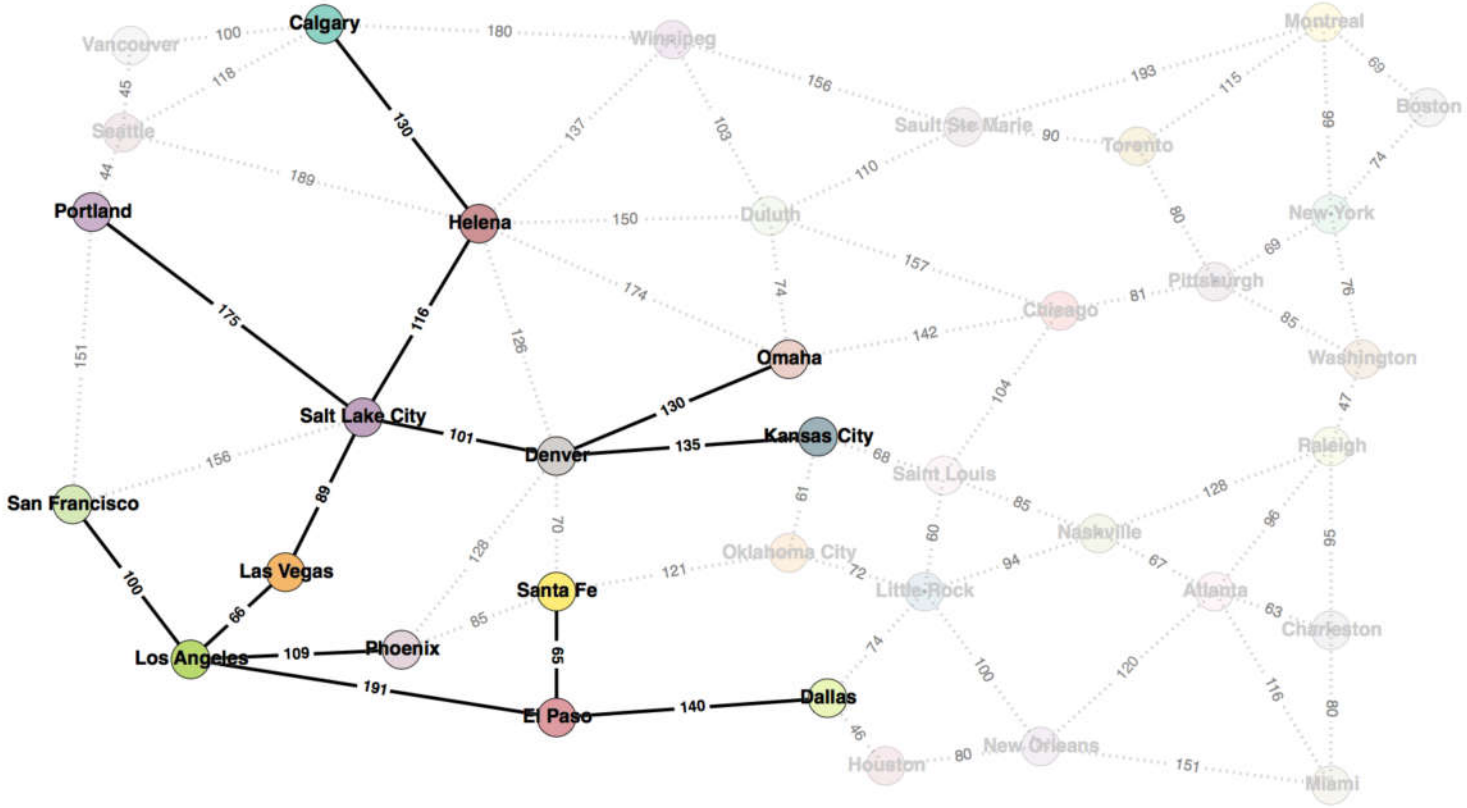
Artificial Intelligence

Search Agents

Uninformed search



Uninformed search

Use no domain knowledge!

Strategies:

1. Breadth-first search (BFS): Expand shallowest node

Uninformed search

Use no domain knowledge!

Strategies:

1. Breadth-first search (BFS): Expand shallowest node
2. Depth-first search (DFS): Expand deepest node

Uninformed search

Use no domain knowledge!

