Search Terminology

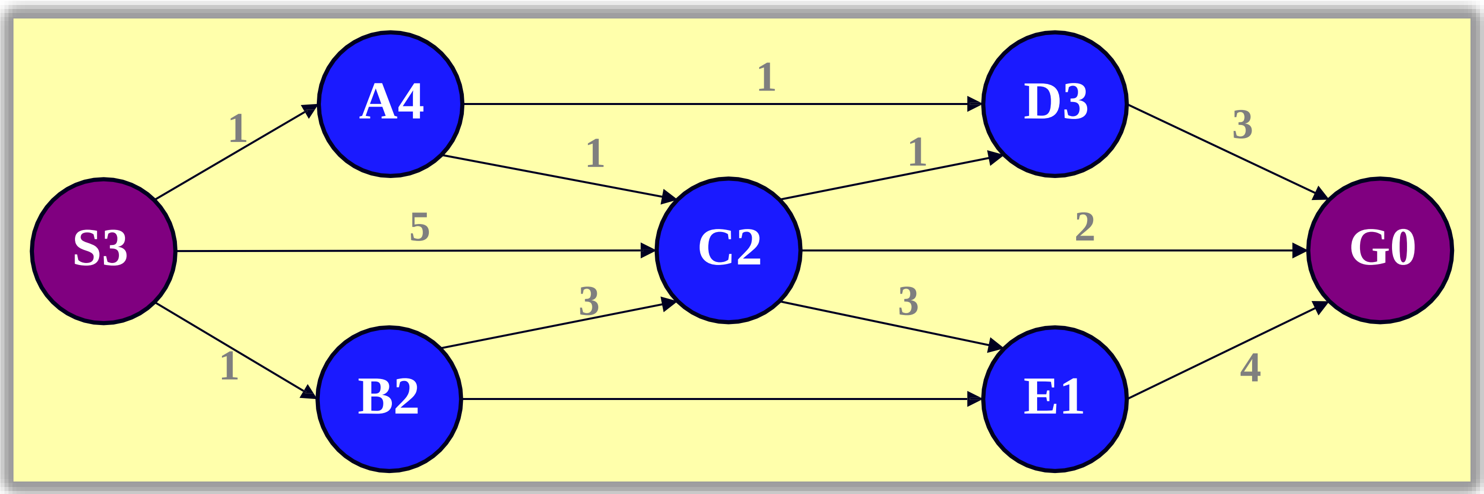
Search Tree

- Generated as the search space is traversed
  - The search space itself is not necessarily a tree, frequently it is a graph
  - The tree specifies possible paths through the search space
- Expansion of nodes
  - As states are explored, the corresponding nodes are expanded by applying the successor function
    - this generates a new set of (child) nodes
  - The *fringe* (frontier/queue) is the set of nodes not yet visited
    - newly generated nodes are added to the fringe
- Search strategy
  - Determines the selection of the next node to be expanded
  - Can be achieved by ordering the nodes in the fringe
    - e.g. queue (FIFO), stack (LIFO), “best” node w.r.t. some measure (cost)

Search Tree Vs Graph Tree

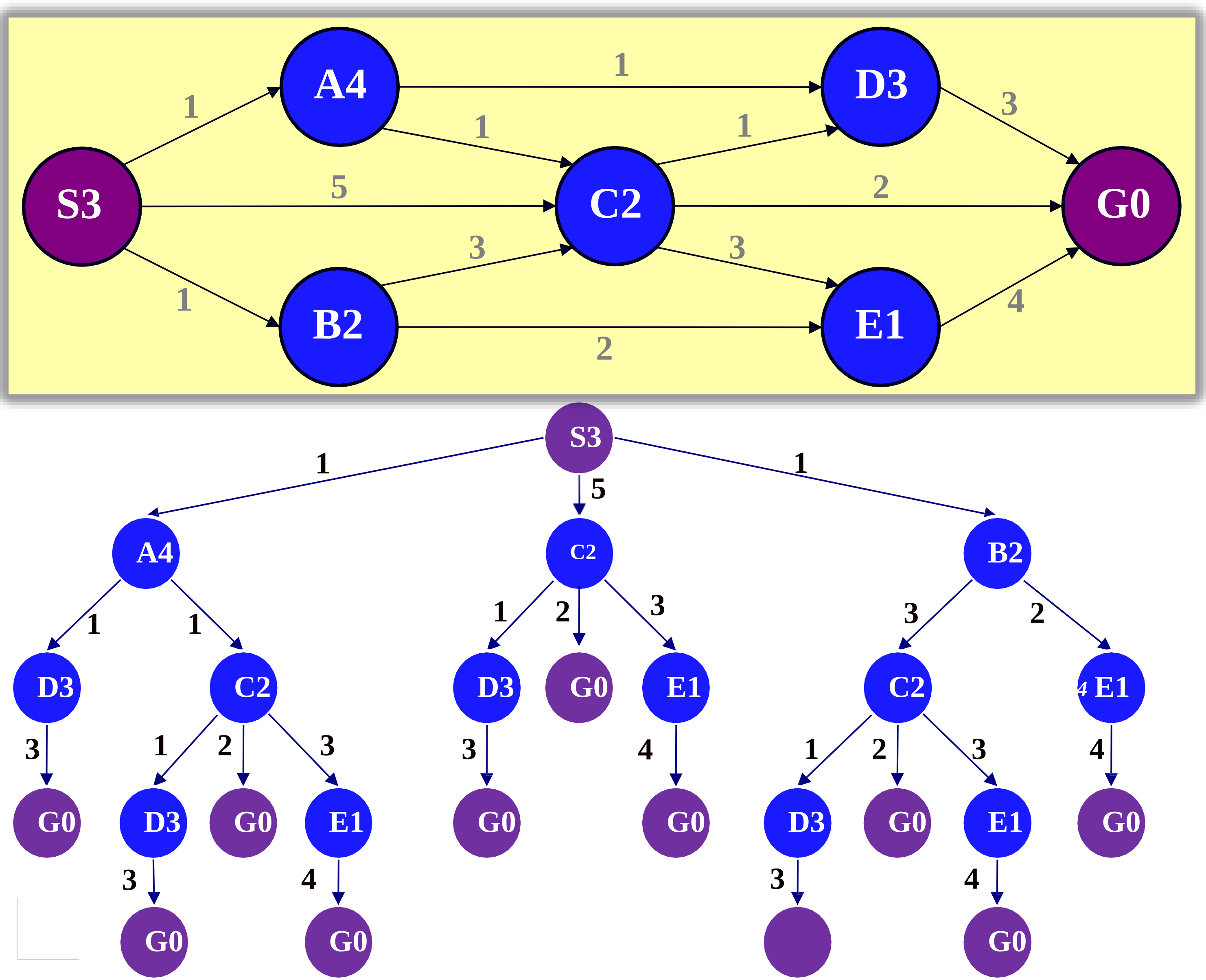
BASIS FOR COMPARISON	TREE	GRAPH
Path	Only one between two vertices.	More than one path is allowed.
Root node	It has exactly one root node.	Graph doesn't have a root node.
Loops	No loops are permitted.	Graph can have loops.
Complexity	Less complex	More complex comparatively
Traversal techniques	Pre-order, In-order and Post-order.	Breadth-first search and depth-first search.
Number of edges	n-1 (where n is the number of nodes)	Not defined
Model type	Hierarchical	Network

Example: Graph Search



- The graph describes the search (state) space
  - Each node in the graph represents one state in the search space
    - e.g. a city to be visited in a routing or touring problem
- This graph has additional information
  - Names and properties for the states (e.g. S3)
  - Links between nodes, specified by the successor function
    - properties for links (distance, cost, name, ...)

Traversing a Graph as Tree



- A tree is generated by traversing the graph.
- The same node in the graph may appear repeatedly in the tree.
- The arrangement of the tree depends on the traversal strategy (search method)
- The initial state becomes the root node of the tree
- In the fully expanded tree, the goal states are the leaf nodes.
- Cycles in graphs may result in infinite branches

Kruskal's Algorithm

Kruskal's Algorithm includes 4 Steps:

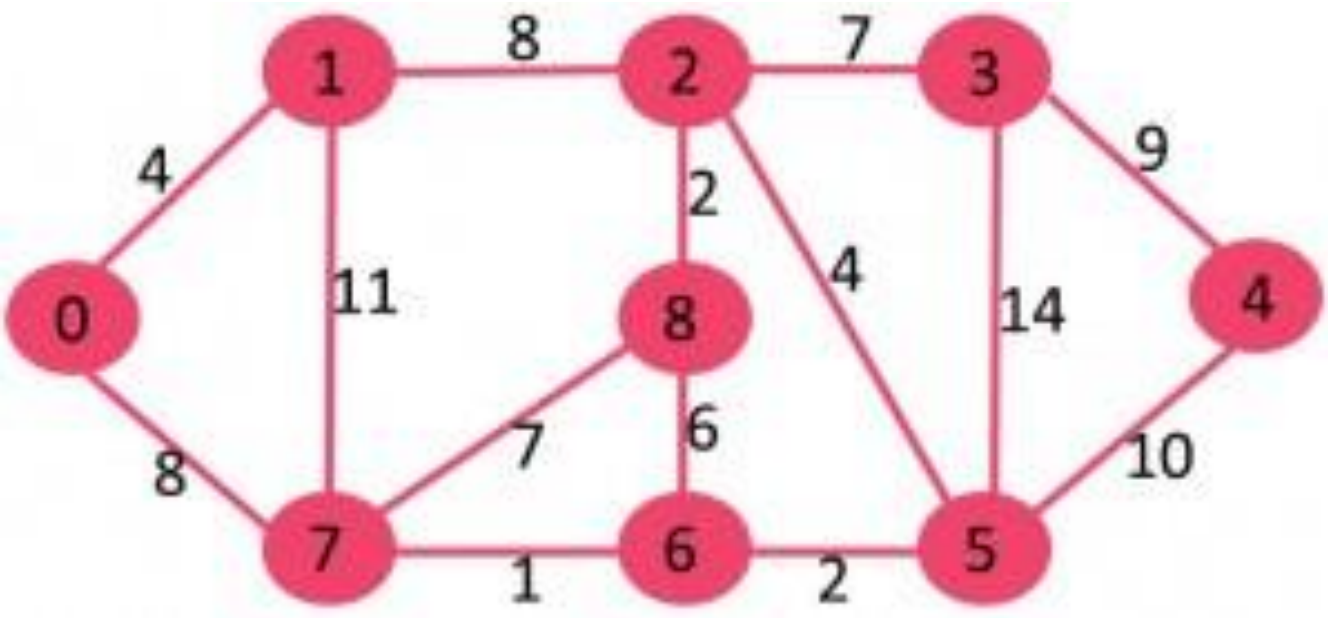
Step 1: List all the edges of the graph in order of increasing weights.

Step 2: Select the smallest edge of the graph.

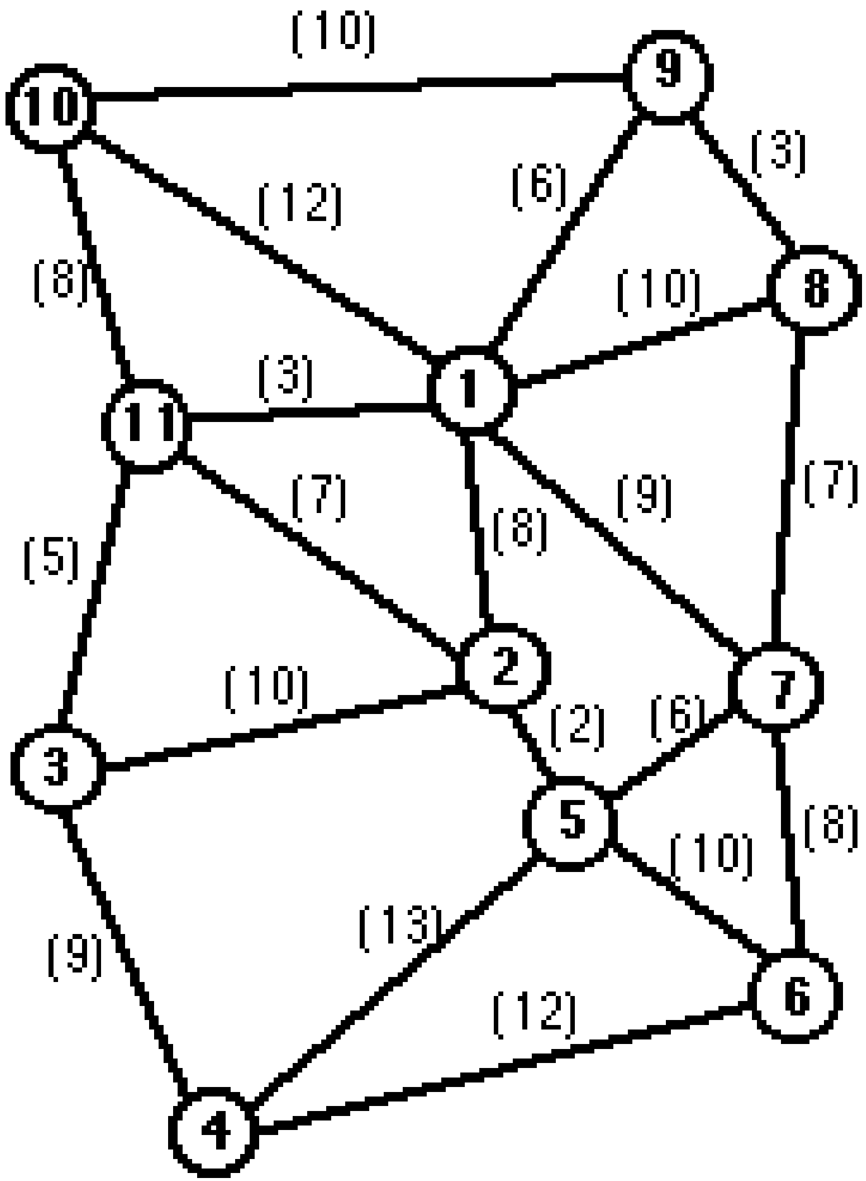
Step 3: Select the next smallest edge that do not makes any circuit.

Step 4: Continue this process until all the Vertices are explored and (V_n-1) edges have been selected

Minimum Spanning Tree (MST) Kruskal’s Algorithm



Find MST using Kruskal's Algorithm



Prim’s Algorithm

Prim’s Algorithm includes 4 Steps:

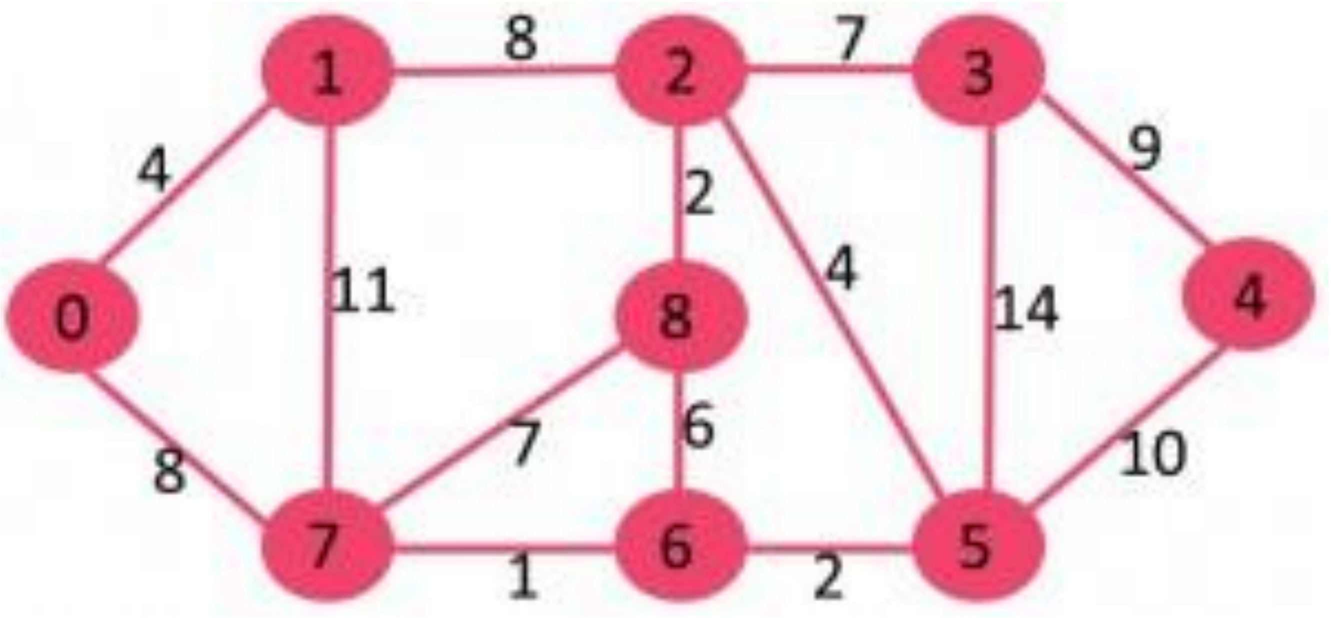
Step 1: Draw an n by n (n x n) vertices’ matrix and label them as V_1, V_2.....V_n along with the given weights of the edges.

