Artificial Intelligence (AI)

CCS-3880 – 3rd Semester 2023

CO2: Blind Search

Dr. Abdullah Alshanqiti

Blind Search

CO2. Investigate the problem state spaces and uninformed search algorithms.

There are two types of search strategies:

- Blind (uninformed) search
- Heuristic (directed, informed) search

Search Strategies

- **Blind search** → traversing the search space until the goal nodes is found (might be doing exhaustive search).

- **Techniques** :
    - Breadth-first search (BFS)
    - Uniform cost search
    - Depth-first search (DFS)
    - Depth-limited search
    - Iterative deepening search

- **Guarantees solution**
  but under some constraints.

- **Heuristic search** → search process takes place by traversing search space with applied rules (information).

- **Techniques:**
    - Greedy Best First Search,
    - A* Algorithm
    - Local Search Algorithms
        - Hill-climbing search
        - Gradient Descent
        - Simulated annealing
          (suited for either local or global search)
    - Global Search Algorithms
        - Genetic Algorithm

There is no guarantee that solution is found.

Outline

Search Strategies (Blind Search**)**

- Breadth-First Search (BFS)
- Uniform Cost Search (UCS)
- Depth-First Search (DFS)
- Depth-Limited Search (DLS)
- Iterative Deepening Search (IDS)

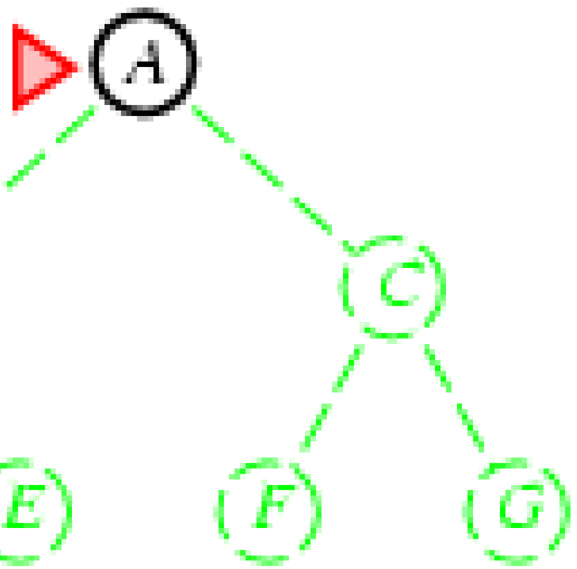
The consideration

Typical questions that need to be answered before choosing an algorithm:

- Is the problem solver **guaranteed** to find a solution?
- Will the system always terminate or caught in an infinite loop?
- If the solution is found, is it **optimal**?
- What is the **complexity** of searching process?
- How the system be able to reduce searching complexity?
- How can it effectively utilize the representation paradigm?

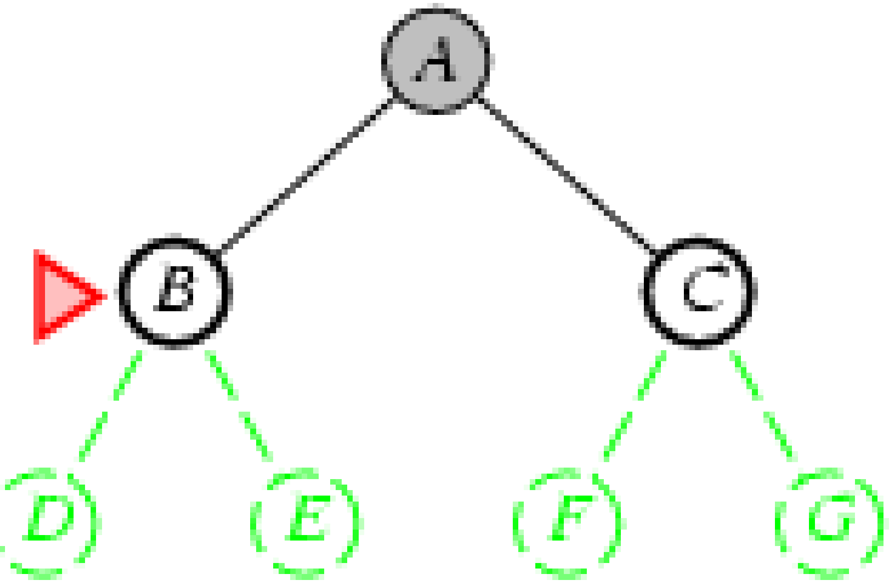
Breadth-first search

- The simplest blind search strategy
- Upon expanding the root node, we expand its children, then we expand their children, etc.
- In general, nodes at level d are expanded only after all nodes at depth d−1 have been expanded, i.e., we search *level-by-level*
- Implementation: based on FIFO **queue**, i.e., new successors go at **end**



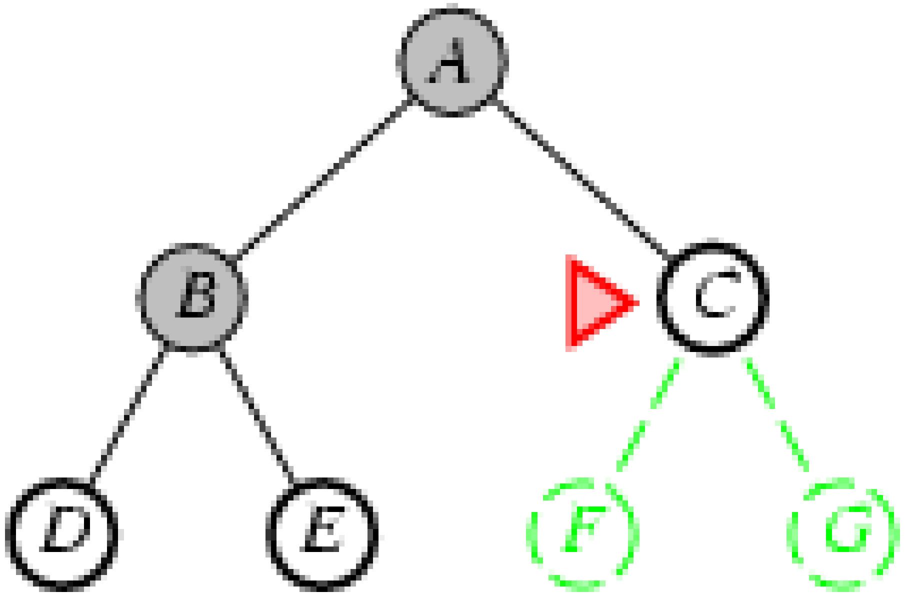
Breadth-first search

- The simplest blind search strategy
- Upon expanding the root node, we expand its children, then we expand their children, etc.
- In general, nodes at level d are expanded only after all nodes at depth d−1 have been expanded, i.e., we search *level-by-level*
- Implementation: based on FIFO **queue**, i.e., new successors go at **end**



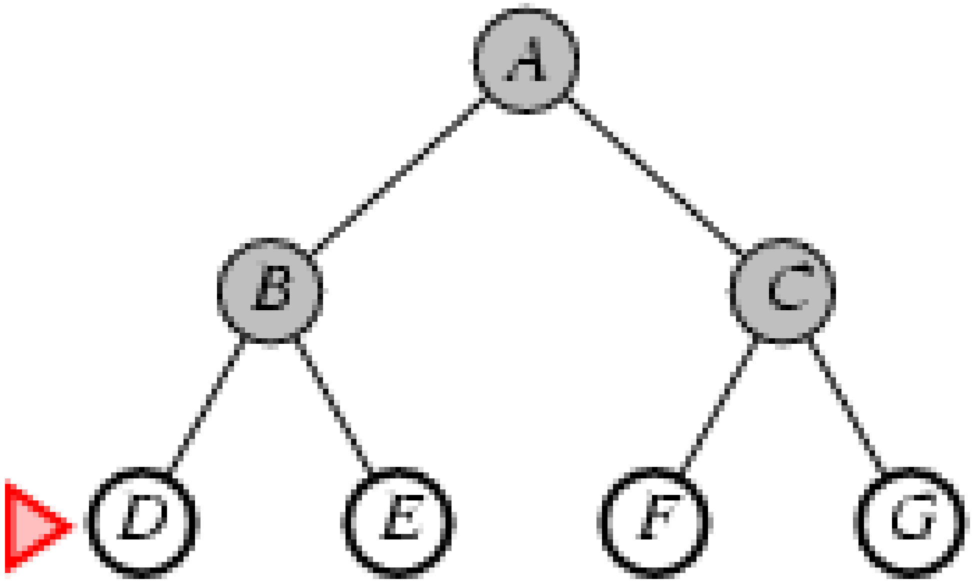
Breadth-first search

- The simplest blind search strategy
- Upon expanding the root node, we expand its children, then we expand their children, etc.
- In general, nodes at level d are expanded only after all nodes at depth d−1 have been expanded, i.e., we search *level-by-level*
- Implementation: based on FIFO **queue**, i.e., new successors go at **end**



