Logistics

- AI Nugget presentations
  - Section 1: Thomas Soria, Gagandeep Singh Kohli, Alex Ledwith
  - Section 3: Martin Silverio
- Project Team Wikis, pages
  - project description refined:
    - Features, Requirements, Schedule
- PolyLearn: Does everybody have access?
  - groups set up for project teams; some team membership info incomplete
- Lab and Homework Assignments
  - Lab 2 due tonight (23:59); demos during the lab time Tue, Thu
  - Lab 3 available: breadth-first, depth-first search
- Quizzes
  - Quiz 2 - Available Tue, Sep. 24, all day (0:00 - 23:59)
  - Make-up question details TBA soon; most likely
    - submission formats: Web form, tab-delimited, NL-delimited
    - 2 submitted questions are the equivalent of one quiz questions (10 points)

Chapter Overview
Search

- Motivation
- Objectives
- Search as Problem-Solving
  - problem formulation
  - problem types
- Uninformed Search
  - breadth-first
  - depth-first
  - uniform-cost search
  - depth-limited search
  - iterative deepening
  - bi-directional search
- Informed Search
  - best-first search
  - search with heuristics
  - memory-bounded search
  - iterative improvement search
- Non-Traditional Search
  - local search and optimization
  - constraint satisfaction
  - search in continuous spaces
  - partially observable worlds
- Important Concepts and Terms
- Chapter Summary

Examples

- getting from home to Cal Poly
  - start: home on Clearview Lane
  - goal: Cal Poly CSC Dept.
  - operators: move one block, turn
- loading a moving truck
  - start: apartment full of boxes and furniture
  - goal: empty apartment, all boxes and furniture in the truck
  - operators: select item, carry item from apartment to truck, load item
- getting settled
  - start: items randomly distributed over the place
  - goal: satisfactory arrangement of items
  - operators: select item, move item

Motivation

- search strategies are important methods for many approaches to problem-solving
- the use of search requires an abstract formulation of the problem and the available steps to construct solutions
- search algorithms are the basis for many optimization and planning methods

Objectives

- formulate appropriate problems as search tasks
  - states, initial state, goal state, successor functions (operators), cost
- know the fundamental search strategies and algorithms
  - uninformed search
    - breadth-first, depth-first, uniform-cost, iterative deepening, bi-directional
  - informed search
    - best-first (greedy, A*), heuristics, memory-bounded, iterative improvement
- evaluate the suitability of a search strategy for a problem
  - completeness, time & space complexity, optimality

Problem-Solving Agents

- agents whose task it is to solve a particular problem
  - goal formulation
    - what is the goal state
    - what are important characteristics of the goal state
    - how does the agent know that it has reached the goal
    - are there several possible goal states
      - are they equal or are some more preferable
  - problem formulation
    - what are the possible states of the world relevant for solving the problem
    - what information is accessible to the agent
    - how can the agent progress from state to state

Problem Formulation

- formal specification for the task of the agent
  - goal specification
  - states of the world
  - actions of the agent
- identify the type of the problem
  - what knowledge does the agent have about the state of the world and the consequences of its own actions
  - does the execution of the task require up-to-date information
    - sensing is necessary during the execution

Well-Defined Problems

- problems with a readily available formal specification
  - initial state
    - starting point from which the agent sets out
  - actions (operators, successor functions)
    - describe the set of possible actions
  - state space
    - set of all states reachable from the initial state by any sequence of actions
  - path
    - sequence of actions leading from one state in the state space to another
  - goal test
    - determines if a given state is the goal state

Well-Defined Problems (cont.)

- solution
  - path from the initial state to a goal state
- search cost
  - time and memory required to calculate a solution
- path cost
  - determines the expenses of the agent for executing the actions in a path
  - sum of the costs of the individual actions in a path
- total cost
  - sum of search cost and path cost
  - overall cost for finding a solution

Selecting States and Actions

- states describe distinguishable stages during the problem-solving process
  - dependent on the task and domain
- actions move the agent from one state to another one by applying an operator to a state
  - dependent on states, capabilities of the agent, and properties of the environment
- choice of suitable states and operators
  - can make the difference between a problem that can or cannot be solved (in principle, or in practice)

Example: From Home to Cal Poly

- states
  - locations:
    - obvious: buildings that contain your home, Cal Poly CSC dept.
    - more difficult: intermediate states
      - blocks, street corners, sidewalks, entryways, ...
      - continuous transitions
  - agent-centric states
    - moving, turning, resting, ...
- operators
  - depend on the choice of states
  - e.g. `move_one_block`
- abstraction is necessary to omit irrelevant details
  - valid: can be expanded into a detailed version
  - useful: easier to solve than in the detailed version

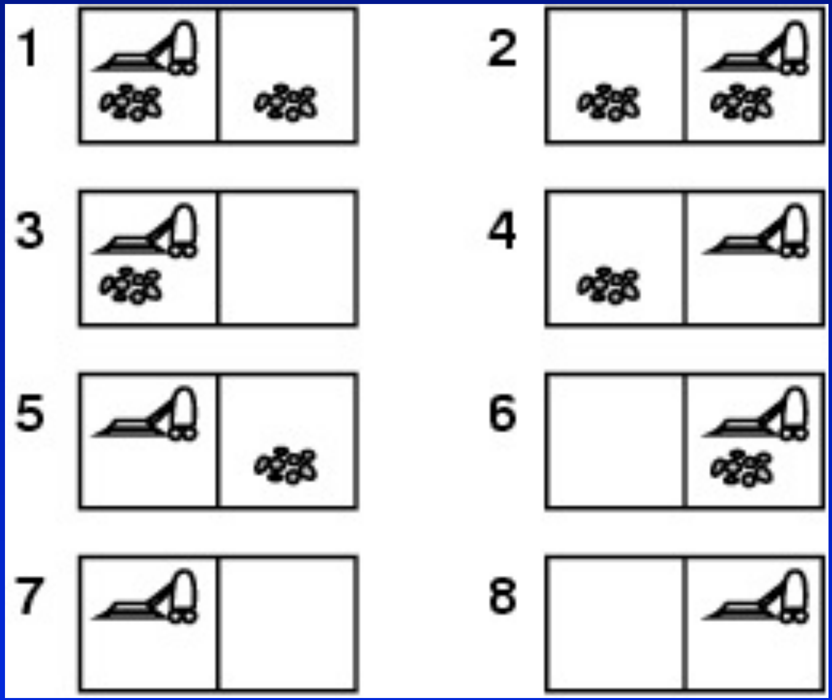
Example Problems

- toy problems
  - vacuum world
  - 8-puzzle
  - 8-queens
  - cryptarithmetic
  - vacuum agent
  - missionaries and cannibals
- real-world problems
  - route finding
  - touring problems
    - traveling salesperson
  - VLSI layout
  - robot navigation
  - assembly sequencing
  - Web search

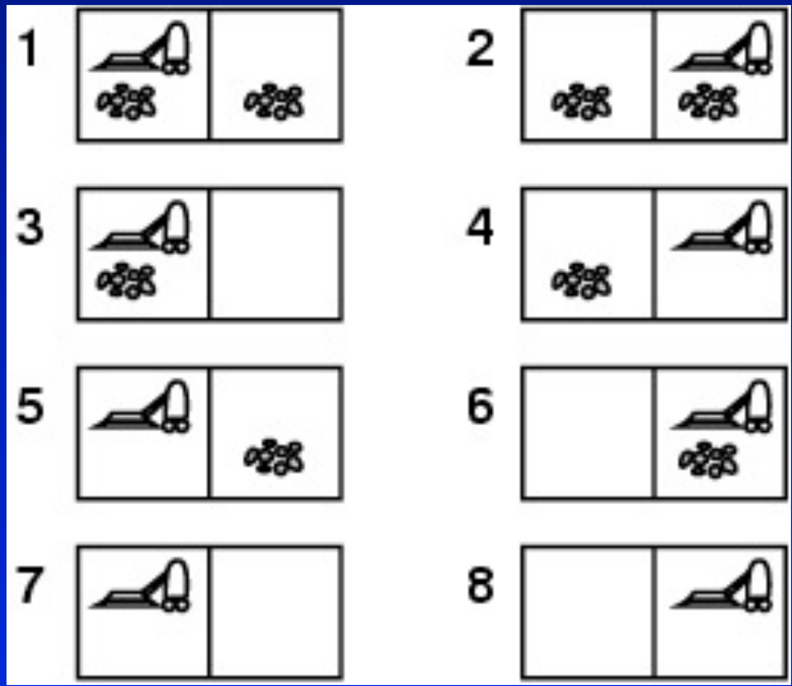
Example: Vacuum World

Example: vacuum world

- Single-state, start in #5.
  Solution?

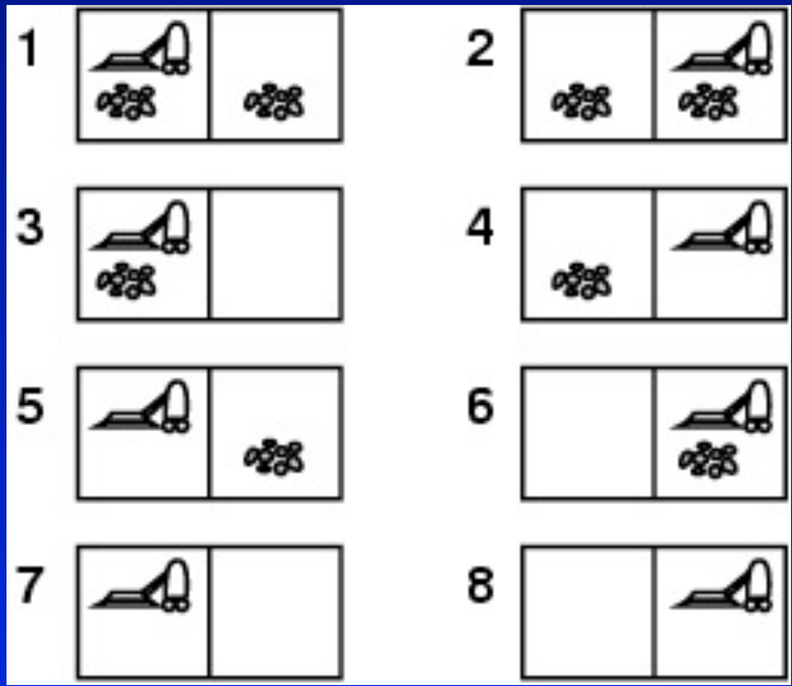


Example: vacuum world



Example: vacuum world

- Sensorless, start in {1,2,3,4,5,6,7,8} e.g., Right goes to {2,4,6,8} Solution?
  [Right,Suck,Left,Suck]

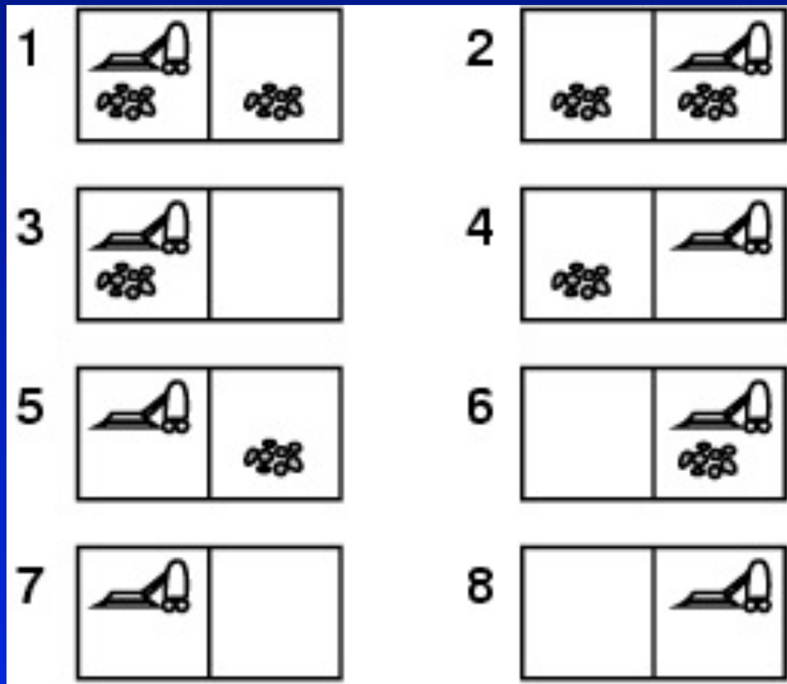


- Contingency
  - Nondeterministic: Suck may dirty a clean carpet
  - Partially observable: location, dirt at current location.
  - Percept: [L, Clean], i.e., start in #5 or #7
    Solution?

Monday, October 8, 12

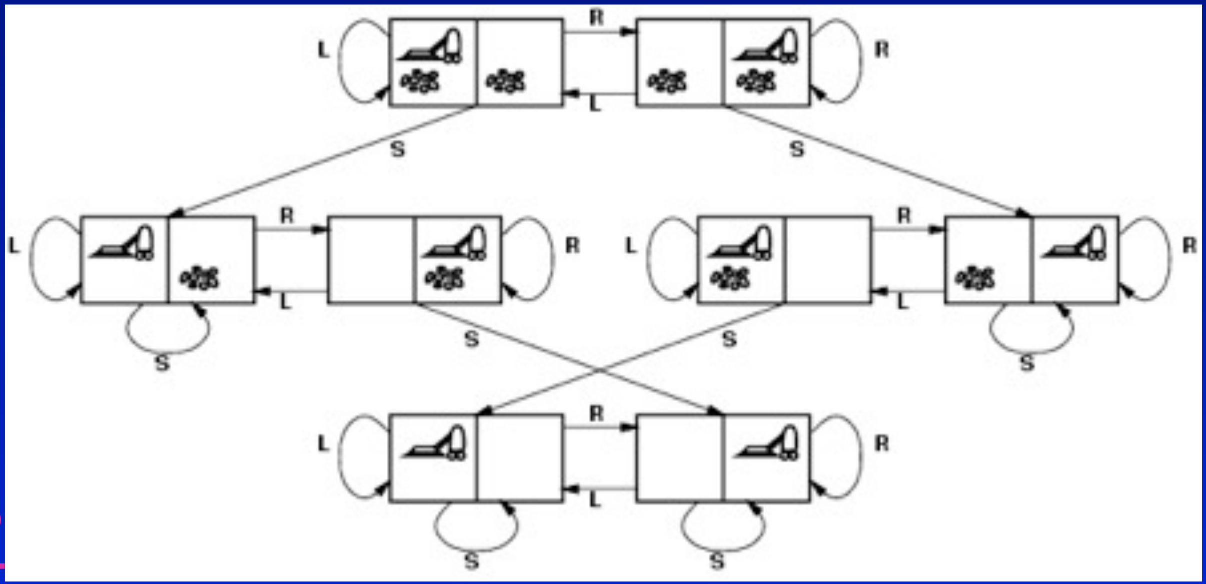
Example: vacuum world

- Sensorless, start in {1,2,3,4,5,6,7,8} e.g., Right goes to {2,4,6,8} Solution? [Right,Suck,Left,Suck]



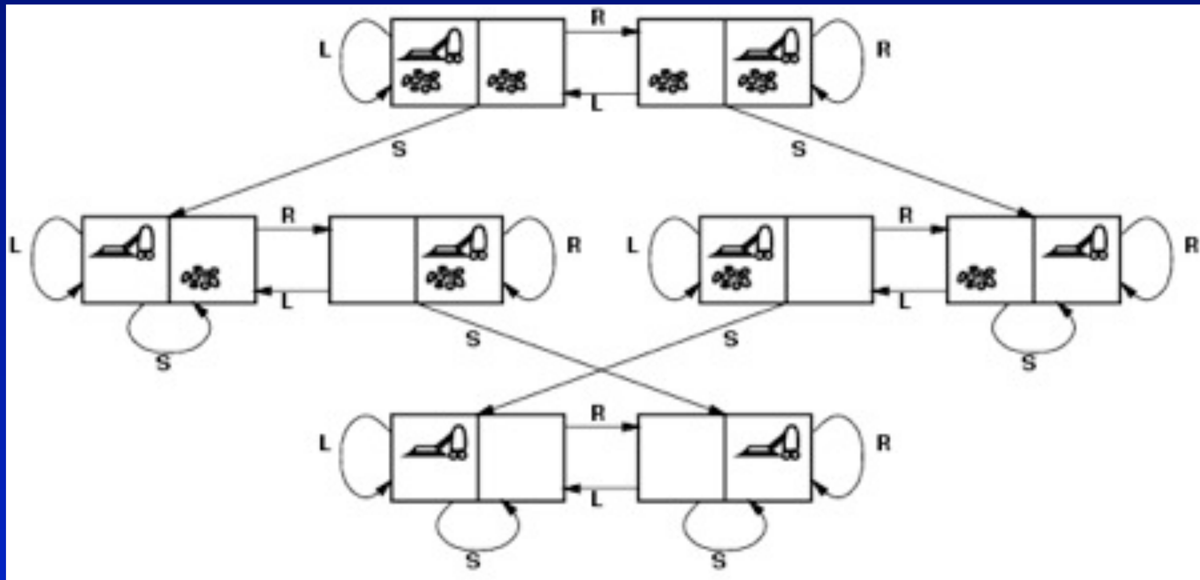
- Contingency
  - Nondeterministic: Suck may dirty a clean carpet
  - Partially observable: location, dirt at current location.
  - Percept: [L, Clean], i.e., start in #5 or #7 Solution? [Right, **if** dirt **then** Suck]

Vacuum world state space graph



- states?
- actions?
- goal test?
- path cost?

Vacuum world state space graph



- states? integer dirt and robot location
- actions? Left, Right, Suck
- goal test? no dirt at all locations
- path cost? 1 per action

Simple Vacuum World

- states
  - two locations
  - dirty, clean
- initial state
  - any legitimate state
- successor function (operators)
  - left, right, suck
- goal test
  - all squares clean
- path cost
  - one unit per action

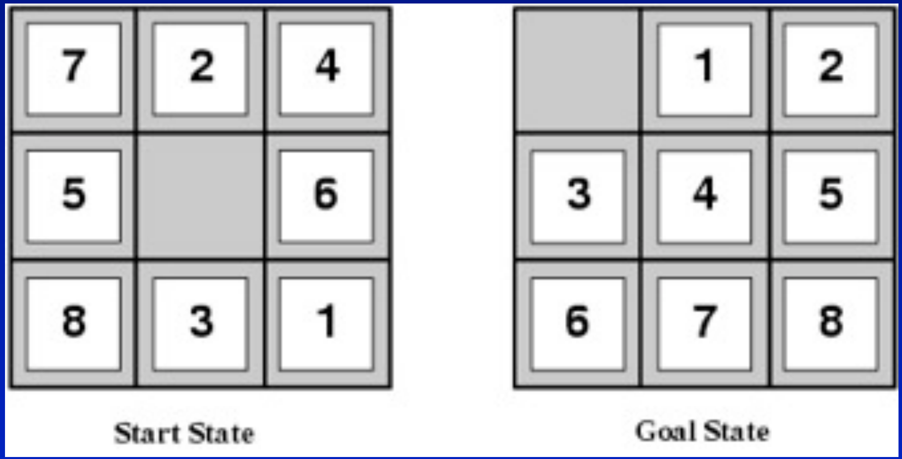
Properties: discrete locations, discrete dirt (binary), deterministic

More Complex Vacuum Agent

- states
  - configuration of the room
    - dimensions, obstacles, dirtiness
- initial state
  - locations of agent, dirt
- successor function (operators)
  - move, turn, suck
- goal test
  - all squares clean
- path cost
  - one unit per action

Properties: discrete locations, discrete dirt, deterministic,
$d * 2^n$ states for dirt degree d,n locations

Example: The 8-puzzle



- states?
- actions?
- goal test?
- path cost?