Step 2: Starting from vertex V_1 and connect it to its nearest neighbor by searching in row 1.

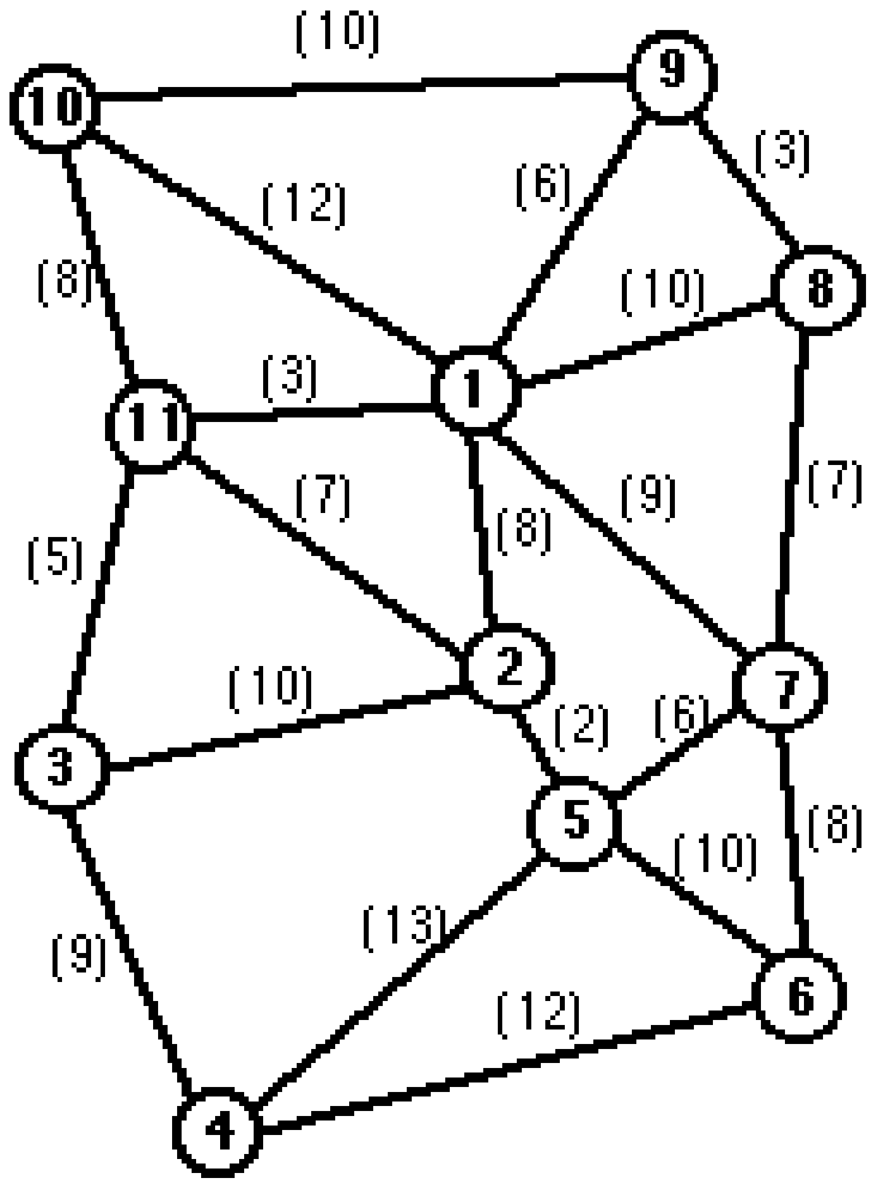
Step 3: Consider V1 and Vi as one subgraph and connect as step 2 while do not forming any circuit.

Step 4: Continue this process until we get the MST having n vertices and (Vi-1) edges.

Prim’s Algorithm



Find MST using Prim's Algorithm



Searching Strategies

Uninformed Search

- breadth-first
- uniform-cost search
- depth-first
- depth-limited search
- iterative deepening
- bi-directional search

Informed Search

- best-first search
- search with heuristics
- memory-bounded search
- iterative improvement search

Most of the effort is often spent on the selection of an appropriate search strategy for a given problem:

- Uninformed Search (blind search)
  - number of steps, path cost unknown
  - agent knows when it reaches a goal
- Informed Search (heuristic search)
  - agent has background information about the problem
    - map, costs of actions

Evaluation of Search Strategies

A search strategy is defined by picking the order of node expansion

Strategies are evaluated along the following dimensions:

- Completeness: if there is a solution, will it be found
- Time complexity: How long does it takes to find the solution
- Space complexity: memory required for the search
- Optimality: will the best solution be found

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 ∞)

1. Breadth-First Search (BFS) Algorithm

Breadth-First Search

- It is the most common search strategy for traversing a tree or graph.
- This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all the successor nodes at the current level before moving to node of next level.
- BFS is implemented using FIFO Queue data structure.

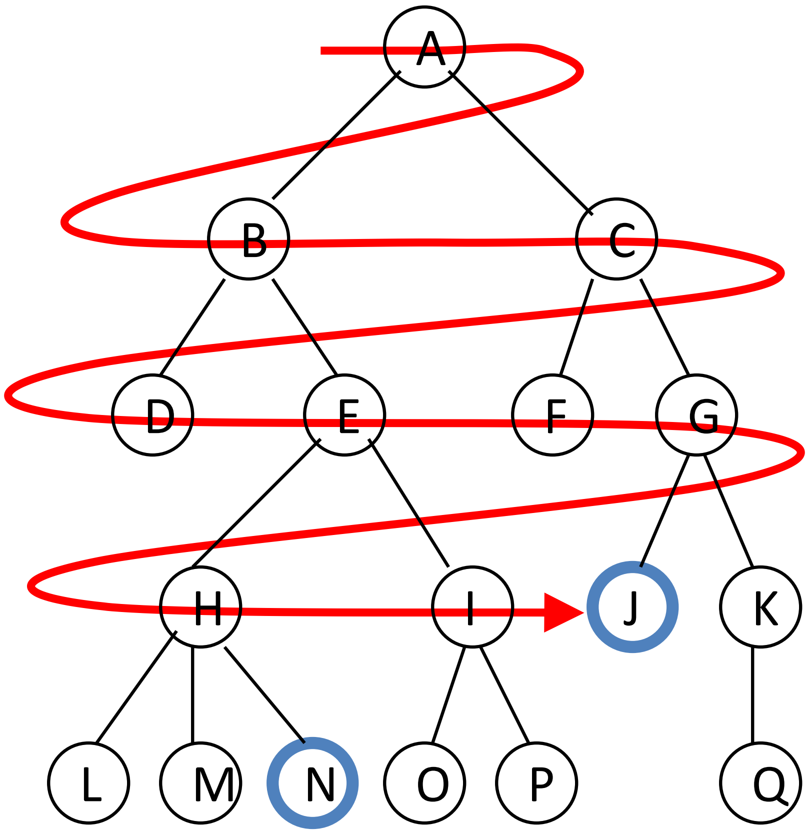
Breadth-First Search

All the nodes reachable from the current node are explored first (shallow nodes are expanded before deep nodes).

Algorithm (Informal)

1. Enqueue the root/initial node (Queue Structure).
2. Dequeue a node and examine it.
    1. If the element sought is found in this node, quit the search and return a result.
    2. Otherwise enqueue any successors (the direct child nodes) that have not yet been discovered.
3. If the queue is empty, every node on the graph has been examined – quit the search and return "not found".
4. Repeat from Step 2.

Breadth-First Search



A breadth-first search (BFS) explores nodes nearest the root before exploring nodes further away on each level

For example, after searching A, then B, then C, the search proceeds with D, E, F, G

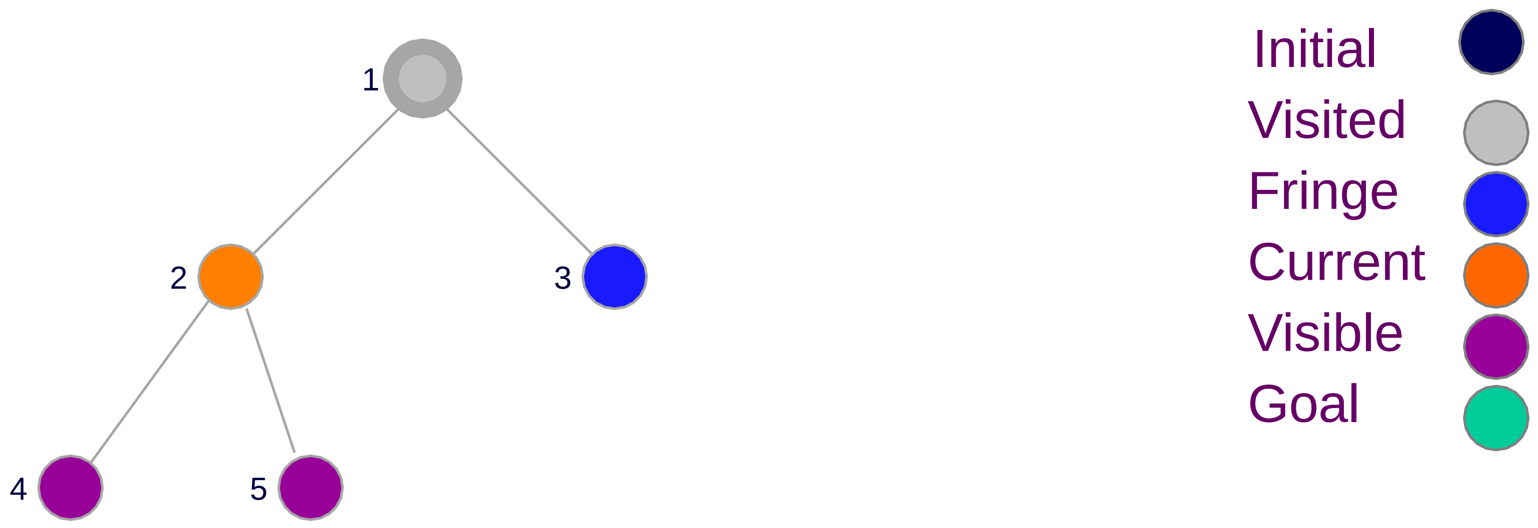
Node are explored in the Level order A B C D E F G H I J K L M N O P Q

J will be found before N

Breadth-First Snapshot 1

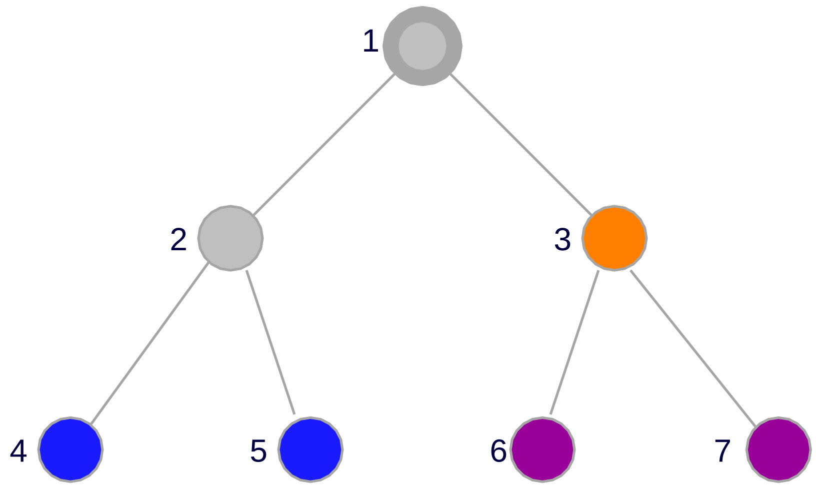
Fringe: [] + [2,3]

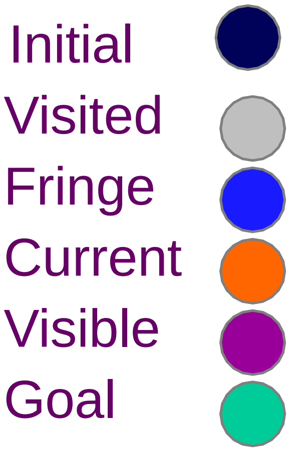
Breadth-First Snapshot 2



Fringe: [3] + [4,5]

