Admissible heuristics

A good heuristic can be powerful.

Only if it is of a "good quality"

Admissible heuristics

A good heuristic can be powerful.

Only if it is of a "good quality"

A good heuristic must be admissible.

Admissible heuristics

- An **admissible** heuristic never overestimates the cost to reach the goal, that is it is **optimistic**
- A heuristic h is admissible if

$$\forall\ node\ n, h(n) \leq h^*(n)$$

  where h^* is true cost to reach the goal from n.

- h_{SLD} (used as a heuristic in the map example) is admissible because it is by definition the shortest distance between two points.

A* Optimality

If $h(n)$ is admissible, A* using tree search is optimal.

A* Optimality

If $h(n)$ is admissible, A* using tree search is optimal.

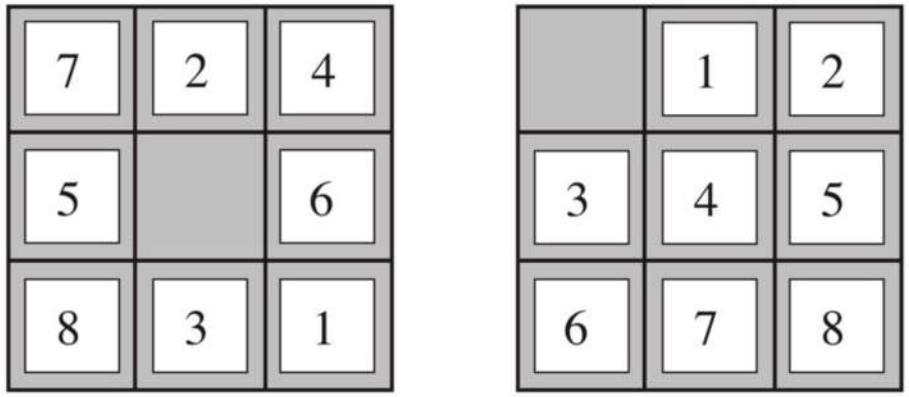
Rationale:

- Suppose G_o is the optimal goal.
- Suppose G_s is some suboptimal goal.
- Suppose n is an unexpanded node in the fringe such that n is on the shortest path to G_o.
- $f(G_s) = g(G_s)$ since $h(G_s) = 0$
  $f(G_o) = g(G_o)$ since $h(G_o) = 0$
  $f(G_s) > g(G_o)$ since G_s is suboptimal
  Then $f(G_s) > f(G_o) \ldots (1)$
- $h(n) \leq h^*(n)$ since h is admissible
  $g(n) + h(n) \leq g(n) + h^*(n) = g(G_o) = f(G_o)$
  Then, $f(n) \leq f(G_o) \ldots (2)$
  From (1) and (2) $f(G_s) > f(n)$
  so A* will never select G_s during the search and hence A* is optimal.

A* search criteria

- **Complete**: Yes
- **Time**: exponential
- **Space**: keeps every node in memory, the biggest problem
- **Optimal**: Yes!

Heuristics



- The solution is 26 steps long.
- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ =total Manhattan distance (sum of the horizontal and vertical distances).
- $h_1(n) = 8$
- Tiles 1 to 8 in the start state gives: $h_2 = 3+1+2+2+2+3+3+2 = 18$ which does not overestimate the true solution.

Search Algorithms: Recap

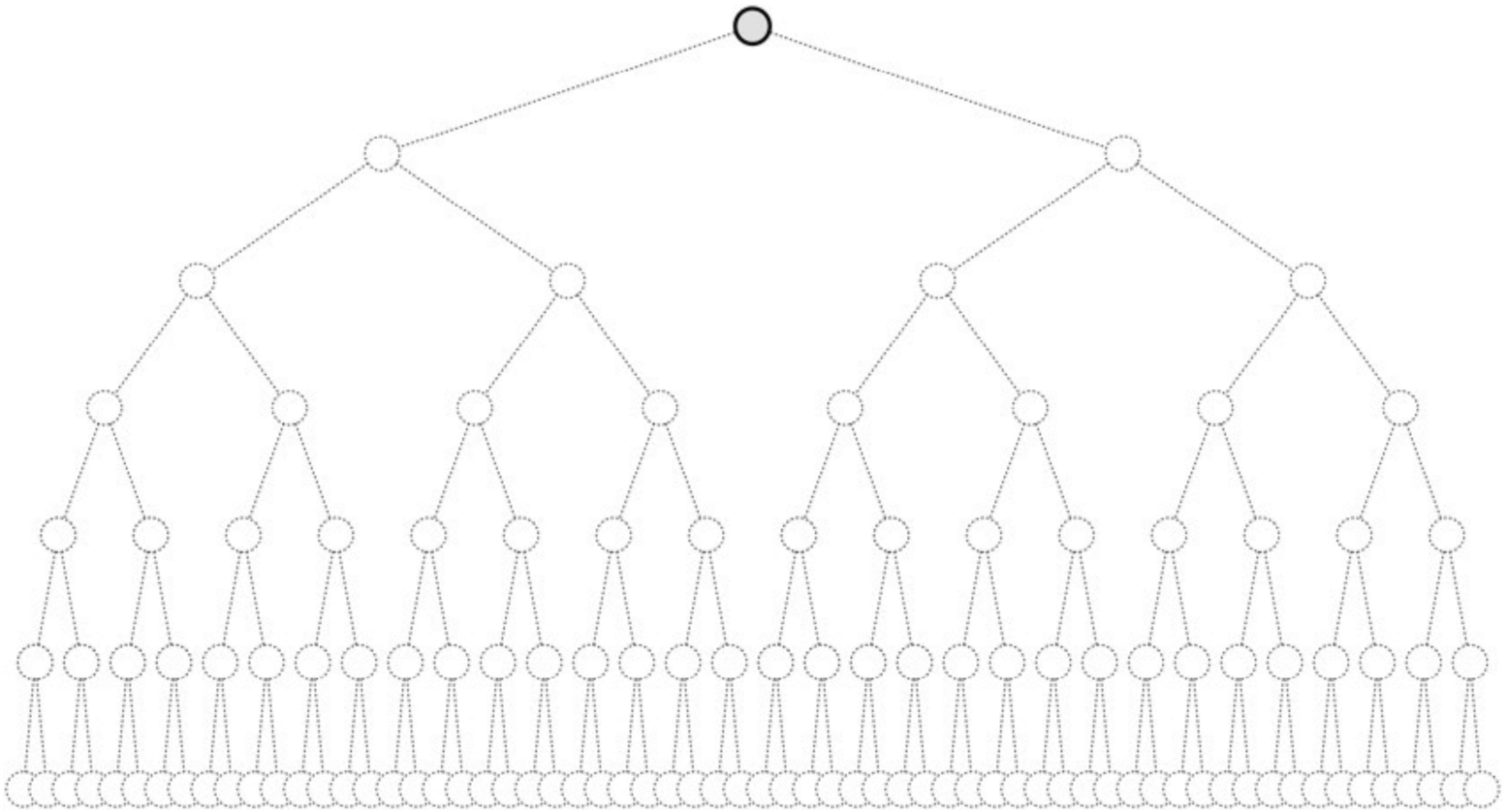
- **Uninformed Search**: Use no domain knowledge.

  BFS, DFS, DLS, IDS, UCS.

- **Informed Search**: Use a heuristic function that estimates how close a state is to the goal.

  Greedy search, A*, IDA*.

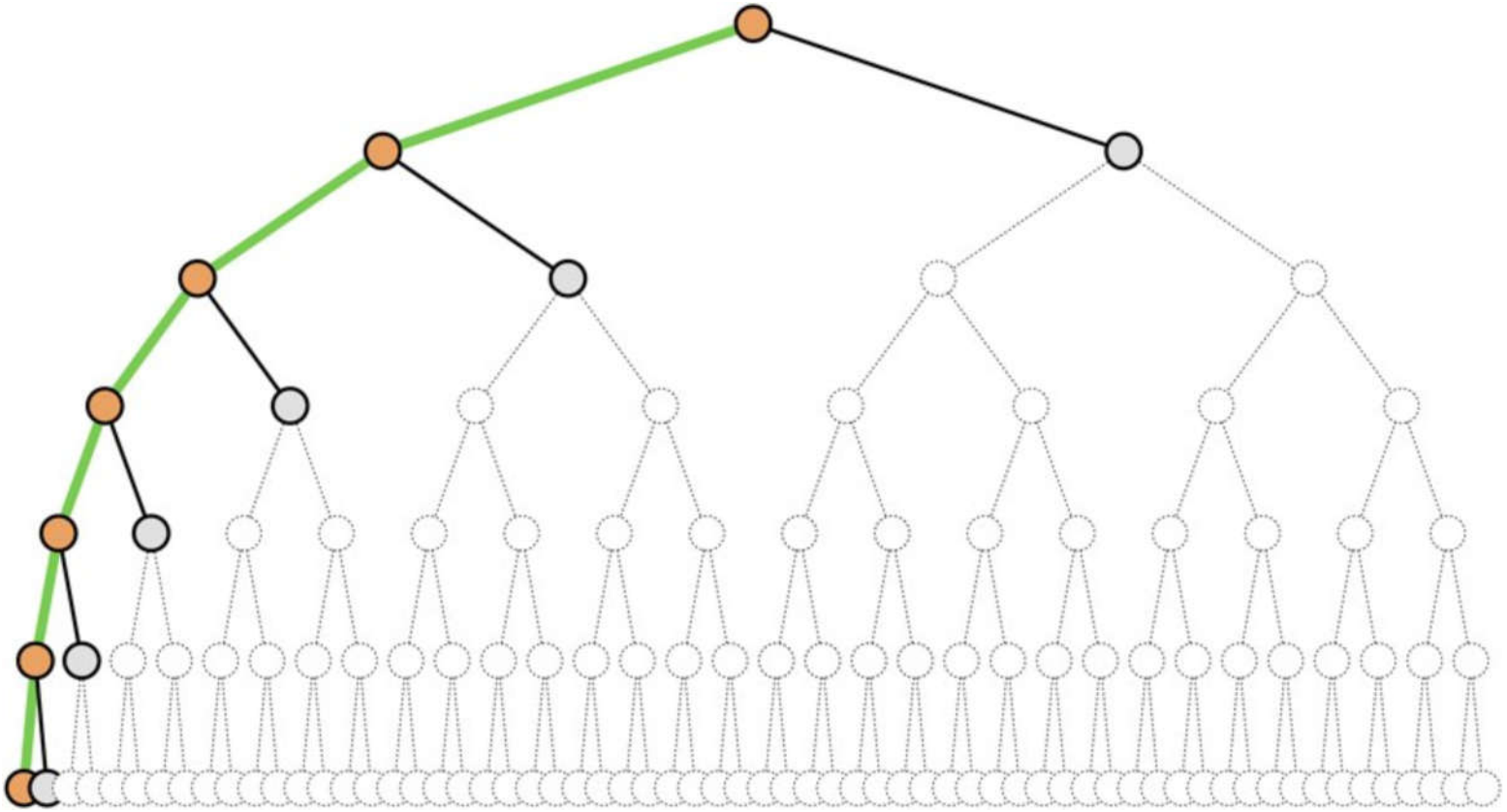
DFS



Searches branch by branch ...

... with each branch pursued to maximum depth.

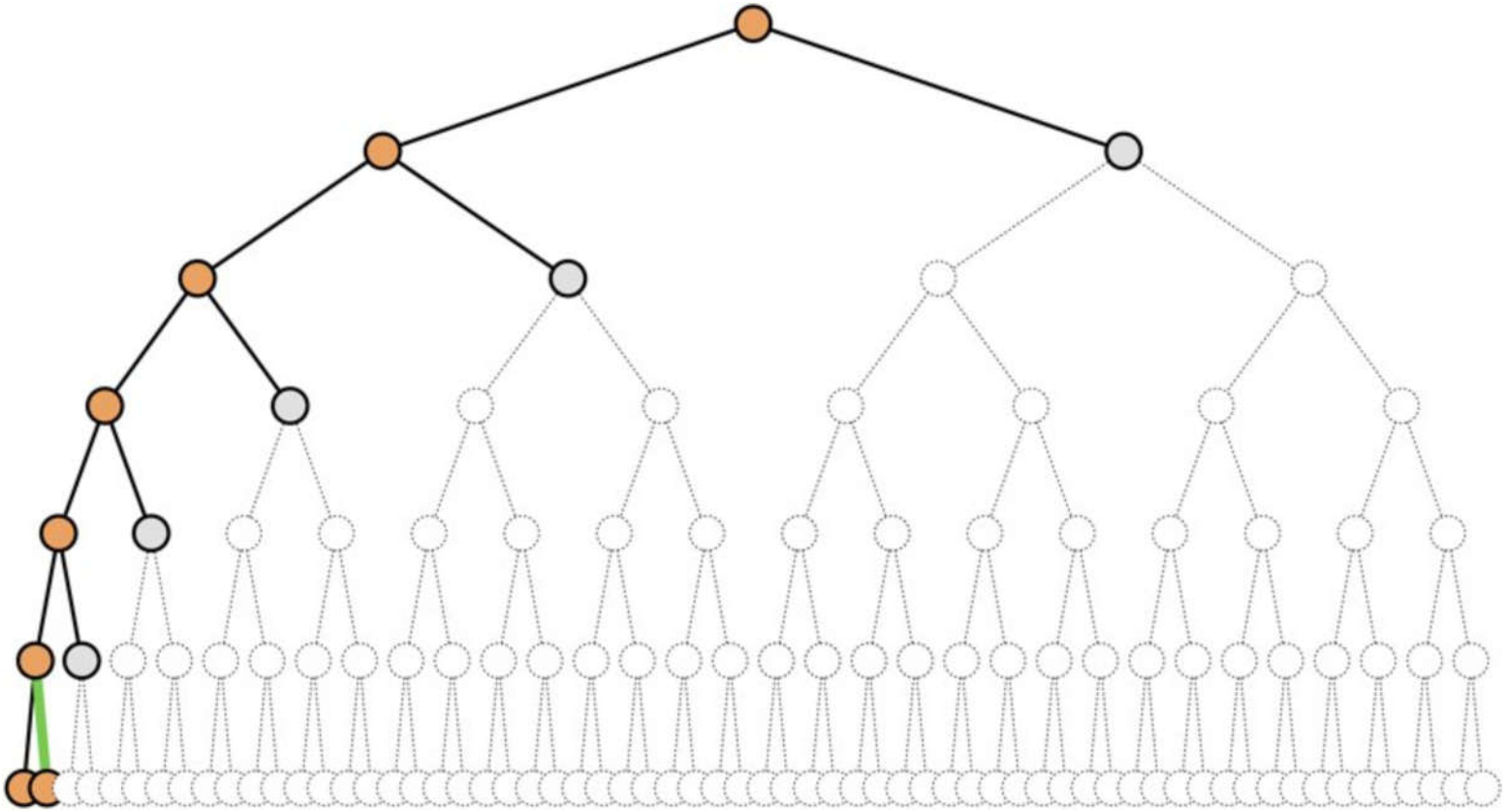
DFS



Searches branch by branch ...

... with each branch pursued to maximum depth.

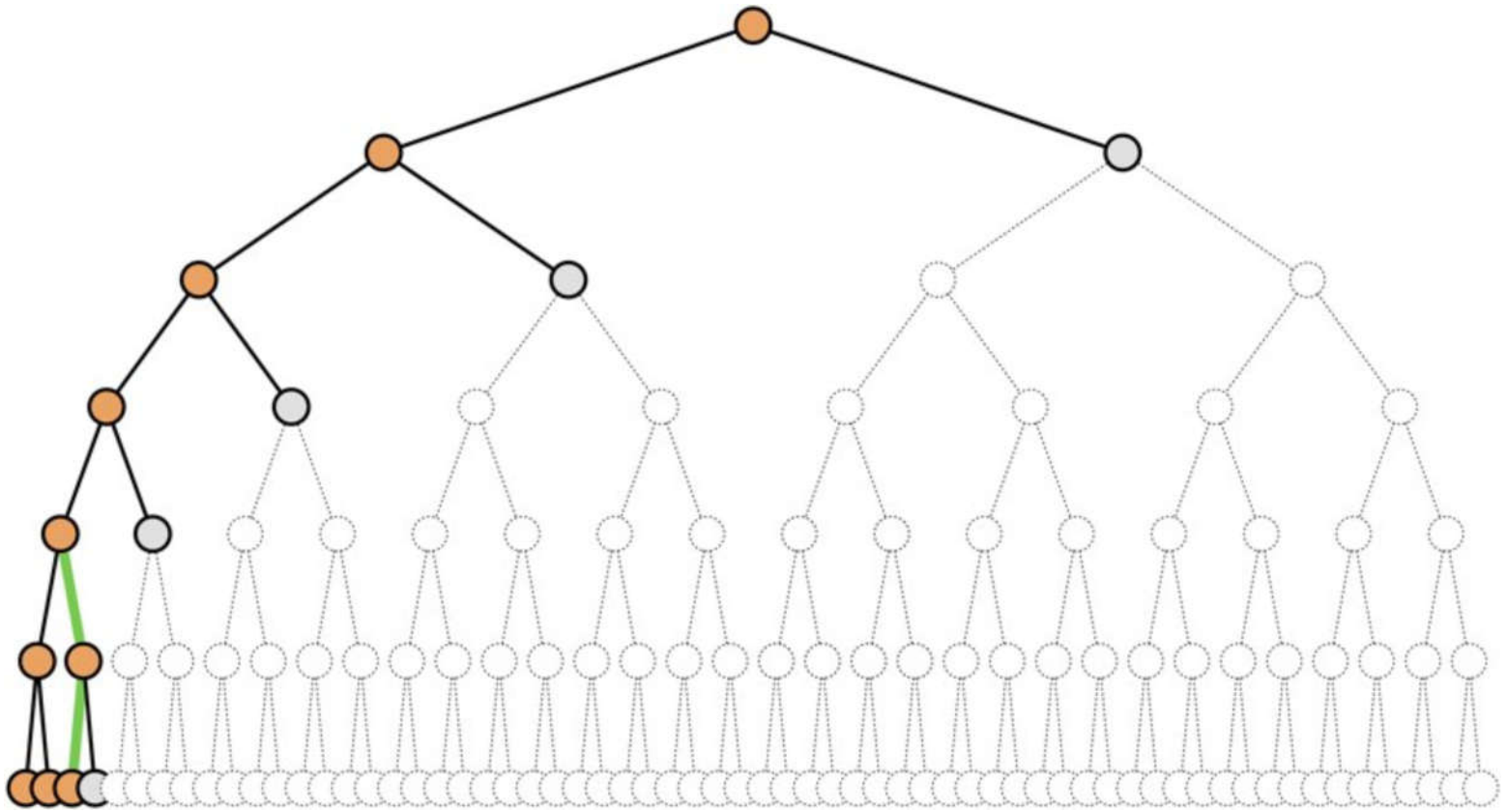
DFS



Searches branch by branch ...

... with each branch pursued to maximum depth.

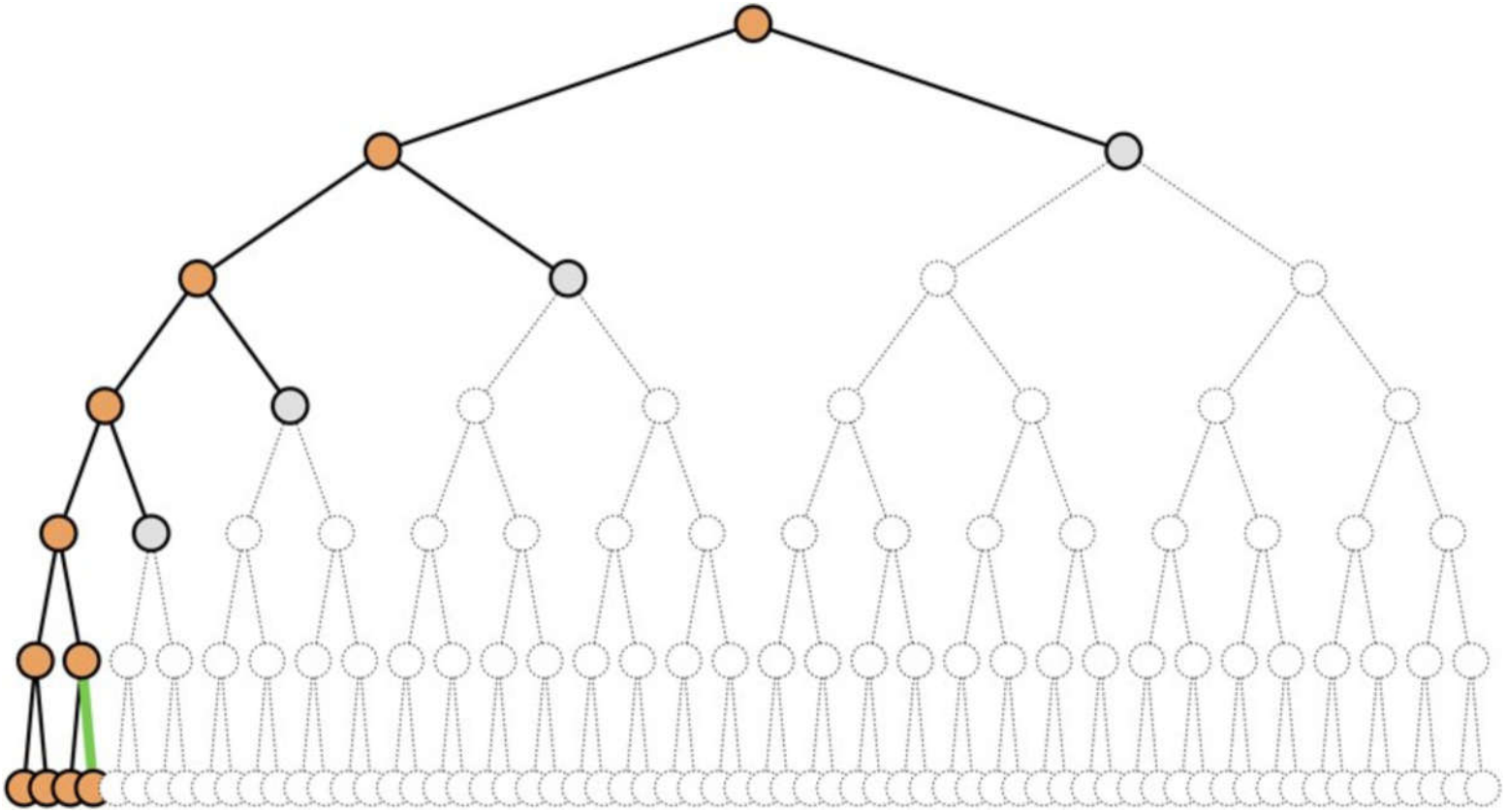
DFS



Searches branch by branch ...

... with each branch pursued to maximum depth.

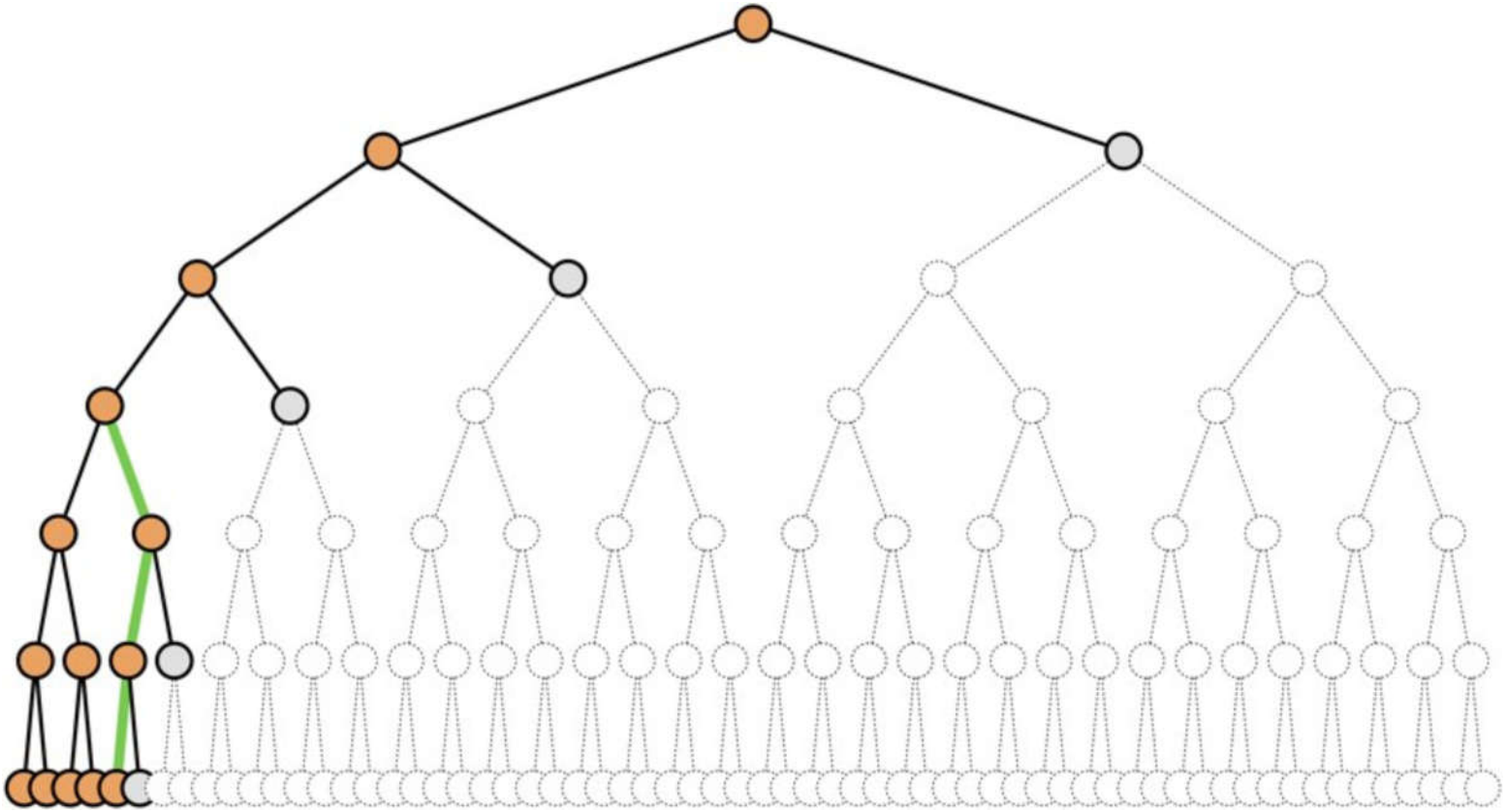
DFS



Searches branch by branch ...

