CSC 480: Artificial Intelligence

Franz J. Kurfess

Visiting Professor
Department of Computer Science and Mathematics
Munich University of Applied Sciences
Germany

Professor
Computer Science Department
California Polytechnic State University
San Luis Obispo, CA, U.S.A.

Logistics Project

- **Moodle: Artificial Intelligence (Kurfess)**
  - enrollment key: AI-MUAS-S15
  - groups being set up for project teams
  - project description
    - project overview
    - tentative schedule
    - features, requirements, evaluation criteria

Logistics

- **Lab and Homework Assignments**
  - Lab 2 due tonight (23:59); demos during the lab time
  - Lab 3: implementation of breadth-first, depth-first search
- **Quizzes**
  - Quiz 2
    - Available Tue, April 14, all day (0:00 - 23:59)
    - longer availability of quizzes?

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**

Motivation and Objectives

Pre-Test
Motivation
Objectives
Evaluation Criteria

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

Background

What is Search in AI?
Problem-Solving and Search
Terminology

Search as Problem-Solving Strategy

- **many problems can be viewed as reaching a goal state from a given starting point**
  - often there is an underlying state space that defines the problem and its possible solutions in a more formal way
  - the space can be traversed by applying a successor function (operators) to proceed from one state to the next
  - if possible, information about the specific problem or the general domain is used to improve the search
    - experience from previous instances of the problem
    - strategies expressed as heuristics
    - simpler versions of the problem
    - constraints on certain aspects of the problem

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

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 points or periods during the problem-solving process**
  - dependent on the task and domain
- **actions move the agent from one state to another one**
  - an operator is applied to the initial state and takes the agent to the successor 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: Commute Home => MUAS

- **states**
  - locations:
    - obvious: buildings that contain your home, MUAS Computer Science building
    - more difficult: intermediate states
      - public transit: stops (bus, street car, U-Bahn, S-Bahn)
      - walking: ??
      - bicycling: ??
      - car: ??
  - environment-centric states
    - e.g., public transit stops
  - 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

Search Example: Traveling by Car

- **Route finding: Starting Point => Goal Point**
  - focus on
    - communities, landmarks
    - highways, streets, intersections
    - street addresses
  - constraints
    - one-way, avoid highways/ferries/toll roads, scenic vs. direct, ...
    - traffic considerations

Search Example: Traveling by Train

- **Route finding: Starting Station => Goal Station**
  - focus on railway network
    - train stations
  - constraints
    - train type: ICE vs. regional
    - passenger special needs: bicycle transport
    - traffic considerations

Example Problems for Search

Toy Problems
Real-World Problems

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

http://www.google.com/url?sa=i&source=images&cd=&cad=rja&docid=Pz3Wn3dXJTa6wM&tbnid=wU_RhC16EqguGM:&ved=&url=http%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DOf2HU3LGdbo&ei=ihISUpO9KuvMigKJuYCADw&psig=AFQjCNHI8trHQoemZh0VyNhoWvcDwSatMQ&ust=1381198602732118

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

CAL POLY

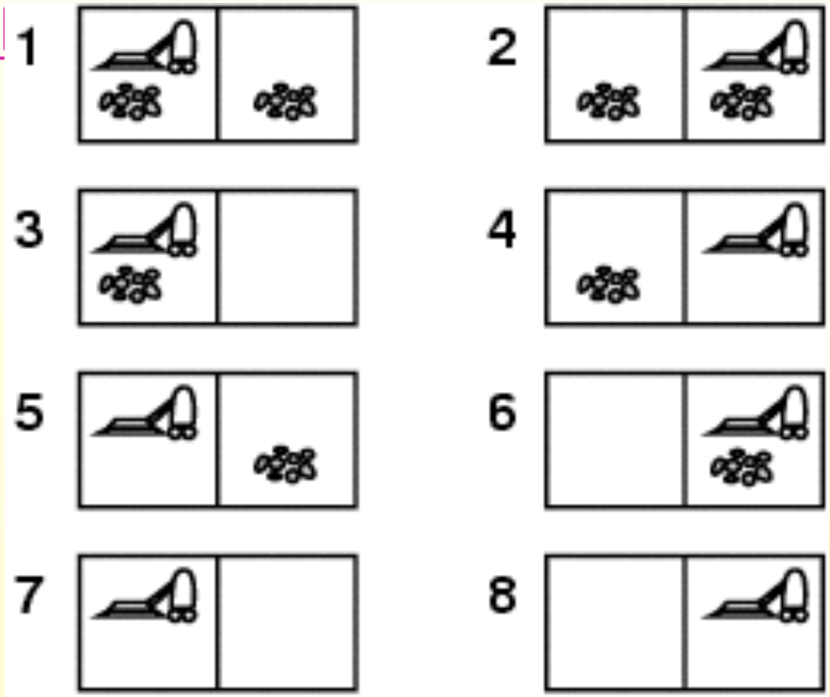
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Vacuum World Example

- from the AIMA textbook slides
- conversion from PowerPoint to Keynote

Example: vacuum world

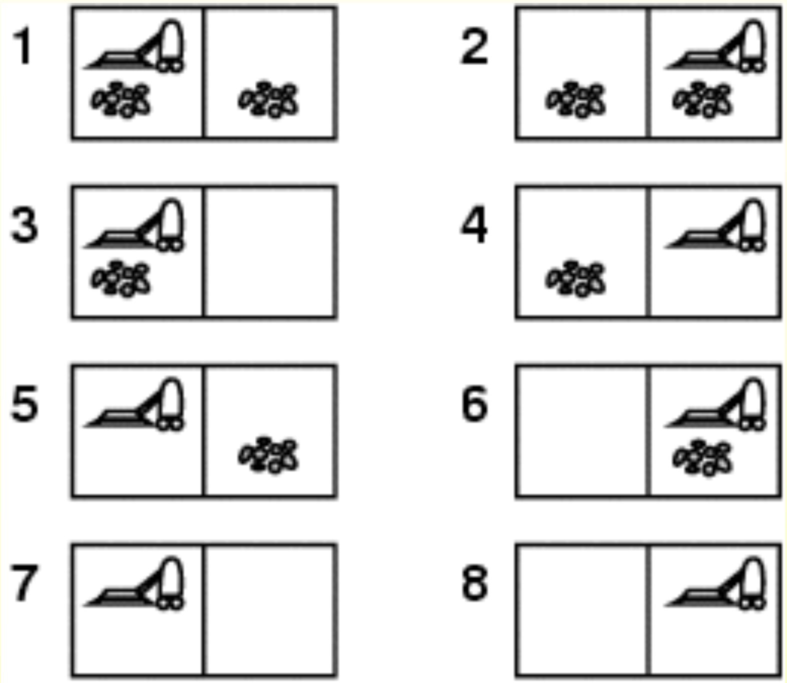
- Single-state, start in #5. So



Example: vacuum world

- Single-state, start in #5.
  Solution? [Right, Suck]

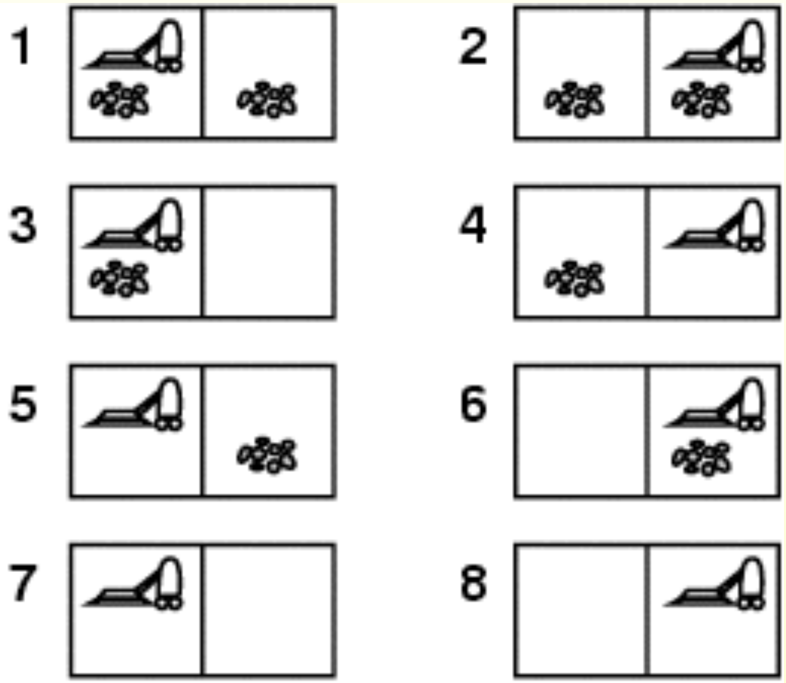
- Sensorless, start in {1,2,3,4,5,6,7,8} e.g., Right goes to {2,4,6,8}
  Solution?



http://aima.eecs.berkeley.edu/slides-ppt/

Example: vacuum world

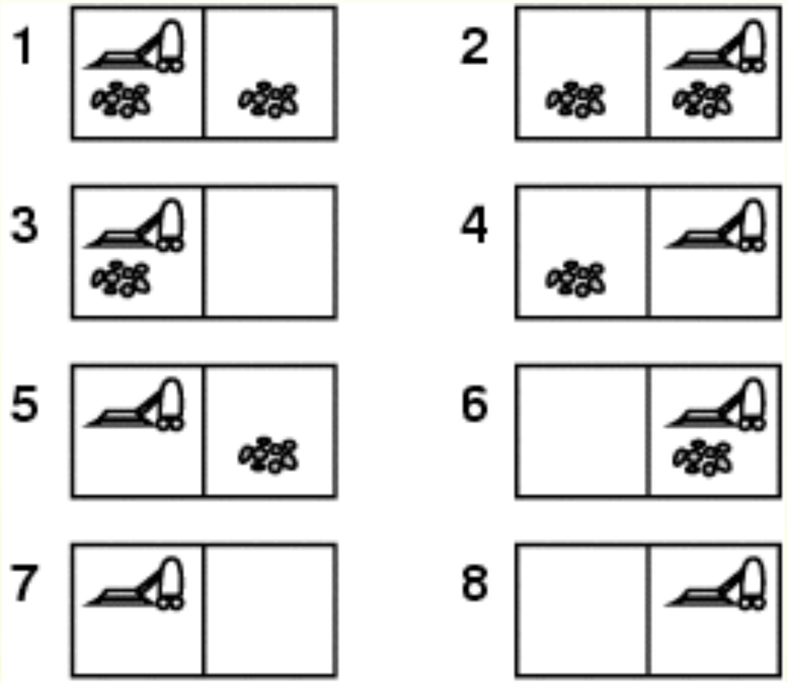
- **Sensorless, start in {1,2,3,4,5,6,7,8} e.g., Right goes to {2,4,6,8}**
  Solution?
  [Right,Suck,Left,Suck]



- **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?

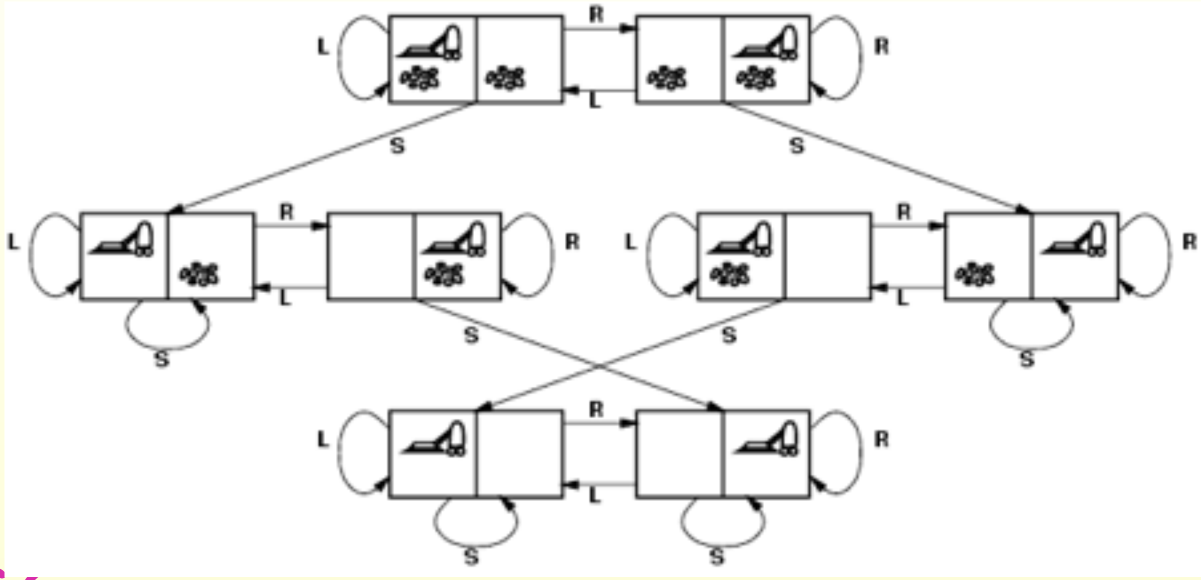
Example: vacuum world

- **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]

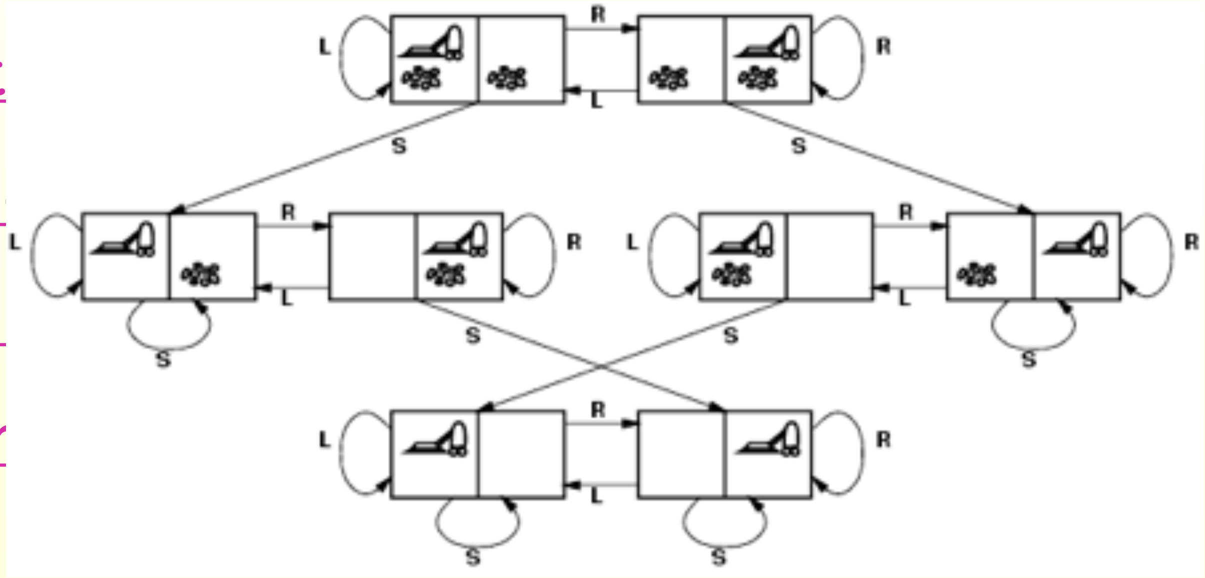
Vacuum world state space graph



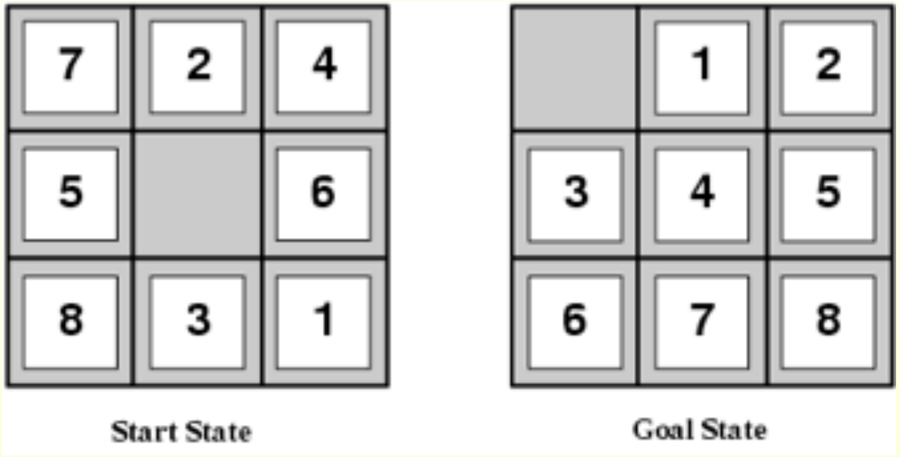
- states?
- actions?
- goal test?
- path cost?