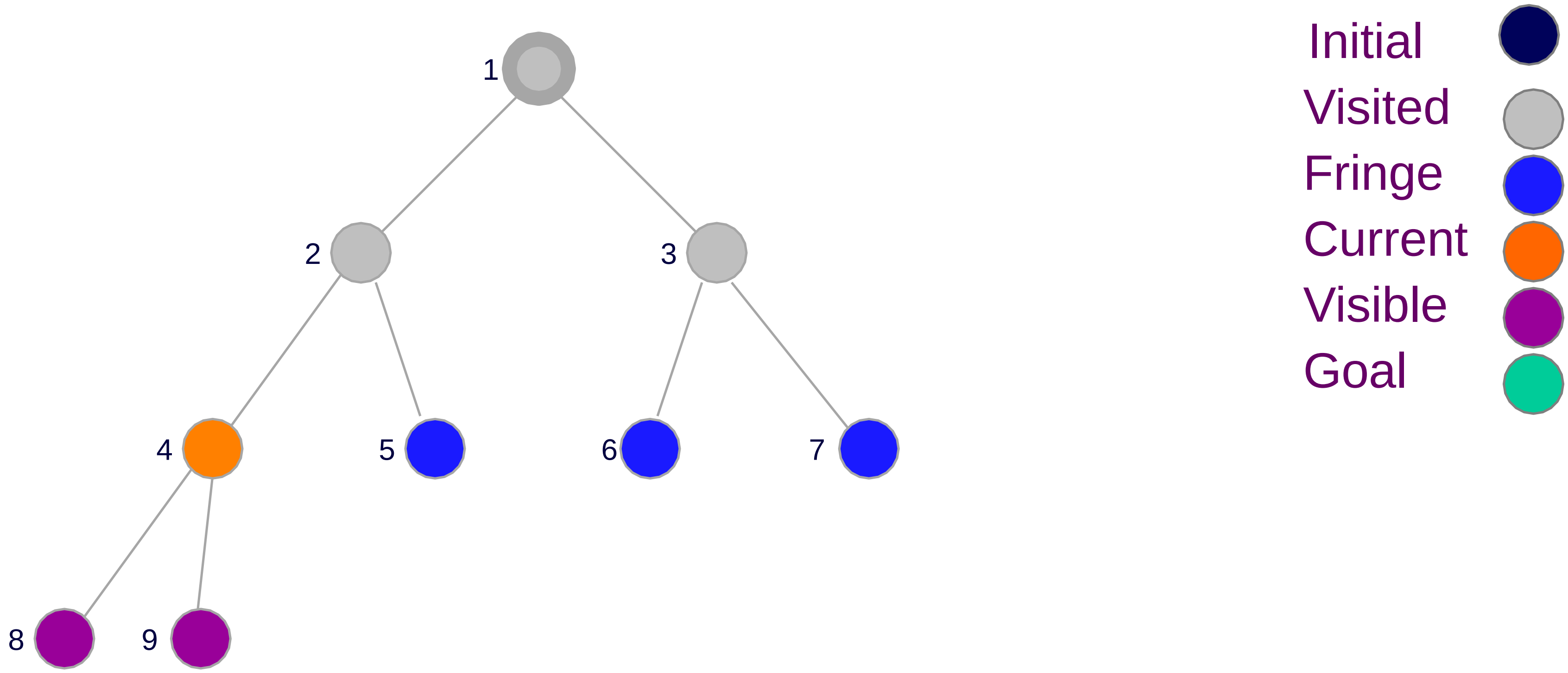
Breadth-First Snapshot 3





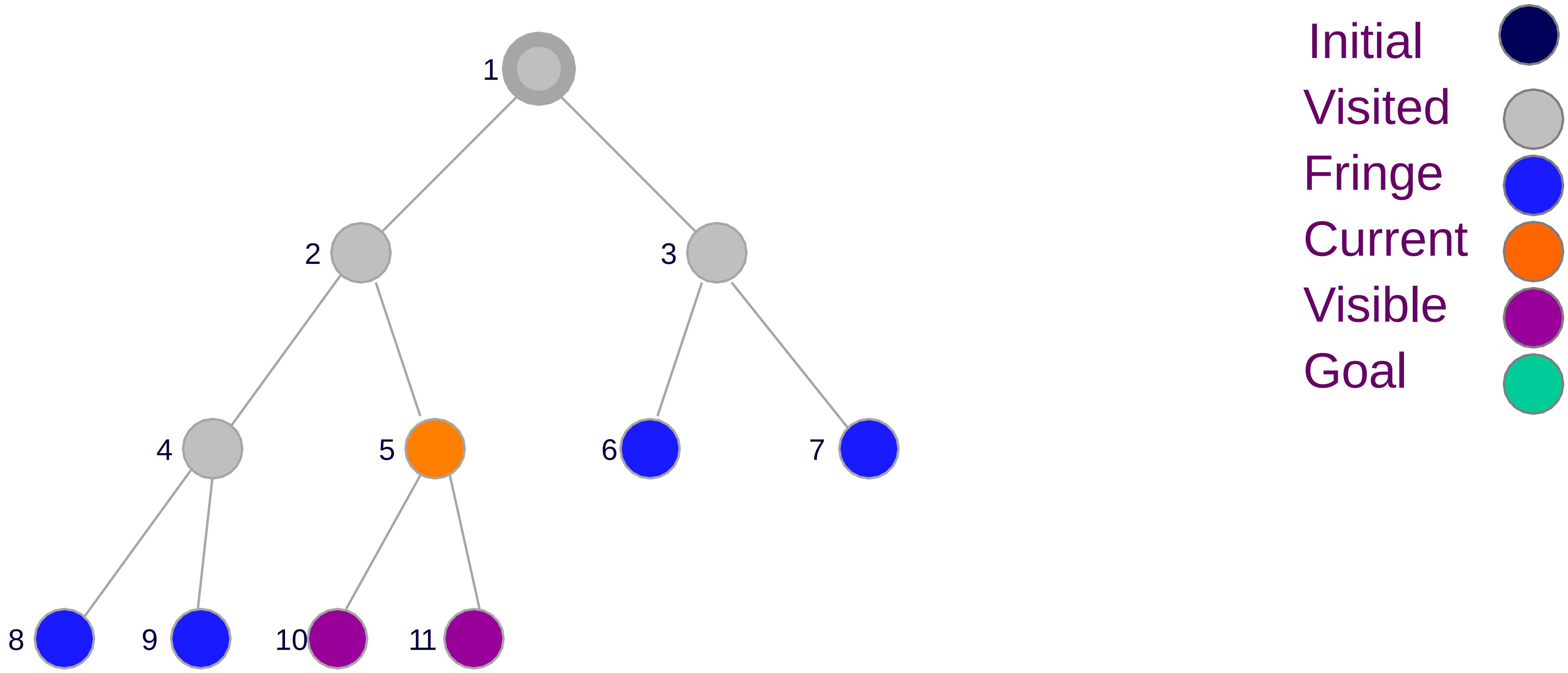
Fringe: [4,5] + [6,7]

Breadth-First Snapshot 4



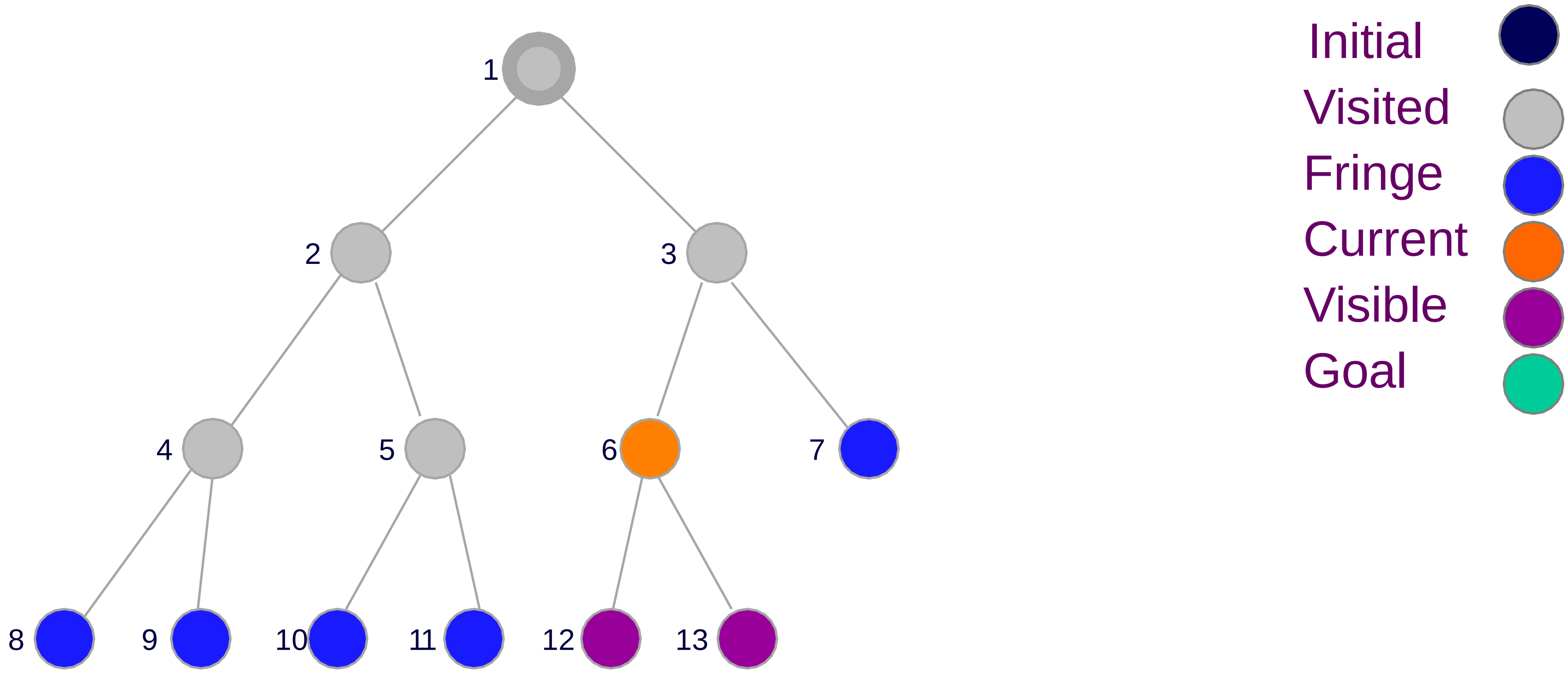
Fringe: [5,6,7] + [8,9]

Breadth-First Snapshot 5



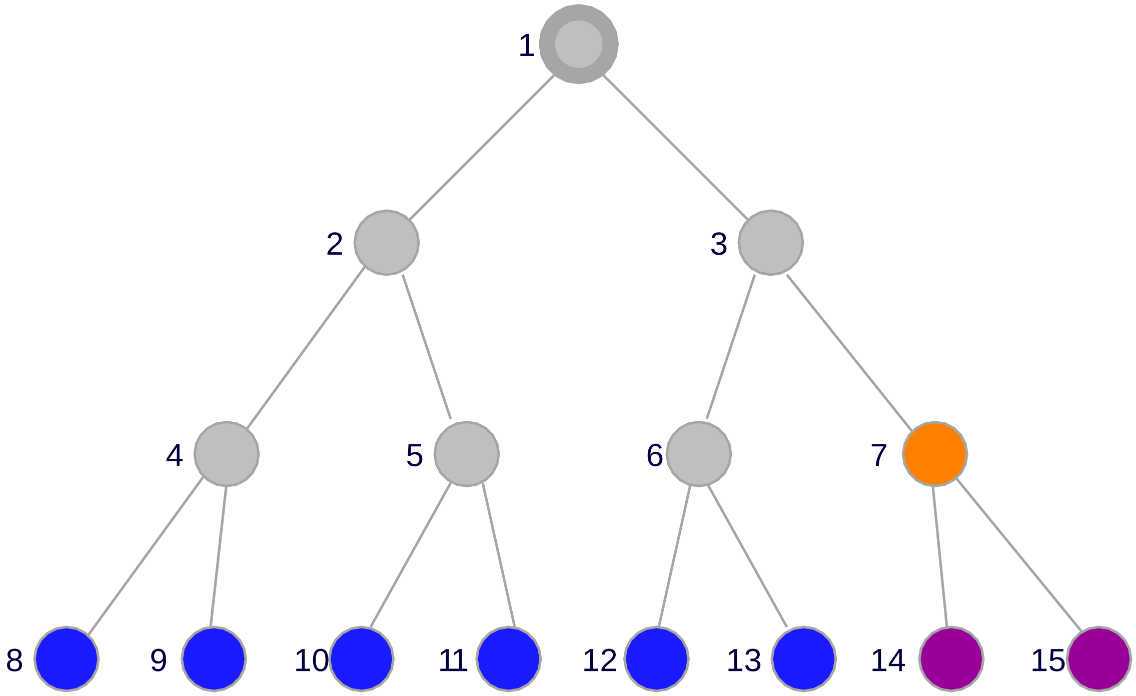
Fringe: [6,7,8,9] + [10,11]

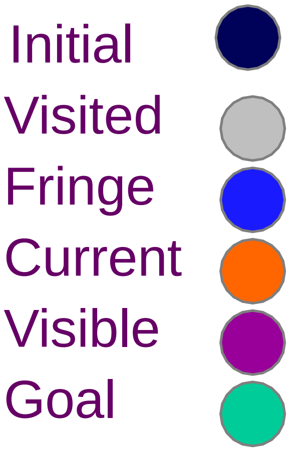
Breadth-First Snapshot 6



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

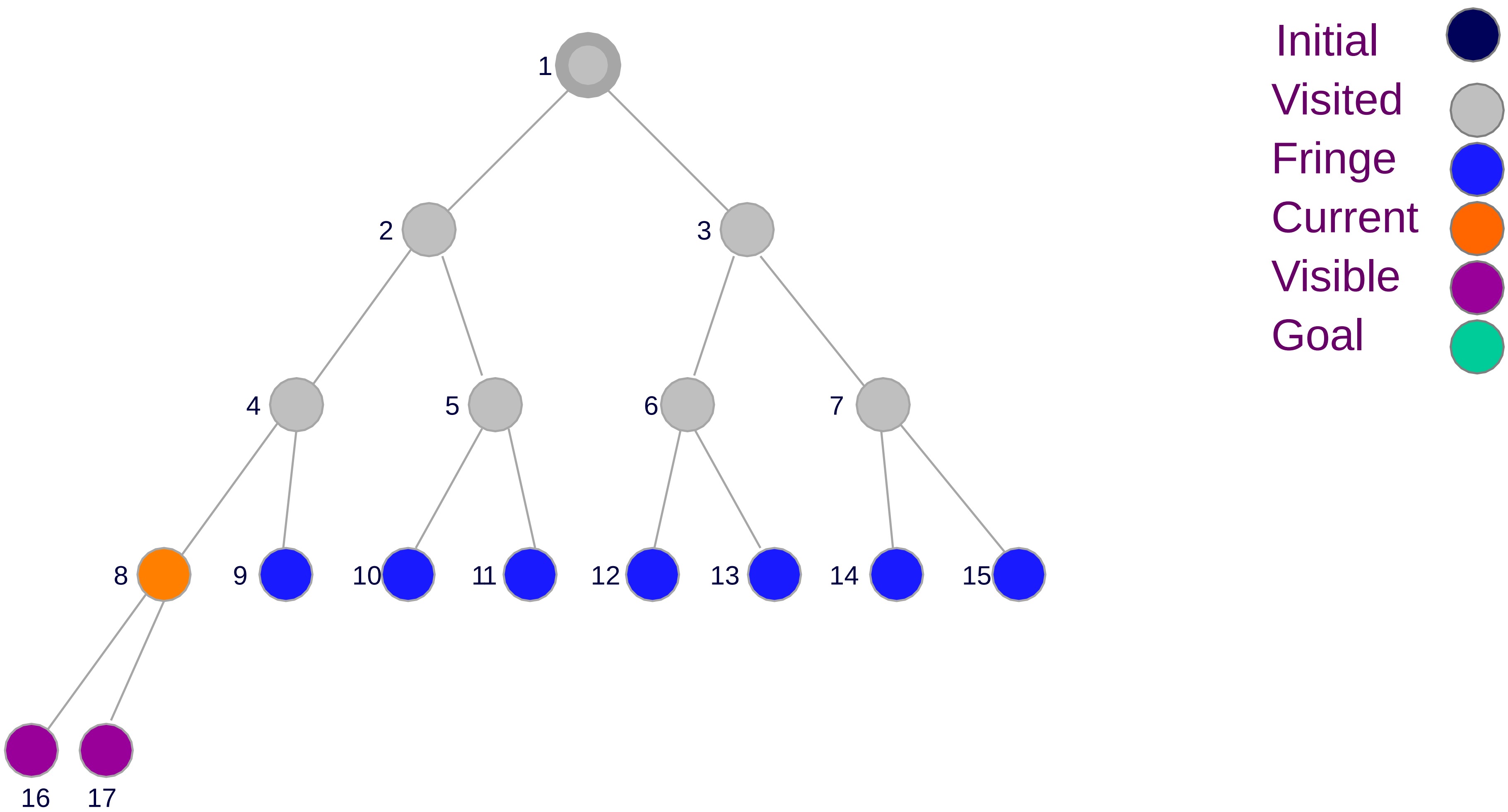
Breadth-First Snapshot 7





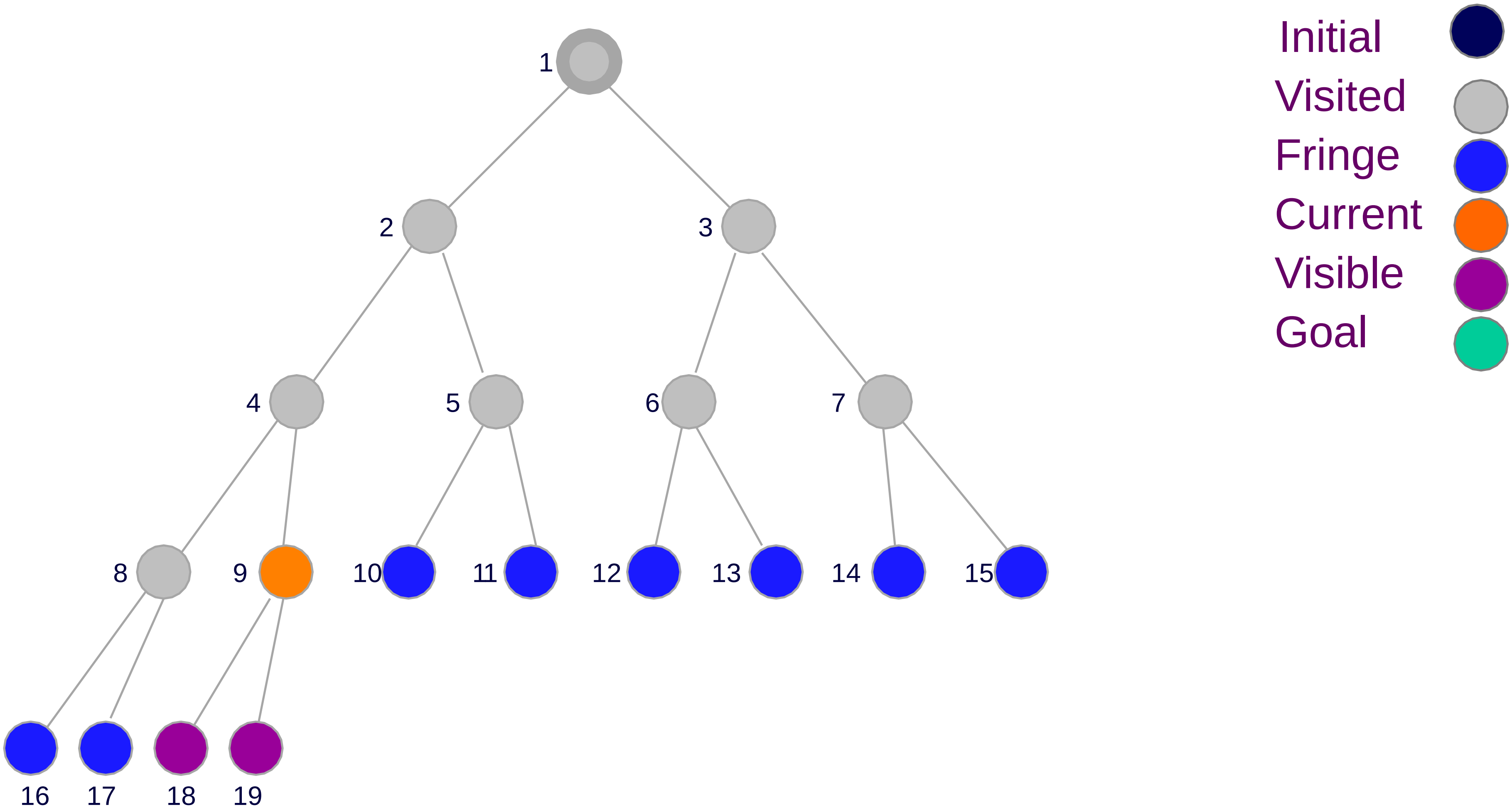
Fringe: [8,9,10,11,12,13] + [14,15]