... with each branch pursued to maximum depth.

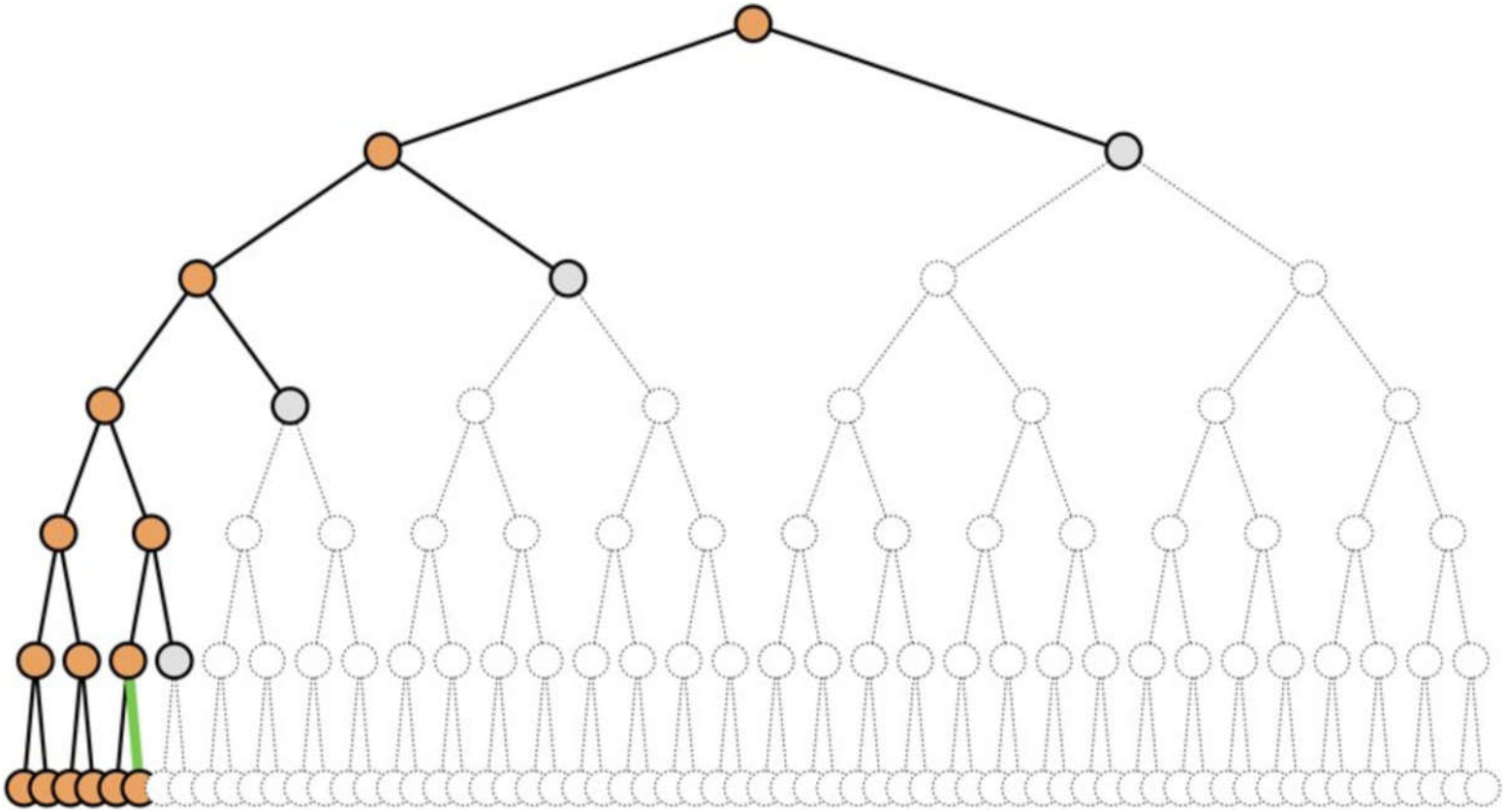
DFS



Searches branch by branch ...

... with each branch pursued to maximum depth.

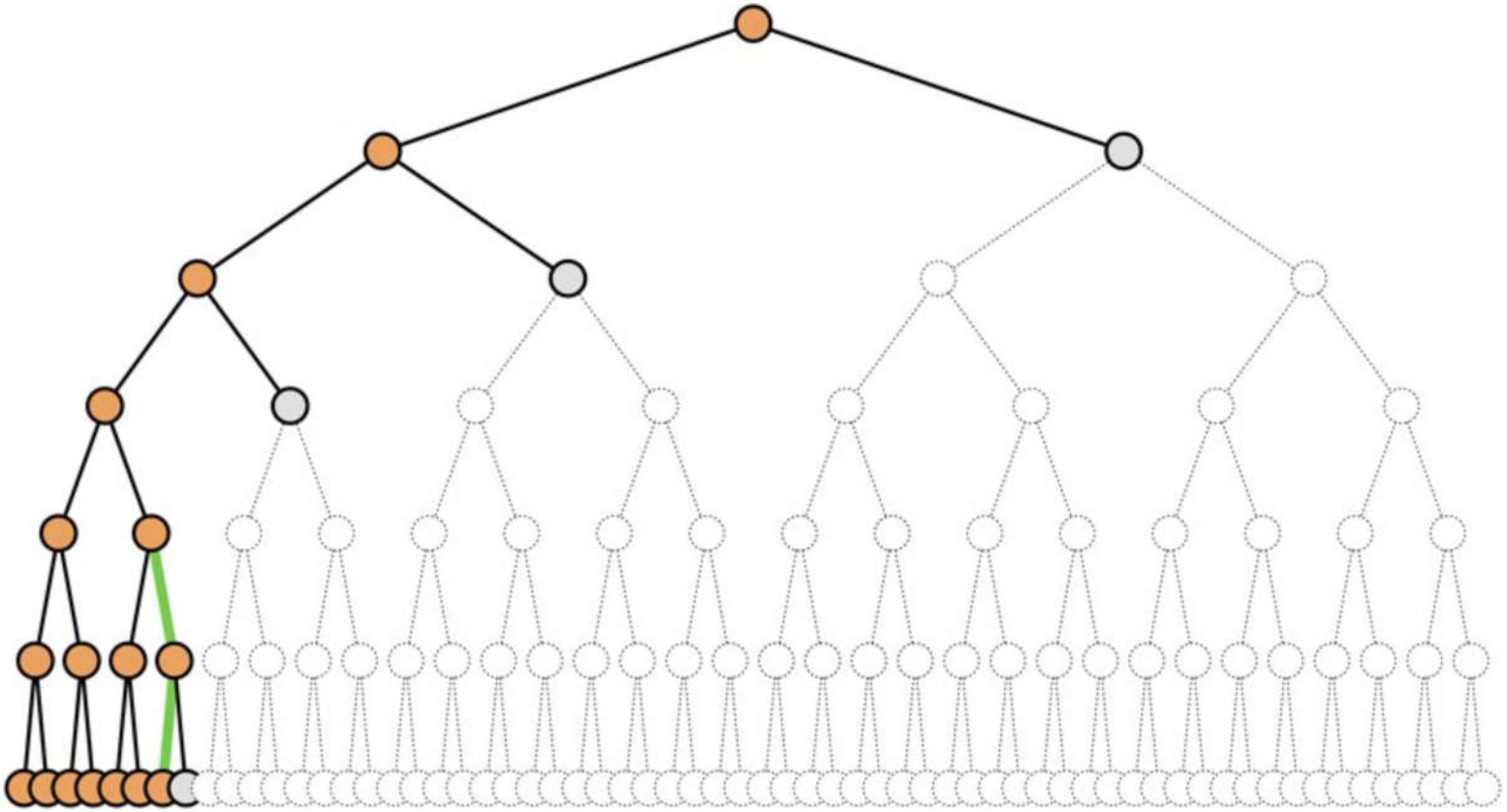
DFS



Searches branch by branch ...

... with each branch pursued to maximum depth.

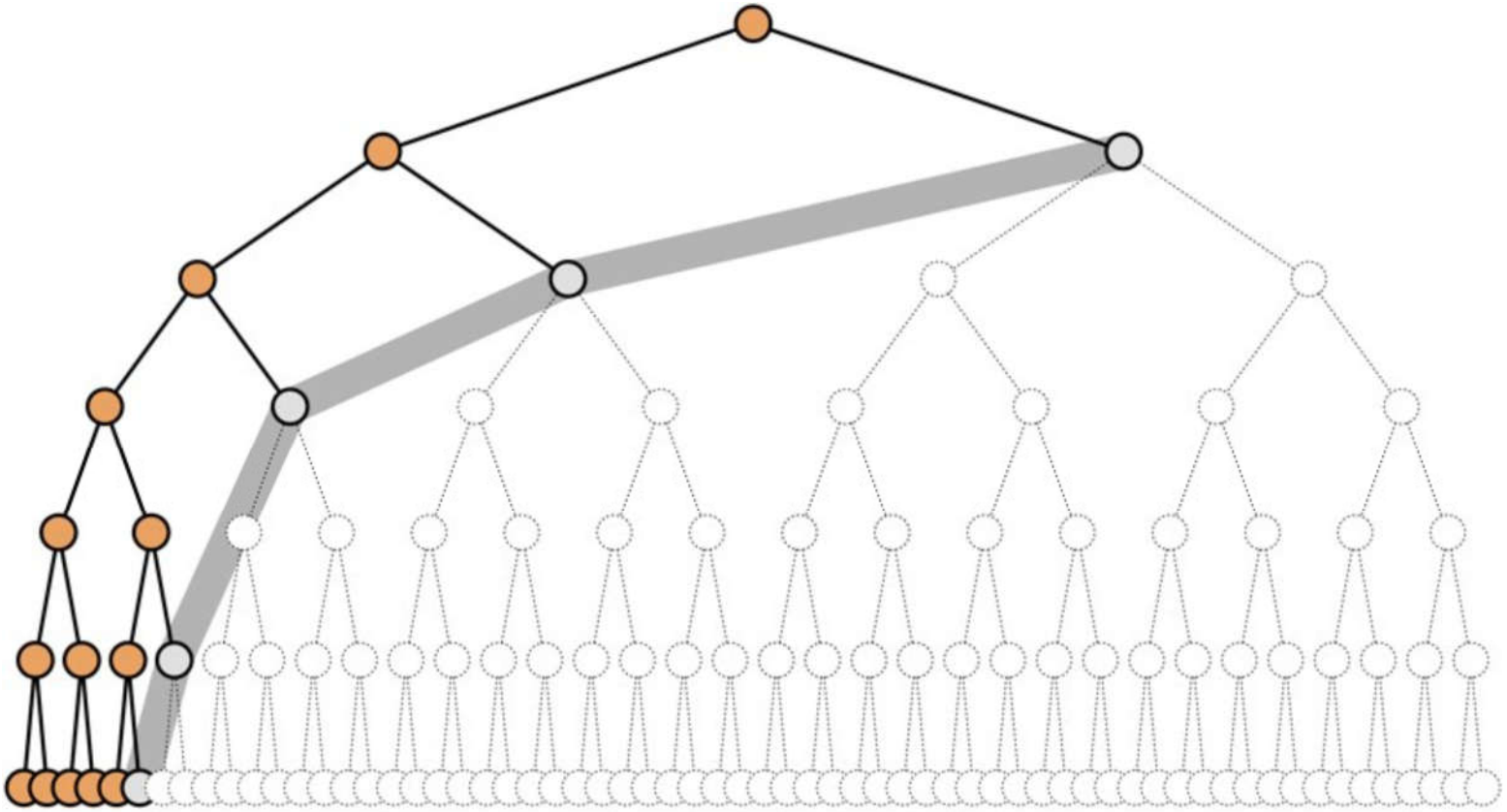
DFS



Searches branch by branch ...

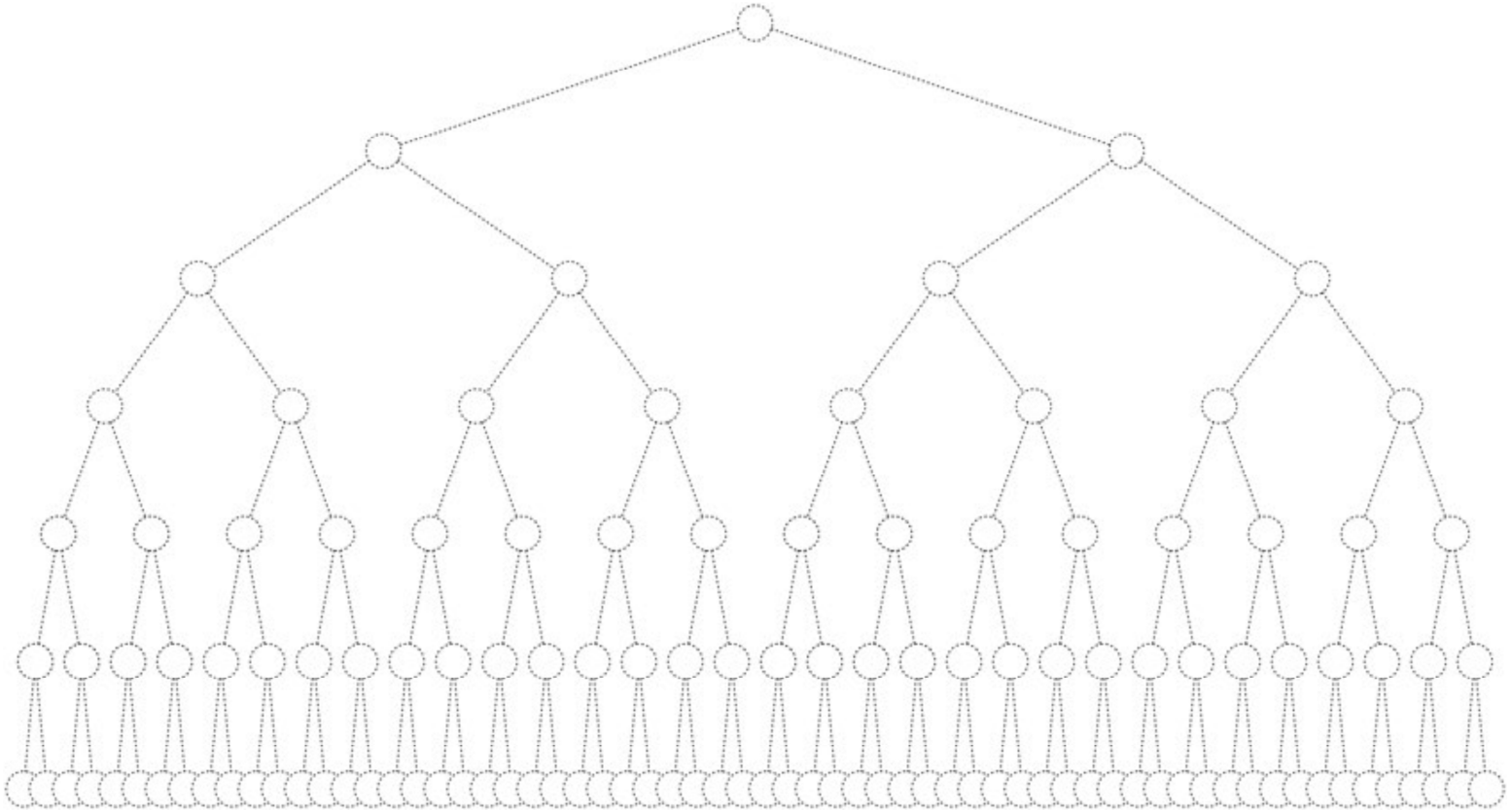
... with each branch pursued to maximum depth.

DFS



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

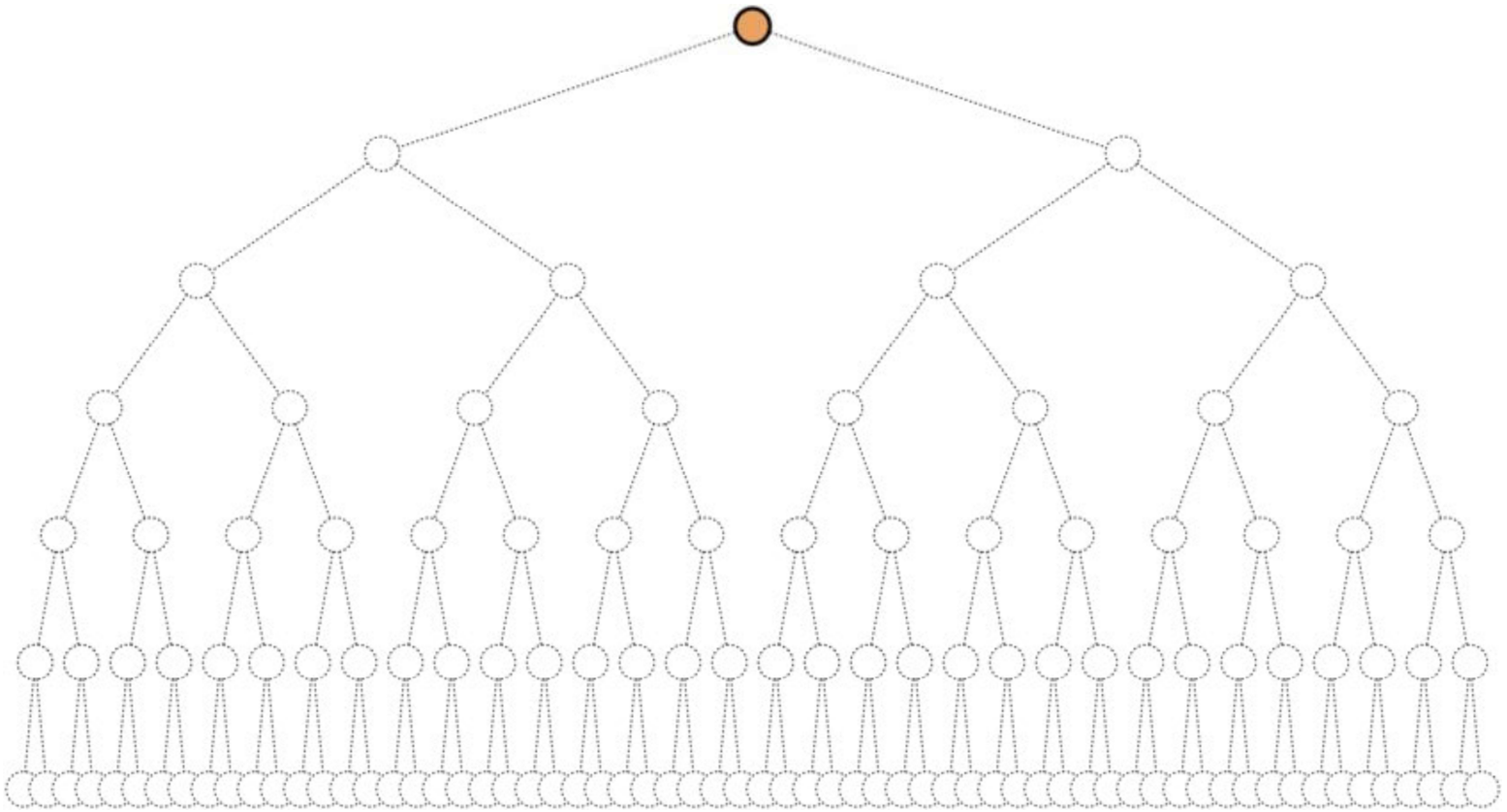
IDS



Searches subtree by subtree ...

... with each subtree increasing by depth limit.

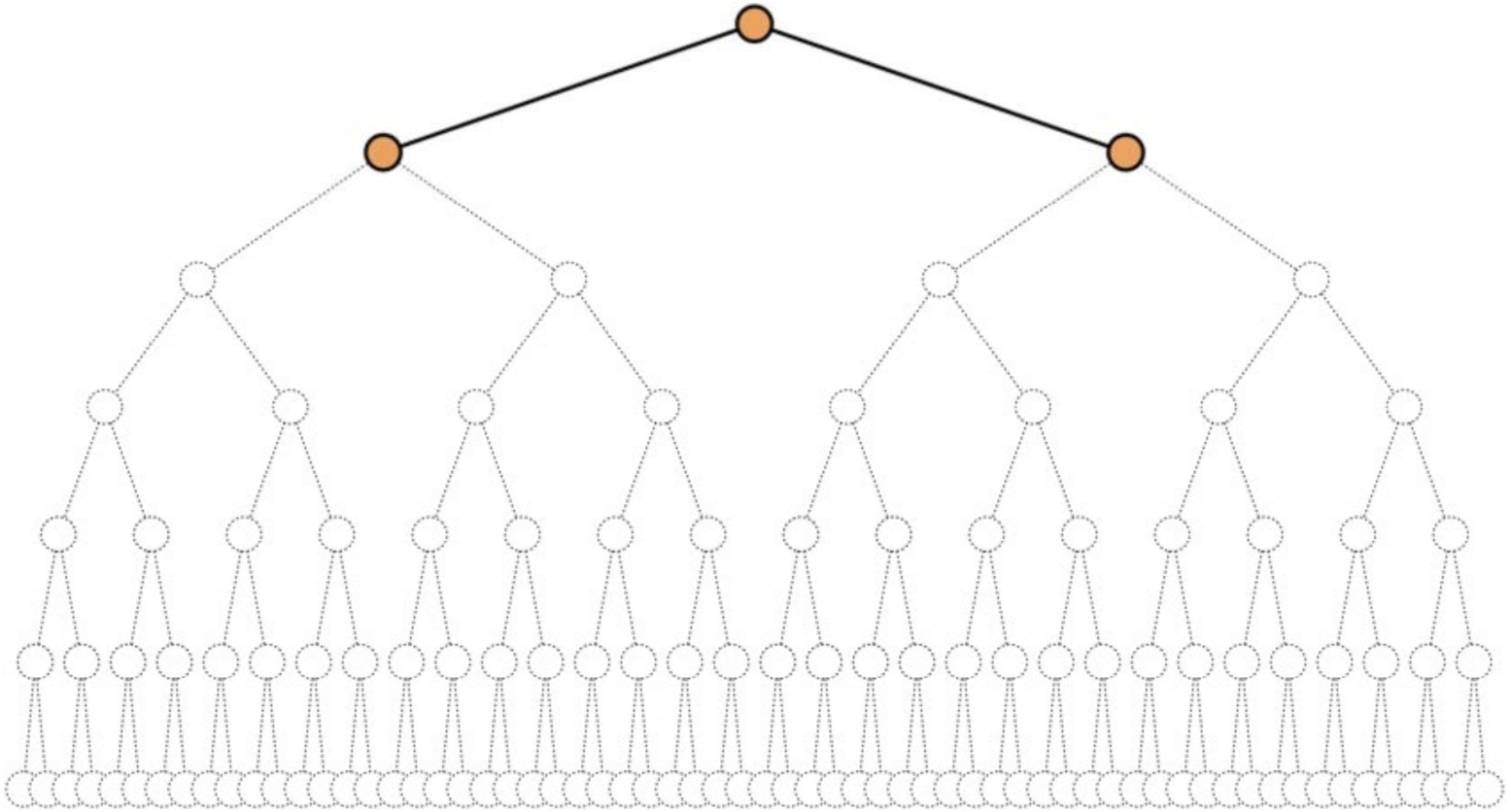
IDS



Searches subtree by subtree ...

... with each subtree increasing by depth limit.

