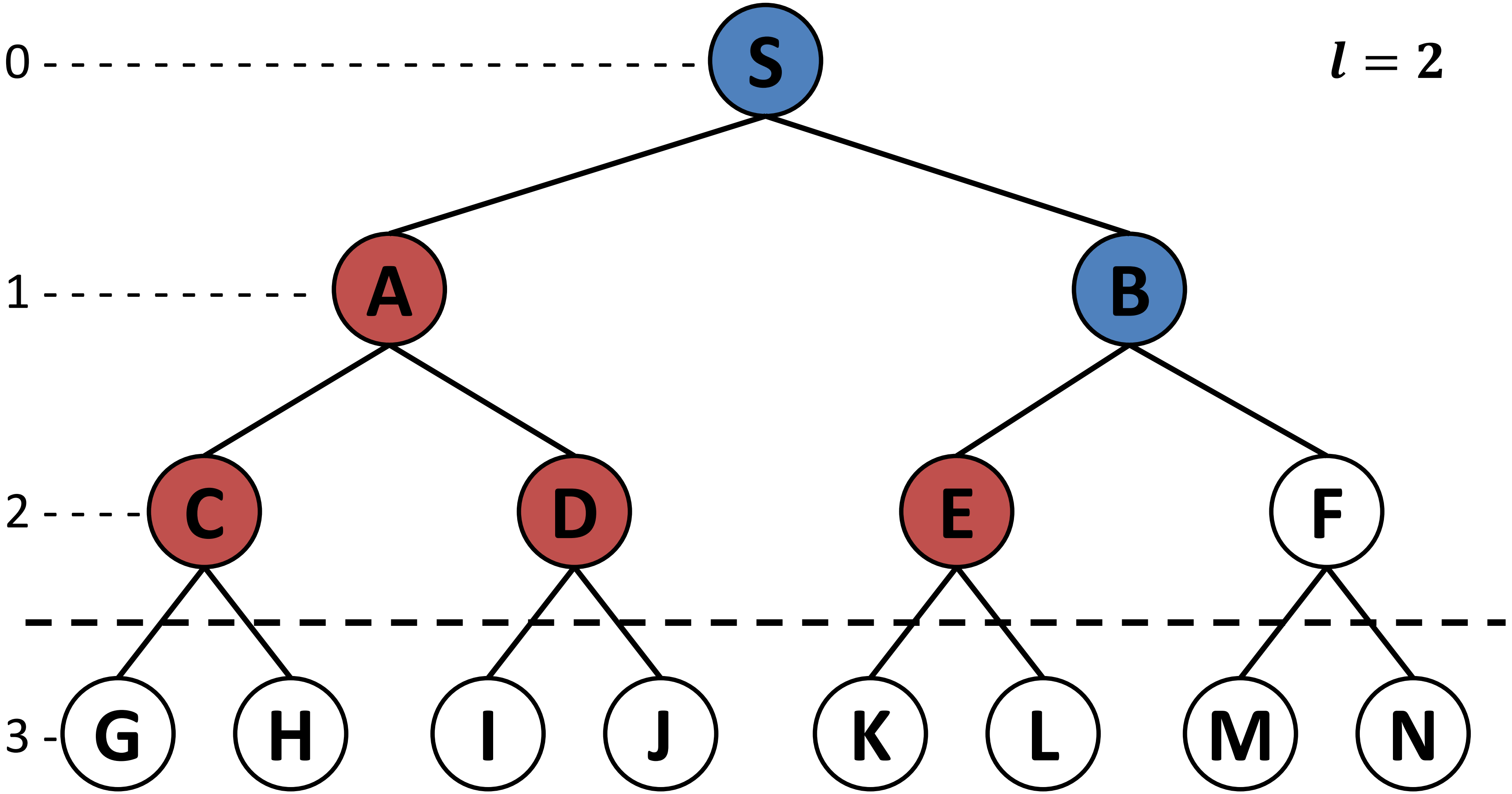
Iterative Deepening Search



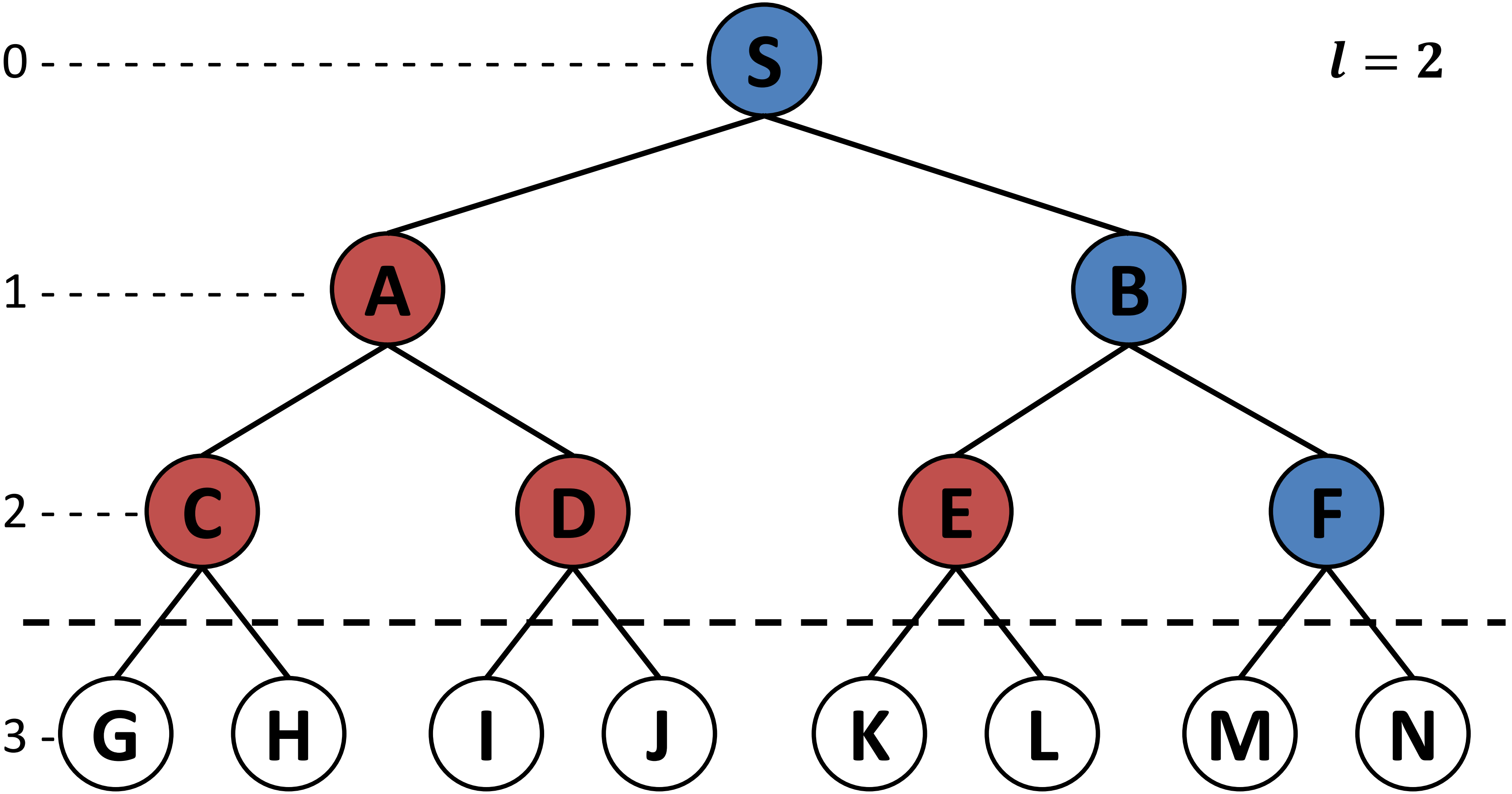
Iterative Deepening Search



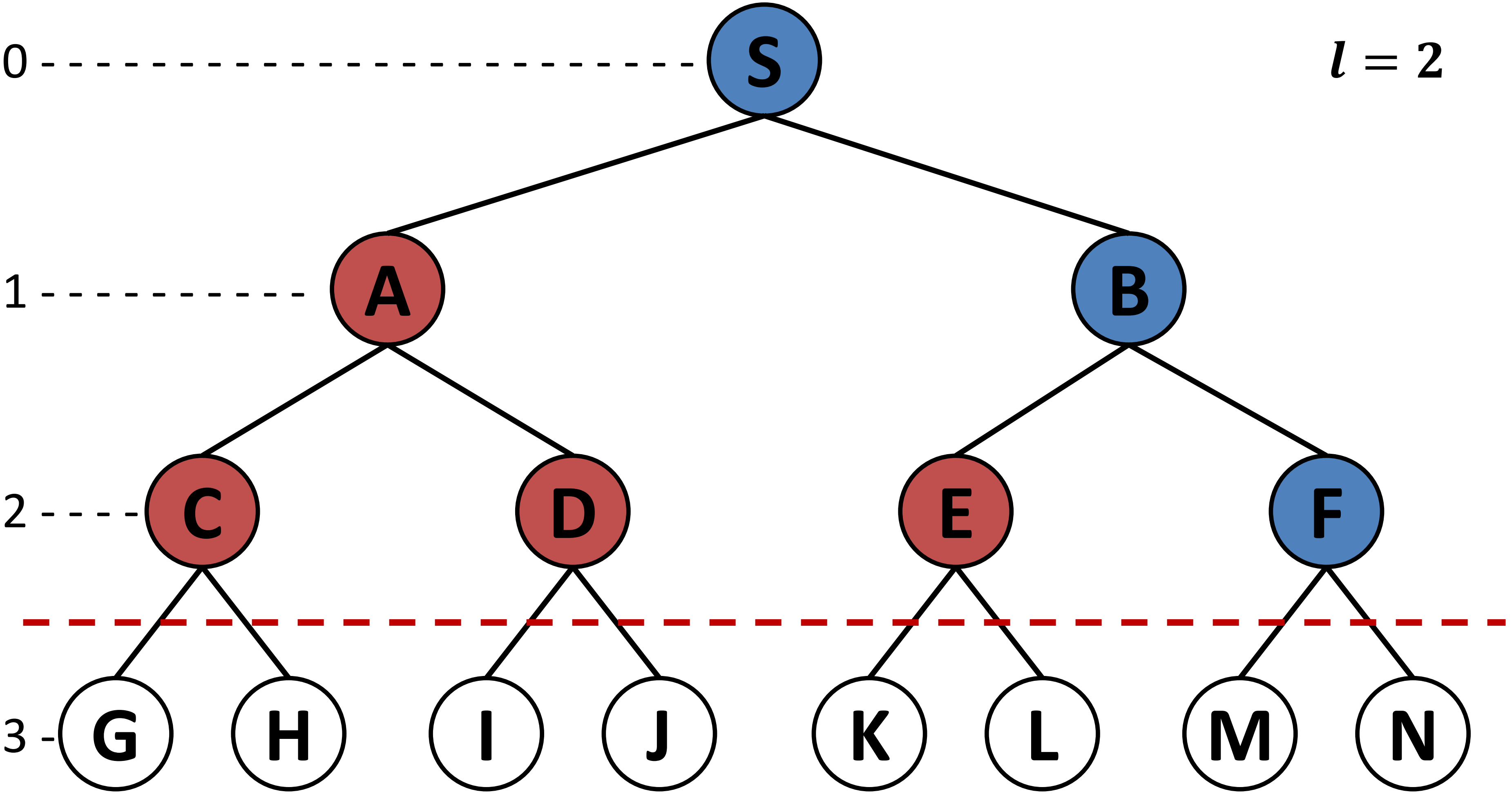
Iterative Deepening Search



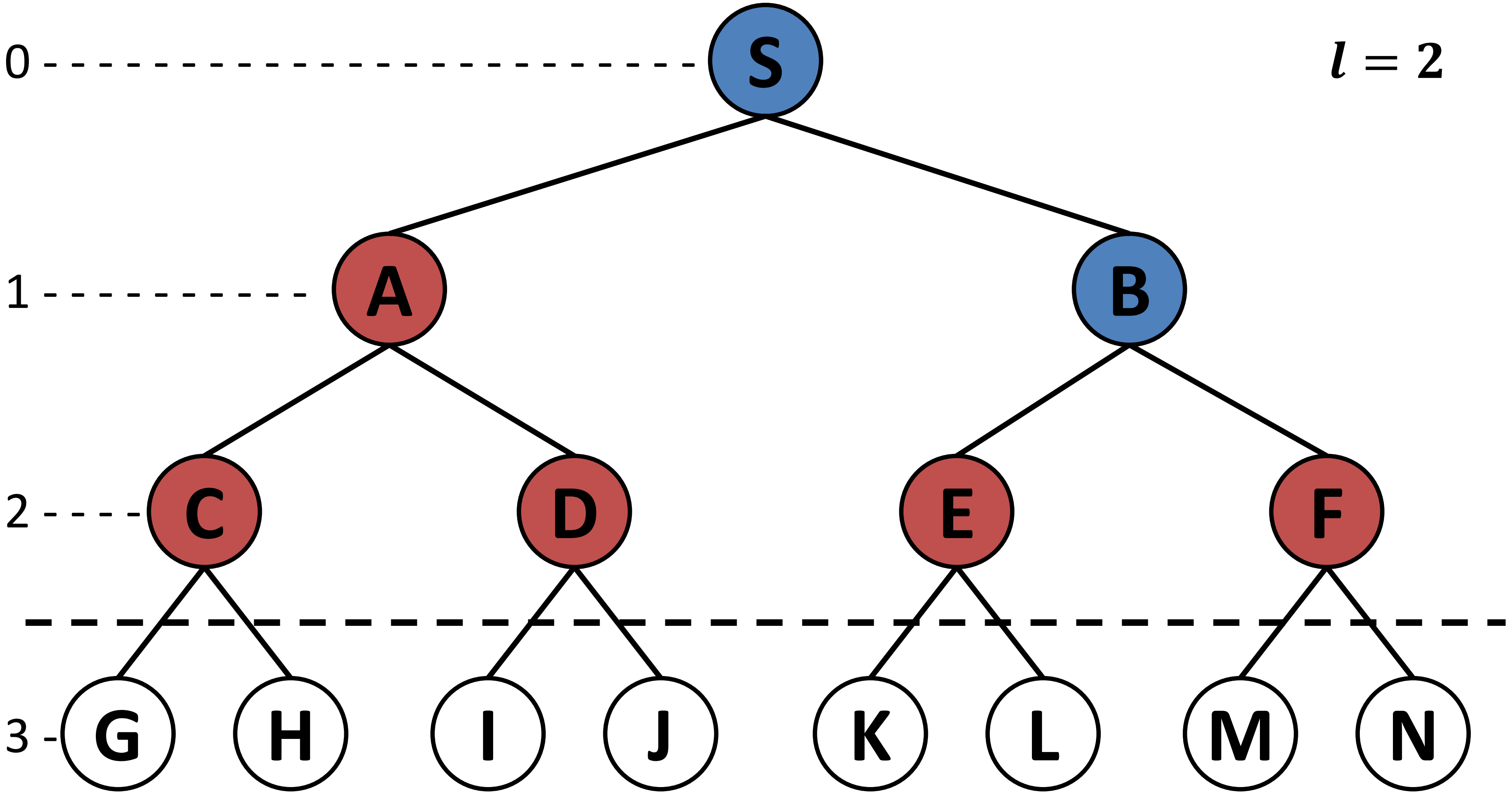
Iterative Deepening Search



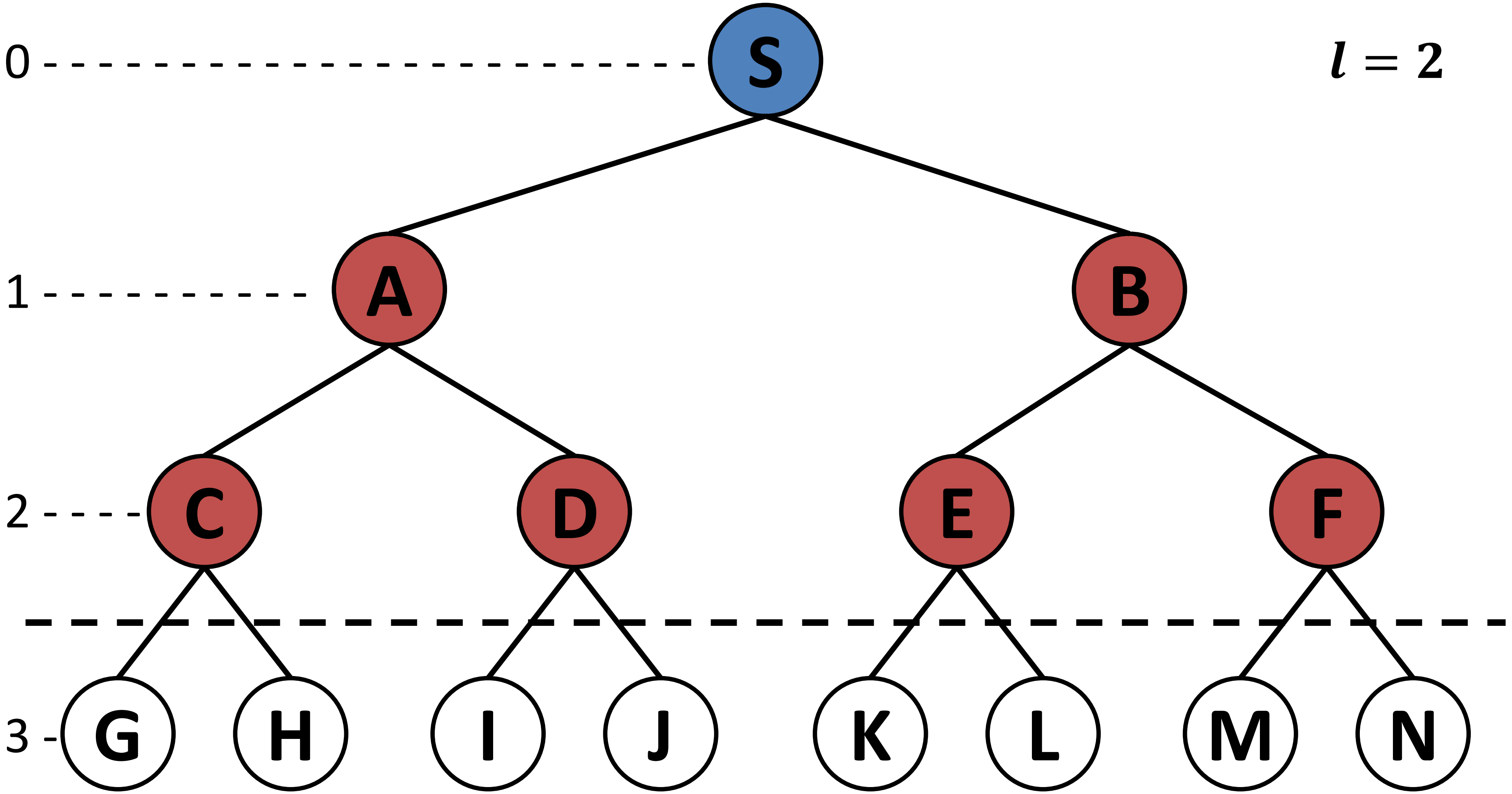
Iterative Deepening Search



Iterative Deepening Search

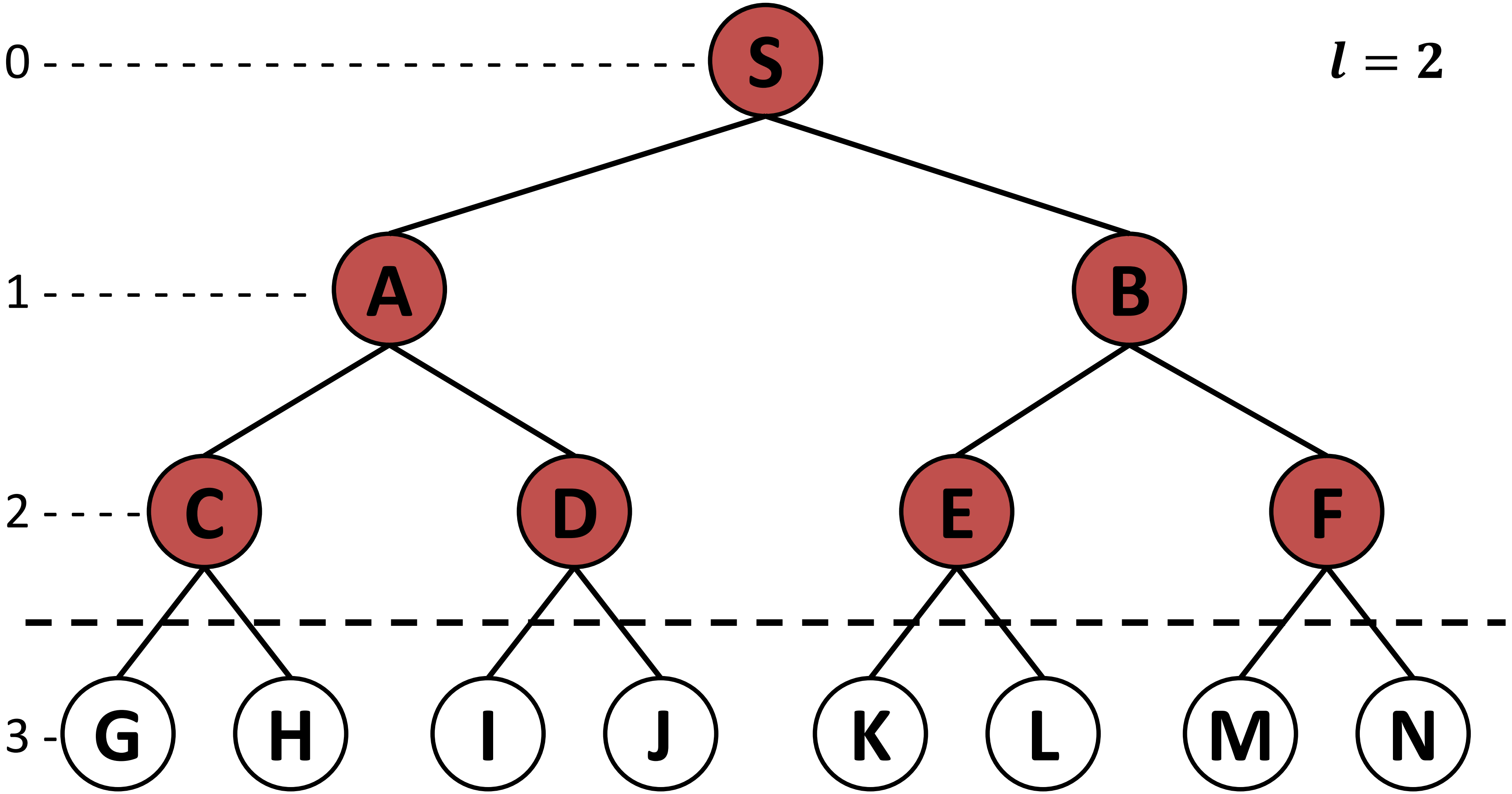


Iterative Deepening Search

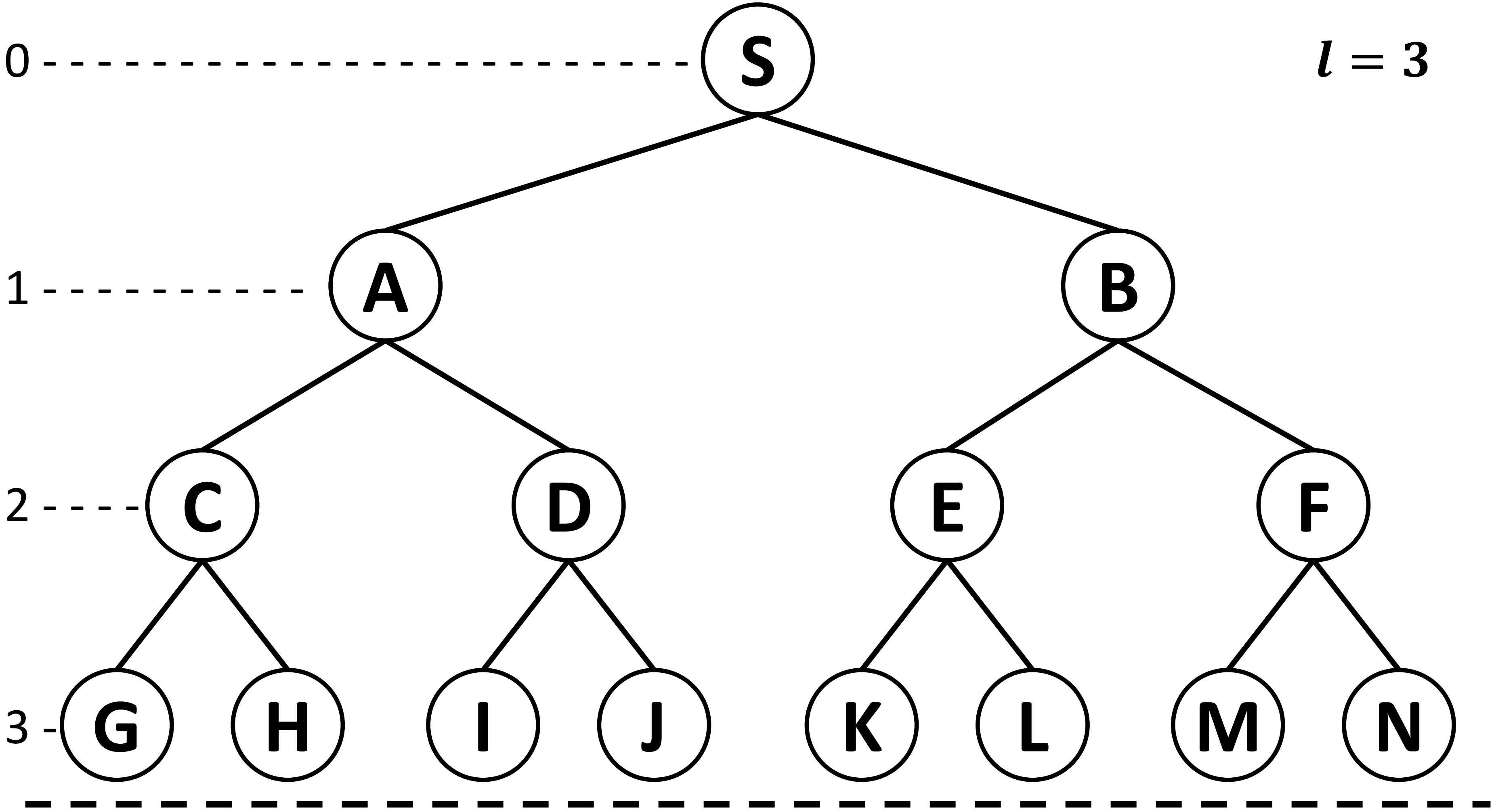


$l = 2$

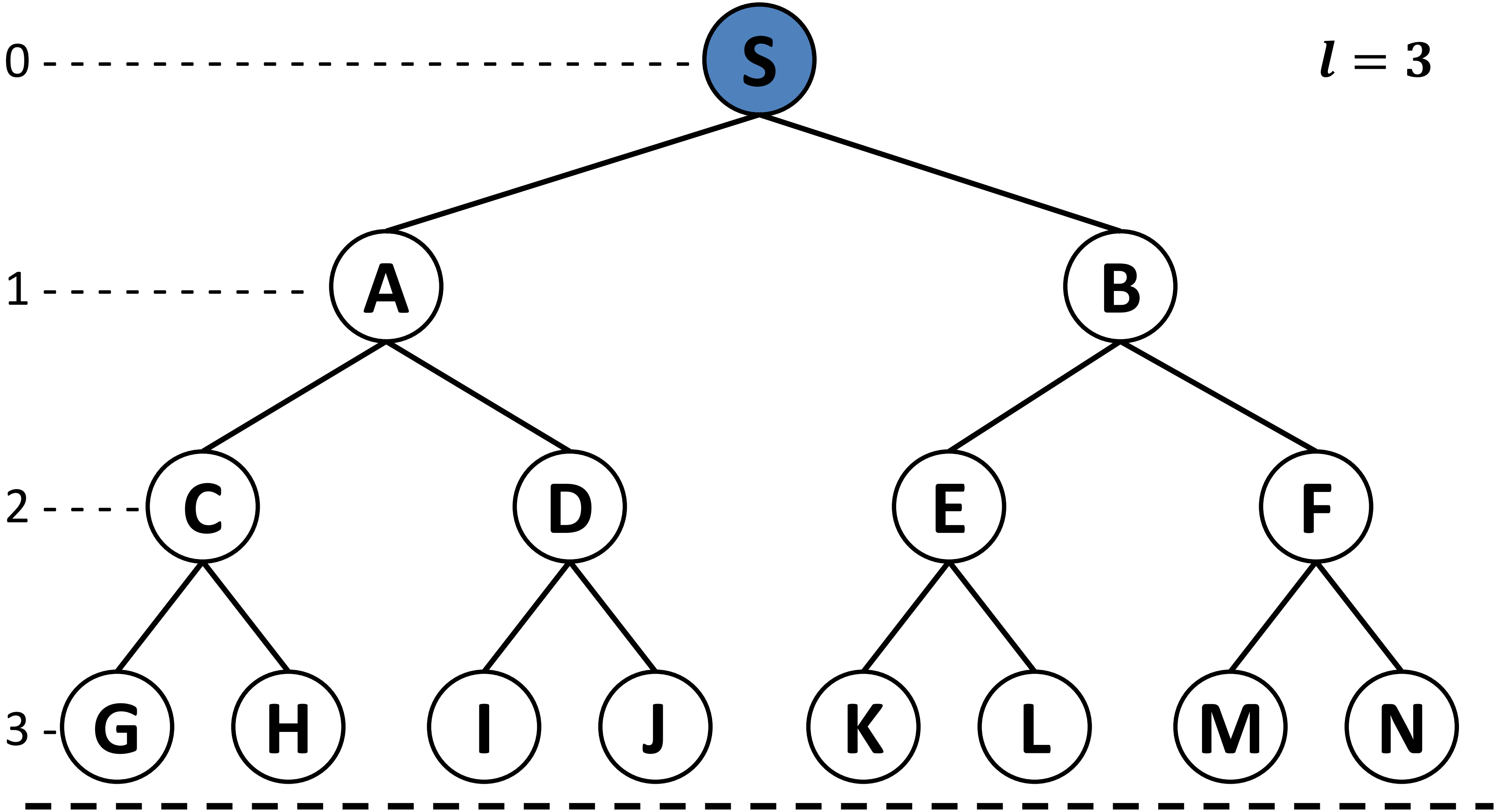
Iterative Deepening Search



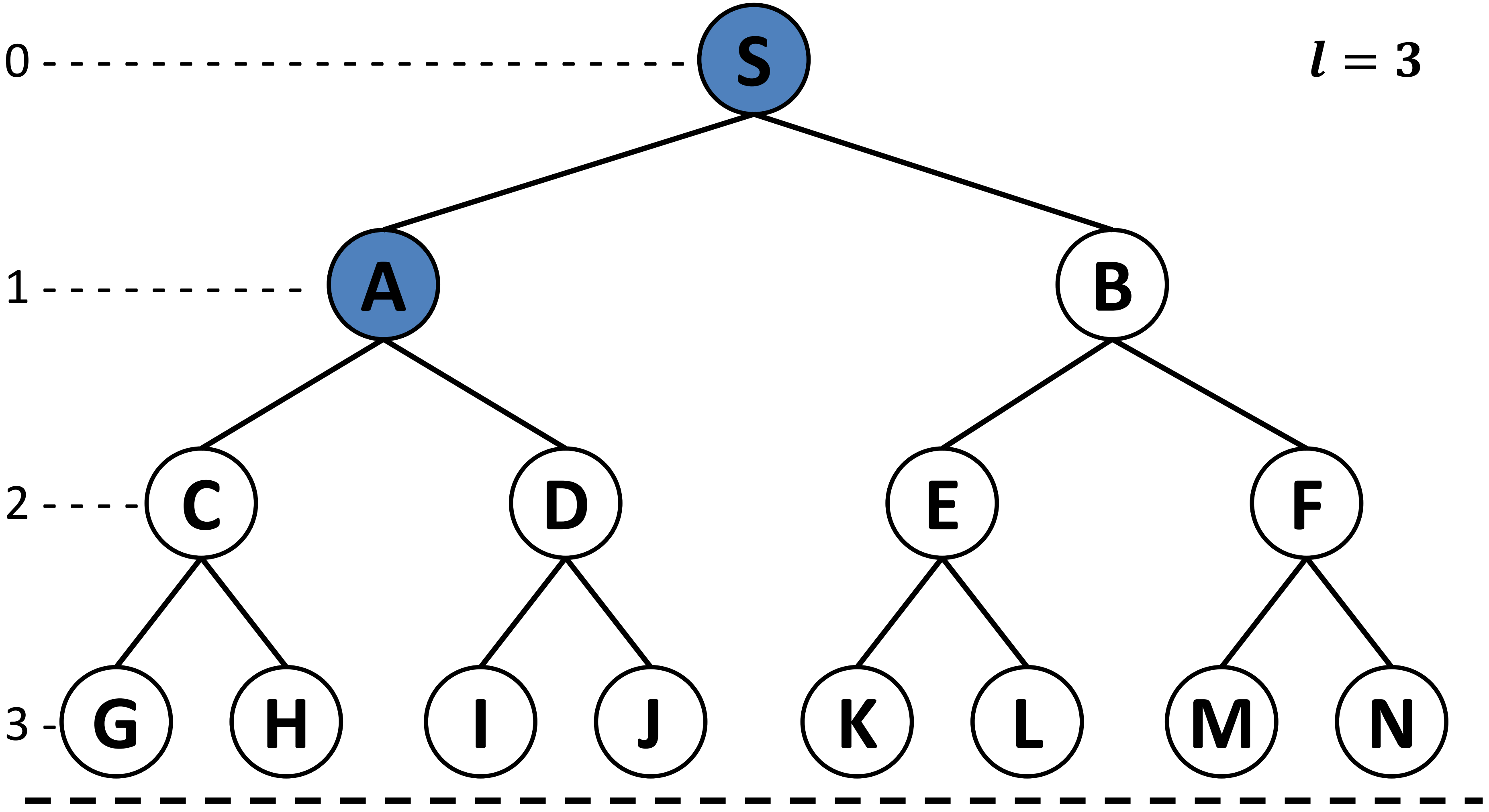
Iterative Deepening Search



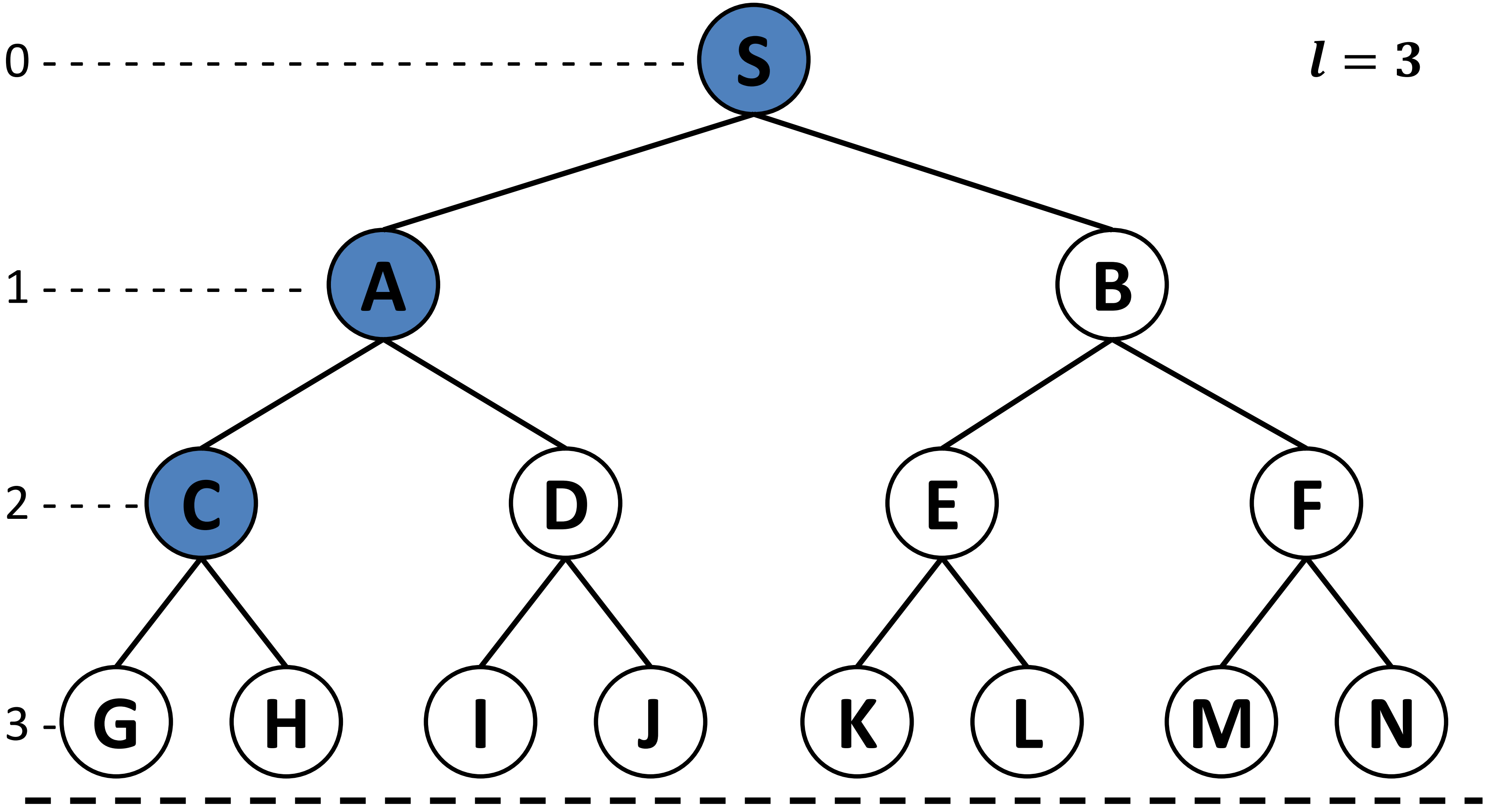
Iterative Deepening Search



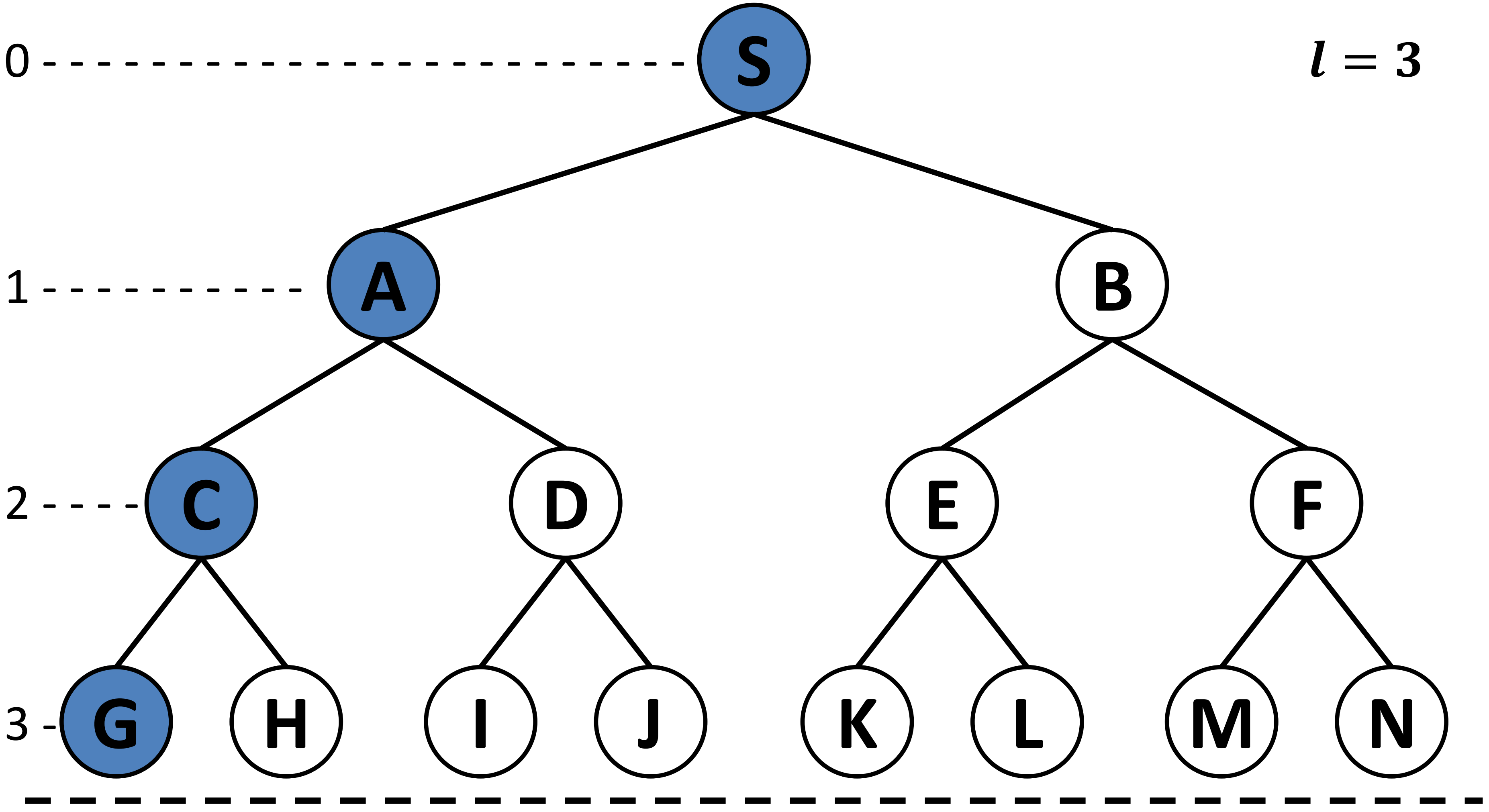
Iterative Deepening Search



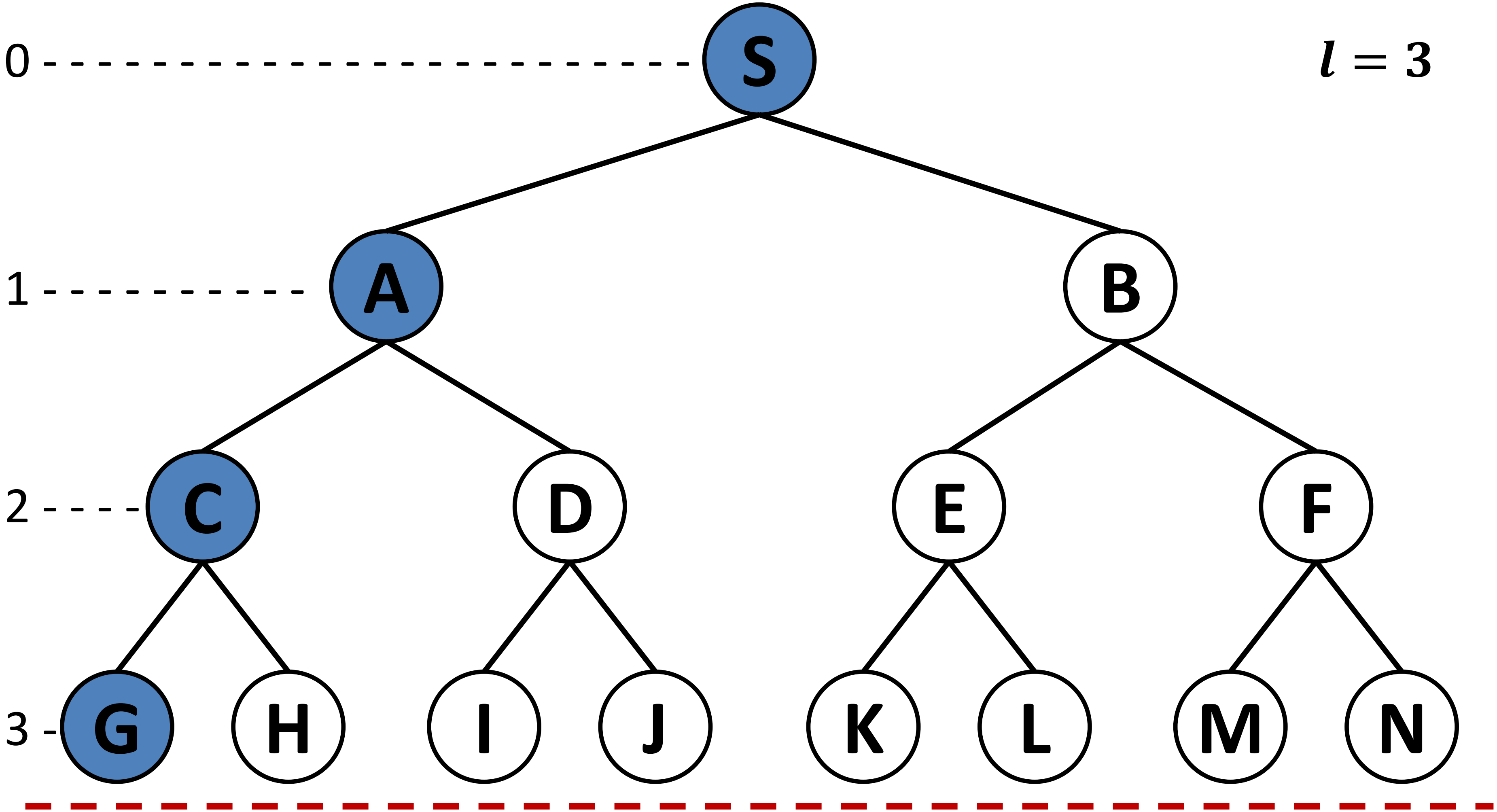
