ENCS3340 - Artificial Intelligence

Uninformed Search

Uninformed Search Strategies

- Uninformed search strategies use only the information available in the problem definition

- Examples:
  - Breadth-first search
  - Depth-first search
  - Iterative deepening search
  - Uniform-cost search

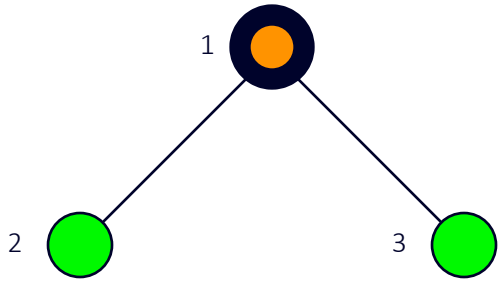
Breadth-First Search

- all the nodes reachable from the current node are explored first
  - Expand shallowest unexpanded node
  - achieved by the TREE-SEARCH method by appending newly generated nodes at the end of the search queue

```
function GENERAL-SEARCH(problem) returns solution

        return TREE-SEARCH(problem, FIFO-QUEUE())
```

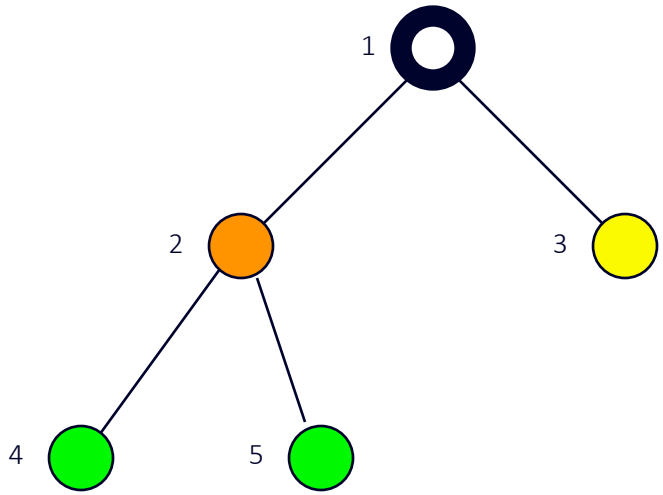
- Implementation: fringe is a FIFO queue, i.e., new successors go at end

Breadth-First Snapshot 1



Fringe: [] + [2,3]

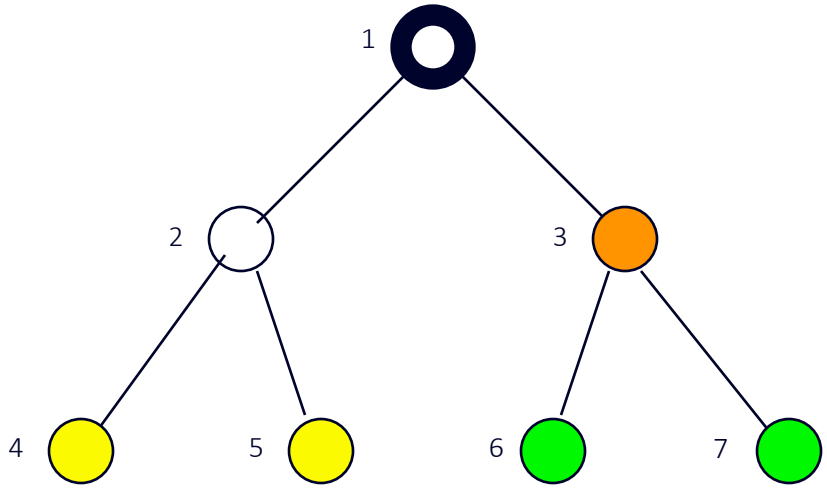
Breadth-First Snapshot 2



Initial
Visited
Fringe
Current
Visible
Goal

Fringe: [3] + [4,5]

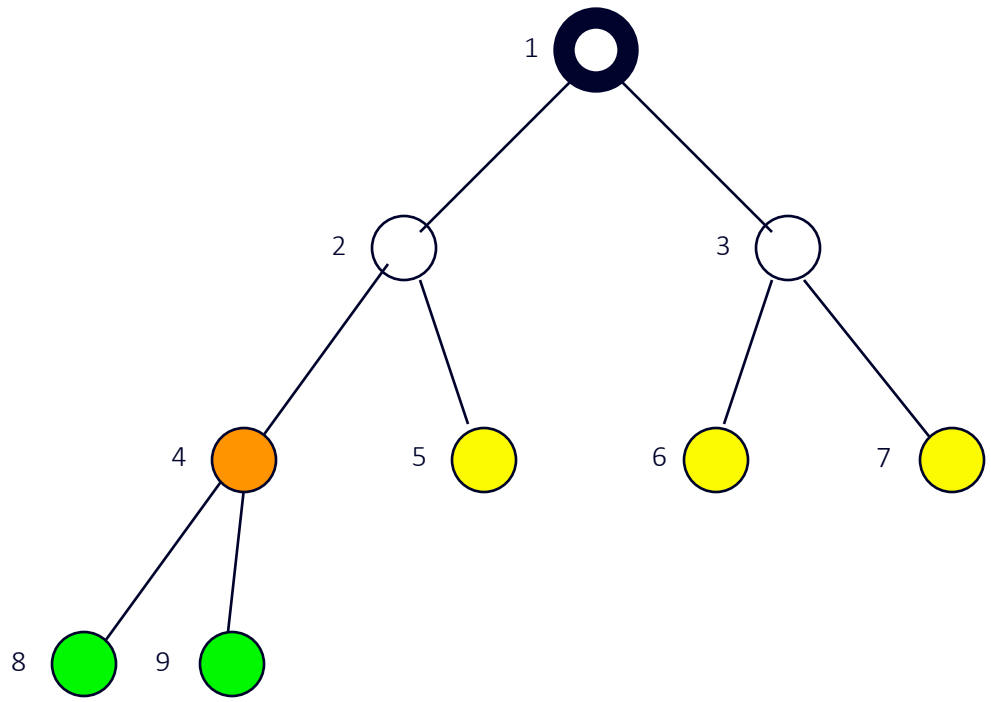
Breadth-First Snapshot 3



Initial
Visited
Fringe
Current
Visible
Goal

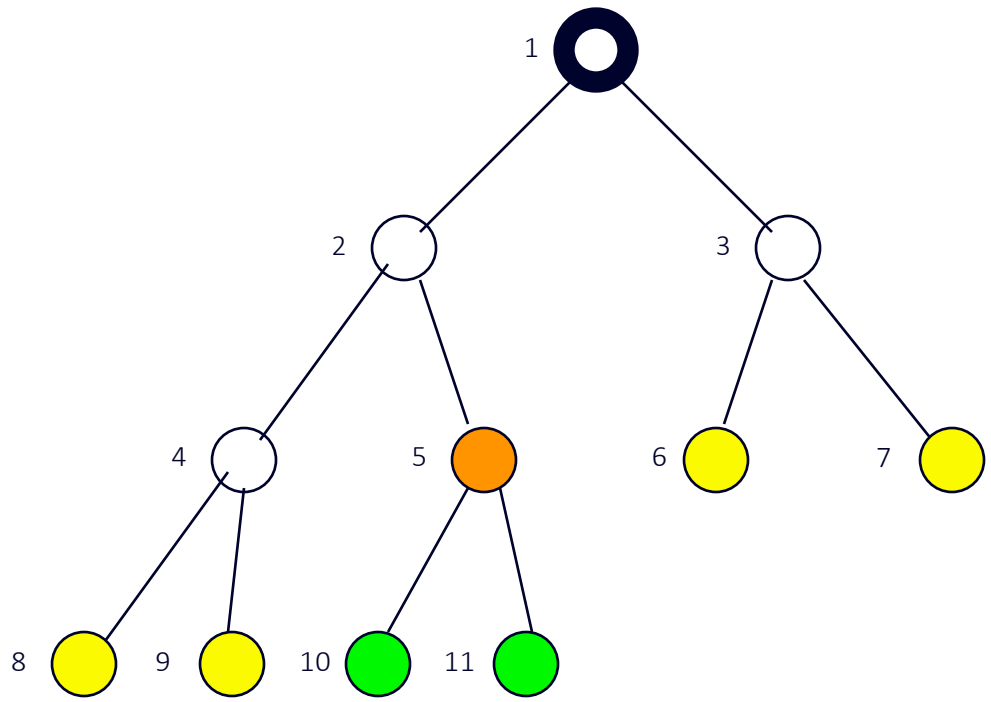
Fringe: [4,5] + [6,7]

Breadth-First Snapshot 4



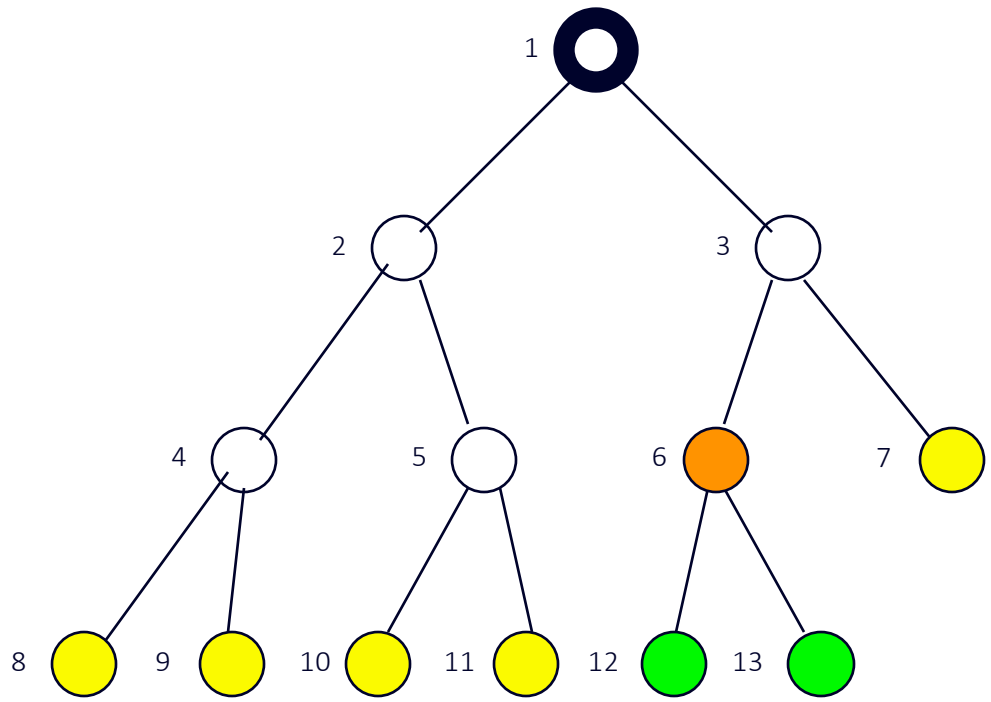
Fringe: [5,6,7] + [8,9]

Breadth-First Snapshot 5



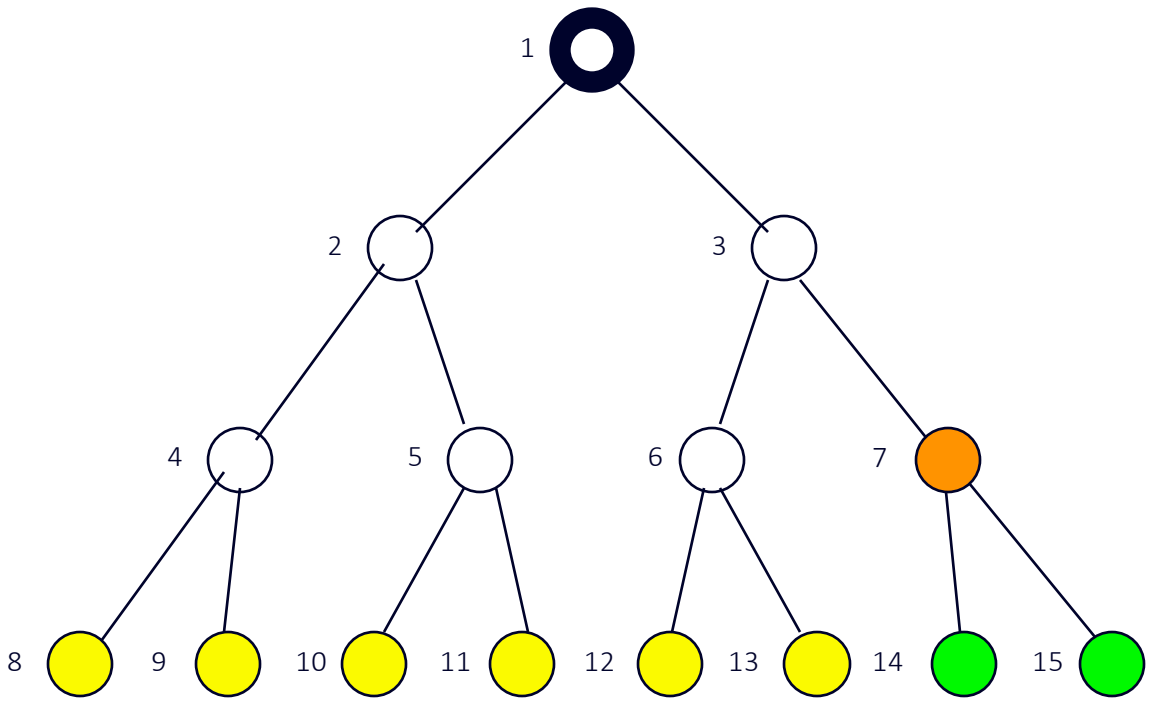
Fringe: [6,7,8,9] + [10,11]

Breadth-First Snapshot 6



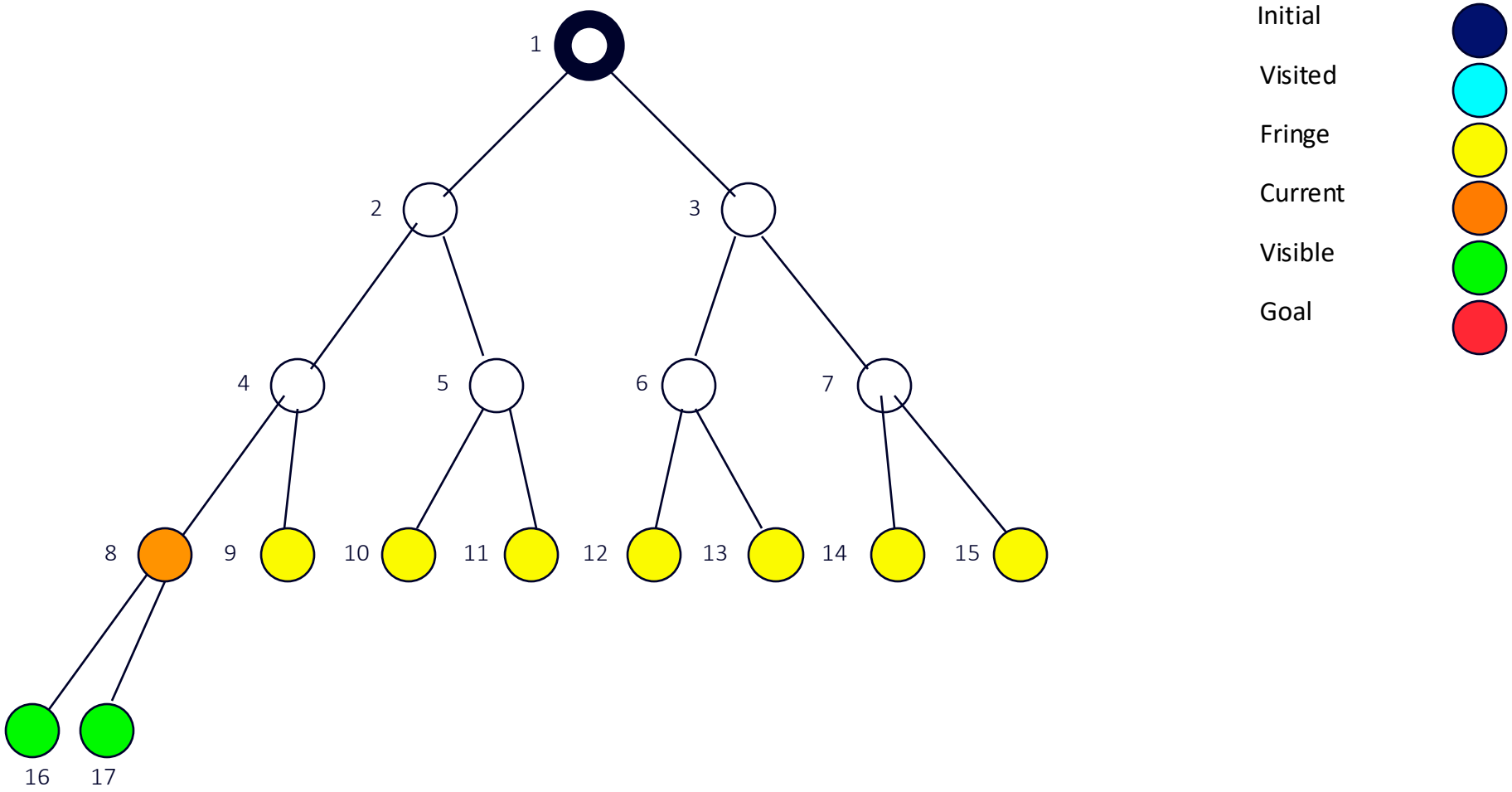
Fringe: [7,8,9,10,11] + [12,13]

