COMP 8620
Advanced Topics in AI

Lecturers:
Philip Kilby &
Jinbo Huang

Part 1: Search

Lecturer: Dr Philip Kilby

Philip.Kilby@nicta.com.au

Weeks 1-7

(Week 7 is assignment seminars)

Part 2: Probabilistic Reasoning with Bayesian networks

Lecturer: Dr Jinbo Huang

Jinbo.Huang@nicta.com.au

Weeks 8-13

Course Outline – Part 1

1-2 Introduction

Systematic Search

3-4 Linear Programming /

Integer Programming /
Branch and Bound

5-6 Neighbourhood-based methods:

- Simulated Annealing
- Tabu Search
- Genetic Algorithms

7-8 Constraint Programming
(Guest lecturer: Jason Li)

7-8 Constraint Programming
(Guest lecturer: Jason Li)

9-10 Case Studies:

- TSP
- Planning (Jussi Rintanen)

11 Dynamic Programming

12 Qualitative Reasoning
(Guest lecturer: Jason Li)

13-14 Seminars

(Guest Lecturers: You lot!)

Assessment

- 40% Exam
- 60% Assignment

- Search
  - 1 x Assignment (next week) 15%
  - 1 x Seminar 15%

- Reasoning
  - 3 x Assignment (10% each) *(tentative)*

What is Search?

- Search finds a *solution* to a *problem*

- Landscape of solutions
- Some are “good”
- Lots (and lots and lots) are “bad”

How do we find the good ones?

What is Search?

- Landscape can continuous or discrete
- (We will only deal with discrete)
- Landscape can be (approximated by?) a *tree*
- Landscape can be (approximated by?) a *graph*

What is Search?

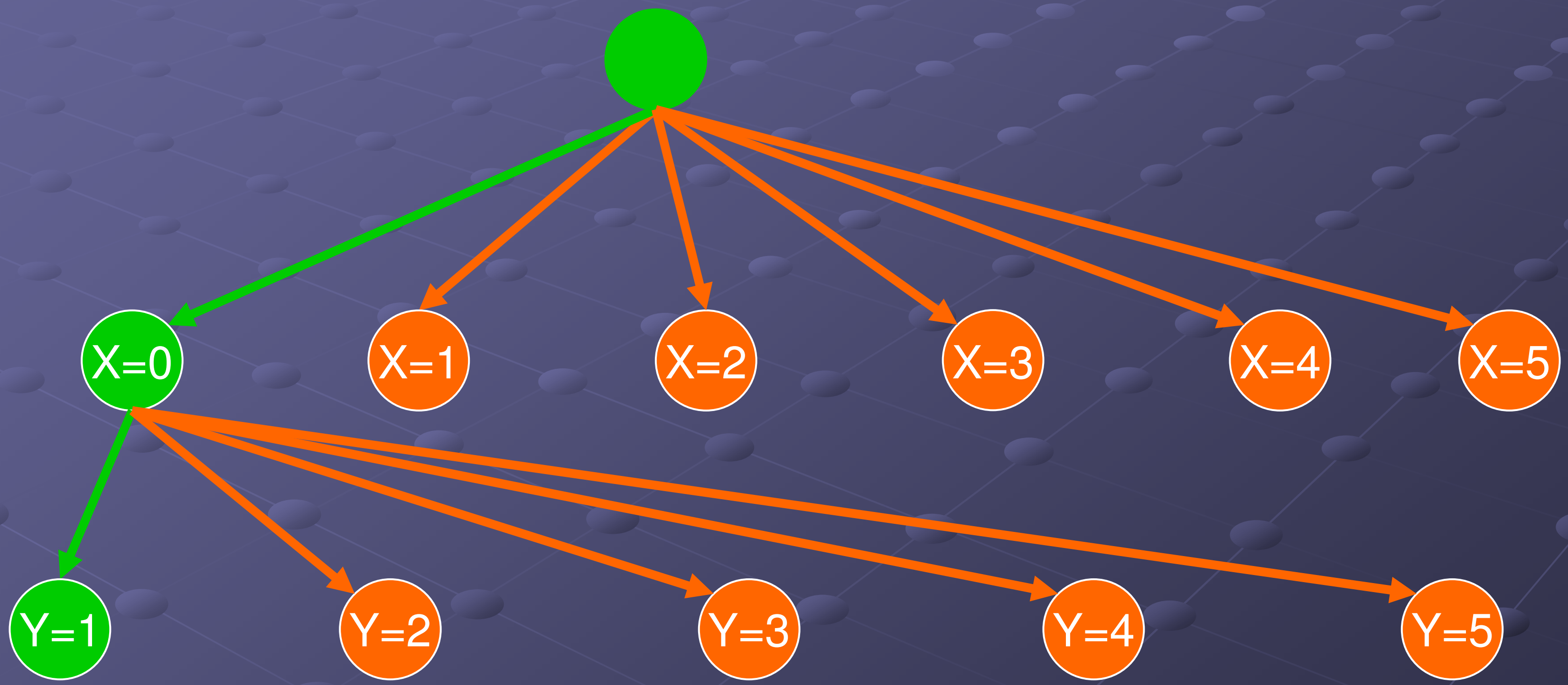
$X + 2 * Y < 3, \quad X \in [0,5] \quad Y \in [1,5]$

What is Search?

$X + 2 * Y < 3, \quad X \in [0,5] \quad Y \in [1,5]$

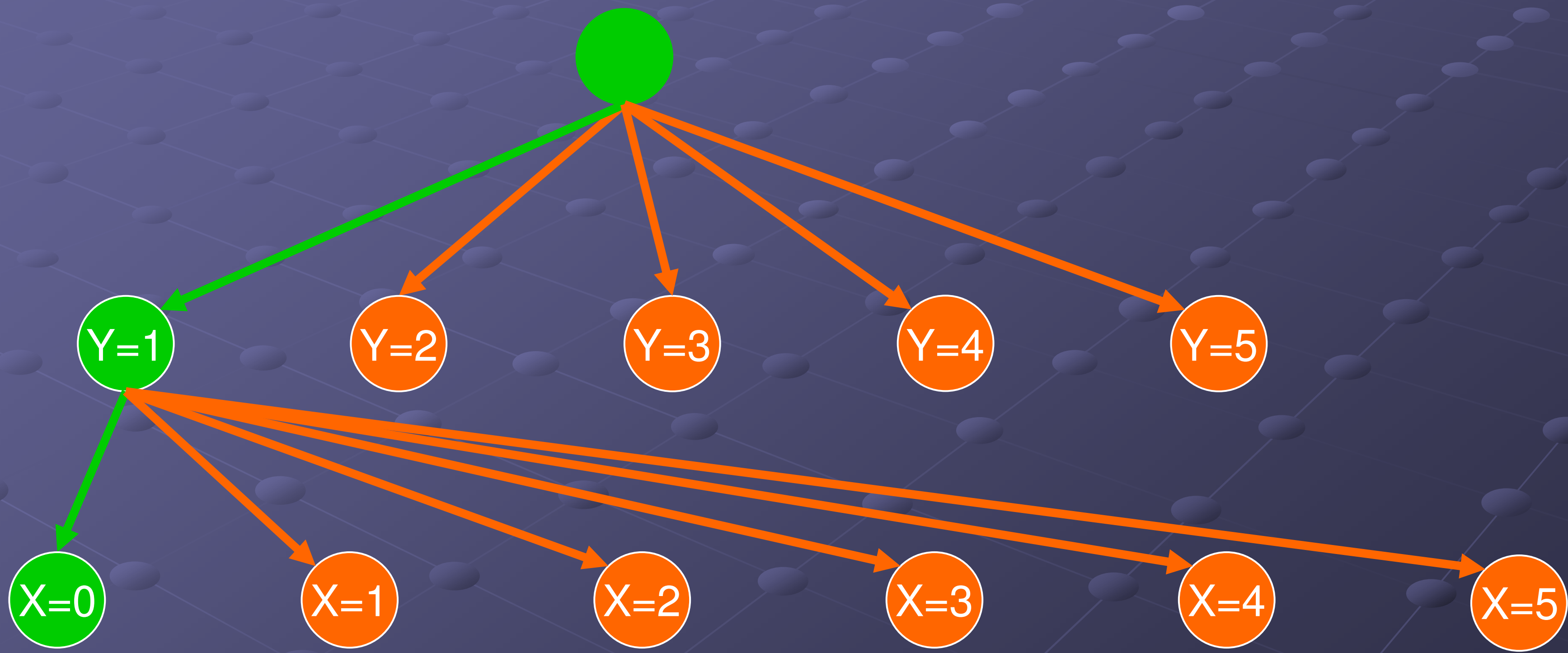
What is Search?

X + 2 * Y < 3,   X ∈ [0,5] Y ∈ [1,5]



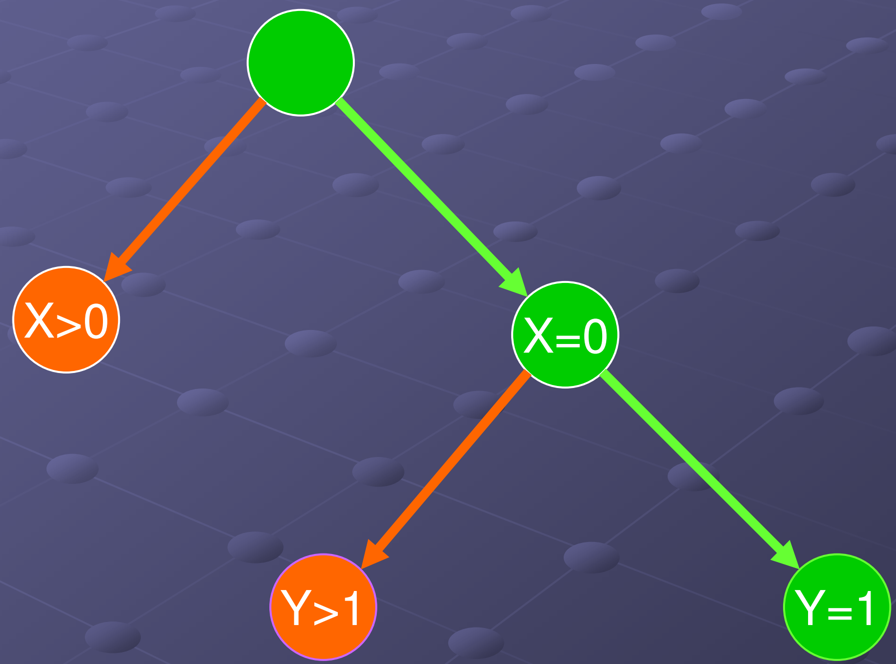
What is Search?

$X + 2 * Y < 3, \quad X \in [0,5] \quad Y \in [1,5]$



What is Search?

$X + 2 * Y < 3, \quad X \in [0,5] \quad Y \in [1,5]$



What is Search?

- Different flavours
  - Find a solution (Satisfaction/Decision)
  - Find the *best* solution (Combinatorial Optimisation)
- Decision problems:
  - Search variant: Find a solution (or show none exists)
  - Decision variant: Does a solution exist?

What is Search

- Constructive Search – Find a solution by construction
- Local Search – Given a solution, explore the "neighbourhood" of that solution to find other (possibly better) solutions

What is Search?

Anytime algorithm (for optimisation prob):

- After a (short) startup time, an answer is available
- The longer the alg is run, the better the solution
- Quality guarantee may be available (e.g. solution is within 5% of optimal)

One-shot algorithm

- Answer only available at completion

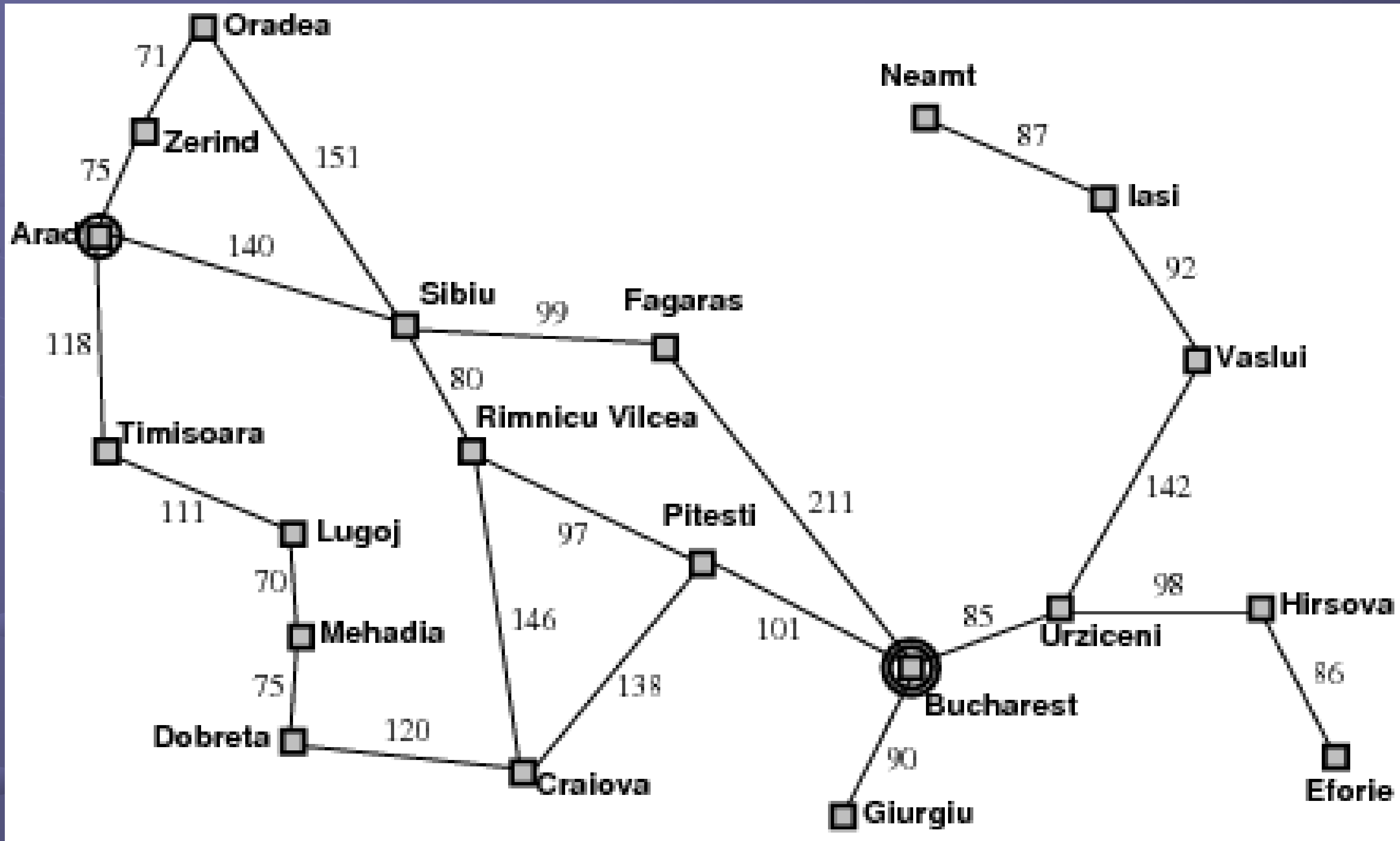
What is Search?

A problem is defined by

- Initial state
- Successor function
  (an action leading from one state to another)
- Goal test
- Path cost

- A solution is a sequence of actions leading from the start state to a goal state

Example: Romania



Problem formulation

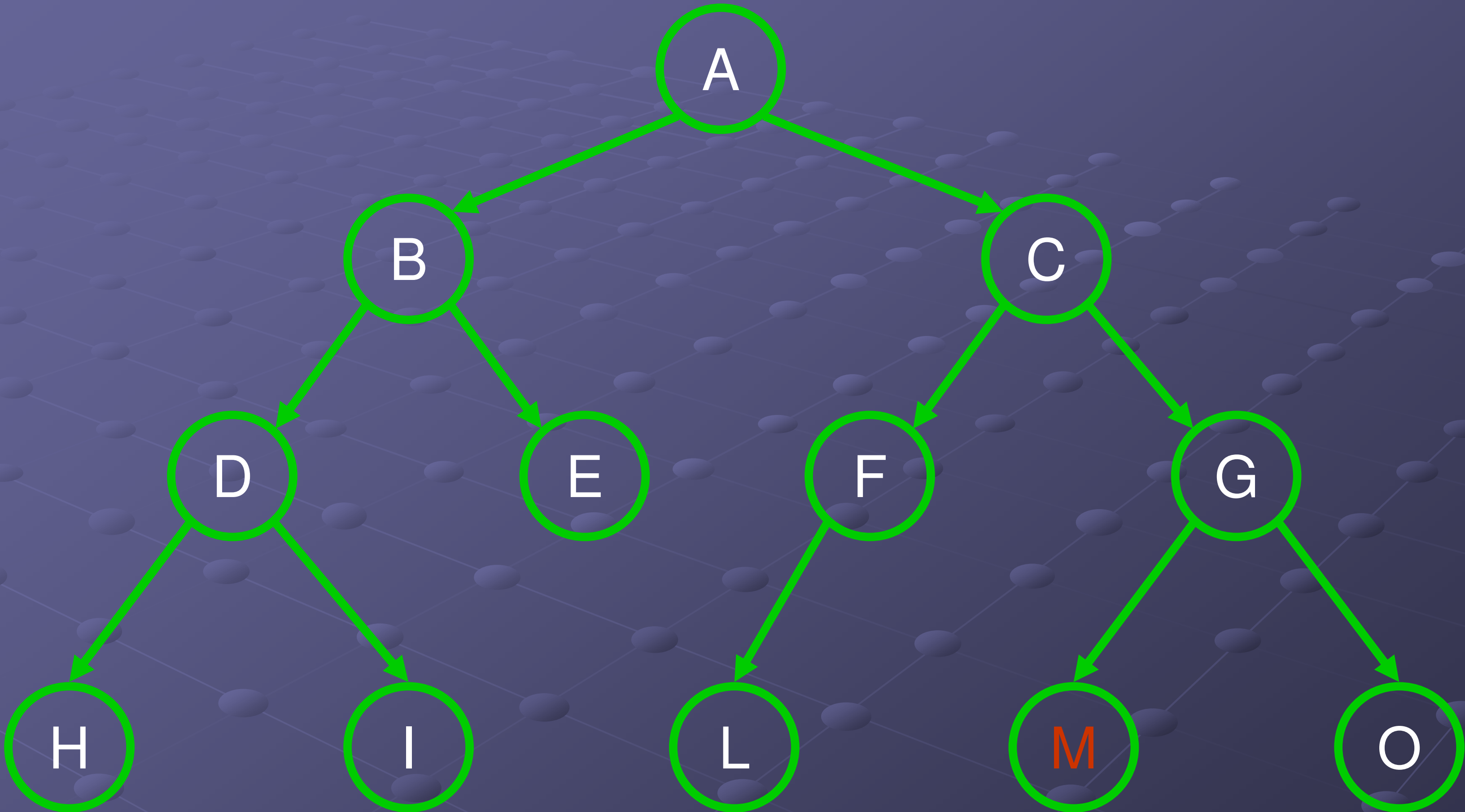
- A problem is defined by four items:
- initial state
  - e.g., “at Arad”
- successor function S(x)
  - set of action–state pairs
  - e.g., S(Arad) = {<Arad → Zerind, at Zerind>, . . .}
- goal test e.g., x = “at Bucharest”
  - can be implicit, e.g., HasAirport(x)
- path cost (additive)
  - e.g. sum of distances, number of actions executed, etc.
  - c(x, a, y) is the step cost, assumed to be ≥ 0

Tree Search

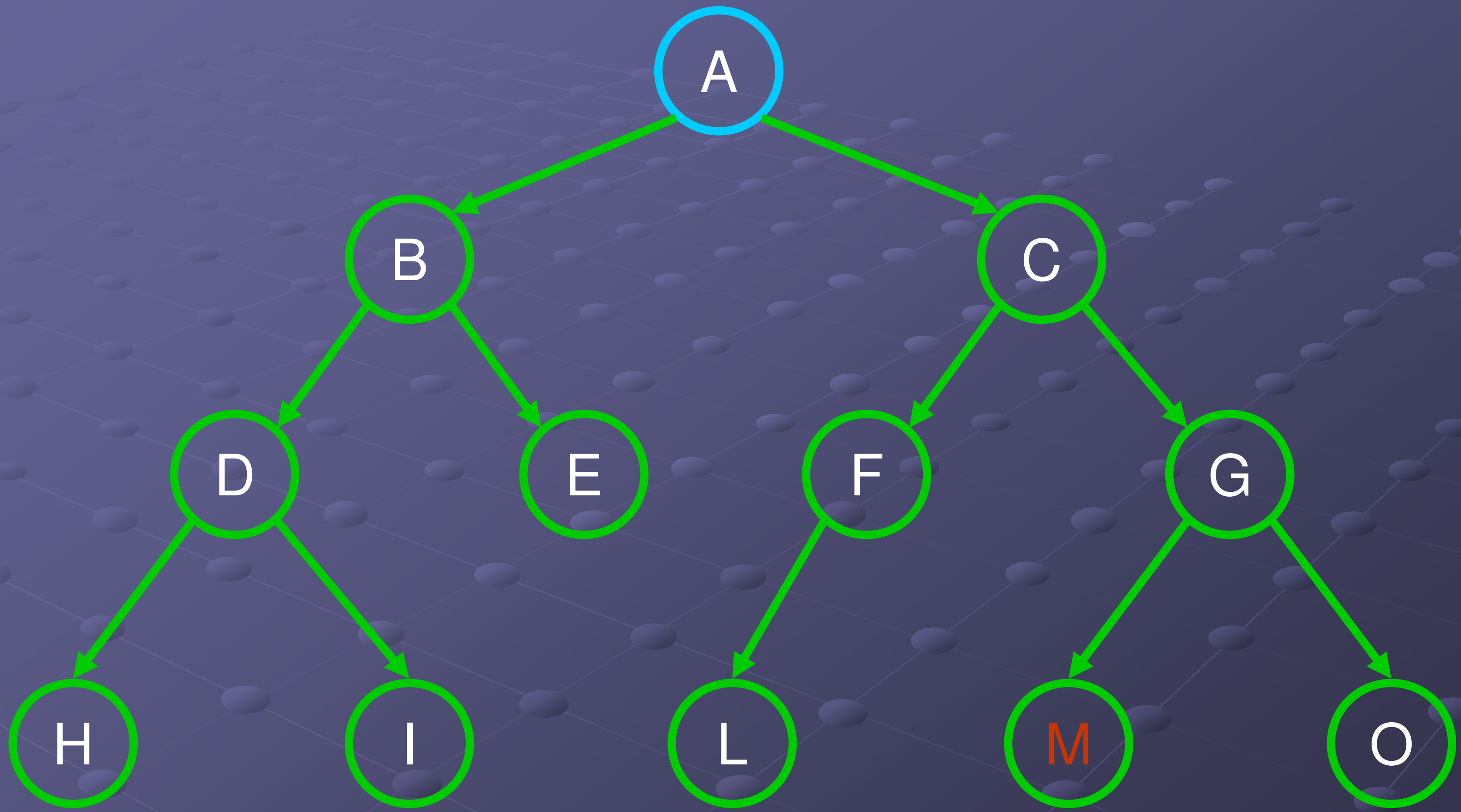
```
function Tree-Search (problem, fringe) returns a solution, or failure
    fringe ← Insert (Make-Node (Initial-State (problem)), fringe)
    loop do
        if Is-Empty (fringe) then return failure
        node ← Remove-Front (fringe)
        if Goal-Test (problem, State[node]) then return node
        fringe ← InsertAll (Expand (node, problem), fringe)


function Expand (node, problem) returns a set of nodes
    successors ← the empty set
    for each (action, result) in Successor-Fn (problem, State[node]) do
        s ← a new Node
        Parent-Node[s] ← node; Action[s] ← action; State[s] ← result
        Path-Cost[s] ← Path-Cost[node] + Step-Cost(State[node], action, result)
        Depth[s] ← Depth[node] + 1
        successors ← Insert (s)
    return successors
```