Iterative Deepening Search



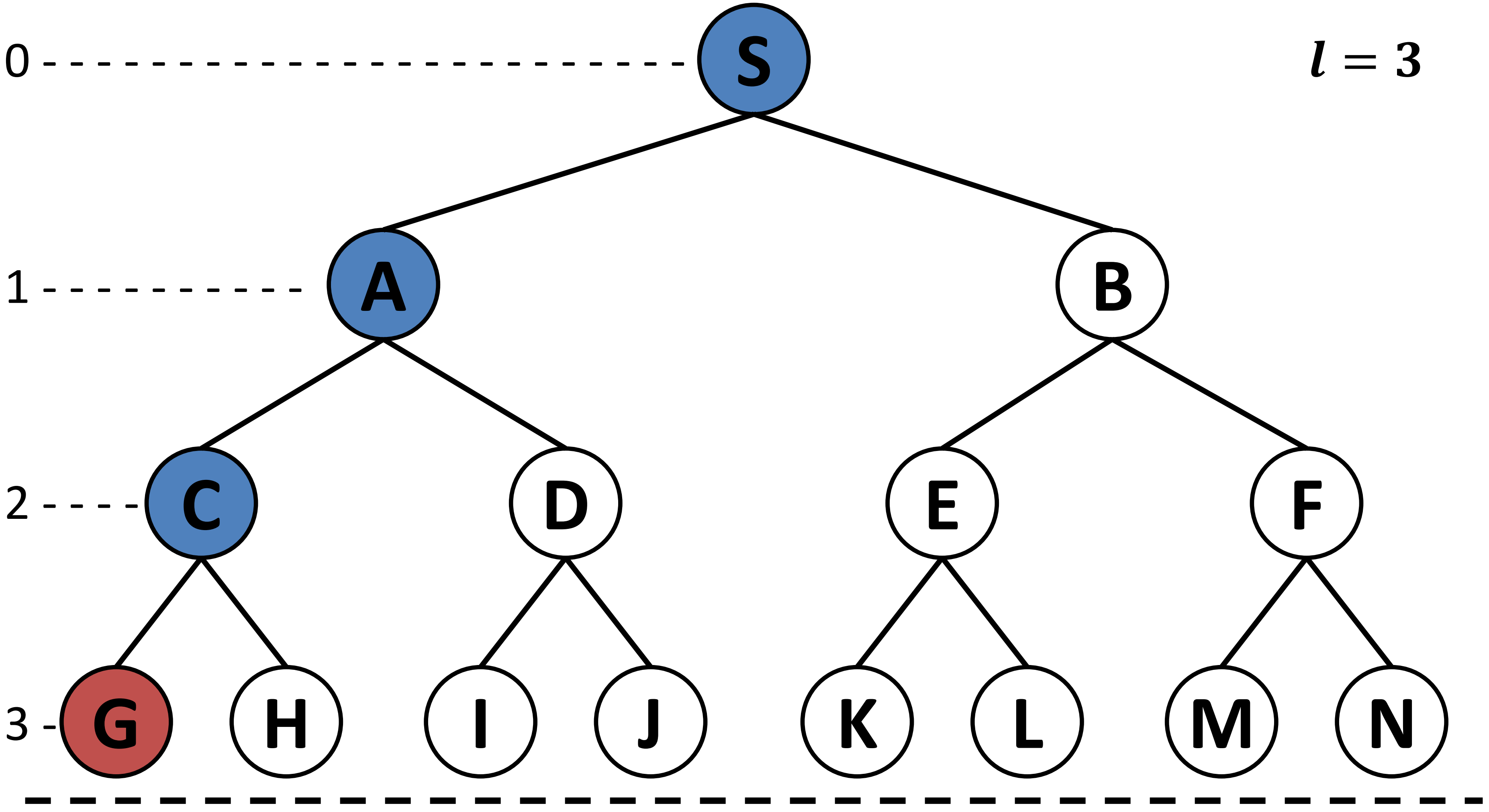
Iterative Deepening Search



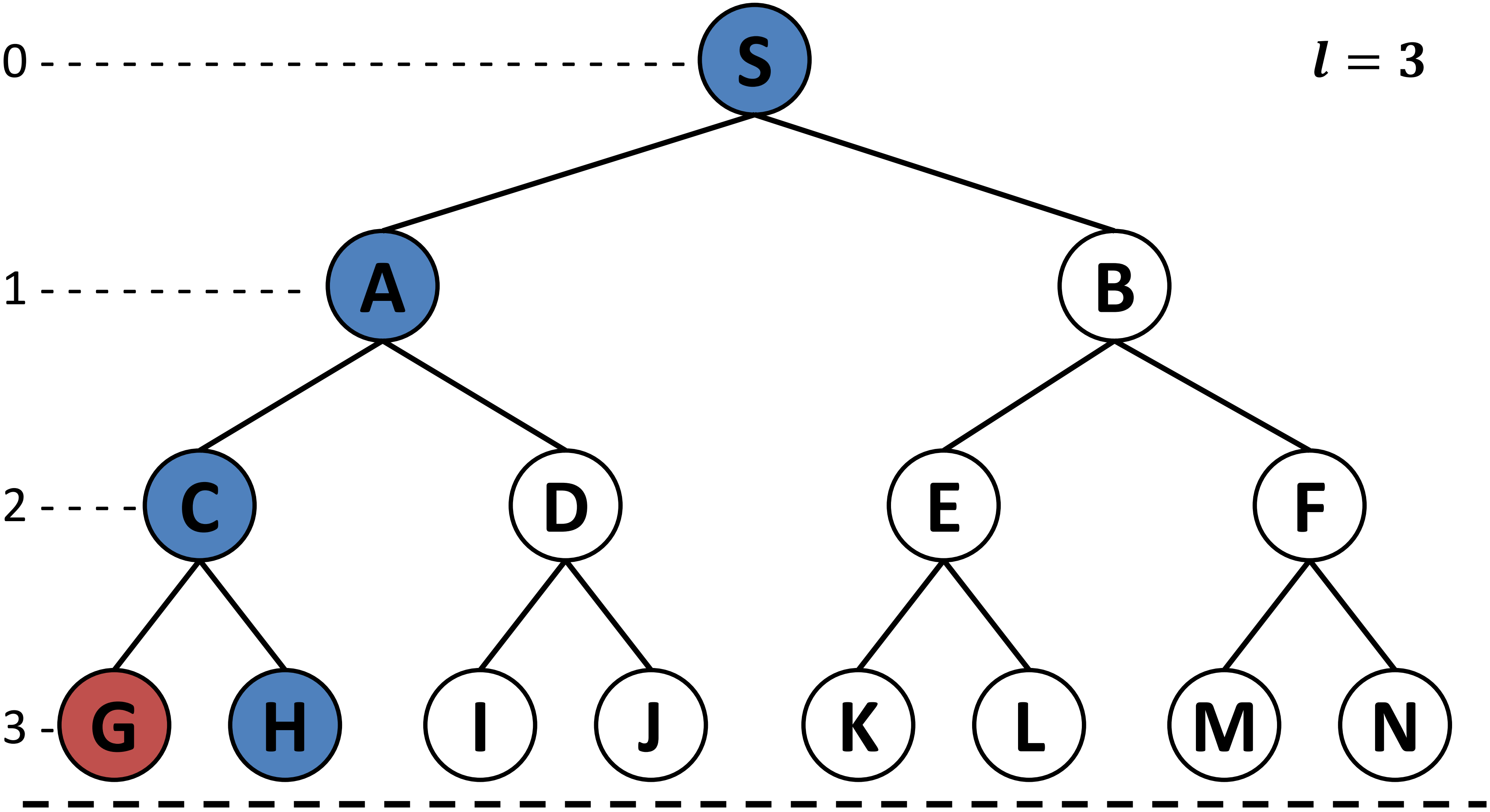
Iterative Deepening Search



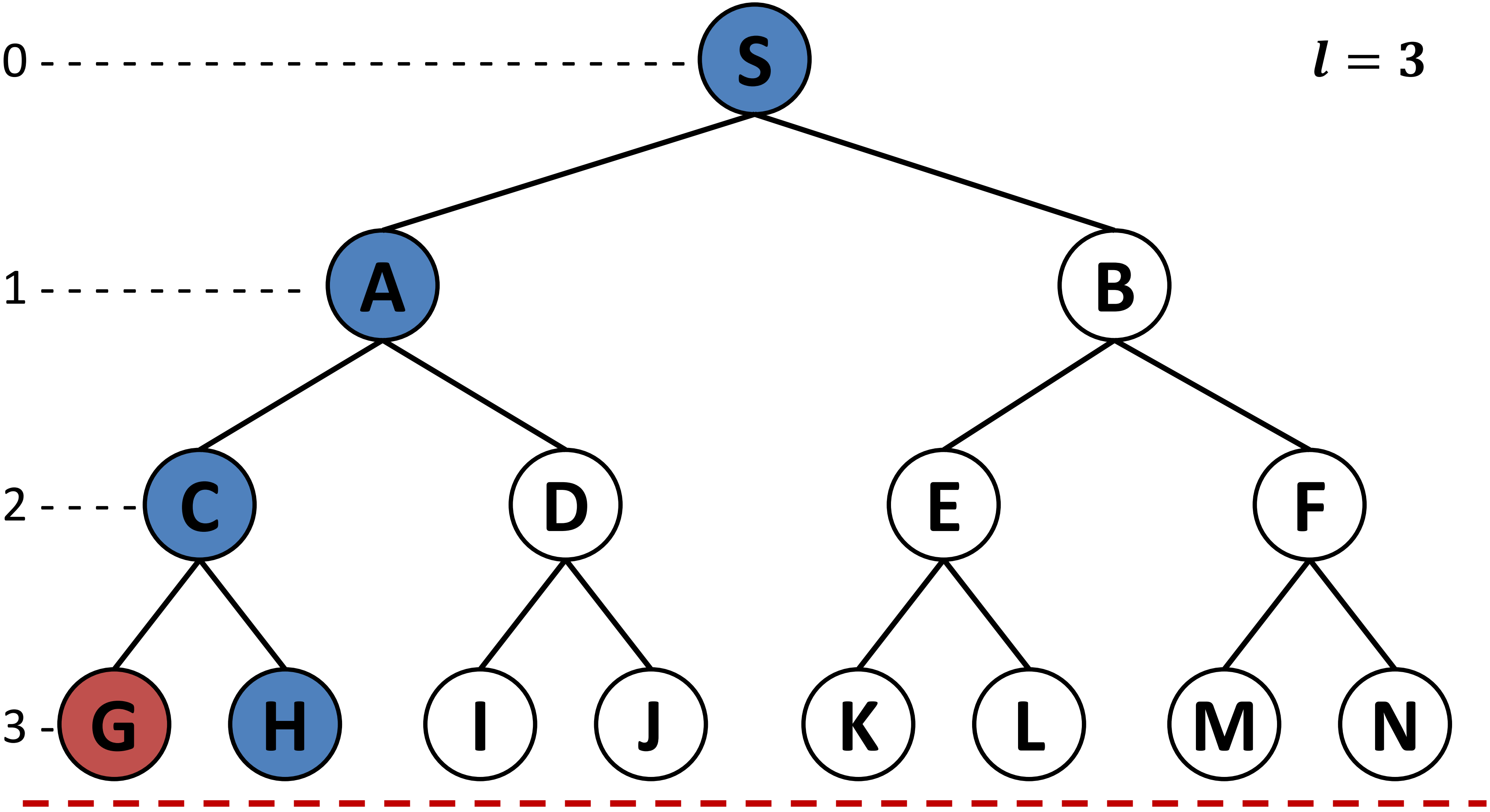
Iterative Deepening Search



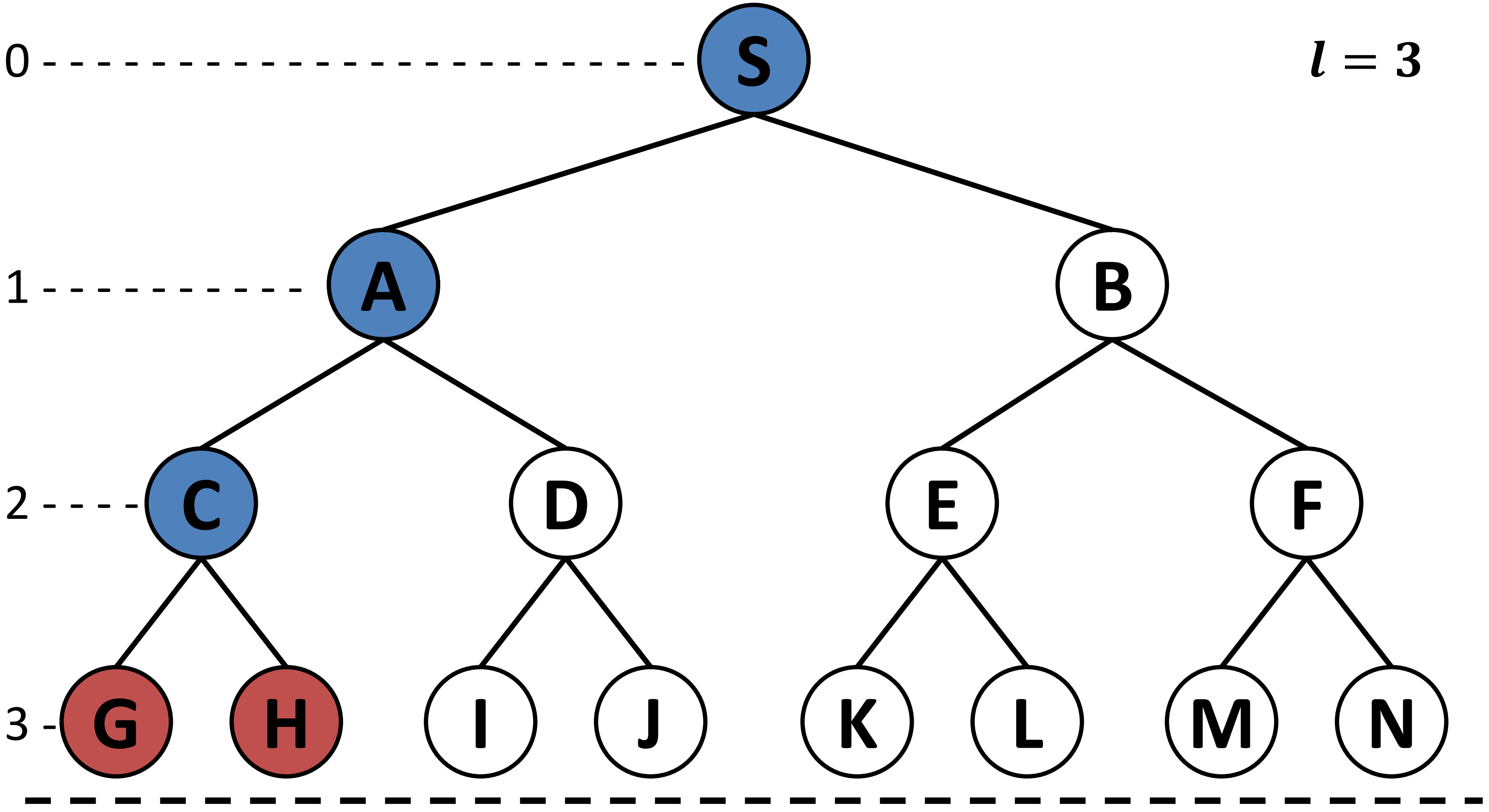
Iterative Deepening Search



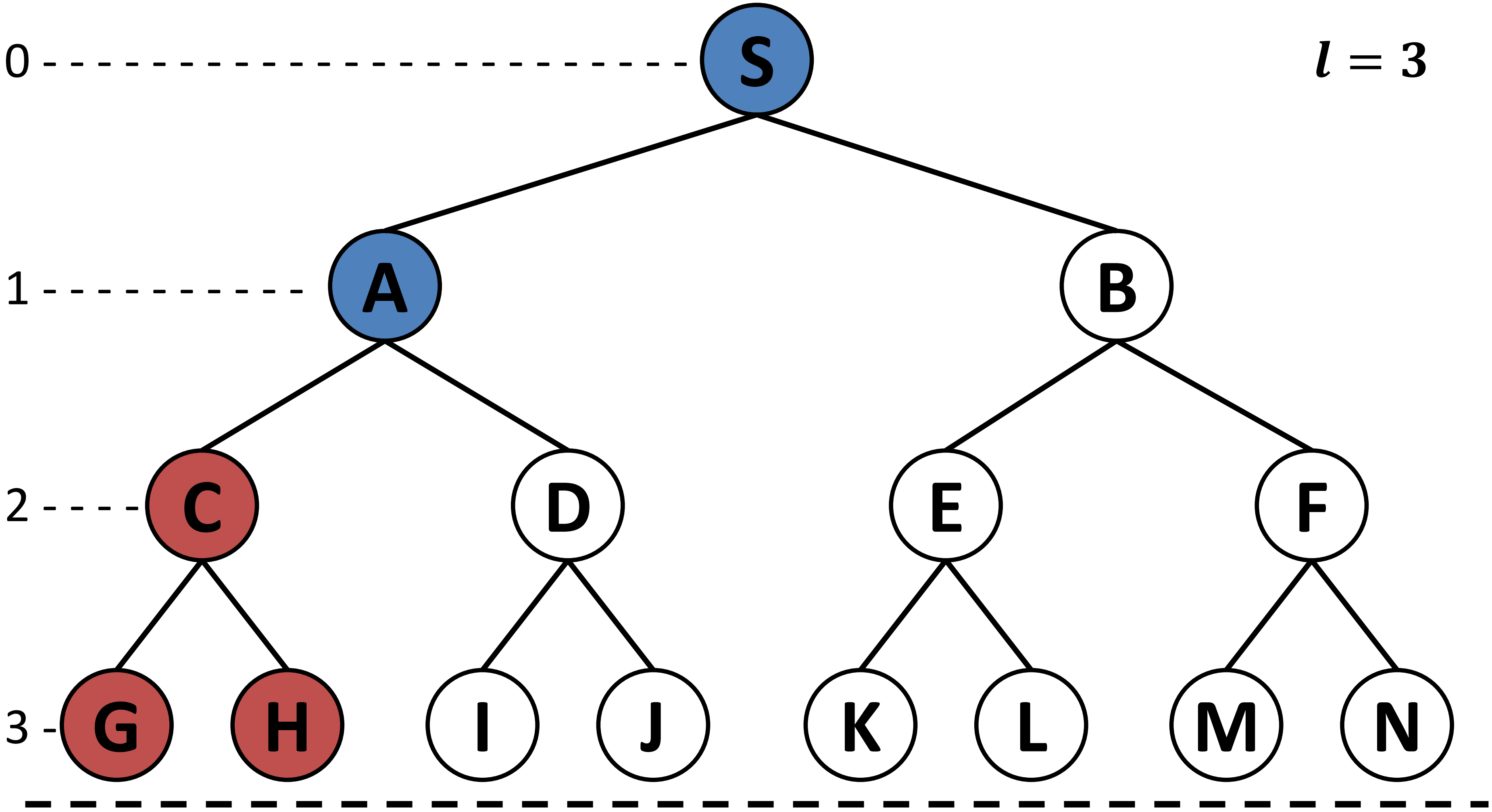
Iterative Deepening Search



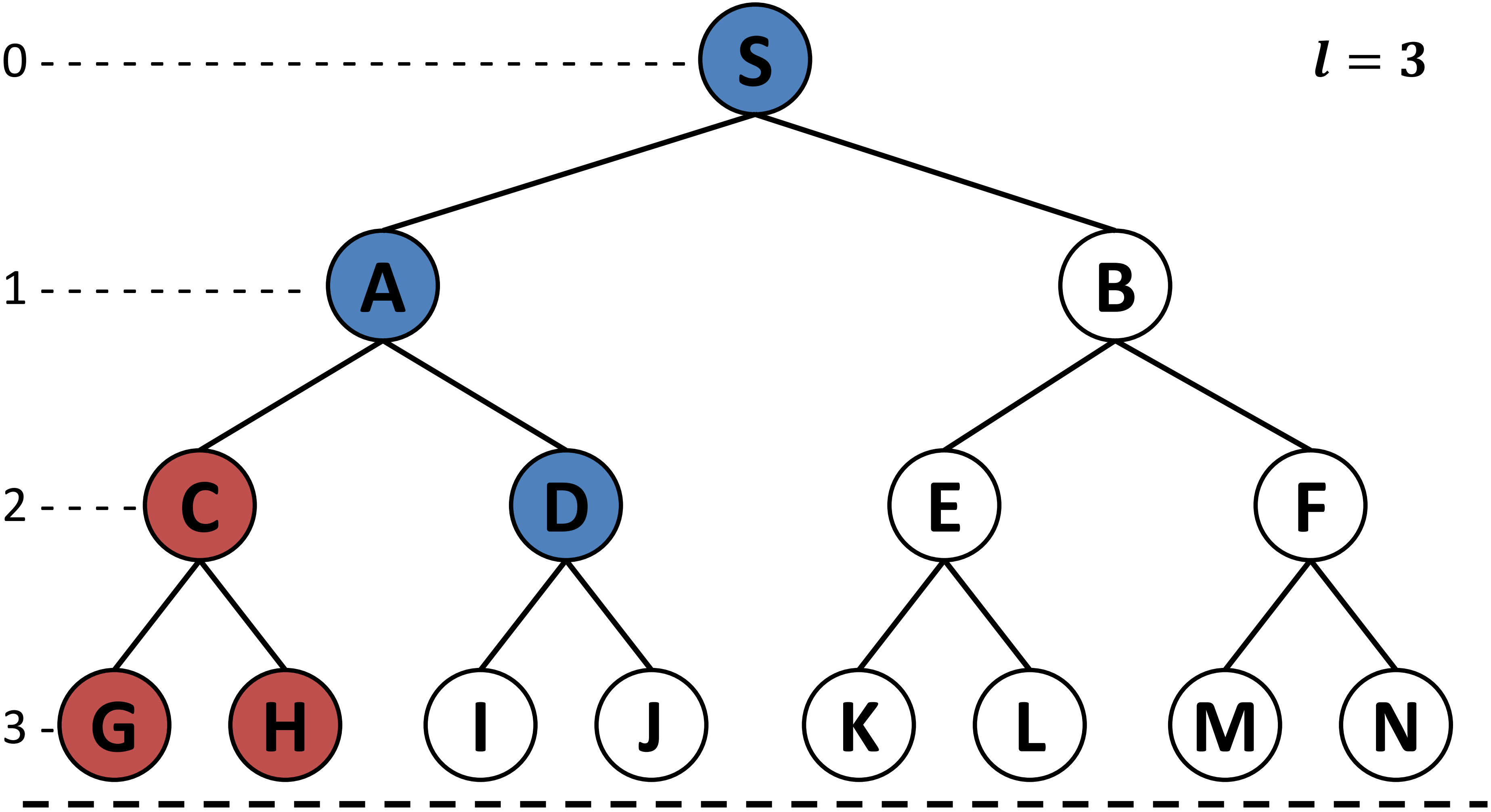
Iterative Deepening Search



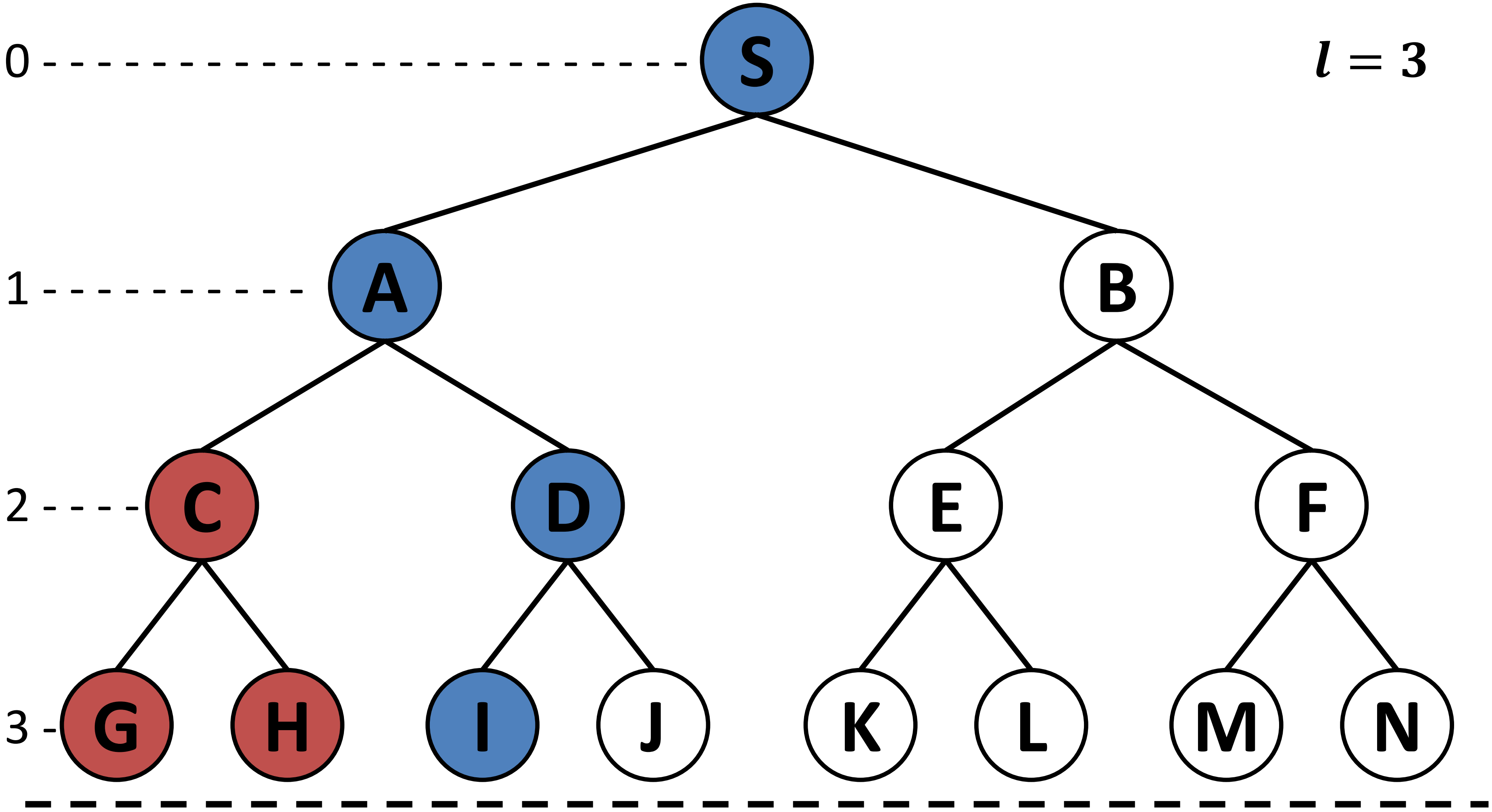
Iterative Deepening Search



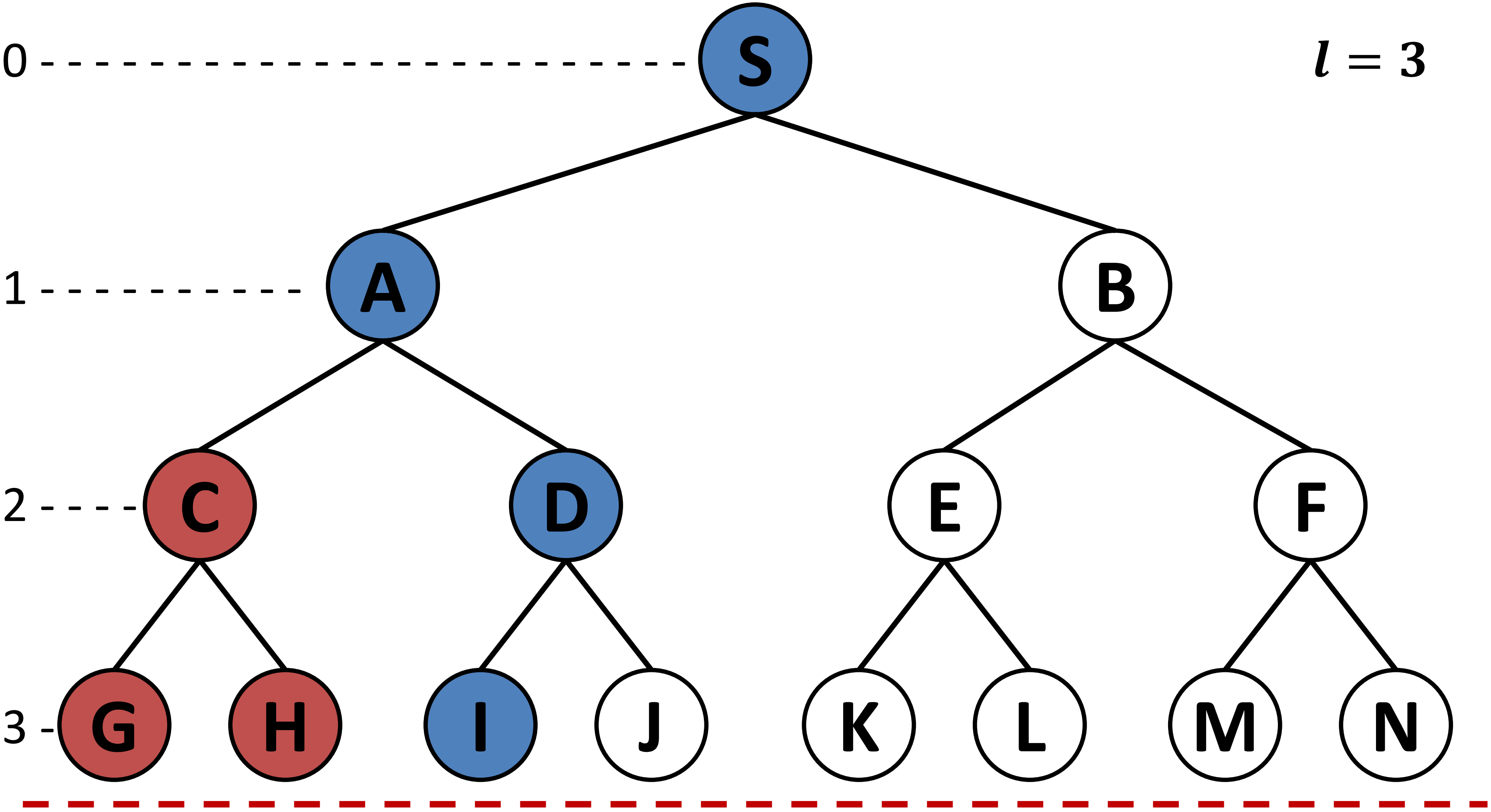
Iterative Deepening Search



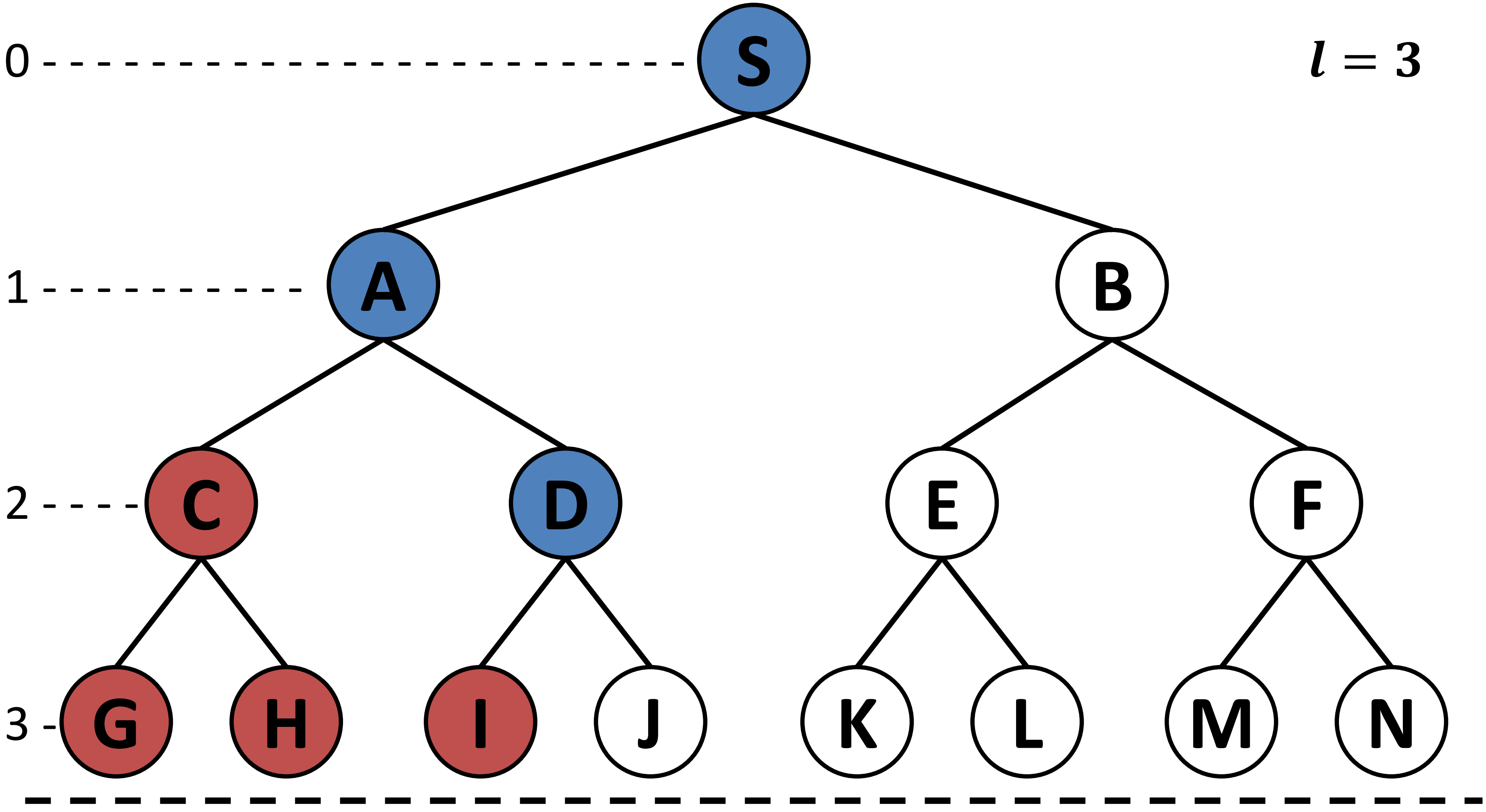
Iterative Deepening Search



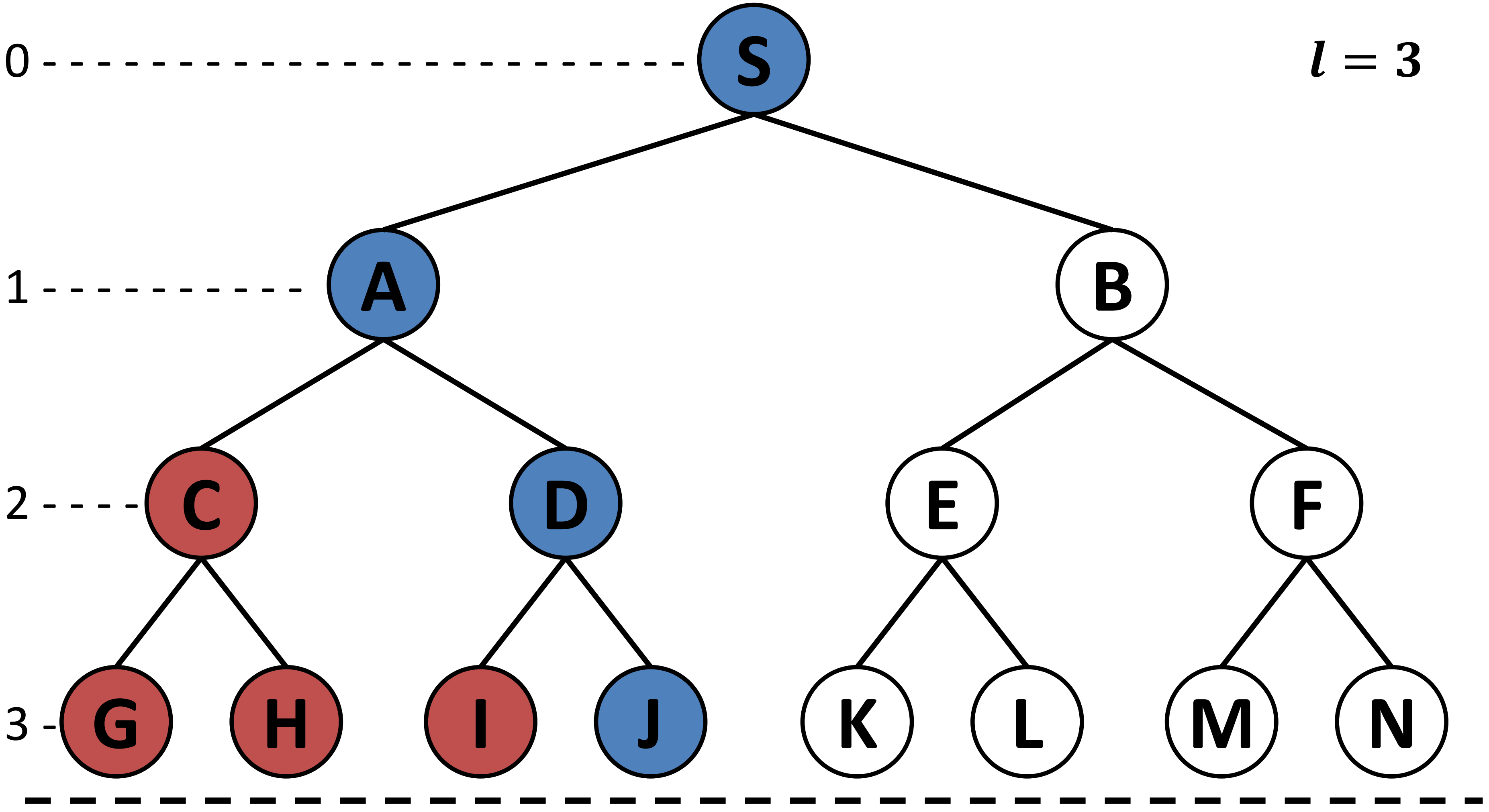
Iterative Deepening Search



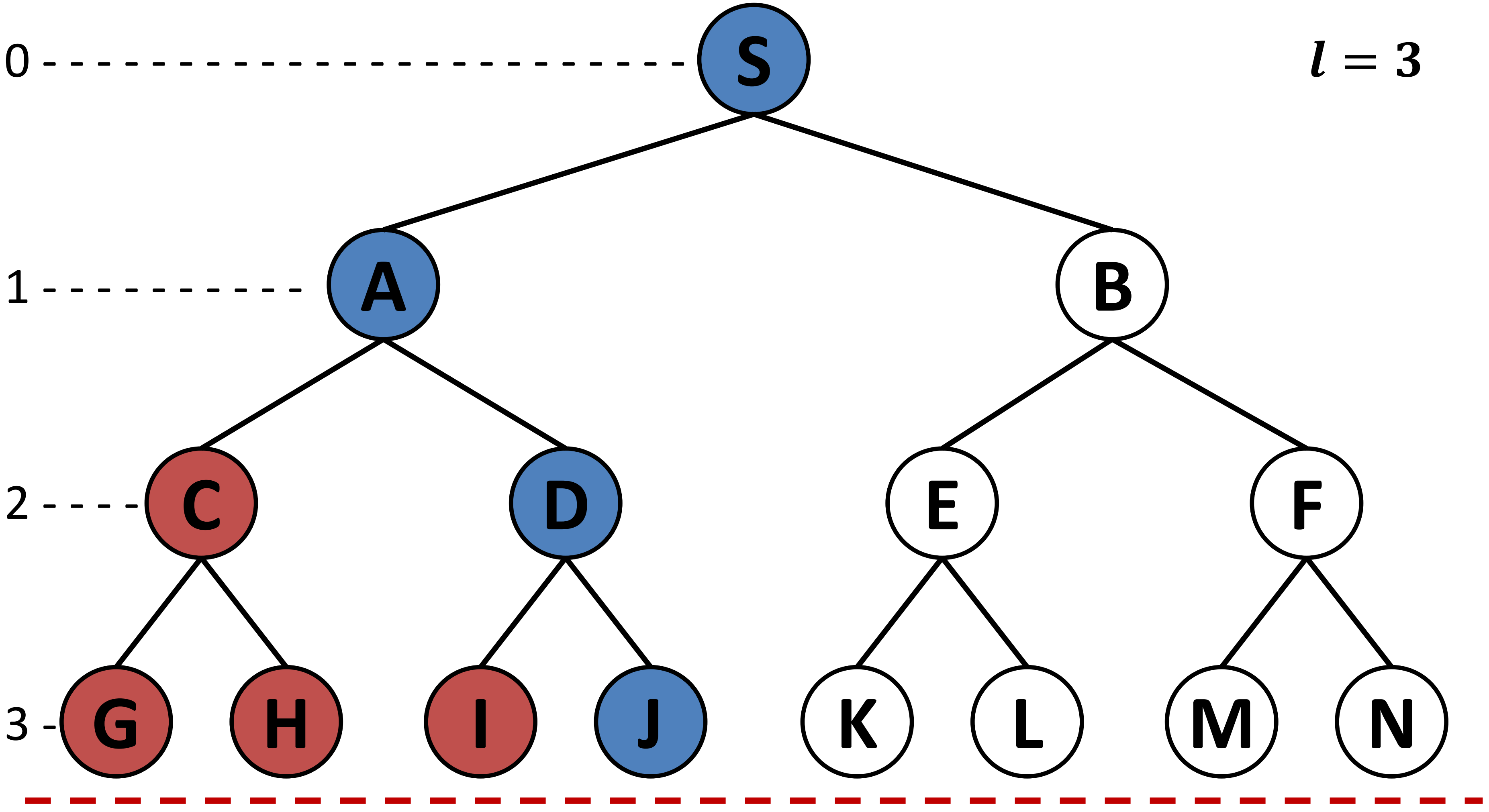
Iterative Deepening Search



