Artificial Intelligence
Bishoy Nader Fathy (21)
Marc Magdi Kamel (55)

Eight Puzzle Solver

OVERVIEW

An instance of the 8-puzzle game consists of a board holding 8 distinct movable tiles, plus an empty space. For any such board, the empty space may be legally swapped with any tile horizontally or vertically adjacent to it. In this assignment, the blank space is going to be represented with the number 0.
Given an initial state of the board, the search problem is to find a sequence of moves that transitions this state to the goal state; that is, the configuration with all tiles arranged in ascending order 0,1,2,3,4,5,6,7,8.
The search space is the set of all possible states reachable from the initial state. The blank space may be swapped with a component in one of the four directions Up, Down, Left, Right, one move at a time. The cost of moving from one configuration of the board to another is the same and equal to one. Thus, the total cost of path is equal to the number of moves made from the initial state to the goal state.

Data Structures Used

A Set is used to contain the explored states in the three algorithms

1. **Depth First Search (DFS)**

   Used Data Structure: Last In First Out Stack (LIFO)

2. **Breadth First Search (BFS)**

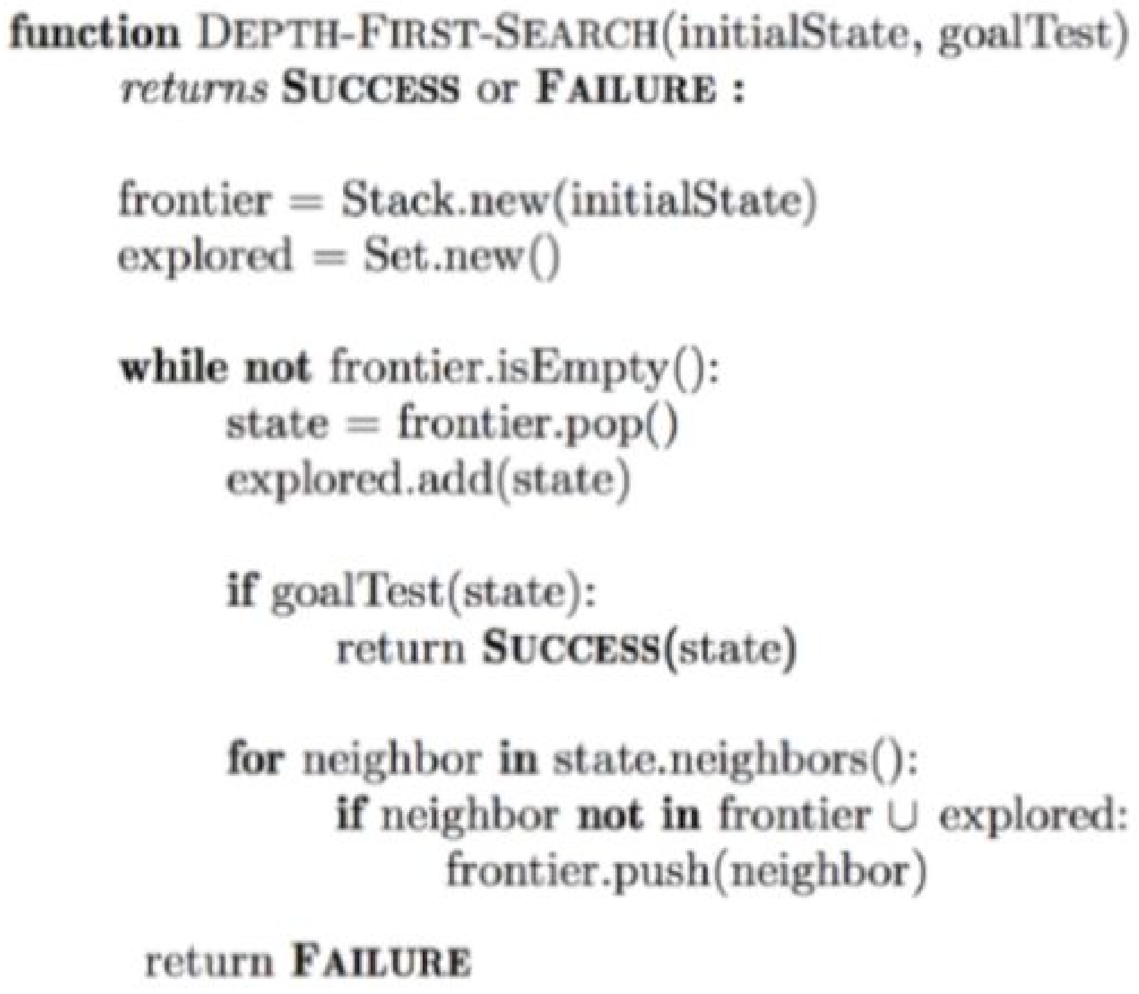
   Used Data Structure: First In First Out Queue (FIFO)

3. **A Star Search (A*)**

   Used Data Structure: Min Heap (Priority Queue in JAVA)

Algorithms Used

1. Depth First Search (DFS)

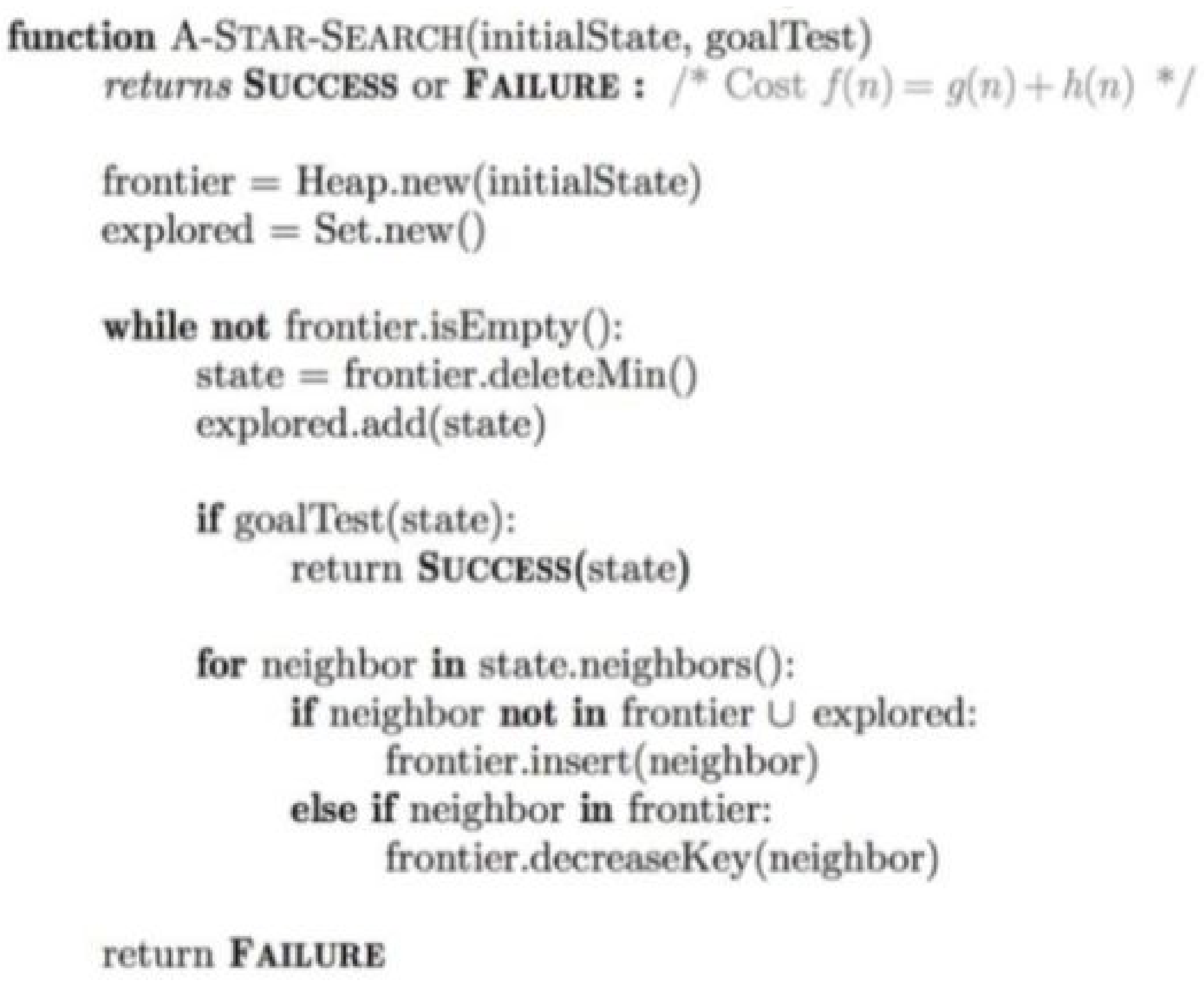


```
function DEPTH-FIRST-SEARCH(initialState, goalTest)
        returns SUCCESS or FAILURE :

        frontier = Stack.new(initialState)
        explored = Set.new()

        while not frontier.isEmpty():
                state = frontier.pop()
                explored.add(state)

                if goalTest(state):
                        return SUCCESS(state)

                for neighbor in state.neighbors():
                        if neighbor not in frontier ∪ explored:
                                frontier.push(neighbor)

        return FAILURE
```

2. Breadth First Search (BFS)

```
function BREADTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Queue.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.dequeue()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier $\cup$ explored:
                frontier.enqueue(neighbor)

    return FAILURE
```