Breadth-First Snapshot 7



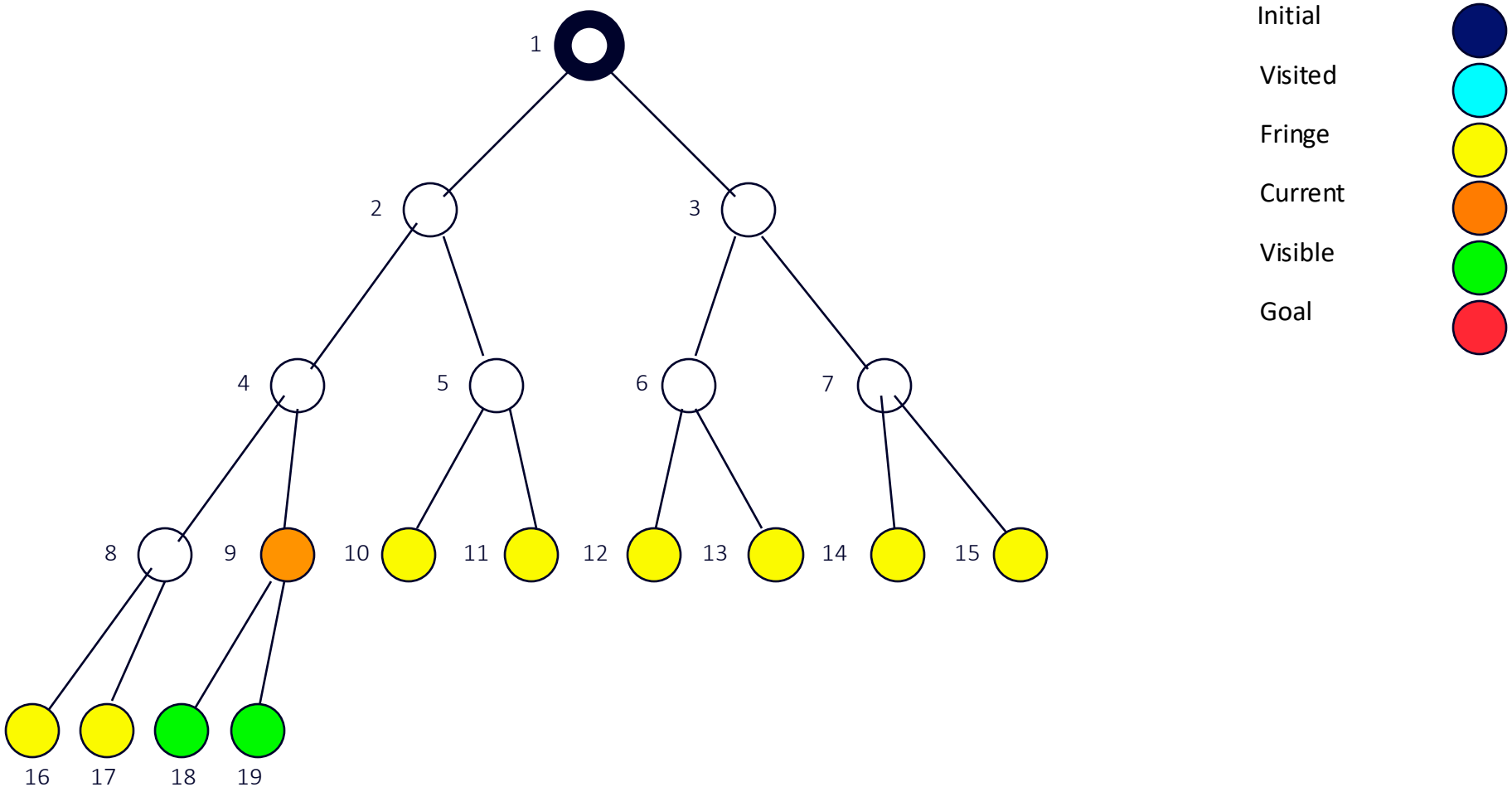
Fringe: [8,9.10,11,12,13] + [14,15]

Breadth-First Snapshot 8



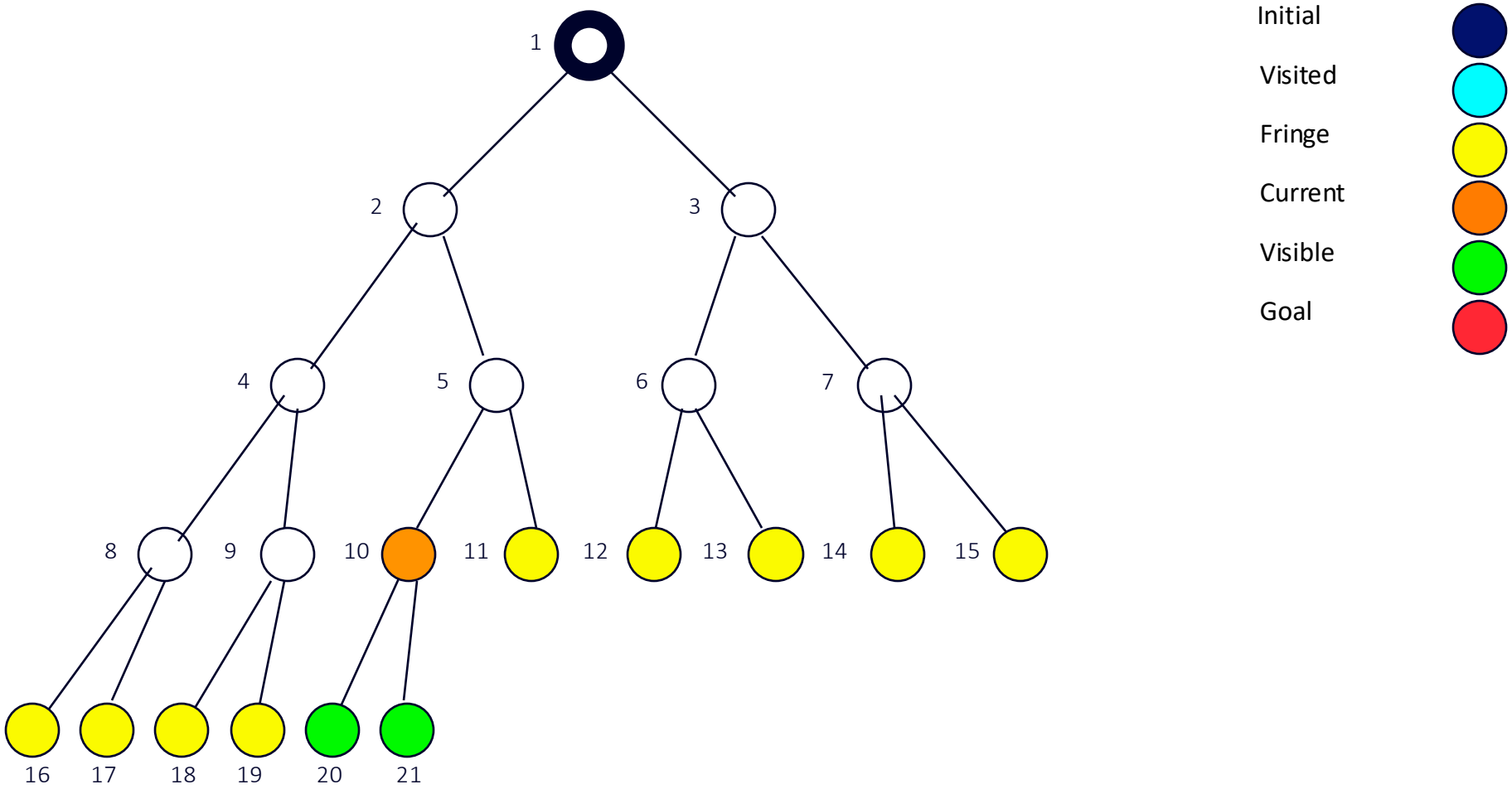
Fringe: [9,10,11,12,13,14,15] + [16,17]

Breadth-First Snapshot 9



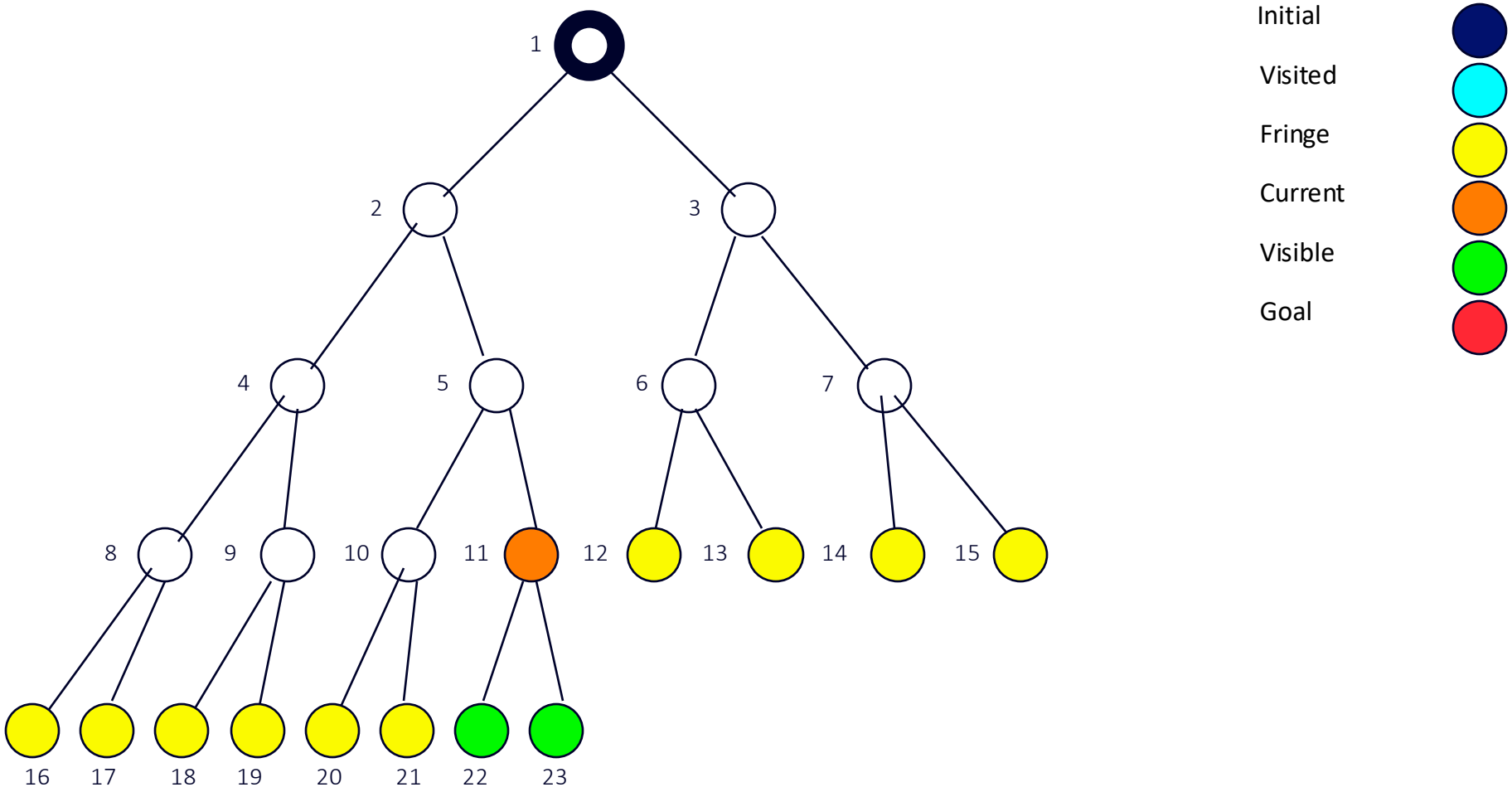
Fringe: [10,11,12,13,14,15,16,17] + [18,19]

Breadth-First Snapshot 10



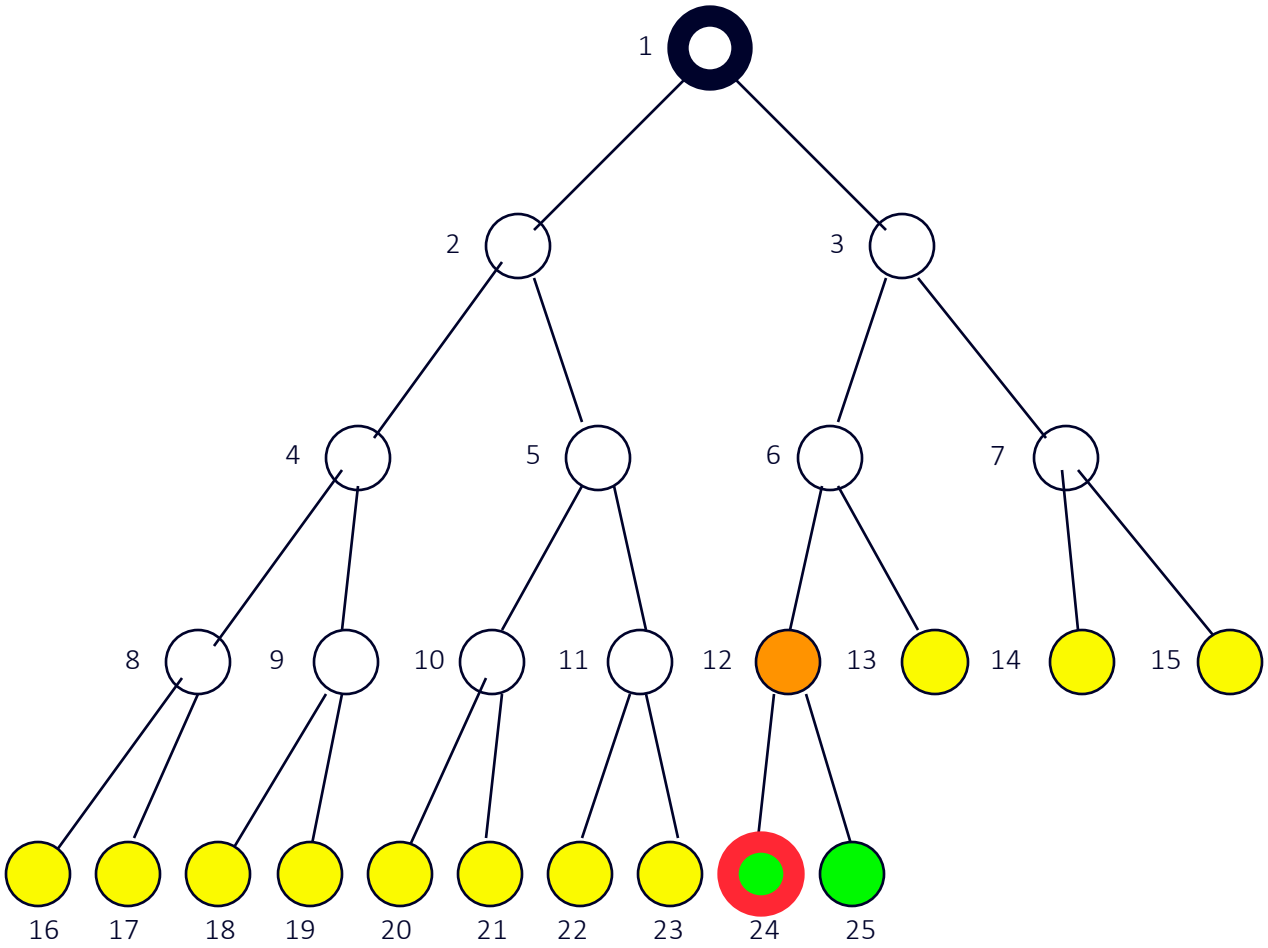
Fringe: [11,12,13,14,15,16,17,18,19] + [20,21]

