Uninformed search algorithms

Chapter 3, Section 4

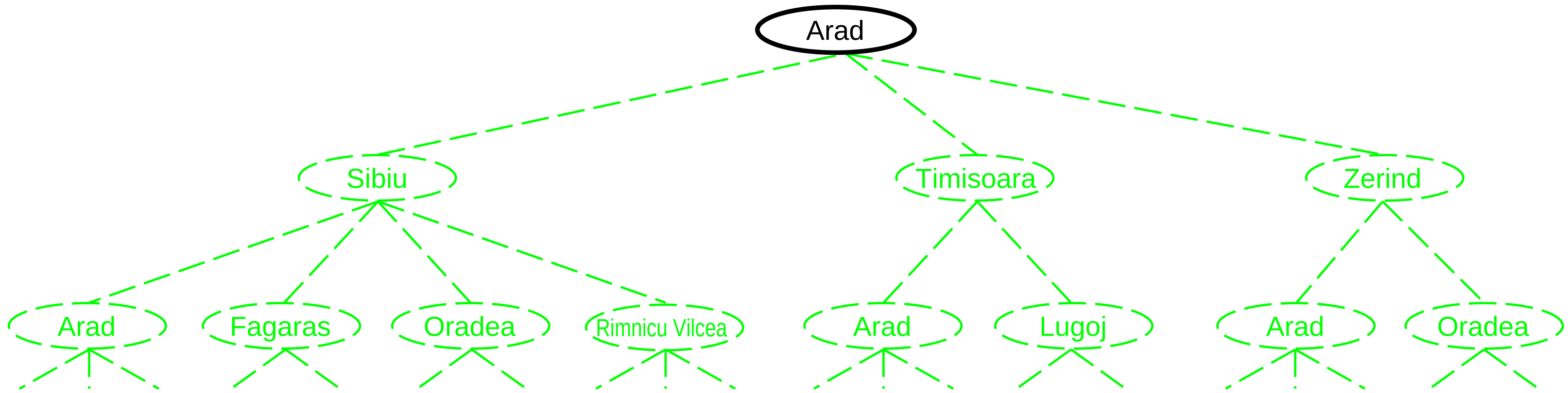
Tree search algorithms

Basic idea:

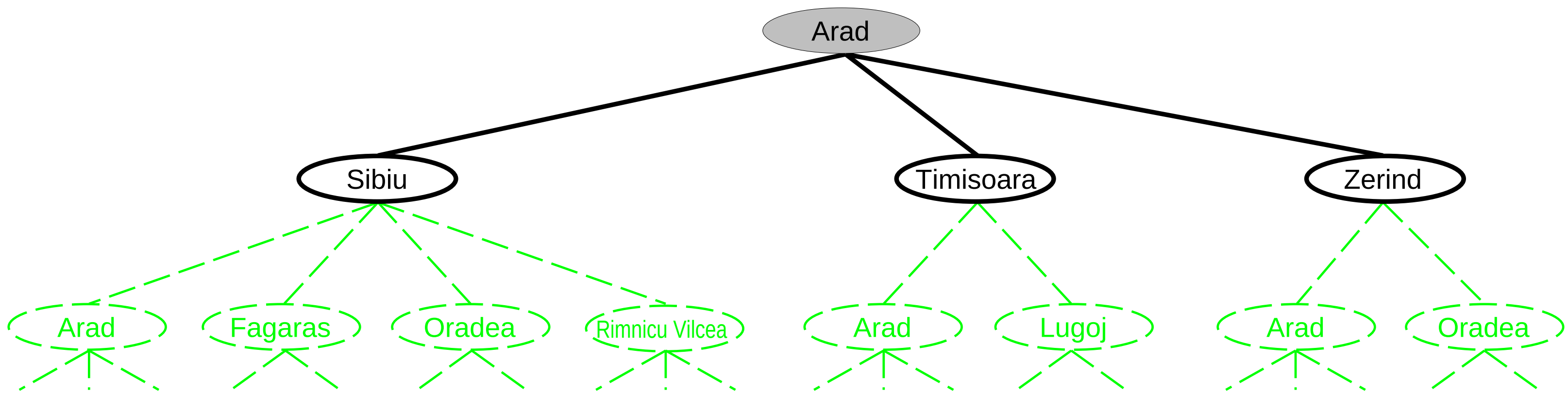
offline, simulated exploration of state space
by generating successors of already-explored states
(a.k.a. expanding states)

```
function TREE-SEARCH( problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node and add the resulting nodes to the frontier
    end
```