Breadth-First Snapshot 8



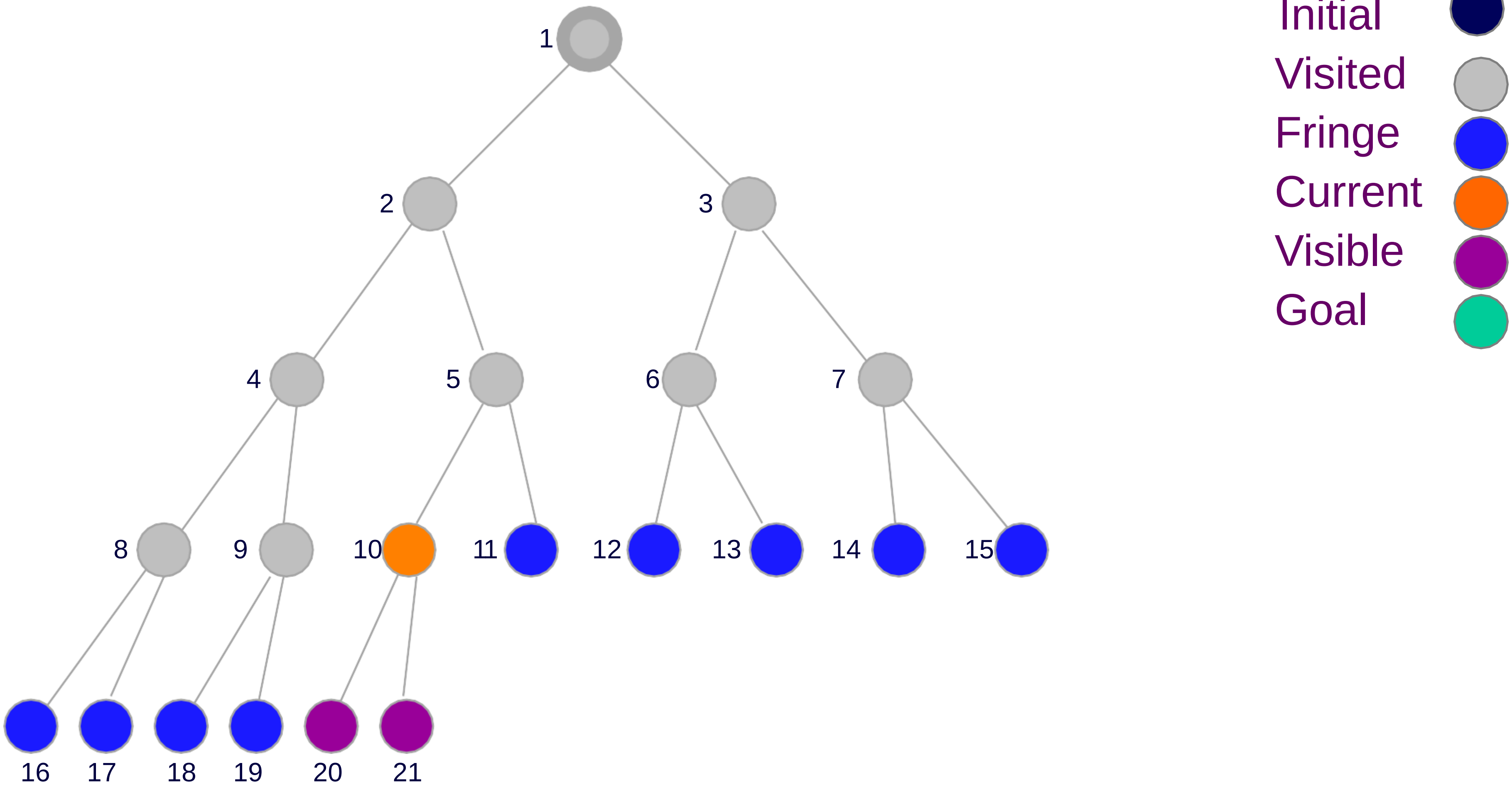
Fringe: [9,10,11,12,13,14,15] + [16,17]

Breadth-First Snapshot 9



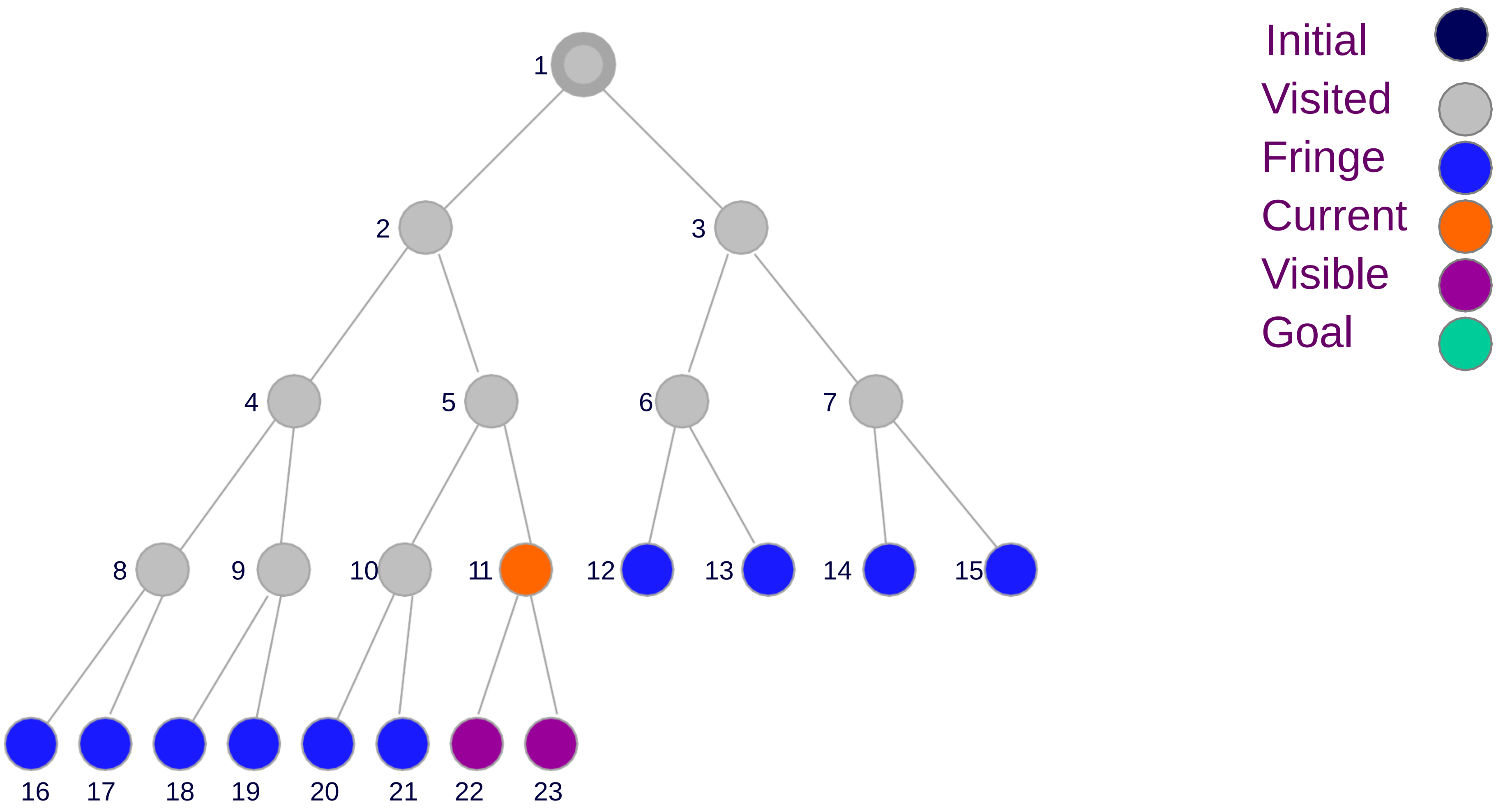
Fringe: [10,11,12,13,14,15,16,17] + [18,19]

Breadth-First Snapshot 10



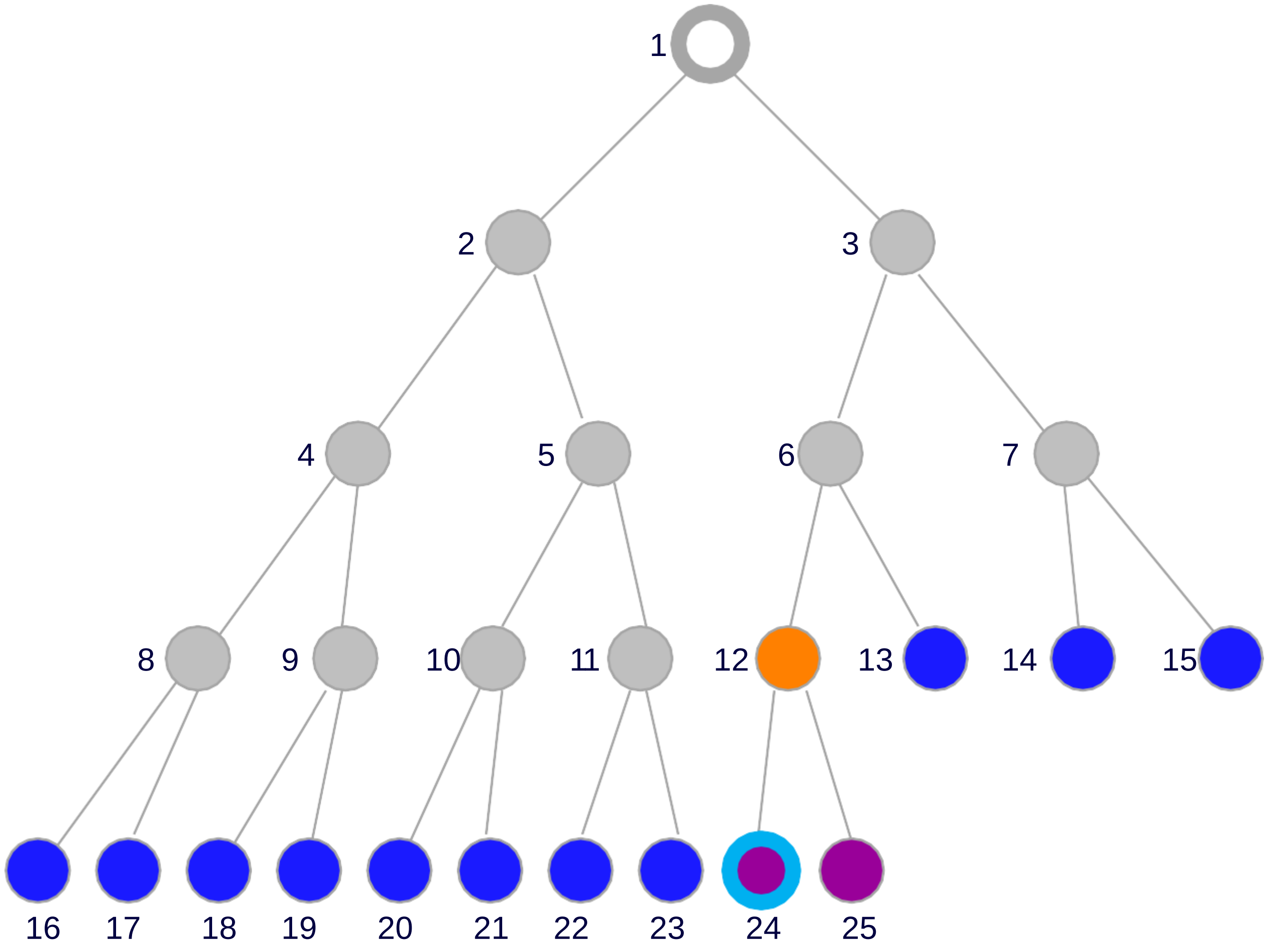
Fringe: [11,12,13,14,15,16,17,18,19] + [20,21]

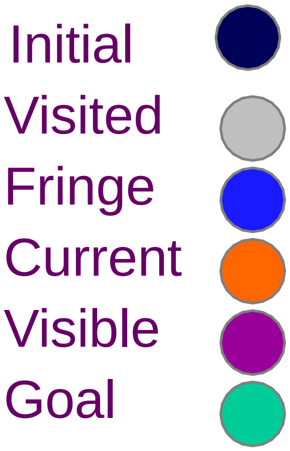
Breadth-First Snapshot 11



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

Breadth-First Snapshot 12

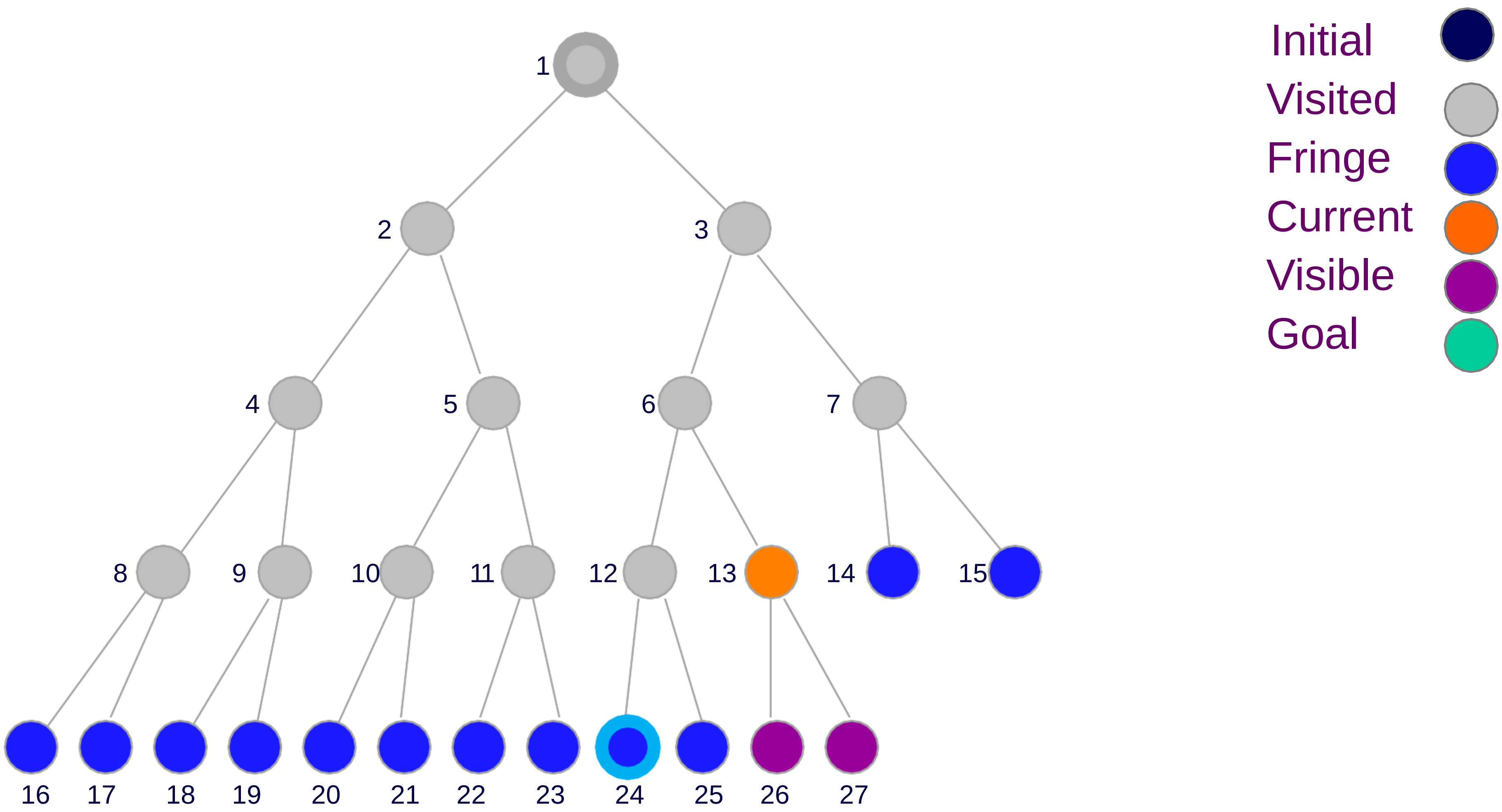




Note:
The goal node is "visible" here, but we can not perform the goal test yet.