Breadth-first search

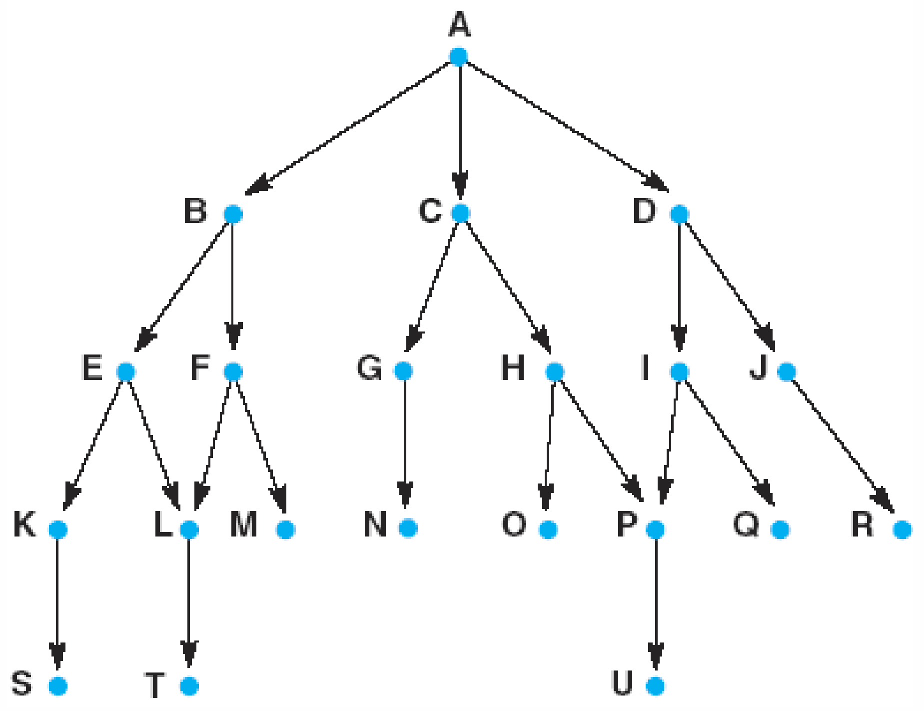
- The simplest blind search strategy
- Upon expanding the root node, we expand its children, then we expand their children, etc.
- In general, nodes at level d are expanded only after all nodes at depth d−1 have been expanded, i.e., we search *level-by-level*
- Implementation: based on FIFO **queue**, i.e., new successors go at **end**



Breadth-first search

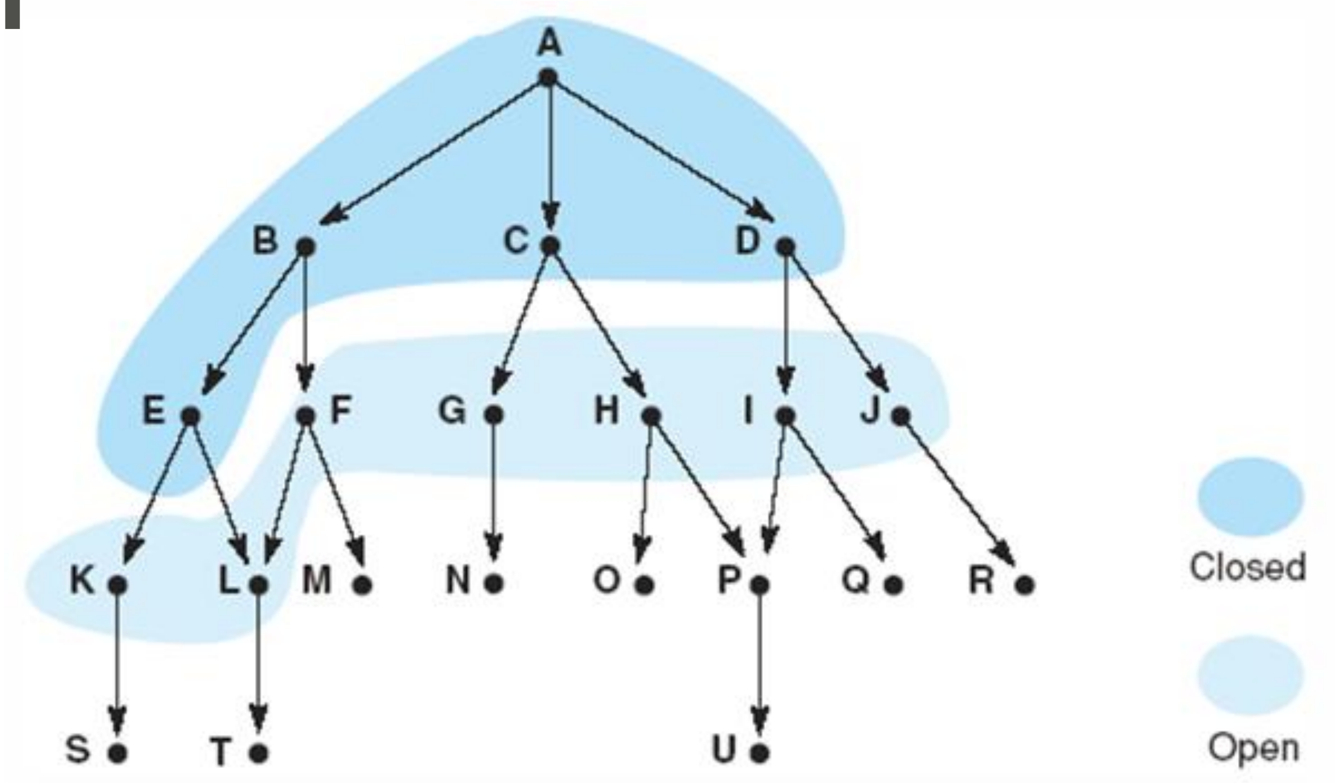
Breadth-first search

function $\mathrm{breadthFirstSearch}(s_0, \mathrm{succ}, \mathrm{goal})$
    $open \leftarrow [\,\mathrm{initial}(s_0)\,]$
    while $open \neq [\,]$ **do**
        $n \leftarrow \mathrm{removeHead}(open)$
        if $\mathrm{goal}(\mathrm{state}(n))$ **then return** n
        for $m \in \mathrm{expand}(n, \mathrm{succ})$ **do**
            $\mathrm{insertBack}(m, open)$
    return $fail$



Starting at node A, our search would generate the nodes in alphabetical order from A to U

Breadth-first search



1. **open = [A]; closed = [ ]**
2. **open = [B,C,D]; closed = [A]**
3. **open = [C,D,E,F]; closed = [B,A]**
4. **open = [D,E,F,G,H]; closed = [C,B,A]**
5. **open = [E,F,G,H,I,J]; closed = [D,C,B,A]**
6. **open = [F,G,H,I,J,K,L]; closed = [E,D,C,B,A]**
7. **open = [G,H,I,J,K,L,M]** (as L is already on open); **closed = [F,E,D,C,B,A]**
8. **open = [H,I,J,K,L,M,N]; closed = [G,F,E,D,C,B,A]**
9. and so on until either U is found or **open** = [ ]

Properties of Breadth-first search

- Complete? Yes (if *b* is finite)
- Time? $b+b^2+b^3+\ldots+b^d = O(b^d) \Rightarrow O(b^{d+1})$ because nodes at depth d would be expanded before the goal was detected
- Space? $O(b^d)$ (keeps every node in memory)
- Optimal? Yes (if cost = 1 per step)
- Space is the bigger problem (more than time)
- BFS is applicable only to small problems

Properties of Breadth-first search

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

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

Lessons:

- The **memory** requirements are a bigger problem for BFS than is the execution time. One might wait 13 days for the solution to an important problem with search depth 12, but no personal computer has the petabyte of memory it would take.
- **Time** is still a major factor. If your problem has a solution at depth 16, then (given our assumptions) it will take about 350 years for breadth-first search to find it.
- In general, **exponential-complexity** search problems cannot be solved by uninformed methods for any but the smallest instances.