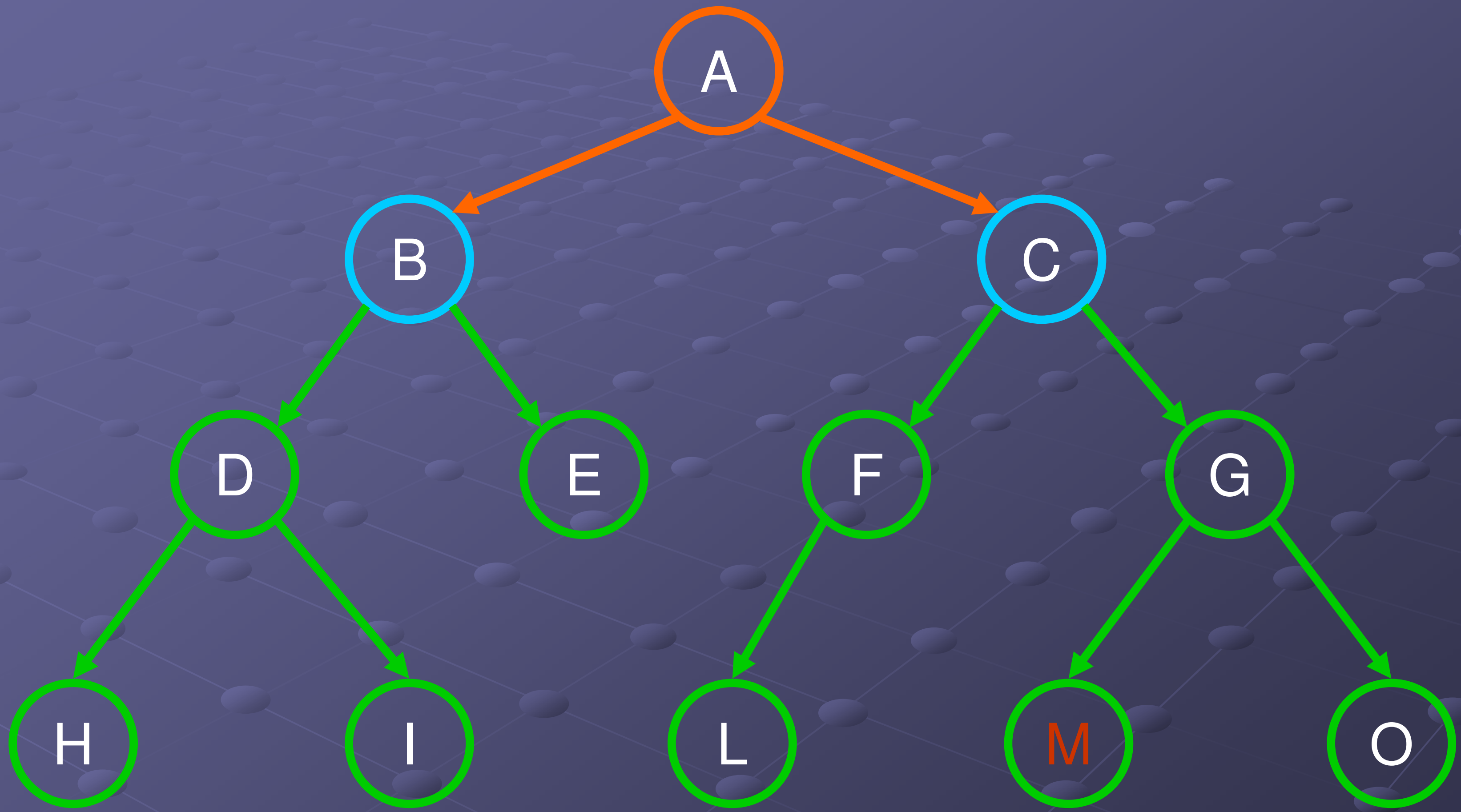
Breadth-first Search



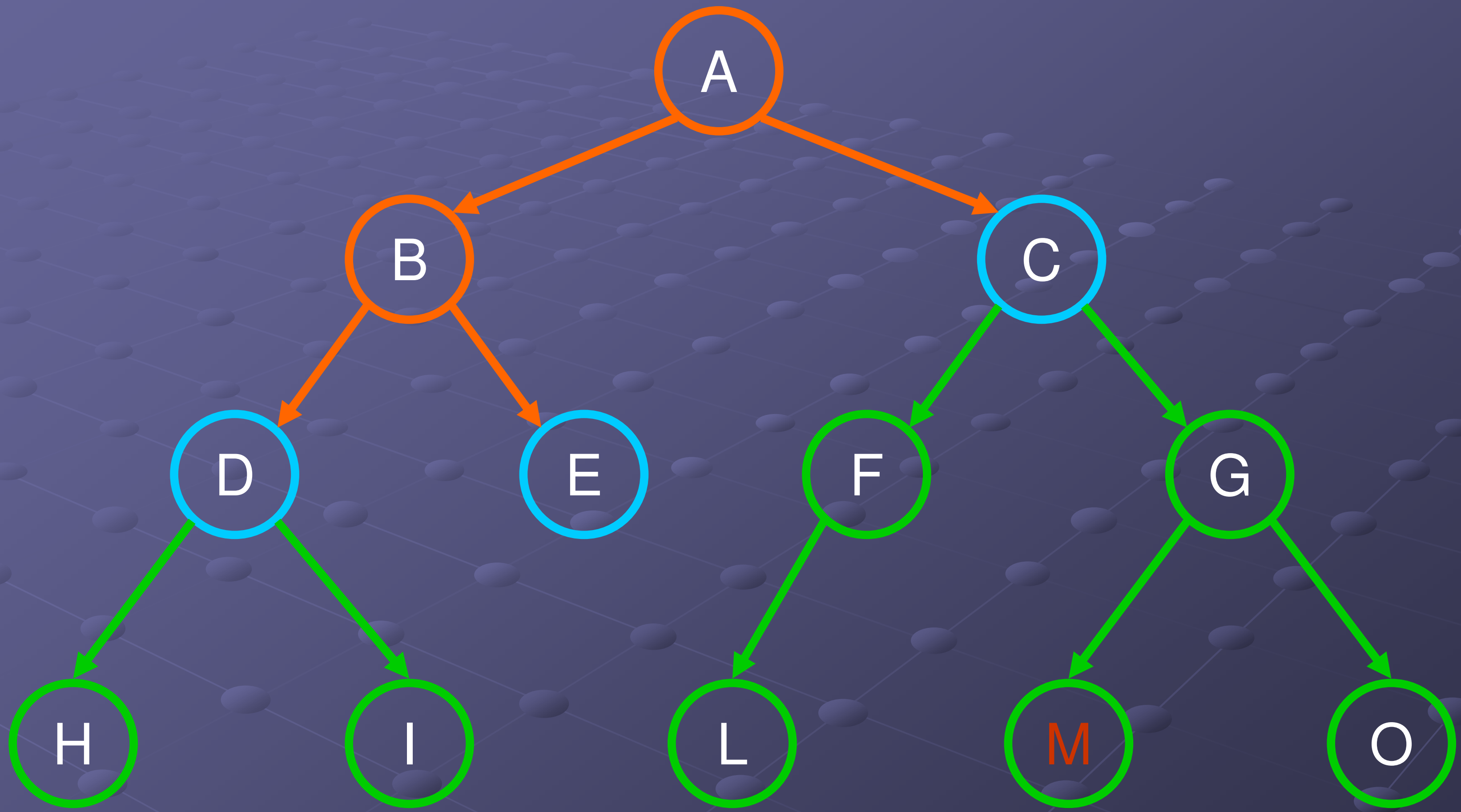
Breadth-first Search



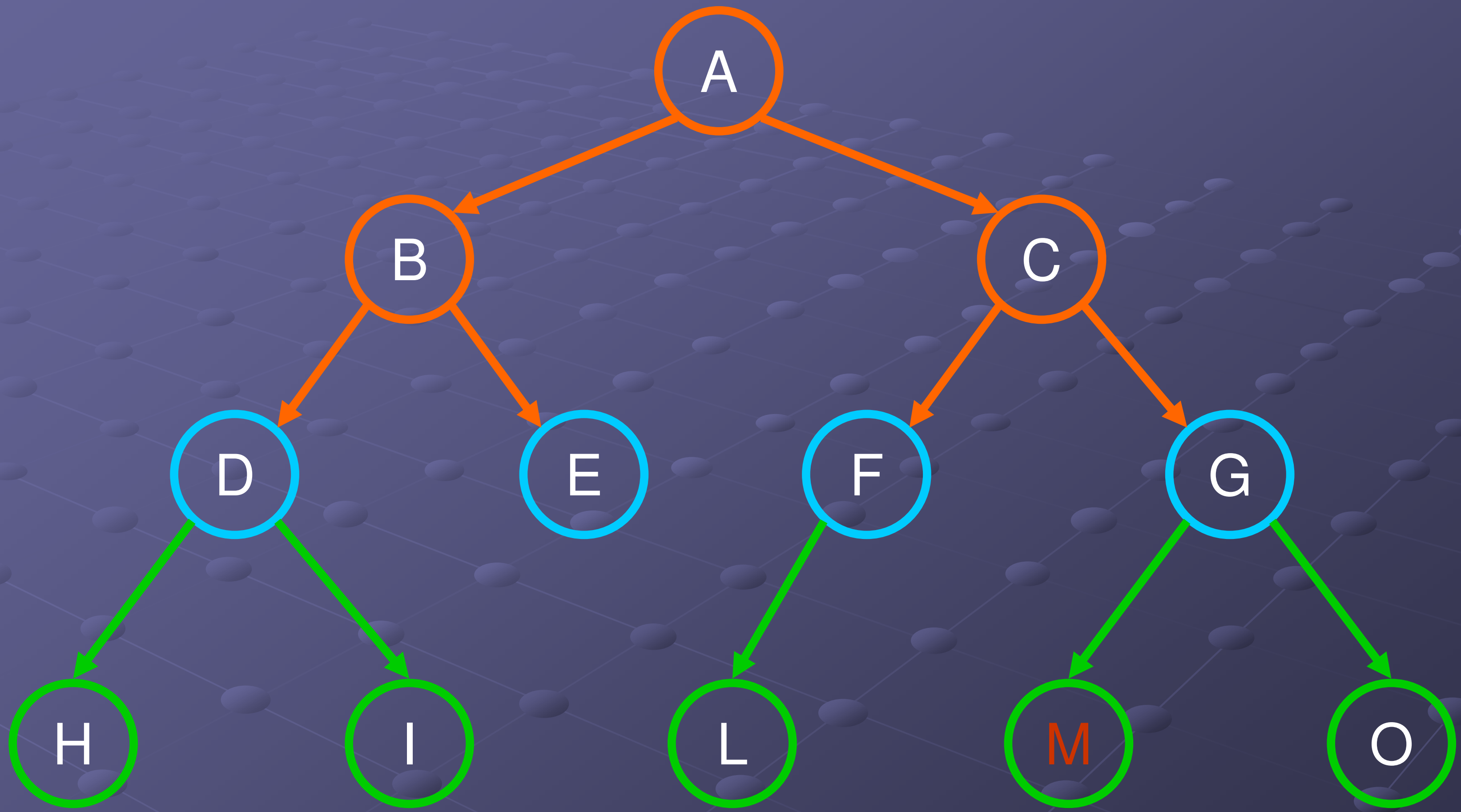
Breadth-first Search



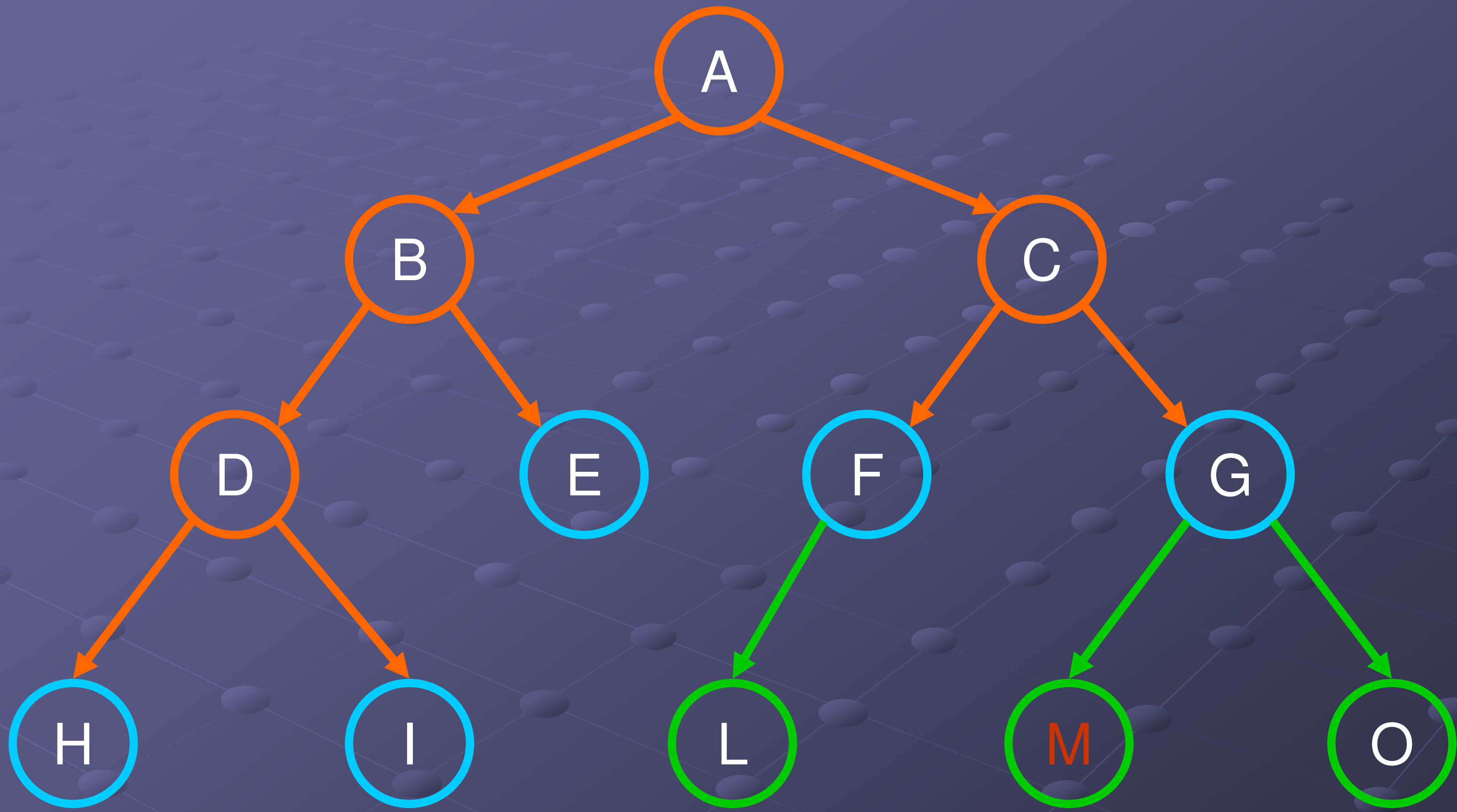
Breadth-first Search



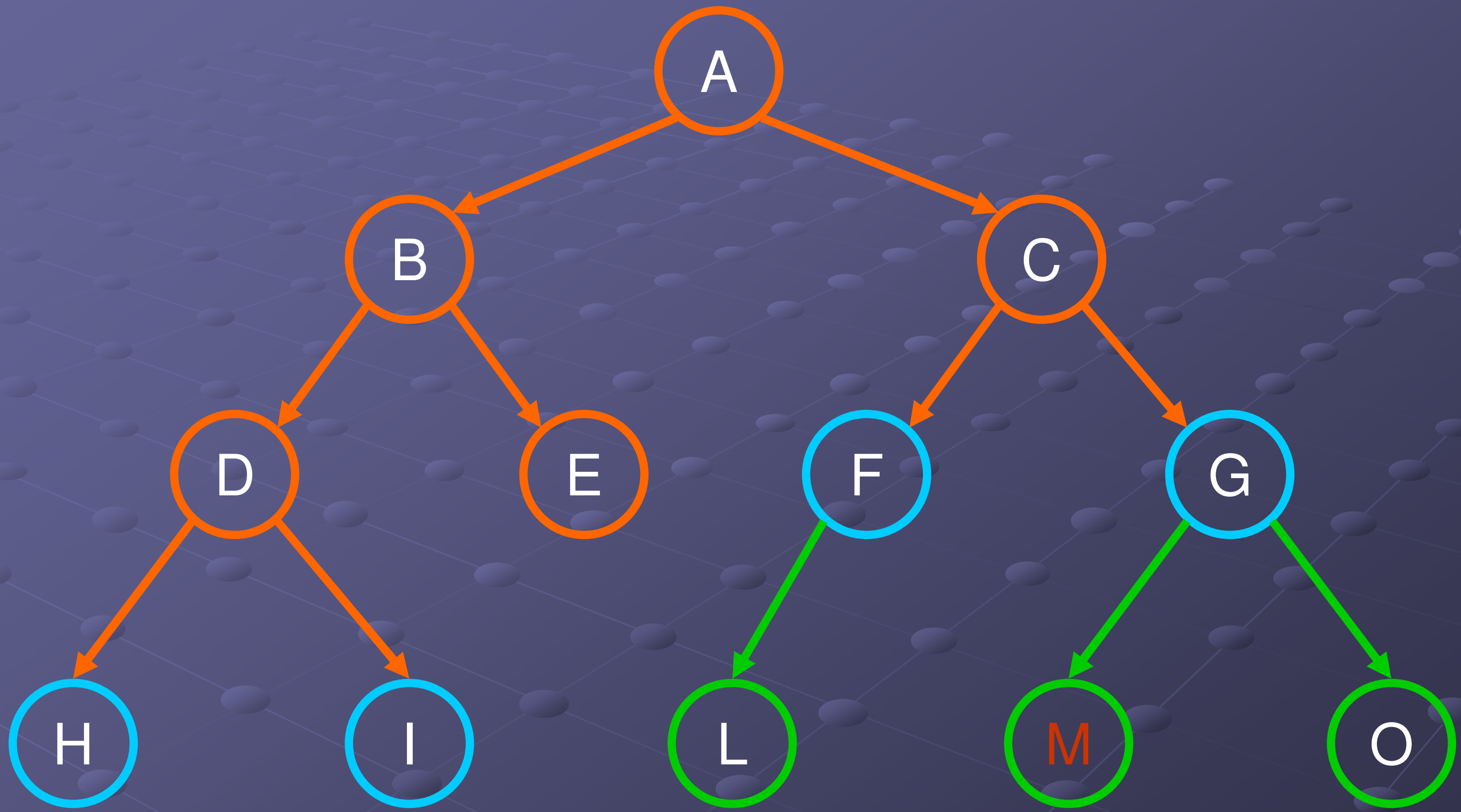
Breadth-first Search



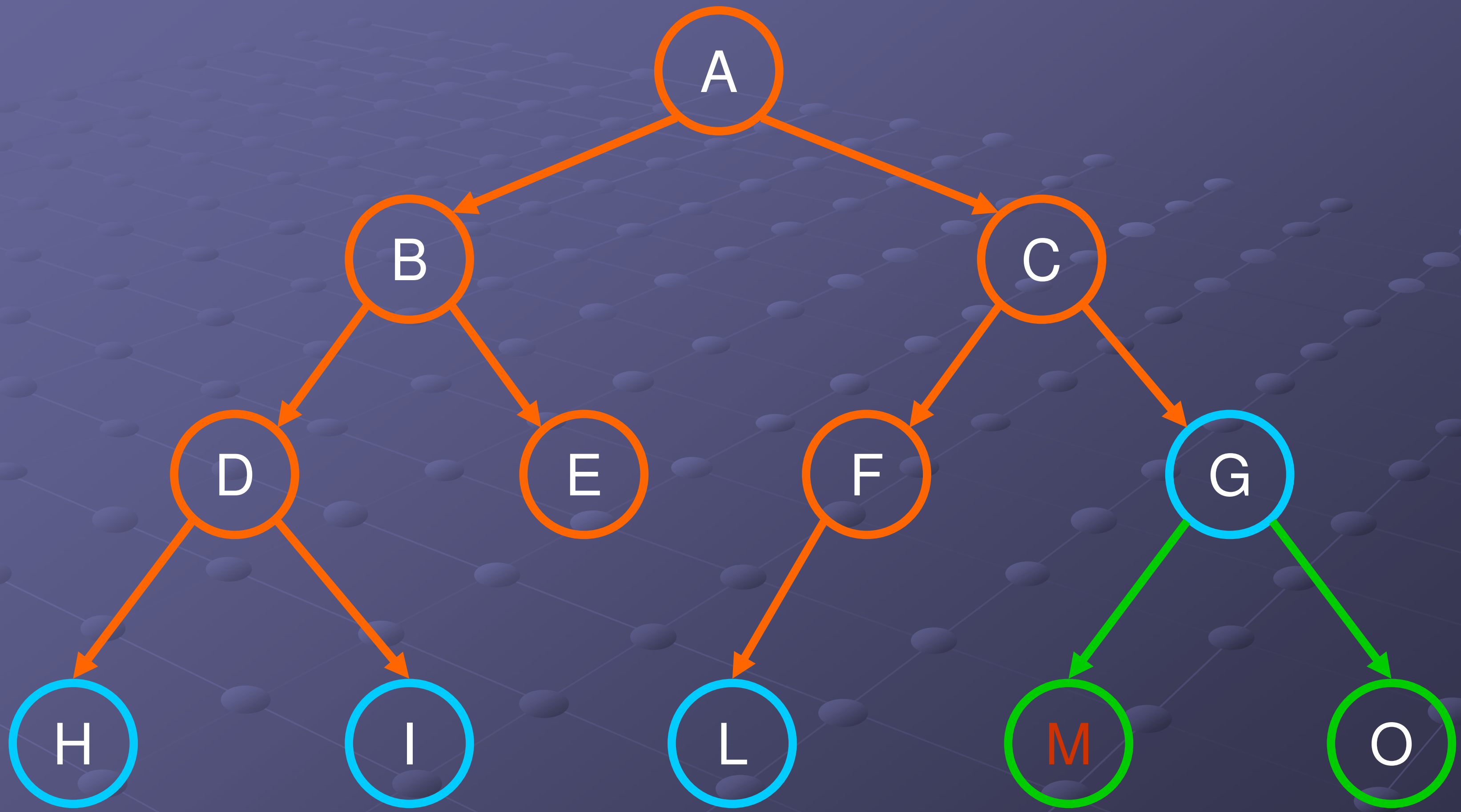
Breadth-first Search



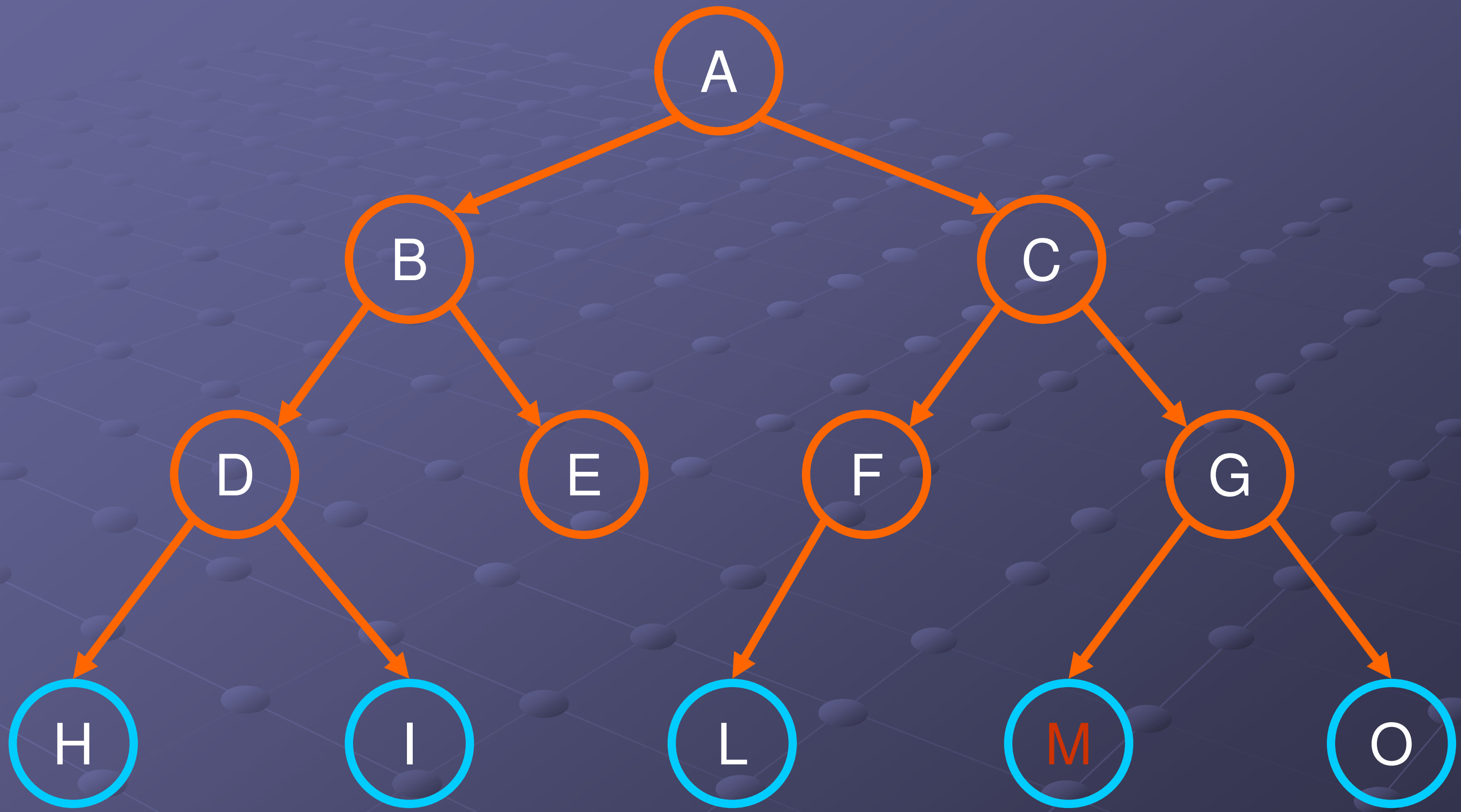
Breadth-first Search



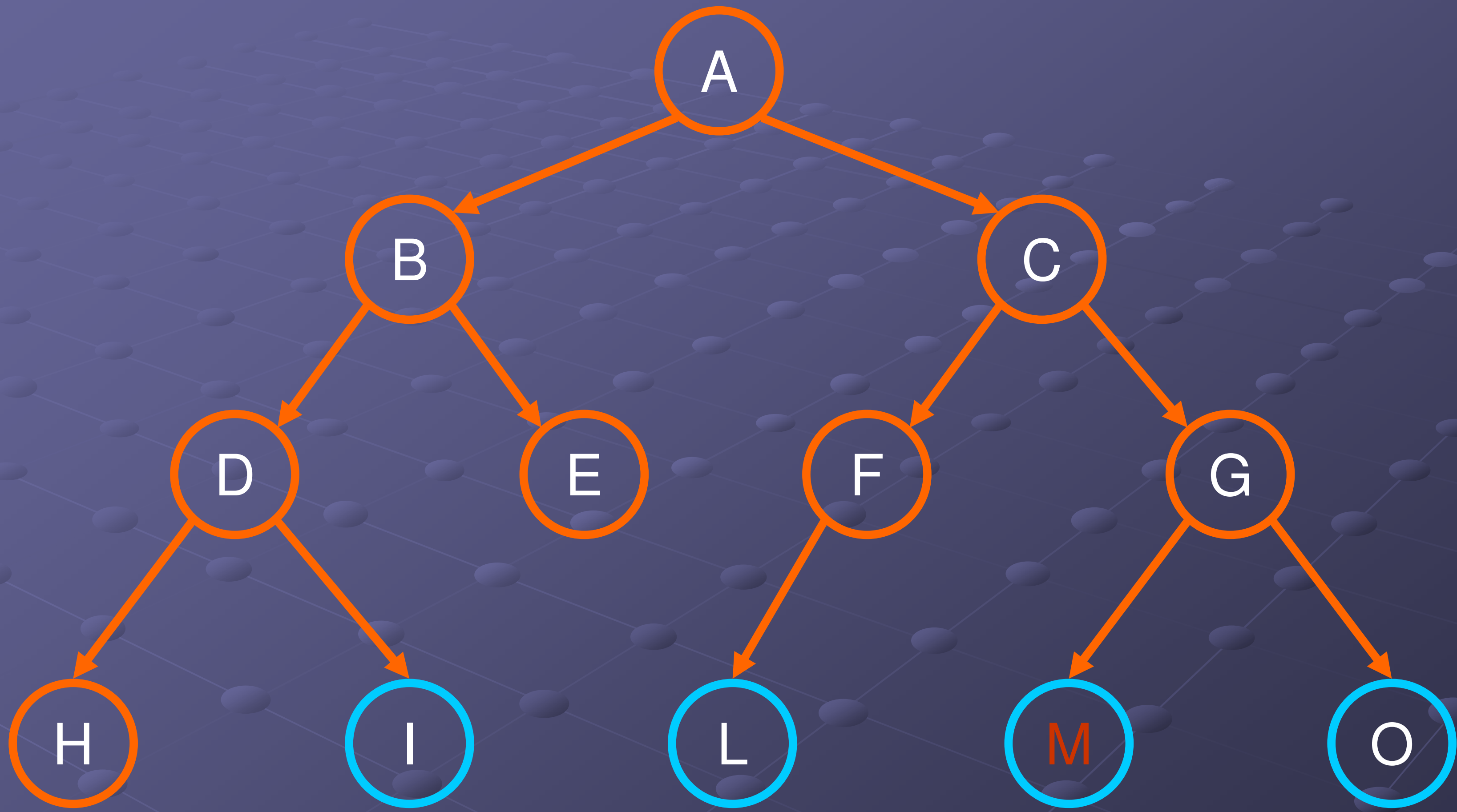
Breadth-first Search



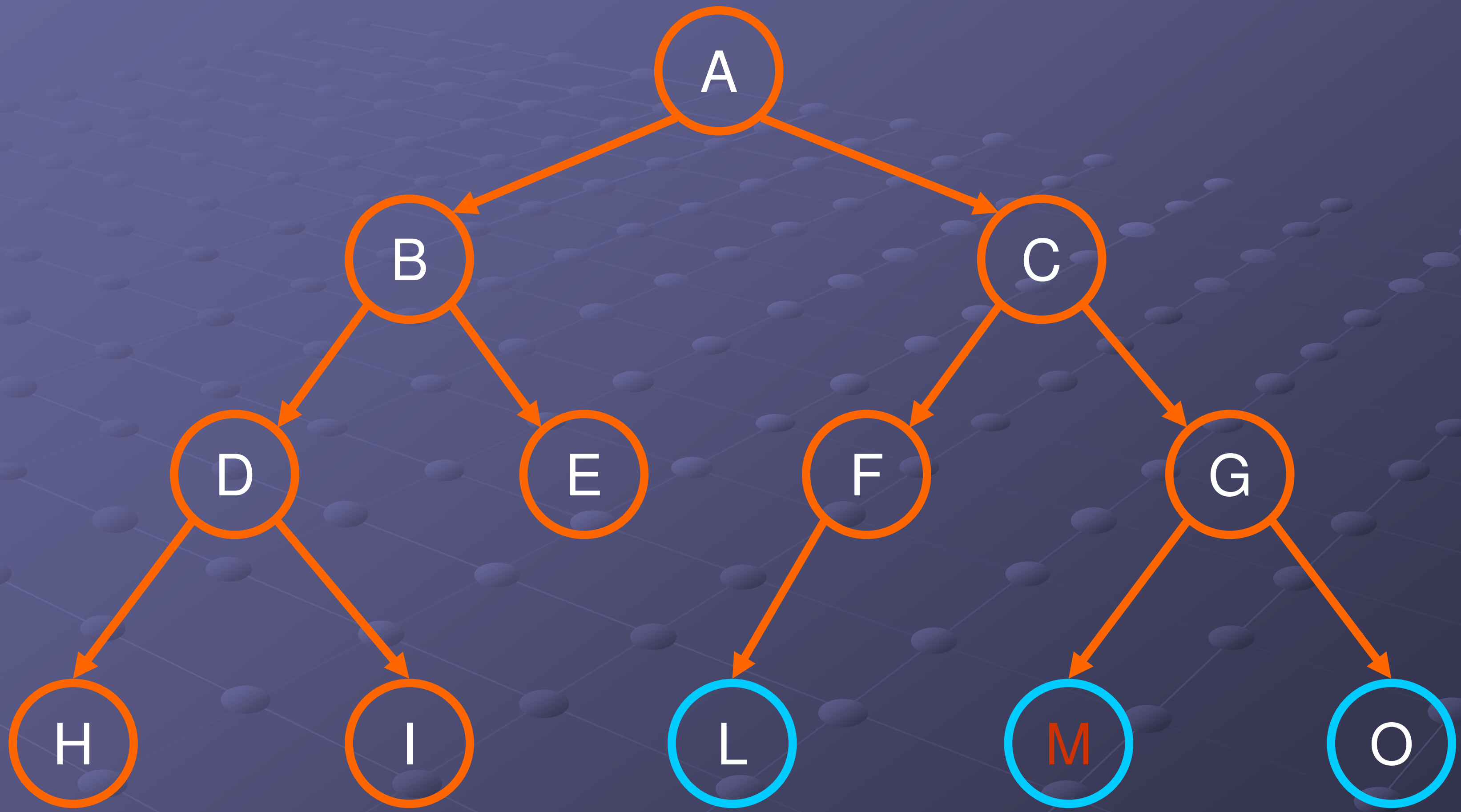
Breadth-first Search



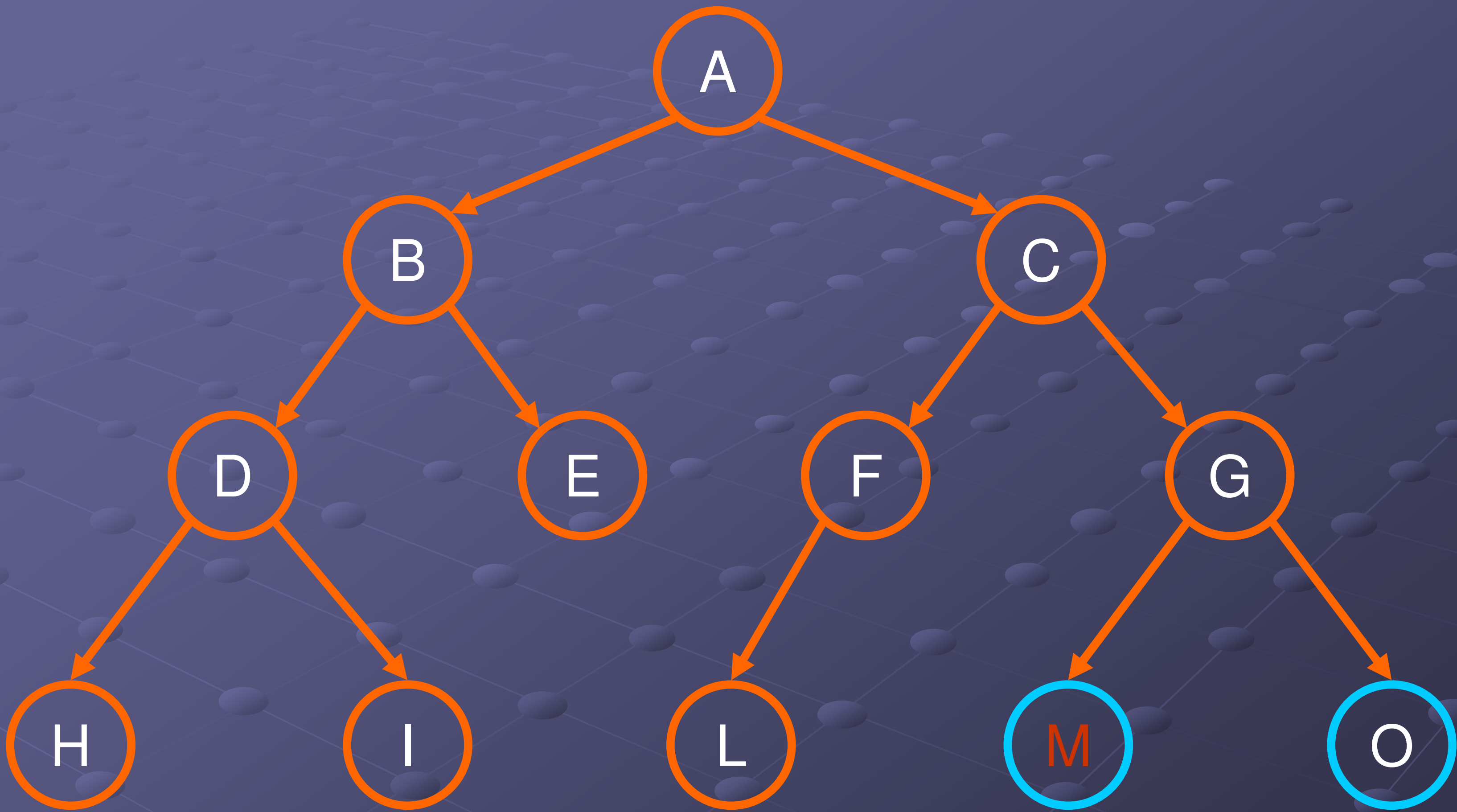
Breadth-first Search



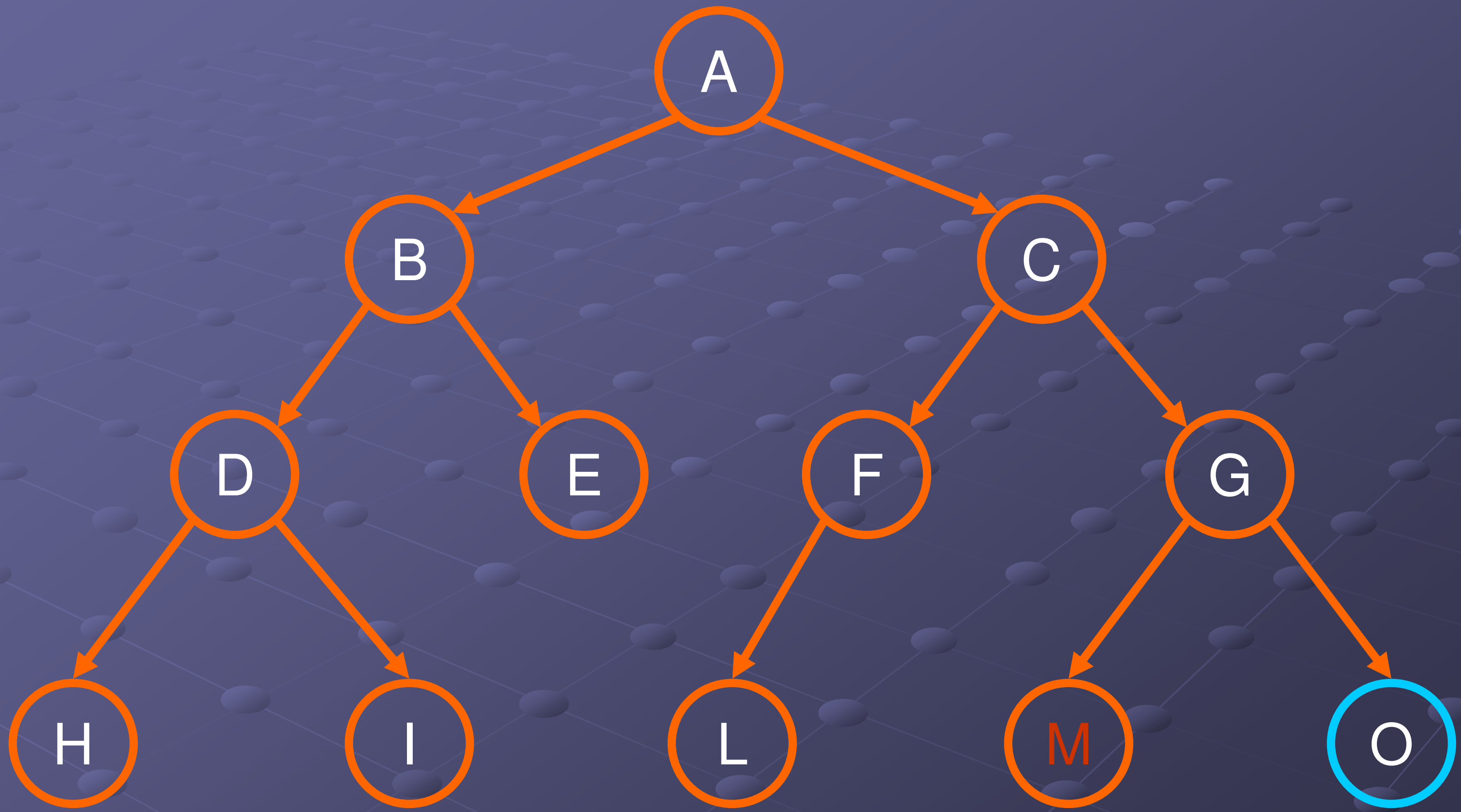
Breadth-first Search



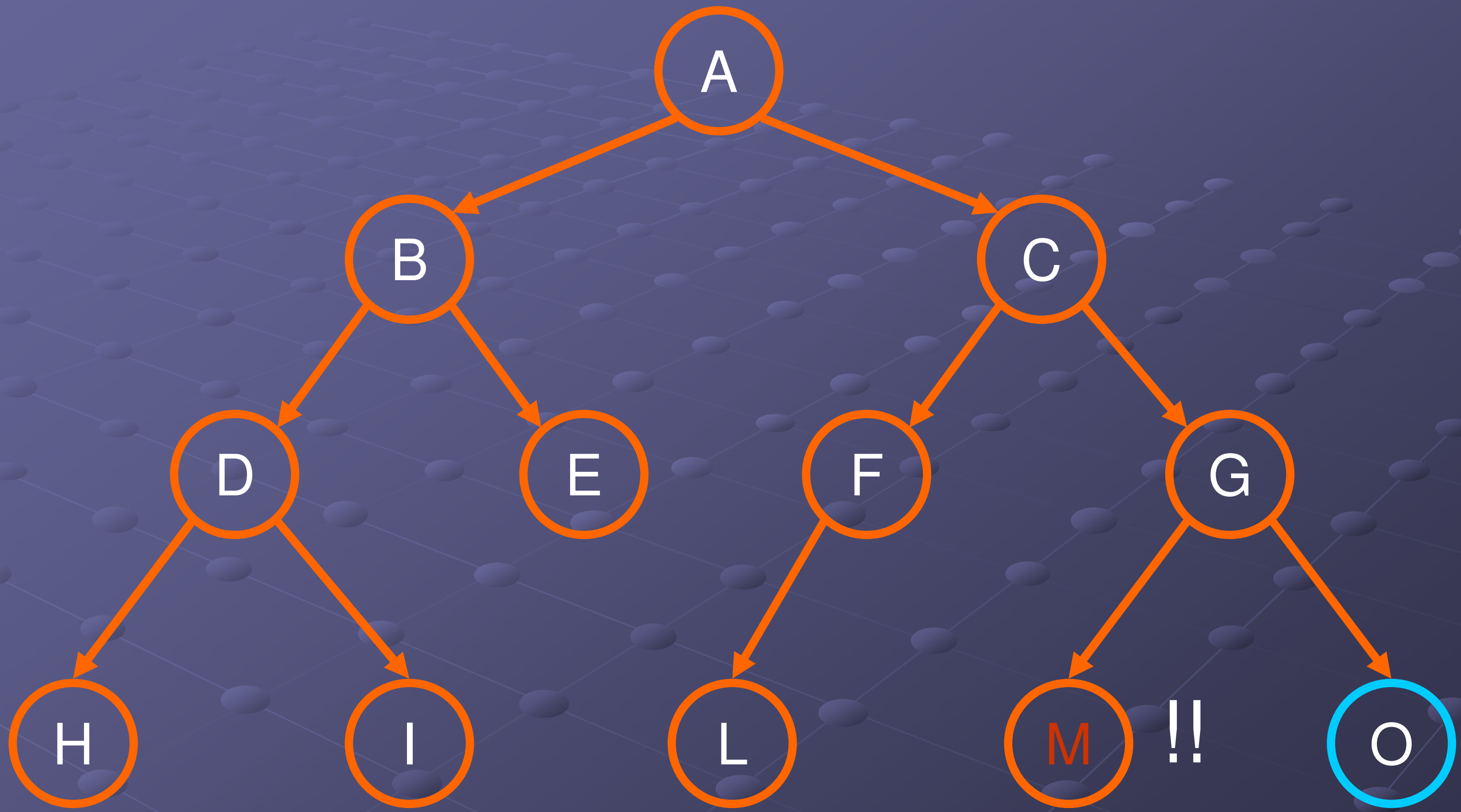
Breadth-first Search



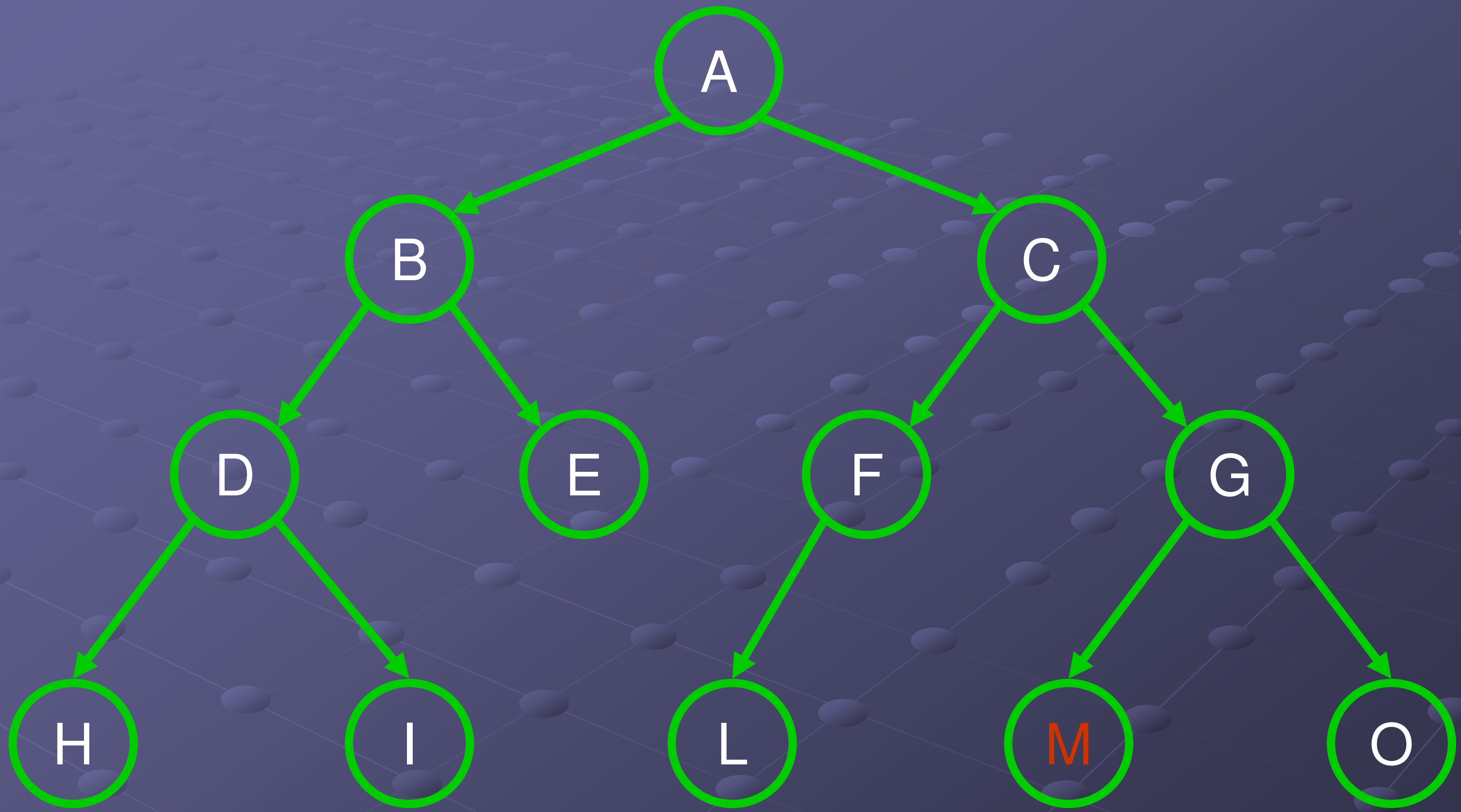
Breadth-first Search



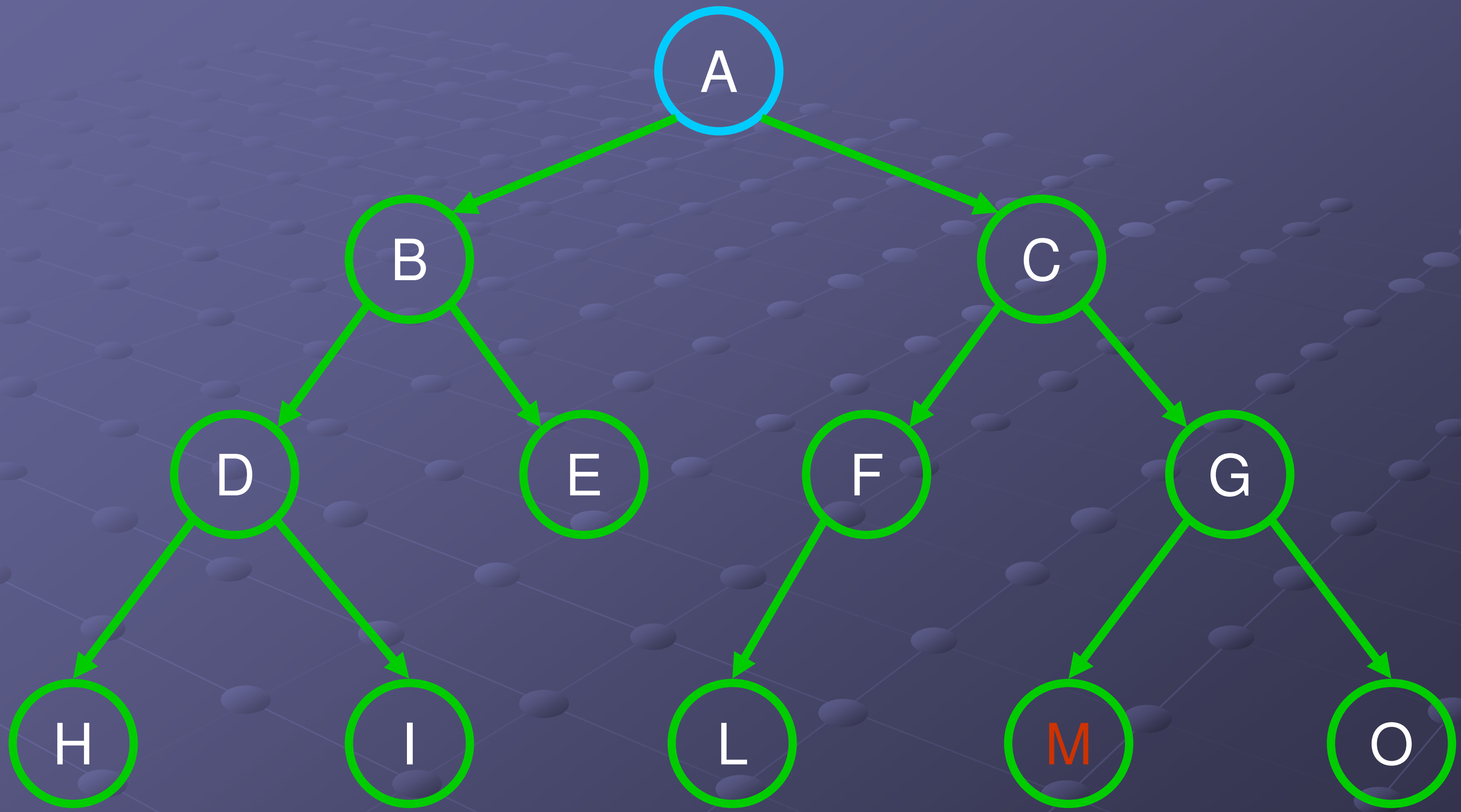
