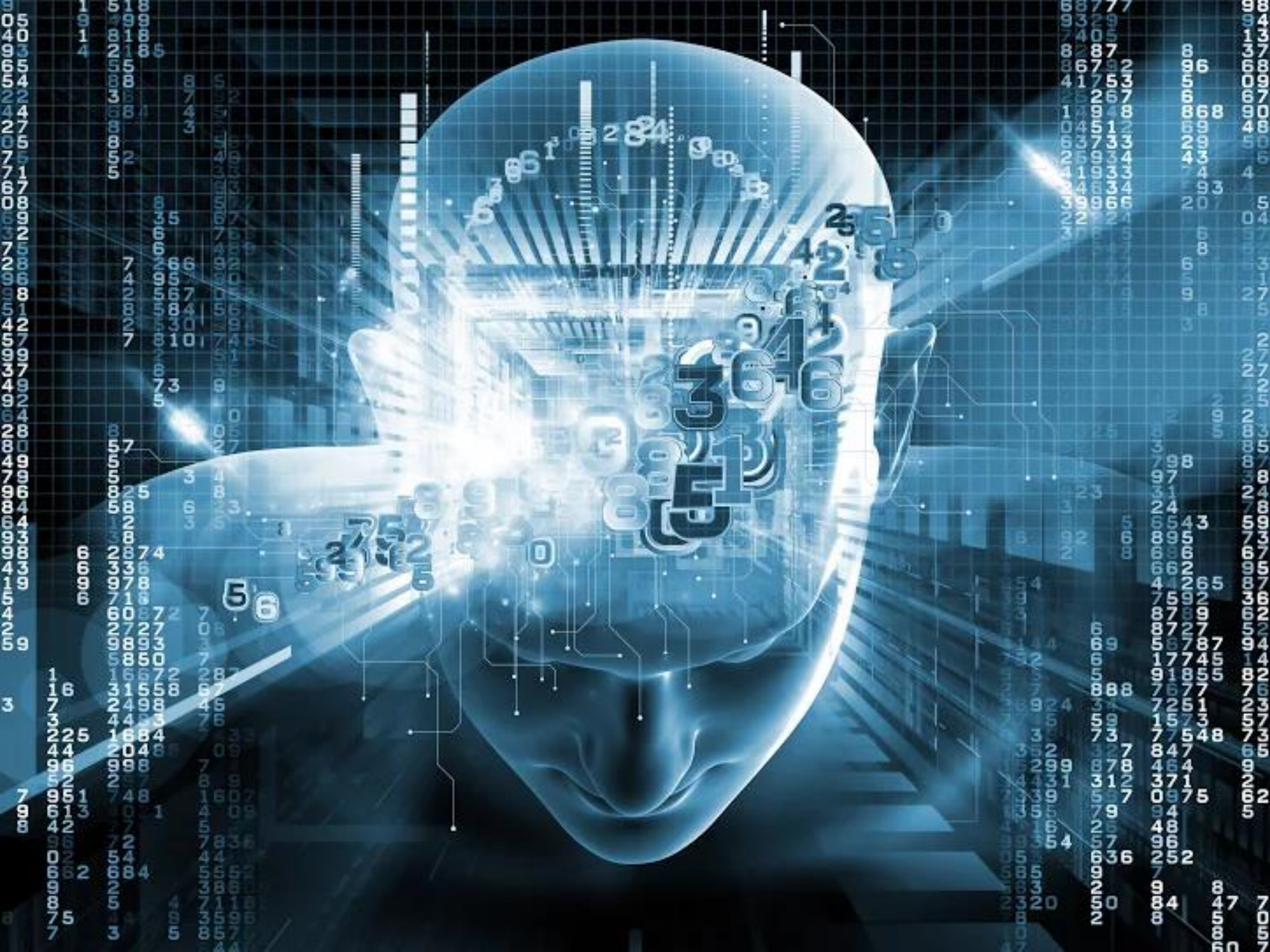
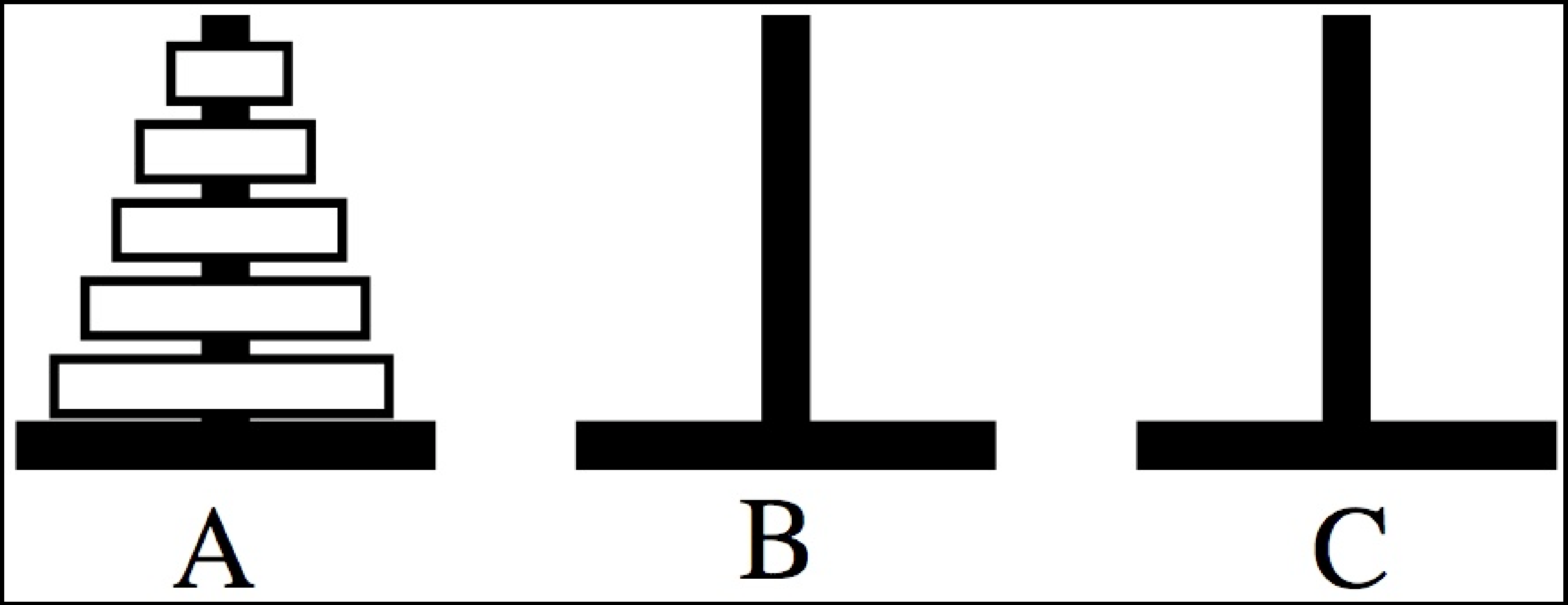
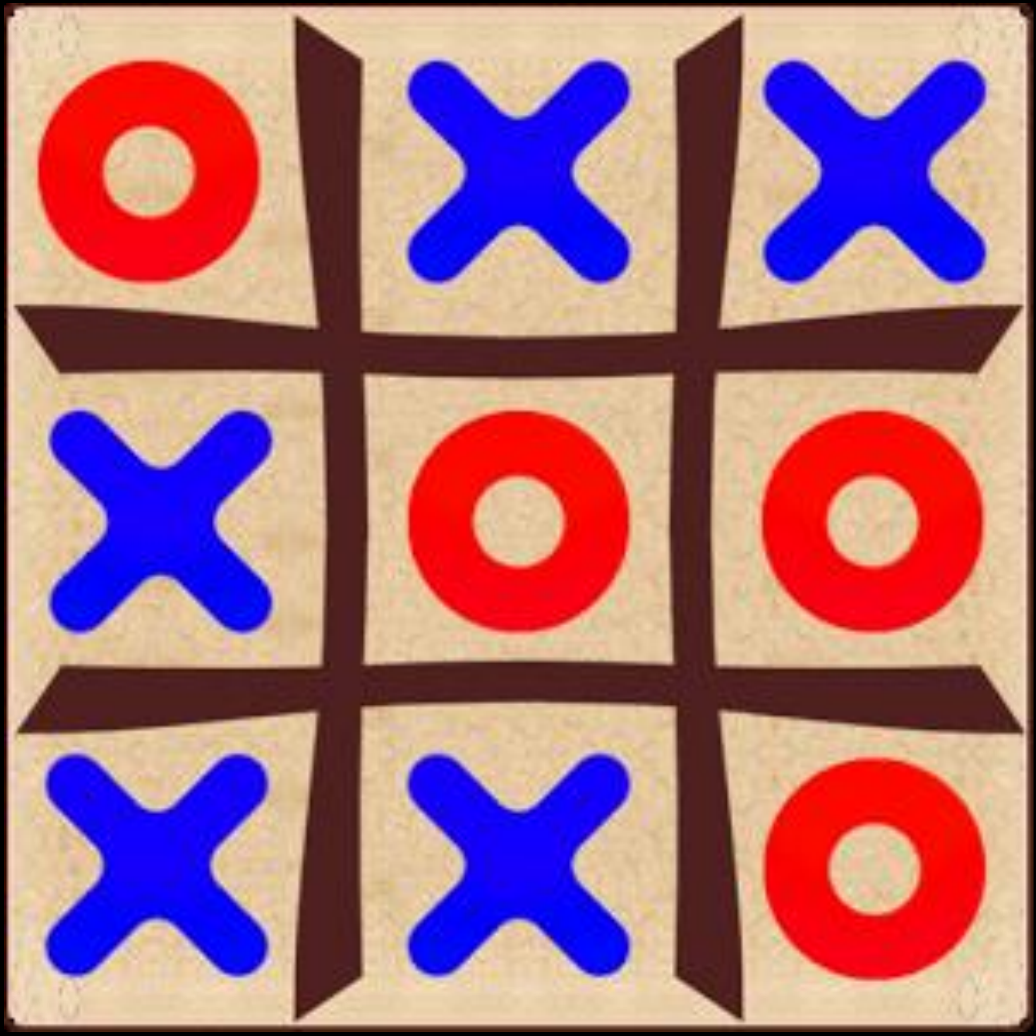
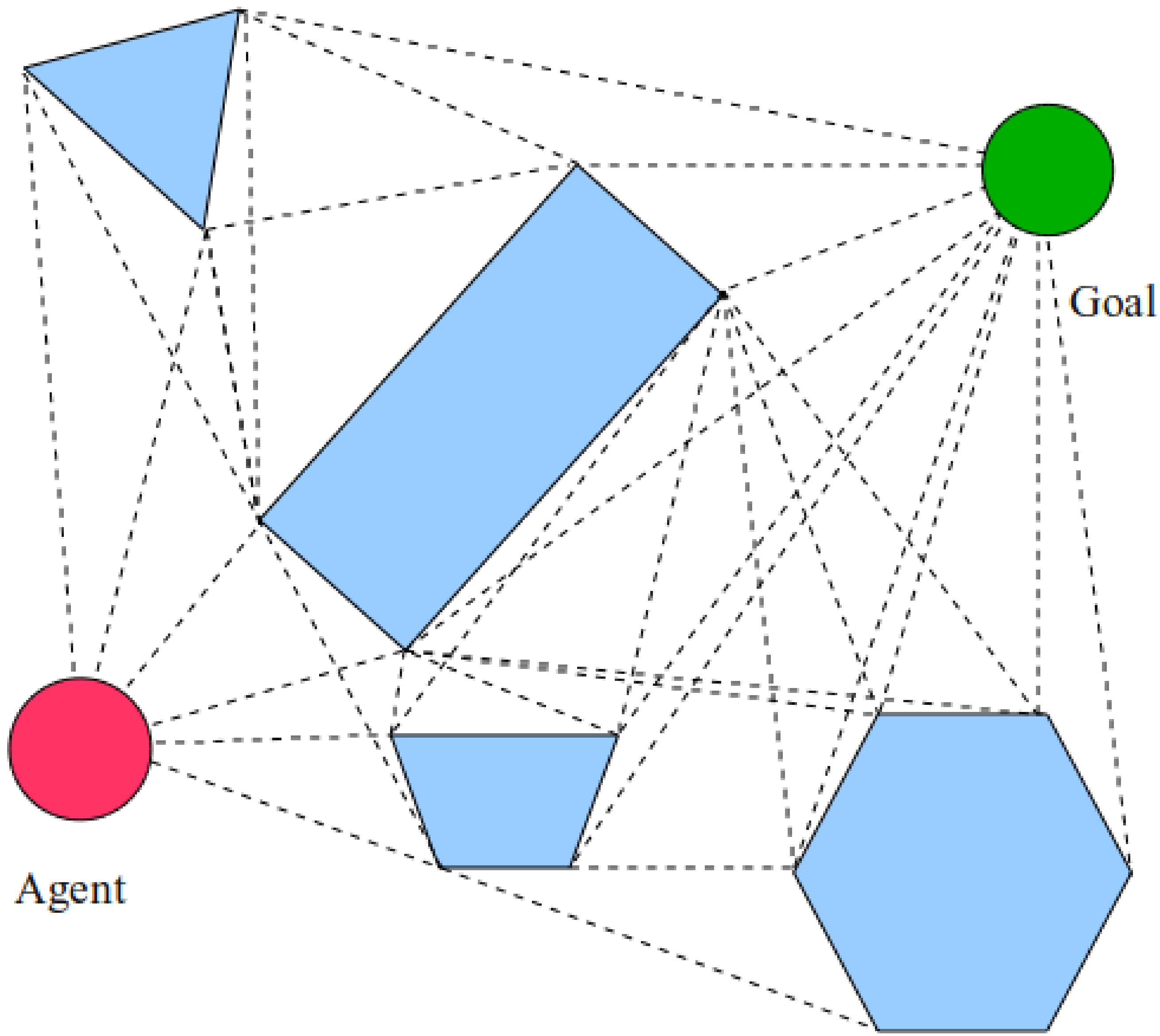
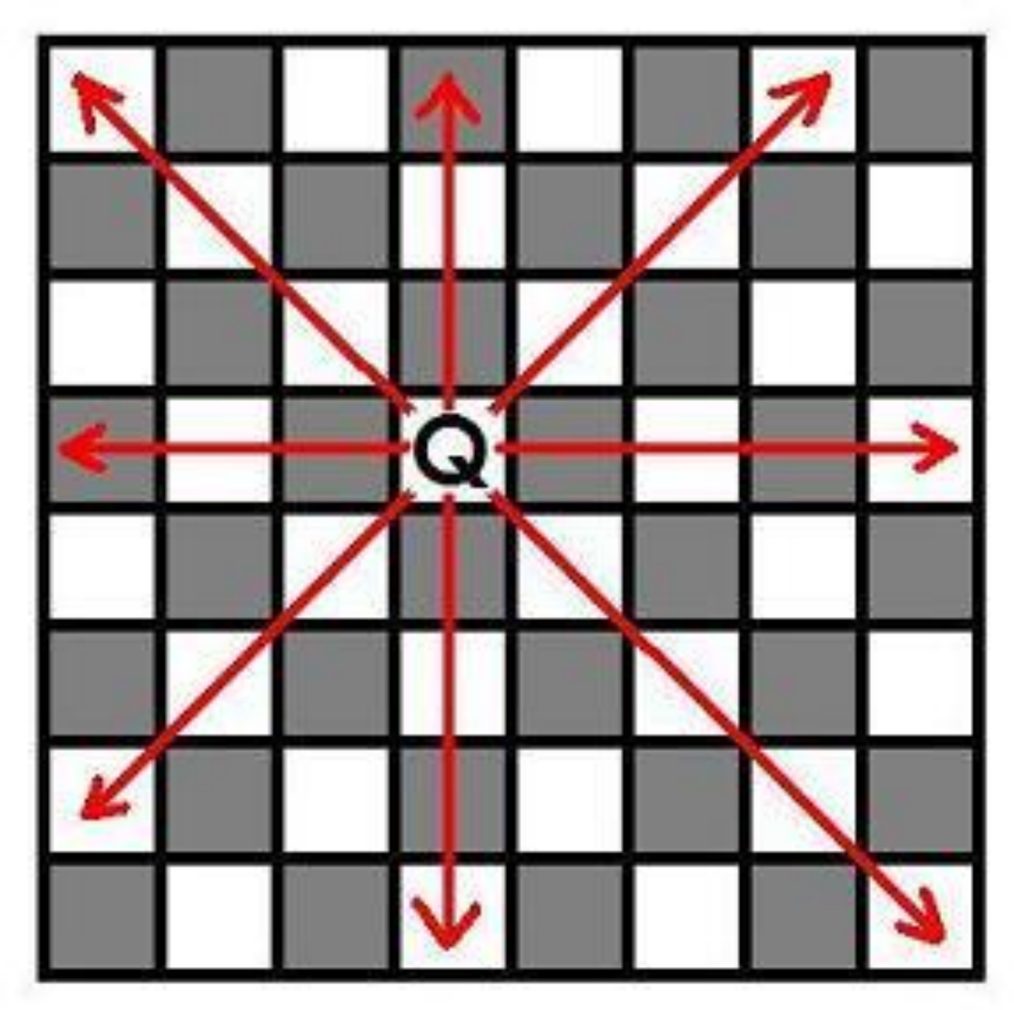