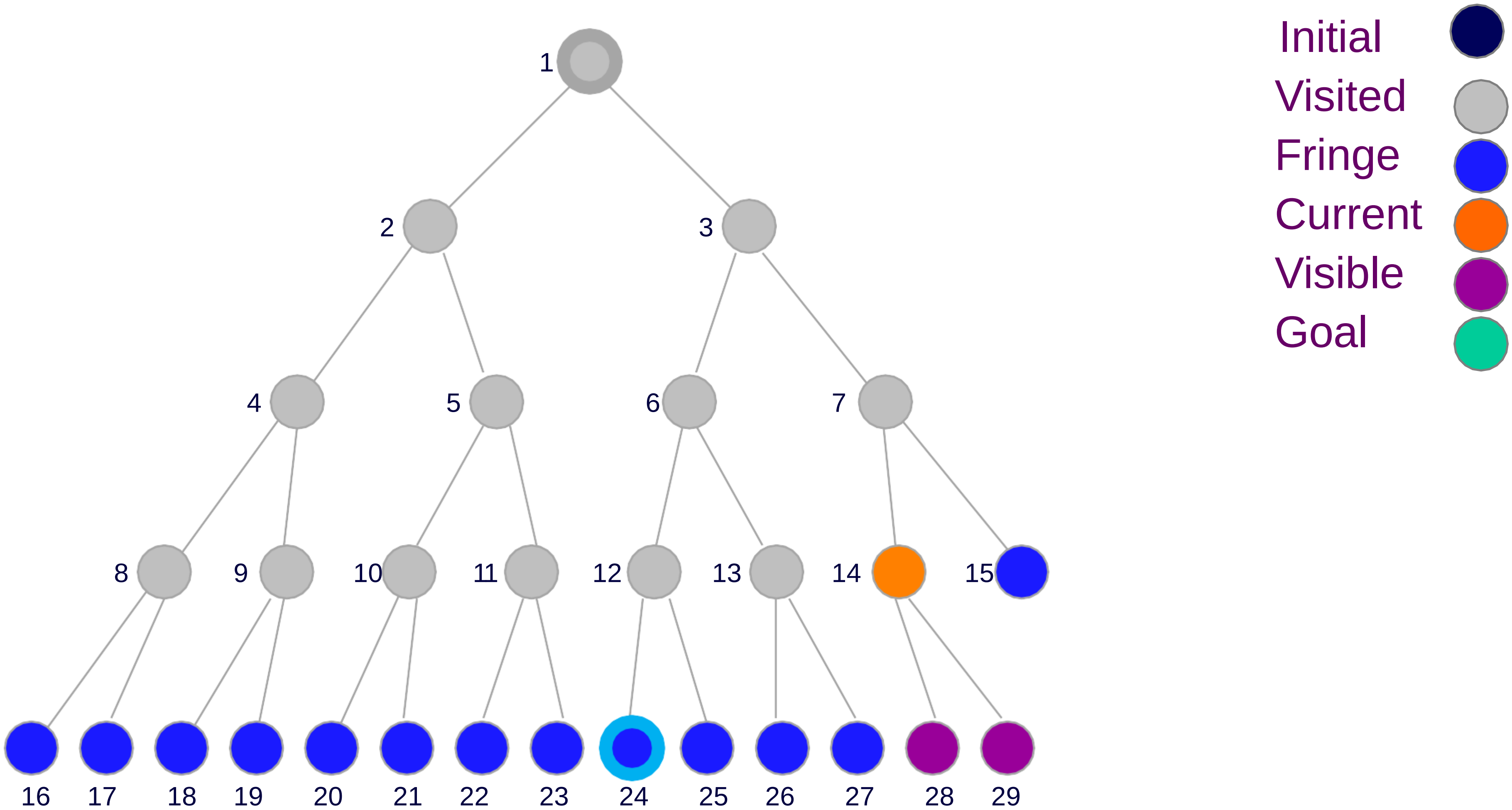
Fringe: [13,14,15,16,17,18,19,20,21] + [22,23]

Breadth-First Snapshot 13



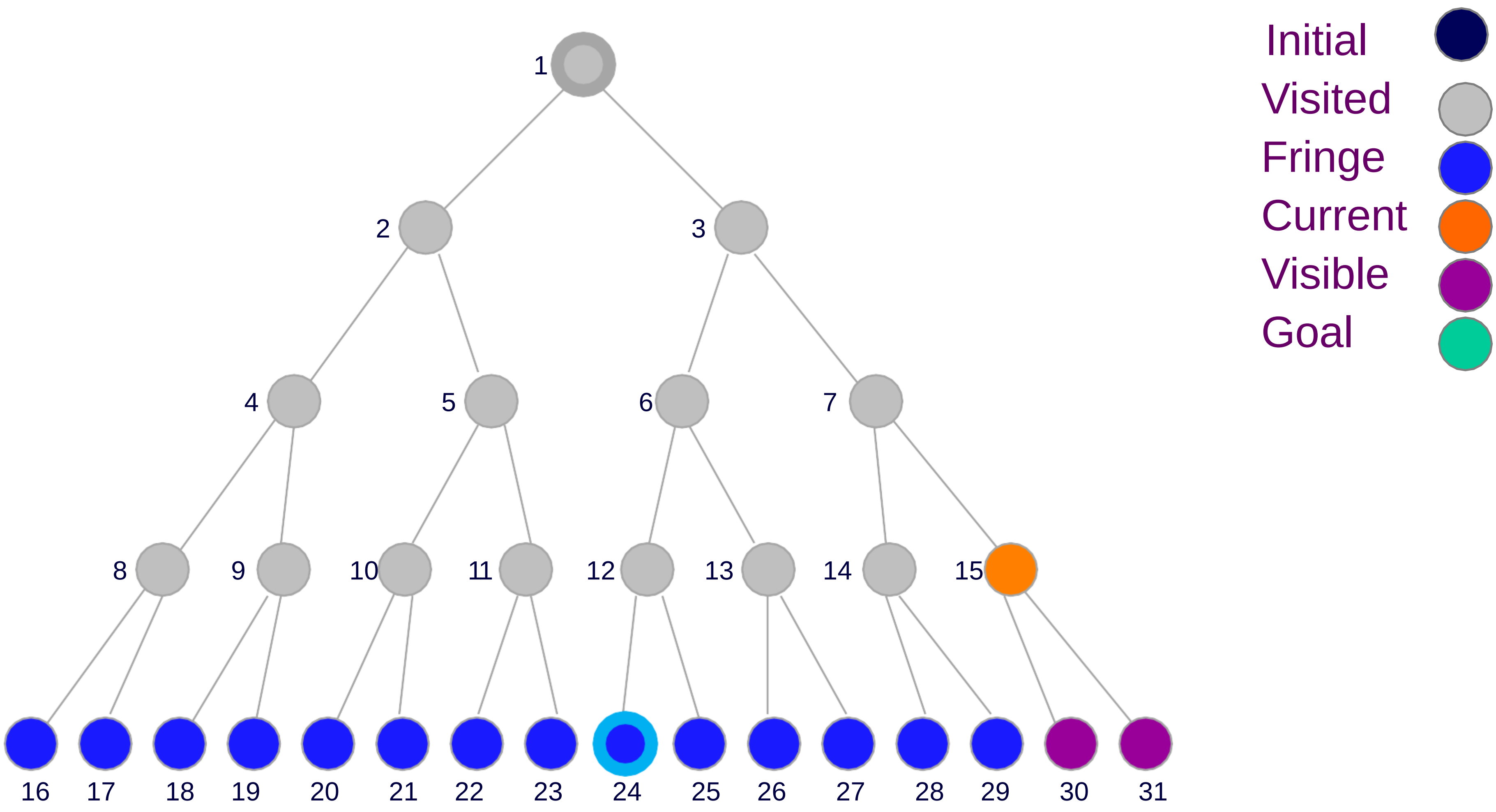
Fringe: [14,15,16,17,18,19,20,21,22,23,24,25] + [26,27]

Breadth-First Snapshot 14



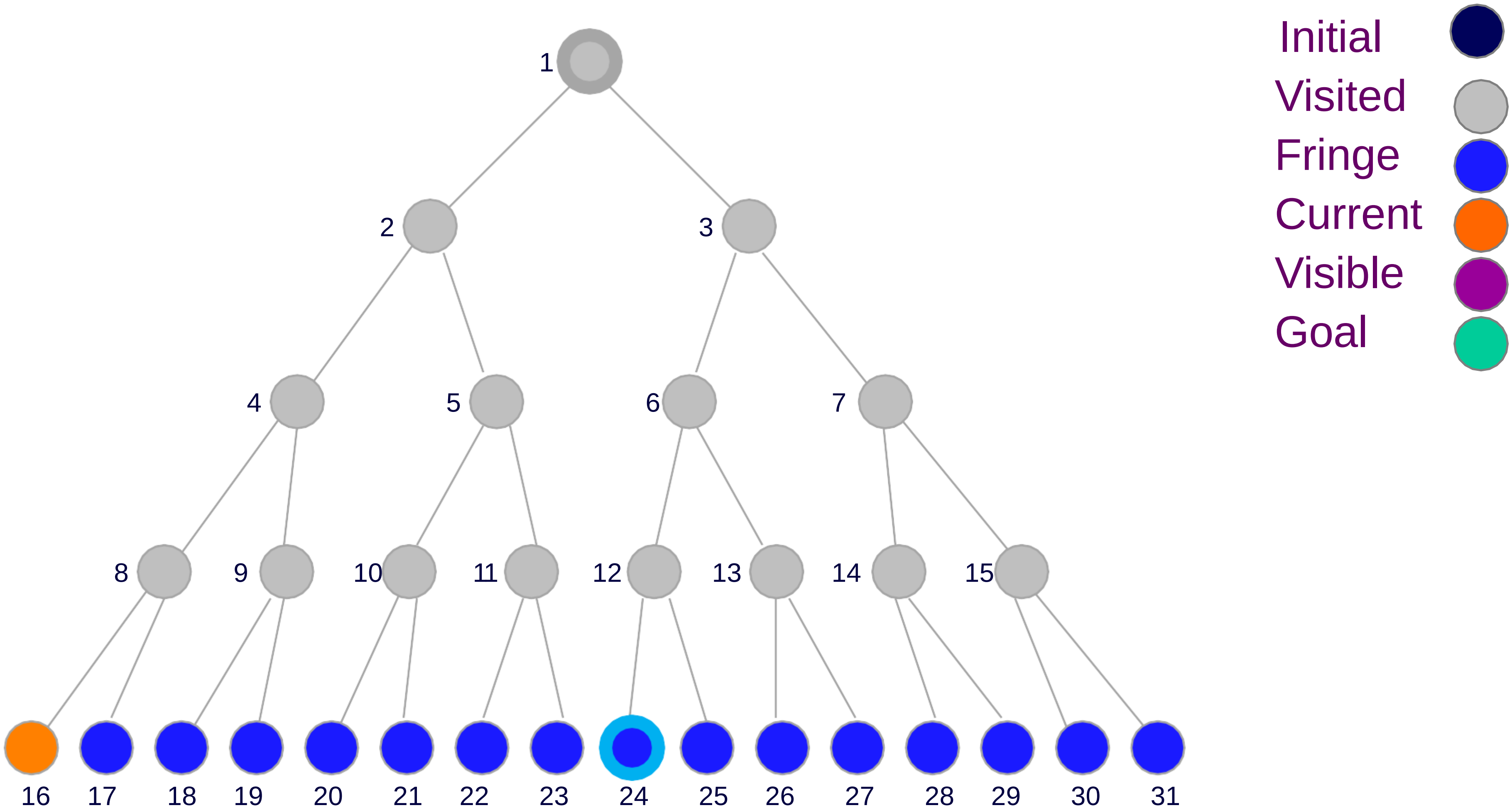
Fringe: [15,16,17,18,19,20,21,22,23,24,25,26,27] + [28,29]

Breadth-First Snapshot 15



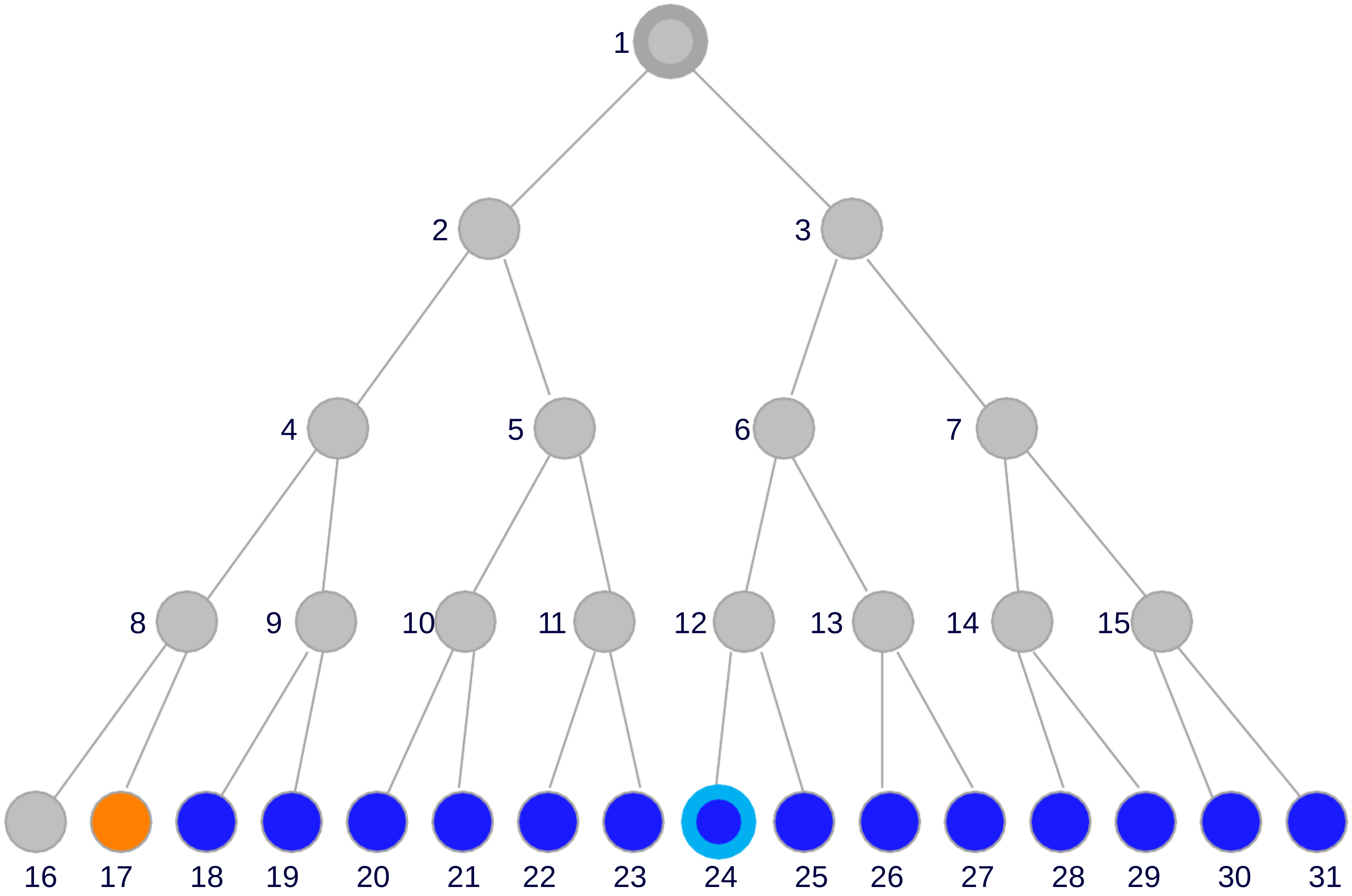
Fringe: [15,16,17,18,19,20,21,22,23,24,25,26,27,28,29] + [30,31]