3. A Star Search (A*)



```
function A-STAR-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost $f(n) = g(n) + h(n)$ */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier $\cup$ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

Extra Work

We used an auxiliary set in DFS and BFS for the frontier data structure to check the element existence in the frontier data structure in Constant order rather than linear time.

We used and auxiliary hashmap to in A Star Algorithm for the frontier data structure to get the nodes faster in Constant order than logarithmic order from the heap.

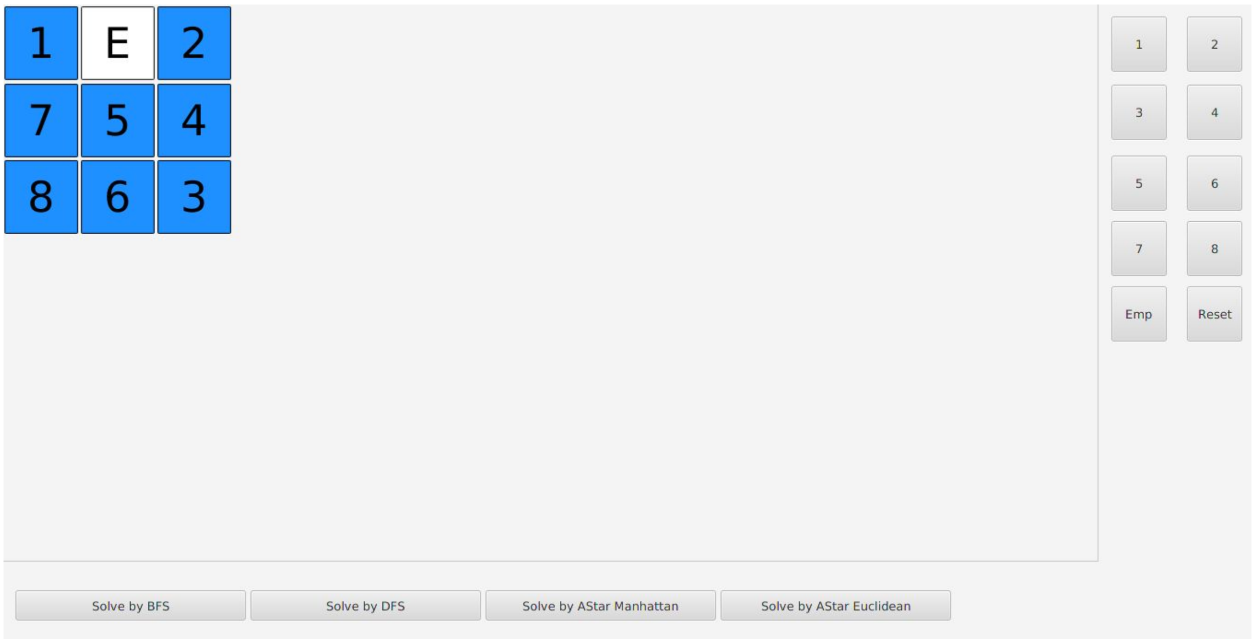
In general A Star using Manhattan Heuristic is better than using Euclidean Heuristic as it is closer to real cost.

Assumptions and details necessary to be clarified

1. At any puzzle state there are a maximum of 4 neighbour states if the 0 square is in the center, 2 neighbour states if the 0 square is in one of the four corners of the puzzle and 3 neighbours otherwise.