Uniform-cost Search

- Like BFS, but accounts for transition costs
- Expand least-cost unexpanded node
- Implementation: queue ordered by path cost
- Equivalent to breadth-first if step costs all equal

Uniform cost search

function uniformCostSearch(s_0, succ, goal)
    $open \leftarrow [\,\text{initial}(s_0)\,]$
    while $open \neq [\,]$ **do**
        $n \leftarrow \text{removeHead}(open)$
        if goal(state(n)) **then return** n
        for $m \in \text{expand}(n, \text{succ})$ **do**
            $\text{insertSortedBy}(g, m, open)$
    return *fail*

The first goal node selected for expansion must be the optimal solution

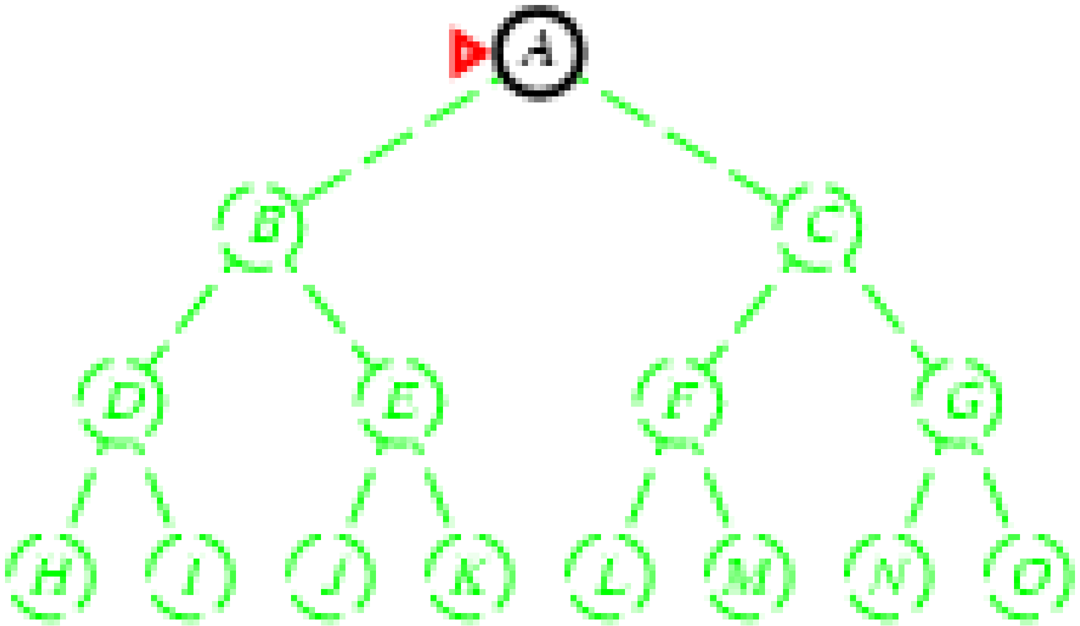
Properties of uniform-cost Search

Uniform-cost search is guided by path costs rather than depths, so its complexity is not easily characterized in terms of *b* and *d*.

- Complete? Yes, if the cost of every step is ≥ ε (epsilon here is just a small positive constant)
- Time? # of nodes with $g \le$ cost of optimal solution, $O(b^{1+(C^*/\varepsilon)})$ where C^* is the cost of the optimal solution and *g* is queue
  of paths. The complexity here is much greater than b^d
- Space? # of nodes with $g \le$ cost of optimal solution, $O(b^{1+(C^*/\varepsilon)})$
- Optimal? Yes – nodes expanded in increasing order of *g(n)* (i.e., lowest path cost)

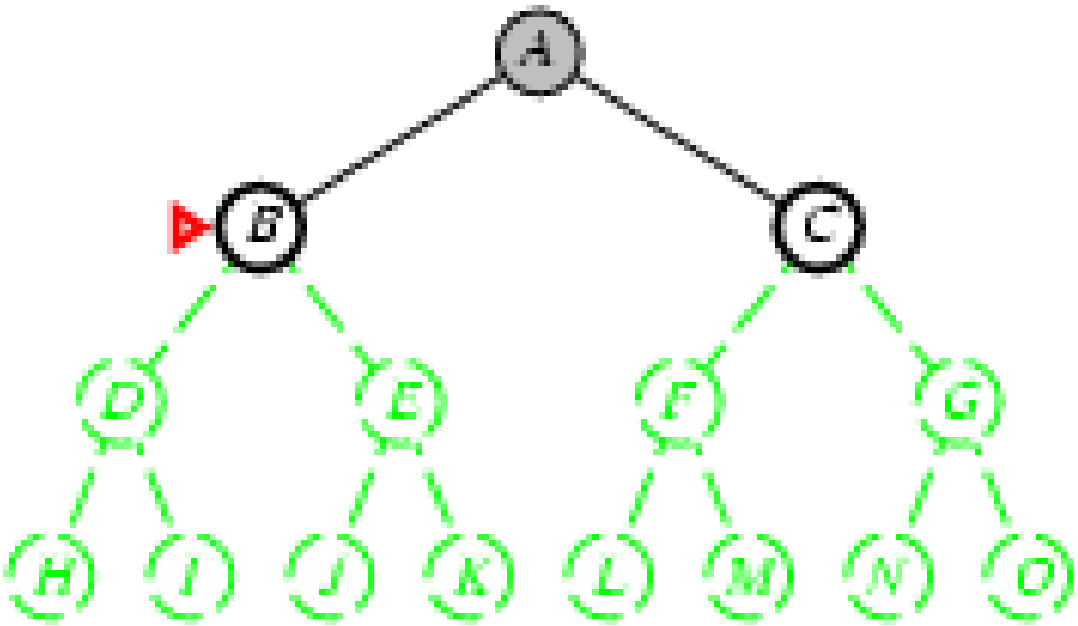
Depth-first search

- Depth-first search (DFS) always expands the deepest node in the search tree
- The search returns to the upper-level nodes only after reaching the leaf node (a node without descendants)
- Implementation: based on LIFO **stack**, i.e., put successors at **front**



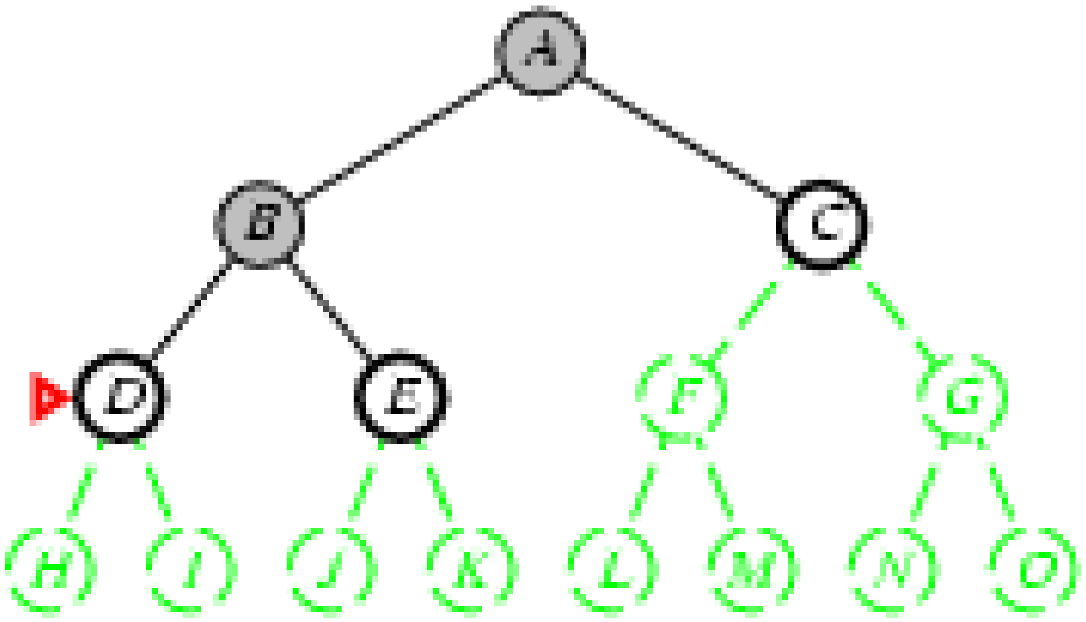
Depth-first search

- Depth-first search (DFS) always expands the deepest node in the search tree
- The search returns to the upper-level nodes only after reaching the leaf node (a node without descendants)
- Implementation: based on LIFO **stack**, i.e., put successors at **front**



