Methods of Knowledge Based Software Development

Tanel Tammet, Juhan Ernits
Department of Computer Science
Tallinn University of Technology
tanel.tammet@ttu.ee, juhan.ernits@ttu.ee
2016

Search strategies

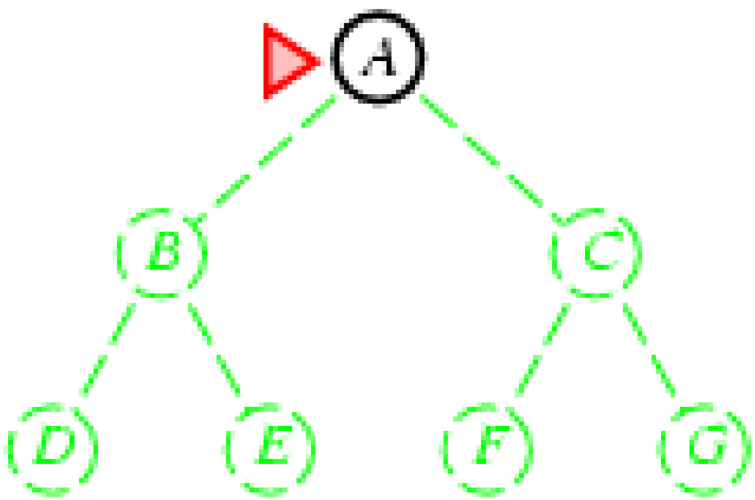
- A search strategy is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?
- 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 search 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

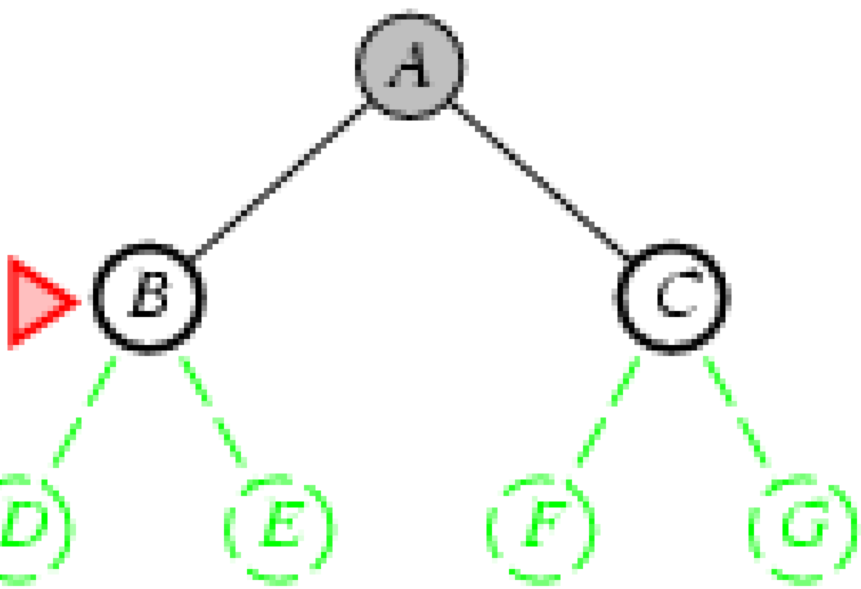
Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  – *frontier* is a FIFO queue, i.e., new successors go at end