Breadth-First Snapshot 11



Fringe: [12, 13, 14, 15, 16, 17, 18, 19, 20, 21] + [22,23]

Breadth-First Snapshot 12



Initial

Visited

Fringe

Current

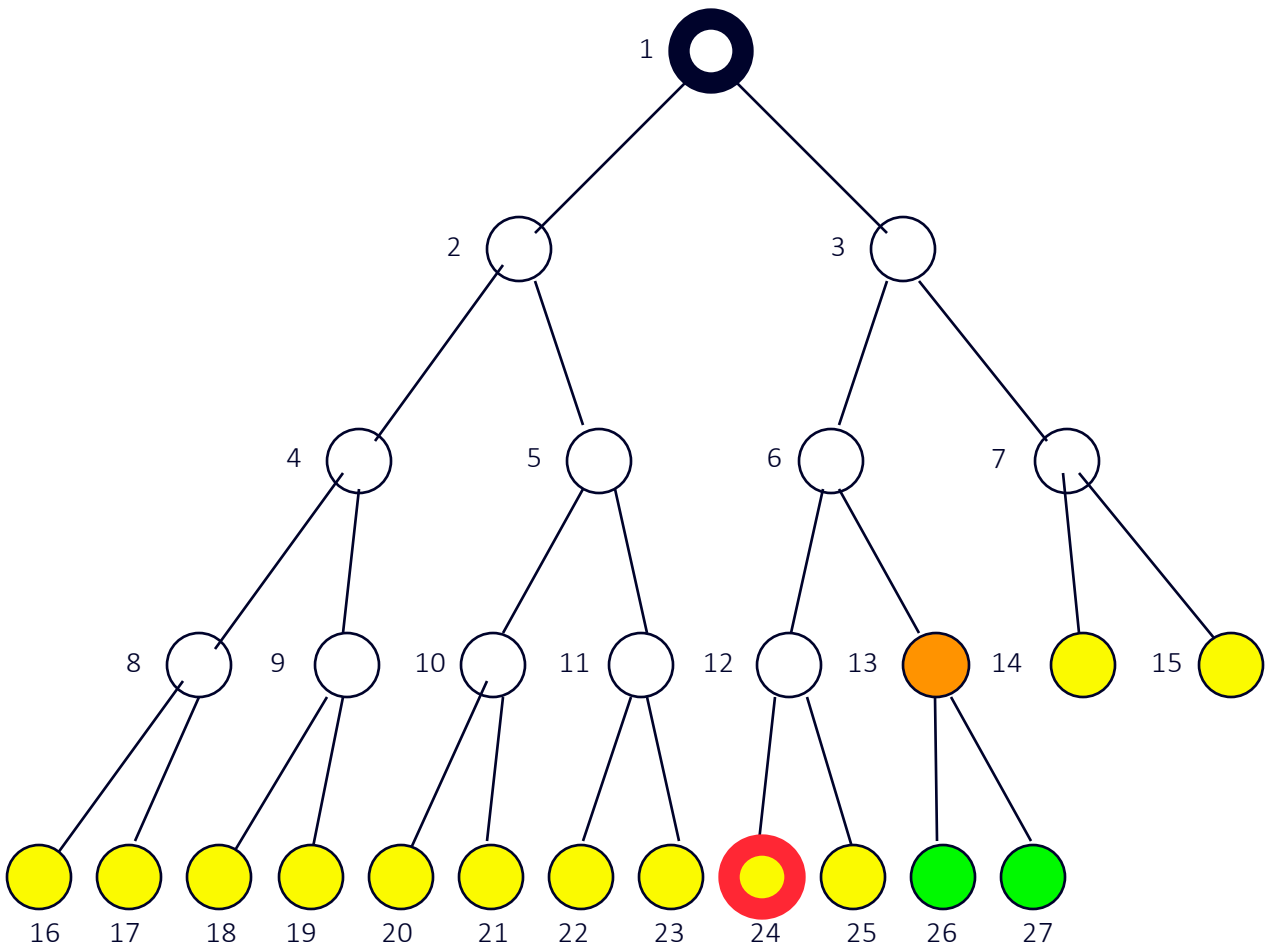
Visible

Goal

Note:
The goal node is "visible" here, but we can not perform the goal test yet.

Fringe: [13,14,15,16,17,18,19,20,21, 22, 23] + [24,25]

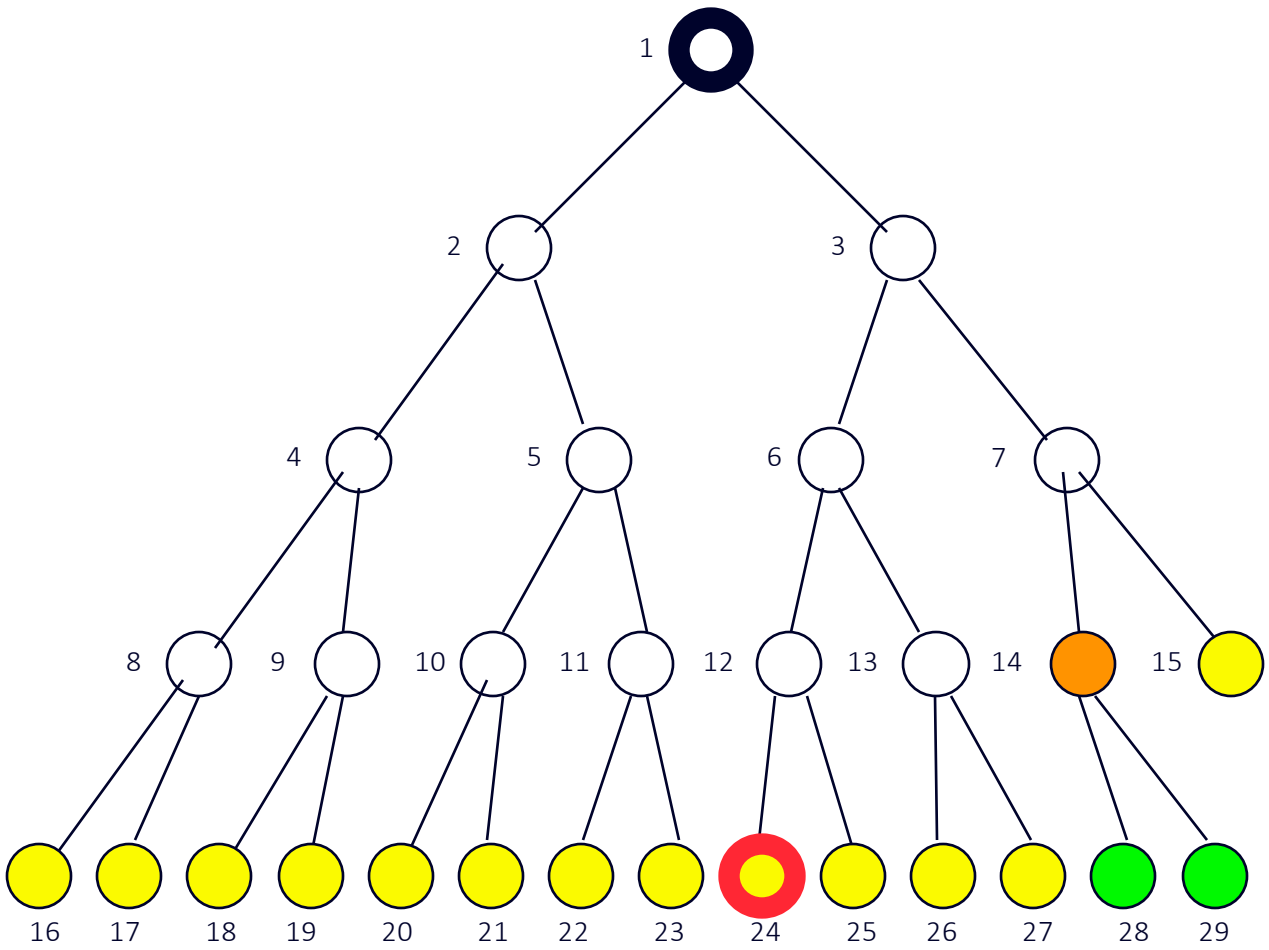
Breadth-First Snapshot 13



Initial
Visited
Fringe
Current
Visible
Goal

Fringe: [14,15,16,17,18,19,20,21,22,23,24,25] + [26,27]

Breadth-First Snapshot 14



Initial

Visited

Fringe

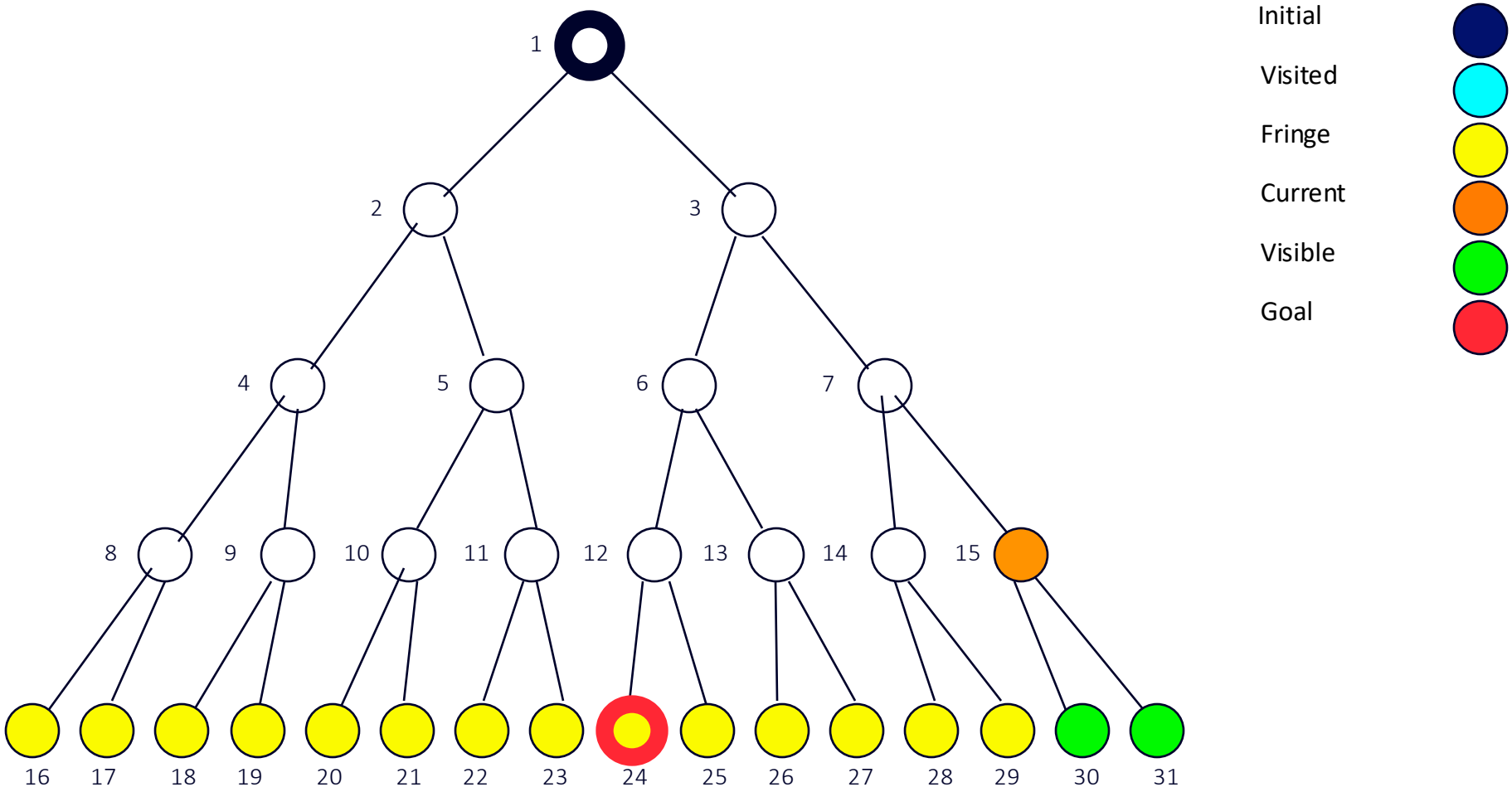
Current

Visible

Goal

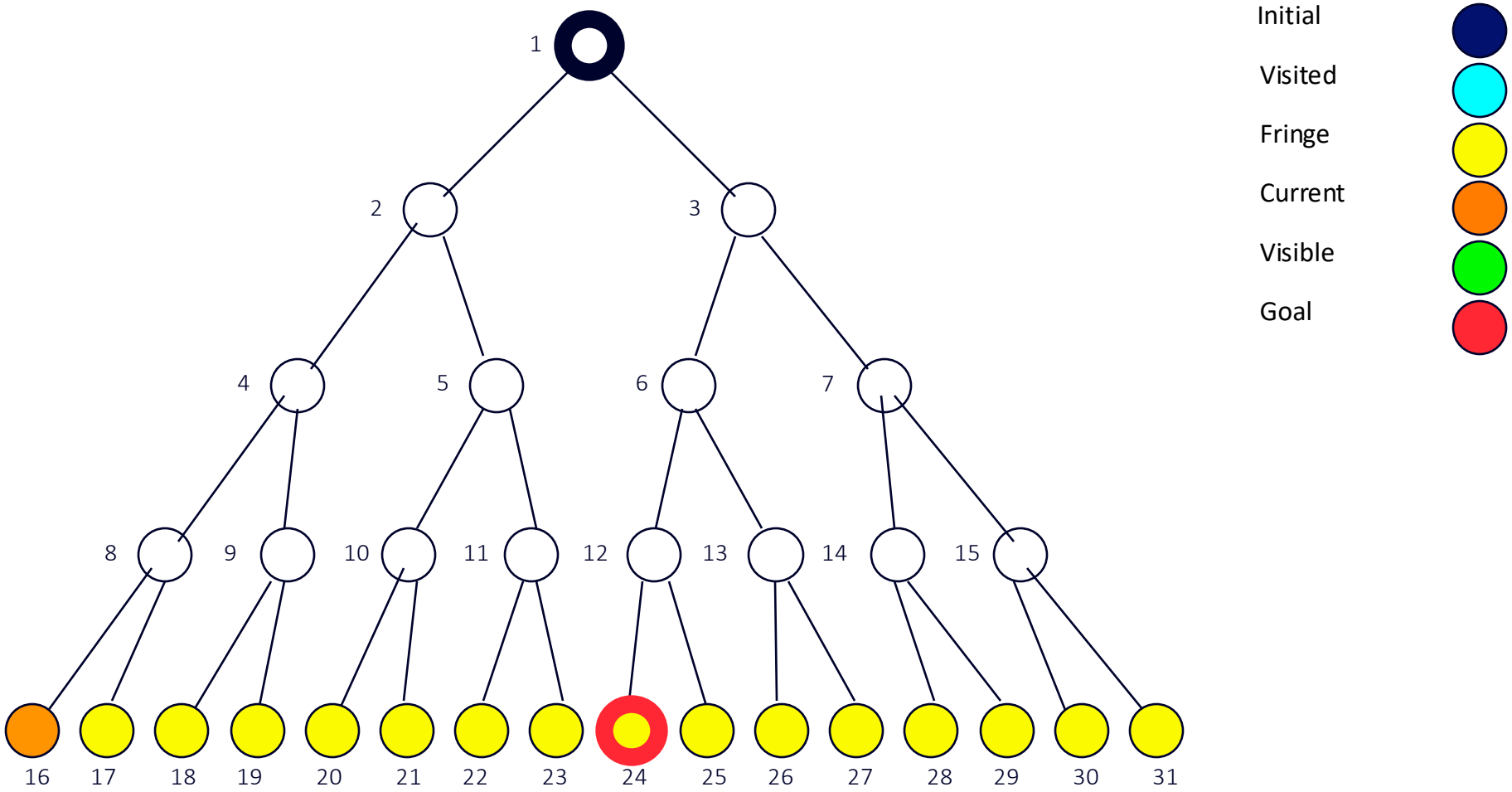
Fringe: [15,16,17,18,19,20,21,22,23,24,25,26,27] + [28,29]