[Note: optimal solution of n-Puzzle family is NP-hard]

8-Puzzle

- states
  - location of tiles (including blank tile)
- initial state
  - any legitimate configuration
- successor function (operators)
  - move tile
  - alternatively: move blank
- goal test
  - any legitimate configuration of tiles
- path cost
  - one unit per move

Properties: abstraction leads to discrete configurations, discrete moves, deterministic

9!/2 = 181,440 reachable states

Example: *n*-queens

- Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal

8-Queens

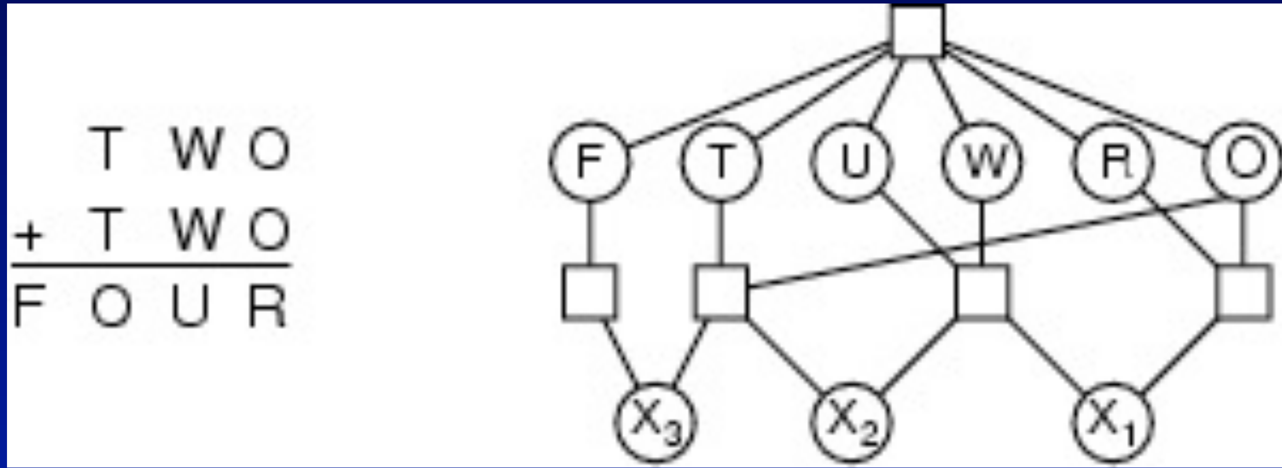
- incremental formulation
  - states
    - arrangement of up to 8 queens on the board
  - initial state
    - empty board
  - successor function (operators)
    - add a queen to any square
  - goal test
    - all queens on board
    - no queen attacked
  - path cost
    - irrelevant (all solutions equally valid)
- Properties: $3*10^{14}$ possible sequences; can be reduced to 2,057

- complete-state formulation
  - states
    - arrangement of 8 queens on the board
  - initial state
    - all 8 queens on board
  - successor function (operators)
    - move a queen to a different square
  - goal test
    - no queen attacked
  - path cost
    - irrelevant (all solutions equally valid)
- Properties: good strategies can reduce the number of possible sequences considerably

8-Queens Refined

- simple solutions may lead to very high search costs
  - 64 fields, 8 queens ==> 64^8 possible sequences
- more refined solutions trim the search space, but may introduce other constraints
  - place queens on “unattacked” places
    - much more efficient
    - may not lead to a solutions depending on the initial moves
  - move an attacked queen to another square in the same column, if possible to an “unattacked” square
    - much more efficient

Crypt-arithmetic



- states
  - puzzle with letters and digits
- initial state
  - only letters present
- successor function (operators)
  - replace all occurrences of a letter by a digit not used yet
- goal test
  - only digits in the puzzle
  - calculation is correct
- path cost
  - all solutions are equally valid

Missionaries and Cannibals

- states
  - number of missionaries, cannibals, and boats on the banks of a river
  - illegal states
    - missionaries are outnumbered by cannibals on either bank
- initial states
  - all missionaries, cannibals, and boats are on one bank
- successor function (operators)
  - transport a set of up to two participants to the other bank
    - {1 missionary} | { 1cannibal} | {2 missionaries} | {2 cannibals} | {1 missionary and 1 cannibal}
- goal test
  - nobody left on the initial river bank
- path cost
  - number of crossings

also known as “goats and cabbage”, “wolves and sheep”, etc

Route Finding

- states
  - locations
- initial state
  - starting point
- successor function (operators)
  - move from one location to another
- goal test
  - arrive at a certain location
- path cost
  - may be quite complex
    - money, time, travel comfort, scenery, ...

Traveling Salesperson

- states
  - locations / cities
  - illegal states
    - each city may be visited only once
    - visited cities must be kept as state information
- initial state
  - starting point
  - no cities visited
- successor function (operators)
  - move from one location to another one
- goal test
  - all locations visited
  - agent at the initial location
- path cost
  - distance between locations

VLSI Layout

- states
  - positions of components, wires on a chip
- initial state
  - incremental: no components placed
  - complete-state: all components placed (e.g. randomly, manually)
- successor function (operators)
  - incremental: place components, route wire
  - complete-state: move component, move wire
- goal test
  - all components placed
  - components connected as specified
- path cost
  - may be complex
    - distance, capacity, number of connections per component

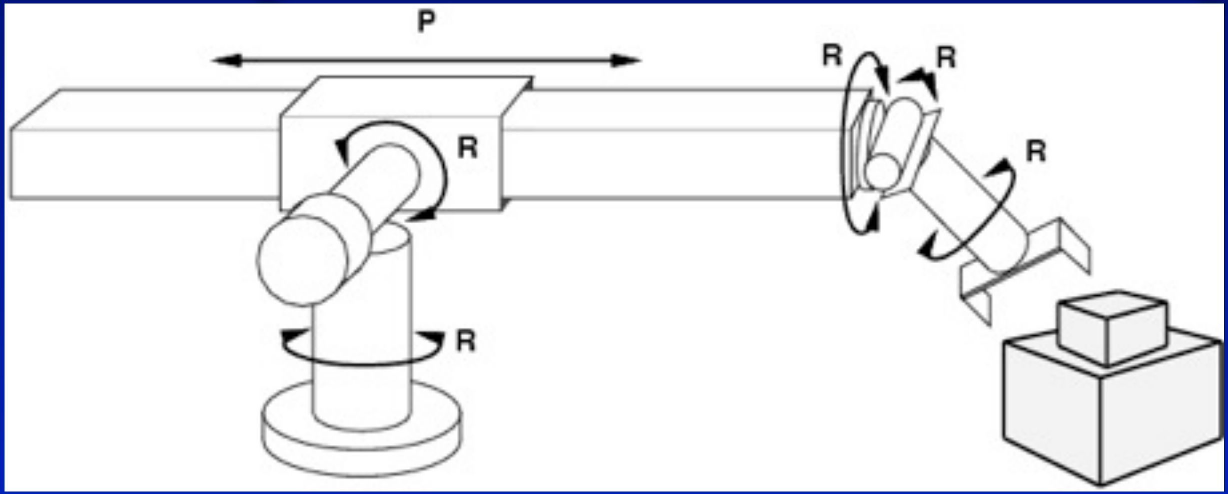
Robot Navigation

- states
  - locations
  - position of actuators
- initial state
  - start position (dependent on the task)
- successor function (operators)
  - movement, actions of actuators
- goal test
  - task-dependent
- path cost
  - may be very complex
    - distance, energy consumption

Assembly Sequencing

- states
  - location of components
- initial state
  - no components assembled
- successor function (operators)
  - place component
- goal test
  - system fully assembled
- path cost
  - number of moves

Example: robotic assembly



- states?: real-valued coordinates of robot joint angles parts of the object to be assembled
- actions?: continuous motions of robot joints
- goal test?: complete assembly
- path cost?: time to execute

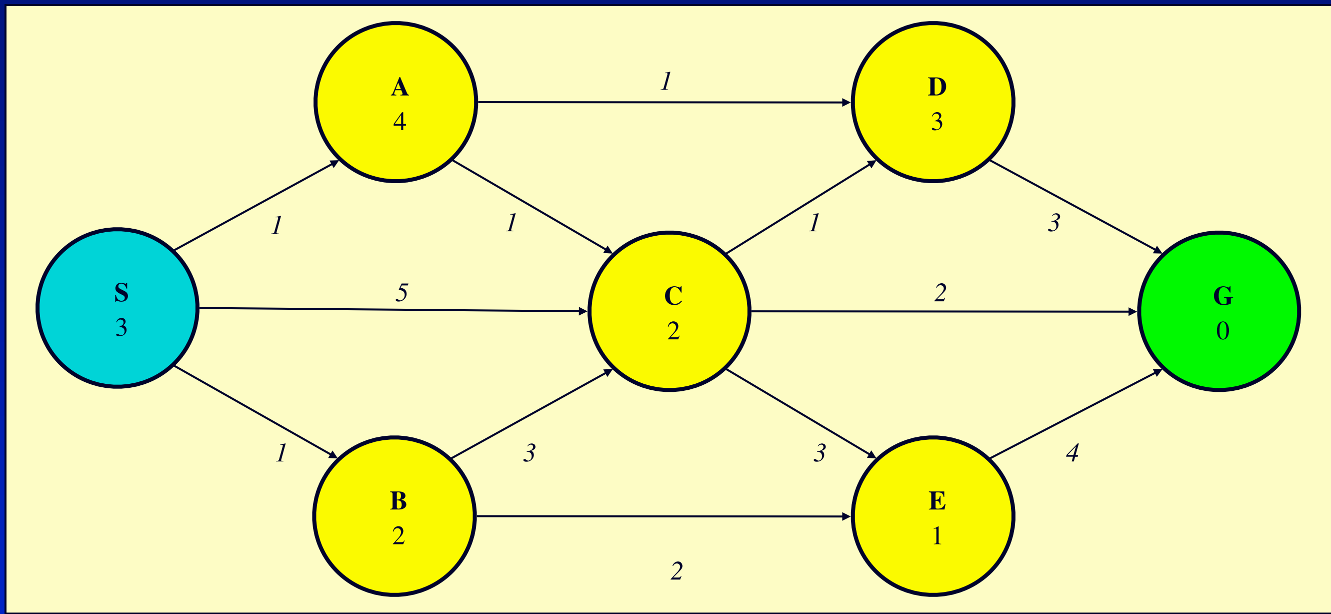
Searching for Solutions

- traversal of the search space
  - from the initial state to a goal state
  - legal sequence of actions as defined by successor function (operators)
- general procedure
  - check for goal state
  - expand the current state
    - determine the set of reachable states
    - return “failure” if the set is empty
  - select one from the set of reachable states
  - move to the selected state
- a search tree is generated
  - nodes are added as more states are visited

Search Terminology

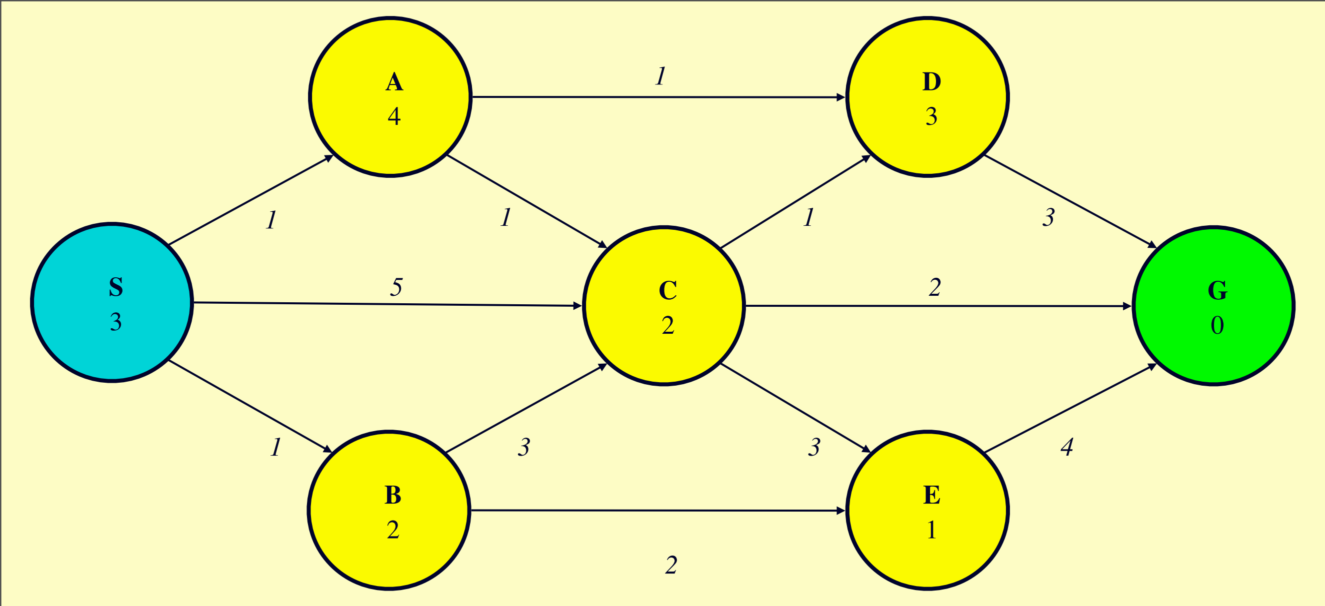
- **search tree**
  - generated as the search space is traversed
    - the search space itself is not necessarily a tree, frequently it is a graph
    - the tree specifies possible paths through the search space
  - expansion of nodes
    - as states are explored, the corresponding nodes are expanded by applying the successor function
      - this generates a new set of (child) nodes
    - the **fringe** (frontier) 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)

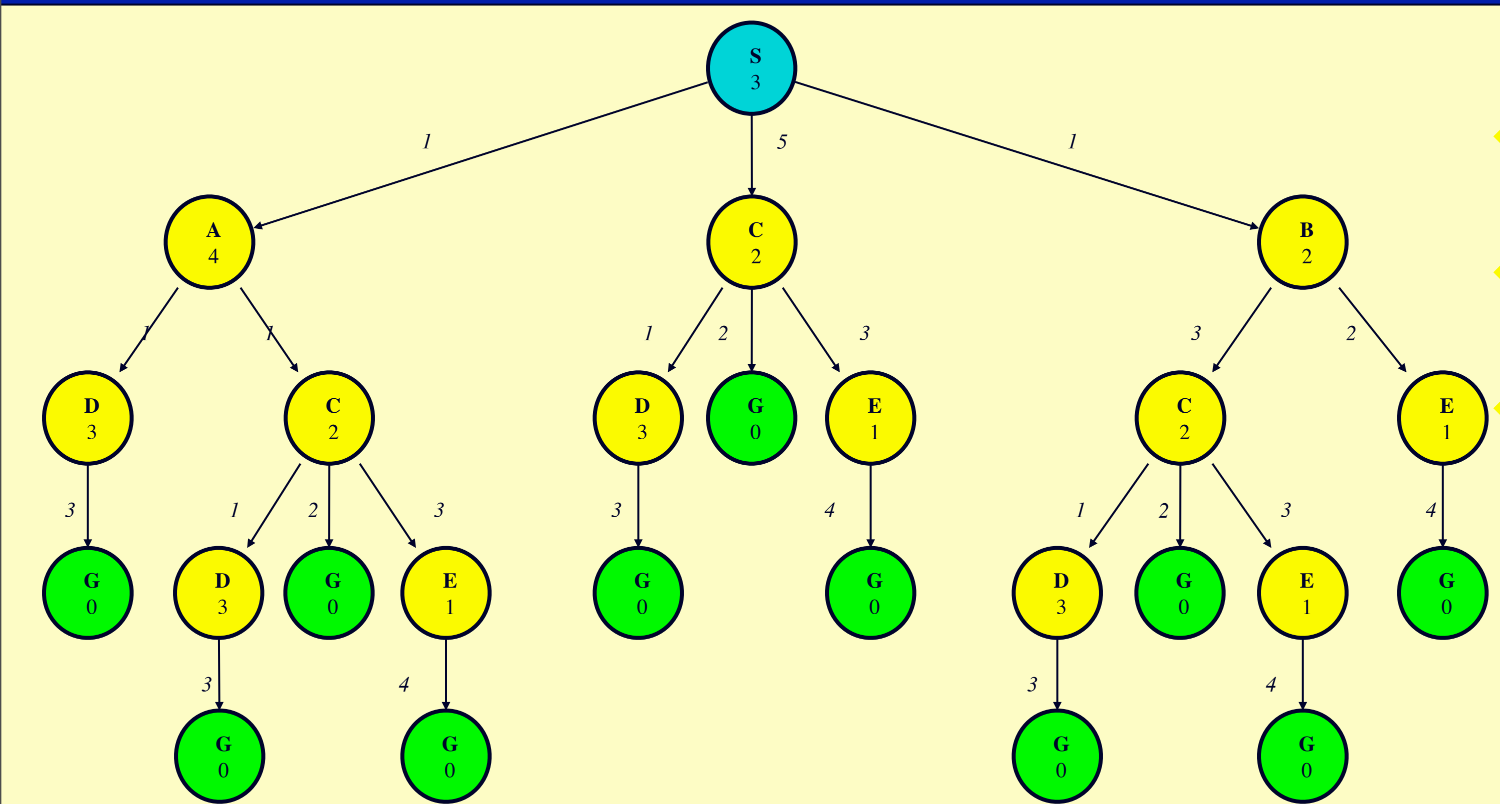
Example: Graph Search



- 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. **S**, 3)
  - links between nodes, specified by the successor function
    - properties for links (distance, cost, name, ...)

Graph and Tree





- the 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

Logistics

- AI Nugget presentations
  - Section 1:
    - Ray Tam: Image Processing
    - Andrew Sinclair: Autonomous Bitcoin Agents
  - Section 3: -

General Tree Search Algorithm

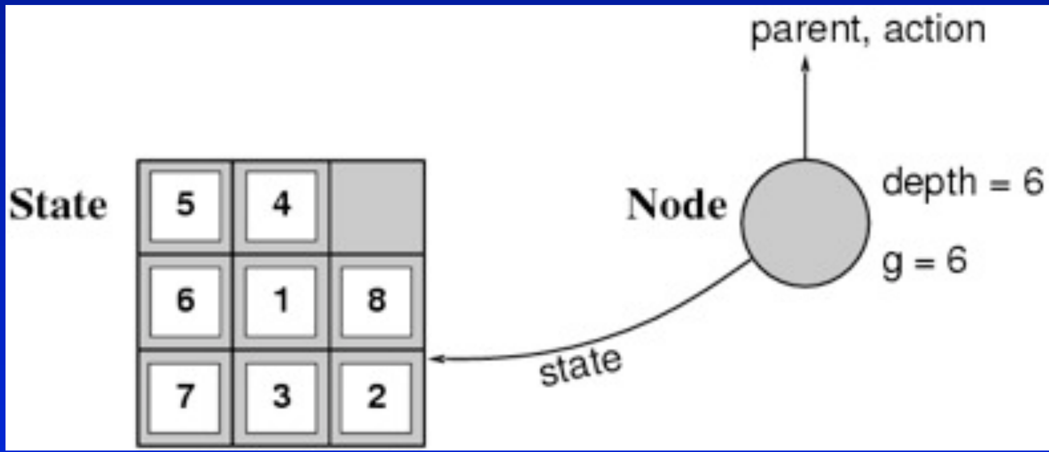
```
function TREE-SEARCH(problem, fringe) returns solution
  fringe  := INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
   loop do
       if EMPTY?(fringe) then return failure
       node  := REMOVE-FIRST(fringe)
       if GOAL-TEST[problem] applied to STATE[node] succeeds
          then return SOLUTION(node)
       fringe := INSERT-ALL(EXPAND(node, problem), fringe)
```

- generate the node from the initial state of the problem
- repeat
  - return failure if there are no more nodes in the fringe
  - examine the current node; if it's a goal, return the solution
  - expand the current node, and add the new nodes to the fringe

Note: This method is called "General-Search" in earlier AIMA editions

Implementation: states vs. nodes

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree includes state, parent node, action, path cost g(x), depth



- The `Expand` function creates new nodes, filling in the various fields and using the `SuccessorFn` of the problem to create the corresponding states.

Evaluation Criteria

- completeness
  - if there is a solution, will it be found
- optimality
  - the best solution will be found
- time complexity
  - time it takes to find the solution
  - does not include the time to perform actions
- space complexity
  - memory required for the search

main factors for complexity considerations:

branching factor *b*, depth *d* of the shallowest goal node, maximum path length *m*

Search Cost and Path Cost

- the *search cost* indicates how expensive it is to generate a solution
  - time complexity (e.g. number of nodes generated) is usually the main factor
  - sometimes space complexity (memory usage) is considered as well
- *path cost* indicates how expensive it is to execute the solution found in the search
  - distinct from the search cost, but often related
- *total cost* is the sum of search and path costs

Selection of a Search Strategy

- 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

Search Strategies

- Uninformed Search
  - breadth-first
  - depth-first
  - uniform-cost search
  - depth-limited search
  - iterative deepening
  - bi-directional search
- Informed Search
  - best-first search
  - search with heuristics
  - memory-bounded search
  - iterative improvement search
- Local Search and Optimization
  - hill-climbing
  - simulated annealing
  - local beam search
  - genetic algorithms
  - constraint satisfaction
- Search in Continuous Spaces
- Non-deterministic Actions
- Partial Observations
- Online Search