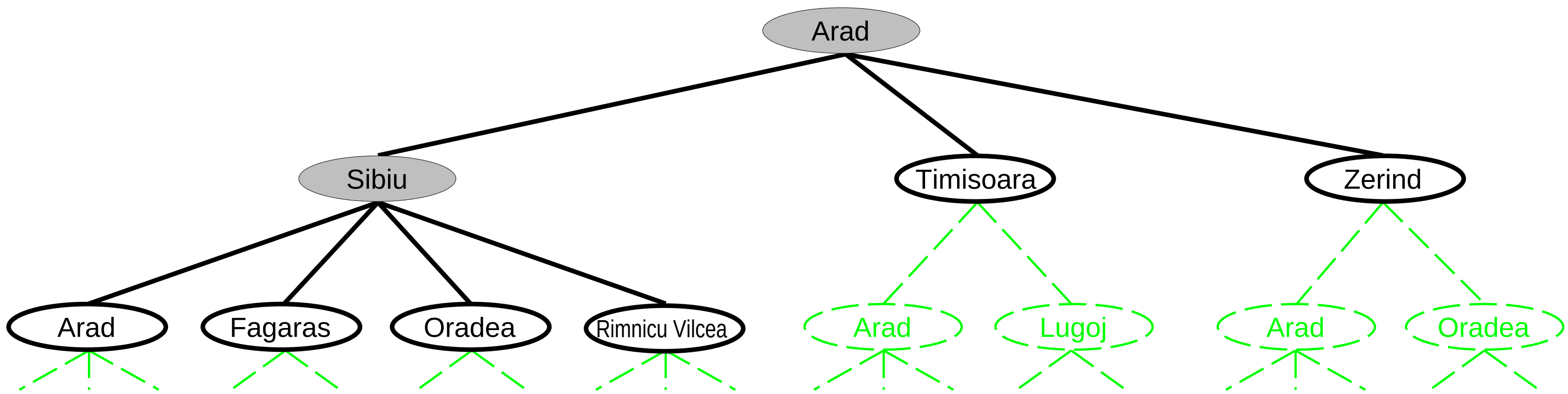
Tree search example



Tree search example



Tree search example



Note: Arad is one of the expanded nodes!

This corresponds to going to Sibiu and then returning to Arad.

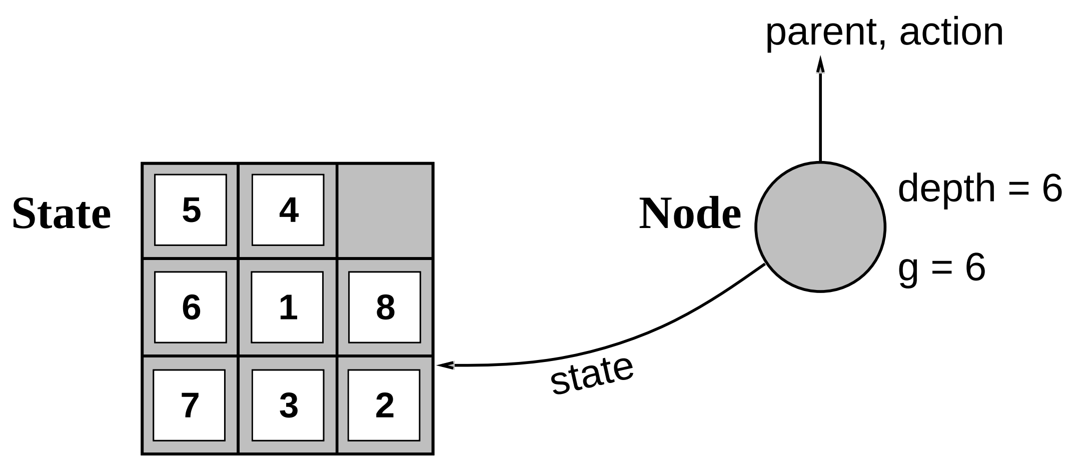
Implementation: states vs. nodes

A state is a (representation of) a physical configuration

A node is a data structure constituting part of a search tree

– includes the state, parent, children, depth, and the path cost $g(x)$

States do not have parents, children, depth, or path cost!



The EXPAND function creates new nodes, filling in the various fields and using the SUCCESSORFN of the problem to create the corresponding states.