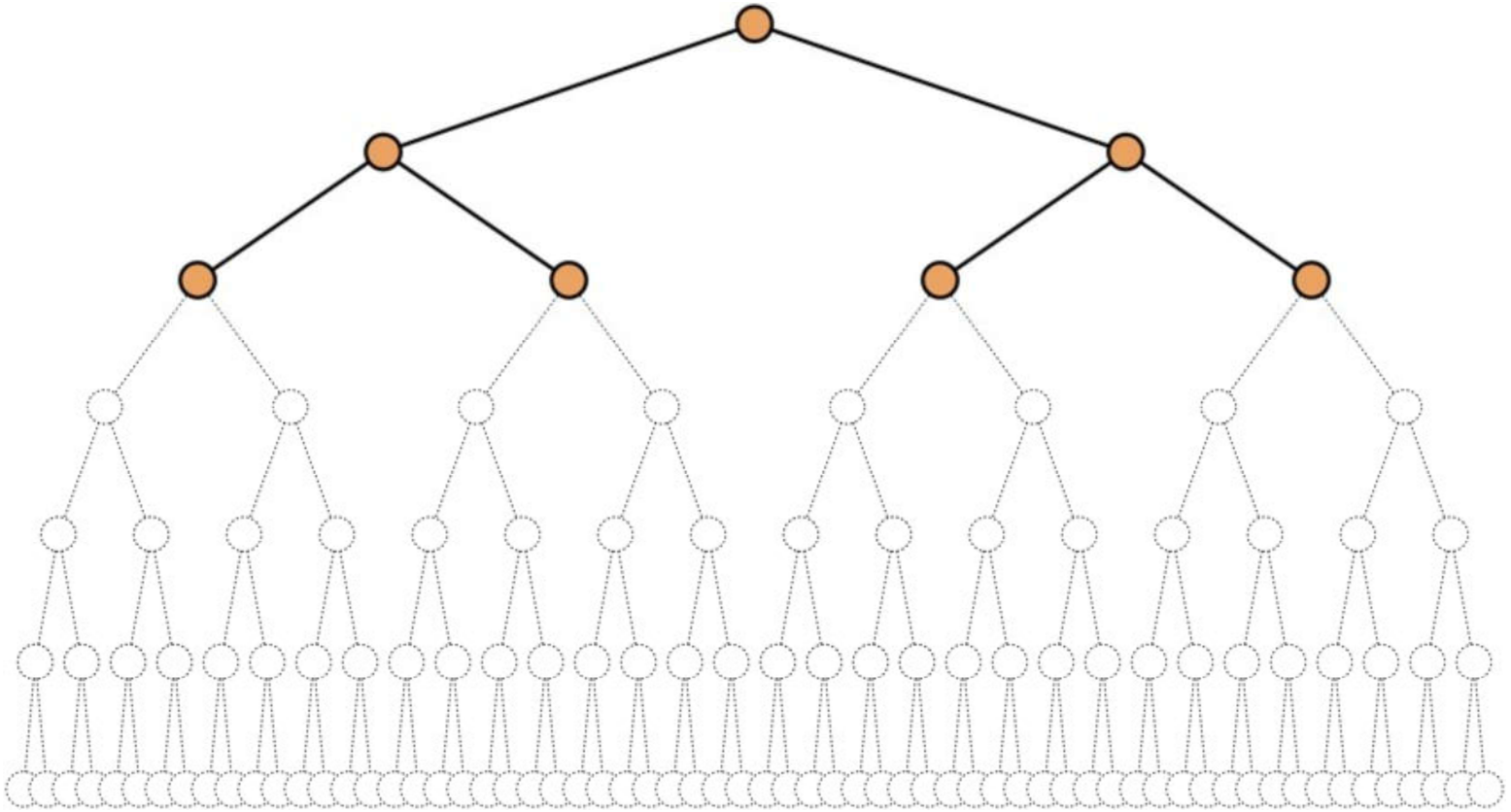
IDS



Searches subtree by subtree ...

... with each subtree increasing by depth limit.

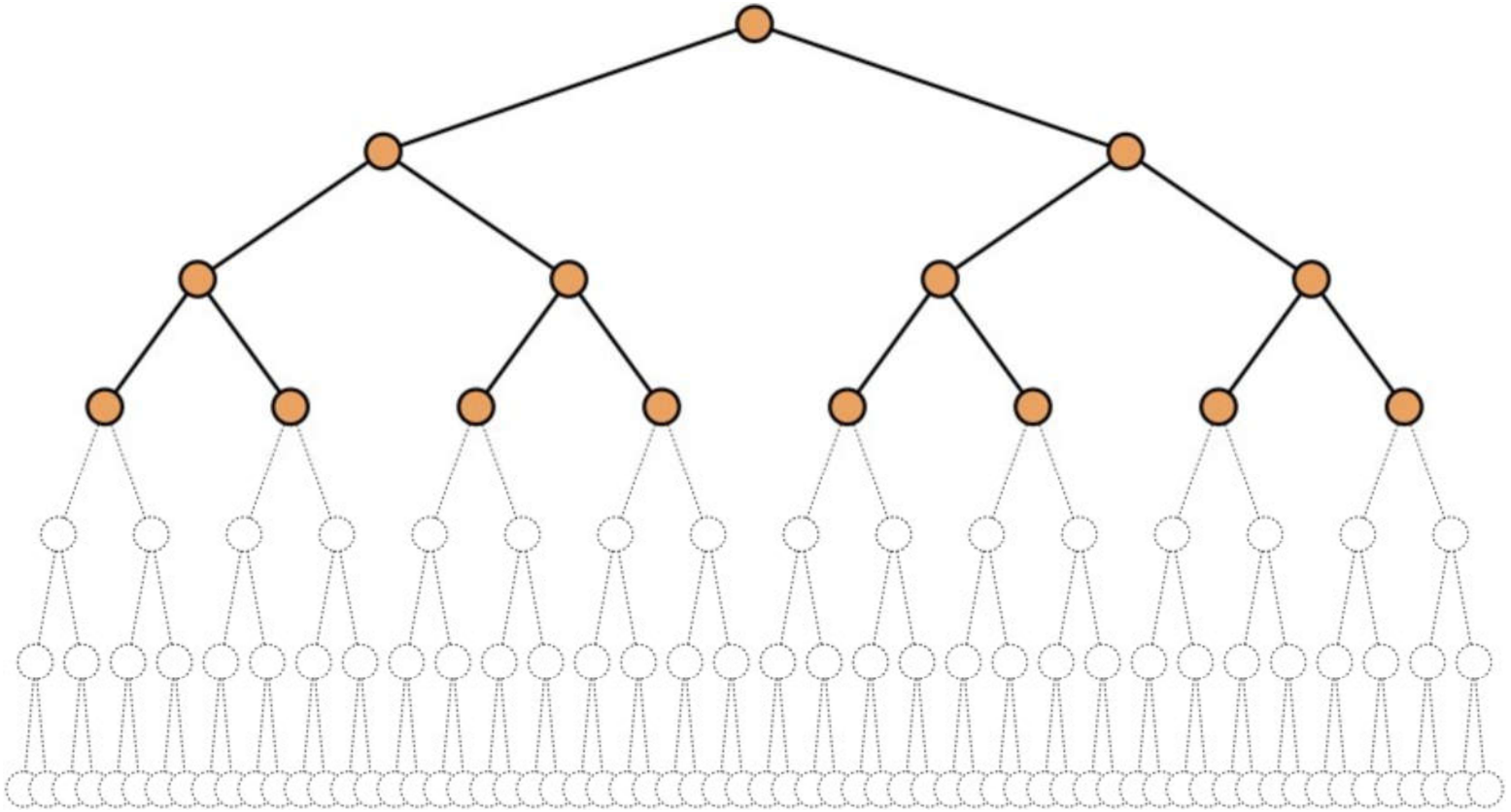
IDS



Searches subtree by subtree ...

... with each subtree increasing by depth limit.

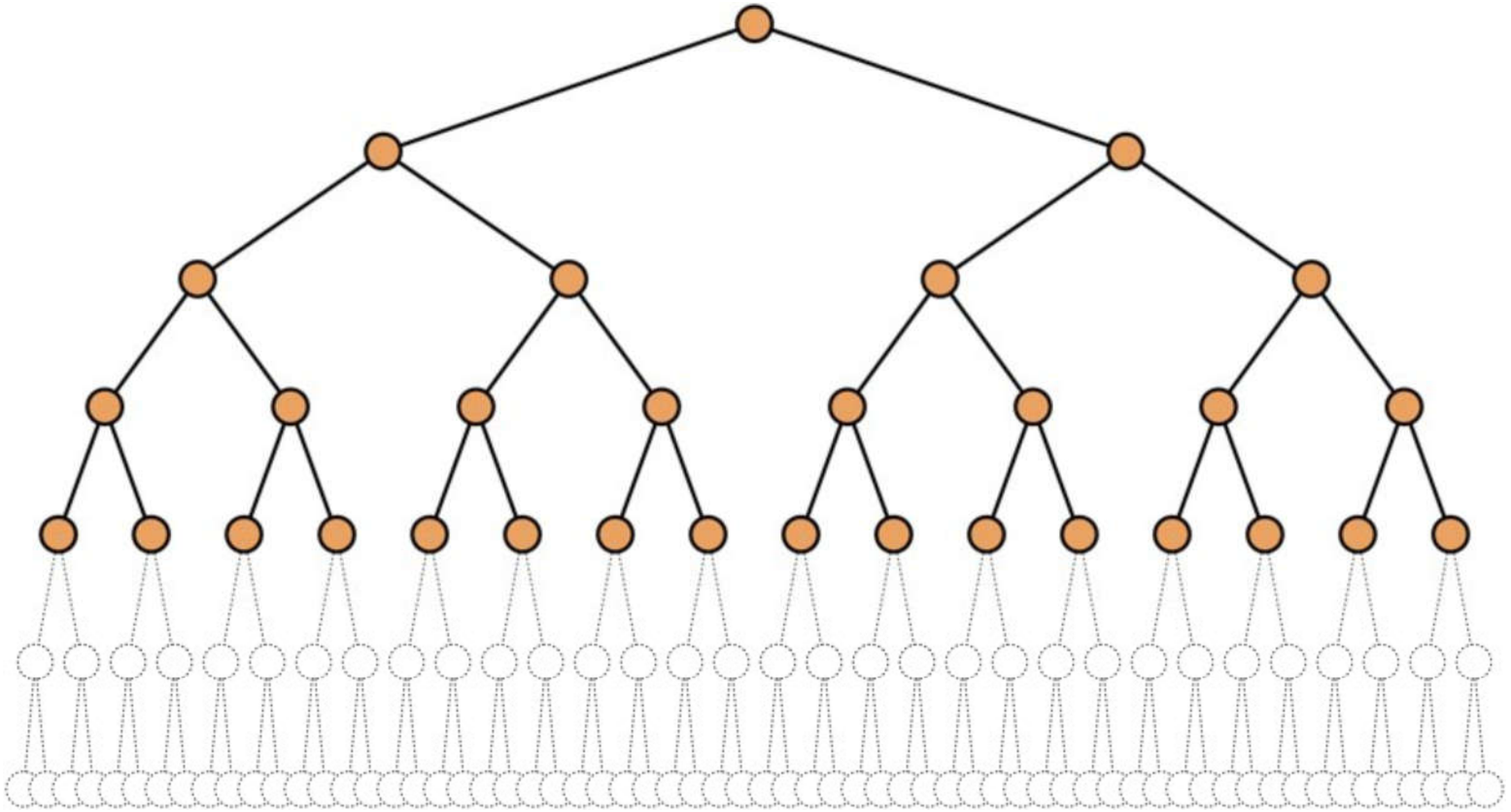
IDS



Searches subtree by subtree ...

... with each subtree increasing by depth limit.

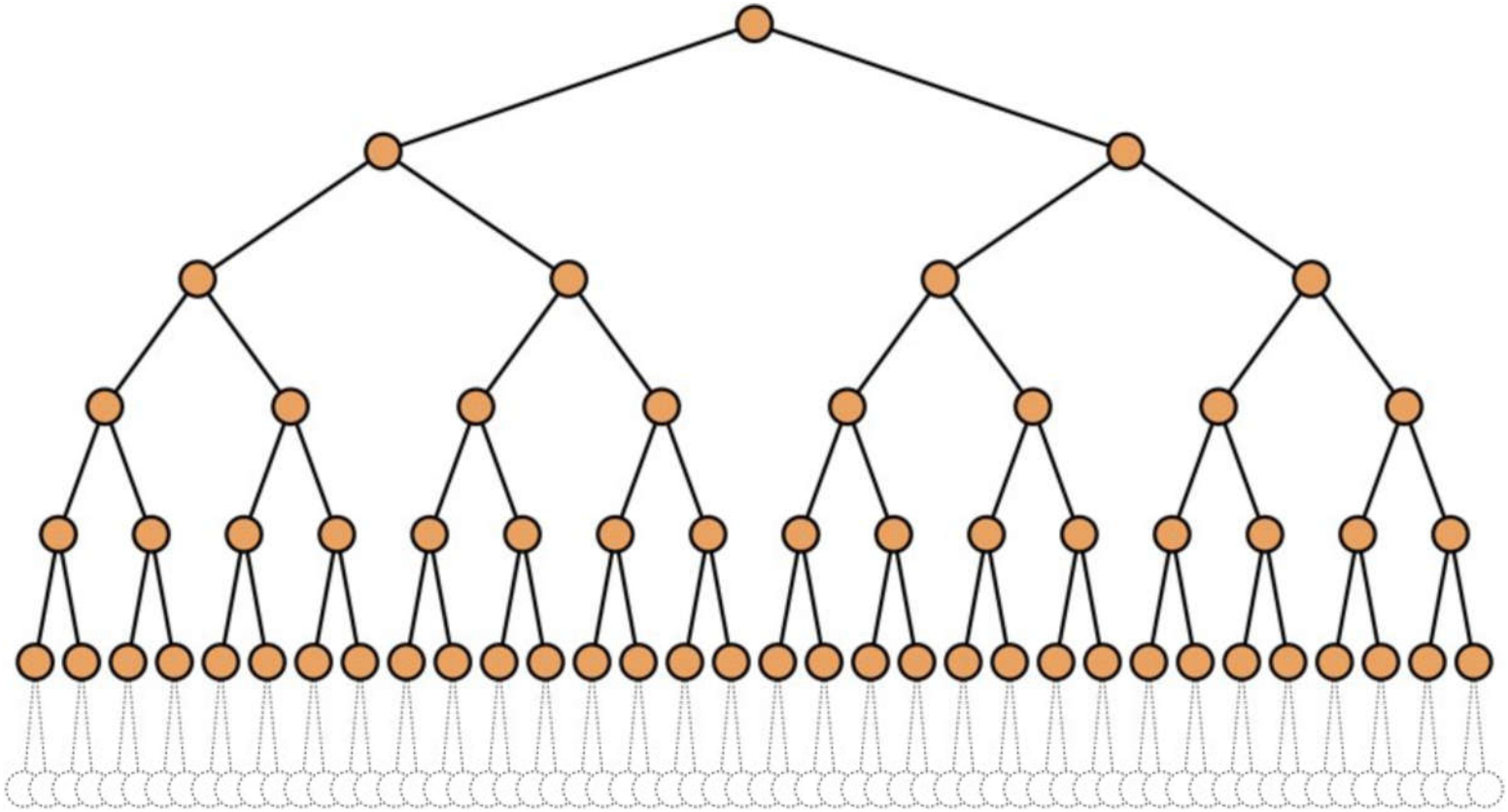
IDS



Searches subtree by subtree ...

... with each subtree increasing by depth limit.

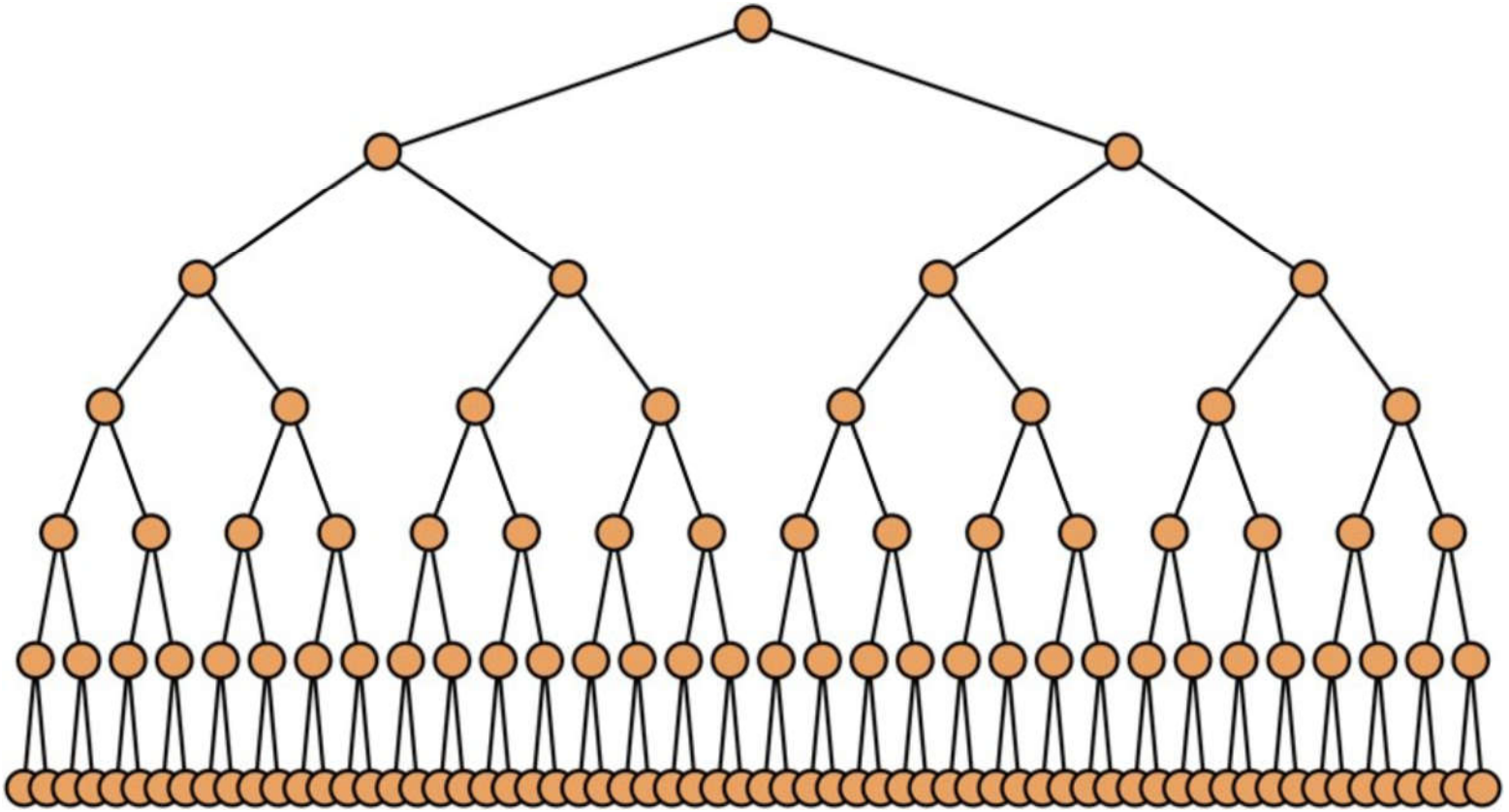
IDS



Searches subtree by subtree ...

... with each subtree increasing by depth limit.

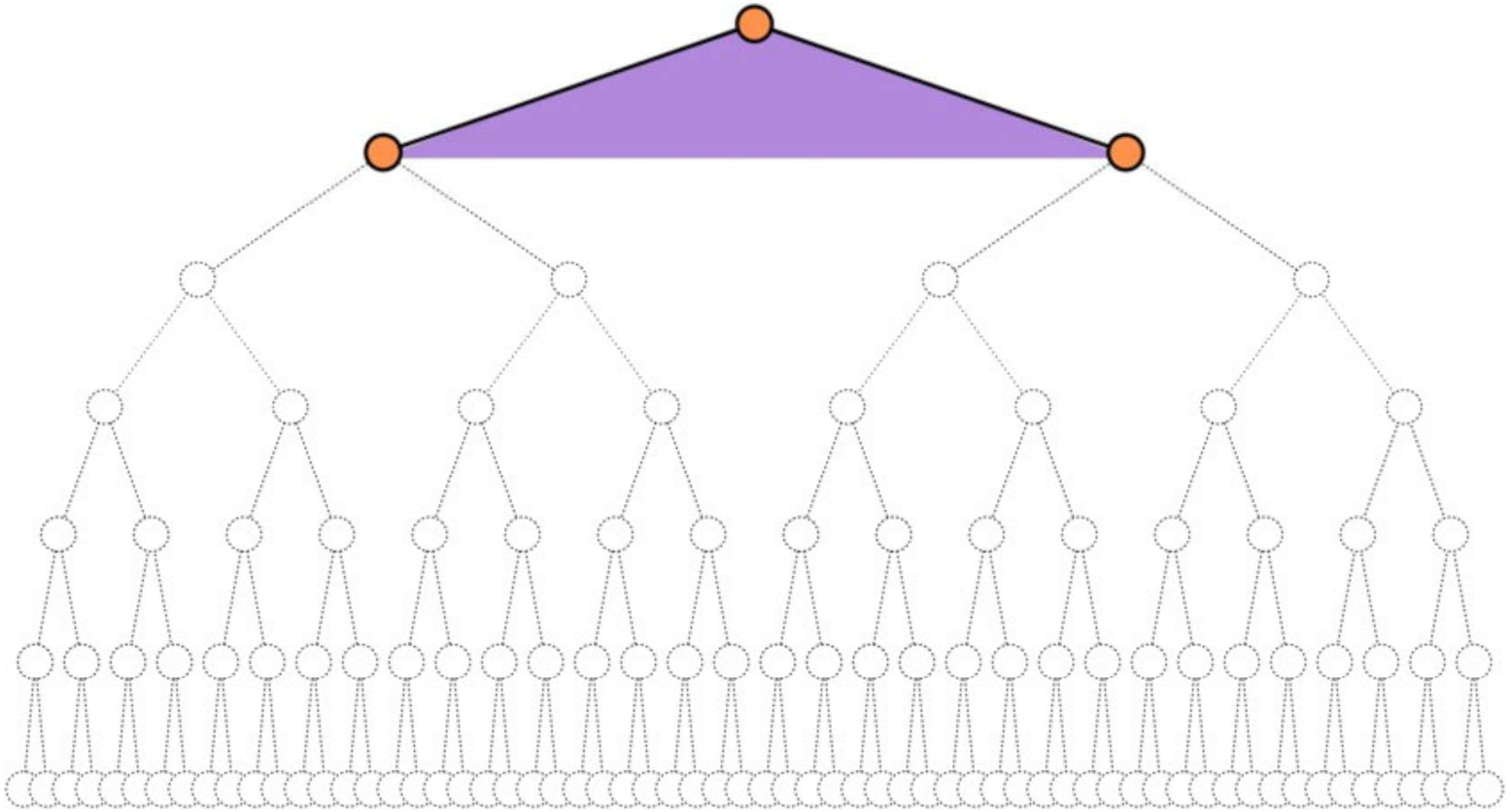
IDS



Searches subtree by subtree ...

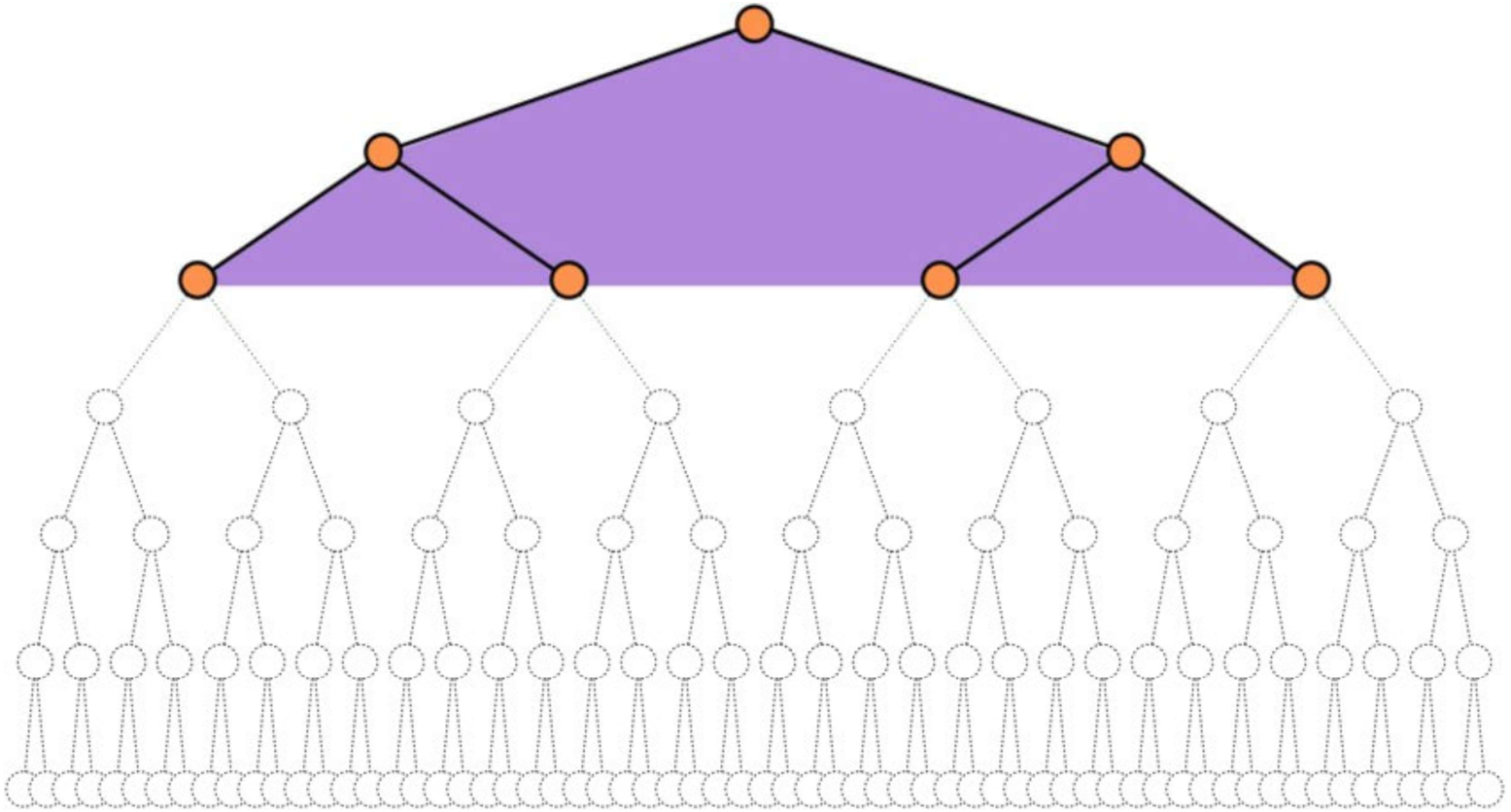
... with each subtree increasing by depth limit.