Breadth-first Search



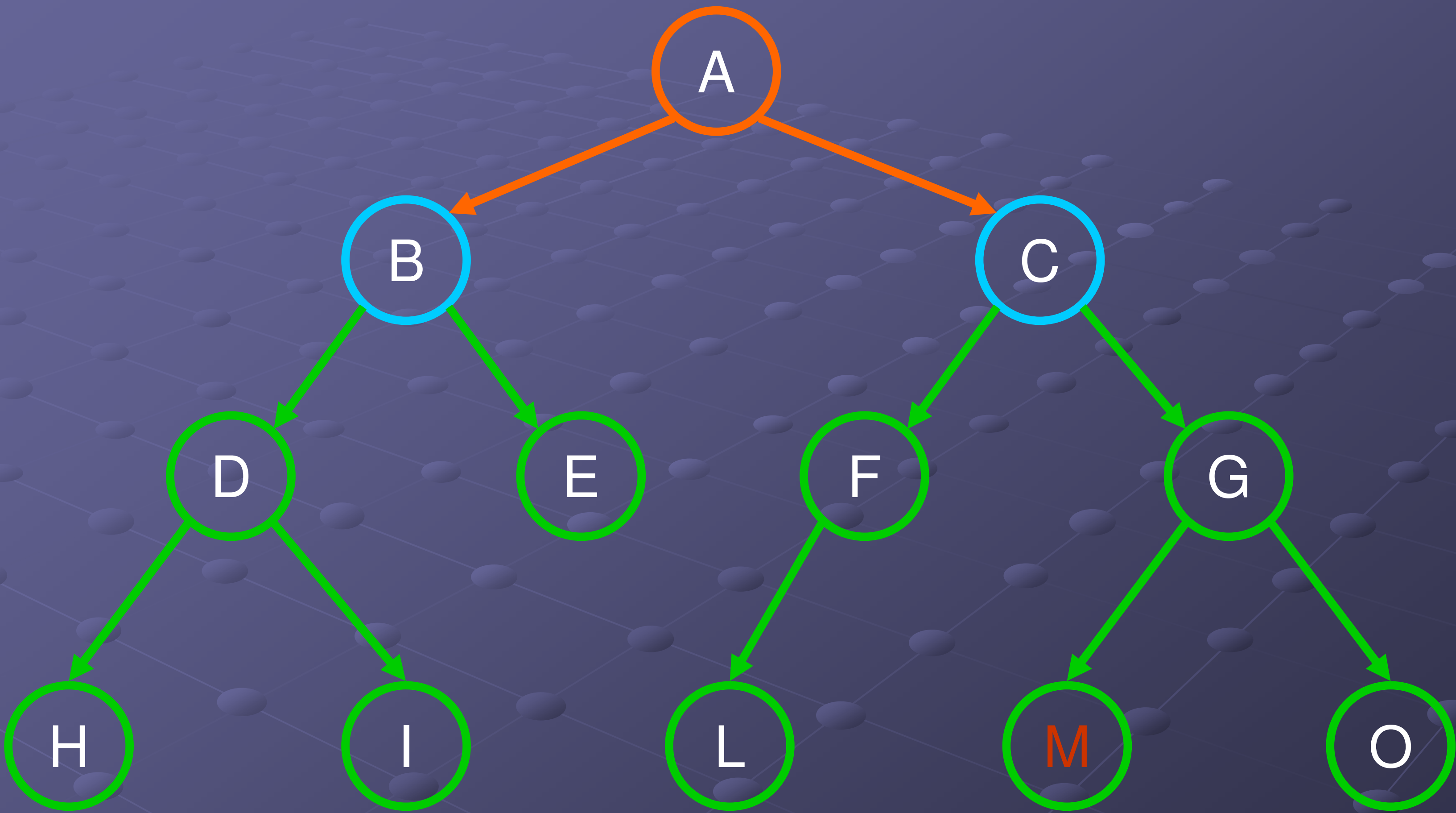
Depth-first Search



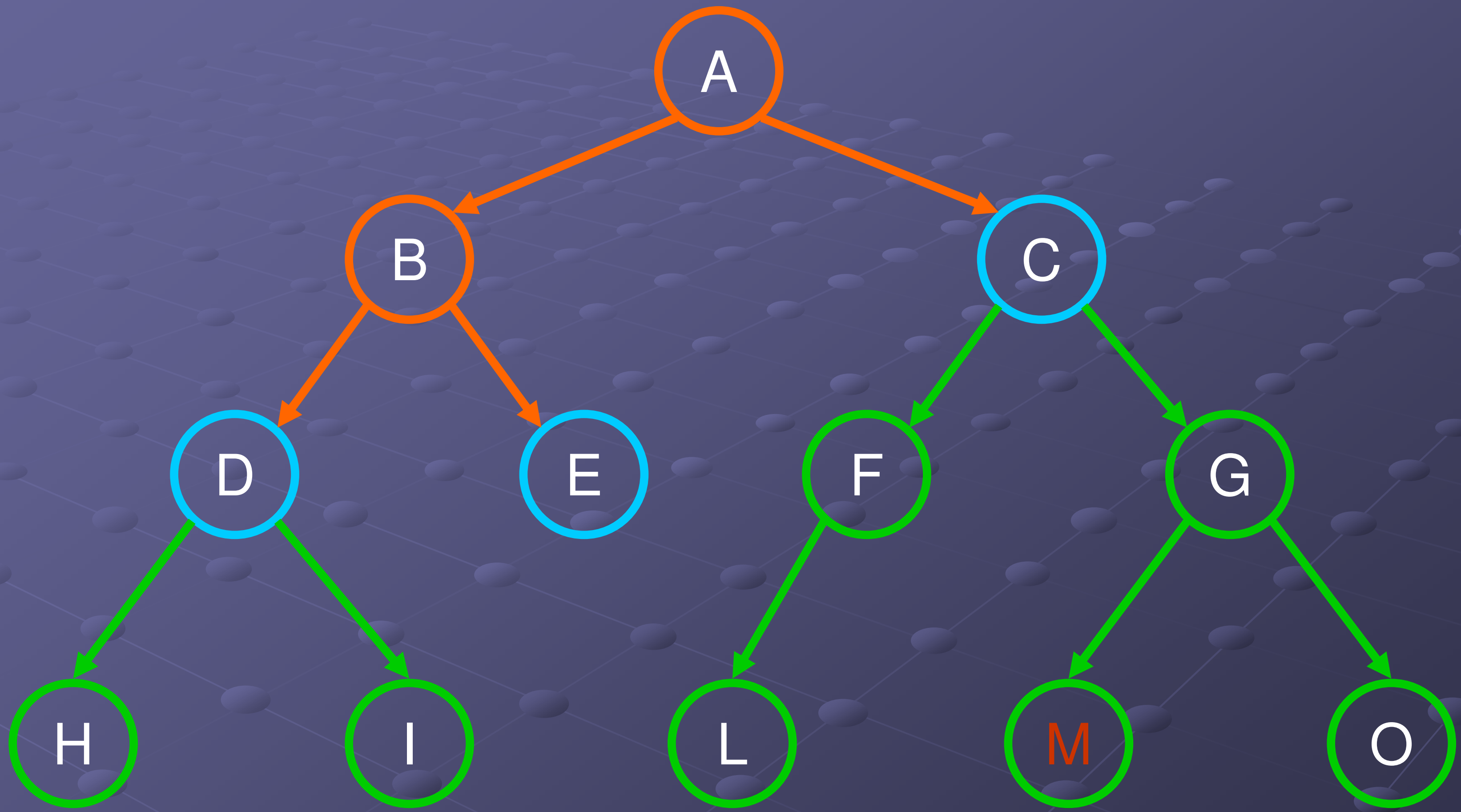
Depth-first Search



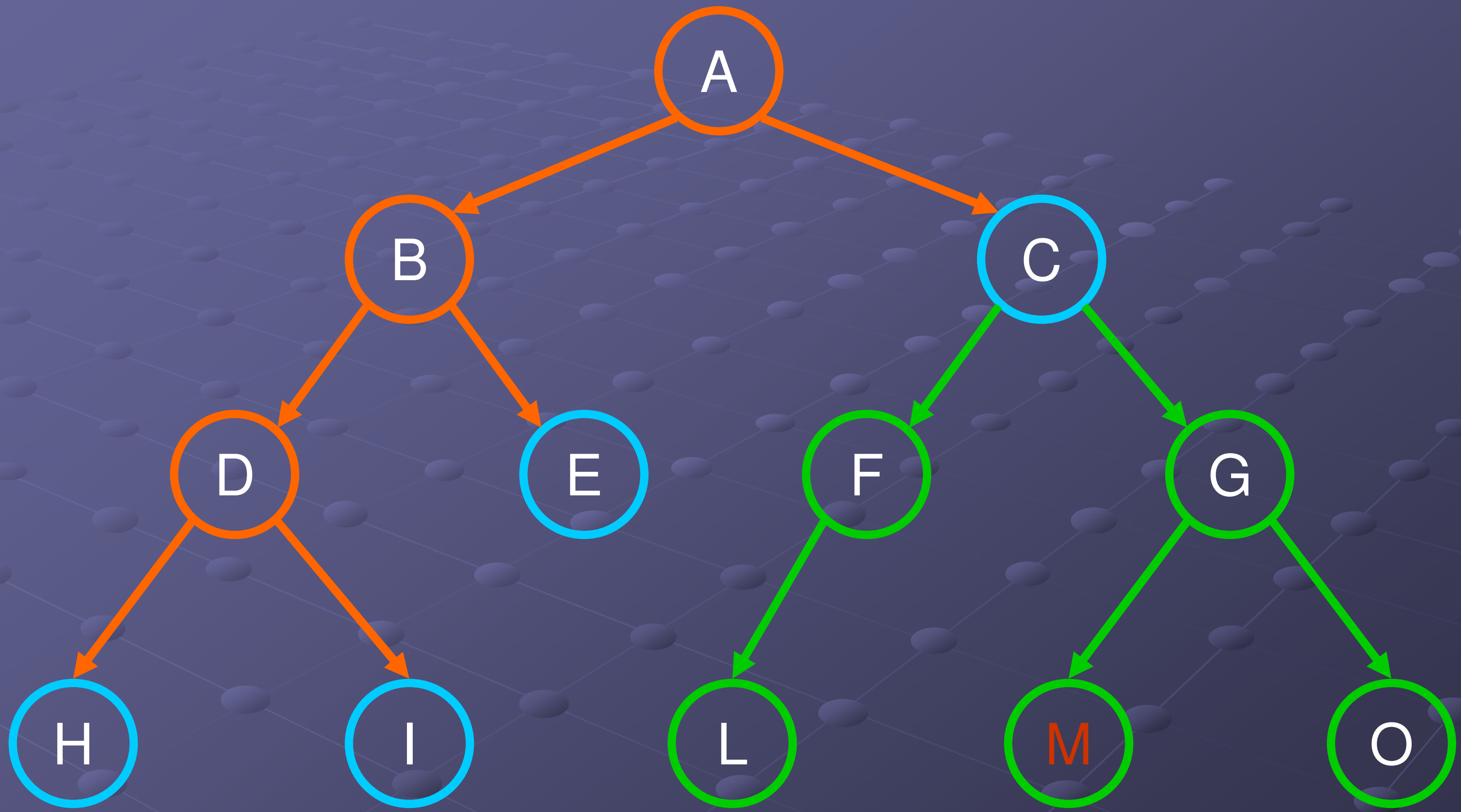
Depth-first Search



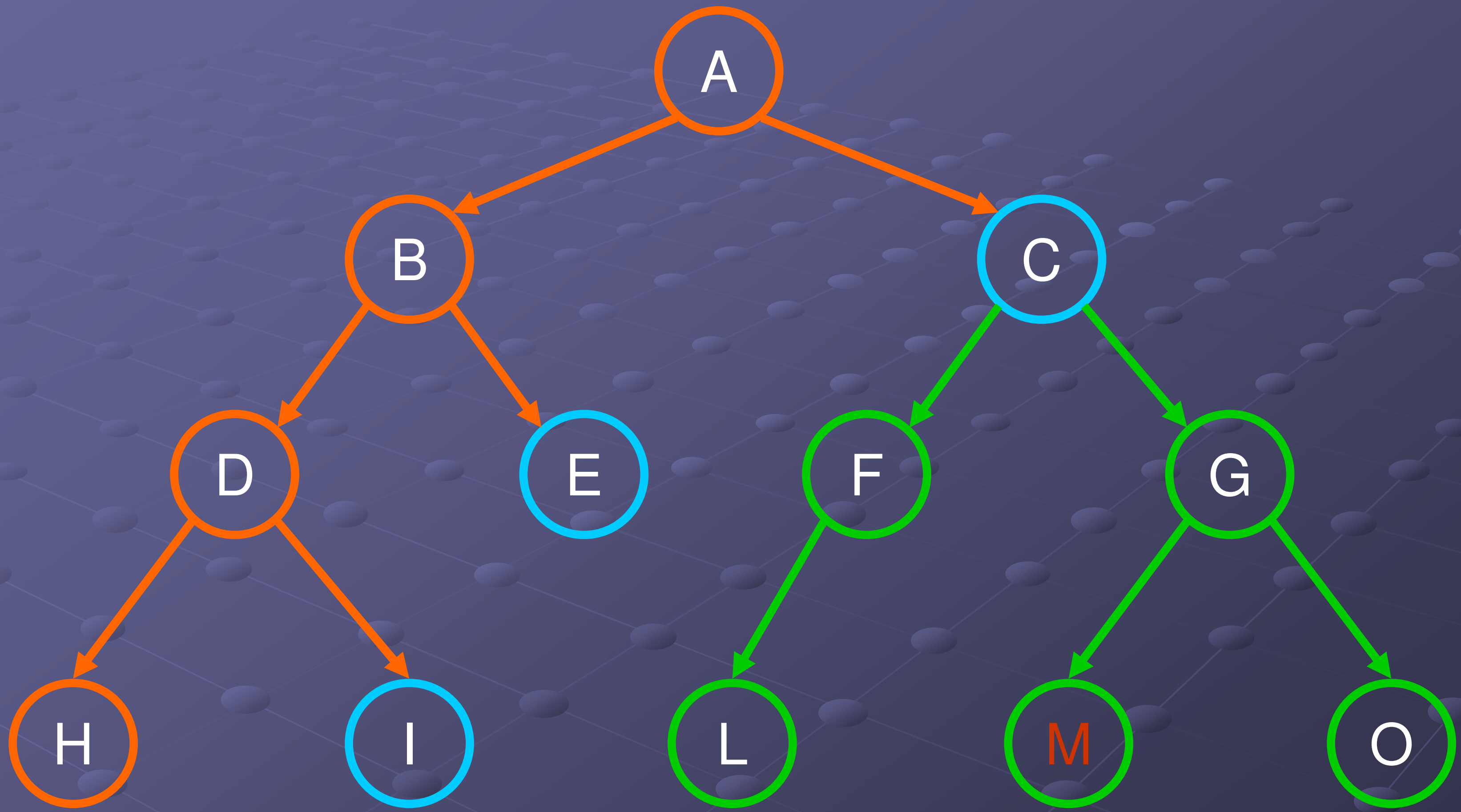
Depth-first Search



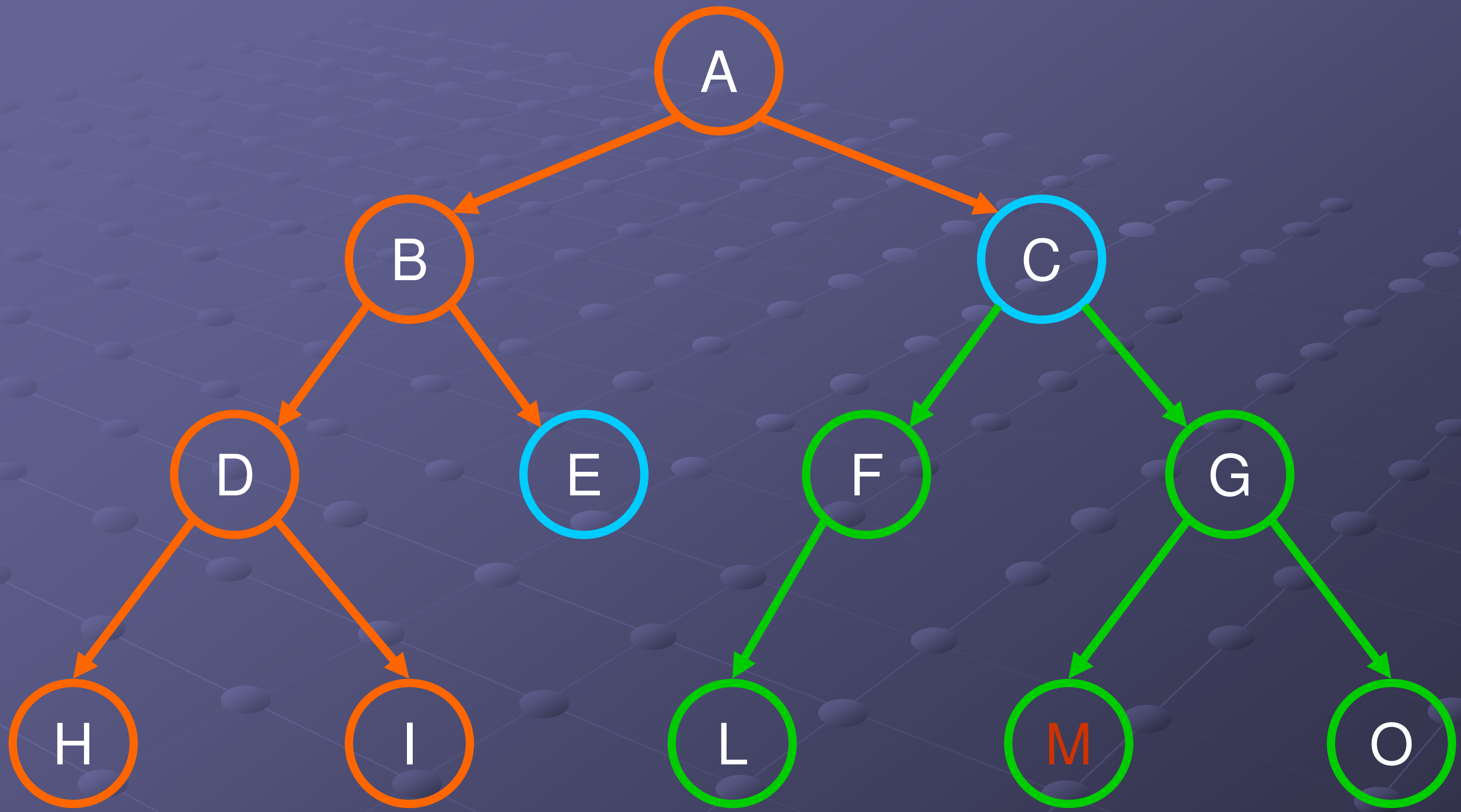
Depth-first Search



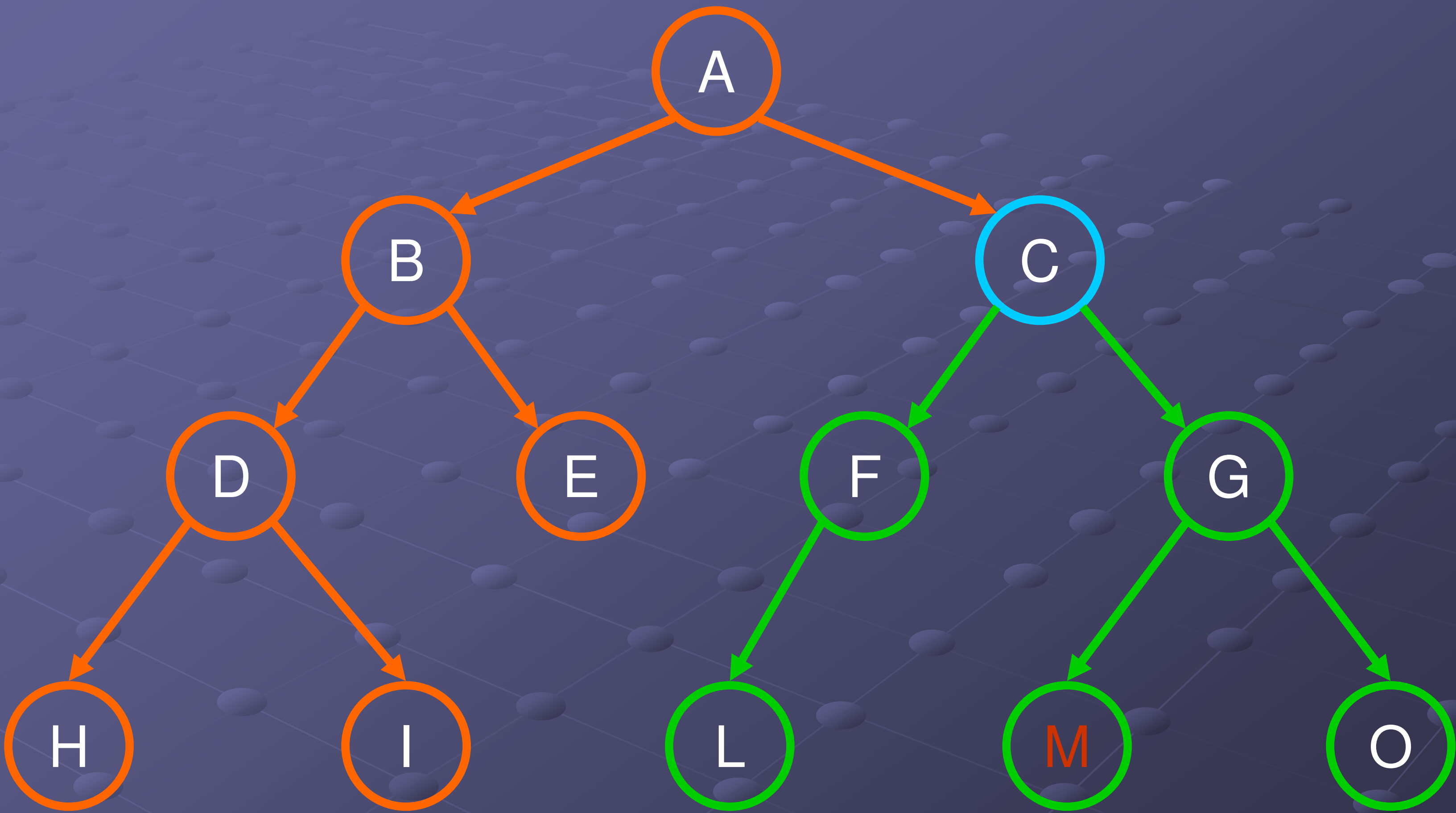
Depth-first Search



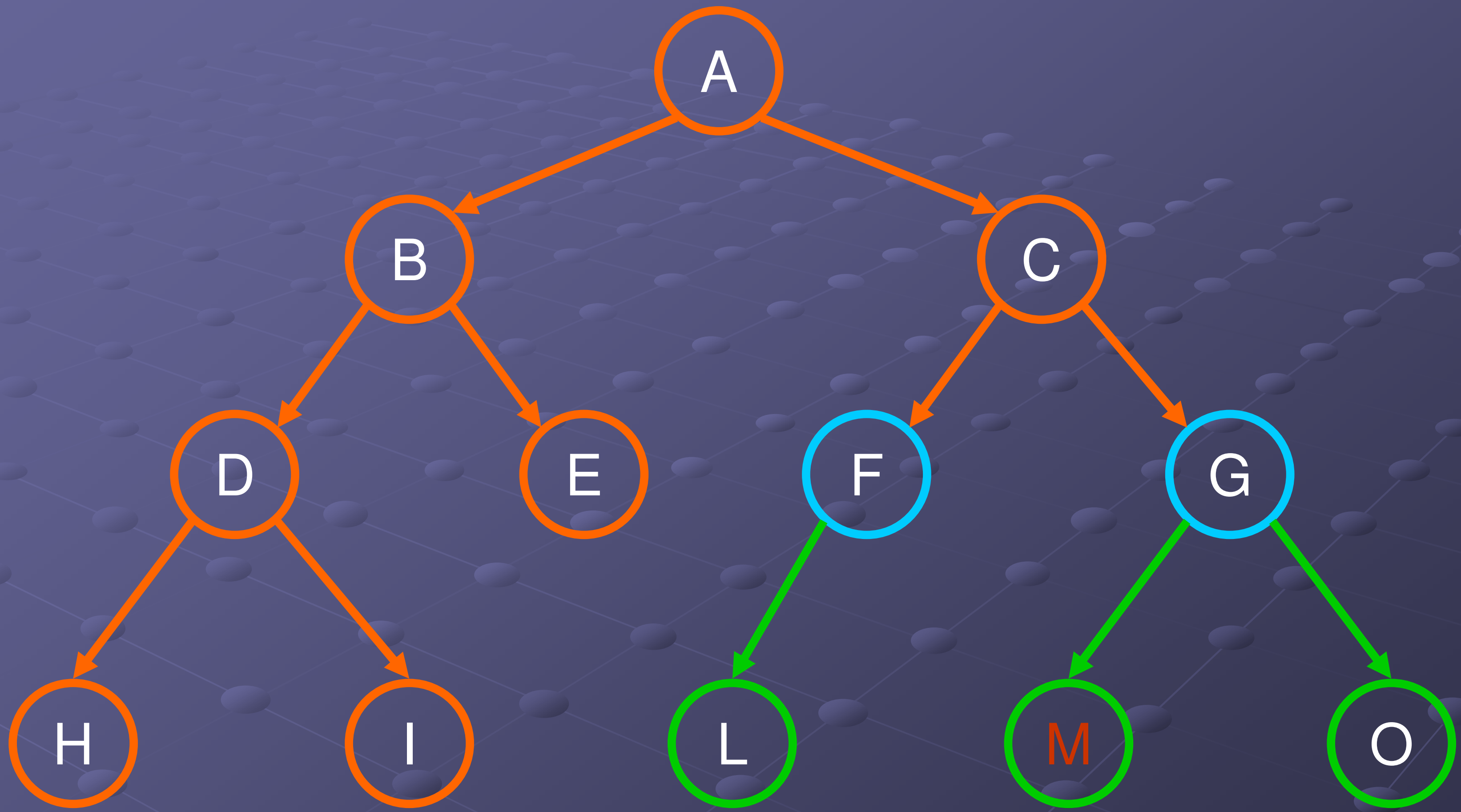
Depth-first Search



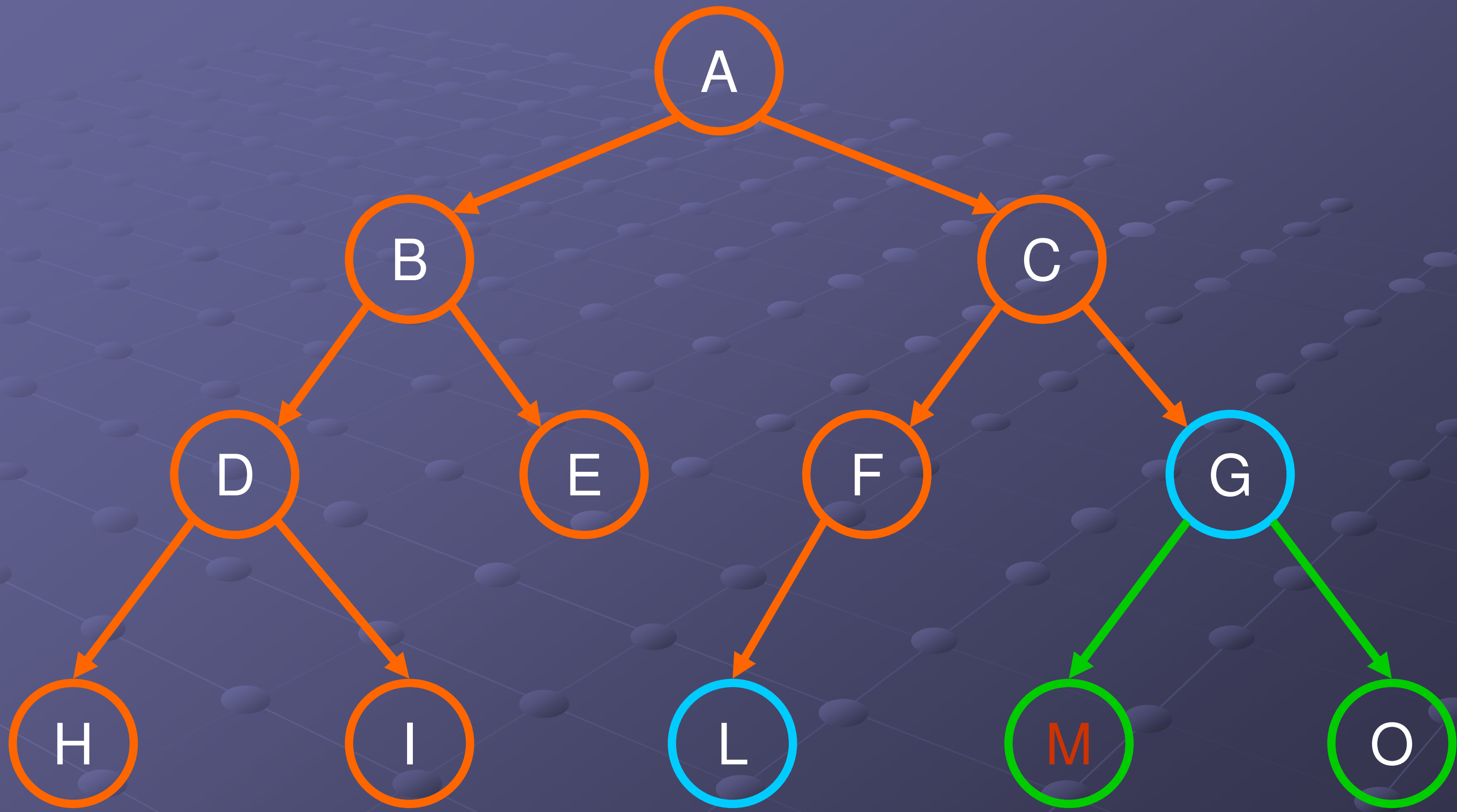
Depth-first Search



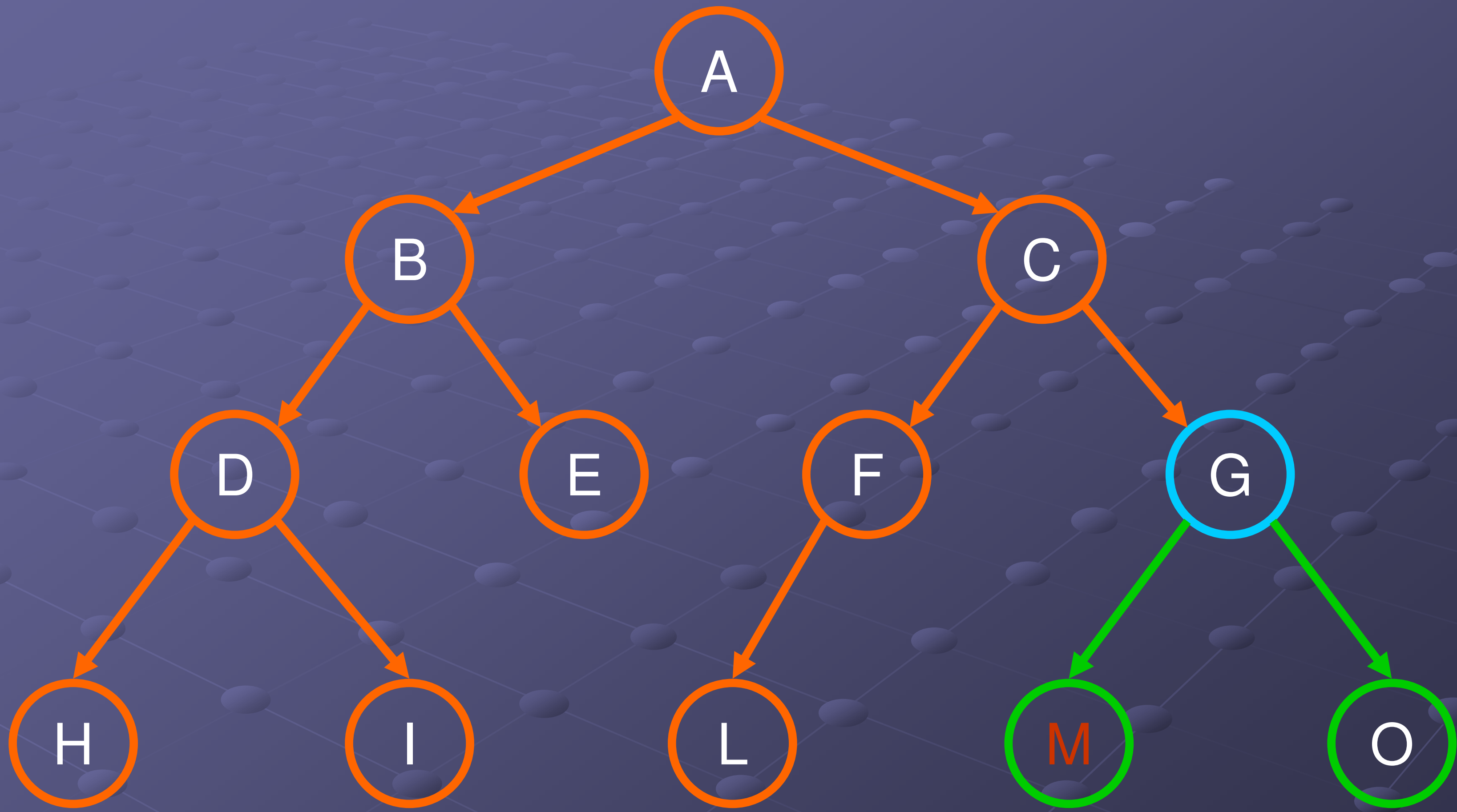
Depth-first Search



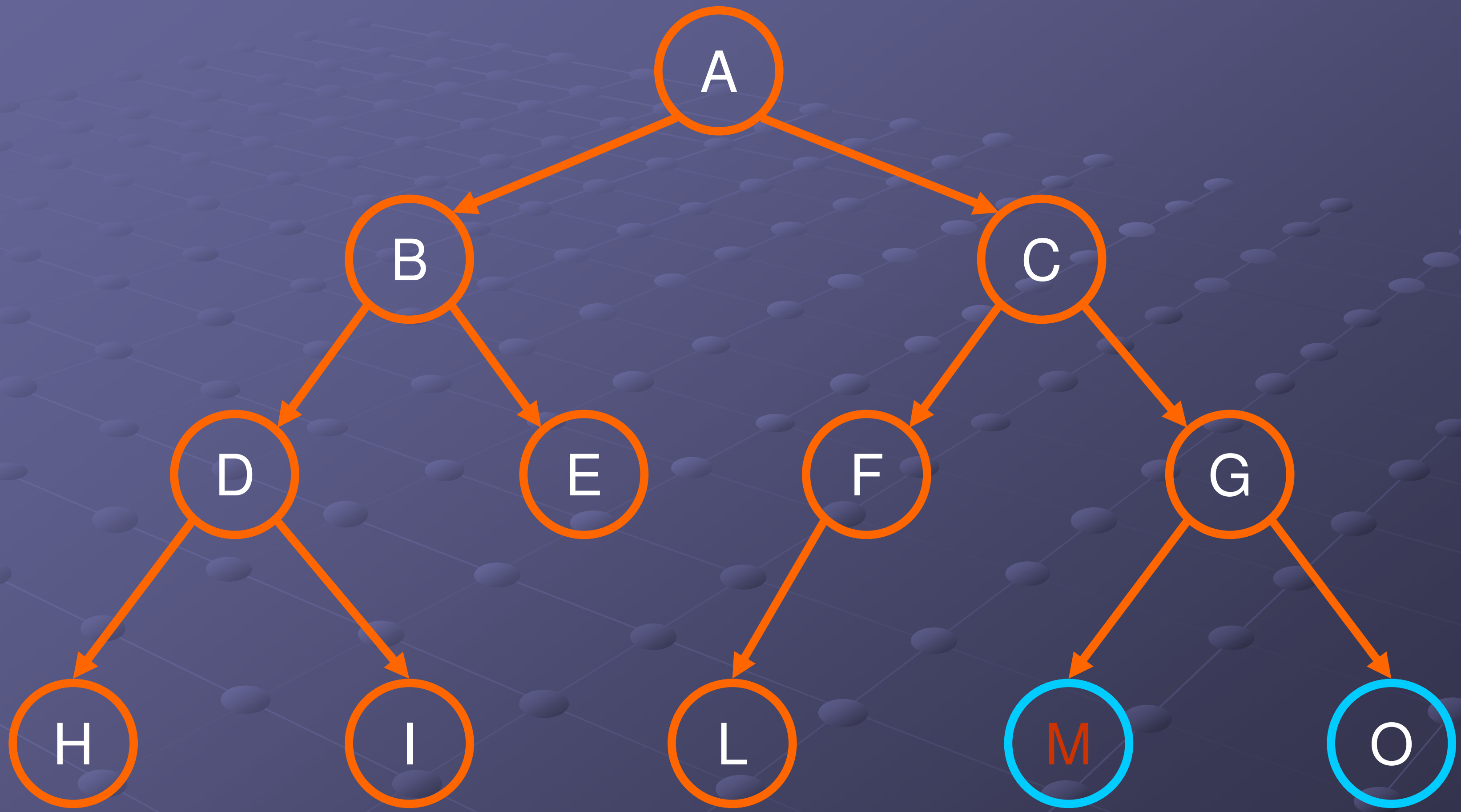
Depth-first Search



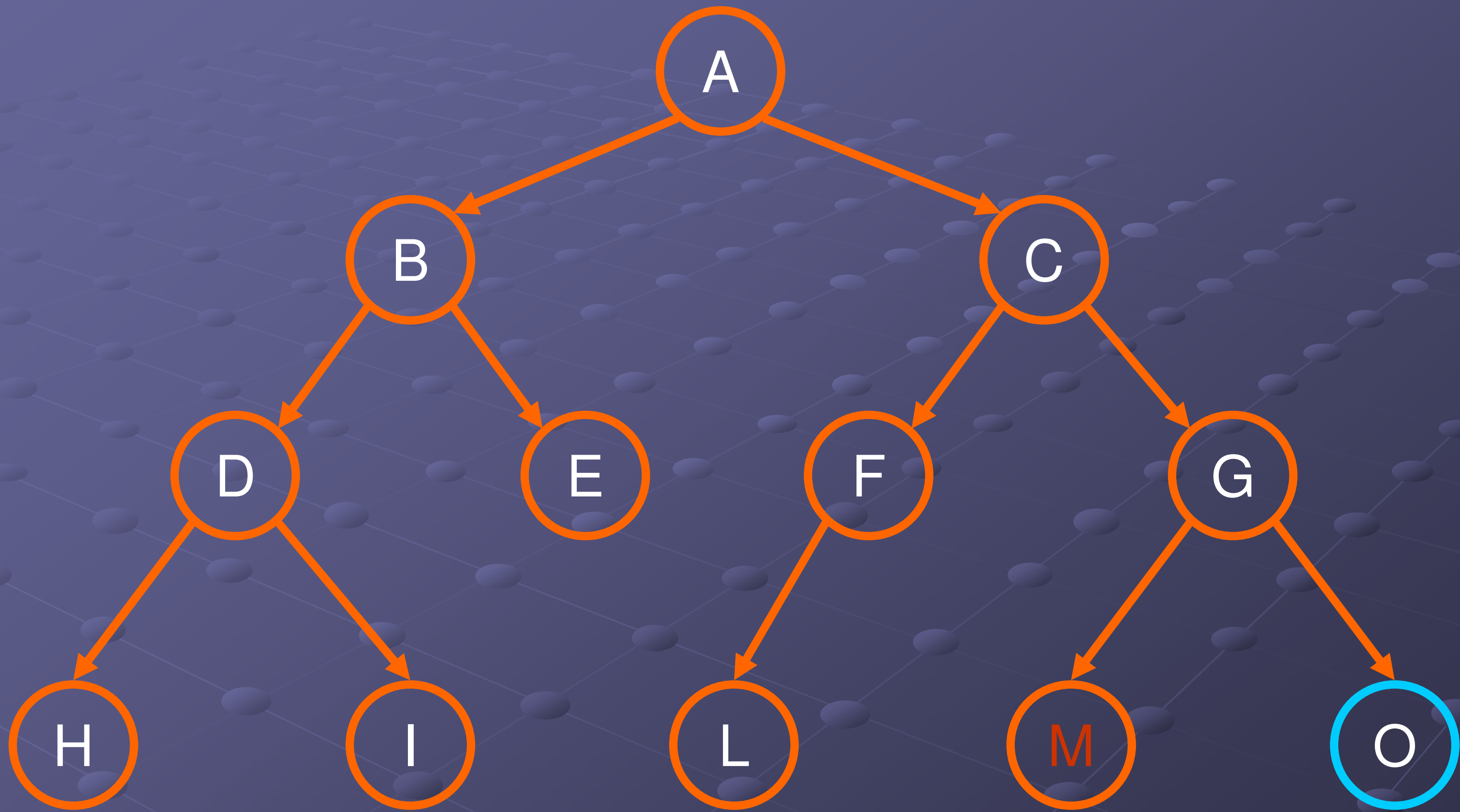
Depth-first Search



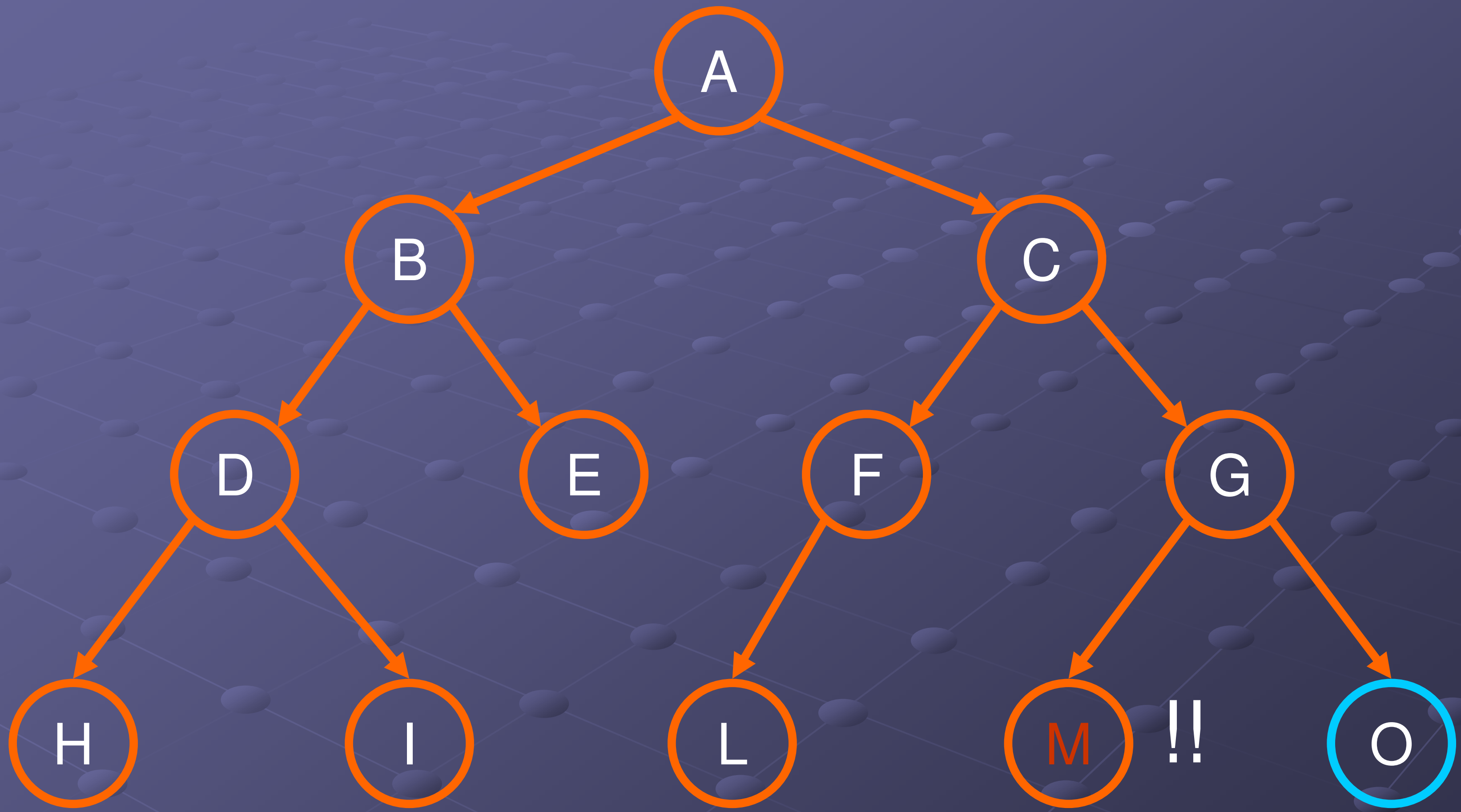
Depth-first Search



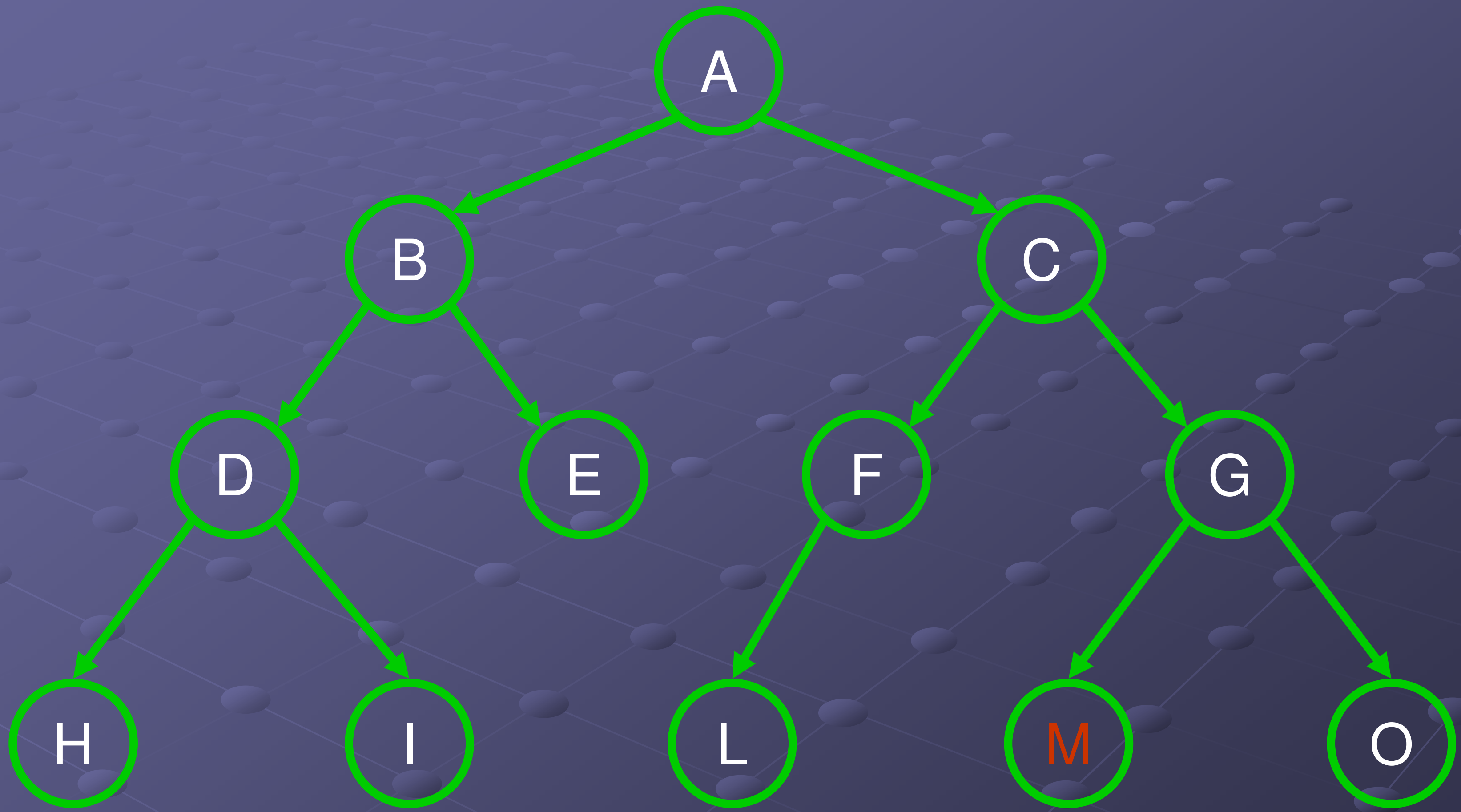
Depth-first Search



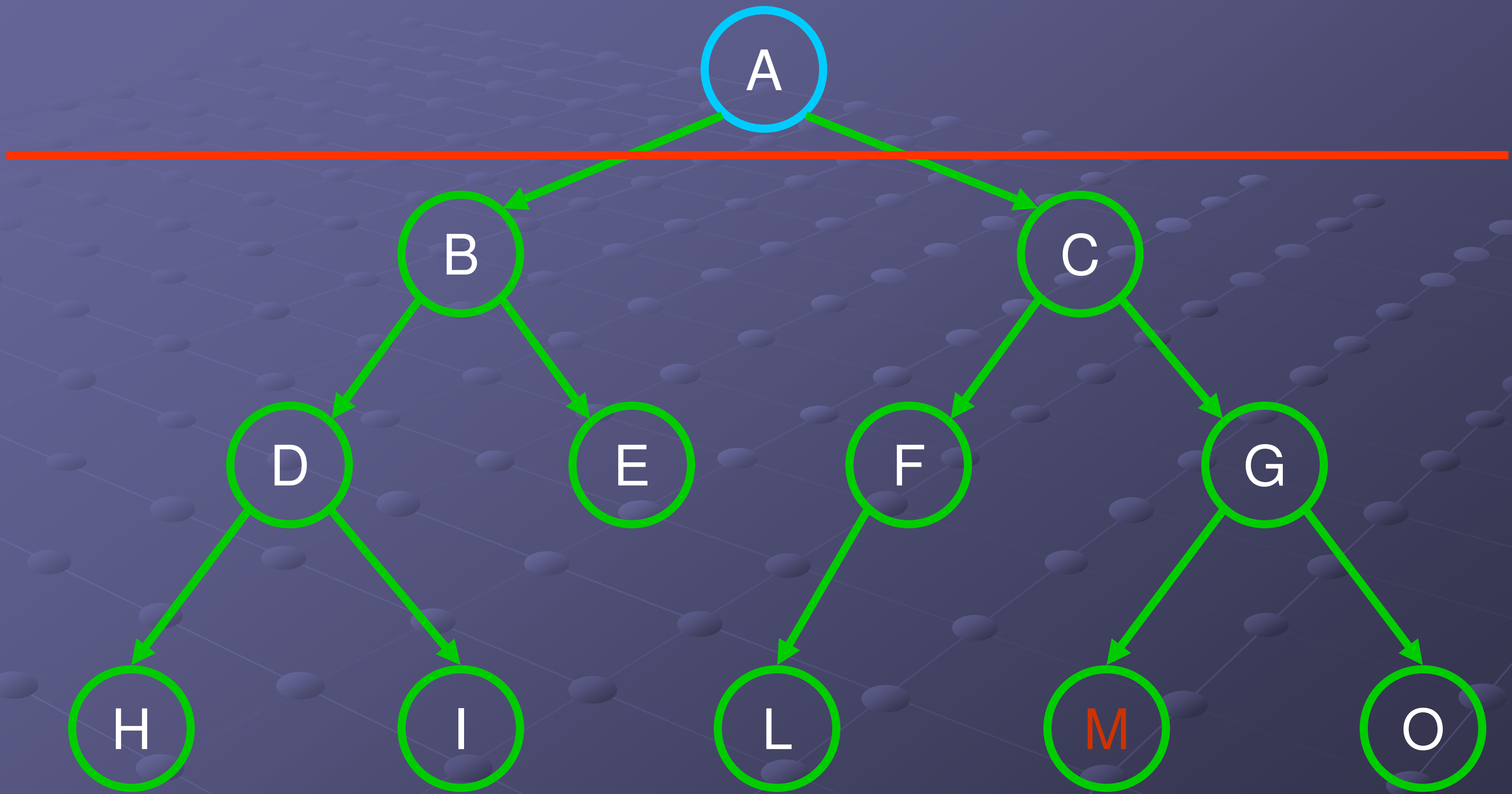
Breadth-first Search