Breadth-First

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

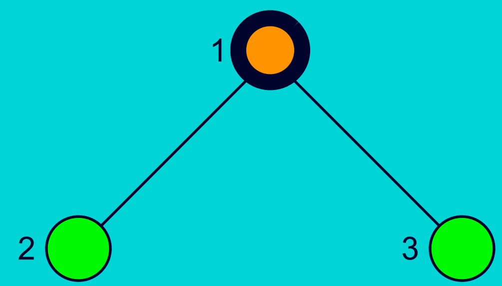
```
function BREADTH-FIRST-SEARCH(problem) returns solution

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

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

b	branching factor
d	depth of the tree

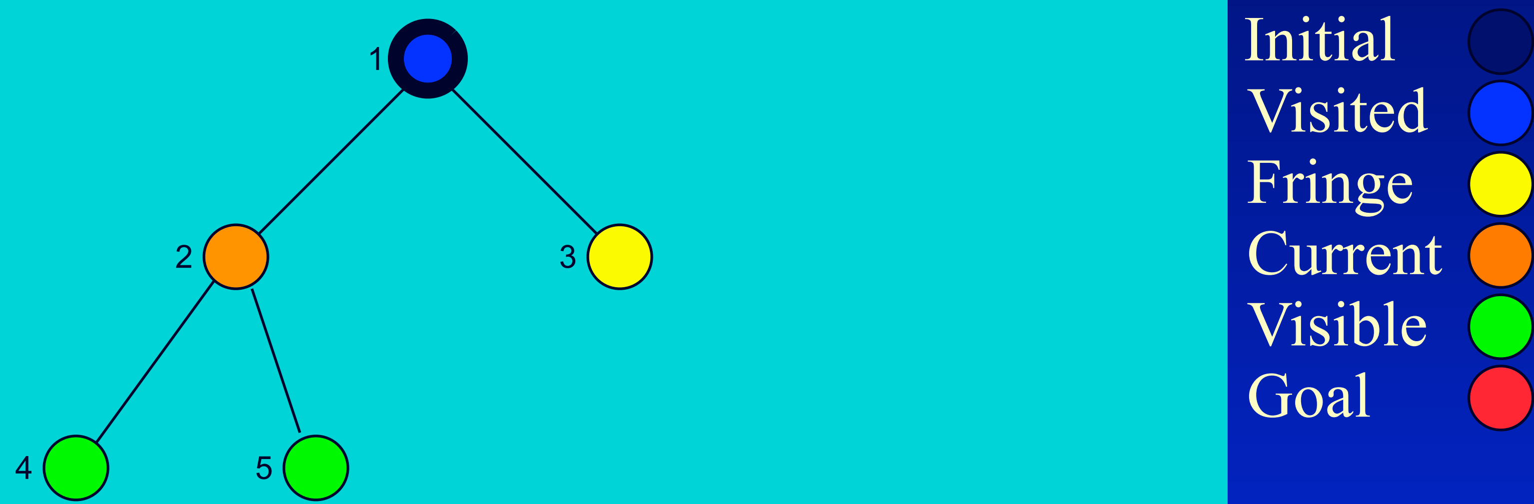
Breadth-First Snapshot 1



Initial
Visited
Fringe
Current
Visible
Goal

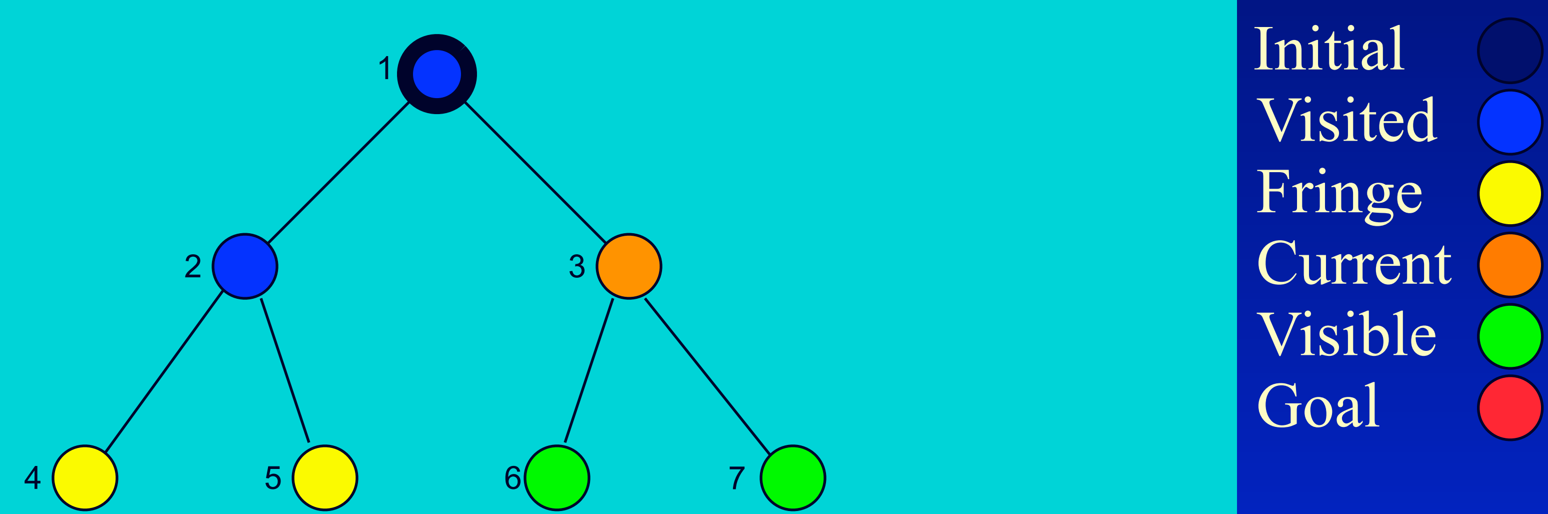
Fringe: [] + [2,3]

Breadth-First Snapshot 2



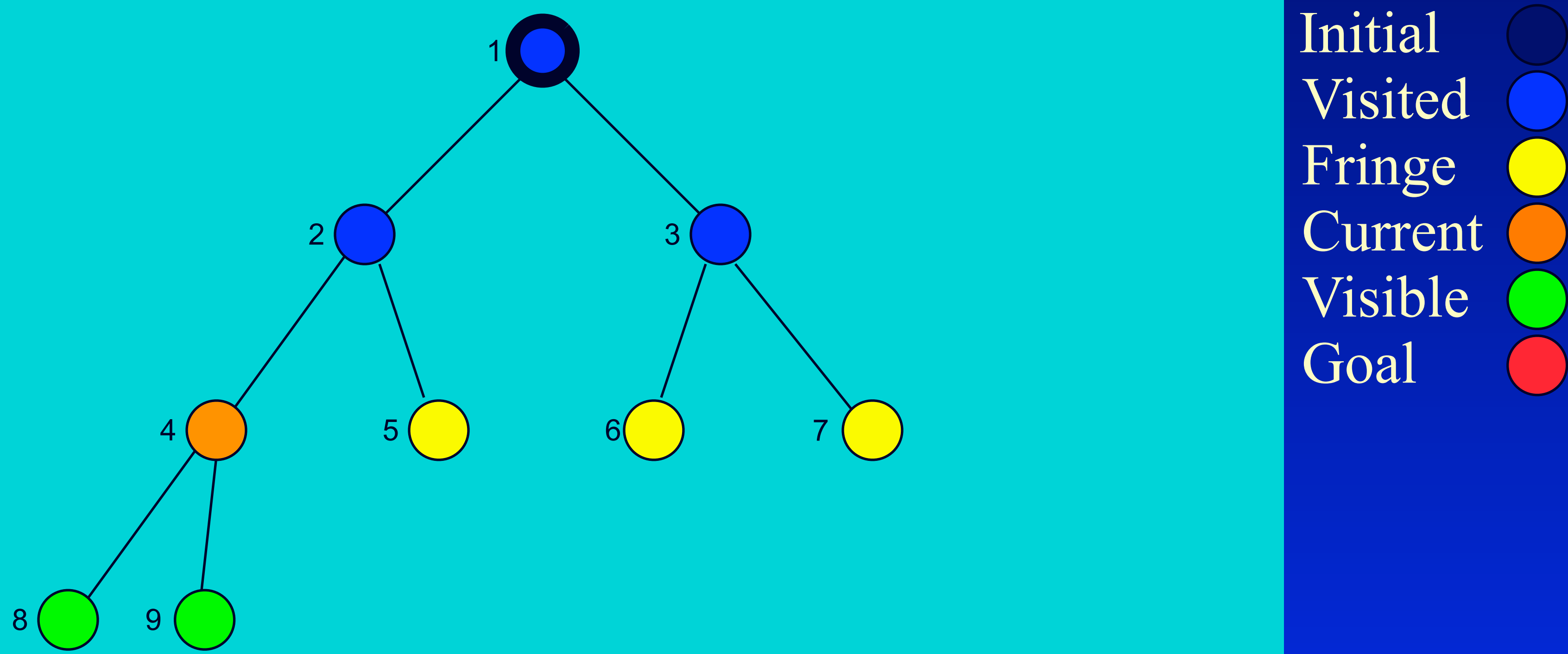
Fringe: [3] + [4,5]

Breadth-First Snapshot 3



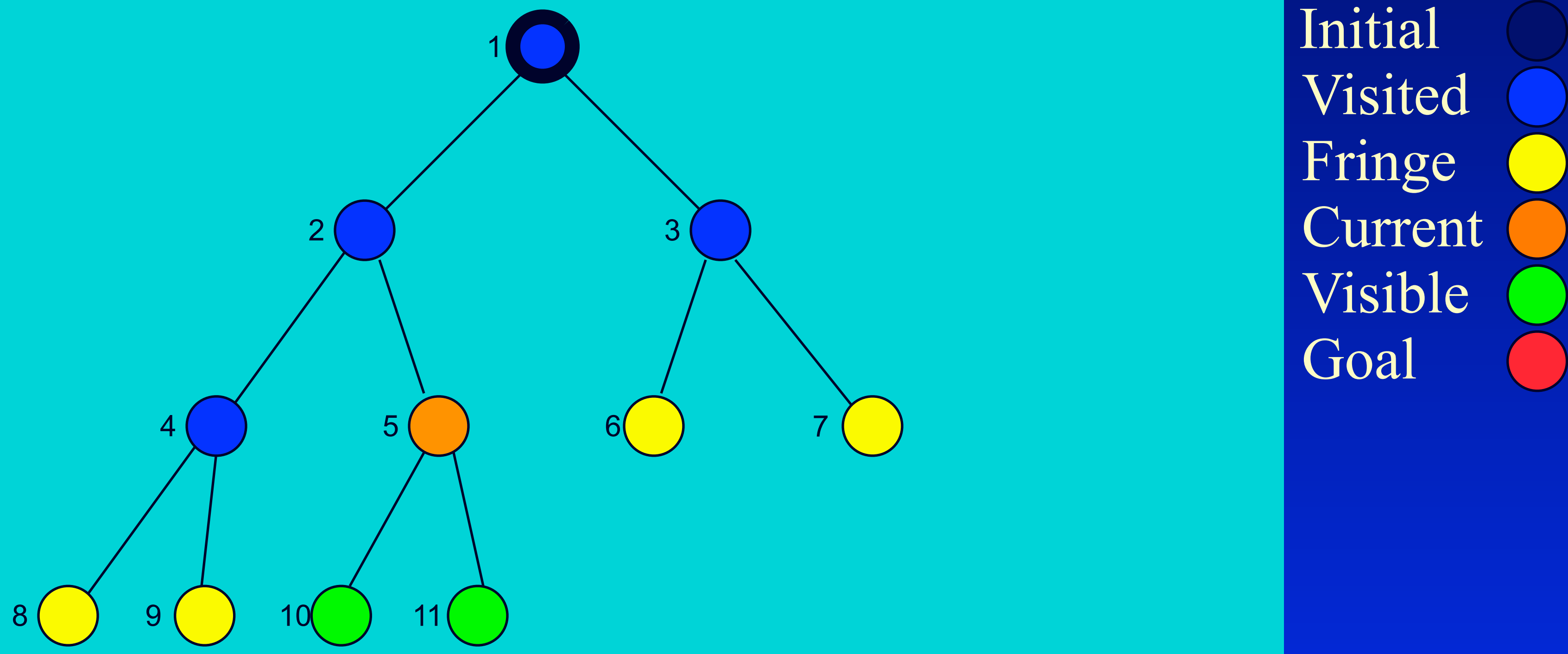
Fringe: [4,5] + [6,7]

Breadth-First Snapshot 4



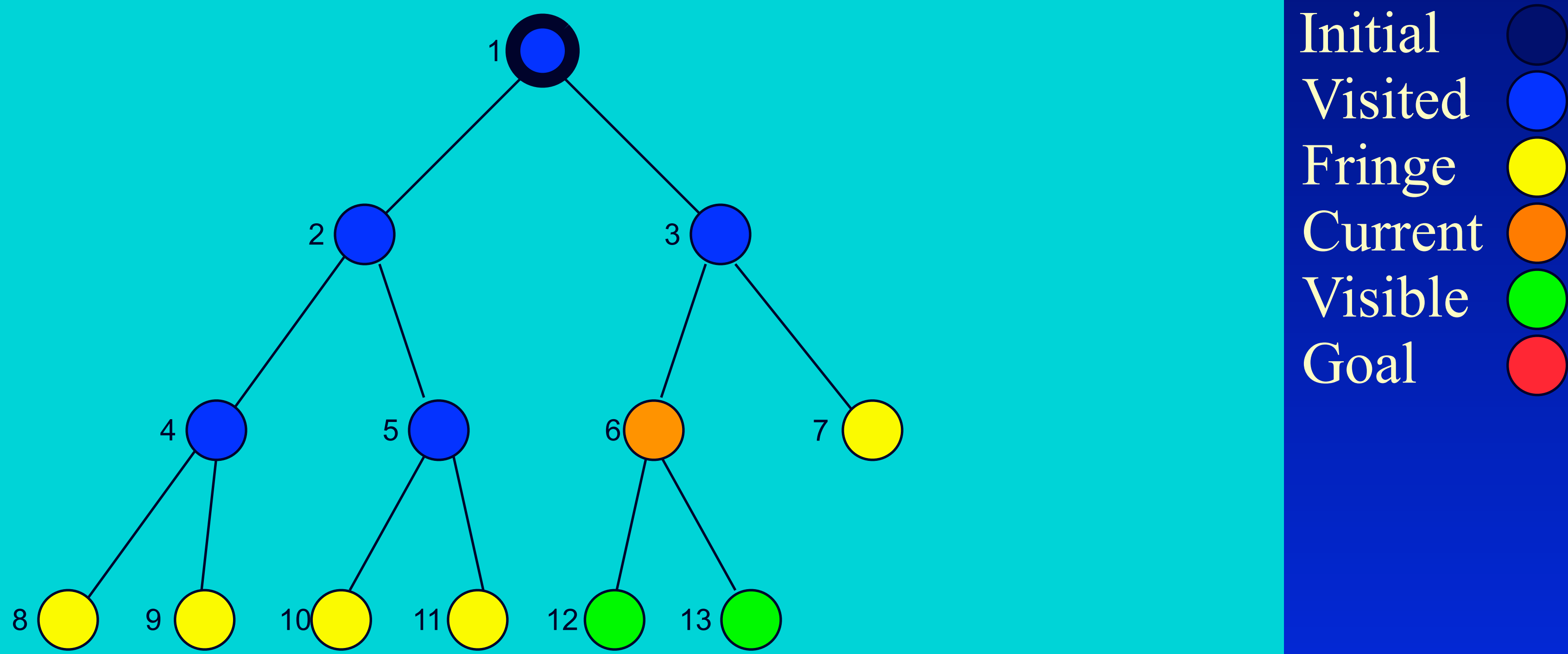
Fringe: [5,6,7] + [8,9]

Breadth-First Snapshot 5



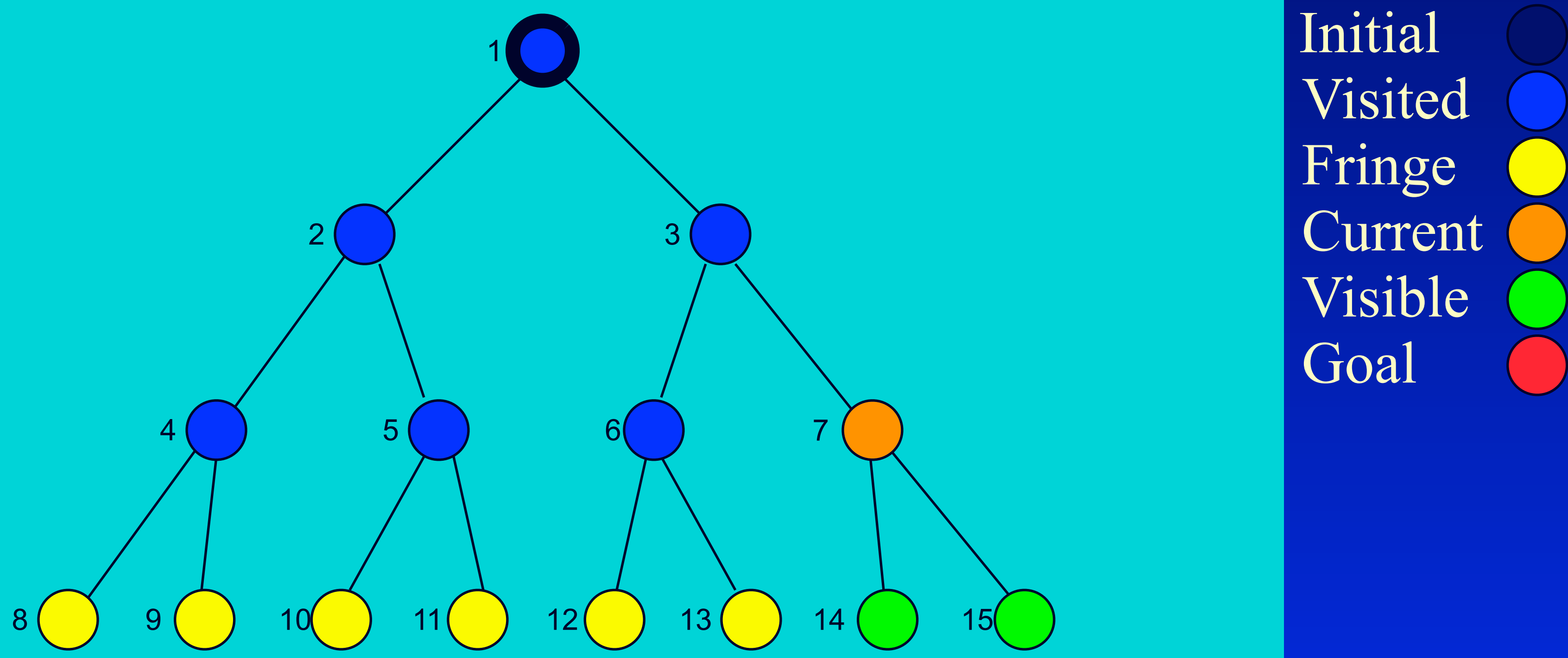