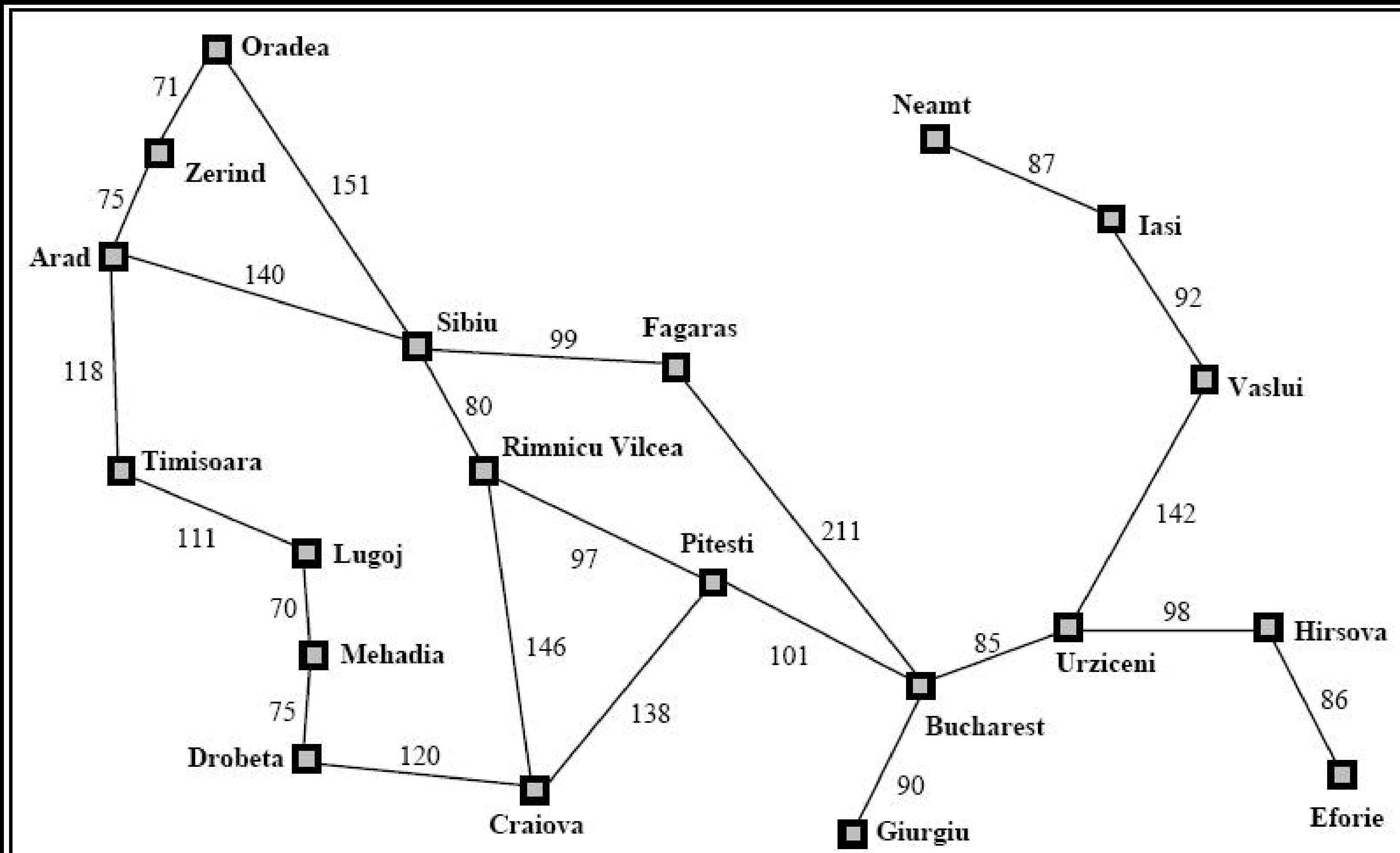

A
B
C



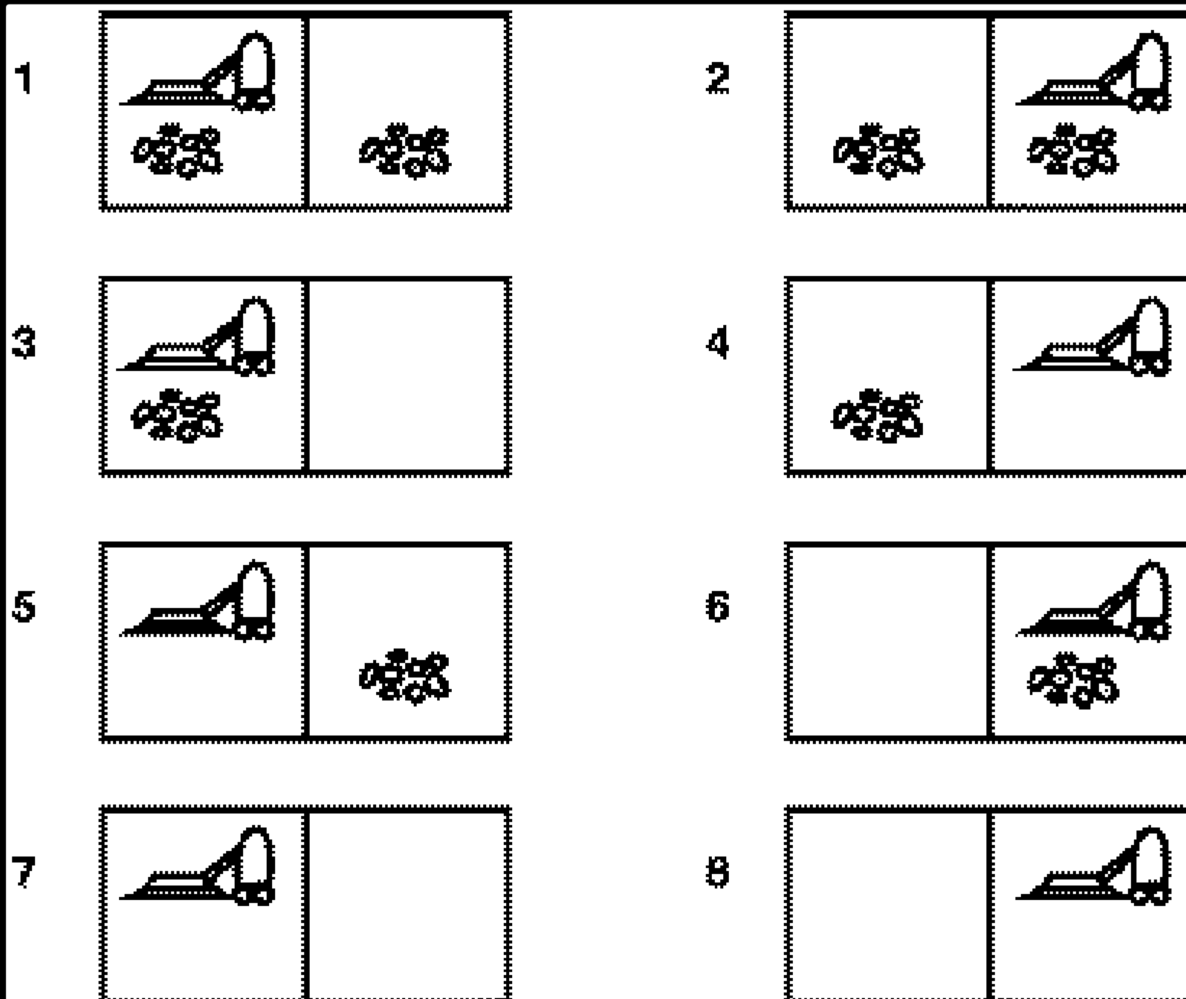

Goal
Agent

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |


Q



Oradea
71
Zerind
151
75
Arad
140
Neamt
87
Iasi
92
Sibiu
99
Fagaras
118
80
Vaslui
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
142
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
Drobeta
120
138
Bucharest
86
Craiova
90
Giurgiu
Eforie


1
2
3
4
5
6
7
8

# Problem Solving as Search

Looking at problems as a space of possibilities where we have to discover solutions by looking at a wide variety of paths.

# 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 reachable states

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

# Understanding Costs

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

# Terminology

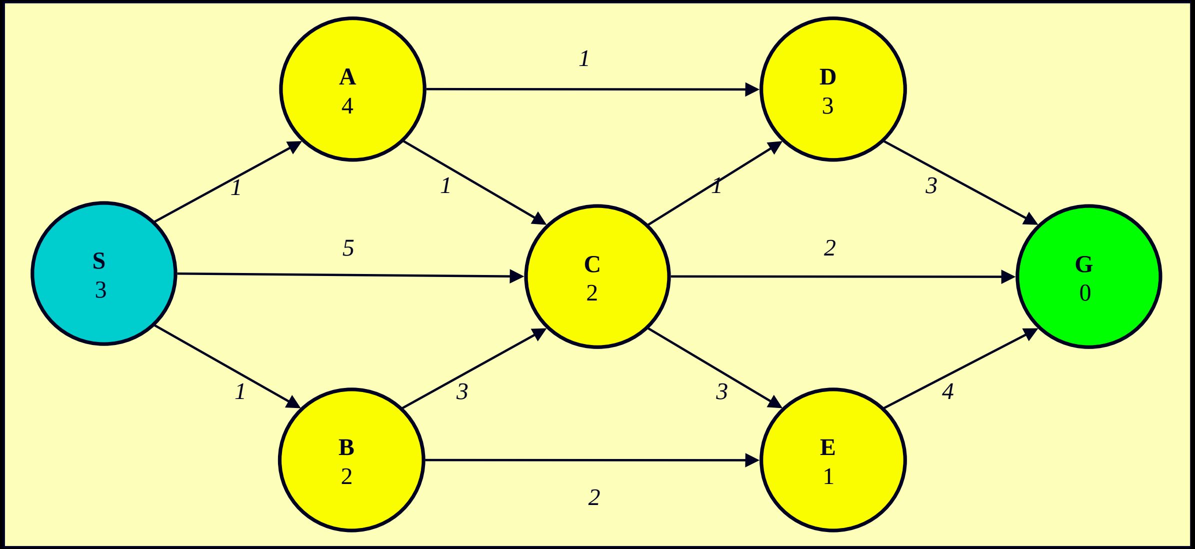
## Search tree

- 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)

# General Approach

- 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

# A very abstract example




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

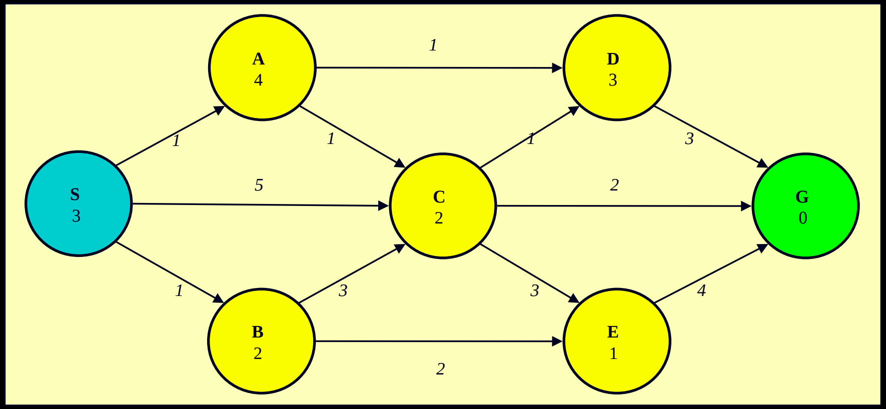
This graph has additional information

Names and properties for the states (e.G. S, 3)

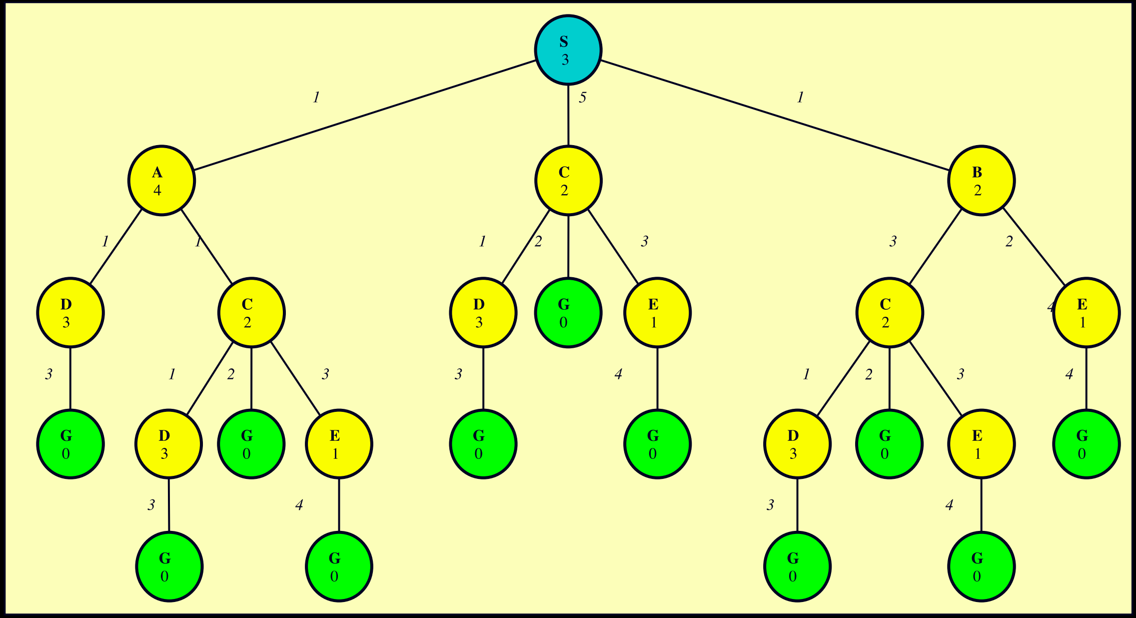
Links between nodes, specified by the successor function

Properties for links (distance, cost, name, ...)

