CS380
Artificial Intelligence for Games

Uninformed Search

Outline

- Uninformed search
    - Breadth-first search
        - Uniform-cost search
    - Depth-first search
        - Backtracking
    - Depth-limited search
    - Iterative deepening search
    - Bidirectional search

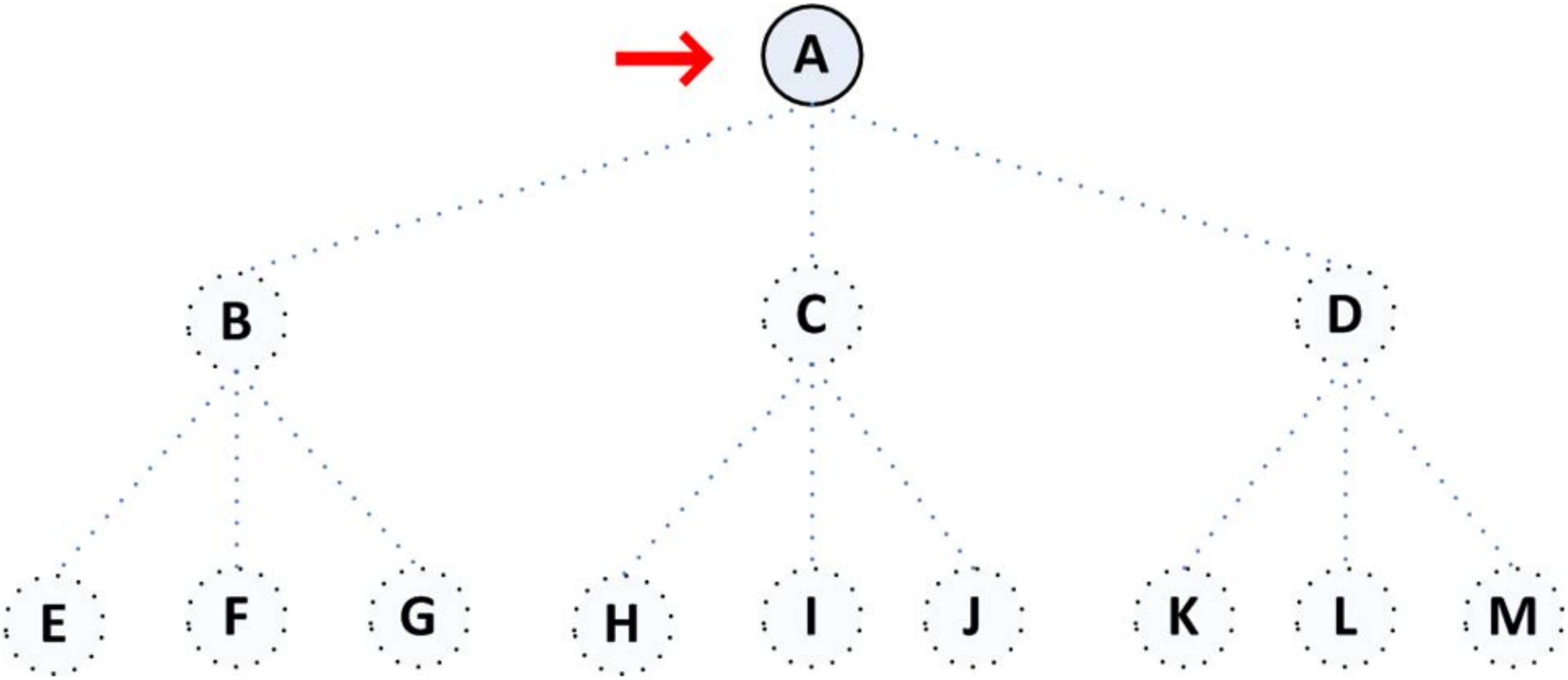
Uninformed search. BFS

- All nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded
  - At a certain level depth d of the search tree, all the previous nodes are visited for sure, and none of the nodes at level d + 1 are visited yet
- Implemented using
  - **Openlist** - a queue (FIFO) of nodes that are next to be expanded
  - **Closedlist** - a list (hash table) of nodes being expanded

Uninformed search. BFS Code

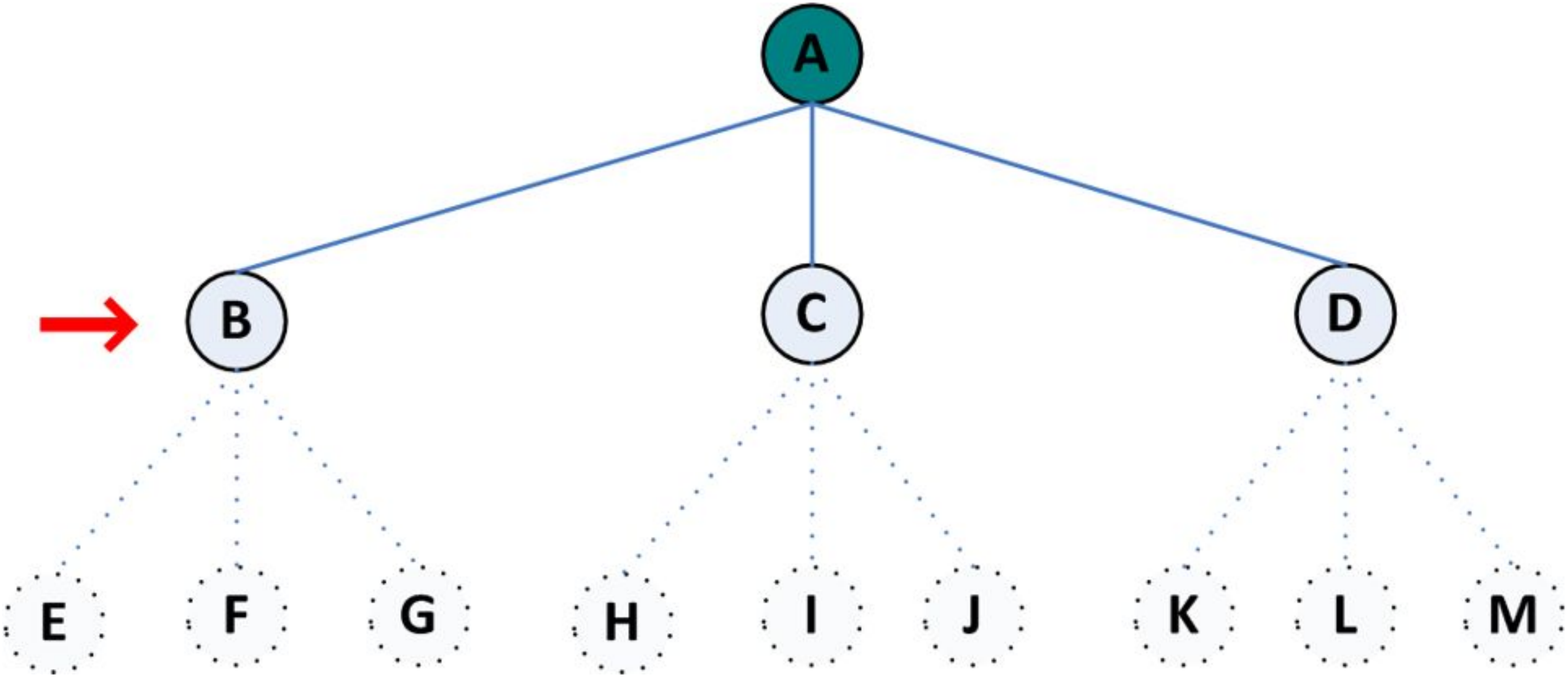
```
var closedlist = new List();

var openlist = new Queue();

var current = null;

openlist.push(starting);

while(true) {

  if (openlist.empty())  { current = null; break; }

  current = openlist.pop();

  closedlist.push(current);

  if (current == target)  break;

  for (var adjacent of getAdjacents(current))

    if (!closedlist.find(adjacent) && !openlist.find(adjacent));

        openlist.push(adjacent);

}

return current;
```