Fringe: [6,7,8,9] + [10,11]

Breadth-First Snapshot 6



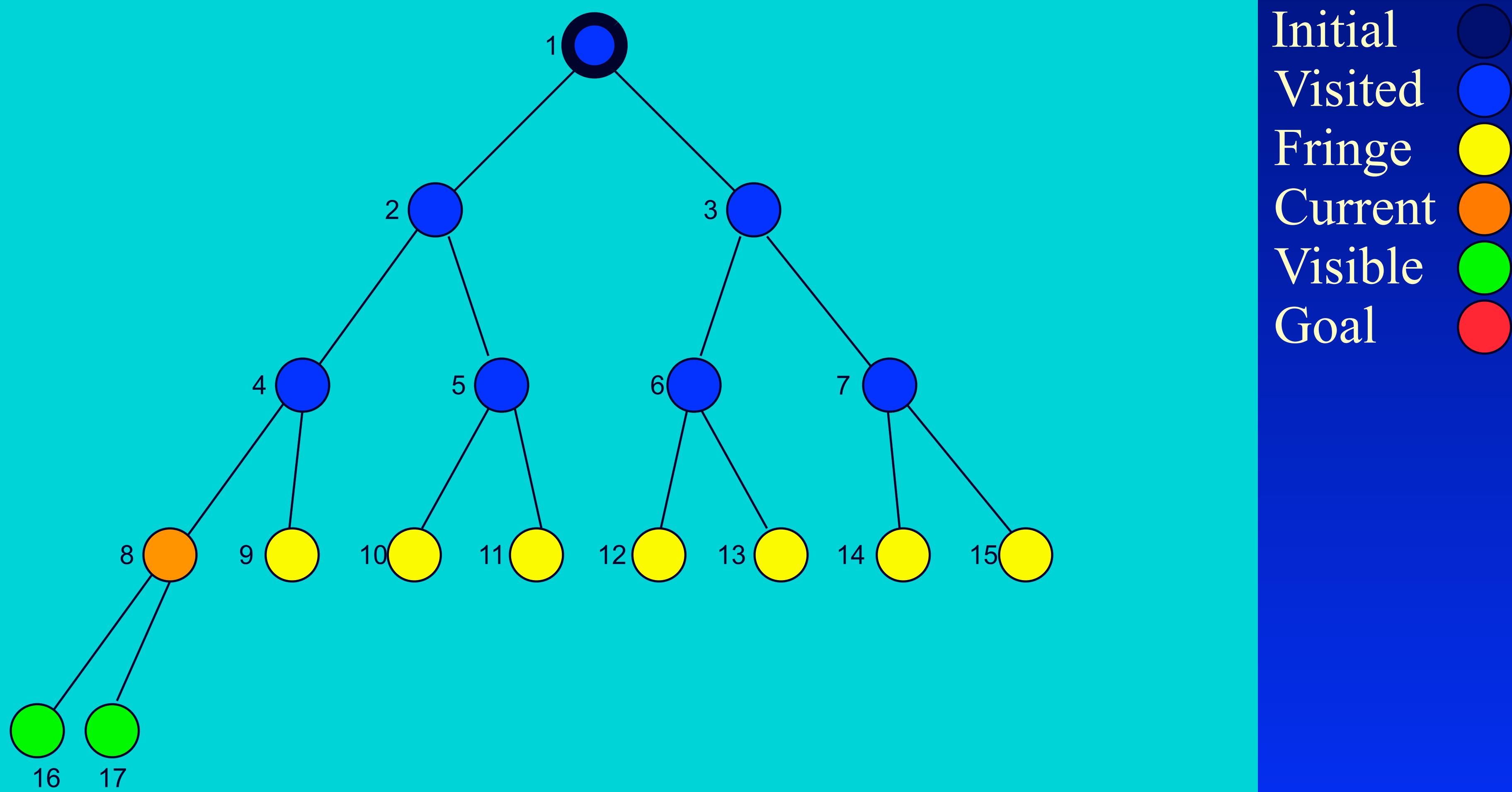
Fringe: [7,8,9,10,11] + [12,13]

Breadth-First Snapshot 7



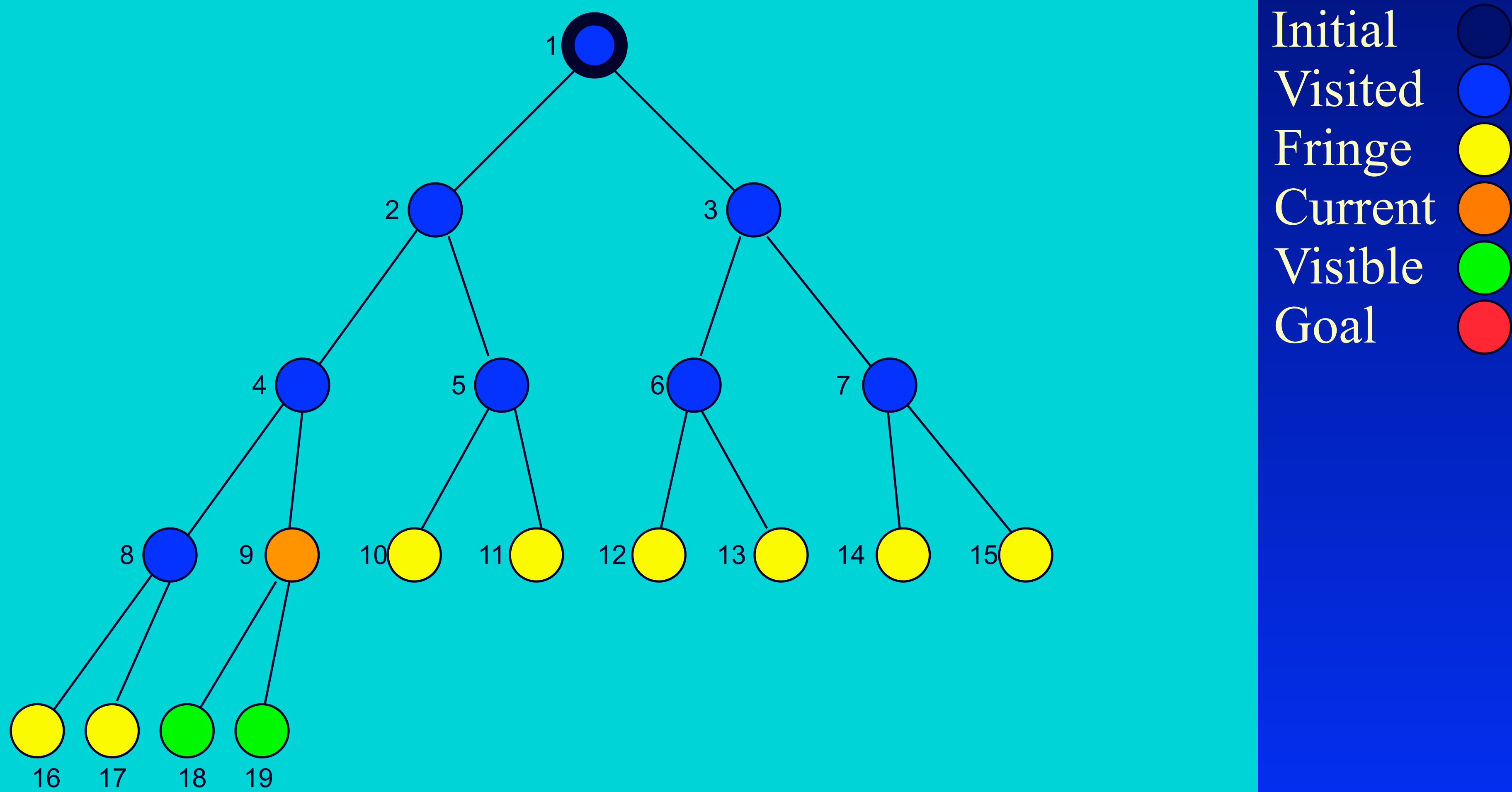
Fringe: [8,9.10,11,12,13] + [14,15]

Breadth-First Snapshot 8



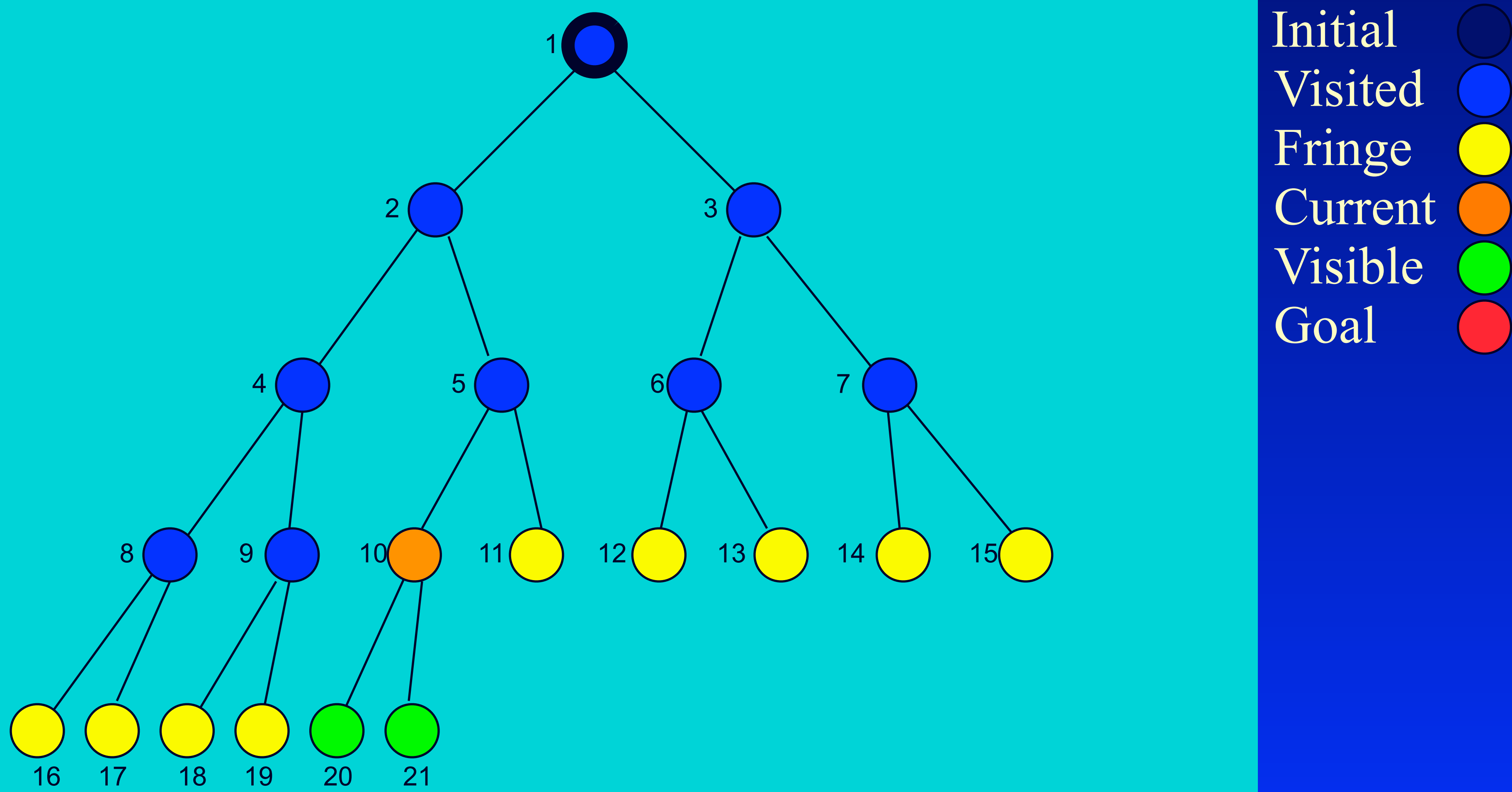
Fringe: [9,10,11,12,13,14,15] + [16,17]

Breadth-First Snapshot 9



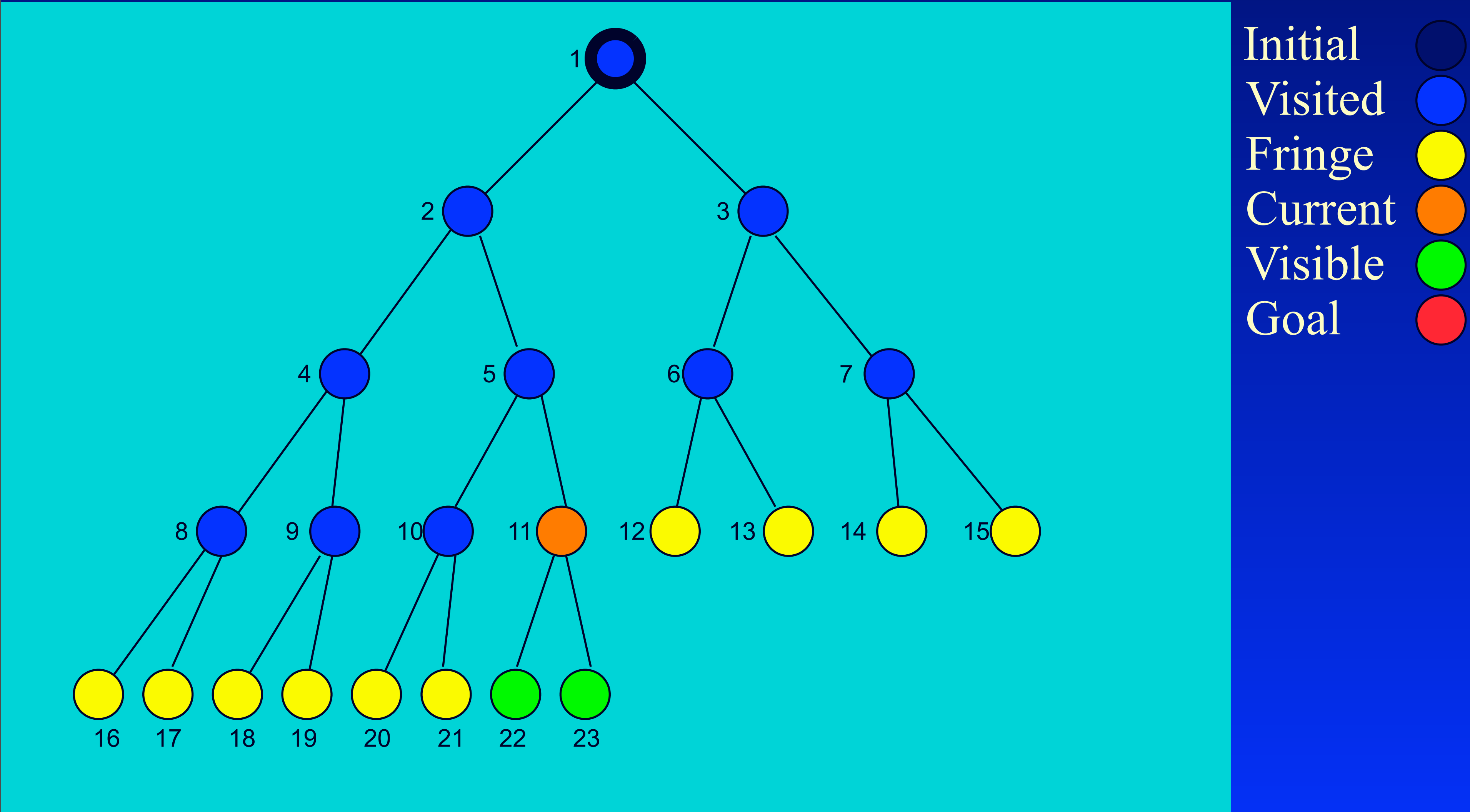
Fringe: [10,11,12,13,14,15,16,17] + [18,19]

Breadth-First Snapshot 10



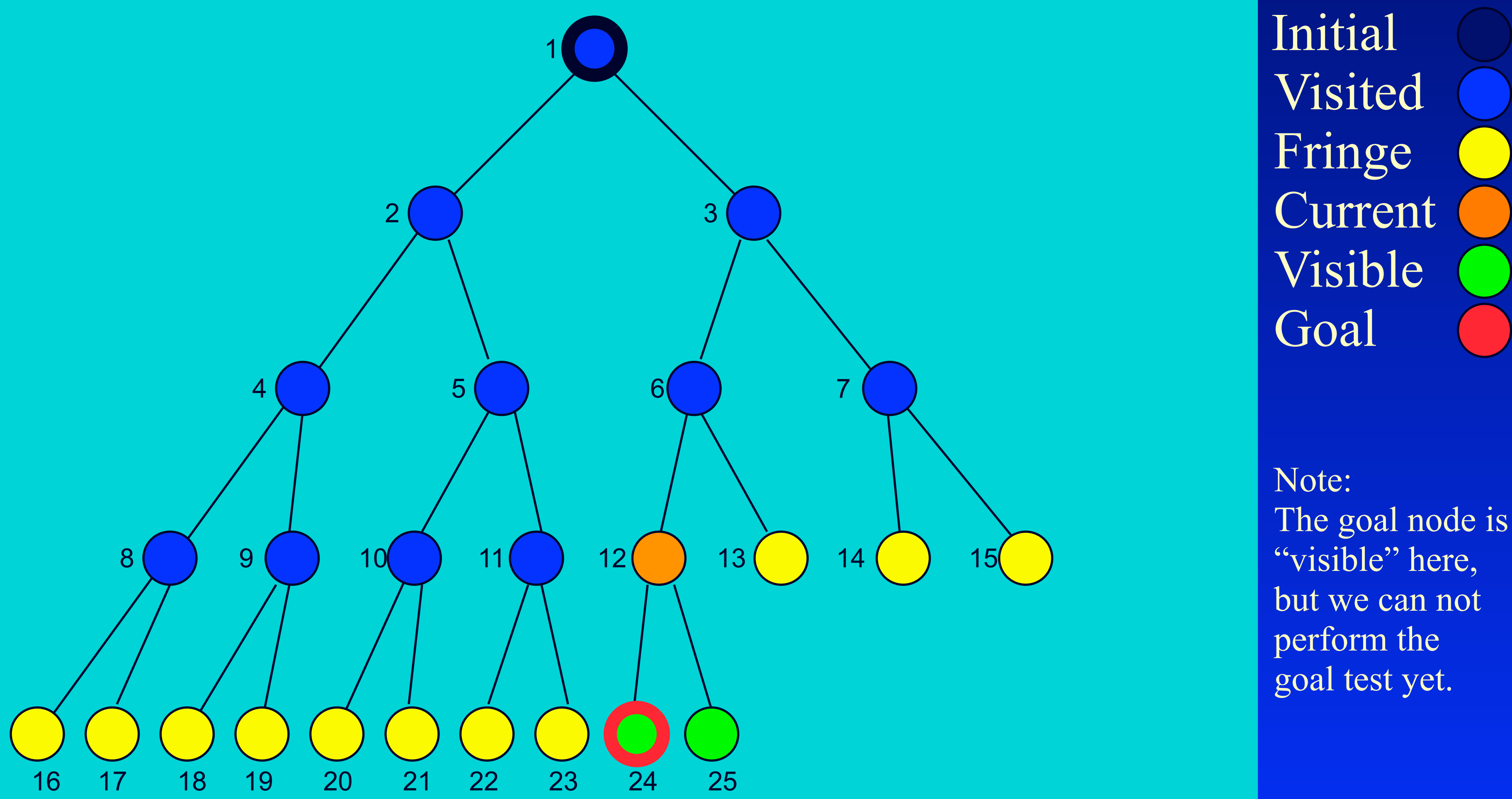
Fringe: [11,12,13,14,15,16,17,18,19] + [20,21]

Breadth-First Snapshot 11



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

Breadth-First Snapshot 12

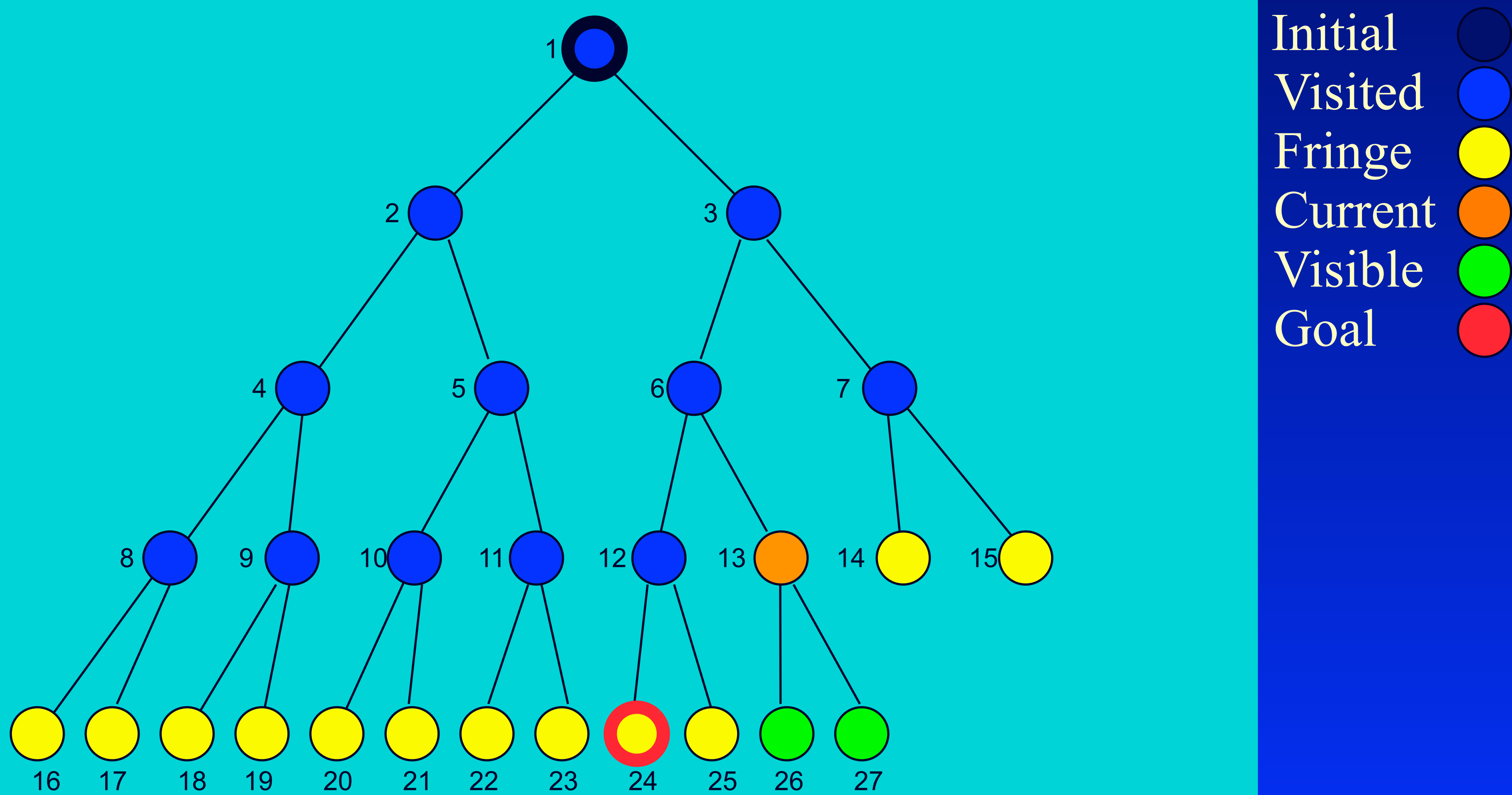


Note:
The goal node is "visible" here, but we can not perform the goal test yet.