Implementation: general tree search

```
function TREE-SEARCH( problem) returns a solution, or failure
    frontier ← {MAKE-NODE(INITIAL-STATE[problem])}
    loop do
        if frontier is empty then return failure
        node ← REMOVE-FRONT(frontier)
        if GOAL-TEST(problem, STATE[node]) return node
        frontier ← INSERTALL(EXPAND(node, problem), frontier)
```

```
function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] +
            STEP-COST(STATE[node], action, result)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Search strategies

A strategy is defined by picking the **order of node expansion**

Strategies are evaluated along the following dimensions:

- completeness—does it always find a solution if one exists?
- optimality—does it always find a least-cost solution?
- time complexity—number of nodes generated/expanded
- space complexity—maximum number of nodes in memory

Time and space complexity are measured in terms of

- b—maximum branching factor of the search tree
- d—depth of the least-cost solution
- m—maximum depth of the state space (may be ∞)

Uninformed search strategies

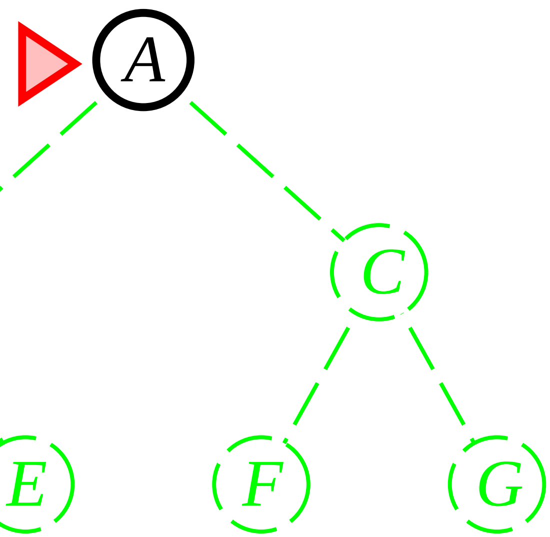
Uninformed strategies use only the information available
in the problem definition

- ◇ Breadth-first search
- ◇ Uniform-cost search
- ◇ Depth-first search
- ◇ Depth-limited search
- ◇ Iterative deepening search

Breadth-first search

Expand the shallowest unexpanded node

Implementation:
frontier is a FIFO queue, i.e., new successors go at end

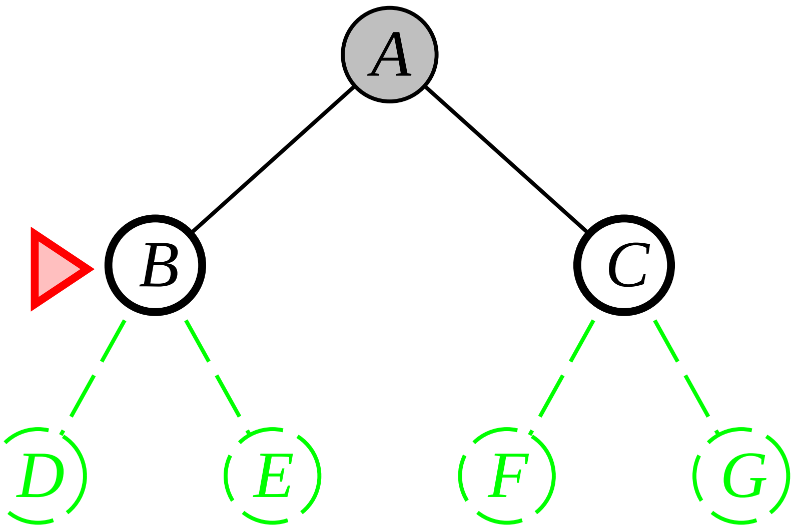


Breadth-first search

Expand shallowest unexpanded node

Implementation:

frontier is a FIFO queue, i.e., new successors go at end

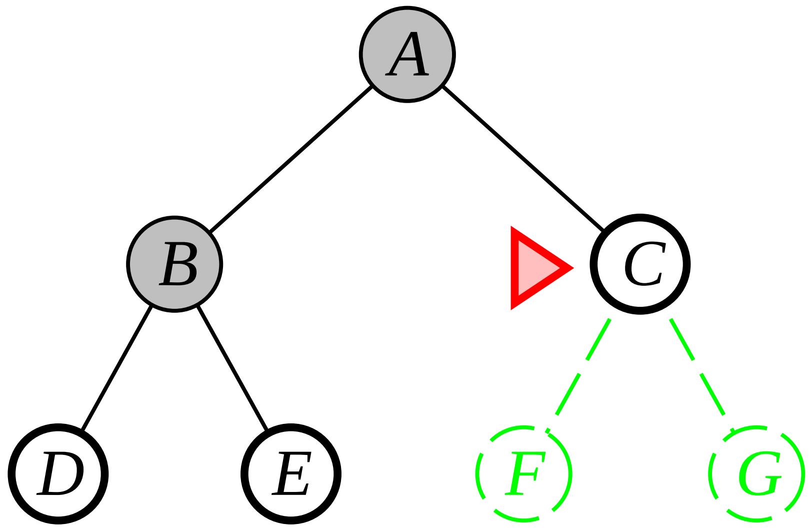


Breadth-first search

Expand shallowest unexpanded node

Implementation:

frontier is a FIFO queue, i.e., new successors go at end

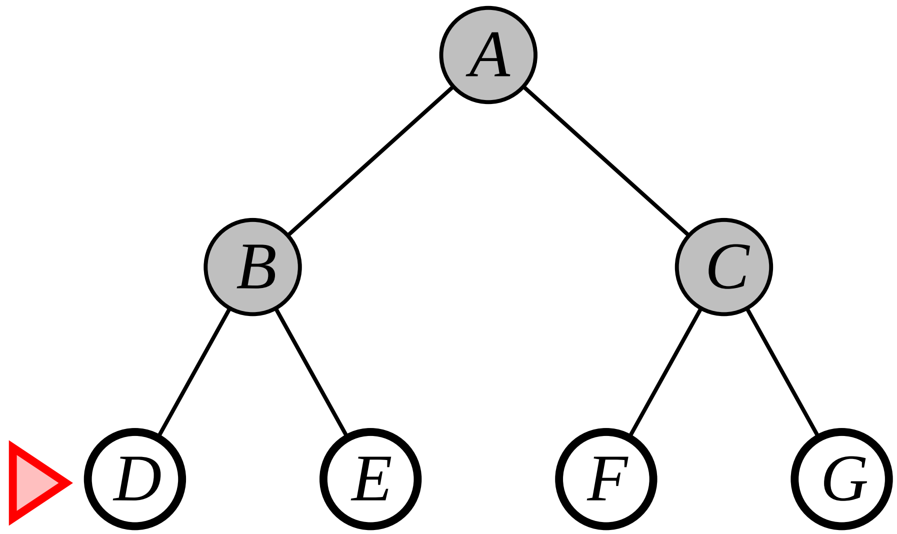


Breadth-first search

Expand shallowest unexpanded node

Implementation:

frontier is a FIFO queue, i.e., new successors go at end



Properties of breadth-first search

Complete?? Yes (if b is finite)

Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^d)$, i.e., exponential in d

Space?? $O(b^d)$ (keeps every node in memory)

Optimal?? Yes, if step cost $= 1$