Uninformed search. BFS Example



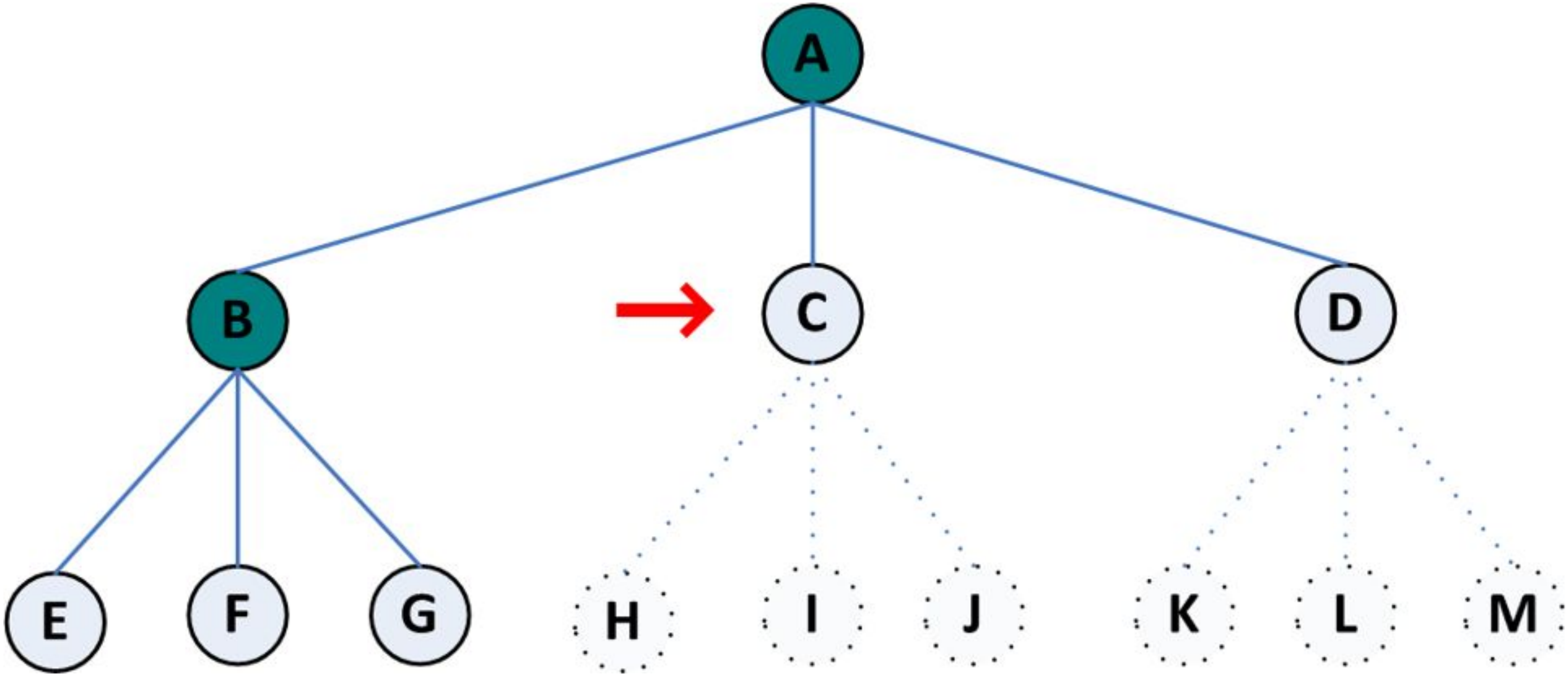
```
closedlist={}
openlist={A}
```

Uninformed search. BFS Example



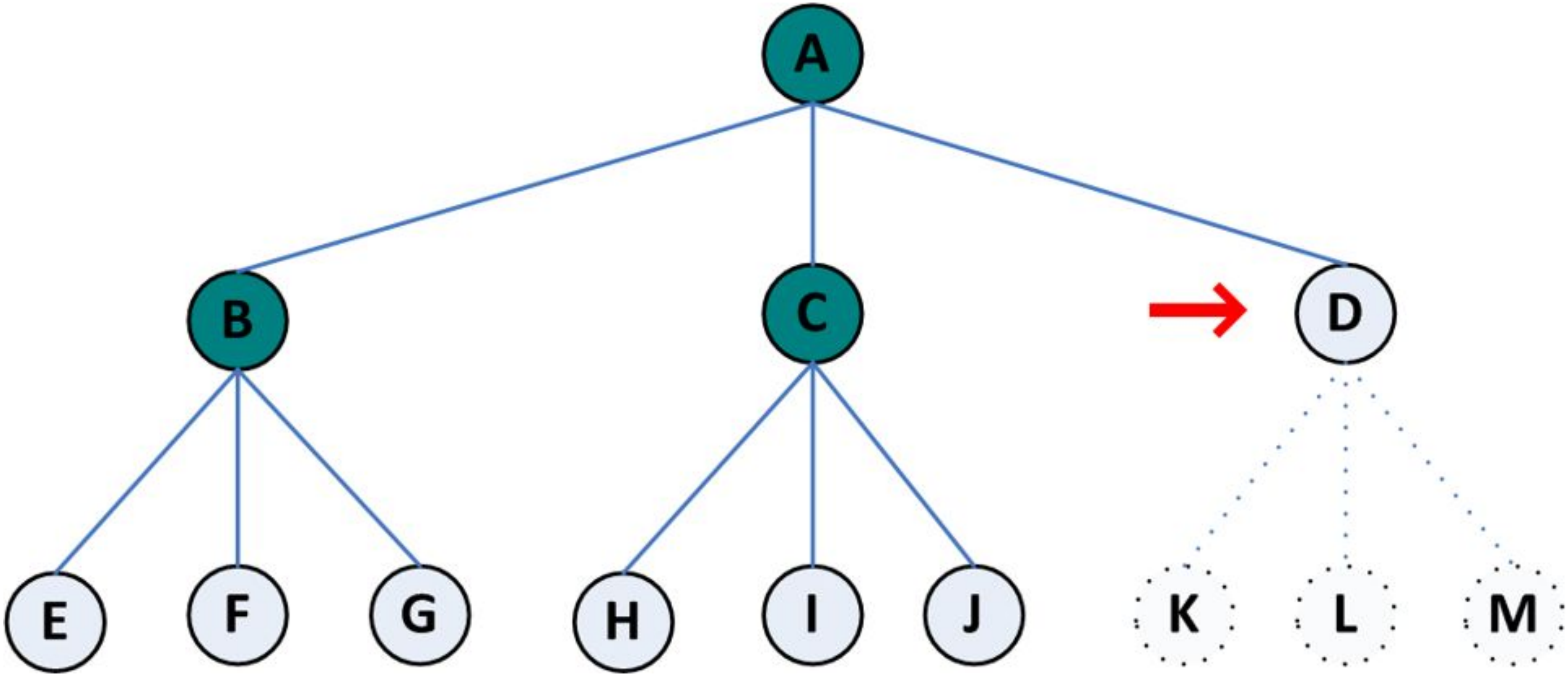
```
closedlist={A}
openlist={B,C,D}
```

Uninformed search. BFS Example



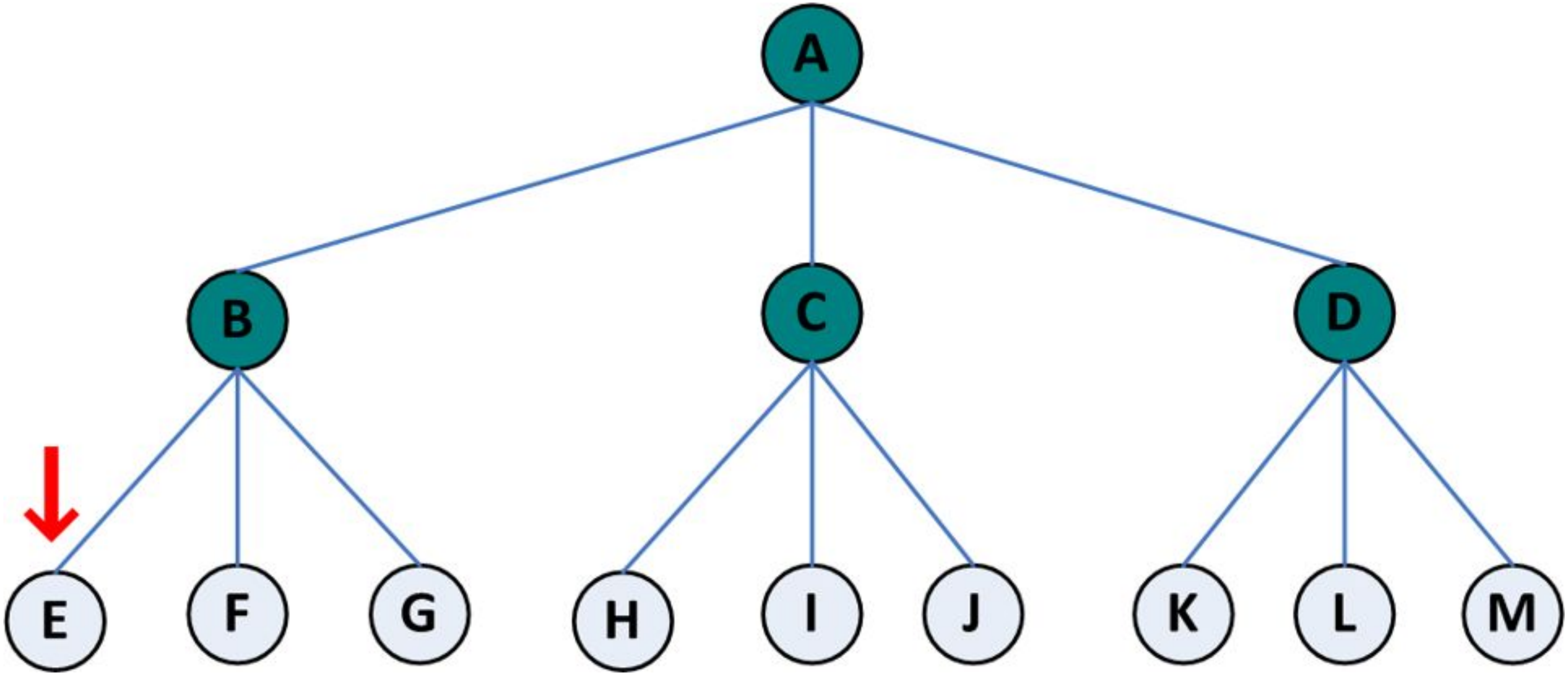
```
closedlist={A,B}
openlist={C,D,E,F,G}
```

Uninformed search. BFS Example



```
closedlist={A,B,C}
openlist={D,E,F,G,H,I,J}
```

