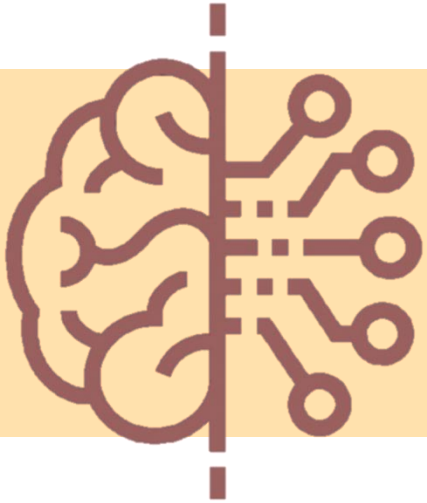
Exhaustive Search Algorithm

(Exhaustive – Uninformed - Blind Search)



Artificial Intelligence

School of Computing
Universiti Teknologi Malaysia

Outline

1. Selection of a Search Strategy
   - Breadth-first
   - Depth-first
   - Comparison of both search strategy
2. Using State Space to Represent Reasoning
   - And/or graph
   - Hypergraph

Selection of a Search Strategy

- Breadth-first
- Depth-first
- Comparison of both search strategy

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

Search Strategy : The order

Uninformed Search

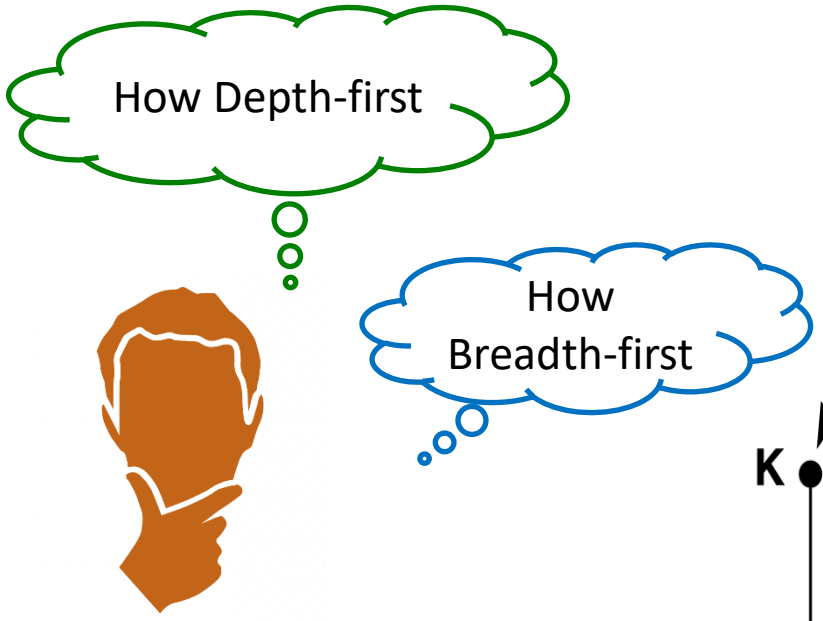
- Breadth-first
- Depth-first
- Iterative deepening
- Uniform-cost search
- Depth-limited search
- Bi-directional search
- Constraint satisfaction

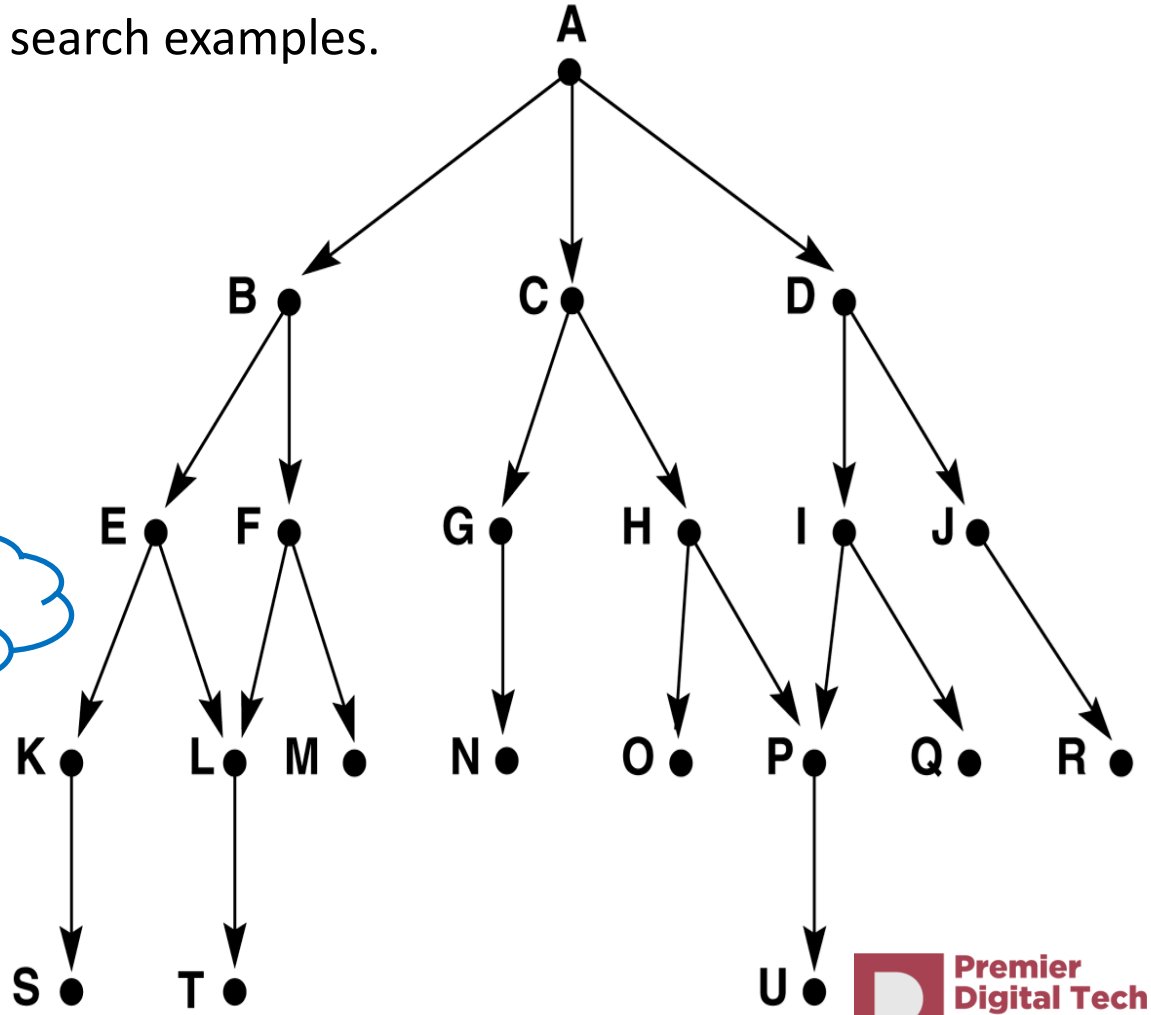
Informed Search

- Best-first search
- Search with heuristics
- Memory-bounded search
- Iterative improvement search

Search Strategy

Graph for breadth and depth-first search examples.





Search Strategy : 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

Solution is found

Search Strategy : Breadth-First

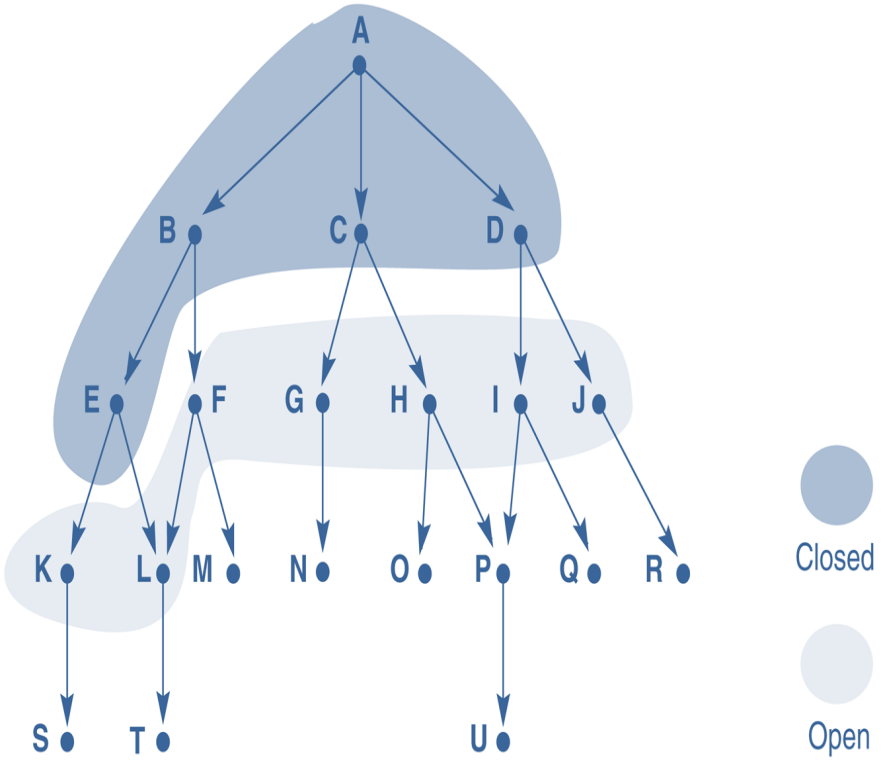
Function Breadth-first search algorithm

```
function breadth_first_search;

begin
   open := [Start];                                          % initialize
   closed := [ ];
   while open ≠ [ ] do                                       % states remain
      begin
         remove leftmost state from open, call it X;
            if X is a goal then return SUCCESS               % goal found
               else begin
                  generate children of X;
                  put X on closed;
                  discard children of X if already on open or closed;   % loop check
                  put remaining children on right end of open           % queue
               end
      end
   return FAIL                                               % no states left
end.
```

Search Strategy : Breadth-First

Function Breadth-first search algorithm

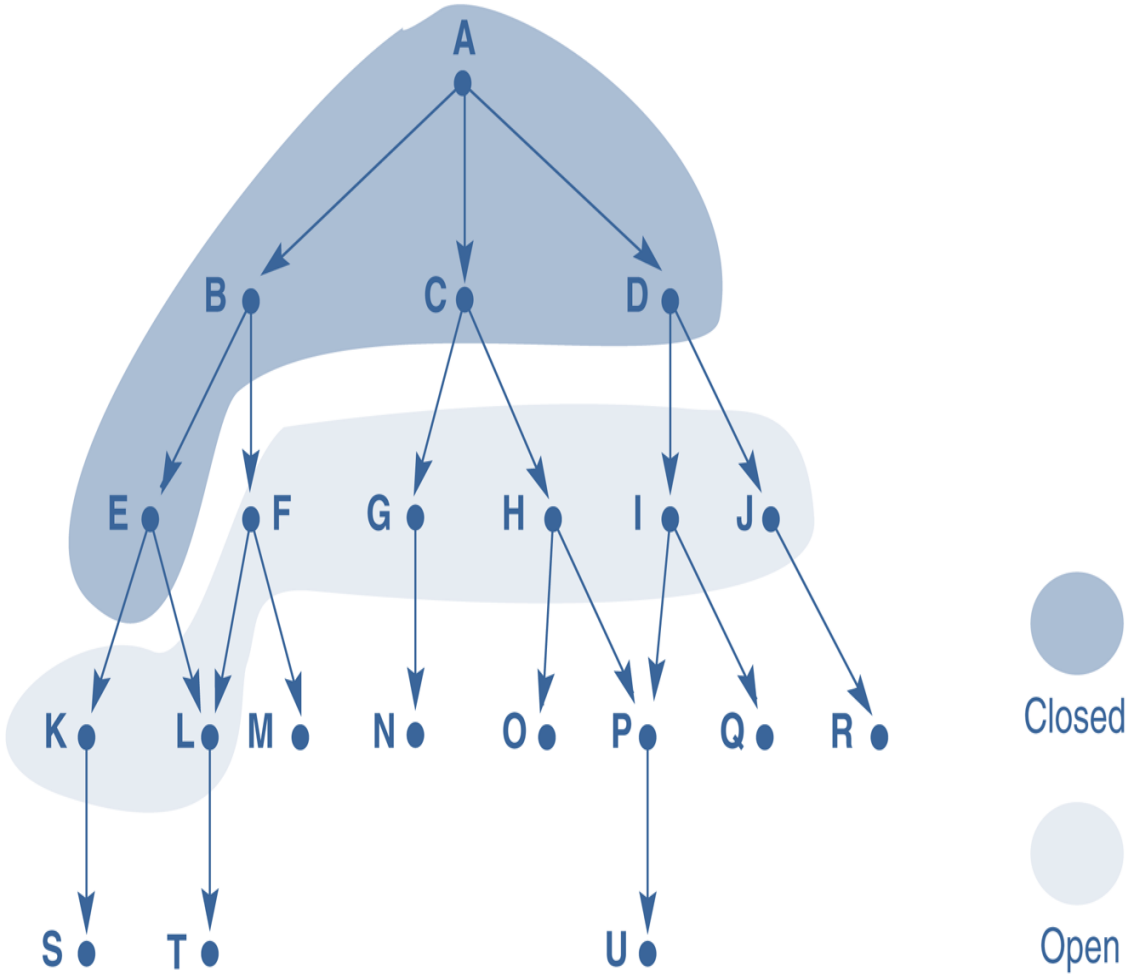


1. **open = [A]; closed = [ ]**
2. **open = [B,C,D]; closed = [A]**
3. **open = [C,D,E,F]; closed = [B,A]**
4. **open = [D,E,F,G,H]; closed = [C,B,A]**
5. **open = [E,F,G,H,I,J]; closed = [D,C,B,A]**
6. **open = [F,G,H,I,J,K,L]; closed = [E,D,C,B,A]**
7. **open = [G,H,I,J,K,L,M]** (as L is already on open); **closed = [F,E,D,C,B,A]**
8. **open = [H,I,J,K,L,M,N]; closed = [G,F,E,D,C,B,A]**
9. and so on until either U is found or **open** = [ ]

Search Strategy : Breadth-First

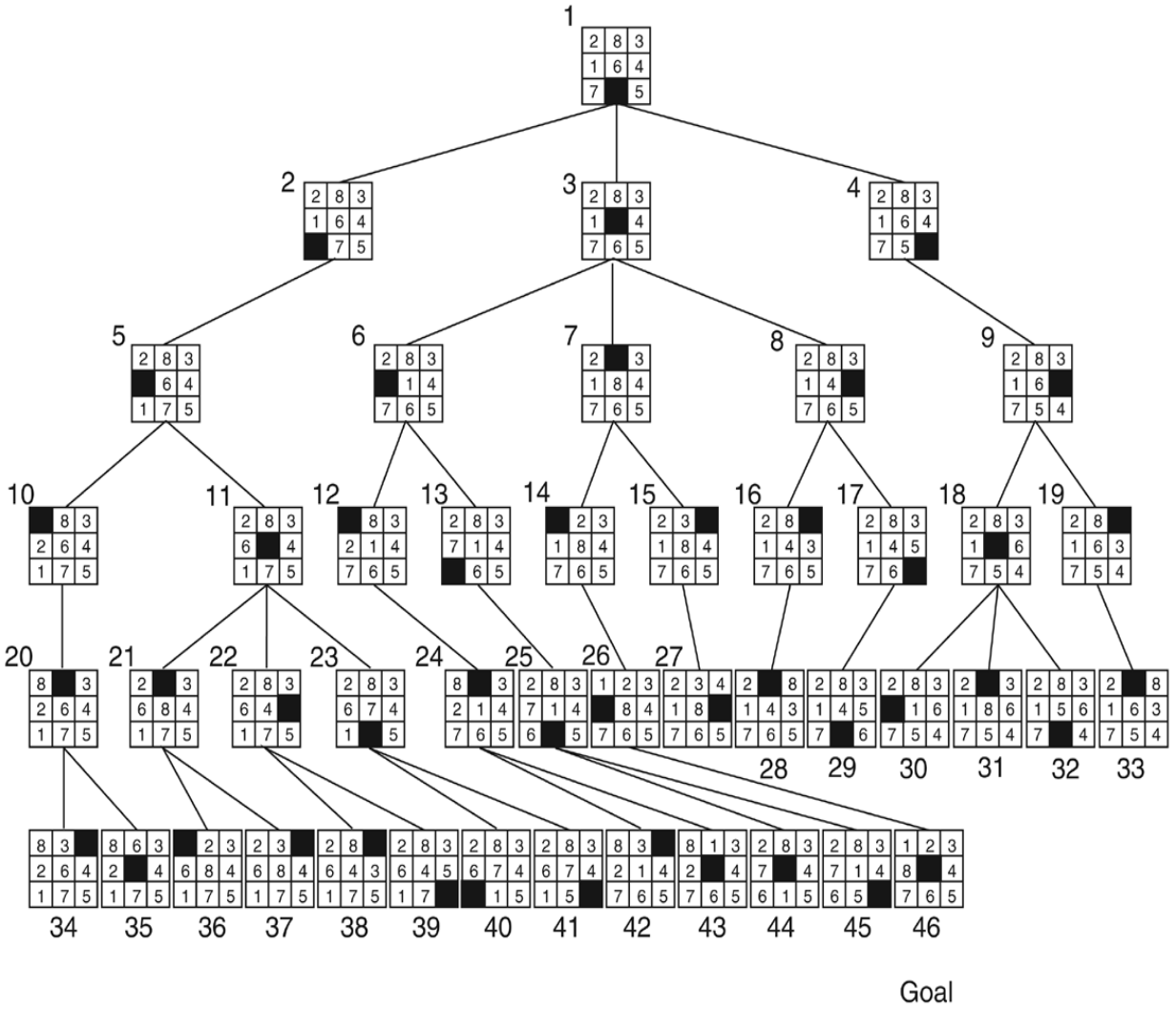
Function Breadth-first search algorithm

- States on open and closed are highlighted.
- Explore level by level
- Use list open in backtrack – states not yet evaluated
- Use list close in backtrack – states already evaluated
- Which state is removed from open determines the order of search- BFS adds at the right list and removes from the left (queue-FIFO)