# Graphs and Trees




- 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 Search

```
function GENERAL-SEARCH(problem, QUEUING-FN) returns solution
    nodes  := MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[problem]))
     loop do
       if nodes is empty then return failure
          node := REMOVE-FRONT(nodes)
       if GOAL-TEST[problem] applied to STATE(node) succeeds
          then return node
       nodes   := QUEUING-FN(nodes, EXPAND(node,
    OPERATORS[problem]))
    end
```

# Our Metrics

- 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

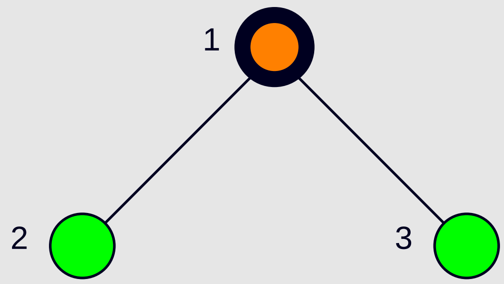
# Uninformed Search

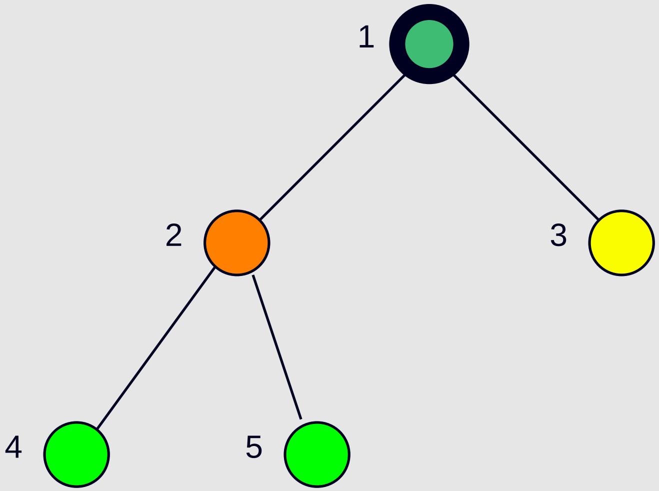
Breadth-first
Depth-first
Uniform-cost Search
Depth-limited Search
Iterative Deepening
Bi-directional Search

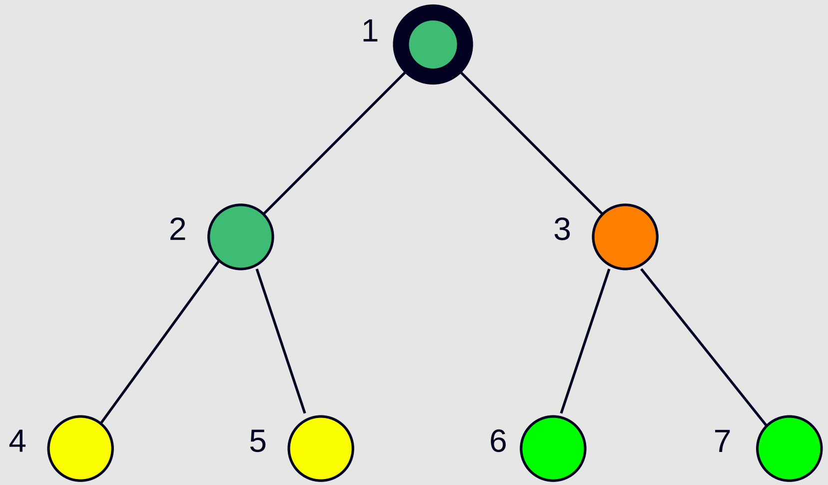
# Breadth First

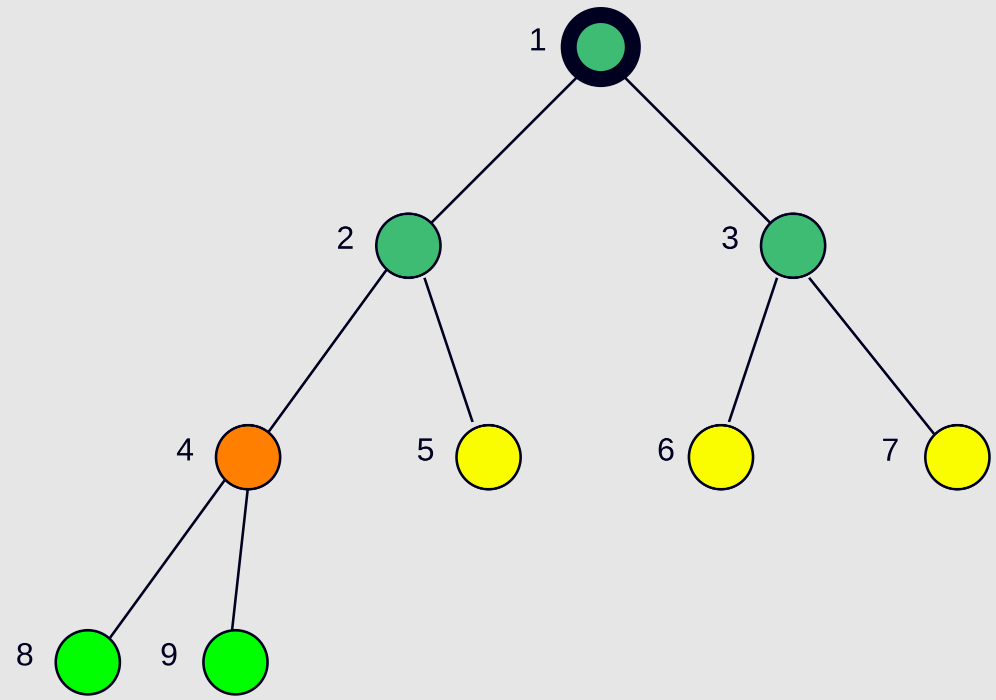
All the nodes reachable from the current node are explored first

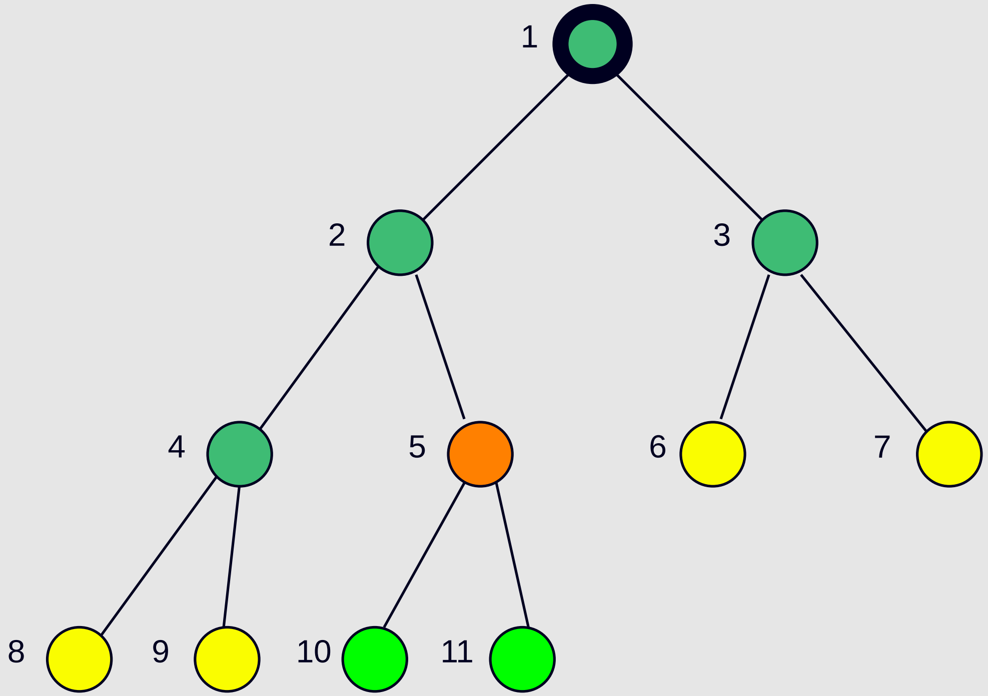
- Achieved by the TREE-SEARCH method by appending newly generated nodes at the end of the search queue