Uninformed search. BFS Example



closedlist={A,B,C,D}

openlist={**E**,F,G,H,I,J,K,L,M}

Uninformed search. BFS

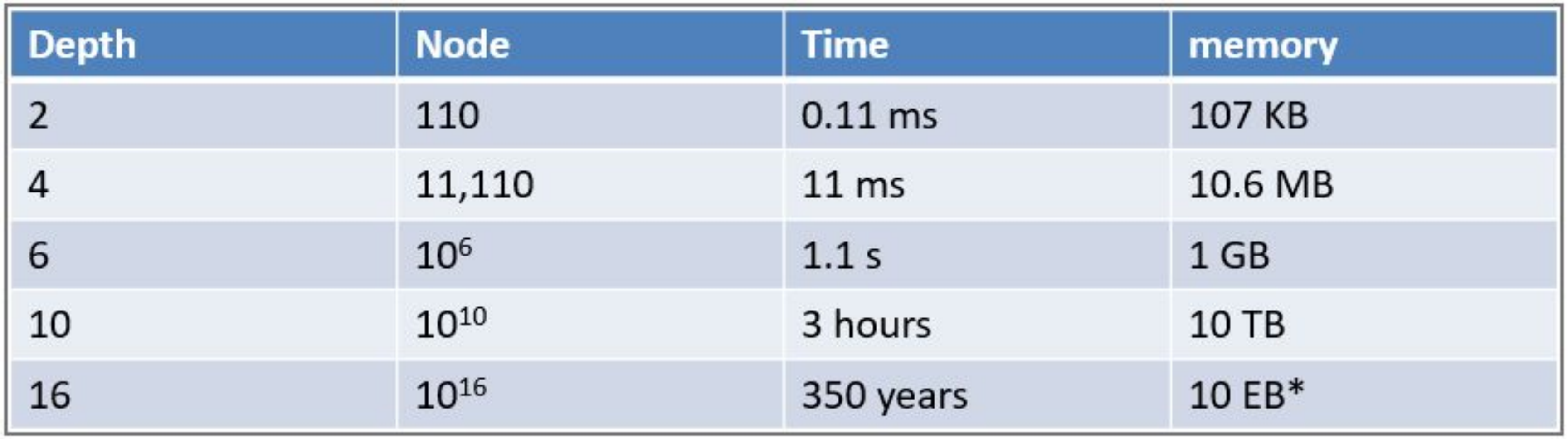
- Complete?
  - Yes, if the shallowest goal node is at some finite depth and the branching factor is also finite.
- Optimal?
  - Yes, no other search algorithm uses less time or space or expands fewer nodes, both with a guarantee of solution quality.

Uninformed search. BFS

- **Complexity** in terms of branching factors: b and depth: d
- **Time complexity**
  - $1+b+b^2+\ldots+b^d = O(b^d)$
- **Space complexity**
  - $O(b^{d-1})$ nodes in the closedlist and $O(b^d)$ nodes in the openlist

Uninformed search. BFS

- Branching factor: b=10
- 1 million nodes can be generated per second
- Each node requires 1KB of storage



Depth	Node	Time	memory
2	110	0.11 ms	107 KB
4	11,110	11 ms	10.6 MB
6	10^6	1.1 s	1 GB
10	10^{10}	3 hours	10 TB
16	10^{16}	350 years	10 EB*

*1EB (ExaByte) = 10^6TB

Outline

- Uninformed search
    - Breadth-first search
        - Uniform-cost search
    - Depth-first search
        - Backtracking
    - Depth-limited search
    - Iterative deepening search
    - Bidirectional search

Uninformed search. Uniform-Cost Search Code

- Extension of the Breadth-First Search: Finds an **optimal** solution with any **positive** step cost
- Expands the node from the openlist with the **cheapest path cost** from the root.
  - **Openlist** is implemented using a priority queue ordered by path cost. It gives maximum priority to the lowest cumulative cost.
- A “relaxation” test can be added in case a better path is found to a node currently on the openlist

Uninformed search. Uniform-Cost Search Code

```
var closedlist = new List();
var openlist = new PriorityQueue();
var current = null;
openlist.push(starting);
while(true) {
  if (openlist.empty())  { current = null; break; }
  current = openlist.pop();
  closedlist.push(current);
  if (current == target)  break;
  for (var adjacent of getAdjacents(current))
    if (!closedlist.find(adjacent) && !openlist.find(adjacent);
        openlist.push(adjacent);
}
return current;
```

Uninformed search. Uniform-Cost Search

- It has **greater** or equal work than BFS.
- It explores large trees of small steps before exploring paths involving large and perhaps useful steps.
- When all step costs are the same, uniform-cost search is similar to BFS, except that the latter stops as soon as it generates a goal, whereas uniform-cost search examines all the nodes at the goal's depth to see if one has a lower cost.

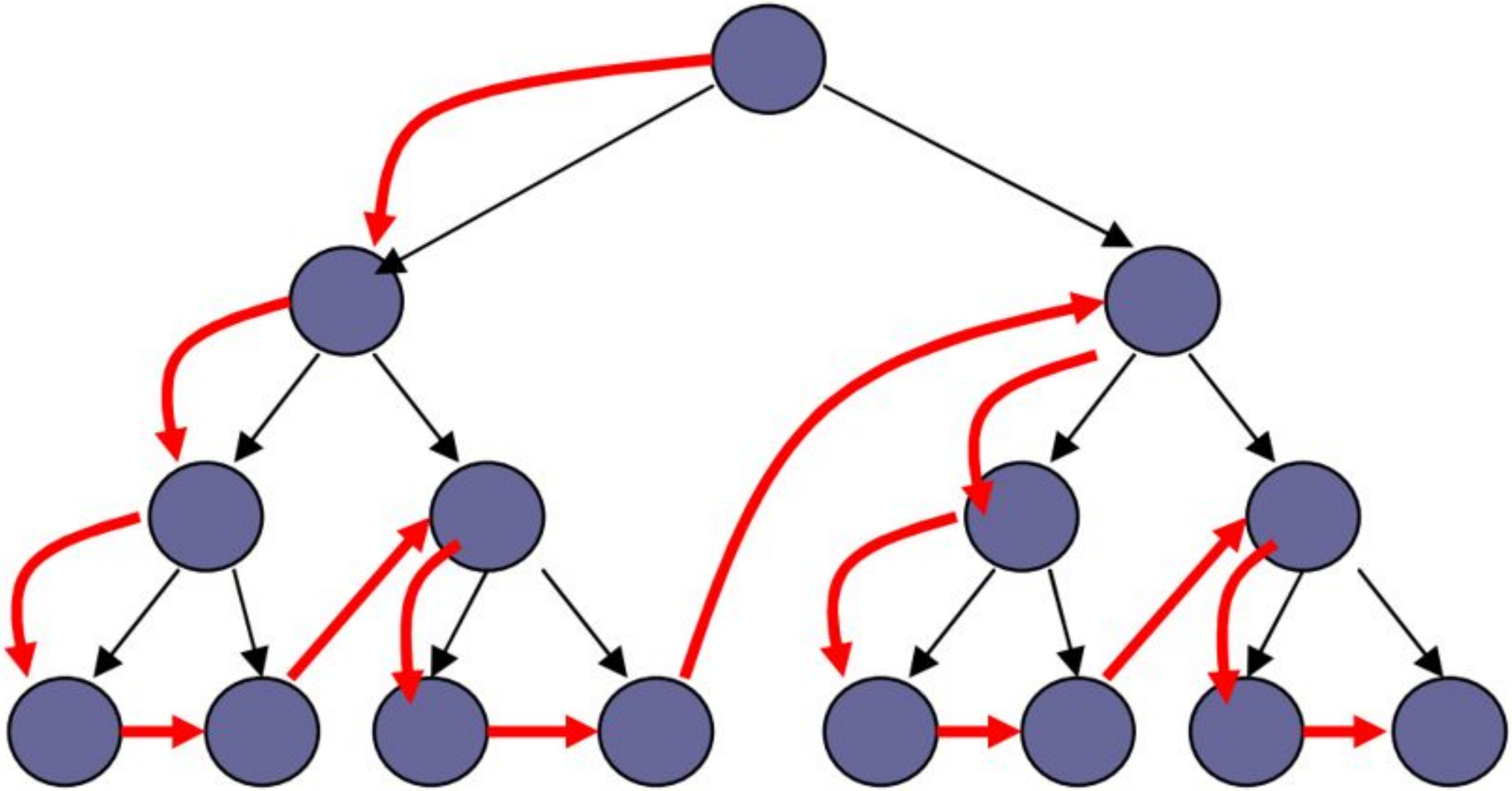
Outline

- Uninformed search
    - Breadth-first search
        - Uniform-cost search
    - Depth-first search
        - Backtracking
    - Depth-limited search
    - Iterative deepening search
    - Bidirectional search

Uninformed search. Depth-First Search

- It visits nodes into the deepest level before going back and visiting the siblings in the tree
- **Openlist** is implemented using a stack (LIFO).
- **Backtrack** when the path cannot be further expanded