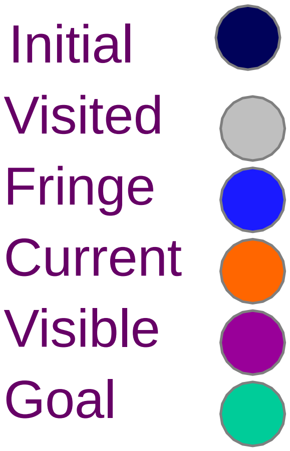
Breadth-First Snapshot 16



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

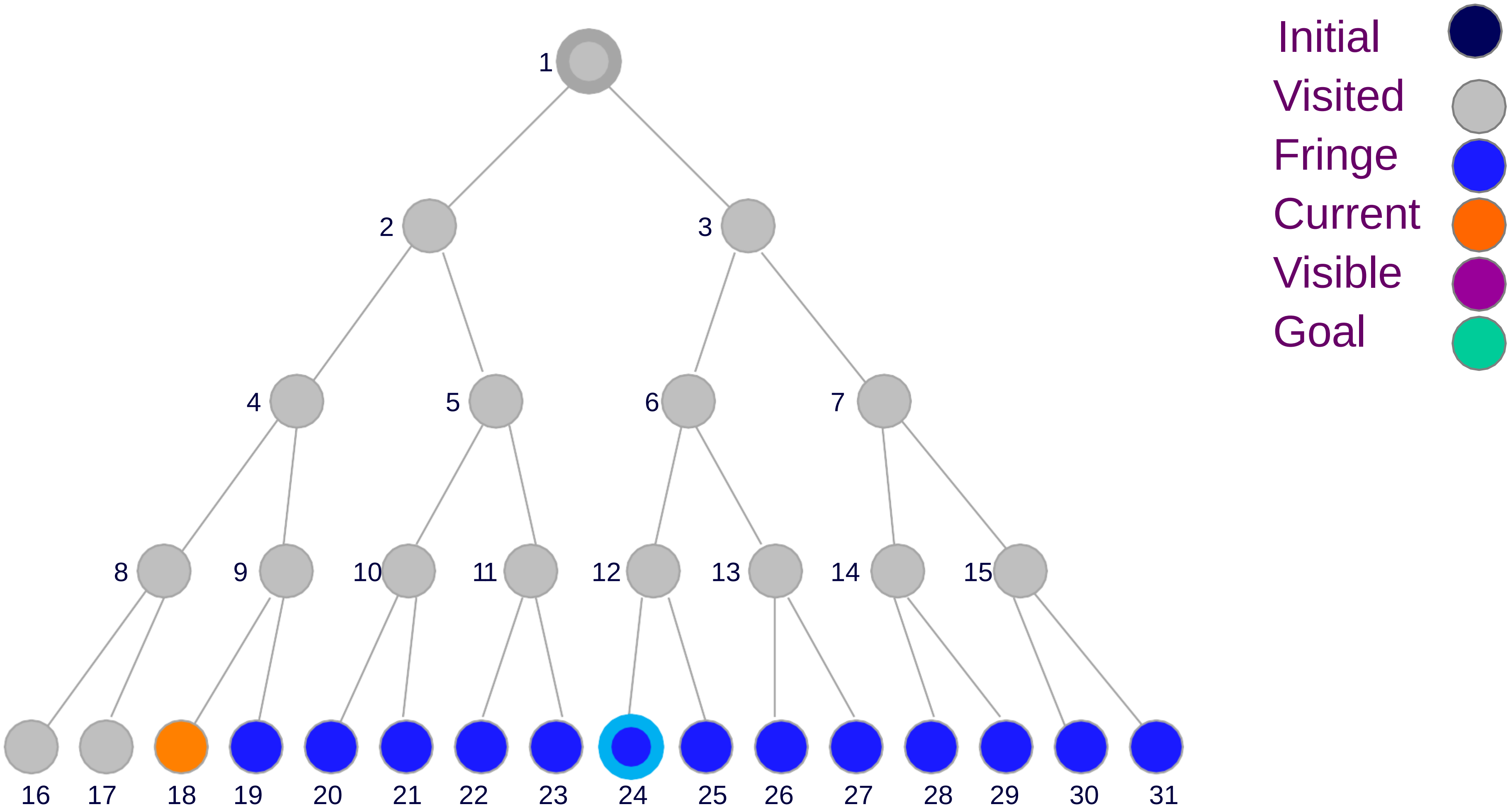
Breadth-First Snapshot 17





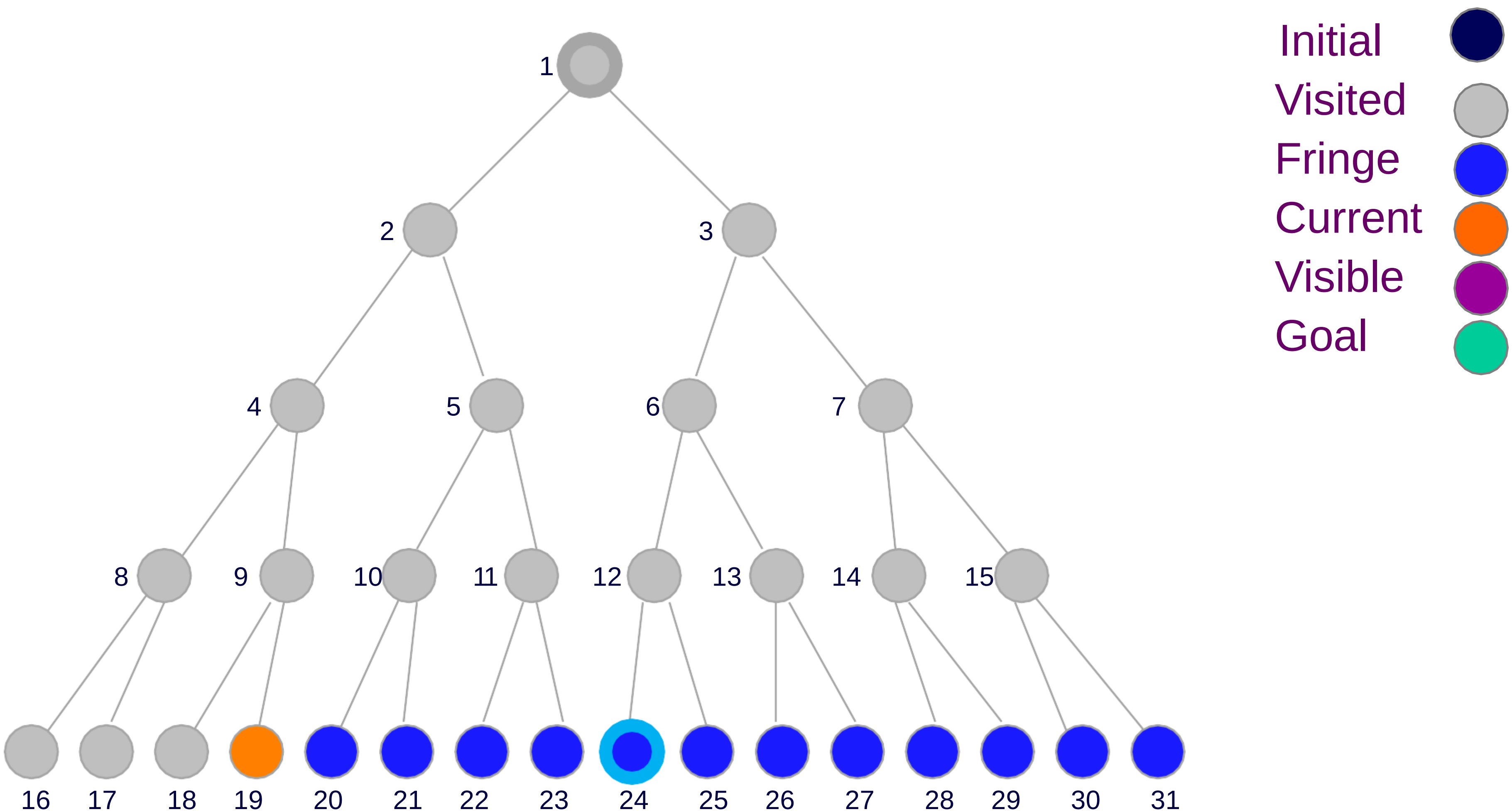
Fringe: [18,19,20,21,22,23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 18



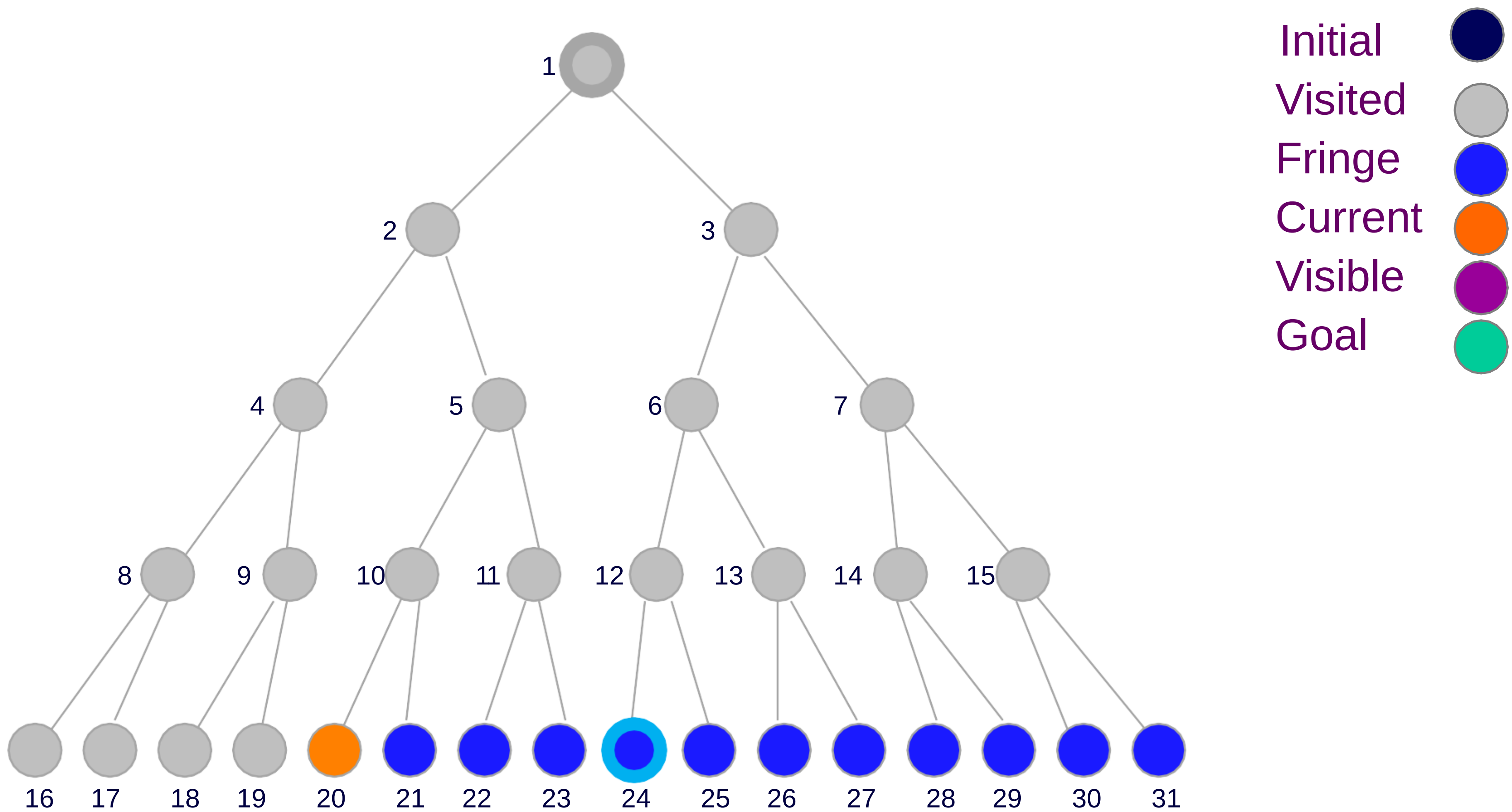
Fringe: [19,20,21,22,23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 19



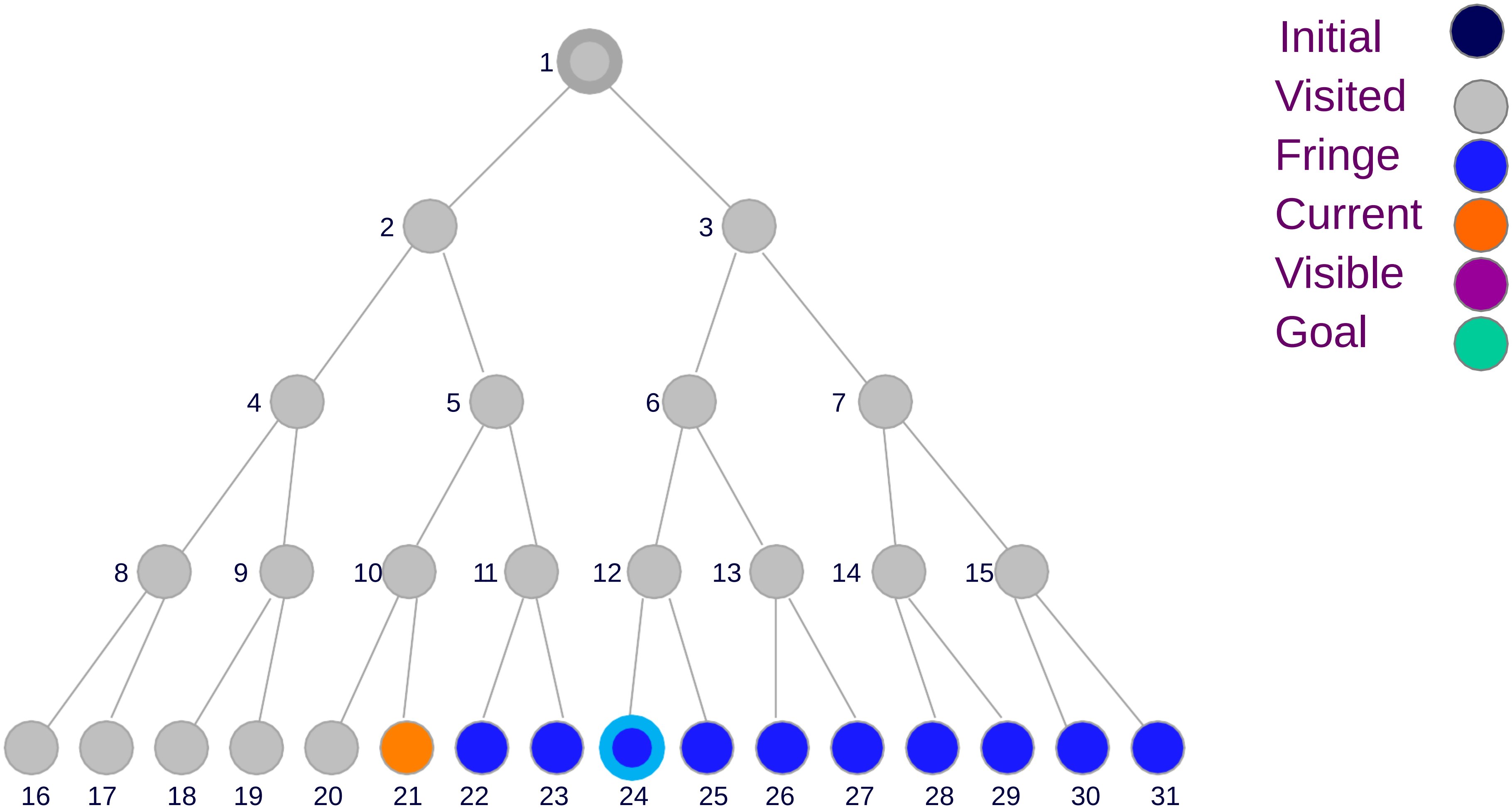
Fringe: [20,21,22,23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 20