Search Strategy : Breadth-First

Breadth-first search of the 8-puzzle, showing order in which states were removed from open



Search Strategy : Depth-First

Function Depth-first search algorithm

```
begin
   open := [Start];                                              % initialize
   closed := [ ];
   while open ≠ [ ] do                                           % states remain
      begin
         remove leftmost state from open, call it X;
         if X is a goal then return SUCCESS                      % goal found
            else begin
               generate children of X;
               put X on closed;
               discard children of X if already on open or closed;   % loop check
               put remaining children on left end of open              % stack
            end
      end;
   return FAIL                                                   % no states left
end.
```

Search Strategy : Depth-First

Function Depth-first search algorithm

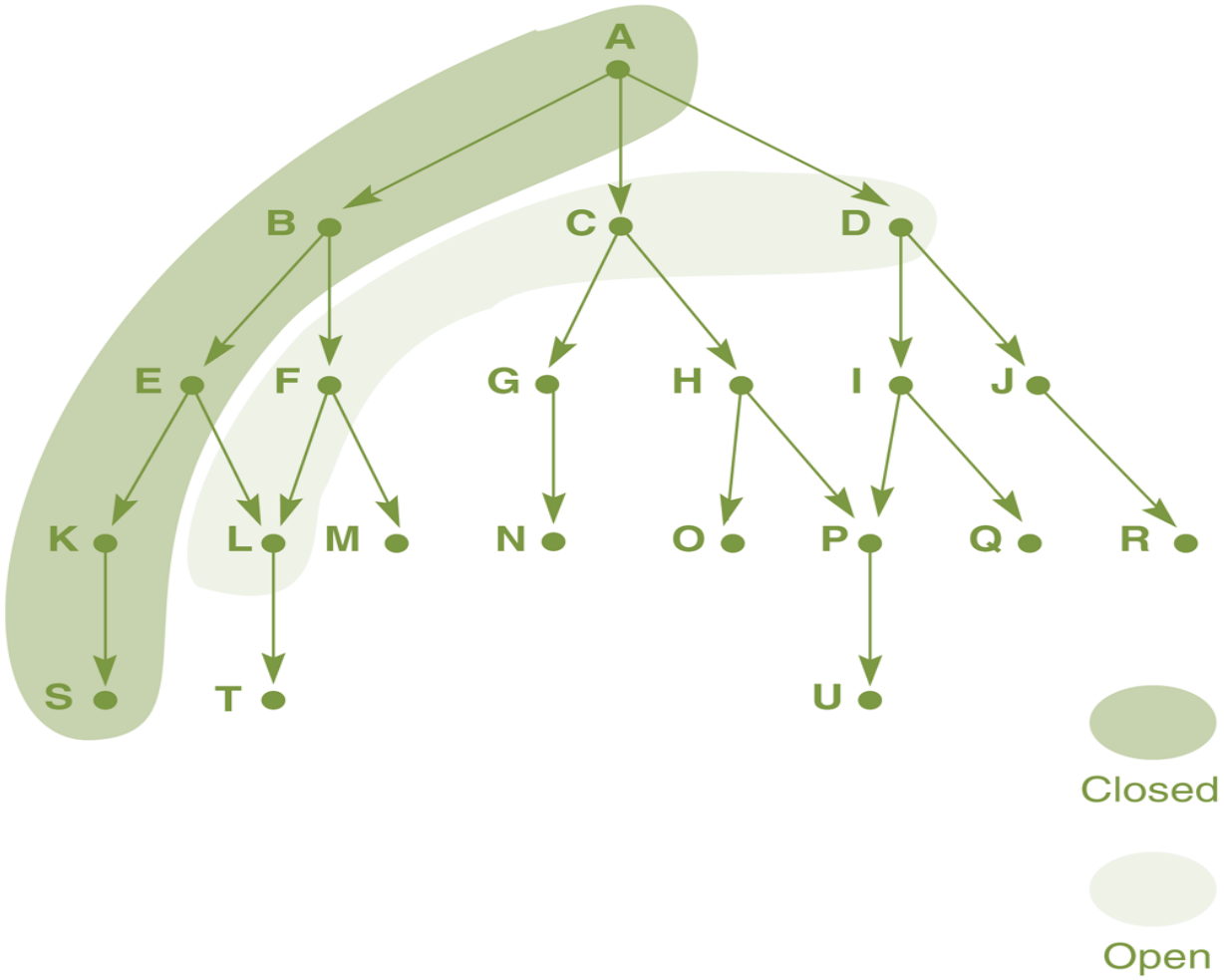
1. open = [A]; closed = [ ]
2. open = [B,C,D]; closed = [A]
3. open = [E,F,C,D]; closed = [B,A]
4. open = [K,L,F,C,D]; closed = [E,B,A]
5. open = [S,L,F,C,D]; closed = [K,E,B,A]
6. open = [L,F,C,D]; closed = [S,K,E,B,A]
7. open = [T,F,C,D]; closed = [L,S,K,E,B,A]
8. open = [F,C,D]; closed = [T,L,S,K,E,B,A]
9. open = [M,C,D], as L is already on closed; closed = [F,T,L,S,K,E,B,A]
10. open = [C,D]; closed = [M,F,T,L,S,K,E,B,A]
11. open = [G,H,D]; closed = [C,M,F,T,L,S,K,E,B,A]

Search Strategy : Depth-First

Function Depth-first search algorithm

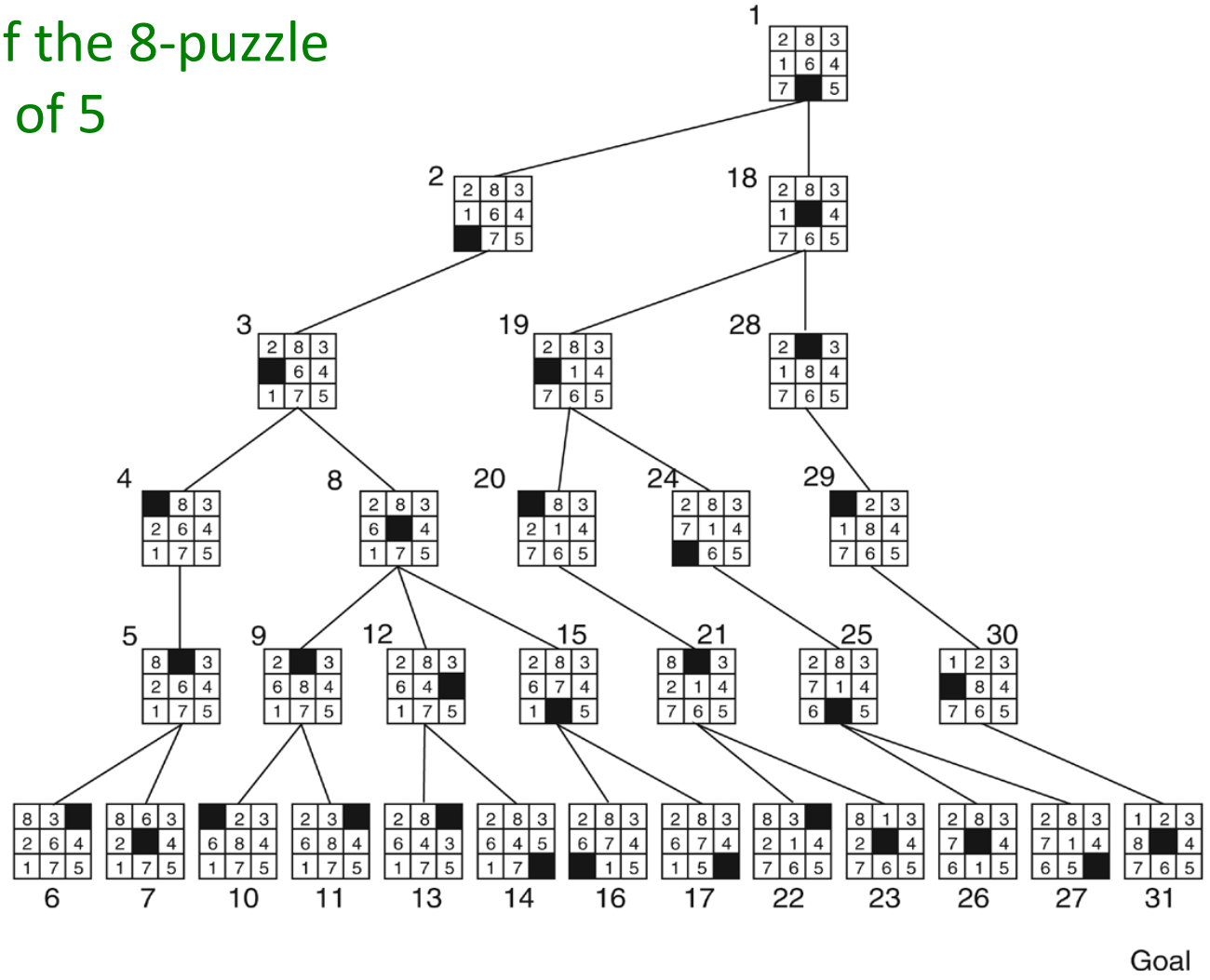
States on open and closed are highlighted.

- Child and descendants are evaluated before siblings
- Goes deeper into search space whenever possible
- Which state is removed from open determines the order of search-DFS adds and removes from the left end (stack-LIFO)

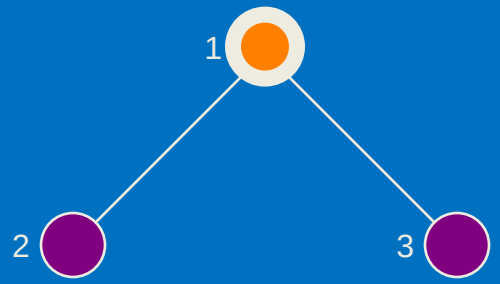


Search Strategy : Depth-First

Depth-first search of the 8-puzzle with a depth bound of 5



Breadth-First Snapshot 1

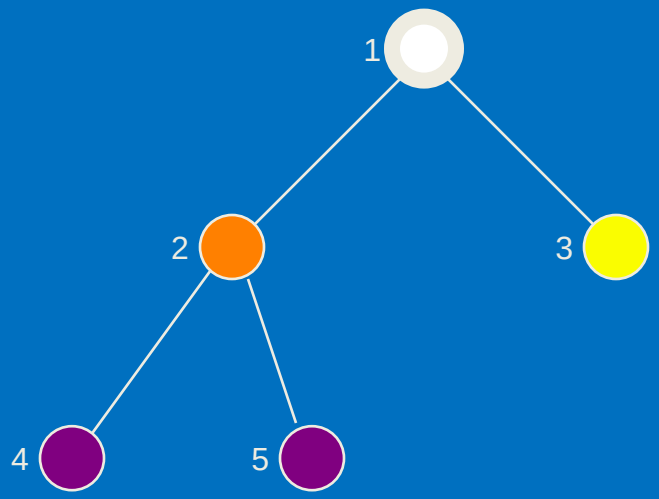


Initial
Visited
Fringe
Current
Visible
Goal

Fringe = nodes waiting in queue to be explored

Fringe: [] + [2,3]

Breadth-First Snapshot 2

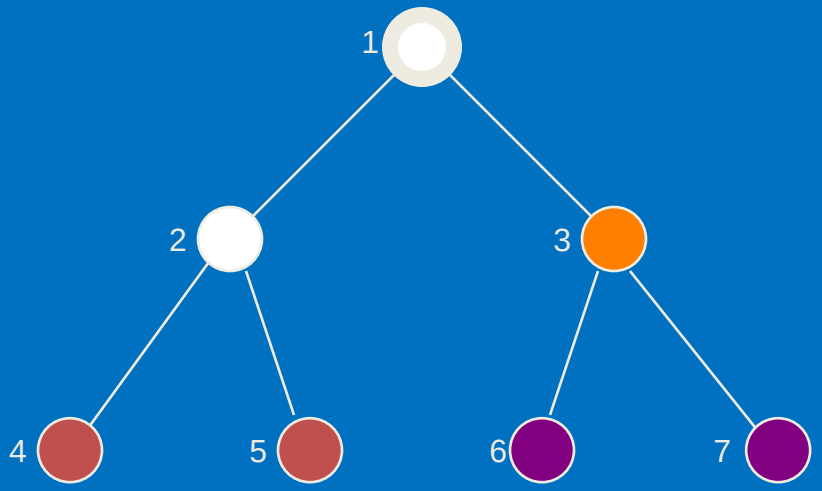


Initial
Visited
Fringe
Current
Visible
Goal

Fringe = nodes waiting in queue to be explored

Fringe: [3] + [4,5]