Uninformed search. DFS path



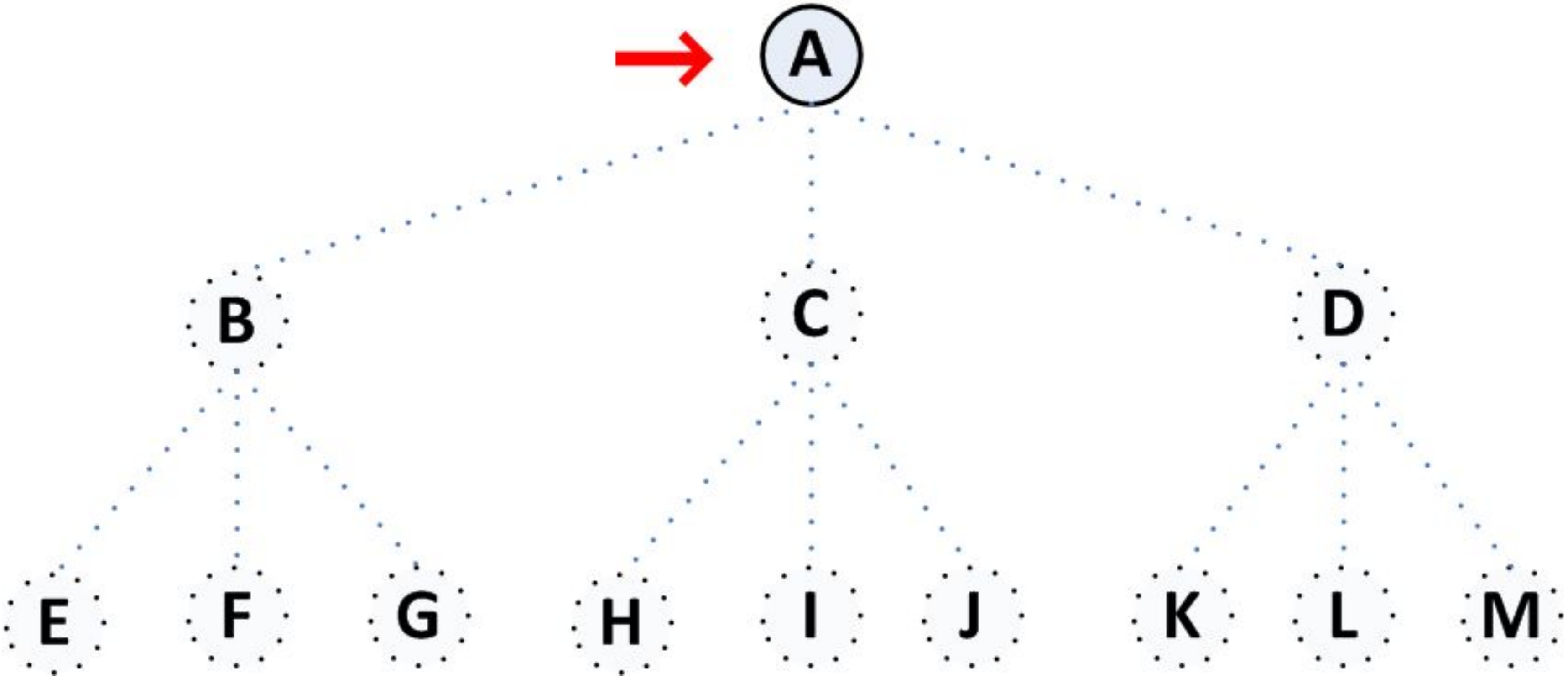
Uninformed search. DFS Code

```
var closedlist = new List();

var openlist = new Stack();

var current = null;

openlist.push(starting);

while(true) {

  if (openlist.empty())  { current = null; break; }

  current = openlist.pop();

  closedlist.push(current);

  if (current == target)  break;

  for (var adjacent of getAdjacents(current))

    if (!closedlist.find(adjacent) && !openlist.find(adjacent);

        openlist.push(adjacent);

}

return current;
```

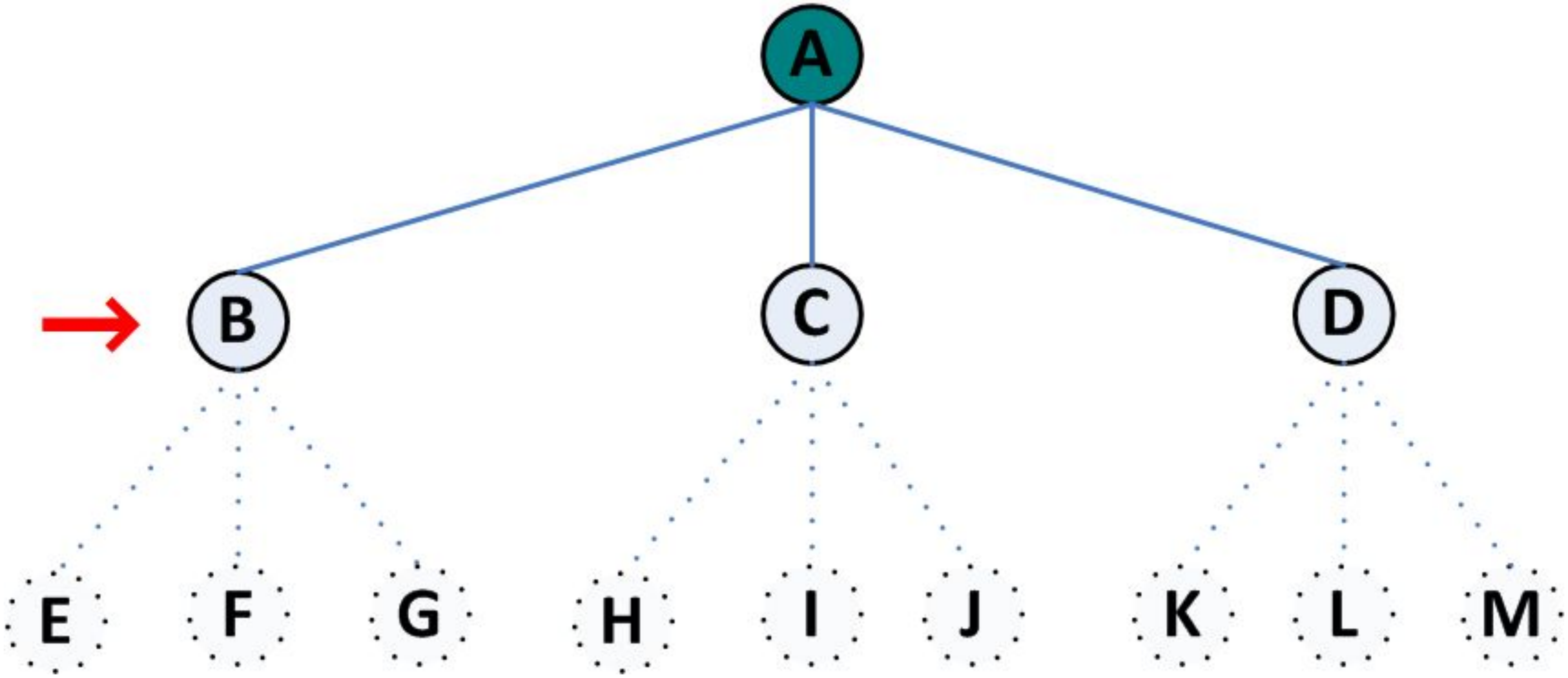
Uninformed search. DFS

- **Complete?**
  - Yes, if the state space is **finite** with **no repeated states** and paths
- **Optimal?**
  - No. We may find a solution at a deep level, whereas a more optimal solution exists at a shallow level to the right side of the search tree