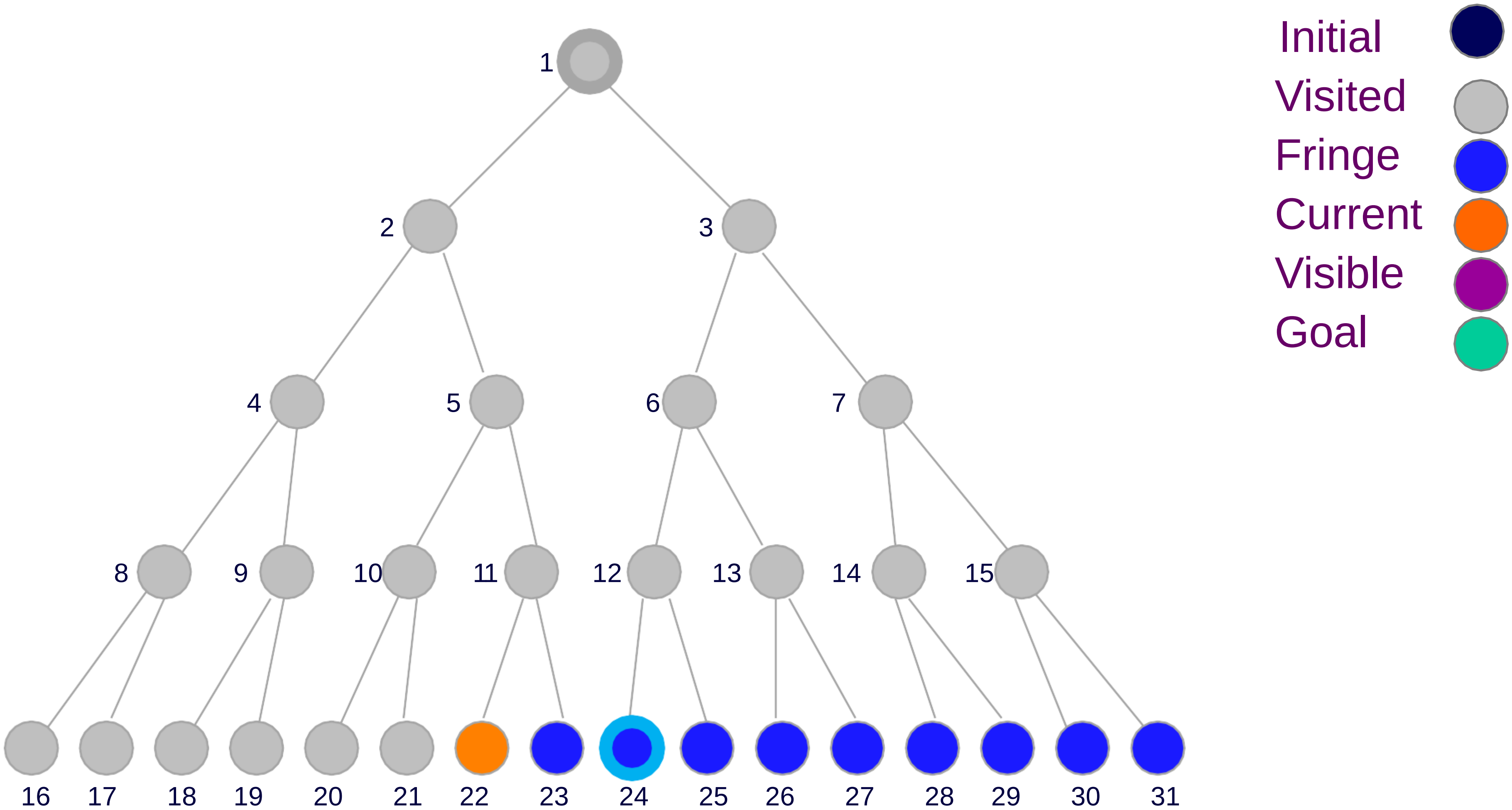
Fringe: [21,22,23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 21



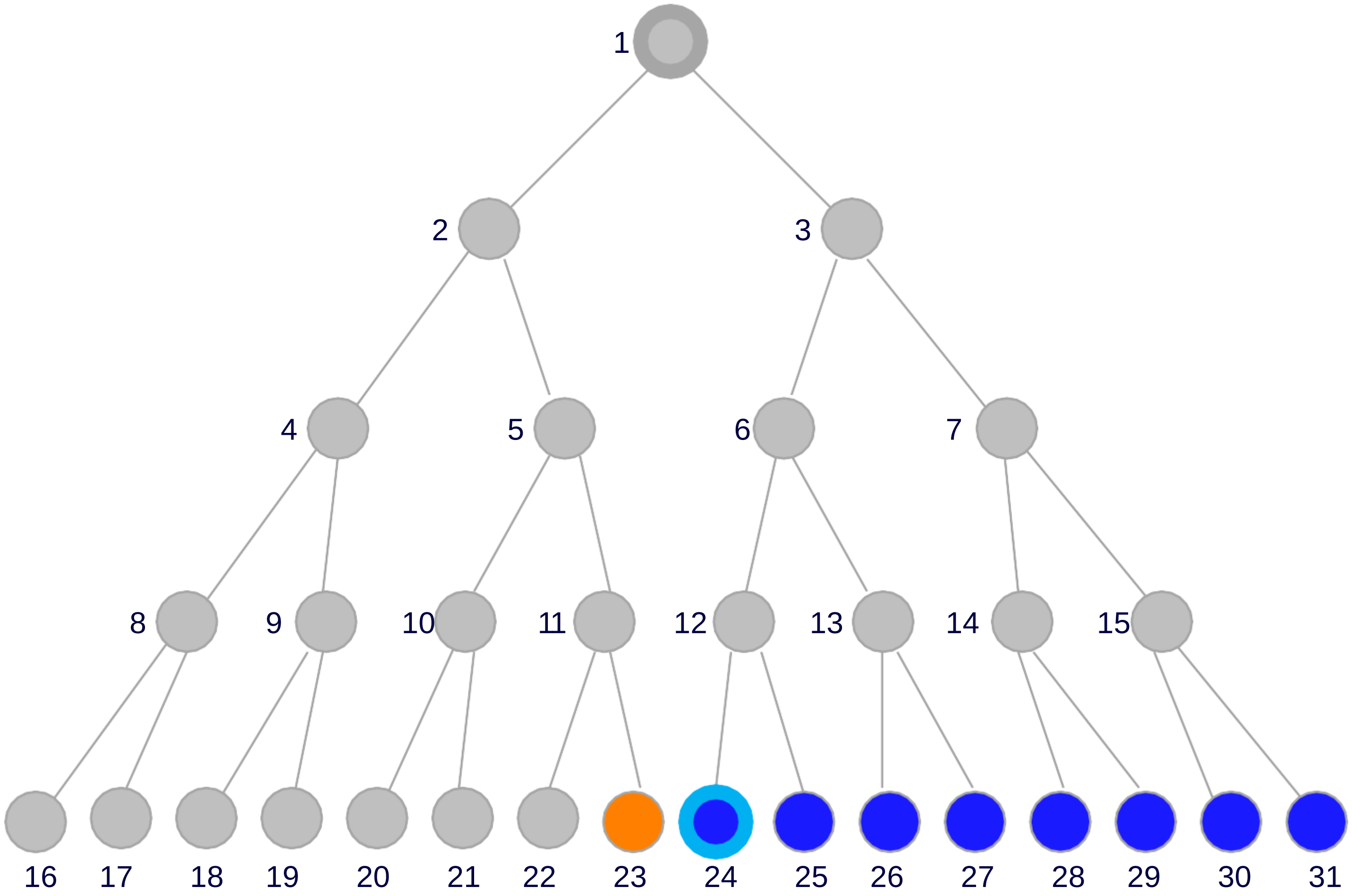
Fringe: [22,23,24,25,26,27,28,29,30,31]

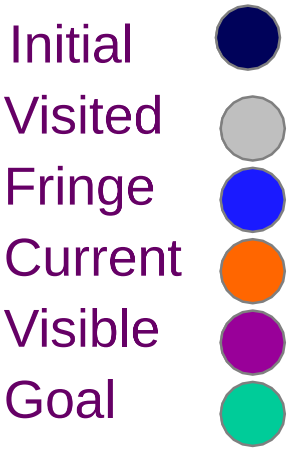
Breadth-First Snapshot 22



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

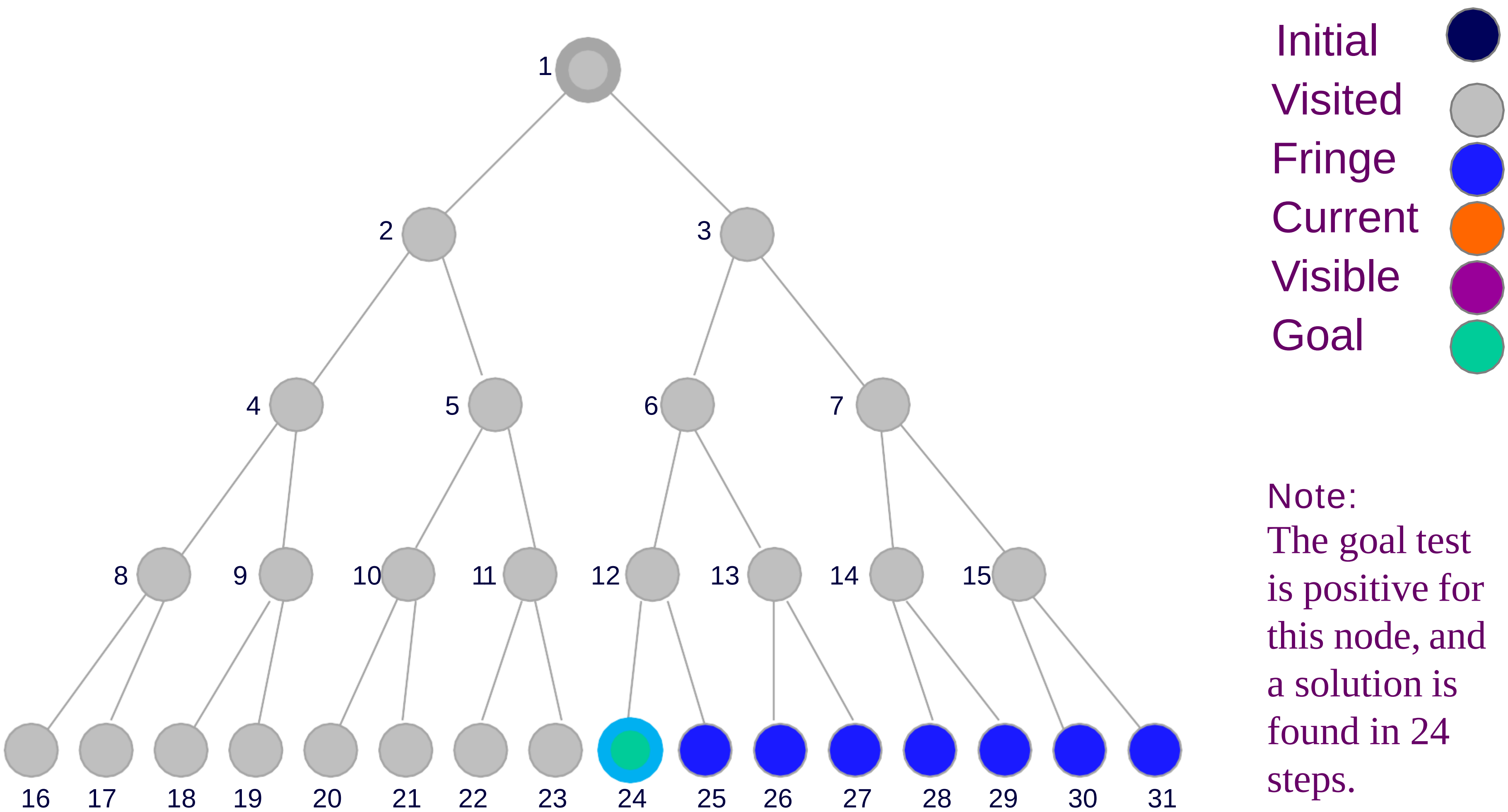
Breadth-First Snapshot 23





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

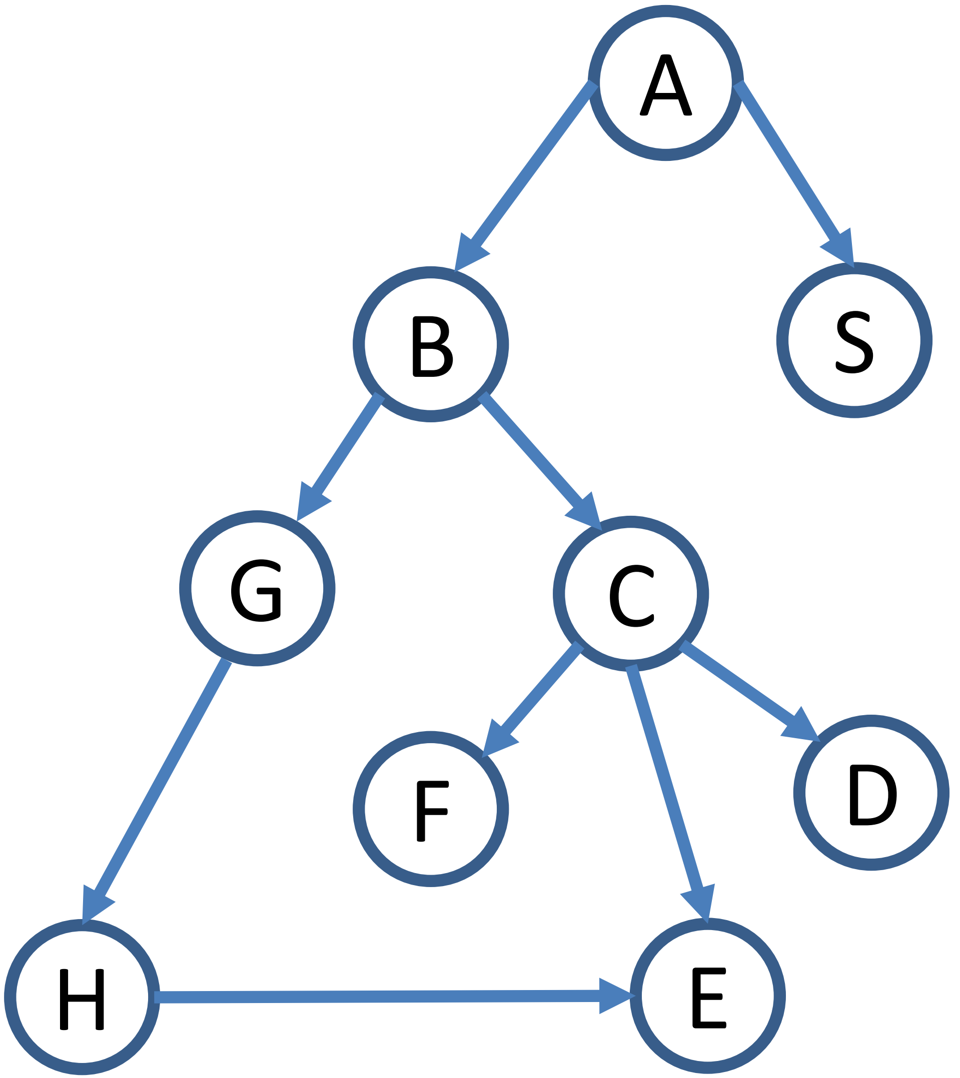
Breadth-First Snapshot 24



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

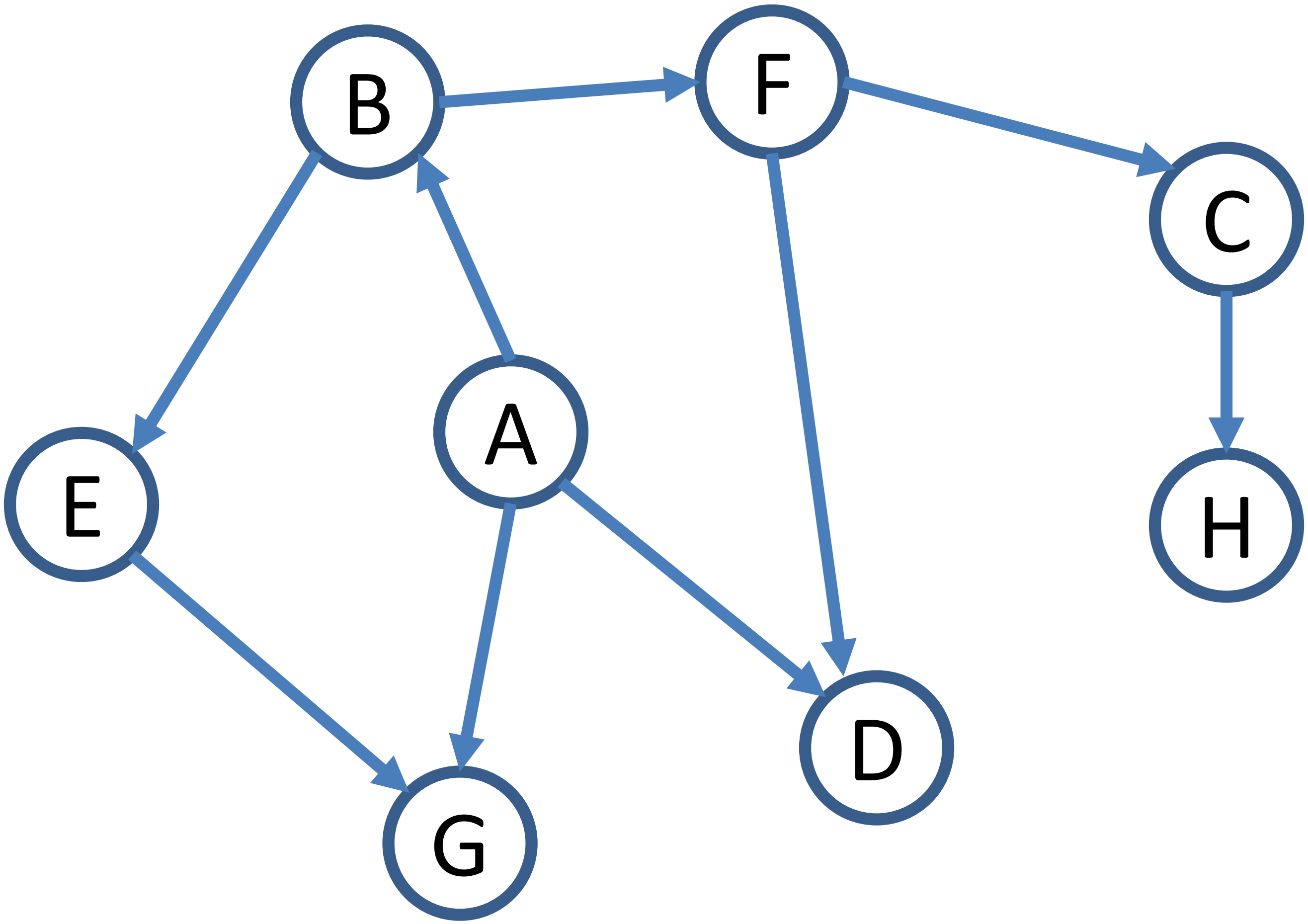
Fringe: [25,26,27,28,29,30,31]

Example BFS



Solve using BFS

ABDGEFCH



Properties of Breadth-First Search (BFS)

Completeness: Yes (if *b* is finite), a solution will be found if exists.