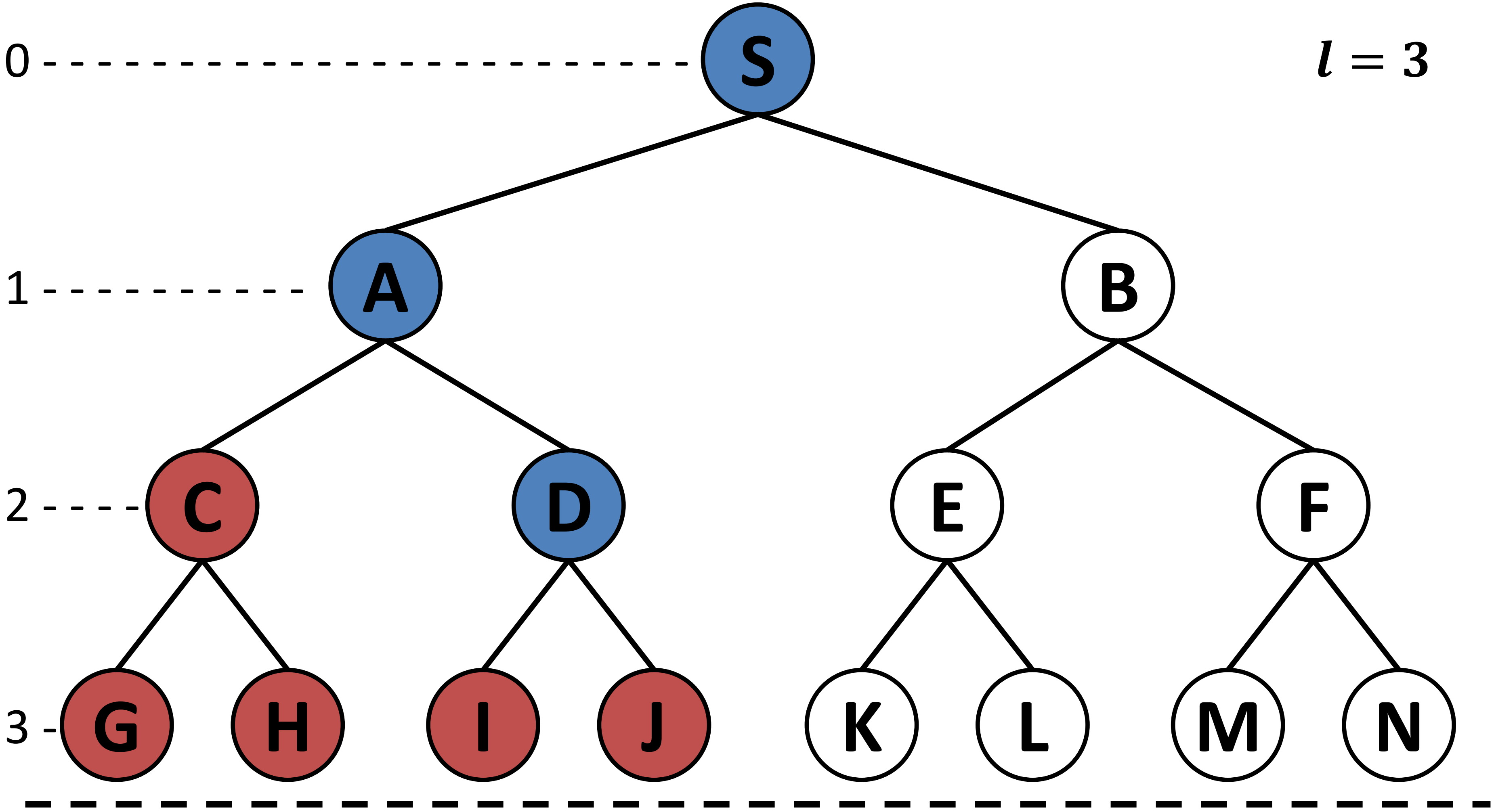
Iterative Deepening Search



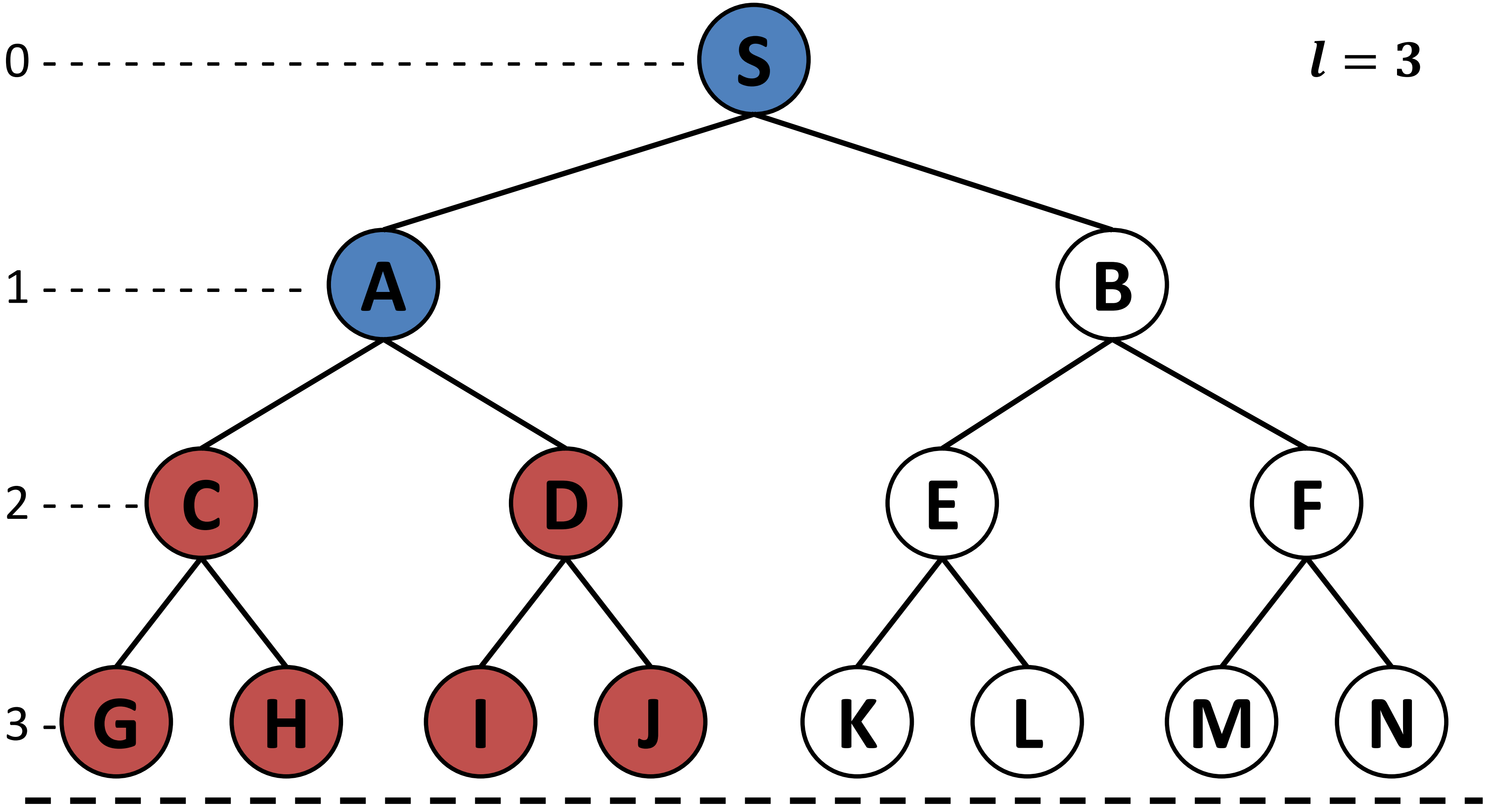
Iterative Deepening Search



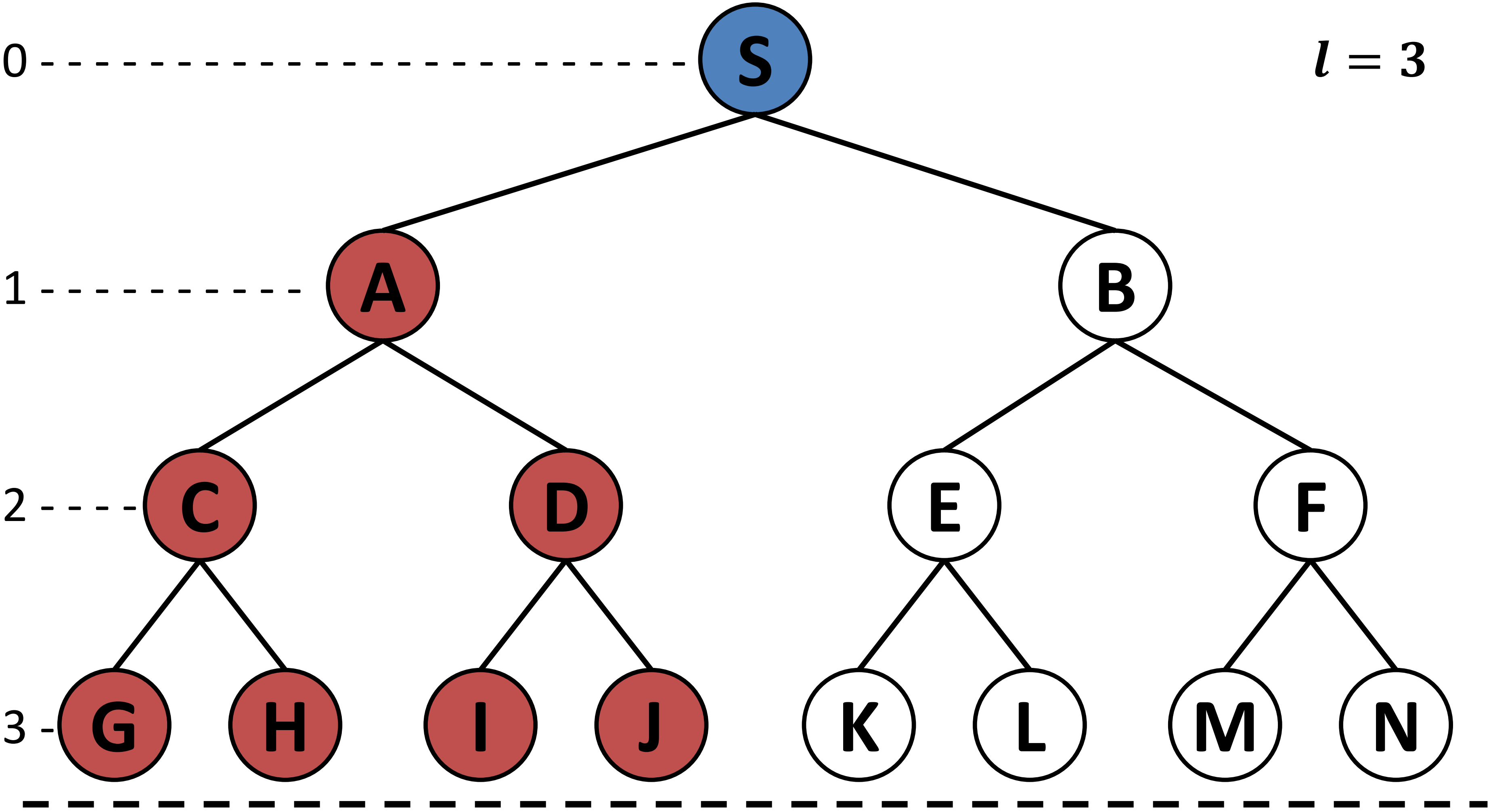
Iterative Deepening Search



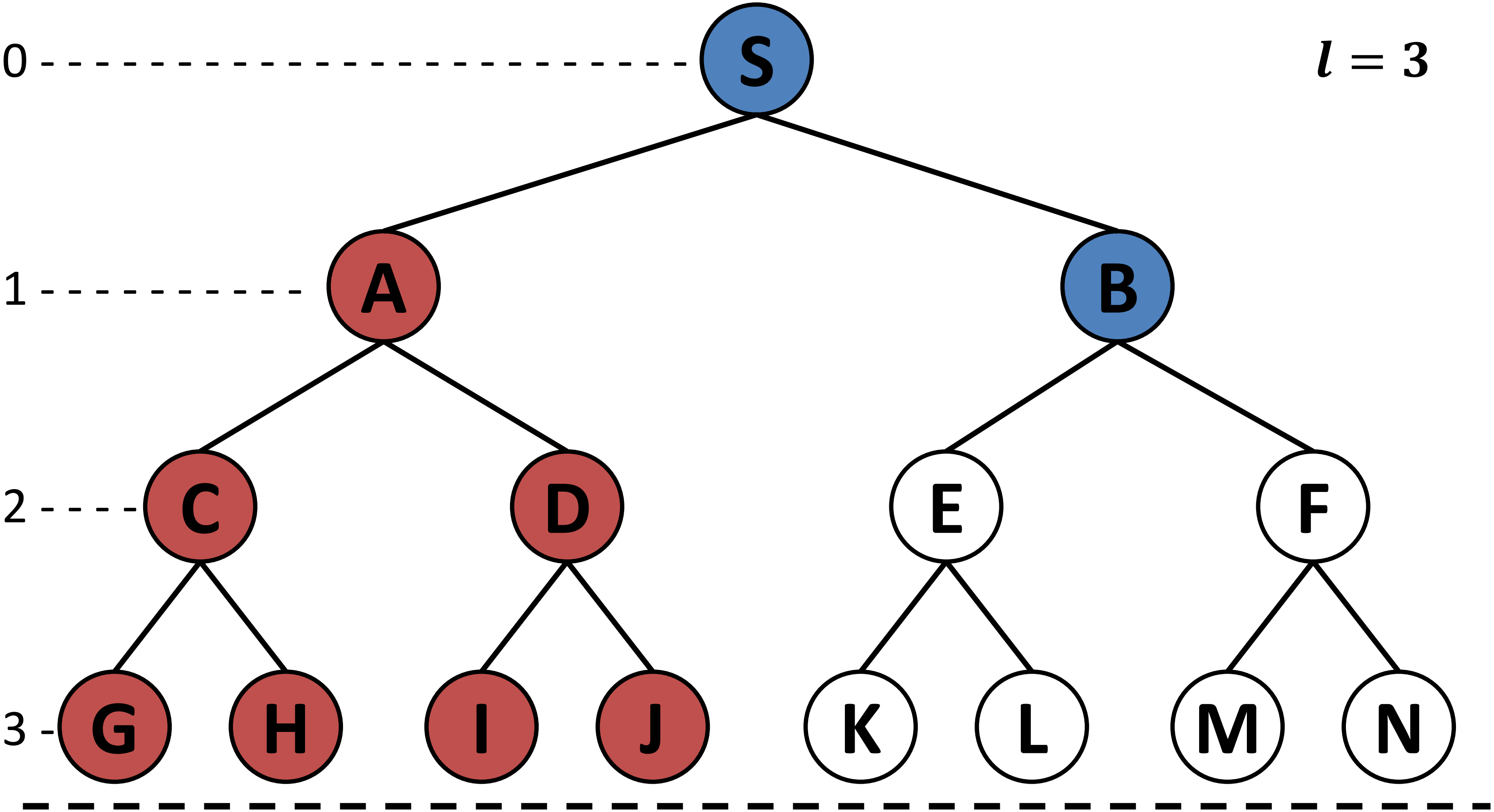
Iterative Deepening Search



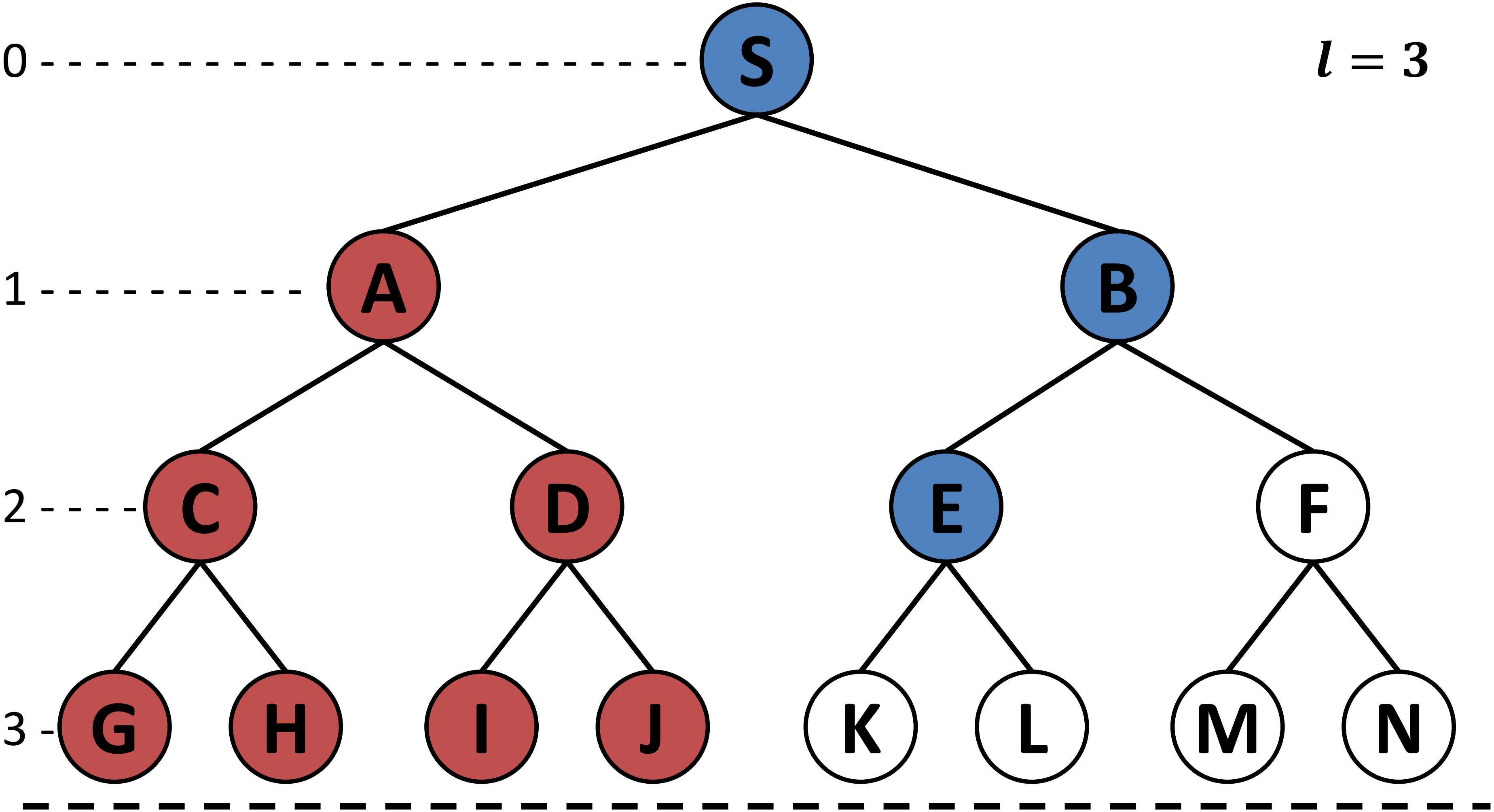
Iterative Deepening Search



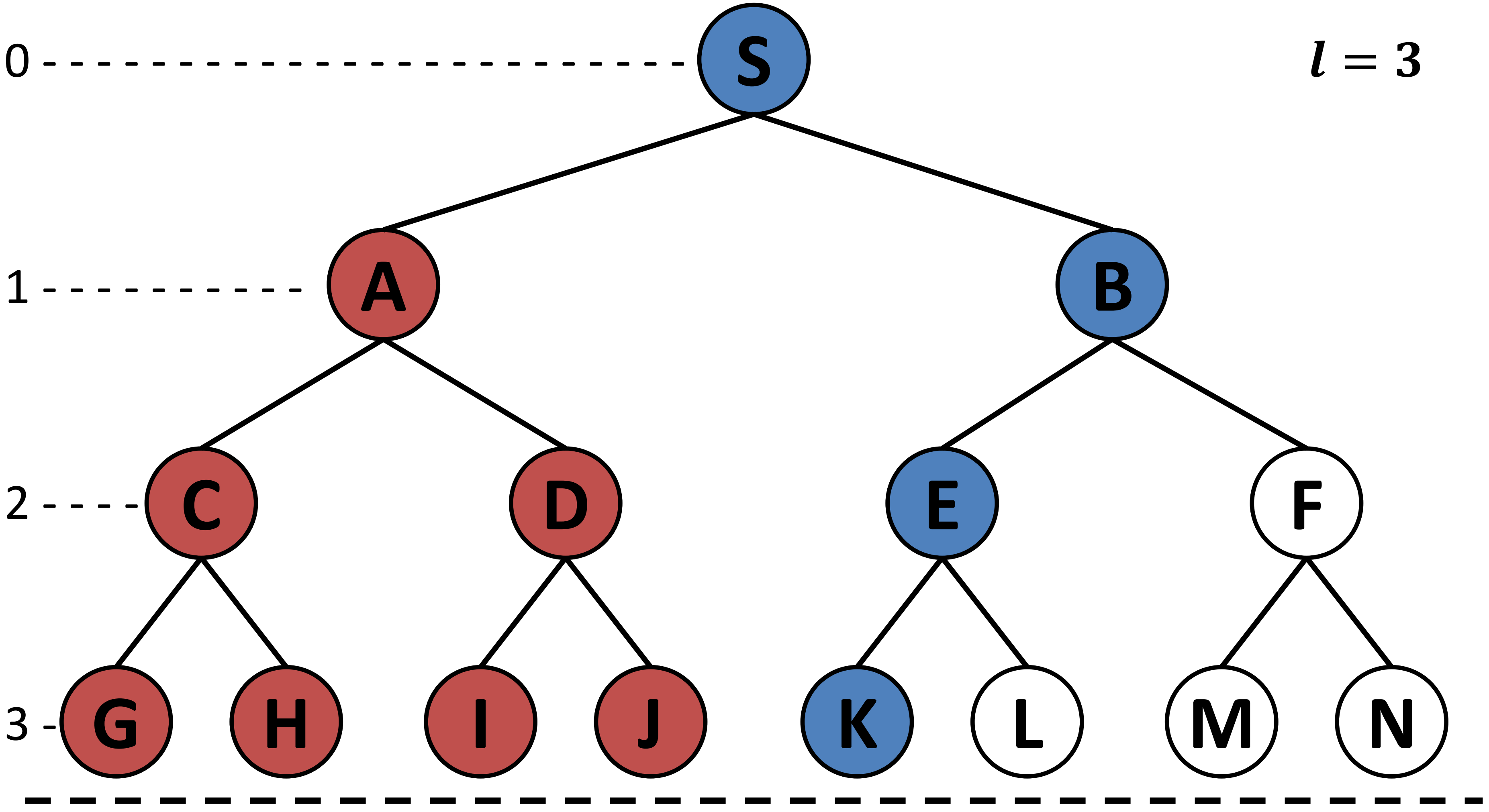
Iterative Deepening Search



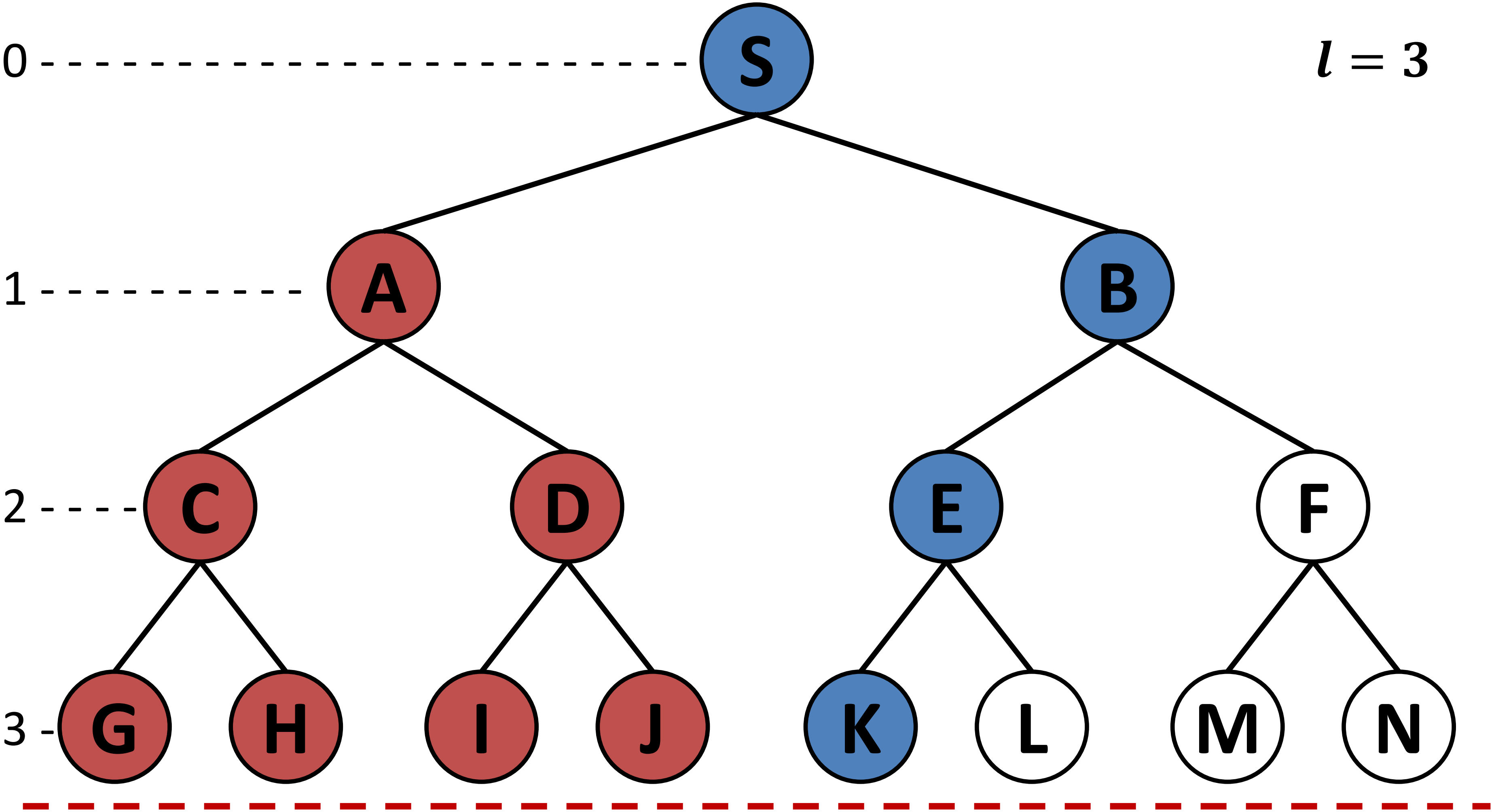
Iterative Deepening Search



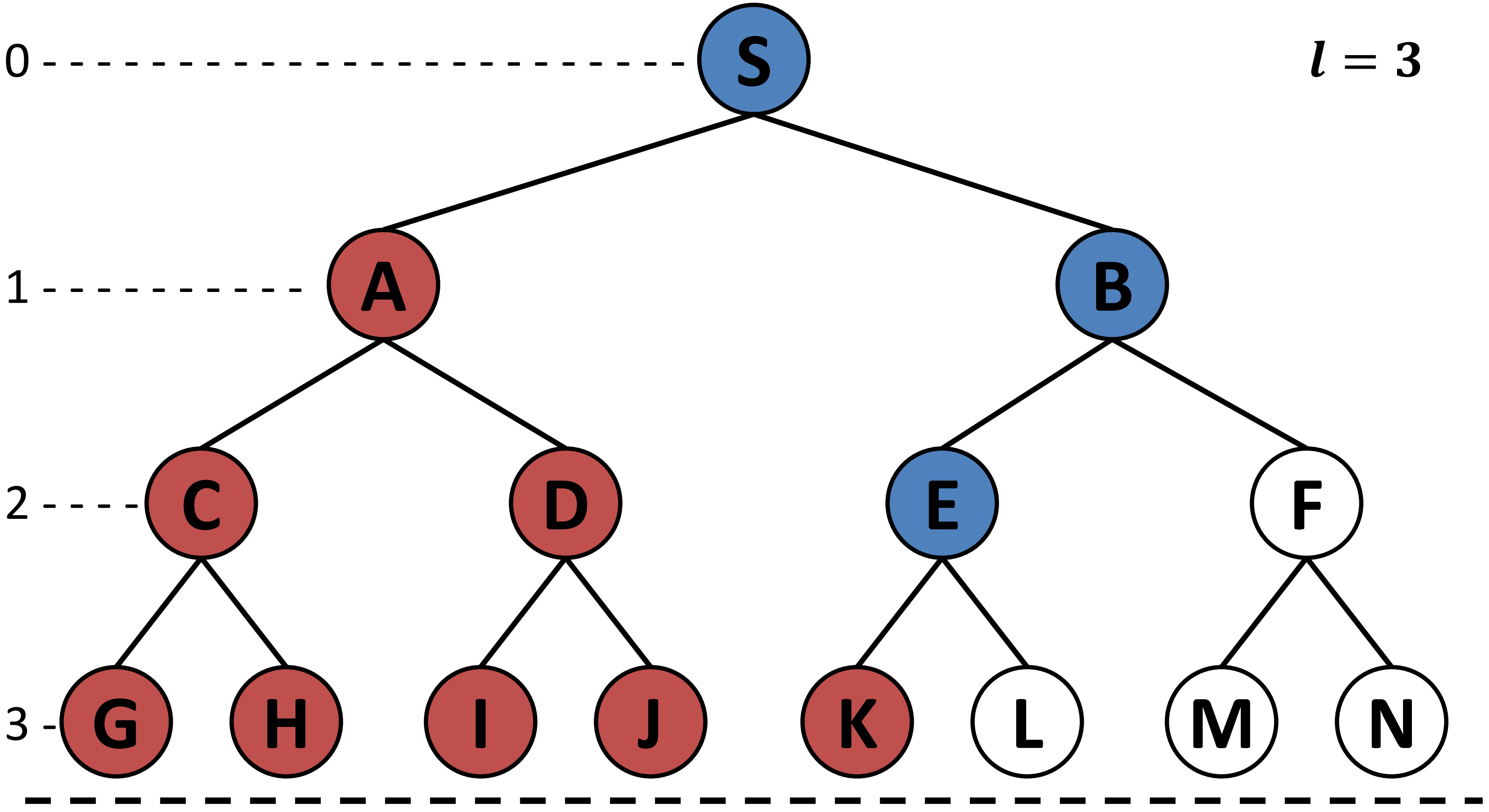
Iterative Deepening Search



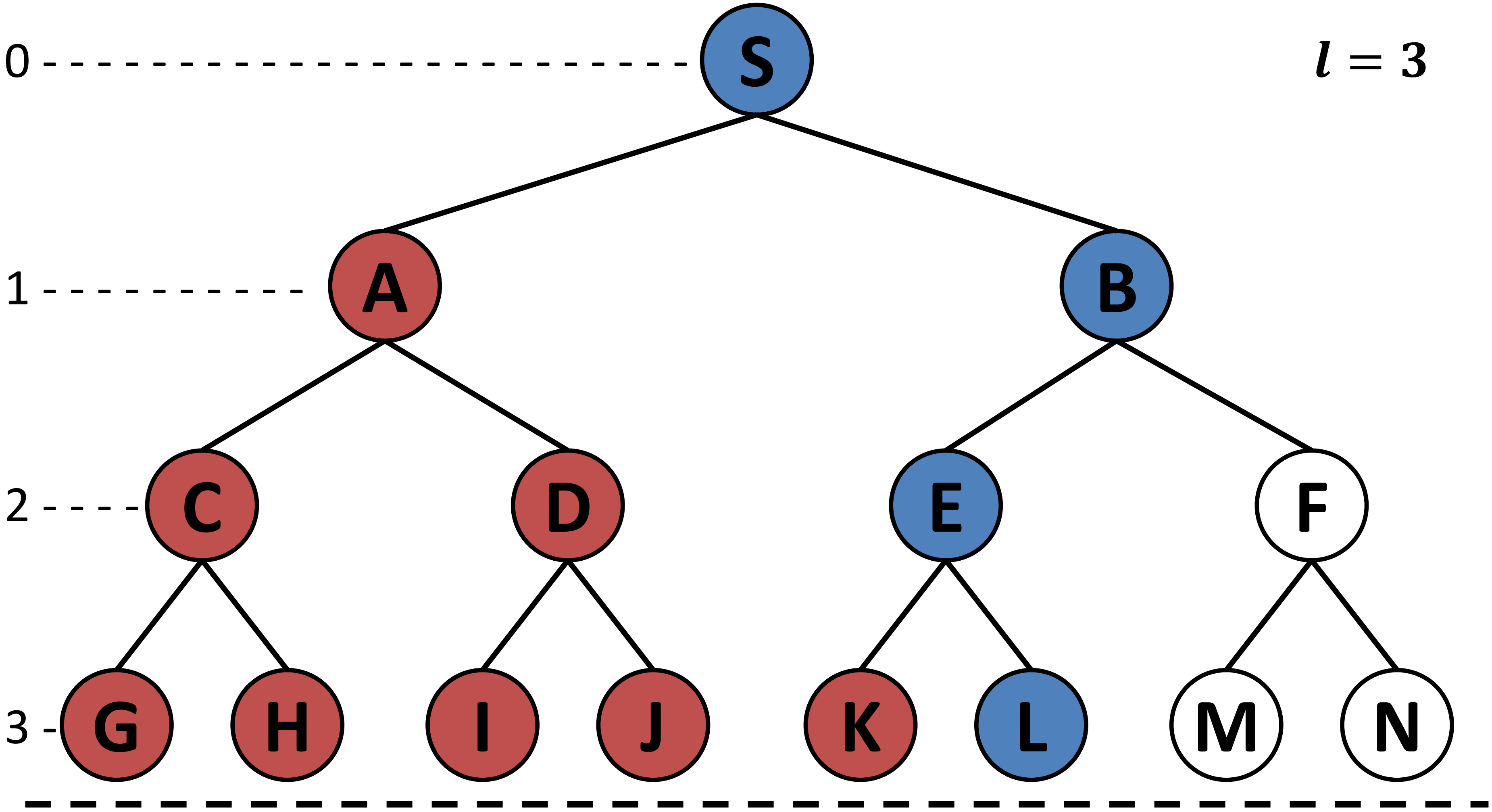
Iterative Deepening Search



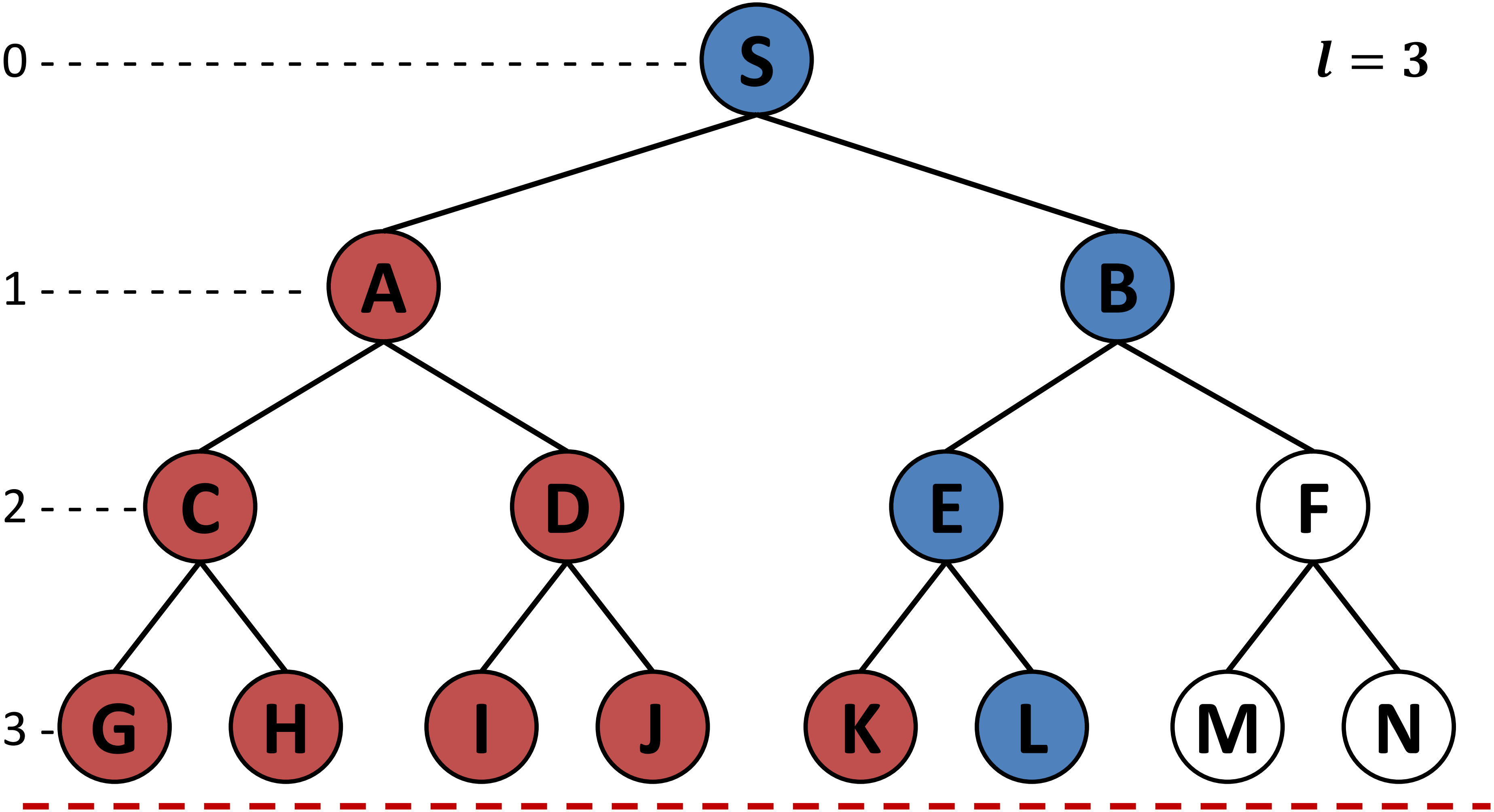
Iterative Deepening Search



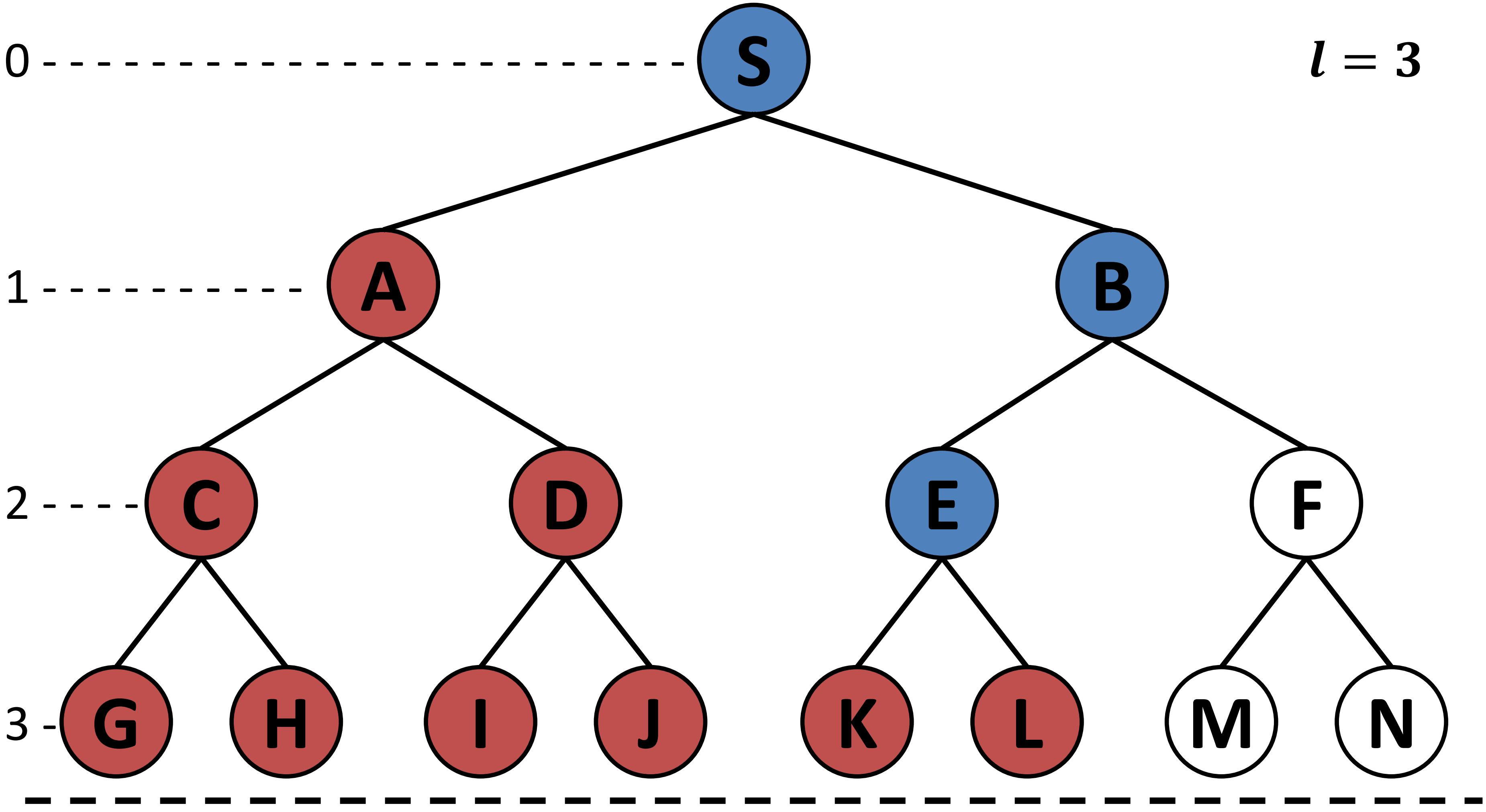
Iterative Deepening Search



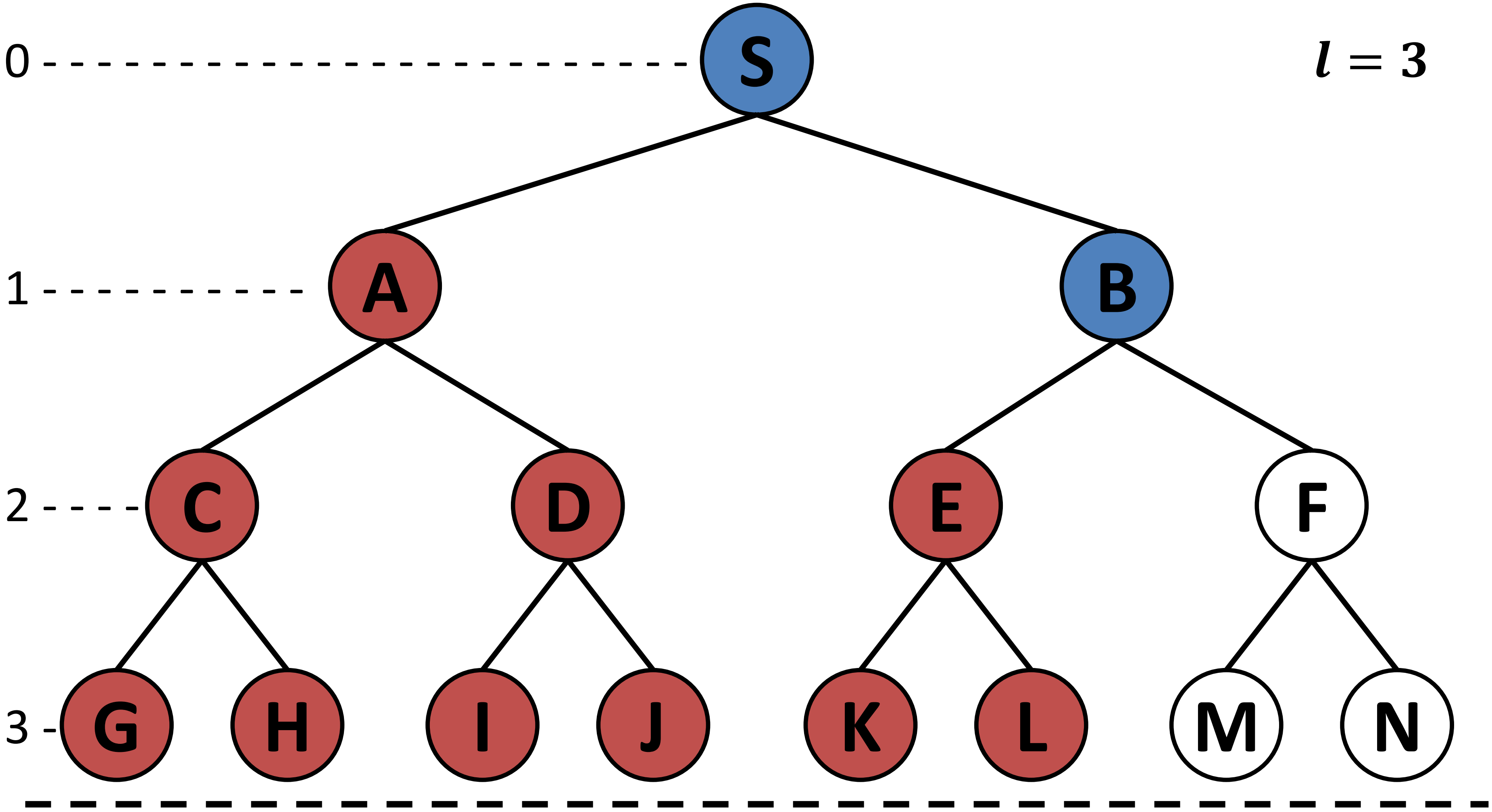
Iterative Deepening Search