Vacuum world state space graph

- stat
- acti
- goa
- path



Example: The 8-puzzle



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

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

http://aima.eecs.berkeley.edu/slides-ppt/

8-Puzzle

- **states**
  - location of tiles (including blank tile)
  - properties of tiles (numbers, letters, image parts, ...)
- **initial state**
  - any legitimate configuration
- **successor function (operators)**
  - move tile
  - alternatively: move blank
- **goal test**
  - specific legitimate configuration of tiles
    - particular arrangement of tiles
    - may reflect ordering relations, words, image
- **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 a n × n board with no two queens on the same row, column, or diagonal**
  - see also Wikipedia Eight Queens Puzzle

8-Queens Incremental Approach

- **start with an empty board**
- **add queens one by one**
  - no violation of constraints
    - different row, column, diagonal from sitting 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*1014 possible sequences; can be reduced to 2,057**

8-Queens Complete-State Approach

- **start with a full board**
  - all n queens placed on the board
  - conflicts are to be expected
- **try to find a better configuration**
  - reduced number of conflicts
- **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 ==> 648 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 solution 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 us
- **goal test**
  - only digits in the puzzle
  - calculation is correct
- **path cost**
  - all solutions are equally valid

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

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

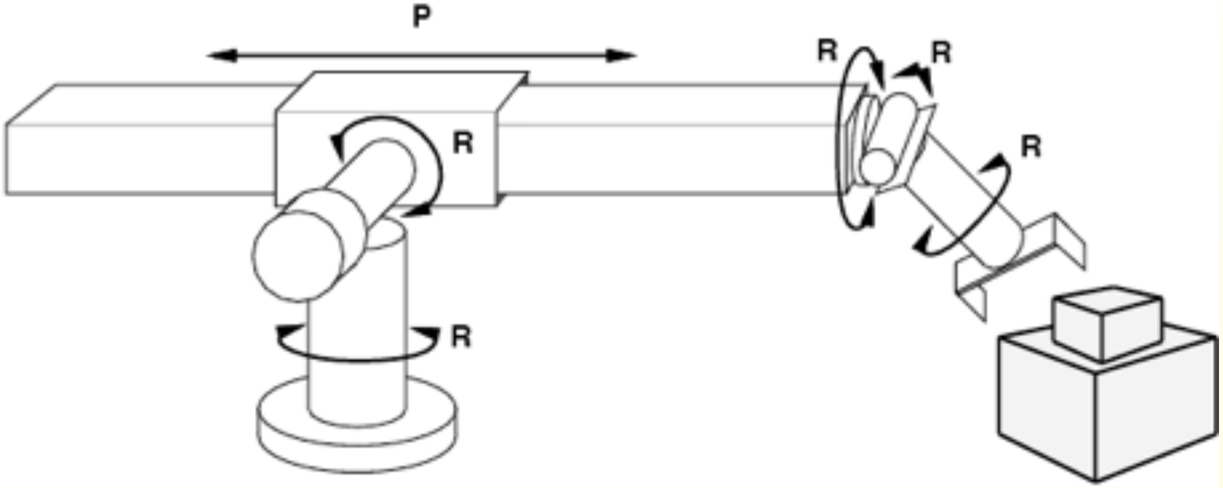
Robot Navigation

- **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

http://aima.eecs.berkeley.edu/slides-ppt/

Search Methods

Terminology
Search and Graphs

Searching for Solutions

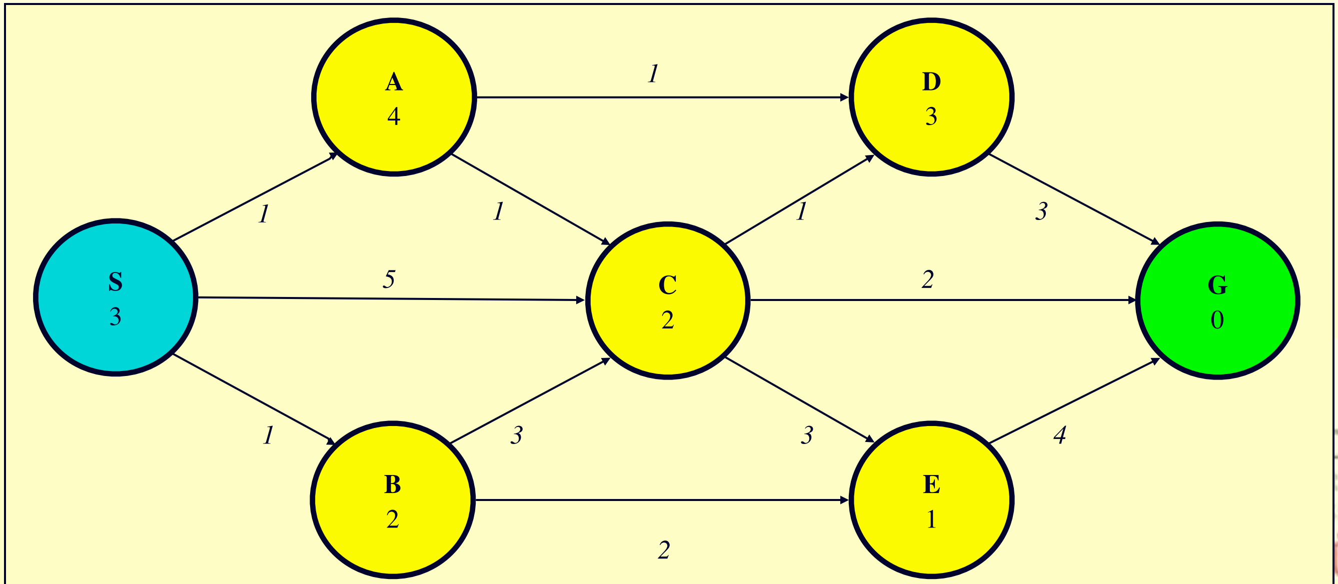
- **traversal of the search space**
  - from the initial state to a goal state
  - legal sequence of actions as defined by successor function (operators)
- **general procedure**
  - check for goal state
  - expand the current state
    - determine the set of reachable states
    - return “failure” if 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

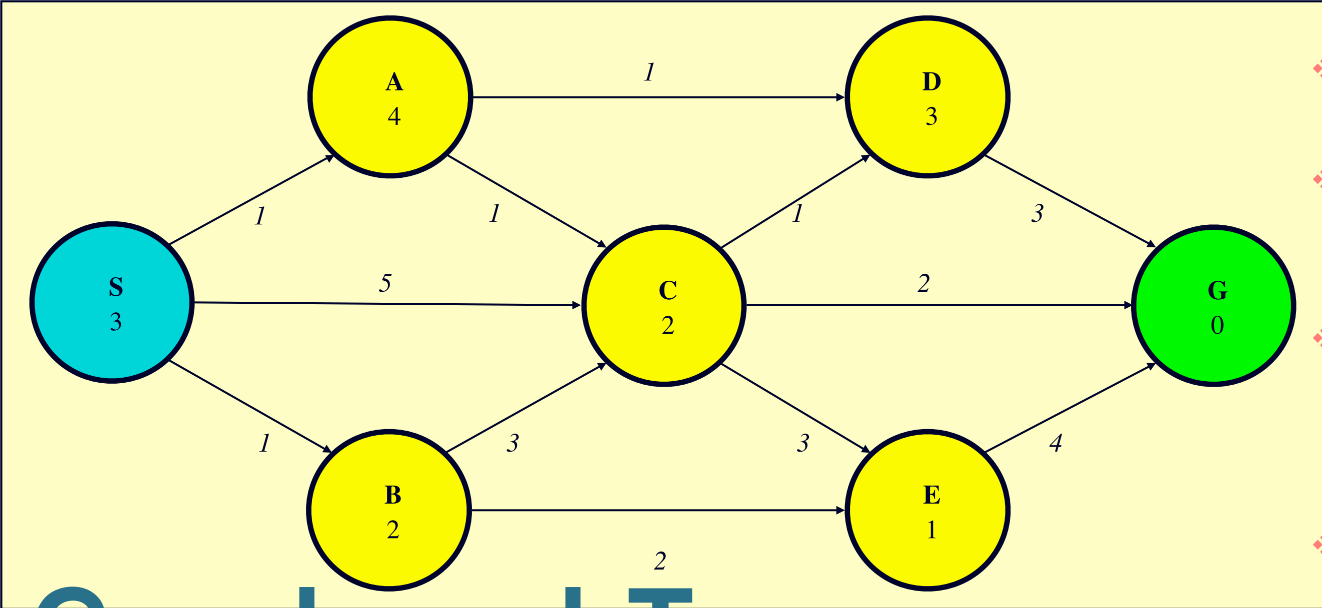
Search Terminology

- **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)

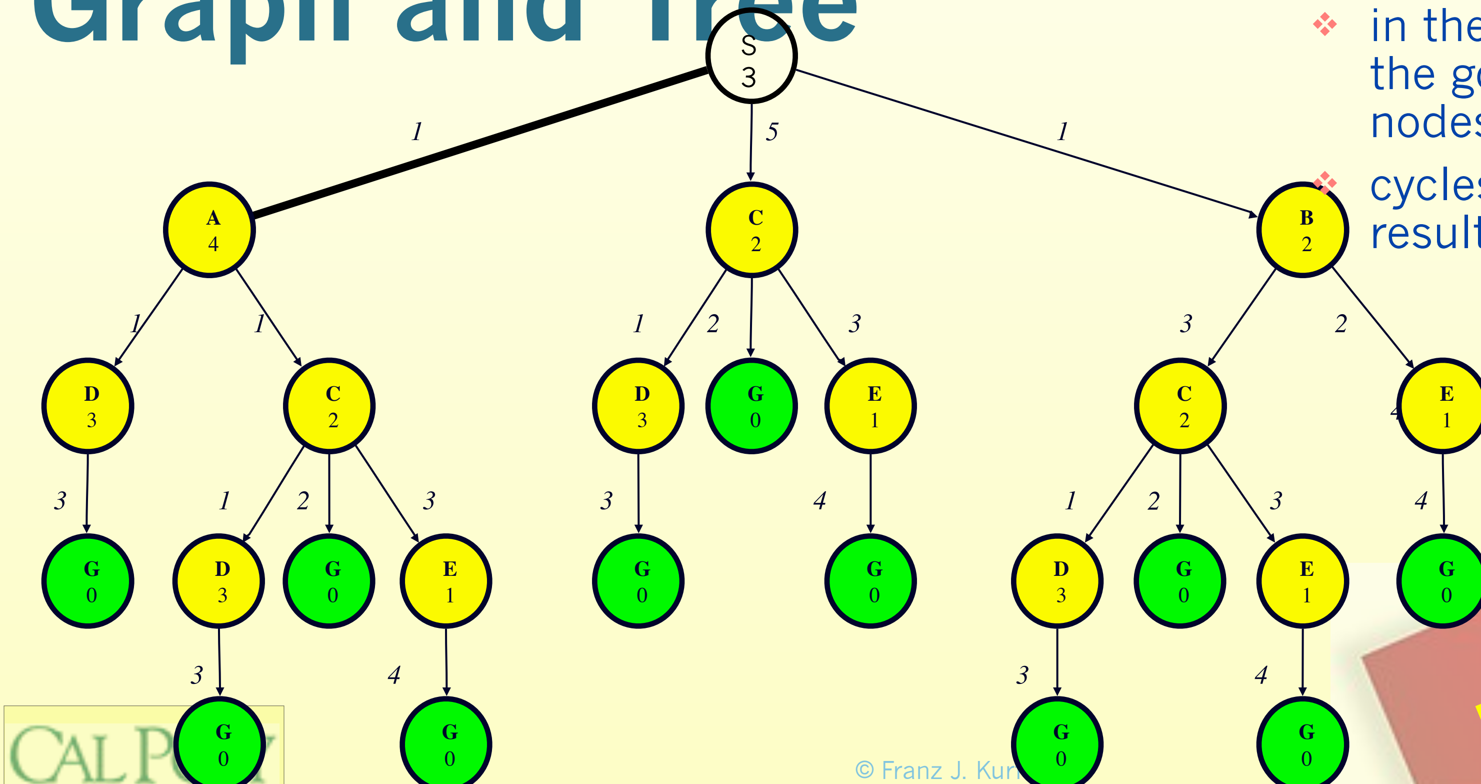
Example: Graph Search

- **describes the search (state) space**
  - each node represents one state in the search space
    - e.g. a city to be visited in a routing or touring problem
  - 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

General Tree Search Algorithm

- **generate the first 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

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)
```

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**

Search Strategies

Uninformed
Informed
Local
Others

Selection of a Search Strategy

- **selection of an appropriate search strategy for a given problem**
  - uninformed search (blind search)
    - number of steps, path cost unknown
    - agent knows it is at a goal only after it reaches a goal
      - goals are not "visible" from a distance
  - informed search (heuristic search)
    - agent has background information about the problem
      - map, costs of actions
      - hints about the location of the goal
      - evaluating hints can be costly as well

Search 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
- **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

plain breadth-first
uniform cost

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

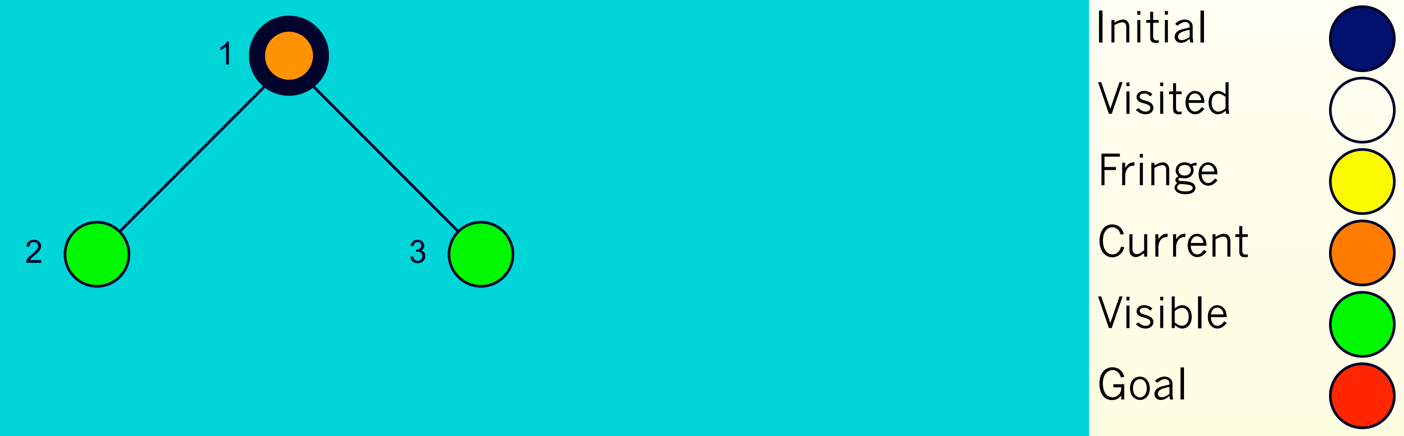
```
function BREADTH-FIRST-SEARCH(problem) returns solution

        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)

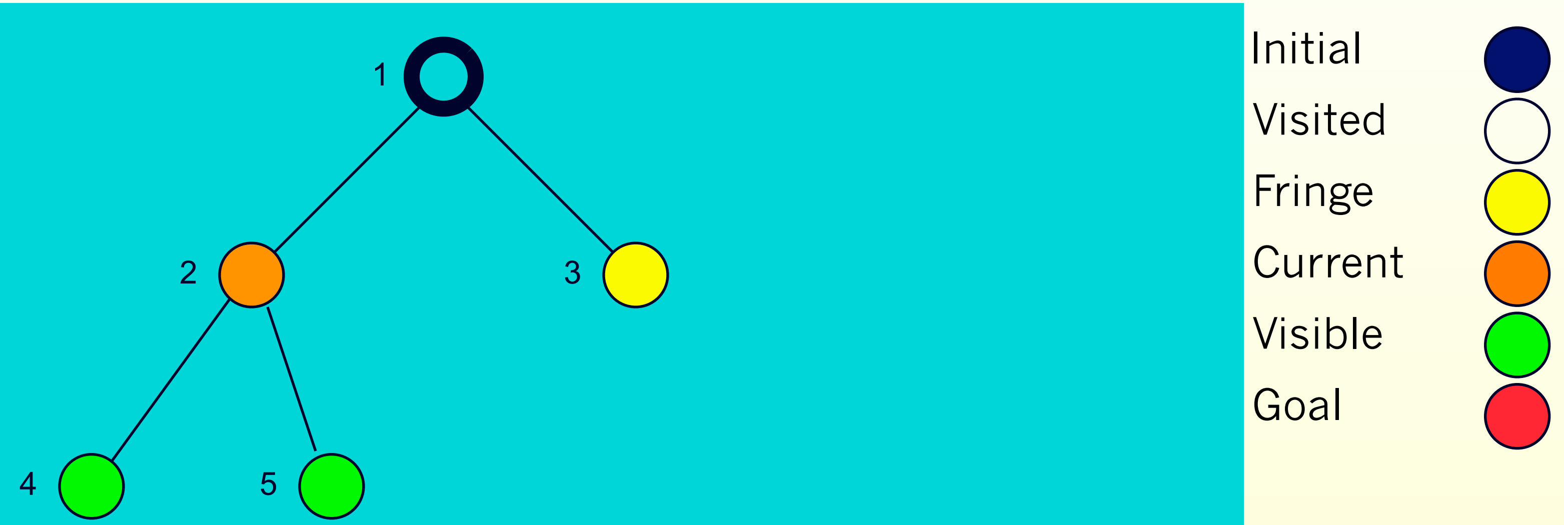
b	branching factor
d	depth of the tree

Breadth-First Snapshot 1



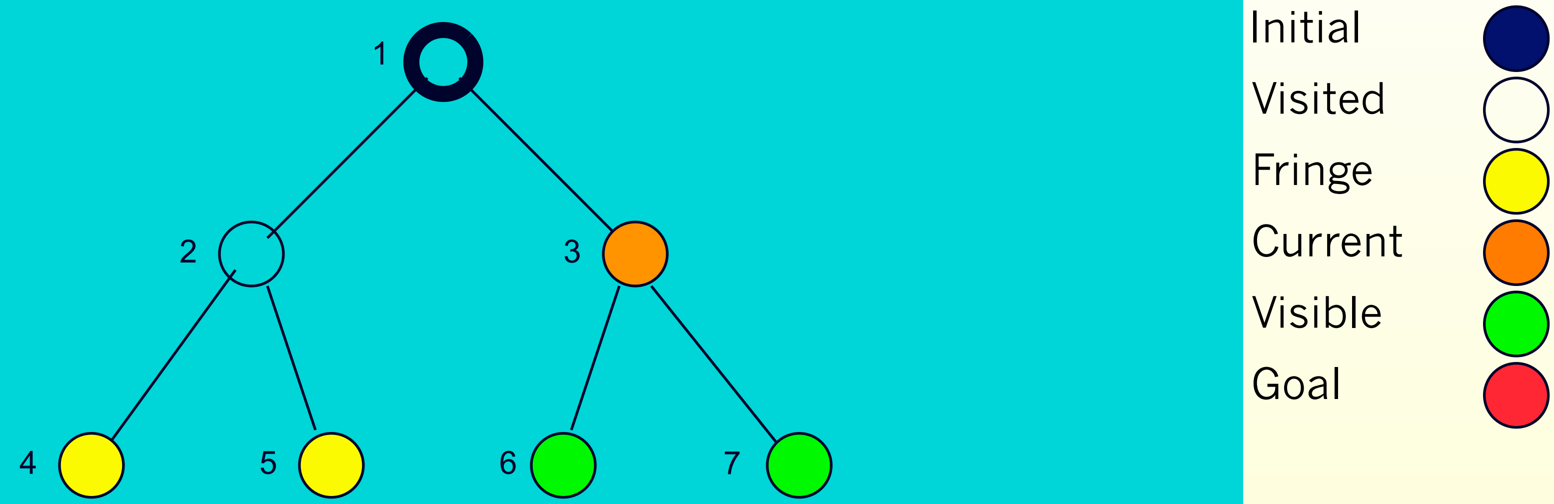
Fringe: [] + [2,3]

Breadth-First Snapshot 2



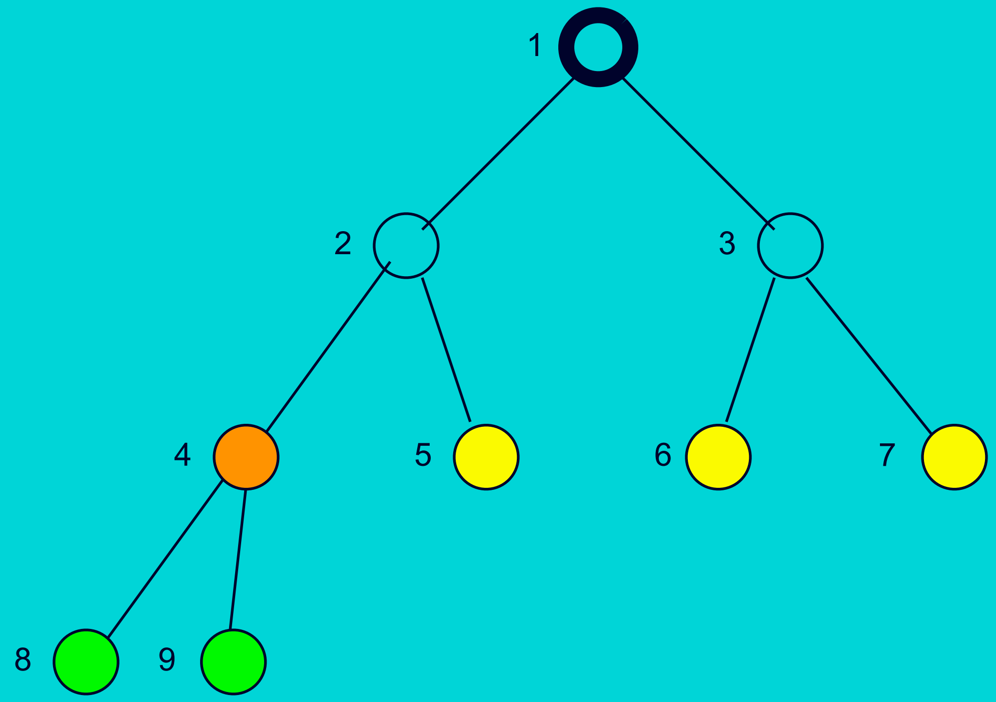
Fringe: [3] + [4,5]