Breadth-First Snapshot 15



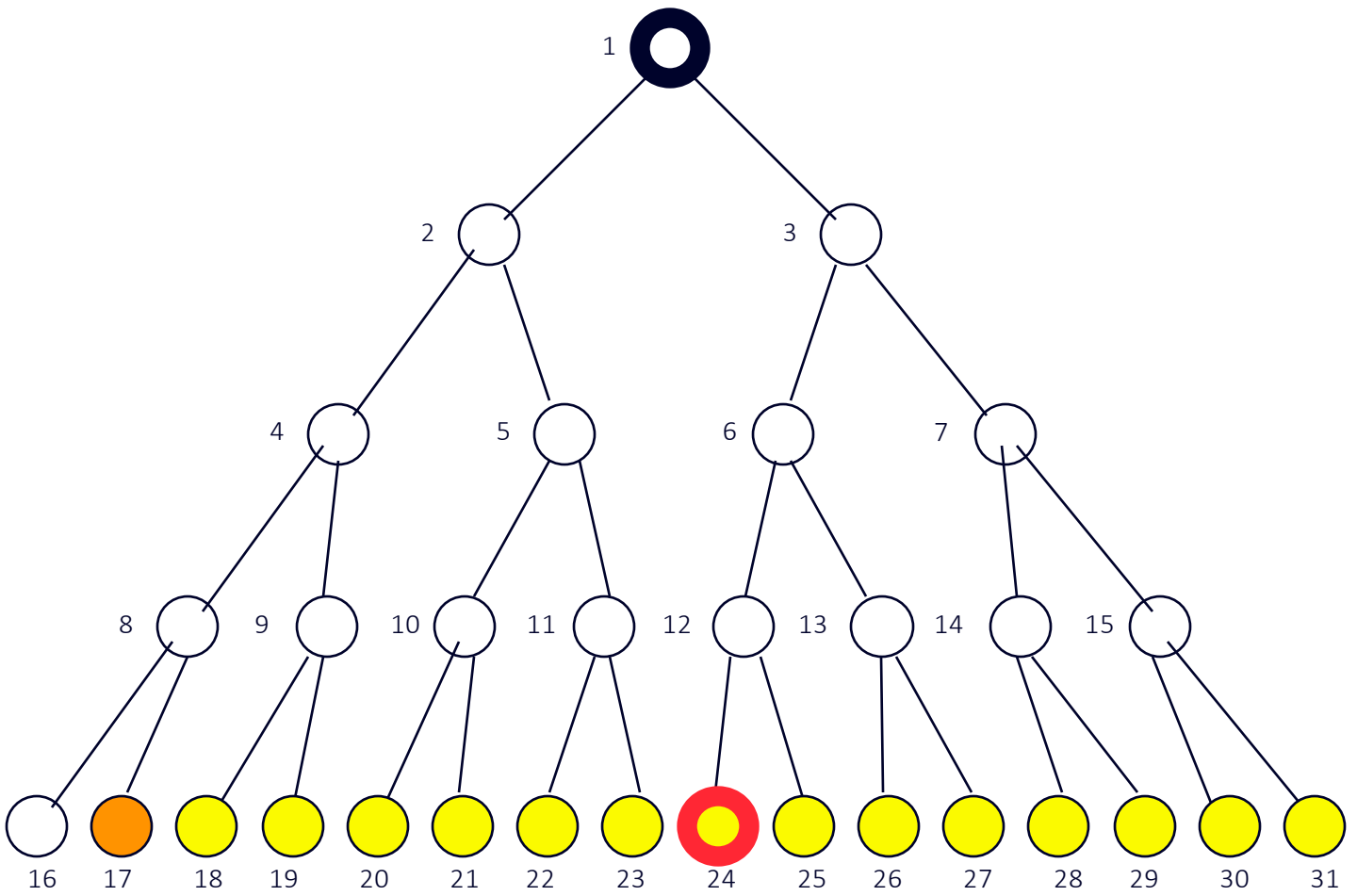
Fringe: [16,17,18,19,20,21,22,23,24,25,26,27,28,29] + [30,31]

Breadth-First Snapshot 16



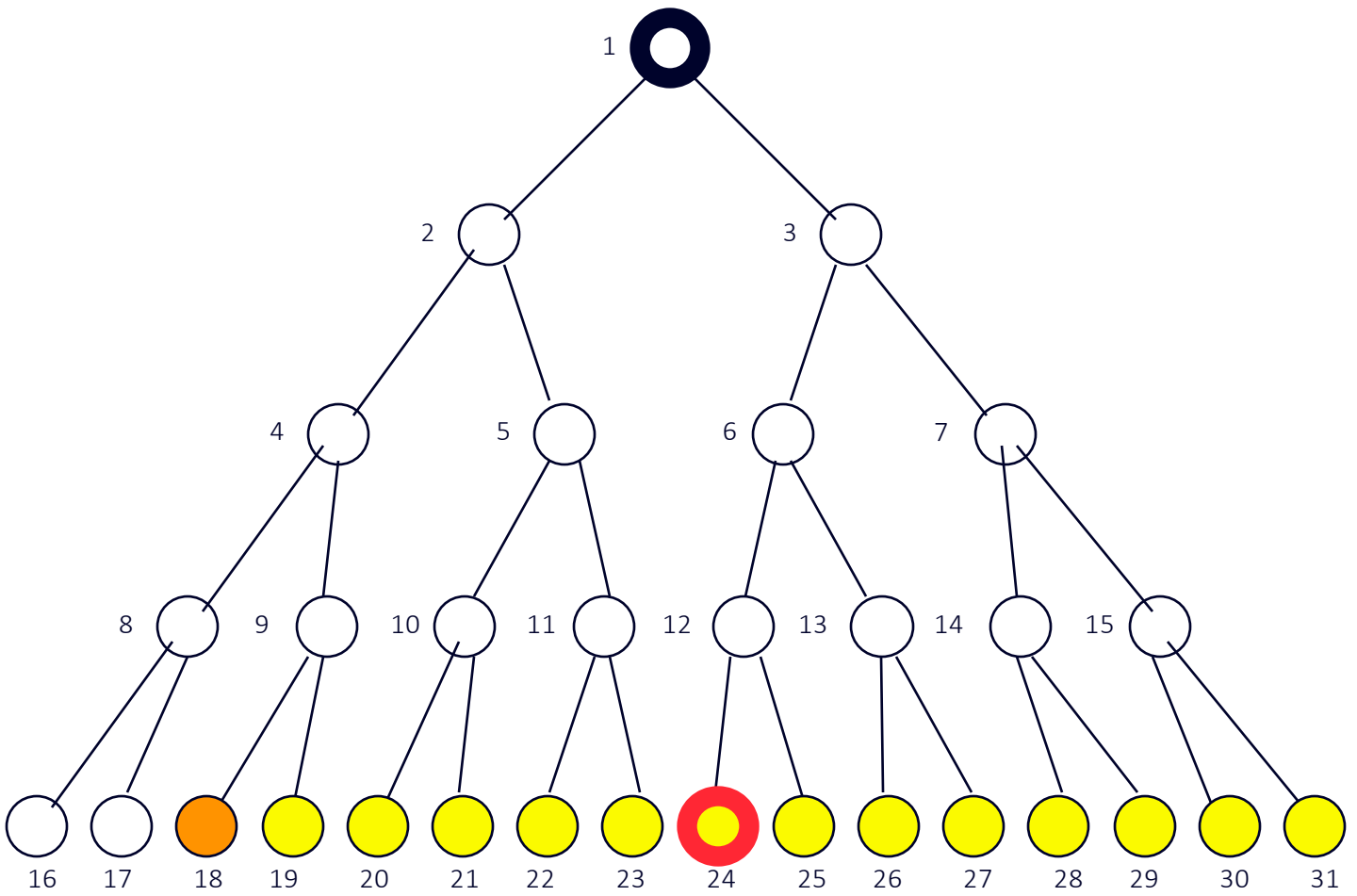
Fringe: [17,18,19,20,21,22,23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 17



Fringe: [18,19,20,21,22,23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 18



Initial

Visited

Fringe

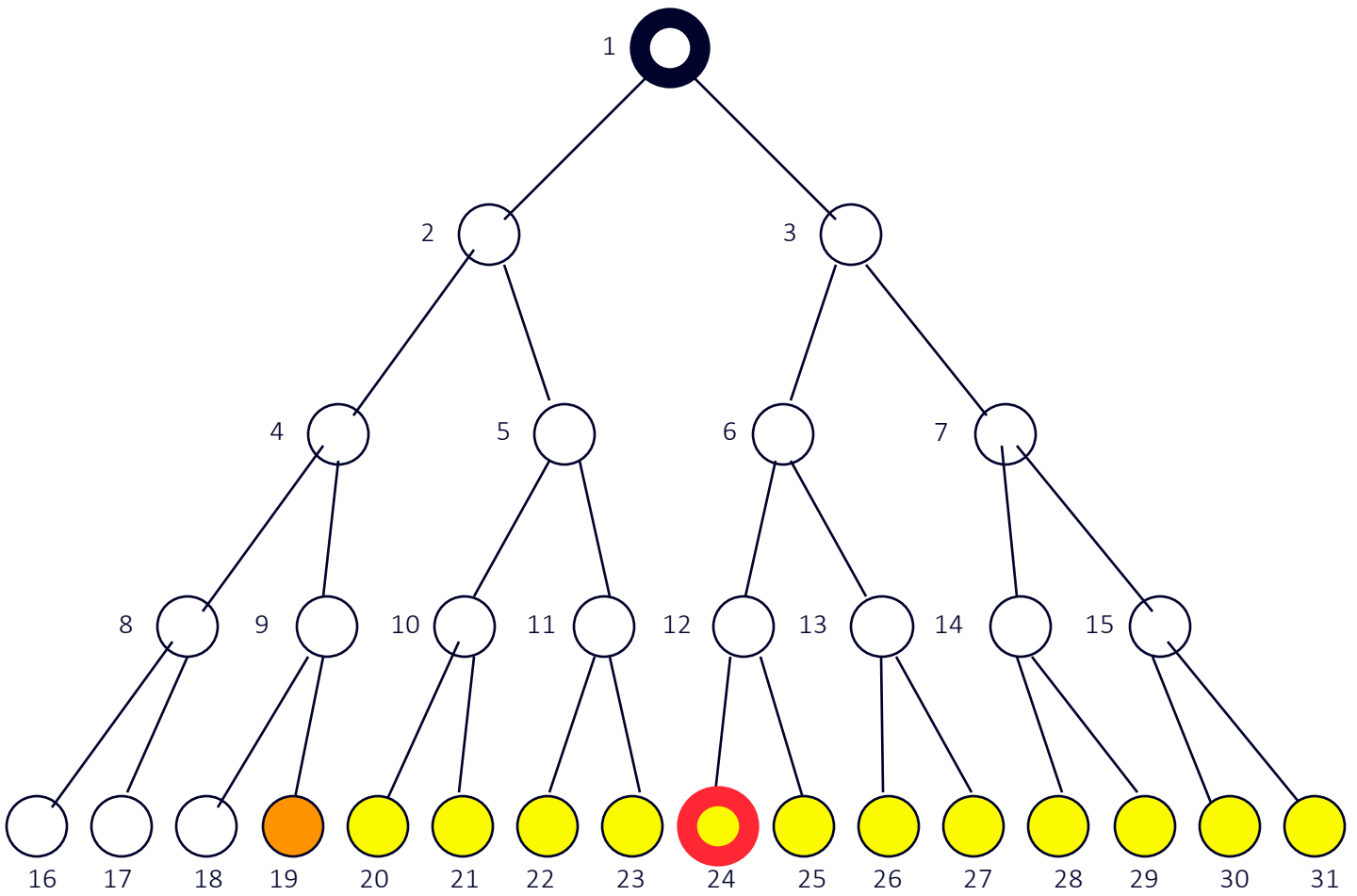
Current

Visible

Goal

Fringe: [19,20,21,22,23,24,25,26,27,28,29,30,31]