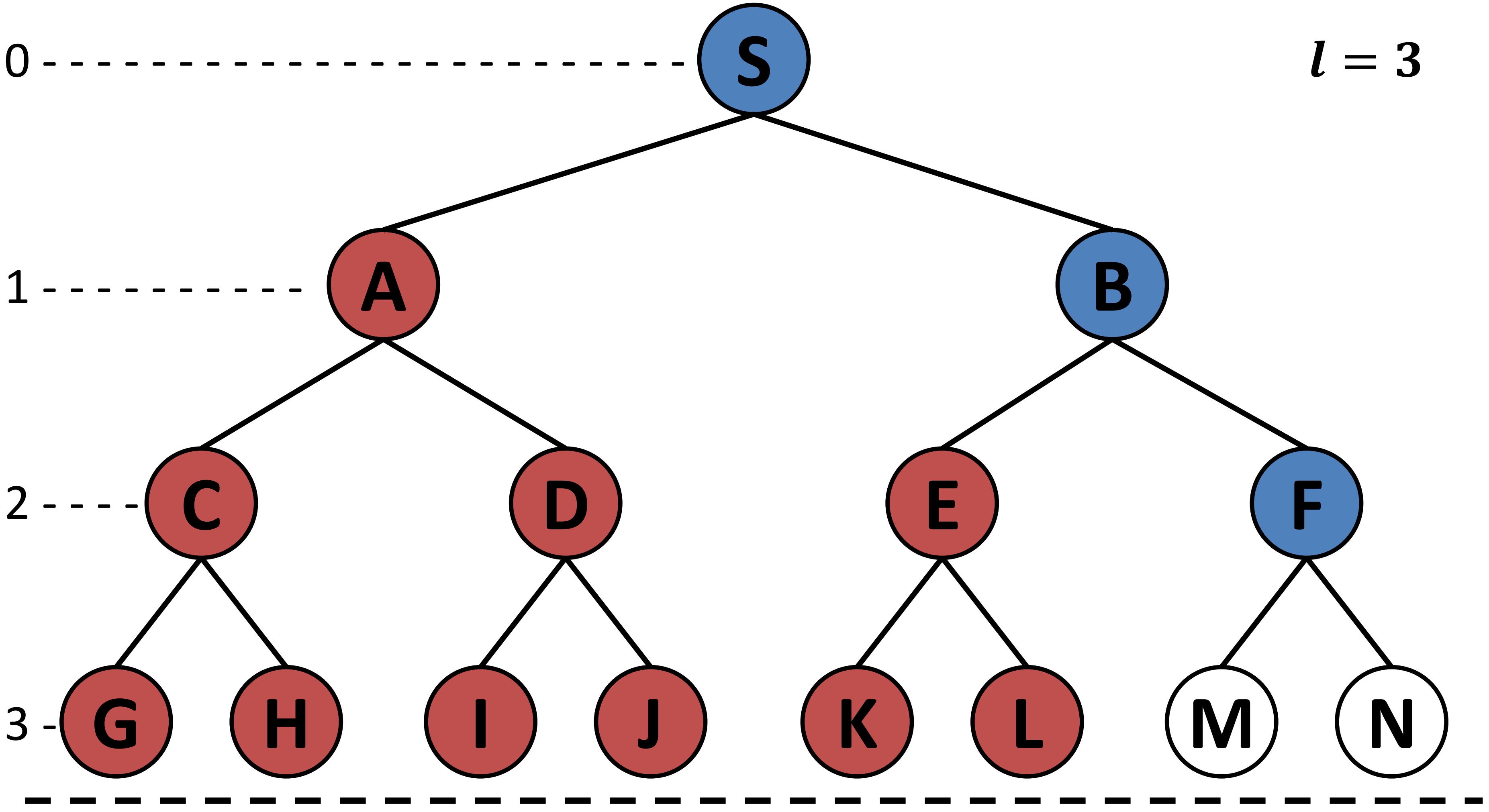
Iterative Deepening Search



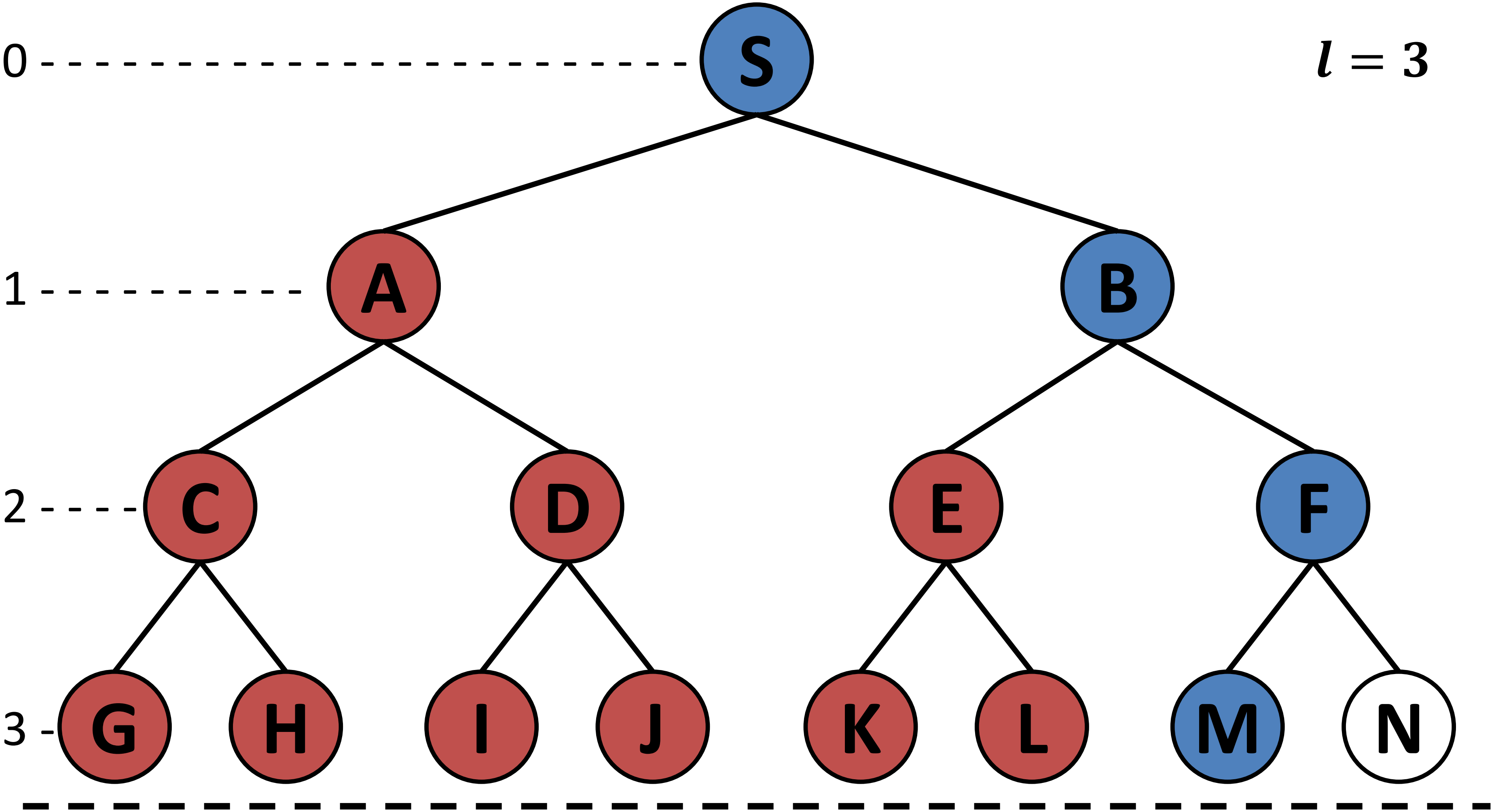
Iterative Deepening Search



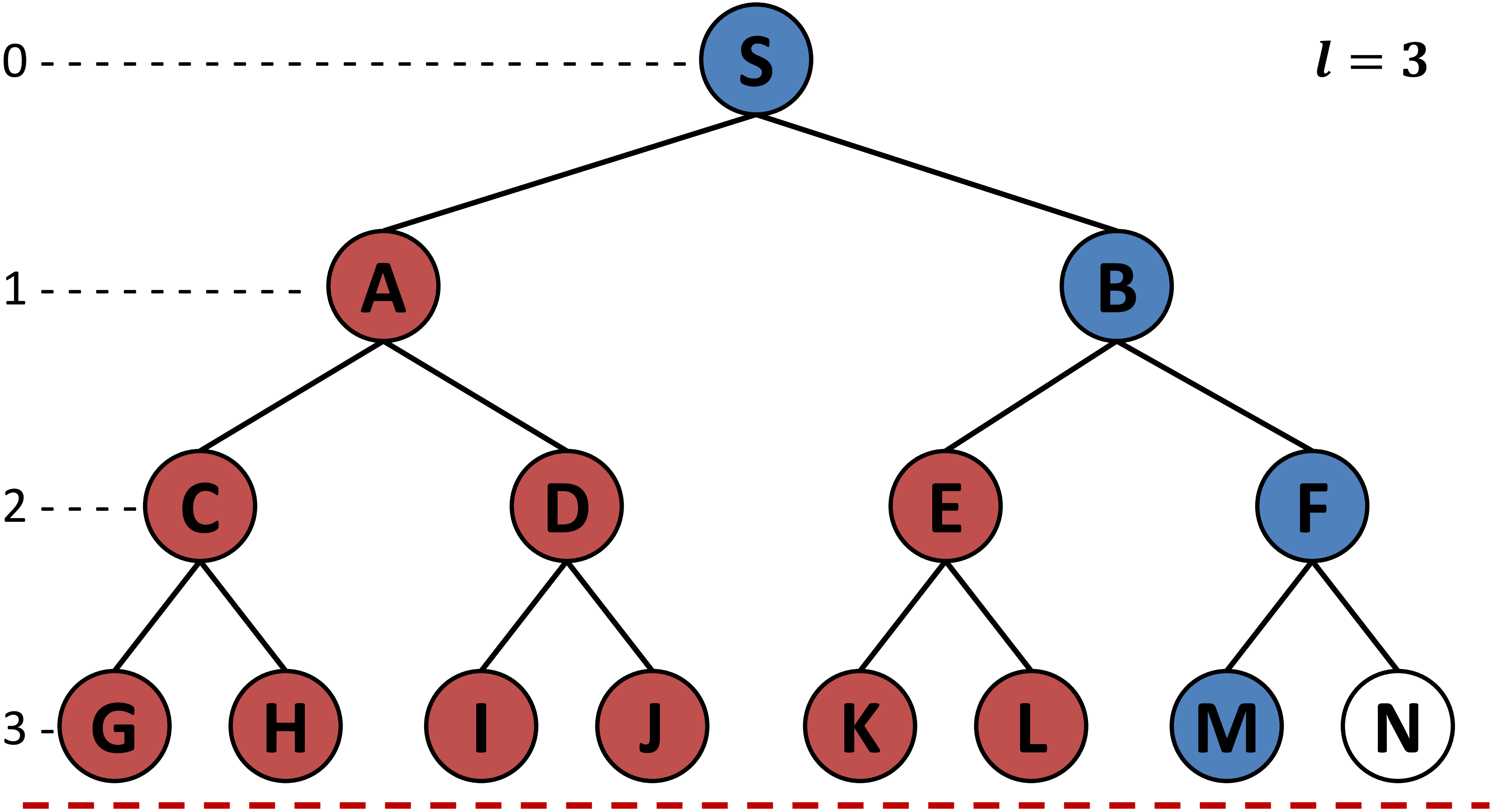
Iterative Deepening Search



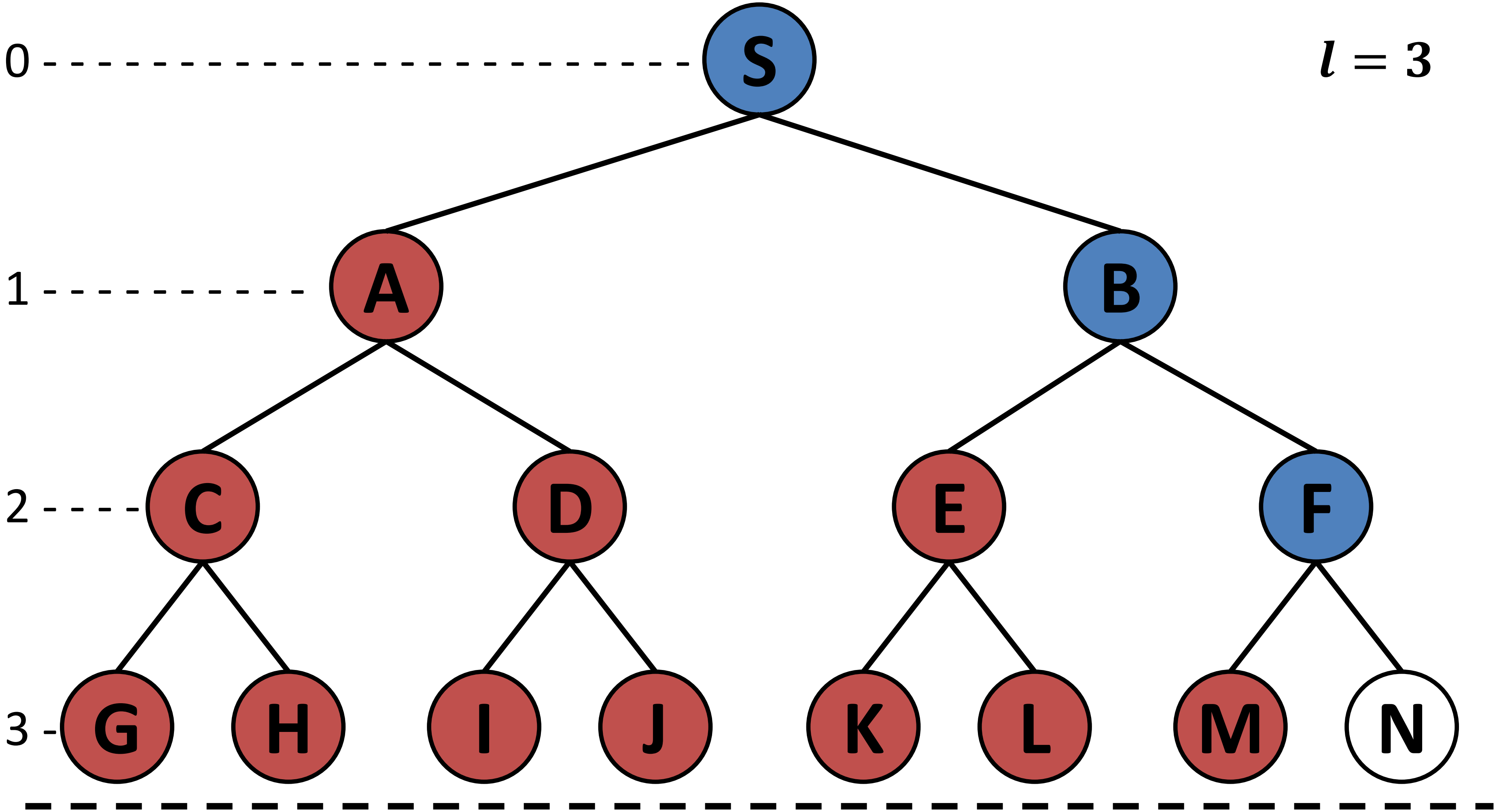
Iterative Deepening Search



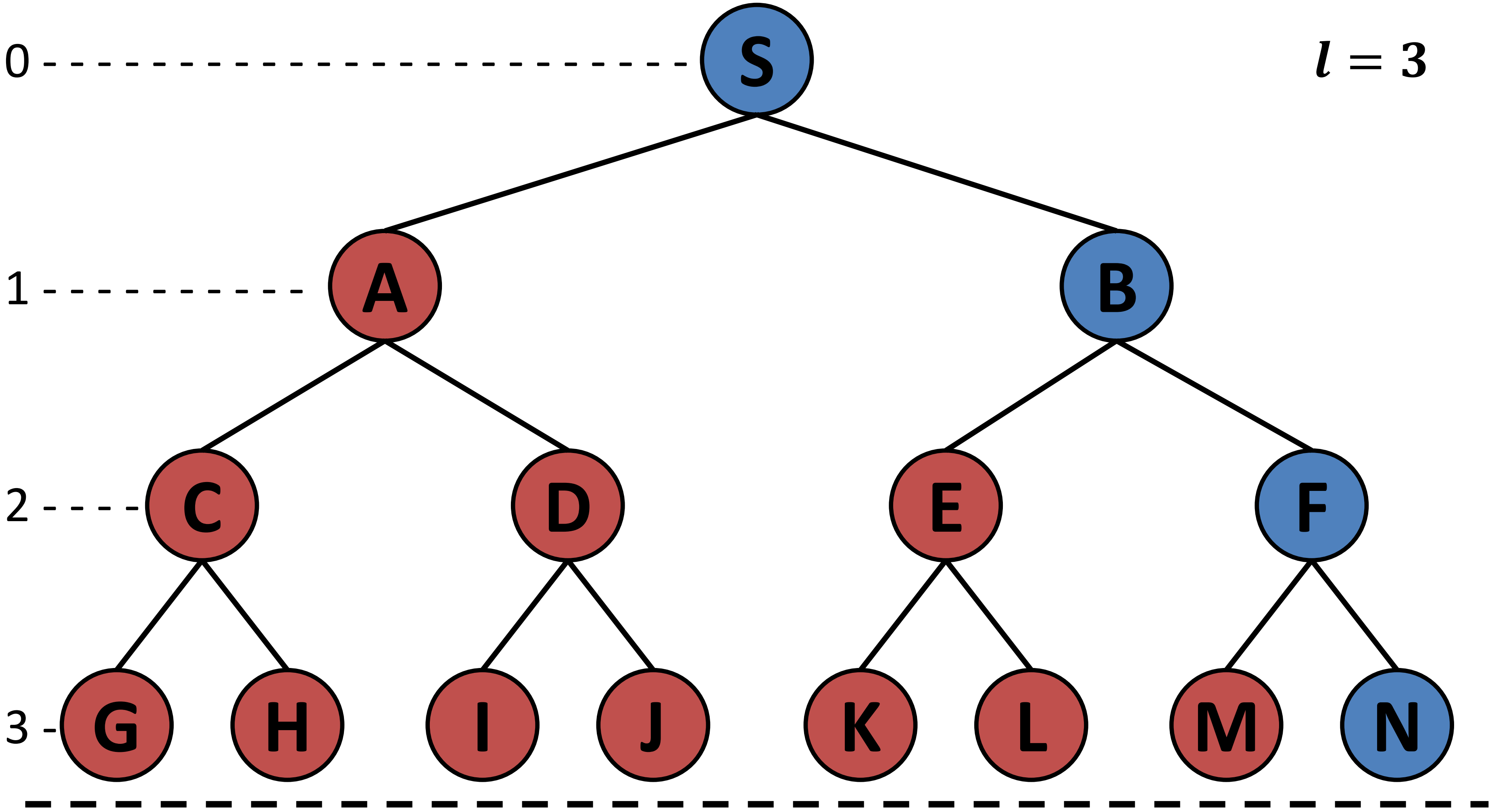
Iterative Deepening Search



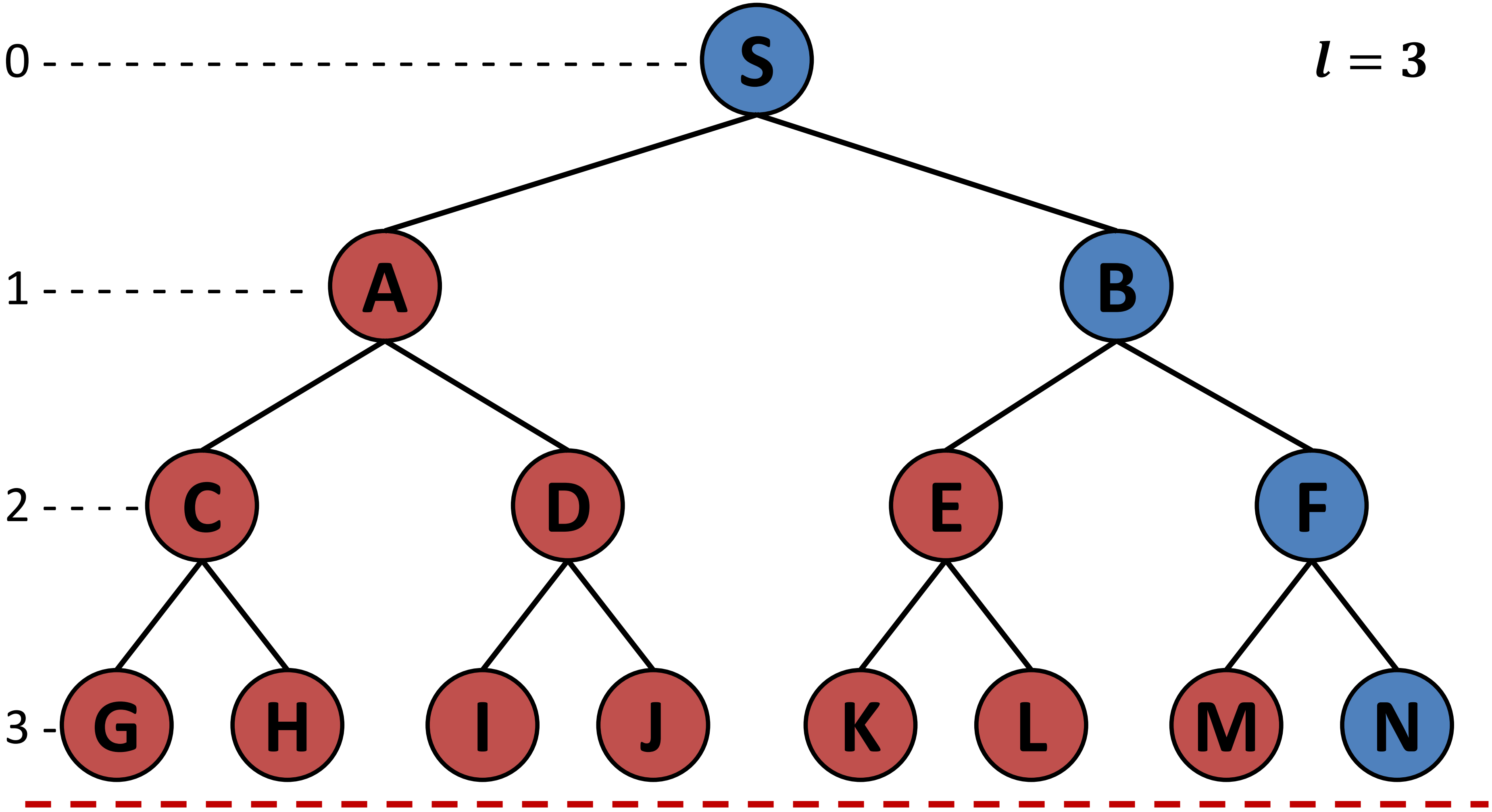
Iterative Deepening Search



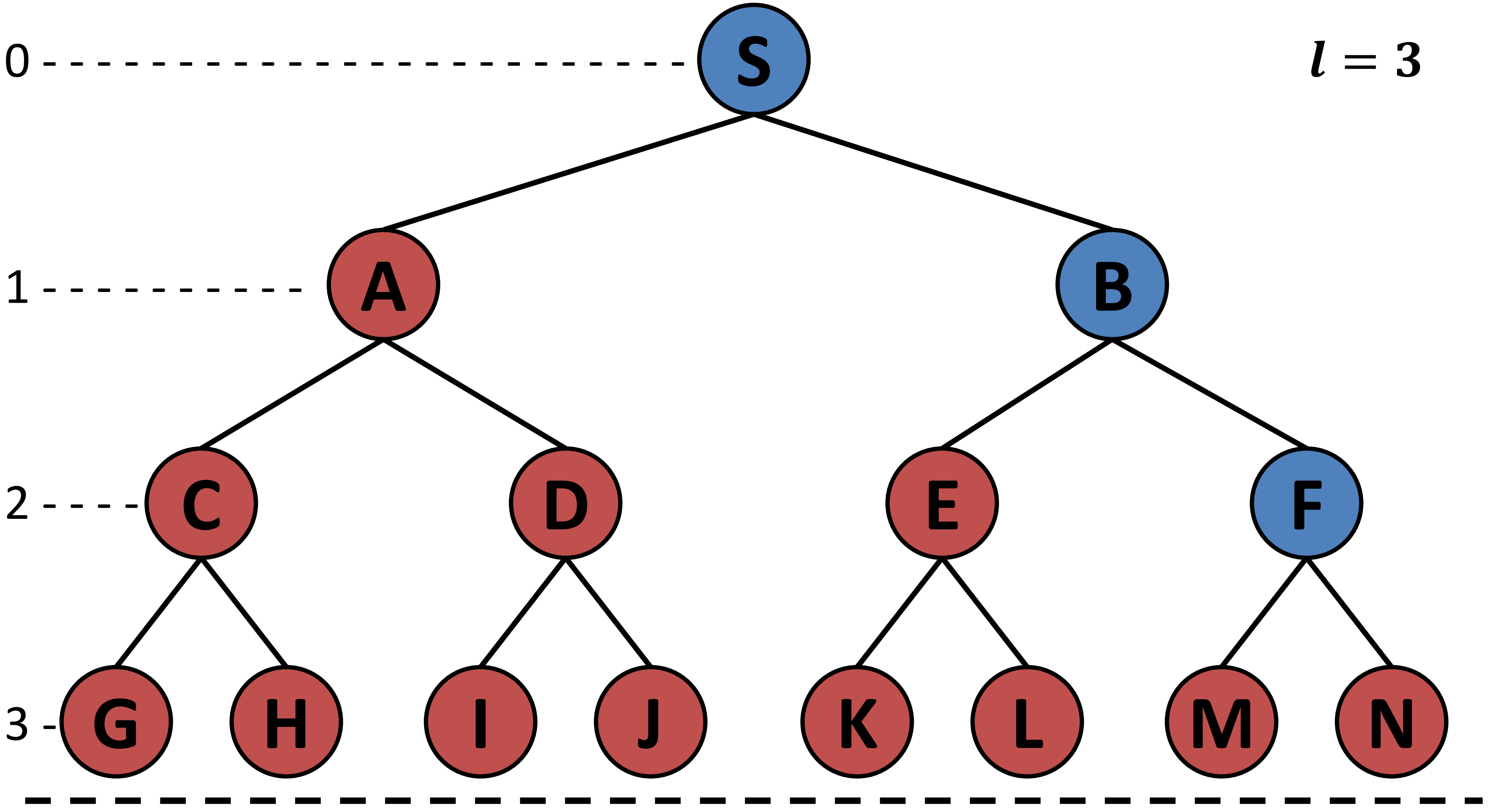
Iterative Deepening Search



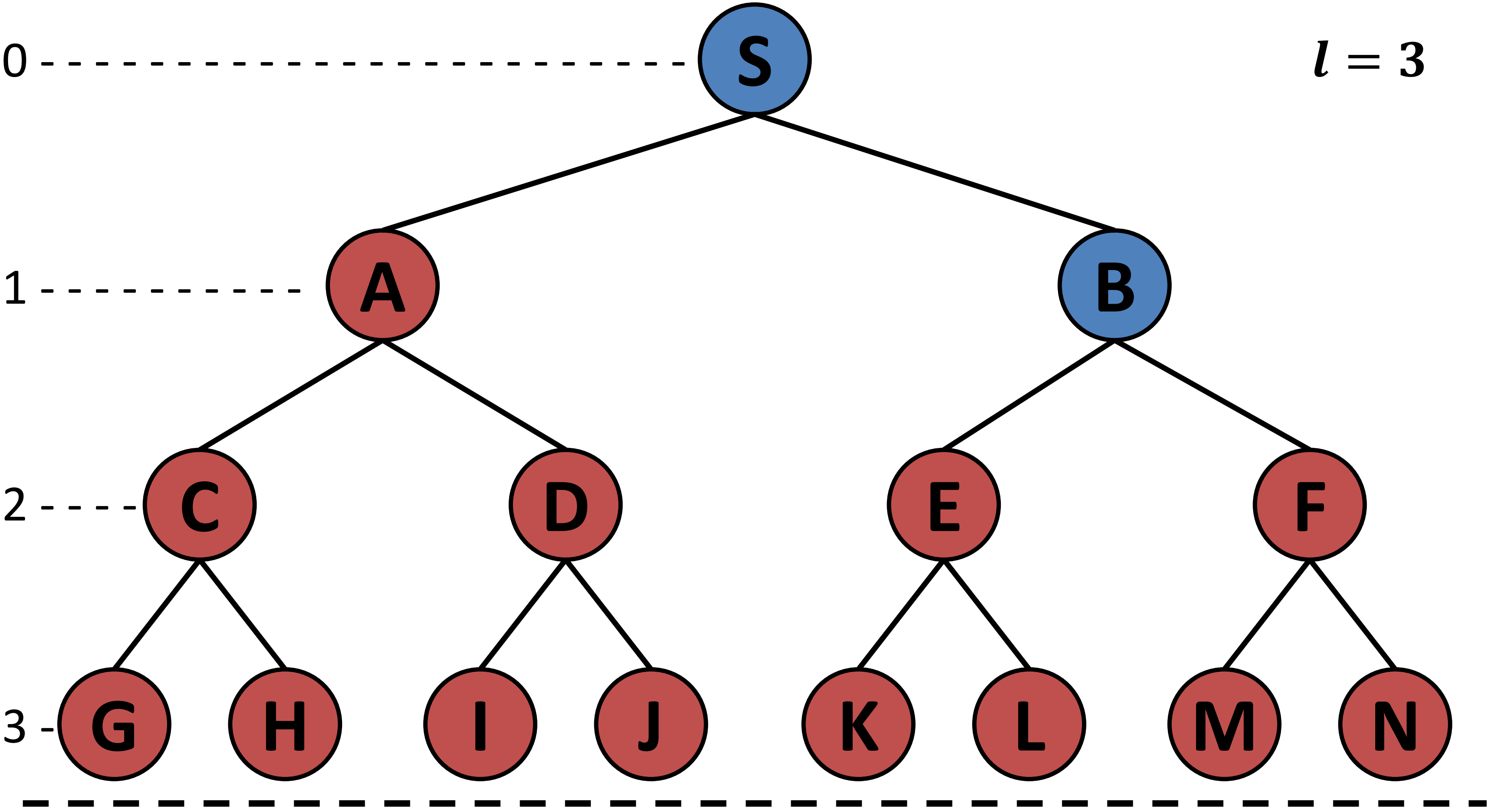
Iterative Deepening Search



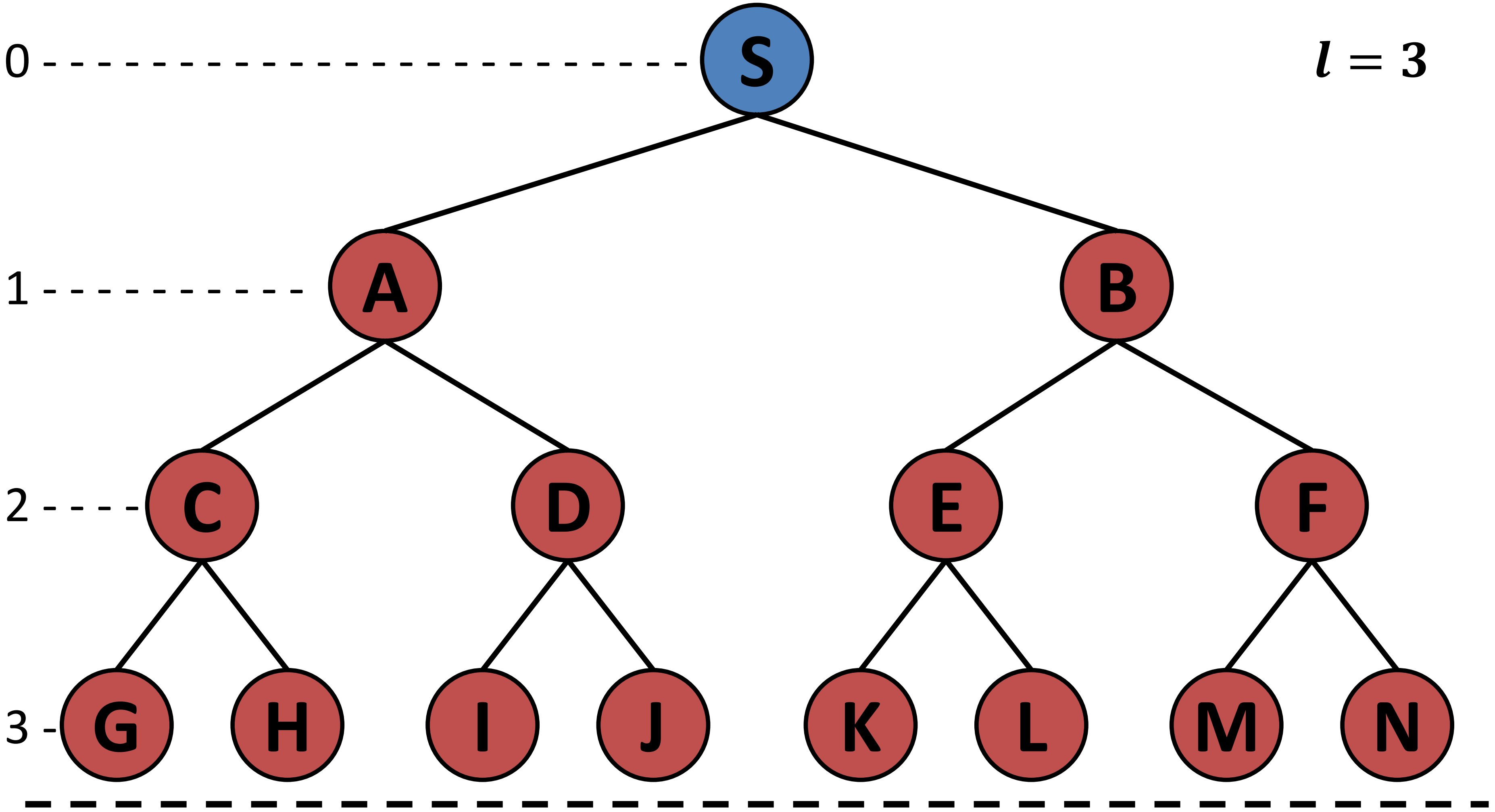
Iterative Deepening Search



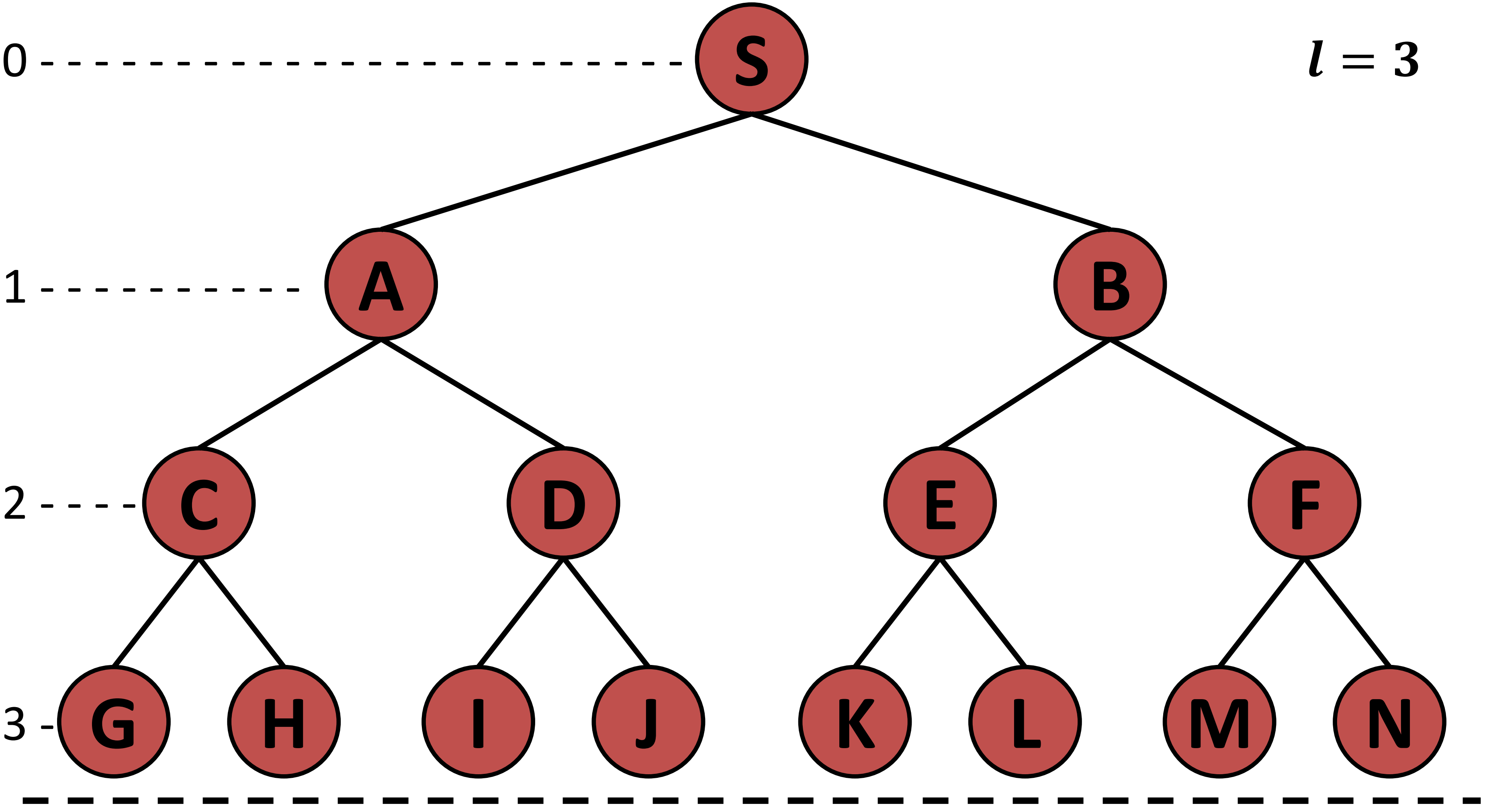
Iterative Deepening Search



Iterative Deepening Search



Iterative Deepening Search



Iterative Deepening Search (IDS)

- ***Complete?***