Breadth-First Snapshot 3

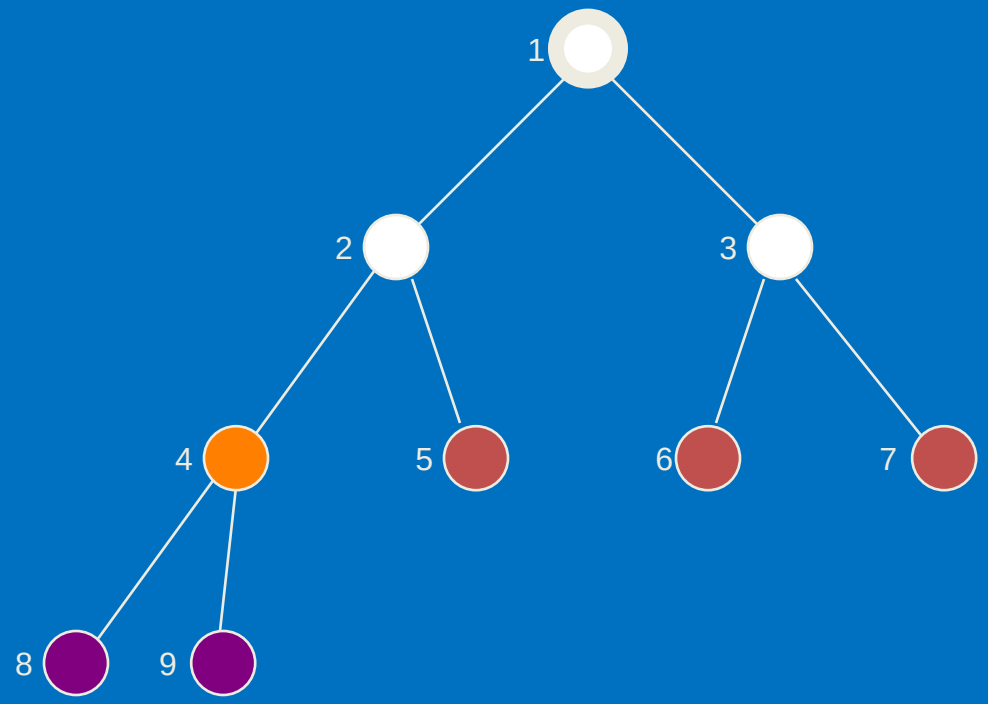


Initial
Visited
Fringe
Current
Visible
Goal

Fringe = nodes waiting in queue to be explored

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

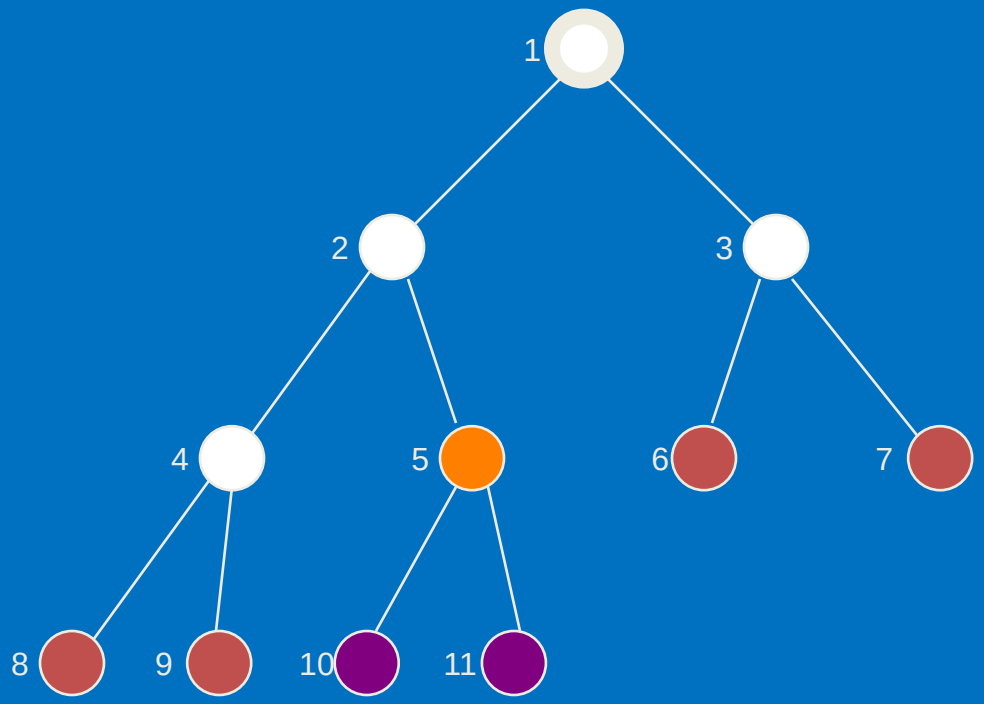
Breadth-First Snapshot 4



Initial
Visited
Fringe
Current
Visible
Goal

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

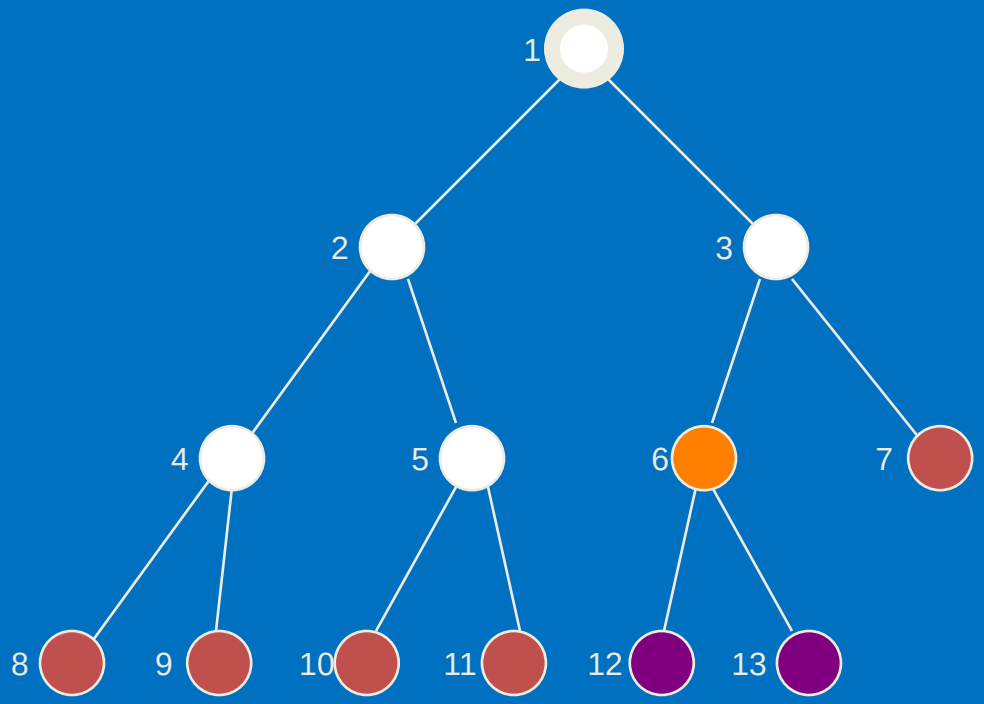
Breadth-First Snapshot 5



Initial
Visited
Fringe
Current
Visible
Goal

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

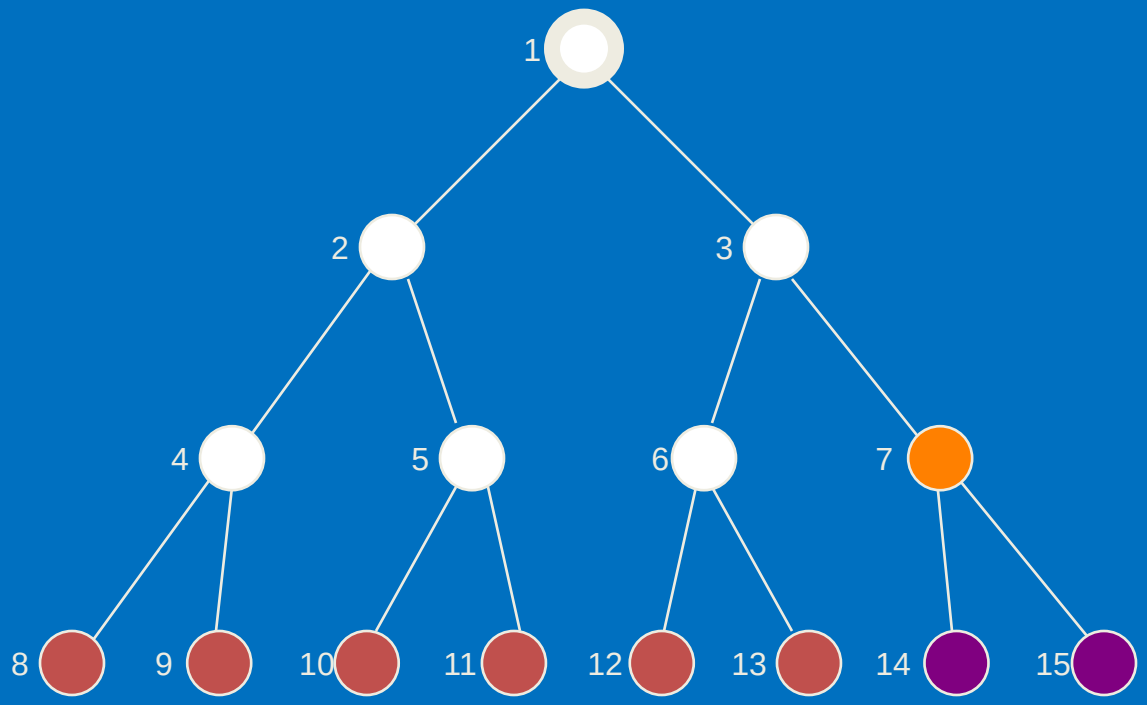
Breadth-First Snapshot 6



Initial
Visited
Fringe
Current
Visible
Goal

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

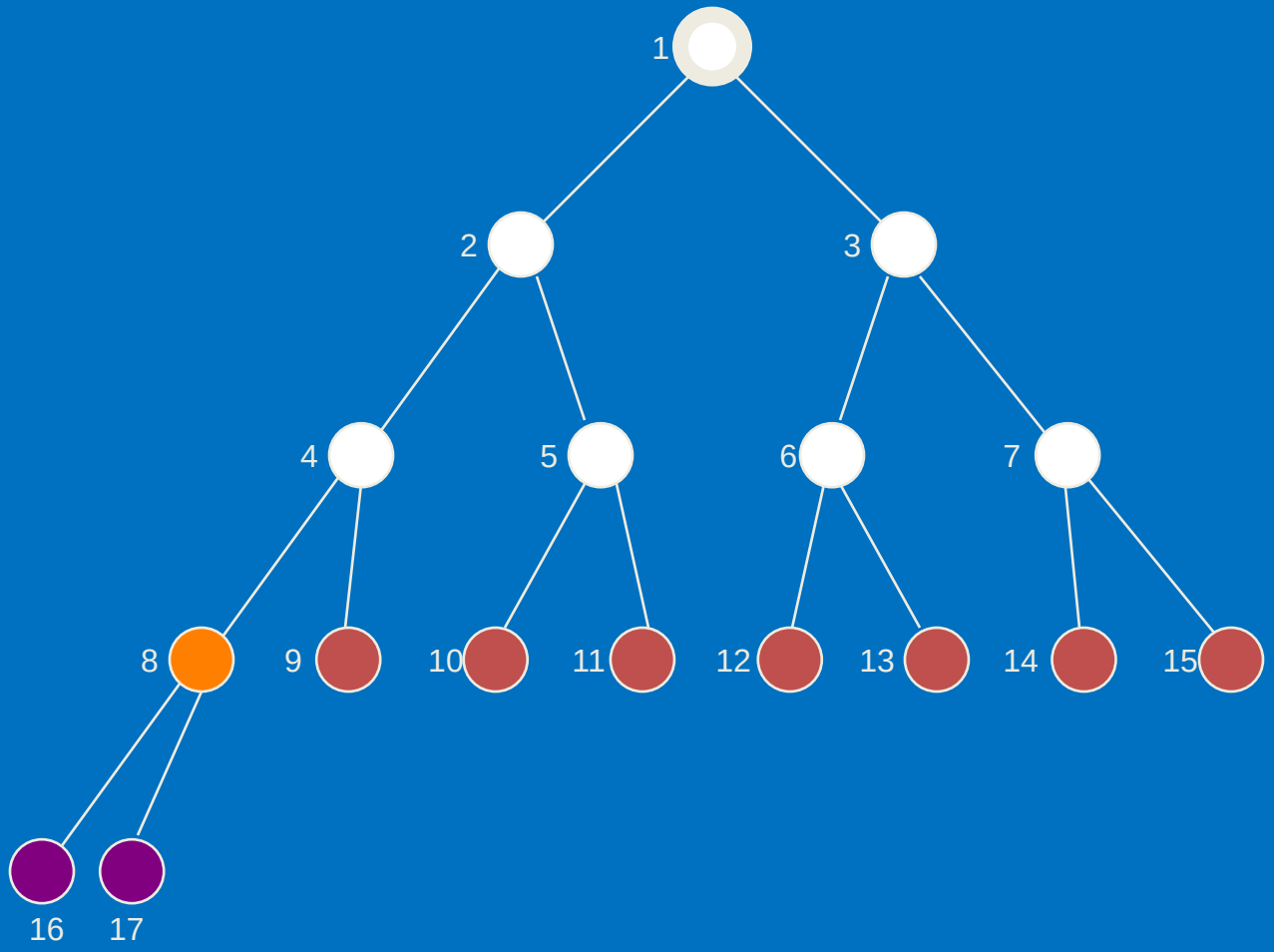
Breadth-First Snapshot 7



Initial
Visited
Fringe
Current
Visible
Goal

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

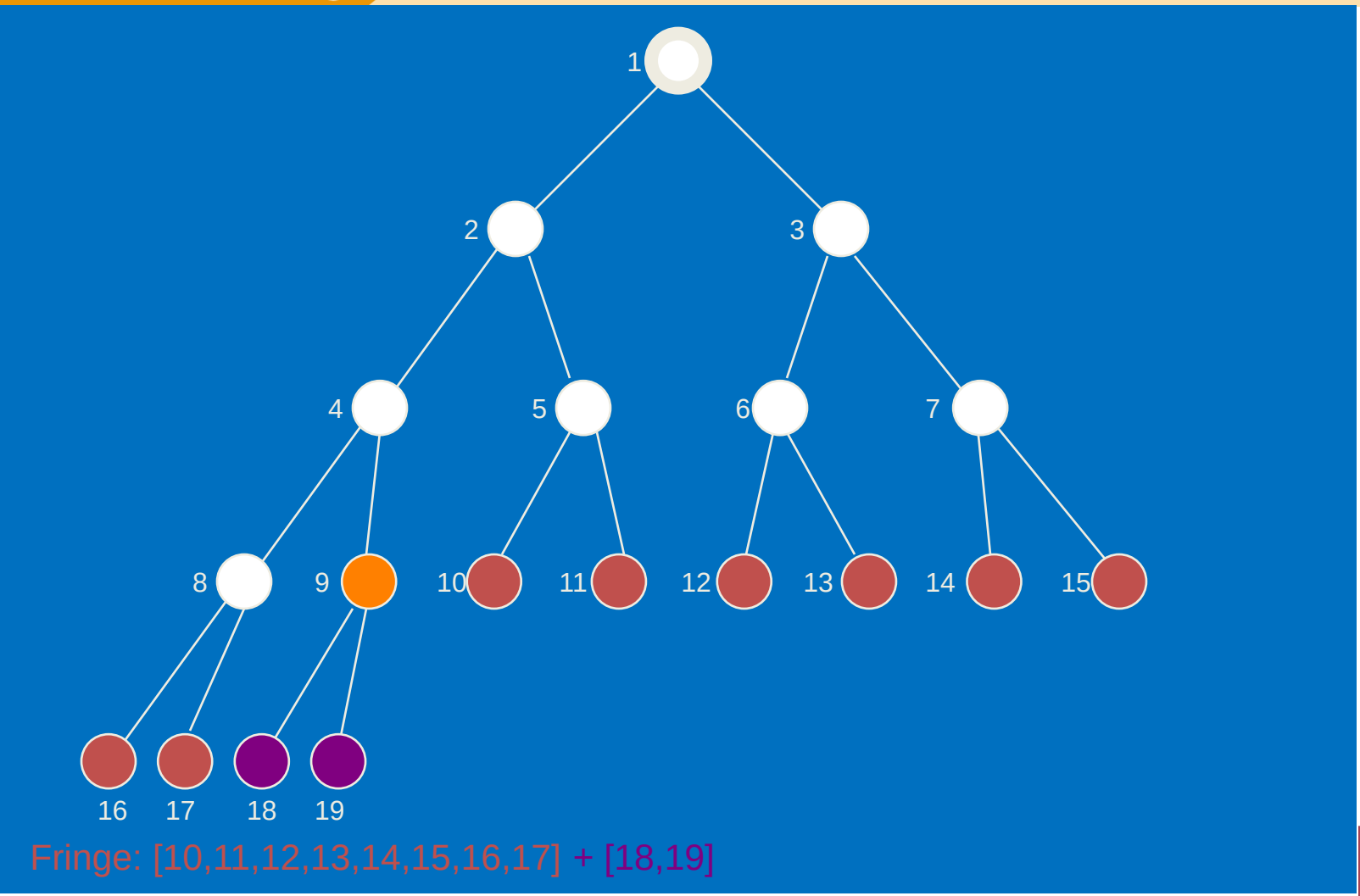
Breadth-First Snapshot 8



Initial
Visited
Fringe
Current
Visible
Goal

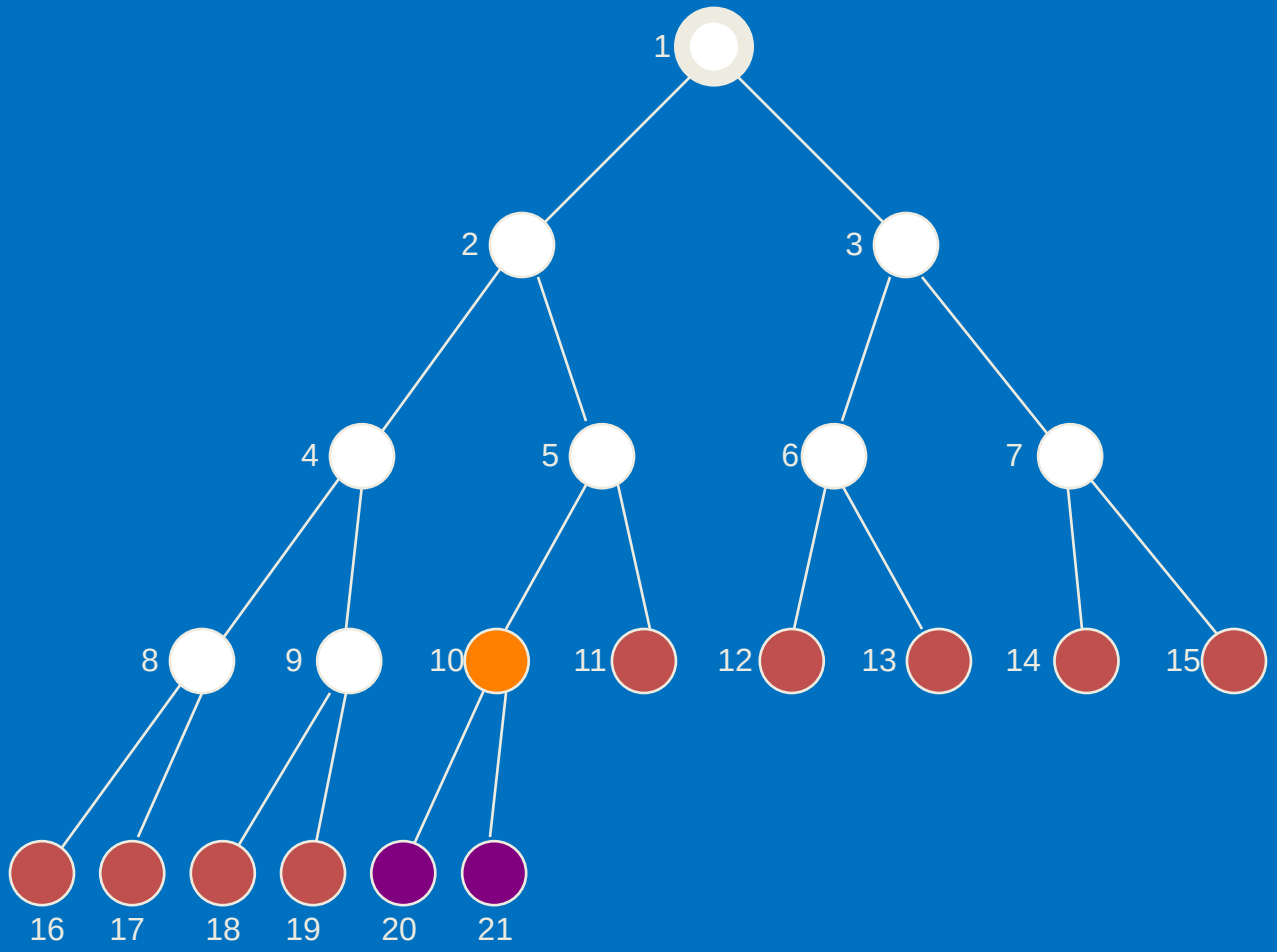
Fringe: [9,10,11,12,13,14,15] + [16,17]