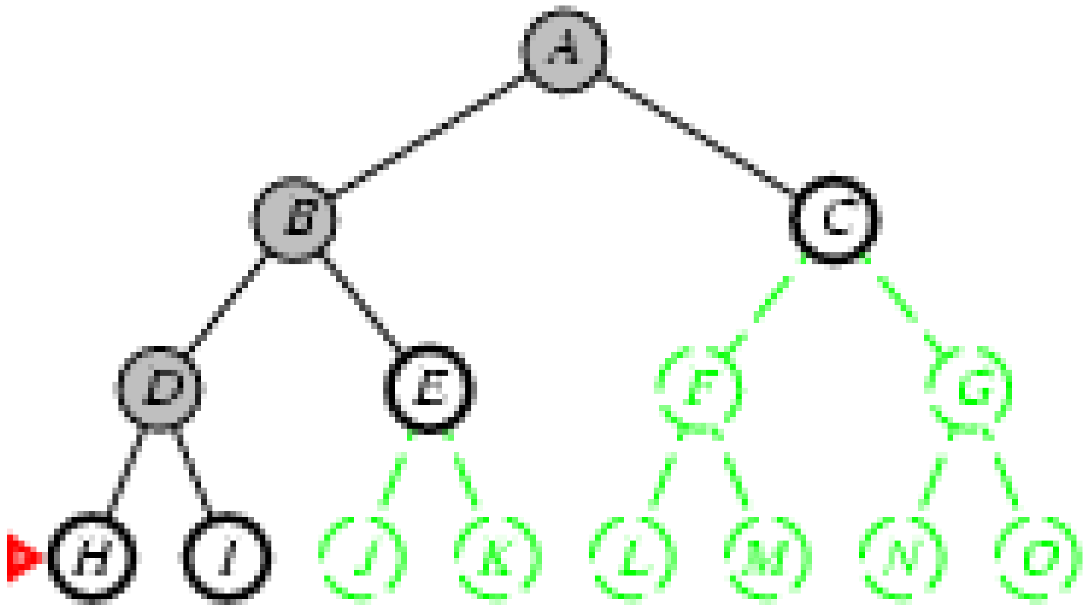
Depth-first search

- Depth-first search (DFS) always expands the deepest node in the search tree
- The search returns to the upper-level nodes only after reaching the leaf node (a node without descendants)
- Implementation: based on LIFO **stack**, i.e., put successors at **front**



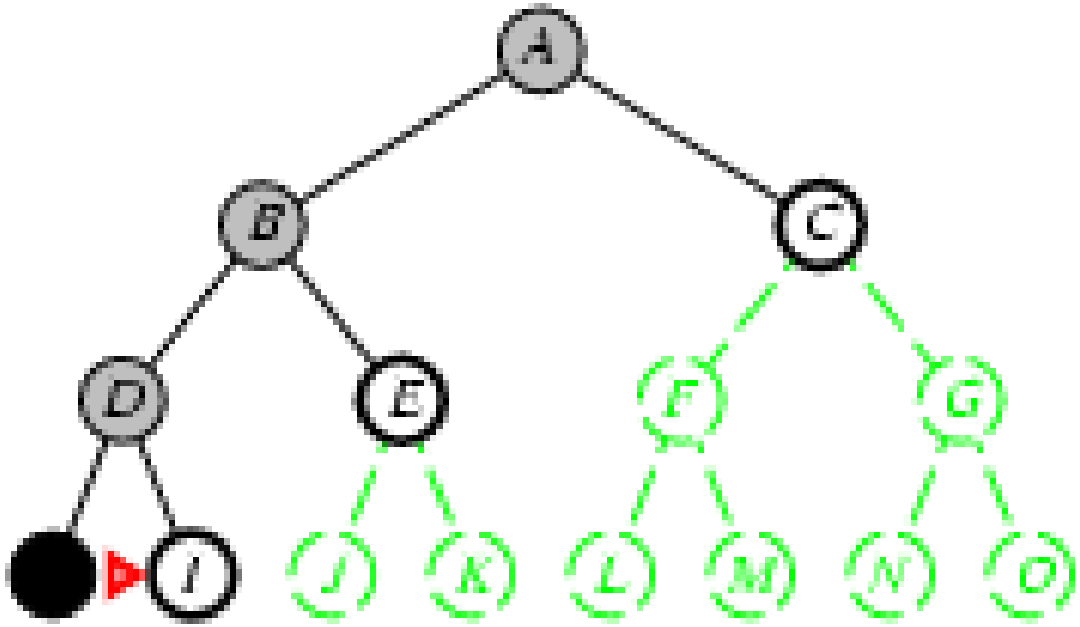
Depth-first search

- Depth-first search (DFS) always expands the deepest node in the search tree
- The search returns to the upper-level nodes only after reaching the leaf node (a node without descendants)
- Implementation: based on LIFO **stack**, i.e., put successors at **front**



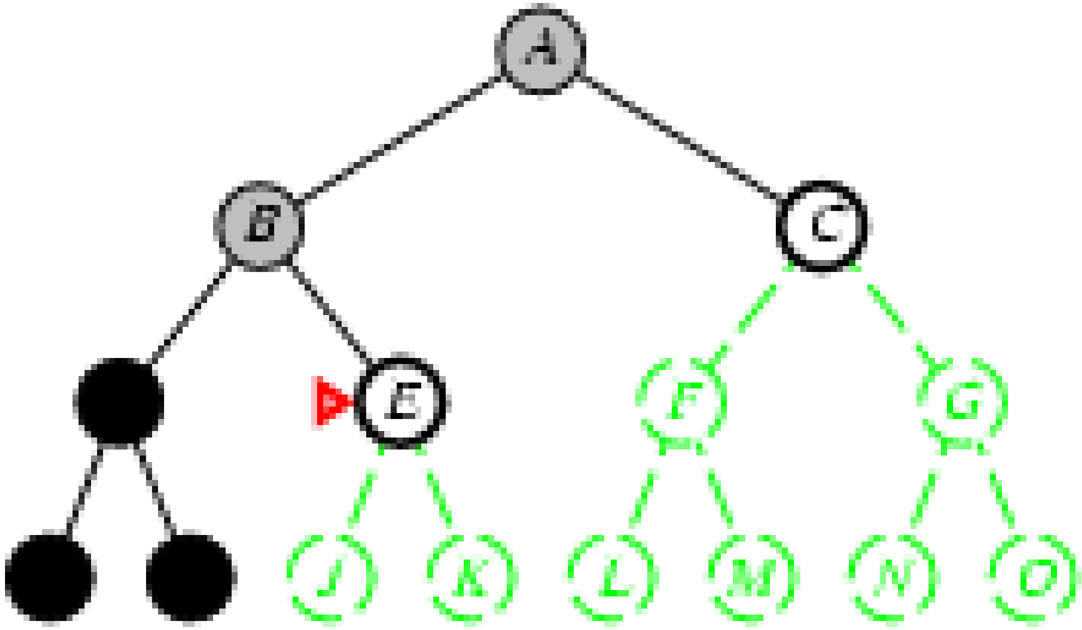
Depth-first search

- Depth-first search (DFS) always expands the deepest node in the search tree
- The search returns to the upper-level nodes only after reaching the leaf node (a node without descendants)
- Implementation: based on LIFO **stack**, i.e., put successors at **front**



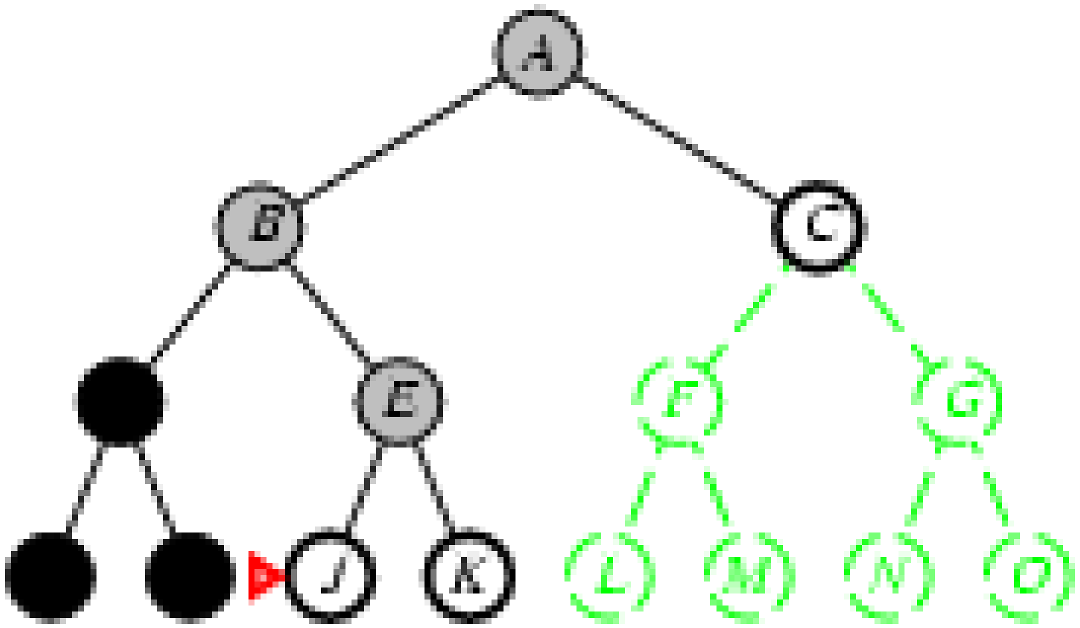
Depth-first search

- Depth-first search (DFS) always expands the deepest node in the search tree
- The search returns to the upper-level nodes only after reaching the leaf node (a node without descendants)
- Implementation: based on LIFO **stack**, i.e., put successors at **front**