Time Complexity: (nodes until the solution)

Optimality: Yes

Criterion	Breadth-First
Complete?	Yes
Time	$O(b^{d+1})$
Space	$O(b^{d+1})$
Optimal?	Yes

b	Branching Factor
d	The depth of the goal

Suppose the branching factor b=10, and the goal is at depth d=12:

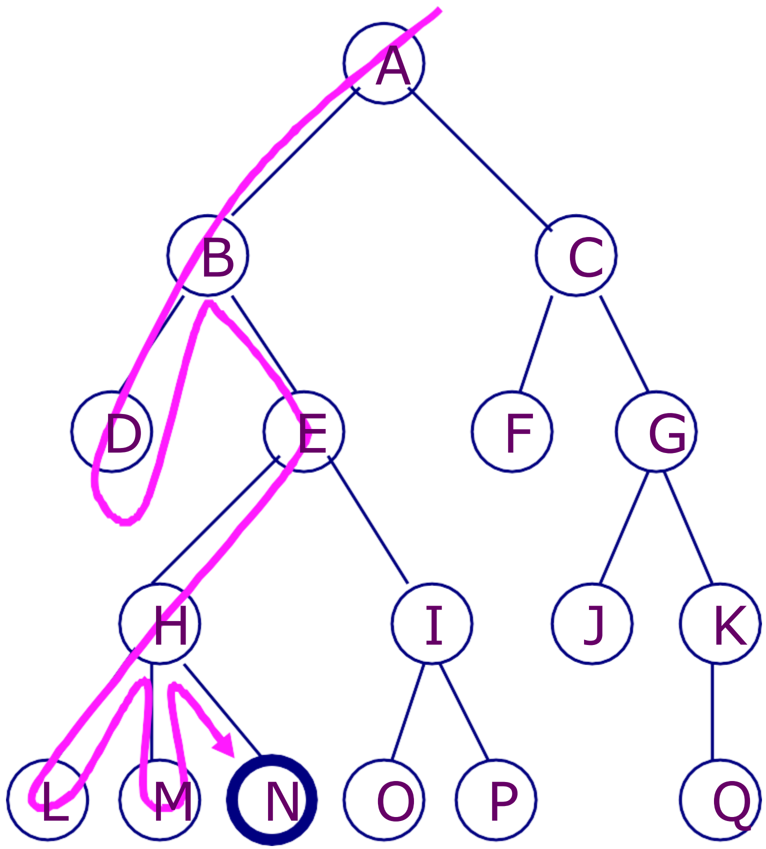
– Then we need $O10^{12}$ time to finish. If O is 0.001 second, then we need 1 billion seconds (31 year). And if each O costs 10 bytes to store, then we also need 1 terabytes.

→Not suitable for searching large graphs

3- Depth-First Search

Depth-First Search

Based on [4]



A depth-first search (DFS) explores a path all the way to a leaf before backtracking and exploring another path.

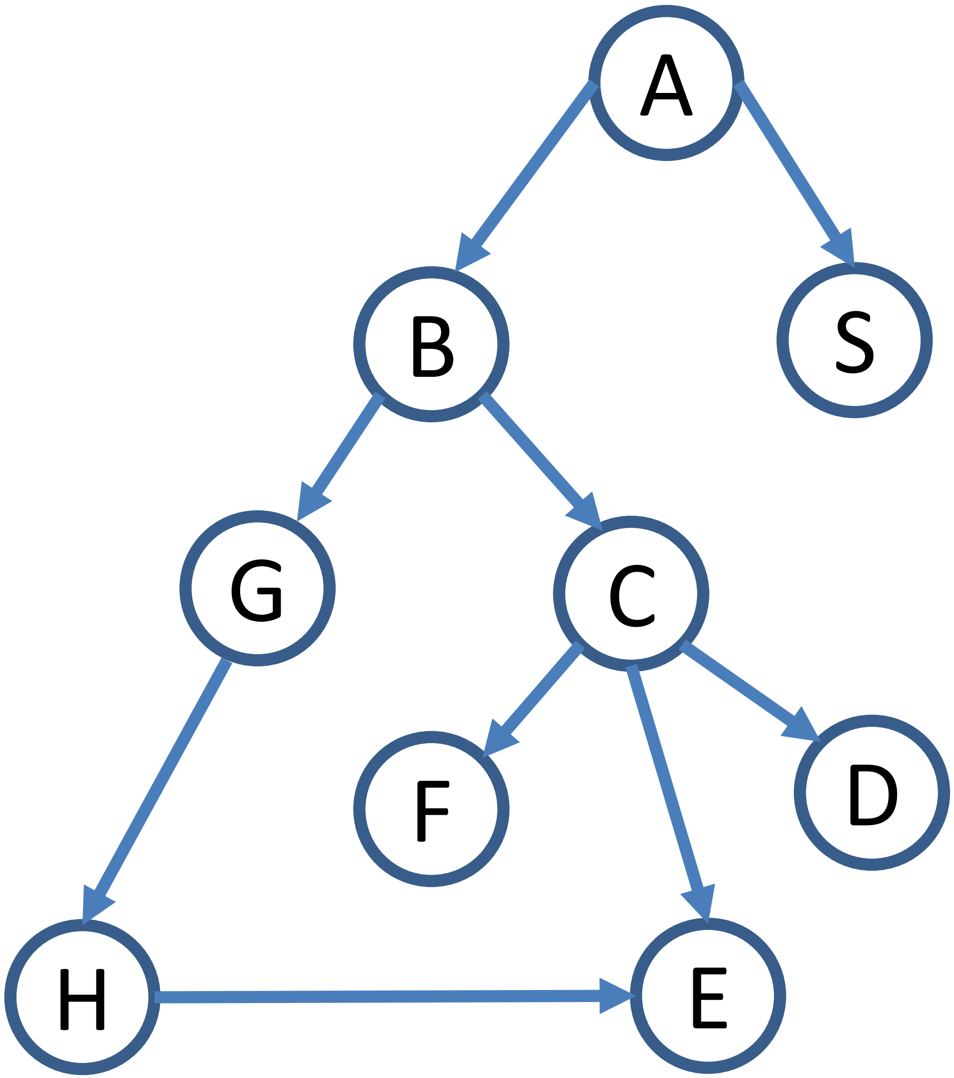
For example, after searching A, then B, then D, the search backtracks and tries another path from B.

Node are explored in the order A B D E H L M N I O P C F G J K Q

Depth-First Search

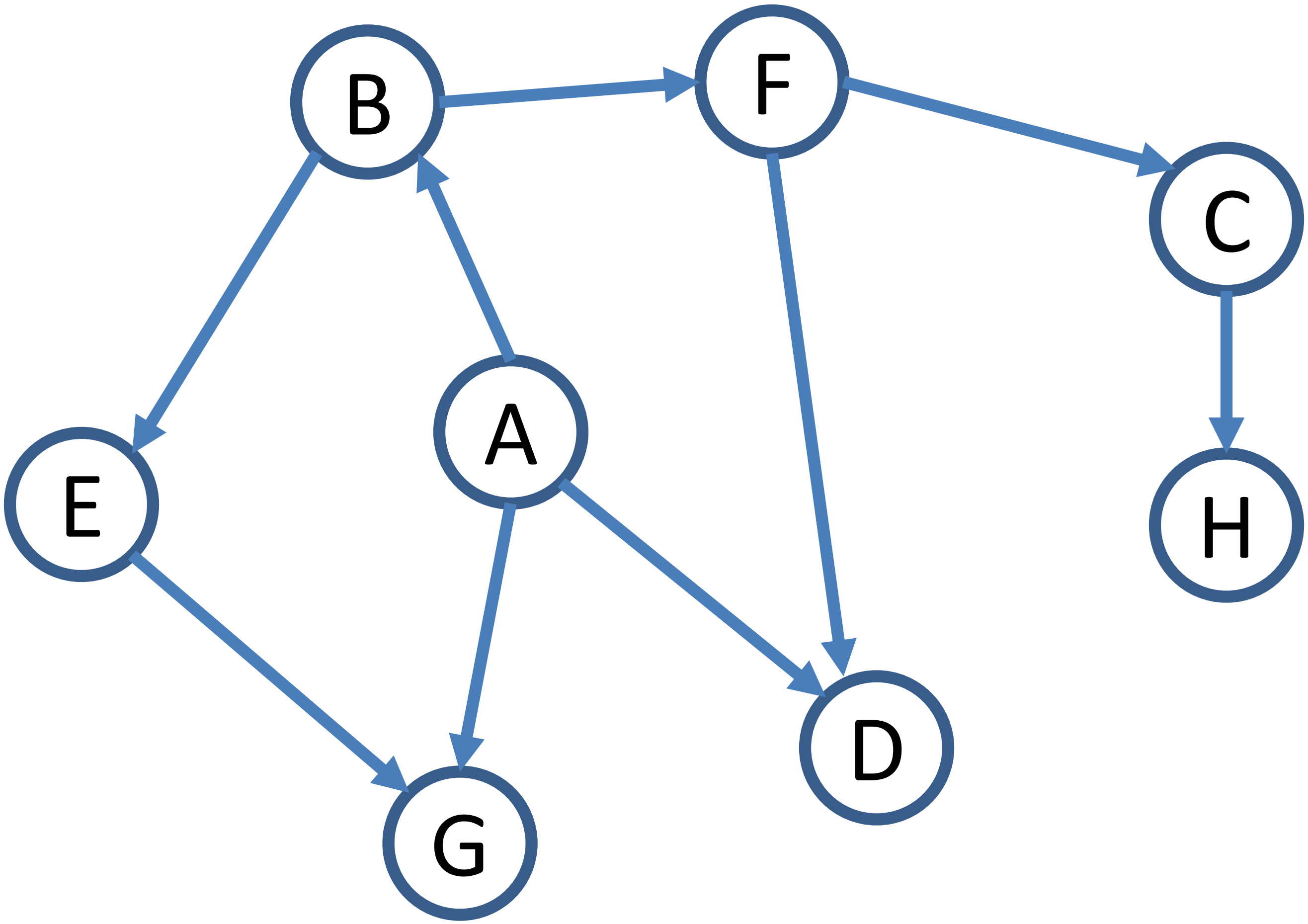
1. Start
2. Push Root Node to Stack
   - Mark Root Node as Visited
   - Print Root Node as Output
3. Check Top of the Stack If Stack is Empty, Go to Step 6
4. Else, Check Adjacent Top of the Stack
   - If Adjacent is not Visited
   - Push Node to Stack
   - Mark Node as Visited
   - Print Node as Output
   - Else Adjacent Visited
5. Go to Step 3
6. Stop

Example DFS



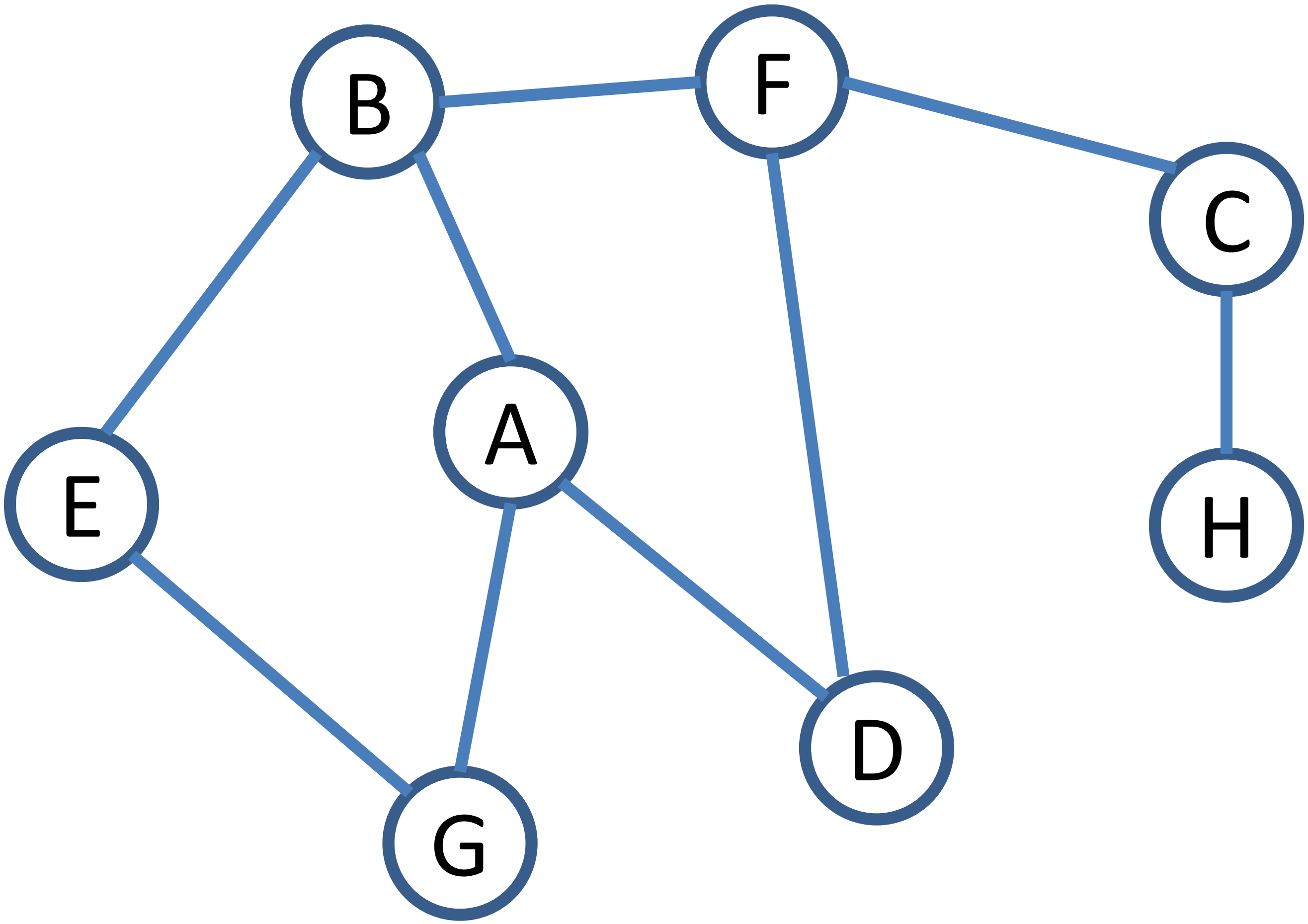
Example DFS

ABEGFCHD



Example DFS

AGEDFCHB



Properties of Depth-First Search

Complete: No: fails in infinite-depth spaces, spaces with loops
– Yes, complete in finite spaces

Time: $O(b^m)$: terrible if m is much larger than d
– but if solutions are dense, may be much faster than breadth-first

Space: $O(bm)$

Optimal: No

Criterion	Depth-First
Complete?	No
Time	$O(b^m)$
Space	$O(bm)$
Optimal?	No

b: maximum branching factor of the search tree
d: depth of the least-cost solution
m: maximum depth of the state space (may be ∞)

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
  - Much fewer visited nodes to keep track, smaller fringe

Breadth-first is more careful by checking all alternatives

- Complete and optimal (Under most circumstances)
- Very memory-intensive

For a large tree, breadth-first search memory requirements maybe excessive

For a large tree, a depth-first search may take an excessively long time to find even a very nearby goal node.

➔ How can we combine the advantages (and avoid the disadvantages) of these two search techniques?