  - – ***Yes***
- ***Optimal?***
  - – ***Yes***, if *step cost = 1* (Can be modified to explore uniform-cost tree)
- ***Time?***
  - – $\boldsymbol{(d + 1)b^0 + db^1 + (d - 1)b^2 + . . . + b^d = O(b^d)}$
- ***Space?***
  - – ***O(bd)***, i.e., *linear space!*

Iterative Deepening Search (IDS)

Numerical comparison for ***b = 10*** and ***d = 5***, solution at far right leaf:

N(IDS) = 50 + 400 + 3, 000 + 20, 000 + 100, 000 = 123, 450

N(BFS) = 10 + 100 + 1, 000 + 10, 000 + 100, 000 + 999, 990 = 1, 111, 100

** **IDS** does better because other nodes at depth d are not expanded*

** **BFS** can be modified to apply goal test when a node is generated*

Summary of Algorithms

Criterion	**DFS**	**BFS**	**UCS**	**DLS**	**IDS**
Complete	No	Yes* (if ***b*** is finite)	Yes* (if *step cost* ≥ *ε*)	Yes* (if *l ≥ d)*	Yes
Time	b^m	b^{d+1}	$b^{\lceil C*/\mathcal{E}\rceil}$	b^l	b^d
Space	bm	b^{d+1}	$b^{\lceil C*/\mathcal{E}\rceil}$	bl	bd
Optimal	No	Yes* (if *step cost = 1*)	Yes	No	Yes* (if *step cost = 1*)
Data Structure	*Stack*	*Queue*	*Priority Queue*	*Stack*	*Stack*

Homework: *“Frog Leap Puzzle”*

- The game has ***2N + 1*** fields of play. In the beginning of the game there are ***N*** frogs which are looking to the left and placed on the rightmost ***N*** fields, and ***N*** frogs which are looking to the right and placed on the leftmost fields. The aim of the game is to swap the places of the frogs and reach the opposite configuration.
- The rules of play are as follows: Each frog can only move in the direction it looks at. Every frog can jump in a free field in front of it or jump over a frog to go to an empty field in front of it.