IDS



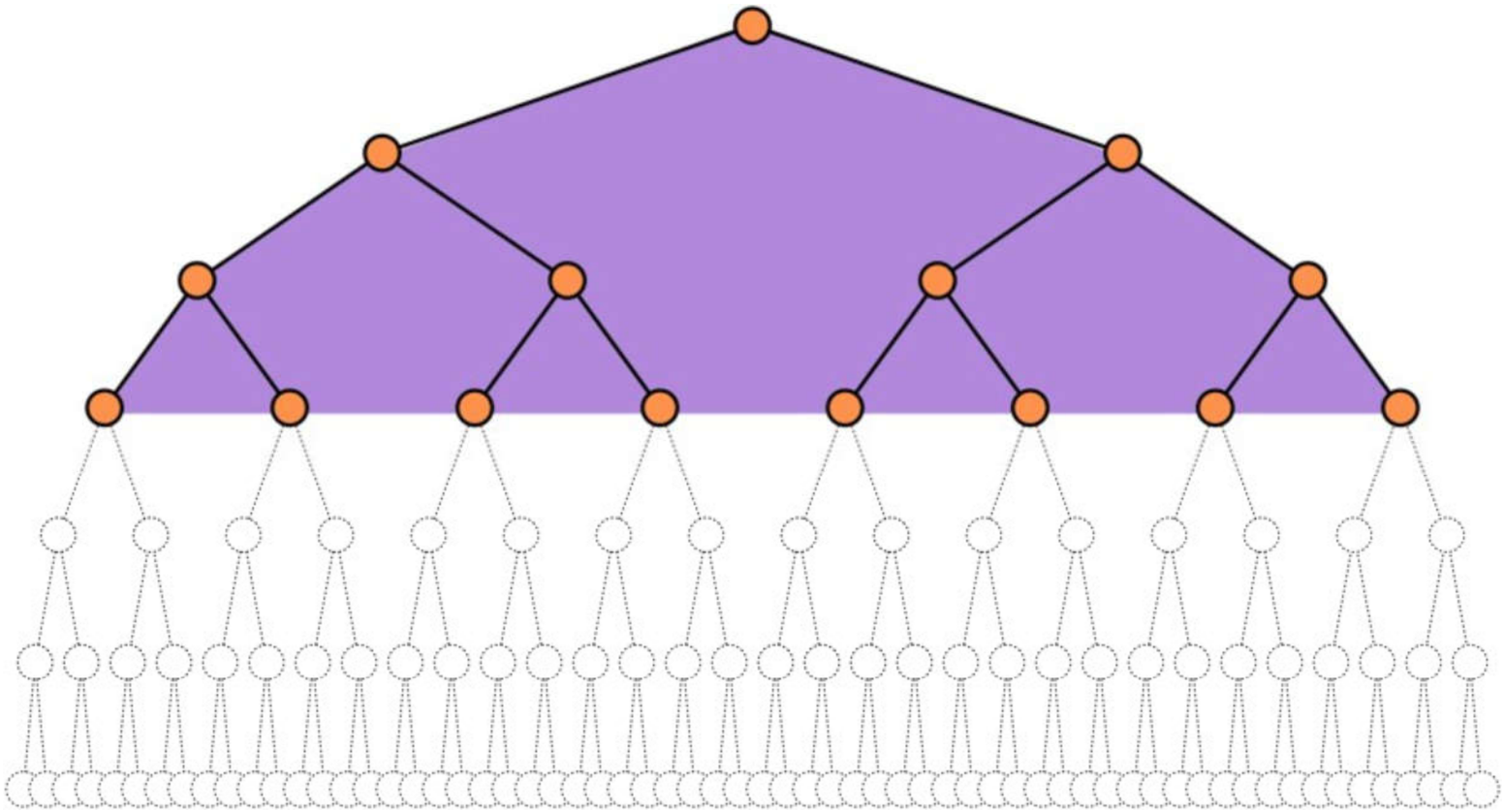
Each iteration is simply an instance of depth-limited search.
Search proceeds by exhausting larger and larger subtrees.

IDS



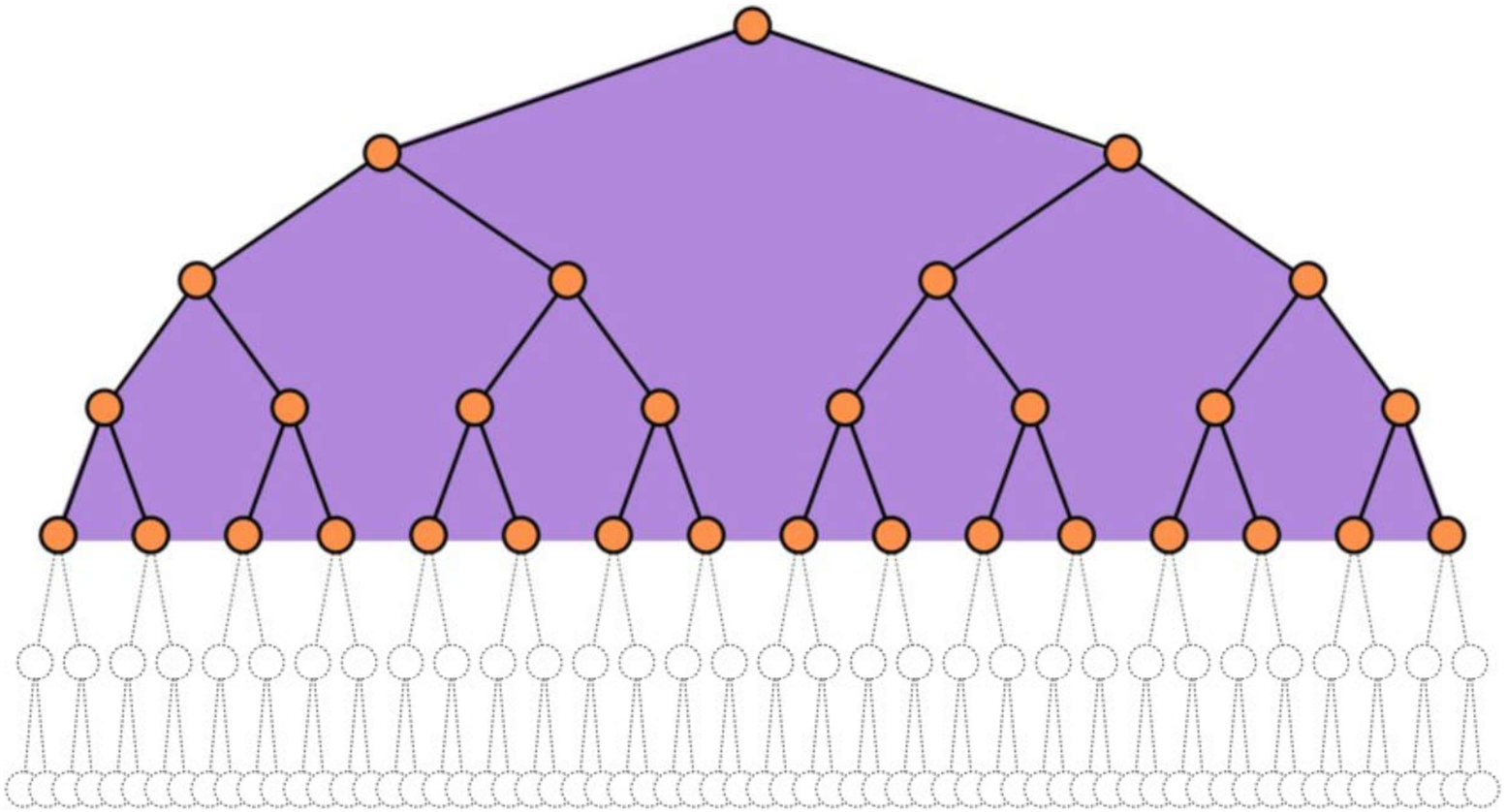
Each iteration is simply an instance of depth-limited search.
Search proceeds by exhausting larger and larger subtrees.

IDS



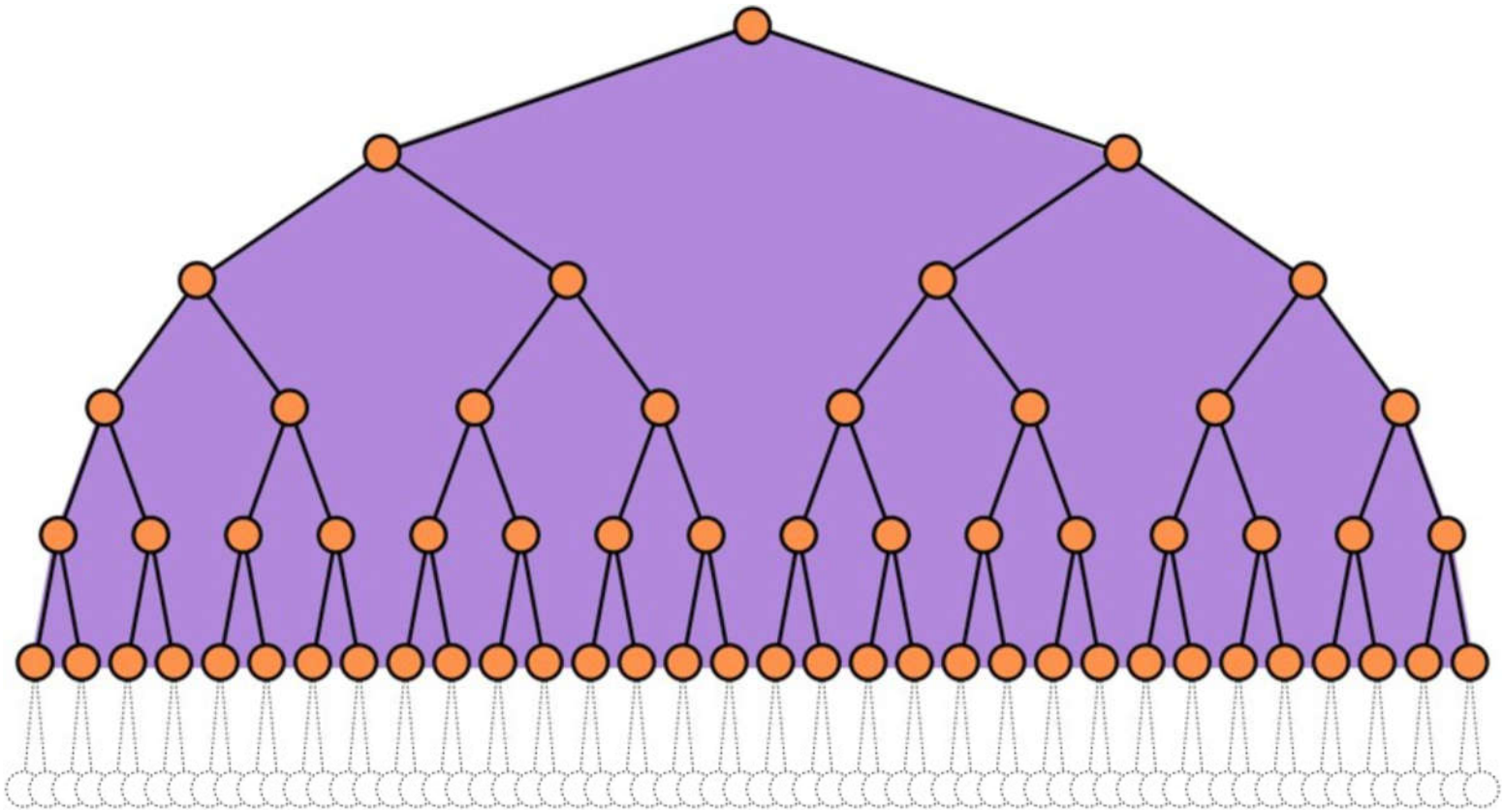
Each iteration is simply an instance of depth-limited search.
Search proceeds by exhausting larger and larger subtrees.

IDS



Each iteration is simply an instance of depth-limited search.
Search proceeds by exhausting larger and larger subtrees.

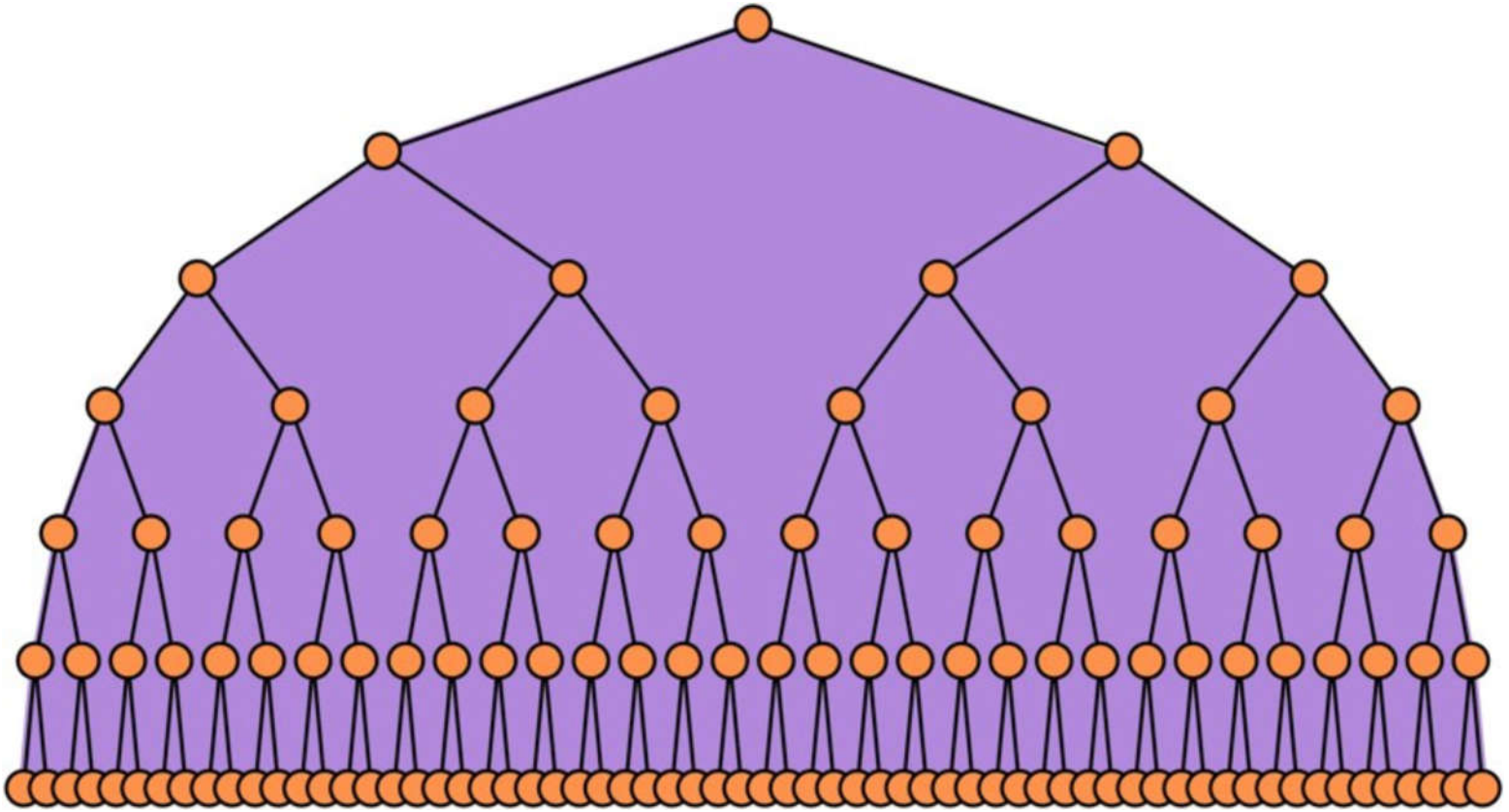
IDS



Each iteration is simply an instance of depth-limited search.

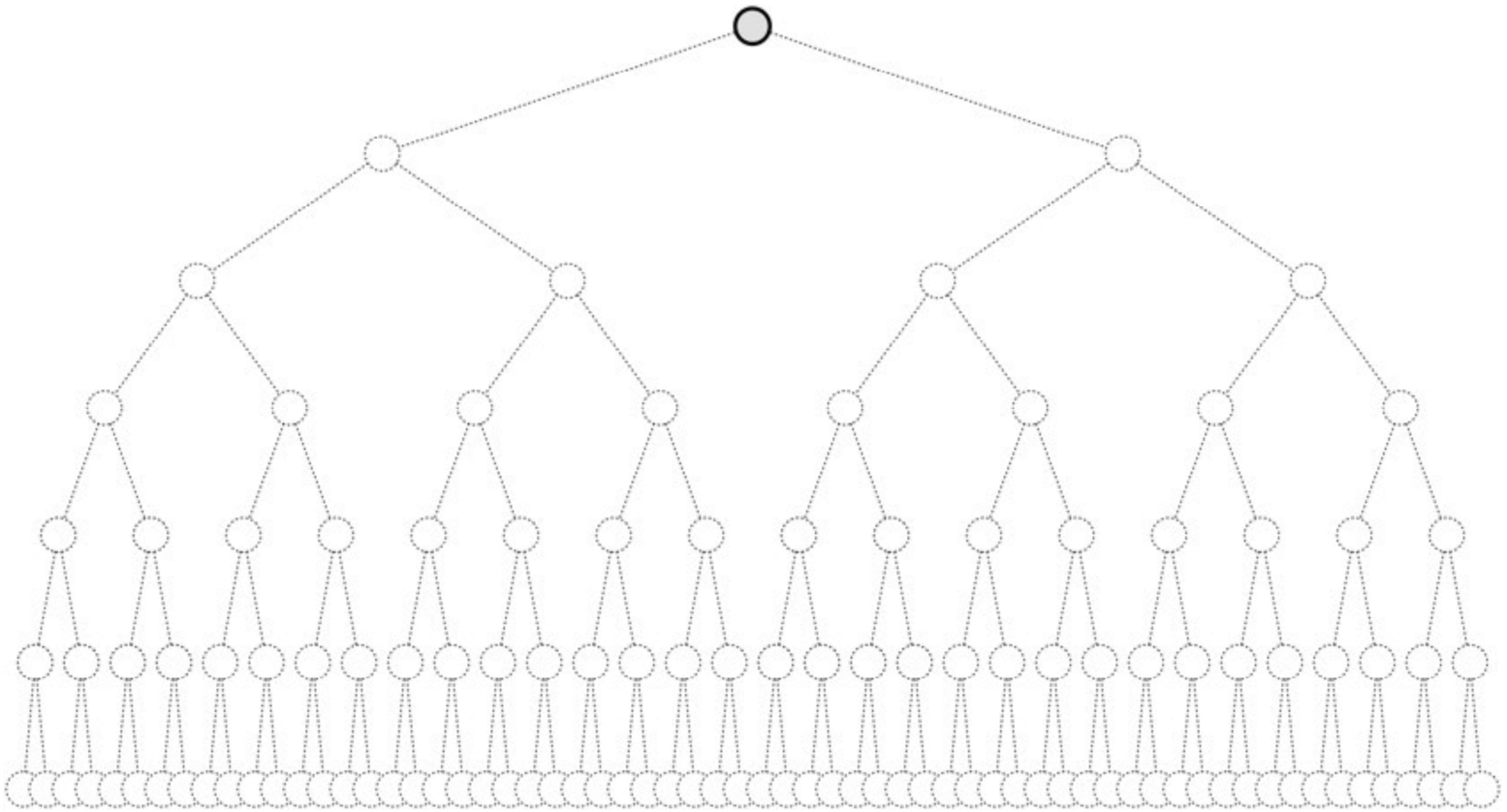
Search proceeds by exhausting larger and larger subtrees.

IDS



Each iteration is simply an instance of depth-limited search.
Search proceeds by exhausting larger and larger subtrees.

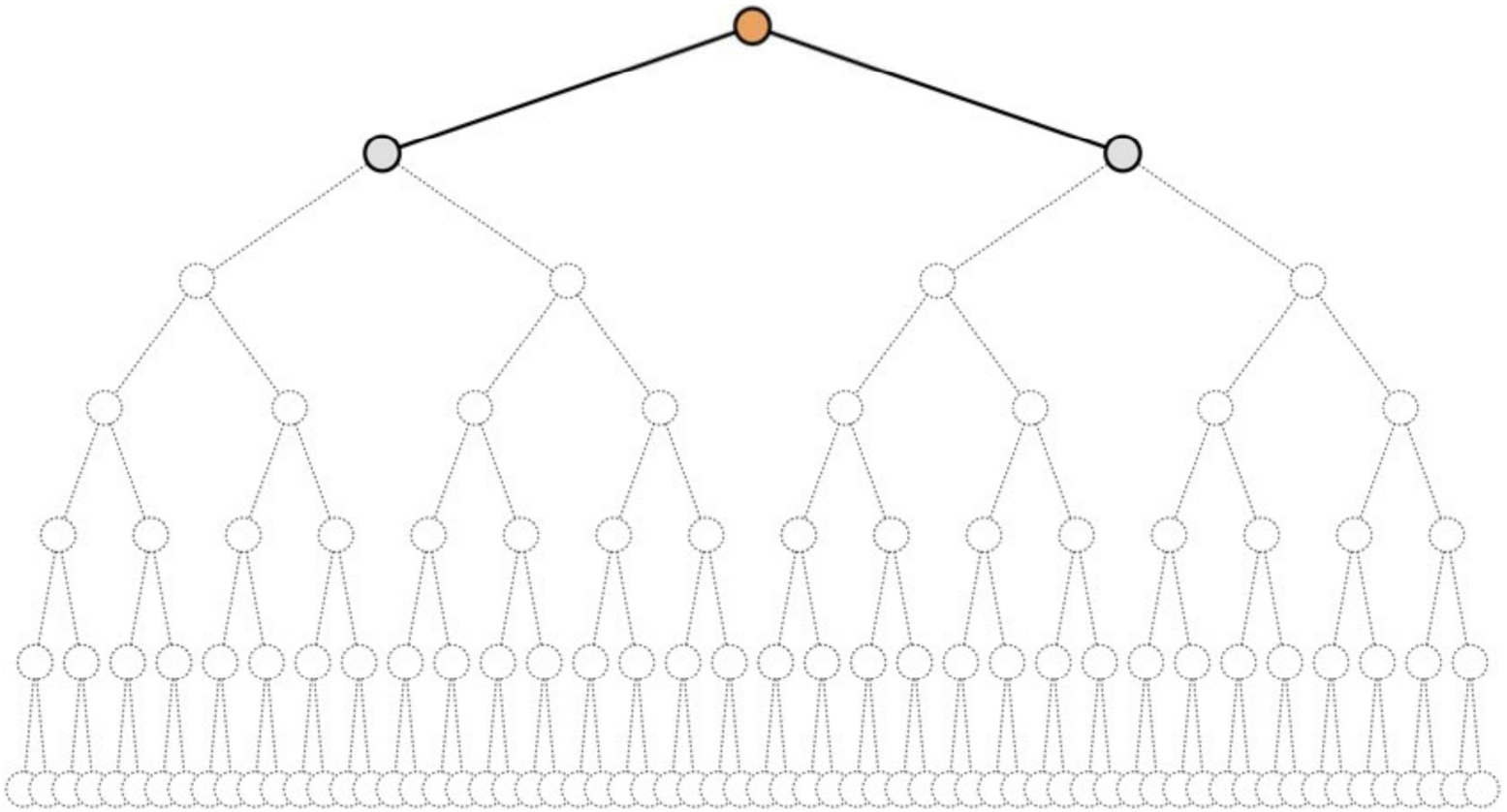
BFS



Searches layer by layer ...

... with each layer organized by node depth.