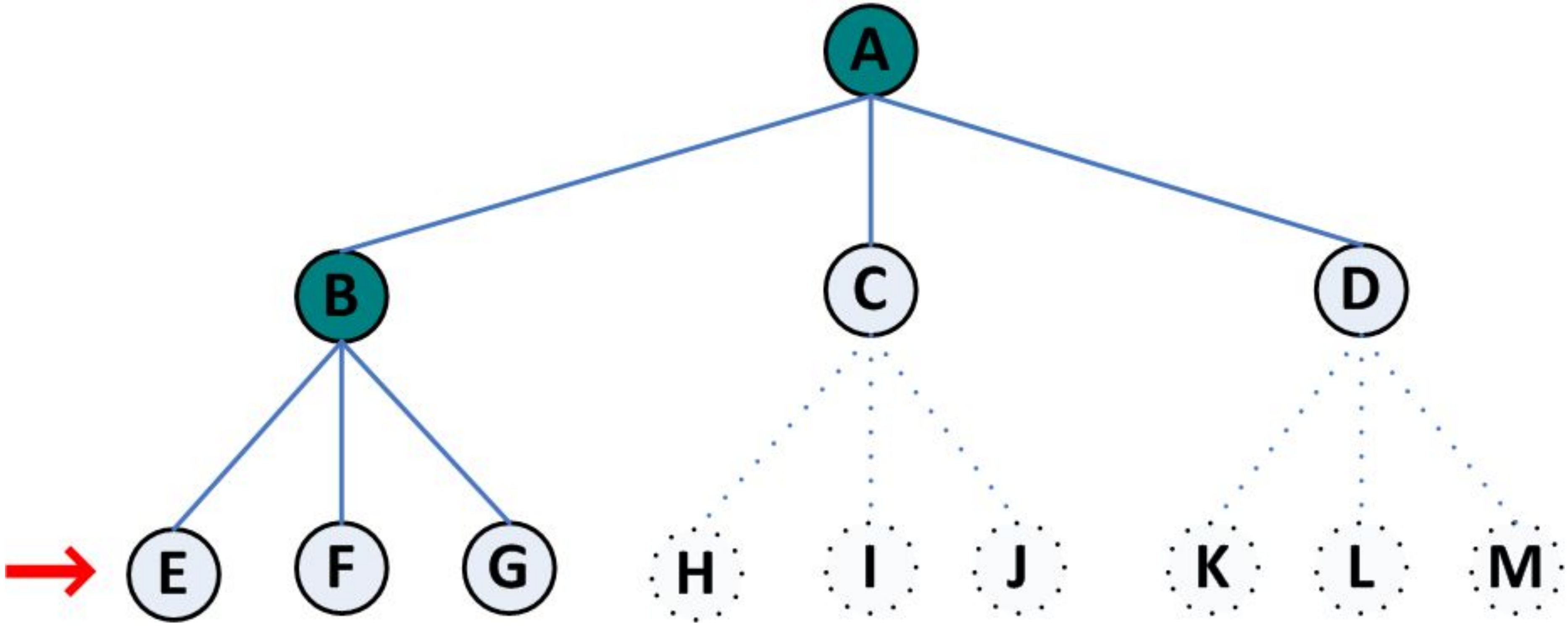
Uninformed search. DFS Example



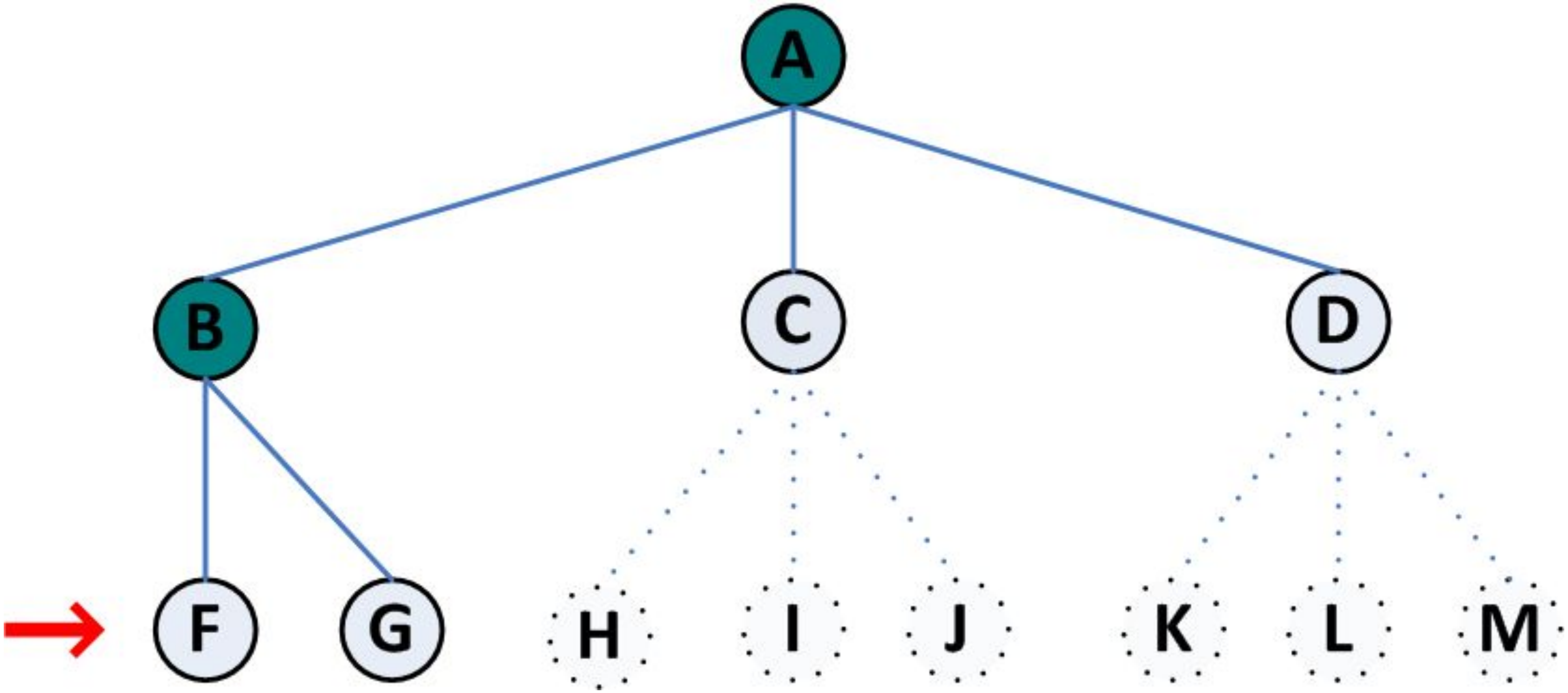
Uninformed search. DFS Example



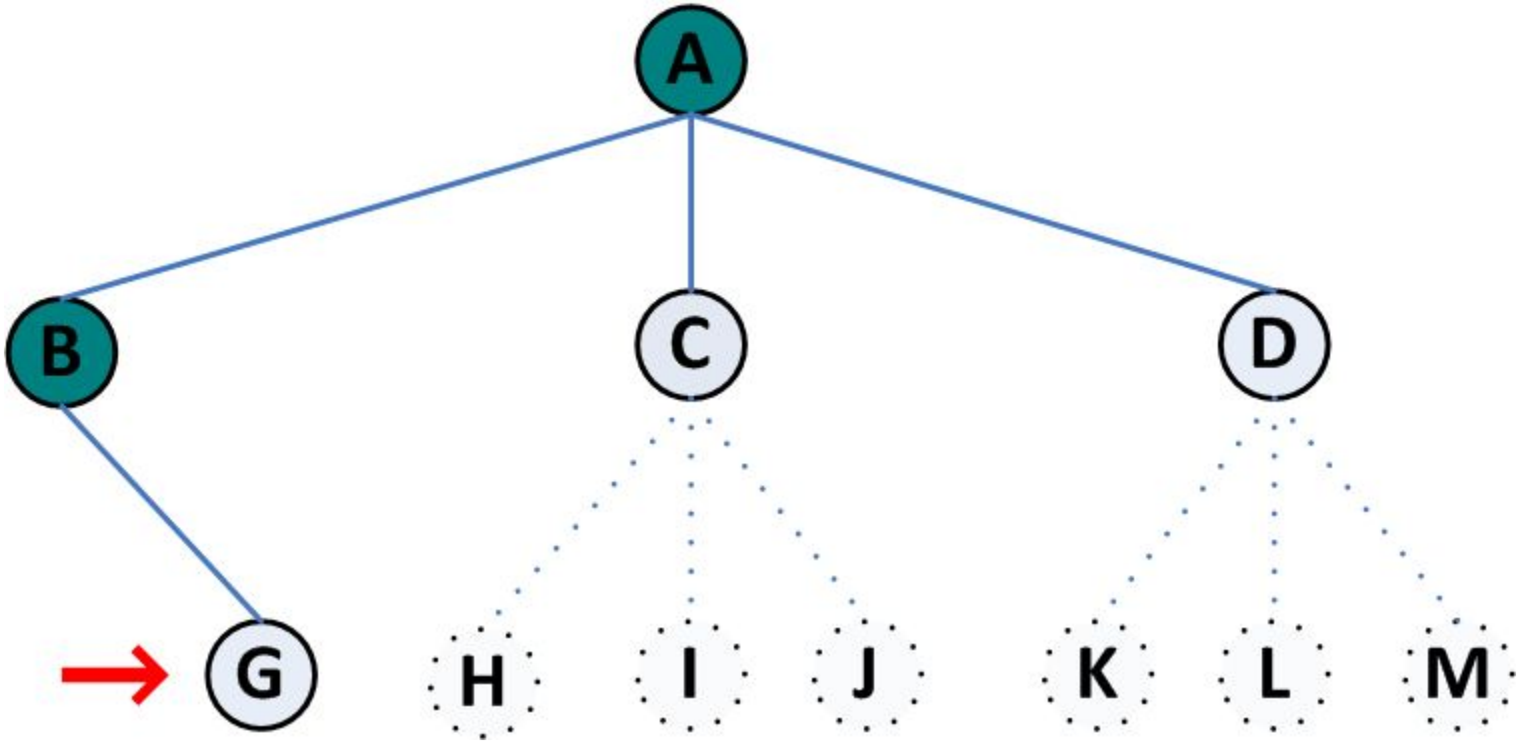
Uninformed search. DFS Example



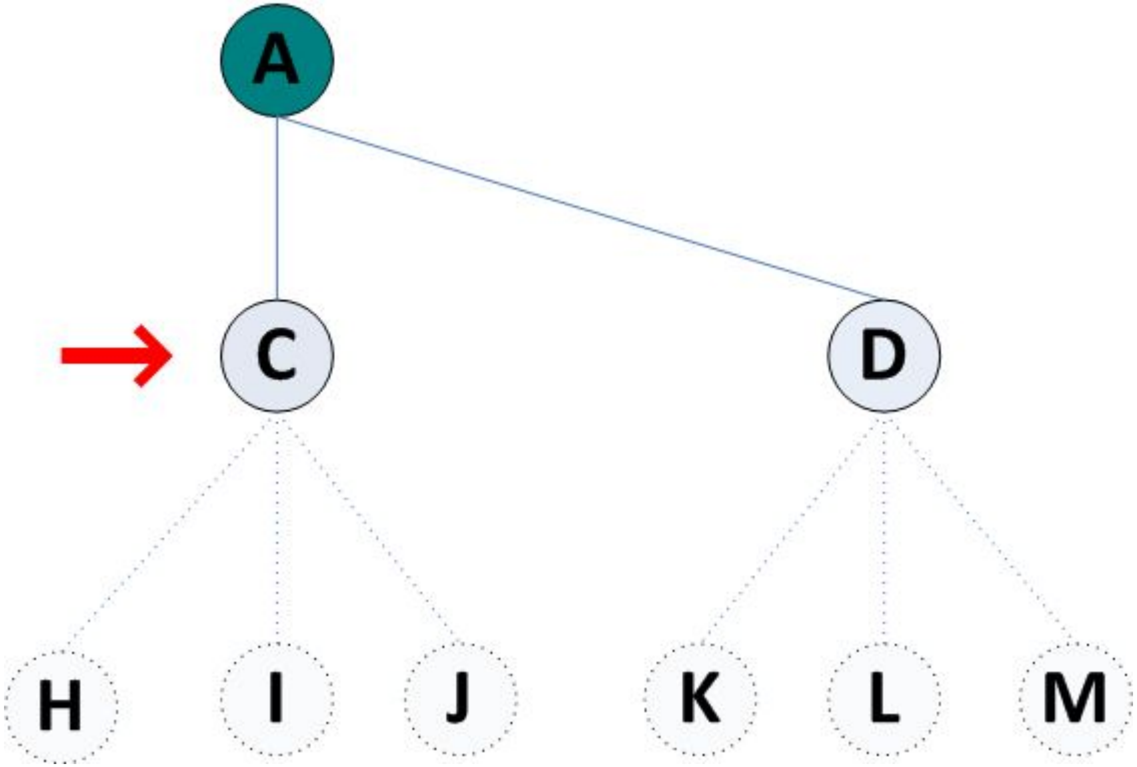
Uninformed search. DFS Example



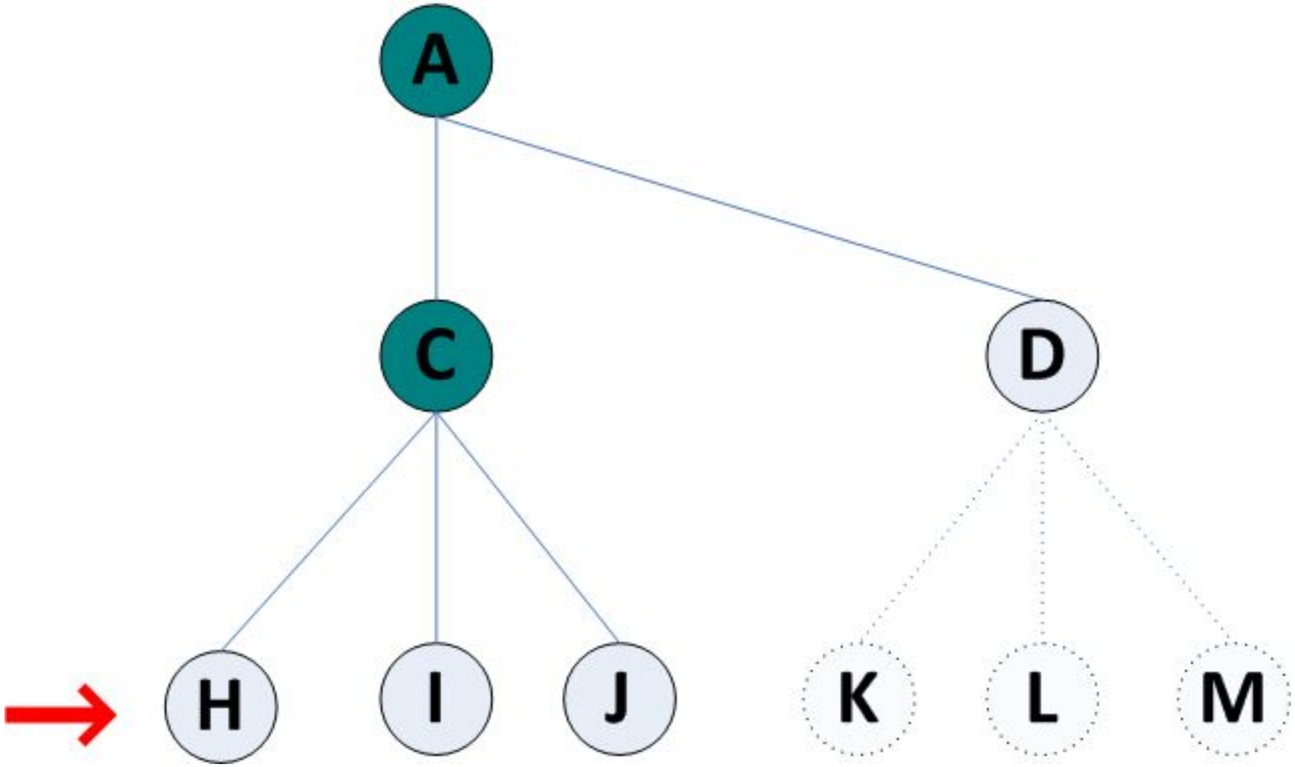
Uninformed search. DFS Example



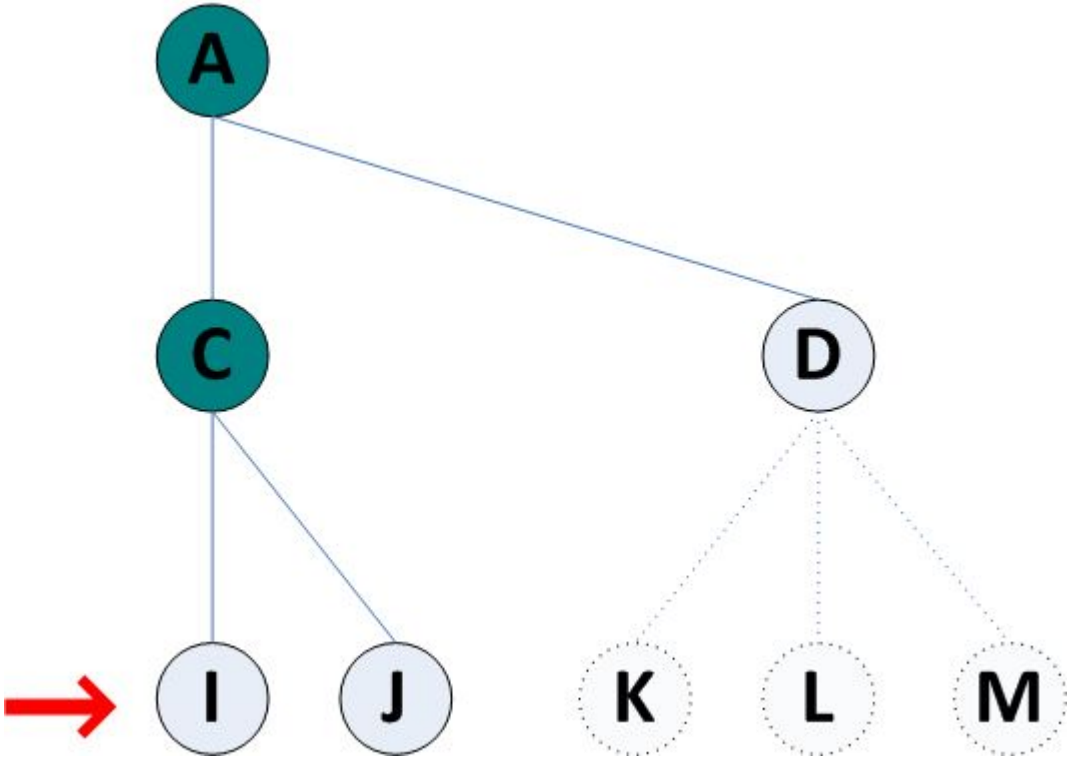
Uninformed search. DFS Example