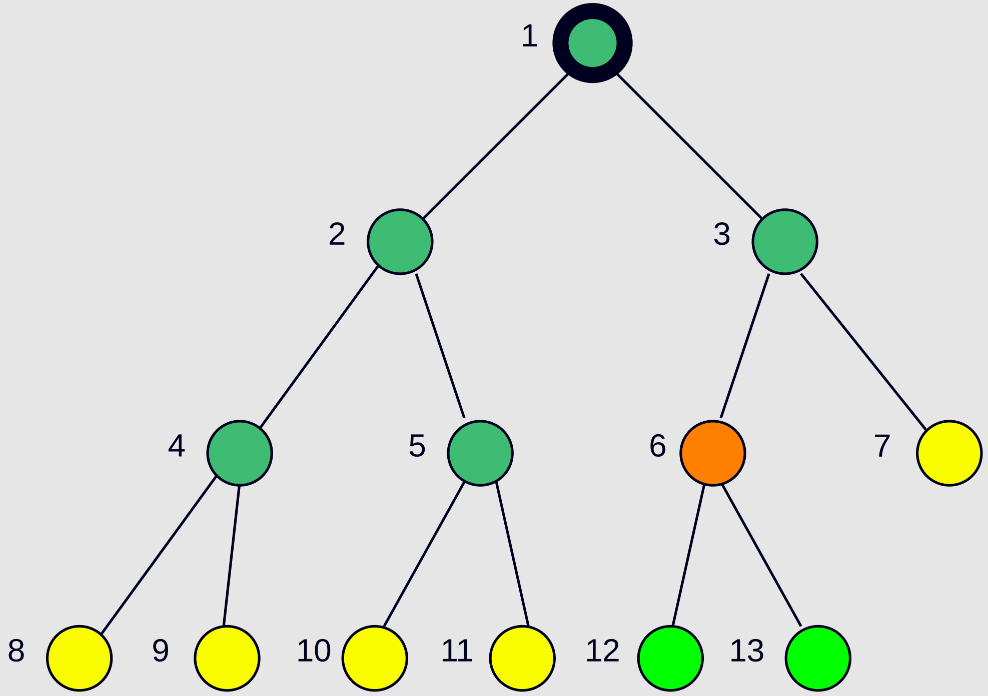

1
2
3

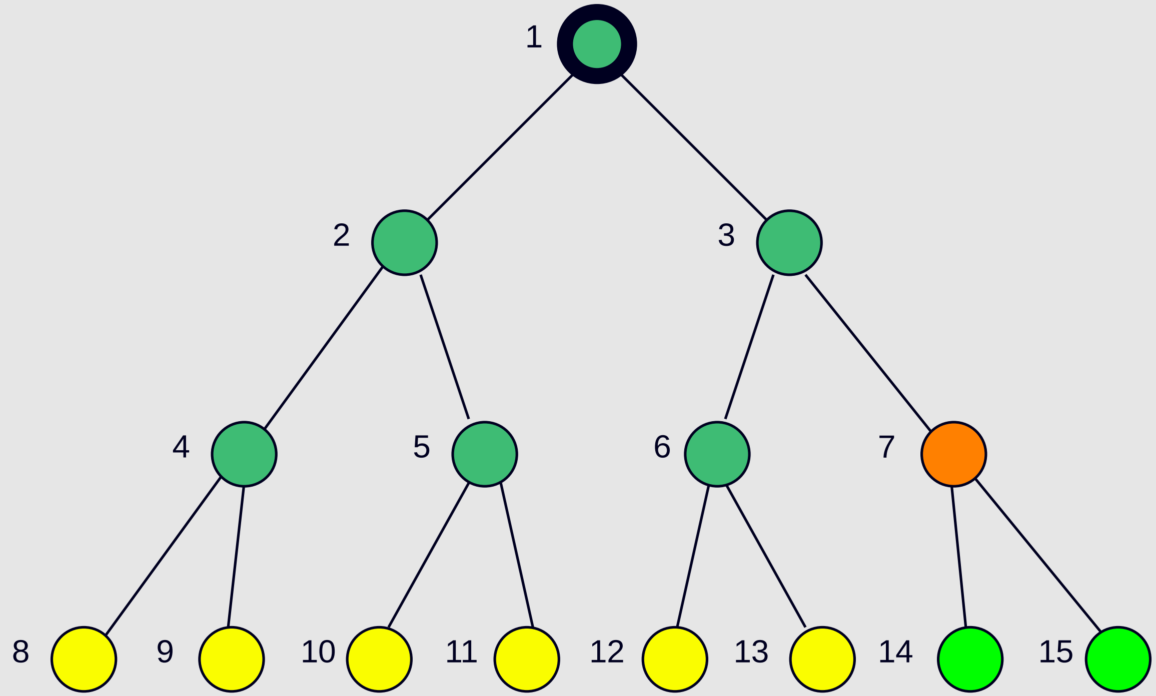

1
2
3
4
5

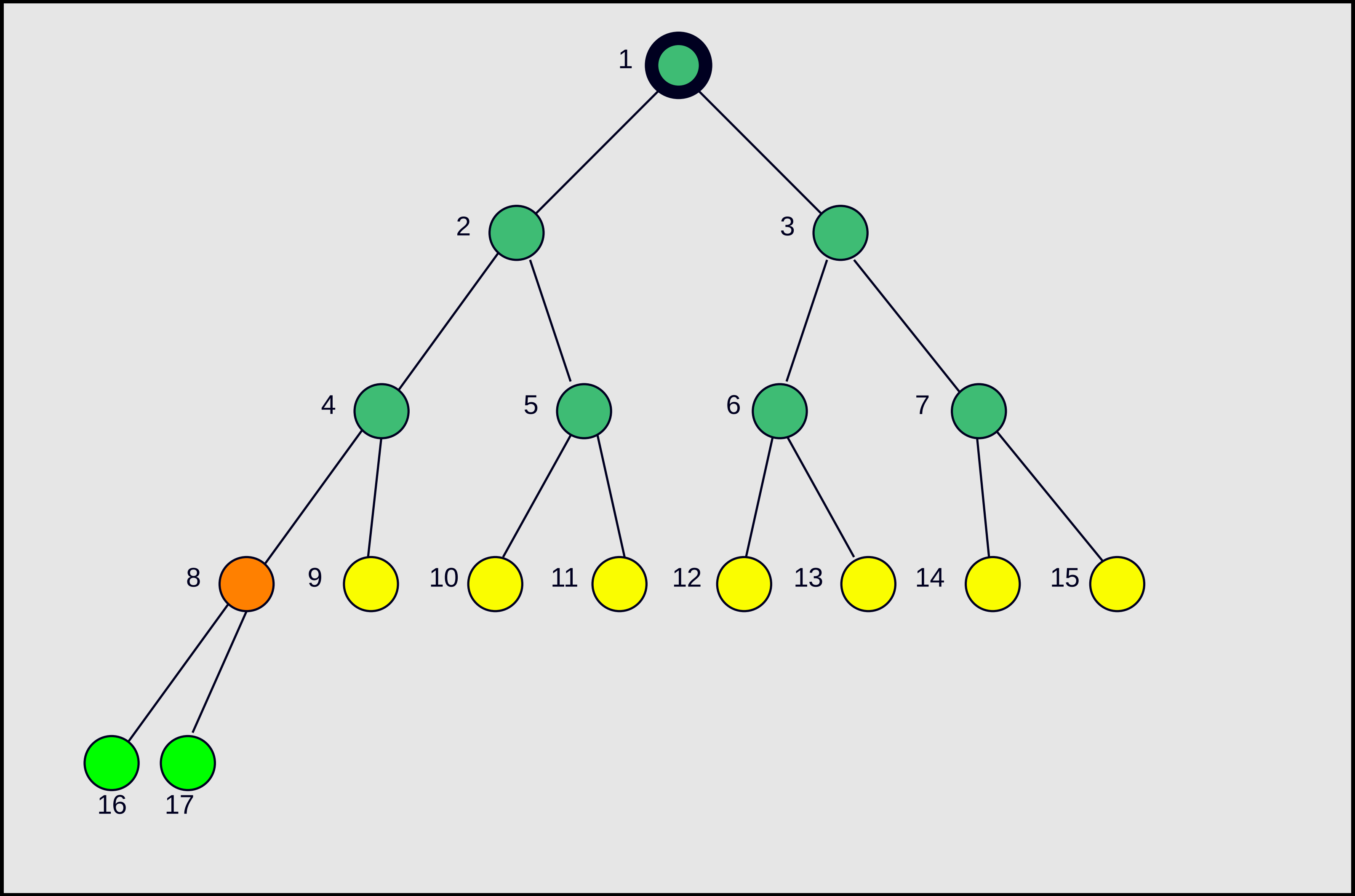

1
2
3
4
5
6
7

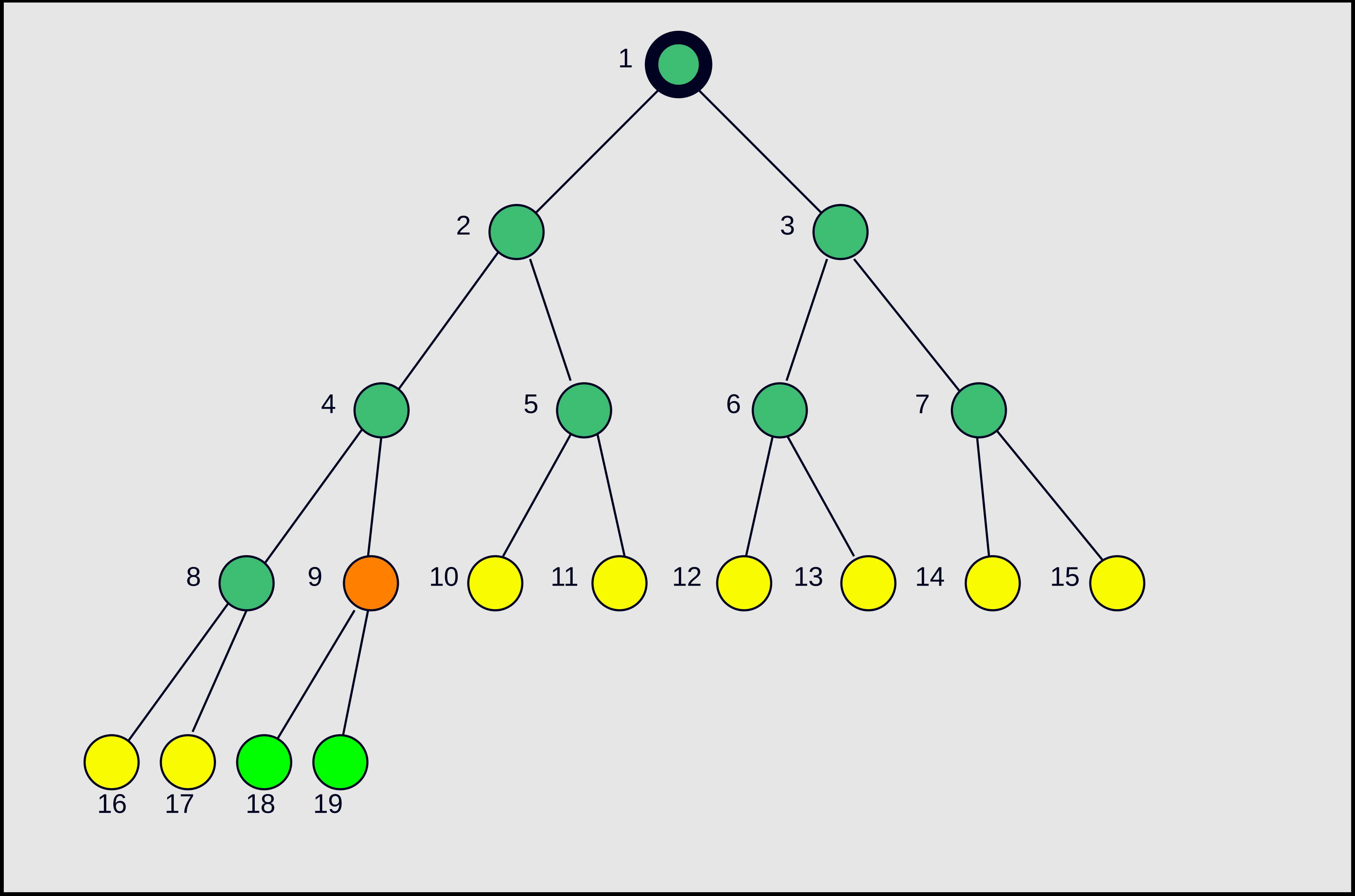

1
2
3
4
5
6
7
8
9

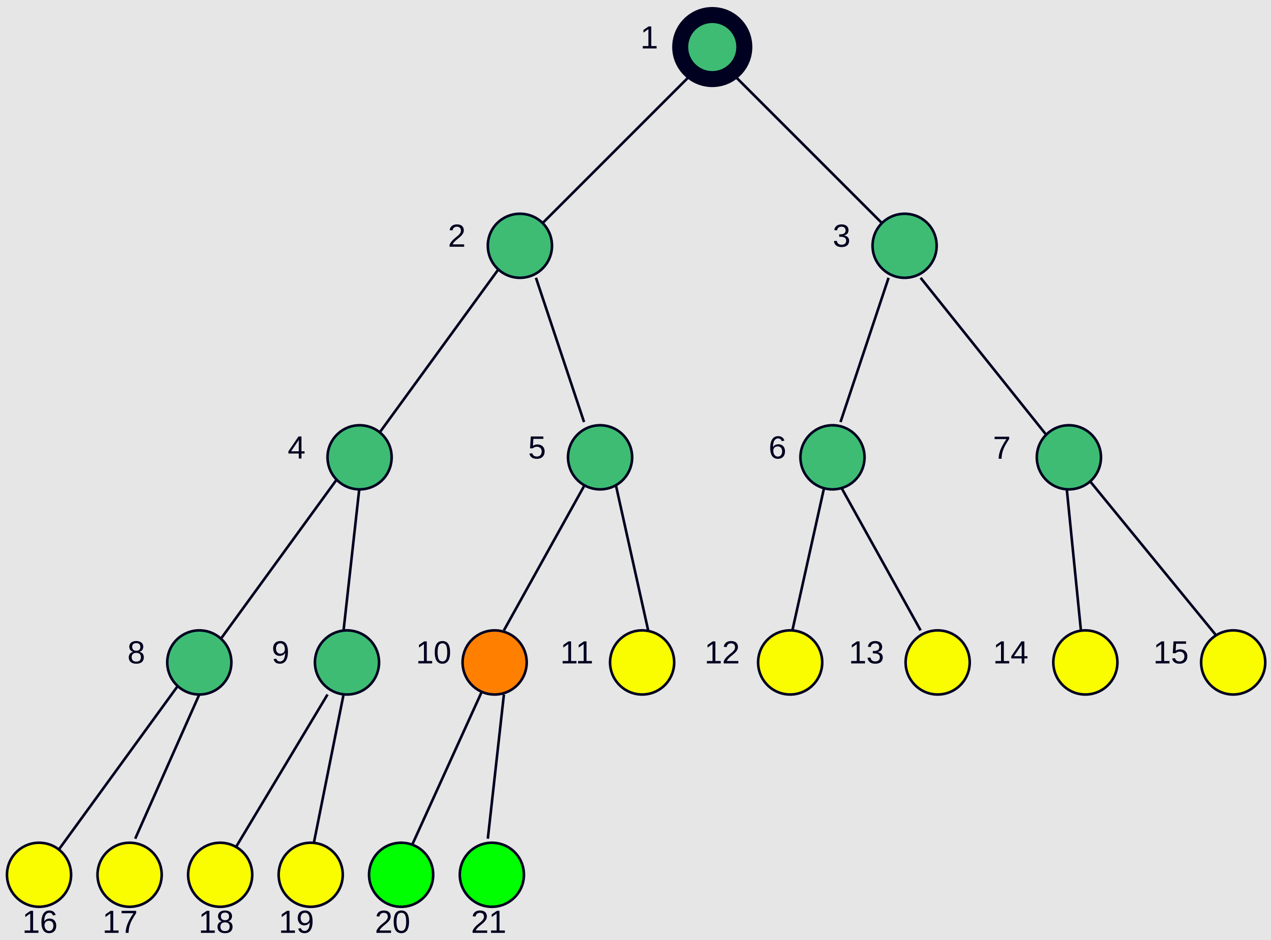

1
2
3
4
5
6
7
8
9
10
11

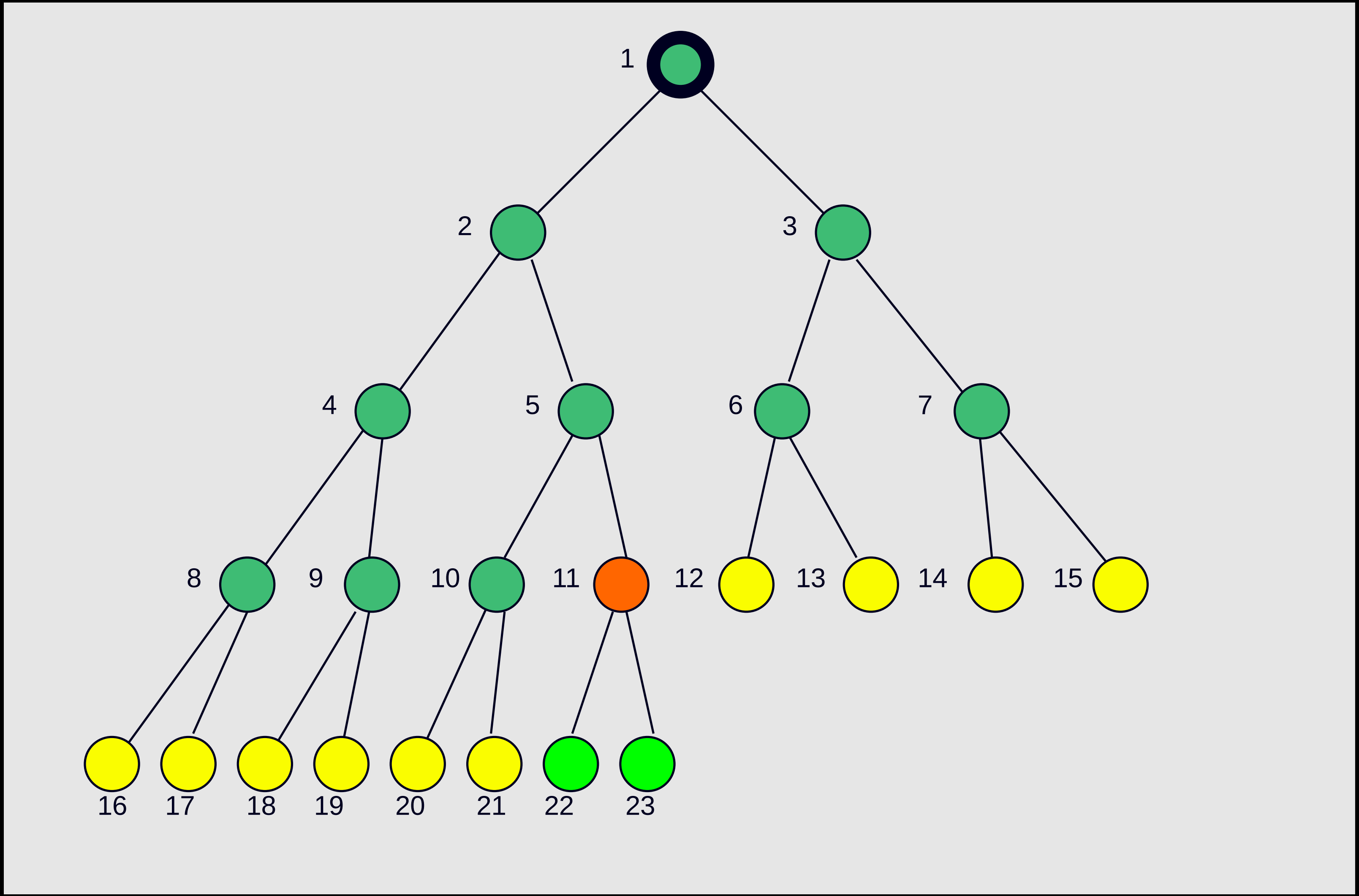

1
2
3
4
5
6
7
8
9
10
11
12
13

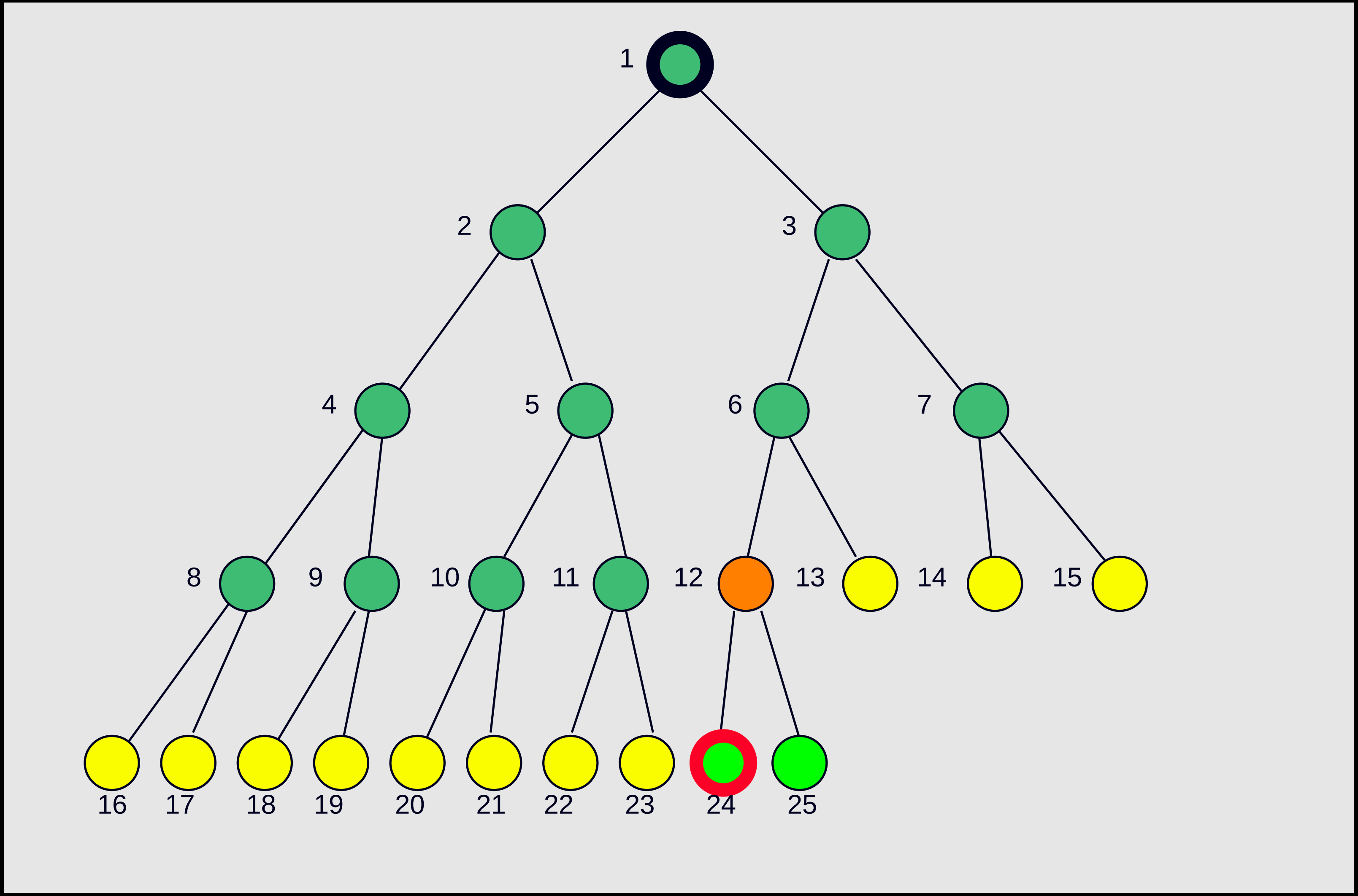

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

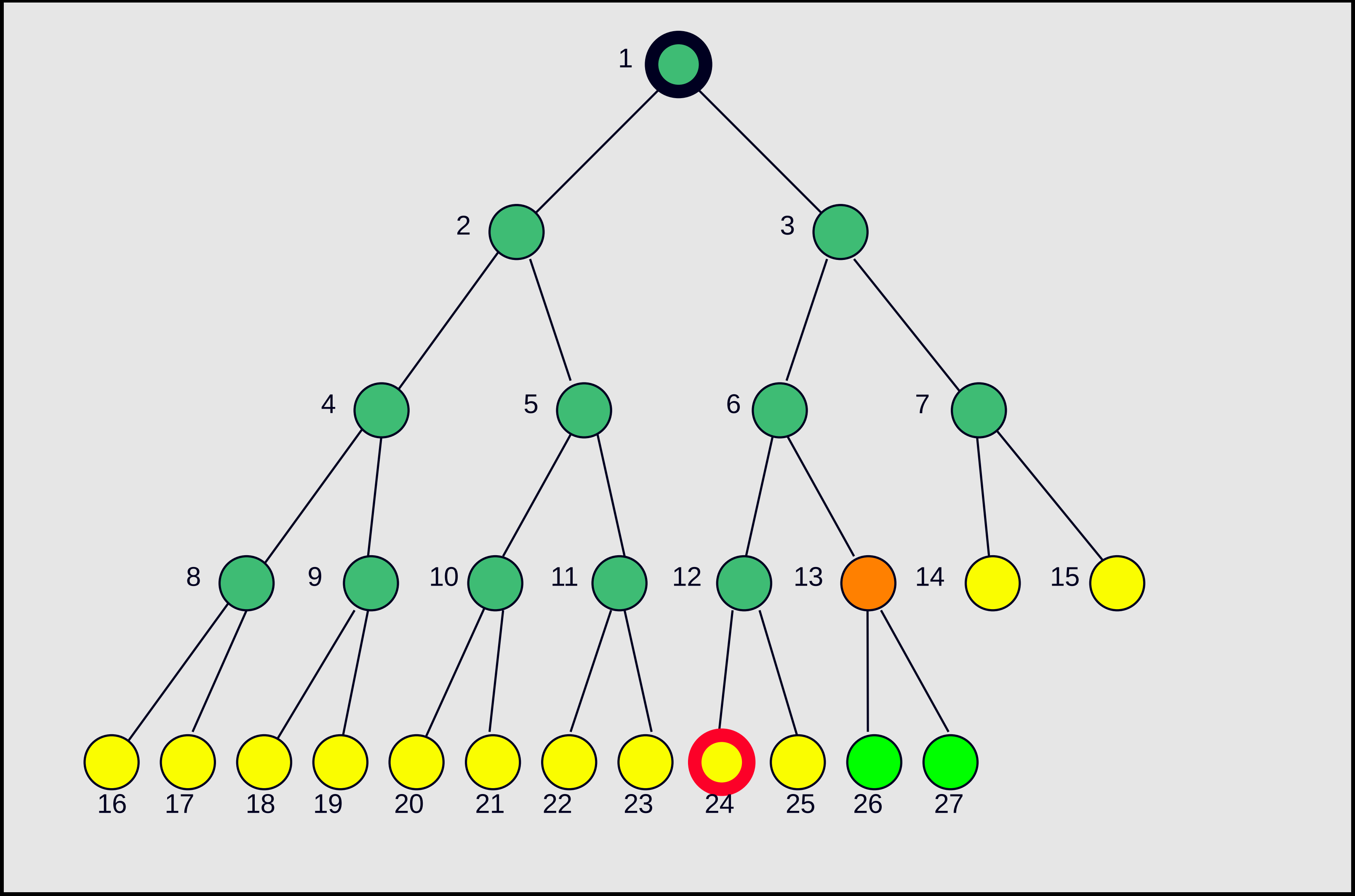

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

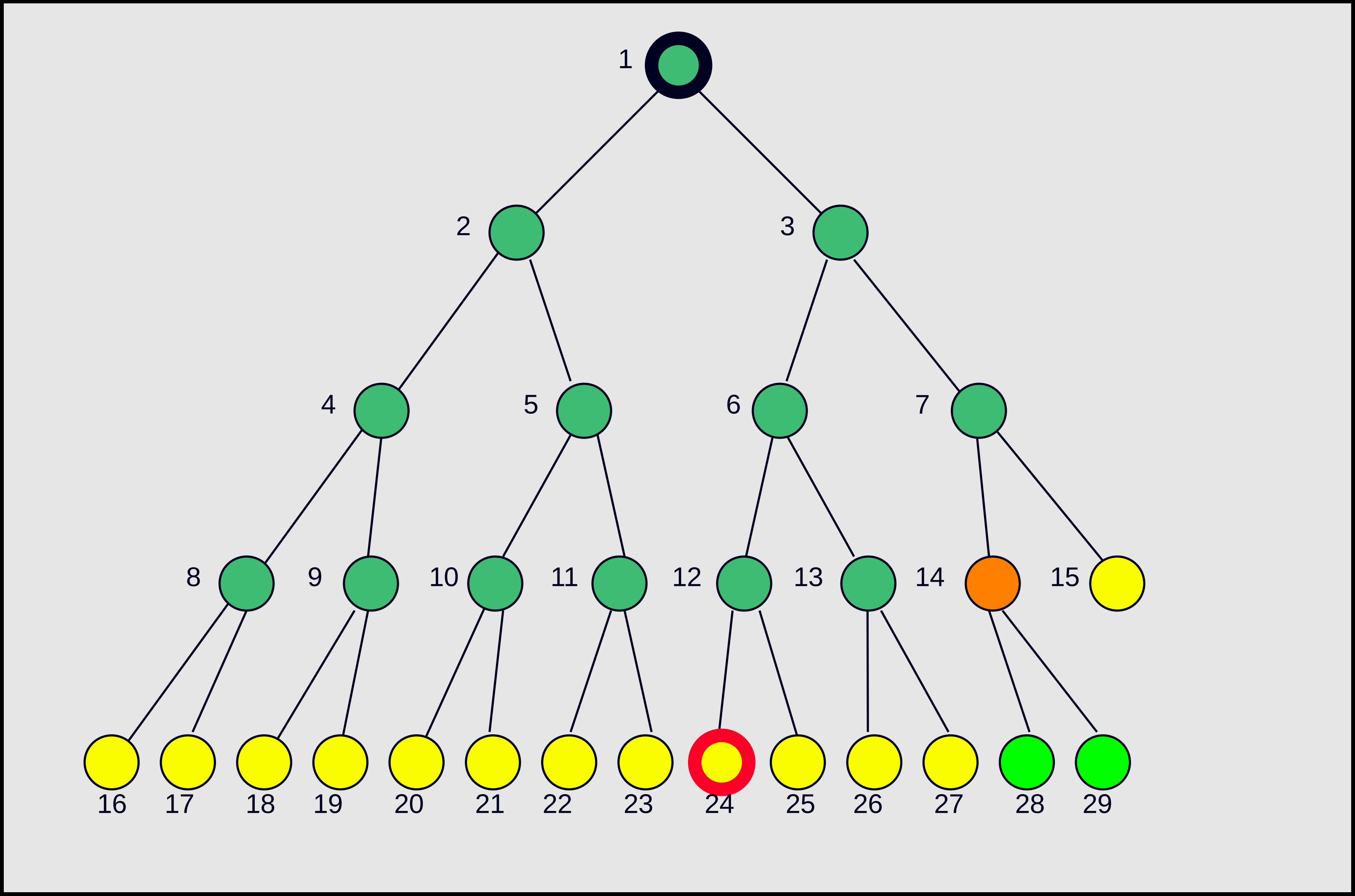

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

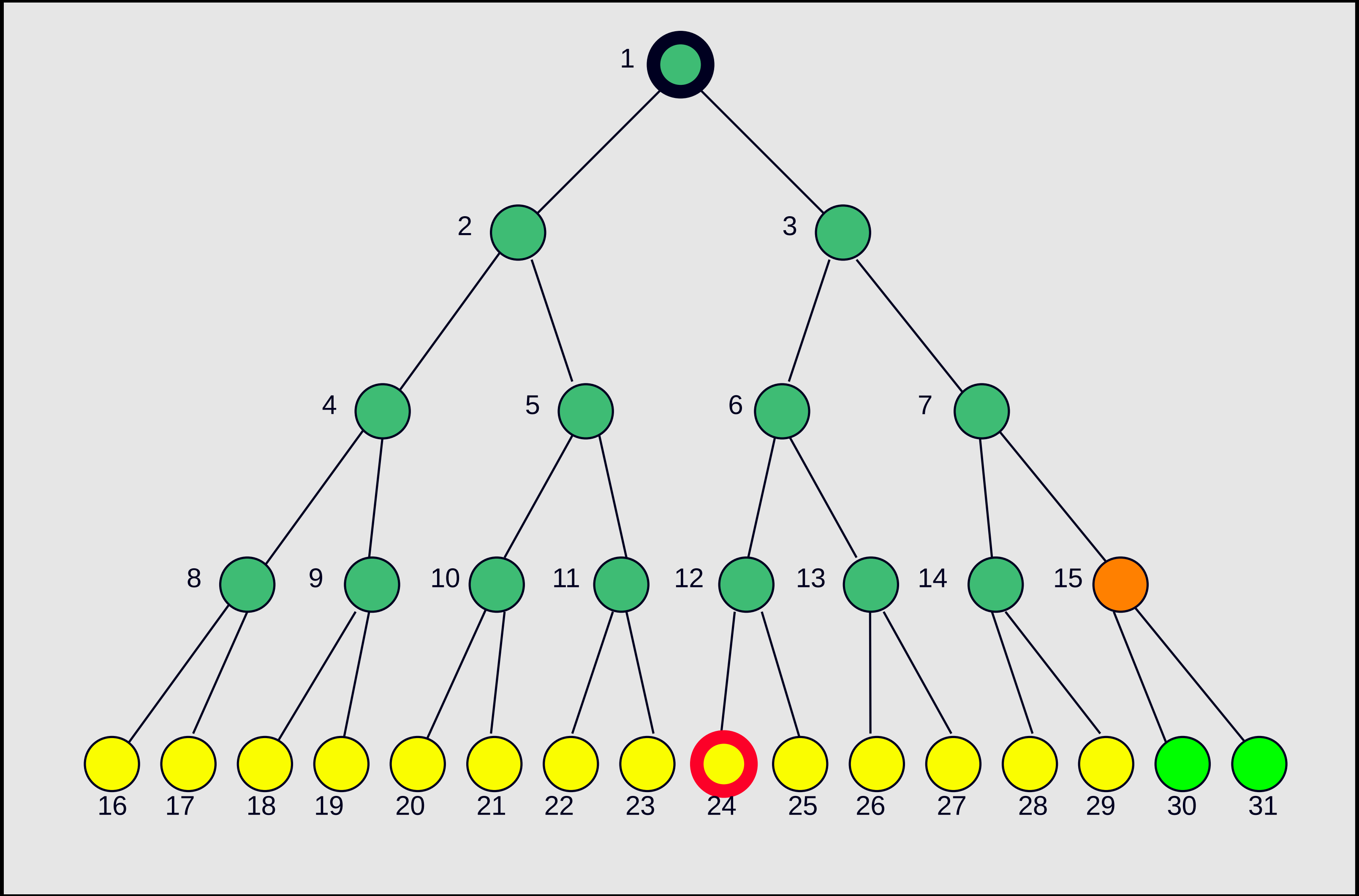