Not optimal in general

Space is the big problem:
it can easily generate 1M nodes/second
so after 24hrs it has used 86,000GB
(and then it has only reached depth 9 in the search tree)

Uniform-cost search

Expand the cheapest unexpanded node

Implementation:

$\mathit{frontier}$ = priority queue ordered by path cost $g(n)$

Equivalent to breadth-first search, if all step costs are equal

Complete?? Yes, if step cost $\geq \epsilon > 0$

Time?? # of nodes with $g(n) \leq C^*$, i.e., $O(b^{\lceil C^*/\epsilon \rceil})$
where C^* is the cost of the optimal solution
and ϵ is the minimal step cost

Space?? Same as time

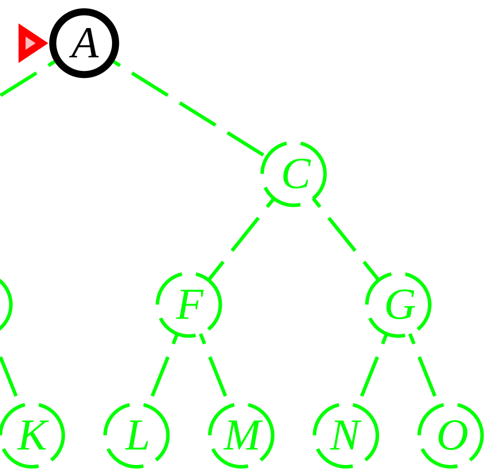
Optimal?? Yes—nodes are expanded in increasing order of $g(n)$

Depth-first search

Expand the deepest unexpanded node

Implementation:

frontier = LIFO queue, i.e., put successors at front

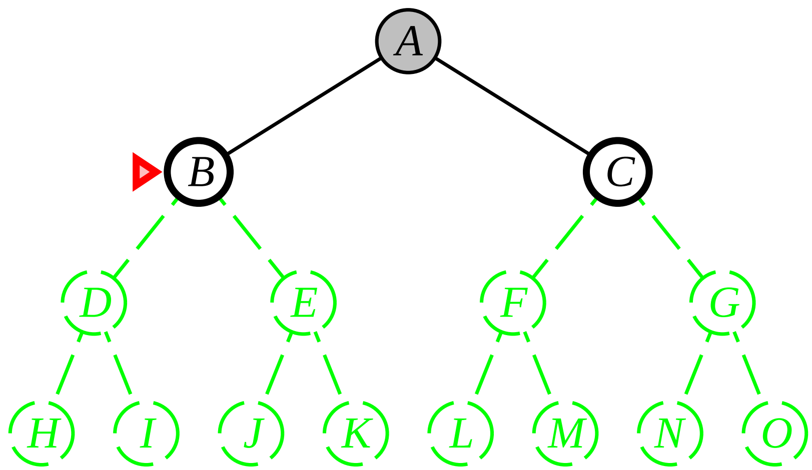


Depth-first search

Expand the deepest unexpanded node

Implementation:

frontier = LIFO queue, i.e., put successors at front

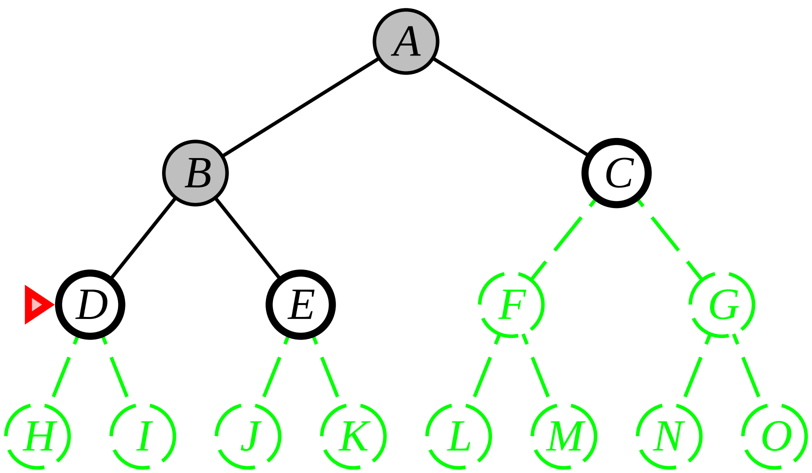


Depth-first search

Expand the deepest unexpanded node