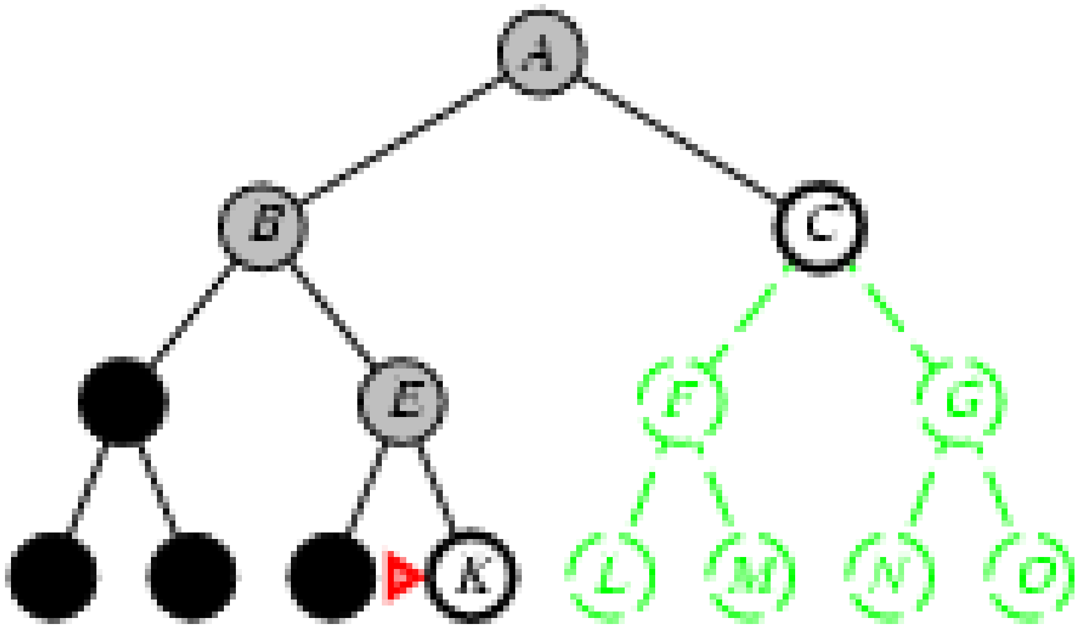
Depth-first search

- Depth-first search (DFS) always expands the deepest node in the search tree
- The search returns to the upper-level nodes only after reaching the leaf node (a node without descendants)
- Implementation: based on LIFO **stack**, i.e., put successors at **front**



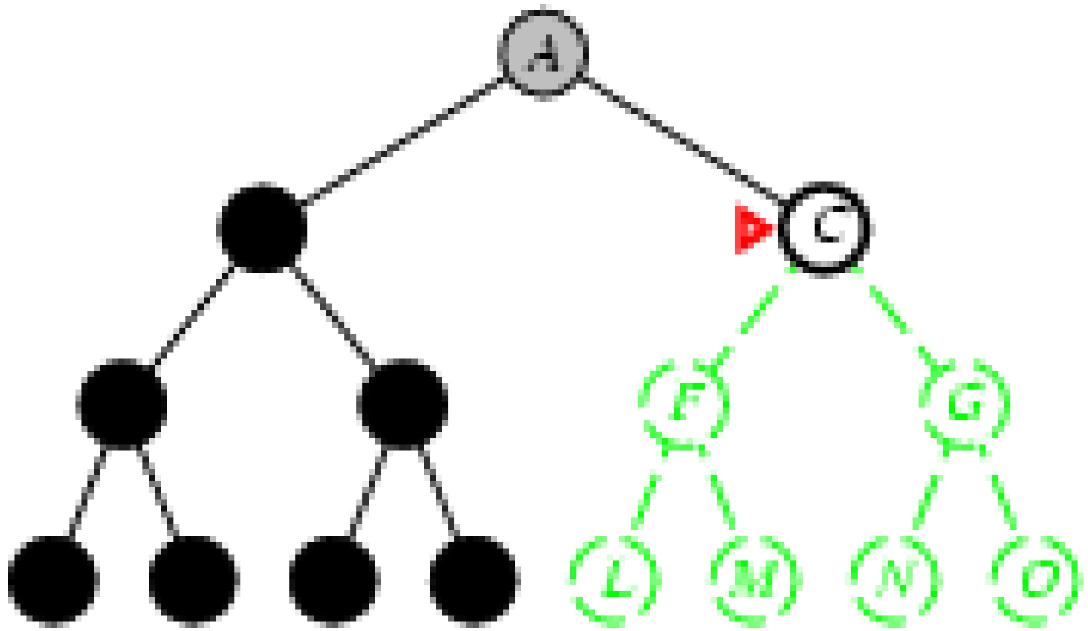
Depth-first search

- Depth-first search (DFS) always expands the deepest node in the search tree
- The search returns to the upper-level nodes only after reaching the leaf node (a node without descendants)
- Implementation: based on LIFO **stack**, i.e., put successors at **front**



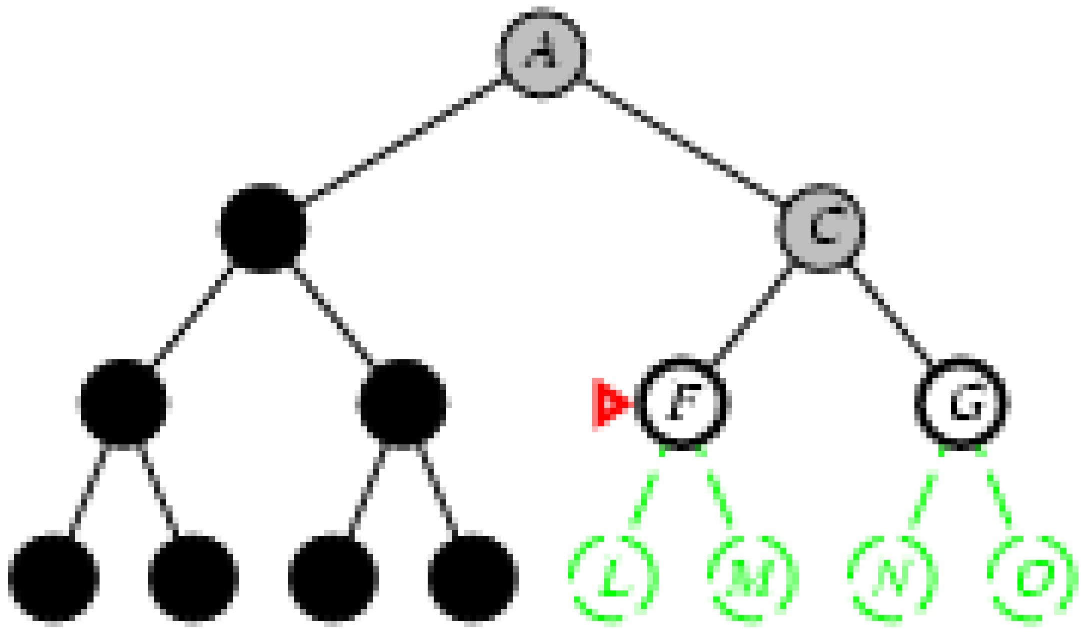
Depth-first search

- Depth-first search (DFS) always expands the deepest node in the search tree
- The search returns to the upper-level nodes only after reaching the leaf node (a node without descendants)
- Implementation: based on LIFO **stack**, i.e., put successors at **front**