2. In Getting the neighbouring states we try to move the 0 square in the following order up -> left -> down -> right.

Sample Runs

Example (1):



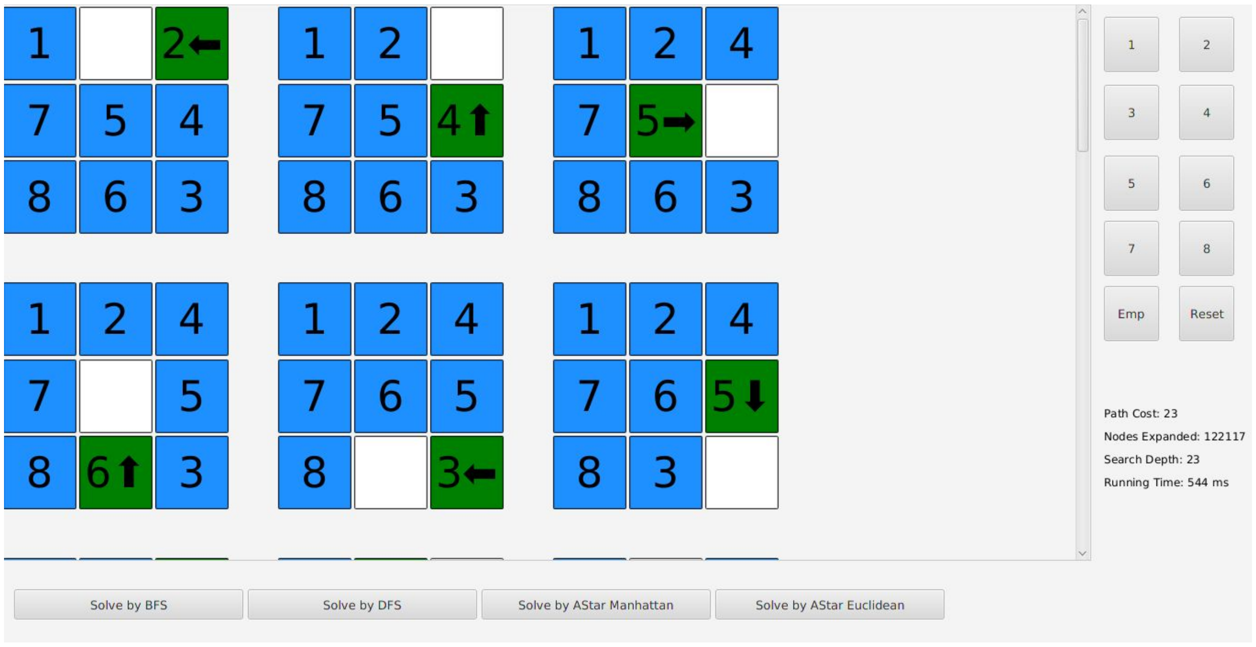
Solving by BFS

Path Cost: 23

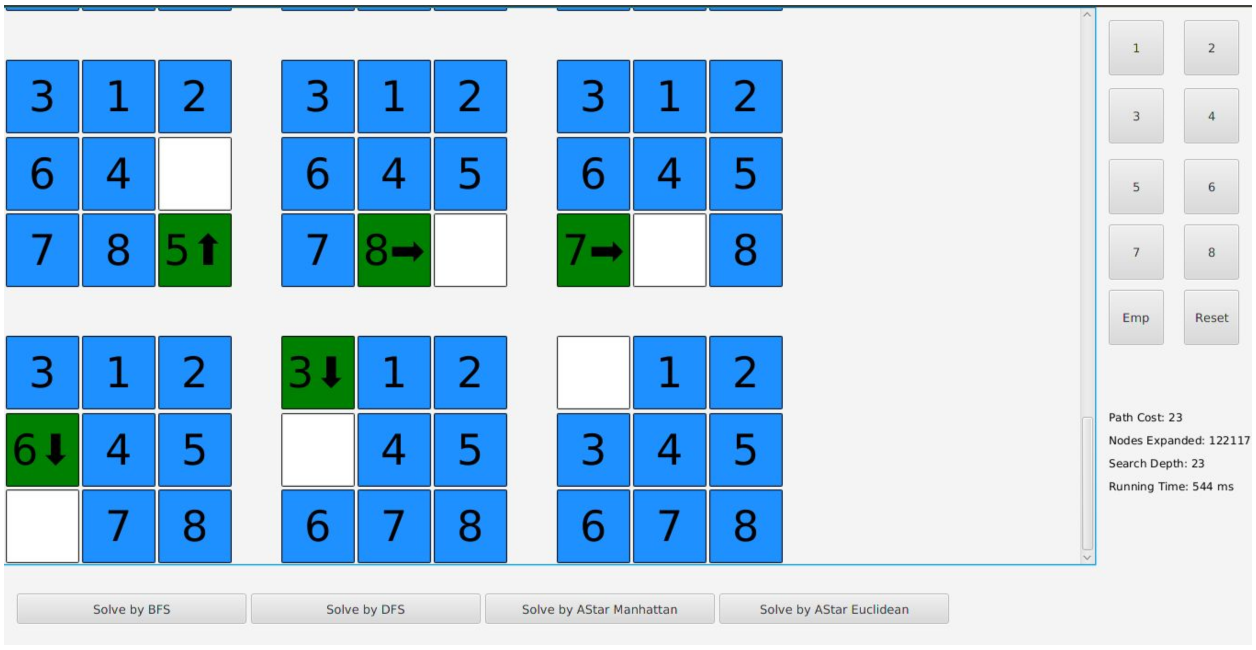
Max Depth: 23

Nodes Expanded: 122117

Running Time: 544ms



Path Cost: 23
Nodes Expanded: 122117
Search Depth: 23
Running Time: 544 ms



Path Cost: 23
Nodes Expanded: 122117
Search Depth: 23
Running Time: 544 ms

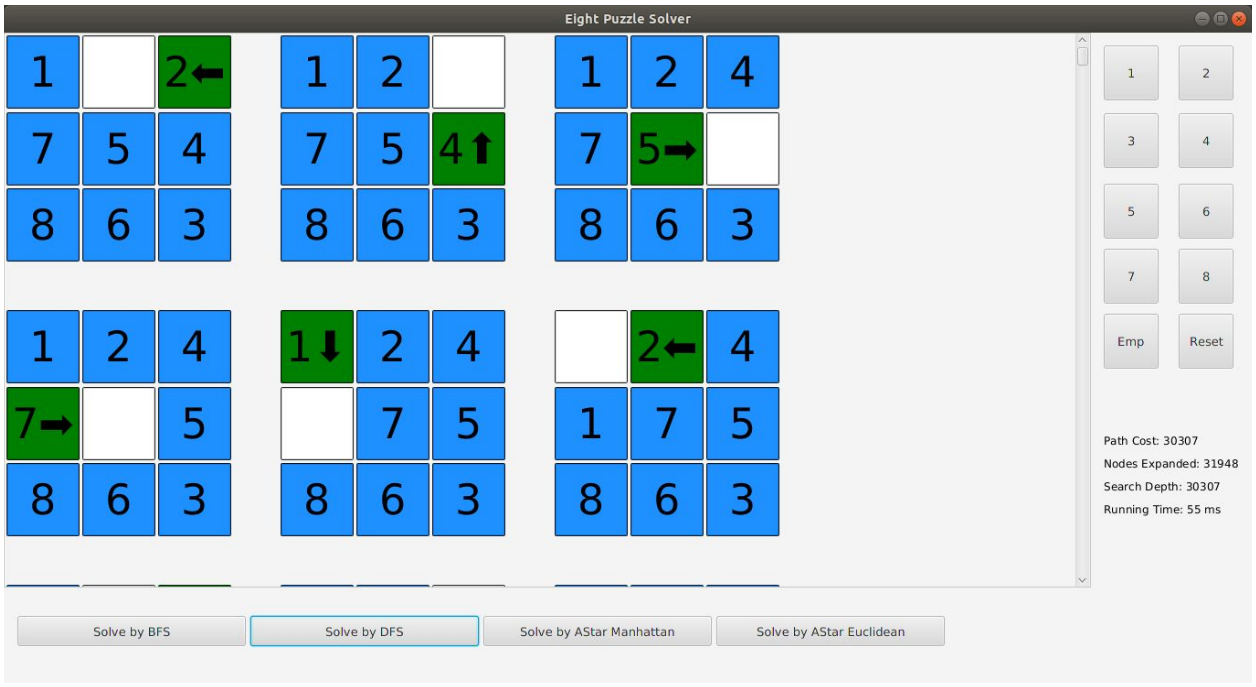
Solving by DFS

Path Cost: 30307

Max Depth: 30307

Nodes Expanded: 31948

Running Time: 55ms



Path Cost: 30307
Nodes Expanded: 31948
Search Depth: 30307
Running Time: 55 ms

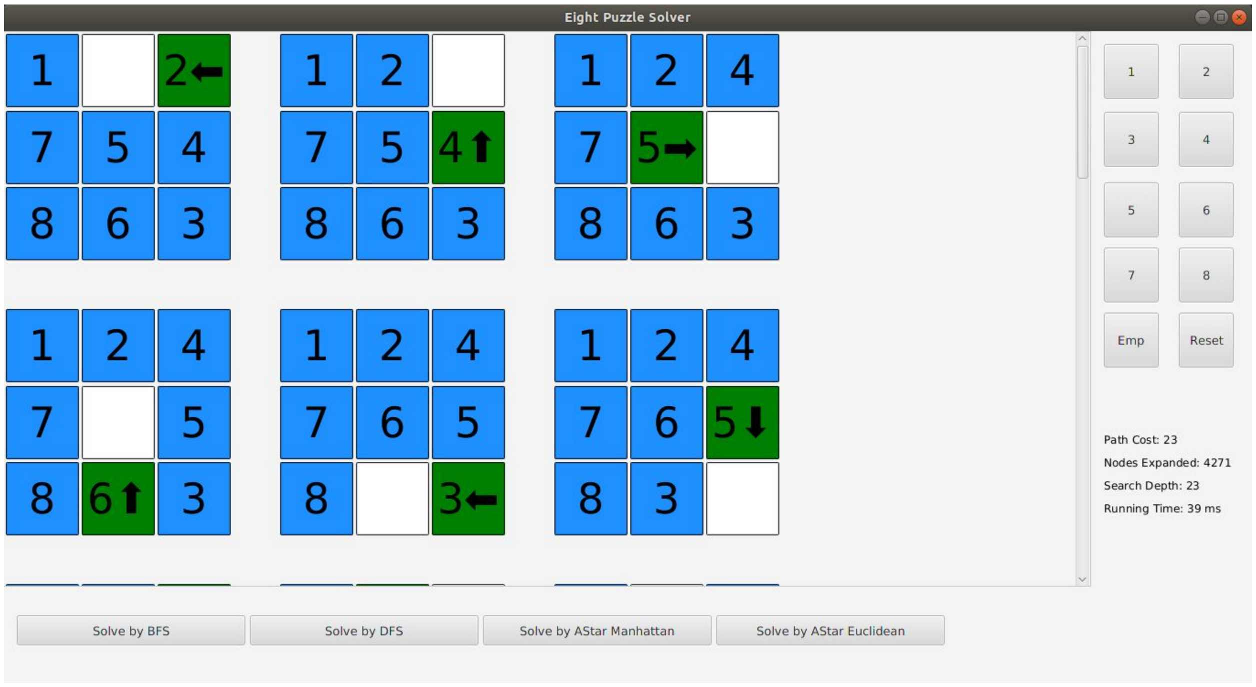
Solving by AStar Manhattan

Path Cost: 23

Max Depth: 23

Nodes Expanded: 4271

Running Time: 39ms



Path Cost: 23
Nodes Expanded: 4271
Search Depth: 23
Running Time: 39 ms

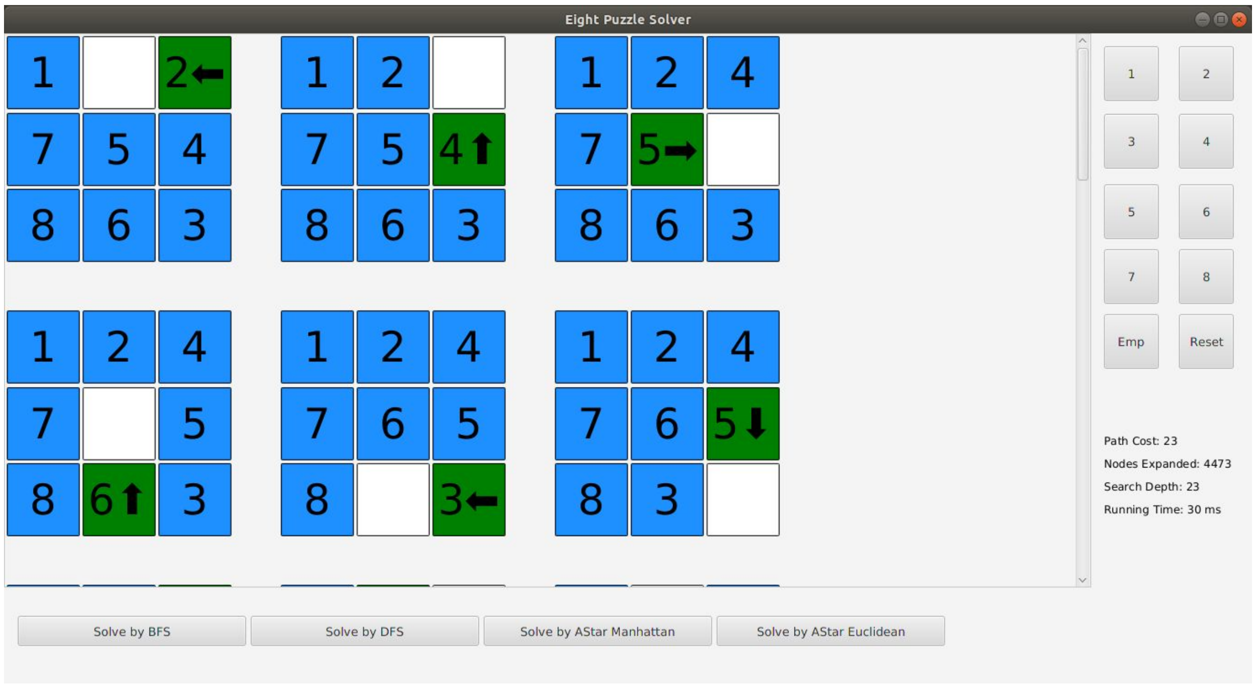
Solving by AStar Euclidean