Implementation:

frontier = LIFO queue, i.e., put successors at front

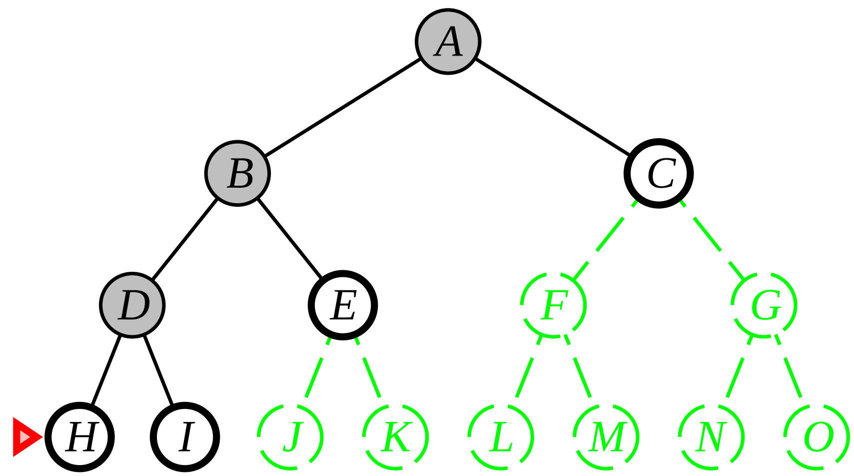


Depth-first search

Expand the deepest unexpanded node

Implementation:

$frontier$ = LIFO queue, i.e., put successors at front

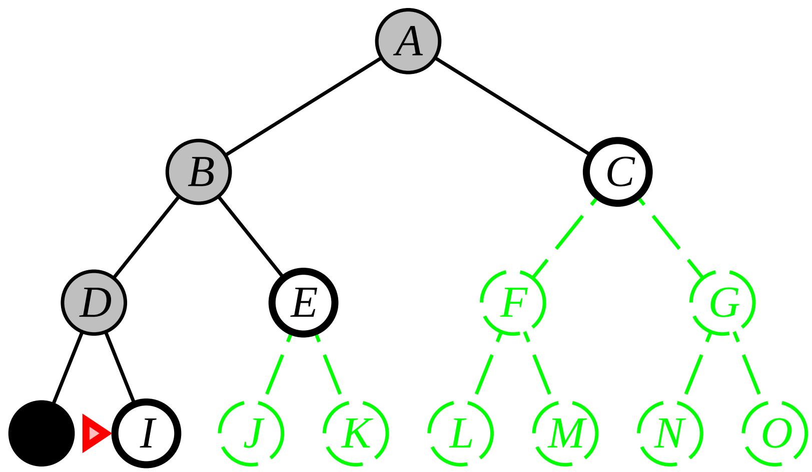


Depth-first search

Expand the deepest unexpanded node

Implementation:

frontier = LIFO queue, i.e., put successors at front

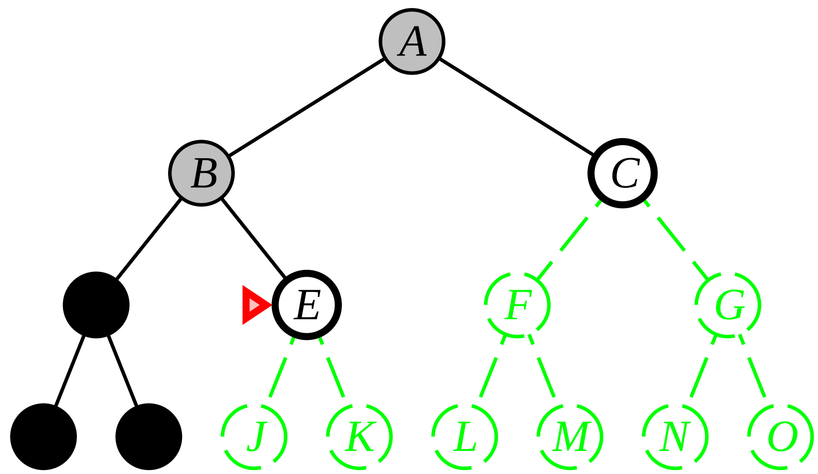


Depth-first search

Expand the deepest unexpanded node

Implementation:

frontier = LIFO queue, i.e., put successors at front



Properties of depth-first search

Complete?? No: it fails in infinite-depth spaces
it also fails in finite spaces with loops
but if we modify the search to avoid repeated states
$\Rightarrow$ complete in finite spaces (even with loops)

Time?? $O(b^m)$: terrible if m is much larger than d
but if solutions are dense, it may be much faster than breadth-first

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

Optimal?? No

Depth-limited search

Depth-first search with depth limit l, i.e., nodes at depth l have no successors

Recursive implementation:

```
function DEPTH-LIMITED-SEARCH( problem, limit) returns soln/failure/cutoff
   RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/failure/cutoff
   if GOAL-TEST(problem, STATE[node]) then return node
   else if limit = 0 then return cutoff
   else
         cutoff-occurred? ← false
         for each action in ACTIONS(STATE[node], problem) do
               child ← CHILD-NODE(problem, node, action)
               result ← RECURSIVE-DLS(child, problem, limit – 1)
               if result = cutoff then cutoff-occurred? ← true
               else if result ≠ failure then return result
         if cutoff-occurred? then return cutoff else return failure
```

Iterative deepening search

Successive depth-limited searches, with higher and higher depth limits, until a goal is found.

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** solution/failure

  for *depth* ← 0 **to** ∞ **do**

    result ← DEPTH-LIMITED-SEARCH(*problem*, *depth*)

    if *result* ≠ cutoff **then return** *result*

  end

Note: This means that shallow nodes will be recalculated several times!

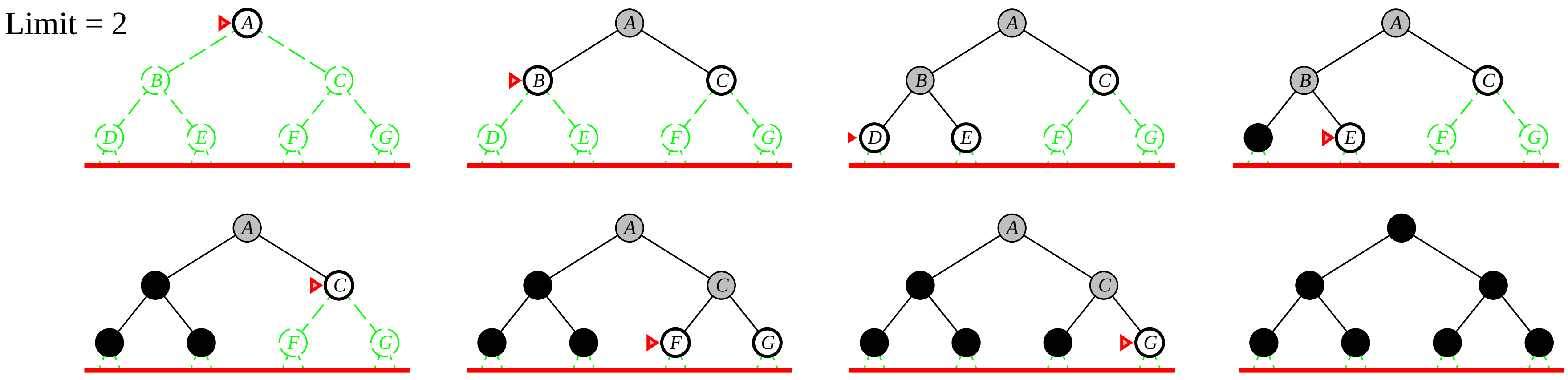
Iterative deepening search $l = 0$

Limit = 0

Iterative deepening search $l = 1$

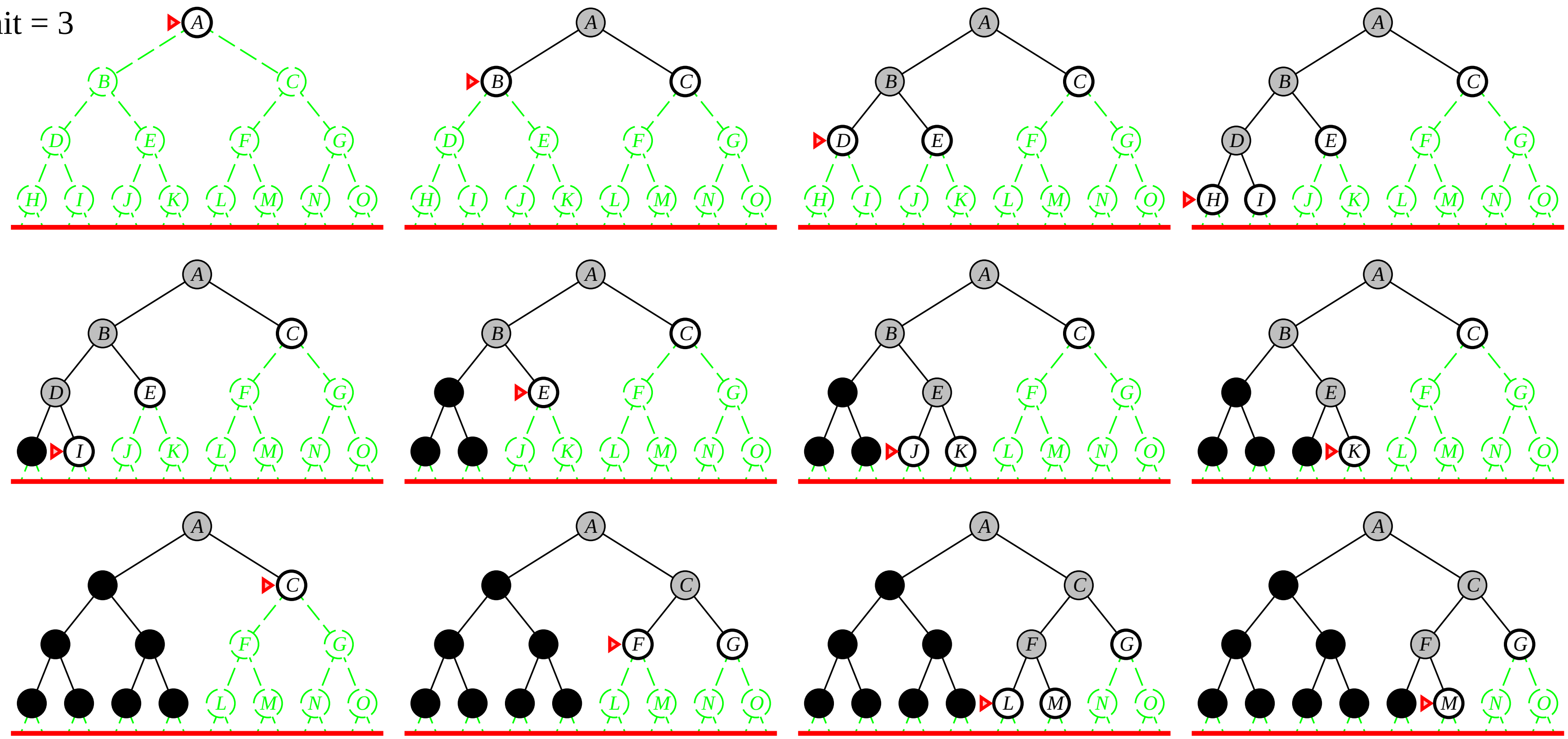
Limit = 1

Iterative deepening search $l = 2$



Iterative deepening search $l = 3$

Limit = 3



Properties of iterative deepening search

Complete?? Yes

Time?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?? $O(bd)$

Optimal?? Yes, if step cost $= 1$
it can be modified to explore a uniform-cost tree

Numerical comparison for $b = 10$ and $d = 5$:

$$N(\text{BFS}) = 10 + 100 + 1,000 + 10,000 + 100,000 = 111,110$$
$$N(\text{IDS}) = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$$