Uninformed search. DFS Example



Uninformed search. DFS Example

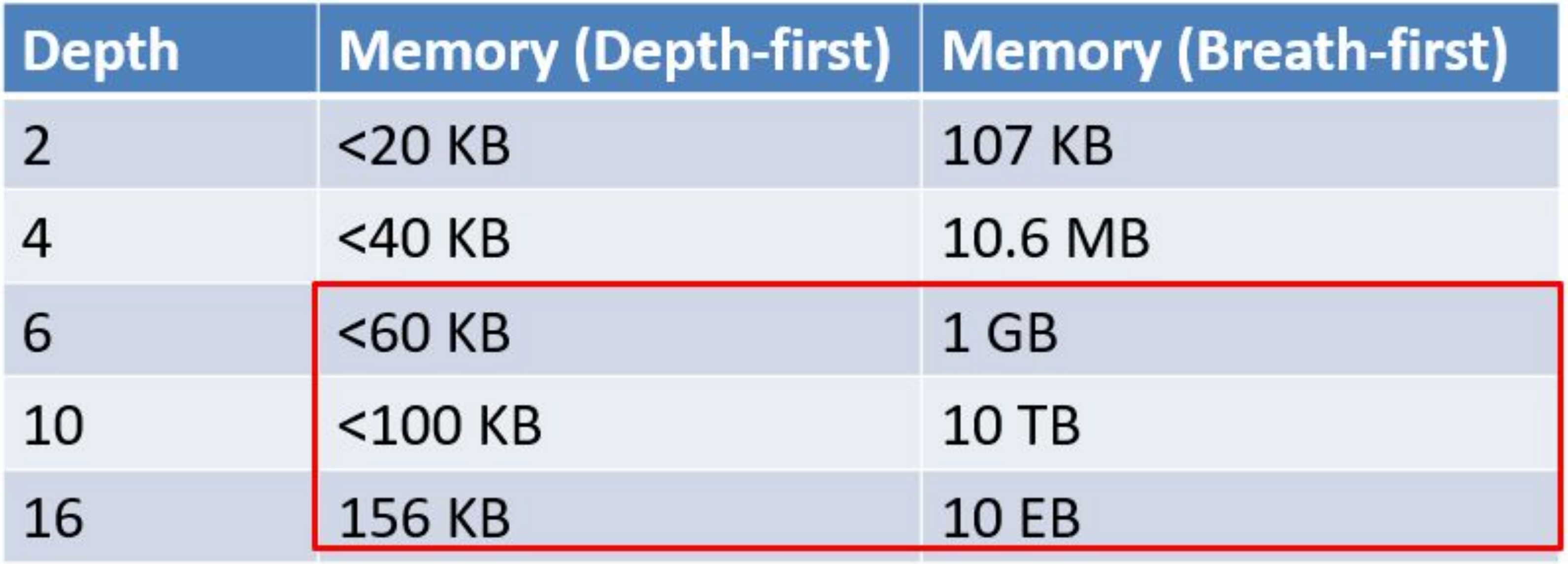


Uninformed search. DFS

- **Complexity** in terms of branching factors b and depth d
- **Time complexity:**
  - $O(b^d)$
- **Space complexity**:
  - $O(bd)$
  - Only the path from the root to the current expanded node needs to be saved
  - Once a node has been expanded, it can be removed from memory as soon as all its descendants have been fully explored