Breadth-First Snapshot 9



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

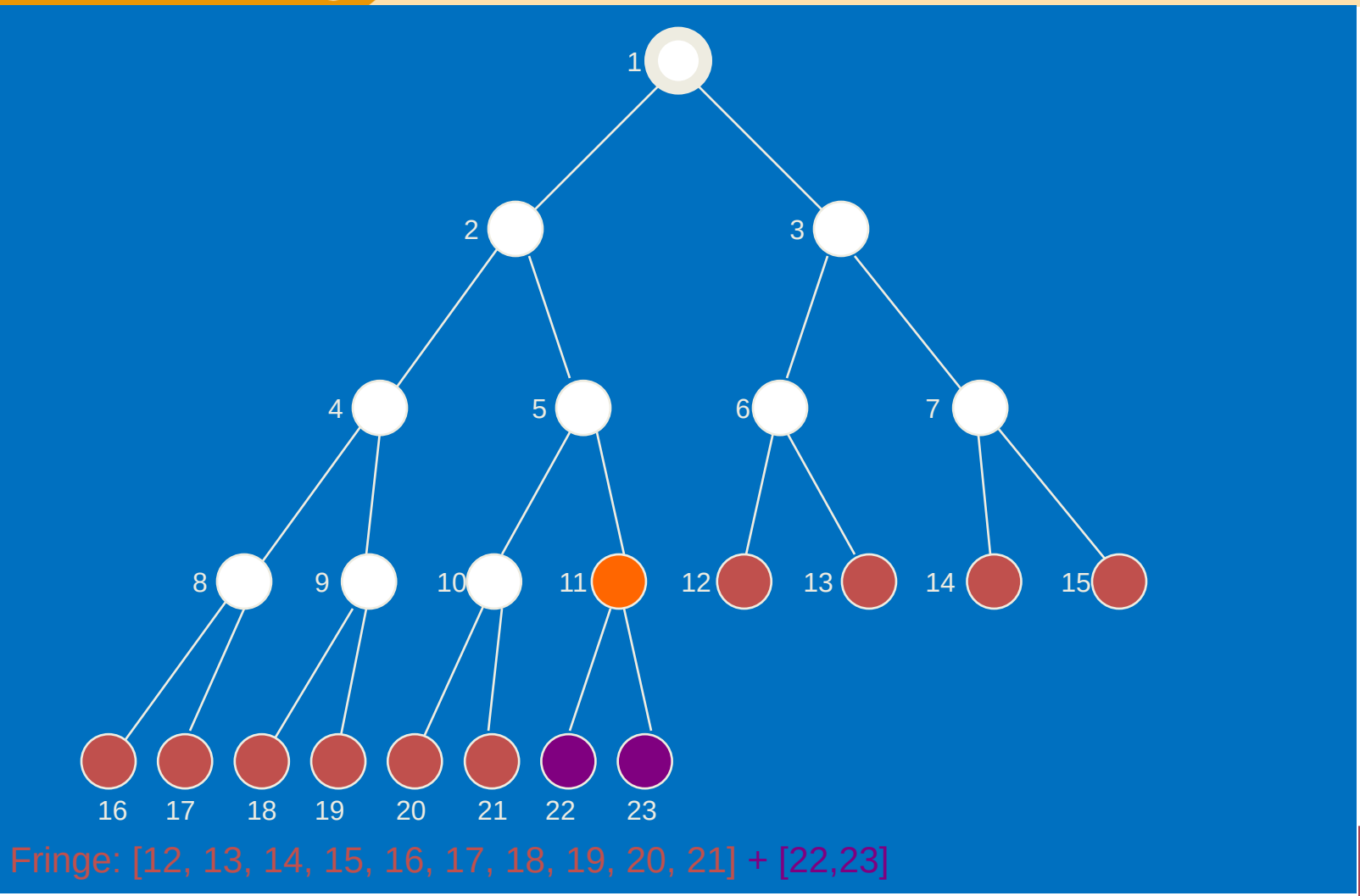
Breadth-First Snapshot 10



Initial
Visited
Fringe
Current
Visible
Goal

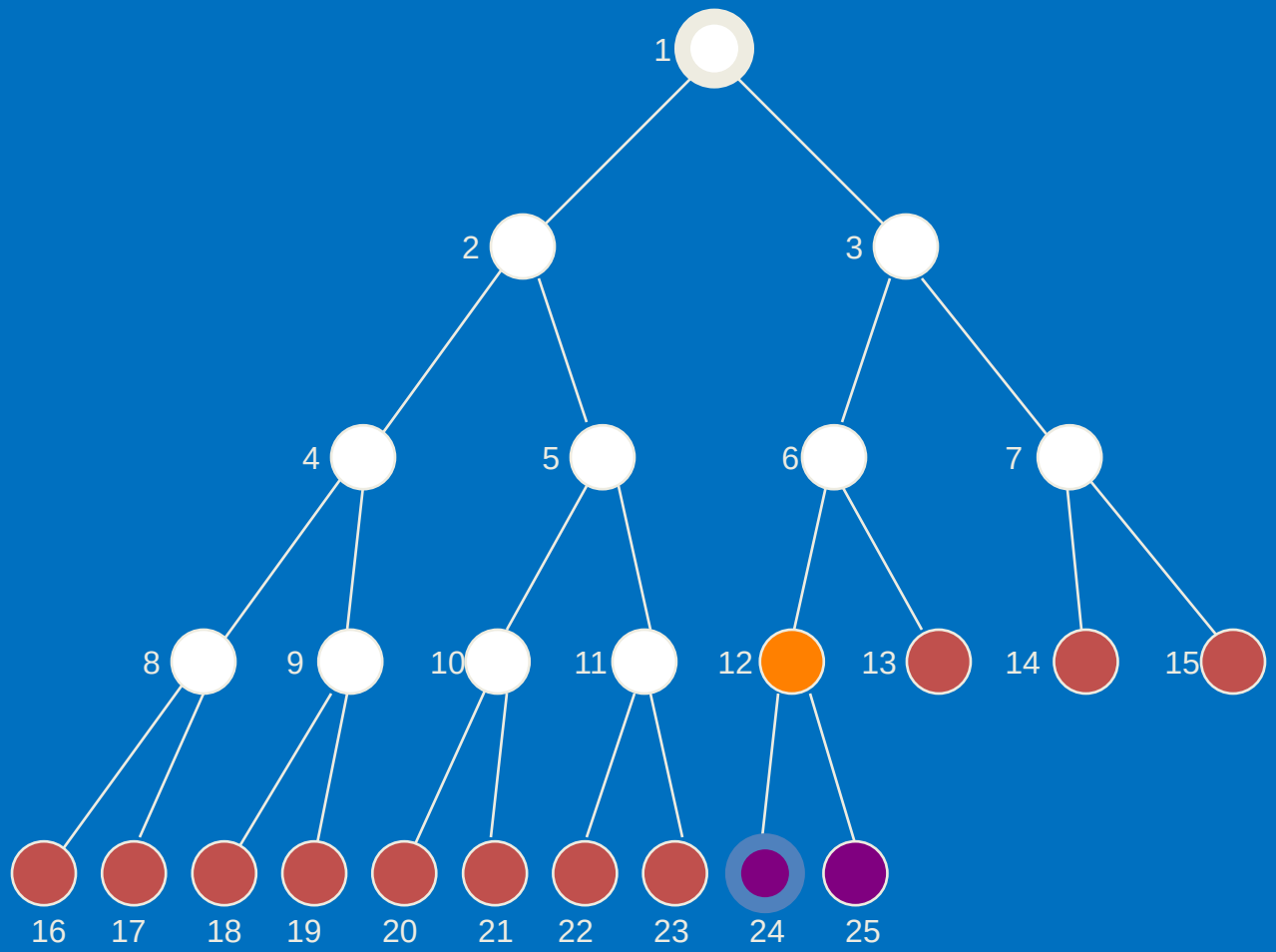
Fringe: [11,12,13,14,15,16,17,18,19] + [20,21]

Breadth-First Snapshot 11



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

Breadth-First Snapshot 12

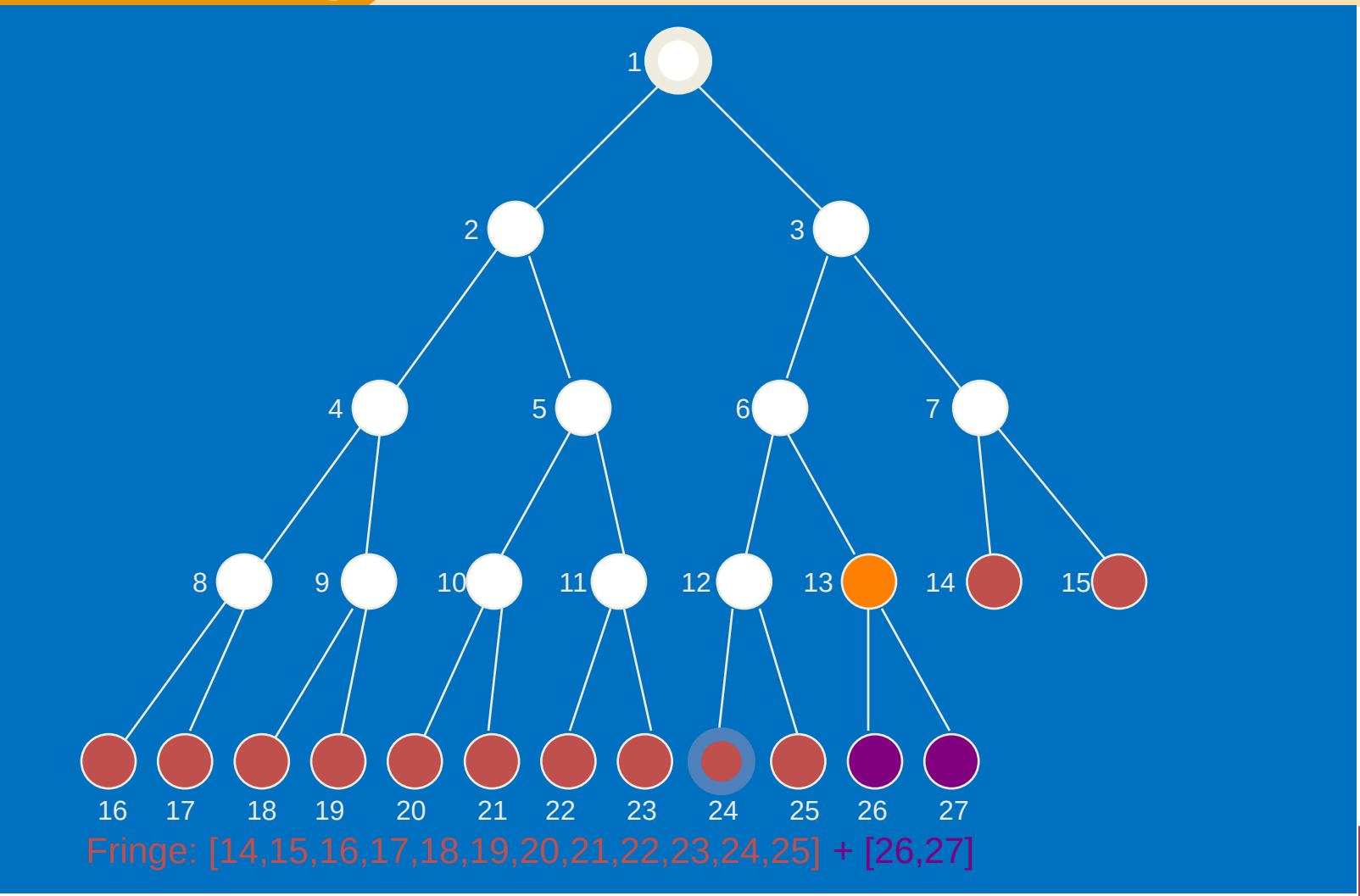


Fringe: [13,14,15,16,17,18,19,20,21] + [22,23]

Initial
Visited
Fringe
Current
Visible
Goal

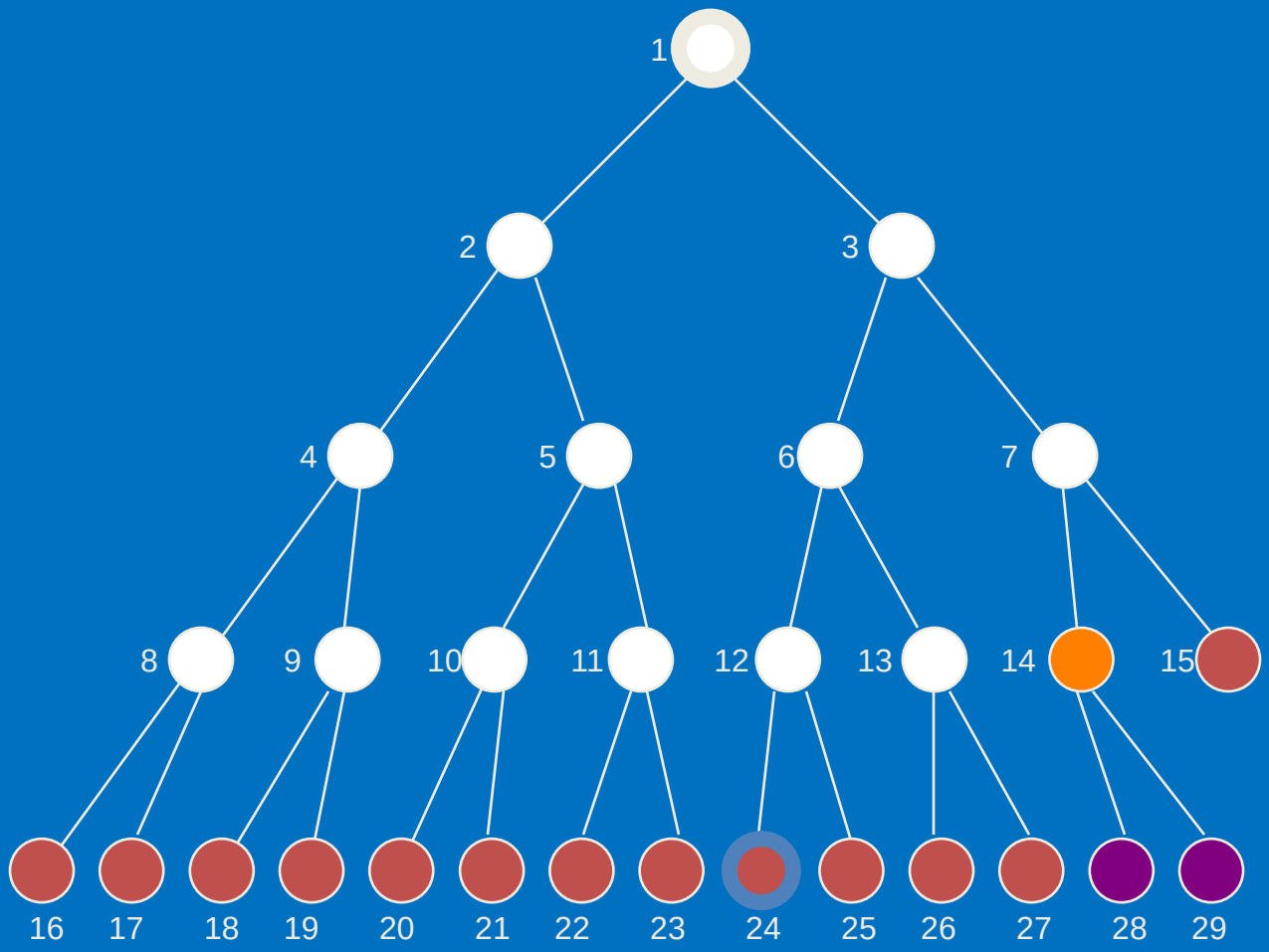
Note:
The goal node is “visible” here, but we can not perform the goal test yet.