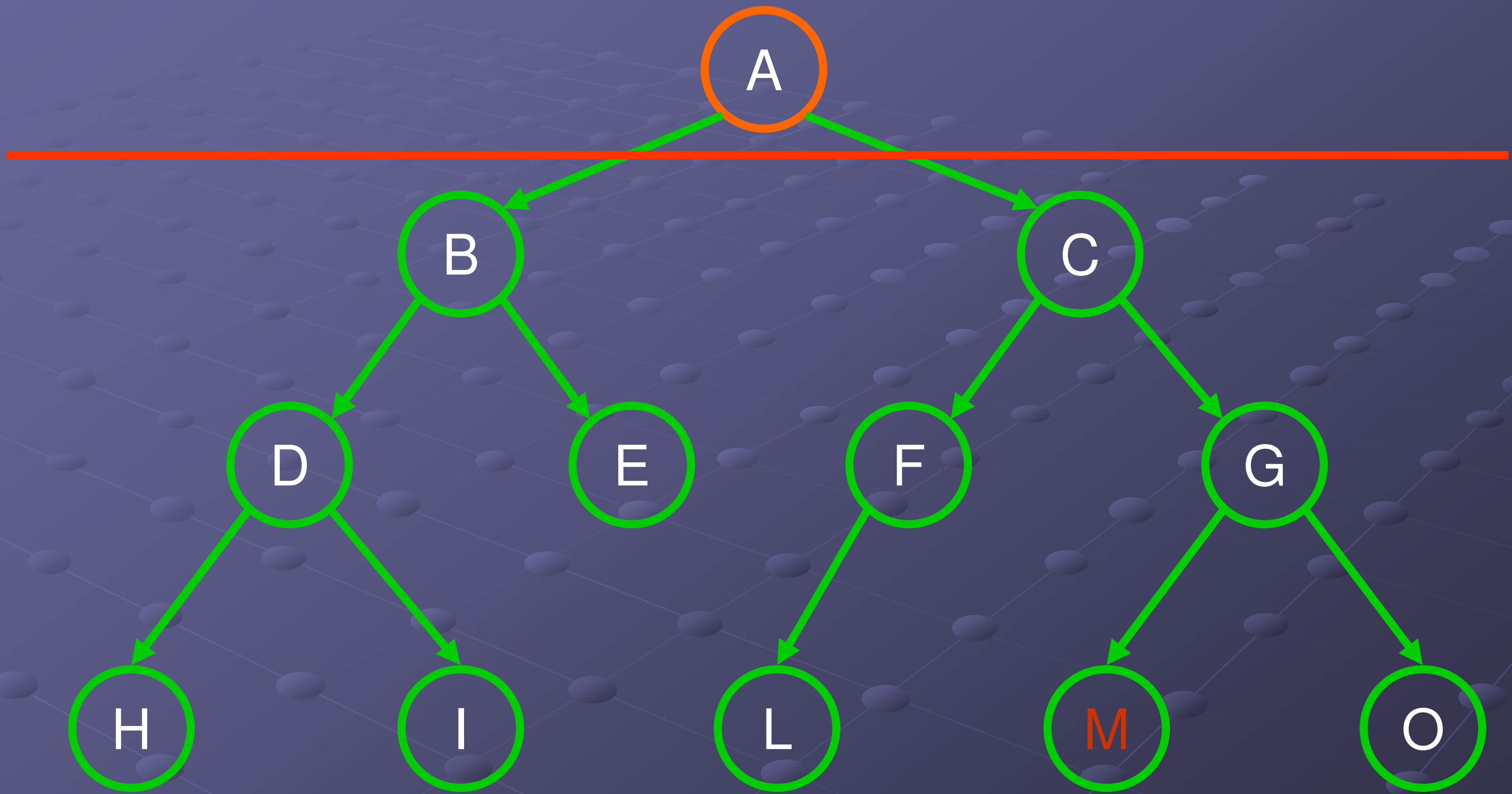
Iterative Deepening DFS Search



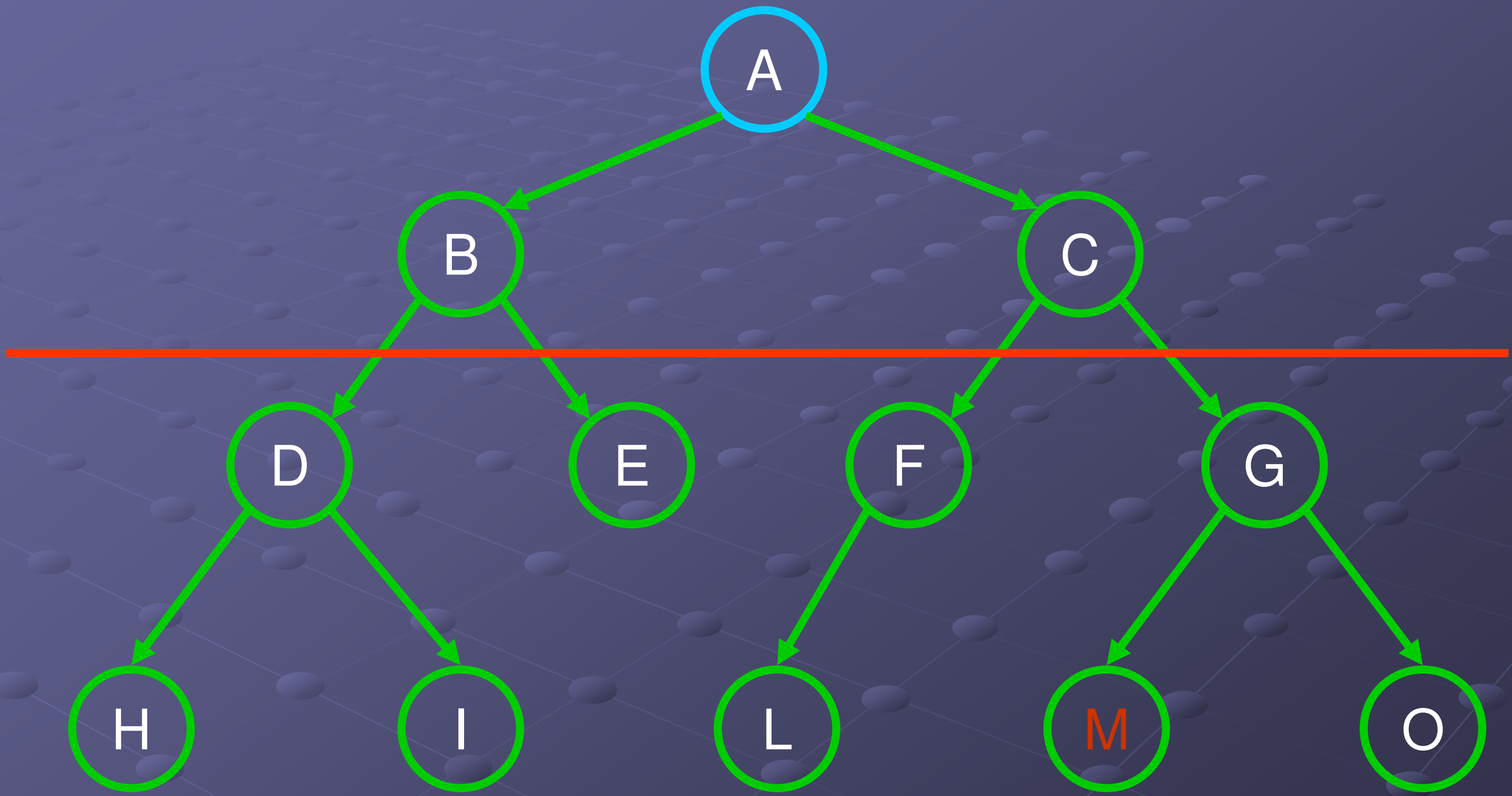
Iterative Deepening DFS Search



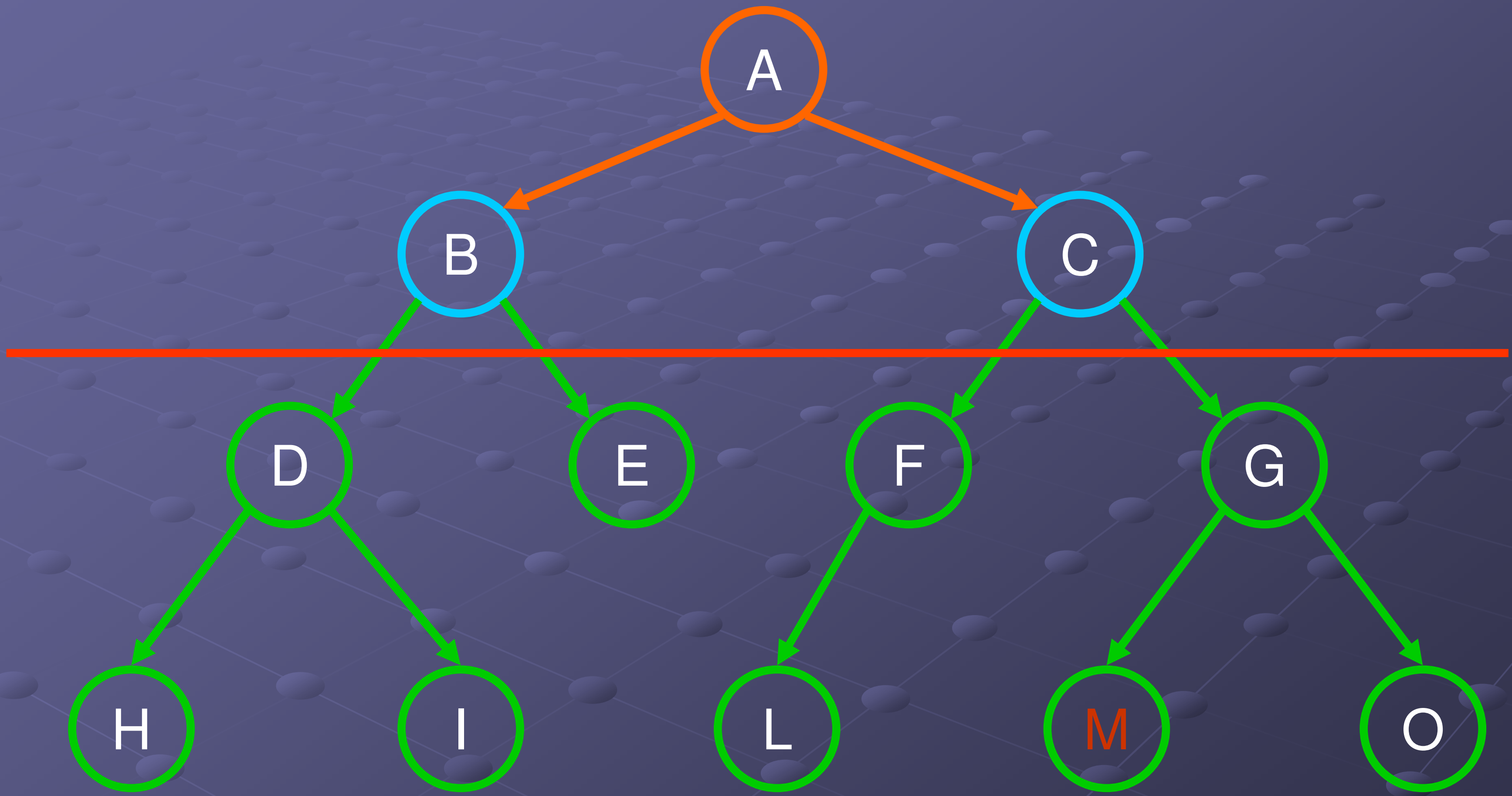
Iterative Deepening DFS Search



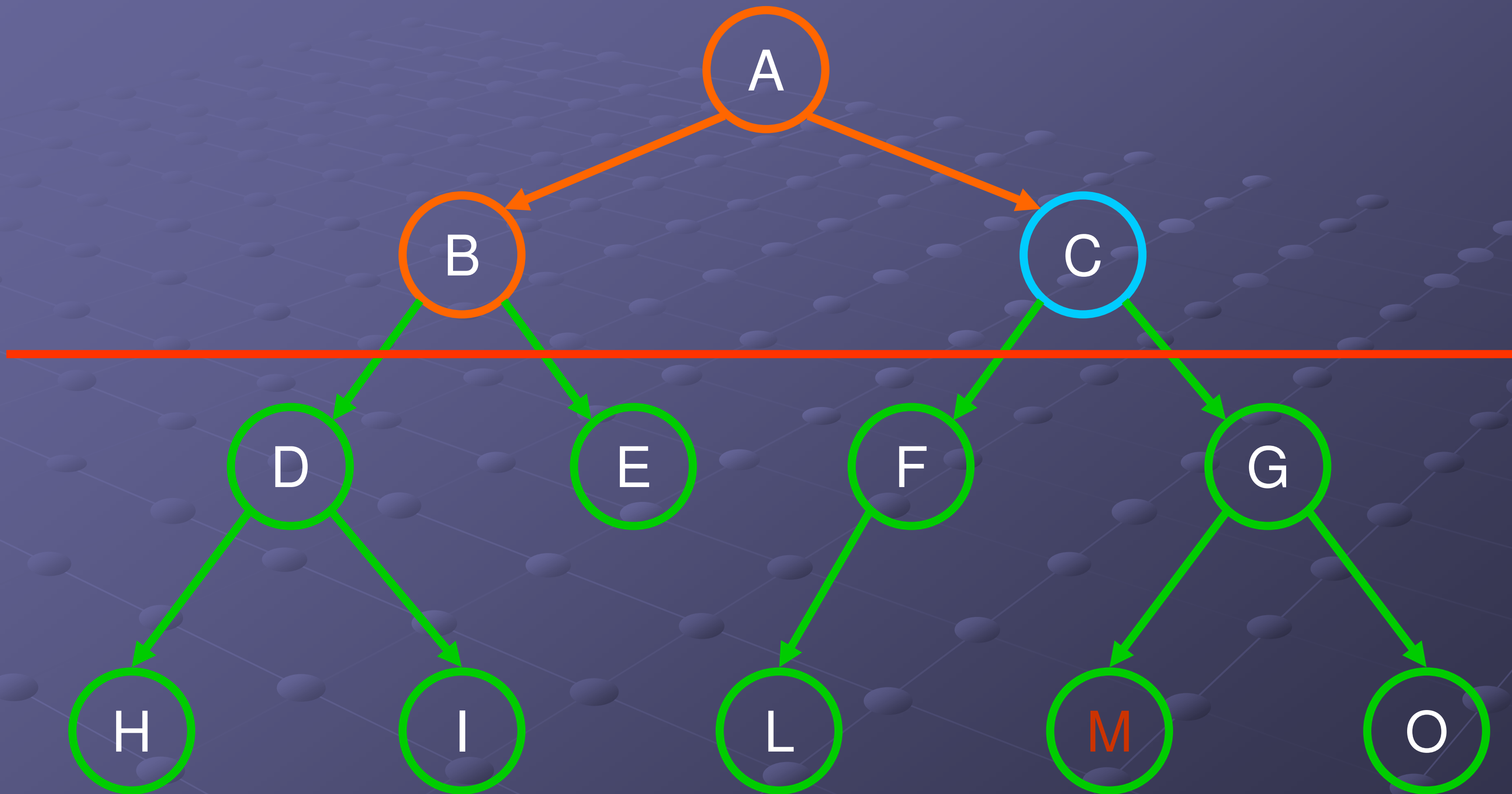
Iterative Deepening DFS Search



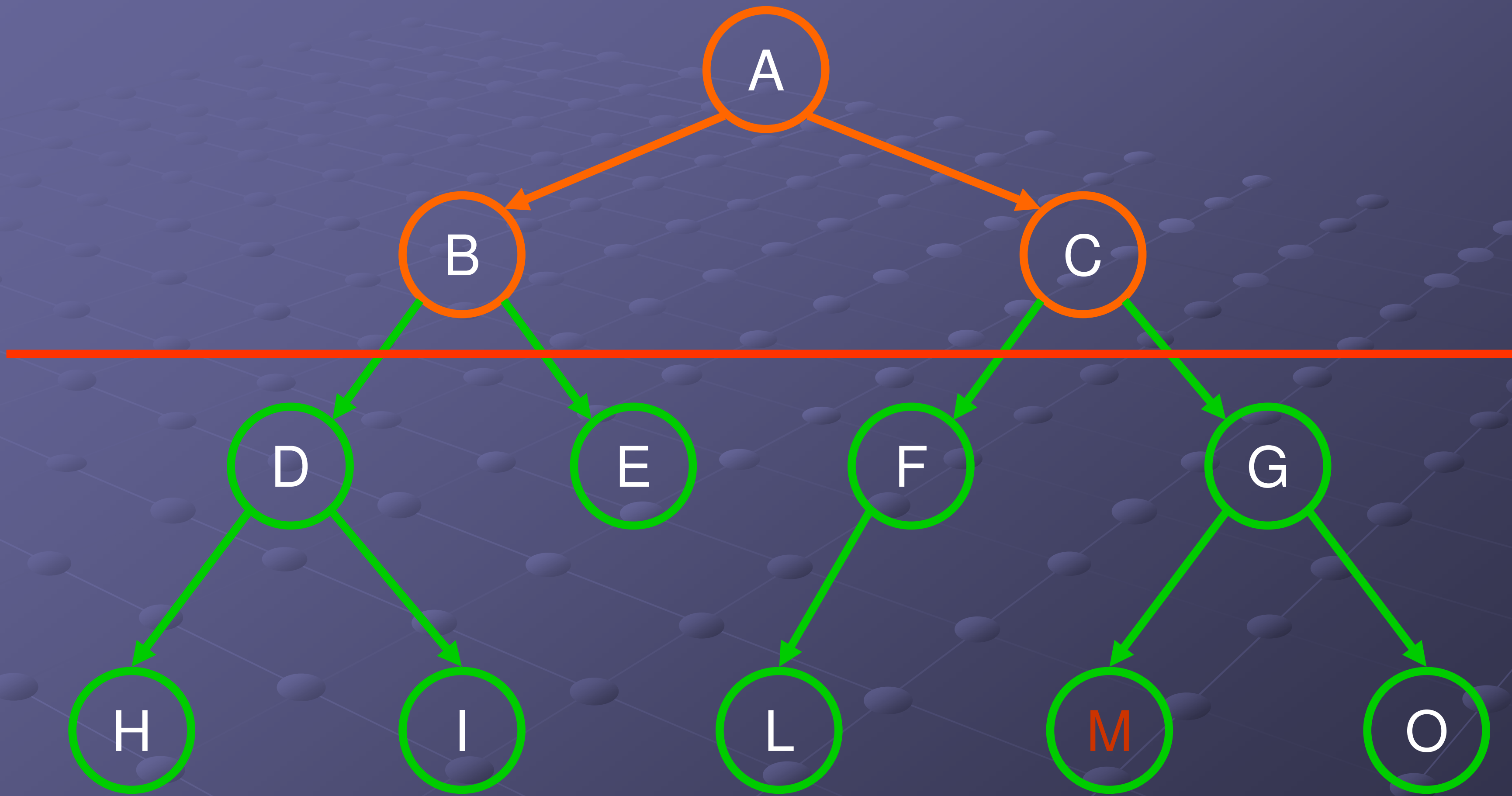
Iterative Deepening DFS Search



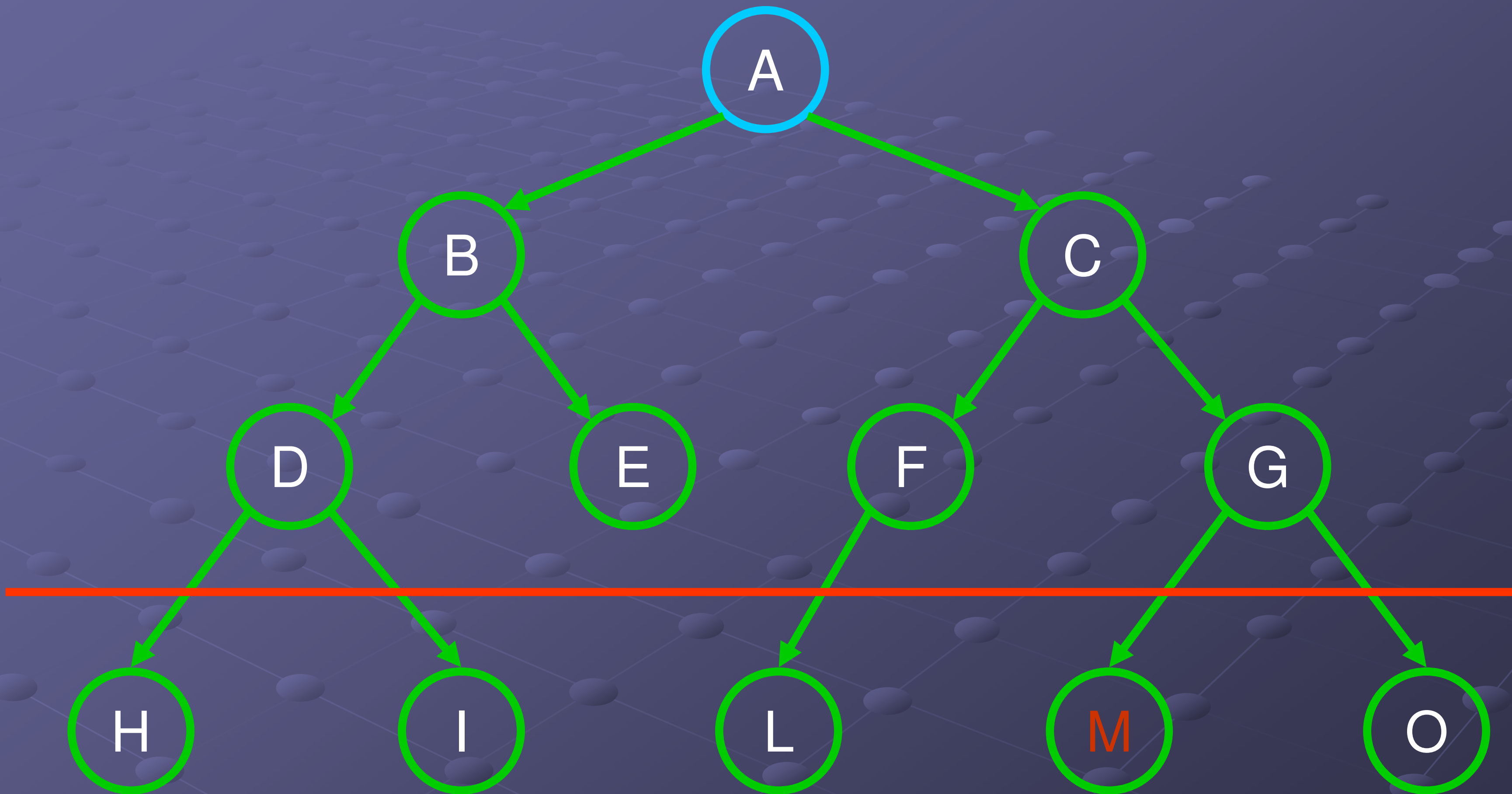
Iterative Deepening DFS Search



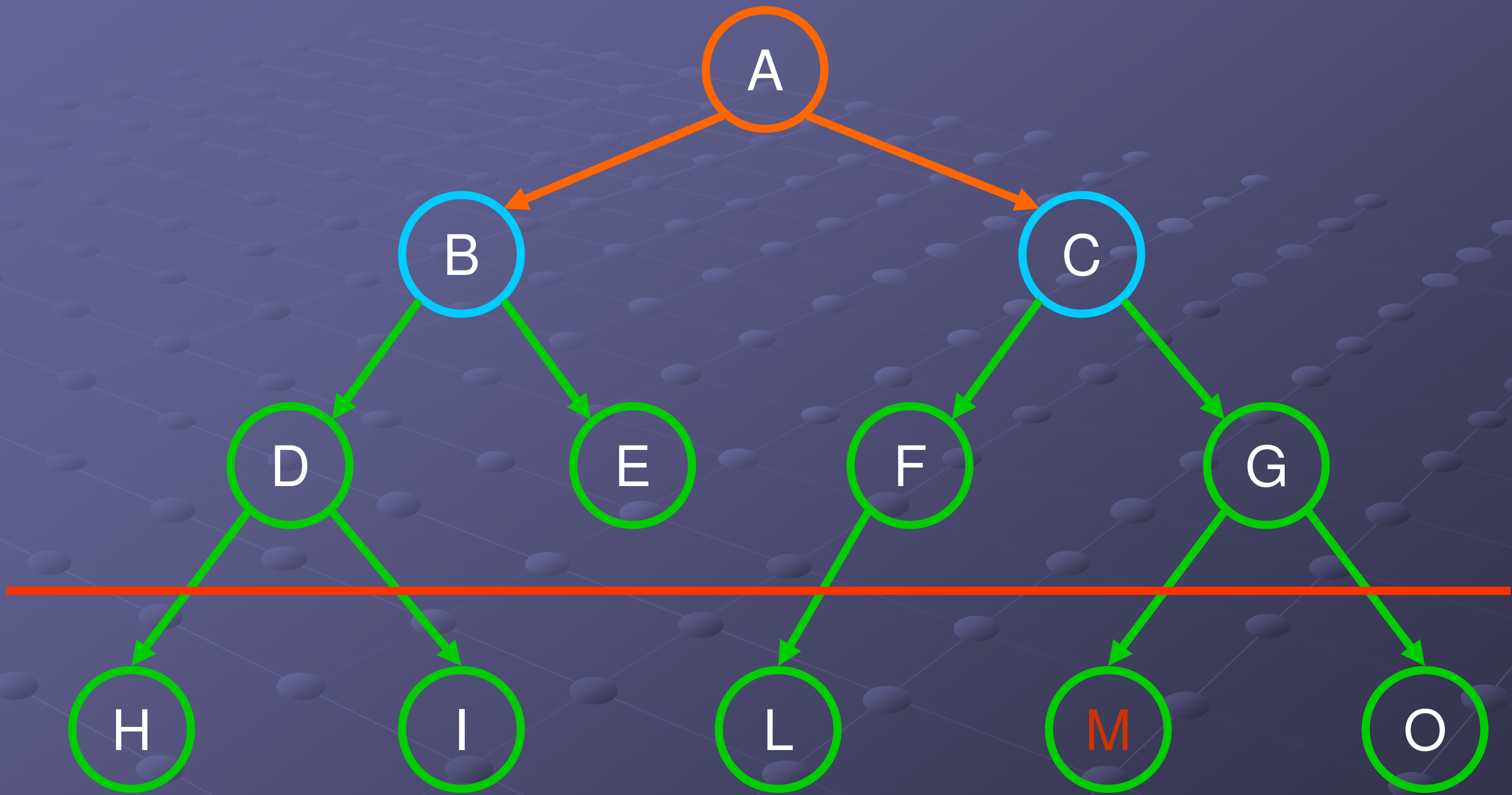
Iterative Deepening DFS Search



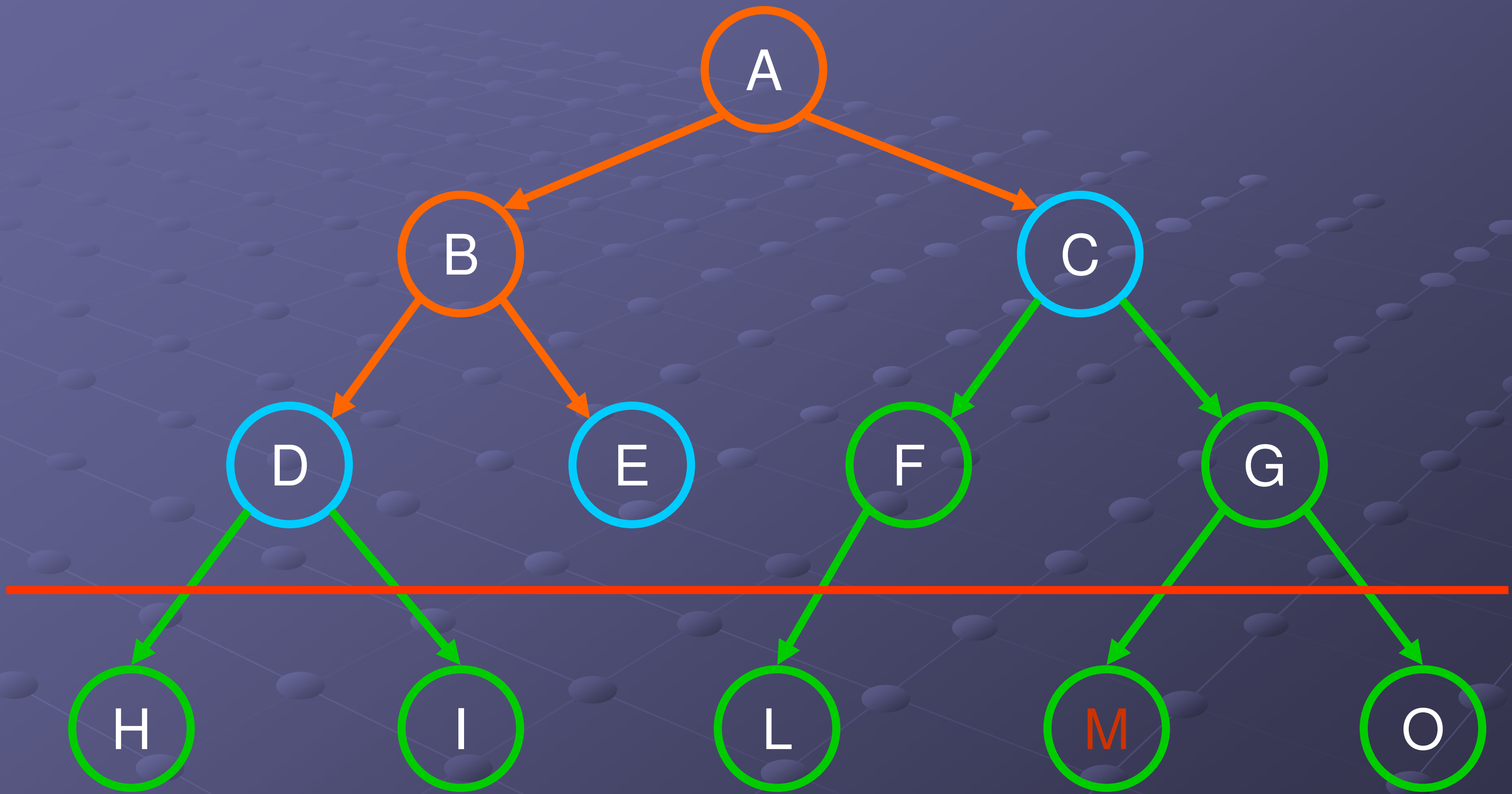
Iterative Deepening DFS Search



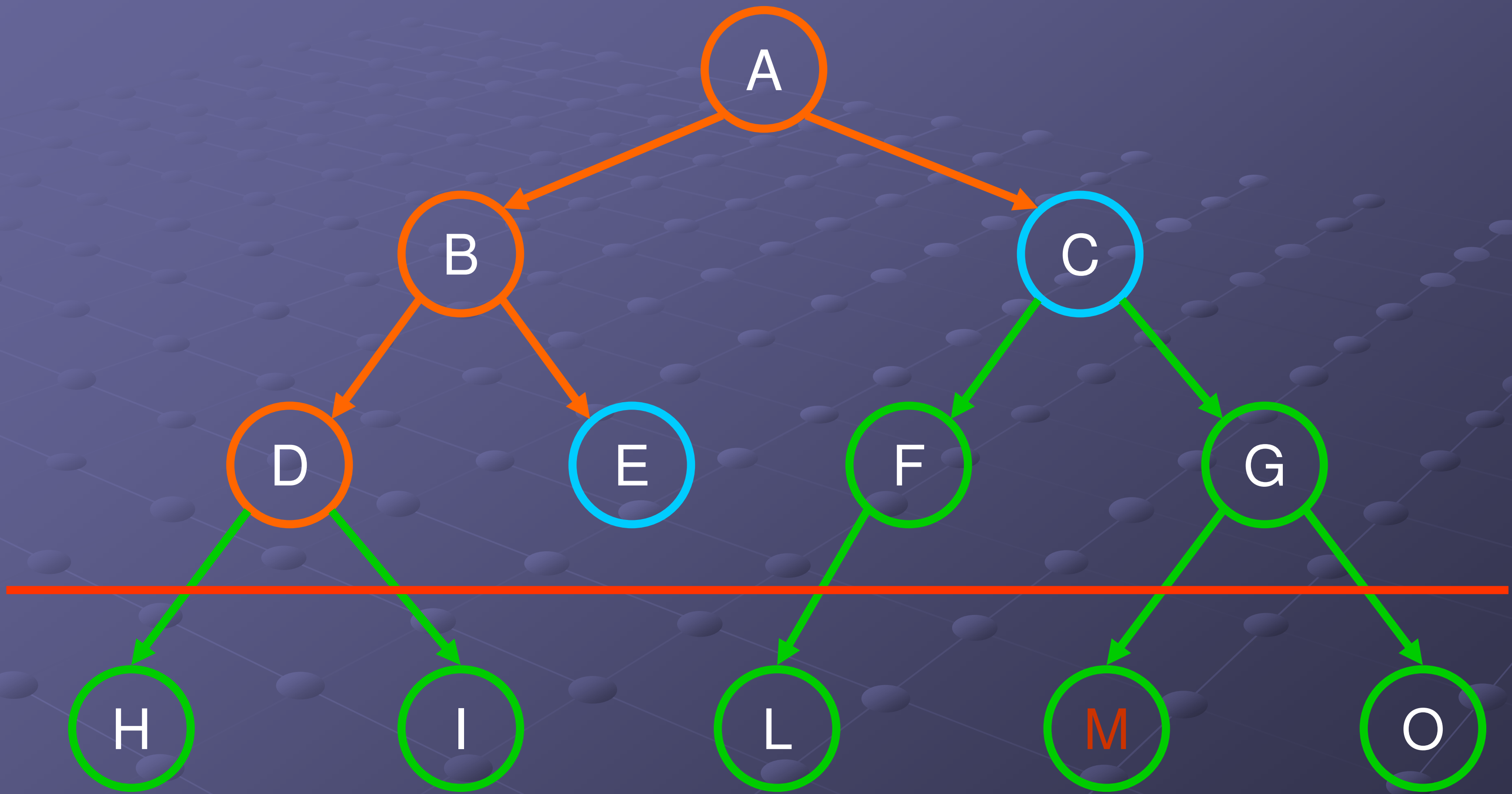
Iterative Deepening DFS Search



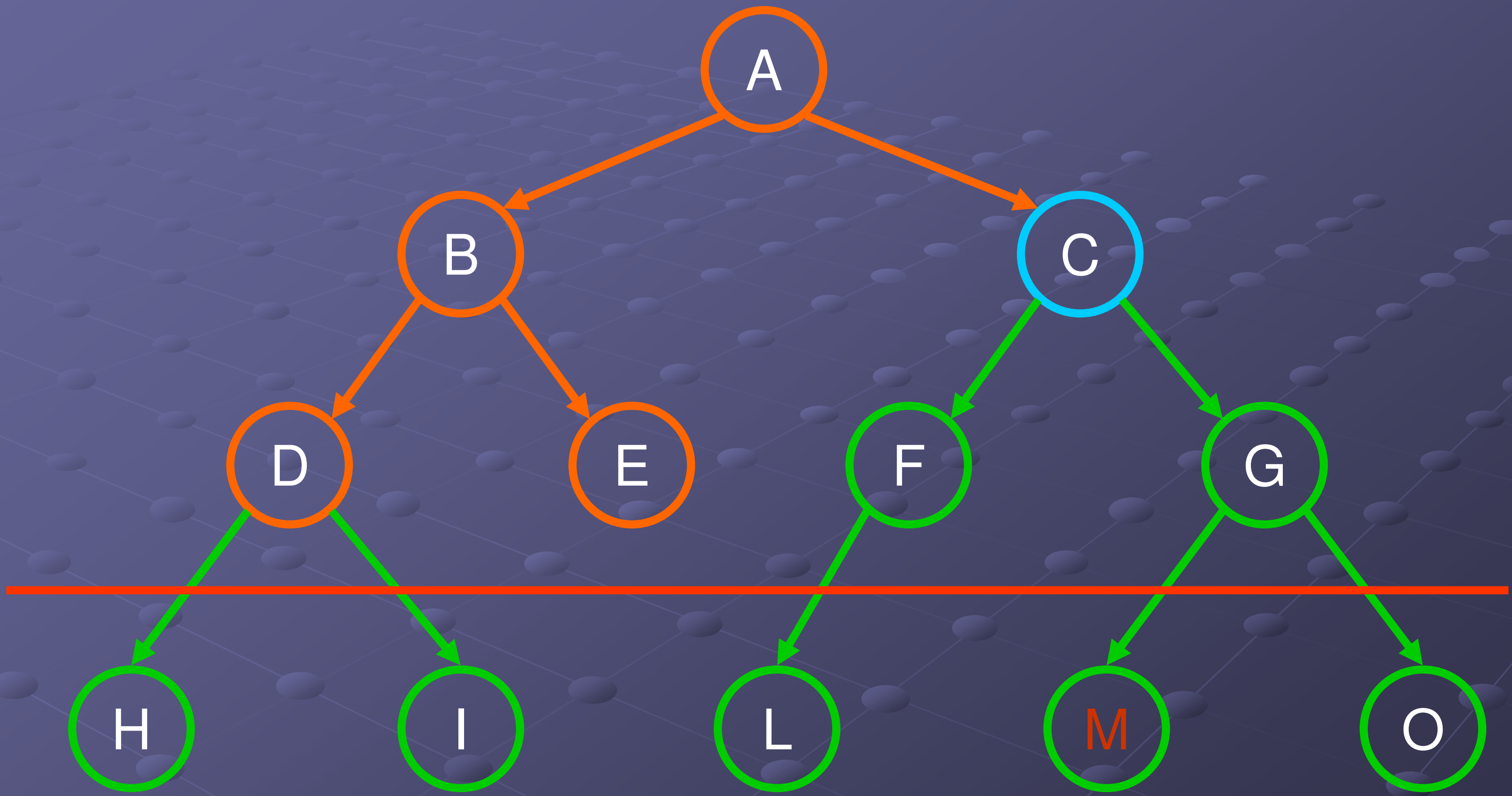
Iterative Deepening DFS Search



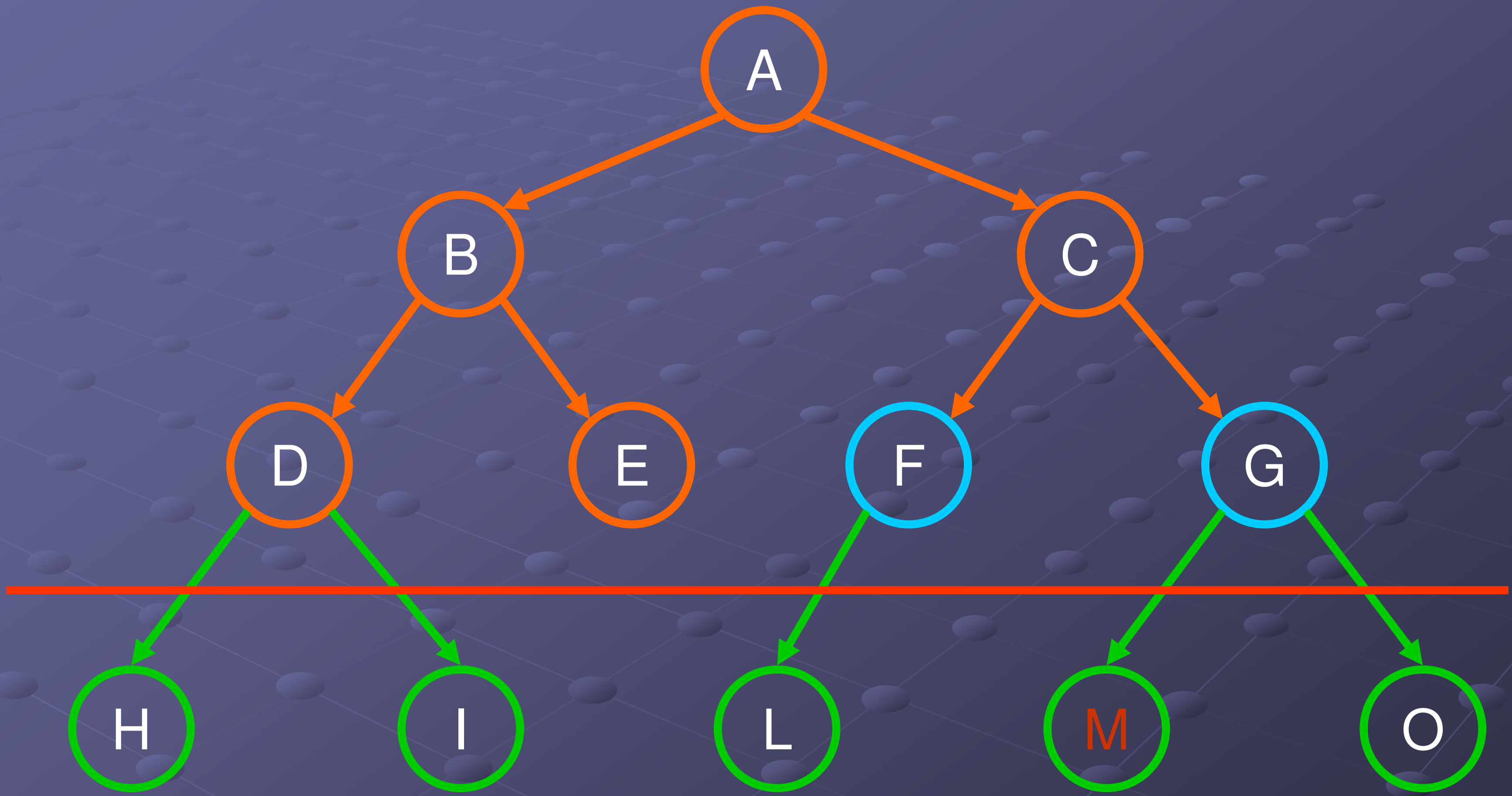
Iterative Deepening DFS Search



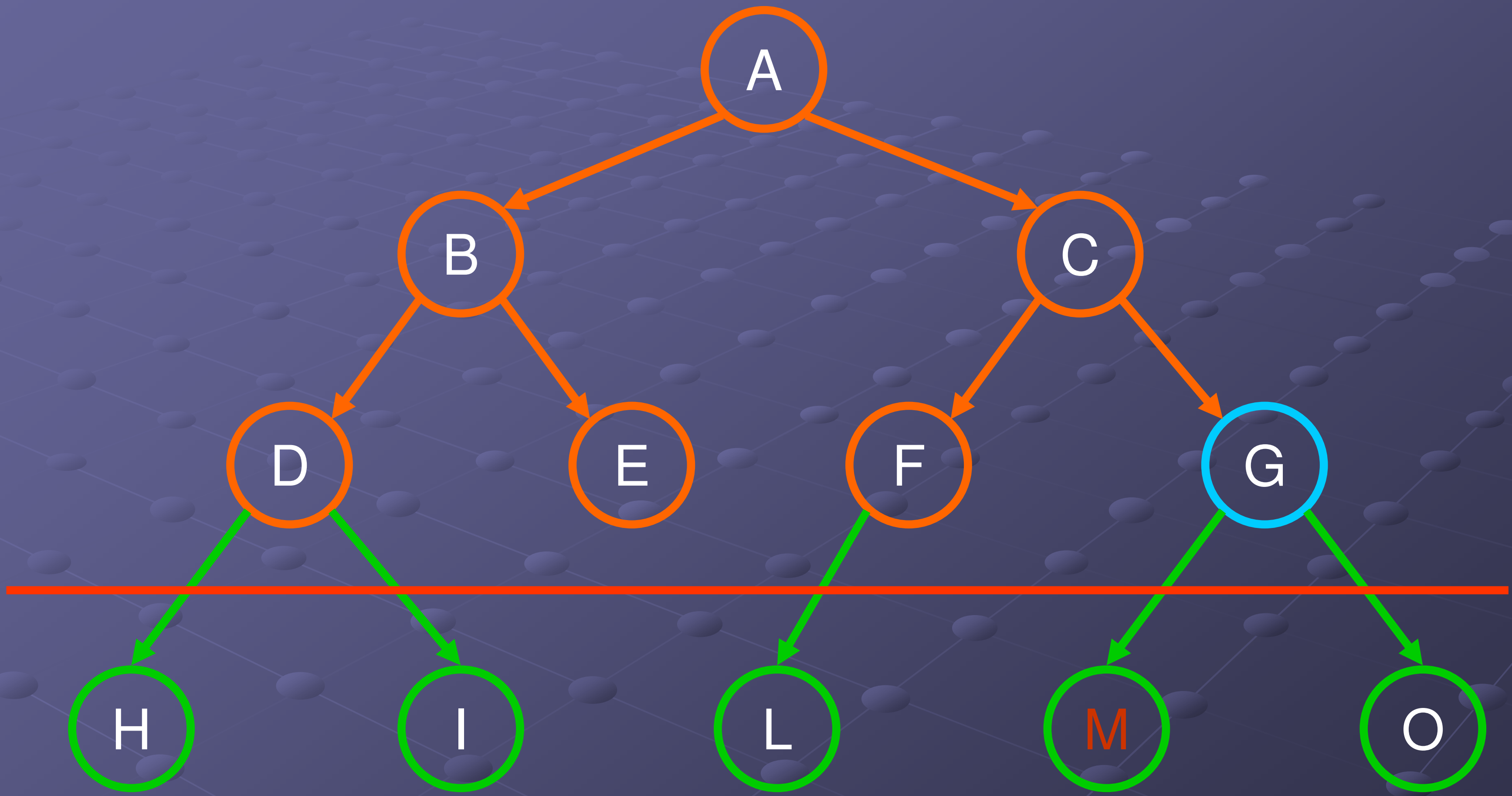
Iterative Deepening DFS Search