Uninformed search. DFS vs BFS

- Branching factor: b=10
- Each node requires 1KB of storage



Depth	Memory (Depth-first)	Memory (Breath-first)
2	<20 KB	107 KB
4	<40 KB	10.6 MB
6	<60 KB	1 GB
10	<100 KB	10 TB
16	156 KB	10 EB

Uninformed search. Backtracking

- DFS, but only one successor is generated at a time rather than all successors
- Each partially expanded node remembers which successor to generate next
- Only O(d) memory is needed rather than O(bd).
- Recursion

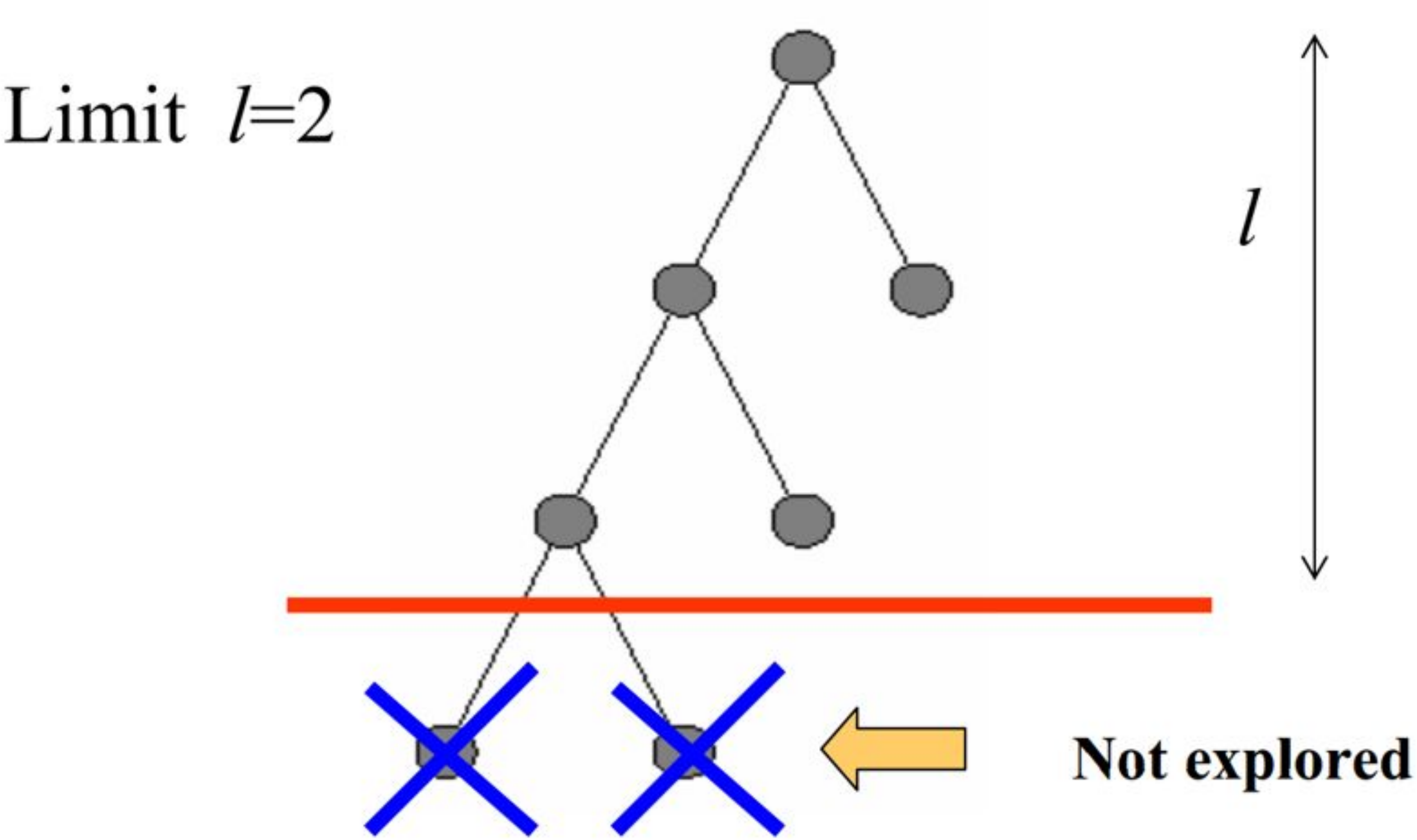
Outline

- Uninformed search
  - Breadth-first search
    - Uniform-cost search
  - Depth-first search
    - Backtracking
  - Depth-limited search
  - Iterative deepening search
  - Bidirectional search

Uninformed search. Depth-limited search

- A depth limit L is assigned to the depth-first search algorithm
    - When the limit is reached, the algorithm returns with a cut off information which is different than a failure (no solution)
- When L < D, where D is the shallowest goal to be reached, the algorithm becomes incomplete
- When L > D the algorithm become non-optimal as described previously
- How to determine L?

Uninformed search. Depth-limited search



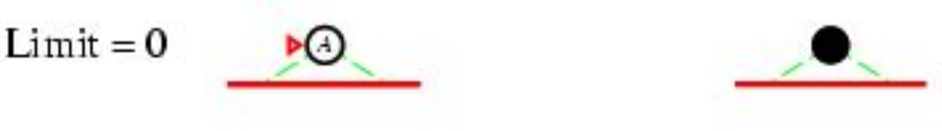
Outline

- Uninformed search
  - Breadth-first search
    - Uniform-cost search
  - Depth-first search
    - Backtracking
  - Depth-limited search
  - Iterative deepening search
  - Bidirectional search

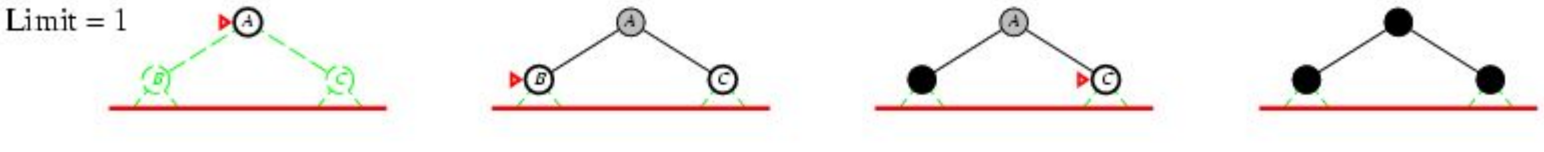
Uninformed search. Iterative Deepening Search

- This search algorithm finds out the best depth limit and does it by gradually **increasing** the limit until a goal is found
- Combines benefits of both Depth-first search and Breadth-first search
  - O(bd) memory requirements
  - Complete when k is finite
  - Optimum when the path cost is a non-decreasing function of the depth of the node
- The main drawback is that it repeats all the work of the previous iteration