Path Cost: 23

Max Depth: 23

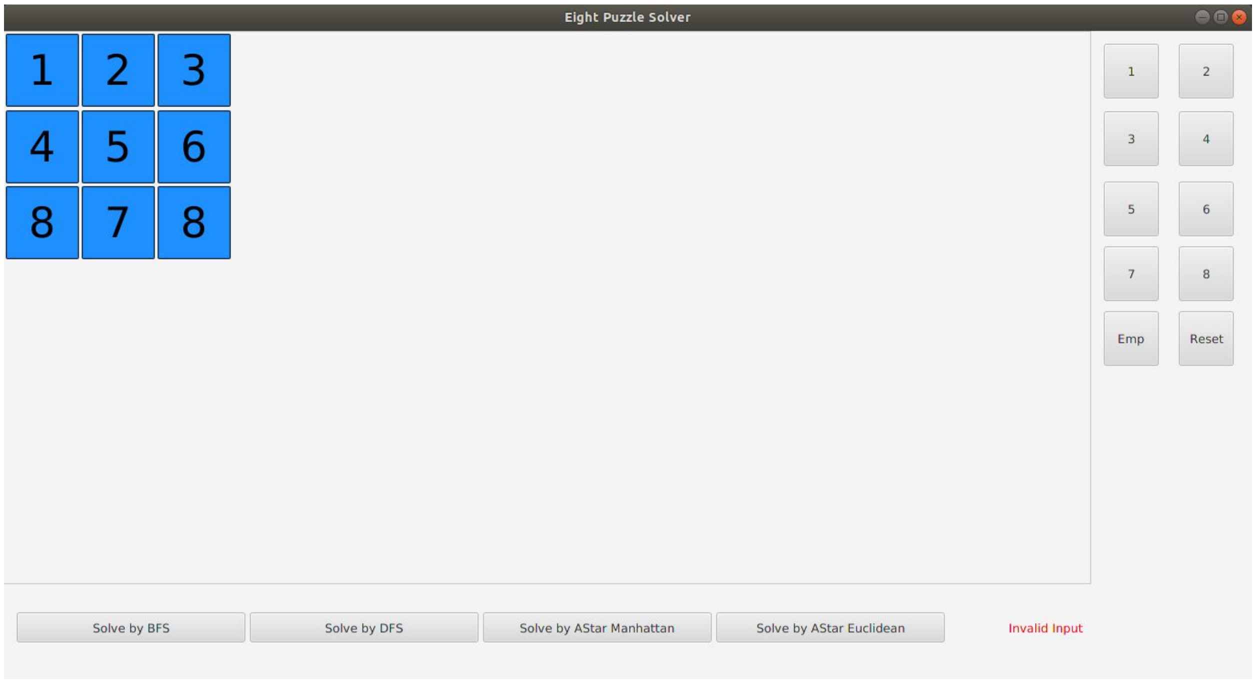
Nodes Expanded: 4473

Running Time: 30ms



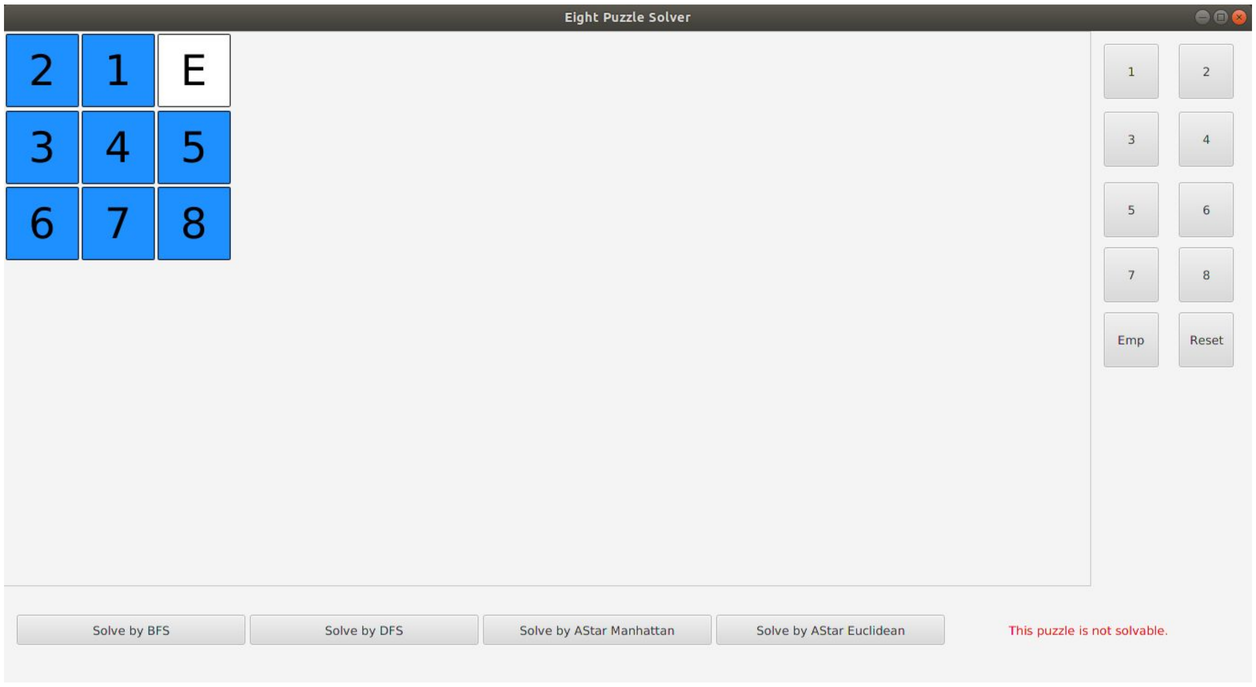
Path Cost: 23
Nodes Expanded: 4473
Search Depth: 23
Running Time: 30 ms

Detection of Invalid Input



Invalid Input

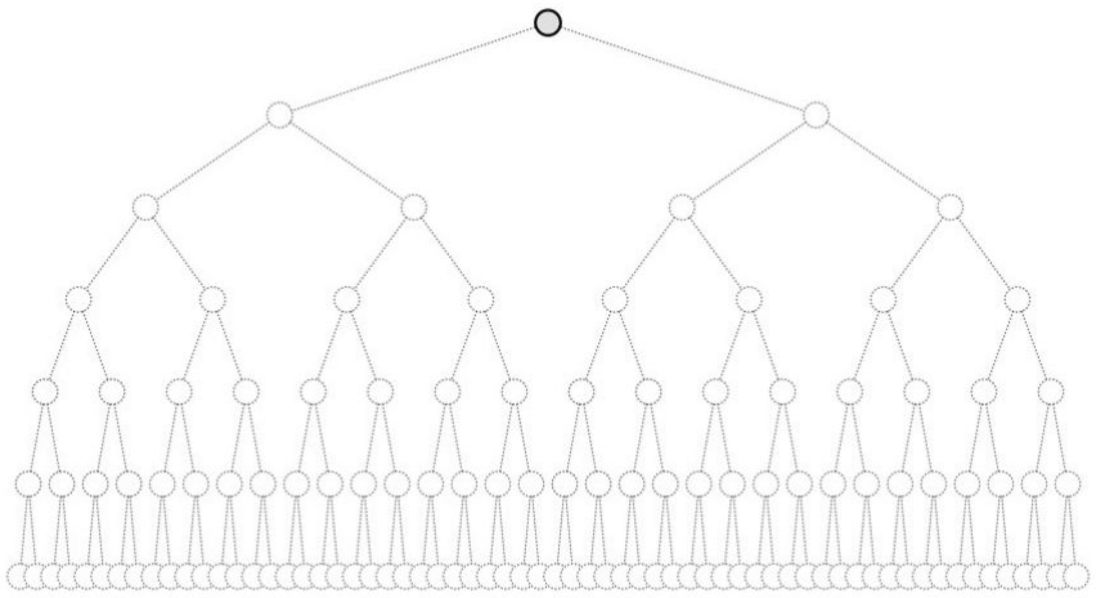
Detection of not solvable puzzles



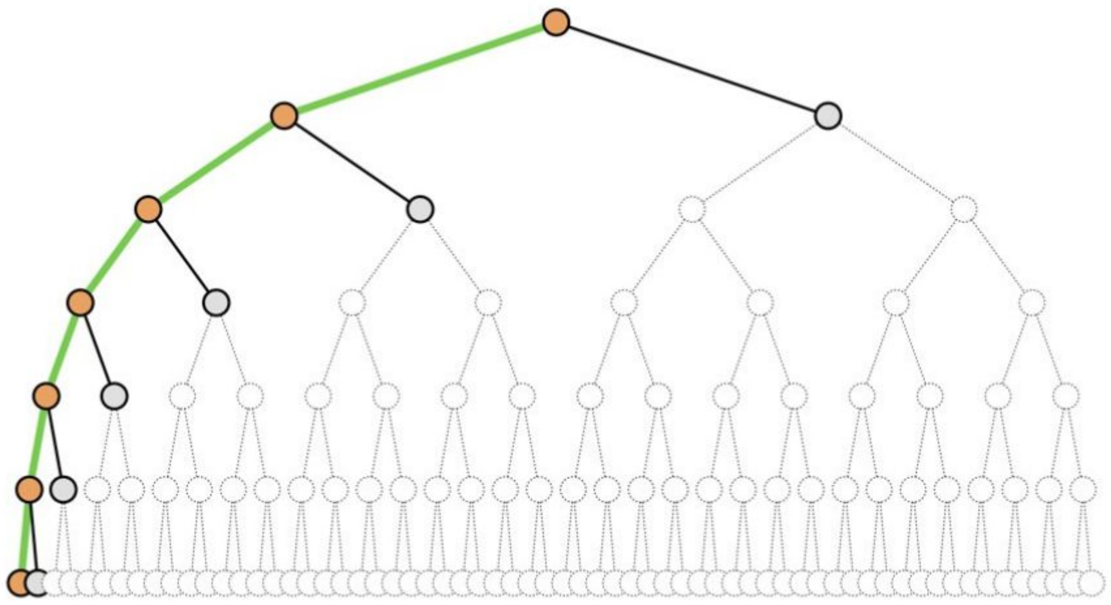
This puzzle is not solvable.

How Algorithms Work

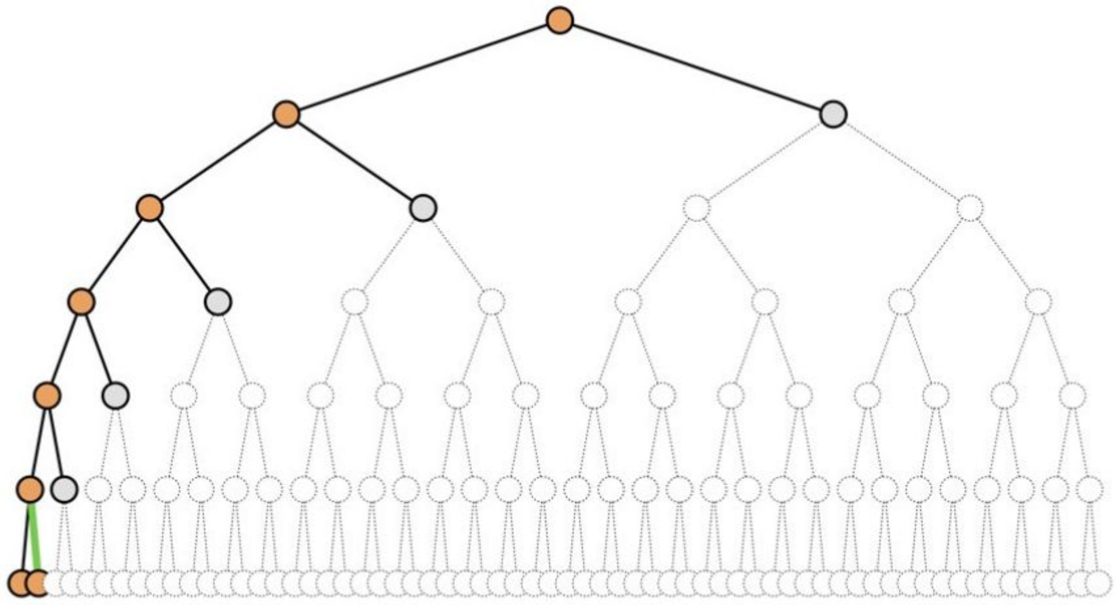
1. Depth First Search (DFS)



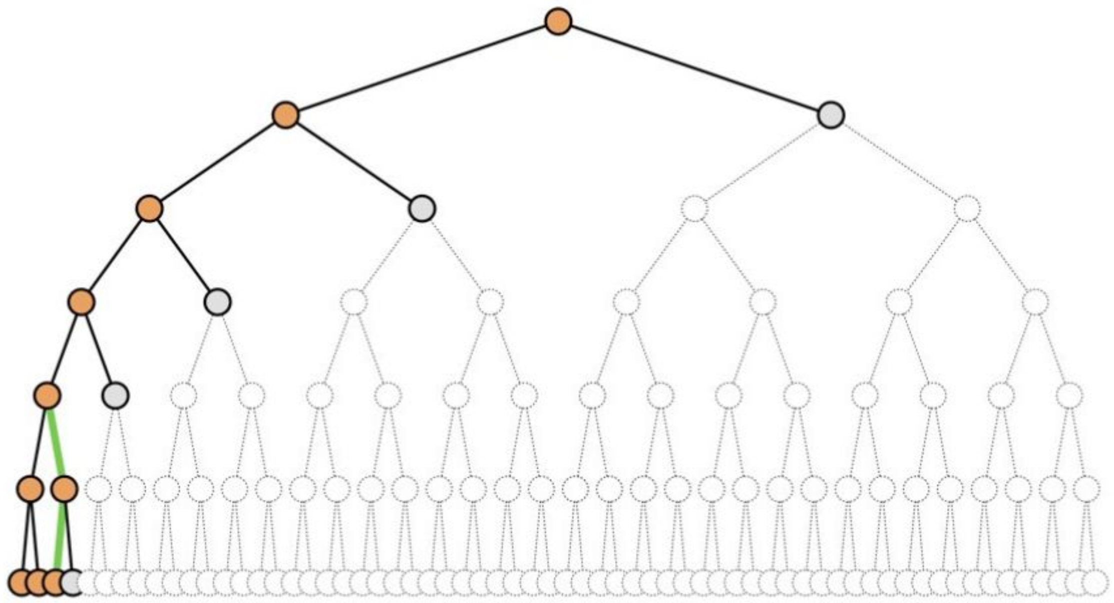
Searches branch by branch ...
... with each branch pursued to maximum depth.



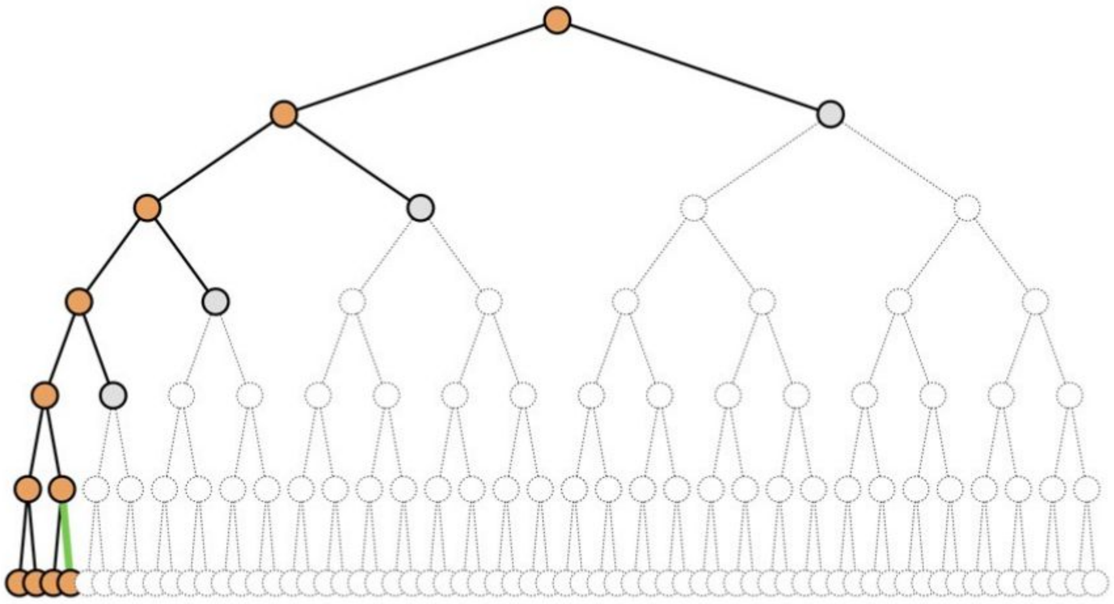
Searches branch by branch ...
... with each branch pursued to maximum depth.



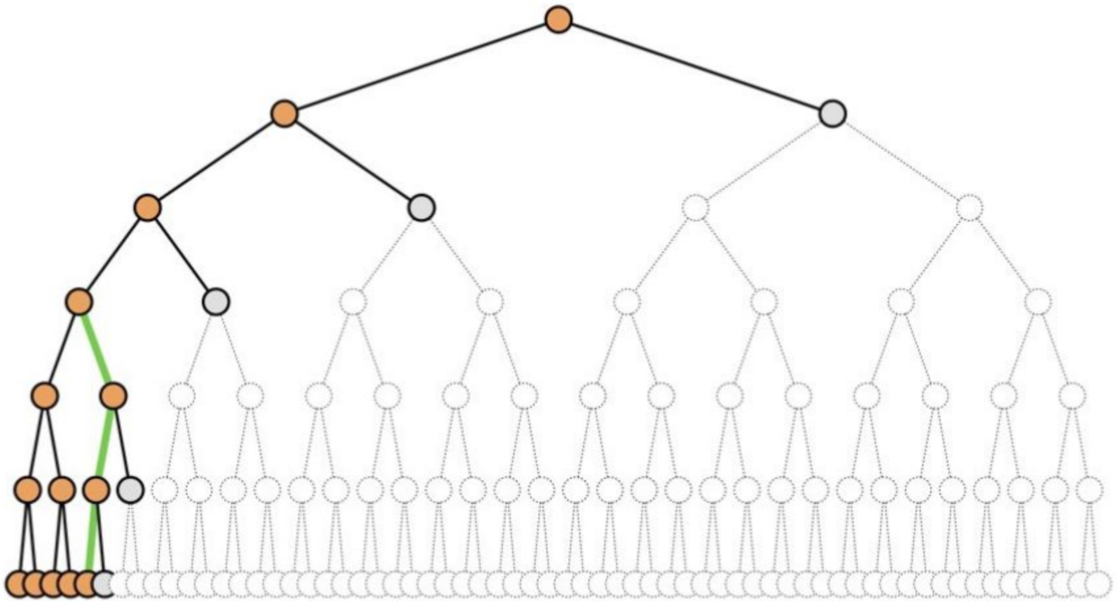
Searches branch by branch ...
... with each branch pursued to maximum depth.



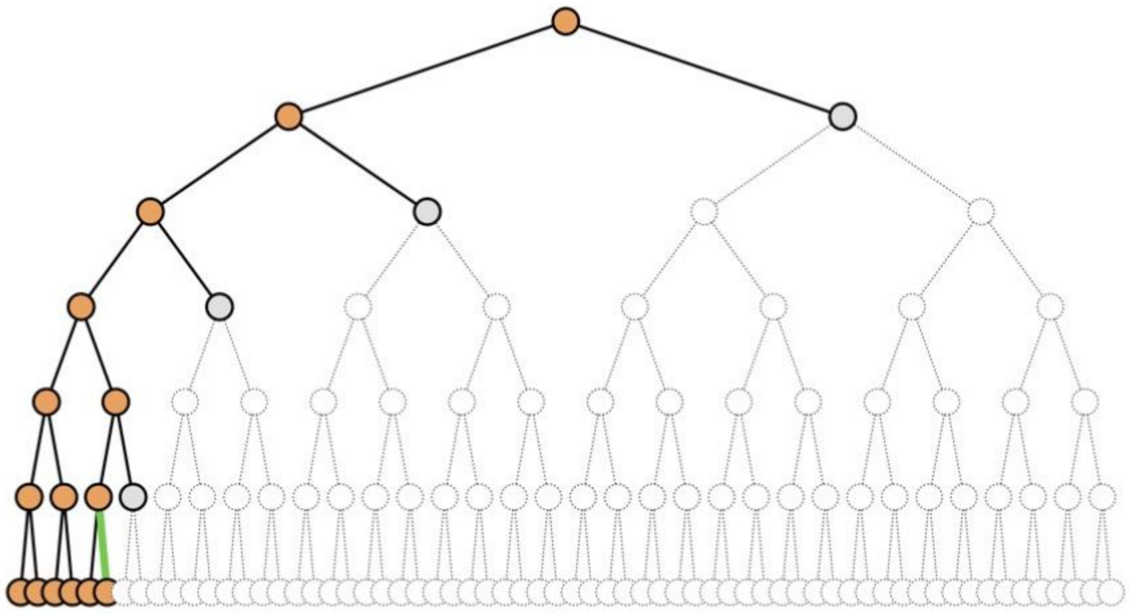
Searches branch by branch ...
... with each branch pursued to maximum depth.