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

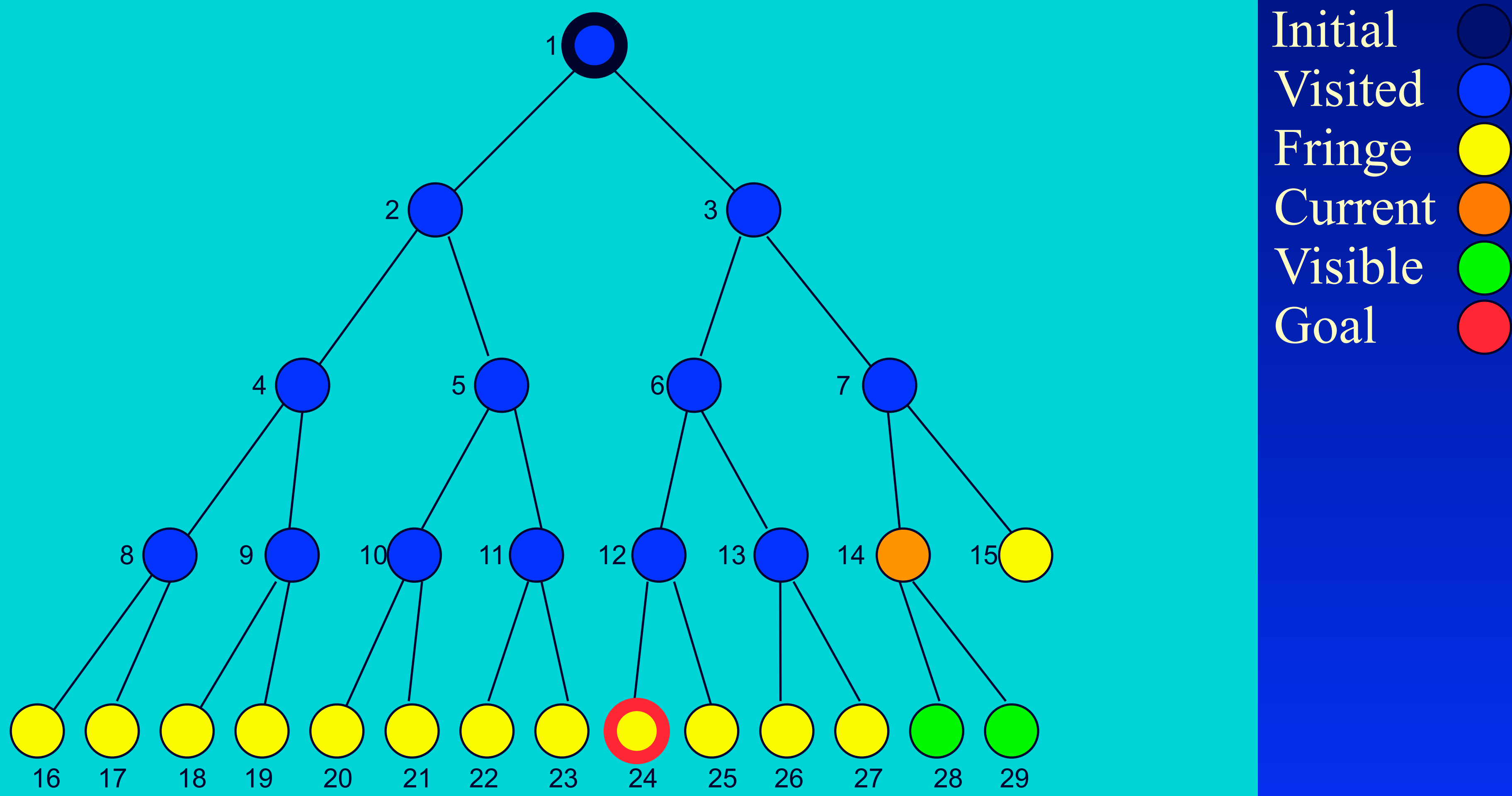
Monday, October 8, 12

Breadth-First Snapshot 13



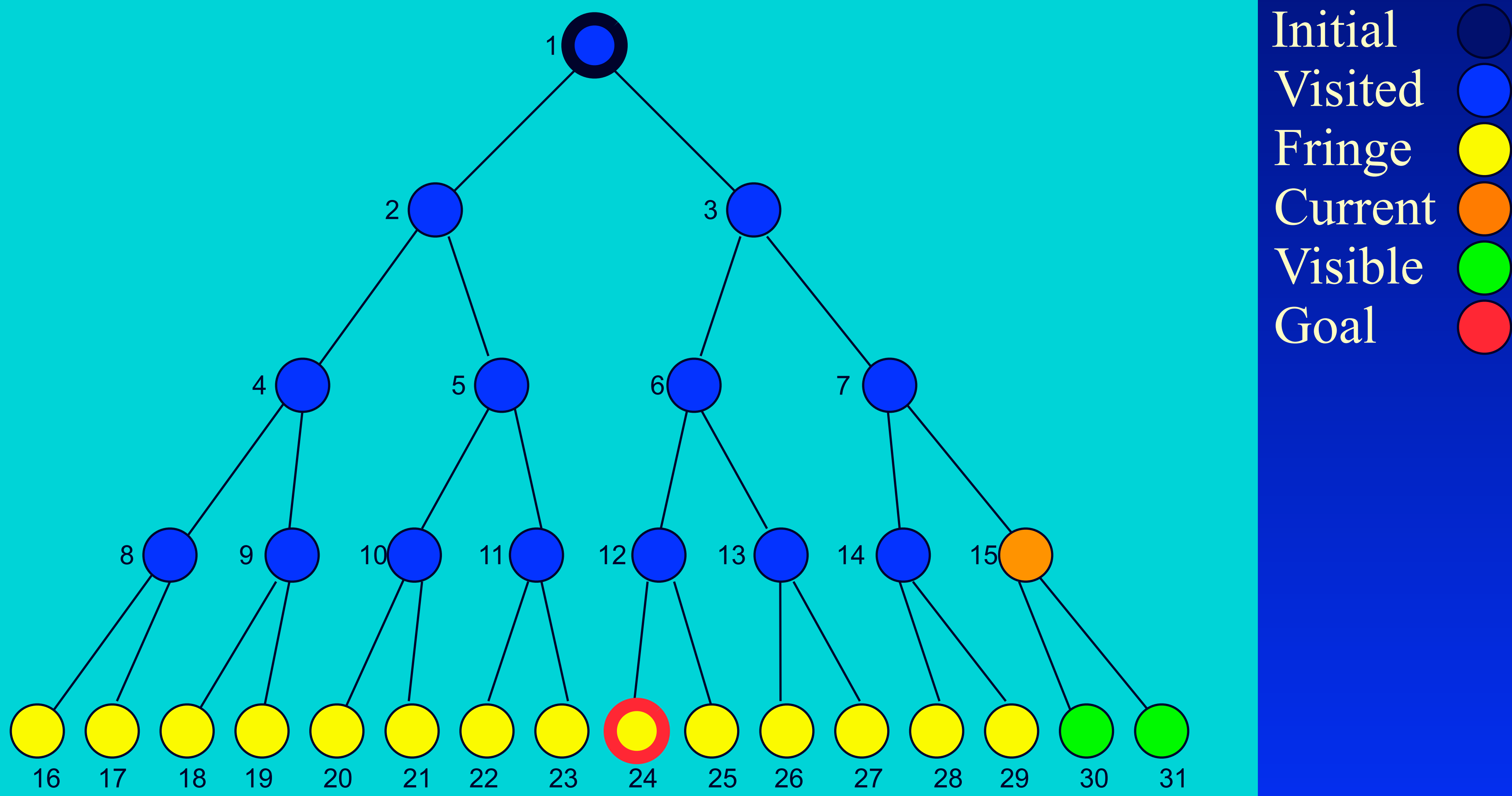
Fringe: [14,15,16,17,18,19,20,21,22,23,24,25] + [26,27]

Breadth-First Snapshot 14



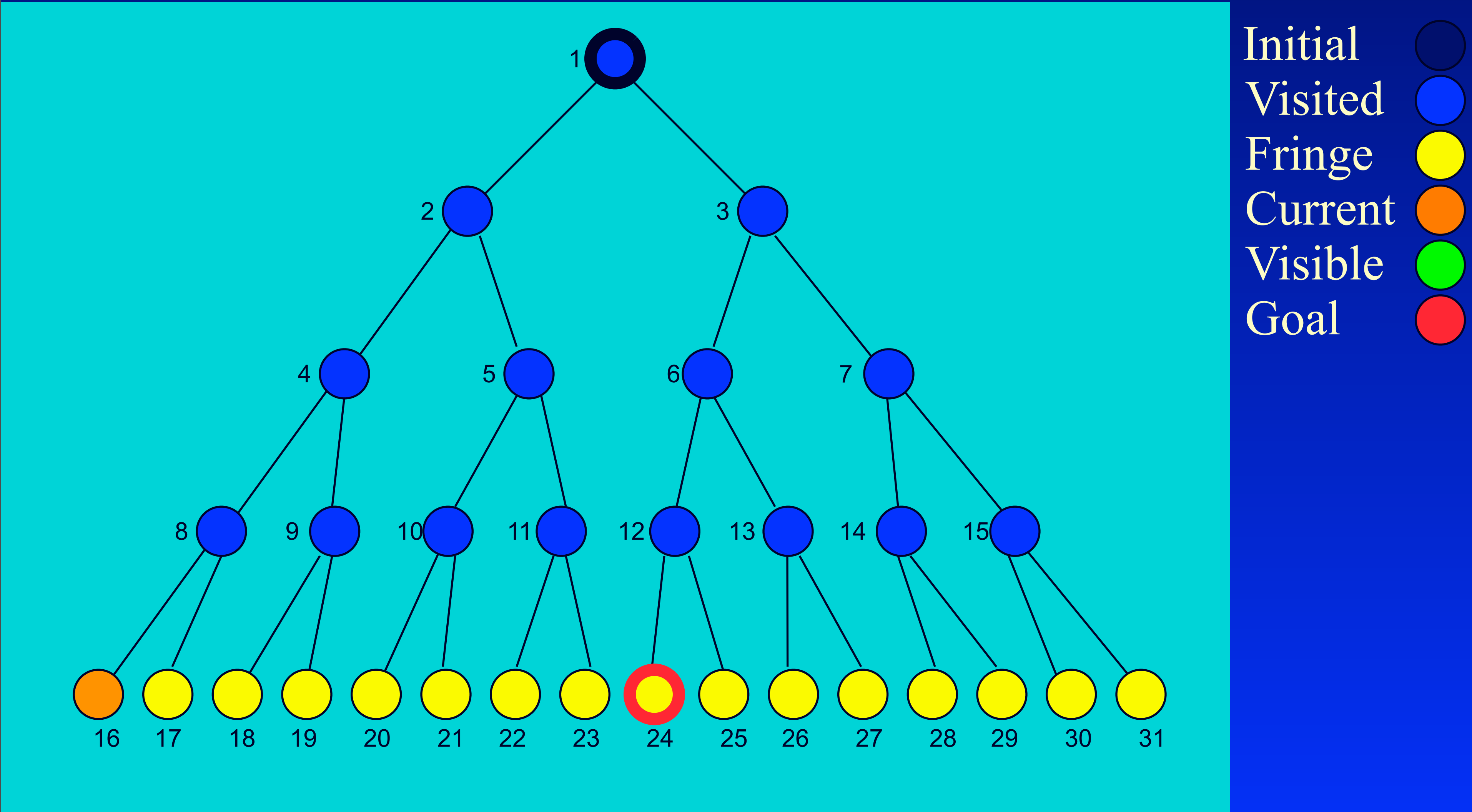
Fringe: [15,16,17,18,19,20,21,22,23,24,25,26,27] + [28,29]

Breadth-First Snapshot 15



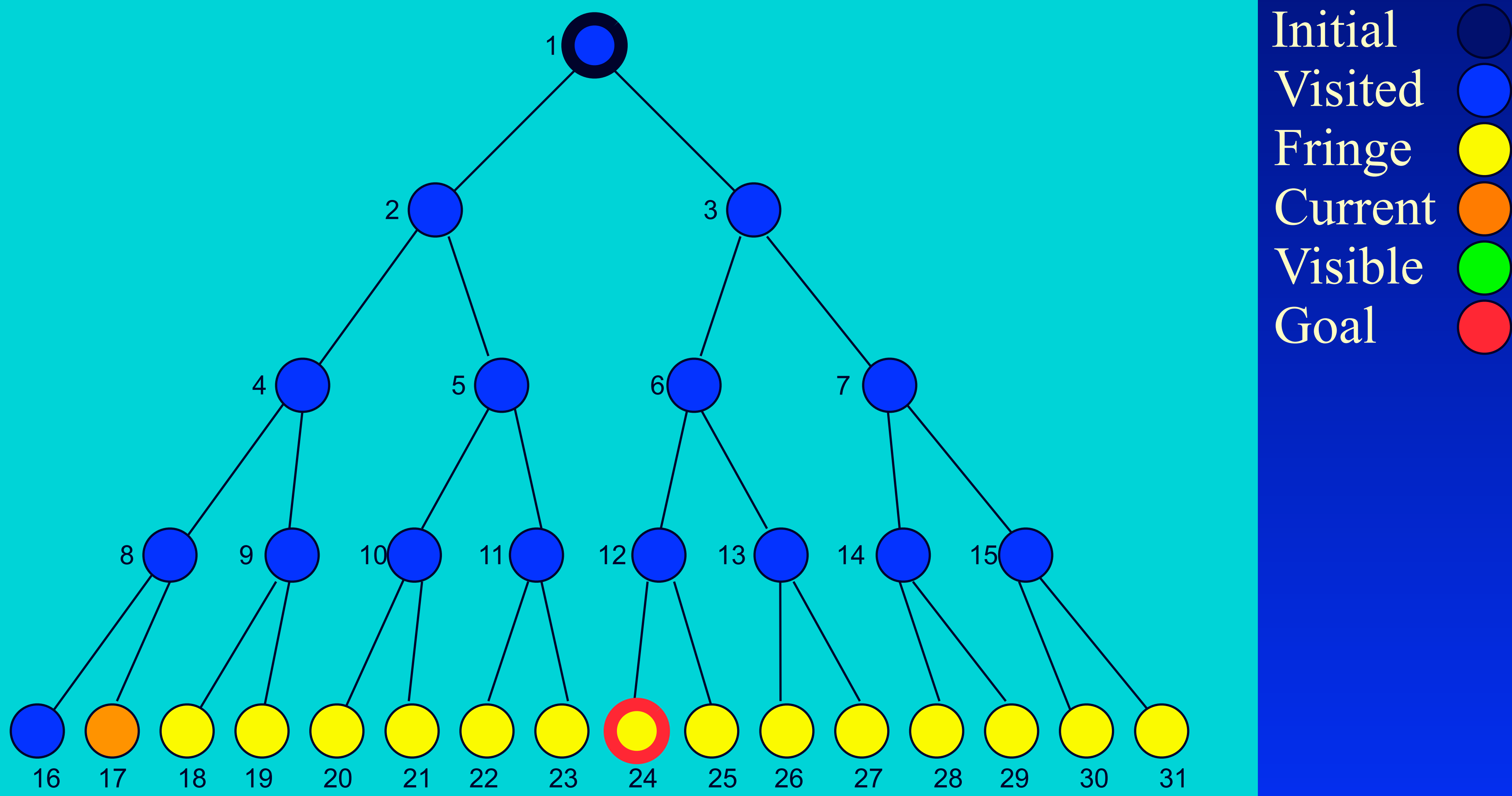
Fringe: [15,16,17,18,19,20,21,22,23,24,25,26,27,28,29] + [30,31]

Breadth-First Snapshot 16



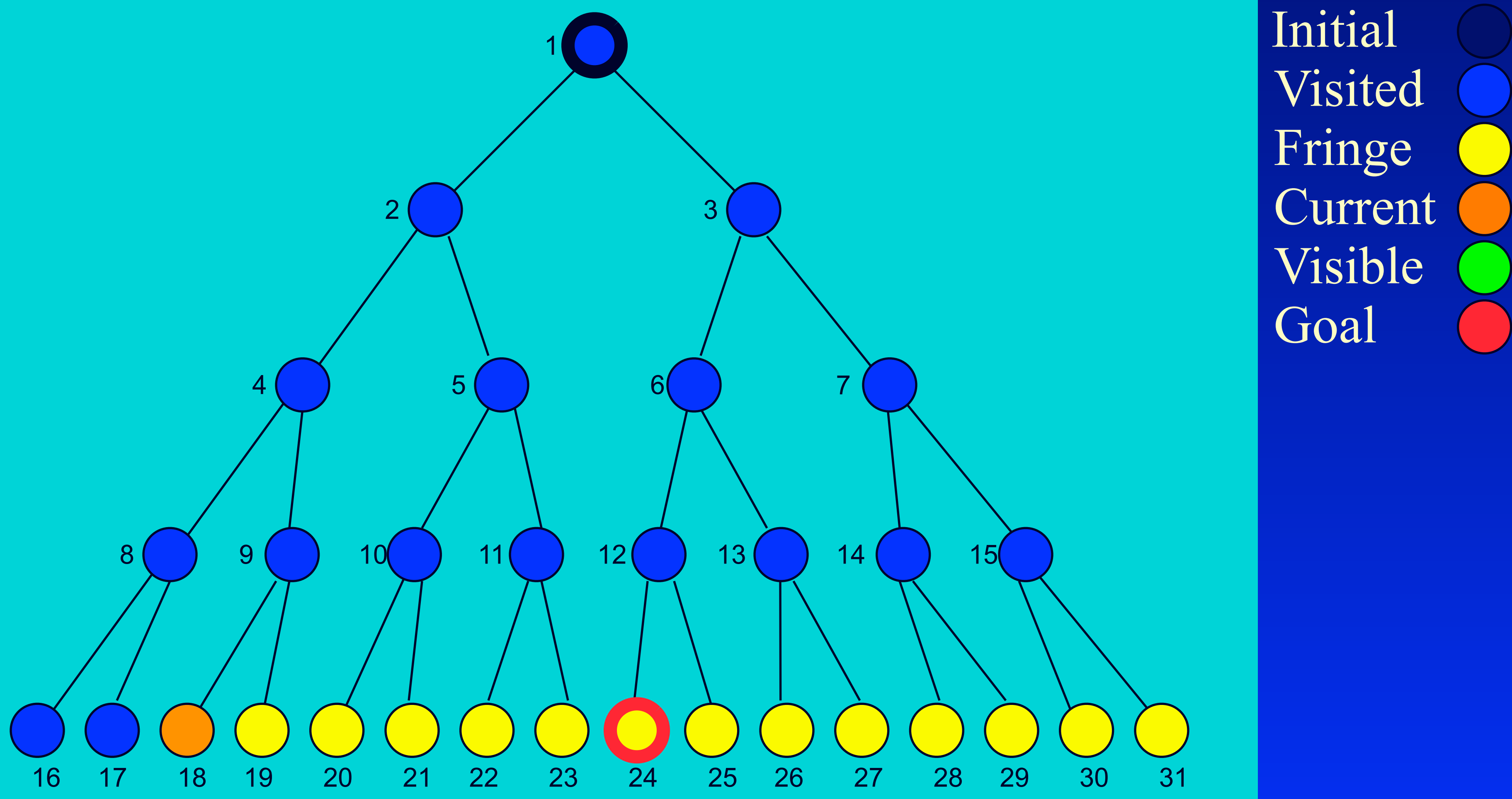
Fringe: [17,18,19,20,21,22,23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 17



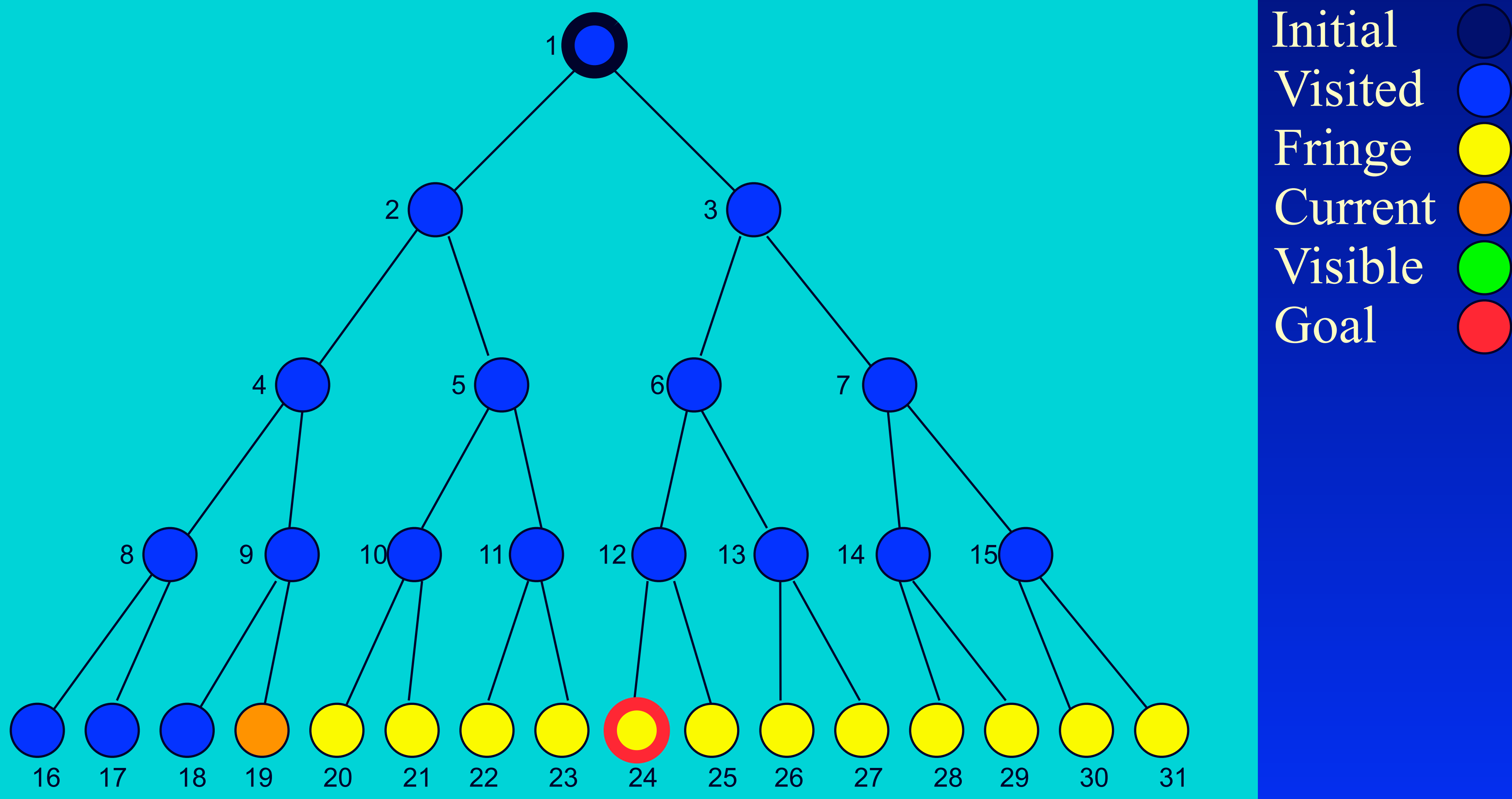
Fringe: [18,19,20,21,22,23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 18



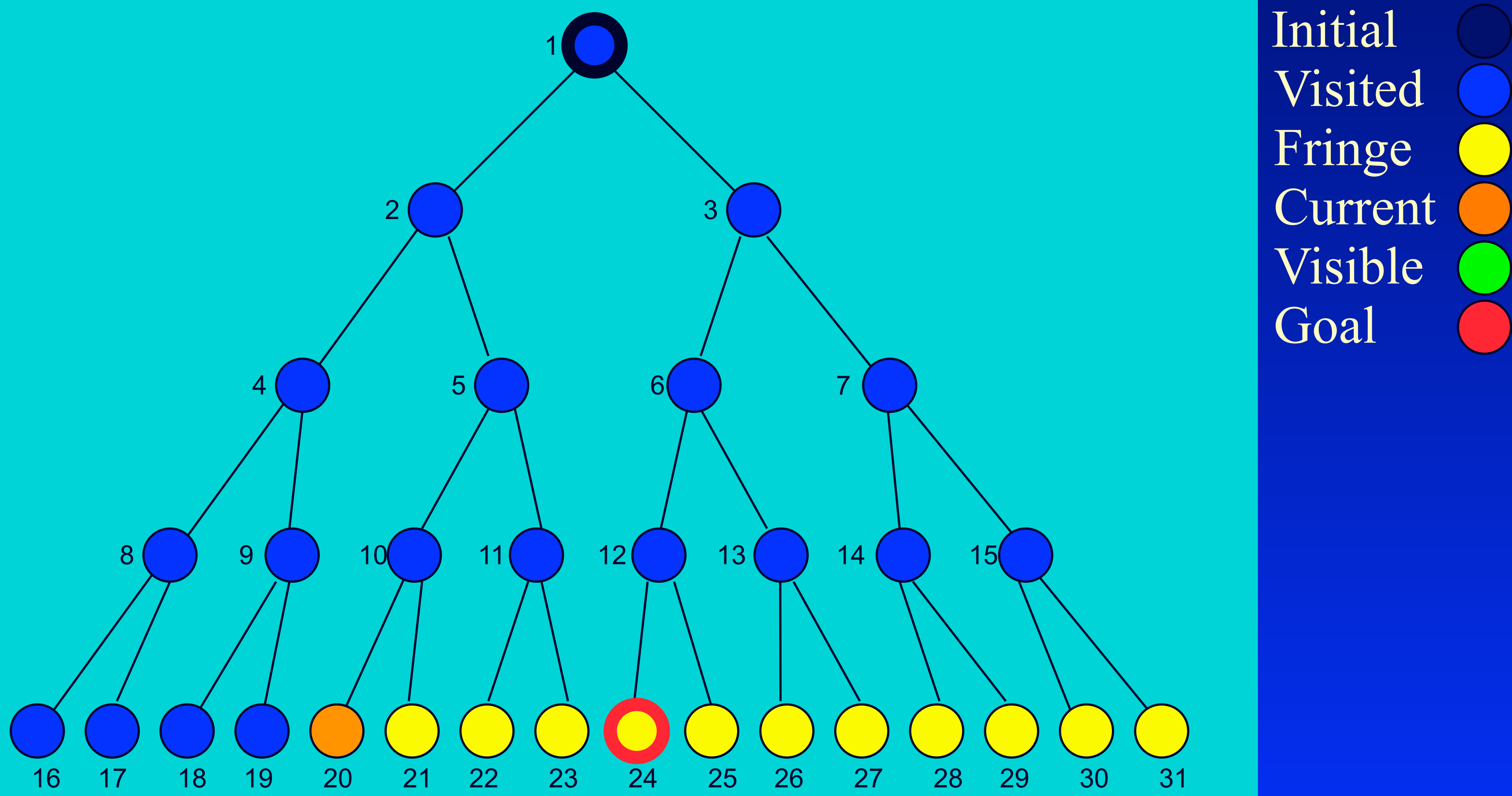