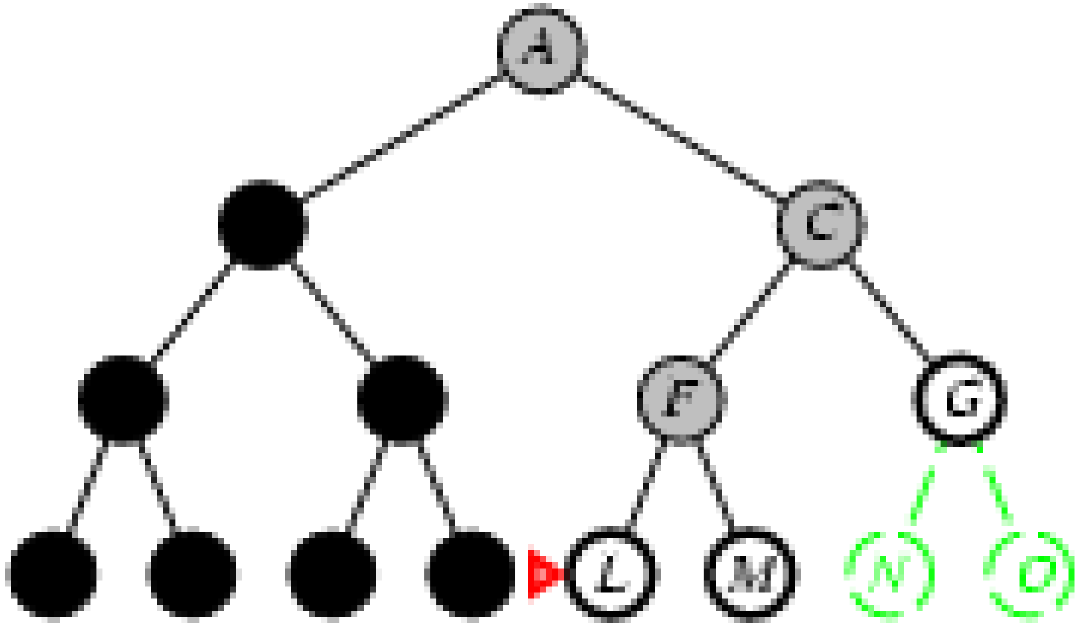
Depth-first search

- Depth-first search (DFS) always expands the deepest node in the search tree
- The search returns to the upper-level nodes only after reaching the leaf node (a node without descendants)
- Implementation: based on LIFO **stack**, i.e., put successors at **front**



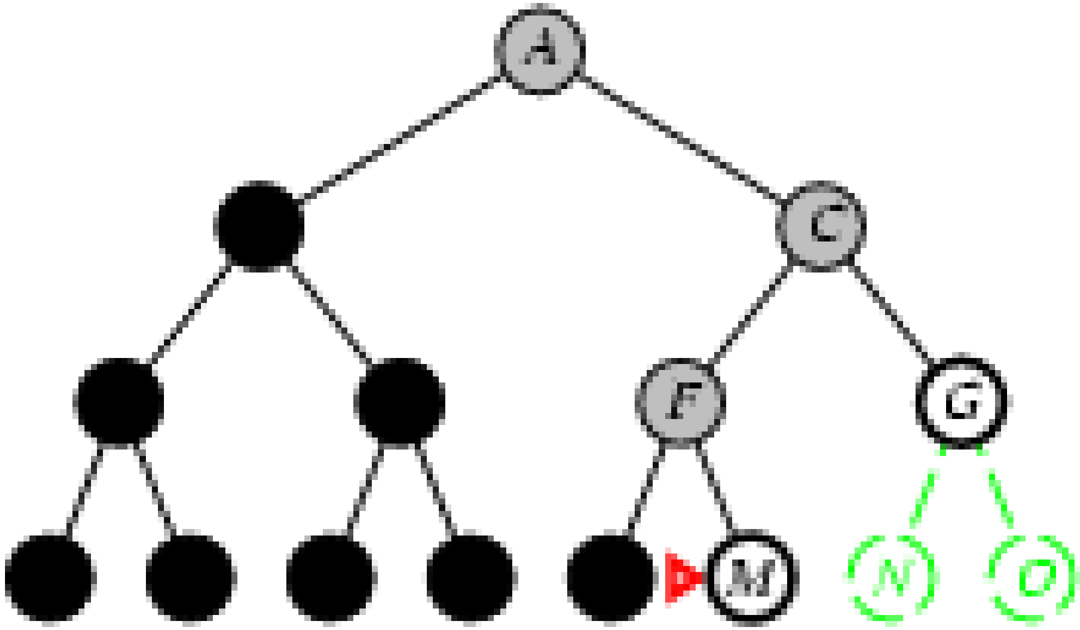
Depth-first search

- Depth-first search (DFS) always expands the deepest node in the search tree
- The search returns to the upper-level nodes only after reaching the leaf node (a node without descendants)
- Implementation: based on LIFO **stack**, i.e., put successors at **front**



Depth-first search

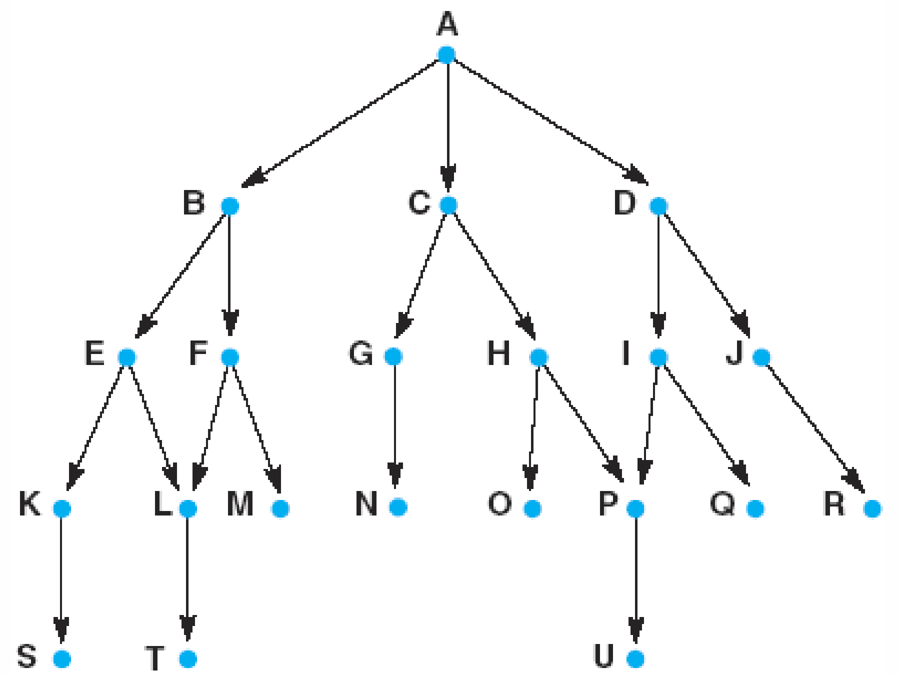
- Depth-first search (DFS) always expands the deepest node in the search tree
- The search returns to the upper-level nodes only after reaching the leaf node (a node without descendants)
- Implementation: based on LIFO **stack**, i.e., put successors at **front**



Depth-first search

Depth-first search

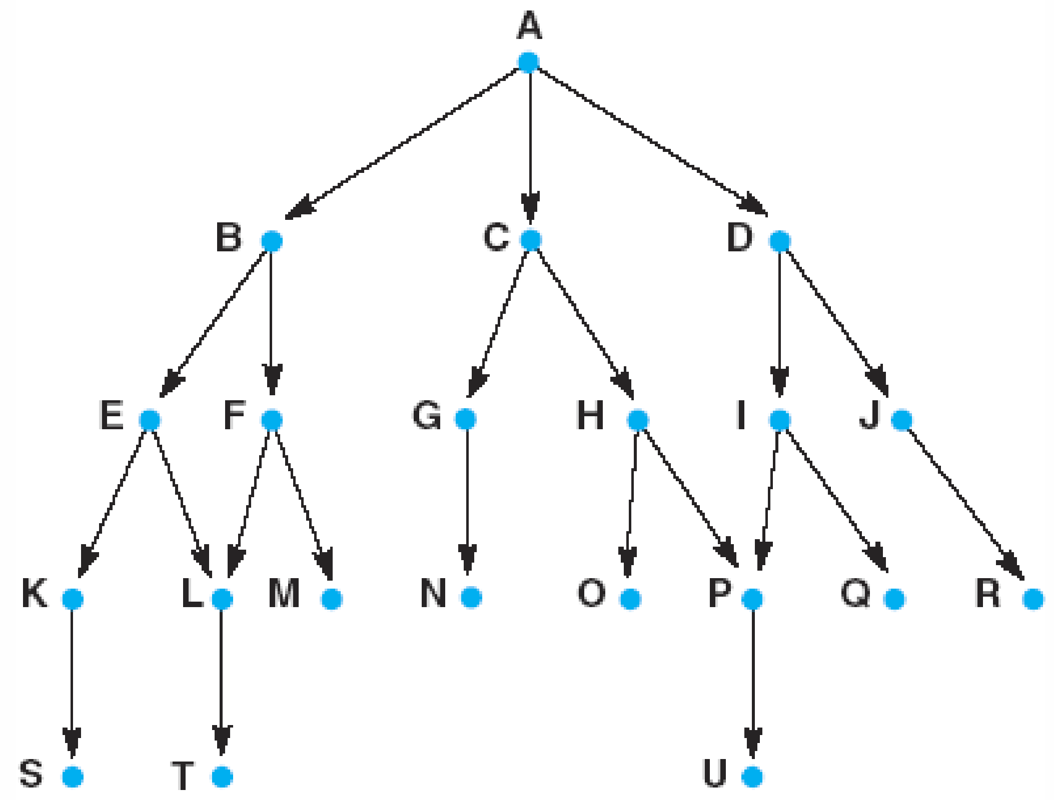
function depthFirstSearch(s_0, succ, goal)
    $open \leftarrow [\,\text{initial}(s_0)\,]$
    while $open \neq [\,]$ **do**
        $n \leftarrow \text{removeHead}(open)$
        if goal(state(n)) **then return** n
        for $m \in \text{expand}(n, \text{succ})$ **do**
            $\text{insertFront}(m, open)$
    return *fail*