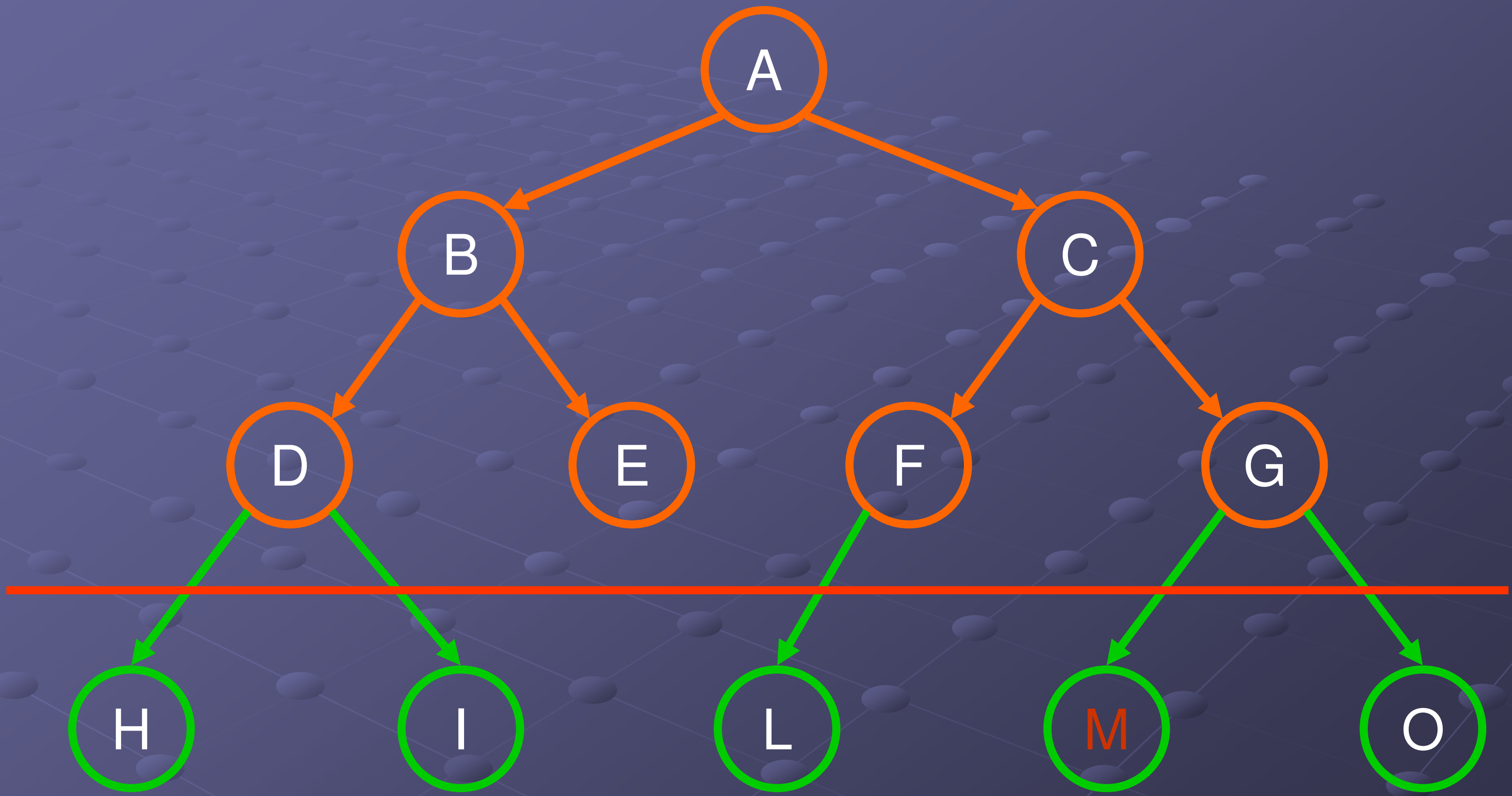
Iterative Deepening DFS Search



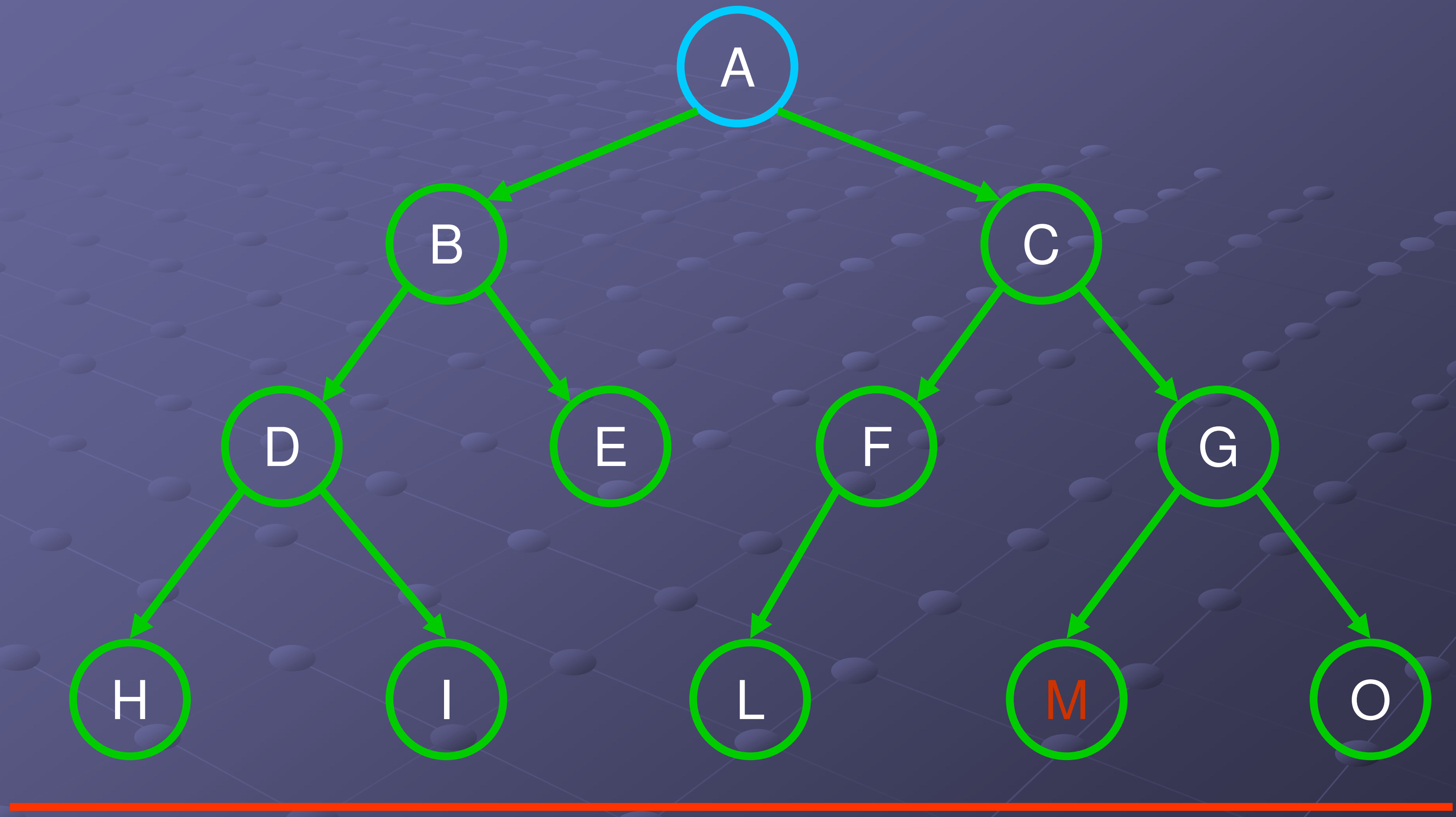
Iterative Deepening DFS Search



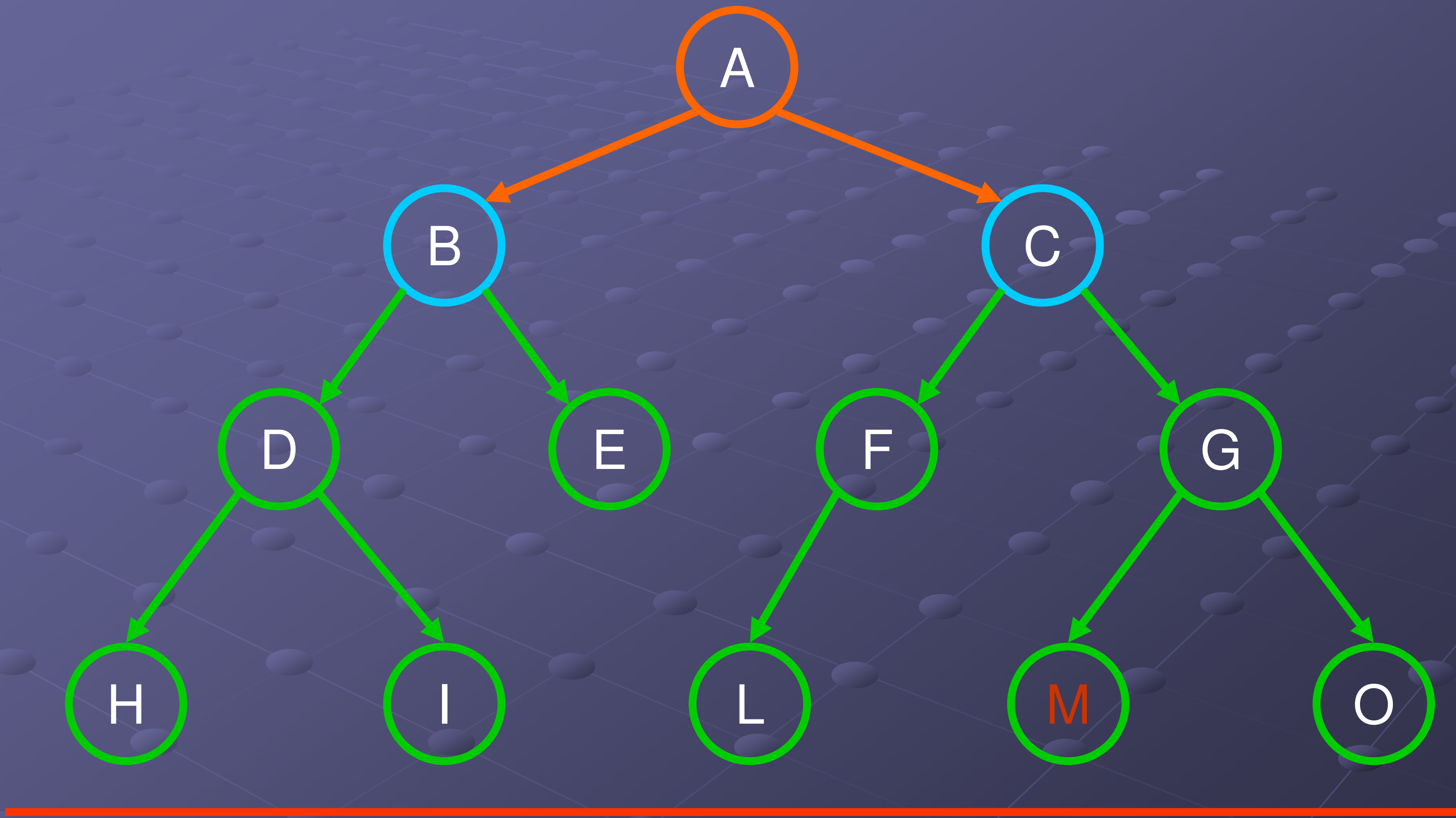
Iterative Deepening DFS Search



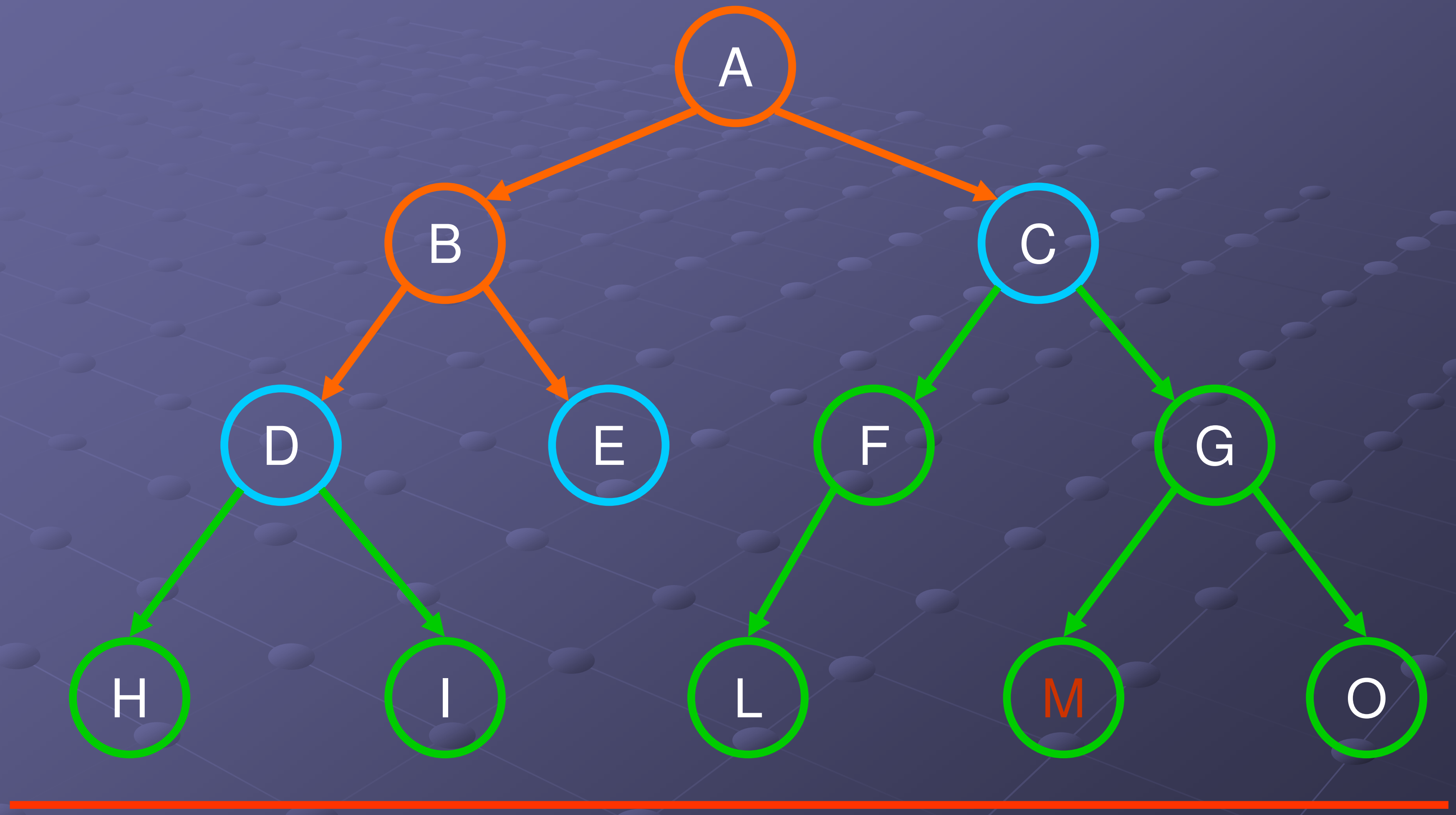
Iterative Deepening DFS Search



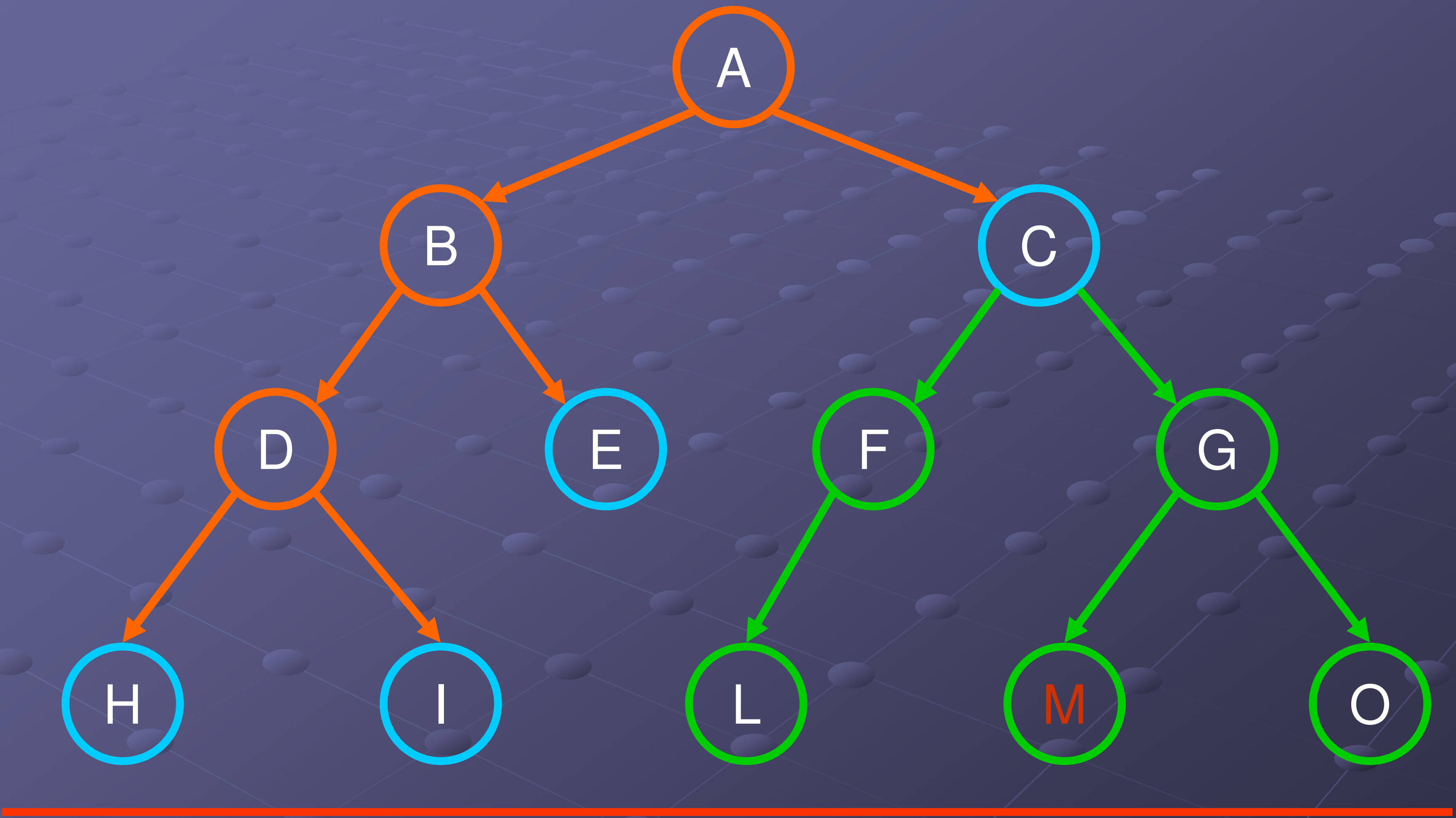
Iterative Deepening DFS Search



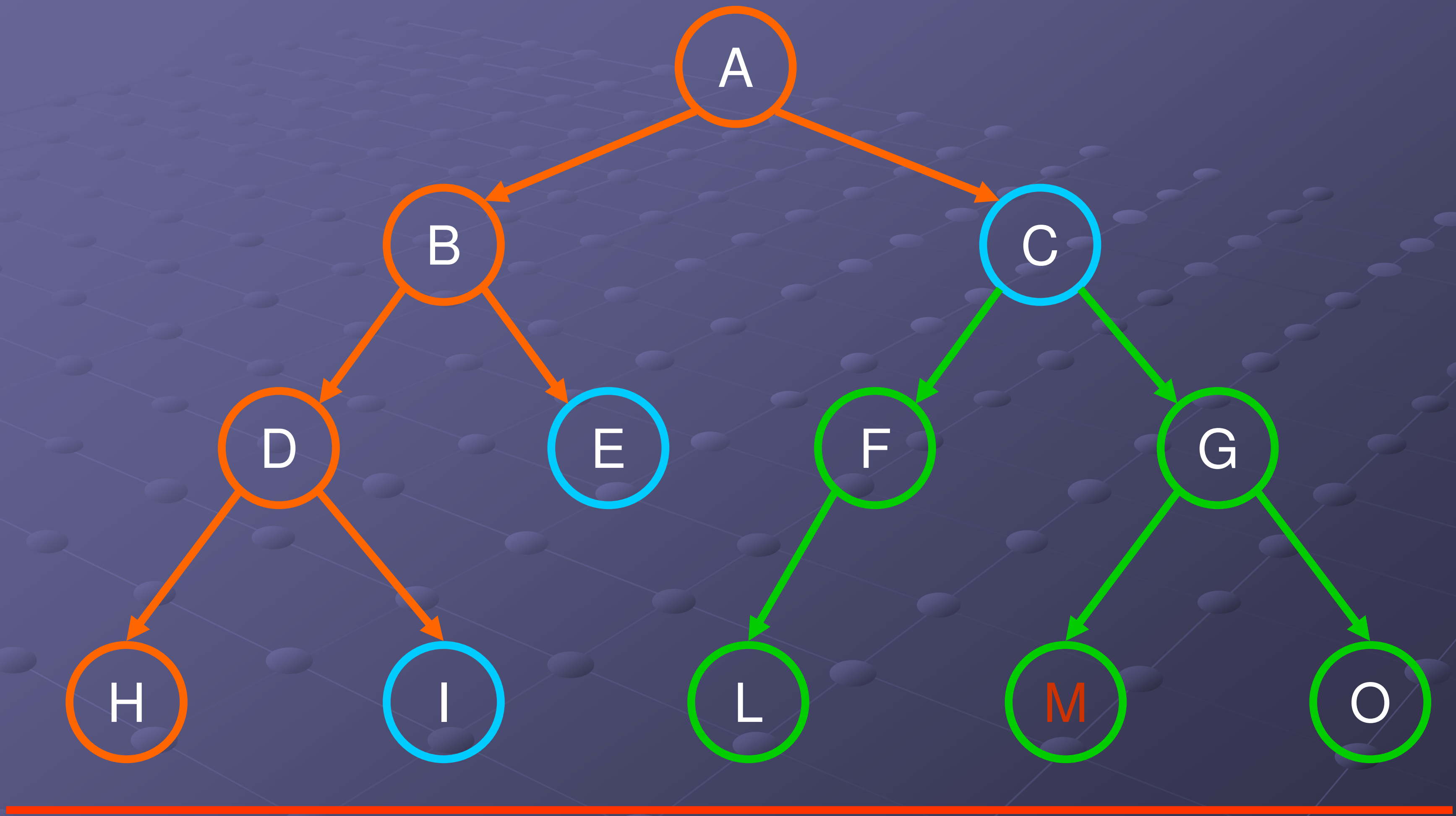
Iterative Deepening DFS Search



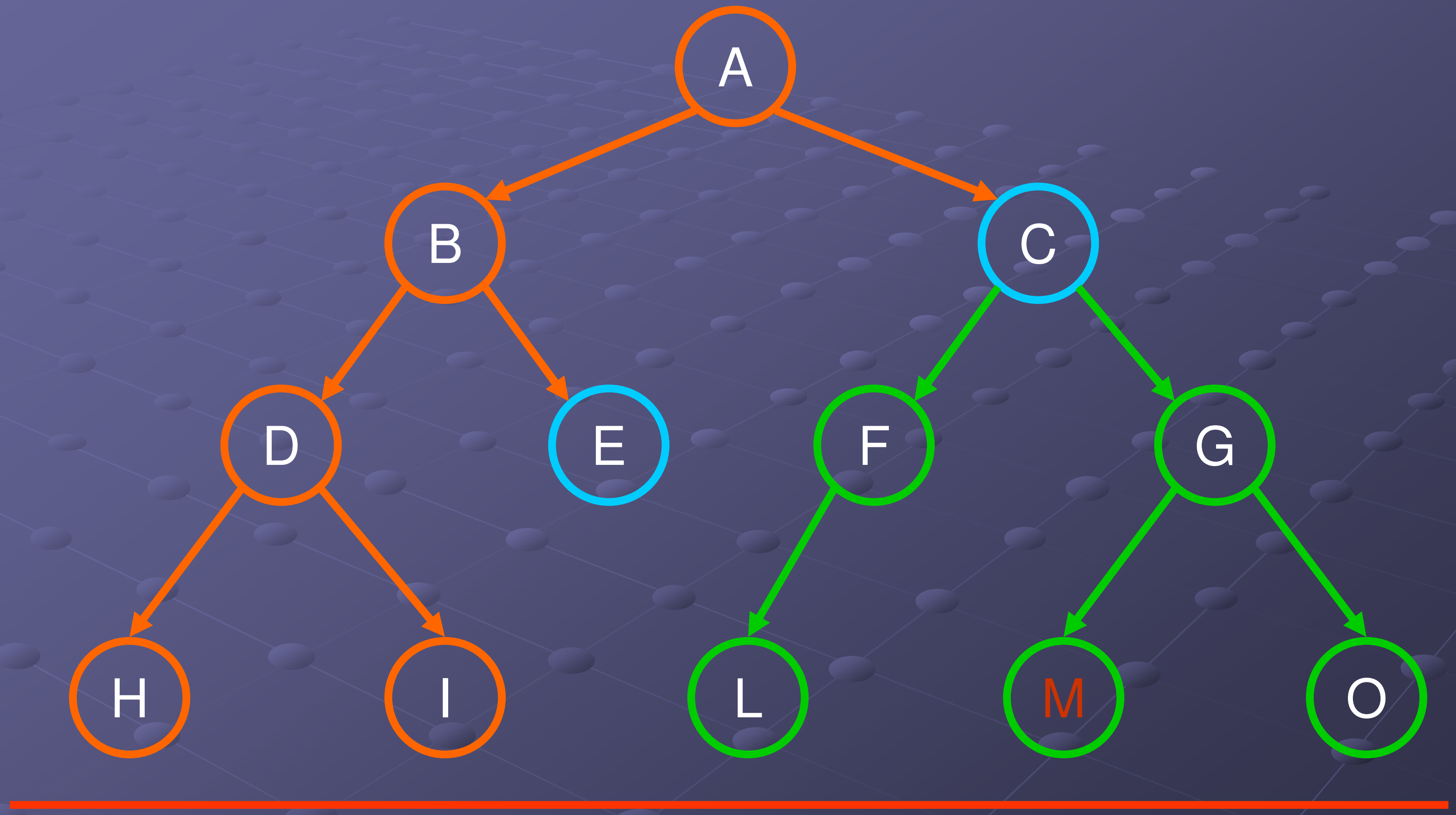
Iterative Deepening DFS Search



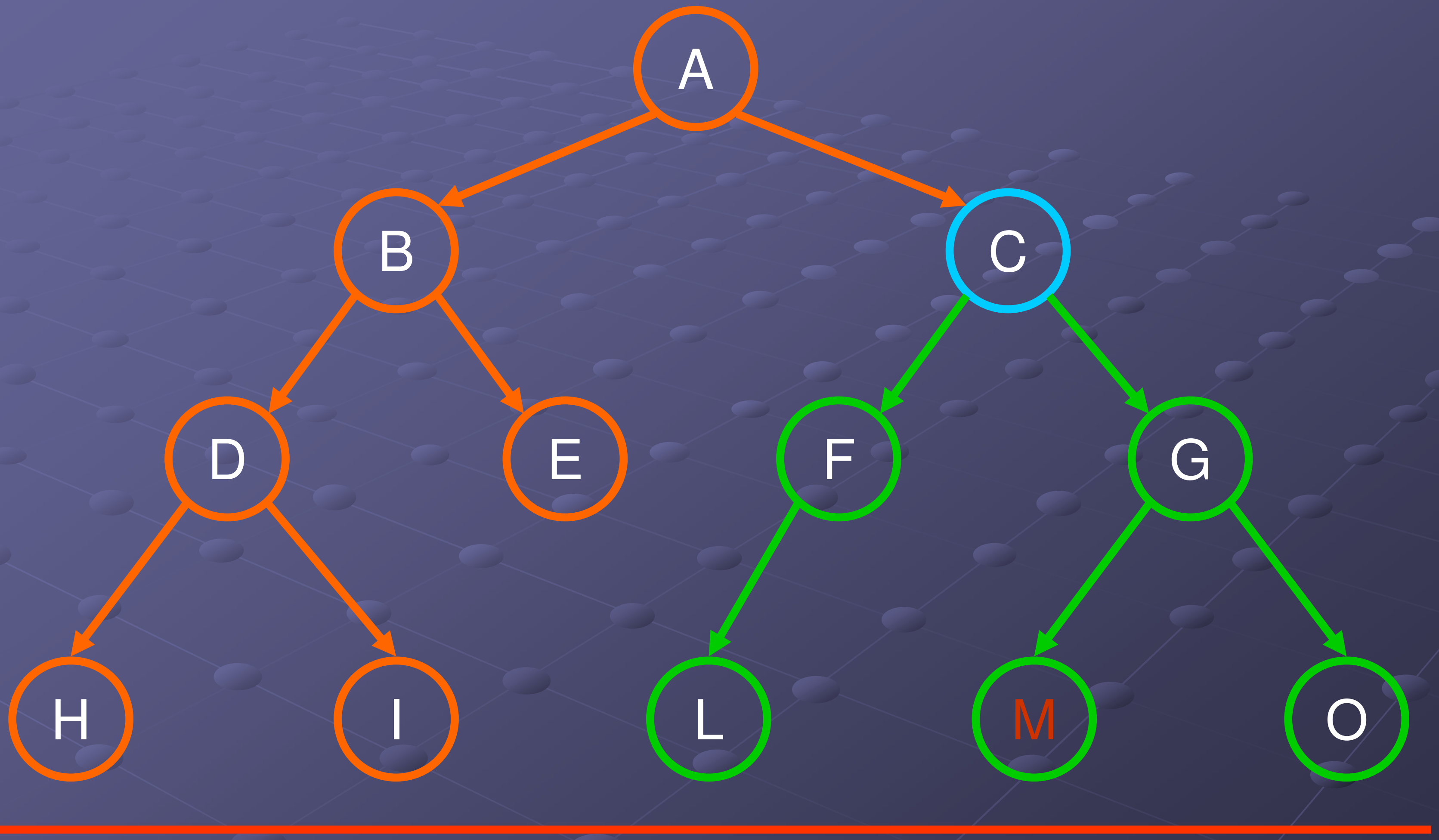
Iterative Deepening DFS Search



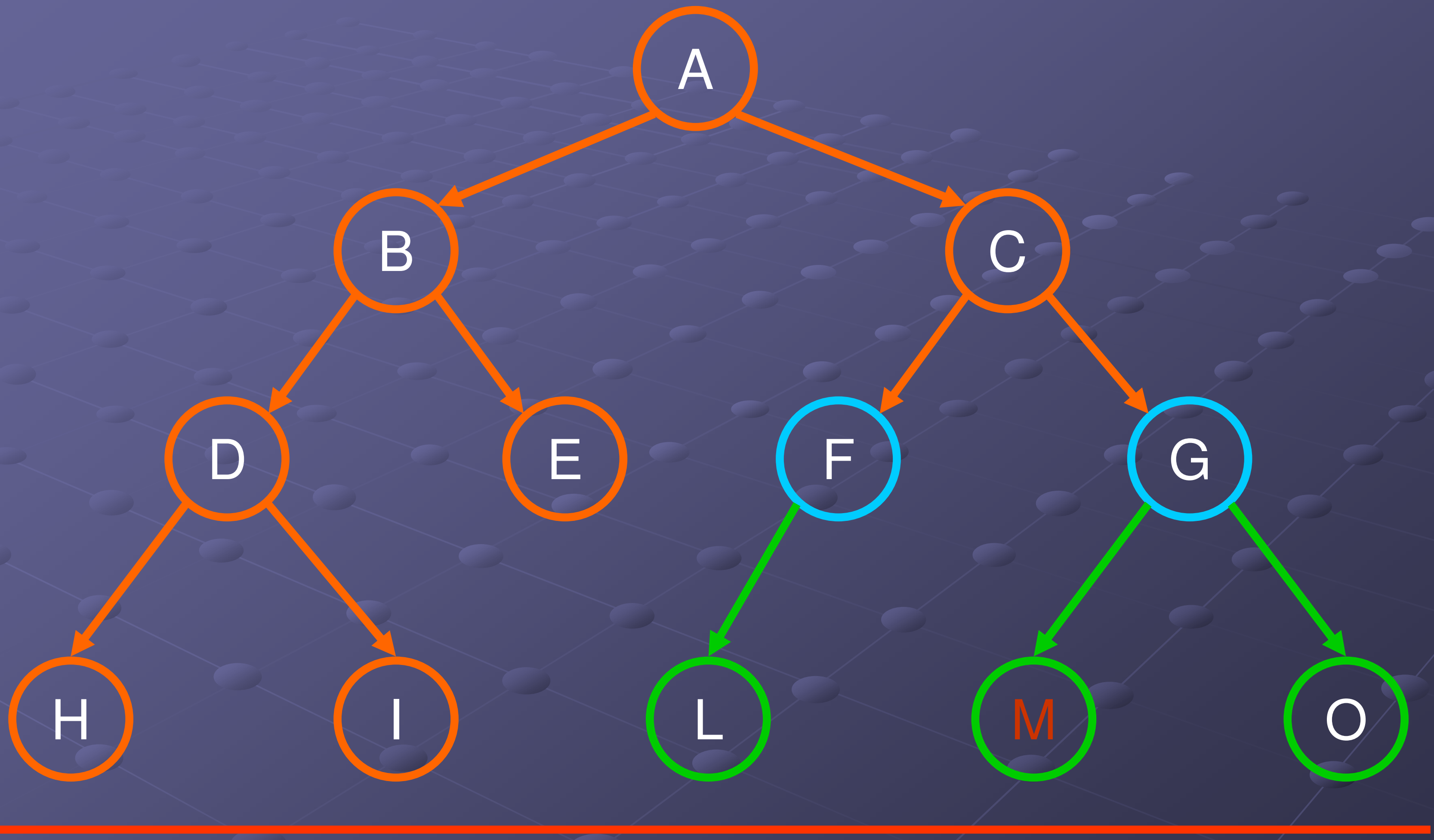
Iterative Deepening DFS Search



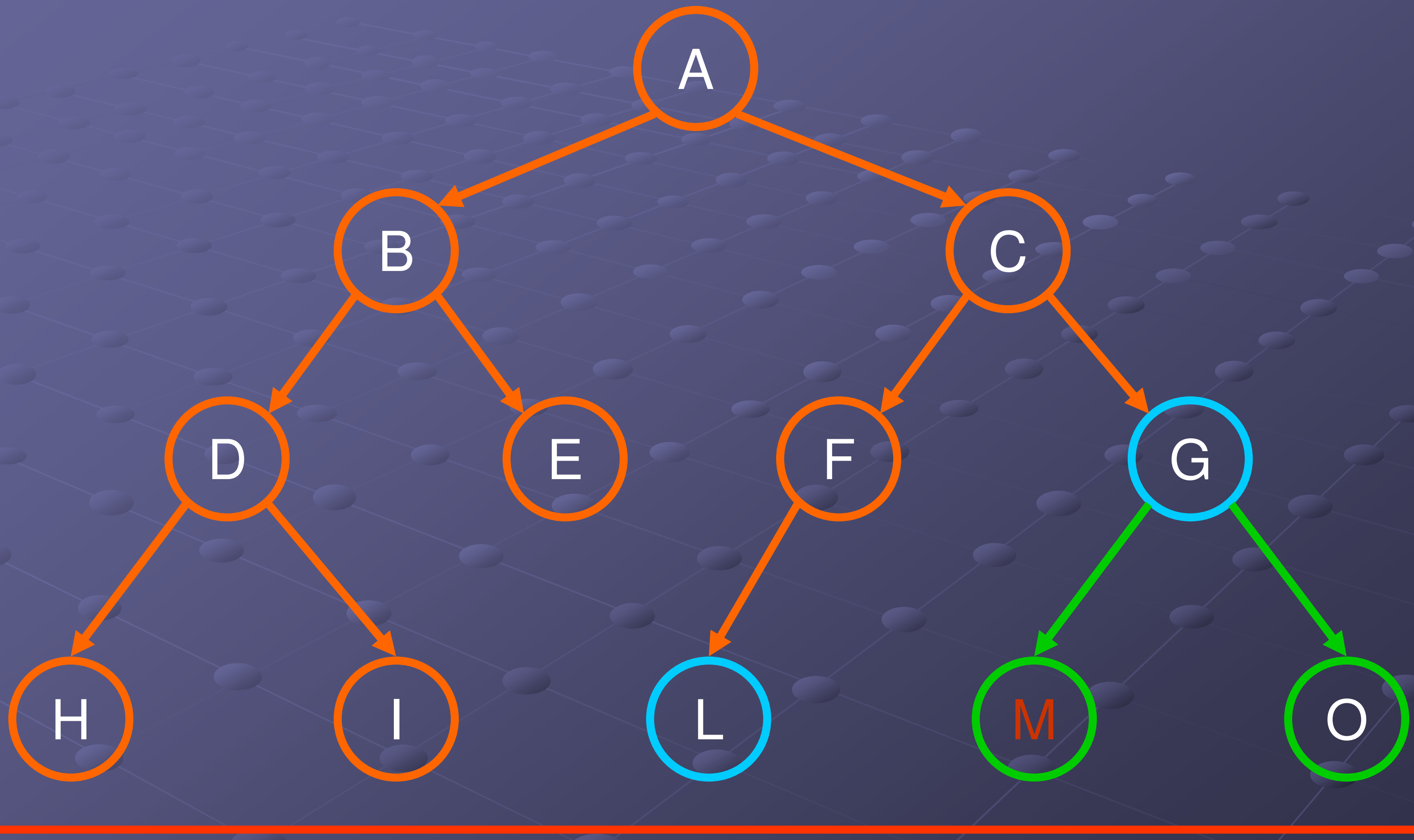
Iterative Deepening DFS Search



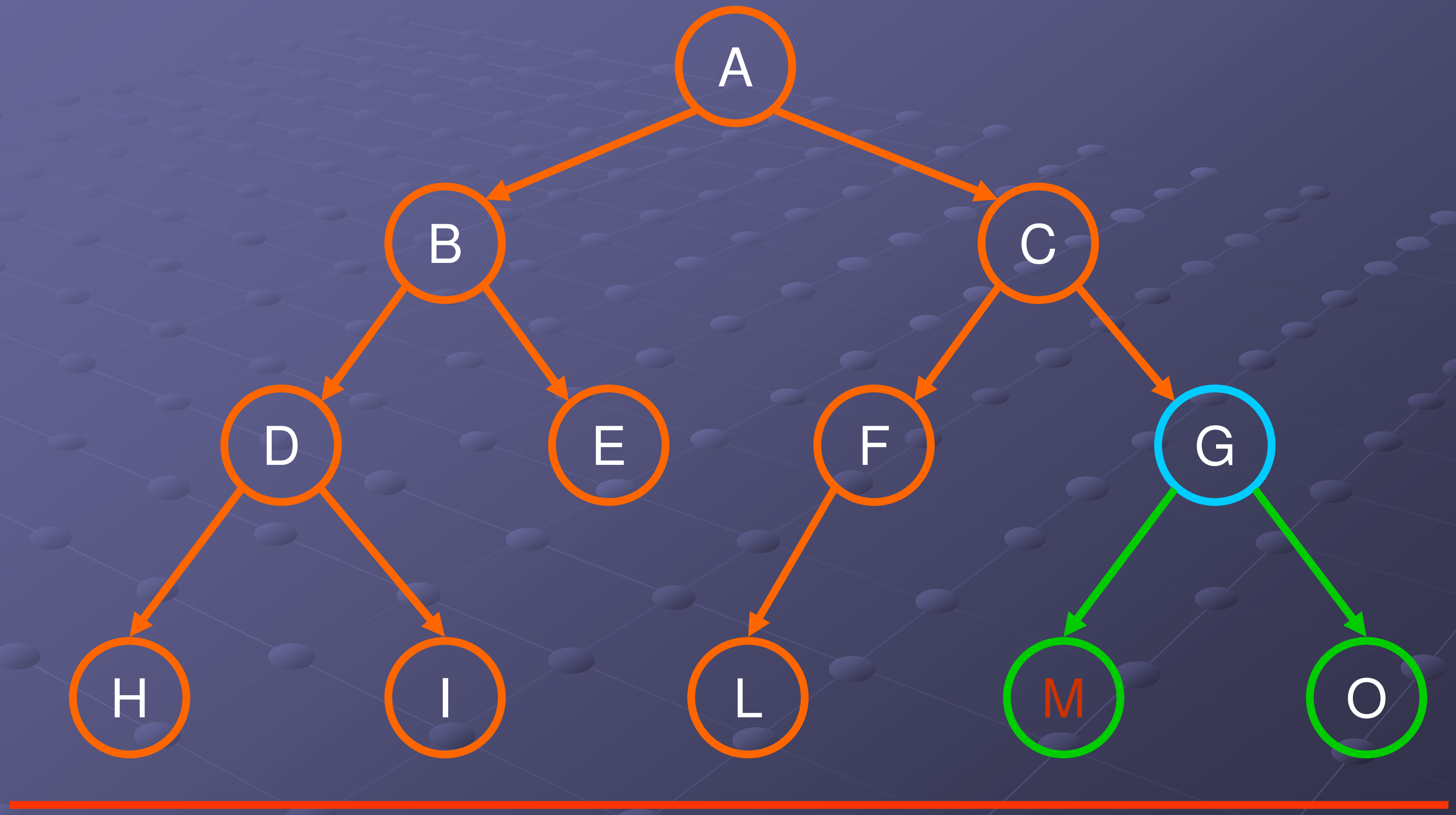
Iterative Deepening DFS Search



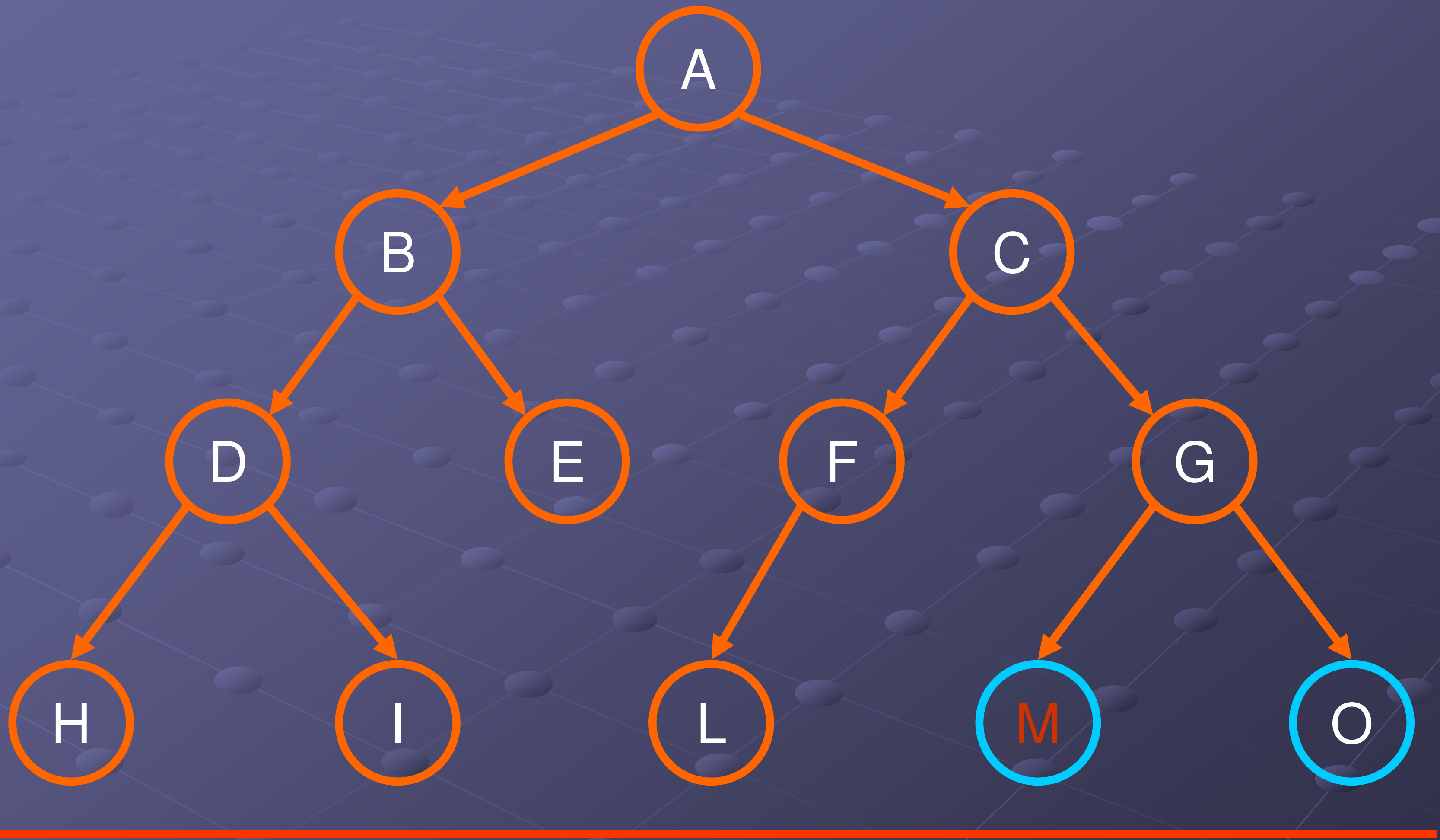
Iterative Deepening DFS Search