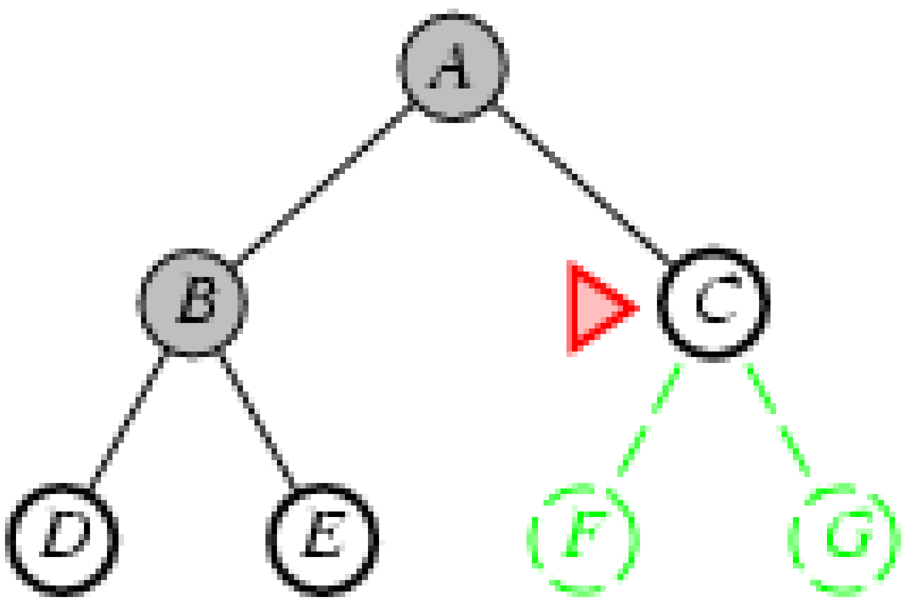
Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *frontier* is a FIFO queue, i.e., new successors go at end



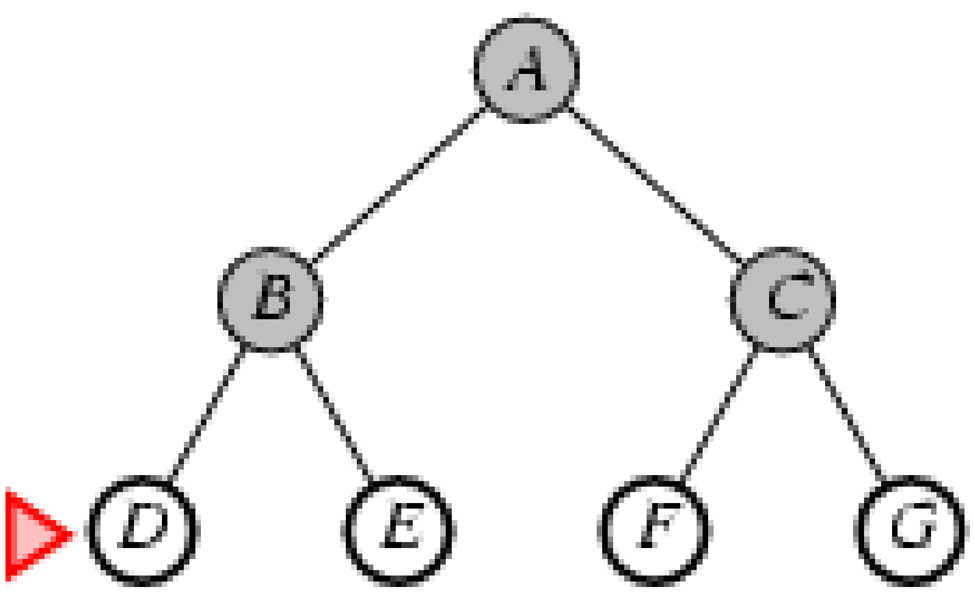
Breadth-first search

- 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?
- Time?
- Space?
- Optimal?

Properties of breadth-first search

- Complete? Yes (if b is finite)
- Time? $1+b+b^2+b^3+\ldots+b^d = O(b^d)$
- Space? $O(b^d)$ (keeps every node in memory)
- Optimal? Yes

- Space is the bigger problem (more than time)

Uniform-cost search

- Expand least-cost unexpanded node
- Implementation:
  - *frontier* = queue ordered by path cost
- Complete?
- Time

- Space?