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

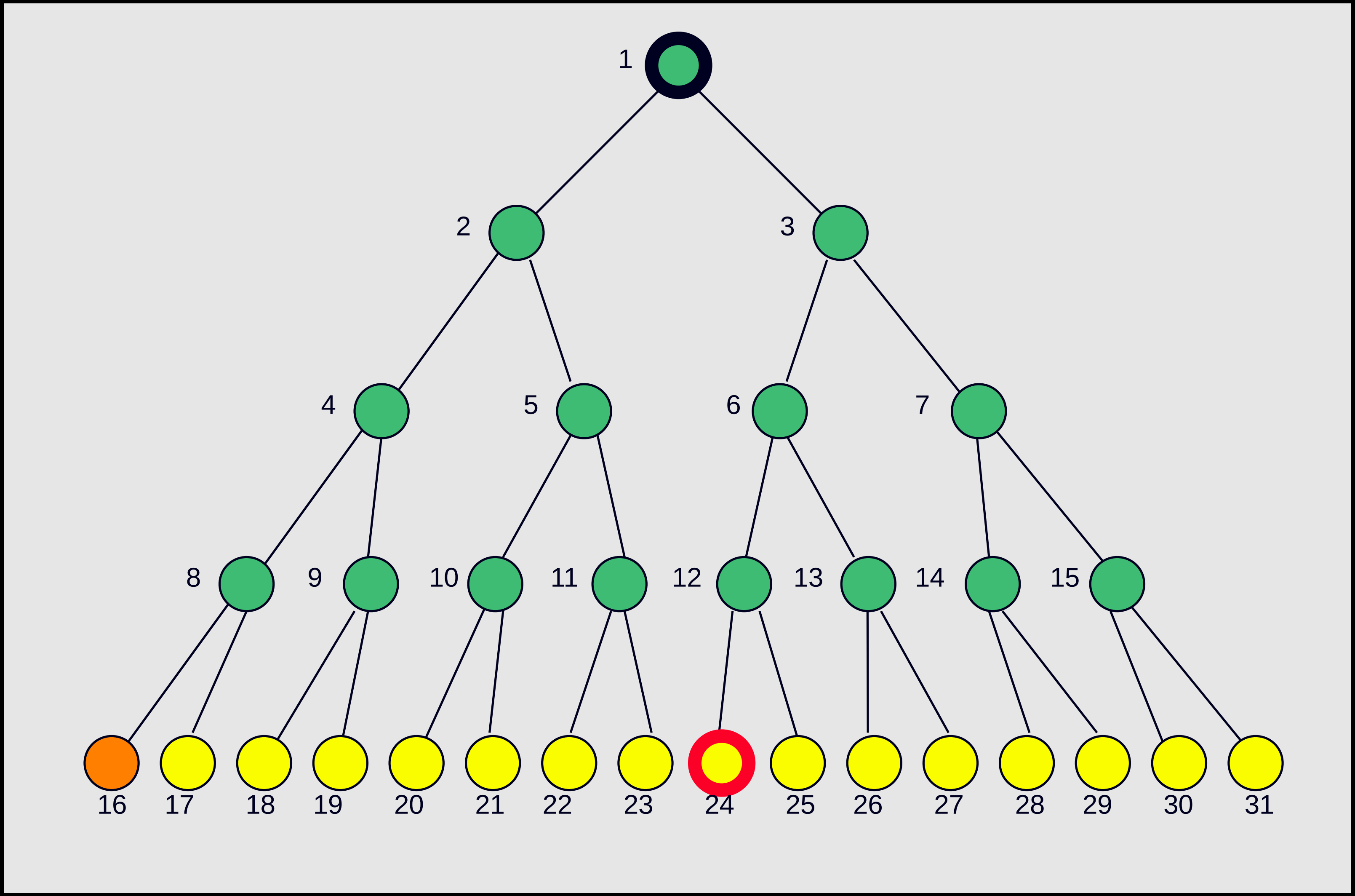

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

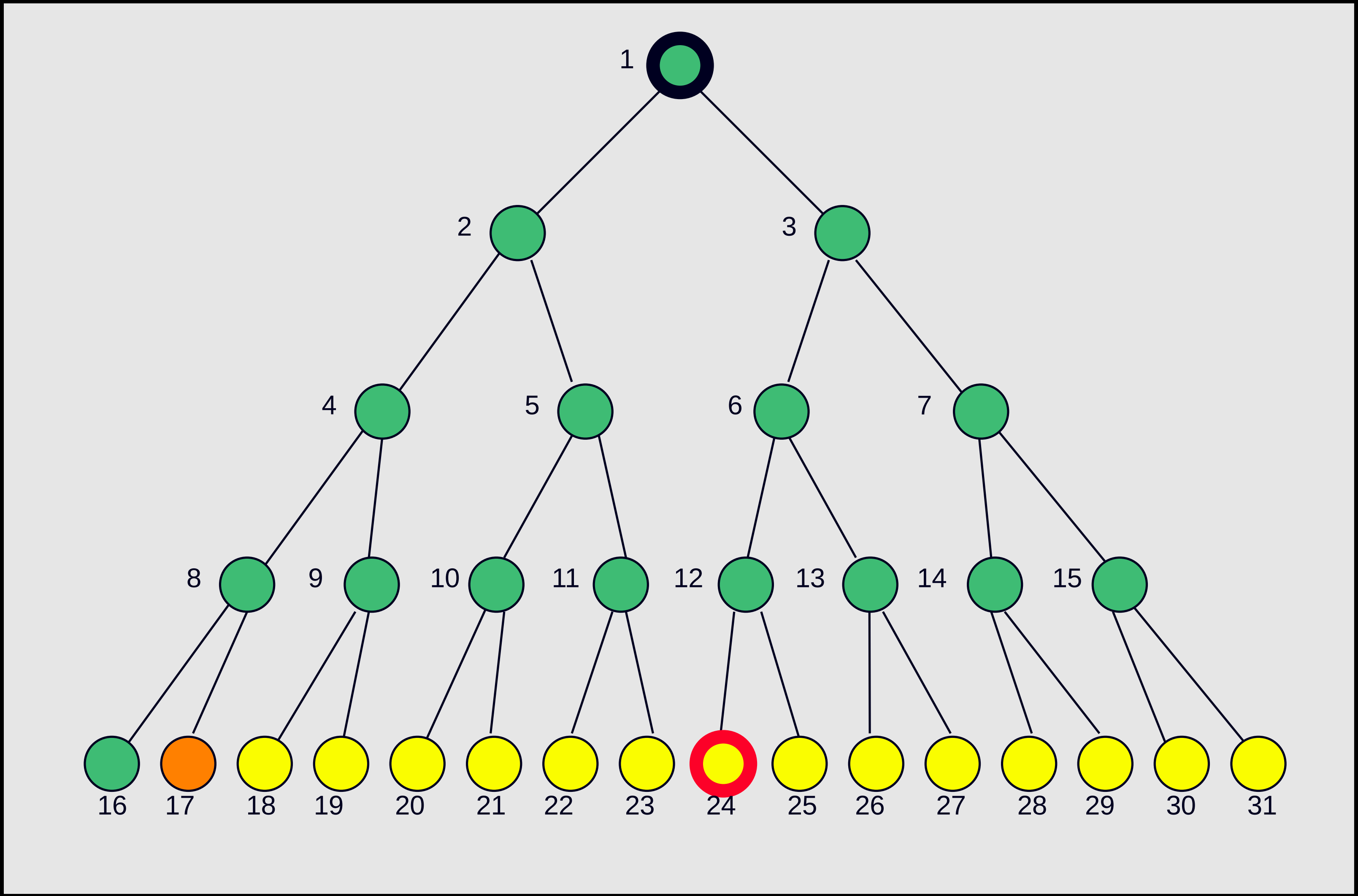

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

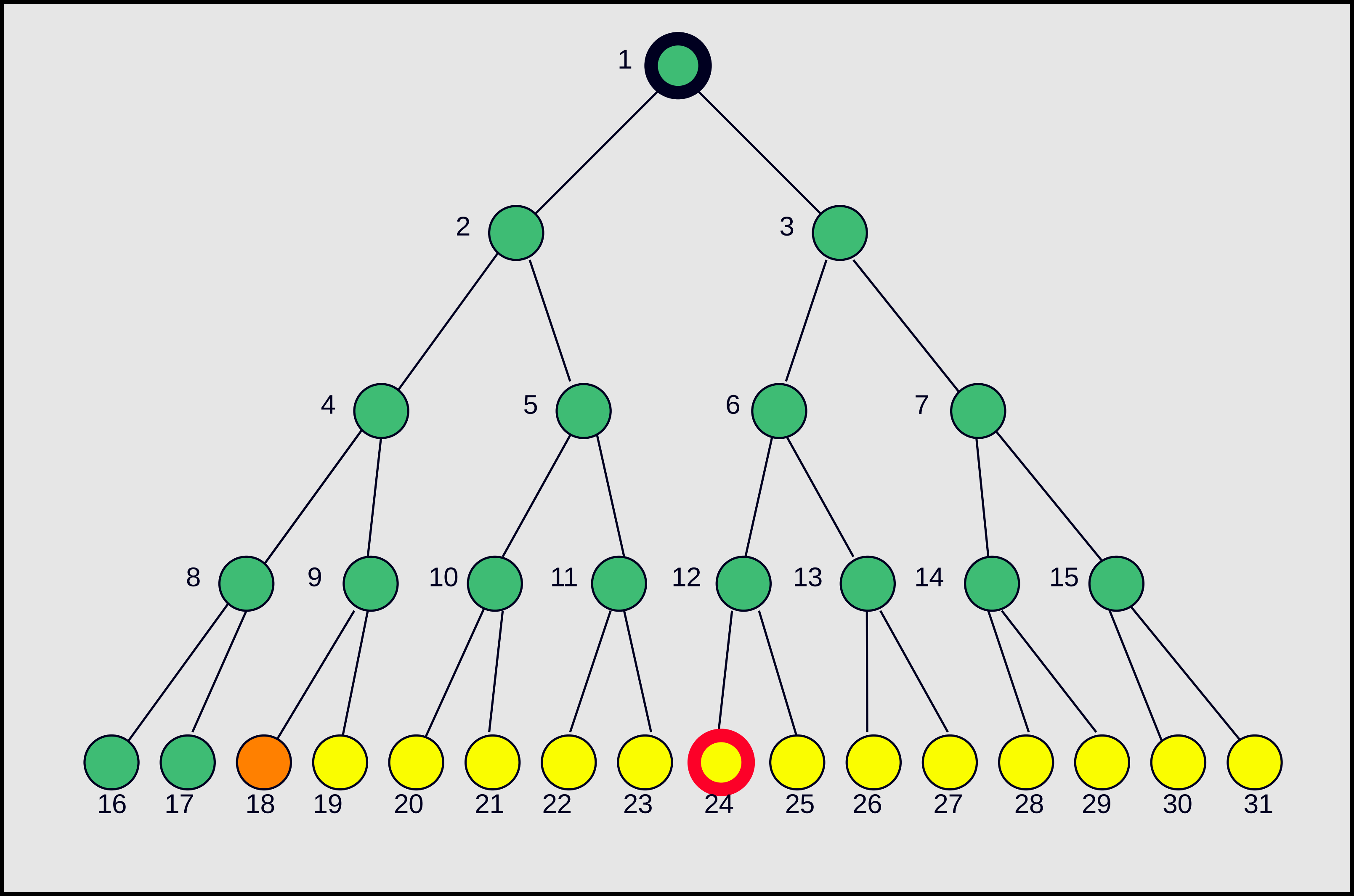

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

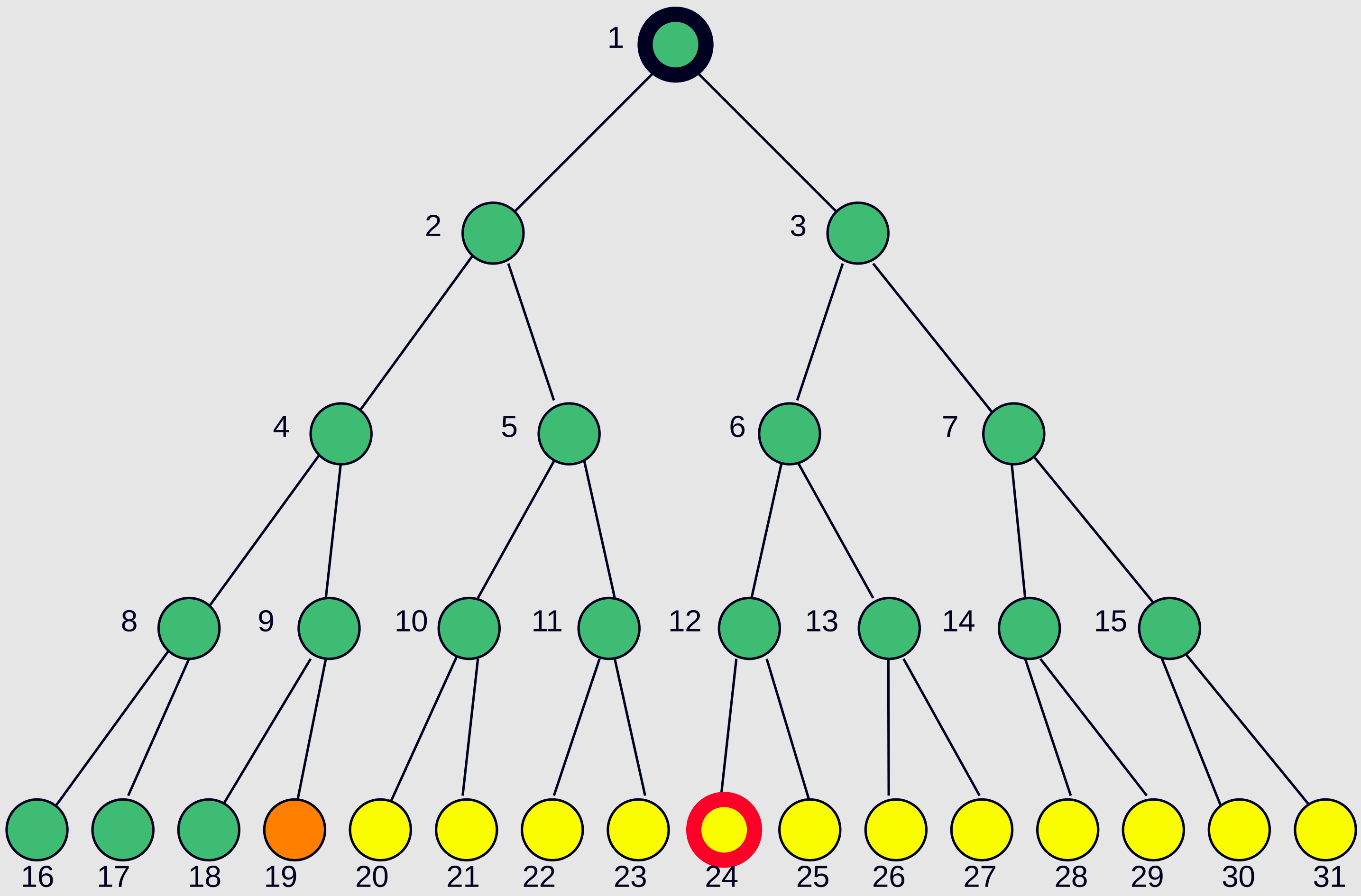

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29

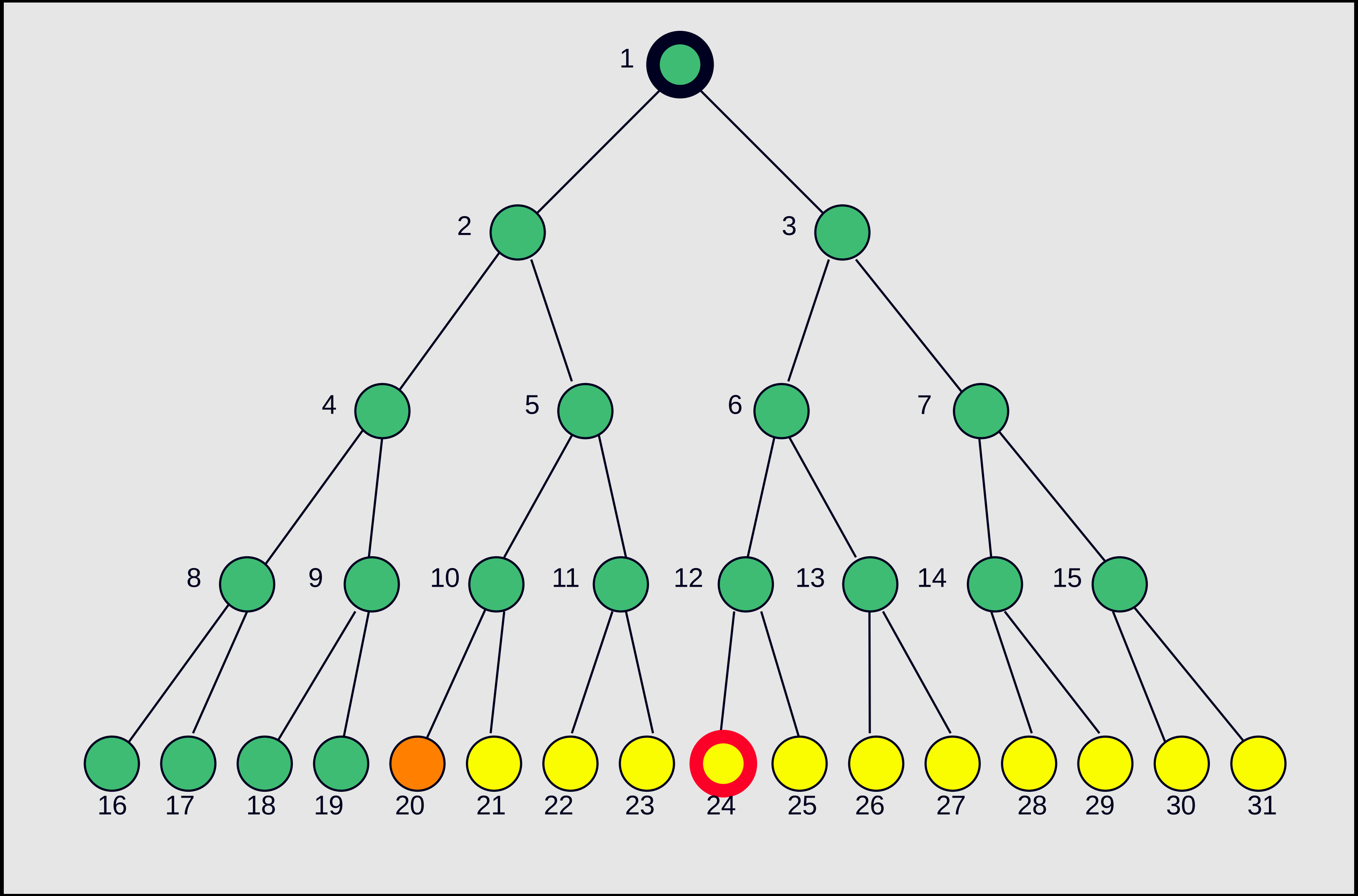

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

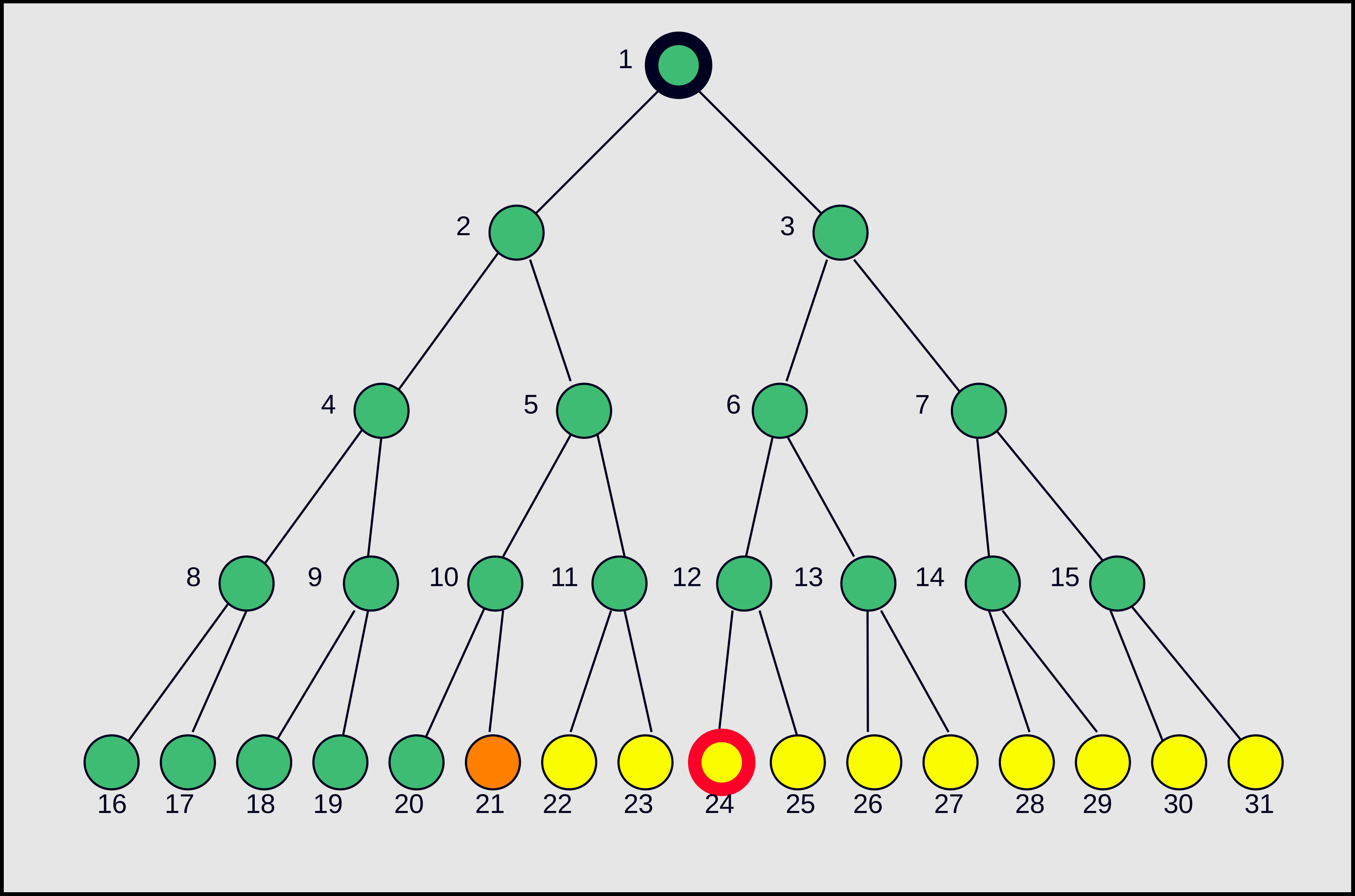

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

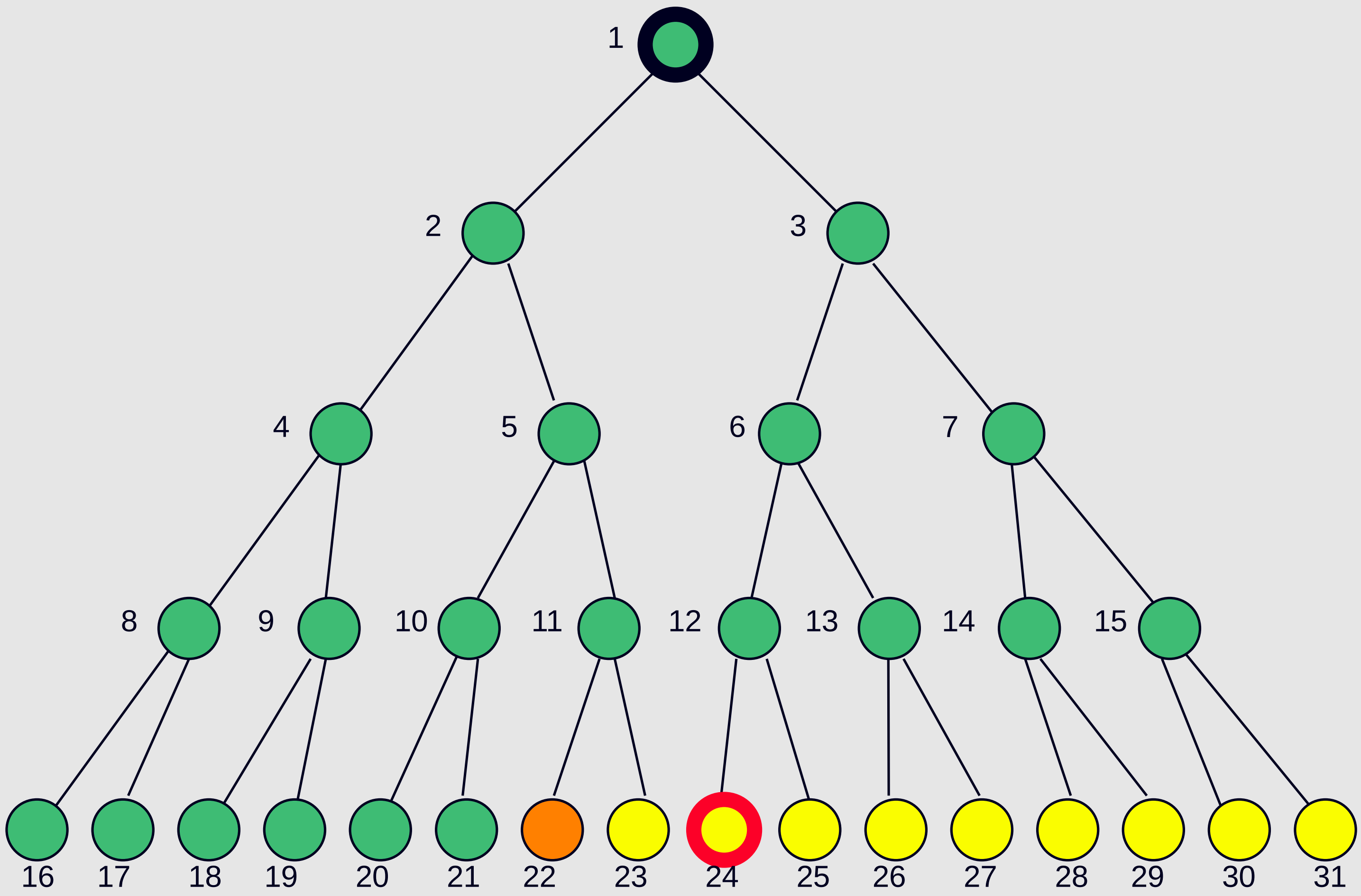