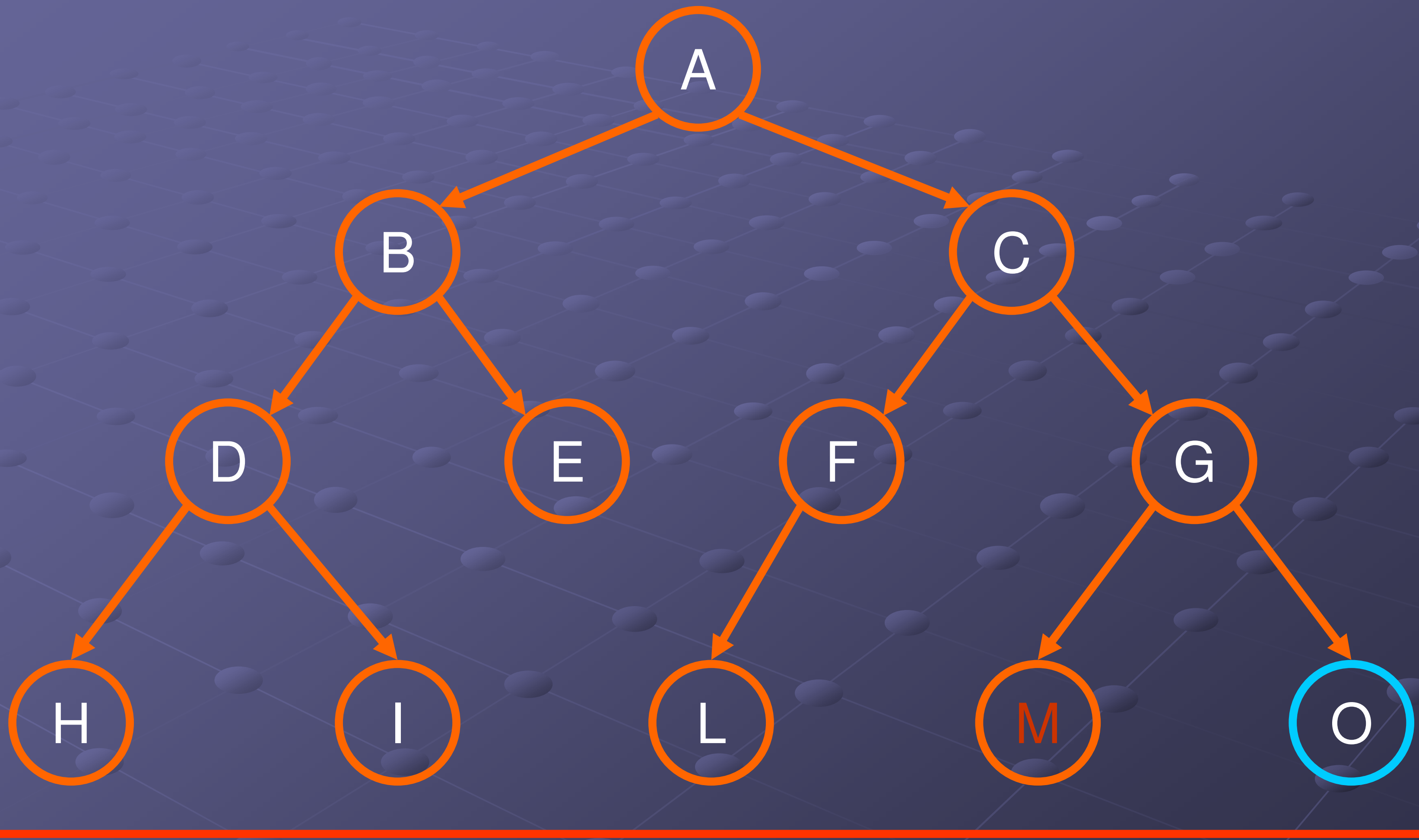
Iterative Deepening DFS Search



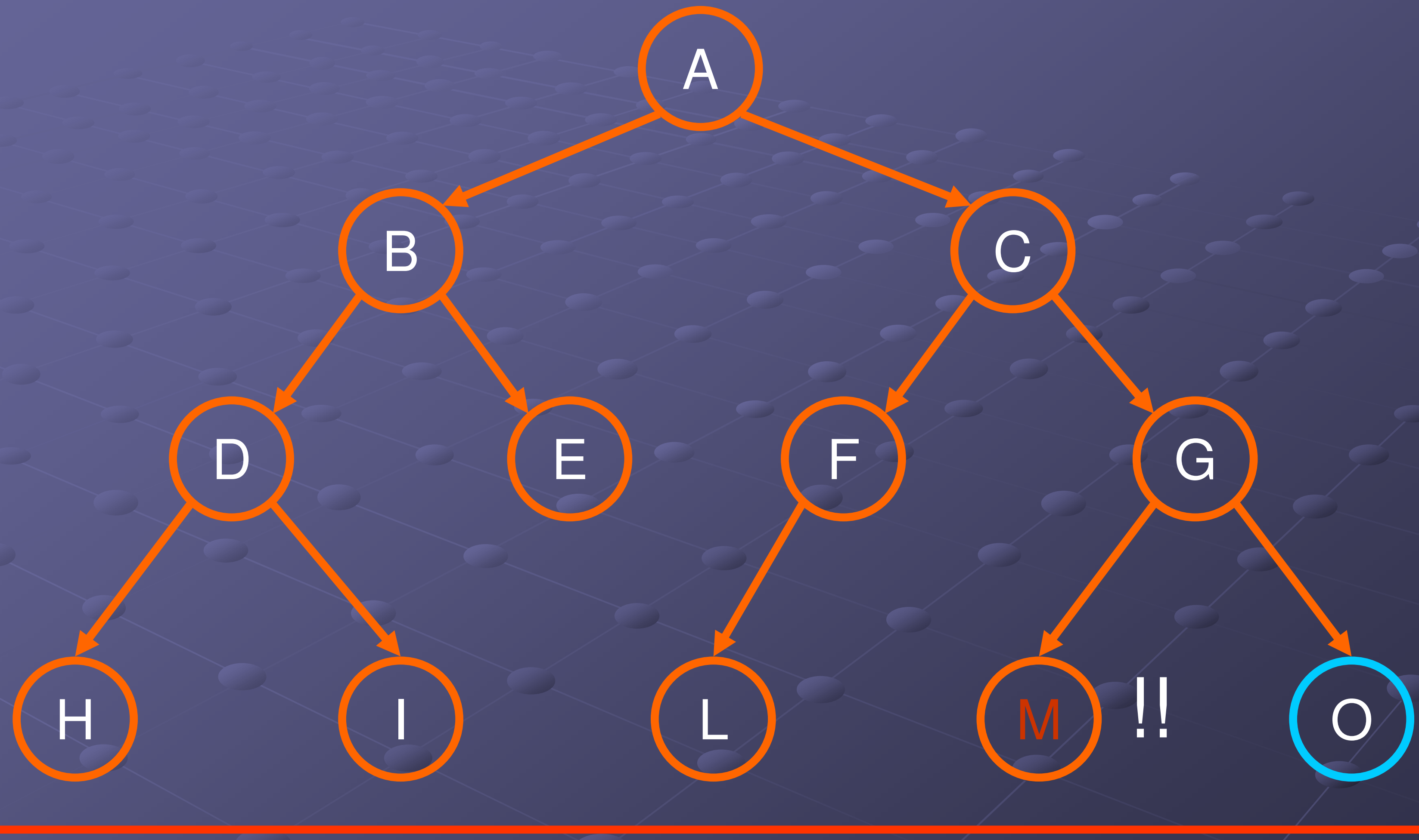
Iterative Deepening DFS Search



Iterative Deepening DFS Search



Iterative Deepening DFS Search



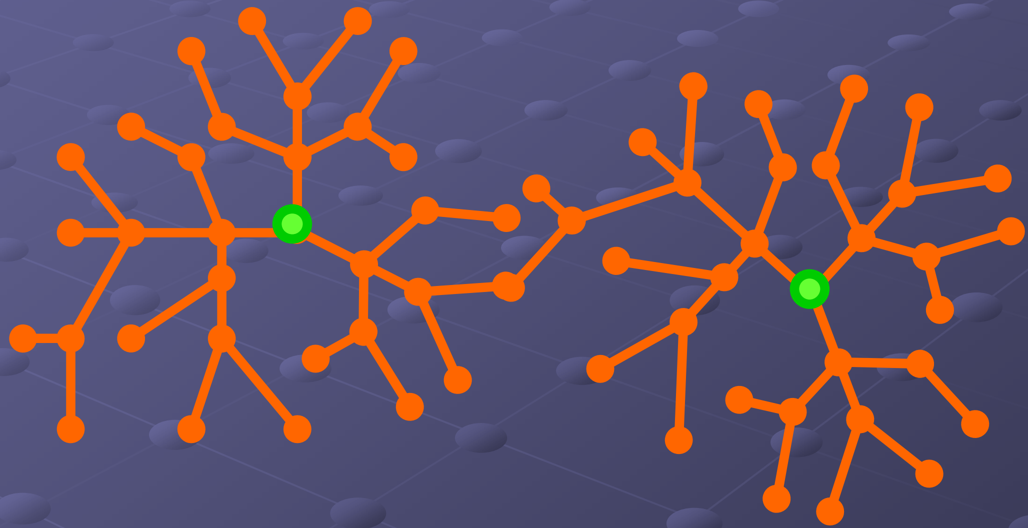
Uniform cost search

- Expand in order of cost
- Identical to Breadth-First Search if all step costs are equal

- Number of nodes expanded depends on $\varepsilon > 0$ – the smallest cost action cost.
- (If $\varepsilon = 0$, uniform-Cost search may infinite-loop)

Bi-directional Search

- Extend BFS from Start and Goal states simultaneously



$$b^{d/2} + b^{d/2} << b^d$$

Bi-directional Search

Restrictions:

- Can only use if 1 (or a few) *known* goal nodes
- Can only use if actions are reversible
  i.e. if we can efficiently calculate Pred*(s)* (for state *s*)
  where $Pred(s) = \{r : \exists a \; r \xrightarrow{a} s\}$