Breadth-First Snapshot 3



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

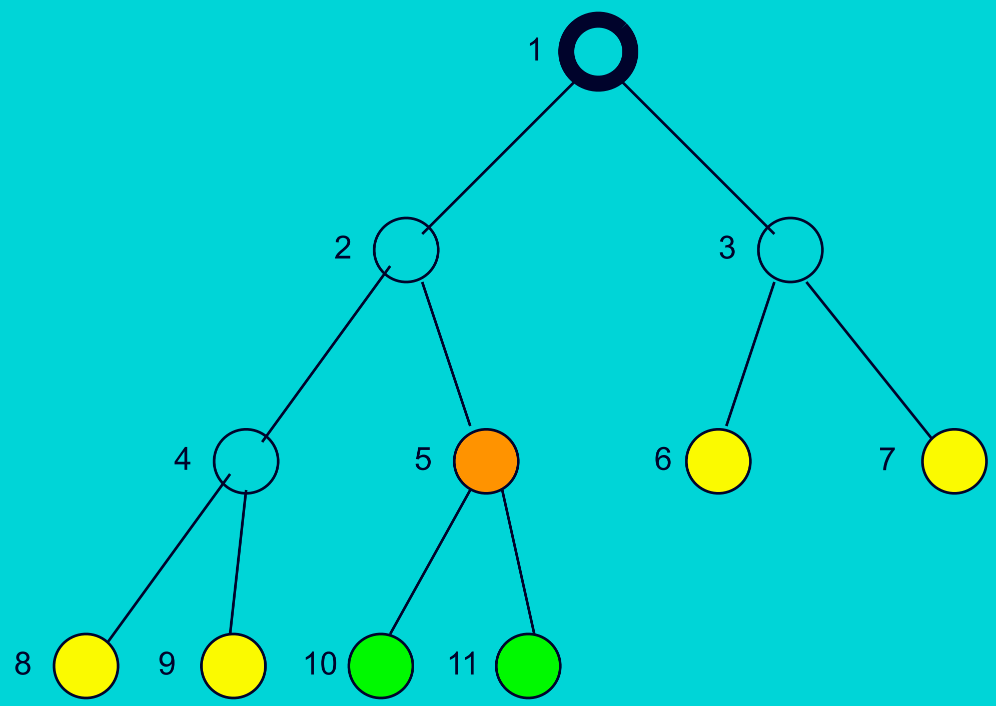
Breadth-First Snapshot 4



Initial
Visited
Fringe
Current
Visible
Goal

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

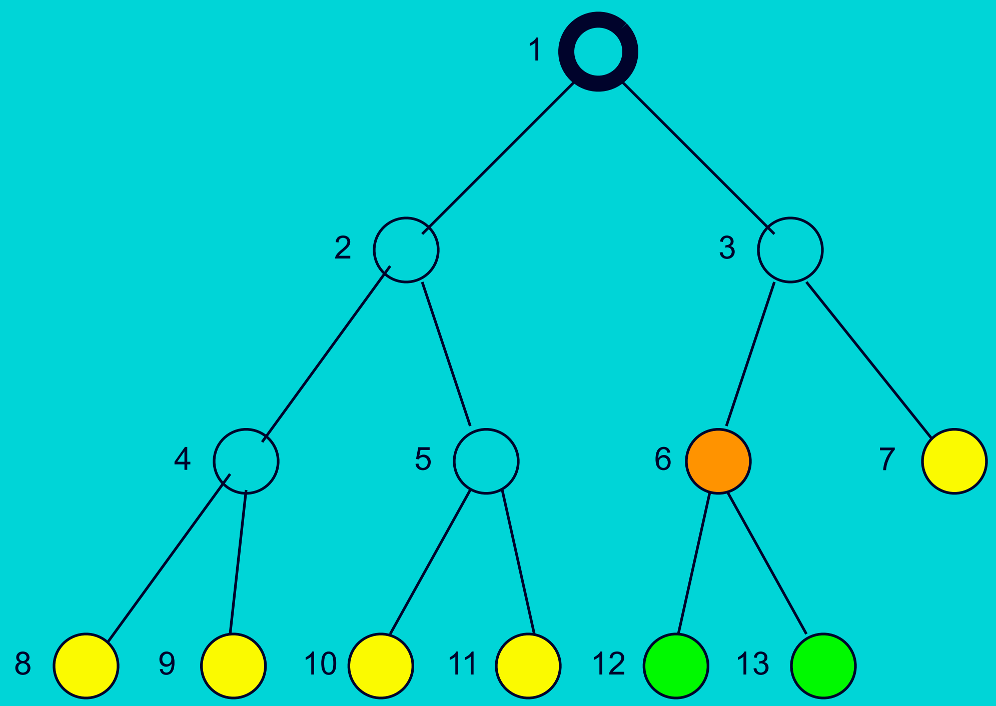
Breadth-First Snapshot 5



Initial
Visited
Fringe
Current
Visible
Goal

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

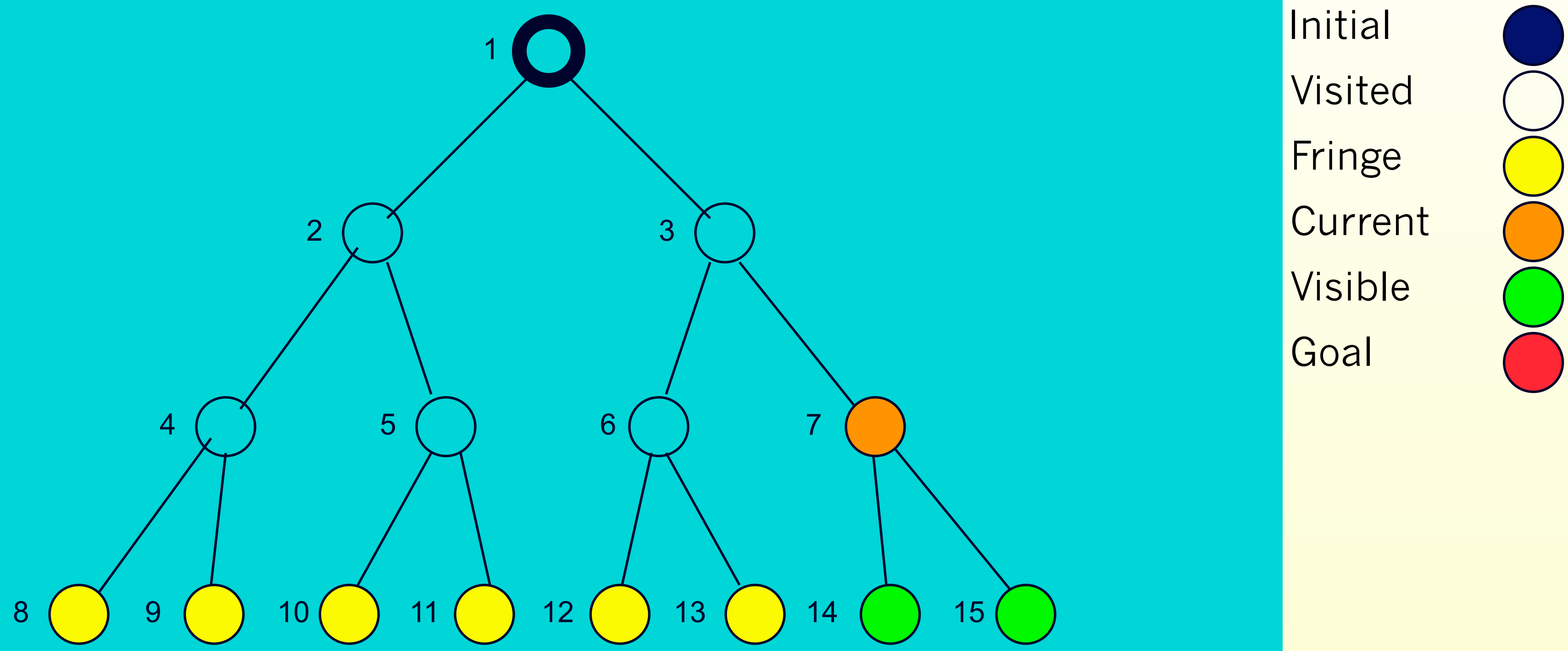
Breadth-First Snapshot 6



Initial
Visited
Fringe
Current
Visible
Goal

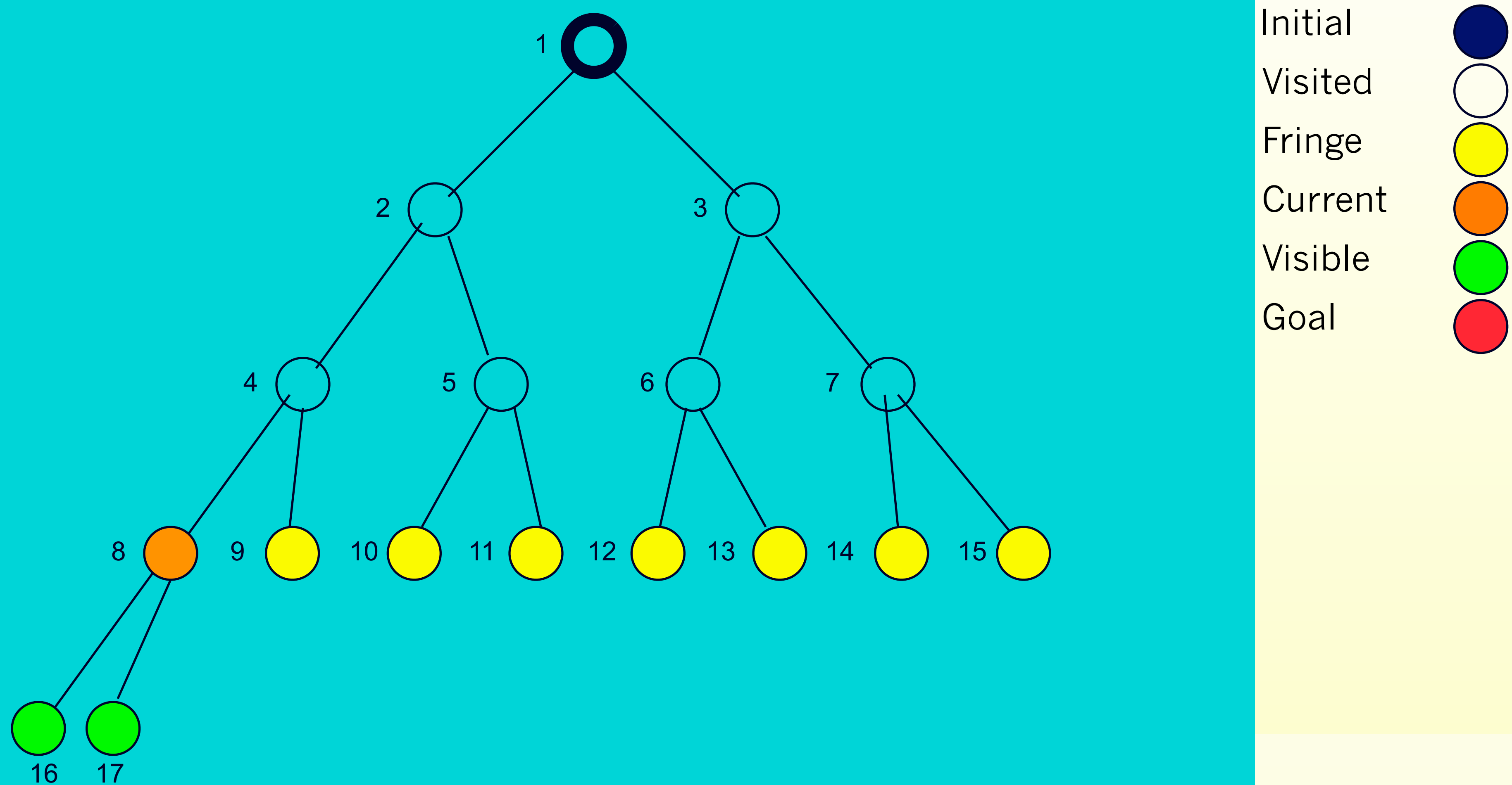
Fringe: [7,8,9,10,11] + [12,13]

Breadth-First Snapshot 7



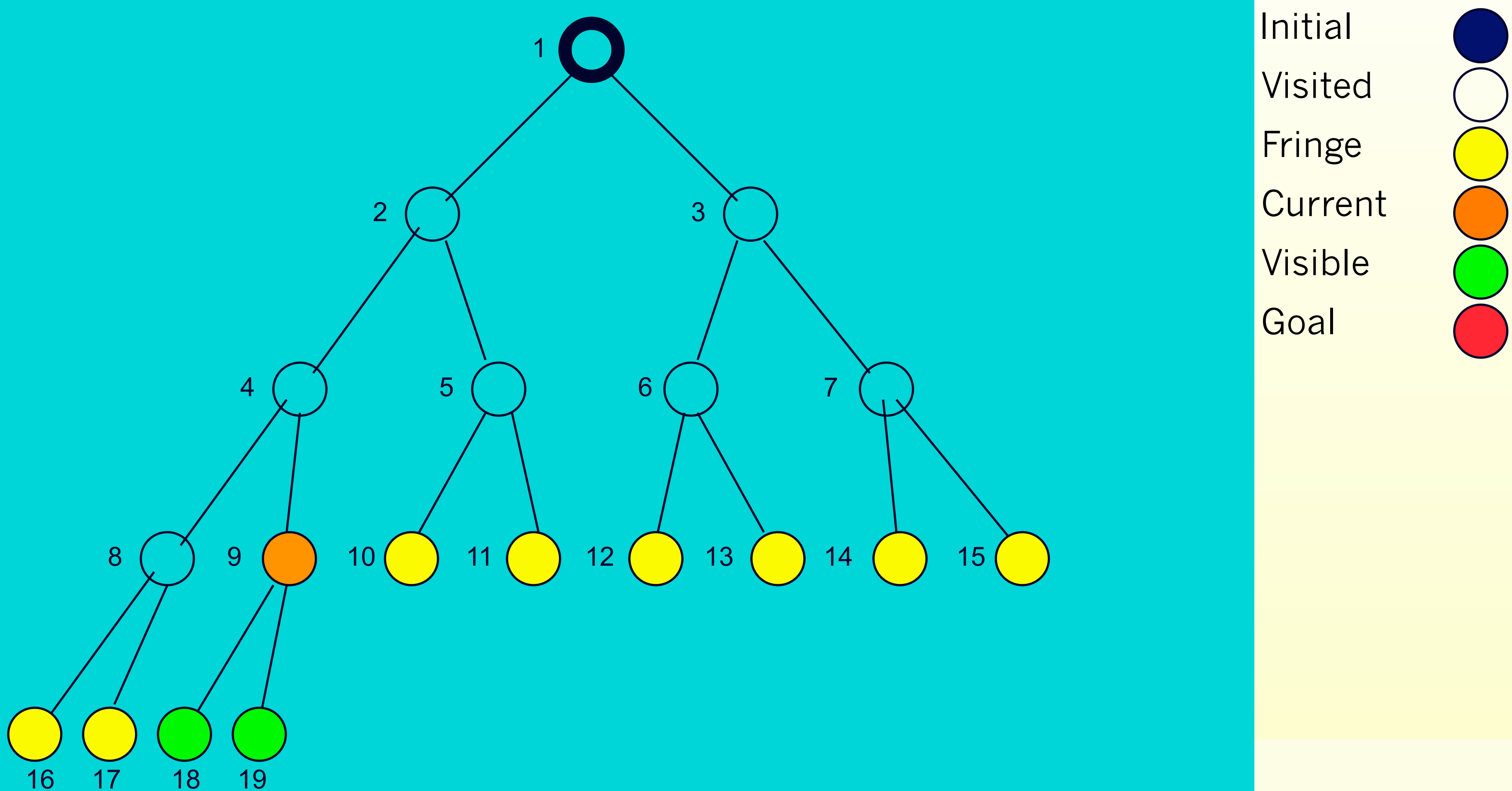
Fringe: [8,9.10,11,12,13] + [14,15]

Breadth-First Snapshot 8



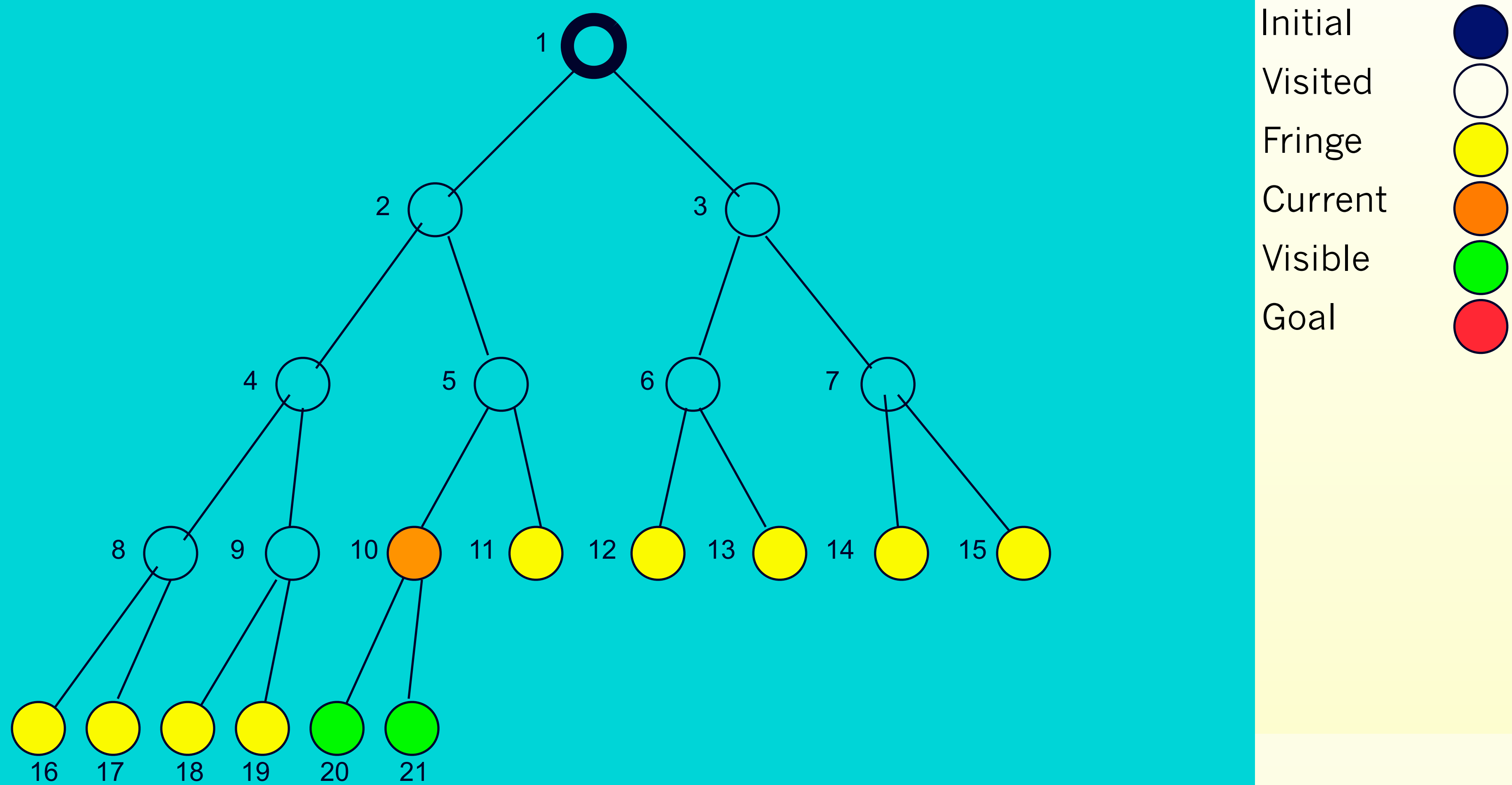
Fringe: [9,10,11,12,13,14,15] + [16,17]