Starting at node A, our search gives us: A, B, E, K, S, L, T, F, M, C, G, N, H, O, P, U, D, I, Q, J, R

Depth-first search

Example



1. **open = [A]; closed = [ ]**
2. **open = [B,C,D]; closed = [A]**
3. **open = [E,F,C,D]; closed = [B,A]**
4. **open = [K,L,F,C,D]; closed = [E,B,A]**
5. **open = [S,L,F,C,D]; closed = [K,E,B,A]**
6. **open = [L,F,C,D]; closed = [S,K,E,B,A]**
7. **open = [T,F,C,D]; closed = [L,S,K,E,B,A]**
8. **open = [F,C,D]; closed = [T,L,S,K,E,B,A]**
9. **open = [M,C,D]**, as L is already on **closed**; **closed = [F,T,L,S,K,E,B,A]**
10. **open = [C,D]; closed = [M,F,T,L,S,K,E,B,A]**
11. **open = [G,H,D]; closed = [C,M,F,T,L,S,K,E,B,A]**

Depth first search – recursive implementation

- We can avoid using the open list

Depth first search (recursive implementation)

function $\mathrm{depthFirstSearch}(s, \mathrm{succ}, \mathrm{goal})$
    if $\mathrm{goal}(s)$ **then return** s
    for $m \in \mathrm{succ}(s)$ **do**
        $r \leftarrow \mathrm{depthFirstSearch}(m, \mathrm{succ}, \mathrm{goal})$
        if $r \neq \mathit{fail}$ **then return** r
    return fail

- We use the system stack instead of explicitly using the open list
- Space complexity reduces to O(m)

Properties of Depth-first search

- Complete? No: with tree-search, the algorithm may allow loop forever
  Modify to avoid repeated states along path → may help to avoid infinite loops only but not redundant paths

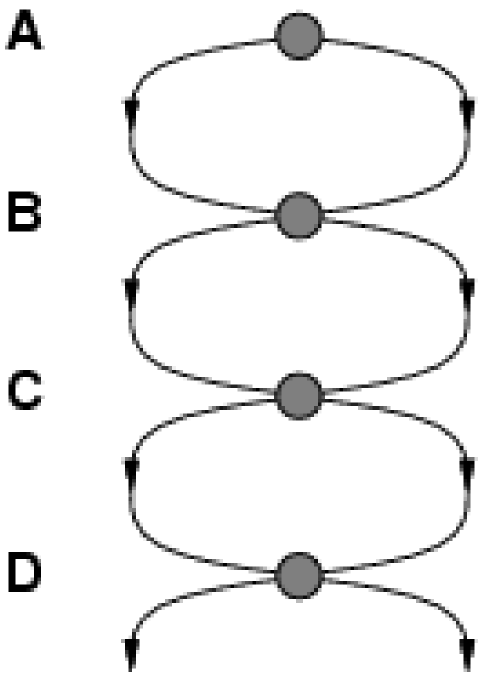
- Time? $O(b^m)$: terrible if m is much larger than d
  - but if solutions are dense, may be much faster than breadth-first.
  - It should be avoided if the maximum search tree depth is large or infinite.

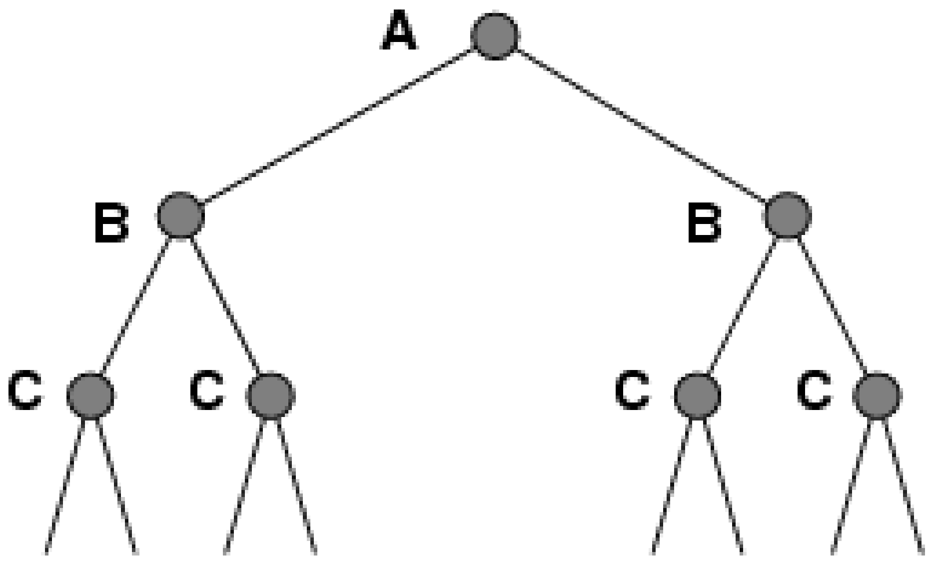
- Space? $O(bm)$, i.e., linear space!

- Optimal? No, because it does not search level-by-level

Repeated states

Failure to detect repeated states can turn a linear problem into an exponential one!





Depth-limited search

- Like depth-first search, but stops at a given limited depth *k*
- It solves the infinite-path problem that is failed in DFS
- Nodes at depth *k is assumed to* have no successors

Depth-limited search

function depthLimitedSearch(s_0, succ, goal, k)
  $open \leftarrow [\,\text{initial}(s_0)\,]$
  while $open \neq [\,]$ **do**
    $n \leftarrow \text{removeHead}(open)$
    if goal(state(n)) **then return** s
    if depth(n) $< k$ **then**
      for $m \in \text{expand}(n, \text{succ})$ **do**
        insertFront(m, $open$)
  return *fail*

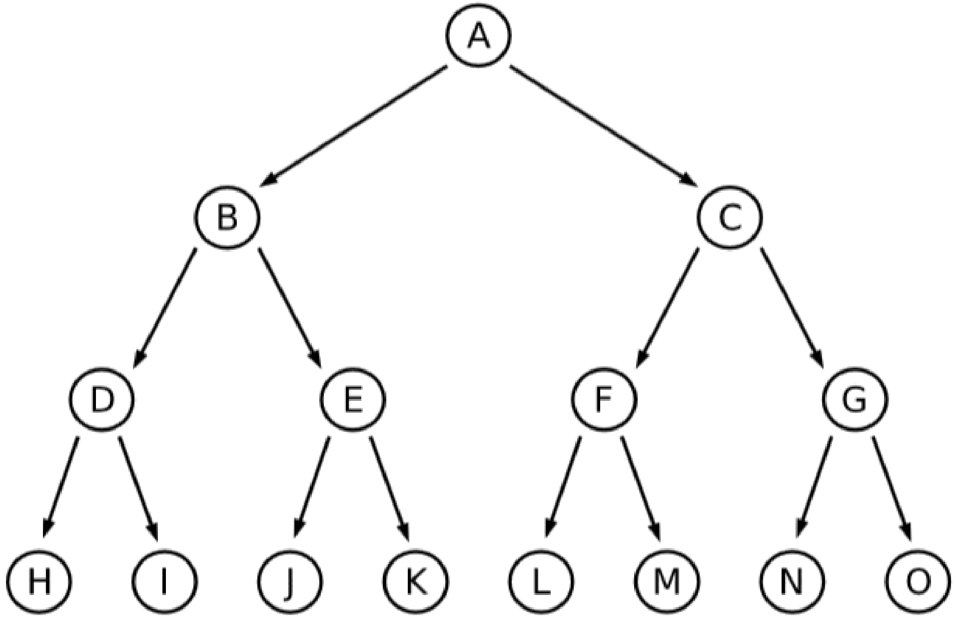
Properties of Depth-limited search

- Complete? No, but may find a solution if $d \leq k$ is provided, and no way if $k<d$
- Time? $O(b^k)$: terrible if k is large
- Space? $O(bk)$, where k is the depth limit time
- Optimal? no, because it does not search level-by-level, especially when $k<d$

This algorithm is useful if we know the solution depth **d**
(we can set k = |S| for reasonably-sized state spaces)

Iterative deepening search

Effectively combines the advantages of DFS and BFS



A, A, B, C, A, B, D, E, C, F,
G A, B, D, H, I, E, J, K, . . .