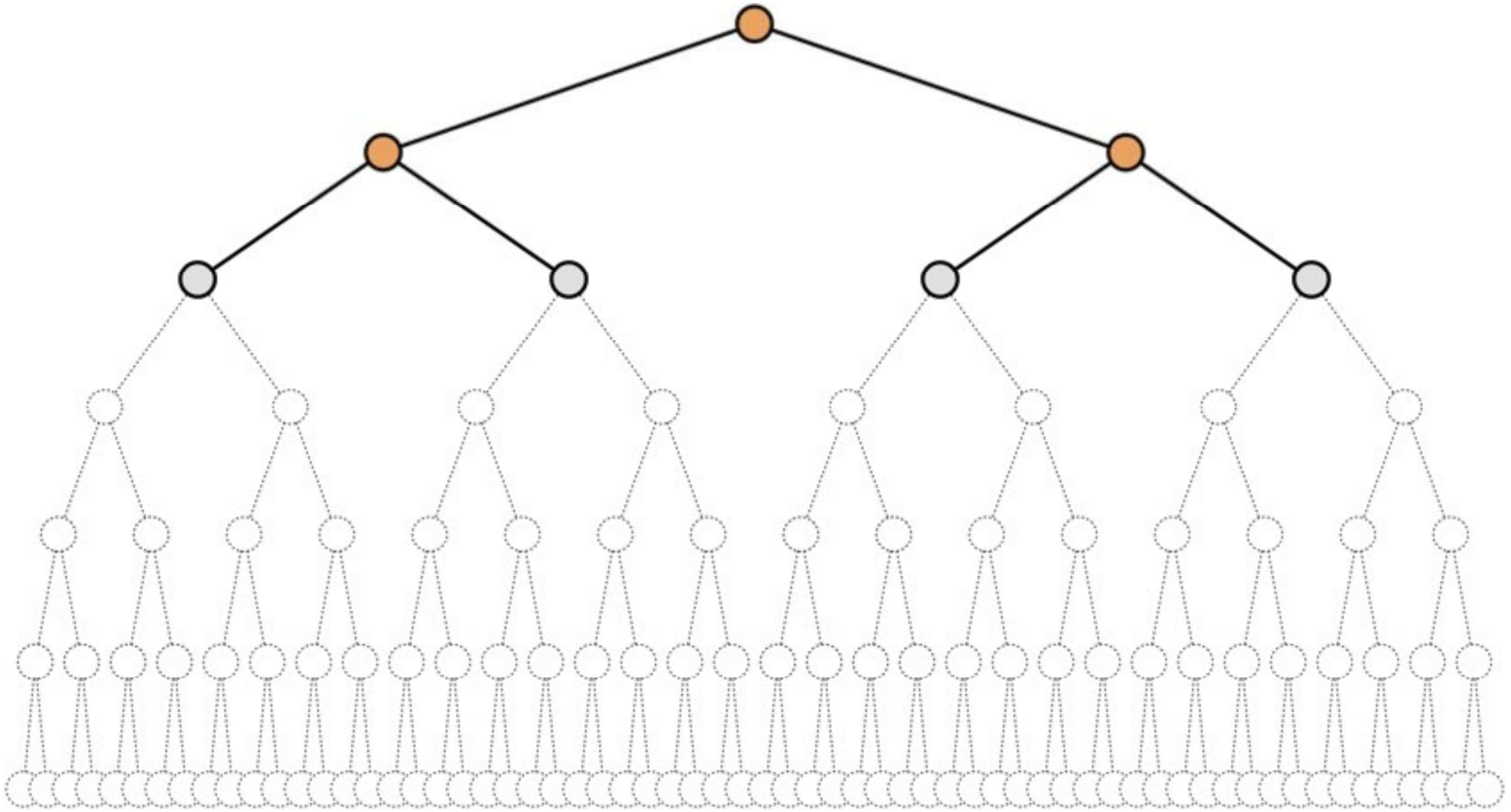
BFS



Searches layer by layer ...

... with each layer organized by node depth.

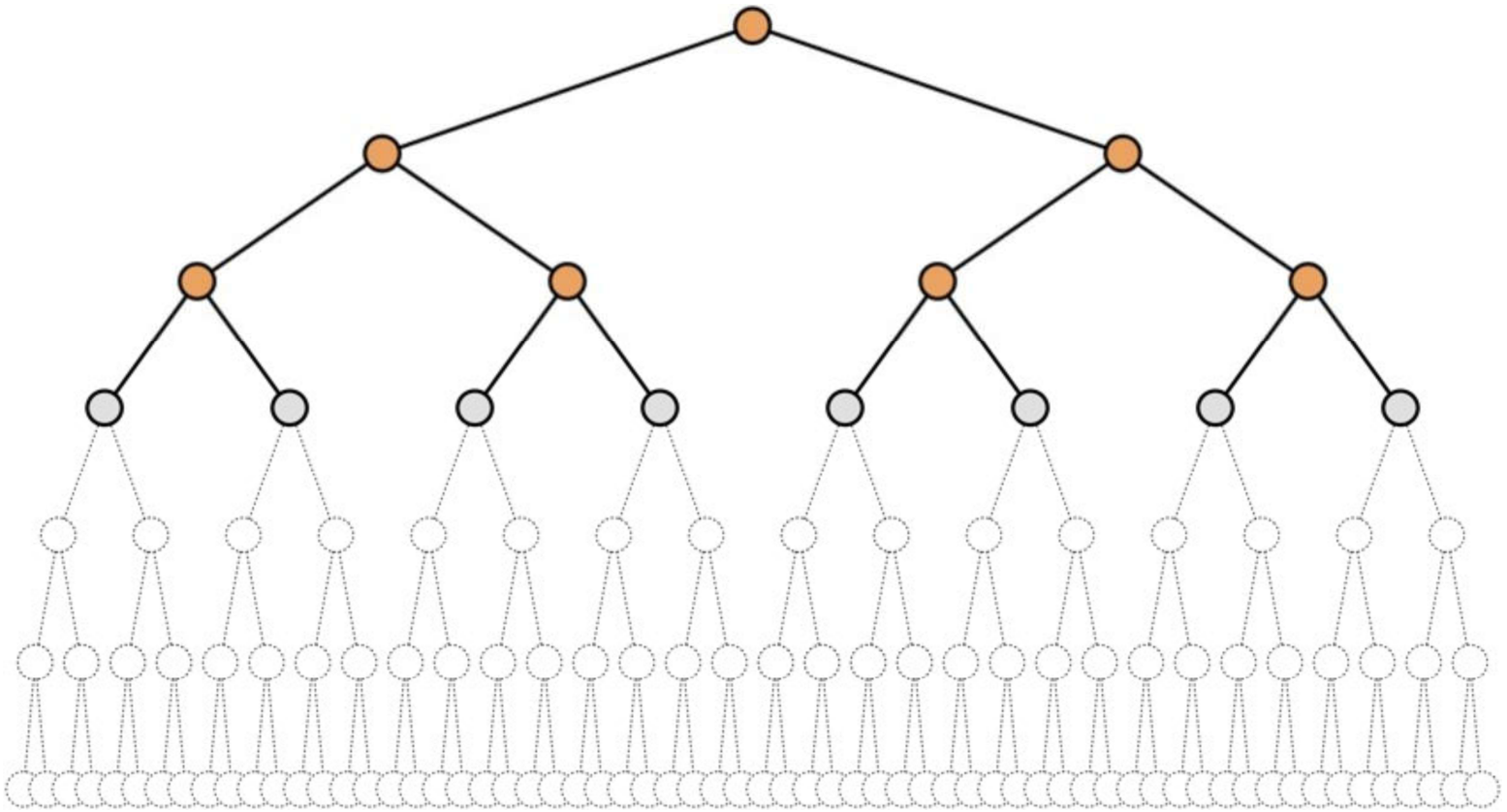
BFS



Searches layer by layer ...

... with each layer organized by node depth.

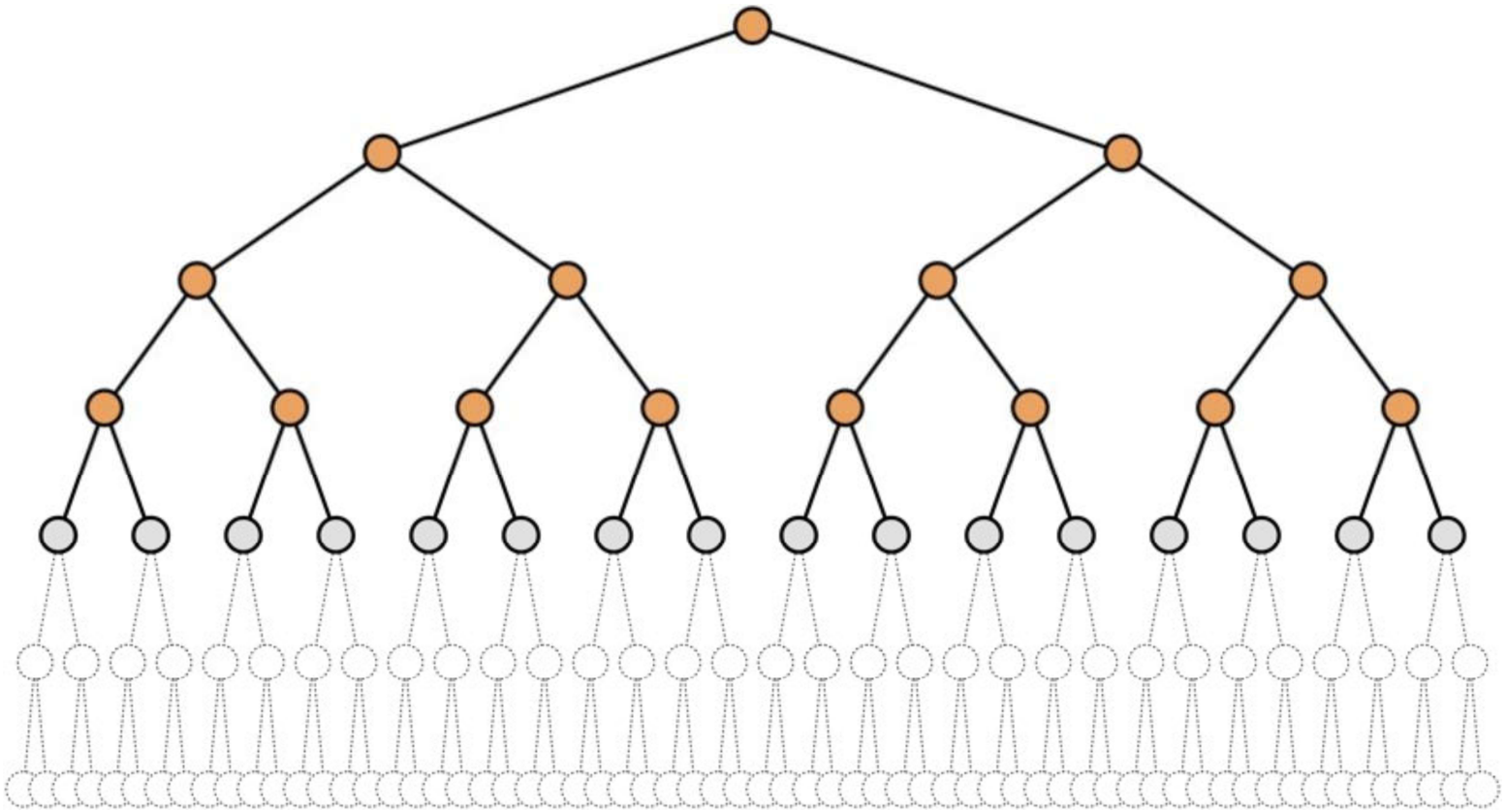
BFS



Searches layer by layer ...

... with each layer organized by node depth.

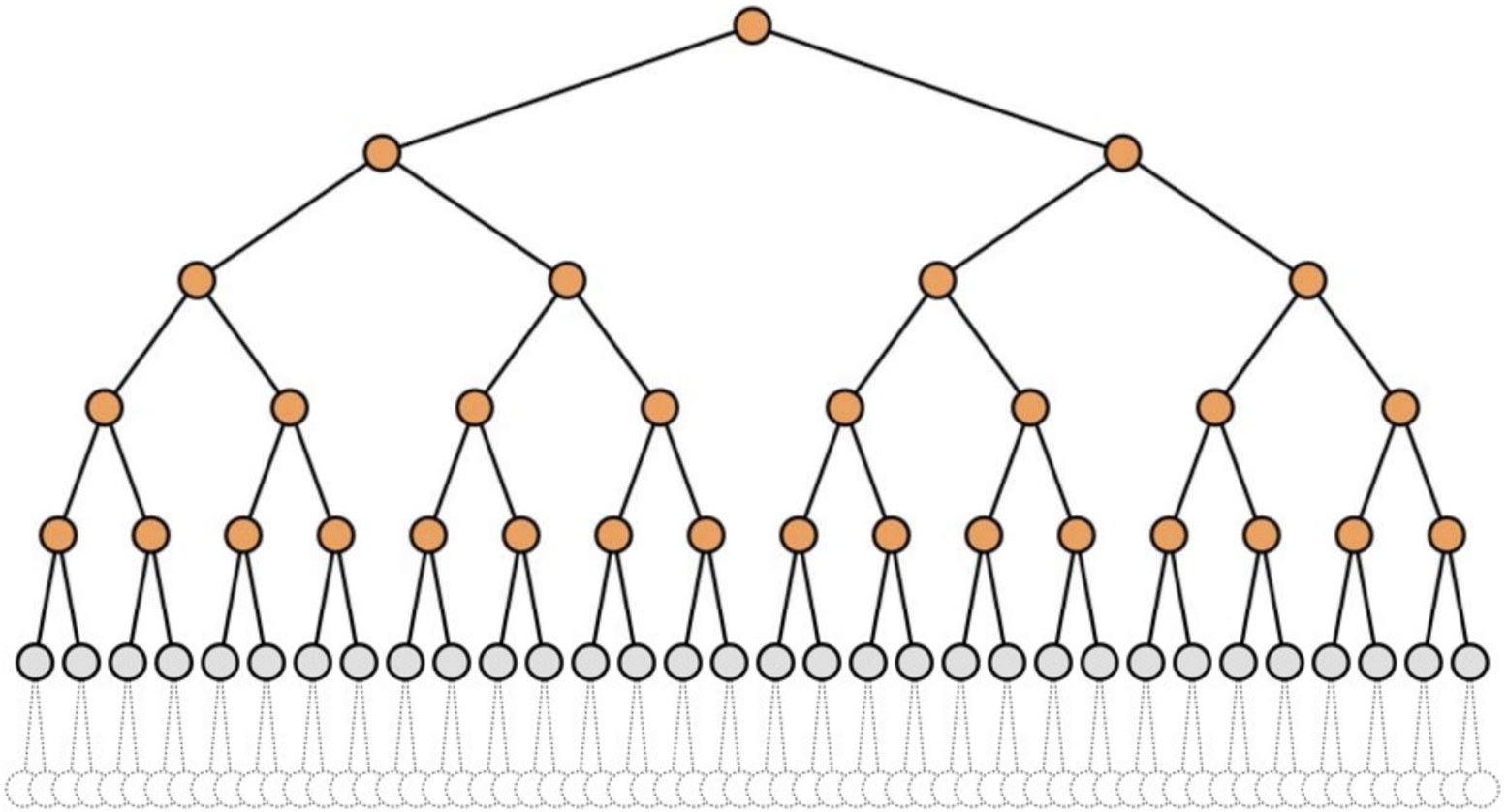
BFS



Searches layer by layer ...

... with each layer organized by node depth.

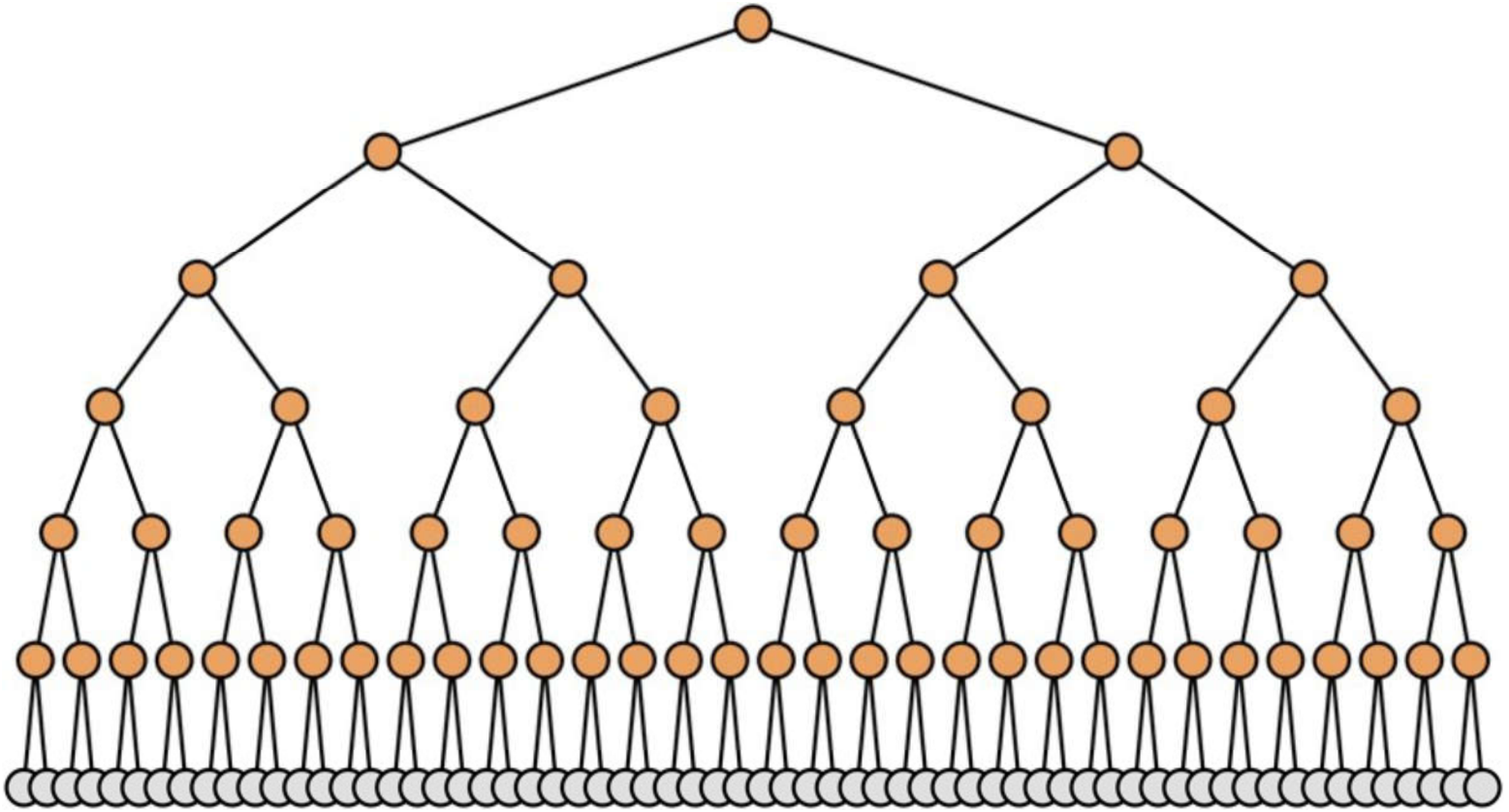
BFS



Searches layer by layer ...

... with each layer organized by node depth.

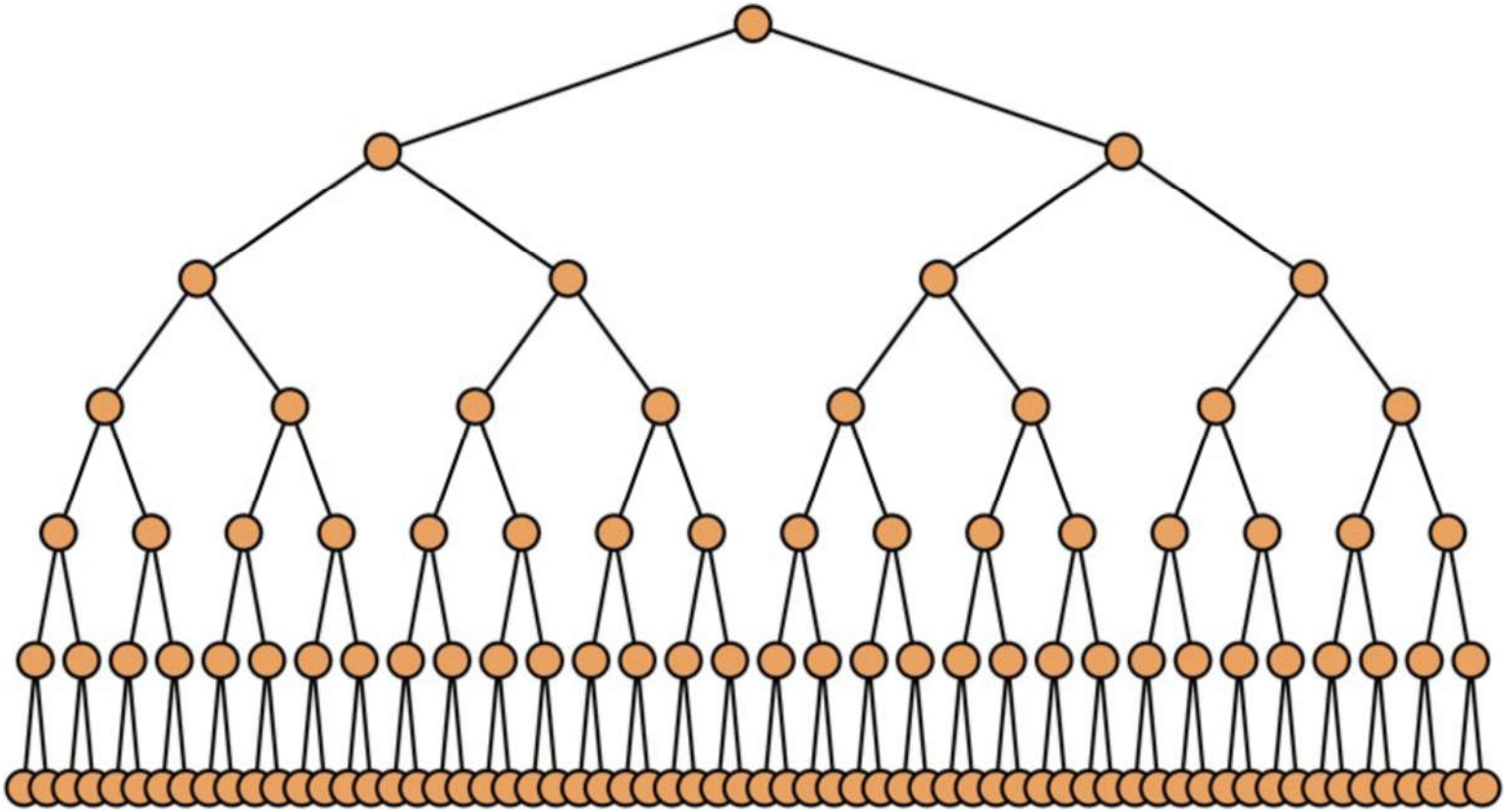
BFS



Searches layer by layer ...

... with each layer organized by node depth.

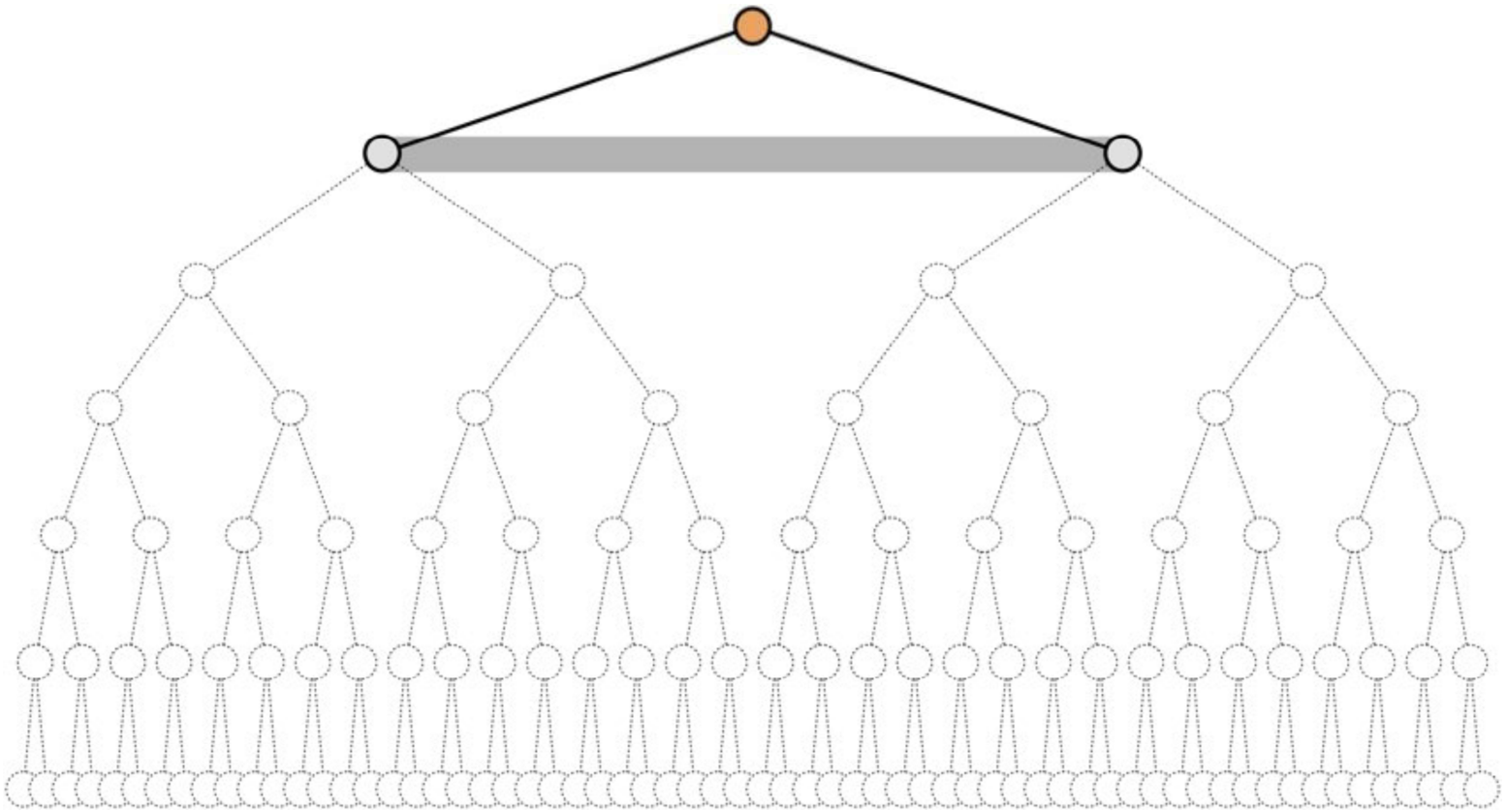
BFS



Searches layer by layer ...