Breadth-First Snapshot 13



Initial
Visited
Fringe
Current
Visible
Goal

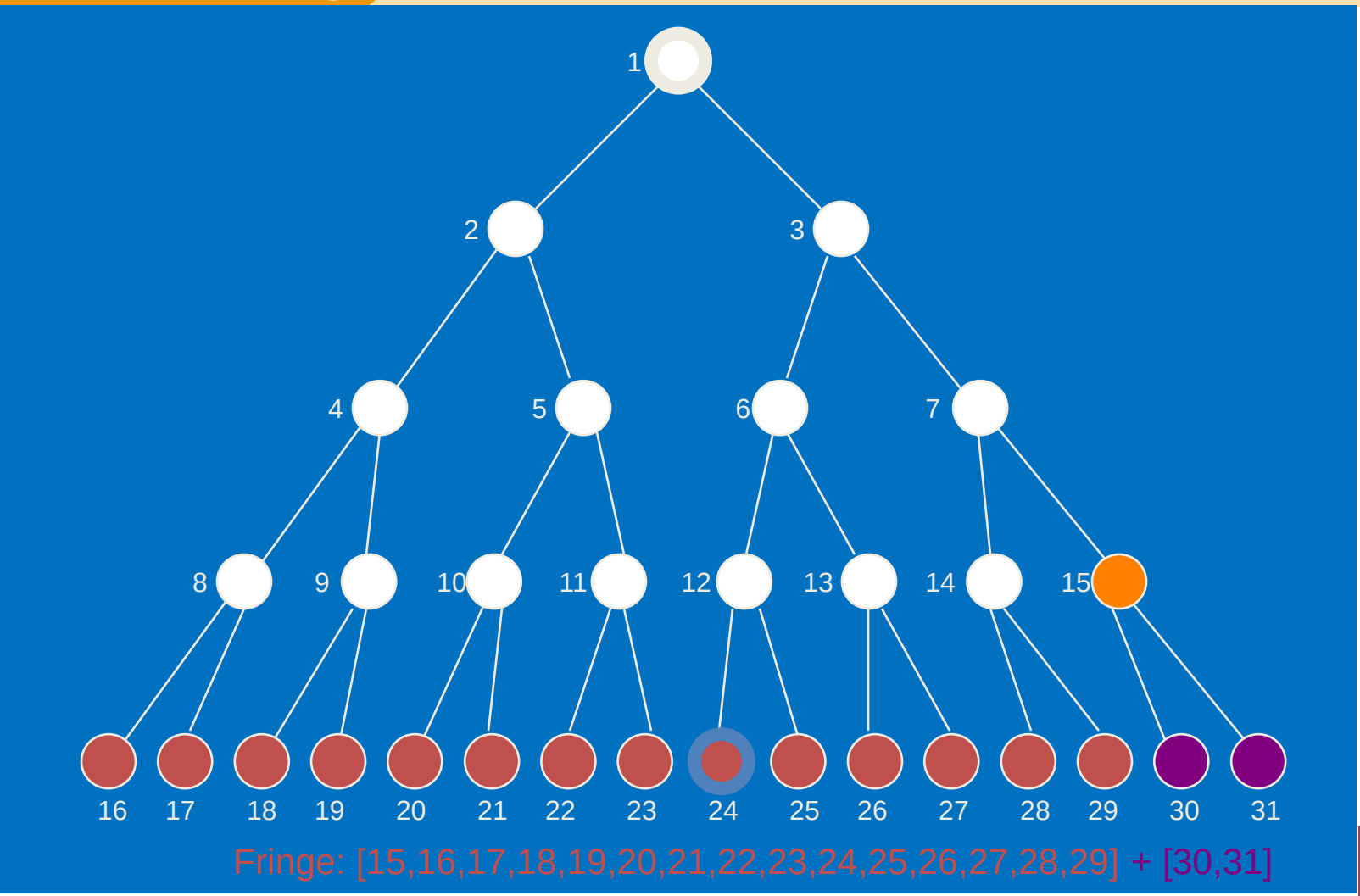
Breadth-First Snapshot 14



Initial
Visited
Fringe
Current
Visible
Goal

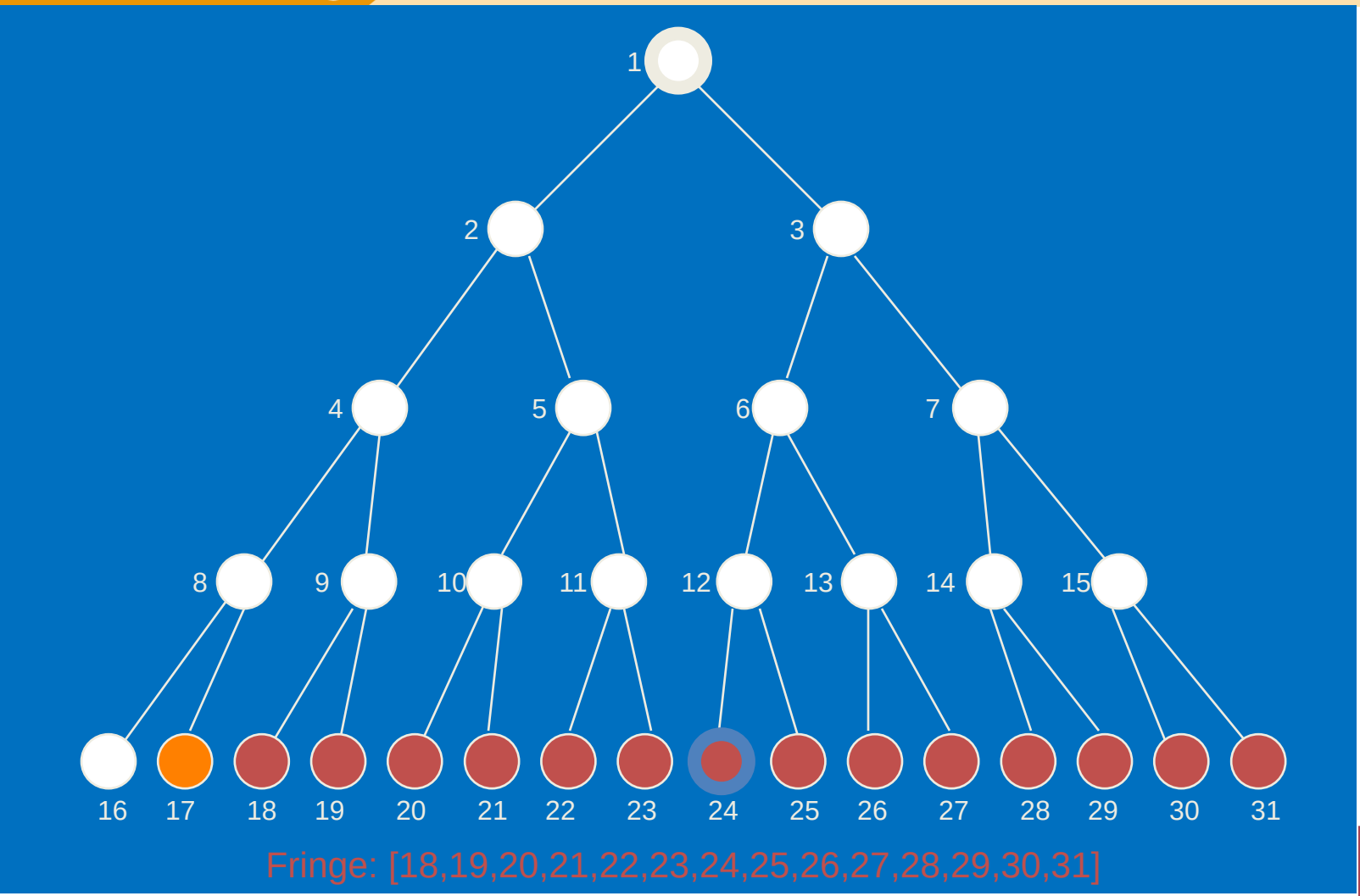
Fringe: [15,16,17,18,19,20,21,22,23,24,25,26,27] + [28,29]

Breadth-First Snapshot 15



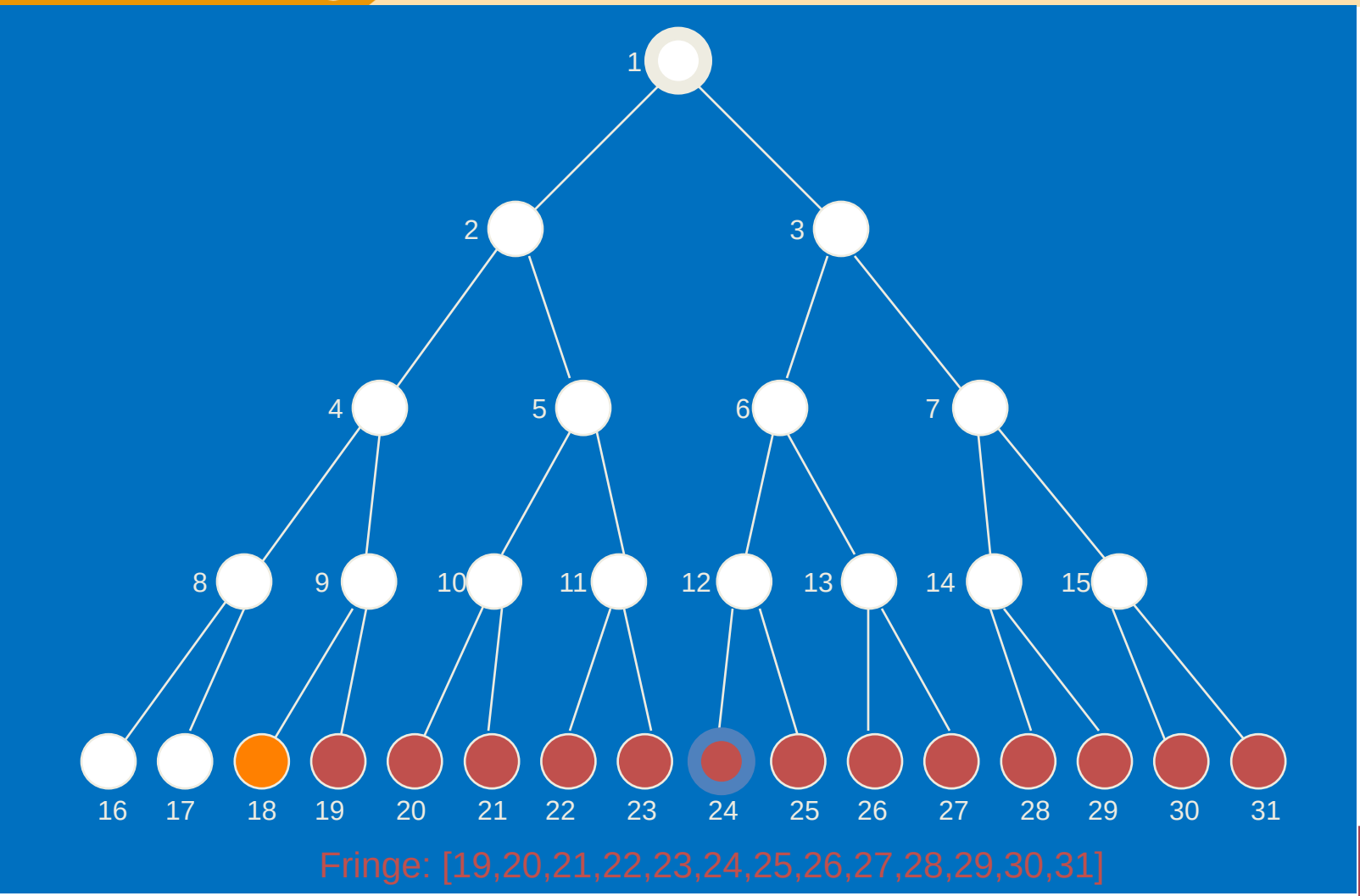
Initial
Visited
Fringe
Current
Visible
Goal

Breadth-First Snapshot 17



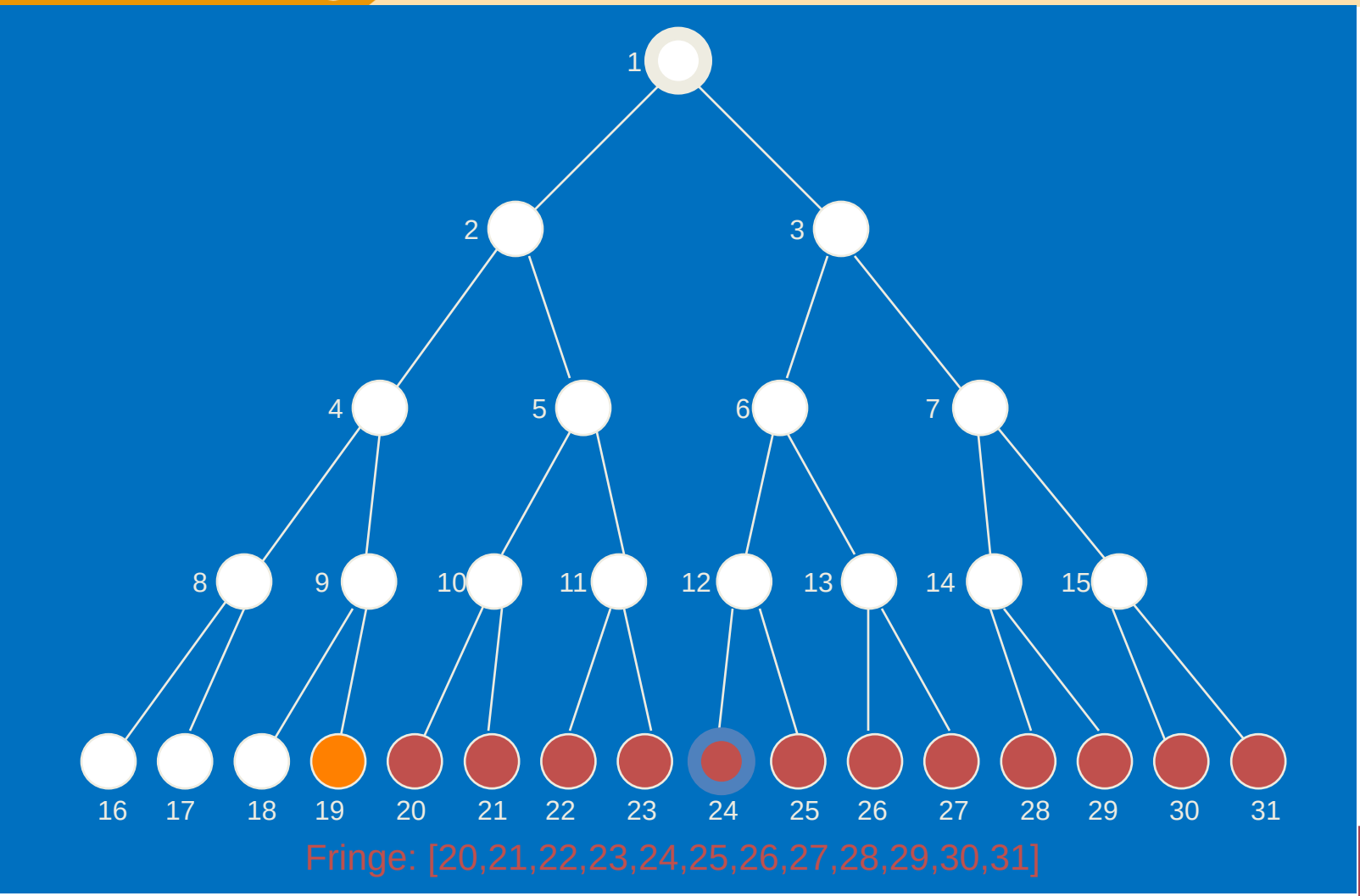
Initial
Visited
Fringe
Current
Visible
Goal