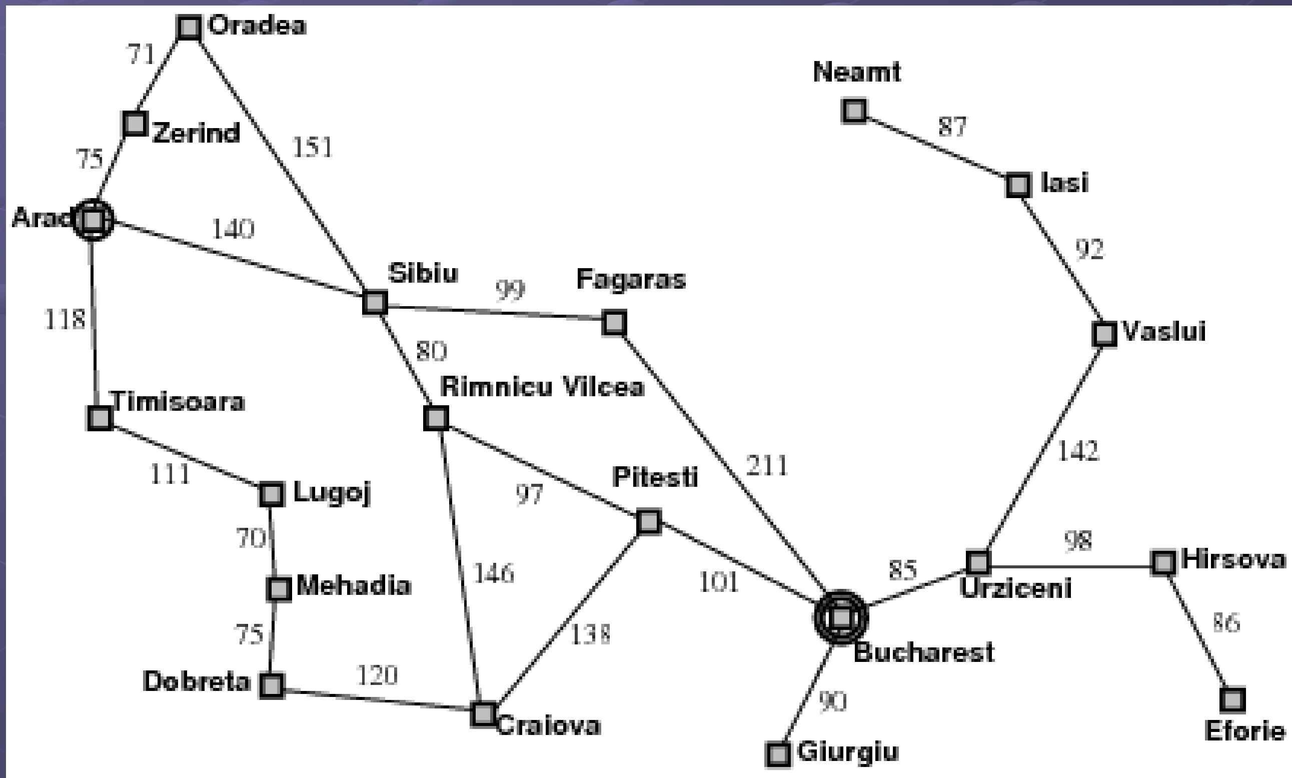
Summary

Criterion	BFS	DFS	IDDFS	Uniform Cost	Bi-direc-tional
Complete	Yes	No	Yes	Yes	Yes
Time	$O(b^{d+1})$	$O(b^m)$	$O(b^d)$	$O(b^{C^*/\varepsilon})$	$O(b^{d/2})$
Space	$O(b^{d+1})$	$O(bm)$	$O(bd)$	$O(b^{C^*/\varepsilon})$	$O(b^{d/2})$
Optimal	Yes, for unit cost	No	Yes, for unit cost	Yes	Yes, for unit cost

b = branching factor ; m = max depth of tree ; d = depth of goal node
C^* = cost of goal ; ε = min action cost

Repeated State Detection

- Test for previously-seen states
- Essentially, search graph mimics state graph



Repeated State Detection

- ✓ Can make exponential improvement in time
- ✓ Can make DFS complete
- ✗ DFS now has potentially exponential space requirement
- ✗ Have to be careful to guarantee optimality

In practice, almost always use repeated state detection

A* Search

Idea: avoid expanding paths that are already expensive

Evaluation function $f(n) = g(n) + h(n)$

$g(n)$ = cost so far to reach n
$h(n)$ = estimated cost from n to the closest goal
f(n) = estimated total cost of path through n to goal

A$*$ search uses an *admissible* heuristic
i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from n.
(Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G.)
E.g., $h_{SLD}(n)$ never overestimates the actual road distance
When $h(n)$ is admissible, $f(n)$ never overestimates the total cost of the shortest path through n to the goal
Theorem: A$*$ search is optimal

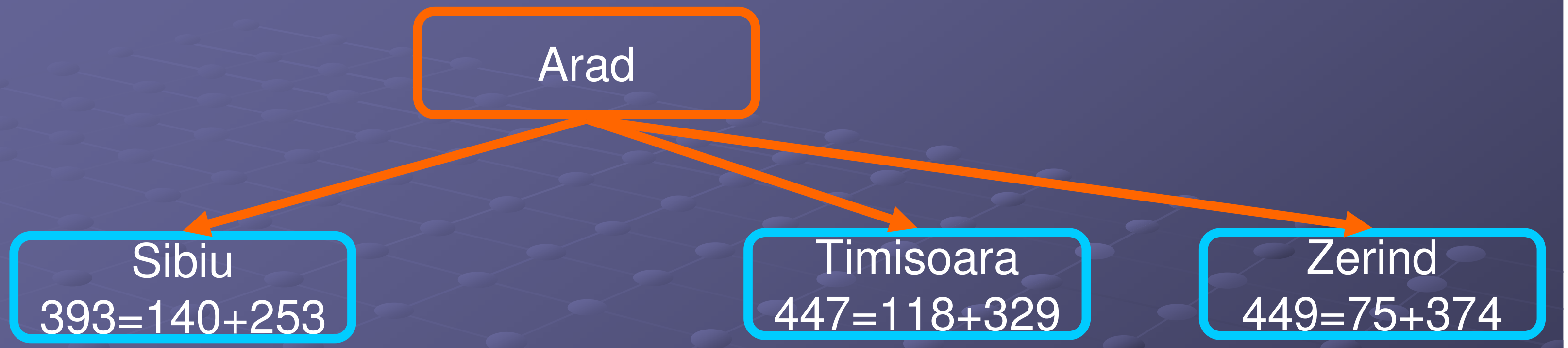
A* Example

Arad
366=0+366

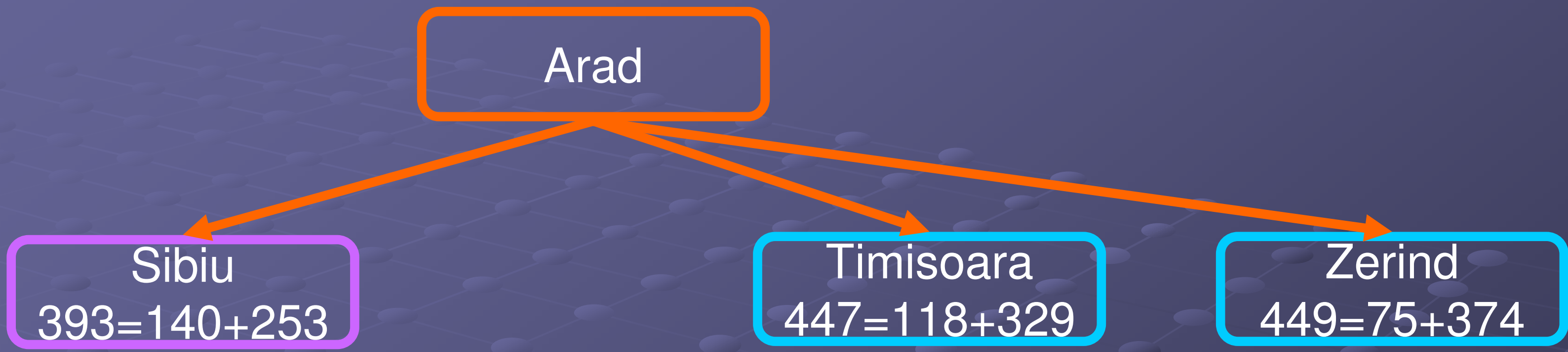
A* Example

Arad
366=0+366

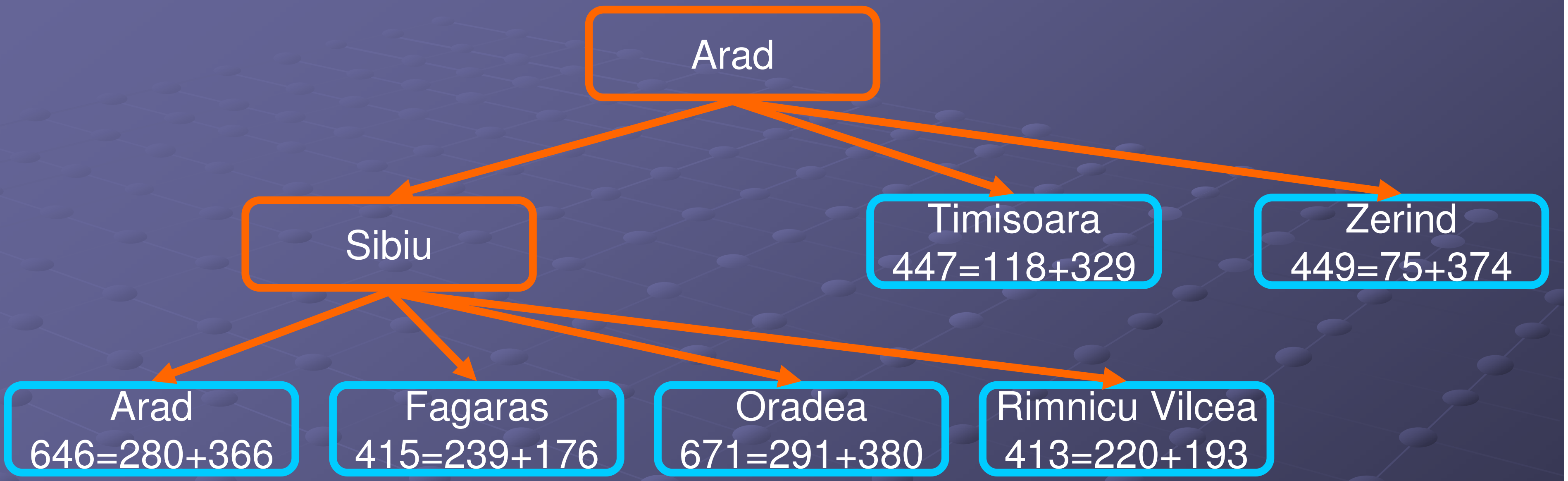
A* Example


Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374

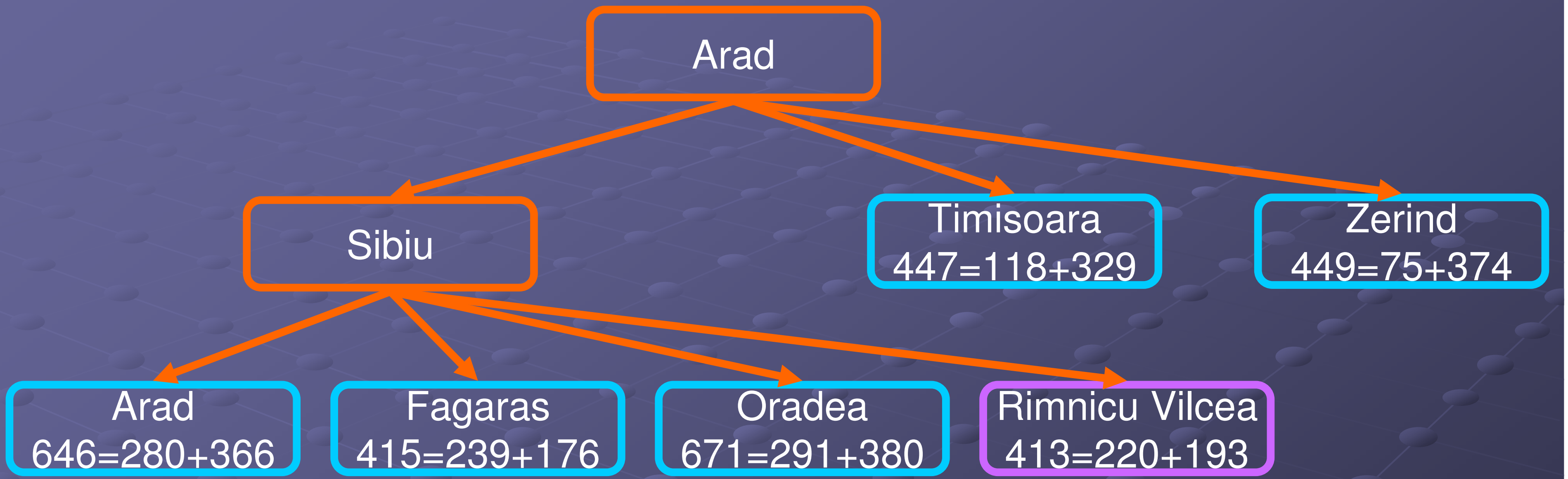
A* Example


Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374

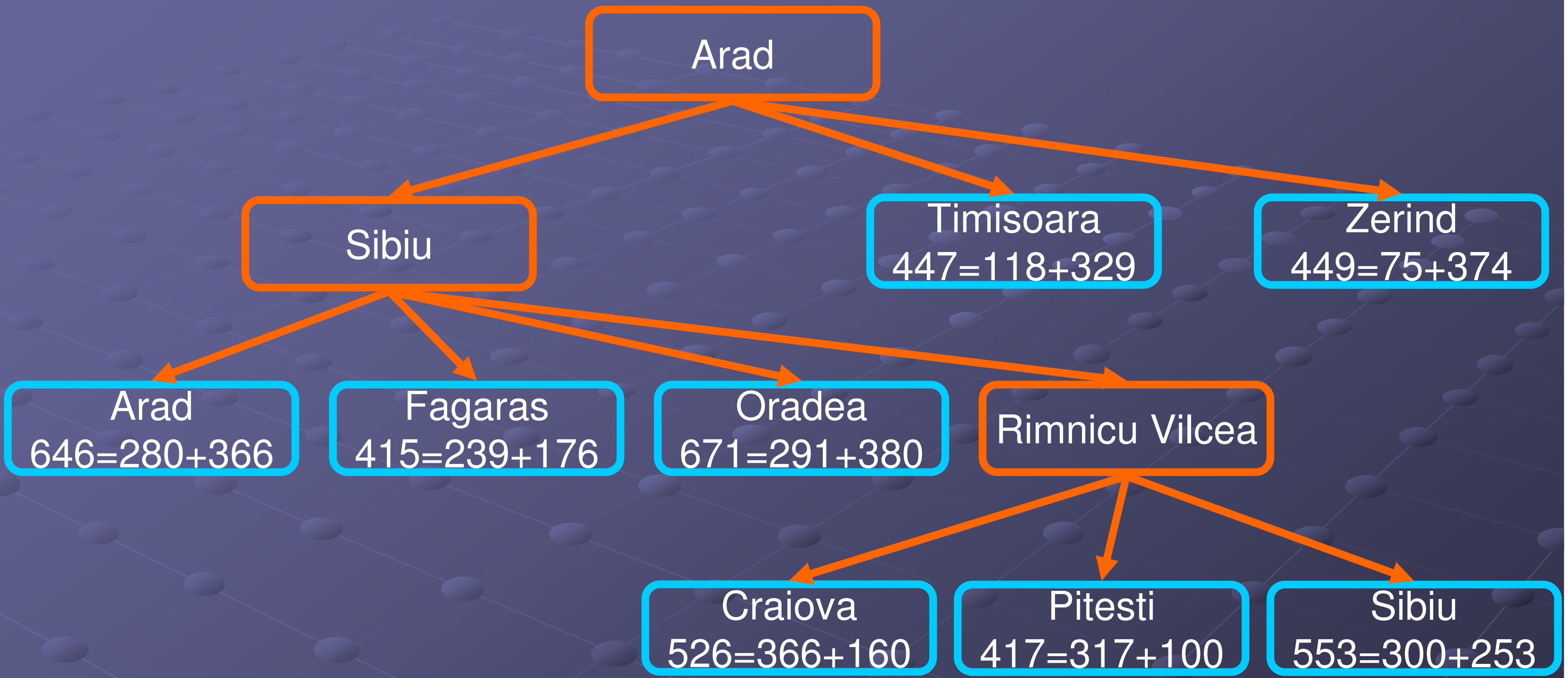
A* Example


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193

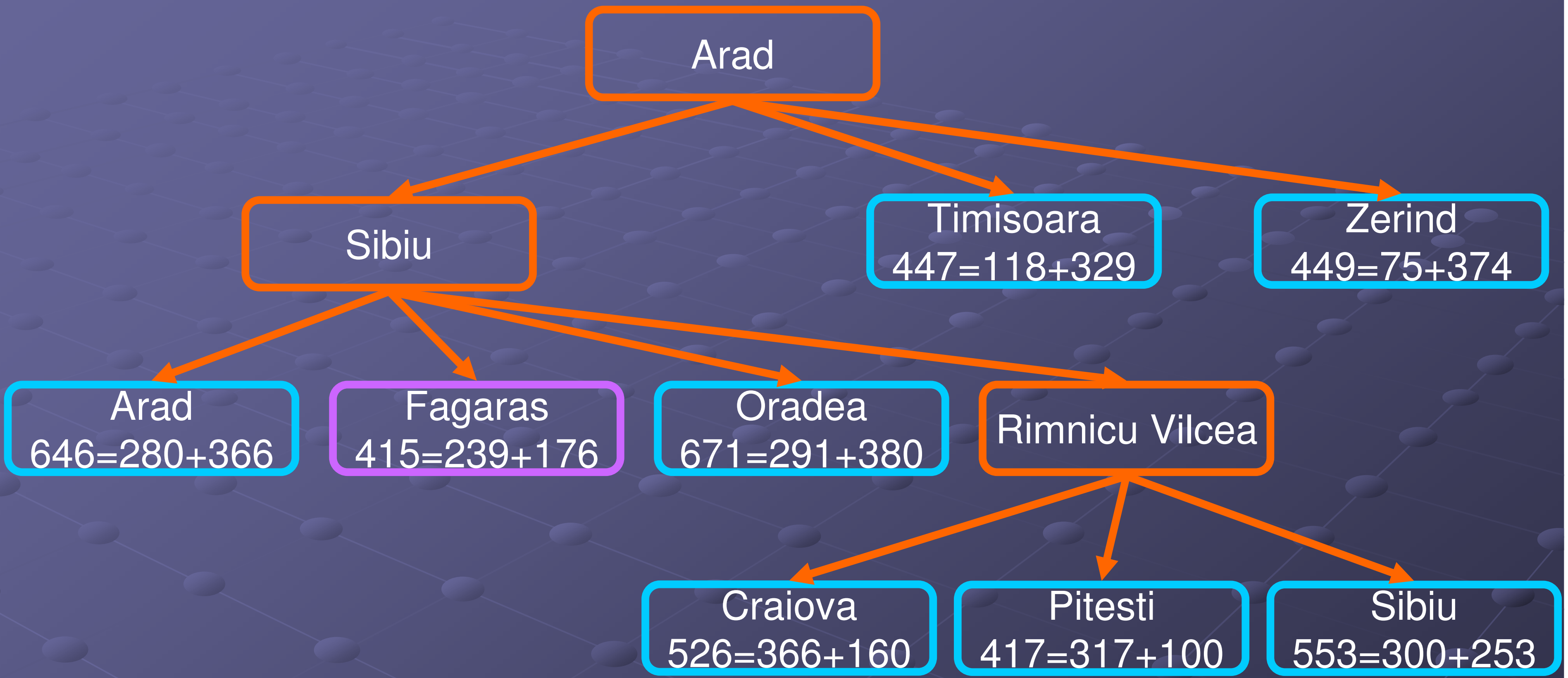
A* Example


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193

A* Example


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

A* Example


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

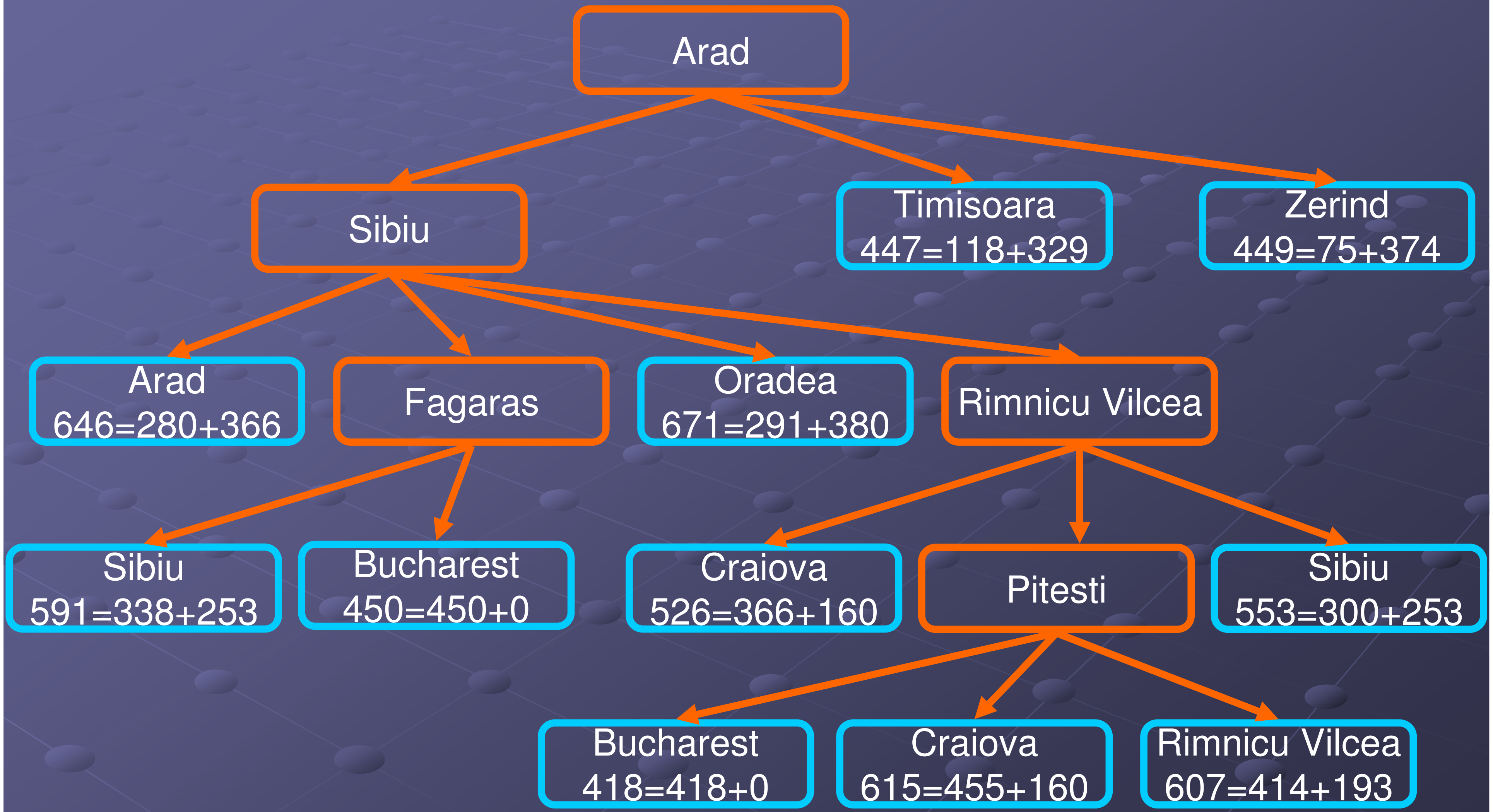
A* Example

Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

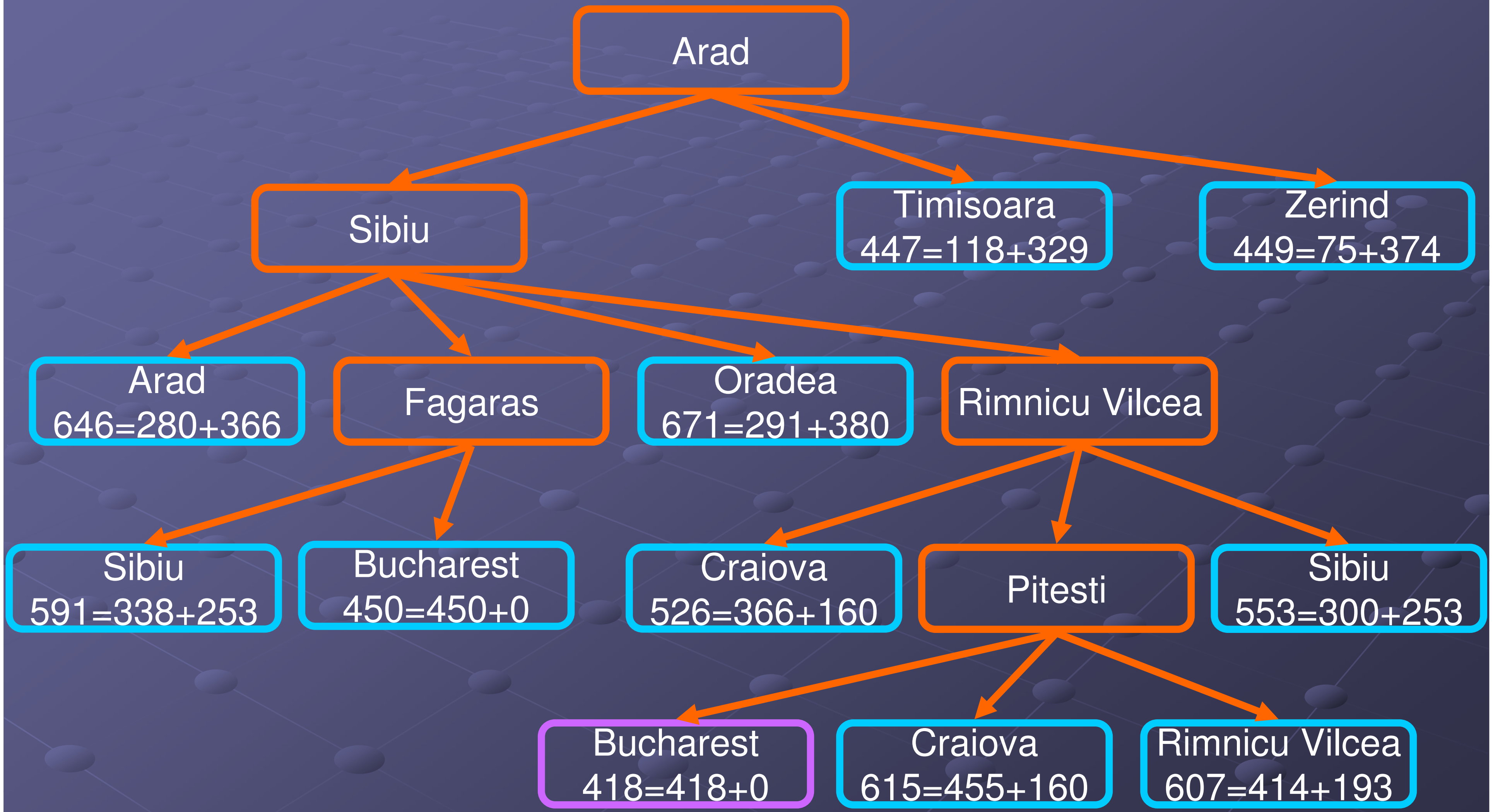
A* Example


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

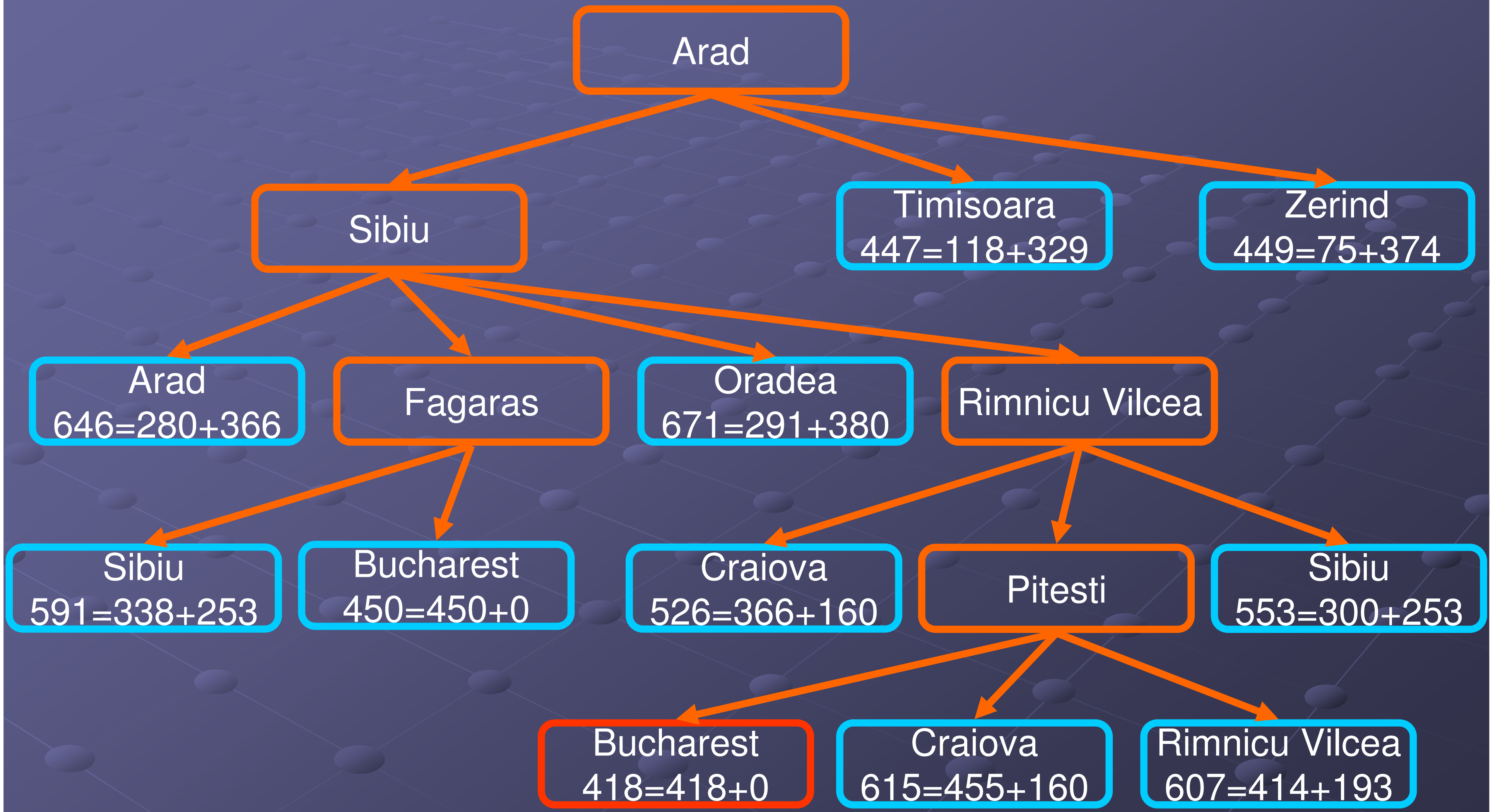
A* Example


Arad
Sibiu
Timisoara 447=118+329
Zerind 449=75+374
Arad 646=280+366
Fagaras
Oradea 671=291+380
Rimnicu Vilcea
Sibiu 591=338+253
Bucharest 450=450+0
Craiova 526=366+160
Pitesti
Sibiu 553=300+253
Bucharest 418=418+0
Craiova 615=455+160
Rimnicu Vilcea 607=414+193

A* Example


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

A* Example


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

A* Search

✓Optimal

✓No other optimal algorithm expands fewer nodes (modulo order of expansion of nodes with same cost)

✗Exponential space requirement

Greedy Search

- Use (always admissible) $h(n) = 0$ in $f(n) = g(n) + h(n)$
- *i.e.* always use the cheapest found so far

Variants of A* Search

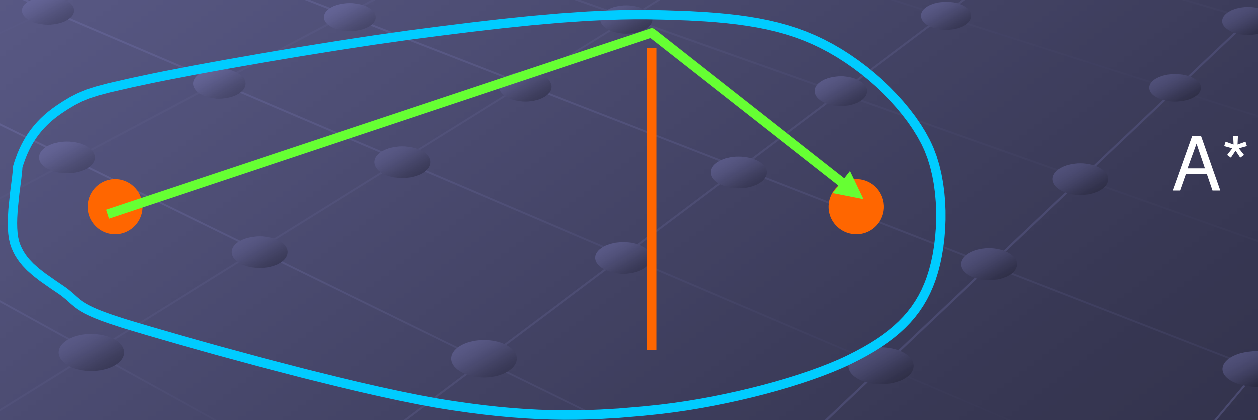
Many variants of A* developed

- Weighted A*: $f(n) = g(n) + W\,h(n)$
  - Not exact
  - Faster
  - Answer is within W of optimal

Variants of A* Search

Many variants of A* developed

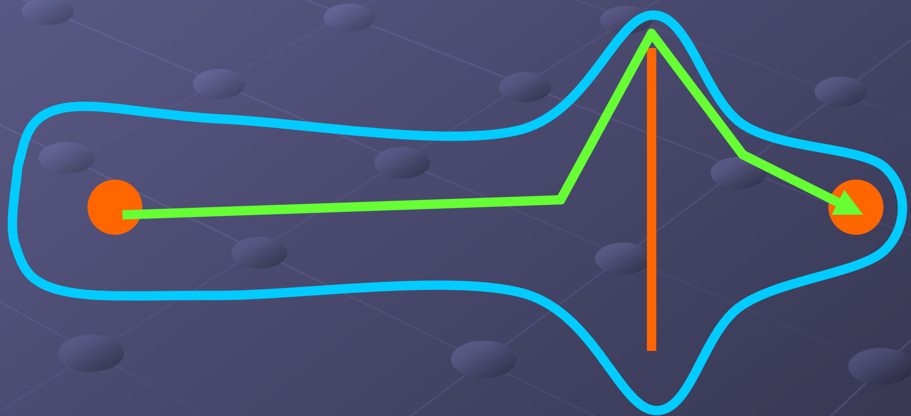
- Weighted A*: $f(n) = g(n) + W\,h(n)$
  - Not exact
  - Faster
  - Answer is within W of optimal



Variants of A* Search

Many variants of A* developed

- Weighted A*: $f(n) = g(n) + W\, h(n)$
  - Not exact
  - Faster
  - Answer is within W of optimal



Weighted A*

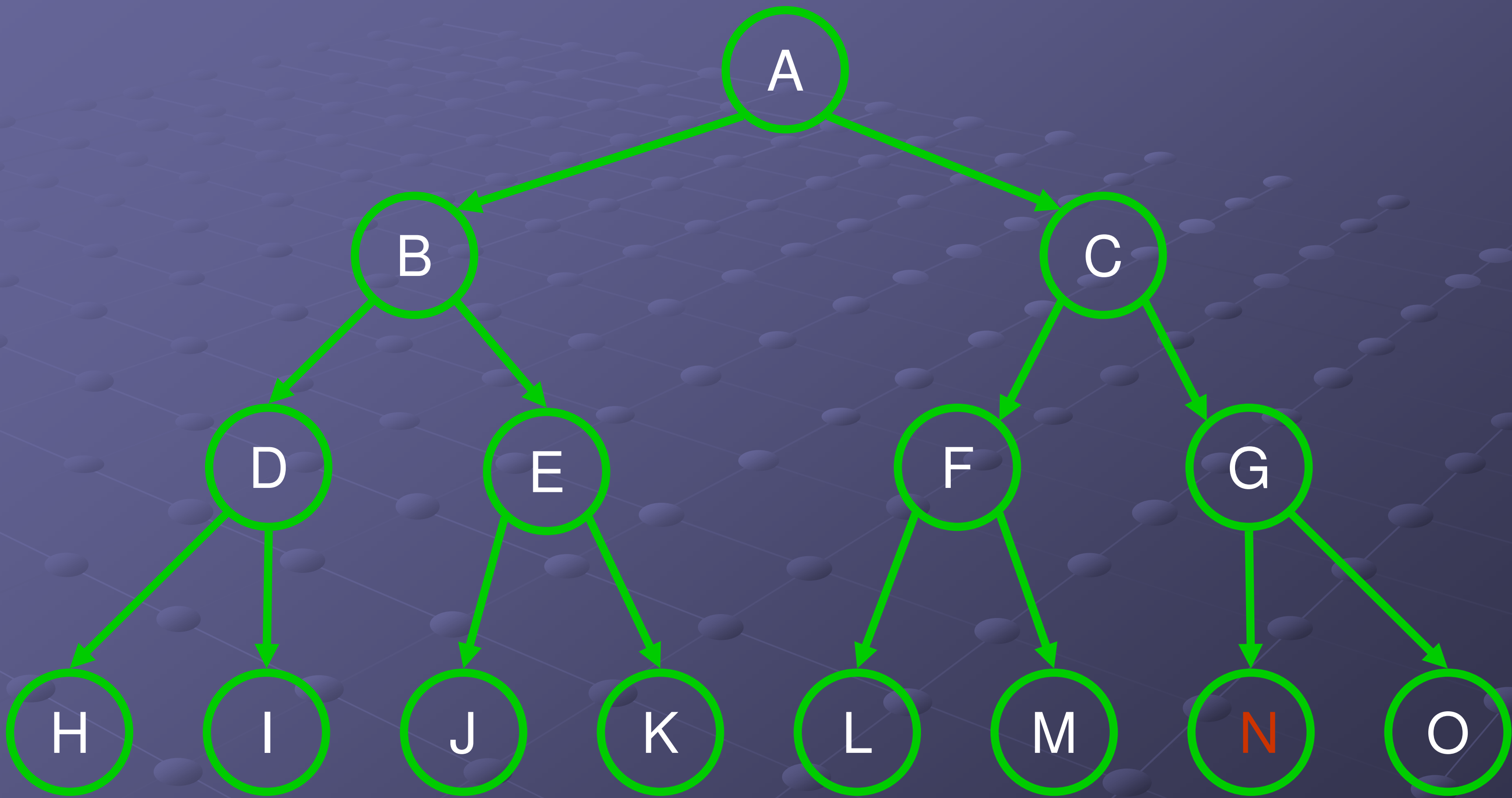
Variants of A* Search

- Iterative Deepening A* (IDA*)
  - Incrementally increase max depth
- Memory-bounded A*
  - Impose bound on memory
  - Discard nodes with largest *f(x)* if necessary
- Others to come in seminars…

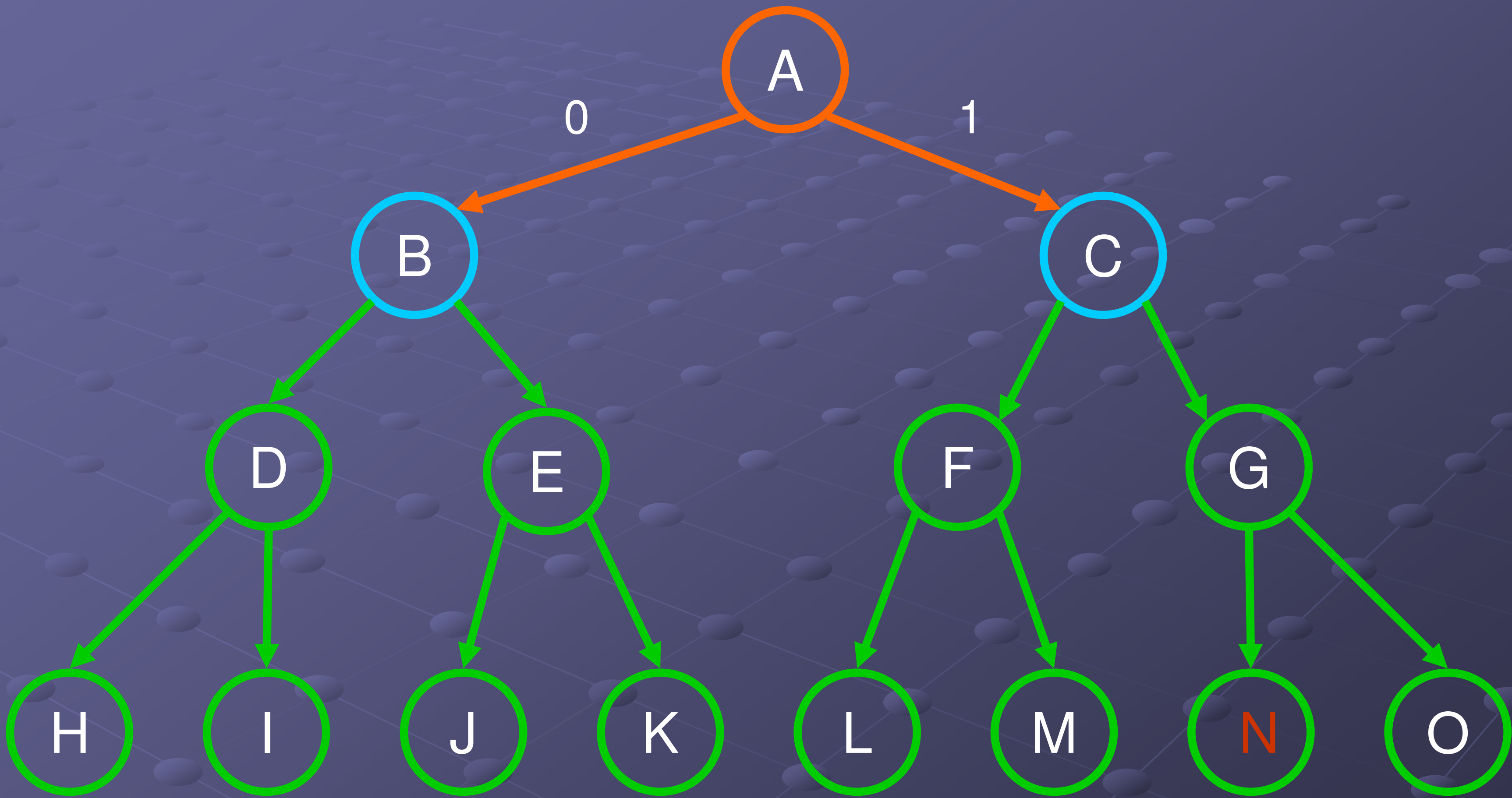
Limited Discrepancy Search

- Requires a heuristic to order choices (e.g. admissible heuristic, but don't require underestimate)
- If heuristic was right, we could always take the choice given by the heuristic
- LDS allows the heuristic to make mistakes: *k*-LDS allows *k* mistakes.