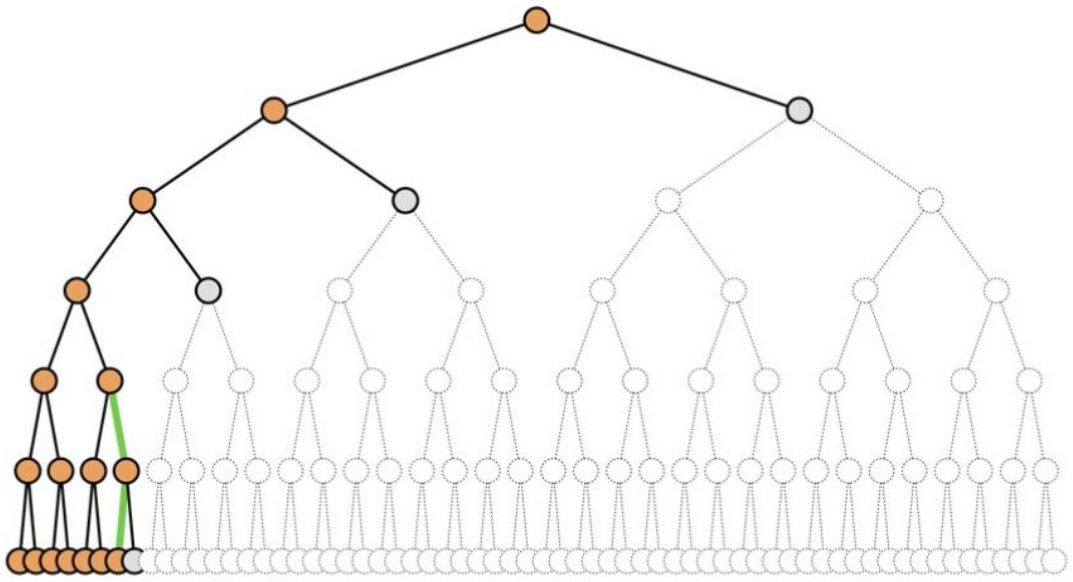
Searches branch by branch ...
... with each branch pursued to maximum depth.



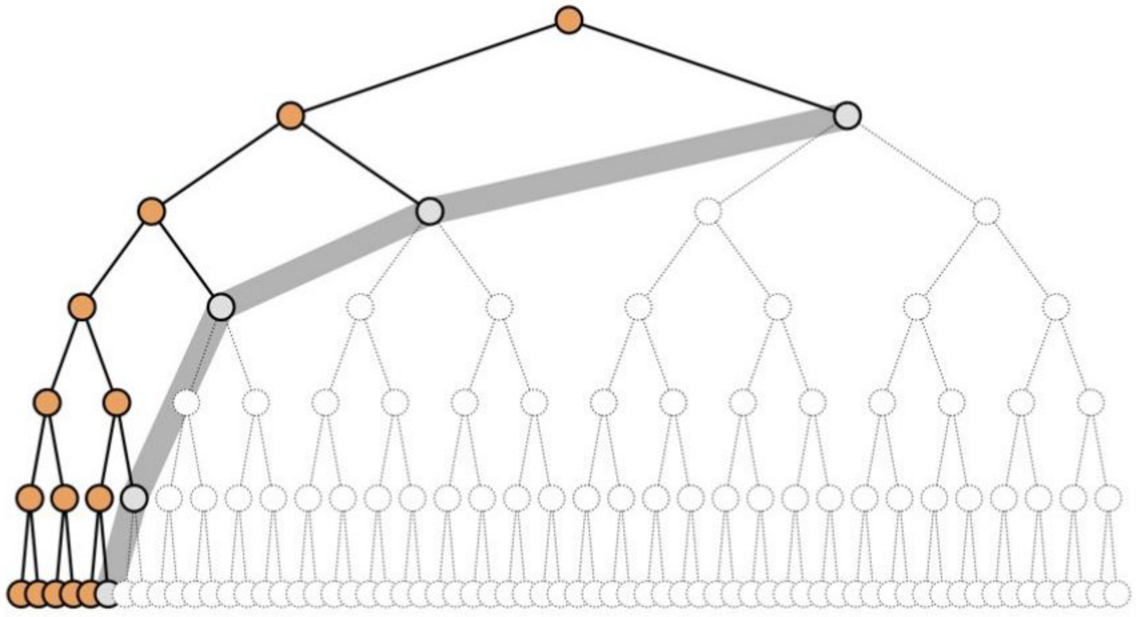
Searches branch by branch ...
... with each branch pursued to maximum depth.



Searches branch by branch ...
... with each branch pursued to maximum depth.



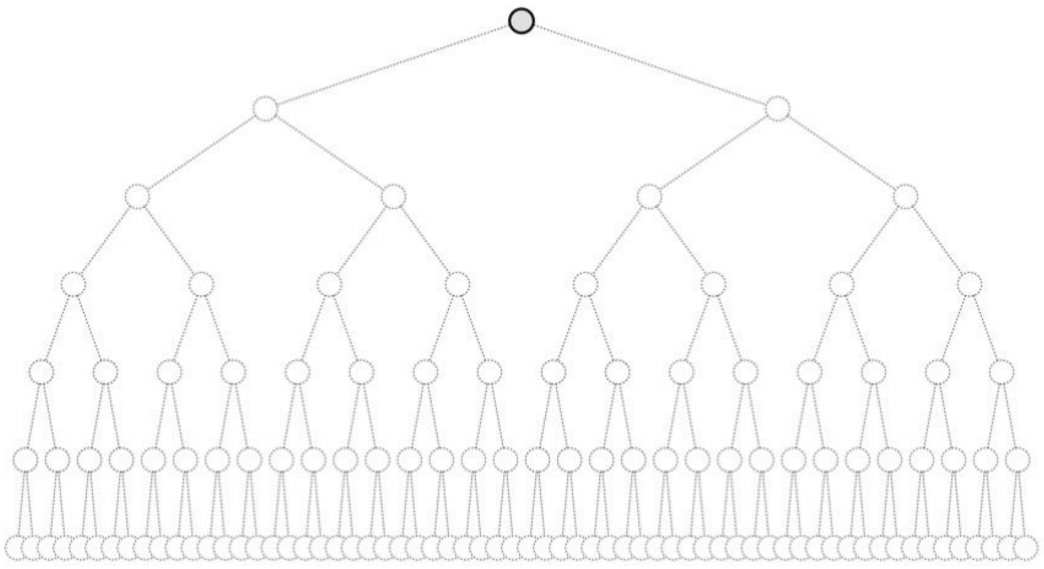
Searches branch by branch ...
... with each branch pursued to maximum depth.



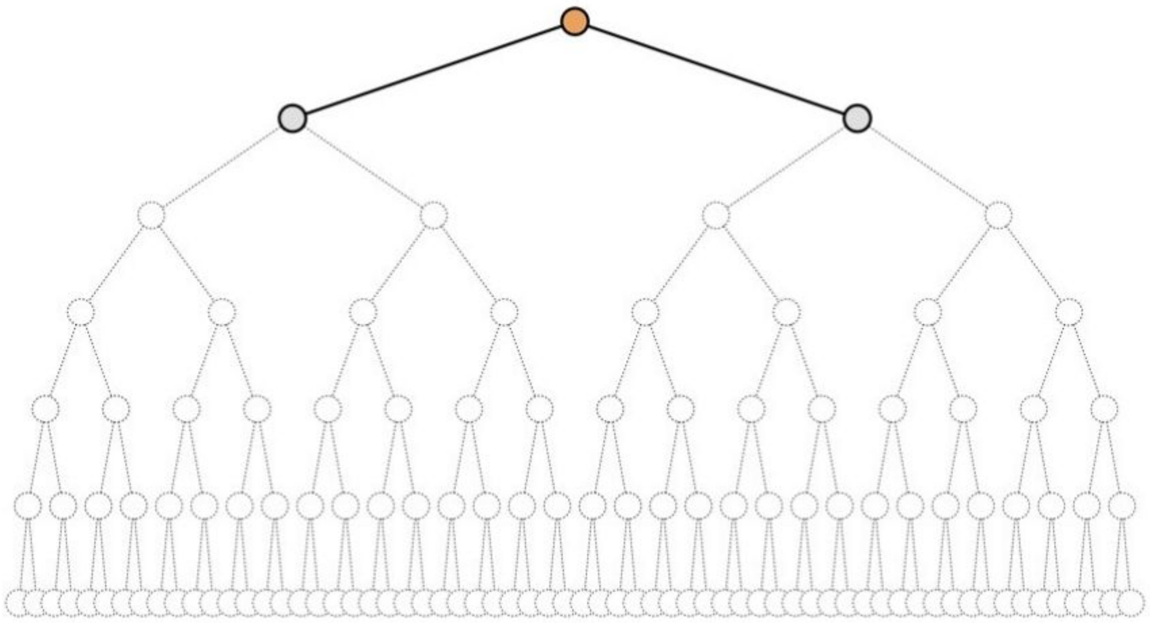
The frontier consists of unexplored siblings of all ancestors.
Search proceeds by exhausting one branch at a time.

2. Breadth First Search (BFS)

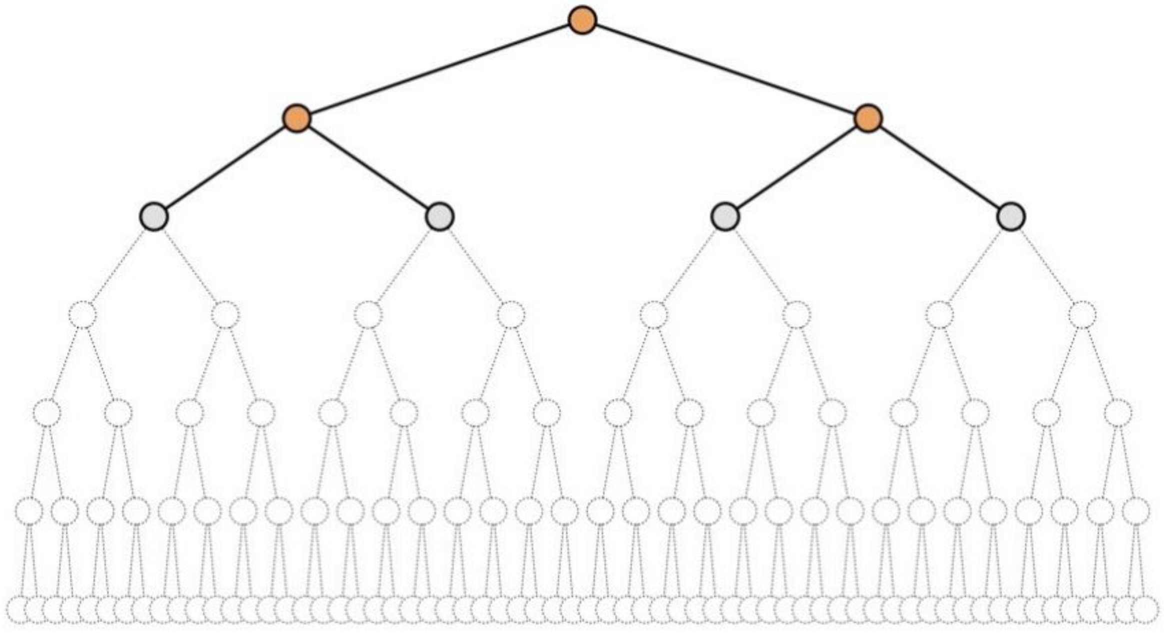
Used Data Structure: First In First Out Queue (FIFO)



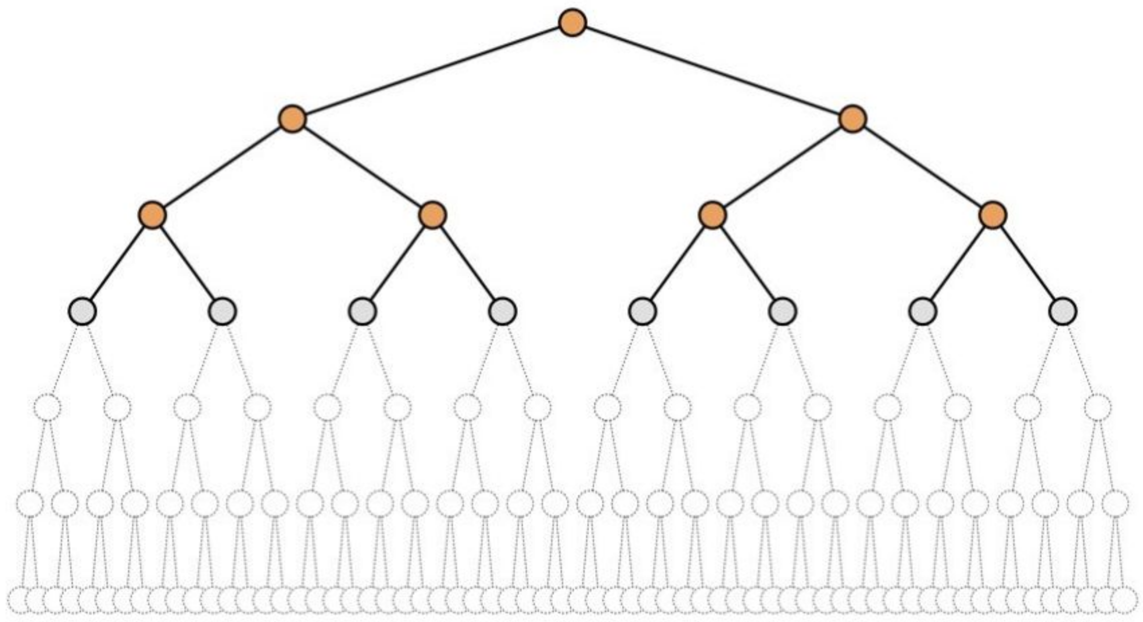
Searches layer by layer ...
... with each layer organized by node depth.



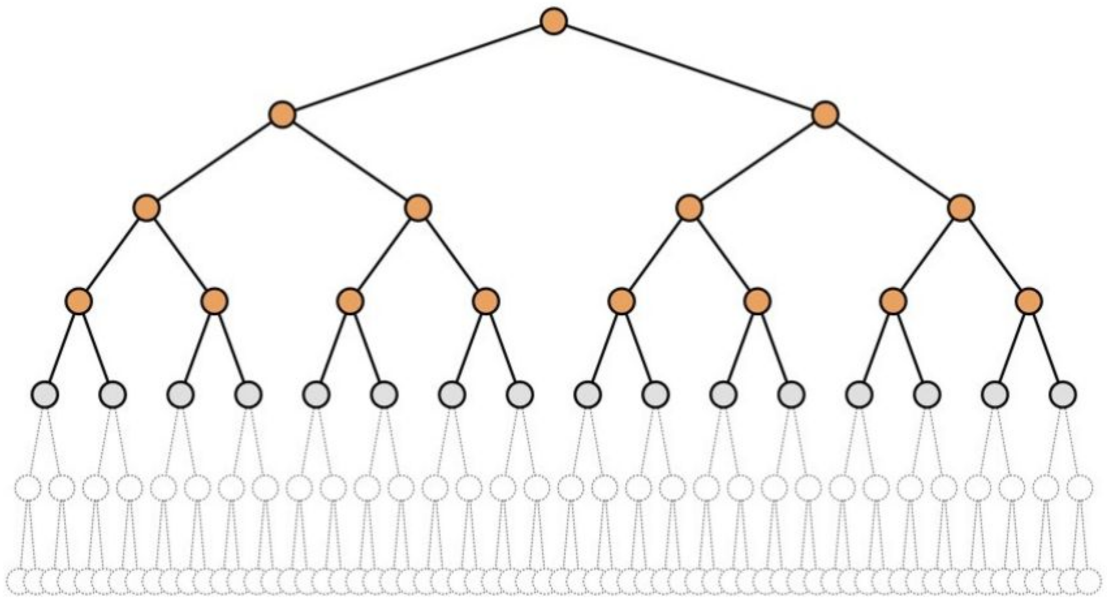
Searches layer by layer ...
... with each layer organized by node depth.



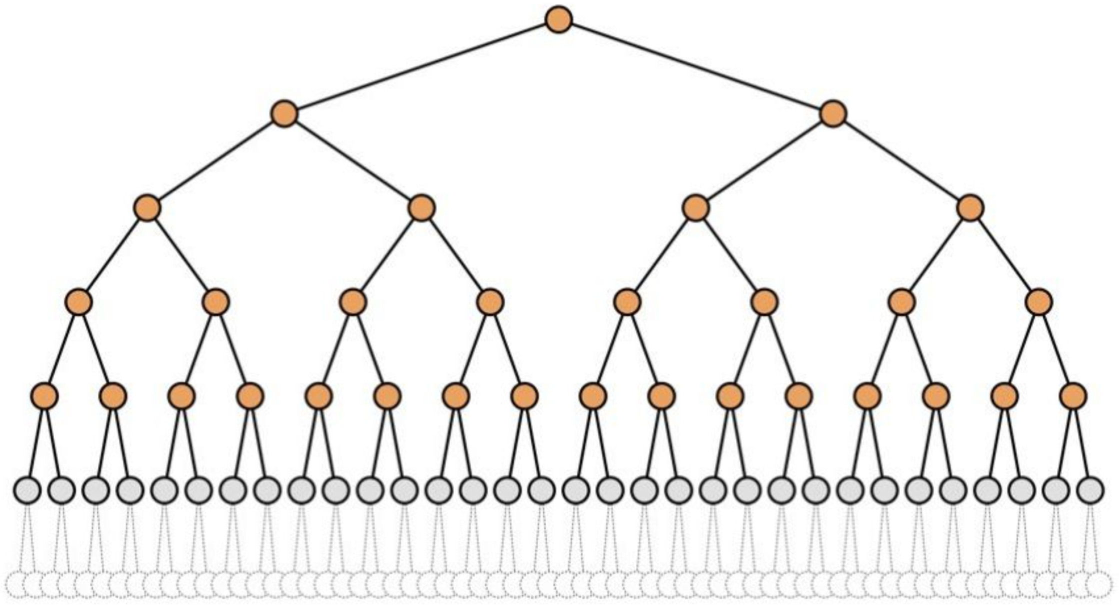