Strategies:

1. Breadth-first search (BFS): Expand shallowest node
2. Depth-first search (DFS): Expand deepest node
3. Depth-limited search (DLS): Depth first with depth limit

Uninformed search

Use no domain knowledge!

Strategies:

1. Breadth-first search (BFS): Expand shallowest node
2. Depth-first search (DFS): Expand deepest node
3. Depth-limited search (DLS): Depth first with depth limit
4. Iterative-deepening search (IDS): DLS with increasing limit

Uninformed search

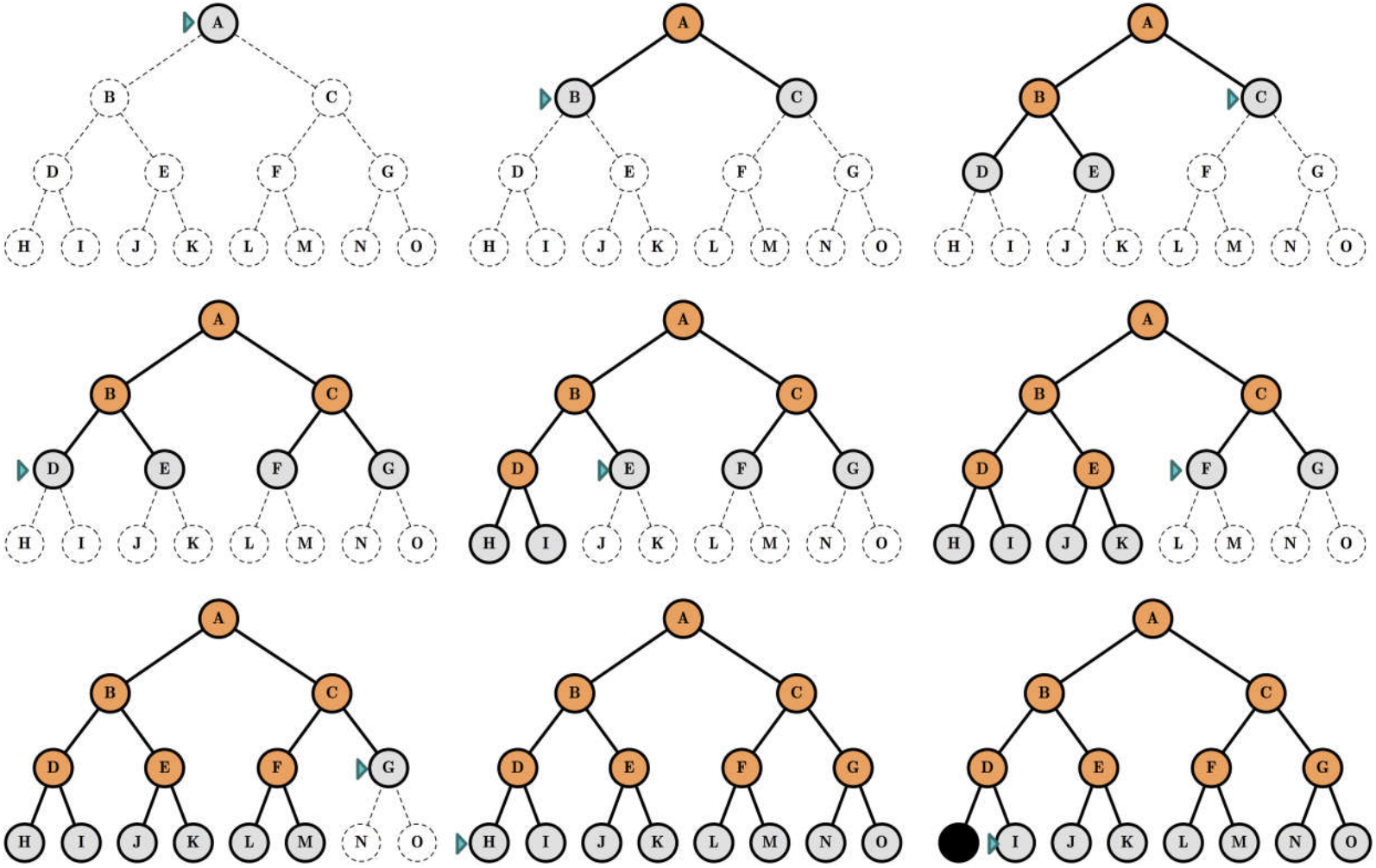
Use no domain knowledge!