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

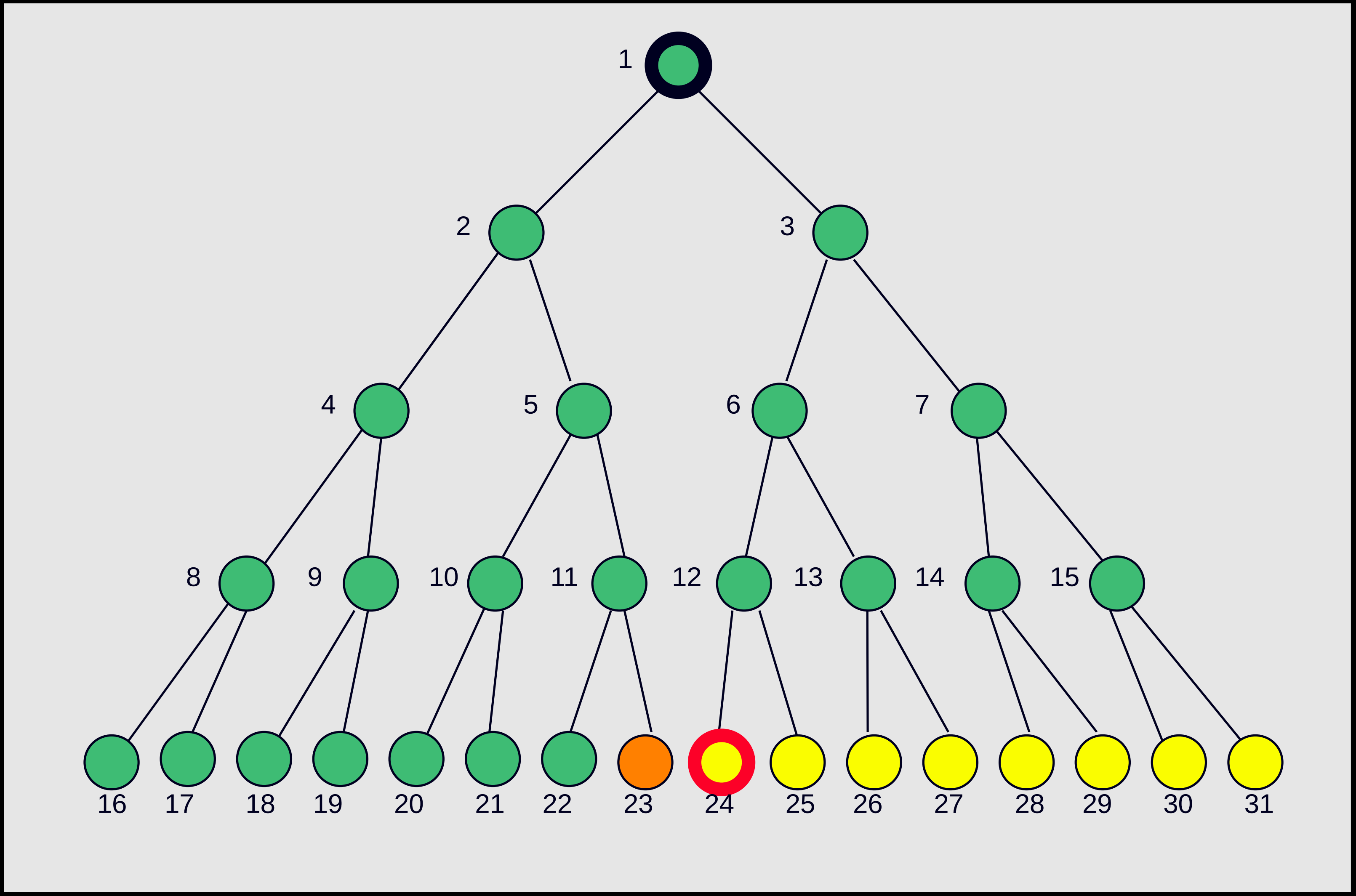

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

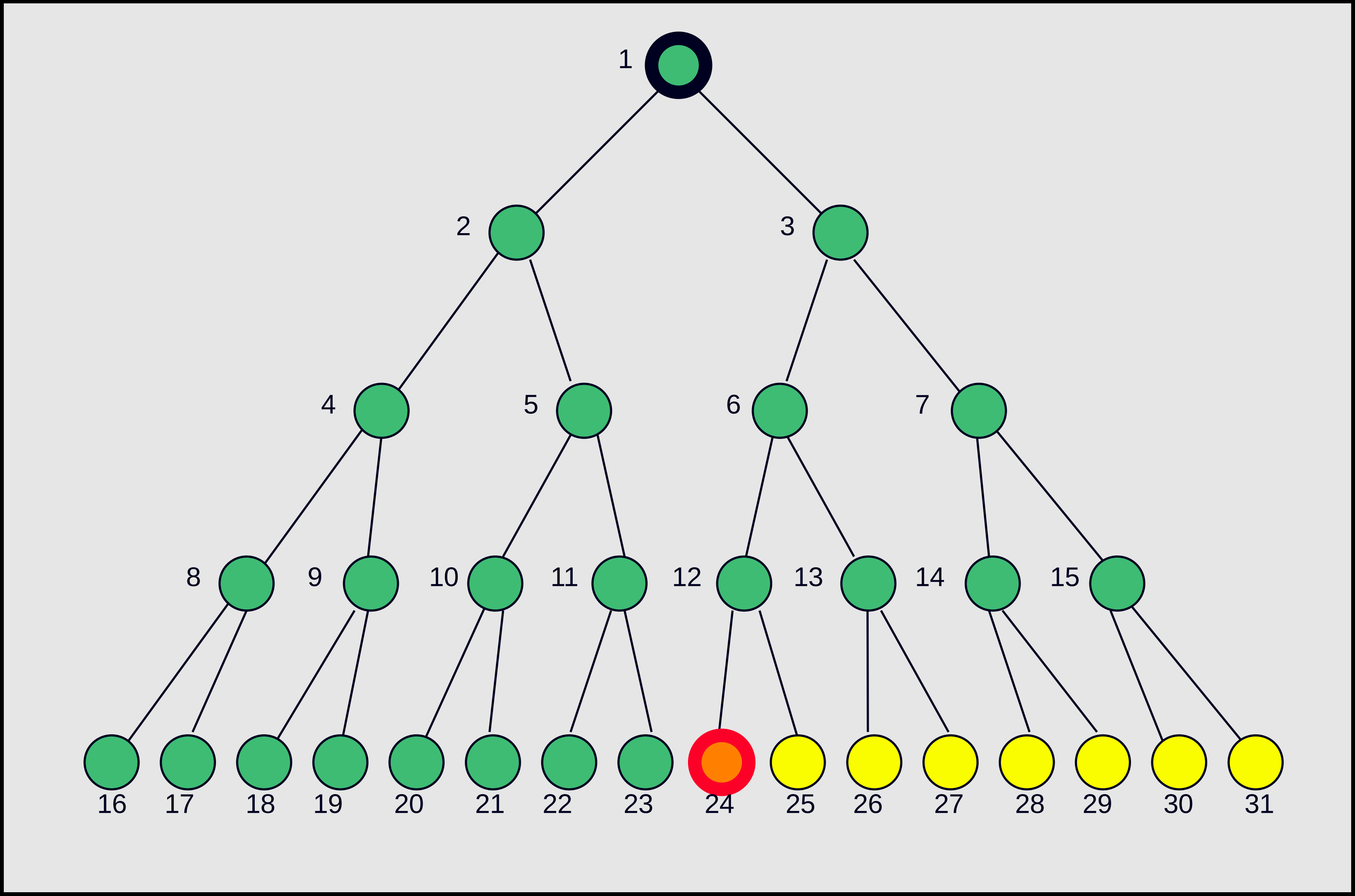

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

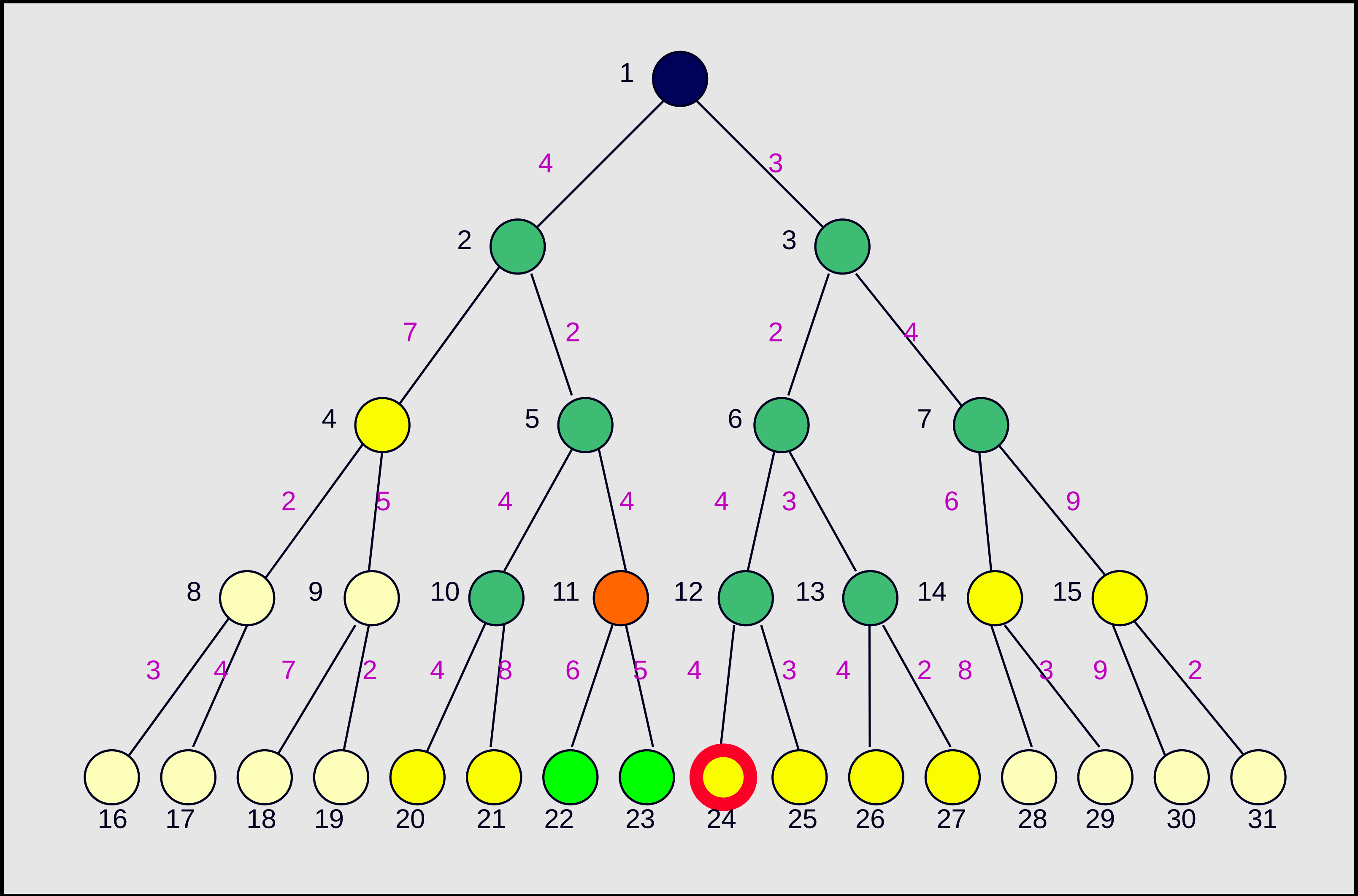

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

# Uniform Cost

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



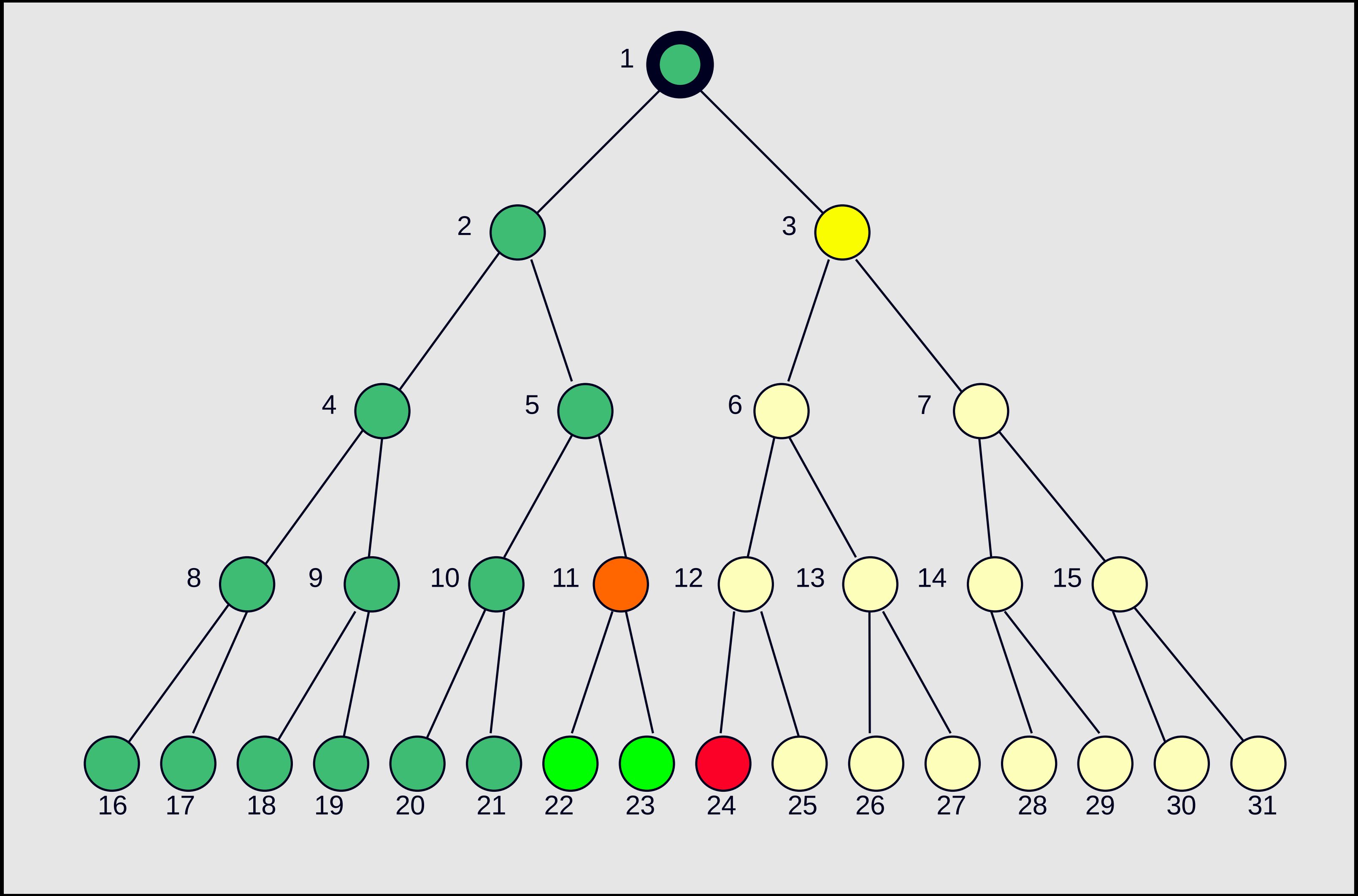
1
4
3
2
3
7
2
2
4
4
5
6
7
2
5
4
4
4
3
6
9
8
9
10
11
12
13
14
15
3
4
7
2
4
8
6
5
4
3
4
2
8
3
9
2
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

# Breadth 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


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