Fringe: [19,20,21,22,23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 19



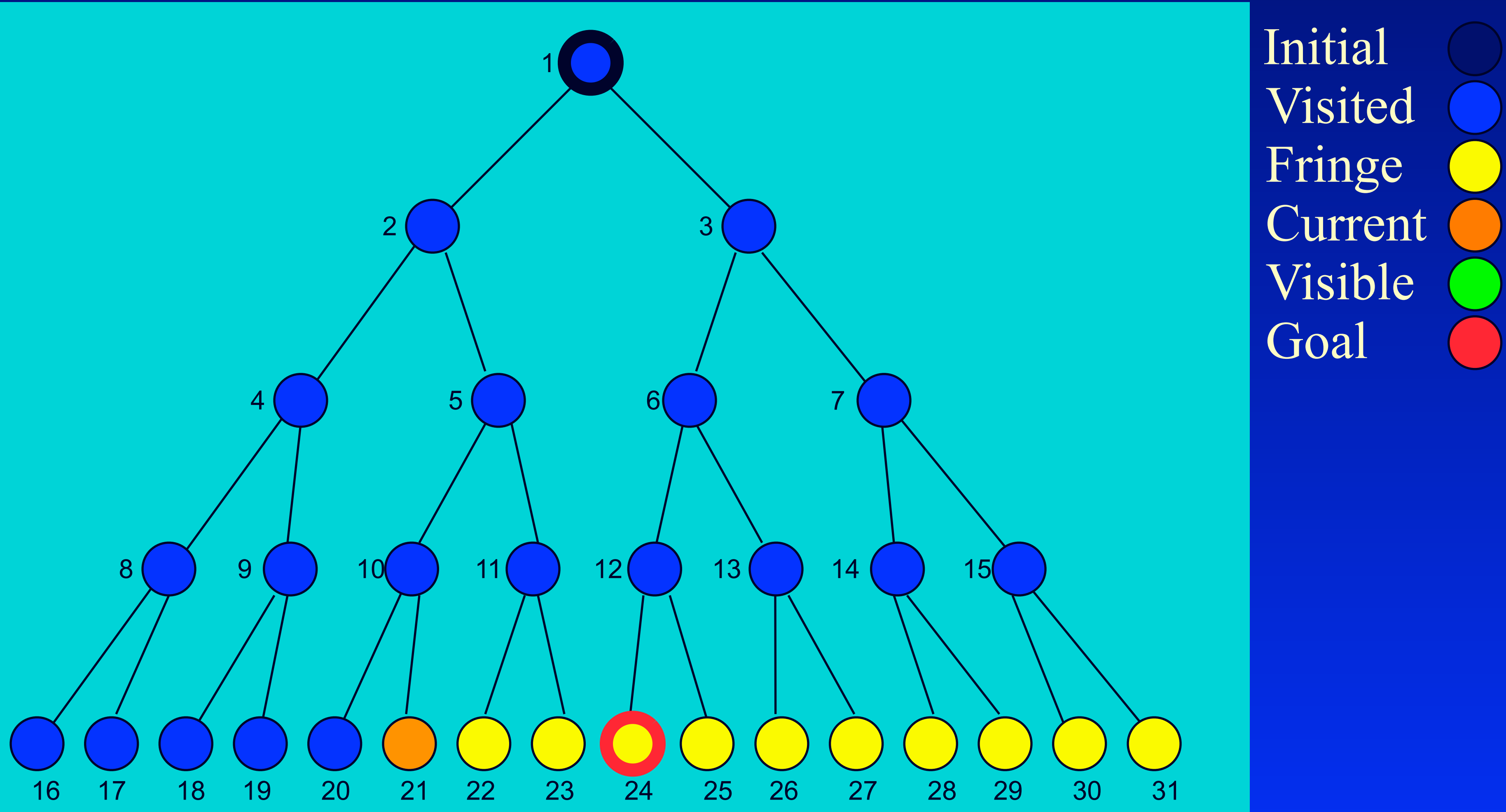
Fringe: [20,21,22,23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 20



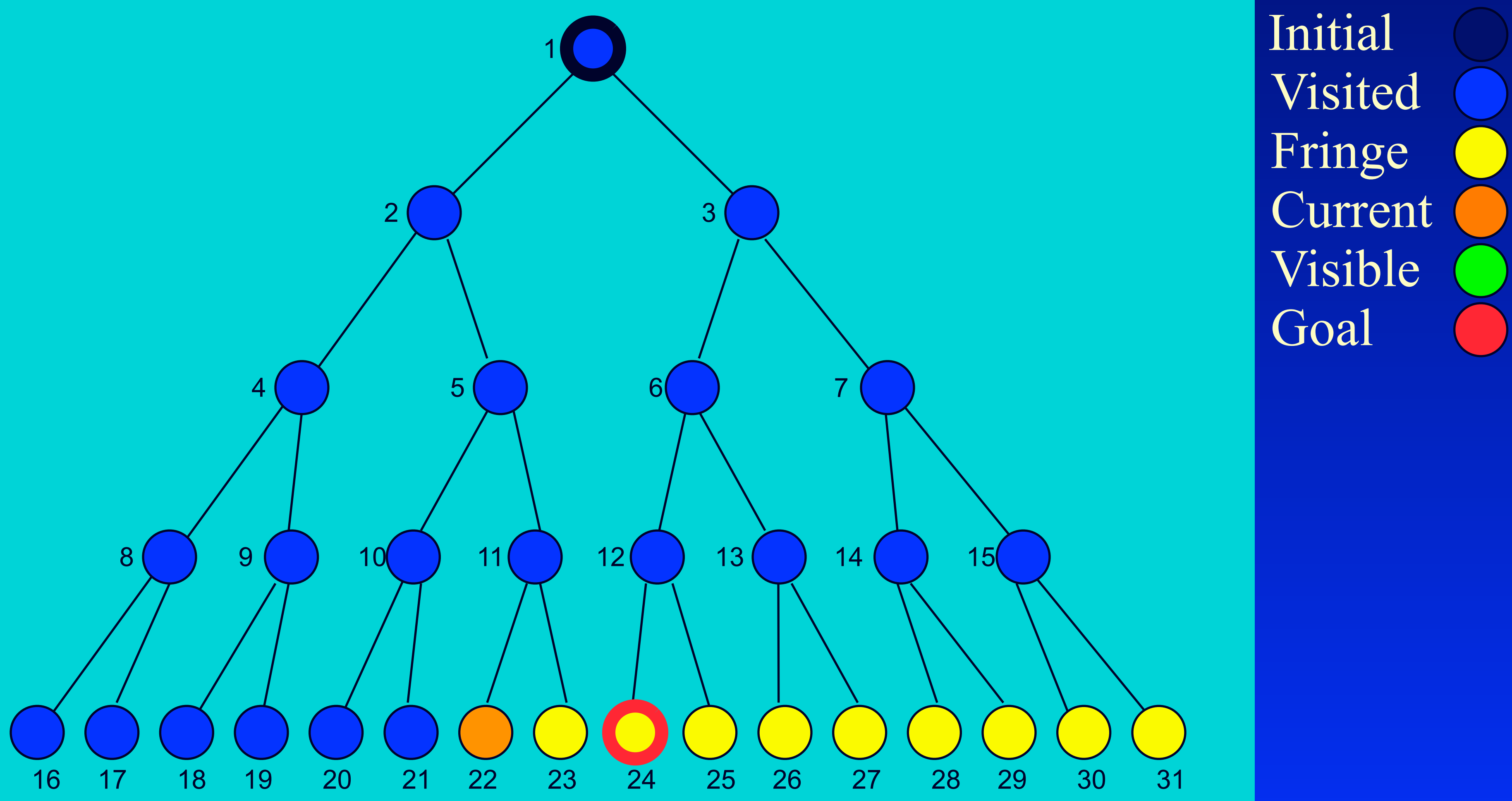
Fringe: [21,22,23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 21



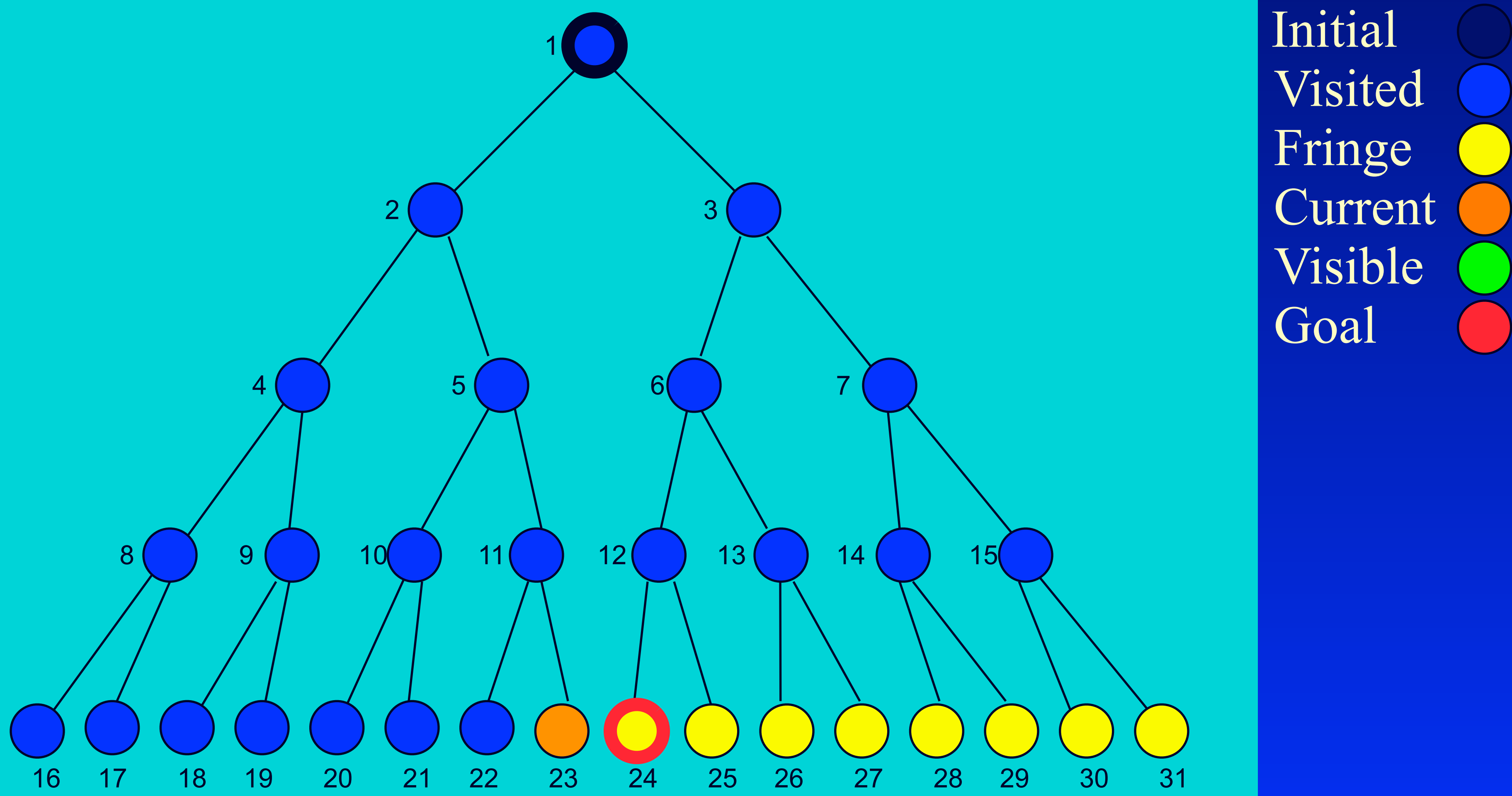
Fringe: [22,23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 22



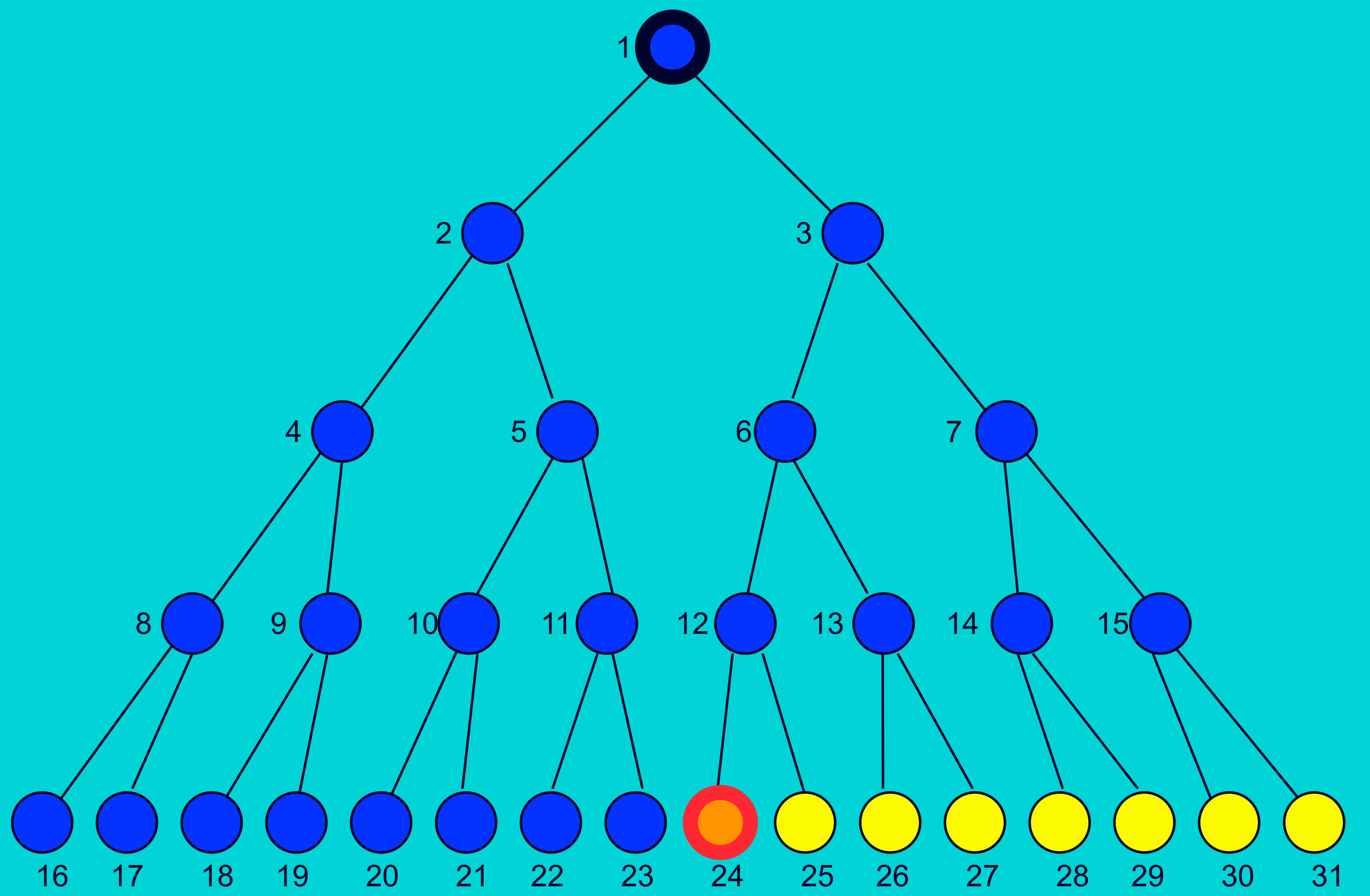
Fringe: [23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 23



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

Breadth-First Snapshot 24



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]

Uniform-Cost -First

- 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

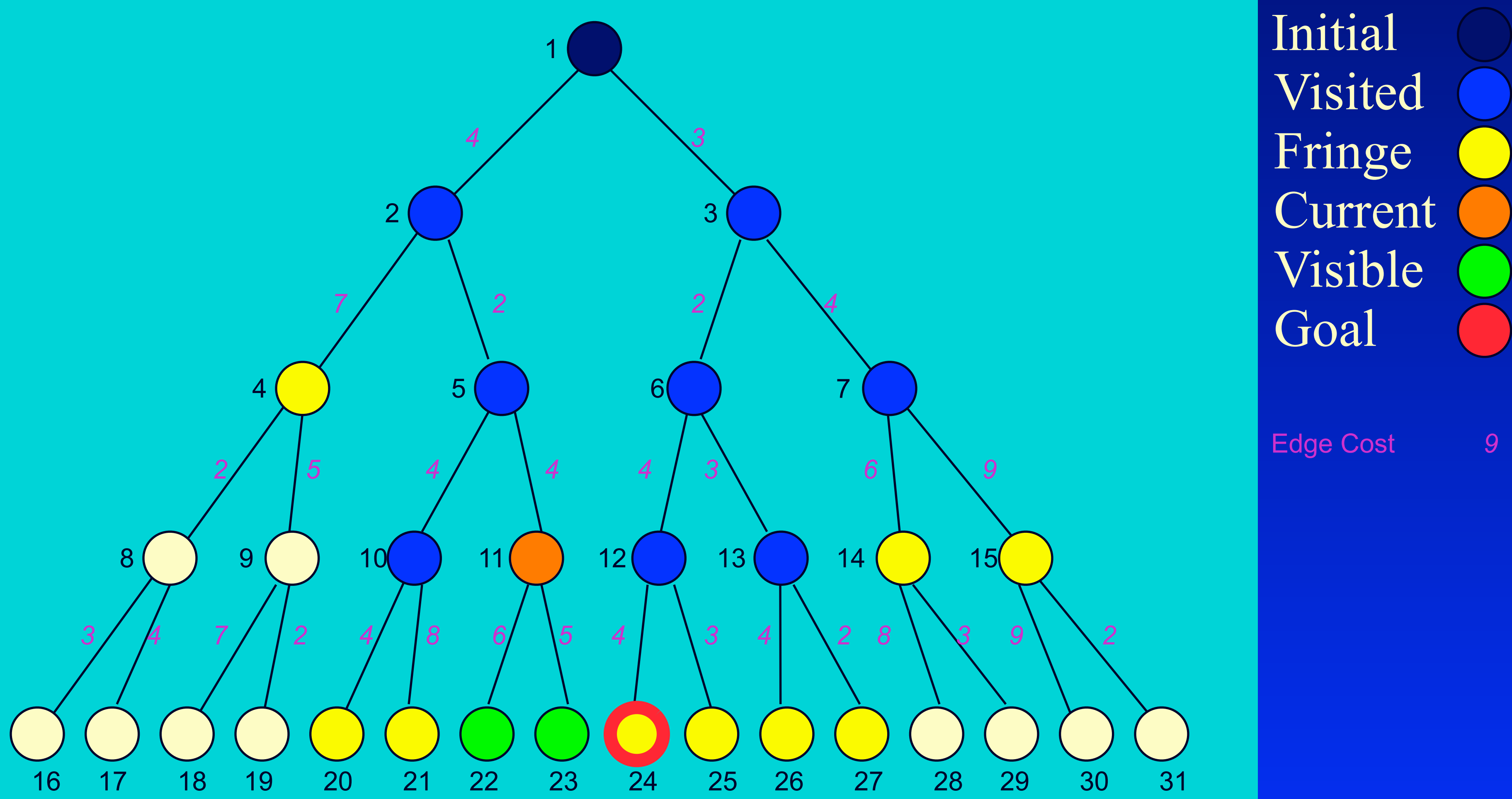
```
function UNIFORM-COST-SEARCH(problem) returns solution