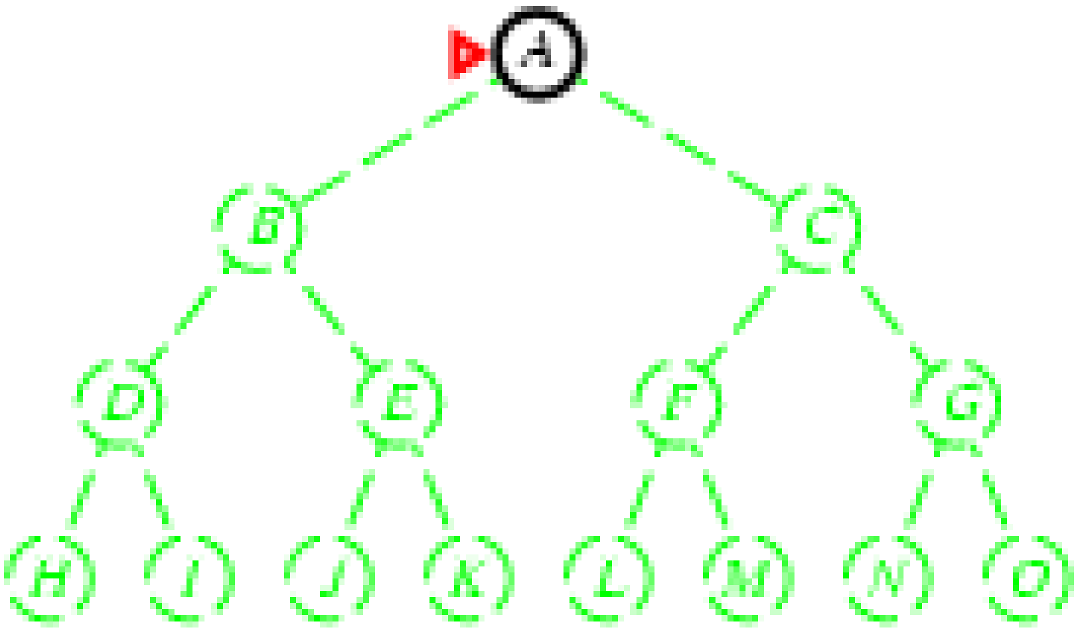
- Optimal?

Uniform-cost search

- Expand least-cost unexpanded node
- Implementation:
  - *frontier* = queue ordered by path cost
- Complete? Yes, if step cost ≥ ε
- Time? # of nodes with *g* ≤ cost of optimal solution, $O(b^{ceiling(C^*/\varepsilon)})$ where C^* is the cost of the optimal solution
- Space? # of nodes with *g* ≤ cost of optimal solution, $O(b^{ceiling(C^*/\varepsilon)})$
- Optimal? Yes – nodes expanded in increasing order of *g(n)*
- Equivalent to breadth-first if step costs all equal

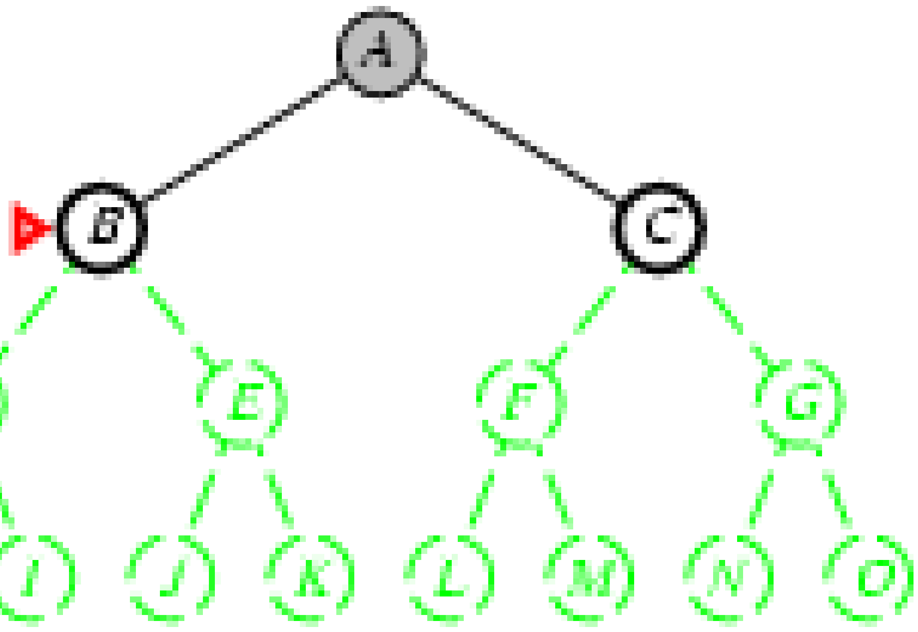
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front