Breadth-First Snapshot 9



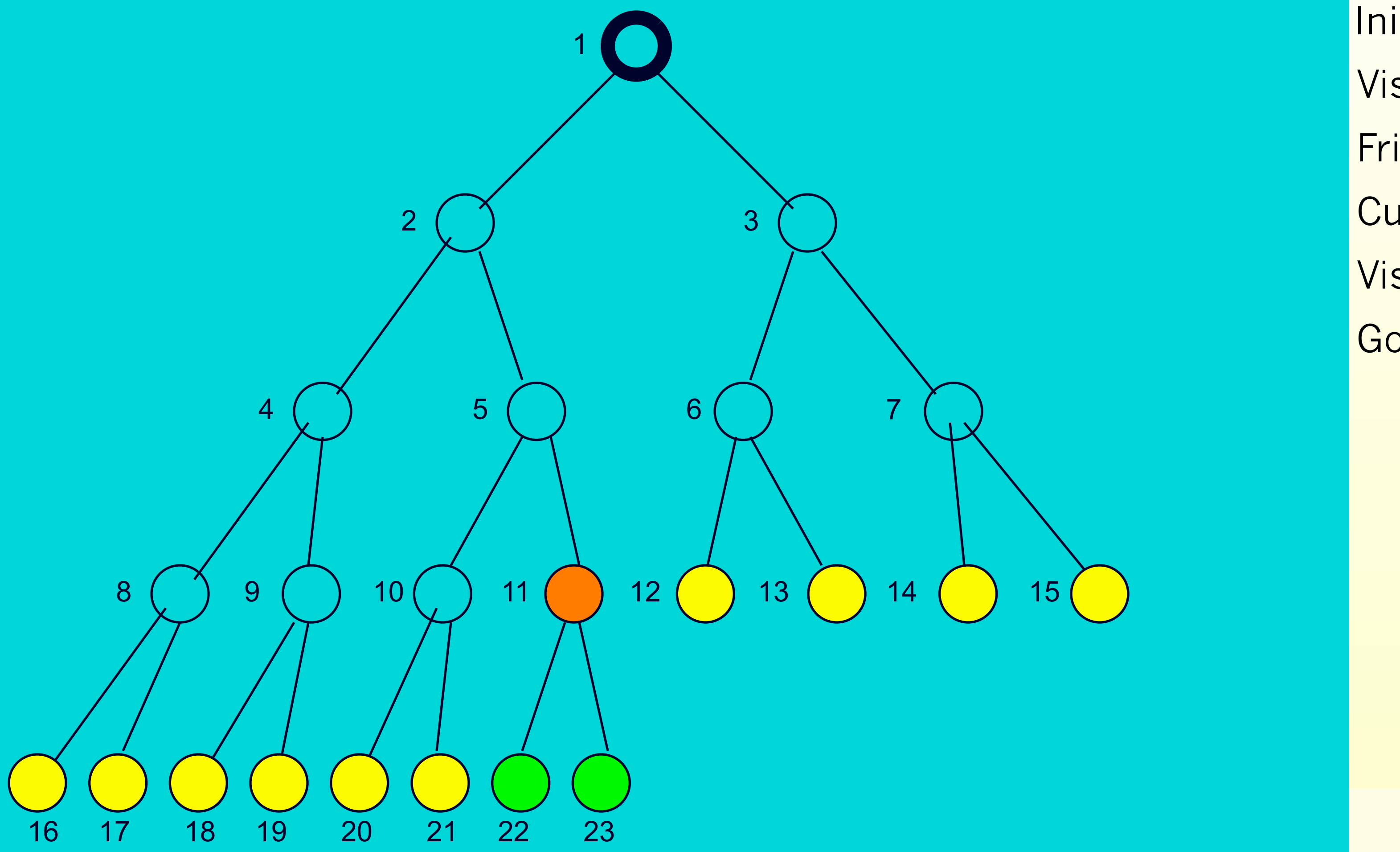
Fringe: [10,11,12,13,14,15,16,17] + [18,19]

Breadth-First Snapshot 10



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

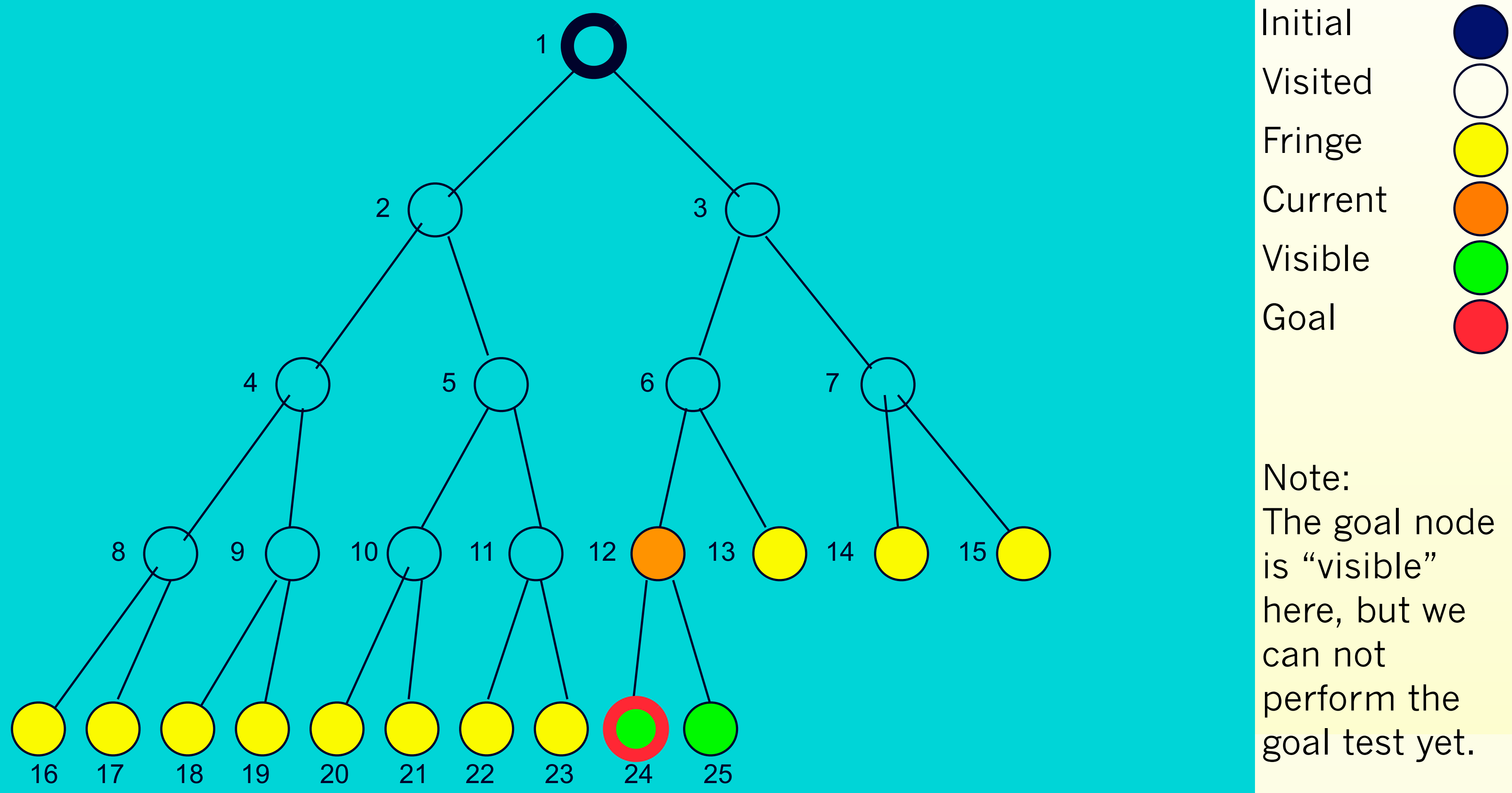
Breadth-First Snapshot 11



Initial
Visited
Fringe
Current
Visible
Goal

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

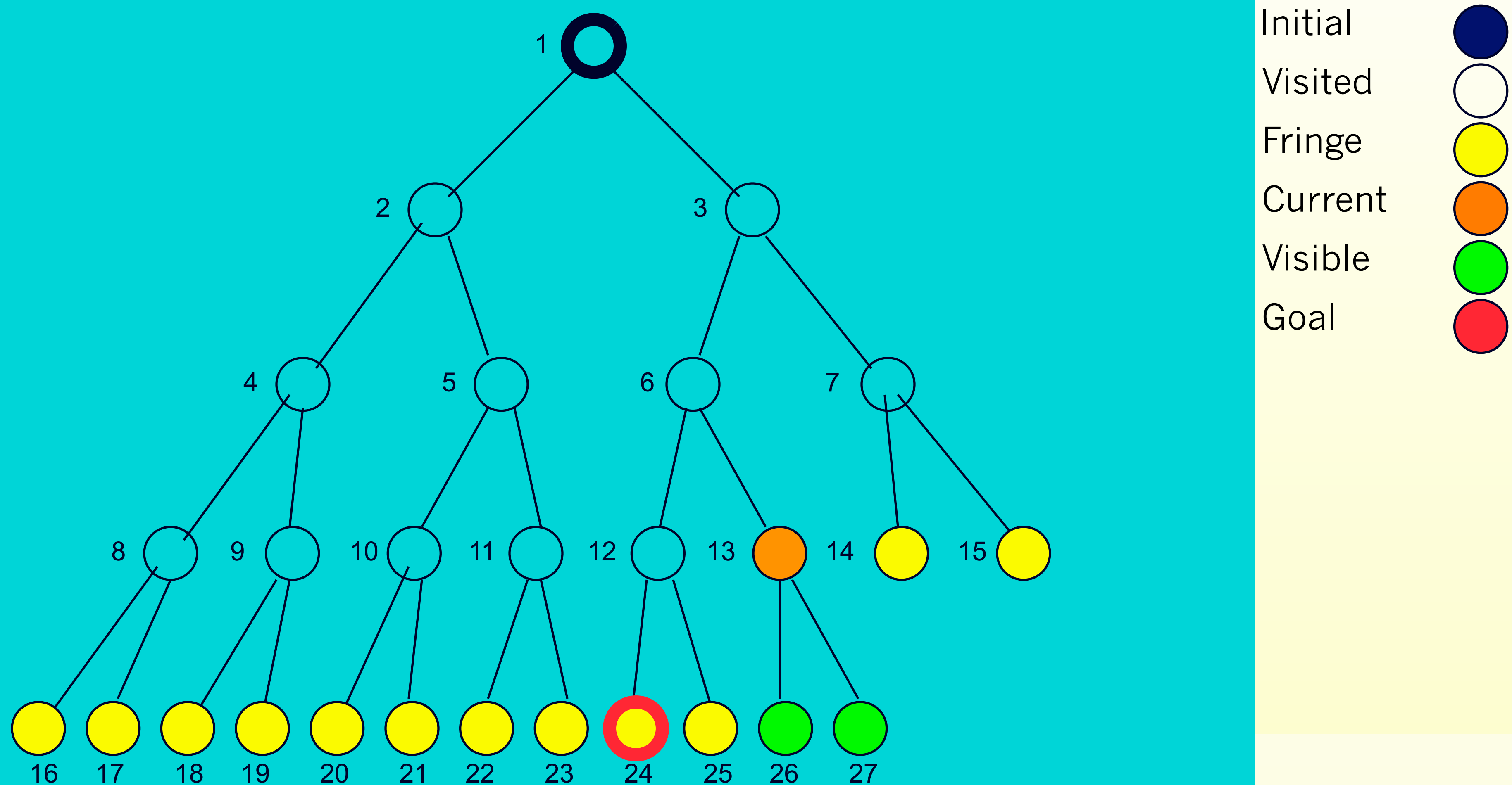
Breadth-First Snapshot 12



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

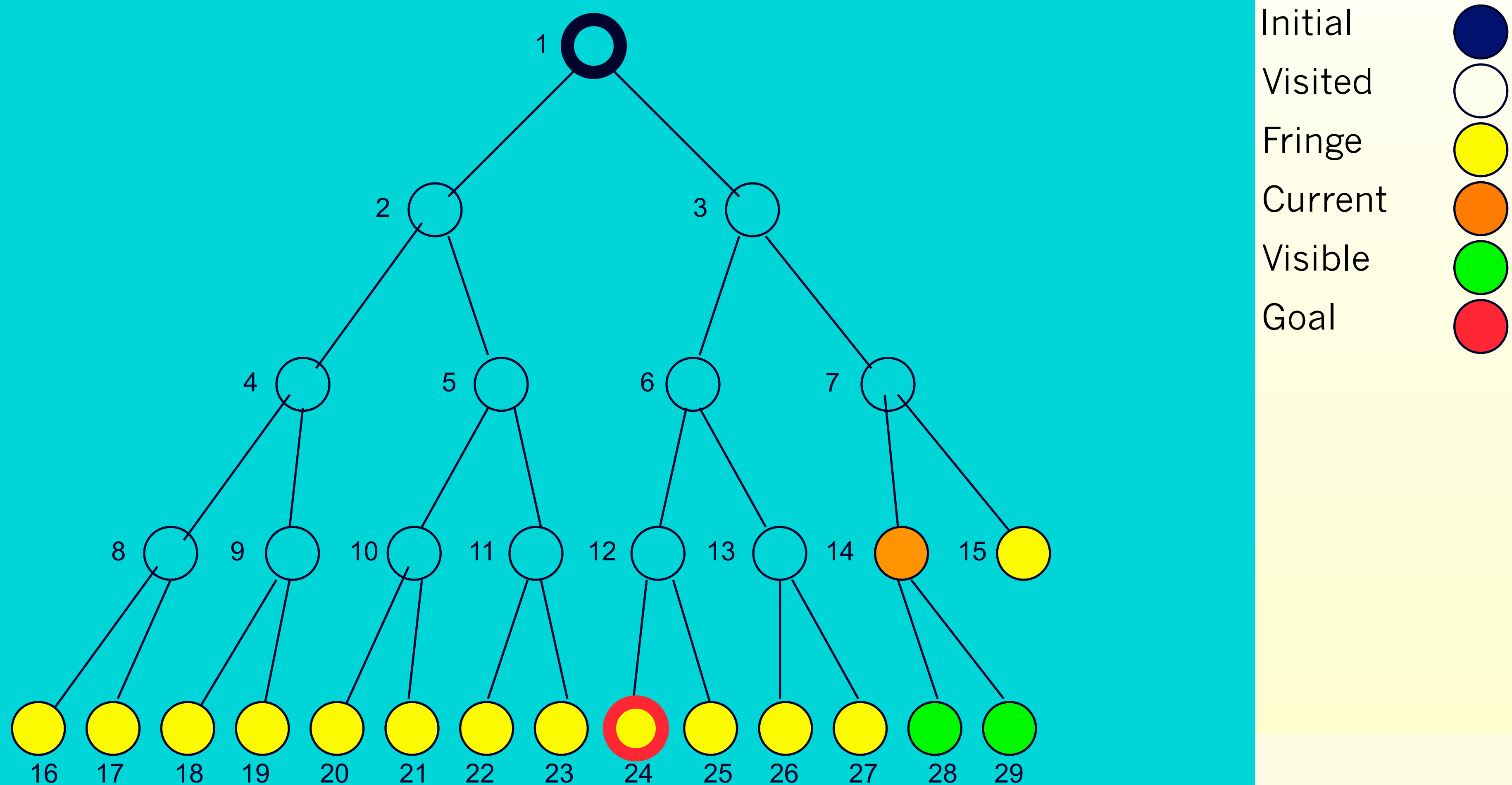
Fringe: [13,14,15,16,17,18,19,20,21] + [22,23]

Breadth-First Snapshot 13



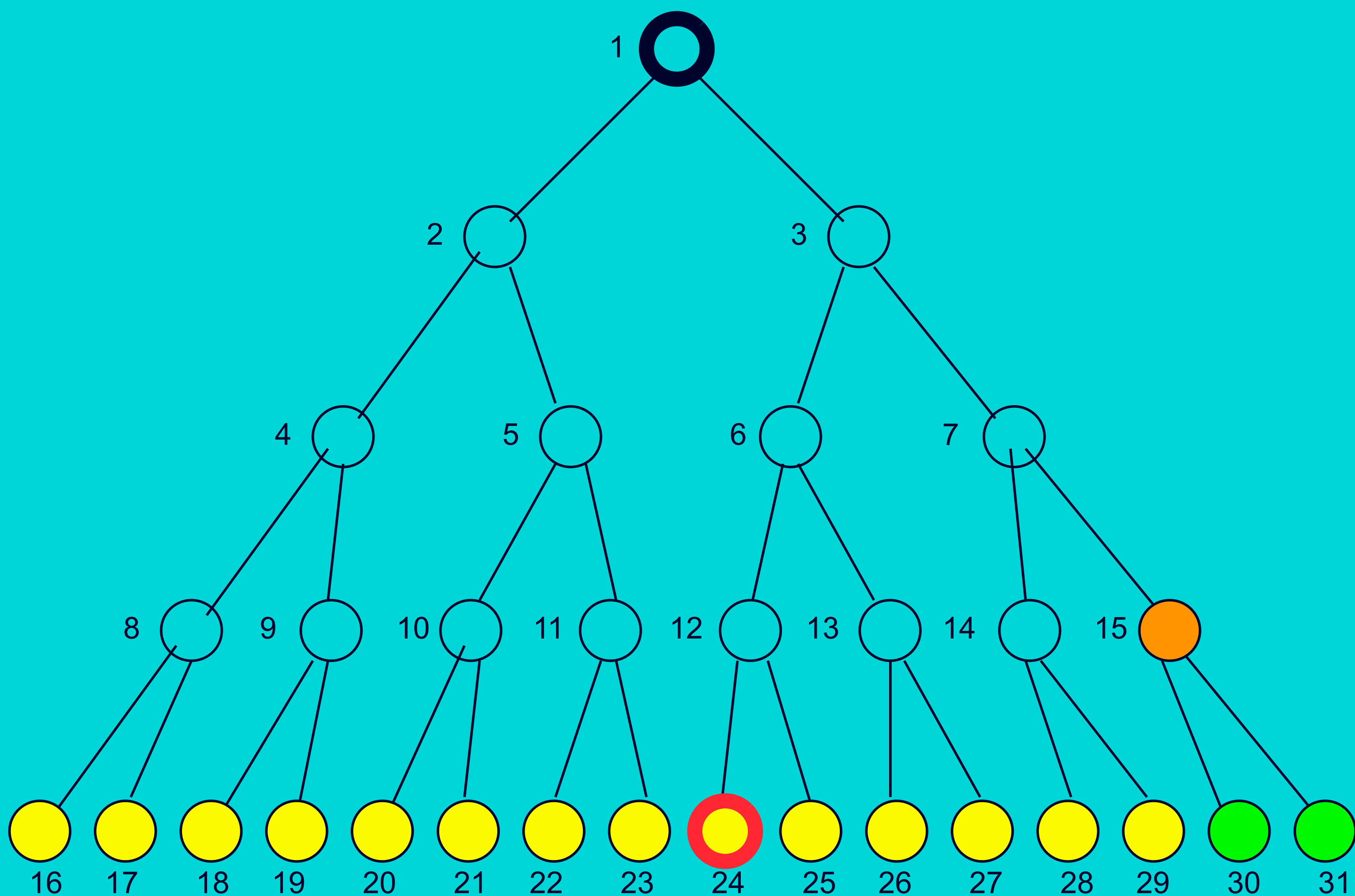
Fringe: [14,15,16,17,18,19,20,21,22,23,24,25] + [26,27]