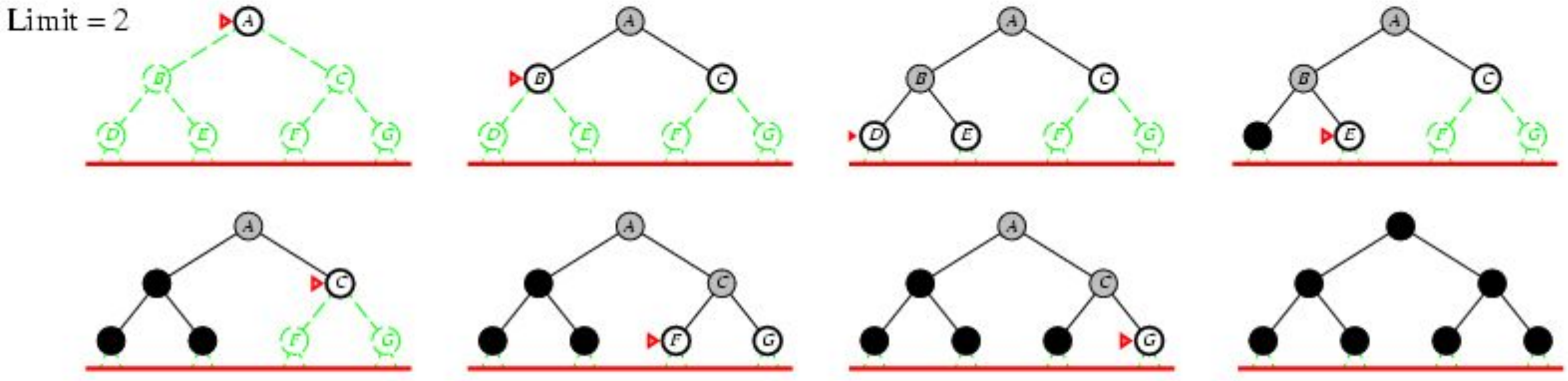
Uninformed search. IDS. L=0



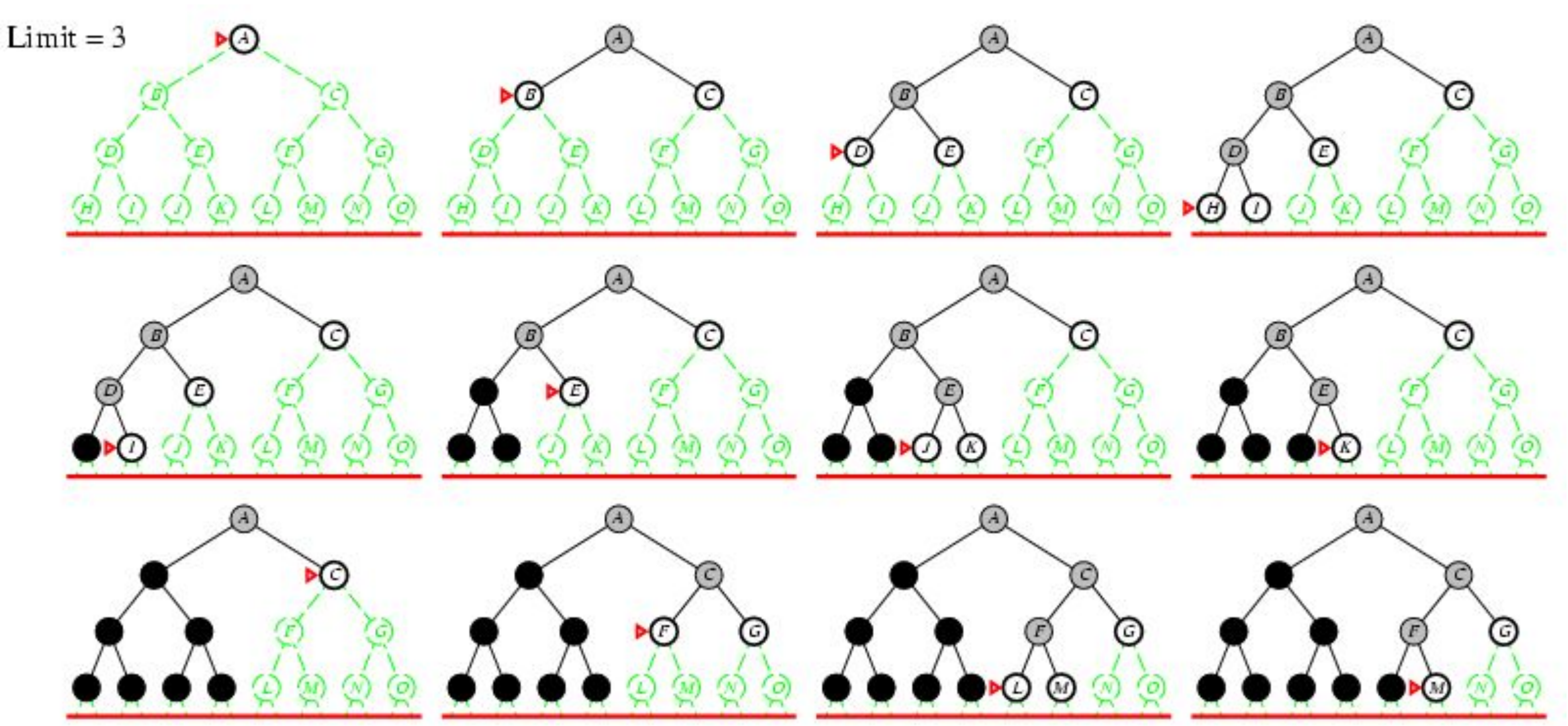
Uninformed search. IDS. L=1



Uninformed search. IDS. L=2



Uninformed search. IDS. L=3



Outline

- Uninformed search
  - Breadth-first search
    - Uniform-cost search
  - Depth-first search
    - Backtracking
  - Depth-limited search
  - Iterative deepening search
  - Bidirectional search

Uninformed search. Bidirectional search

- Bidirectional search algorithm runs two simultaneous searches:
    1. form initial state called as forward-search
    2. from goal node called as backward-search
- Bidirectional search replaces one single search graph with two small subgraphs
- The search stops when subgraphs intersect each other
- Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.

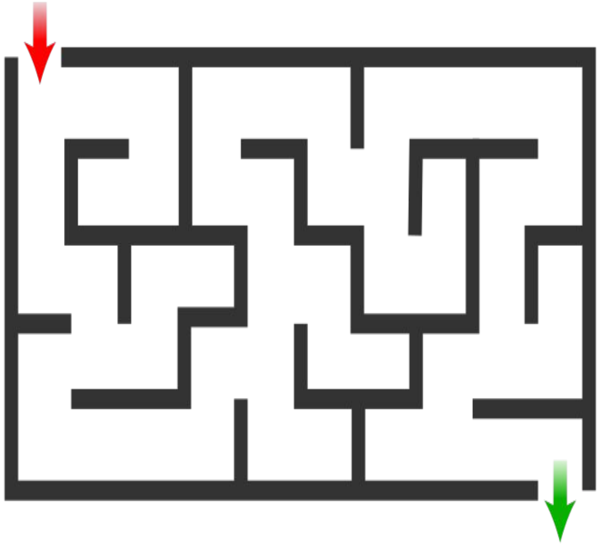
Uninformed search. Bidirectional search

Advantages:

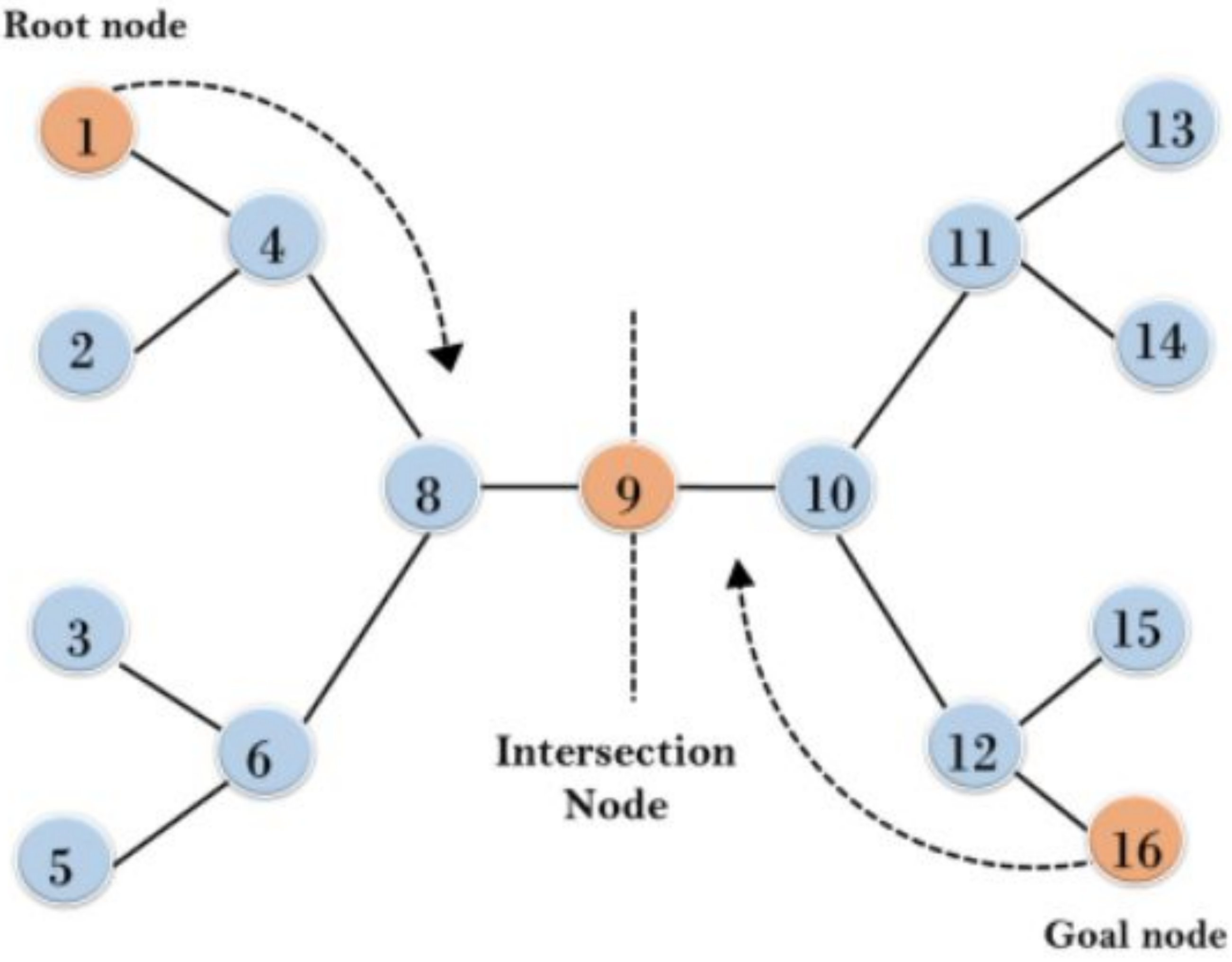
- Bidirectional search is fast
- Bidirectional search requires less memory

Disadvantages:

- Implementation of the bidirectional search tree is difficult
- In bidirectional search, one should know the goal state in advance



Uninformed search. Bidirectional search. Example



Uninformed search. Bidirectional search

Completeness:

Is complete if BFS is used in both searches

Time Complexity:

$O(b^d)$ using BFS

Space Complexity:

$O(b^d)$