Searches layer by layer ...
... with each layer organized by node depth.



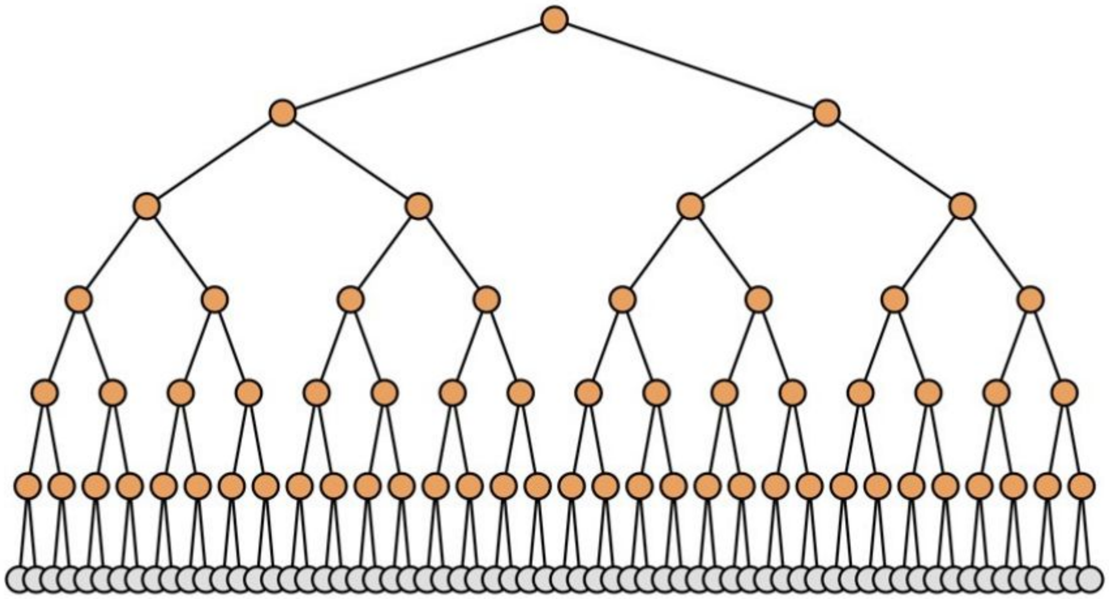
Searches layer by layer ...
... with each layer organized by node depth.



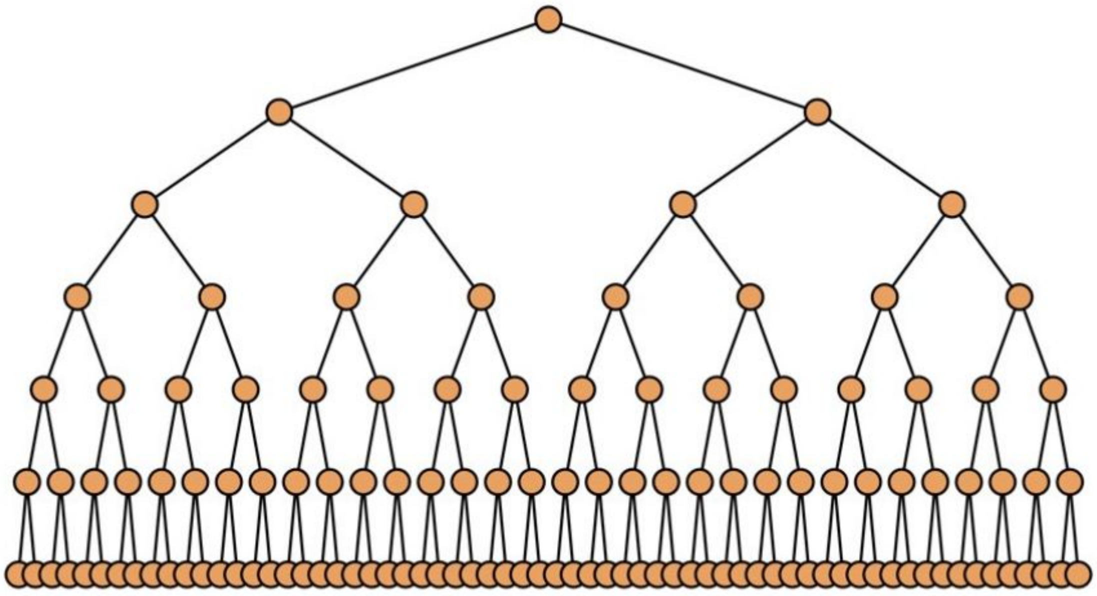
Searches layer by layer ...
... with each layer organized by node depth.



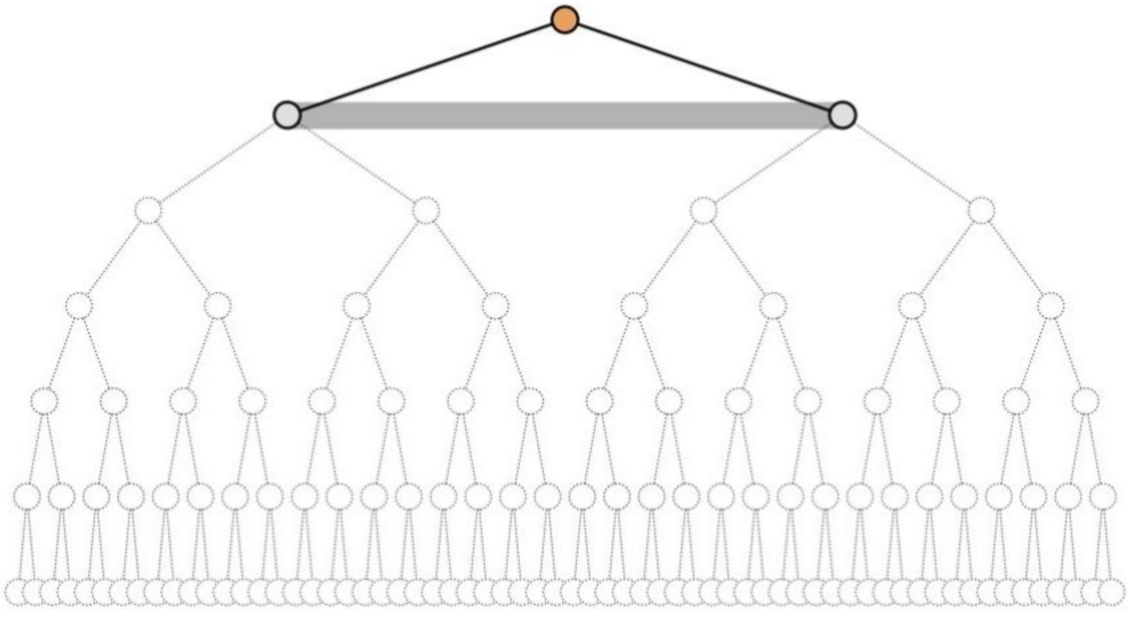
Searches layer by layer ...
... with each layer organized by node depth.



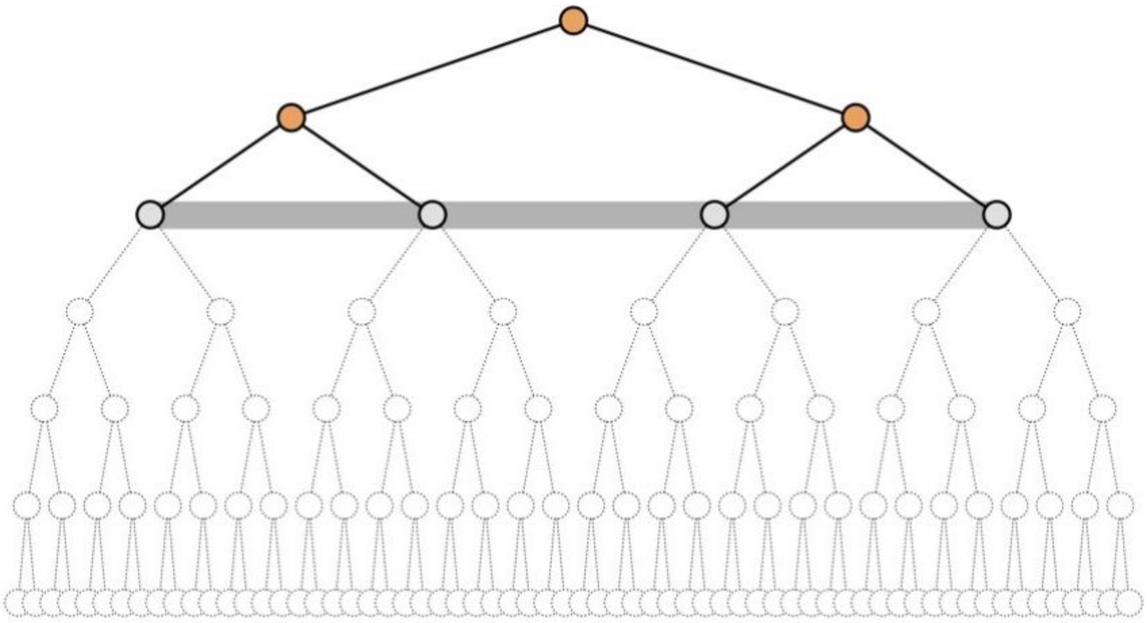
Searches layer by layer ...
... with each layer organized by node depth.



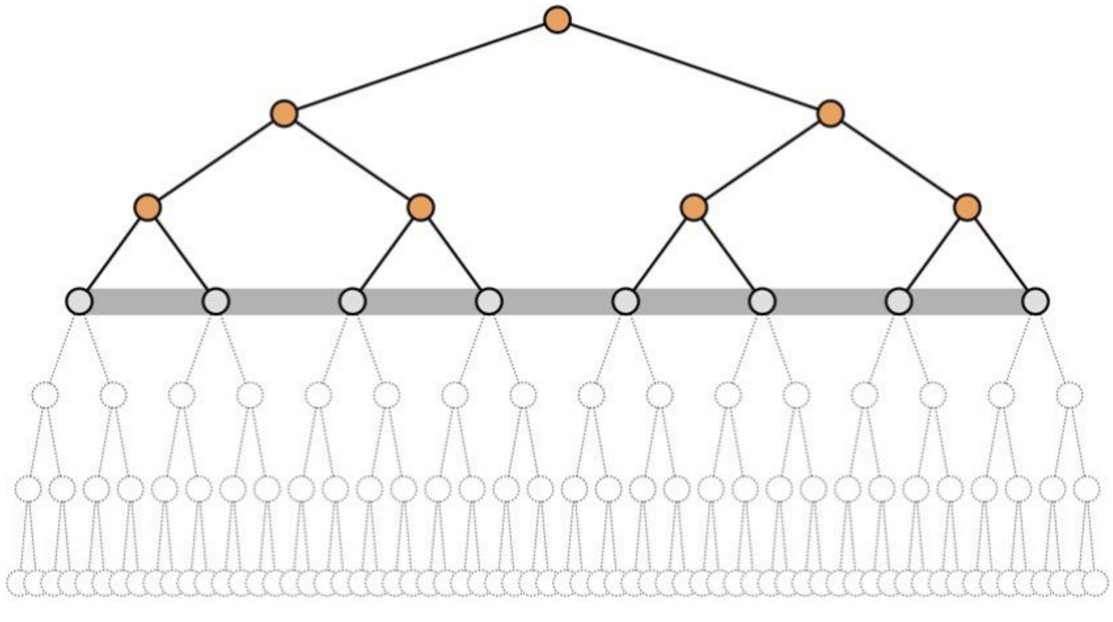
Searches layer by layer ...
... with each layer organized by node depth.



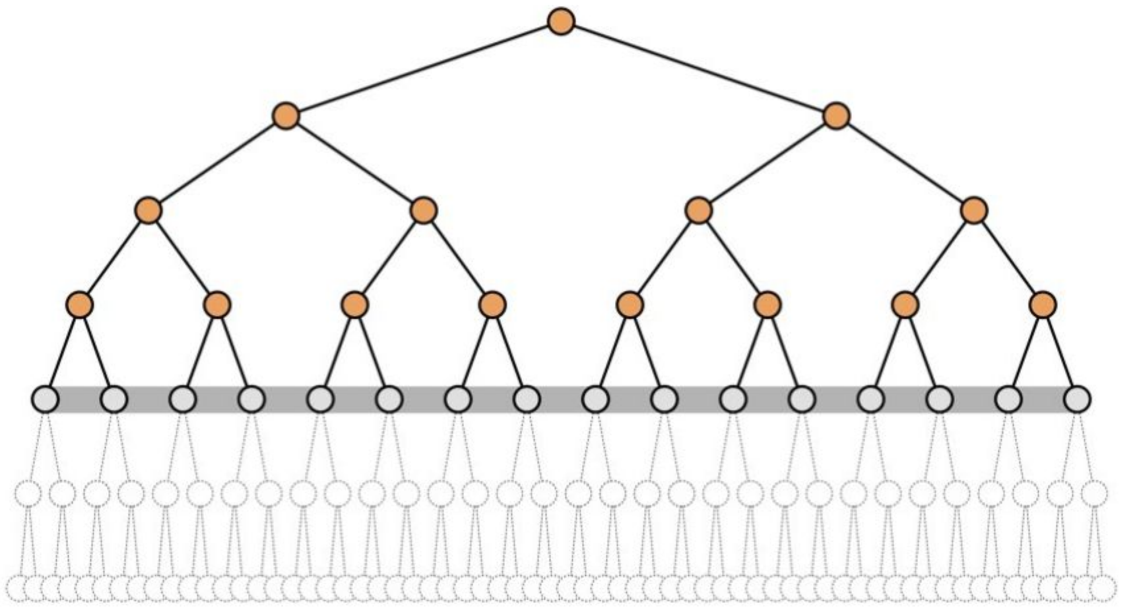
The frontier consists of nodes of similar depth (horizontal).
Search proceeds by exhausting one layer at a time.



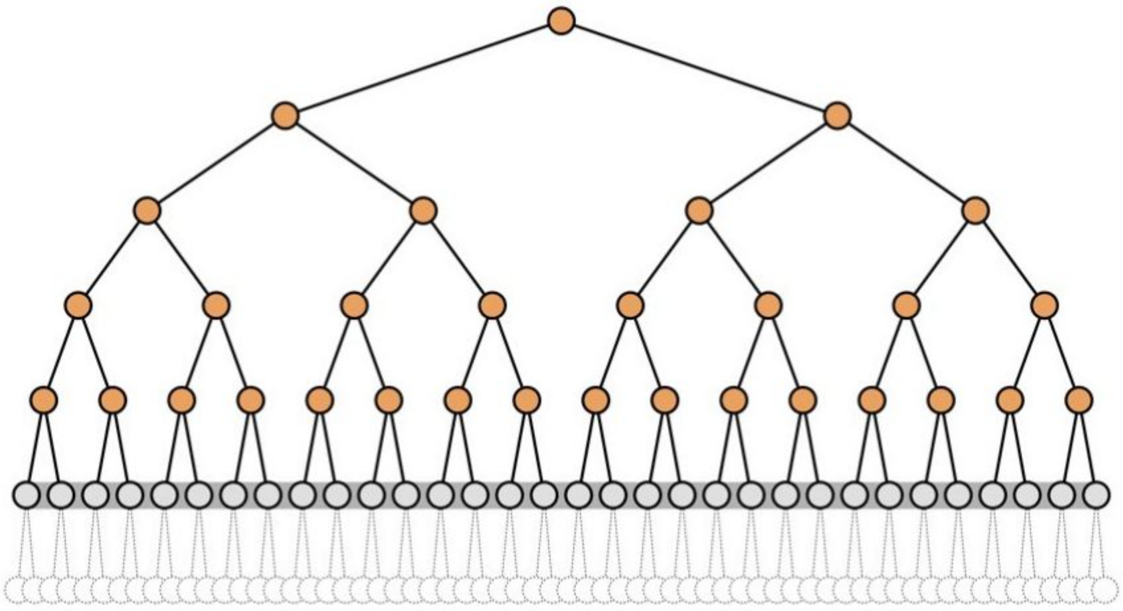
The frontier consists of nodes of similar depth (horizontal).
Search proceeds by exhausting one layer at a time.