Breadth-First Snapshot 14



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

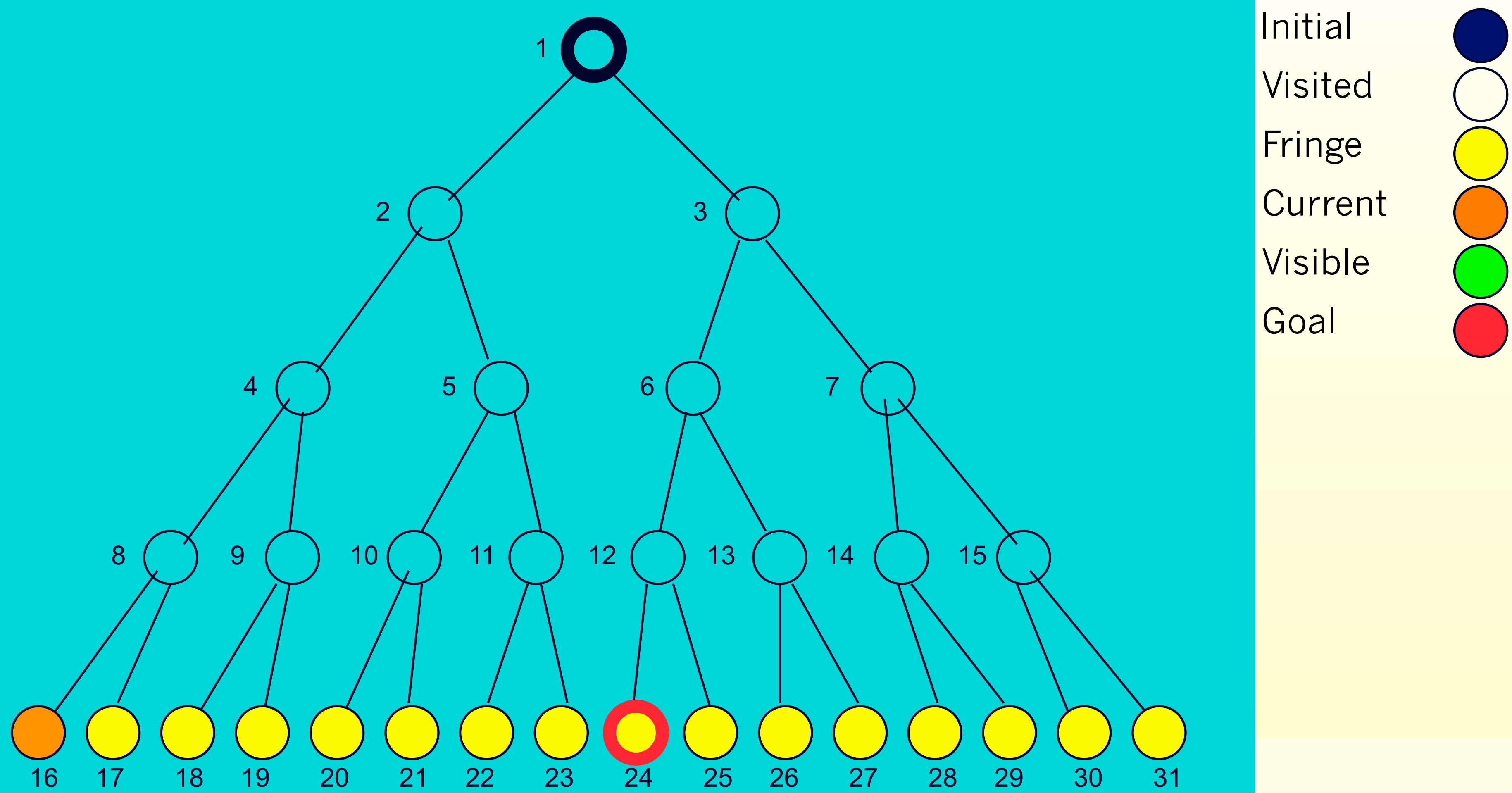
Breadth-First Snapshot 15



Initial
Visited
Fringe
Current
Visible
Goal

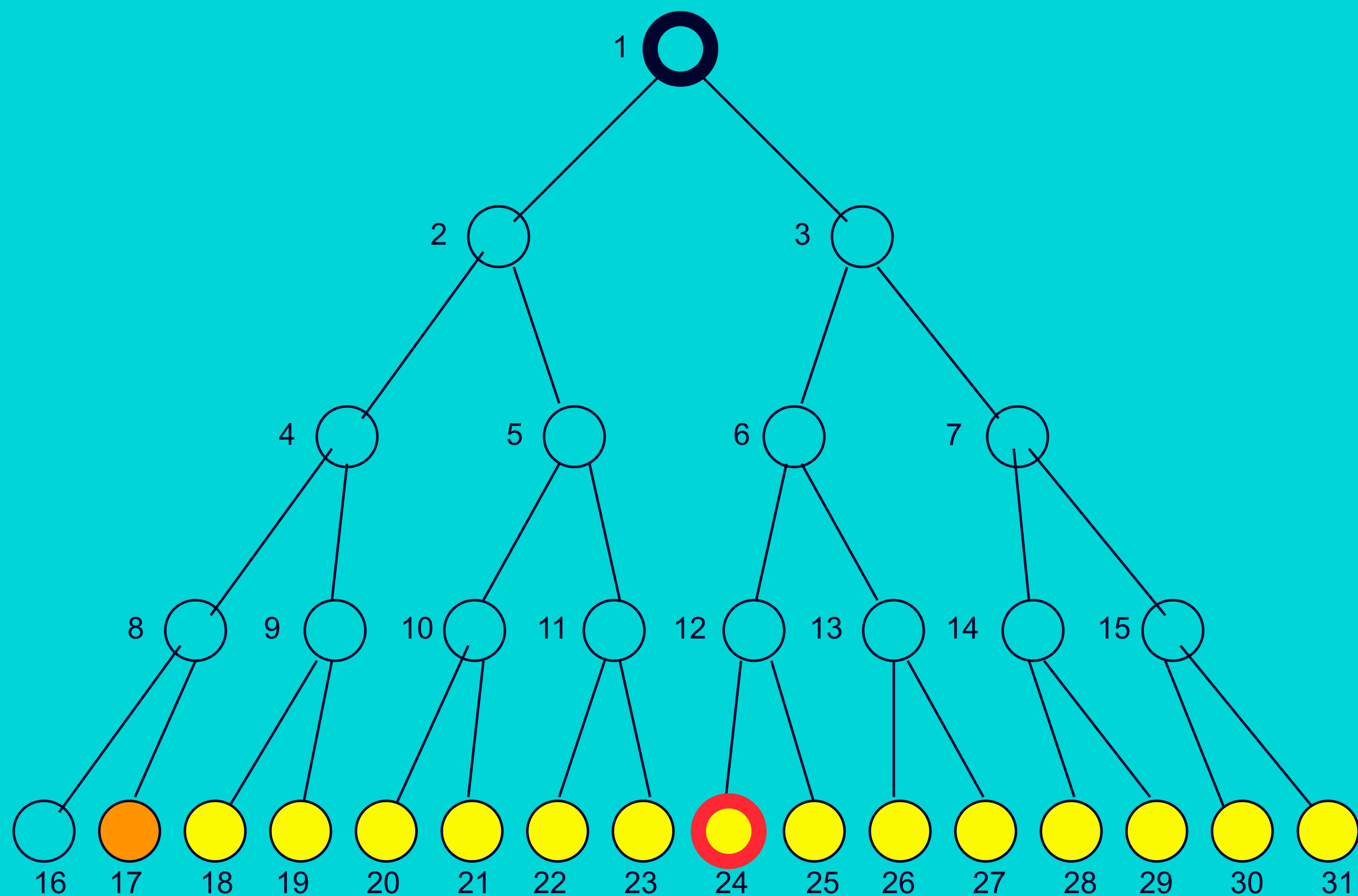
Fringe: [15,16,17,18,19,20,21,22,23,24,25,26,27,28,29] + [30,31]

Breadth-First Snapshot 16



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

Breadth-First Snapshot 17

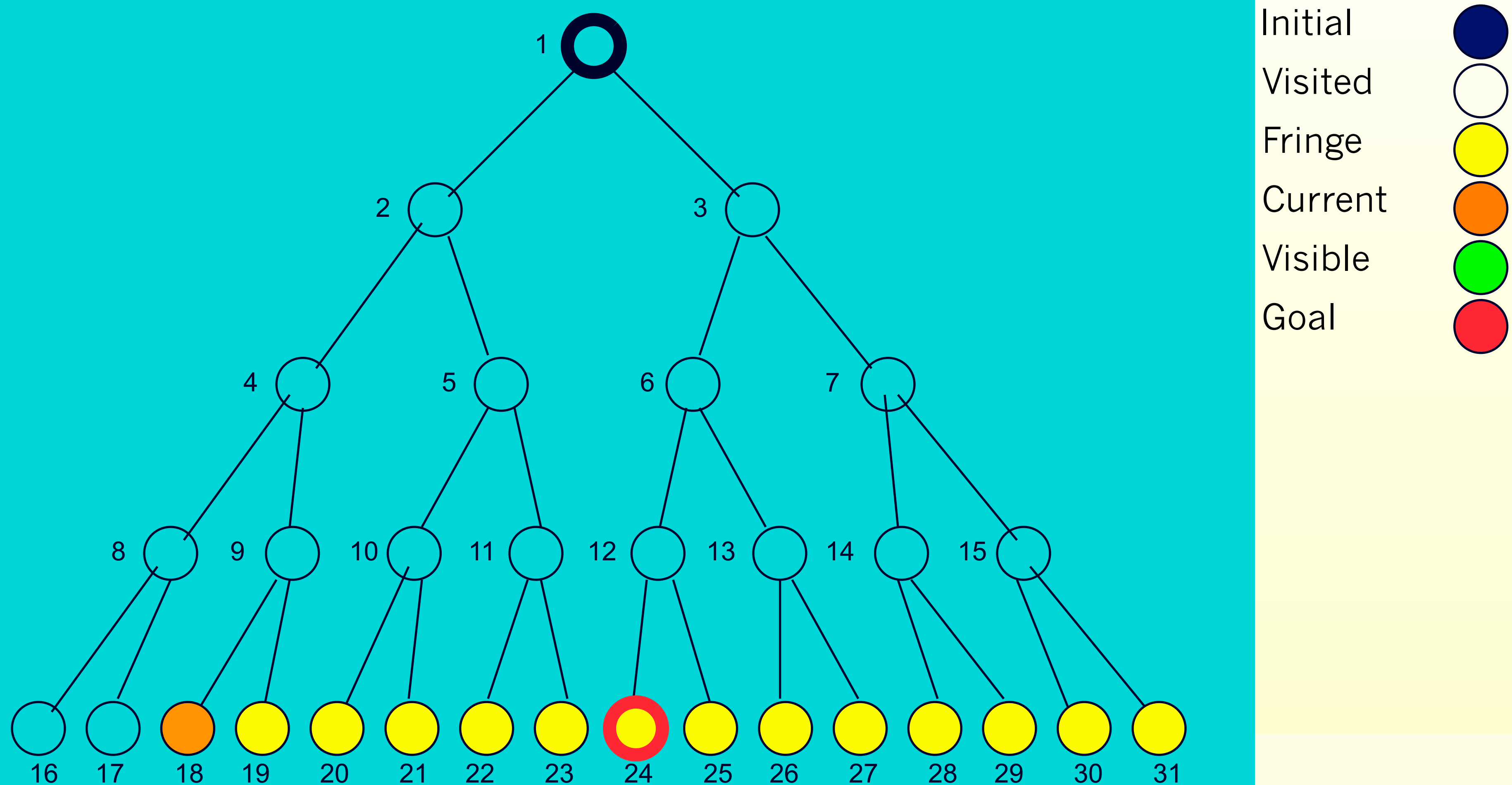


Initial
Visited
Fringe
Current
Visible
Goal

Fringe: [18,19,20,21,22,23,24,25,26,27,28,29,30,3]

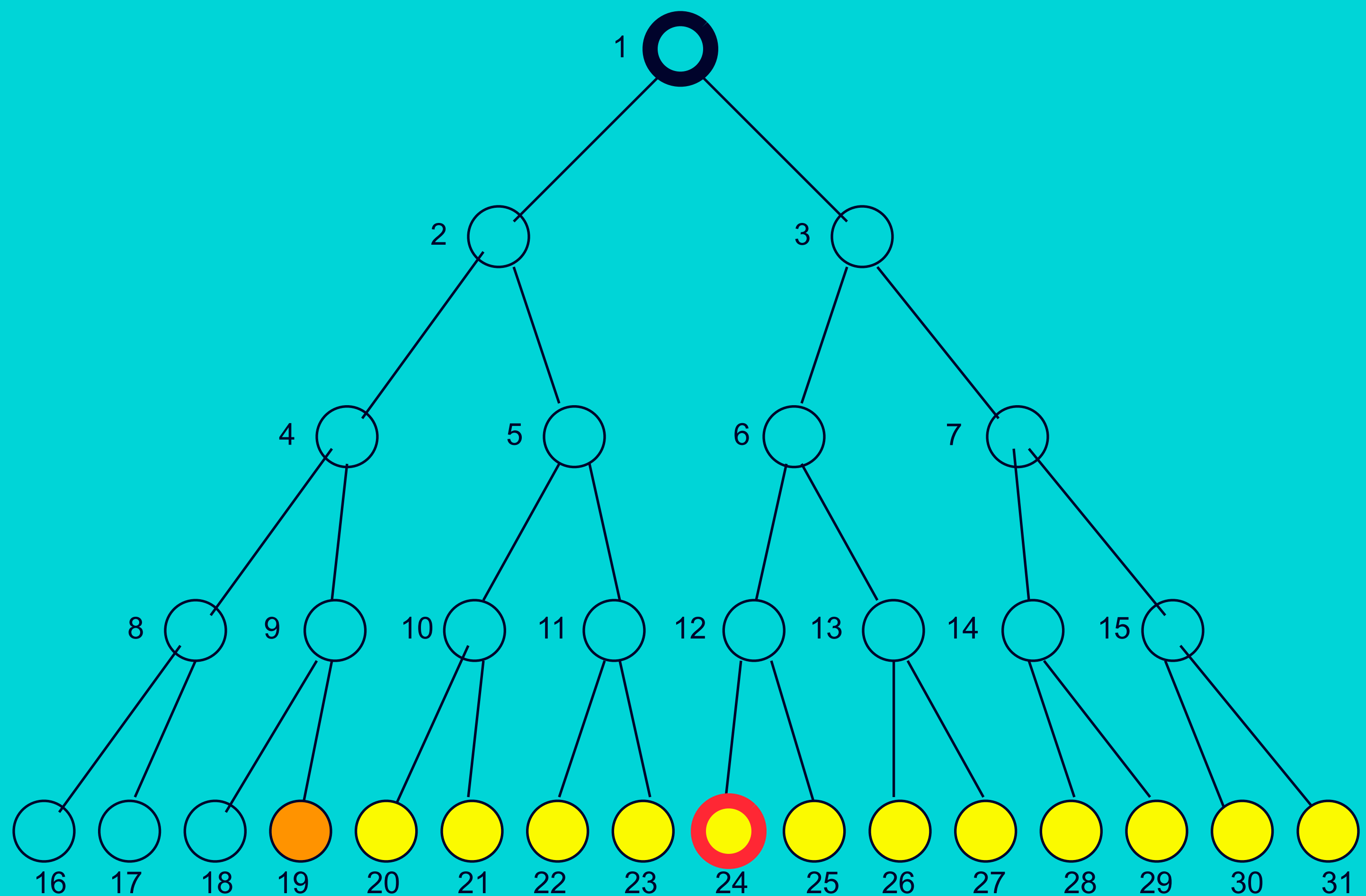
HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Breadth-First Snapshot 18