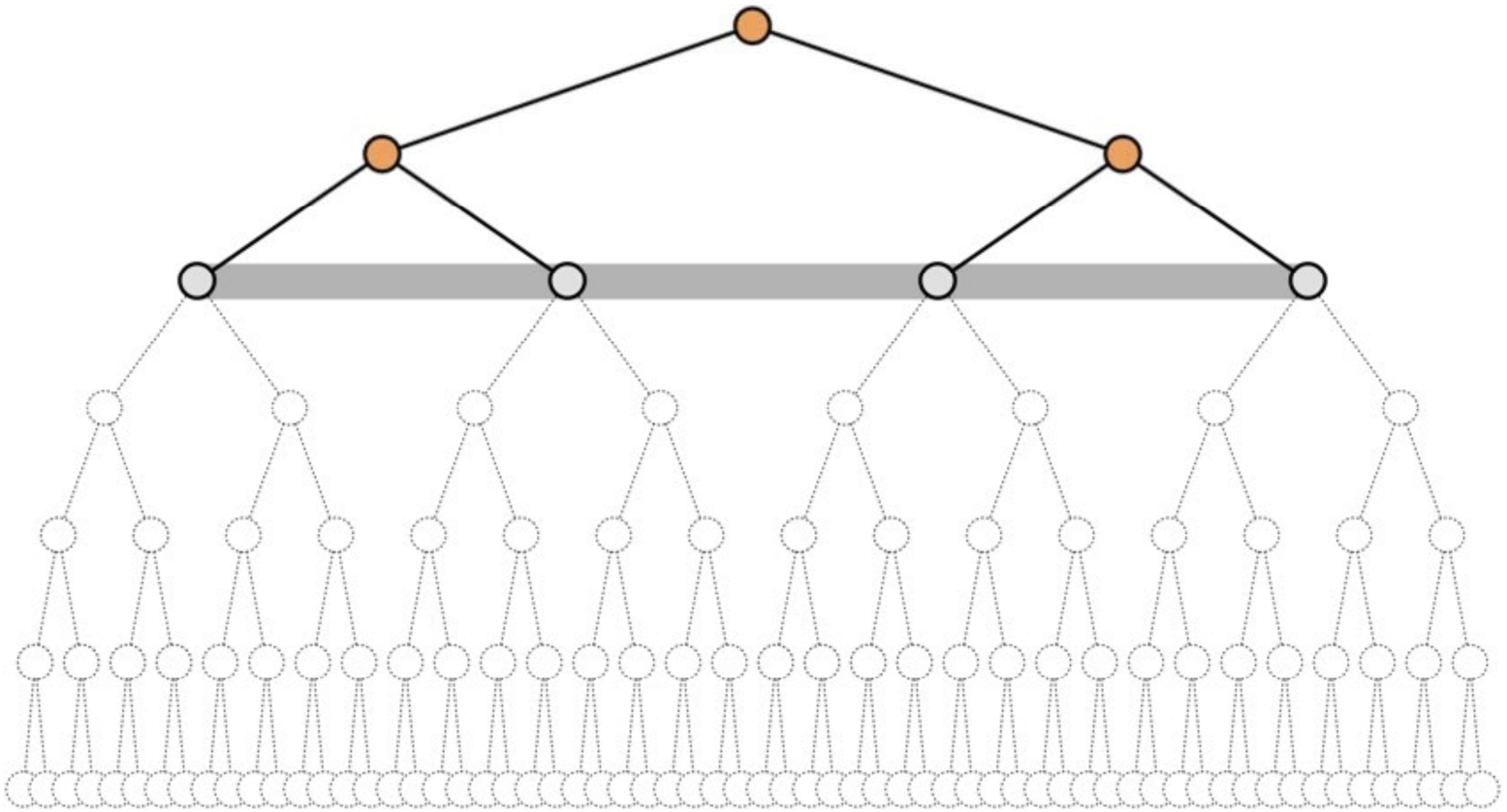
... with each layer organized by node depth.

BFS



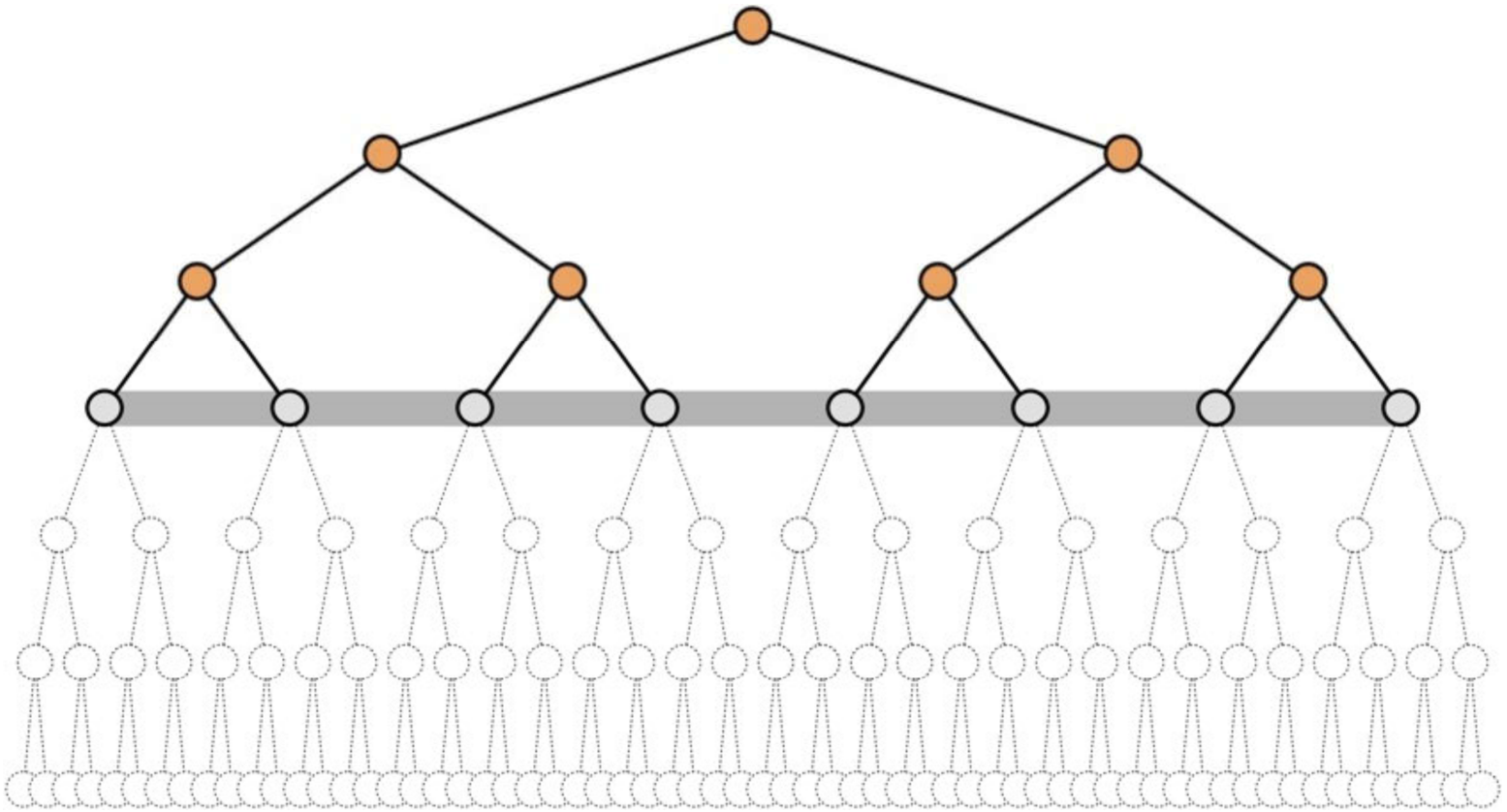
The frontier consists of nodes of similar depth (horizontal).
Search proceeds by exhausting one layer at a time.

BFS



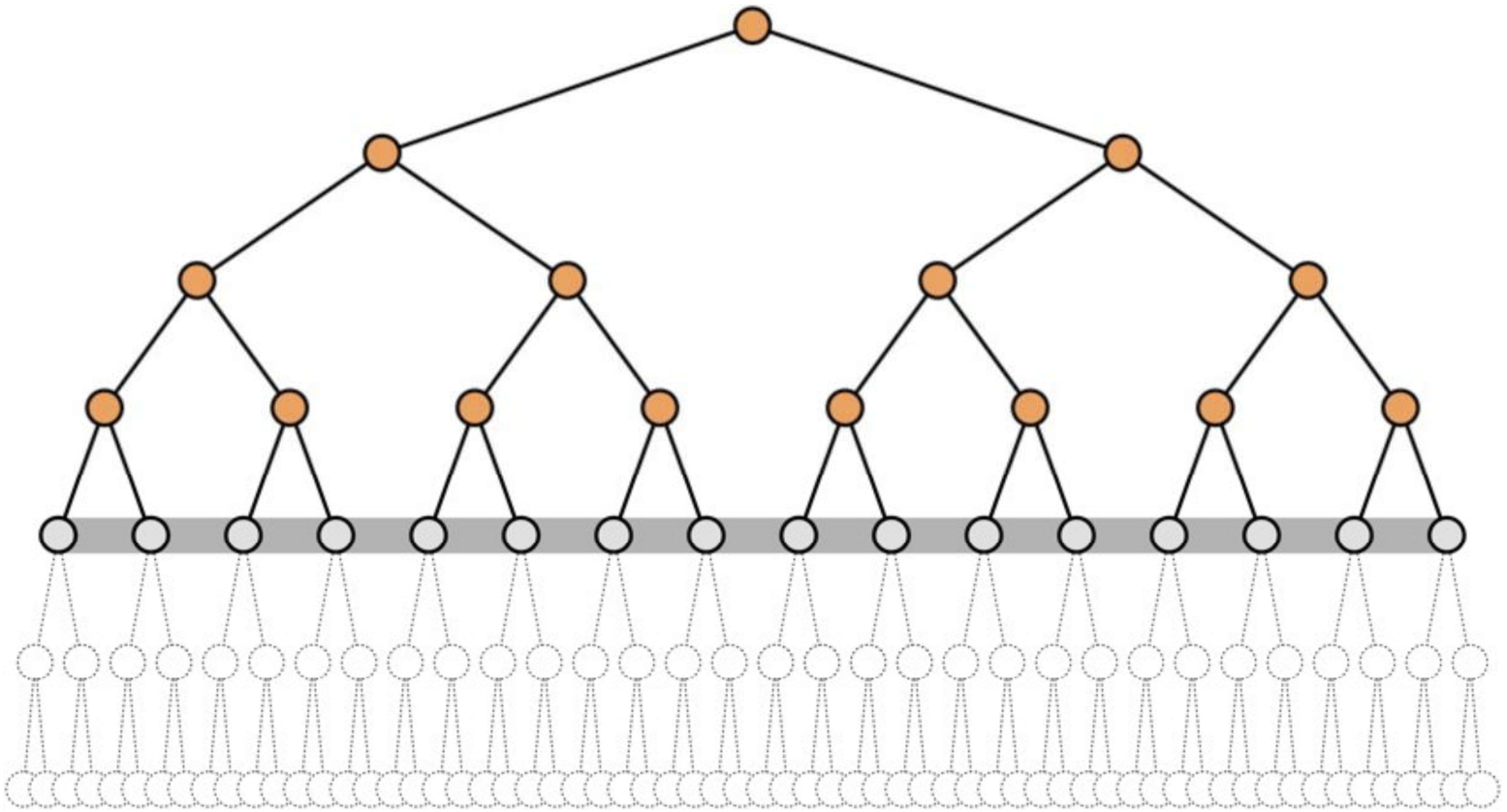
The frontier consists of nodes of similar depth (horizontal).
Search proceeds by exhausting one layer at a time.

BFS



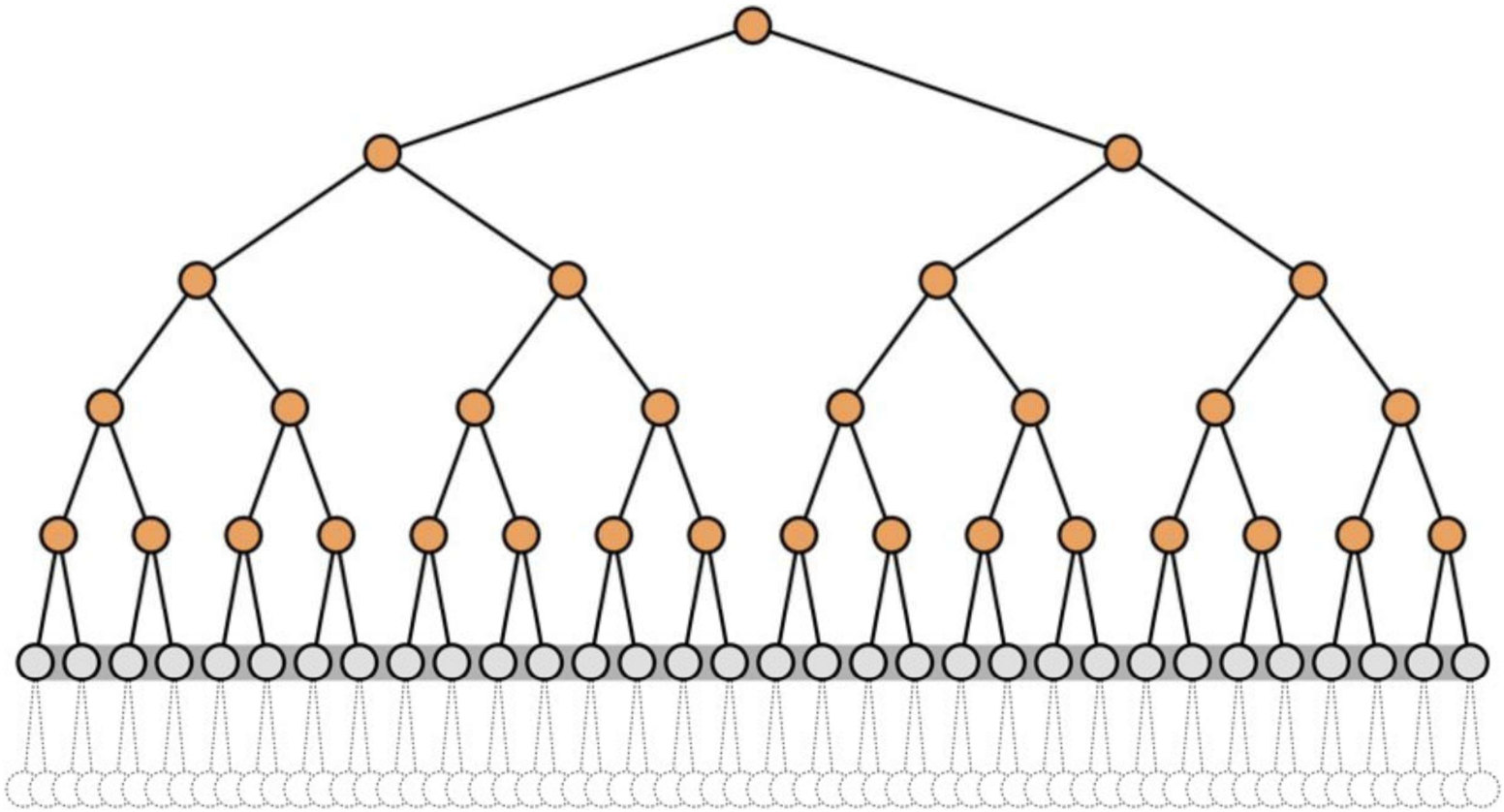
The frontier consists of nodes of similar depth (horizontal).
Search proceeds by exhausting one layer at a time.

BFS



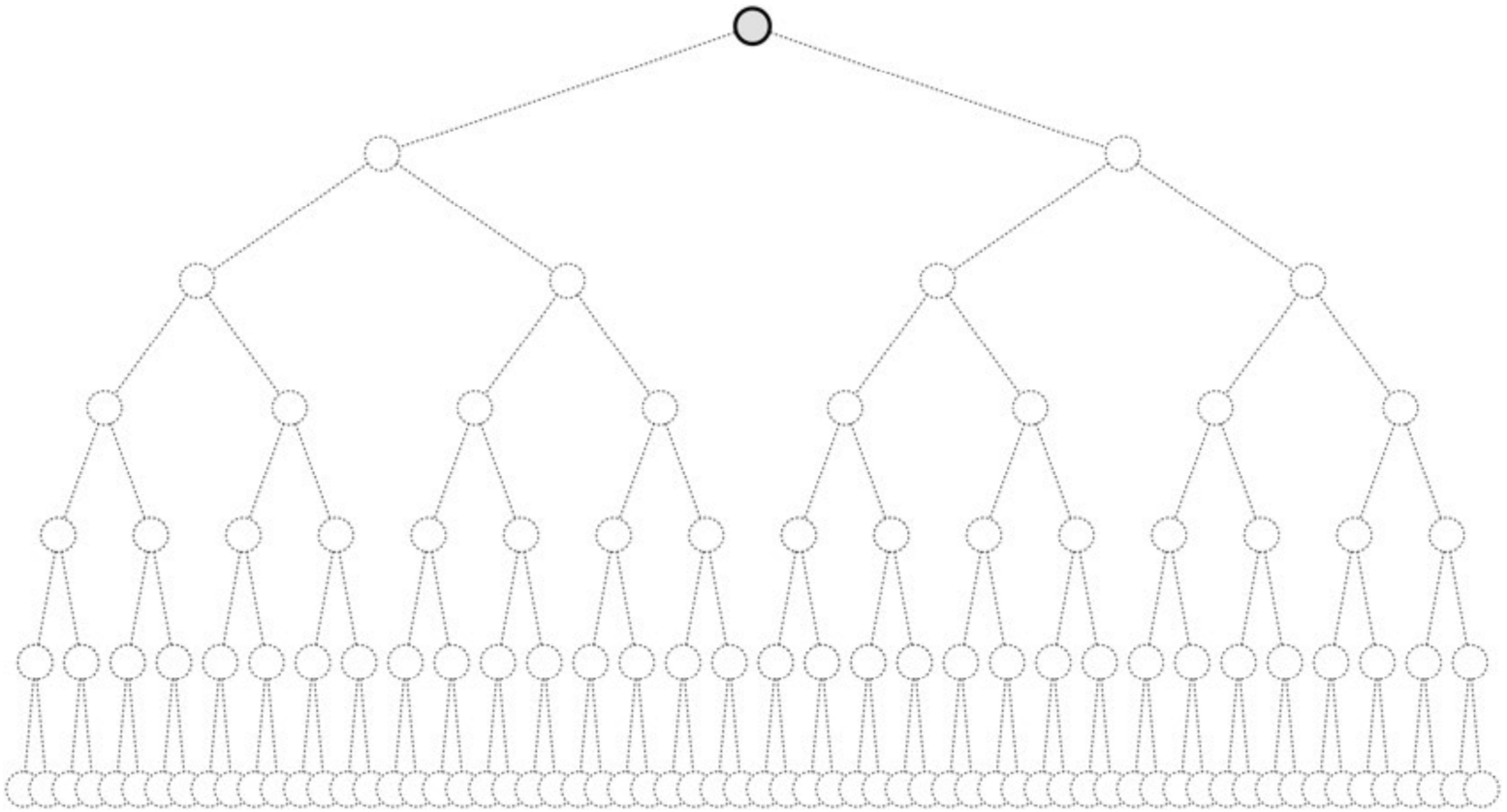
The frontier consists of nodes of similar depth (horizontal).
Search proceeds by exhausting one layer at a time.

BFS



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

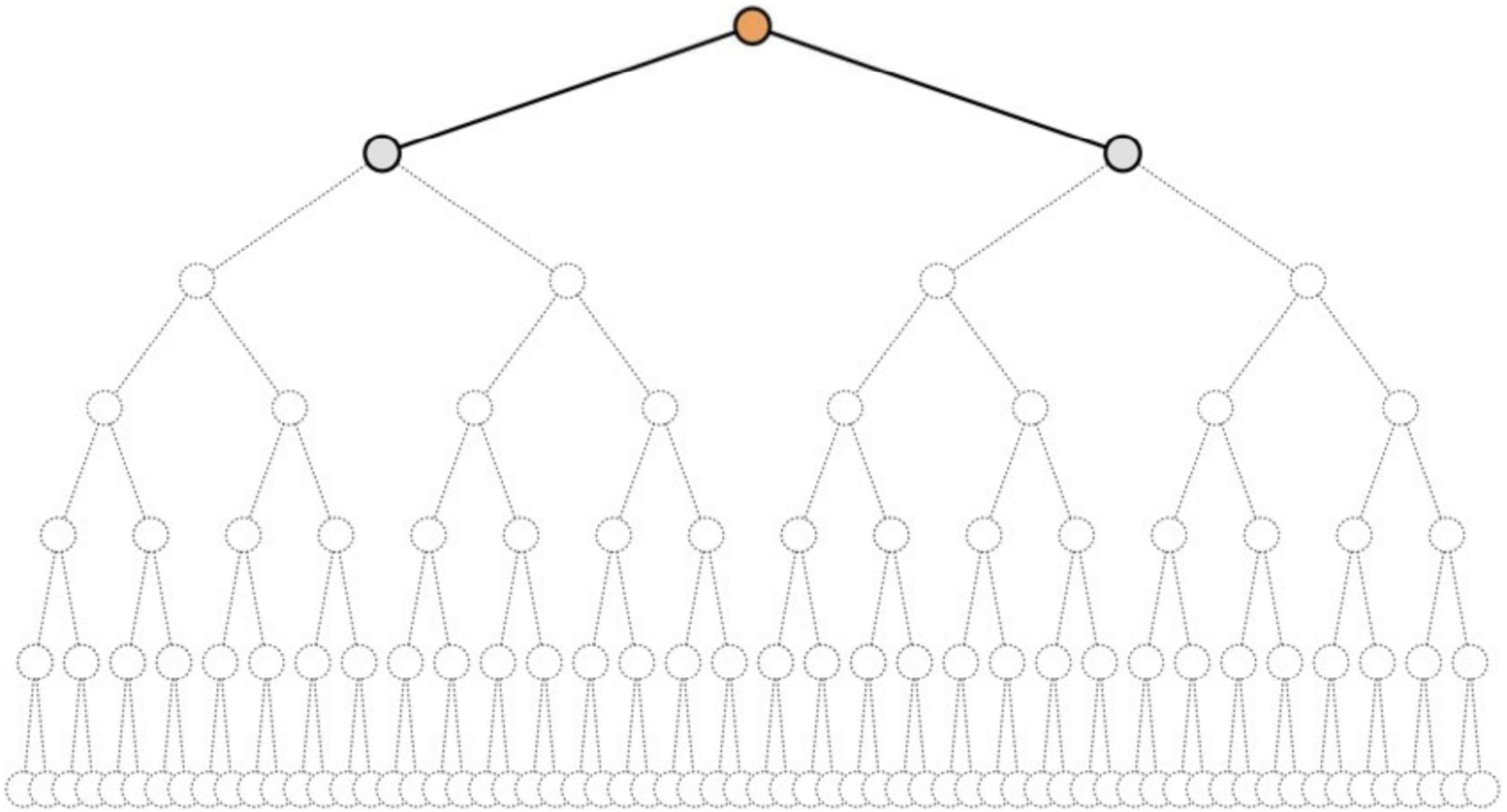
UCS



Searches layer by layer ...

... with each layer organized by path cost.