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

Time Complexity	$b^{C^*/e}$
Space Complexity	$b^{C^*/e}$
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 Snapshot



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.

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

- continues exploring newly generated nodes
  - achieved by the TREE-SEARCH method by appending newly generated nodes at the beginning of the search queue
    - utilizes a Last-In, First-Out (LIFO) queue, or stack

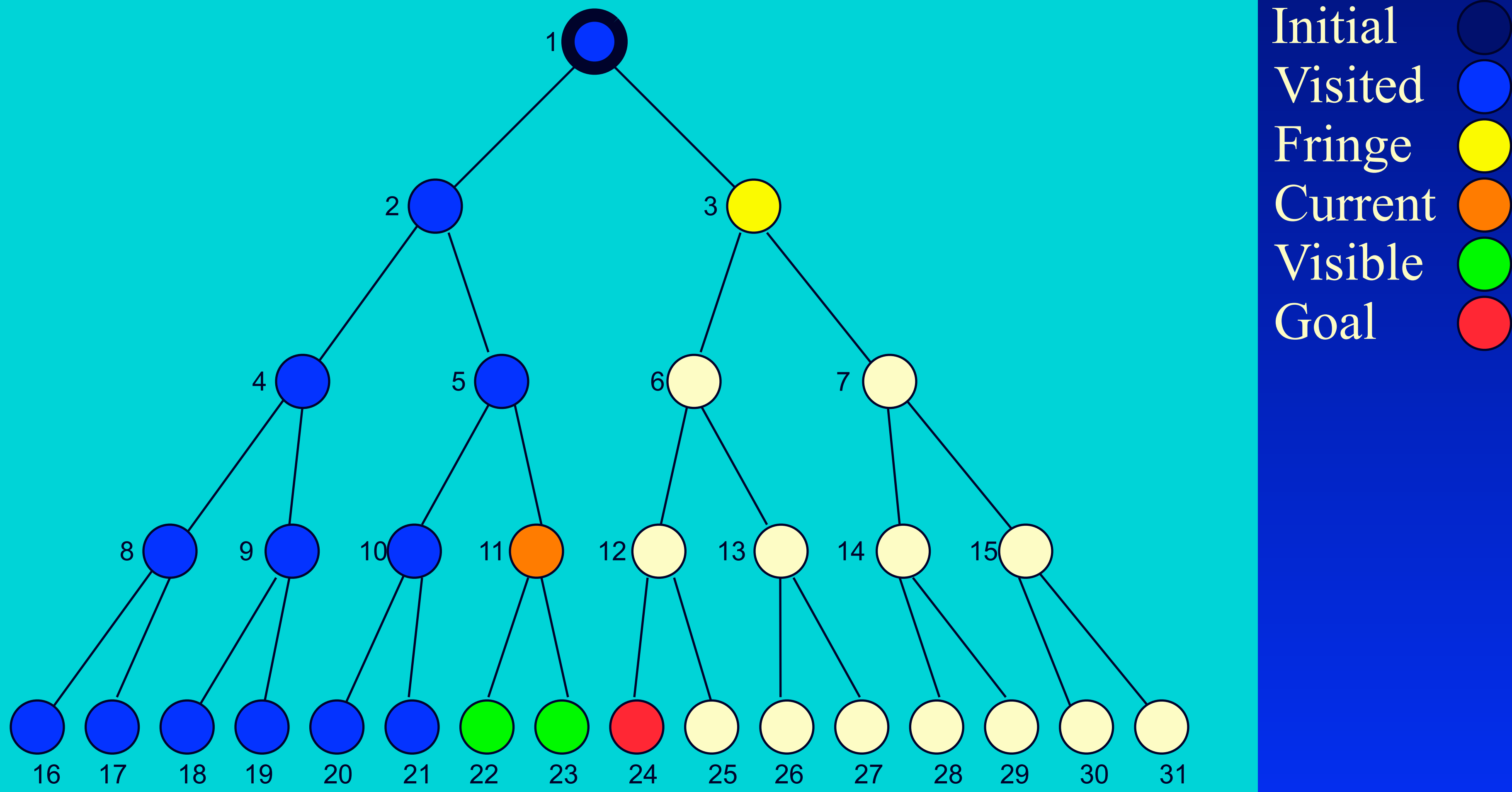
```
function DEPTH-FIRST-SEARCH(problem) returns solution

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

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

b	branching factor
m	maximum path length

Depth-First Snapshot



Fringe: [3] + [22,23]

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 of
    - smaller fringe
- breadth-first is more careful by checking all alternatives
  - complete and optimal
    - under most circumstances
  - very memory-intensive

Backtracking Search

- variation of depth-first search
  - only one successor node is generated at a time
    - even better space complexity: *O(m)* instead of *O(b*m)*
    - even more memory space can be saved by incrementally modifying the current state, instead of creating a new one
      - only possible if the modifications can be undone
      - this is referred to as *backtracking*
    - frequently used in planning, theorem proving

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
    - in other cases, a good depth limit is only known when the problem is solved
  - 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	b^l
Space Complexity	$b*l$
Completeness	no (goal beyond l, or infinite branch length)
Optimality	no

b	branching factor
l	depth limit

Iterative Deepening

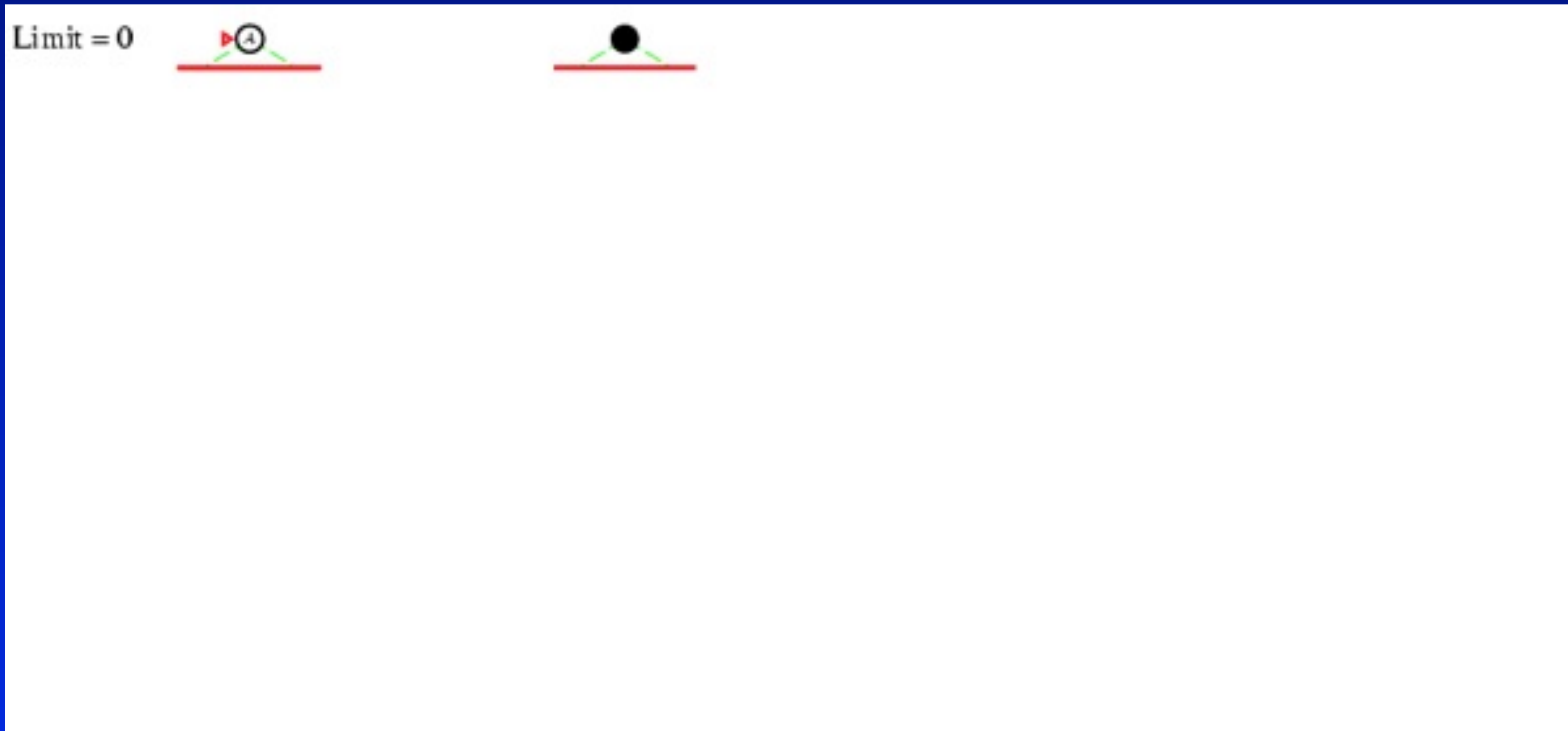
- applies LIMITED-DEPTH with increasing depth limits
  - combines advantages of BREADTH-FIRST and DEPTH-FIRST methods
  - many states are expanded multiple times
    - doesn’t really matter because the number of those nodes is small
  - in practice, one of the best uninformed search methods
    - for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

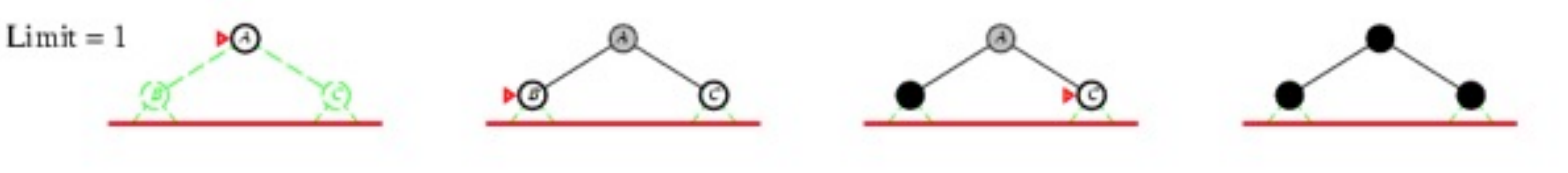
Time Complexity	b^d
Space Complexity	b*d
Completeness	yes (finite b)
Optimality	yes (all step costs identical)

b	branching factor
d	tree depth

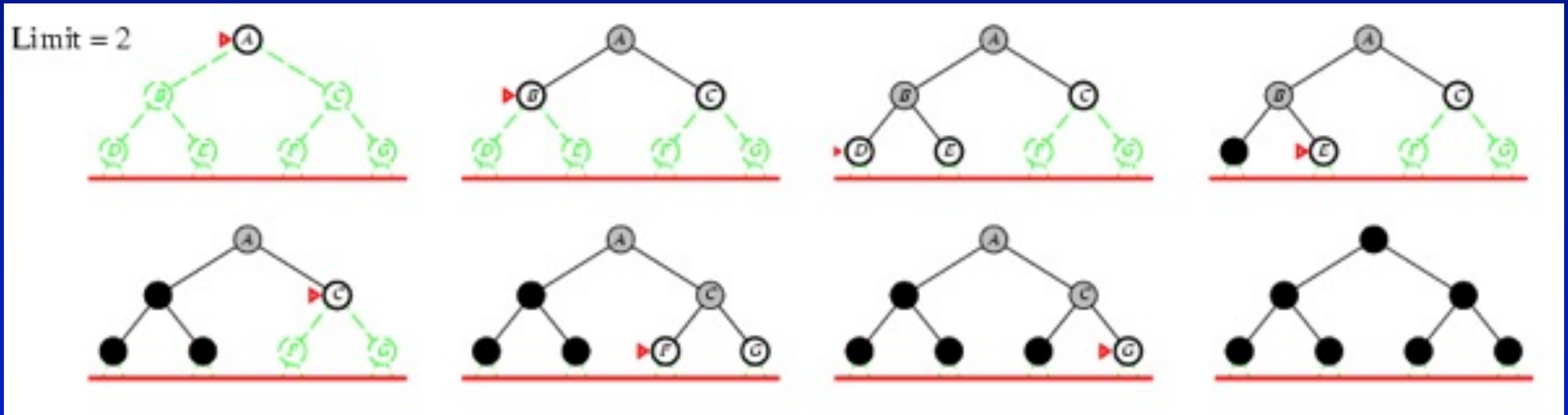
Iterative deepening search l =0



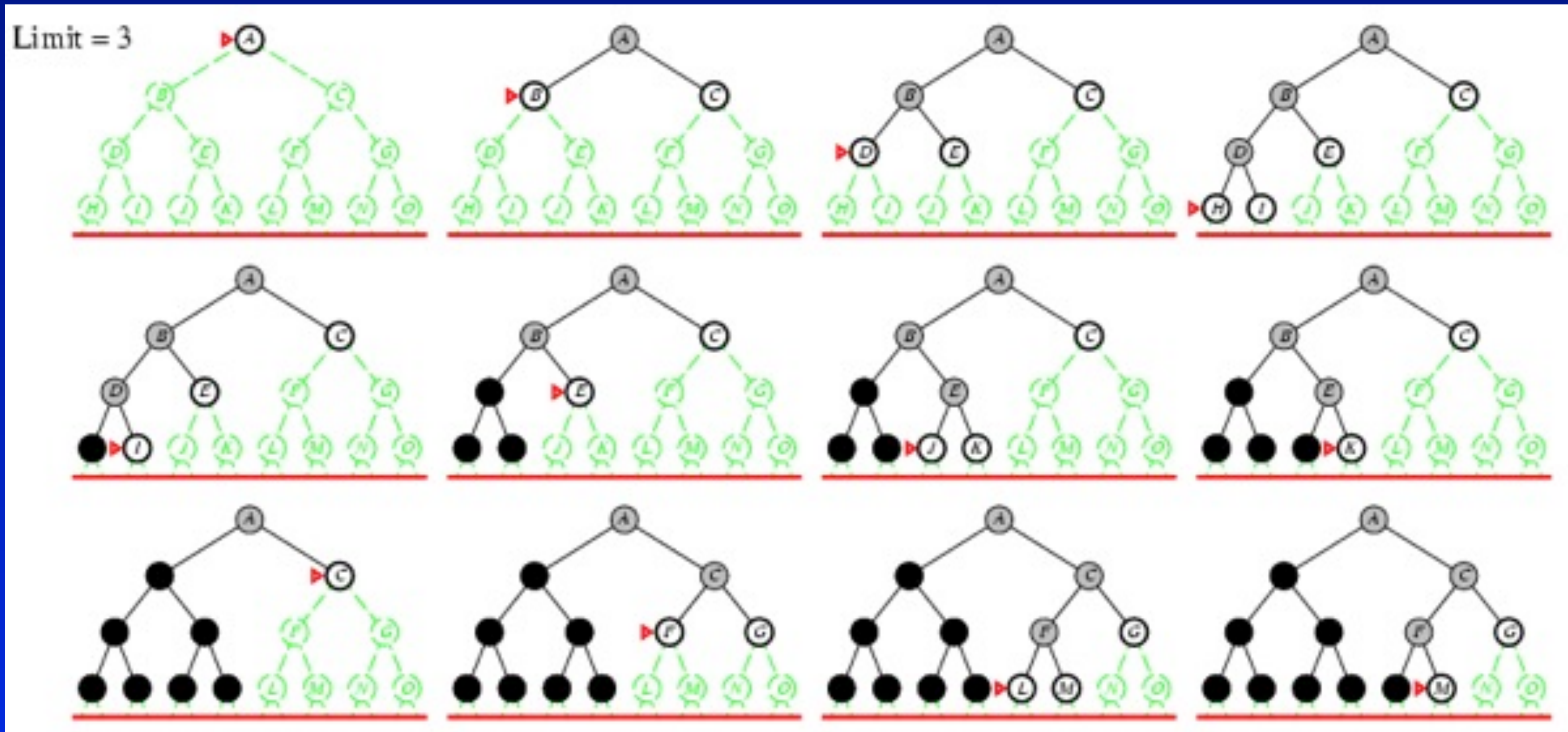
Iterative deepening search l =1



Iterative deepening search l =2



Iterative deepening search l =3



Iterative deepening search

- Number of nodes generated in a depth-limited search to depth d with branching factor b:

$$N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$$

- Number of nodes generated in an iterative deepening search to depth d with branching factor b:

$$N_{IDS} = (d+1)b^0 + d\ b\text{^}^1 + (d-1)b\text{^}^2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d$$

- For b = 10, d = 5,
  - N_{DLS} = 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111
  - N_{IDS} = 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456
- Overhead = (123,456 - 111,111)/111,111 = 11%

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

b	branching factor
d	tree depth

Improving Search Methods

- make algorithms more efficient
  - avoiding repeated states
  - utilizing memory efficiently
- use additional knowledge about the problem
  - 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

Avoiding Repeated States

- in many approaches, states may be expanded multiple times
  - e.g. iterative deepening
  - problems with reversible actions
- eliminating repeated states may yield an exponential reduction in search cost
  - e.g. some n-queens strategies
    - place queen in the left-most non-threatening column
  - rectangular grid
    - 4^d leaves, but only $2d^2$ distinct states

Informed Search

- relies on additional knowledge about the problem or domain
  - frequently expressed through heuristics (“rules of thumb”)
- used to distinguish more promising paths towards a goal
  - may be mislead, depending on the quality of the heuristic
- in general, performs much better than uninformed search
  - but frequently still exponential in time and space for realistic problems

Best-First Search

- relies on an evaluation function that gives an indication of how useful it would be to expand a node
  - family of search methods with various evaluation functions
  - usually gives an estimate of the distance to the goal
  - often referred to as *heuristics* in this context
- the node with the lowest value is expanded first
  - the name is a little misleading: the node with the lowest value for the evaluation function is not necessarily one that is on an optimal path to a goal
  - if we really know which one is the best, there's no need to do a search

```
function BEST-FIRST-SEARCH(problem, EVAL-FN) returns solution
    fringe := queue with nodes ordered by EVAL-FN

    return TREE-SEARCH(problem, fringe)
```

Greedy Best-First Search

- minimizes the estimated cost to a goal
  - expand the node that seems to be closest to a goal
  - utilizes a heuristic function as evaluation function
    - $f(n) = h(n)$ = estimated cost from the current node to a goal
    - heuristic functions are problem-specific
    - often straight-line distance for route-finding and similar problems
  - often better than depth-first, although worst-time complexities are equal or worse (space)

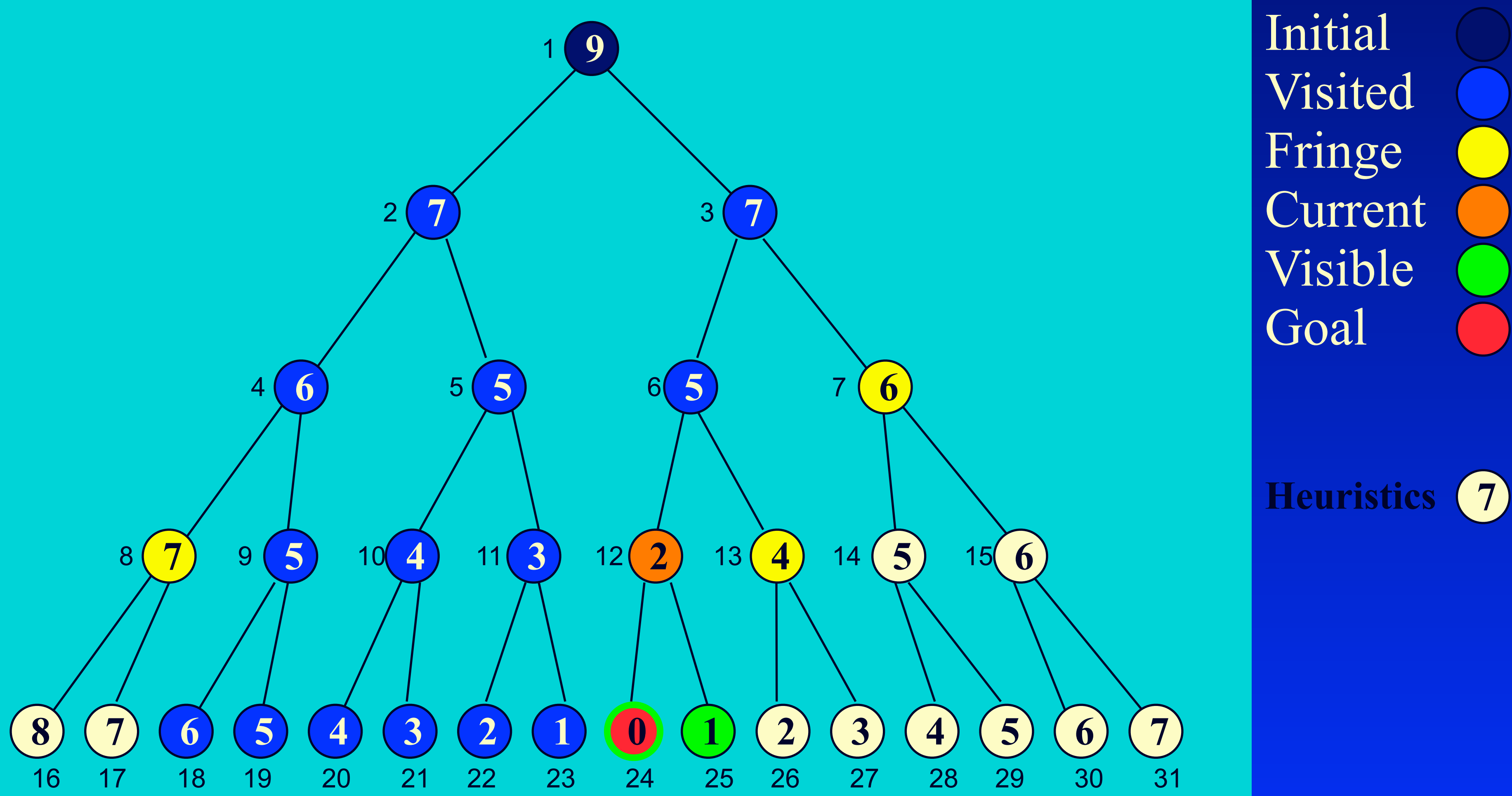
```
function GREEDY-SEARCH(problem) returns solution