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

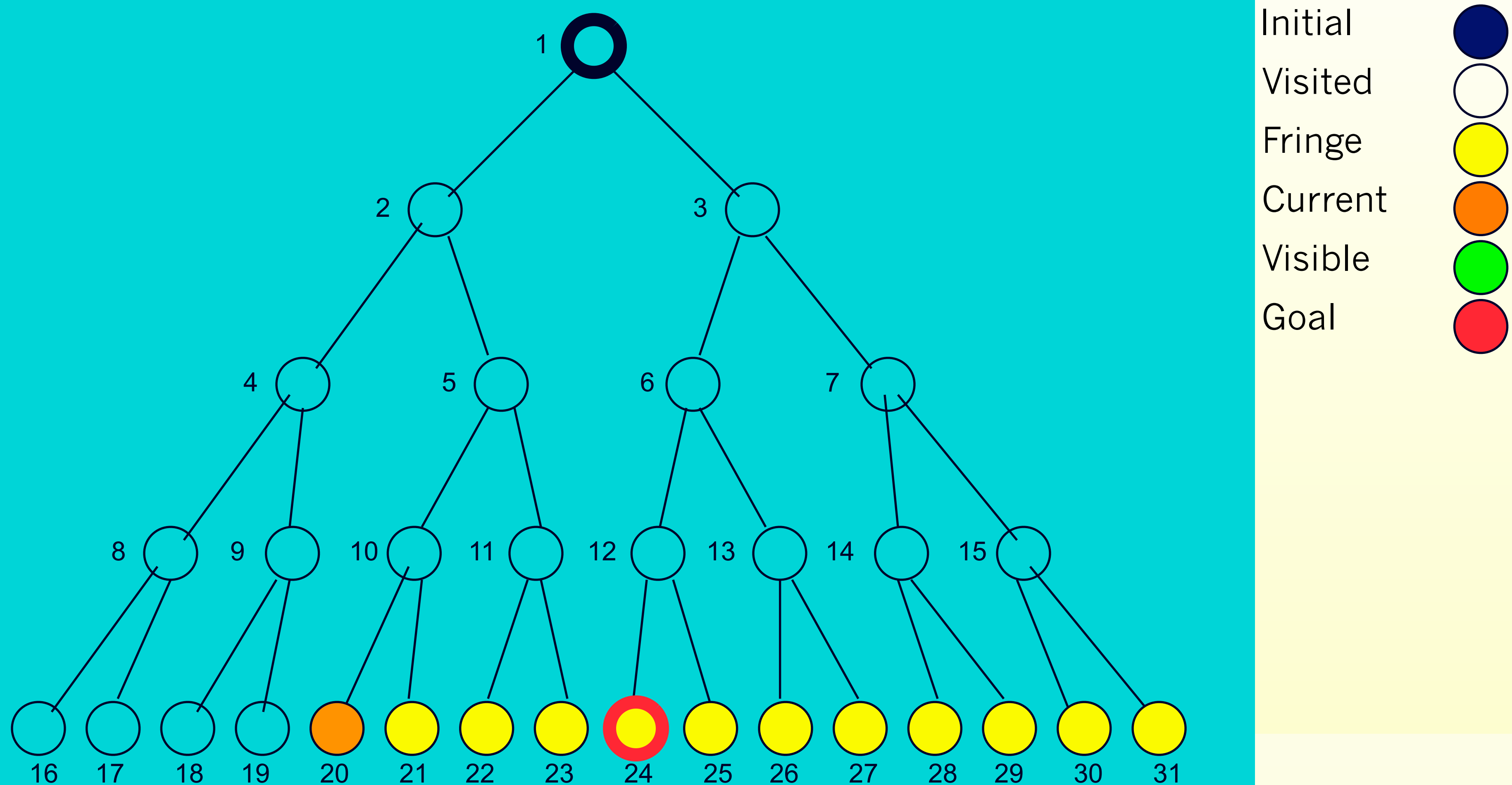
Breadth-First Snapshot 19



Initial
Visited
Fringe
Current
Visible
Goal

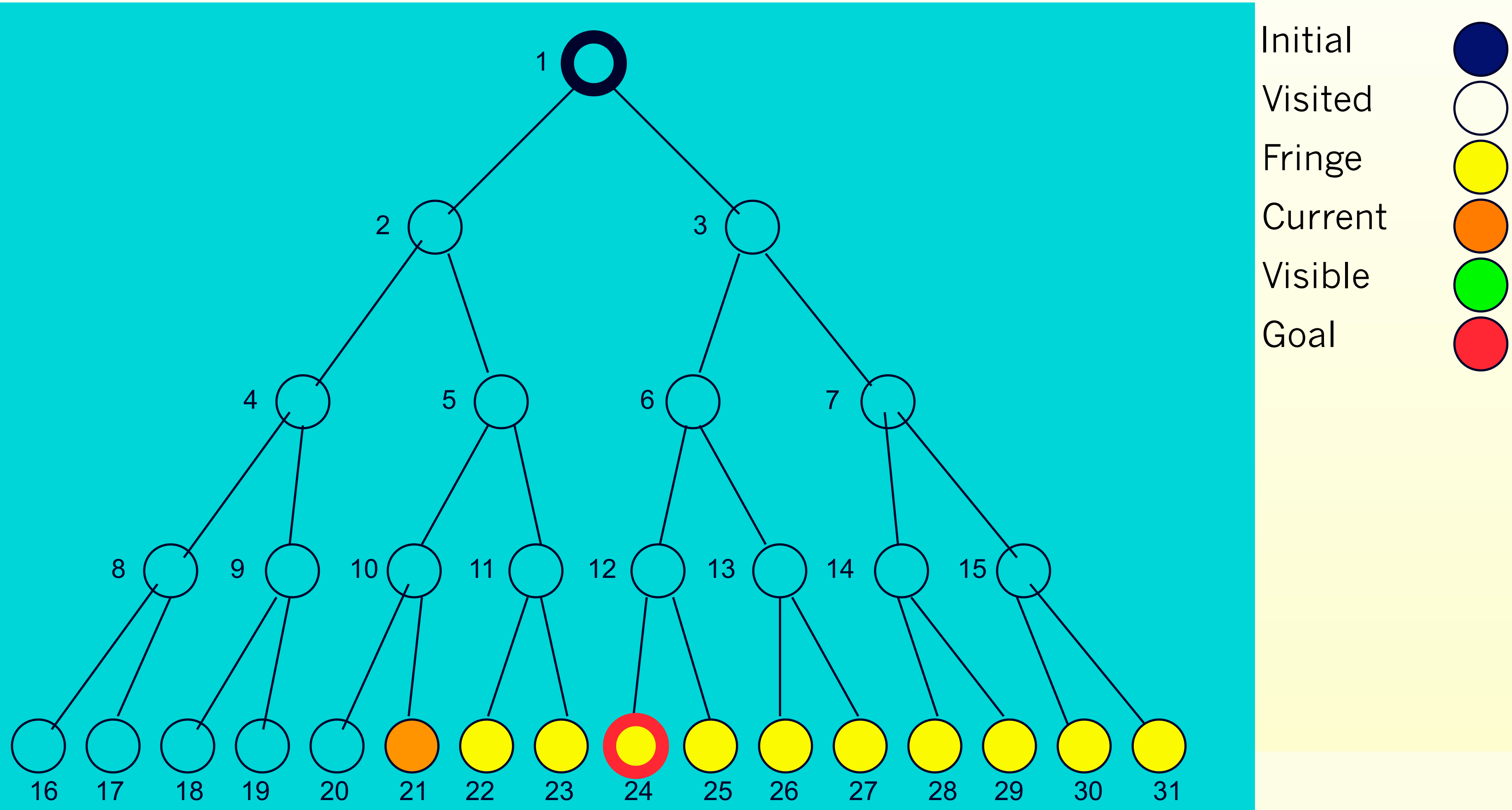
Fringe: [20,21,22,23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 20



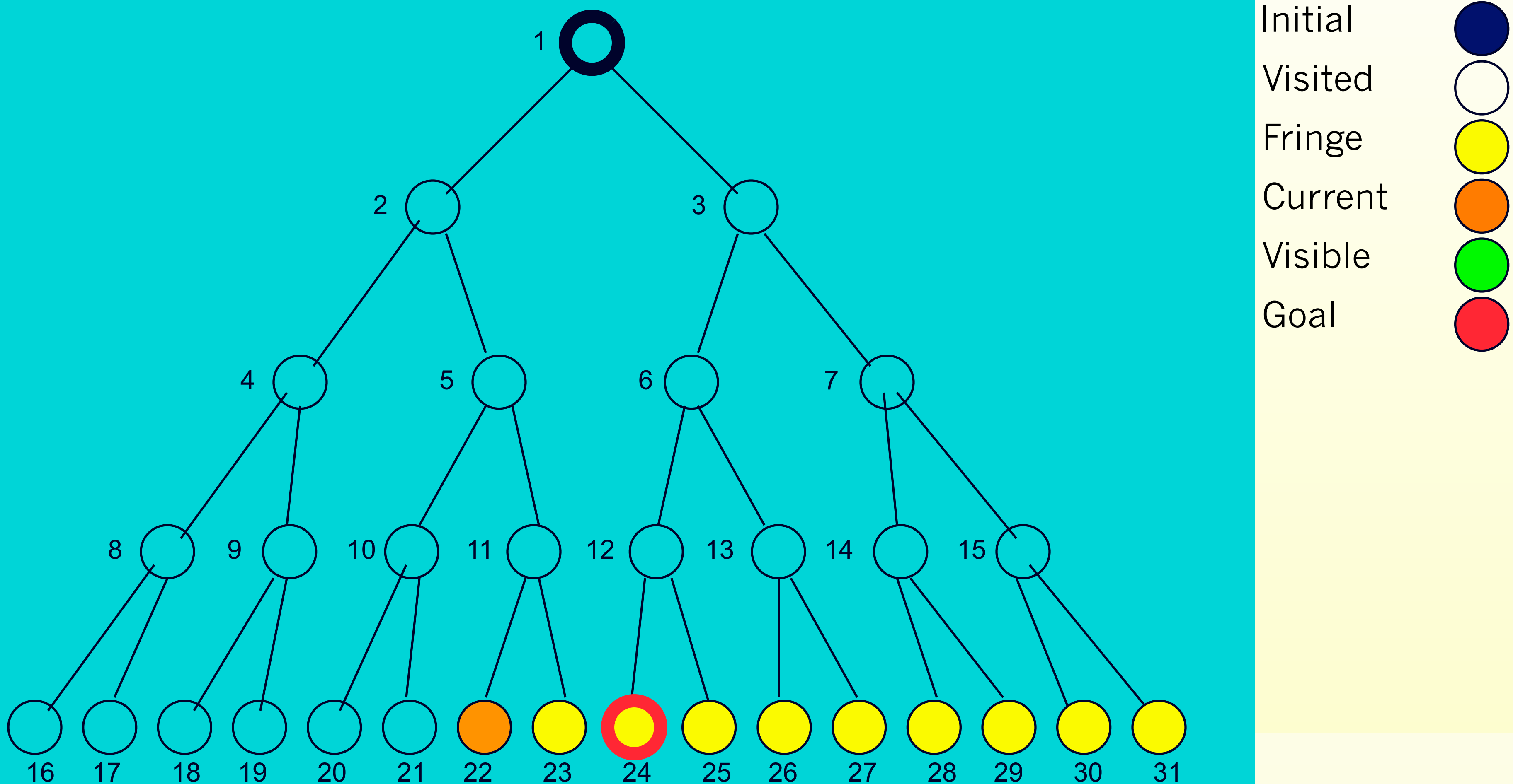
Fringe: [21,22,23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 21



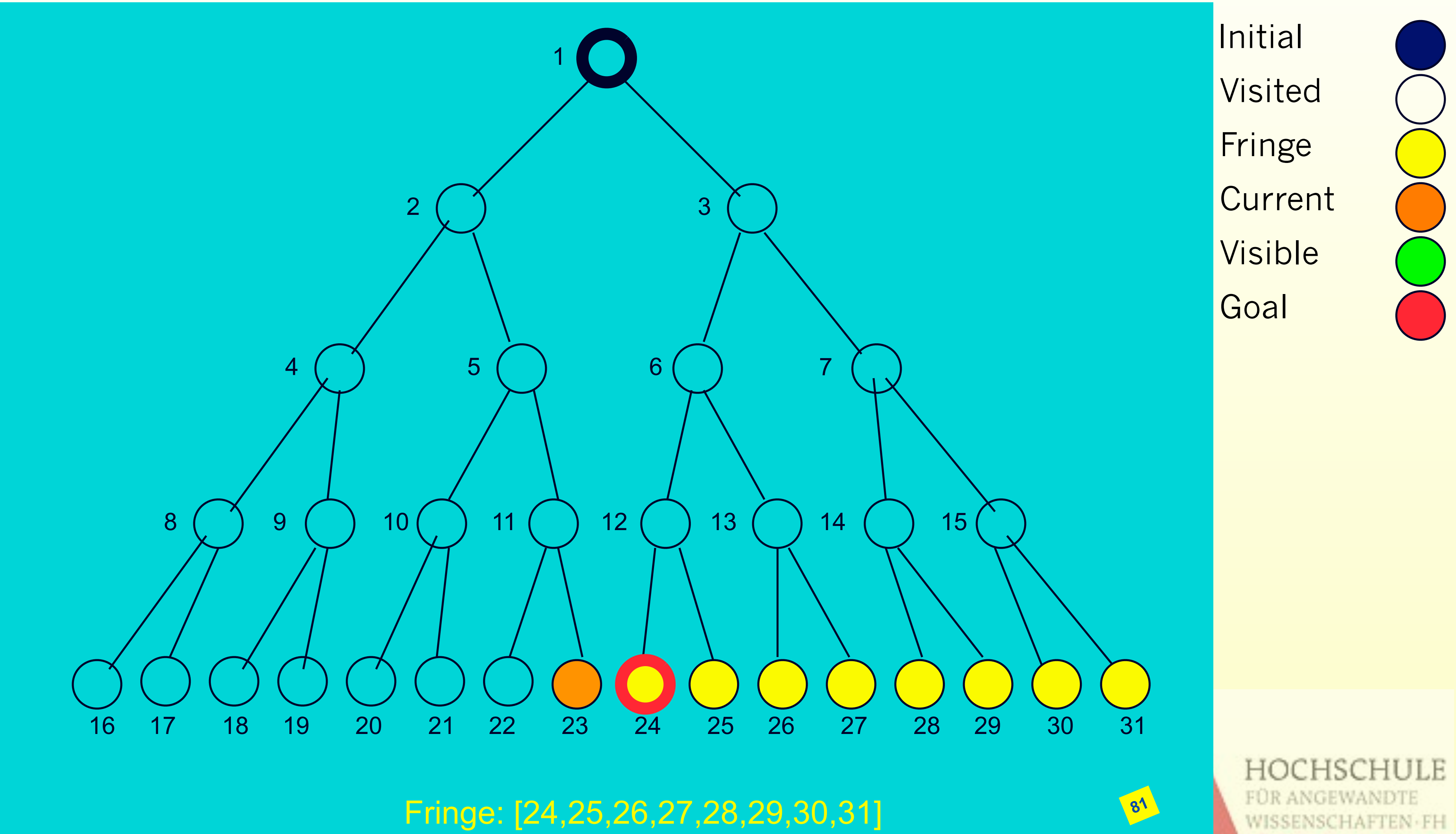
Fringe: [22,23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 22

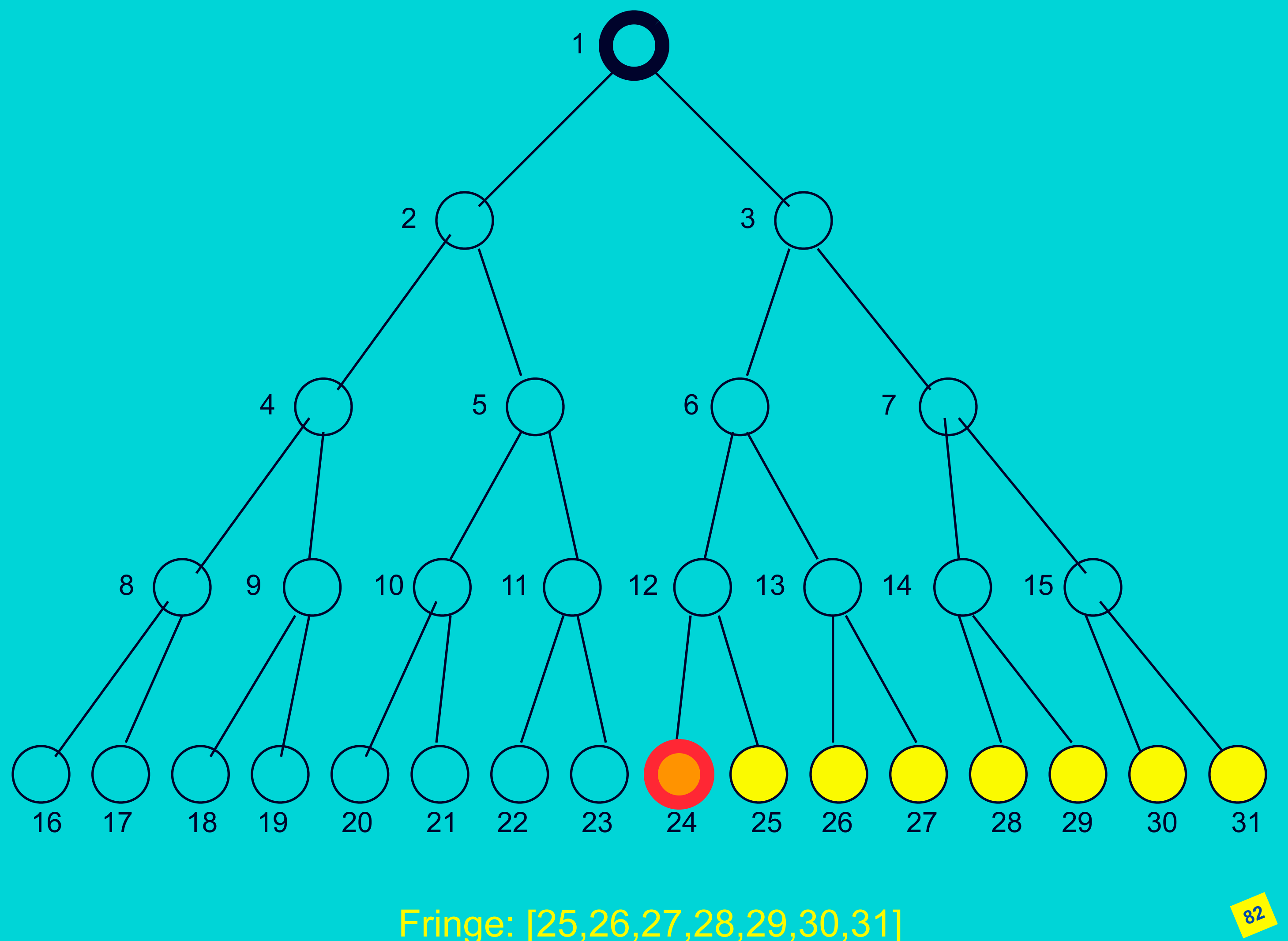


Fringe: [23,24,25,26,27,28,29,30,31]

Breadth-First Snapshot 23



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.