# 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

- 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

# Limited Depth

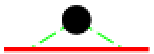
Similar to depth-first, but with a limit

- 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

# Iterative Deepening

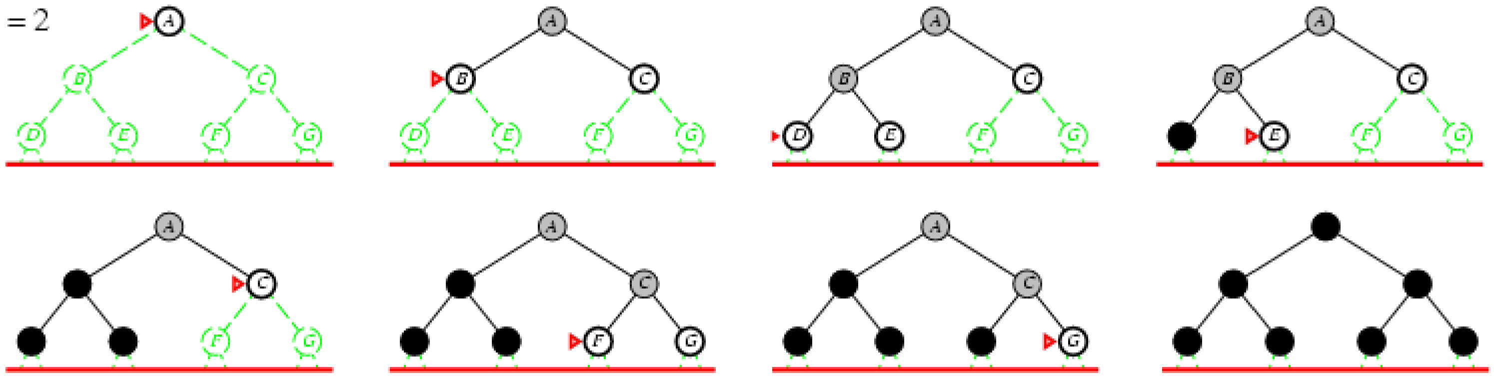
- Applies LIMITED-DEPTH with increasing depth limits
  - Combines advantages of BREADTH-FIRST and DEPTH-FIRST methods
  - Many states are expanded multiple times
    - Doesn’t really matter because the number of those nodes is small
  - In practice, one of the best uninformed search methods