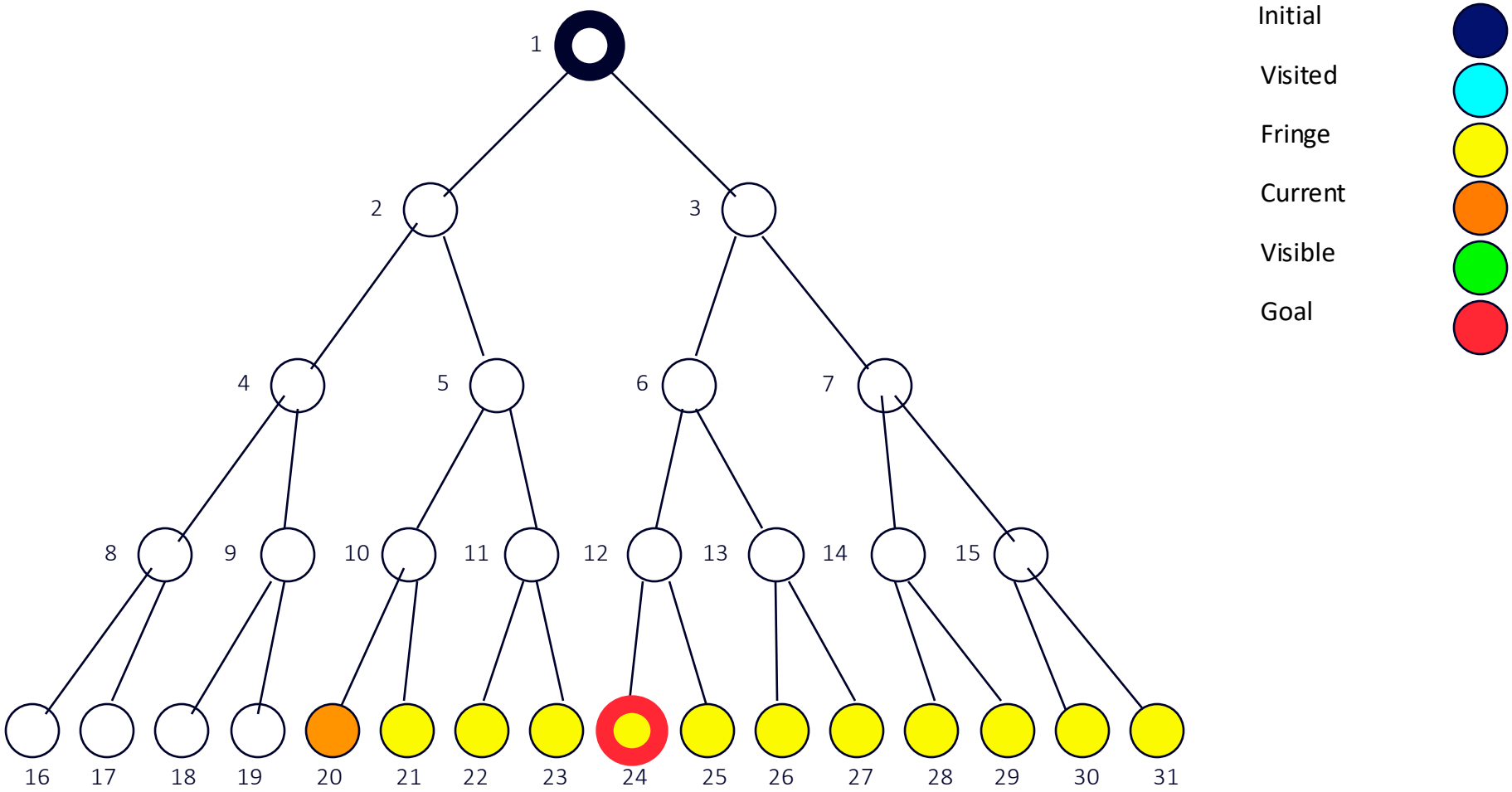
Breadth-First Snapshot 19



Initial
Visited
Fringe
Current
Visible
Goal

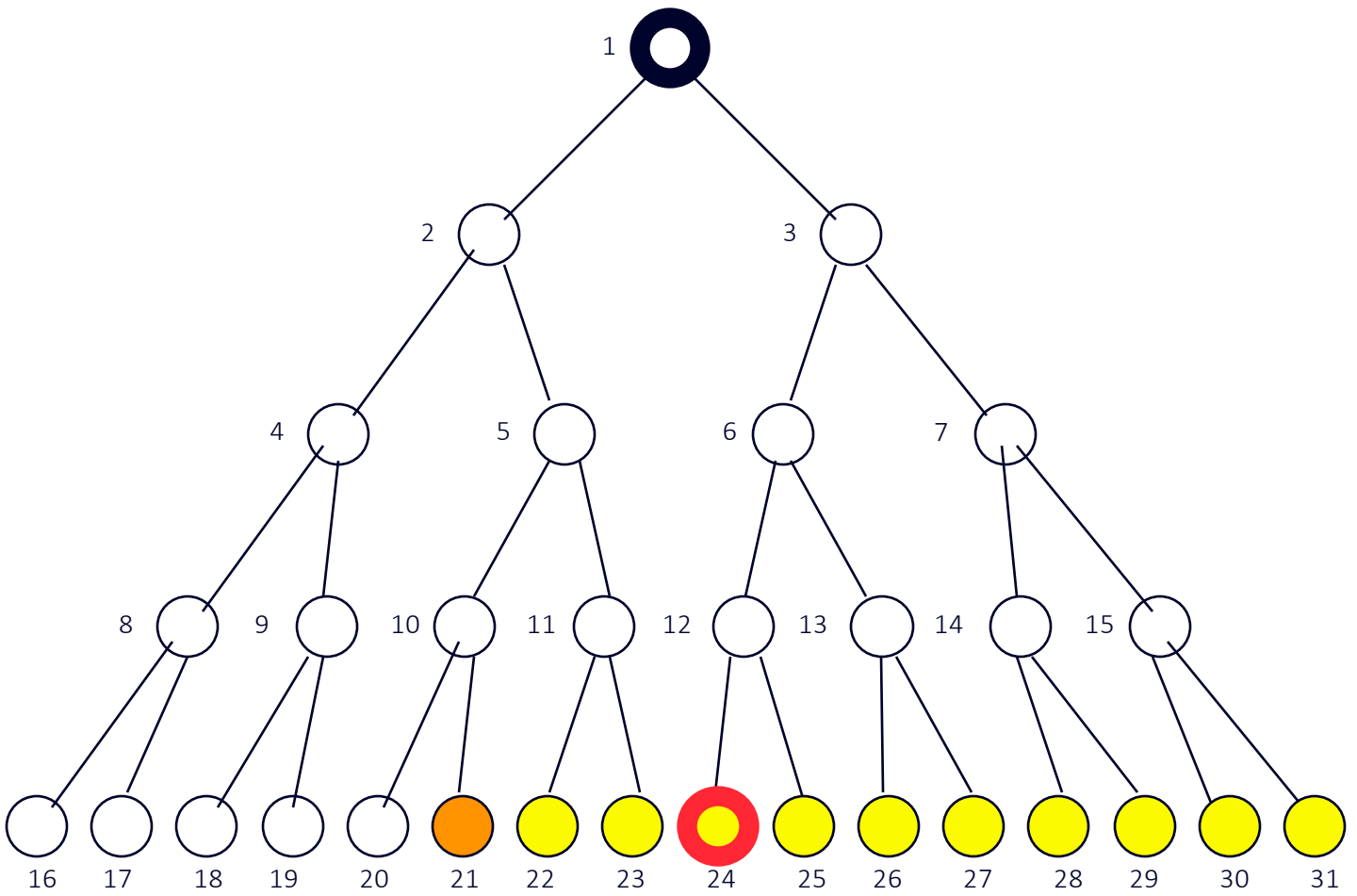
Fringe: [20,21,22,23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 20



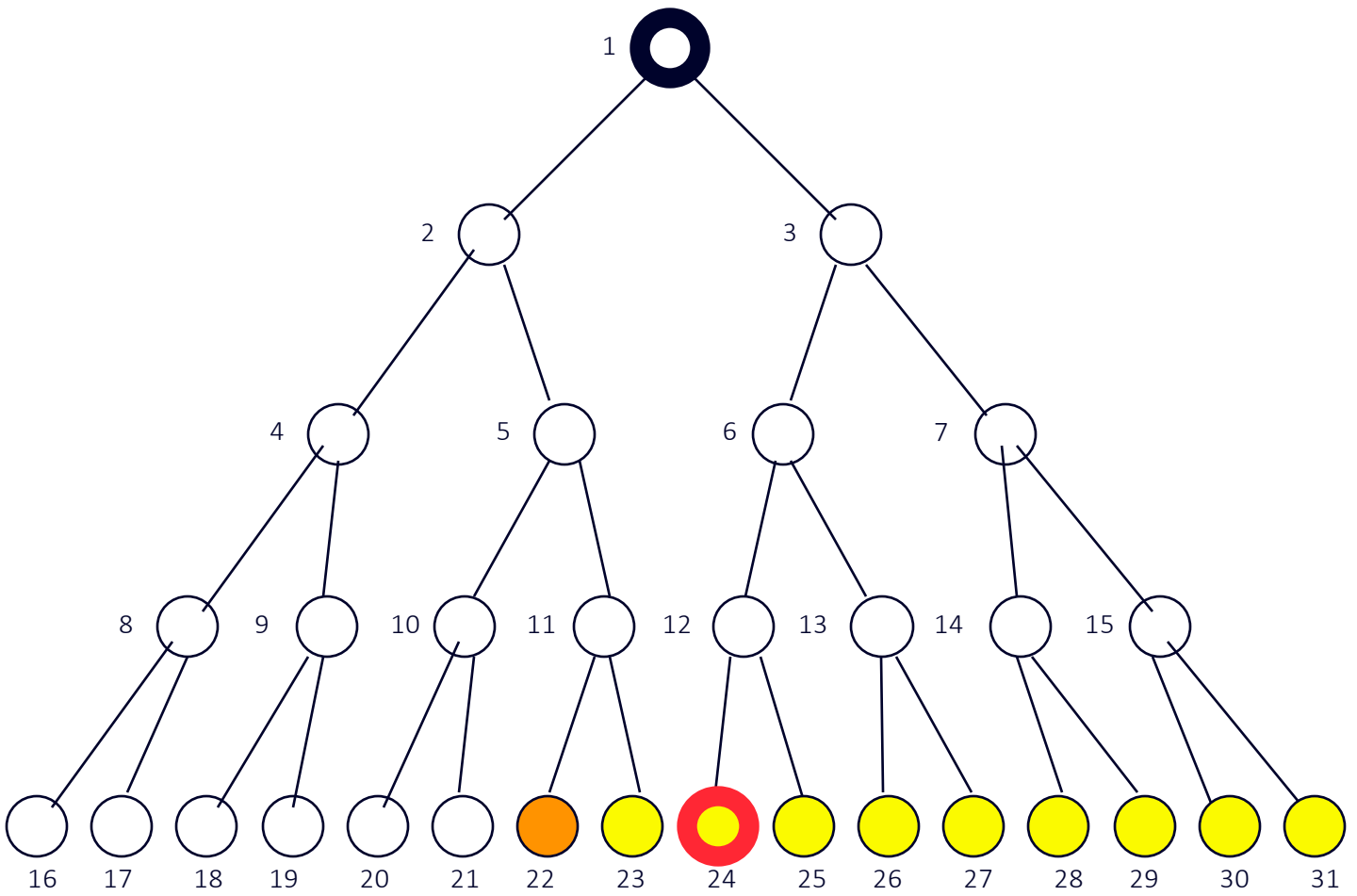
Fringe: [21,22,23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 21



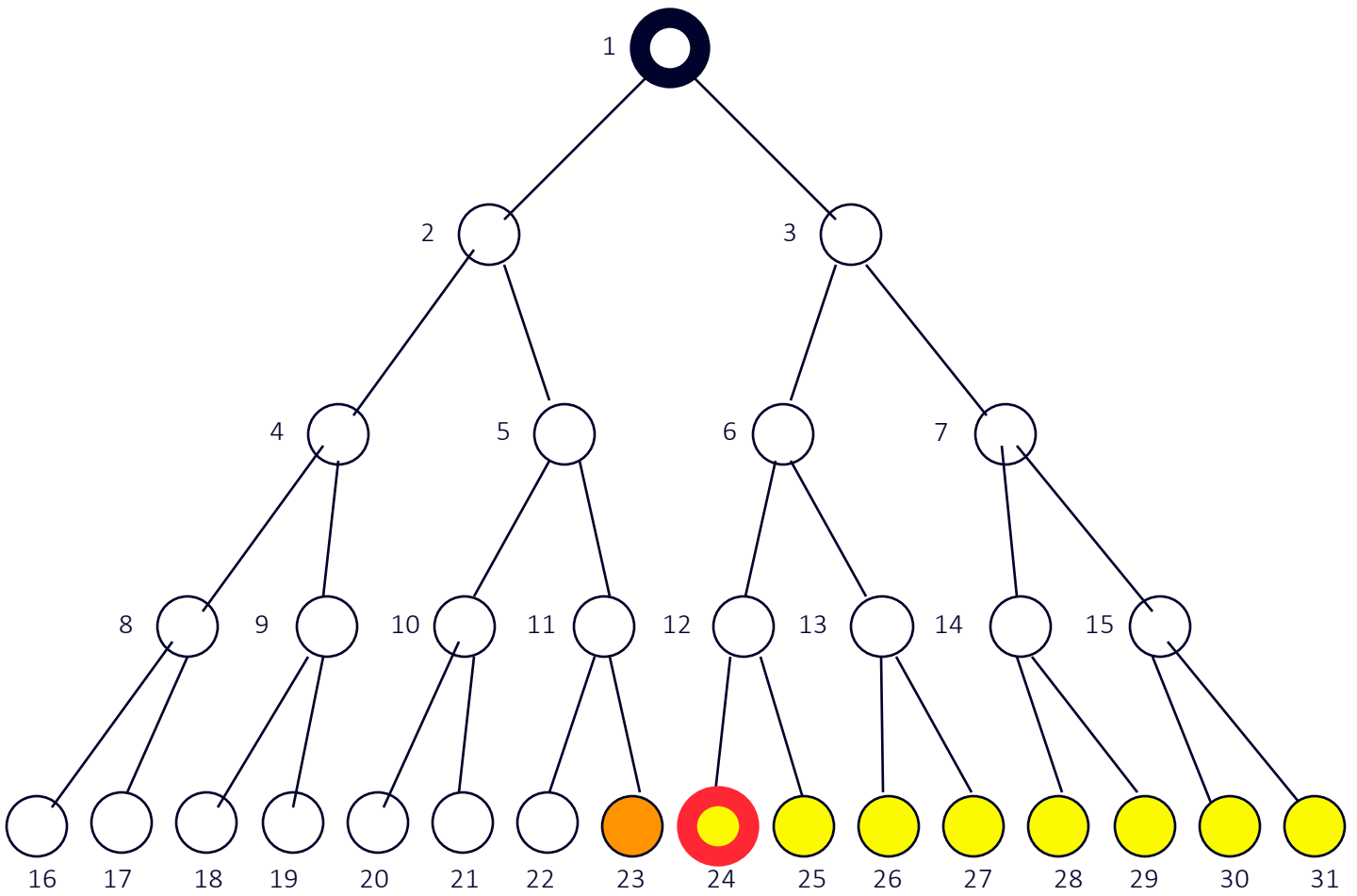
Fringe: [22,23,24,25,26,27,28,29,30,31]

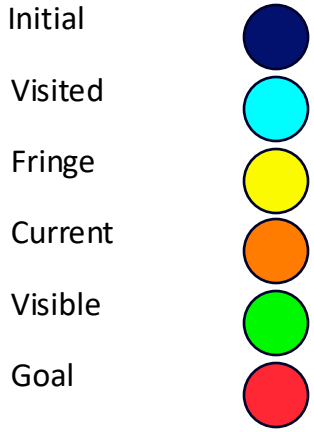
Breadth-First Snapshot 22



Fringe: [23,24,25,26,27,28,29,30,31]