Breadth-First Snapshot 18



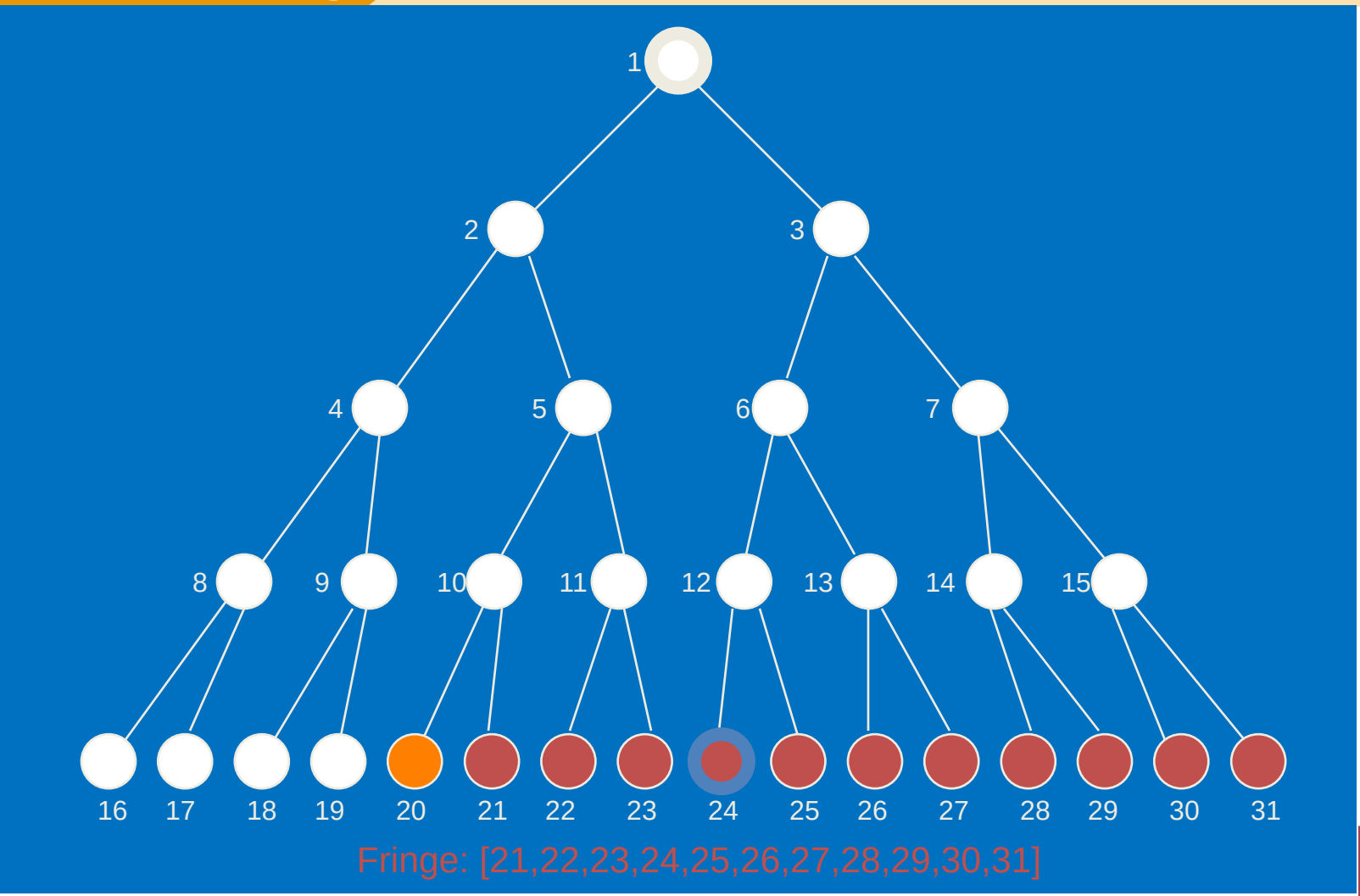
Initial
Visited
Fringe
Current
Visible
Goal

Breadth-First Snapshot 19



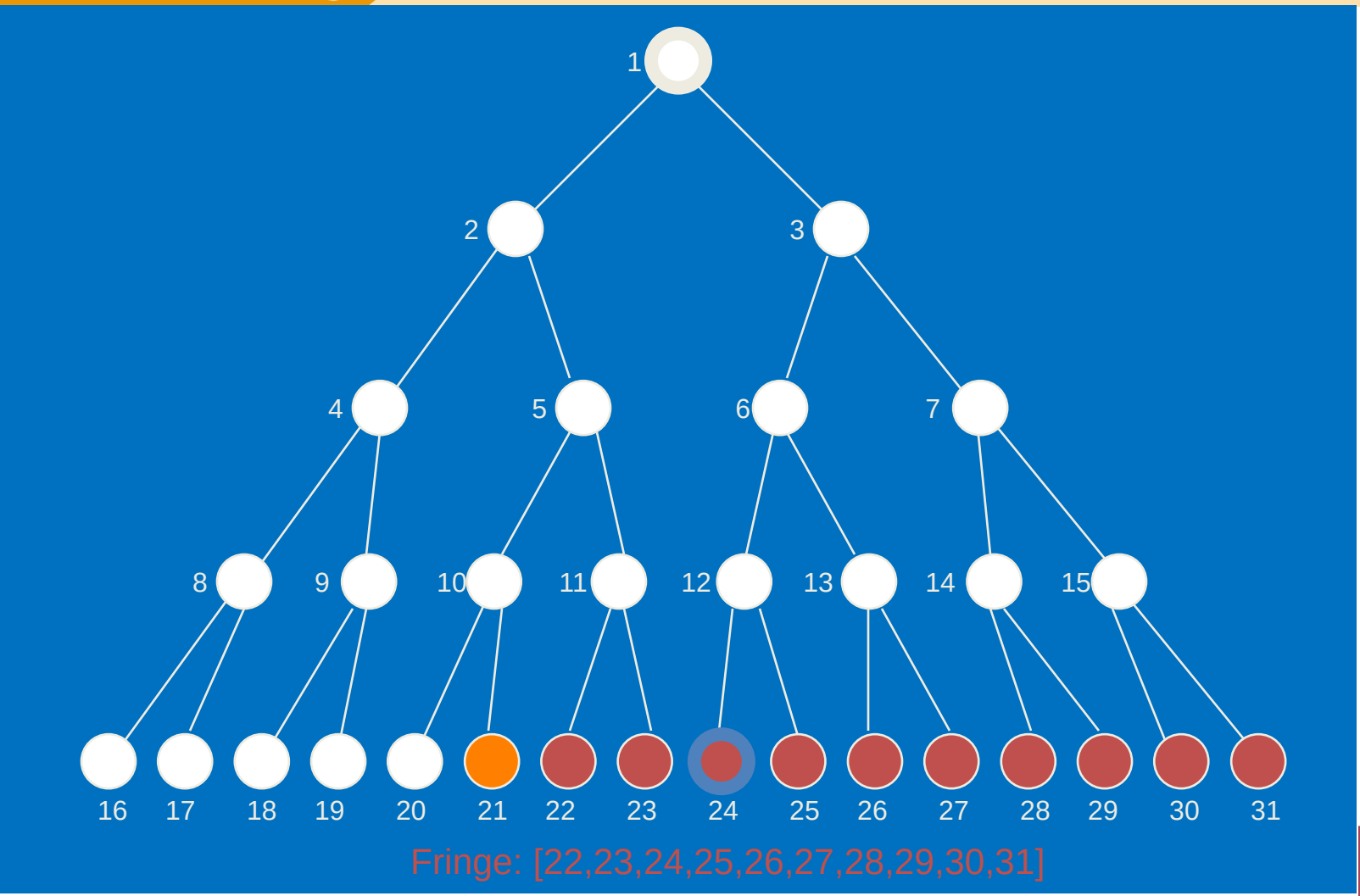
Initial
Visited
Fringe
Current
Visible
Goal

Breadth-First Snapshot 20



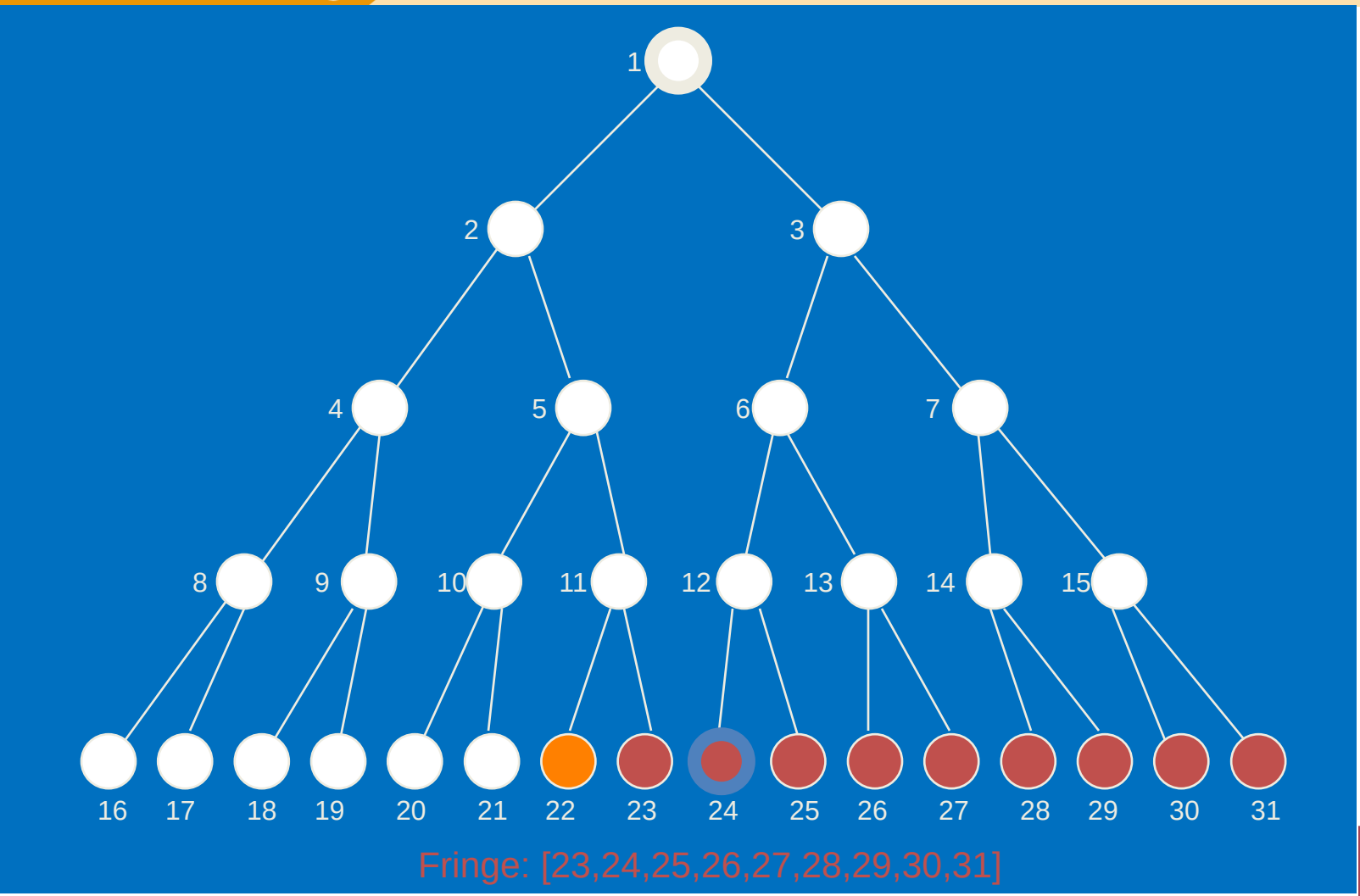
Initial
Visited
Fringe
Current
Visible
Goal

Breadth-First Snapshot 21



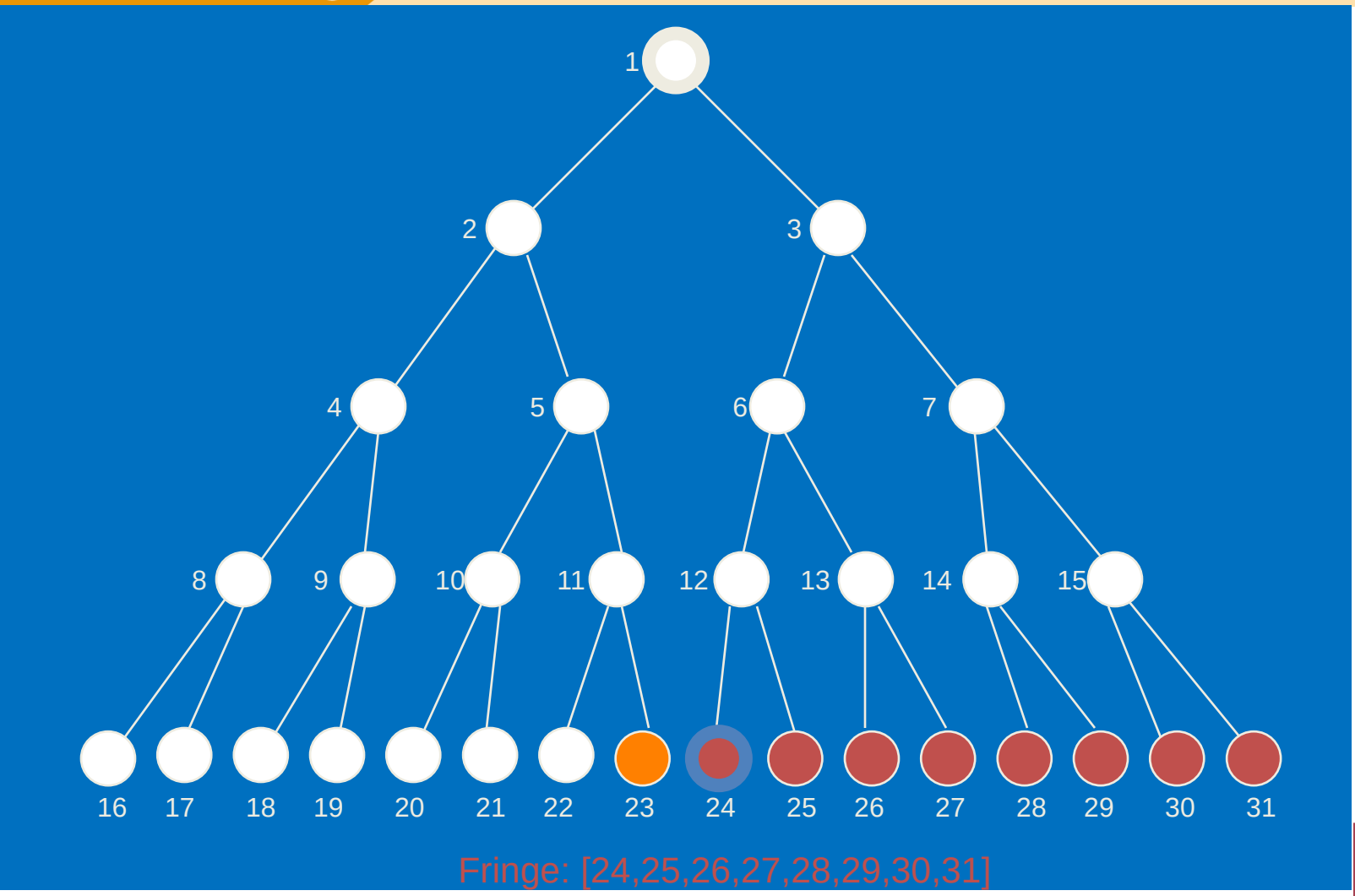
Initial
Visited
Fringe
Current
Visible
Goal