return BEST-FIRST-SEARCH(problem, h)
```

Completeness	Time Complexity	Space Complexity	Optimality
no	b^m	b^m	no

b: branching factor, d: depth of the solution, m: maximum depth of the search tree, l: depth limit

Greedy Best-First Search Snapshot



Fringe: [13(4), 7(6), 8(7)] + [24(0), 25(1)]

Monday, October 8, 12

A* Search

- combines greedy and uniform-cost search to find the (estimated) cheapest path through the current node
  - $f(n) = g(n) + h(n)$
    = path cost + estimated cost to the goal
  - heuristics must be admissible
    - never overestimate the cost to reach the goal
  - very good search method, but with complexity problems

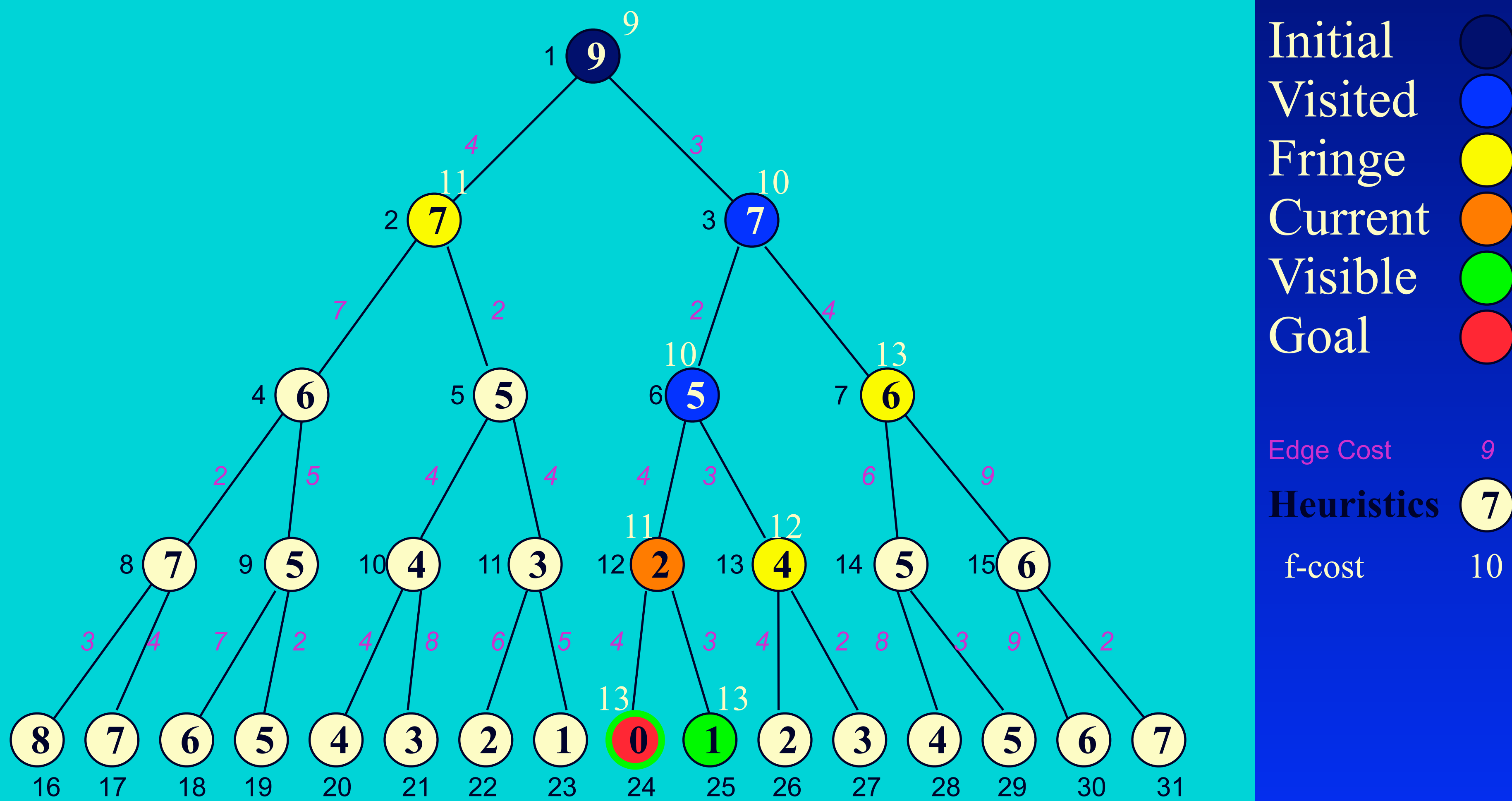
```
function A*-SEARCH(problem) returns solution

return BEST-FIRST-SEARCH(problem, g+h)
```

Completeness	Time Complexity	Space Complexity	Optimality
yes	b^d	b^d	yes

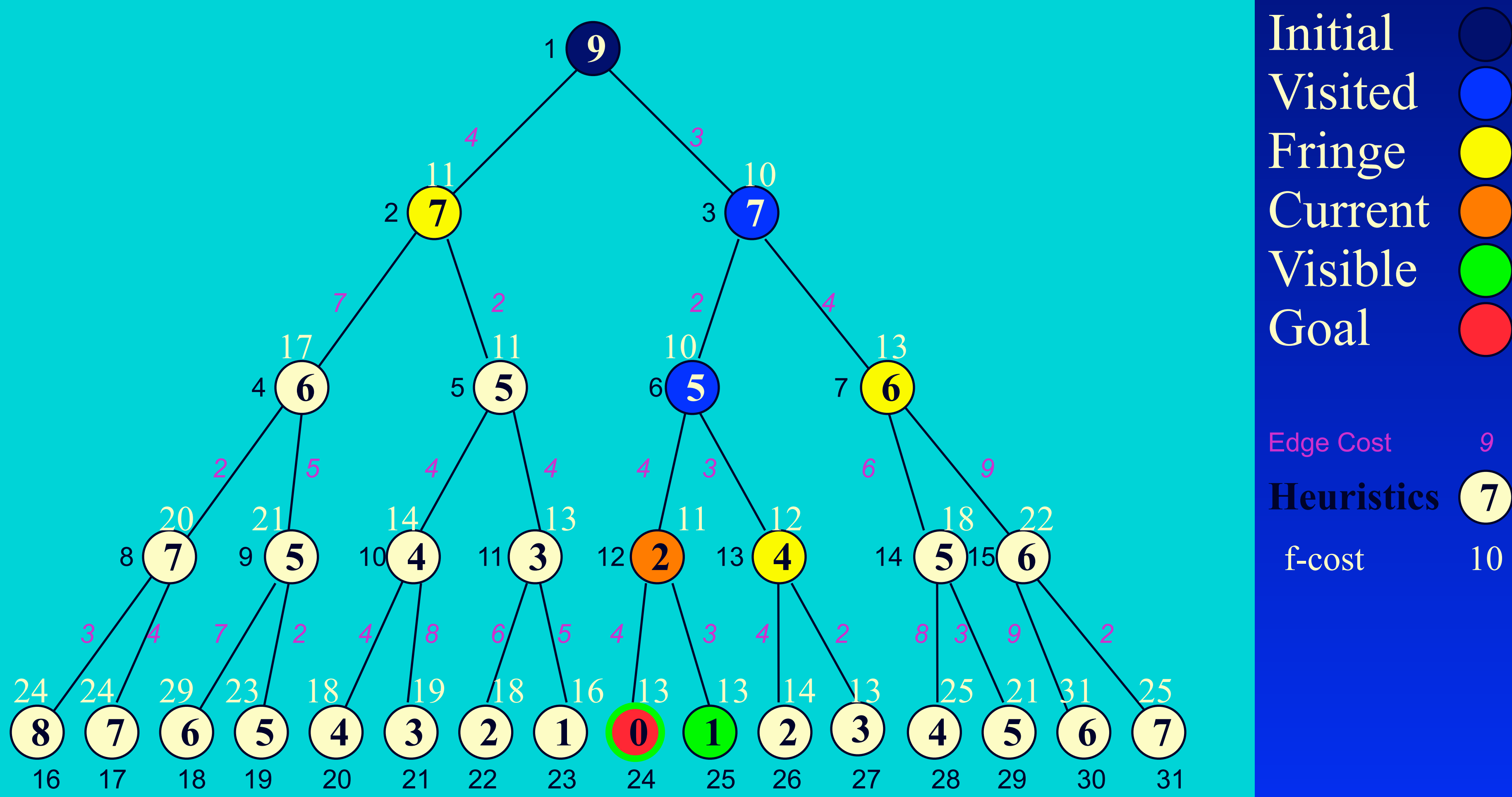
b: branching factor, d: depth of the solution, m: maximum depth of the search tree, l: depth limit

A* Snapshot



Fringe: [2(4+7), 13(3+2+3+4), 7(3+4+6)] + [24(3+2+4+4+0), 25(3+2+4+3+1)]

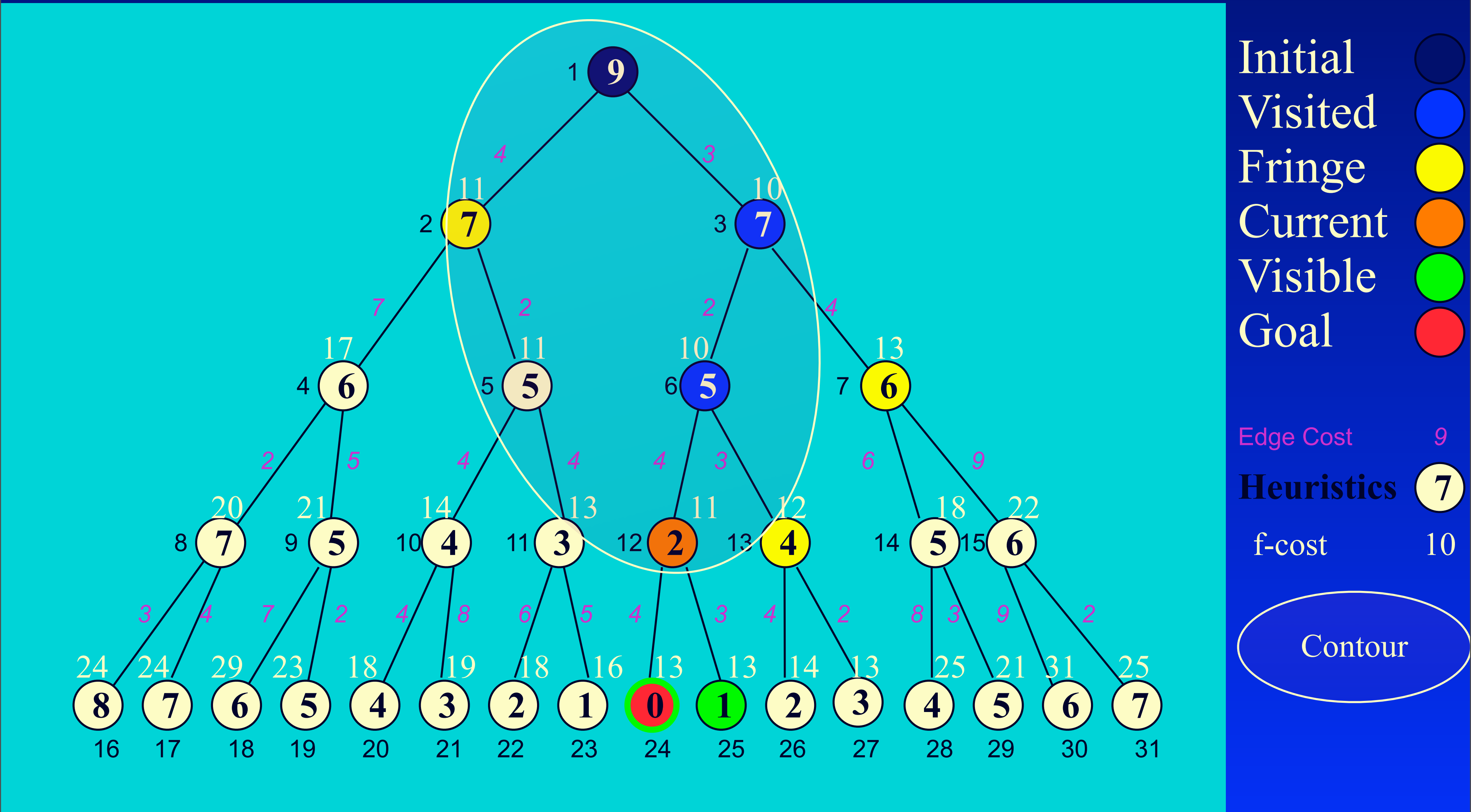
A* Snapshot with all f-Costs



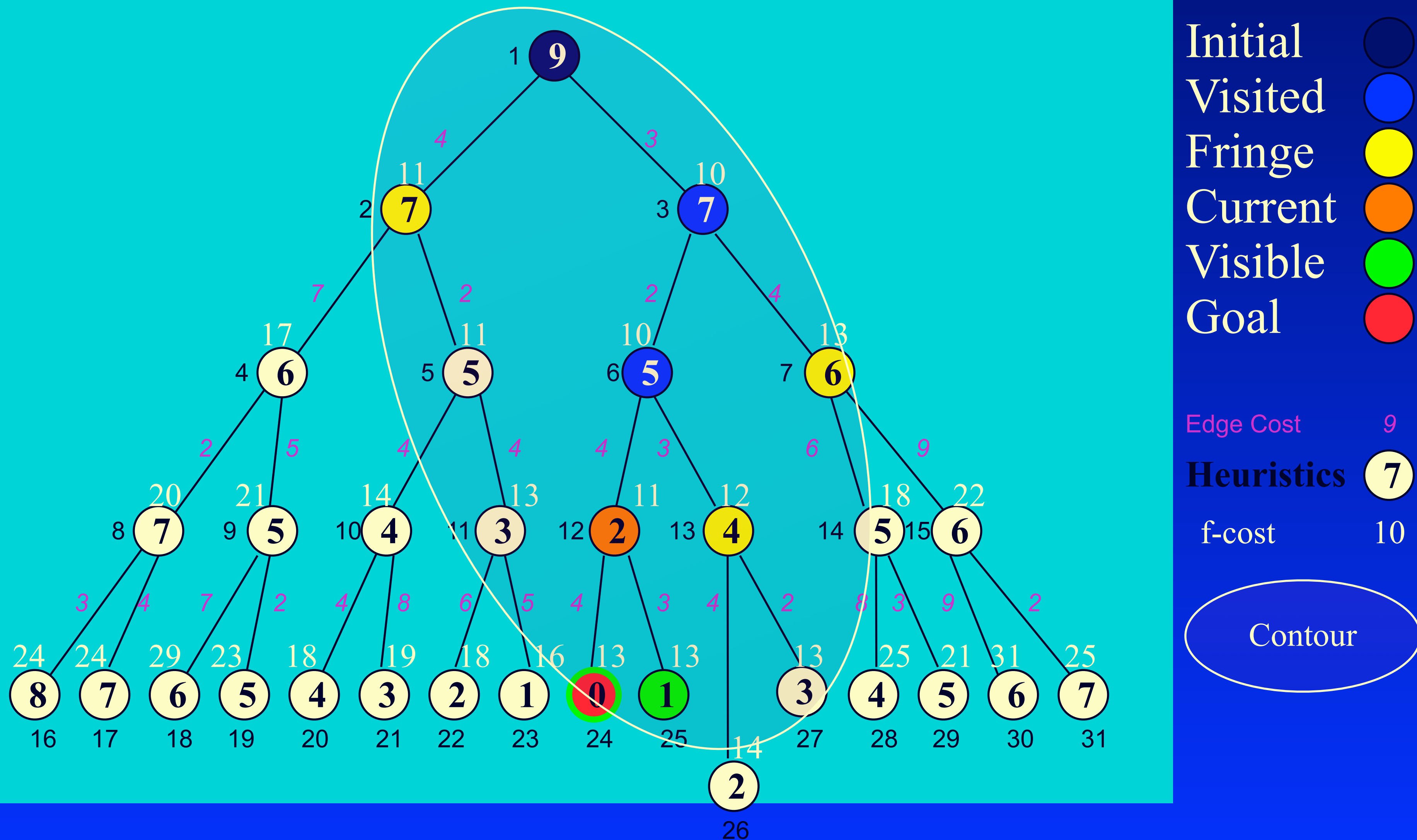
A* Properties

- the value of *f* never decreases along any path starting from the initial node
  - also known as monotonicity of the function
  - almost all admissible heuristics show monotonicity
    - those that don't can be modified through minor changes
- this property can be used to draw contours
  - regions where the f-cost is below a certain threshold
  - with uniform cost search ($h = 0$), the contours are circular
  - the better the heuristics *h*, the narrower the contour around the optimal path

A* Snapshot with Contour f=11



A* Snapshot with Contour f=13



Optimality of A*

- A* will find the optimal solution
  - the first solution found is the optimal one
- A* is optimally efficient
  - no other algorithm is guaranteed to expand fewer nodes than A*
- A* is not always “the best” algorithm
  - optimality refers to the expansion of nodes
    - other criteria might be more relevant
  - it generates and keeps all nodes in memory
    - improved in variations of A*

Complexity of A*

- the number of nodes within the goal contour search space is still exponential
  - with respect to the length of the solution
  - better than other algorithms, but still problematic
- frequently, space complexity is more severe than time complexity
  - A* keeps all generated nodes in memory

Memory-Bounded Search

- search algorithms that try to conserve memory
- most are modifications of A*
  - iterative deepening A* (IDA*)
  - simplified memory-bounded A* (SMA*)

Iterative Deepening A* (IDA*)

- explores paths within a given contour (f-cost limit) in a depth-first manner
  - this saves memory space because depth-first keeps only the current path in memory
    - but it results in repeated computation of earlier contours since it doesn't remember its history
  - was the "best" search algorithm for many practical problems for some time
  - does have problems with difficult domains
    - contours differ only slightly between states
    - algorithm frequently switches back and forth
      - similar to disk thrashing in (old) operating systems

Recursive Best-First Search

- similar to best-first search, but with lower space requirements
  - $O(bd)$ instead of $O(b^m)$
- it keeps track of the best alternative to the current path
  - best f-value of the paths explored so far from predecessors of the current node
  - if it needs to re-explore parts of the search space, it knows the best candidate path
  - still may lead to multiple re-explorations

Simplified Memory-Bounded A* (SMA*)

- uses all available memory for the search
  - drops nodes from the queue when it runs out of space
    - those with the highest f-costs
  - avoids re-computation of already explored area
    - keeps information about the best path of a “forgotten” subtree in its ancestor
  - complete if there is enough memory for the shortest solution path
  - often better than A* and IDA*
    - but some problems are still too tough
    - trade-off between time and space requirements

Heuristics for Searching

- for many tasks, a good heuristic is the key to finding a solution
  - prune the search space
  - move towards the goal
- relaxed problems
  - fewer restrictions on the successor function (operators)
  - its exact solution may be a good heuristic for the original problem

8-Puzzle Heuristics

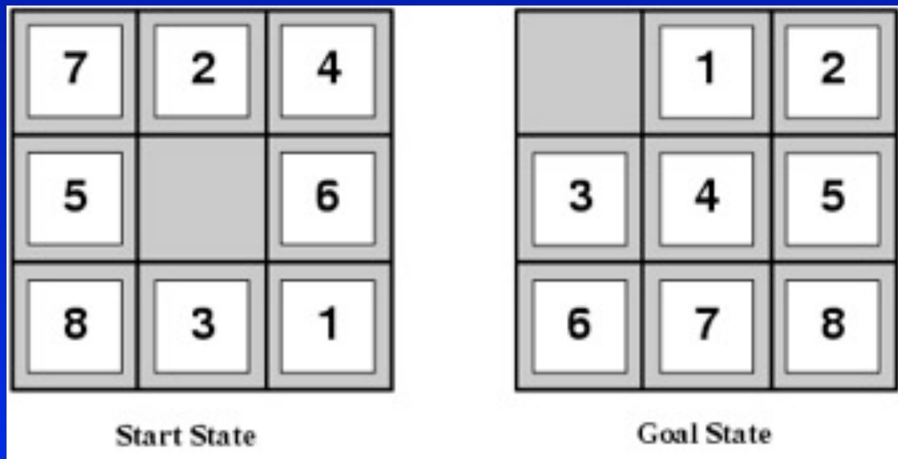
- level of difficulty
  - around 20 steps for a typical solution
  - branching factor is about 3
  - exhaustive search would be 3^{20} =3.5 * 10^9
  - 9!/2 = 181,440 different reachable states
    - distinct arrangements of 9 squares
- candidates for heuristic functions
  - number of tiles in the wrong position
  - sum of distances of the tiles from their goal position
    - city block or Manhattan distance
- generation of heuristics
  - possible from formal specifications

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



- $h_1(S) = ?$ 8
- $h_2(S) = ?$ 3+1+2+2+2+3+3+2 = 18

Important Concepts and Terms

- agent
- A* search
- best-first search
- bi-directional search
- breadth-first search
- depth-first search
- depth-limited search
- completeness
- constraint satisfaction
- depth-limited search
- genetic algorithm
- general search algorithm
- goal
- goal test function
- greedy best-first search
- heuristics
- initial state
- iterative deepening search
- iterative improvement
- local search
- memory-bounded search
- operator
- optimality
- path
- path cost function
- problem
- recursive best-first search
- search
- space complexity
- state
- state space
- time complexity
- uniform-cost search

Chapter Summary

- tasks can often be formulated as search problems
  - initial state, successor function (operators), goal test, path cost
- various search methods systematically comb the search space
  - uninformed search
    - breadth-first, depth-first, and variations
  - informed search
    - best-first, A*, iterative improvement
- the choice of good heuristics can improve the search dramatically
  - task-dependent