    - For large search spaces, unknown depth

Limit = 0

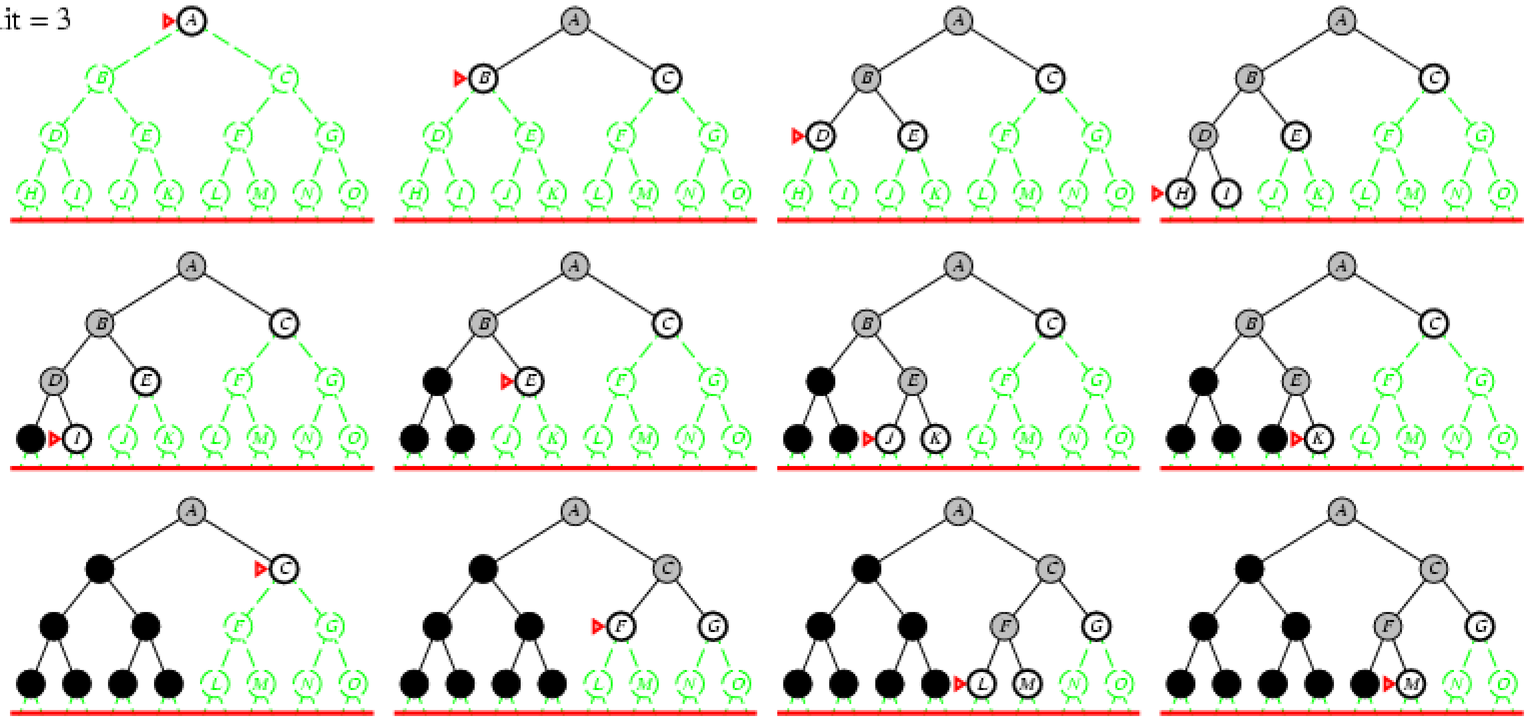



Limit = 1

Limit = 2



Limit = 3



# Bidirectional

- 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

# 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

# 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

- 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

# A*

- Uses the (estimated) cheapest path through the current node
  - $F(n) = g(n) + h(n)$
    $= \text{path cost} + \text{estimated cost to the goal}$
  - Heuristics must be admissible
    - Never overestimate the cost to reach the goal
  - Very good search method, but with complexity problems