- **Input**: ***N*** - number of frogs looking in one direction
- **Output**: All the configurations that needed to go through to get from the start to the final state.

Homework: *“Frog Leap Puzzle”*

- **Sample Input:**

```
2
```

- **Sample Output:**

```
>>_<<
>_><<
><>_<
><><_
><_<>
_<><>
<_><>
<<>_>
<<_>>
```

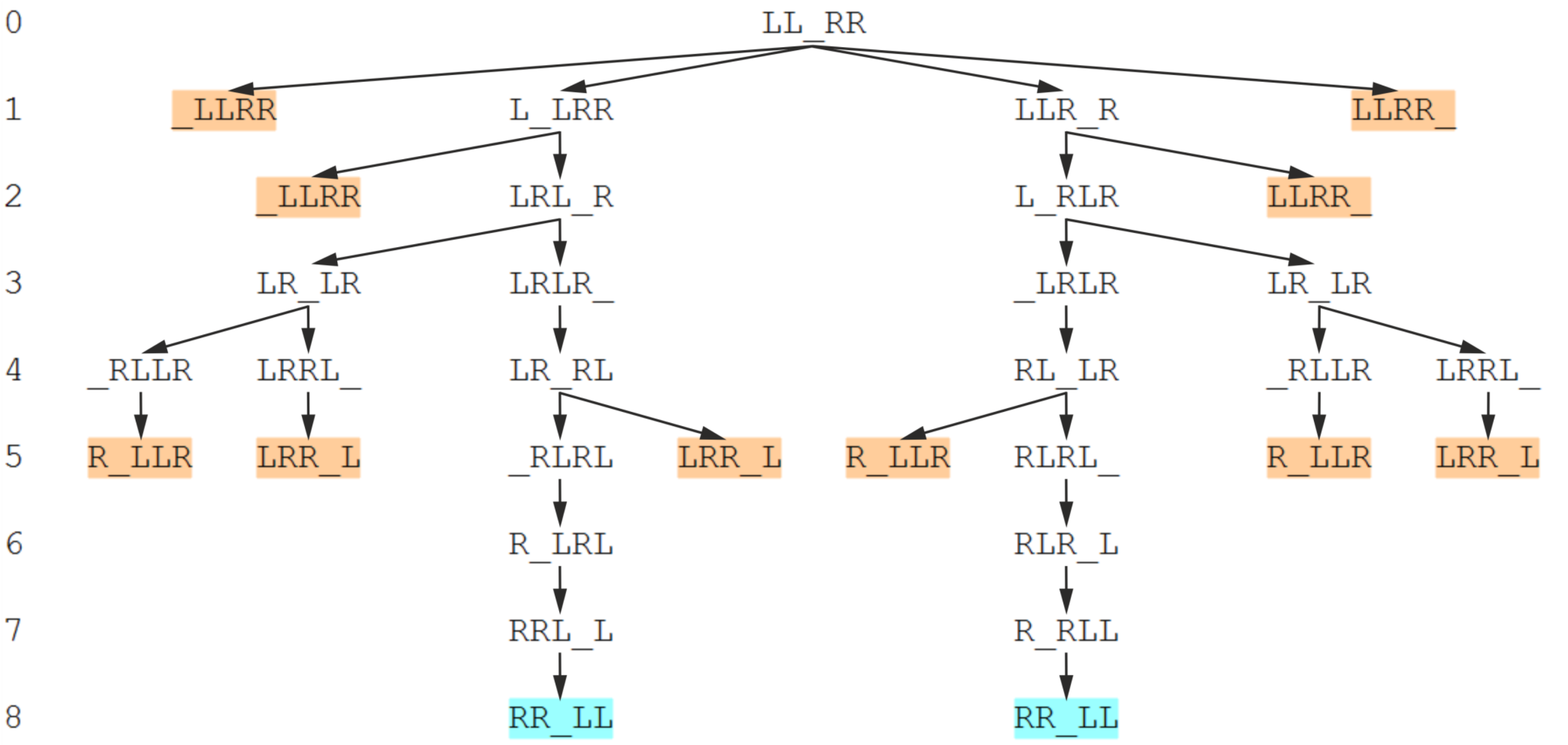
Frog Leap Puzzle

- How can we solve the problem?
- Can we solve the problem using *uninformed search algorithm*?
  - If *"Yes"*, how can we represent the puzzle as a *tree* or a *graph*?
  - Which *uninformed search algorithm* is best? Why?
- Can we solve the problem in a better way?
- What about a *linear time* solution?

Frog Leap Puzzle

- *Initial State*: ***LL_RR***
- *Successor Function* (moves):
  - Left moves (only ***R***)
    - Jump: ***LL_RR*** → ***LLR_R***
    - Double Jump: ***LL_RR*** → ***LLRR_***
  - Right moves (only ***L***)
    - Jump: ***LL_RR*** → ***L_LRR***
    - Double Jump: ***LL_RR*** → ***_LLRR***
- *Goal State*: ***RR_LL***

Frog Leap Puzzle



0 LL_RR

1 _LLRR L_LRR LLR_R LLRR_

2 _LLRR LRL_R L_RLR LLRR_

3 LR_LR LRLR_ _LRLR LR_LR

4 _RLLR LRRL_ LR_RL RL_LR _RLLR LRRL_

5 R_LLR LRR_L _RLRL LRR_L R_LLR RLRL_ R_LLR LRR_L

6 R_LRL RLR_L

7 RRL_L R_RLL

8 RR_LL RR_LL

Frog Leap Puzzle

```
n = readInput();
size = (n * 2) + 1;
board = new board[size];
function dfs(board, zeroState) {
      if isGoalState(board, zeroState)
            return TRUE;
      for move in moves(board, zeroState) {
            if dfs(move.board, move.zeroState) {
                  print(move.board);
                  return TRUE;
            }
      }
      return FALSE;
}
```