Iterative deepening search

function iterativeDeepeningSearch(s_0, succ, goal)
  for $k \leftarrow 0$ **to** ∞ **do**
    result $\leftarrow$ depthLimitedSearch(s_0, succ, goal, k)
    if *result* $\neq$ *fail* **then return** *result*
  return *fail*

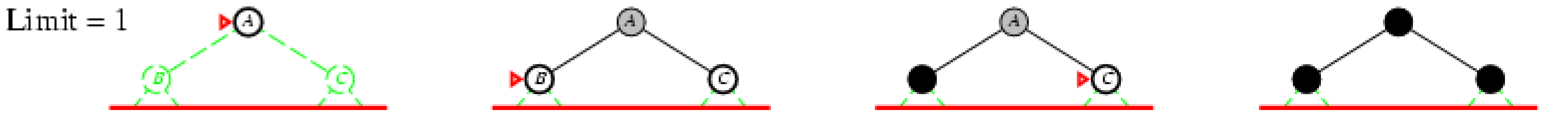
Iterative deepening search

- Avoids the problem of choosing the optimal depth limit by trying out all possible values, starting with depth 0
- The algorithm consists of iterative, depth-first searches, with a maximum depth that increases at each iteration. Maximum depth at the beginning is 1.
- Behavior similar to BFS, but without the spatial complexity.
- Only the actual path is kept in memory; nodes are regenerated at each iteration.
- DFS problems related to infinite branches are avoided.
- To guarantee that the algorithm ends if there is no solution, a general maximum depth of exploration can be defined.

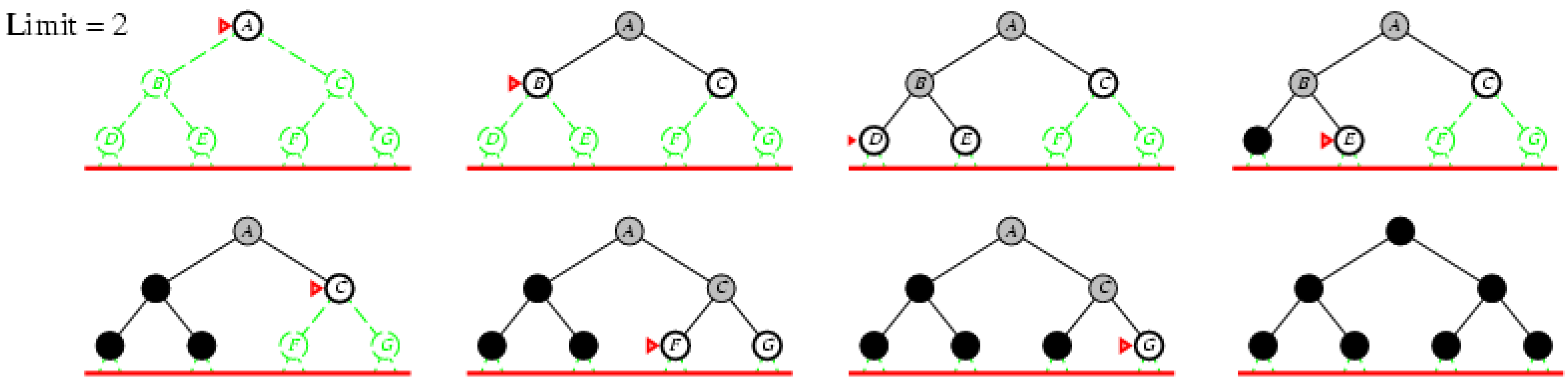
Iterative deepening search l =0

Limit = 0

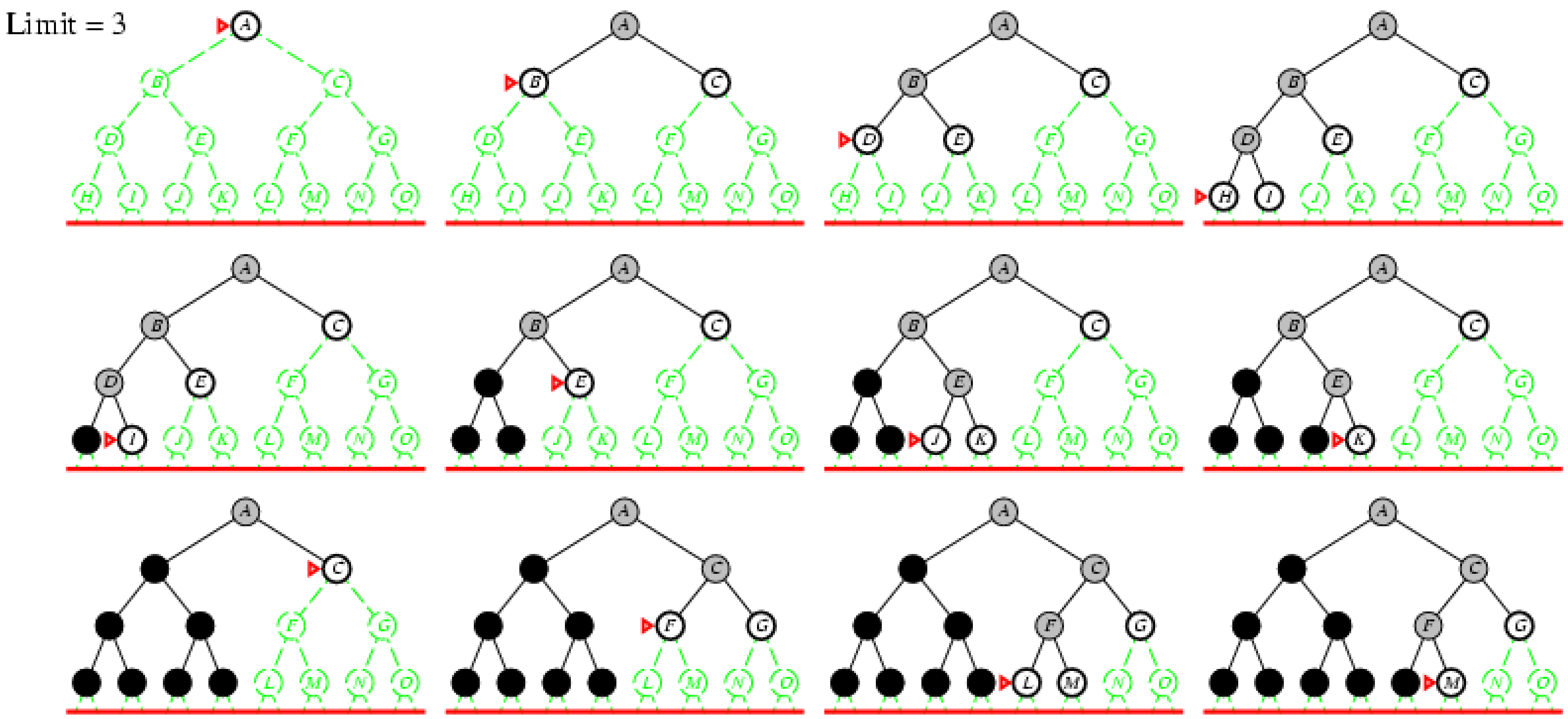
Iterative deepening search l = 1



Iterative deepening search l = 2



Iterative deepening search l = 3



Properties of iterative deepening search

- At first glance, the strategy seems inefficient: the same nodes are expanded many times over again
- In most cases this is not a problem: the majority of nodes are positioned at lower levels, so repeated expansion of the remaining higher-level nodes is not problematic
- Complete? Yes, because it uses a depth limit and gradually increases it
- Time? $O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, because it searches level-by-level

Iterative deepening search is the recommended strategy for problems with big search spaces and unknown solution depth

Comparison between blind algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Summary

- Blind (or uninformed) search algorithms:
  - Information is not taken into account while performing the search for a solution.
- Heuristic (or informed) search algorithms (*it will be covered in detail next week*):
  - A solution cost estimation is used to guide the search.
  - The optimal solution, or even a solution, are not guaranteed.
- All uninformed searching techniques are more alike than different.
- Breadth-first has space issues, and possibly optimality issues.
- Depth-first has time and optimality issues, and possibly completeness issues.
- Depth-limited search has optimality and completeness issues.
- <u>Iterative deepening</u> is the best uninformed search we have explored.

Contents for the next lectures

- **Heuristic Search**
  - Greedy Best First Search,
  - A* Algorithm
  - Local Search Algorithms
    - Hill-climbing search
    - Gradient Descent
    - Simulated annealing
      (suited for either local or global search)
  - Global Search Algorithms
    - Genetic Algorithm

Any questions?