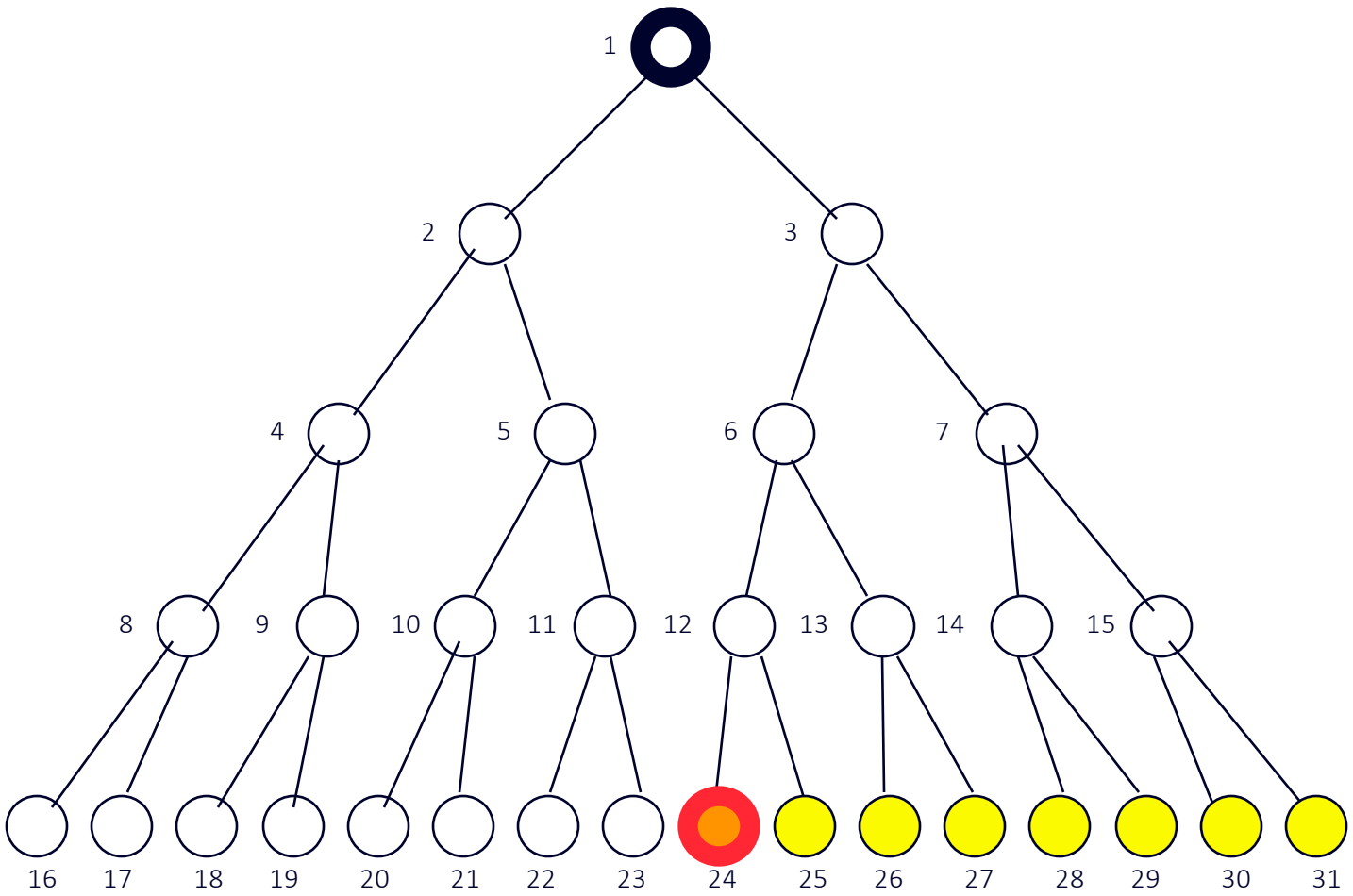
Breadth-First Snapshot 23





Fringe: [24,25,26,27,28,29,30,31]

Breadth-First Snapshot 24



Initial

Visited

Fringe

Current

Visible

Goal

Note:
The goal test is positive for this node, and a solution is found in 24 steps.

Fringe: [25,26,27,28,29,30,31]

Properties of Breadth-First Search

Time Complexity	$O(b^{d+1})$
Space Complexity	$O(b^{d+1})$
Completeness	yes (for finite b)
Optimality	yes (for non-negative path costs)

b	branching factor
d	depth of the optimal solution

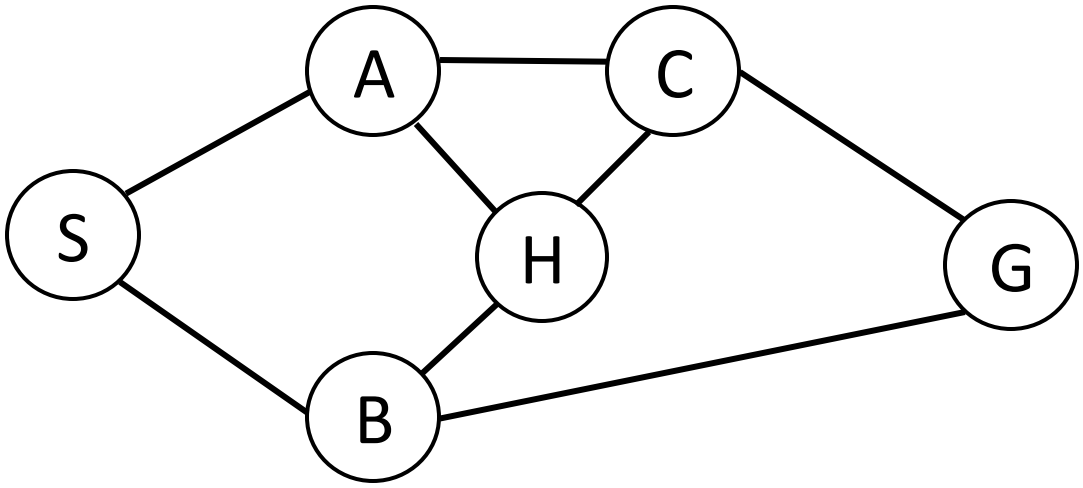
Breadth-First Search for Graphs

- Keep a set of explored (visited) nodes.

- When a node is expanded, add the generated nodes to frontier **only if not in the frontier or explored set**

Example: BFS for Graphs

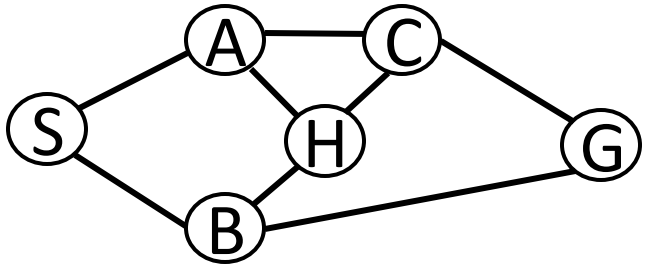
- For the following state space, list all the visited nodes and draw the search tree to go from S to G using BFS. What is the returned solution?

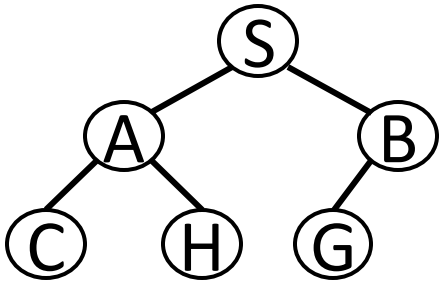


Example: BFS for Graphs (Cont.)

- Visited Nodes: S, A, B, C, H, G

- Solution (Path to goal): S, B, G



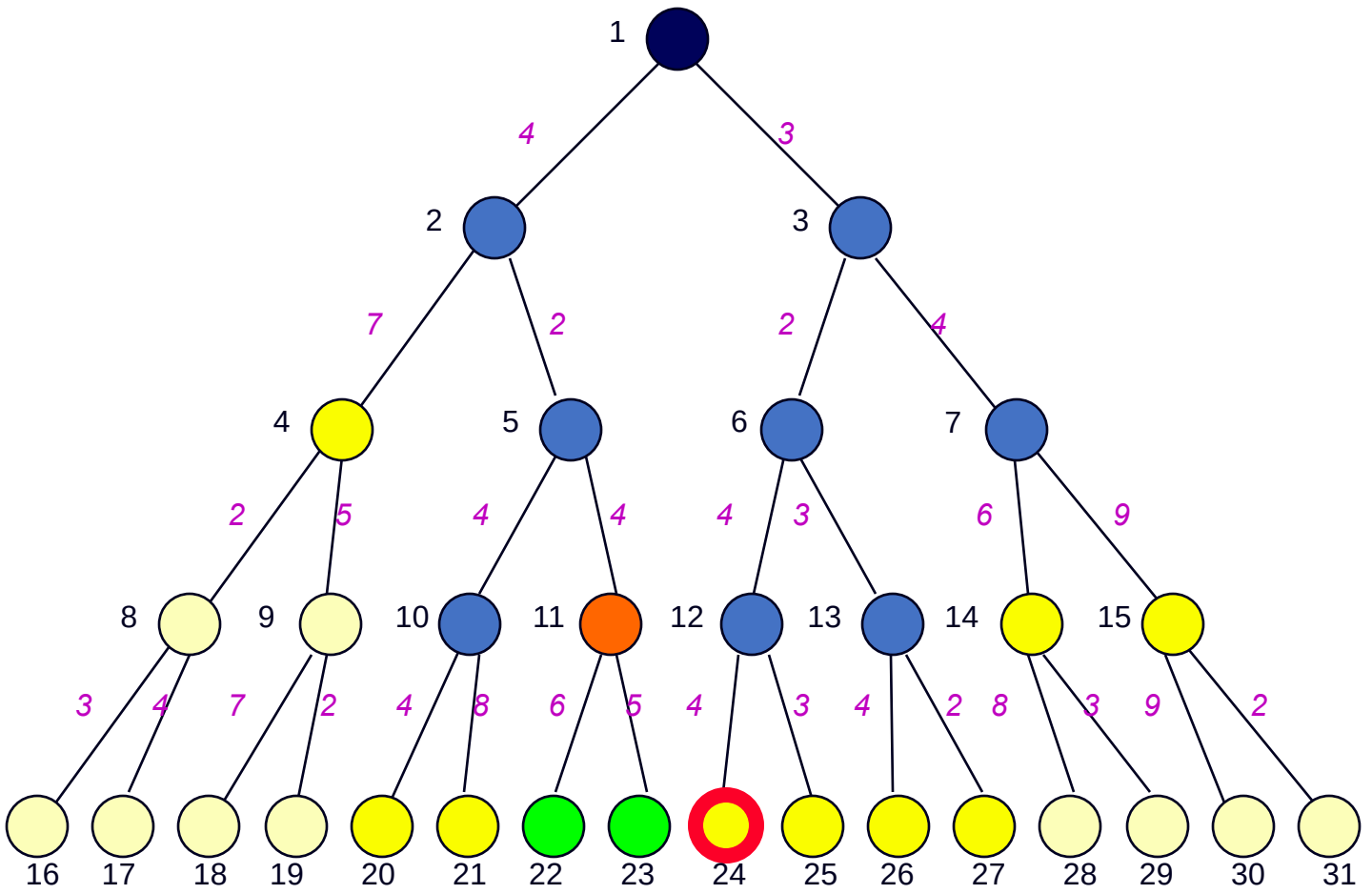


Uniform-Cost Search

- the nodes with the lowest cost are explored first
  - similar to BREADTH-FIRST, but with an evaluation of the cost for each reachable node
  - g(n) = path cost(n) = sum of individual edge costs to reach the current node
- Implementation: fringe is a queue ordered by path cost (priority queue)
- Equivalent to breadth-first if step costs all equal

```
function UNIFORM-COST-SEARCH(problem) returns solution

    return TREE-SEARCH(problem, COST-FN, FIFO-QUEUE())
```

Uniform-Cost Snapshot



Initial
Visited
Fringe
Current
Visible
Goal

Edge Cost 9

Fringe: [27(10), 4(11), 25(12), 26(12), 14(13), 24(13), 20(14), 15(16), 21(18)]
+ [22(16), 23(15)]

Uniform Cost Fringe Trace

1. [1(0)]
2. [3(3), 2(4)]
3. [2(4), 6(5), 7(7)]
4. [6(5), 5(6), 7(7), 4(11)]
5. [5(6), 7(7), 13(8), 12(9), 4(11)]
6. [7(7), 13(8), 12(9), 10(10), 11(10), 4(11)]
7. [13(8), 12(9), 10(10), 11(10), 4(11), 14(13), 15(16)]
8. [12(9), 10(10), 11(10), 27(10), 4(11), 26(12), 14(13), 15(16)]
9. [10(10), 11(10), 27(10), 4(11), 26(12), 25(12), 14(13), 24(13), 15(16)]
10. [11(10), 27(10), 4(11), 25(12), 26(12), 14(13), 24(13), 20(14), 15(16), 21(18)]
11. [27(10), 4(11), 25(12), 26(12), 14(13), 24(13), 20(14), 23(15), 15(16), 22(16), 21(18)]
12. [4(11), 25(12), 26(12), 14(13), 24(13), 20(14), 23(15), 15(16), 23(16), 21(18)]
13. [25(12), 26(12), 14(13), 24(13),8(13), 20(14), 23(15), 15(16), 23(16), 9(16), 21(18)]
14. [26(12), 14(13), 24(13),8(13), 20(14), 23(15), 15(16), 23(16), 9(16), 21(18)]
15. [14(13), 24(13),8(13), 20(14), 23(15), 15(16), 23(16), 9(16), 21(18)]
16. [24(13),8(13), 20(14), 23(15), 15(16), 23(16), 9(16), 29(16),21(18), 28(21)]
    Goal reached!

Notation: [**Bold+Yellow: Current Node**; White: Old Fringe Node; *Green+Italics: New Fringe Node*].
Assumption: New nodes with the same cost as existing nodes are added after the existing node.

Properties of Uniform-Cost Search

Time Complexity	$O(b^{C^*/e})$
Space Complexity	$O(b^{C^*/e})$
Completeness	yes (finite b, step costs >= e)
Optimality	yes

b	branching factor
C*	cost of the optimal solution
e	minimum cost per action

Uniform-Cost Search for Graphs

- Keep a set of explored (visited) nodes.
- When a node is expanded, add the generated nodes to frontier **only if not in the frontier or explored set**
  - **If the state of the generated node is in the frontier but with higher cost, replace that frontier node with newly generated node**

Breadth-First vs. Uniform-Cost

- breadth-first always expands the shallowest node
  - only optimal if all step costs are equal

- uniform-cost considers the overall path cost
  - optimal for any (reasonable) cost function
    - non-zero, positive
  - gets bogged down in trees with many fruitless, short branches
    - low path cost, but no goal node

- both are complete for non-extreme problems
  - finite number of branches
  - strictly positive search function

Depth-First Search

- continues exploring newly generated nodes
  - achieved by the TREE-SEARCH method by appending newly generated nodes at the beginning of the search queue

- Implementation: fringe = LIFO queue, i.e., put successors at front

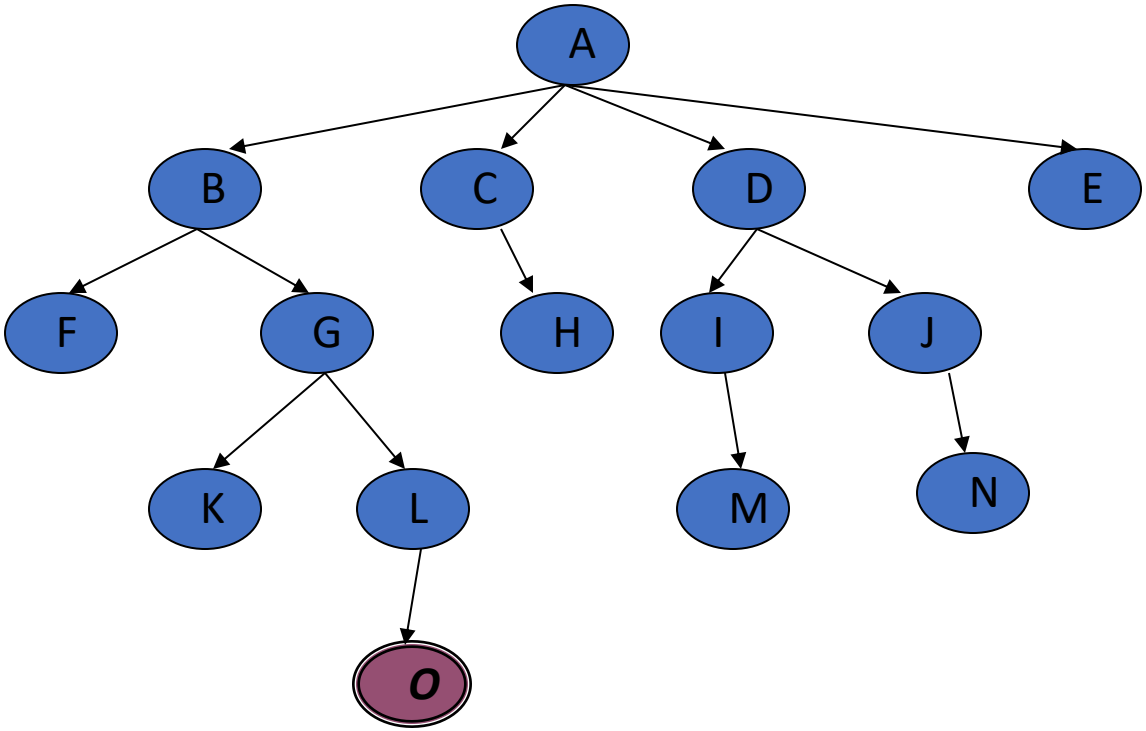
```
function DEPTH-FIRST-SEARCH(problem) returns solution

        return TREE-SEARCH(problem, LIFO-QUEUE())
```