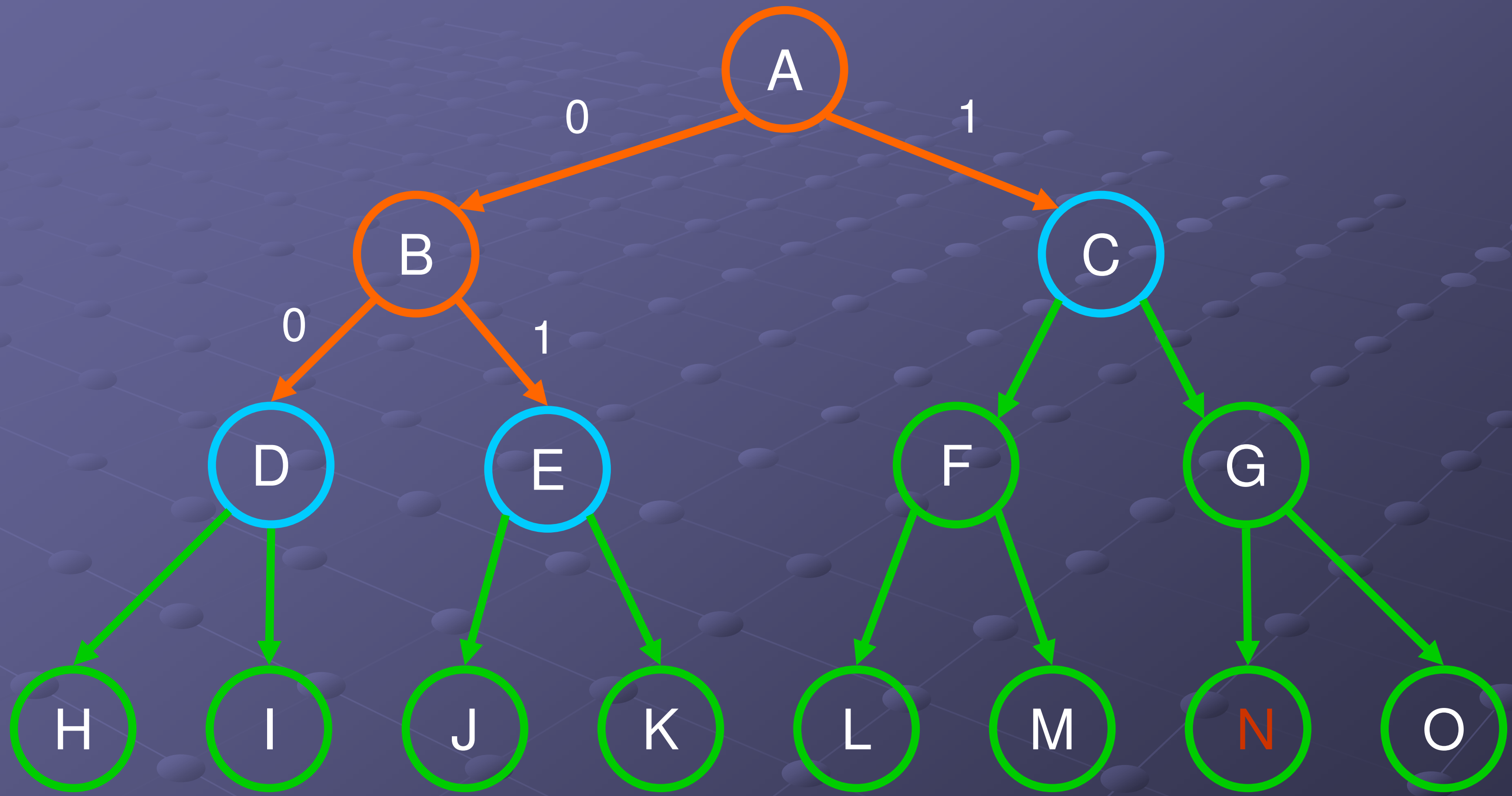
Limited Discrepancy Search (*k* = 1)



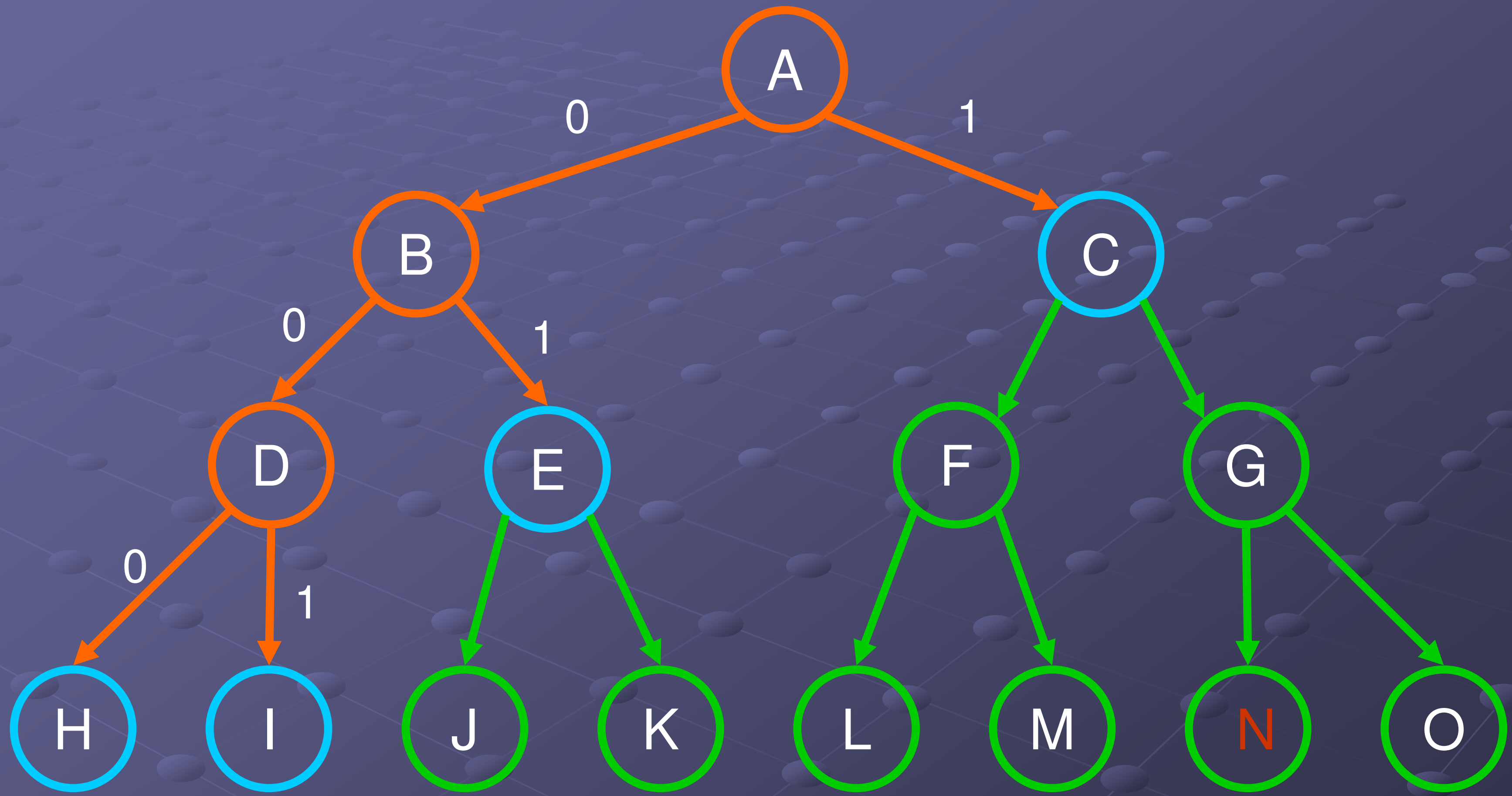
Limited Discrepancy Search ($k = 1$)



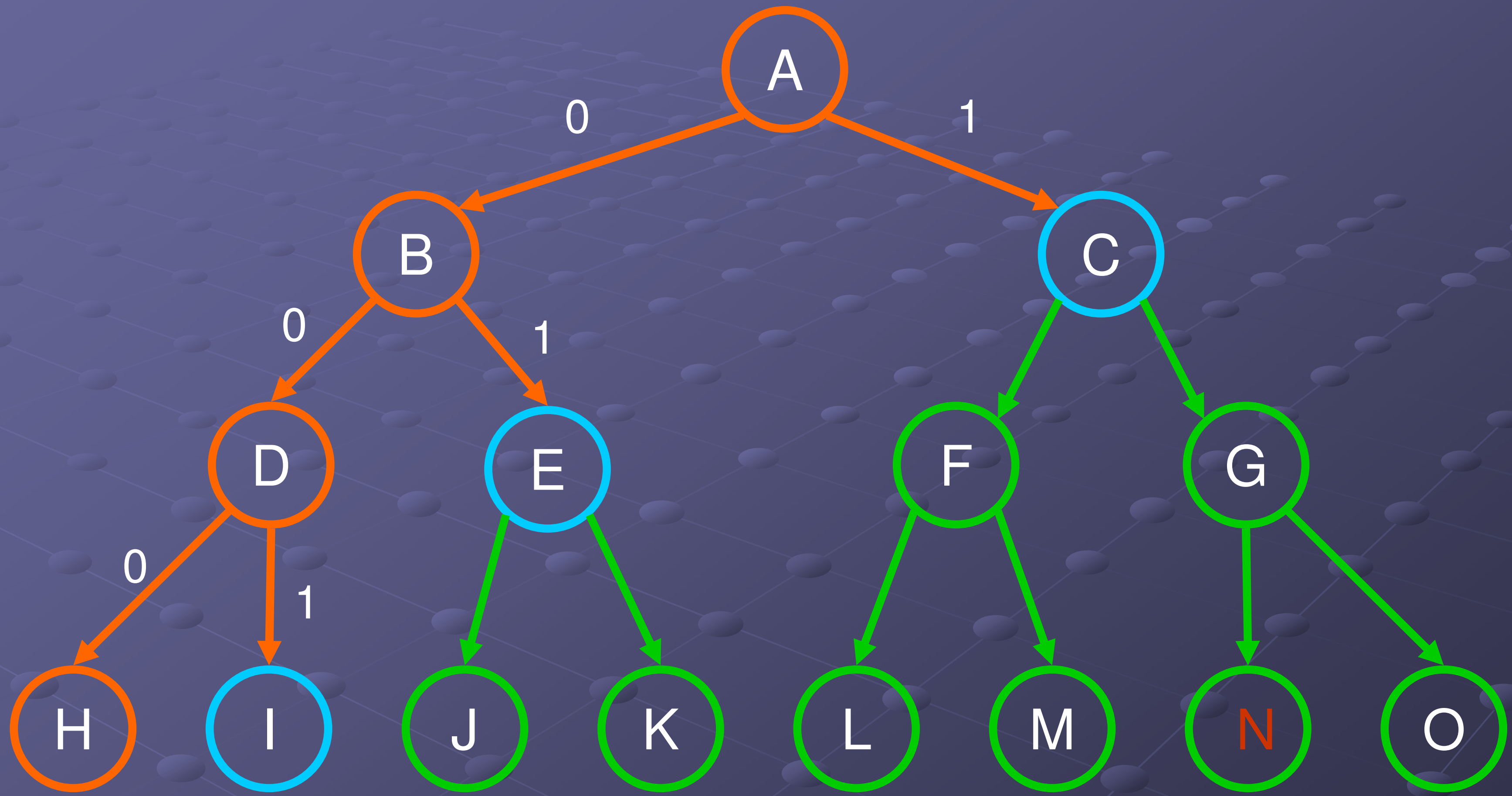
Limited Discrepancy Search ($k = 1$)



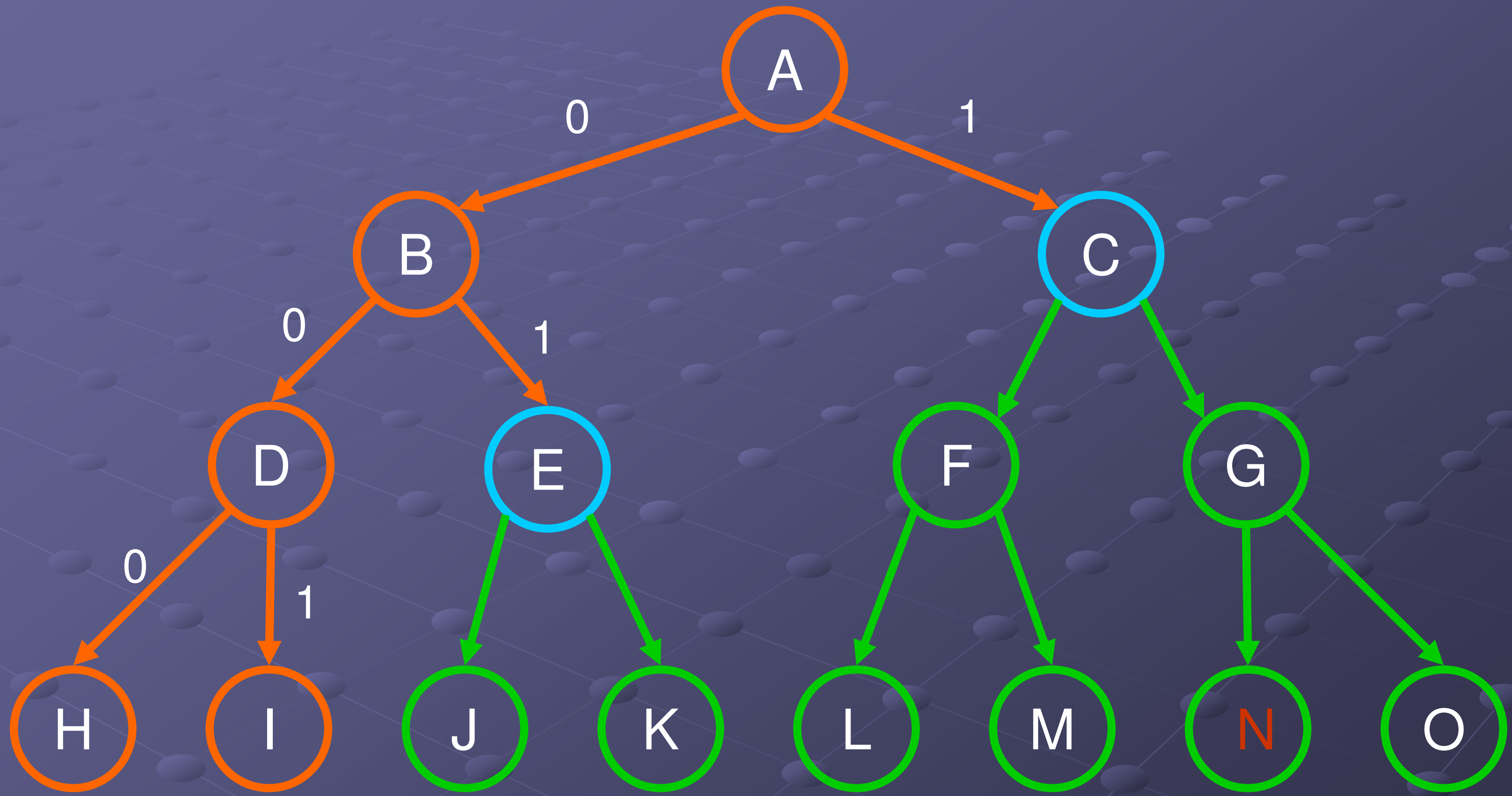
Limited Discrepancy Search (k = 1)



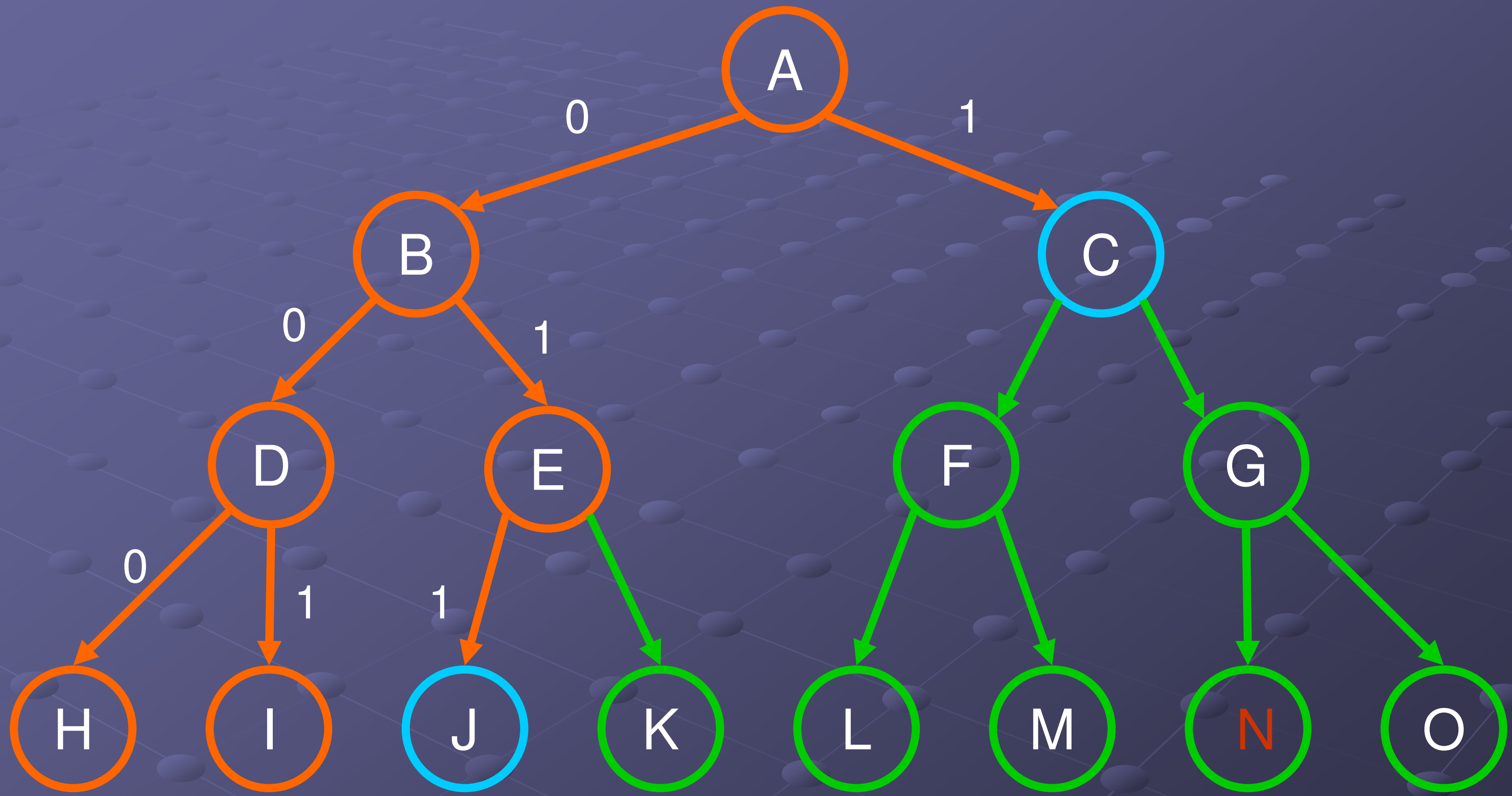
Limited Discrepancy Search (k = 1)



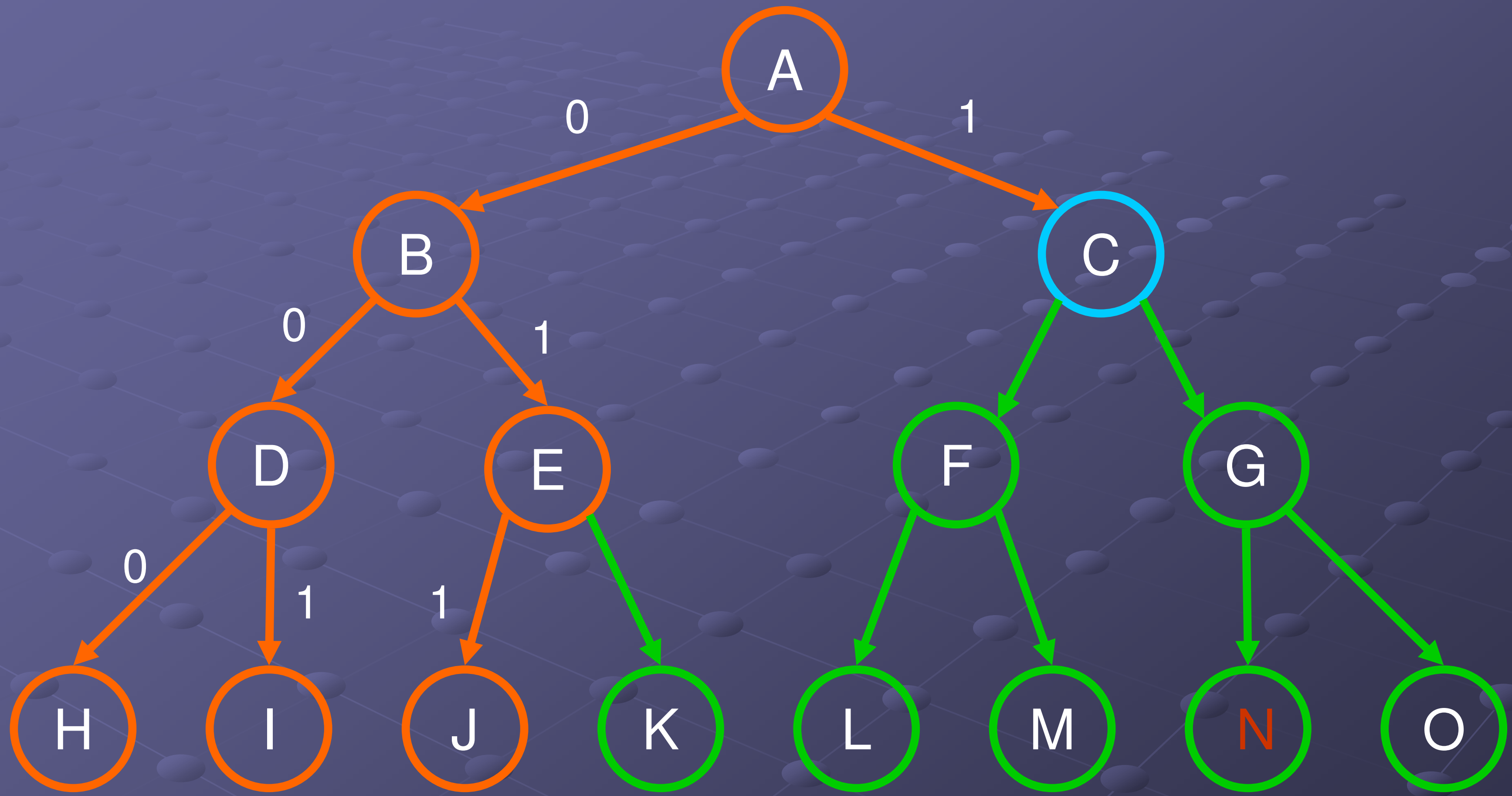
Limited Discrepancy Search (k = 1)



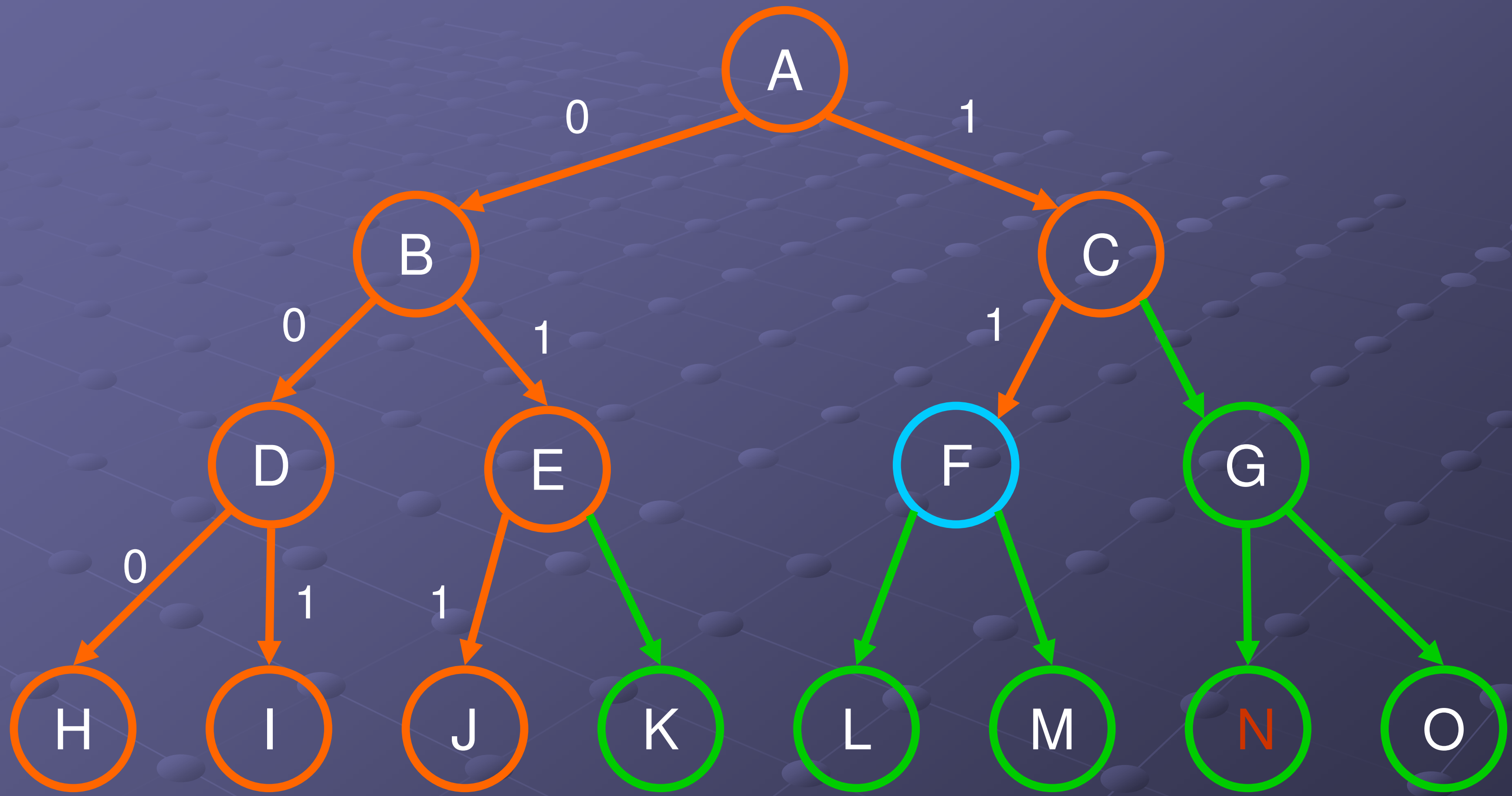
Limited Discrepancy Search (k = 1)



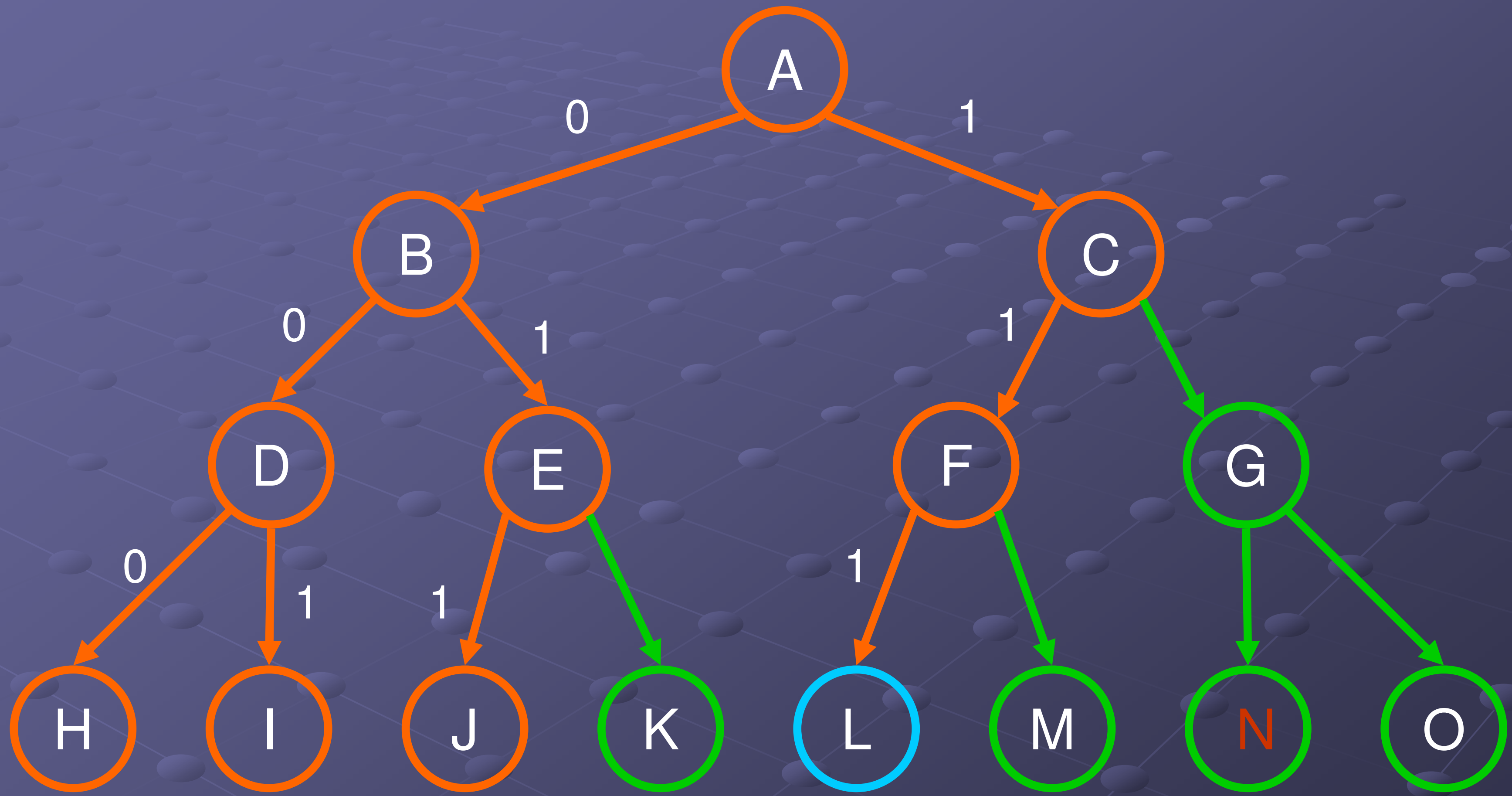
Limited Discrepancy Search (*k* = 1)



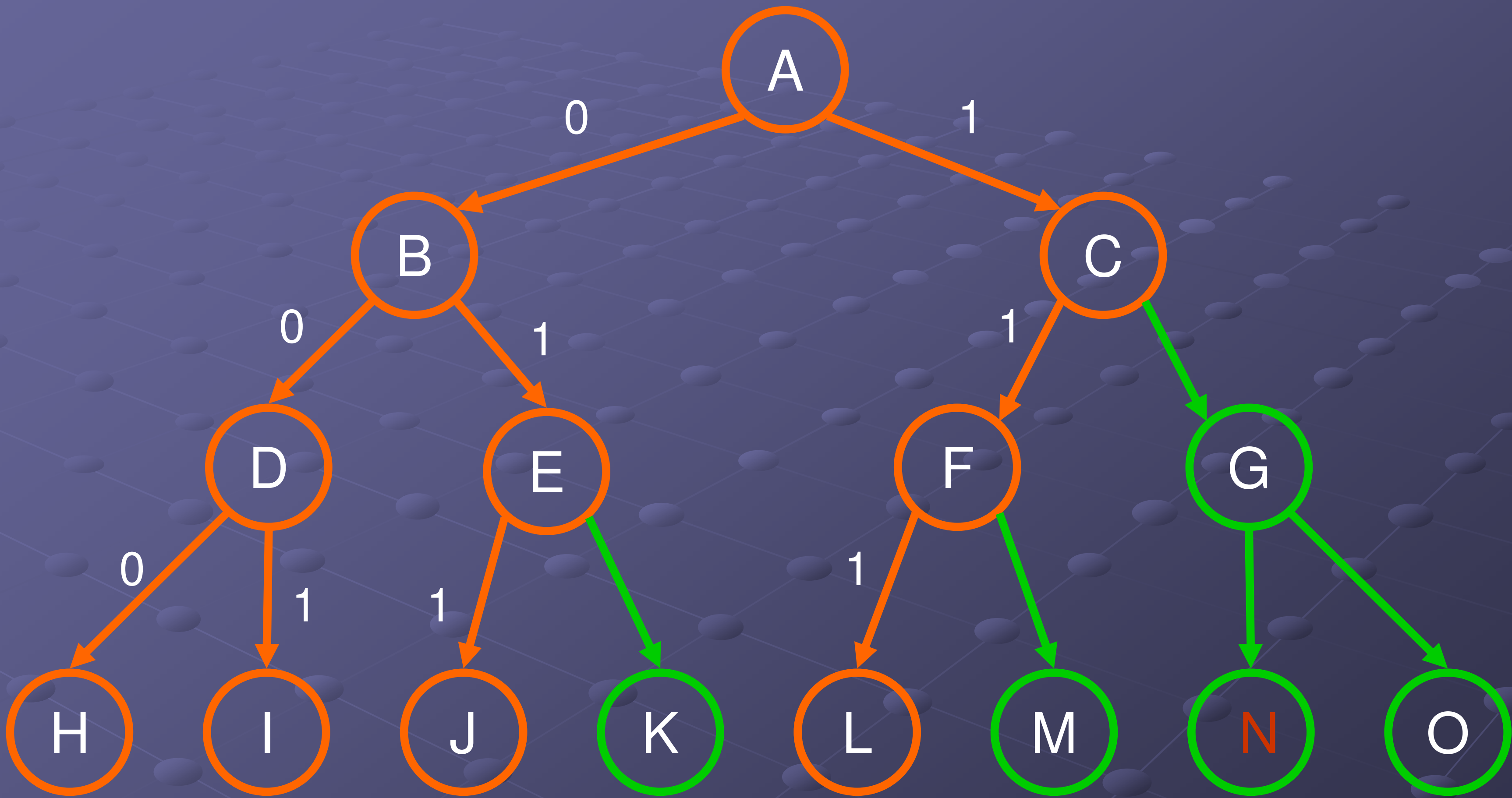
Limited Discrepancy Search (k = 1)



Limited Discrepancy Search ($k = 1$)



Limited Discrepancy Search (k = 1)



Next week….

(Integer) Linear Programming and

Branch & Bound