Breadth-First Snapshot 22



Initial
Visited
Fringe
Current
Visible
Goal

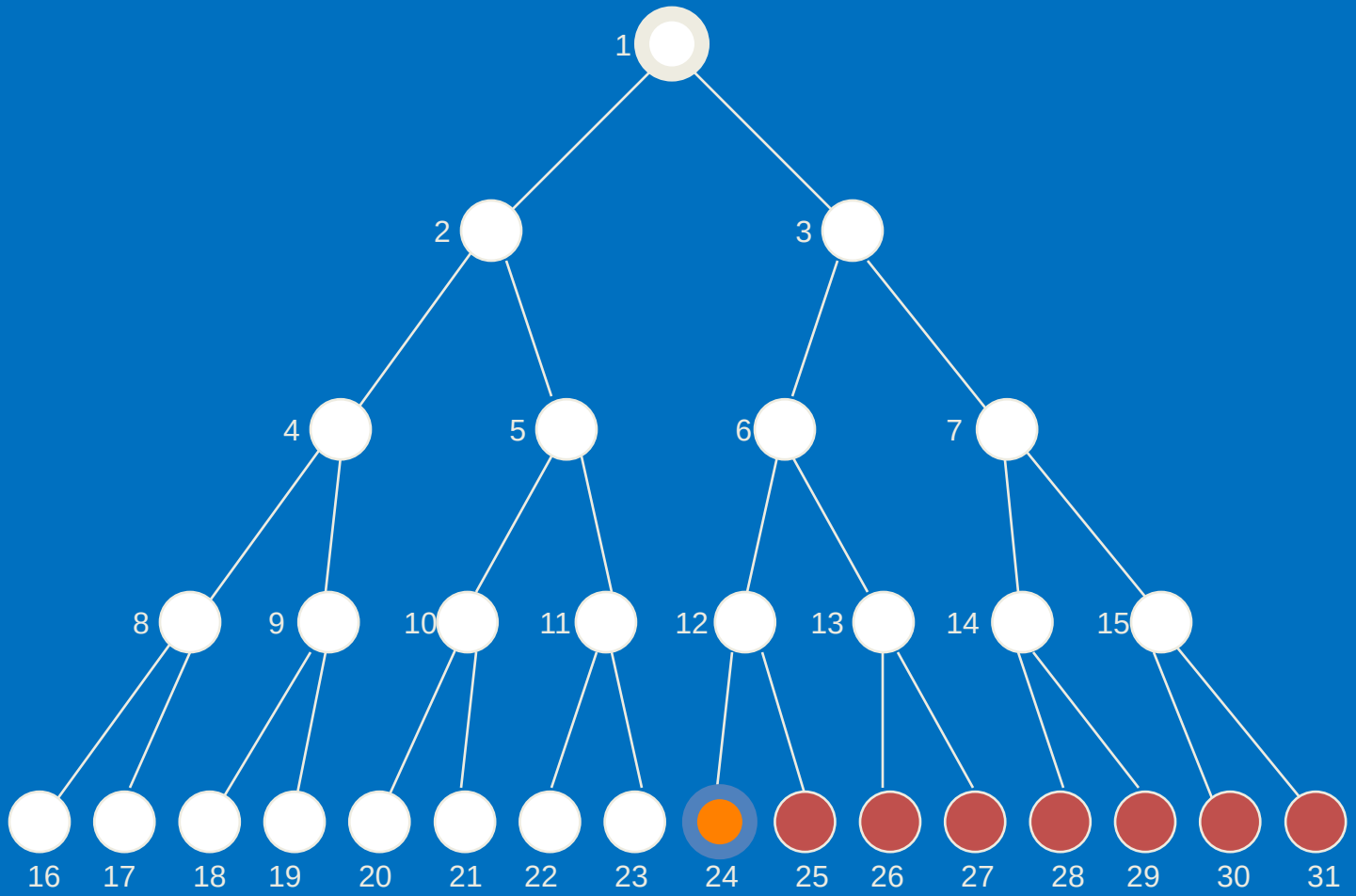
Breadth-First Snapshot 23



Initial
Visited
Fringe
Current
Visible
Goal

Fringe = nodes waiting in queue to be explored

Breadth-First Snapshot 24

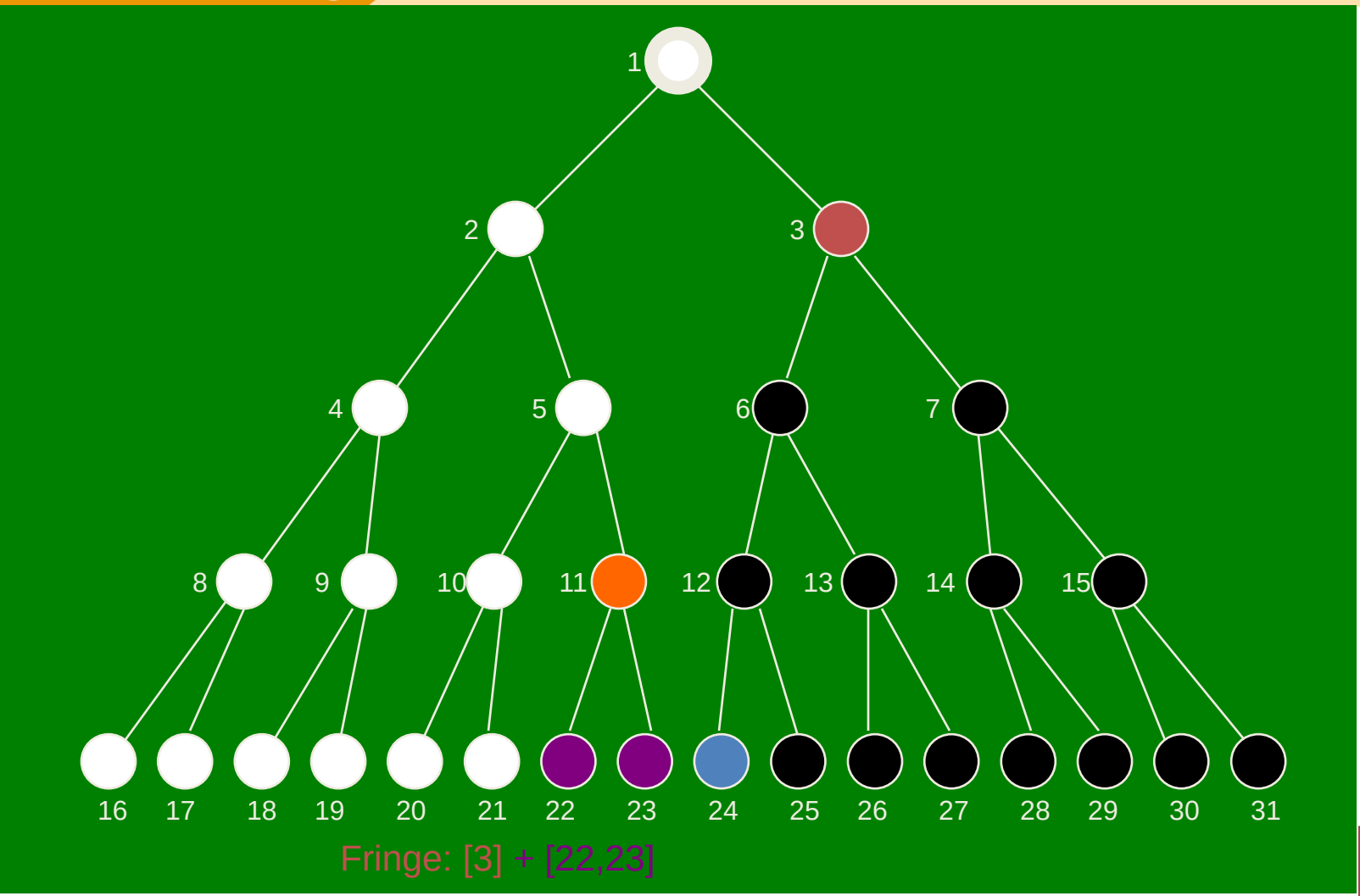


Fringe: [25,26,27,28,29,30,31]

Initial
Visited
Fringe
Current
Visible
Goal

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

Depth-First Snapshot



Initial
Visited
Fringe
Current
Visible
Goal

Fringe = nodes waiting in queue to be explored

Activity 1: 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

Question 1: Give reasons why Depth First search uses less space that breadth first?

- Breadth-first is more careful by checking all alternatives
  - Complete and optimal
    - Under most circumstances
  - But ..Very memory-intensive !!

Question 2: Give a reason why Breadth First search uses very memory-intensive?

Revision (5): Exhaustive Search

The following state space graph below (Figure 1) represents the gas piping network. The node in the graph denotes the gas gate while the arc denotes the pipe that connects the entrance to the gate. Based on this scenario, answer the following questions:

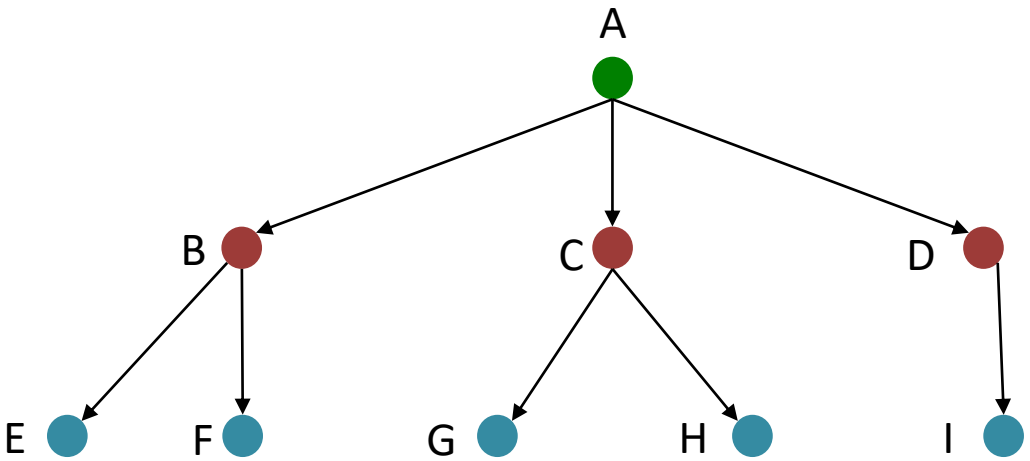


Figure 1. Gas piping network

1. Using graph theory, identify the suitable node/nodes for each description in Table below.

Description	Node (Gas gate)
Start node	A
The child node of A	
The parent node of G. H	
The child node of B	
The parent node of I	

Revision (5): Exhaustive Search

2. Based on Figure 1, let node A is a start node, and the goal node is E. Perform a **breadth-first search**, then list down the order of nodes to be visited (OPEN and CLOSED list) from the starting node to the goal node in Table below.

Iteration	OPEN	CLOSED
0		
1		
2		

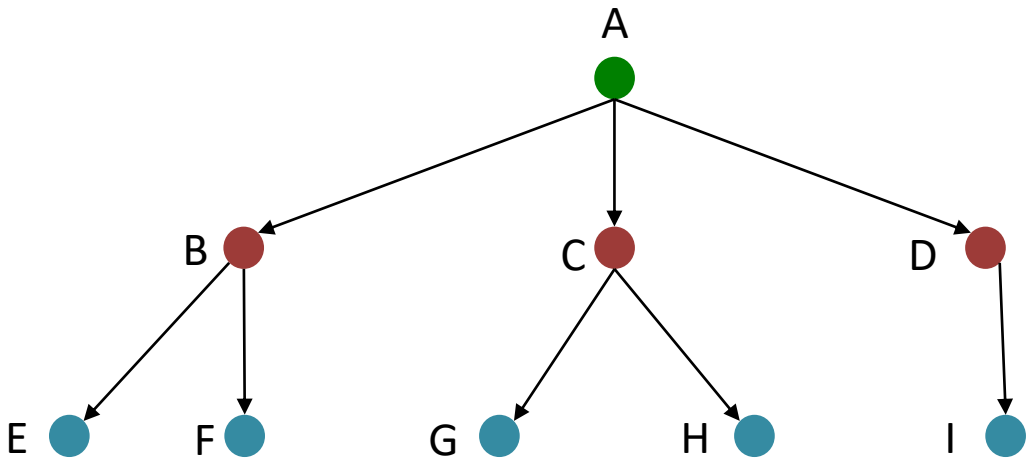


Figure 1. Gas piping network

3. Produce the sequence of nodes for the gas can be flowed from gate A to gate E.

Revision (5): Exhaustive Search

2. Based on Figure 1, let node A is a start node, and the goal node is E. Perform a breadth-first search, then list down the order of nodes to be visited (OPEN and CLOSED list) from the starting node to the goal node in Table below.

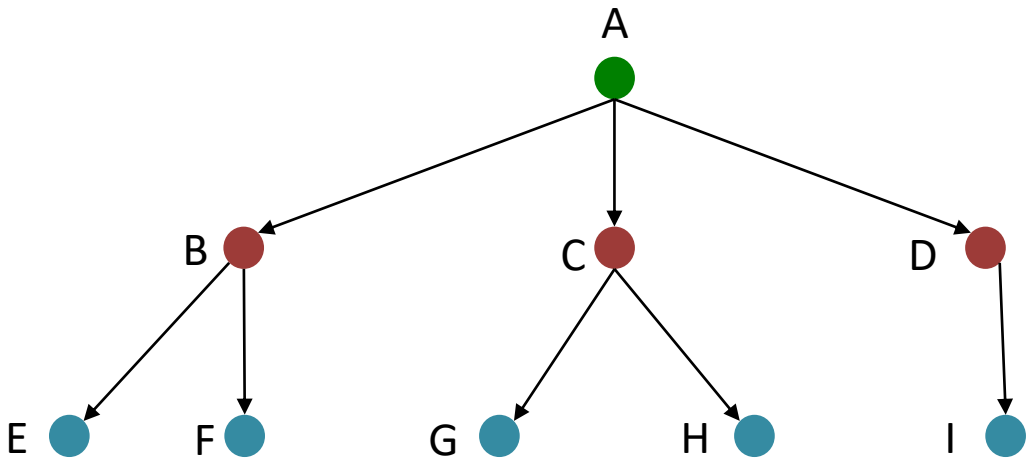


Figure 1. Gas piping network

Iteration	OPEN	CLOSED
0	A	[]
1	B,C,D	A
2	C,D,E,F	B,A
3	D,E,F,G,H	C,B,A
4	E,F,G,H,I	D,C,B,A
5	E is found	

Revision (5): Exhaustive Search

1. Based on Figure 1, let node A is a start node, and the goal node is E. Perform a **depth-first search**, then list down the order of nodes to be visited (OPEN and CLOSED list) from the starting node to the goal node in Table below.

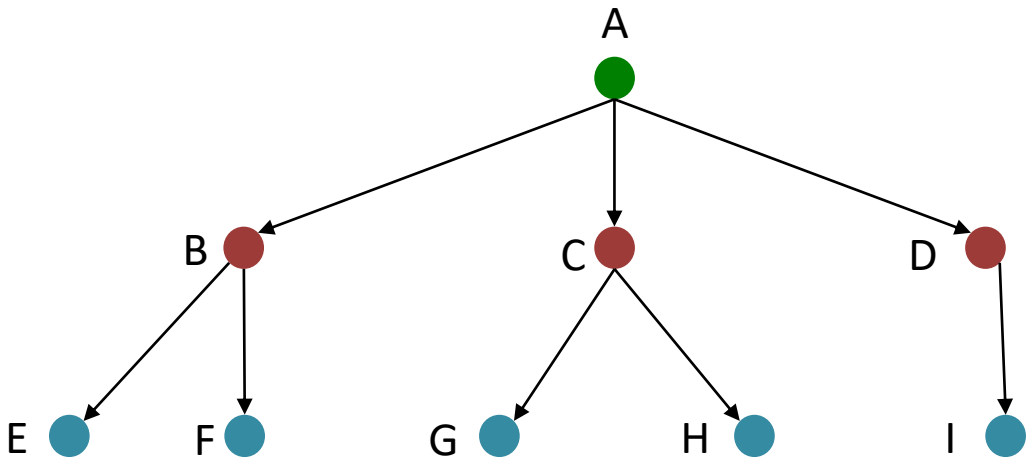


Figure 1. Gas piping network

Iteration	OPEN	CLOSED
0		
1		
2		
3		
4		
5		

2. Produce the sequence of nodes for the gas can be flowed from gate A to gate E.

Revision (5): Exhaustive Search

3. Based on Figure 1, let node A is a start node, and the goal node is E. Perform a depth-first search, then list down the order of nodes to be visited (OPEN and CLOSED list) from the starting node to the goal node in Table below.

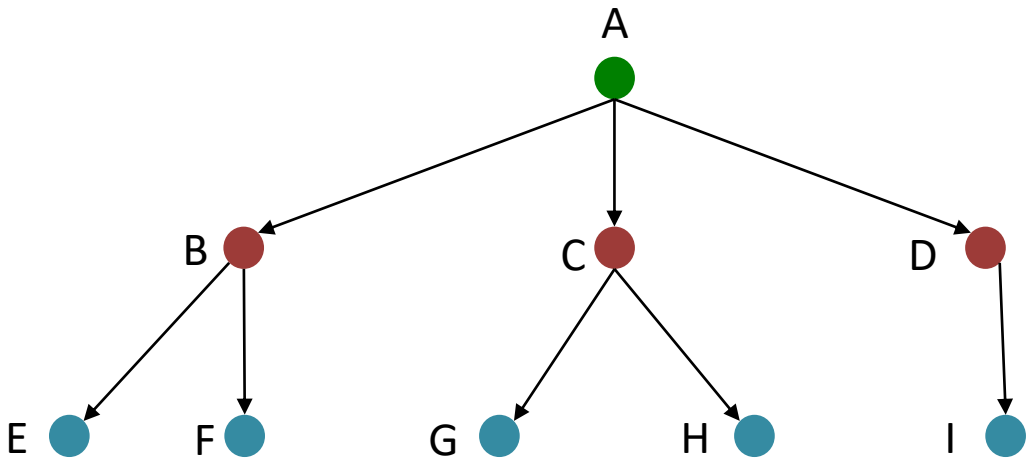


Figure 1. Gas piping network

Iteration	OPEN	CLOSED
0	A	[]
1	B,C,D	A
2	E,F,C,D	B,A
3	E is found	
4		
5		

Using State Space to Represent Reasoning

- And/or graph
- Hypergraph

Using State Space to Represent Reasoning : Predicate calculus

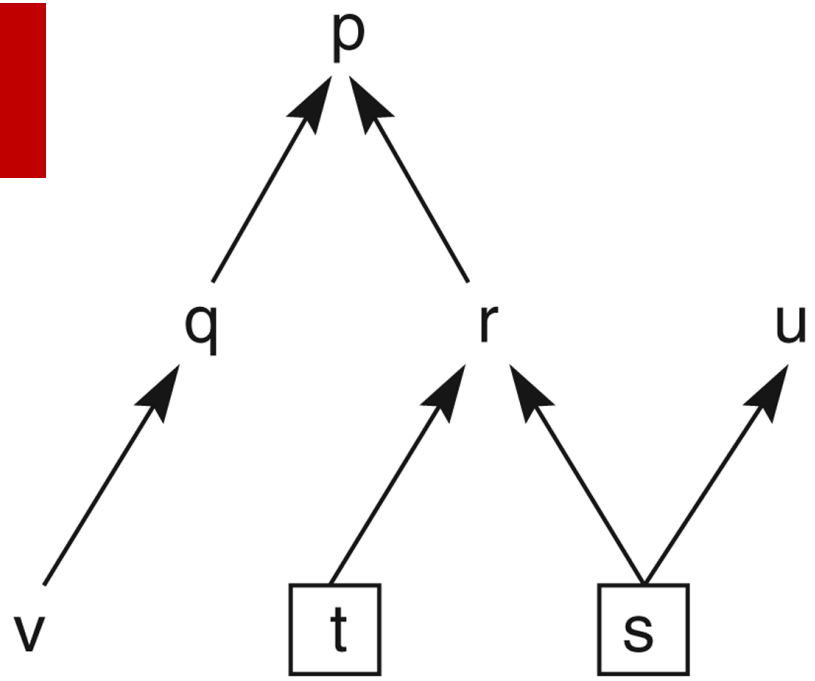
- Predicate calculus – map nodes/some states of a graph onto state space
- Inference rules – describe the arc between states
- Eg. Is expression a logical consequence of given assertion? Use search to solve
- Guarantee correctness of conclusion derived- a formal proof of integrity of solution

Draw a graph: Propositional calculus

Example: Define a graph from propositional calculus to view the logical relationships.