Strategies:

1. Breadth-first search (BFS): Expand shallowest node
2. Depth-first search (DFS): Expand deepest node
3. Depth-limited search (DLS): Depth first with depth limit
4. Iterative-deepening search (IDS): DLS with increasing limit
5. Uniform-cost search (UCS): Expand least cost node

Breadth-first search (BFS)

BFS: Expand shallowest first.



BFS search

```
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 ∪ explored:
                frontier.enqueue(neighbor)

    return FAILURE
```

BFS Criteria

BFS criteria?

BFS

- **Complete** Yes (if b is finite)
- **Time** $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$
- **Space** $O(b^d)$
  Note: If the *goal test* is applied at expansion rather than generation then $O(b^{d+1})$
- **Optimal** Yes (if cost $= 1$ per step).
- **implementation**: fringe: FIFO (Queue)

Question: If time and space complexities are exponential, why use BFS?

BFS

How bad is BFS?

BFS

How bad is BFS?

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Time and Memory requirements for breadth-first search for a branching factor b=10; 1 million nodes per second; 1,000 bytes per node.

BFS

How bad is BFS?

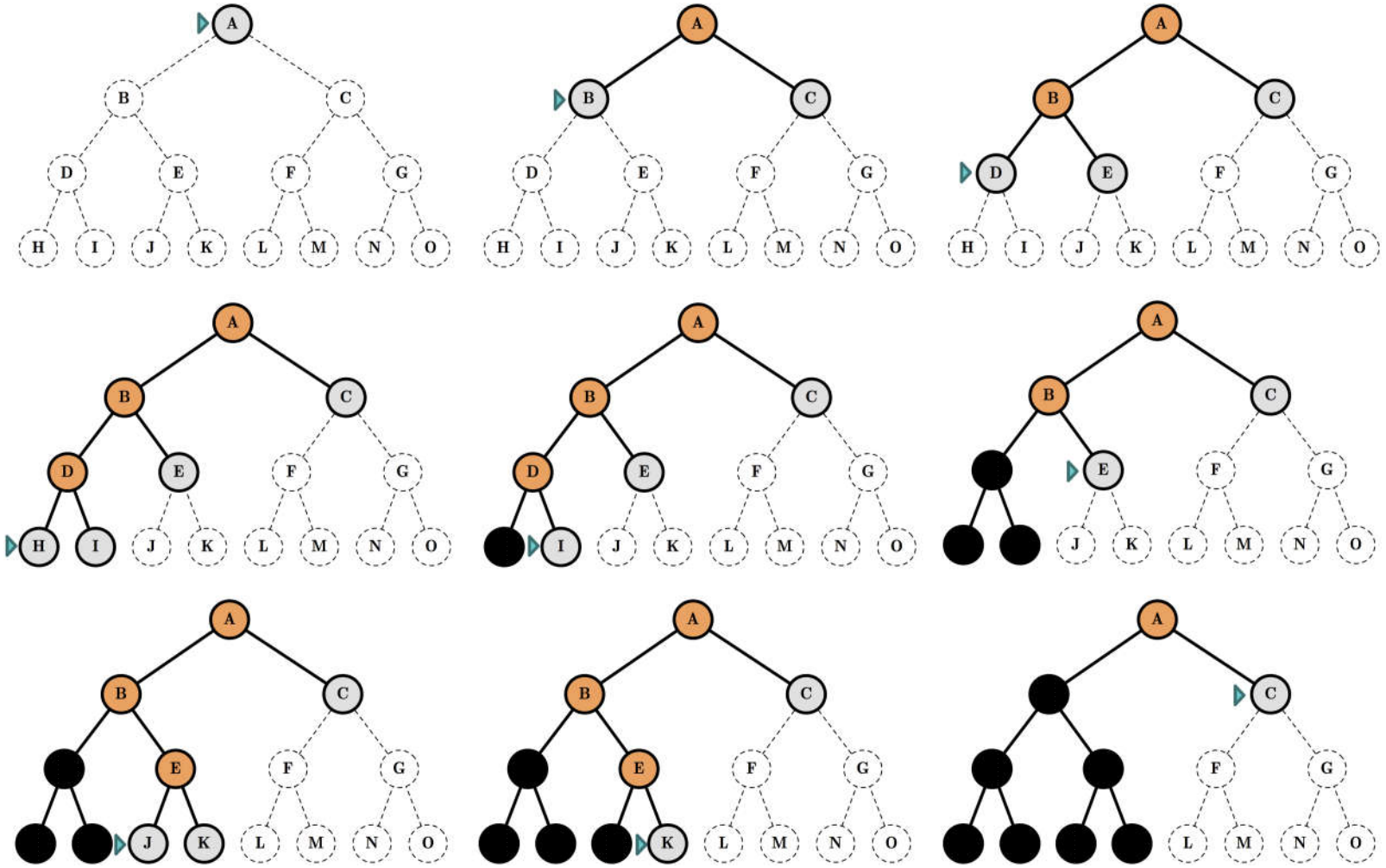
Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Time and Memory requirements for breadth-first search for a branching factor b=10; 1 million nodes per second; 1,000 bytes per node.