Note: IDS recalculates shallow nodes several times,
but this doesn't have a big effect compared to BFS!

Summary of algorithms

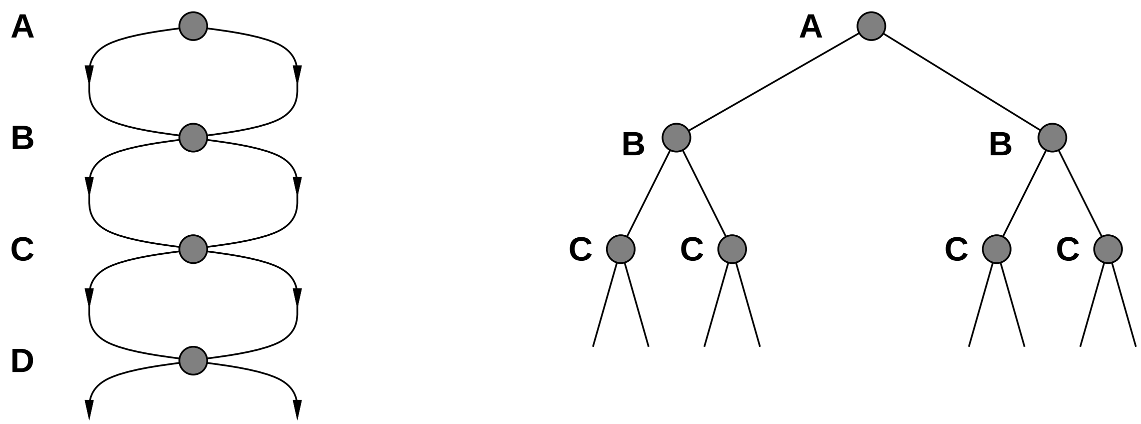
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes, if $\epsilon > 0$	No	Yes, if $l \geq d$	Yes
Time	b^d	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^d	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes	No	No	Yes*

*if all step costs are identical

$$
\begin{aligned}
b &= \text{the branching factor} \\
d &= \text{the depth of the shallowest solution} \\
m &= \text{the maximum depth of the tree} \\
l &= \text{the depth limit} \\
\epsilon &= \text{the smallest step cost} \\
C^* &= \text{the cost of the optimal solution}
\end{aligned}
$$

Repeated states

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



Solution: Use graph search instead of tree search!

Graph search

We augment the tree search algorithm with a set *explored*, which remembers every expanded node

```
function GRAPH-SEARCH( problem) returns a solution, or failure
    frontier ← {MAKE-NODE(INITIAL-STATE[problem])}
    explored ← { }
    loop do
        if frontier is empty then return failure
        node ← REMOVE-FRONT(frontier)
        if GOAL-TEST(problem, STATE[node]) then return node
        add STATE[node] to explored
        if STATE[node] is not in frontier ∪ explored then
            frontier ← INSERTALL(EXPAND(node, problem), frontier)
    end
```

Summary

Variety of uninformed search strategies:

- breadth-first search
- uniform-cost search
- depth-first search
- depth-limited search
- iterative deepening search

Iterative deepening search uses only linear space
and not much more time than other uninformed algorithms

Graph search can be exponentially more efficient than tree search