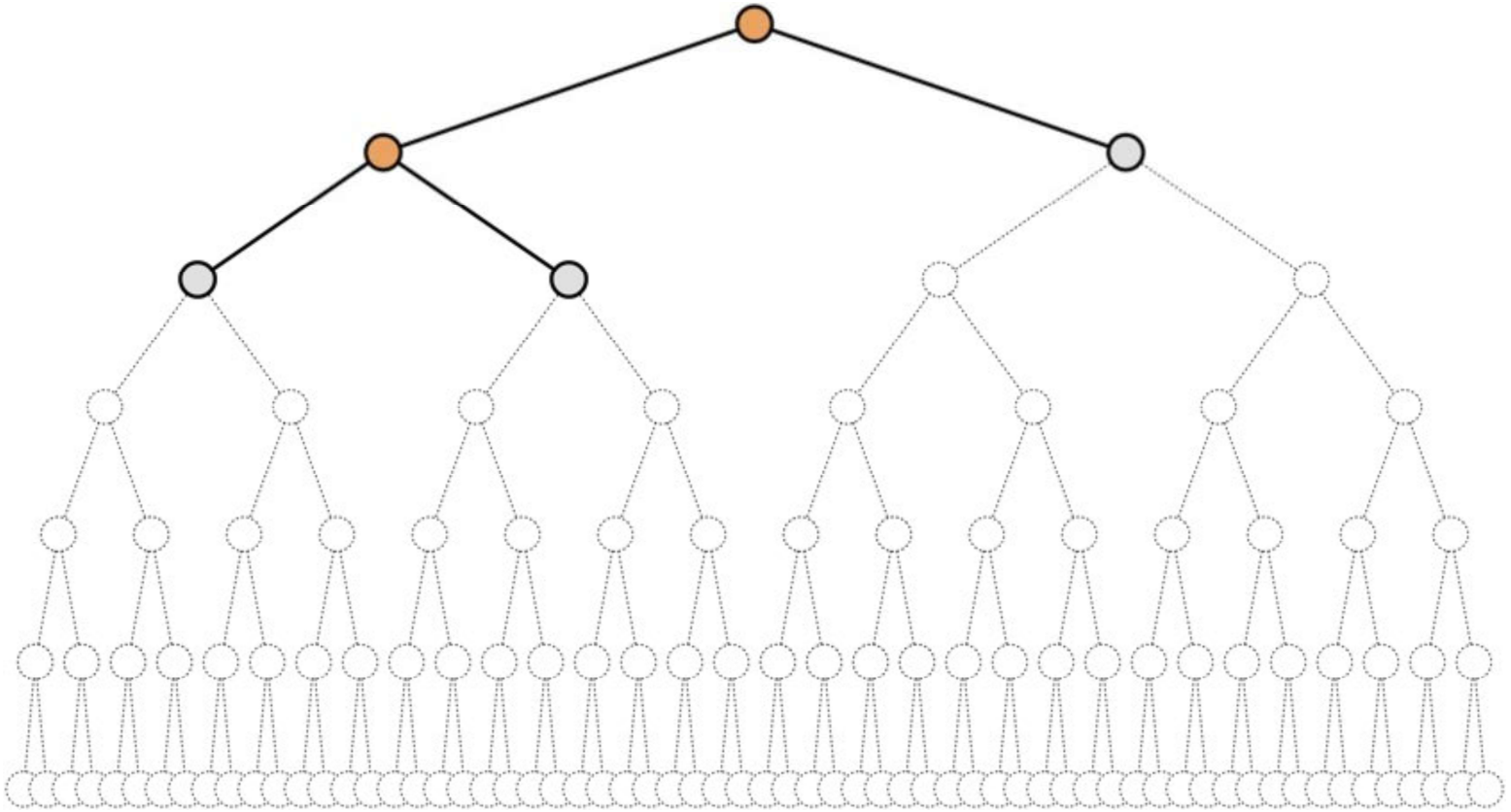
UCS



Searches layer by layer ...

... with each layer organized by path cost.

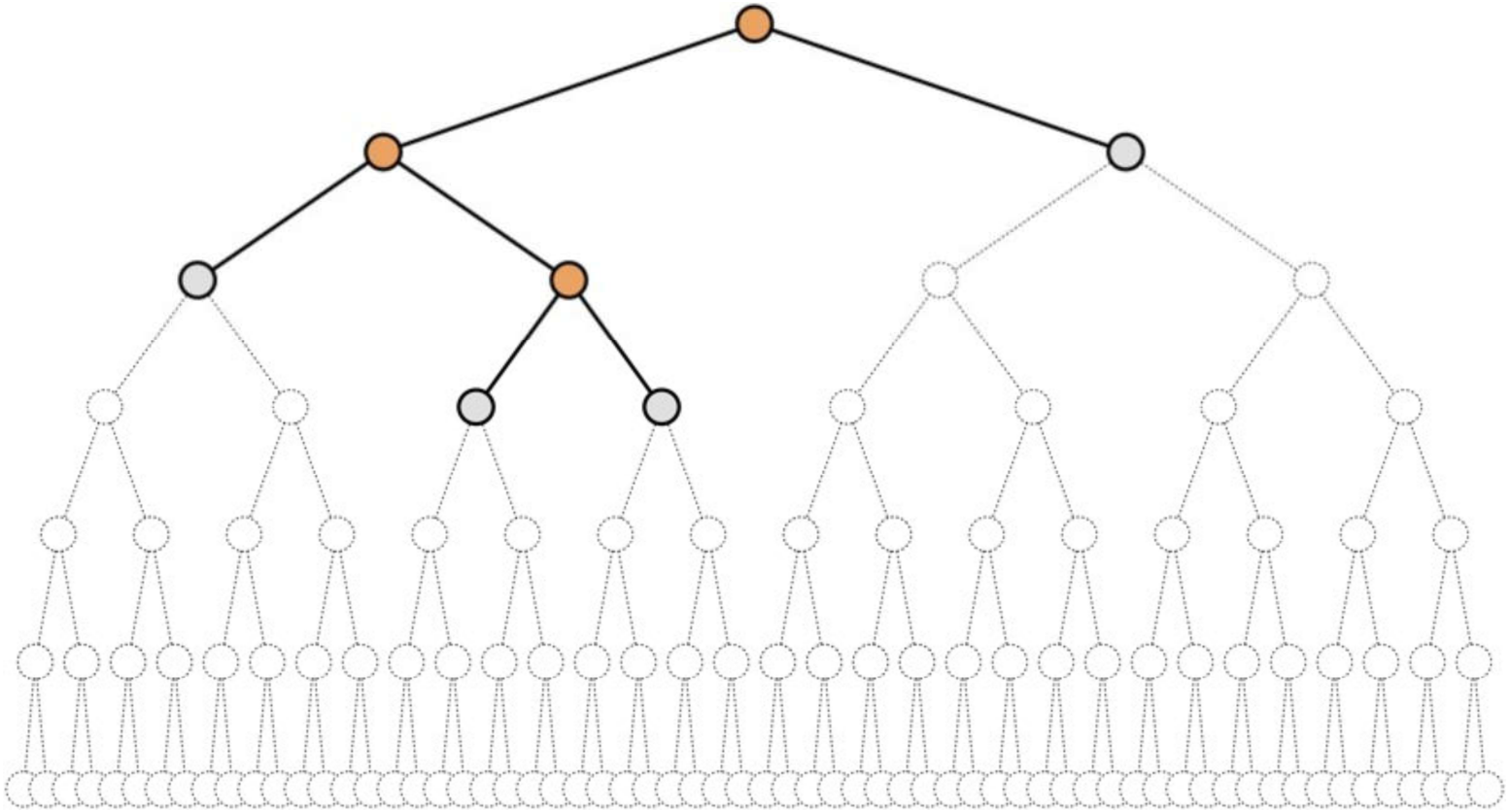
UCS



Searches layer by layer ...

... with each layer organized by path cost.

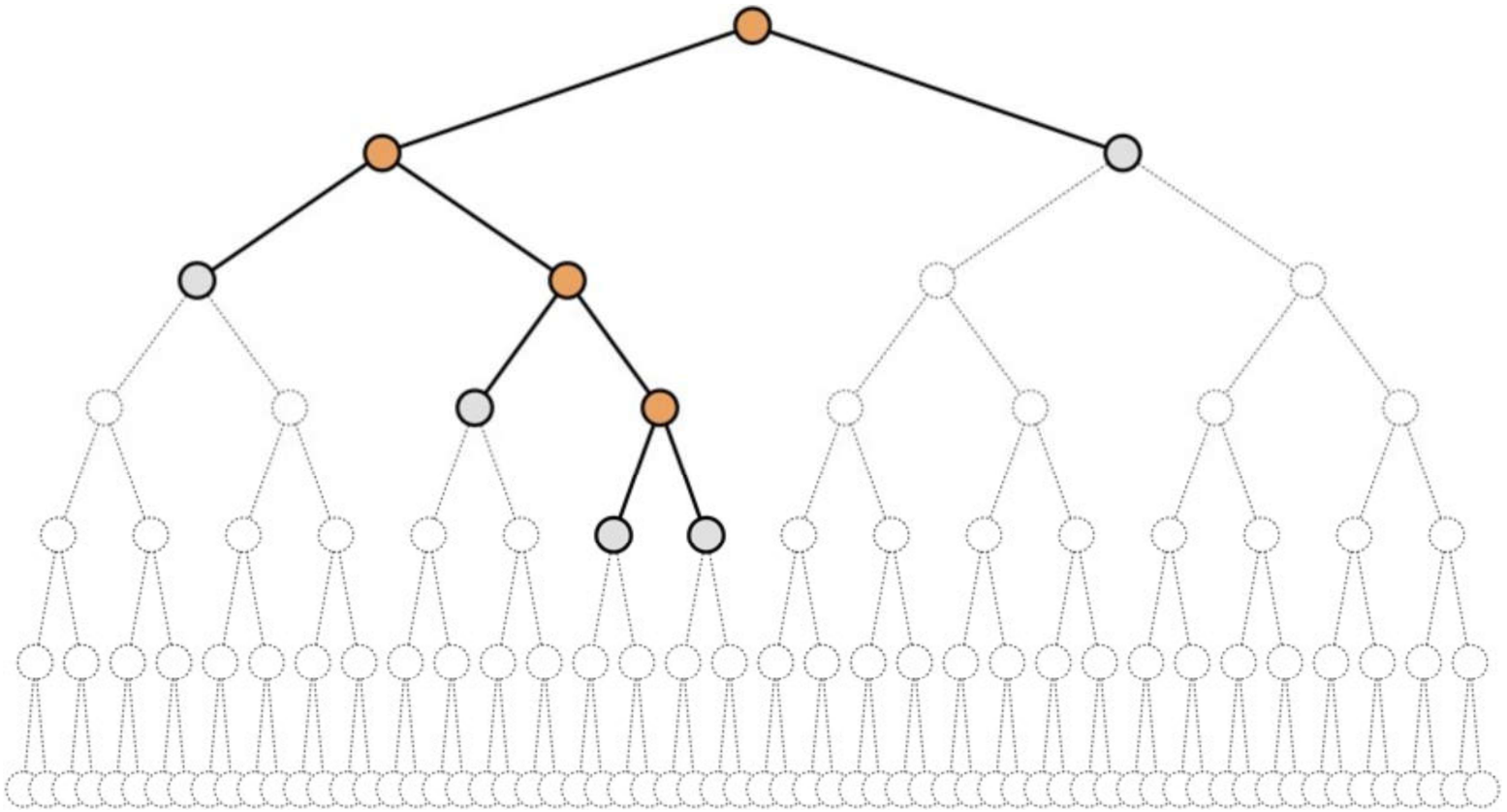
UCS



Searches layer by layer ...

... with each layer organized by path cost.

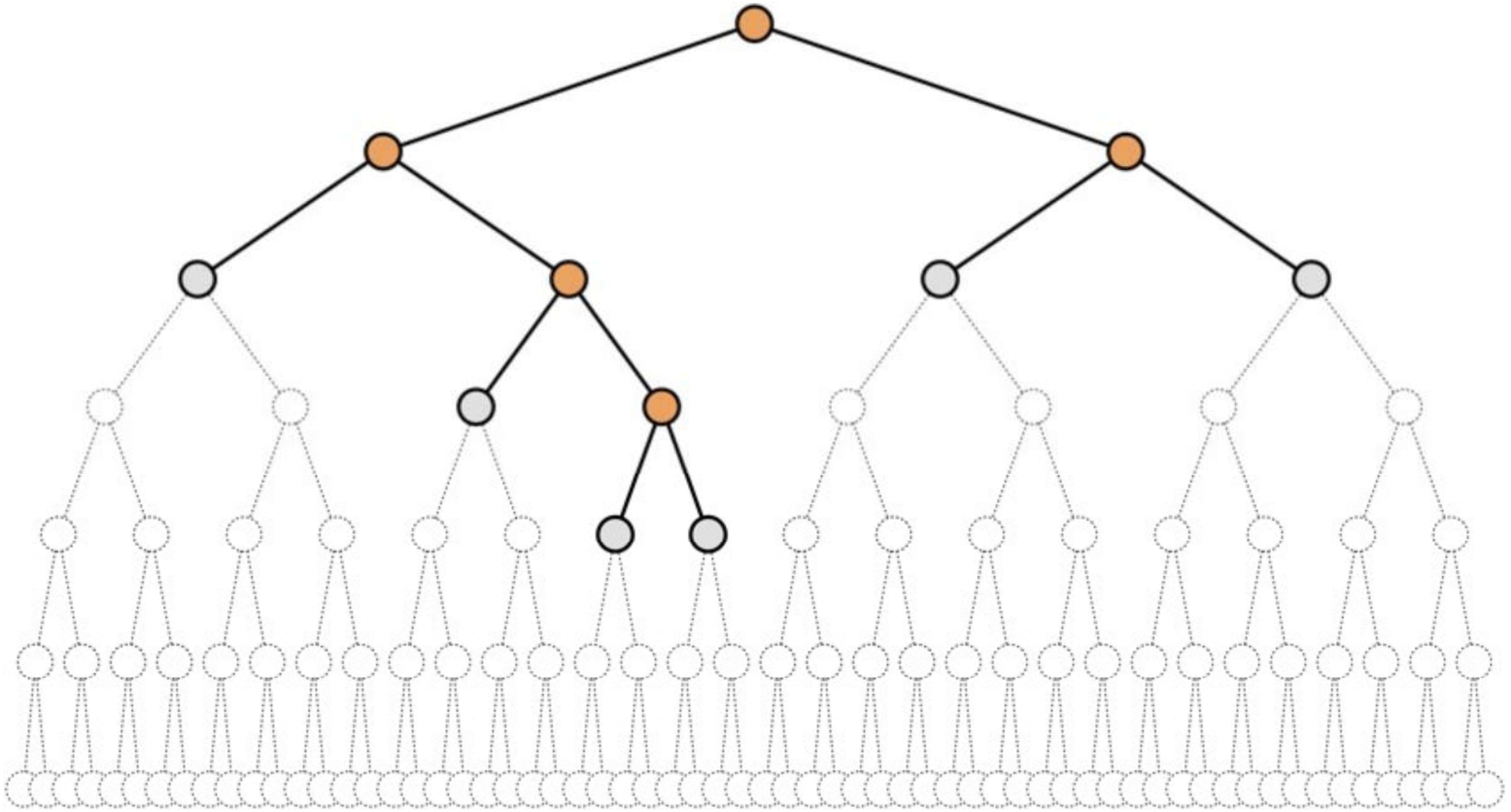
UCS



Searches layer by layer ...

... with each layer organized by path cost.

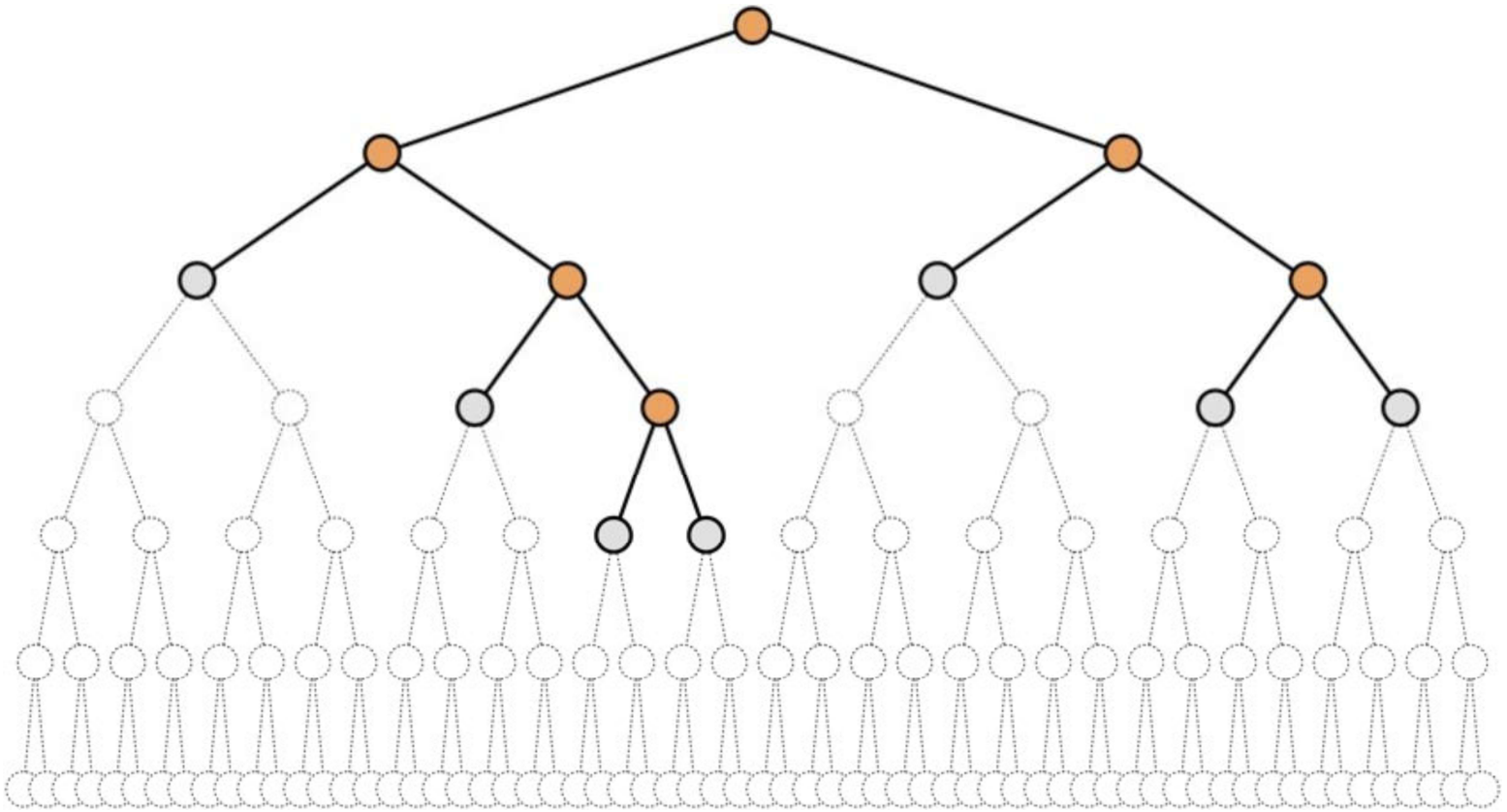
UCS



Searches layer by layer ...

... with each layer organized by path cost.

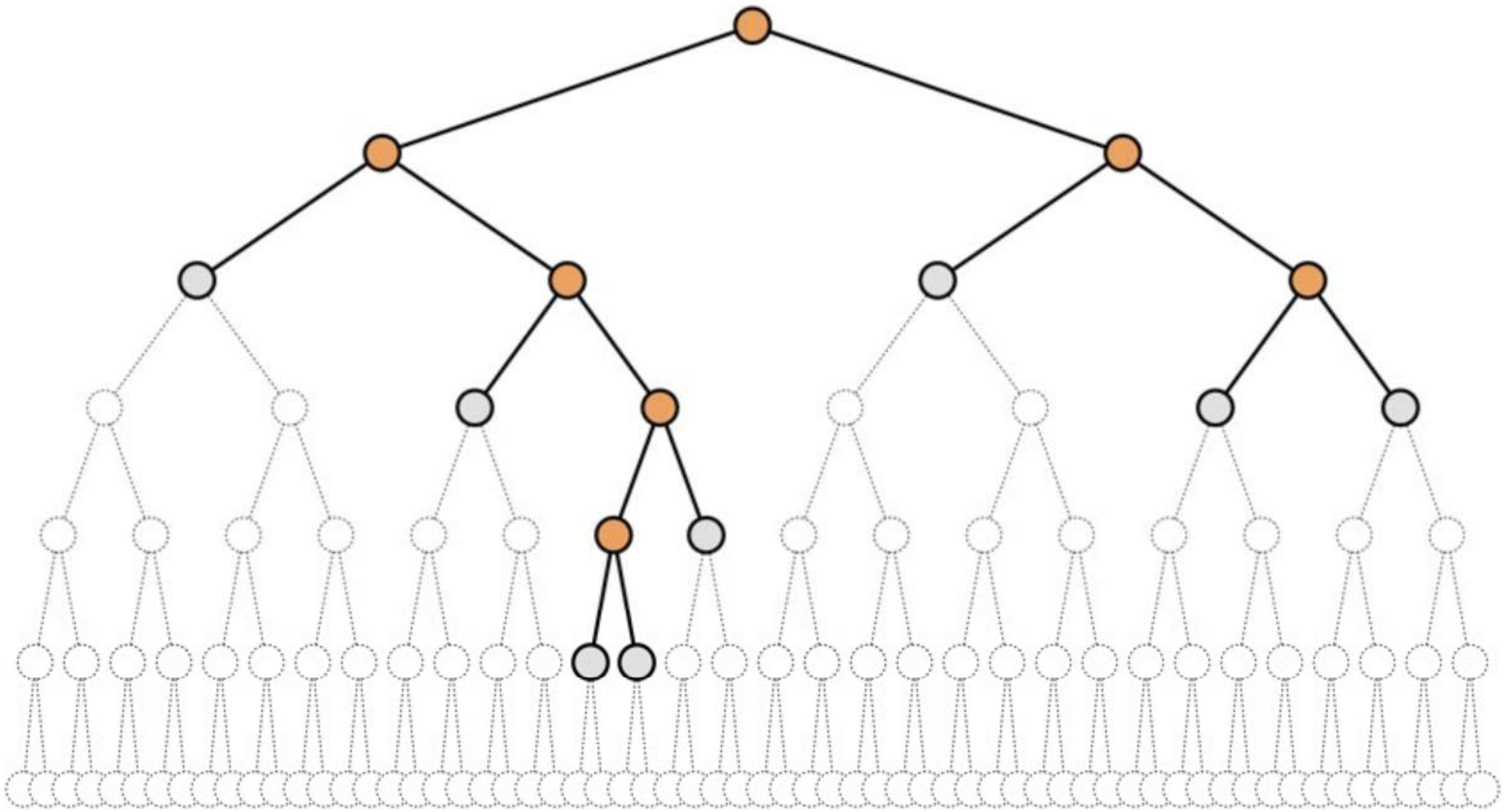
UCS



Searches layer by layer ...

... with each layer organized by path cost.

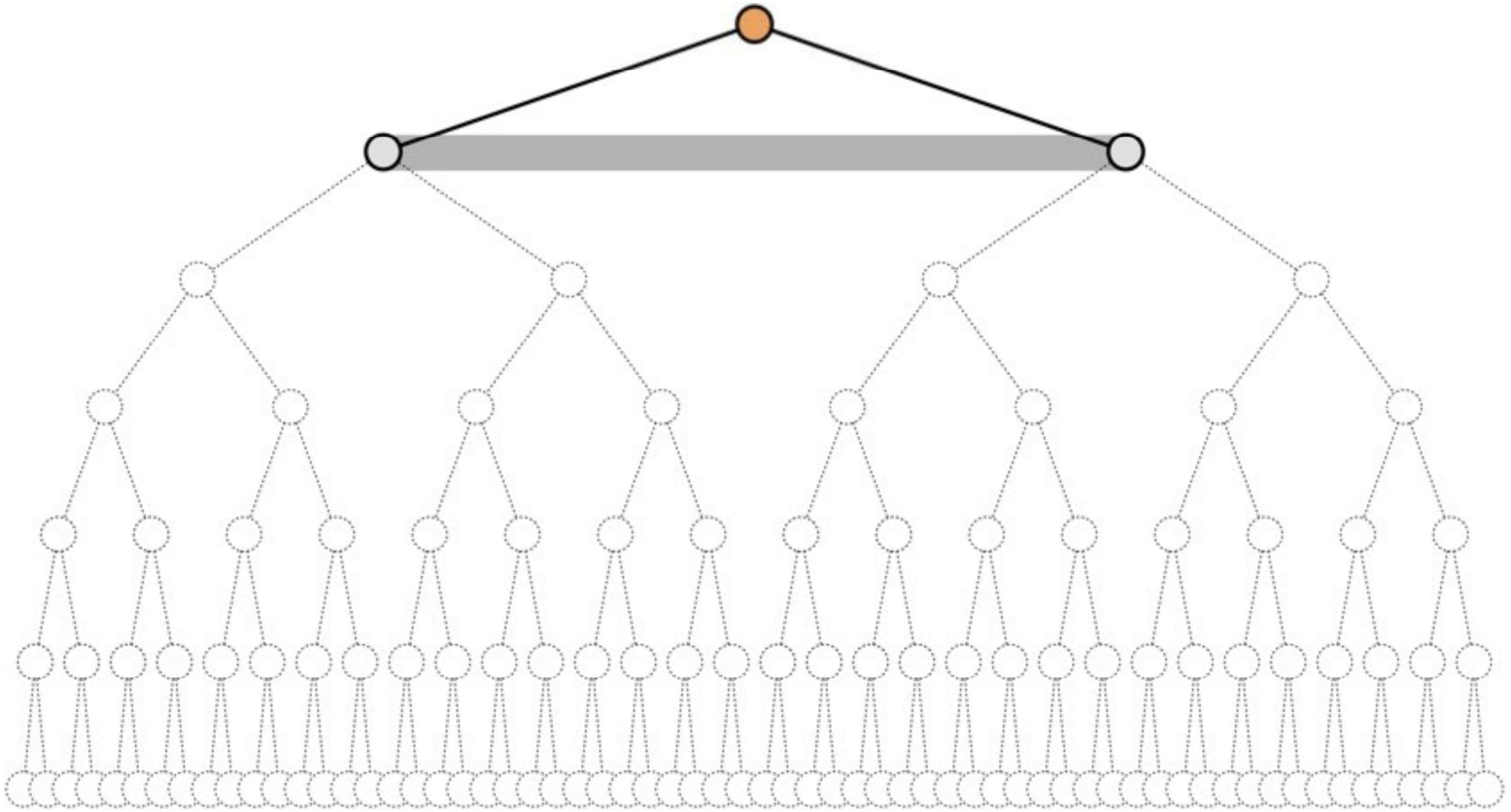
UCS



Searches layer by layer ...