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())
```

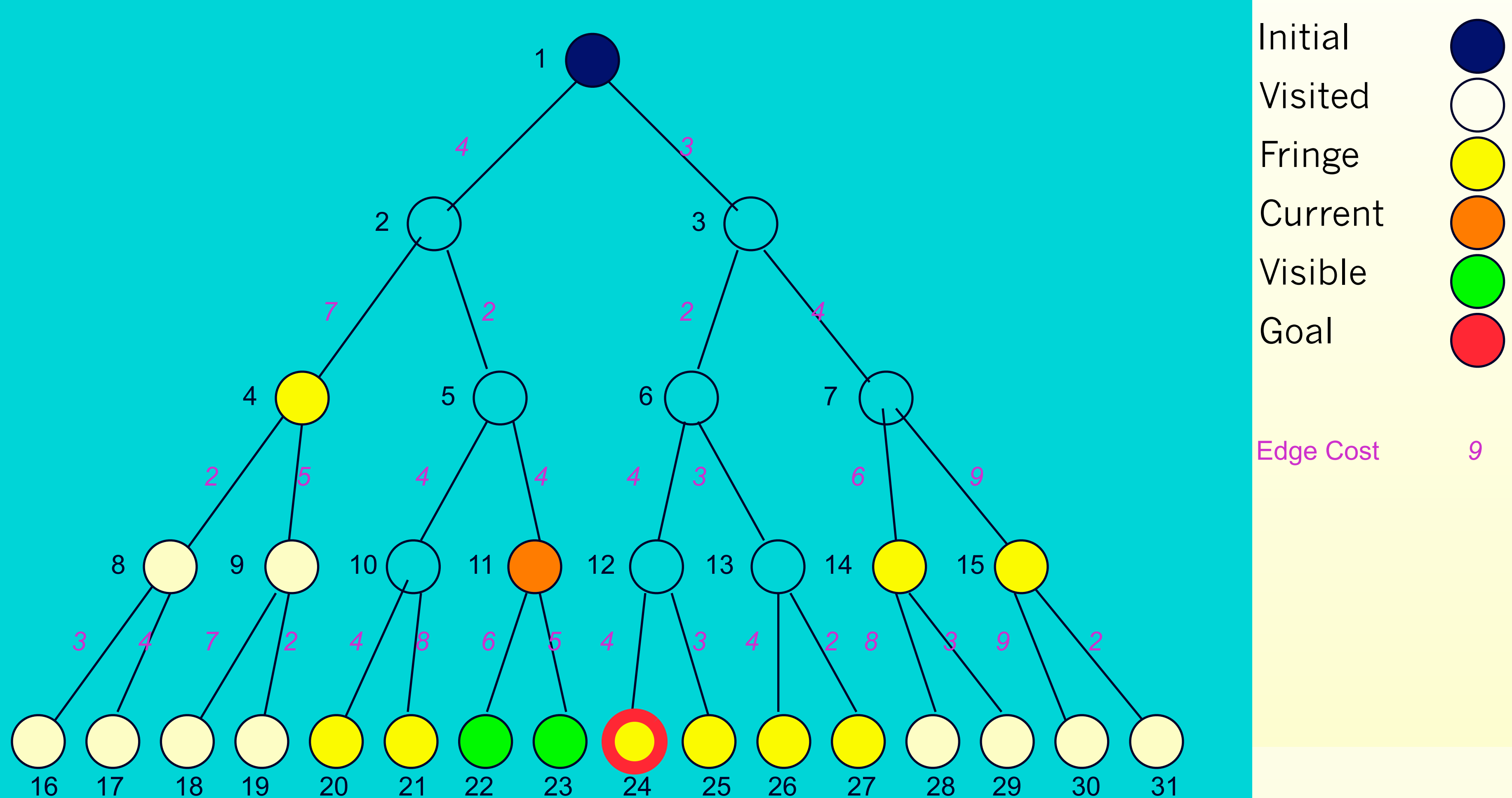
Time Complexity	$b^{C^*/e}$
Space Complexity	$b^{C^*/e}$
Completeness	yes (finite b, step costs >= e)
Optimality	yes

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

CAL POLY

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

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 cost function

Review: Breadth-First vs. Uniform-Cost

- **basic idea:**
  - breadth-first
  - uniform-cost
- **properties**
  - completeness
  - optimality
  - time complexity
  - space complexity

Depth-First

plain depth-first
limited depth
iterative deepening

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())
```

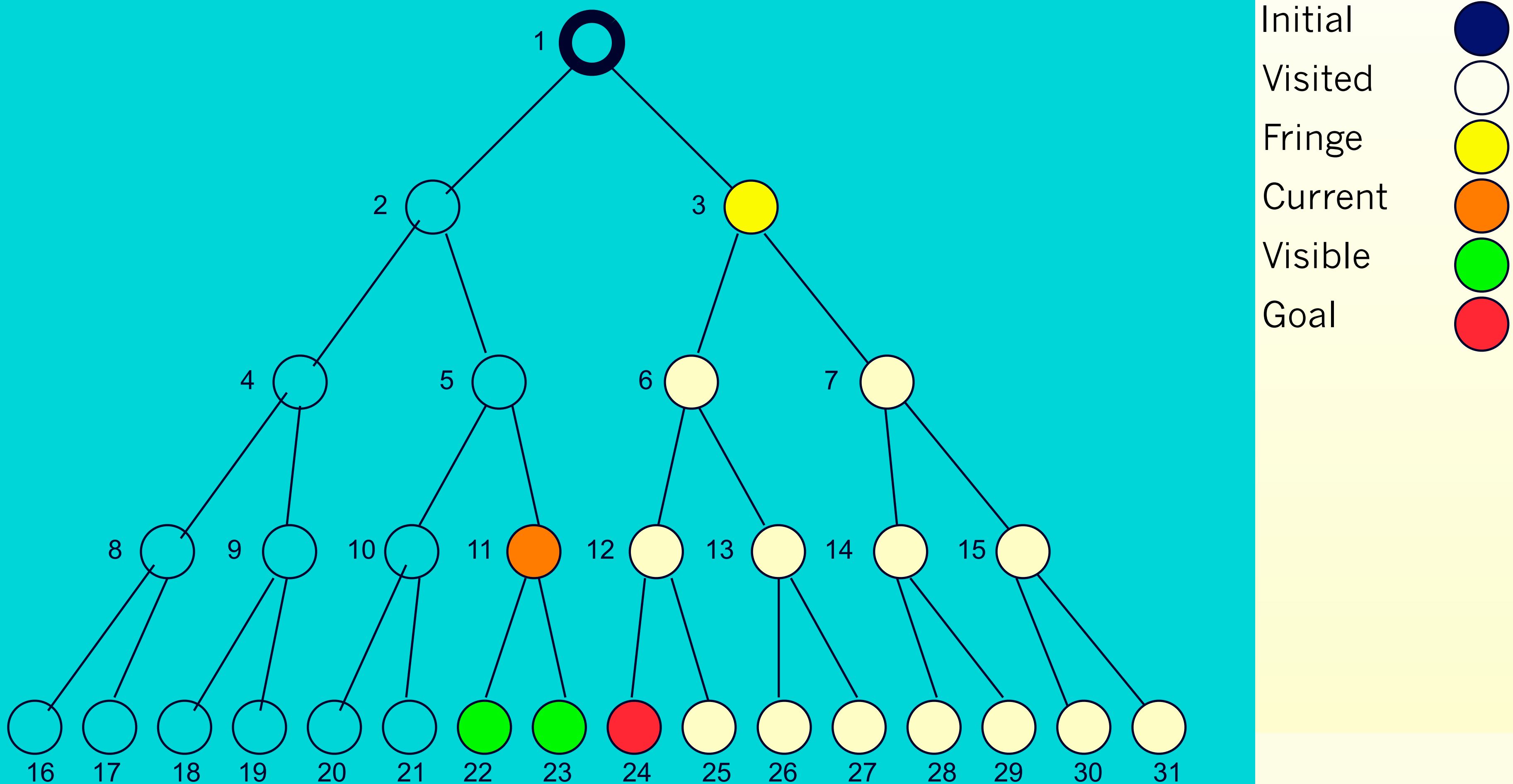
Time Complexity	b^m
Space Complexity	b*m
Completeness	no (for infinite branch length)
Optimality	no

b	branching factor
m	maximum path length

CAL POLY

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

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

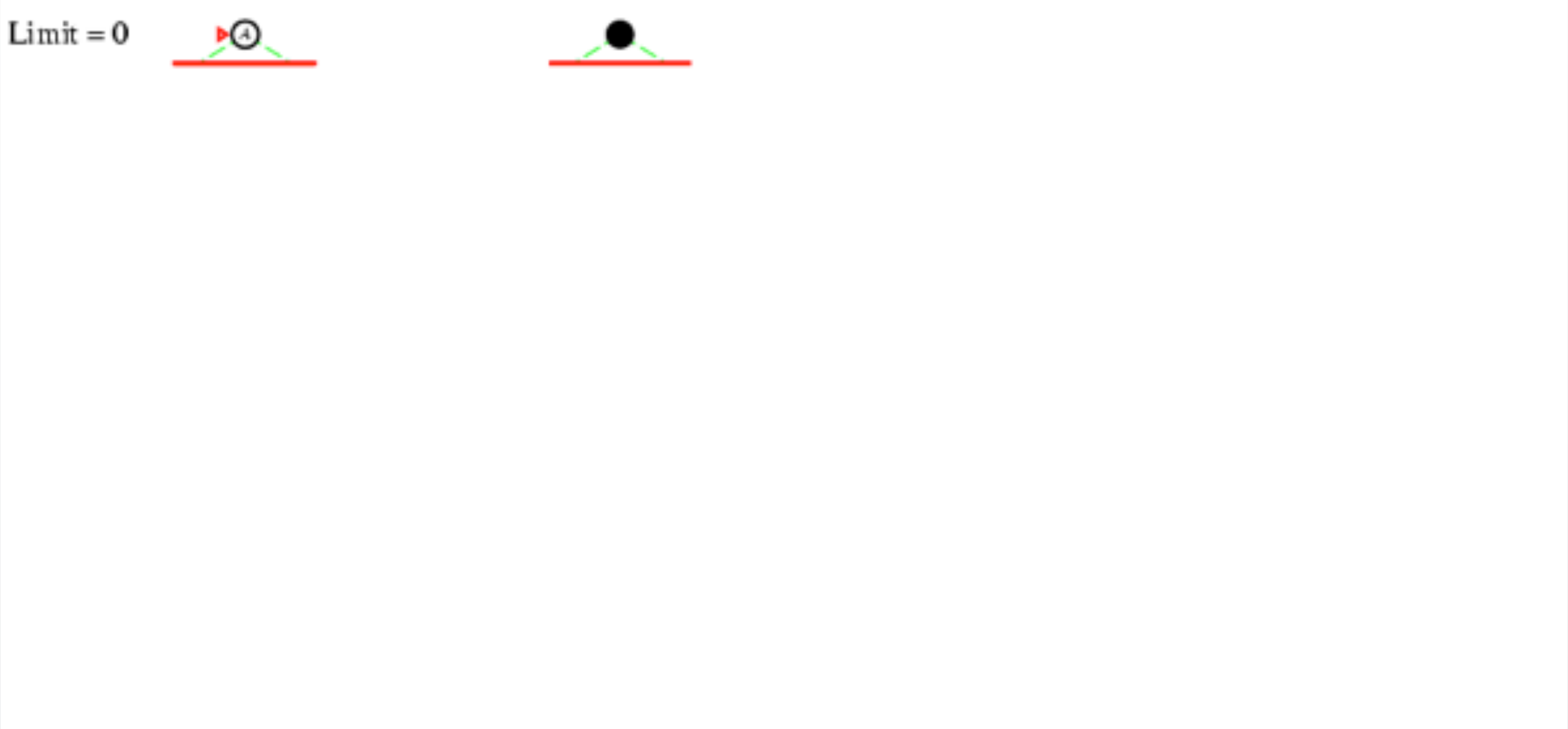
- **applies LIMITED-DEPTH with increasing depth limits**
  - combines advantages of BREADTH-FIRST and DEPTH-FIRST methods
  - many states are expanded multiple times
    - doesn't really matter because the number of those nodes is small
  - in practice, one of the best uninformed search methods
    -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

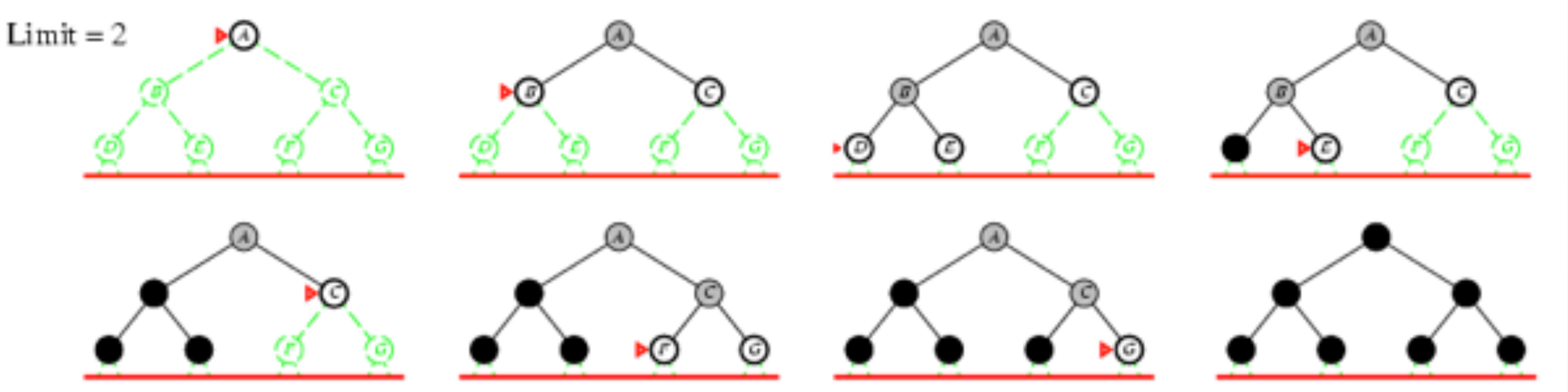


http://aima.eecs.berkeley.edu/slides-ppt/

Iterative deepening search l =1

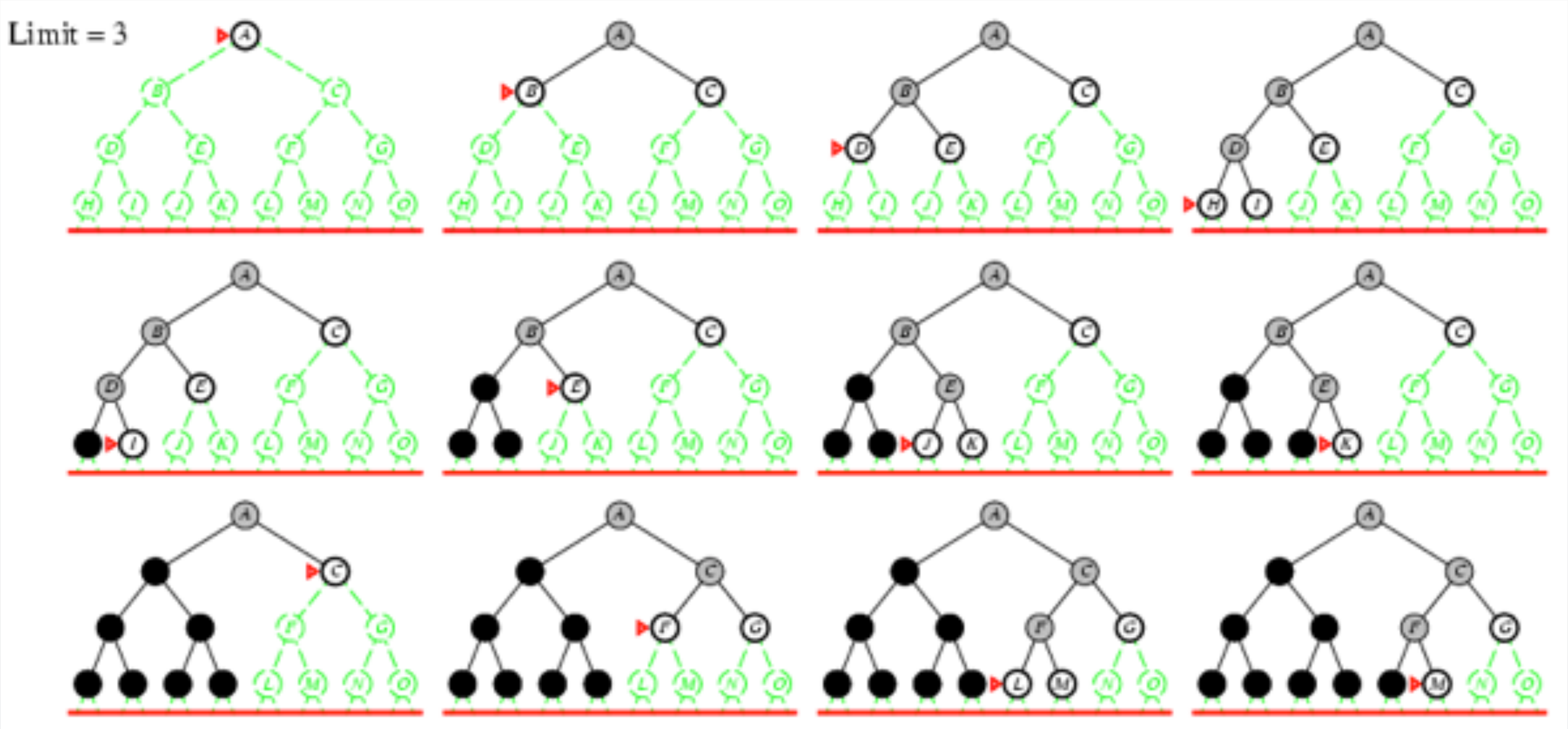
http://aima.eecs.berkeley.edu/slides-ppt/

Iterative deepening search l =2



http://aima.eecs.berkeley.edu/slides-ppt/

Iterative deepening search l =3



http://aima.eecs.berkeley.edu/slides-ppt/

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:

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

- 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%**

http://aima.eecs.berkeley.edu/slides-ppt/

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

- **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**
  - 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

Best-first

Greedy best-first

A*

A* modifications

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

```
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

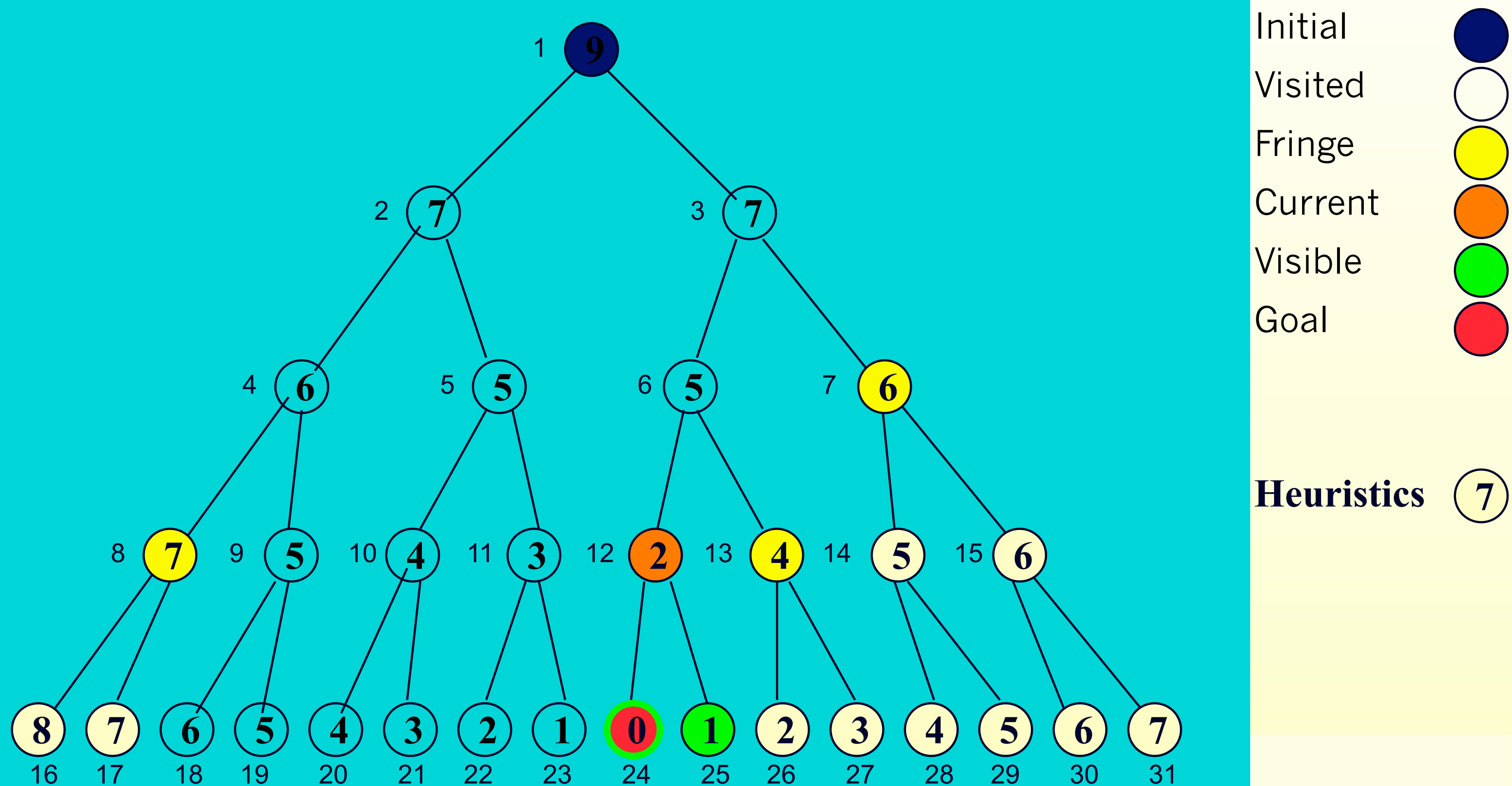
- **minimizes the estimated cost to a goal**
  - expand the node that seems to be closest to a goal
  - utilizes a heuristic function as evaluation function
    - $f(n) = h(n)$ = estimated cost from the current node to a goal
    - heuristic functions are problem-specific
    - often straight-line distance 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**)]

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)
```