Memory requirement + exponential time complexity are the biggest handicaps of BFS!

DFS

DFS: Expand deepest first.



DFS search

```
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
```

DFS

DFS criteria?

DFS

- **Complete** No: fails in infinite-depth spaces, spaces with loops
  Modify to avoid repeated states along path.
  $\Rightarrow$ complete in finite spaces
- **Time** $O(b^m)$: $1 + b + b^2 + b^3 + \ldots + b^m = O(b^m)$
  bad if m is much larger than d
  but if solutions are dense, may be much faster than BFS.
- **Space** $O(bm)$ linear space complexity! (needs to store only a single path from the root to a leaf node, **along with the remaining unexpanded sibling nodes for each node on the path, hence the m factor**.)
- **Optimal** No
- **Implementation:** fringe: LIFO (Stack)

DFS

How bad is DFS?

Recall for BFS...

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Depth =16.

We go down from 10 exabytes in BFS to . . . in DFS?

DFS

How bad is DFS?

Recall for BFS...

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Depth =16.

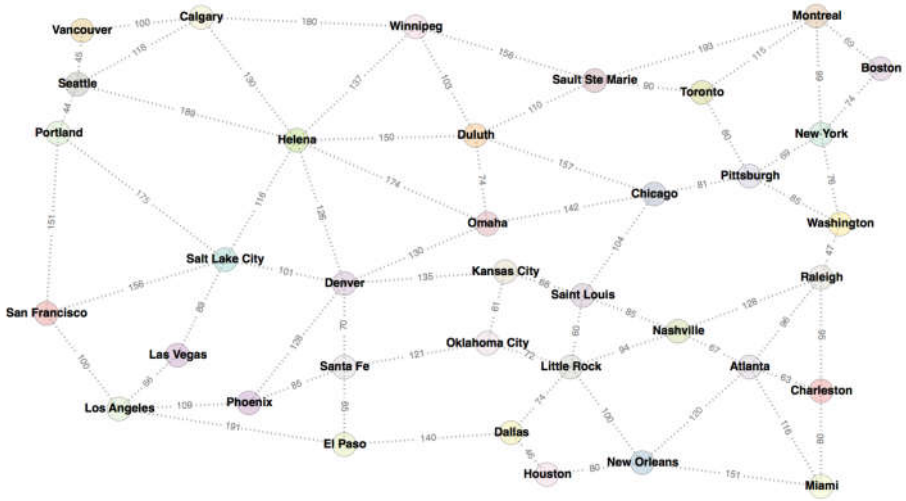
We go down from 10 exabytes in BFS to 156 kilobytes in DFS!

Depth-limited search

- DFS with depth limit l (nodes at level l has no successors).
- Select some limit L in depth to explore with DFS
- Iterative deepening: increasing the limit l

Depth-limited search

- If we know some knowledge about the problem, may be we don't need to go to a full depth.



Idea: any city can be reached from another city in at most L steps with $L < 36$.

Iterative Deepening

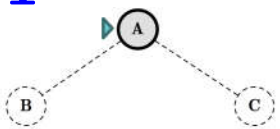
- Combines the benefits of BFS and DFS.
- Idea: Iteratively increase the search limit until the depth of the shallowest solution d is reached.
- Applies DLS with increasing limits.
- The algorithm will stop if a solution is found or if DLS returns a failure (no solution).
- Because most of the nodes are on the bottom of the search tree, it not a big waste to iteratively re-generate the top
- Let's take an example with a depth limit between 0 and 3.