The frontier consists of nodes of similar depth (horizontal).
Search proceeds by exhausting one layer at a time.

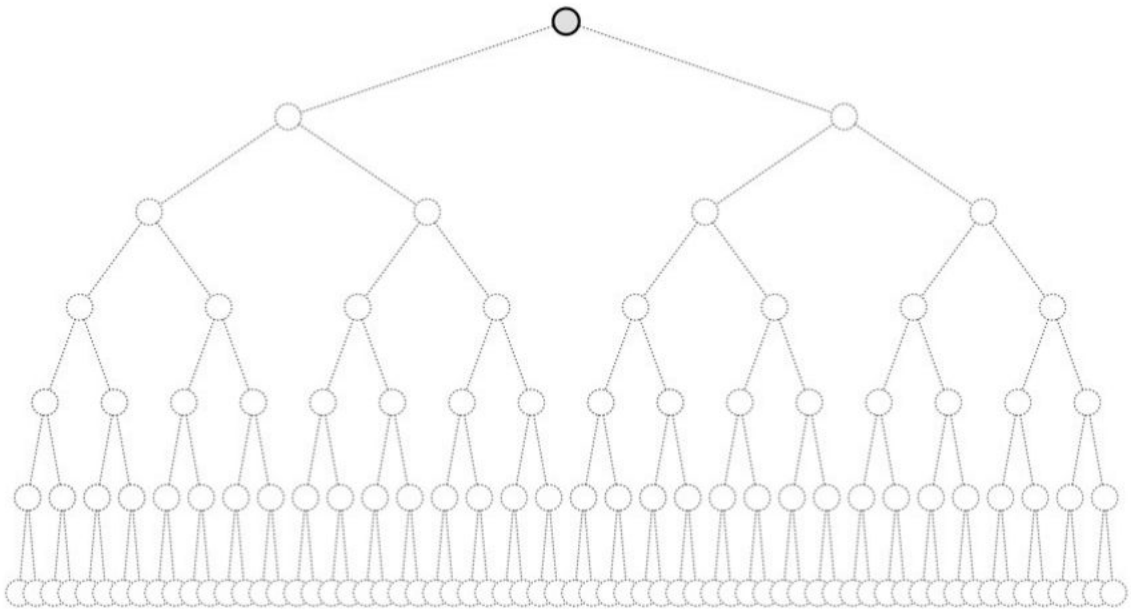


The frontier consists of nodes of similar depth (horizontal). Search proceeds by exhausting one layer at a time.

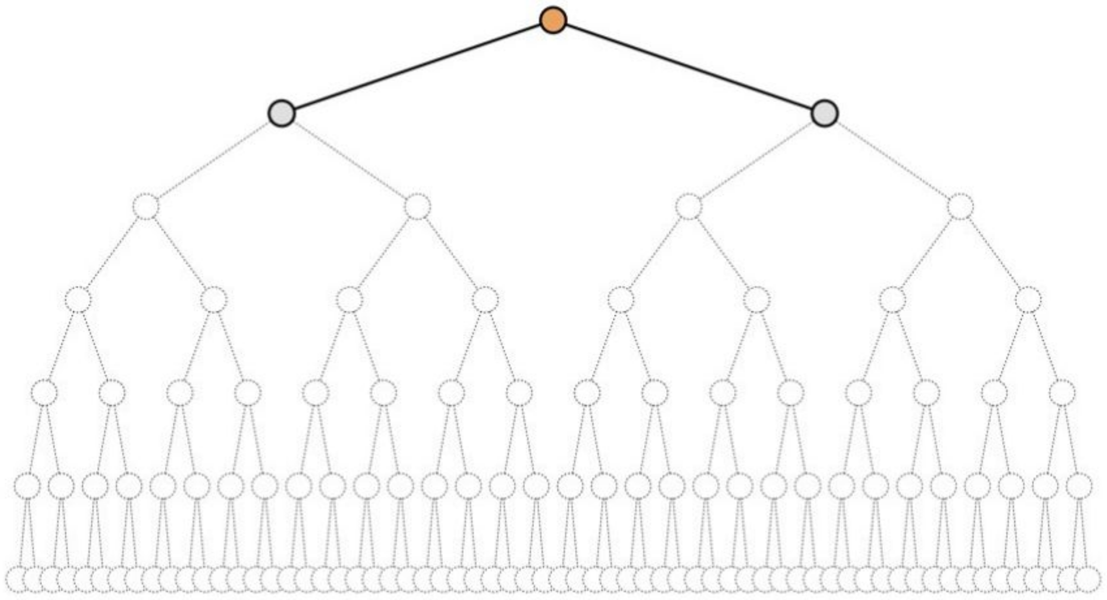


The frontier consists of nodes of similar depth (horizontal). Search proceeds by exhausting one layer at a time.

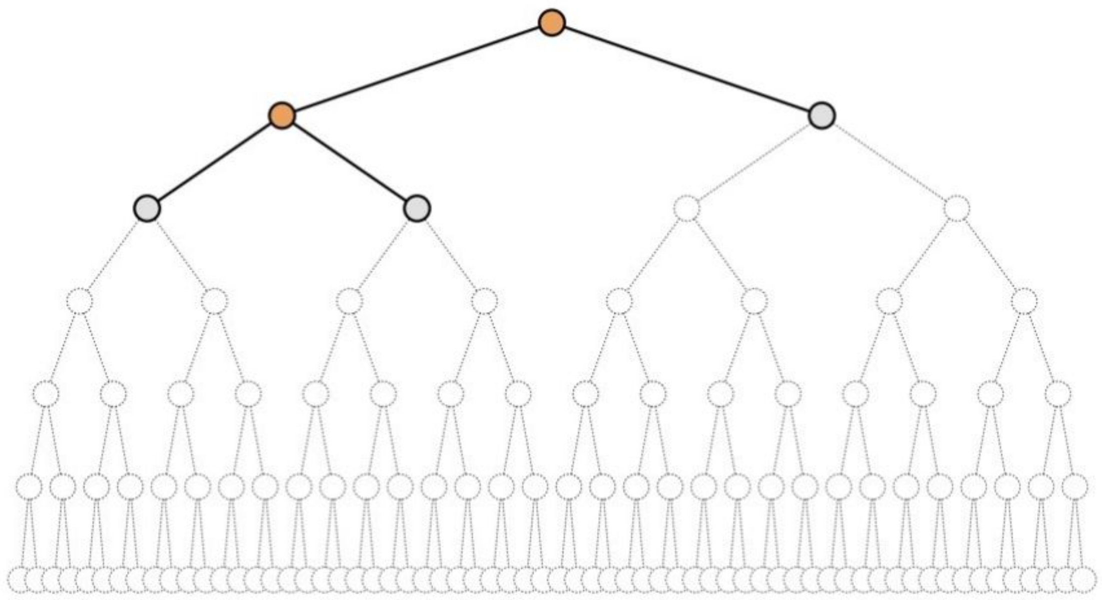
3. A Star Search (A*)



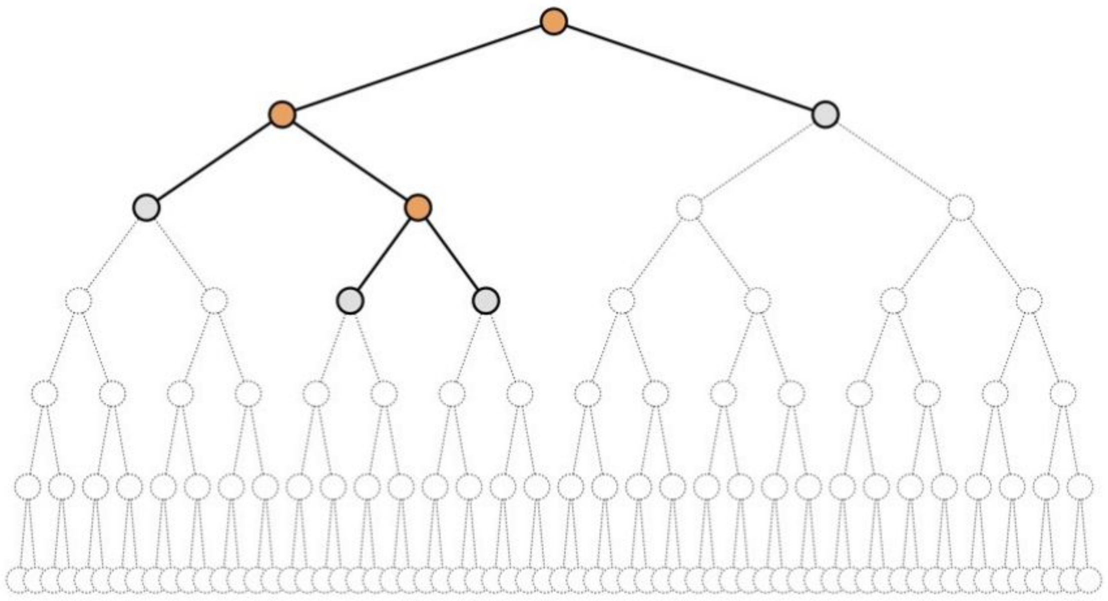
Searches layer by layer ...
... with each layer organized by path cost.



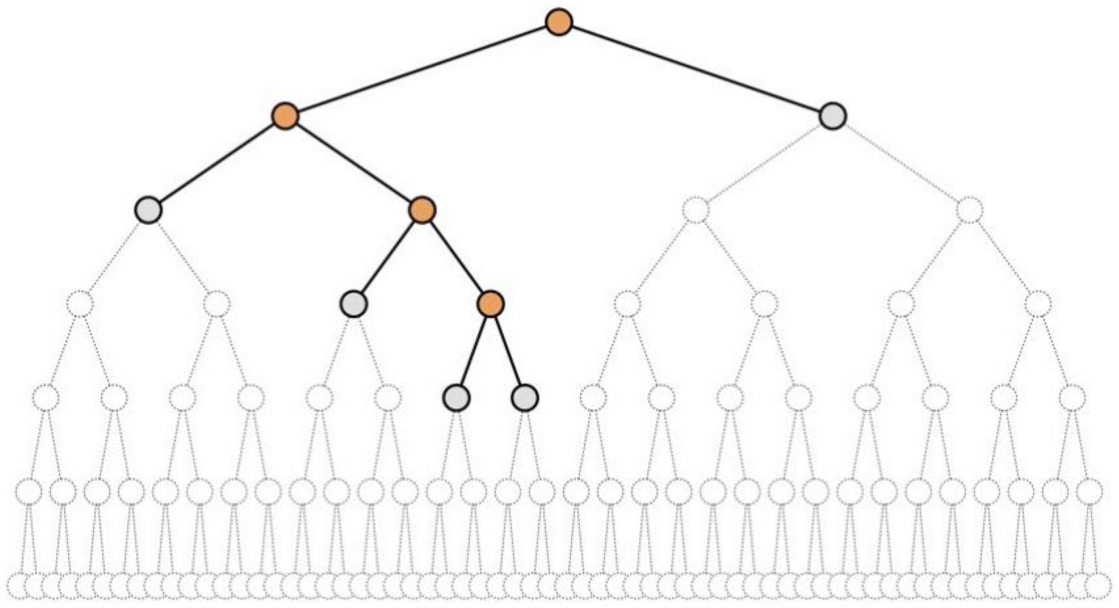
Searches layer by layer ...
... with each layer organized by path cost.



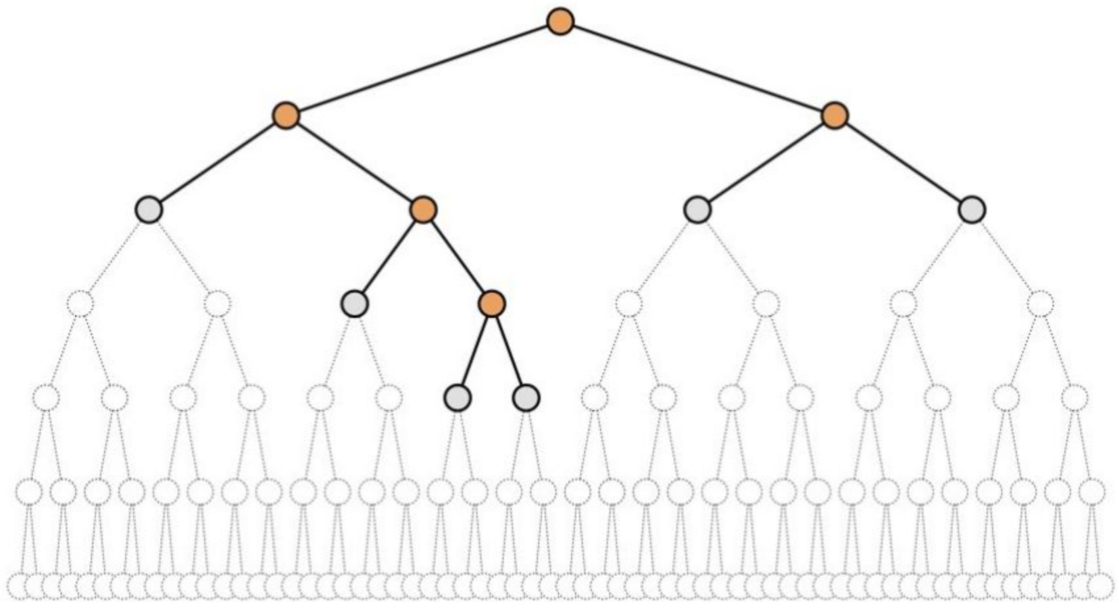
Searches layer by layer ...
... with each layer organized by path cost.



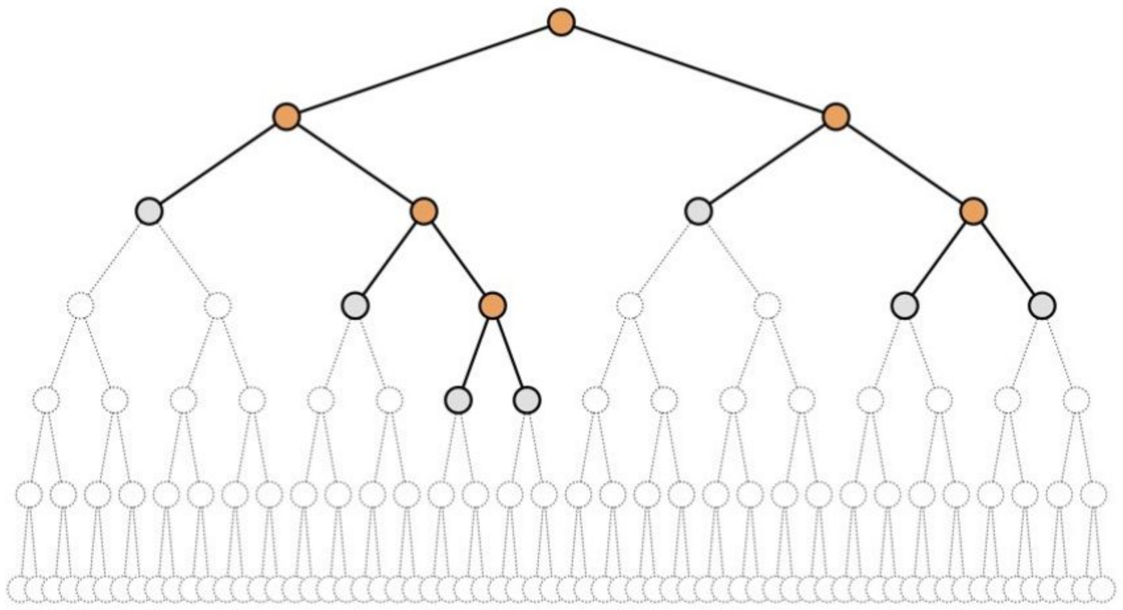
Searches layer by layer ...
... with each layer organized by path cost.



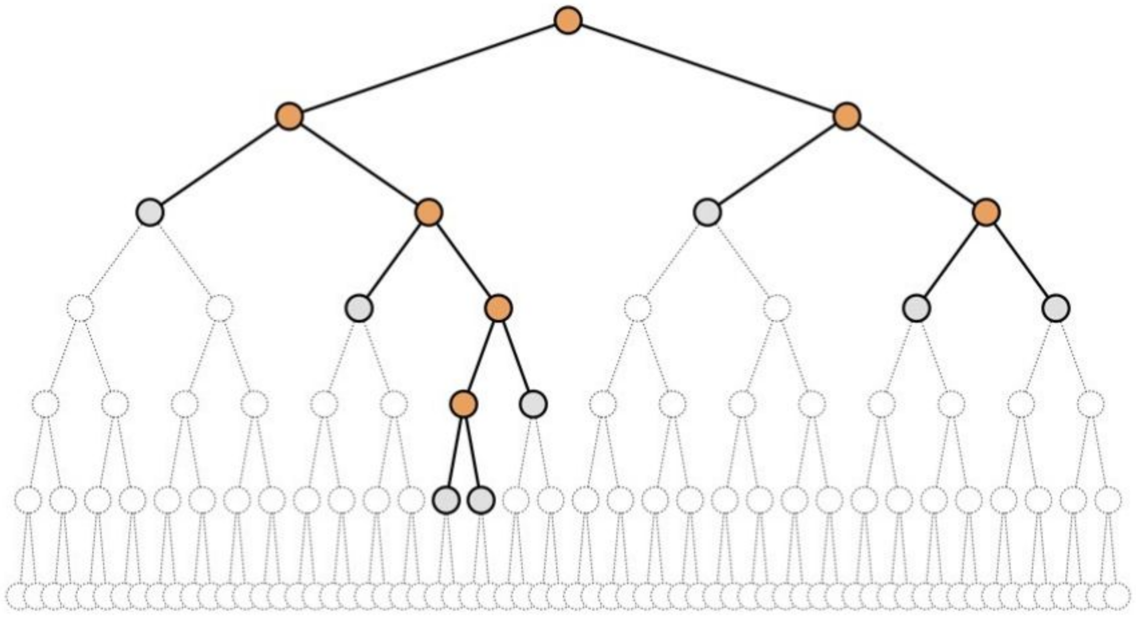