Time Complexity	$O(b^m)$
Space Complexity	O(b*m)
Completeness	no (for infinite branch length)
Optimality	no

b	branching factor
m	maximum path length

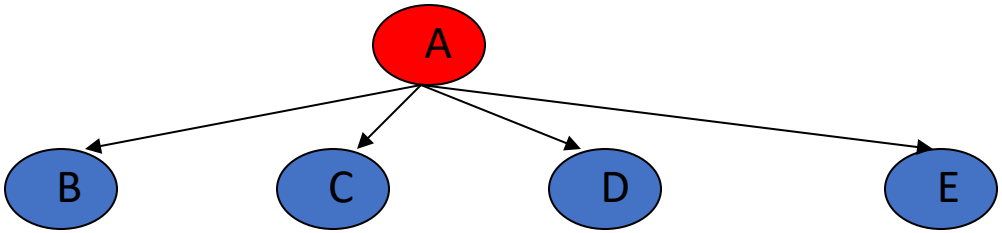
Depth-First Search

- Given the following state space (tree search), give the sequence of visited nodes when using DFS (assume that the node O is the goal state):



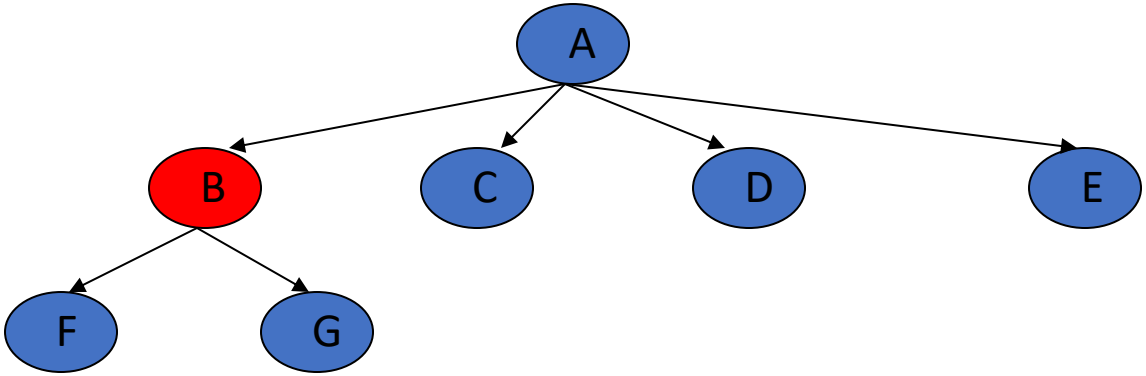
Depth-First Search

- A,



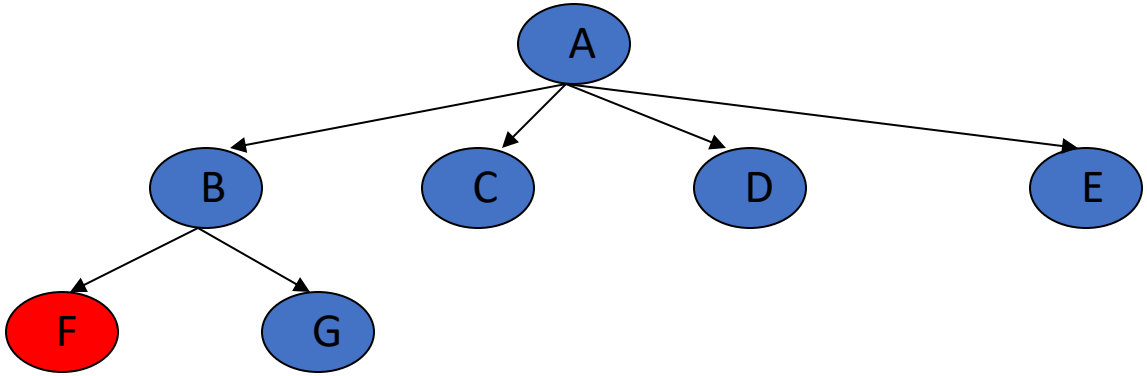
Depth-First Search

- A, B,



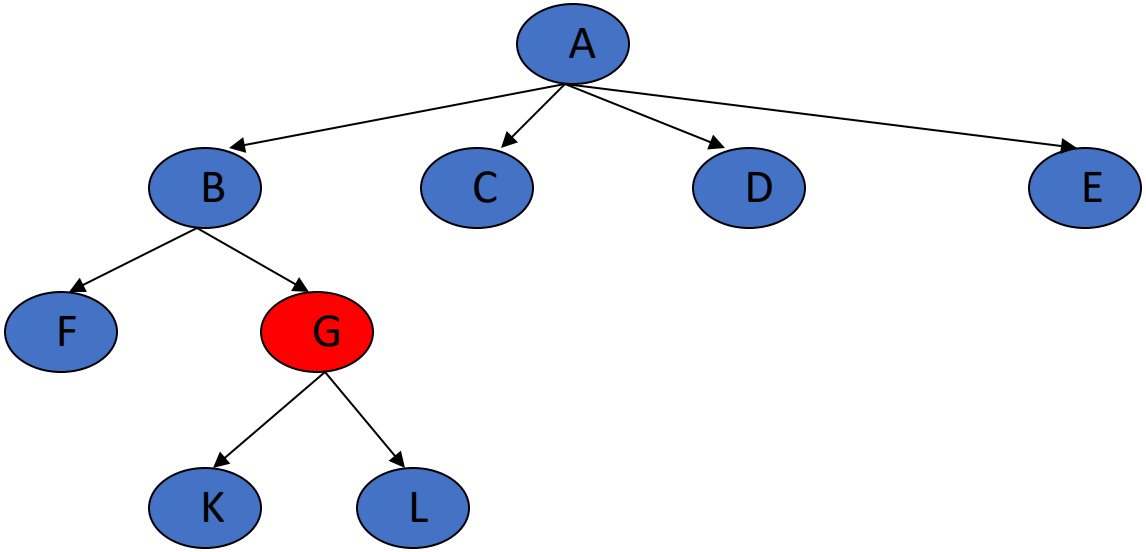
Depth-First Search

- A, B, F,



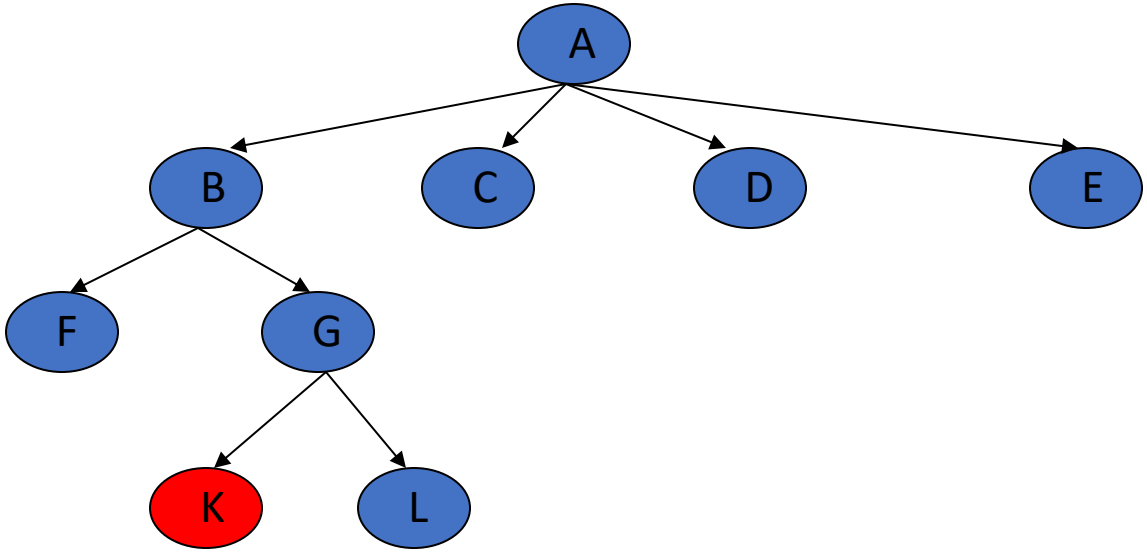
Depth-First Search

- A, B, F,
- G,



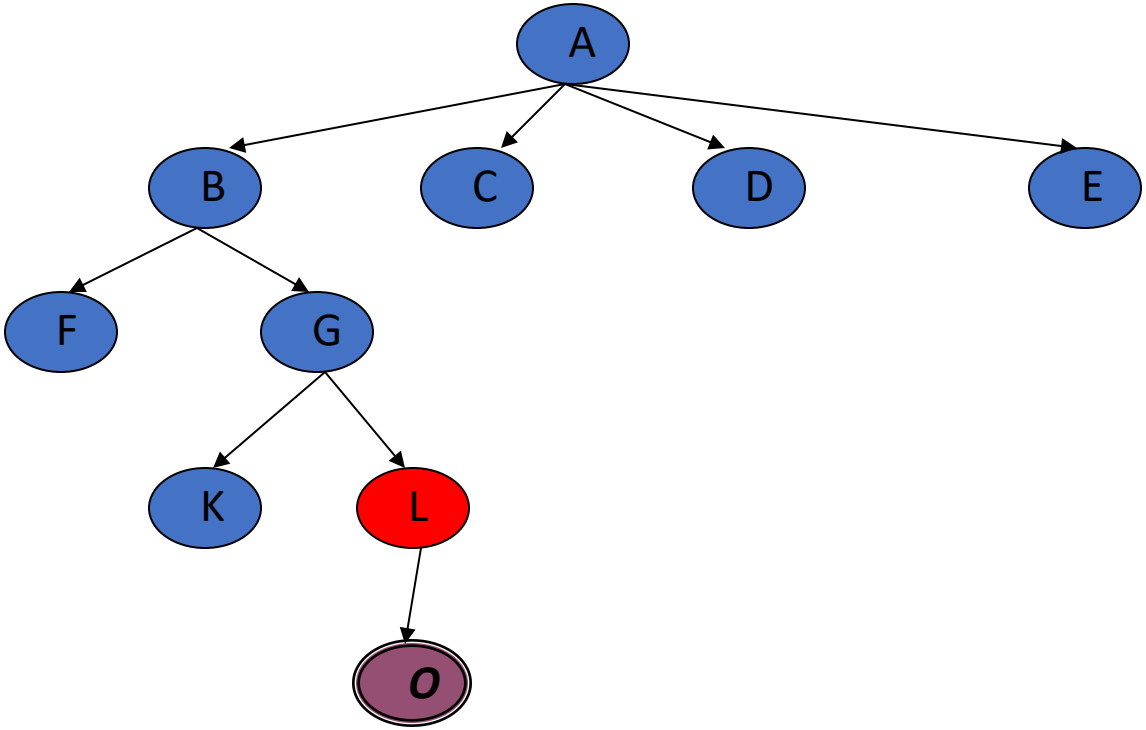
Depth-First Search

- A, B, F,
- G, K,



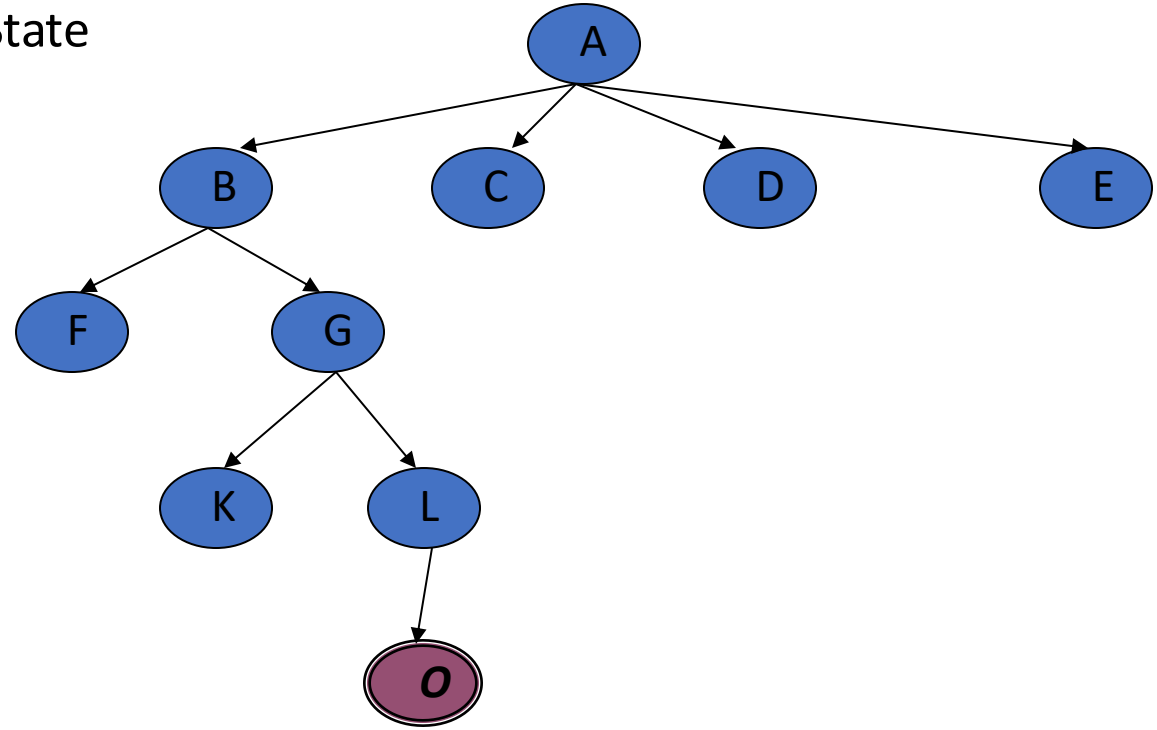
Depth-First Search

- A, B, F,
- G, K,
- L,



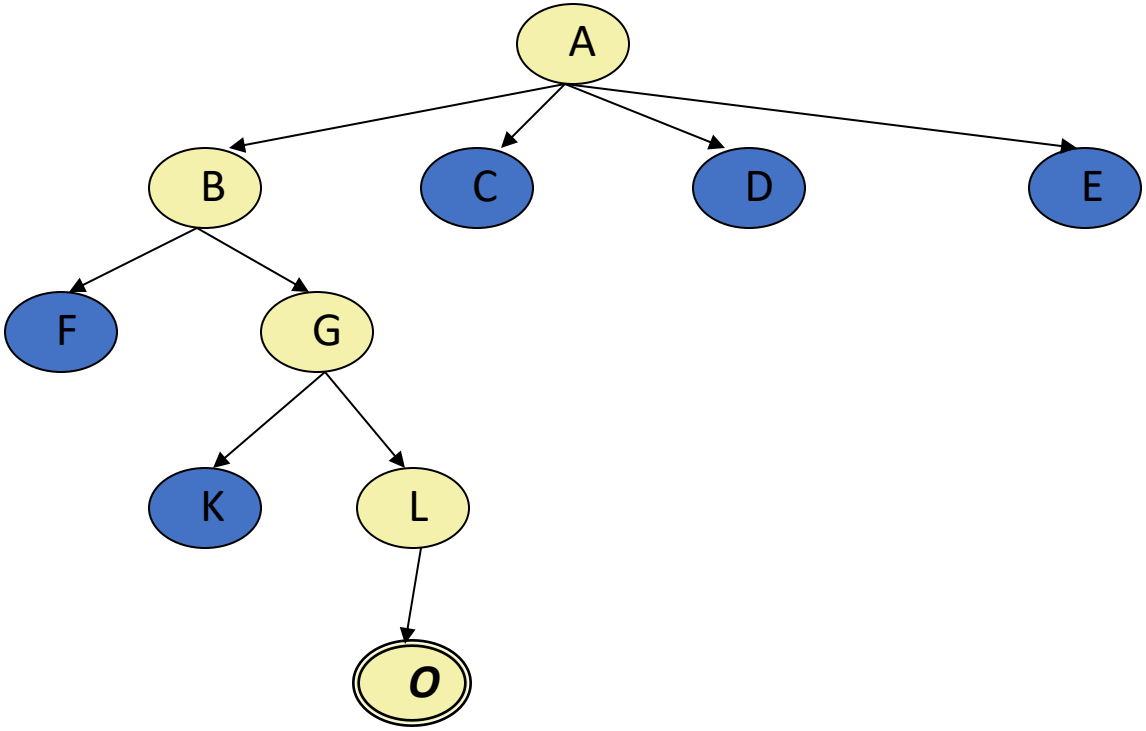
Depth-First Search

- A, B, F,
- G, K,
- L, O: Goal State



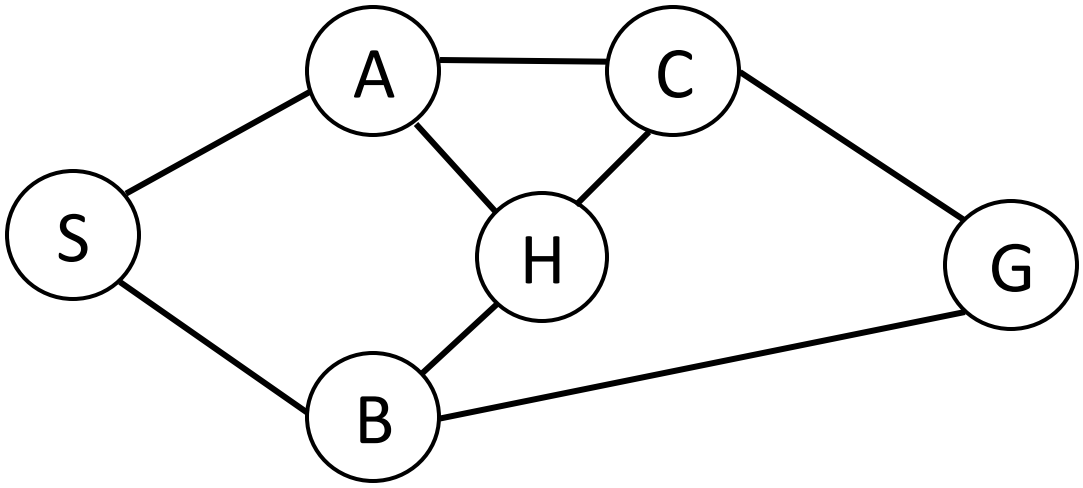
Depth-First Search

- The returned solution is the sequence of operators in the path:
  A, B, G, L, O



Example: DFS for Graphs

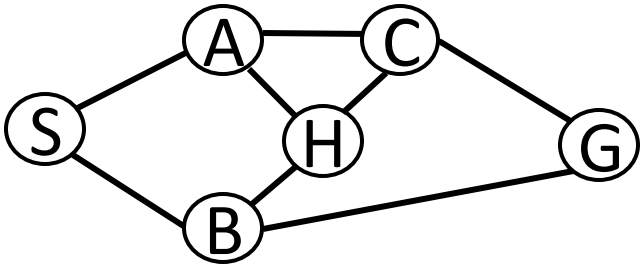
- For the following state space, list all the visited nodes and draw the search tree to go from S to G using DFS. What is the returned solution?

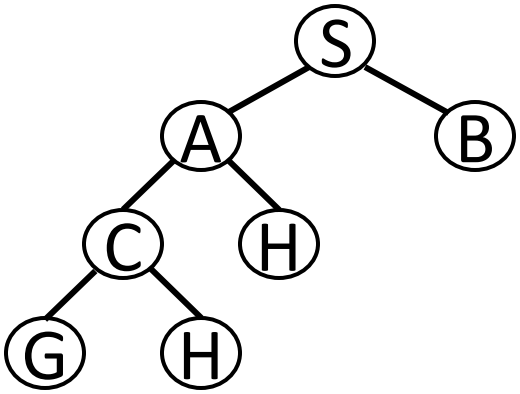


Example: DFS for Graphs (Cont.)

- Visited Nodes: S, A, C, G

- Solution (Path to goal): S, A, C, G





Depth-First vs. Breadth-First

- depth-first goes off into one branch until it reaches a leaf node
  - not good if the goal is on another branch
  - neither complete nor optimal
  - uses much less space than breadth-first

- breadth-first is more careful by checking all alternatives
  - complete and optimal
  - very memory-intensive

Depth-Limited Search

- similar to depth-first, but with a limit
  - overcomes problems with infinite paths
  - sometimes a depth limit can be inferred or estimated from the problem description
  - based on the TREE-SEARCH method
  - must keep track of the depth

```
function DEPTH-LIMITED-SEARCH(problem, depth-limit) returns solution

        return TREE-SEARCH(problem, depth-limit, LIFO-QUEUE())
```

Time Complexity	$O(b^l)$
Space Complexity	$O(b*l)$
Completeness	no (goal beyond l, or infinite branch length)
Optimality	no

b	branching factor
l	depth limit

Iterative Deepening