$q \rightarrow p$
$r \rightarrow p$
$v \rightarrow q$
$s \rightarrow r$
$t \rightarrow r$
$s \rightarrow u$
s
t

Translate into graph?



- State space graph of a set of implications in the propositional calculus.
- Reasoning:- How to infer *p* using moden ponens?

s , s -->r yields r

r , r -->p yields p

Activity 2: From propositional calculus to a graph

Give a set of propositional calculus expressions, you are required to draw a graph.

a

b

c

a → e

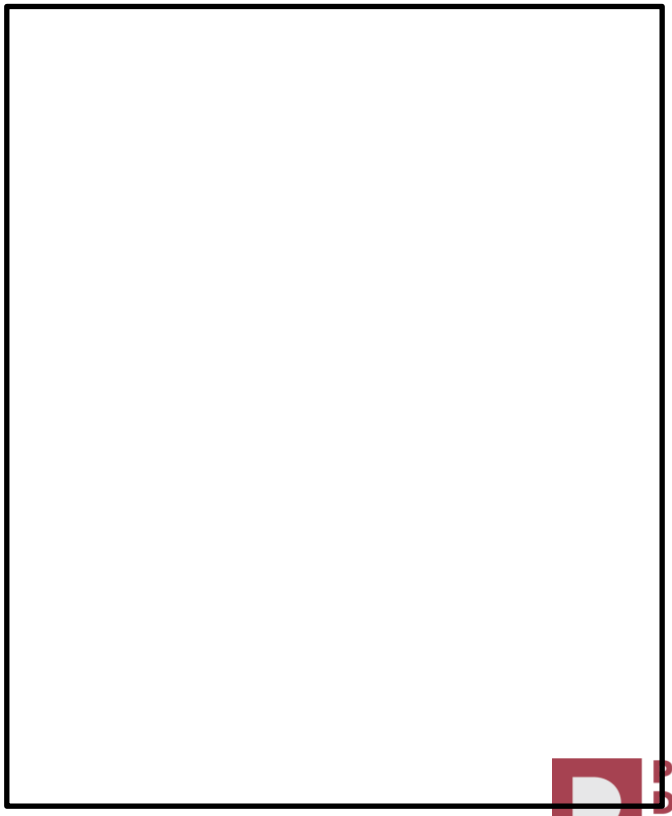
c → e

b → f

c → f

f → g

Draw graph



Activity 2: From propositional calculus to a graph

Give a set of propositional calculus expressions, you are required to draw a graph.

a

b

c

a → e

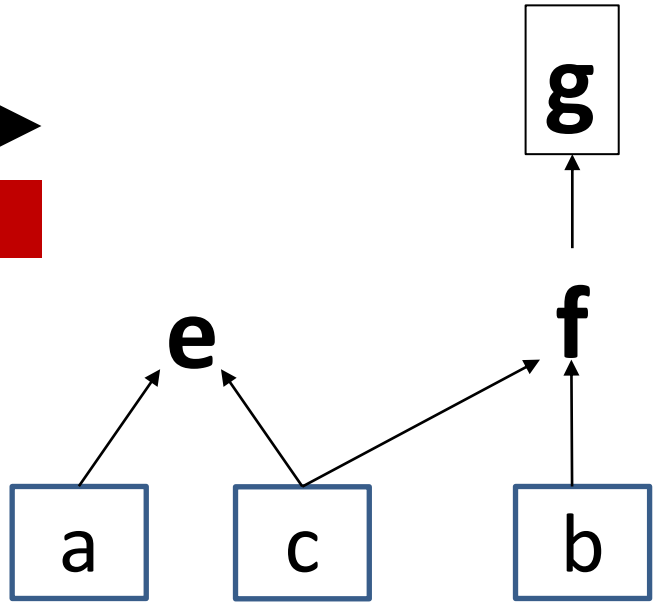
c → e

b → f

c → f

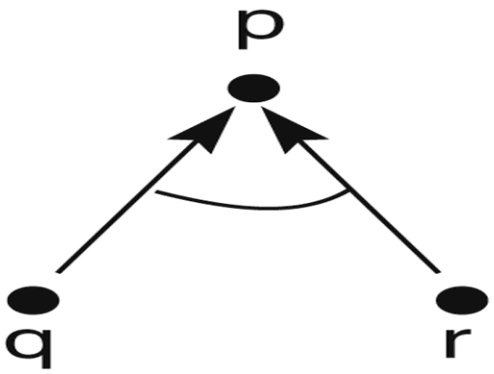
f → g

Draw graph

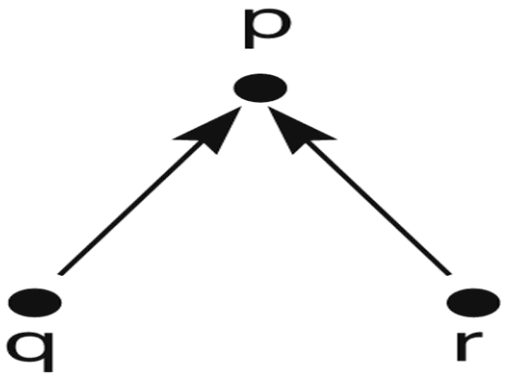


And/or graph

- To represent logical operators AND and OR in predicate calculus
- extension of basic graph model known as and/or graph
- To differ the relationships graphically, curve link indicates operator AND.



$q \wedge r \Rightarrow p$



$q \vee r \Rightarrow p$

Hypergraph

AND/OR graph is a specialization of a type of graph known as *hypergraph*, which connects nodes by sets of arcs rather than a single arcs

DEFINITION

HYPERGRAPH

A hypergraph consists of:

N, a set of nodes.

H, a set of hyperarcs defined by ordered pairs in which the first element of the pair is a single node from **N** and the second element is a subset of **N**.

An ordinary graph is a special case of hypergraph in which all the sets of descendant nodes have a cardinality of 1.

Hypergraph

eg. And/or graph of a set of propositional calculus expressions.

a

b

c

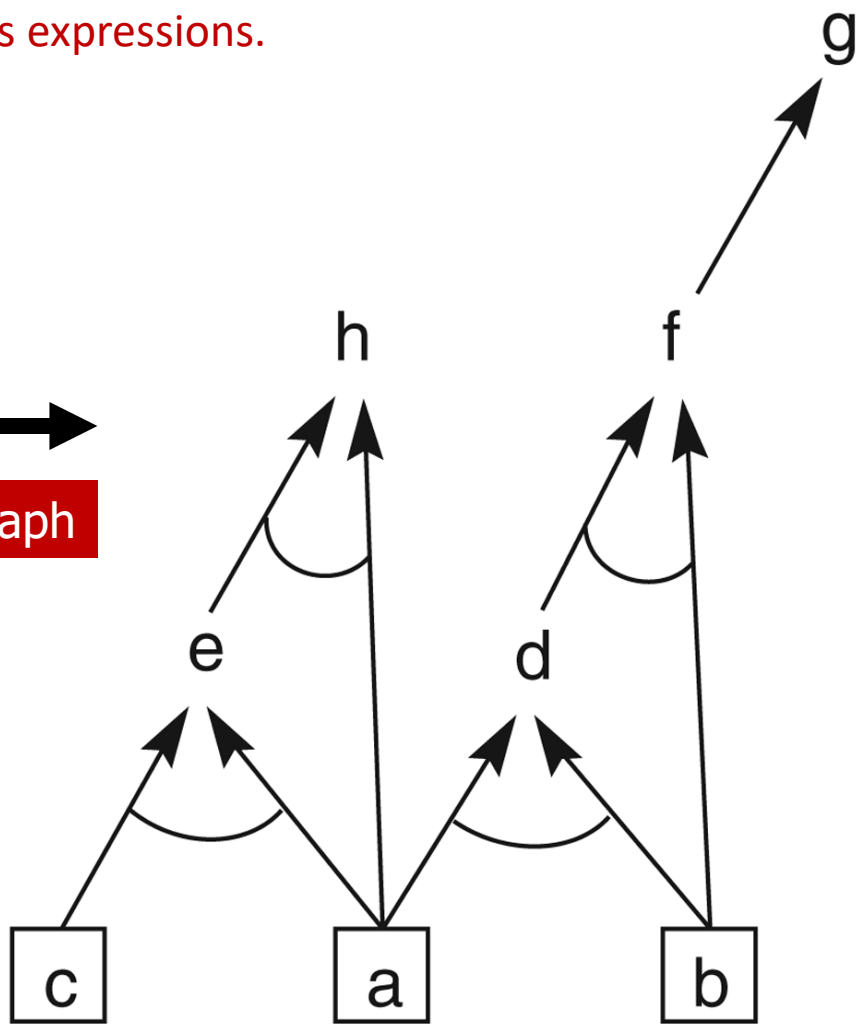
$a \wedge b \rightarrow d$

$a \wedge c \rightarrow e$

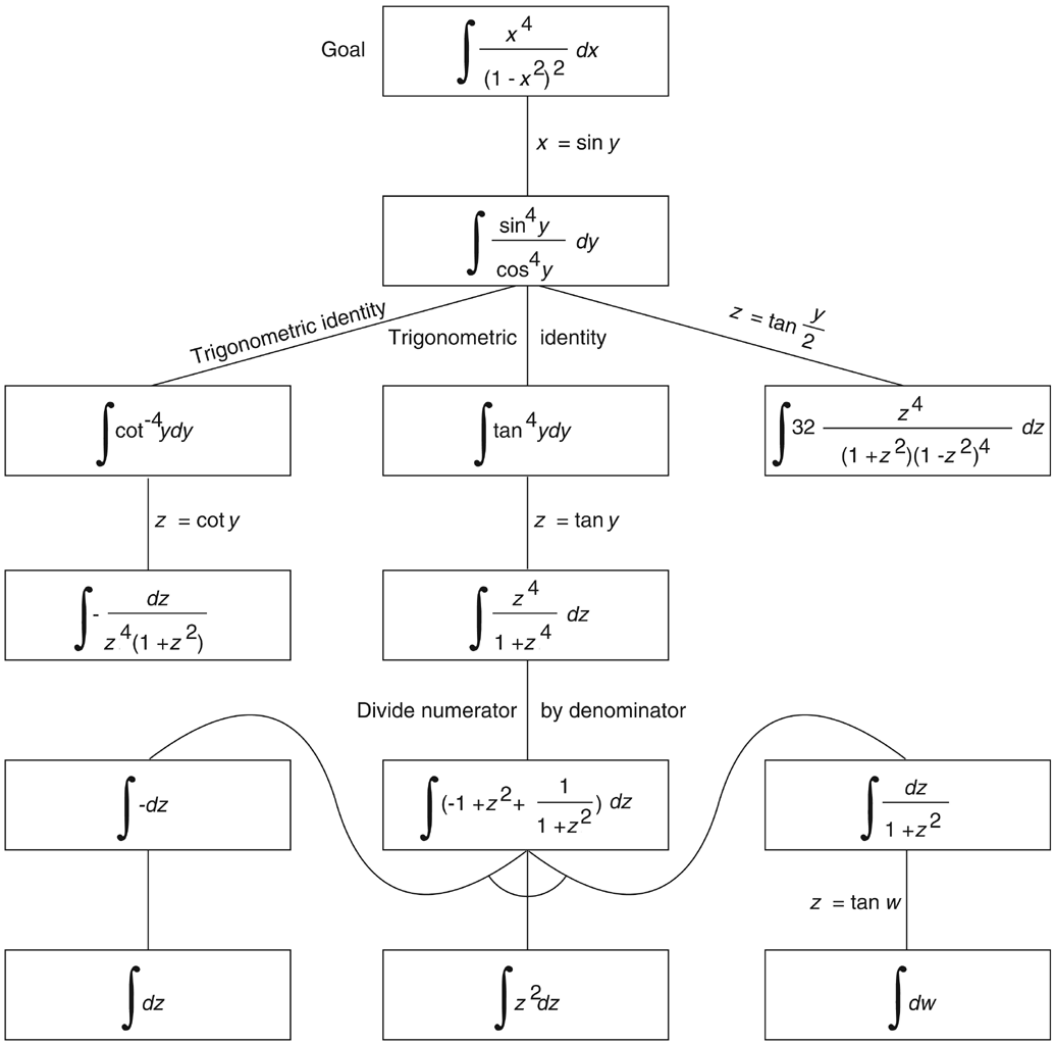
$b \wedge d \rightarrow f$

$f \rightarrow g$

Draw the and/or graph



Hypergraph



More example:
And/or graph of part of the state space for integrating a function, from Nilsson (1971).

Goal-directed search

Hypergraph

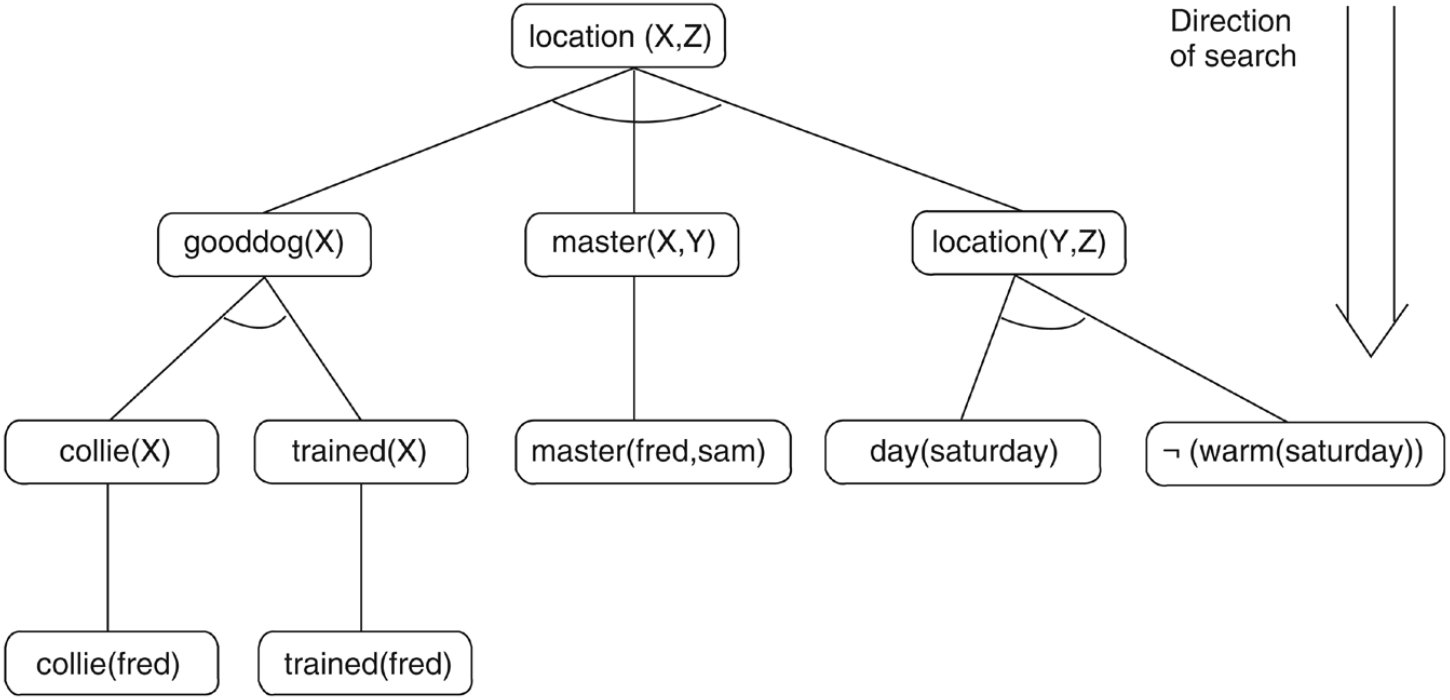
1. Fred is a collie.
   collie(fred).
2. Sam is Fred's master.
   master(fred,sam).
3. The day is Saturday.
   day(saturday).
4. It is cold on Saturday.
   ¬ (warm(saturday)).
5. Fred is trained.
   trained(fred).
6. Spaniels are good dogs and so are trained collies.
   ∀ X[spaniel(X) ∨ (collie(X) ∧ trained(X)) → gooddog(X)]
7. If a dog is a good dog and has a master then he will be with his master.
   ∀ (X,Y,Z) [gooddog(X) ∧ master(X,Y) ∧ location(Y,Z) → location(X,Z)]
8. If it is Saturday and warm, then Sam is at the park.
   (day(saturday) ∧ warm(saturday)) → location(sam,park).
9. If it is Saturday and not warm, then Sam is at the museum.
   (day(saturday) ∧ ¬ (warm(saturday))) → location(sam,museum).

More example:

- Where is Fred??
- The facts and rules of this example are given as English sentences followed by their predicate calculus equivalents:

Hypergraph

Where is Fred???..The solution subgraph showing that fred is ???



at the museum

Substitutions = {fred/X, sam/Y, museum/Z}

Goal-driven search using predicate calculus