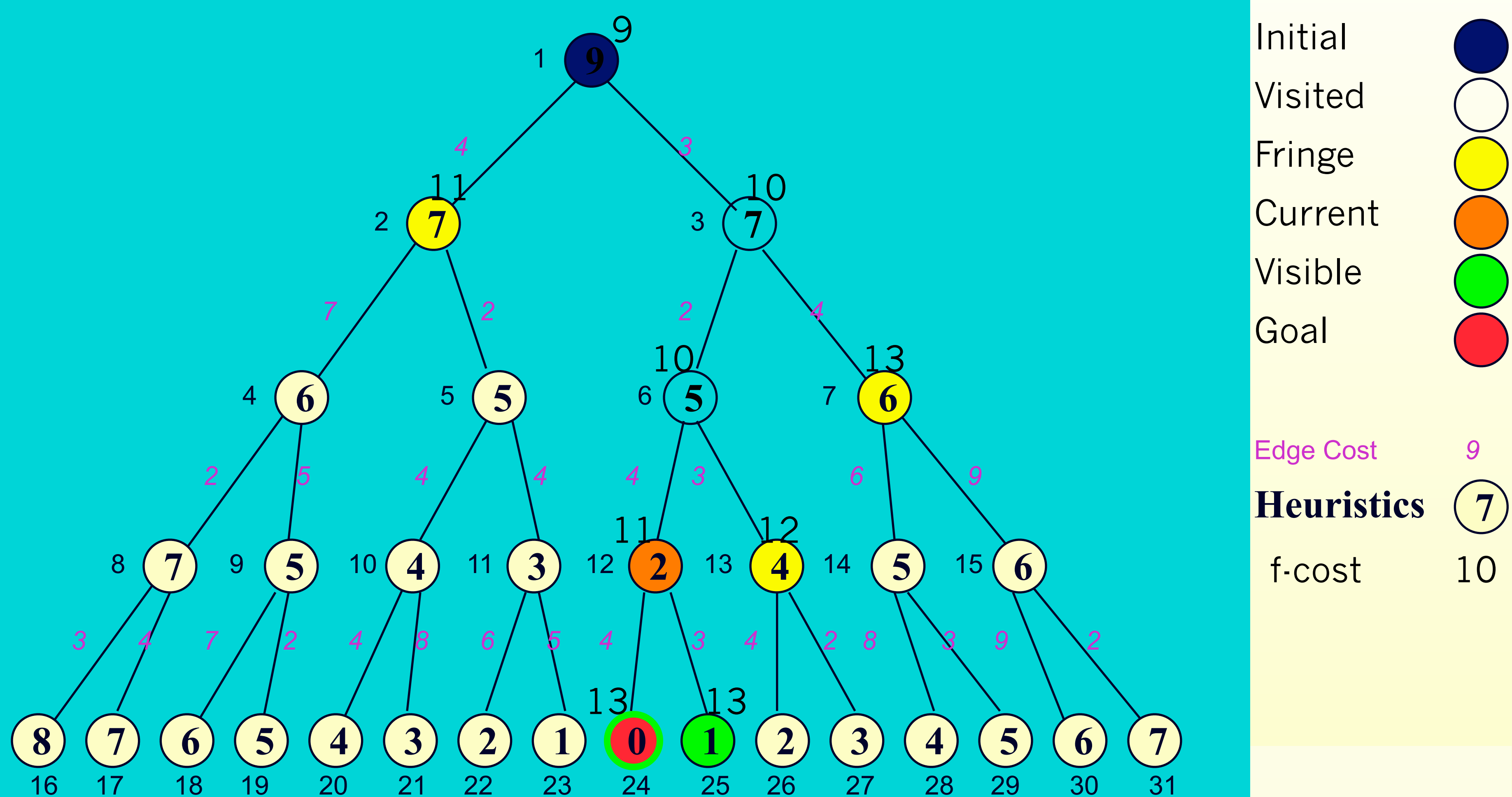
Completeness	Time Complexity	Space Complexity	Optimality
yes	b^d	b^d	yes

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

CAL POLY

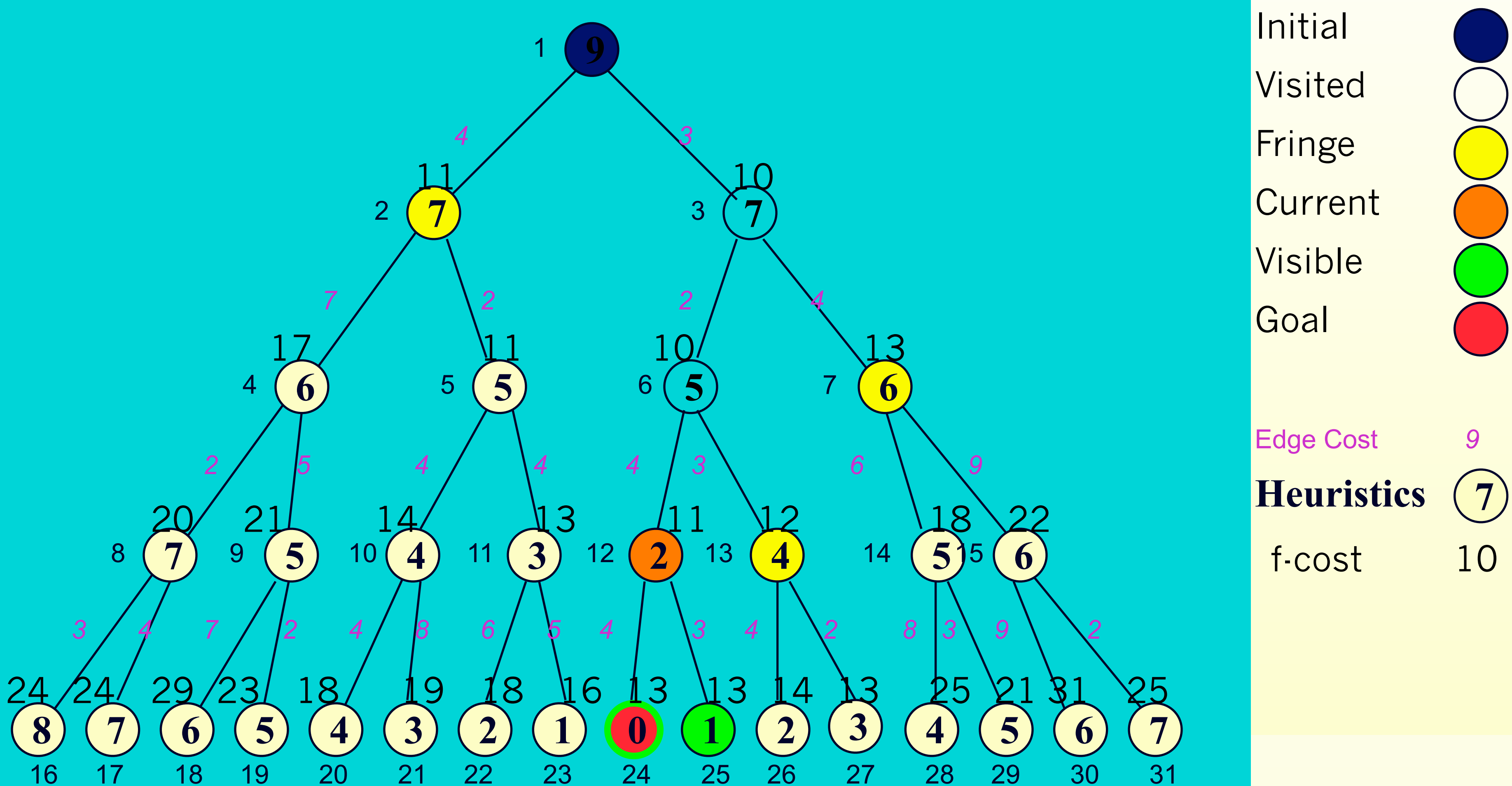
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

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)]

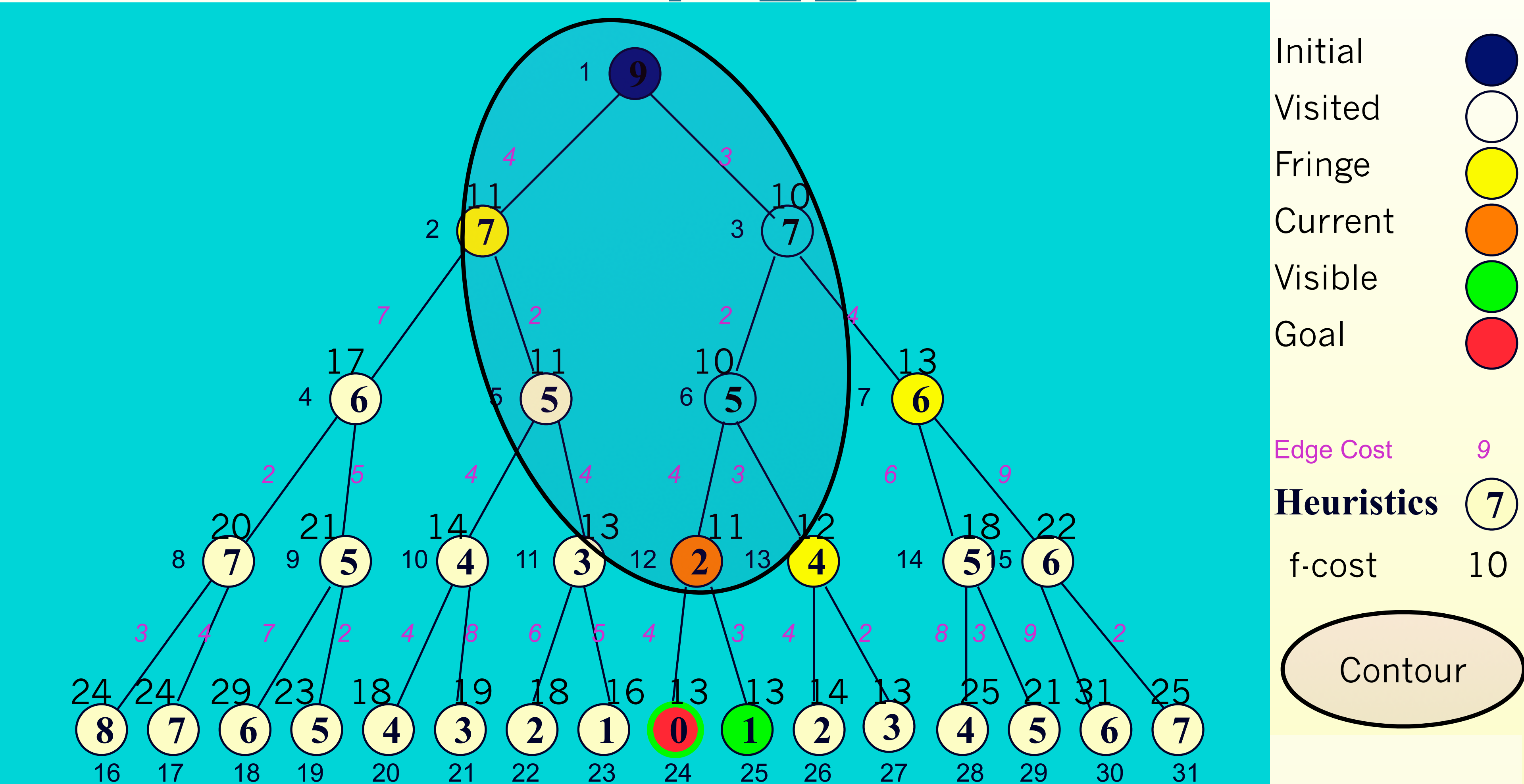
A* Snapshot with all f-Costs



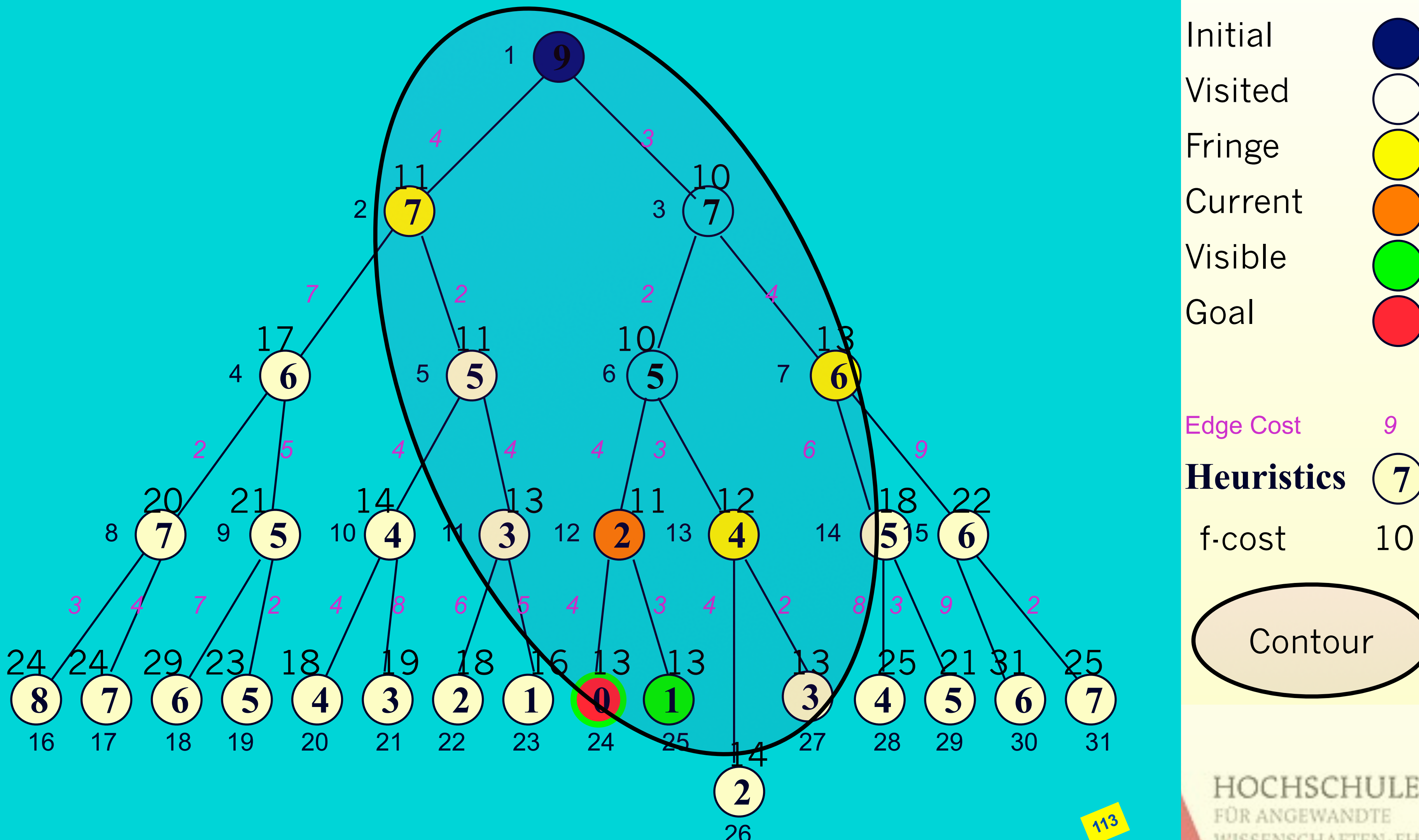
A* Properties

- **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(bm)
- **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

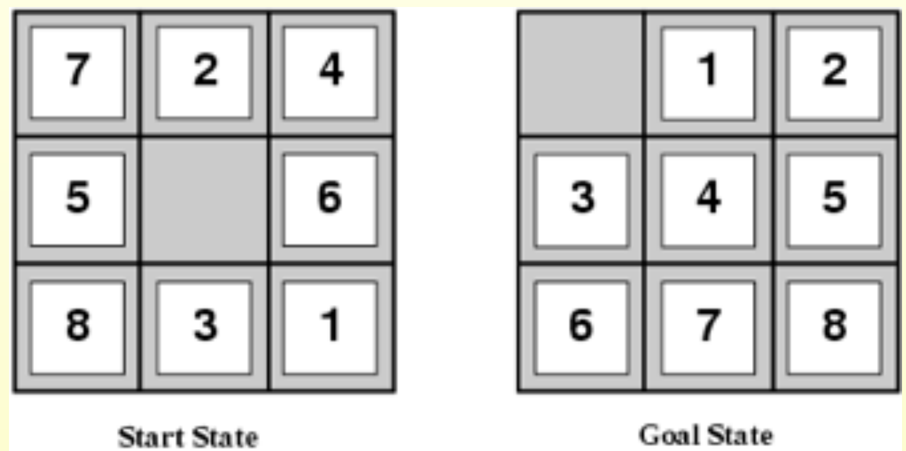
- **for many tasks, a good heuristic is the key to finding a solution**
  - prune the search space
  - move towards the goal
- **relaxed problems**
  - fewer restrictions on the successor function (operators)
  - its exact solution may be a good heuristic for the original problem

8-Puzzle Heuristics

- **level of difficulty**
  - around 20 steps for a typical solution
  - branching factor is about 3
    - exhaustive search would be 320 = 3.5 * 109
  - 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:**
  - h1(n) = number of misplaced tiles
  - h2(n) = total Manhattan distance
  - (i.e., no. of squares from desired location of each tile)



  - h1(S) = ? 8
  - h2(S) = ? 3+1+2+2+2+3+3+2 = 18

http://aima.eecs.berkeley.edu/slides-ppt/

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