- applies LIMITED-DEPTH with increasing depth limits
- combines advantages of BREADTH-FIRST and DEPTH-FIRST methods
- many states are expanded multiple times
  - doesn't really matter because the number of those nodes is small
- in practice, one of the best uninformed search methods
  - for large search spaces, unknown depth

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns solution
   for depth := 0  to unlimited  do
       result := DEPTH-LIMITED-SEARCH(problem, depth-limit)
       if result != cutoff then return result
```

Time Complexity	$O(b^d)$
Space Complexity	O(b*d)
Completeness	yes (finite b)
Optimality	yes (all step costs identical)

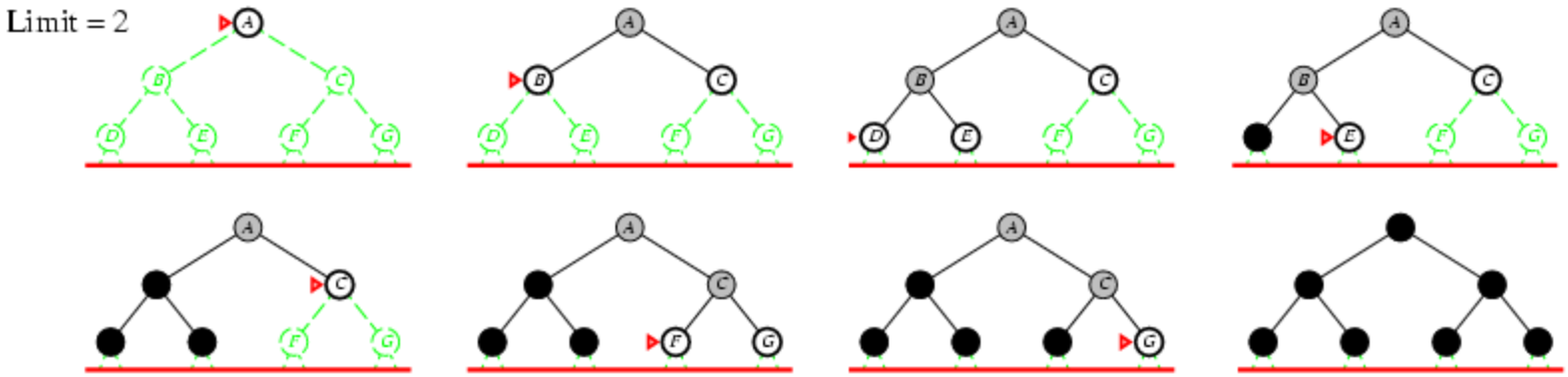
b	branching factor
d	depth of optimal solution

Iterative deepening search

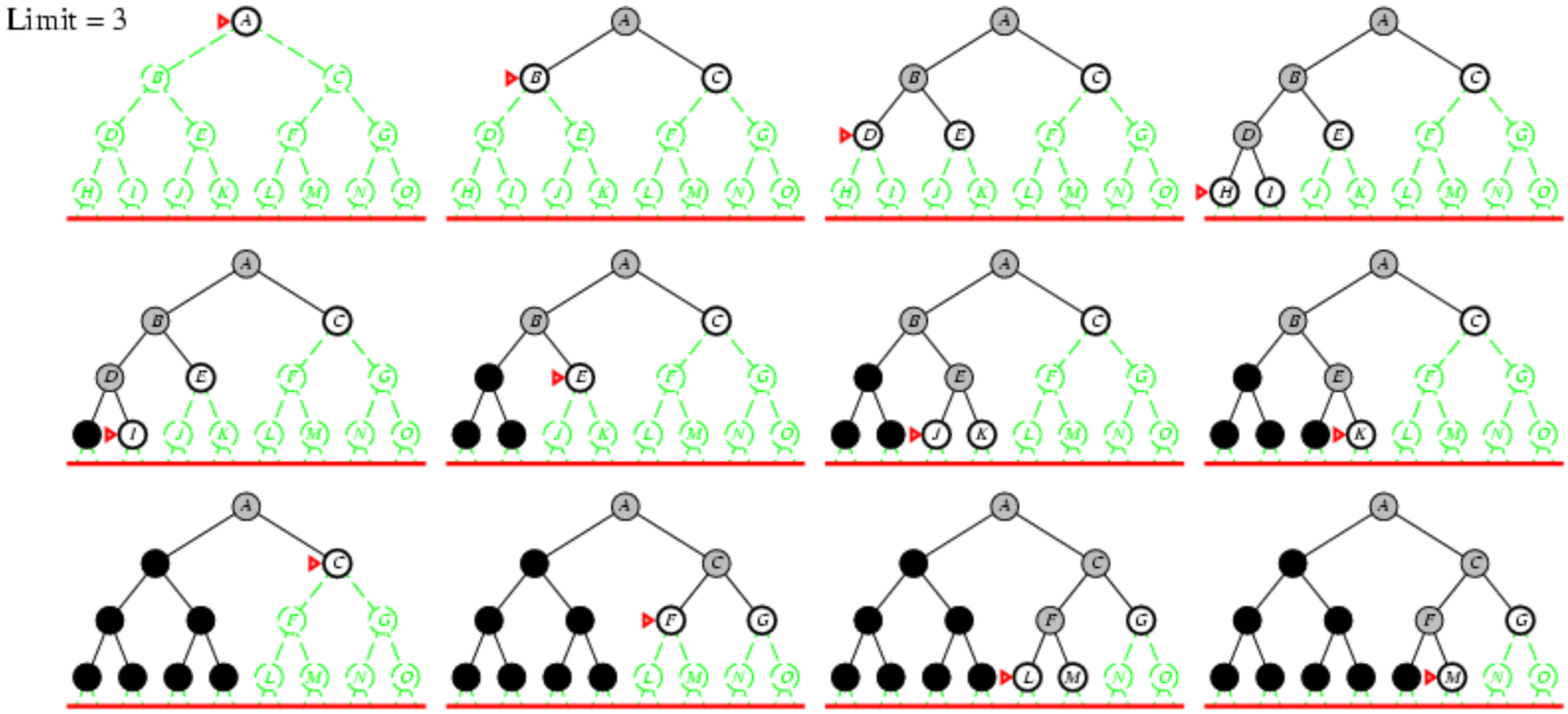
Limit = 0

Iterative deepening search

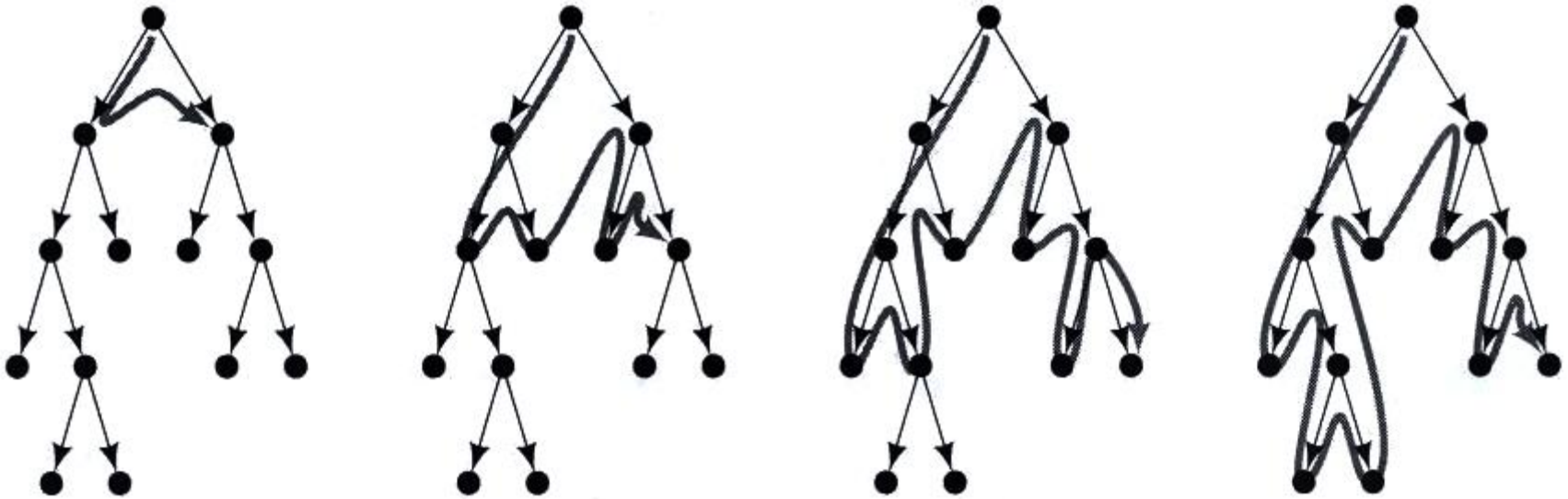
Iterative deepening search



Iterative deepening search



Example IDS



Stages in Iterative-Deepening Search

Bi-directional Search

- search simultaneously from two directions
  - forward from the initial and backward from the goal state
- may lead to substantial savings if it is applicable
- has severe limitations
  - predecessors must be generated, which is not always possible
  - search must be coordinated between the two searches
  - one search must keep all nodes in memory

Time Complexity	$O(b^{d/2})$
Space Complexity	$O(b^{d/2})$
Completeness	yes (b finite, breadth-first for both directions)
Optimality	yes (all step costs identical, breadth-first for both directions)

b	branching factor
d	depth of optimal solution

Improving Search Methods

- assumption for improvements
  - remember information about the search so far
    - all nodes visited so far
    - path to the current node
- make algorithms more efficient
  - avoiding repeated states
  - utilizing memory efficiently
- use additional knowledge about the problem => informed search
  - properties (“shape”) of the search space
    - more interesting areas are investigated first
  - pruning of irrelevant areas
    - areas that are guaranteed not to contain a solution can be discarded

Comparing Uninformed Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^{a}	Yes^{a,b}	No	No	Yes^{a}	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^{c}	Yes	No	No	Yes^{c}	Yes^{c,d}

Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.