Searches layer by layer ...
... with each layer organized by path cost.



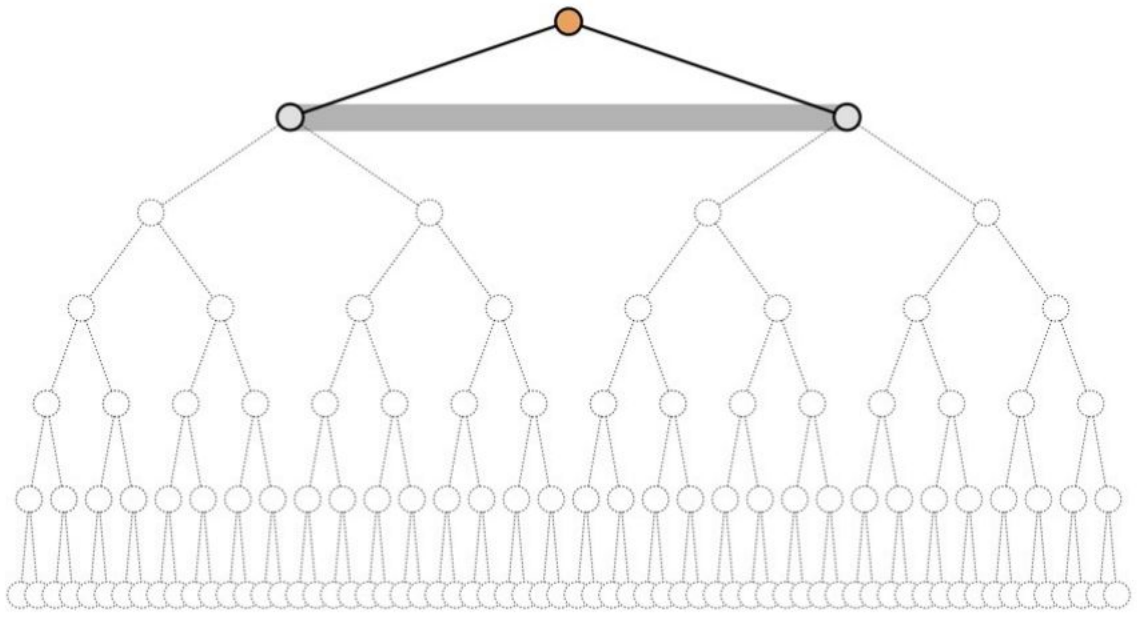
Searches layer by layer ...
... with each layer organized by path cost.



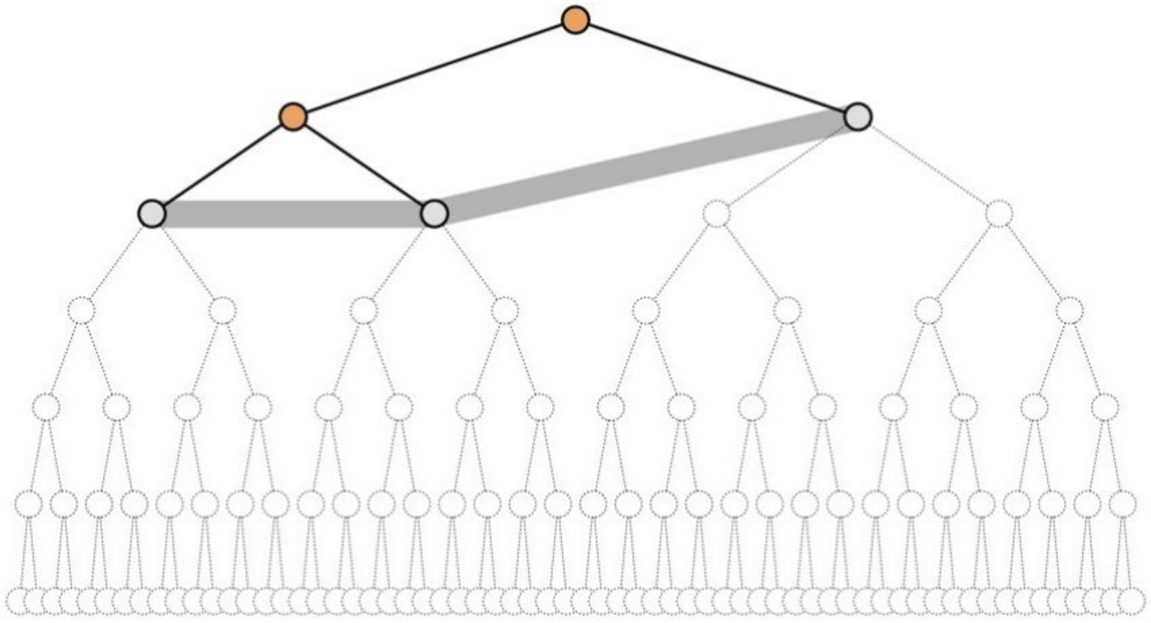
Searches layer by layer ...
... with each layer organized by path cost.



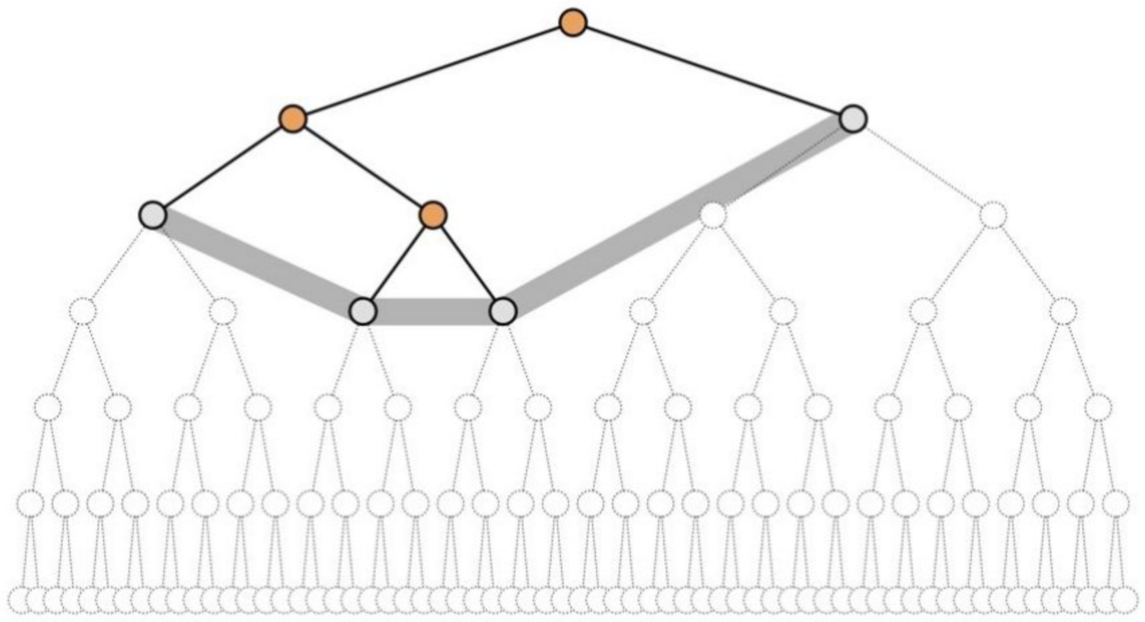
Searches layer by layer ...
... with each layer organized by path cost.



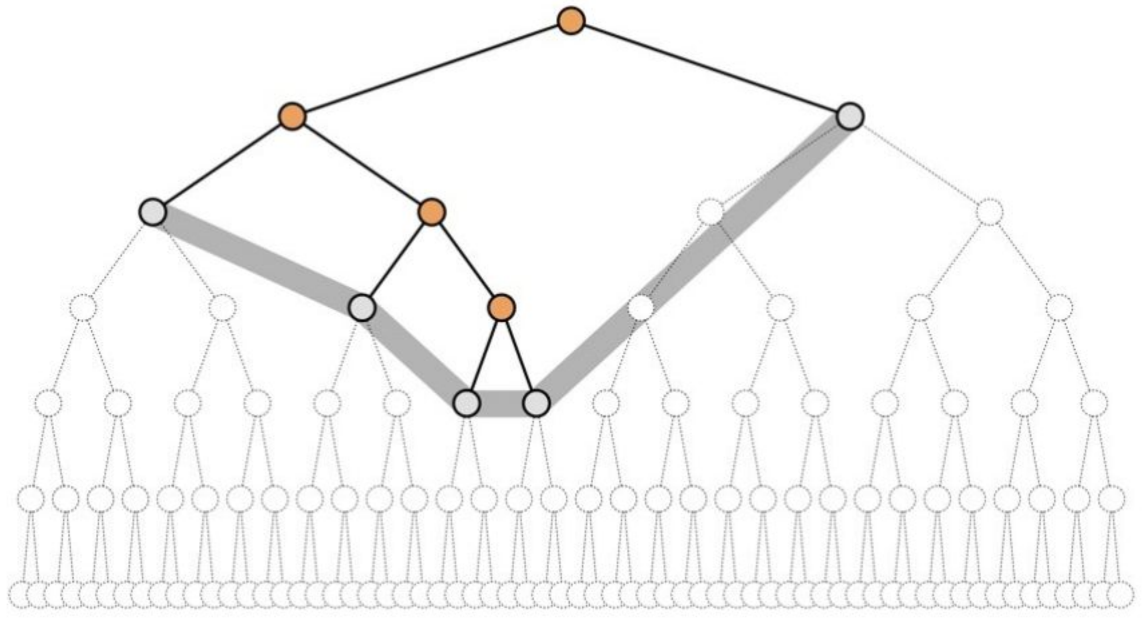
The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.



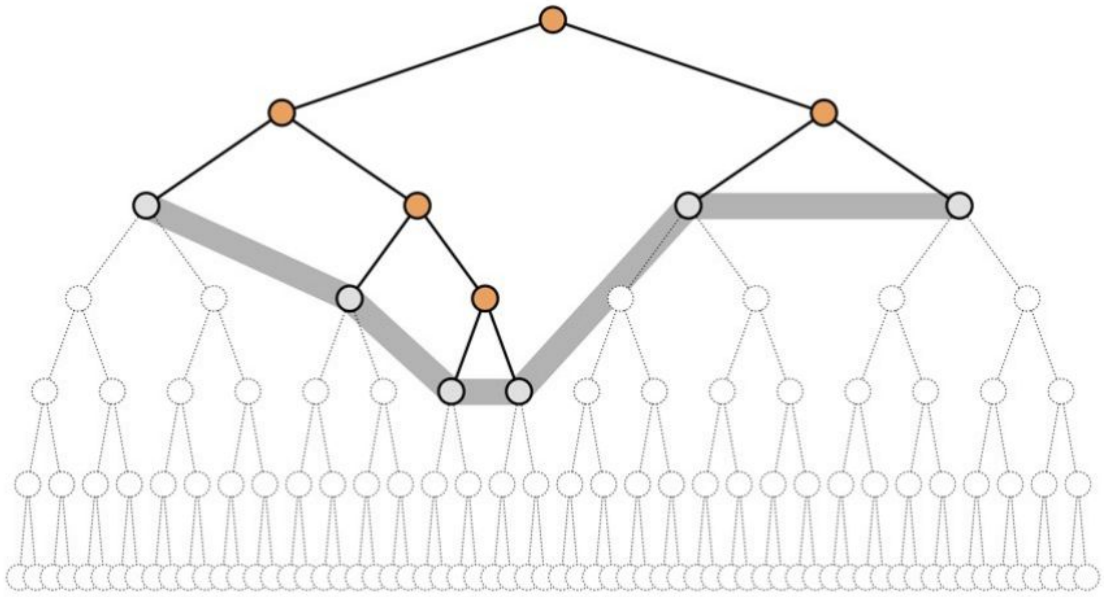
The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.



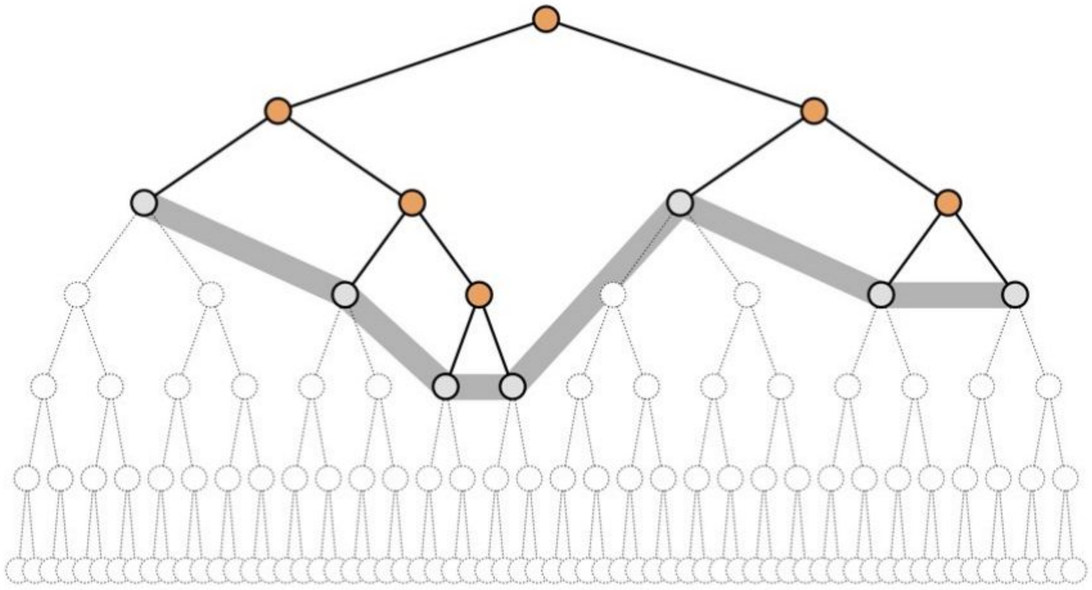
The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.



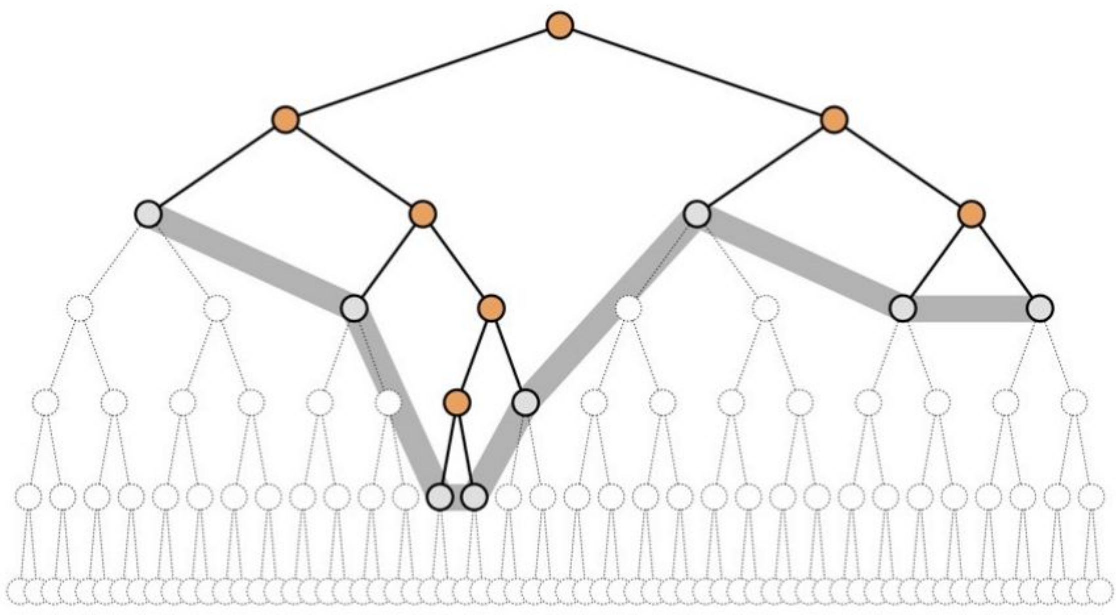
The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.



The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.



The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.



The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.