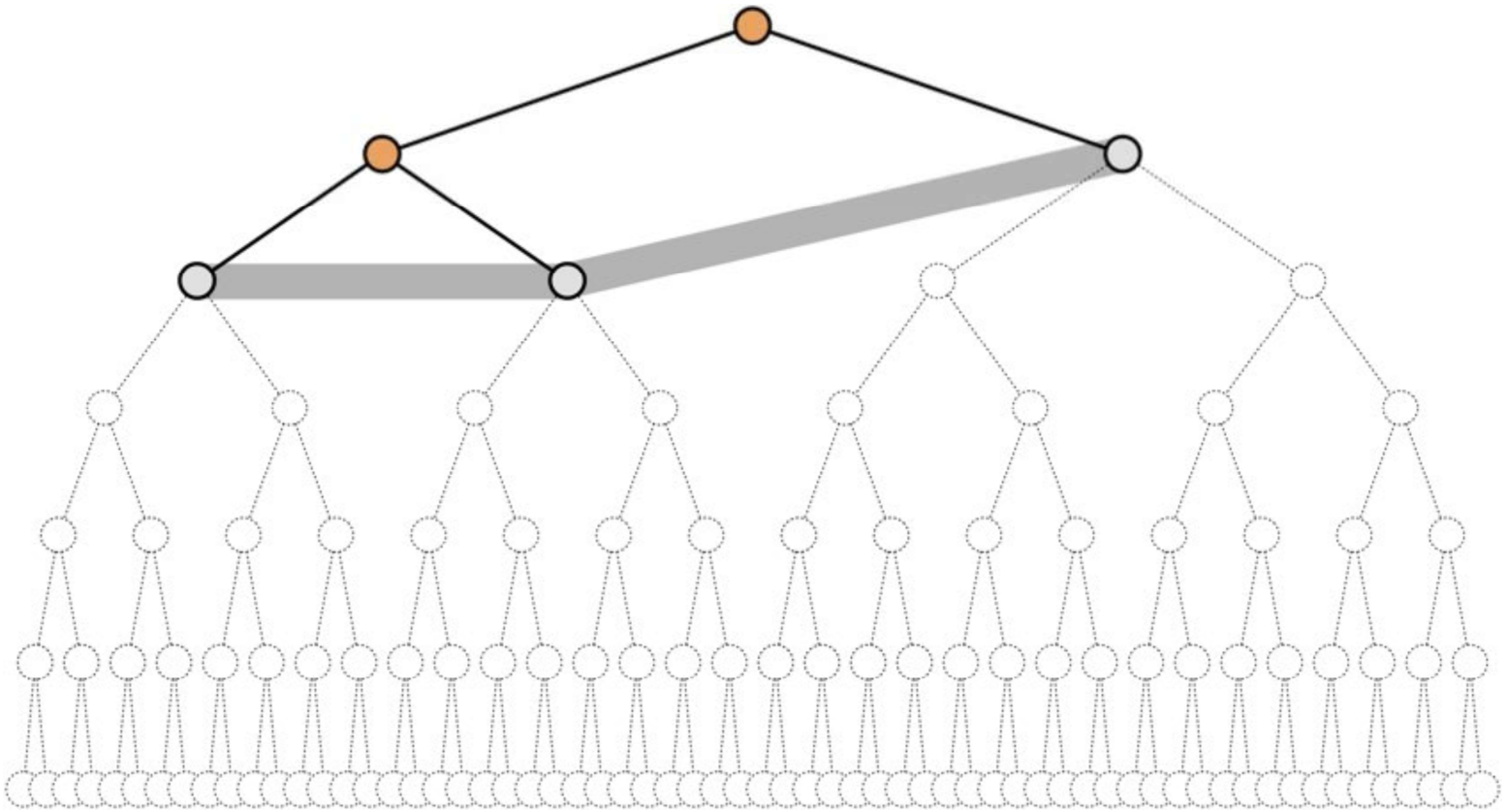
... with each layer organized by path cost.

UCS



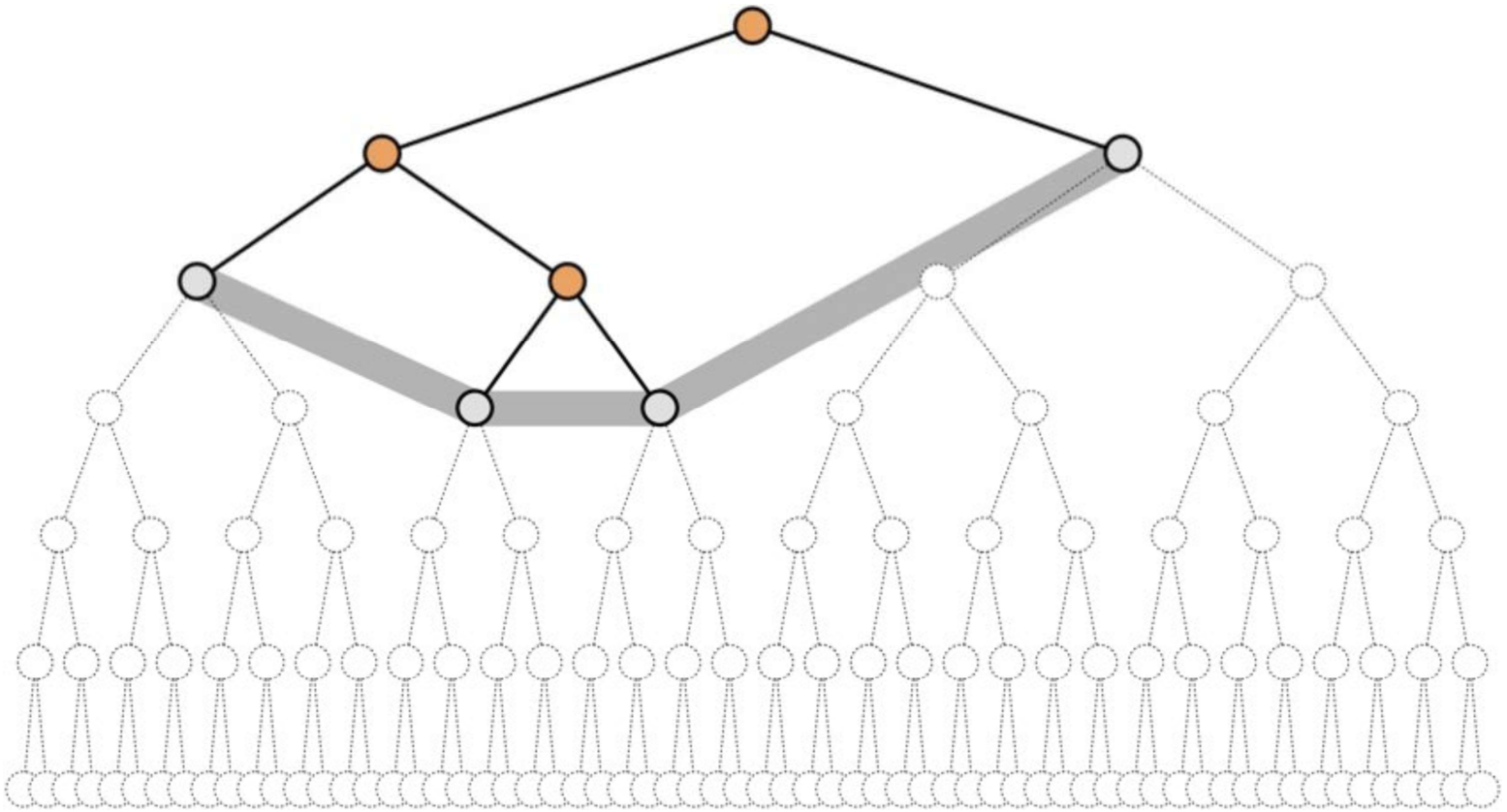
The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.

UCS



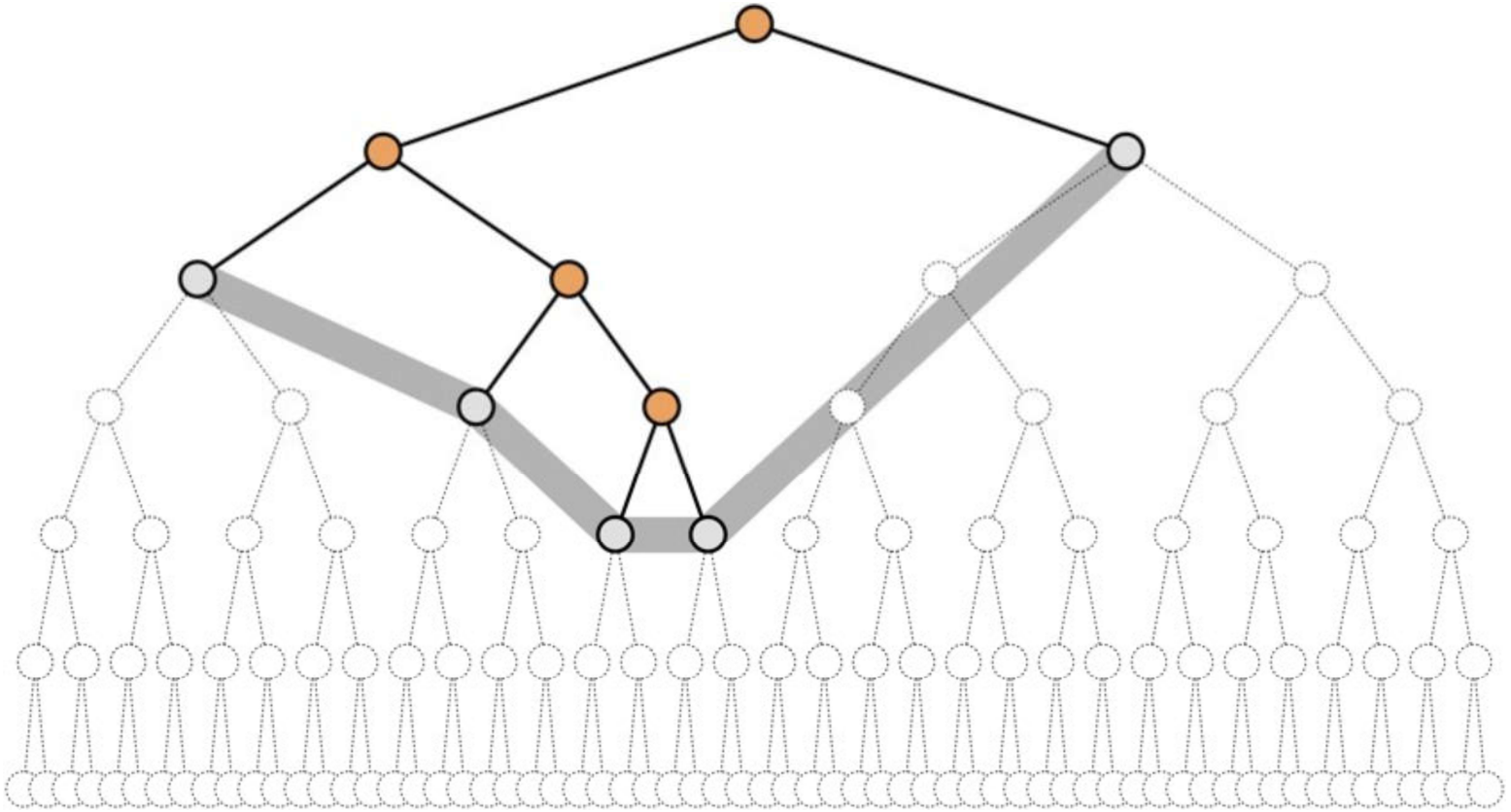
The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.

UCS



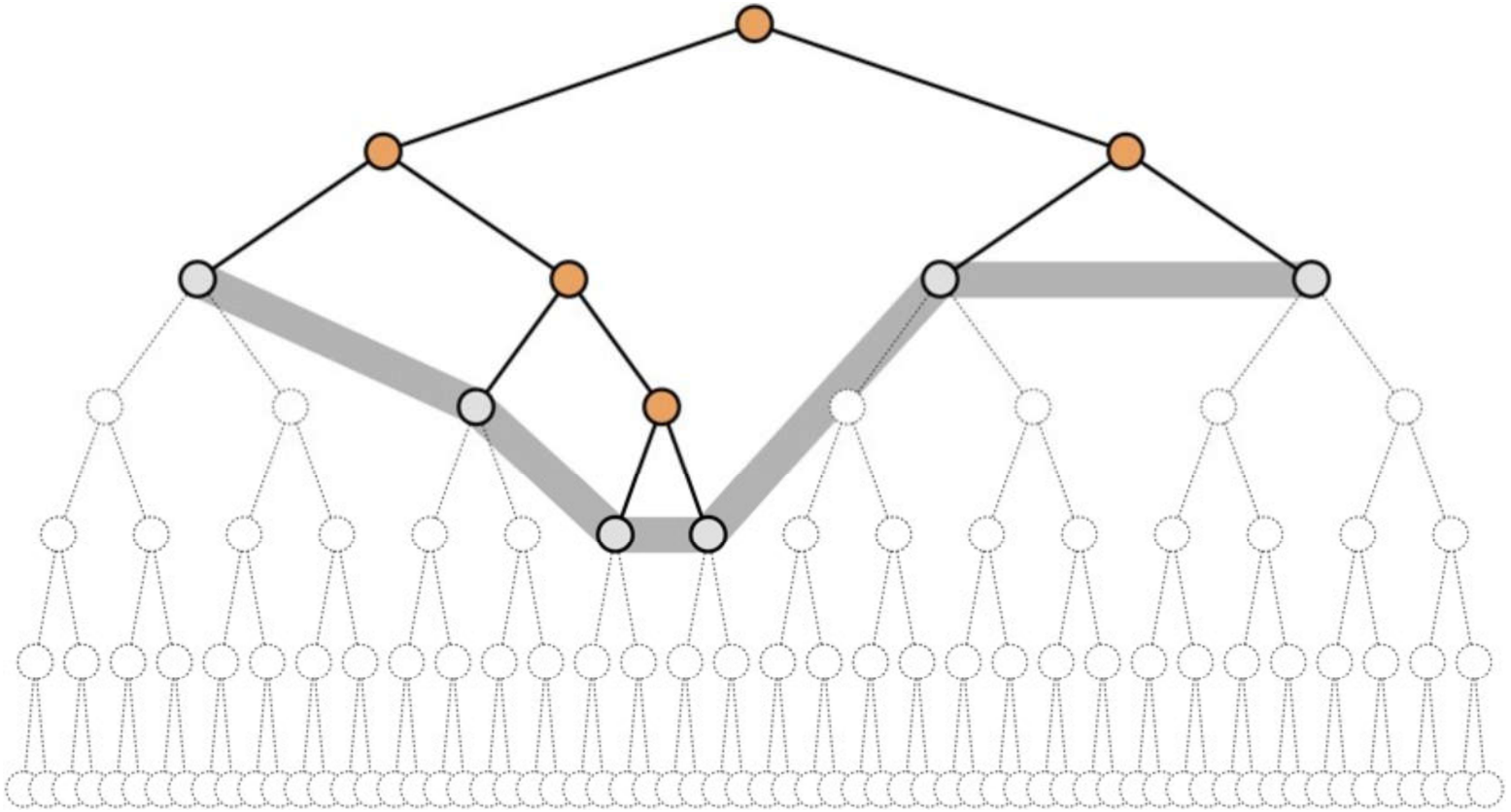
The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.

UCS



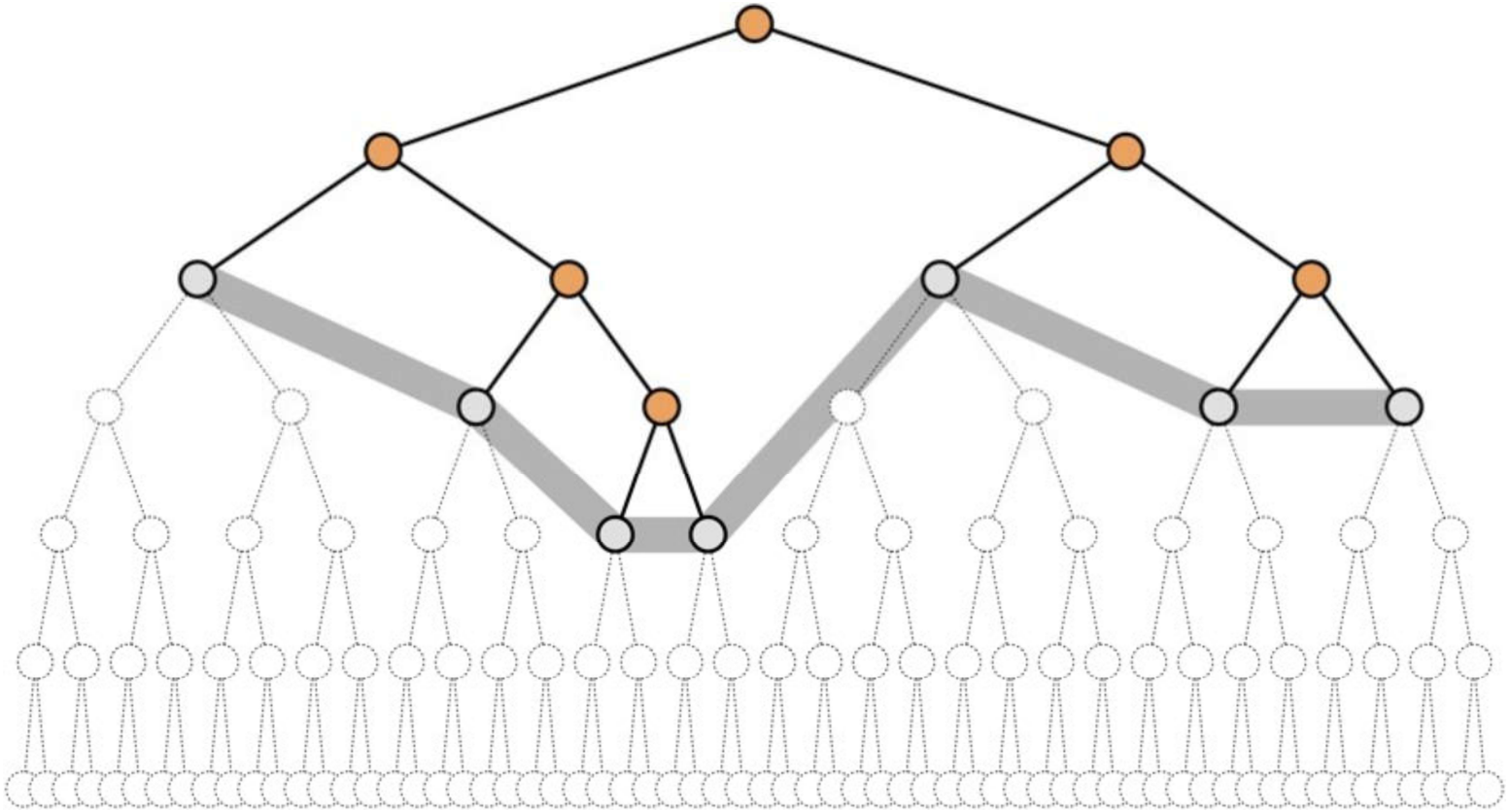
The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.

UCS



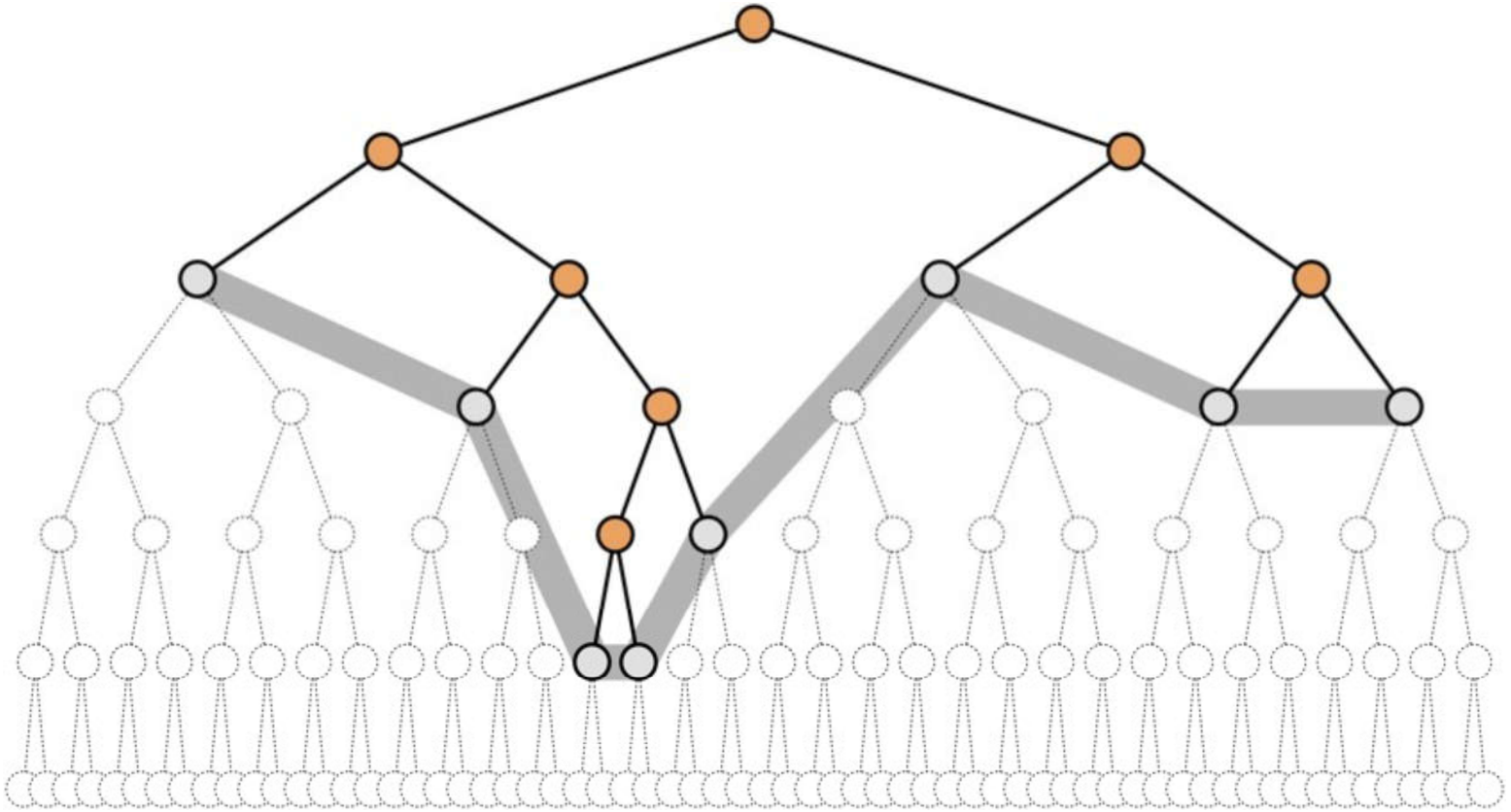
The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.

UCS



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

UCS



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

Recap

We can organize the algorithms into pairs where the first proceeds by layers, and the other proceeds by subtrees.

(1) Iterate on Node Depth:

- BFS searches layers of increasing node depth.
- IDS searches subtrees of increasing node depth.

Recap

We can organize the algorithms into pairs where the first proceeds by layers, and the other proceeds by subtrees.

(1) Iterate on Node Depth:

- BFS searches layers of increasing node depth.
- IDS searches subtrees of increasing node depth.

(2) Iterate on Path Cost + Heuristic Function:

- A* searches layers of increasing path cost + heuristic function.
- IDA* searches subtrees of increasing path cost + heuristic function.

Recap

Which cost function?

- UCS searches layers of increasing path cost.
- Greedy best first search searches layers of increasing heuristic function.
- A* search searches layers of increasing path cost + heuristic function.