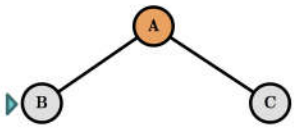
Iterative Deepening

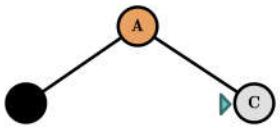
Limit = 0

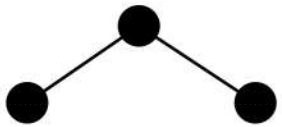
Iterative Deepening

Limit = 1



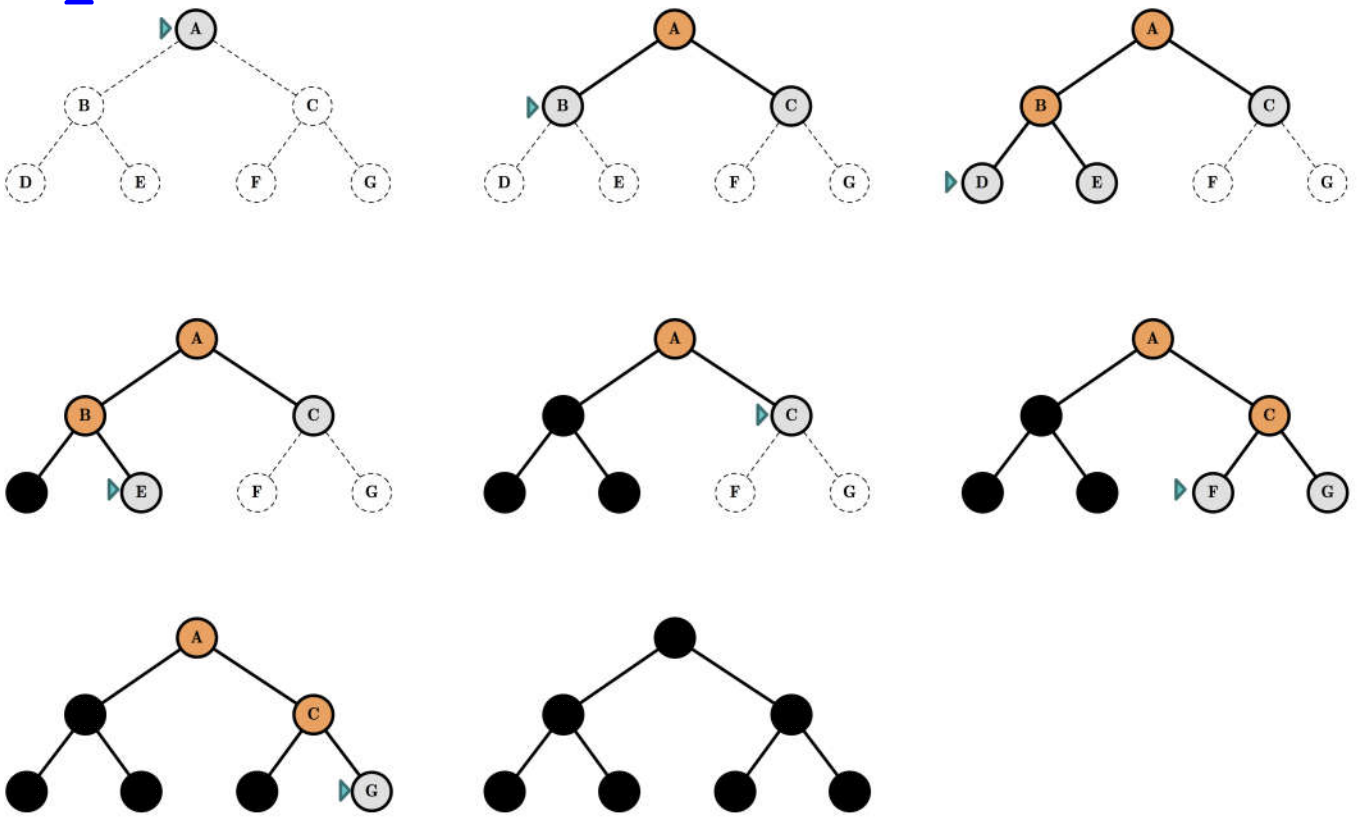






Iterative Deepening

Limit = 2



Iterative Deepening

Limit = 3