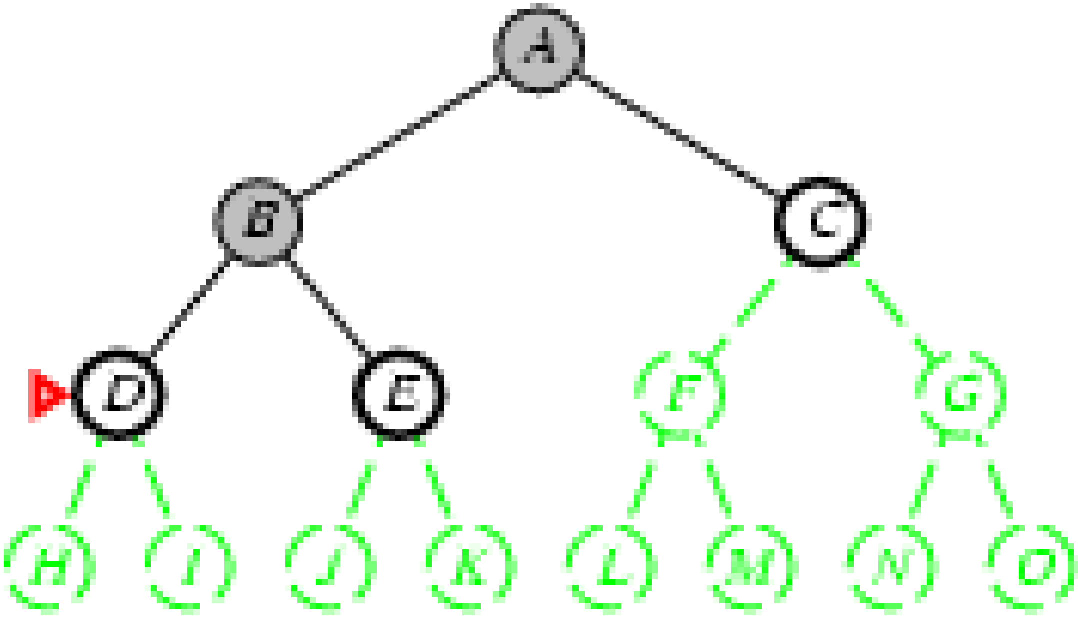
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front



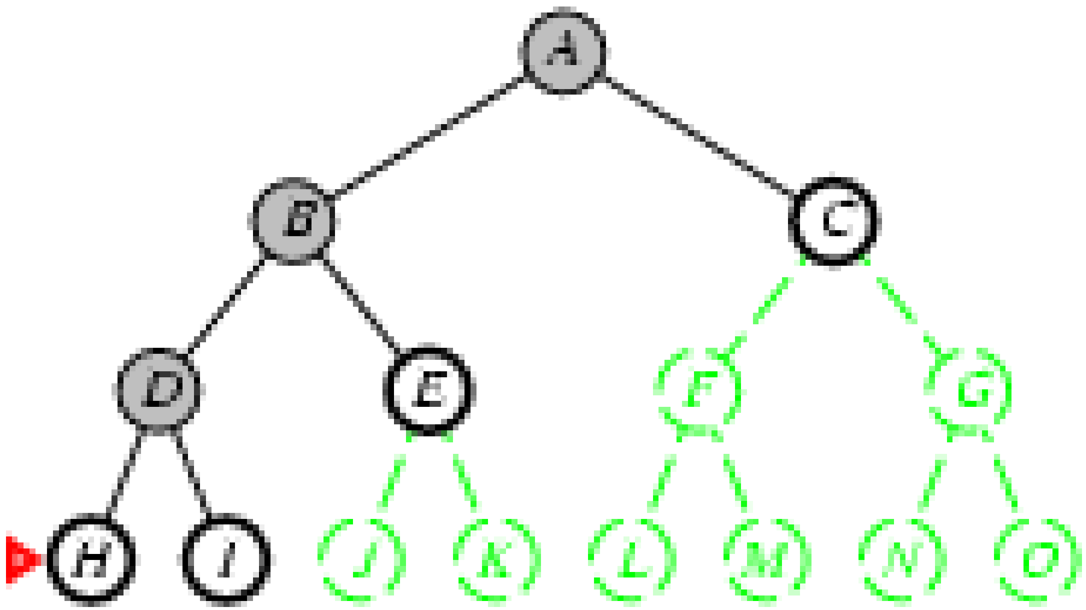
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front



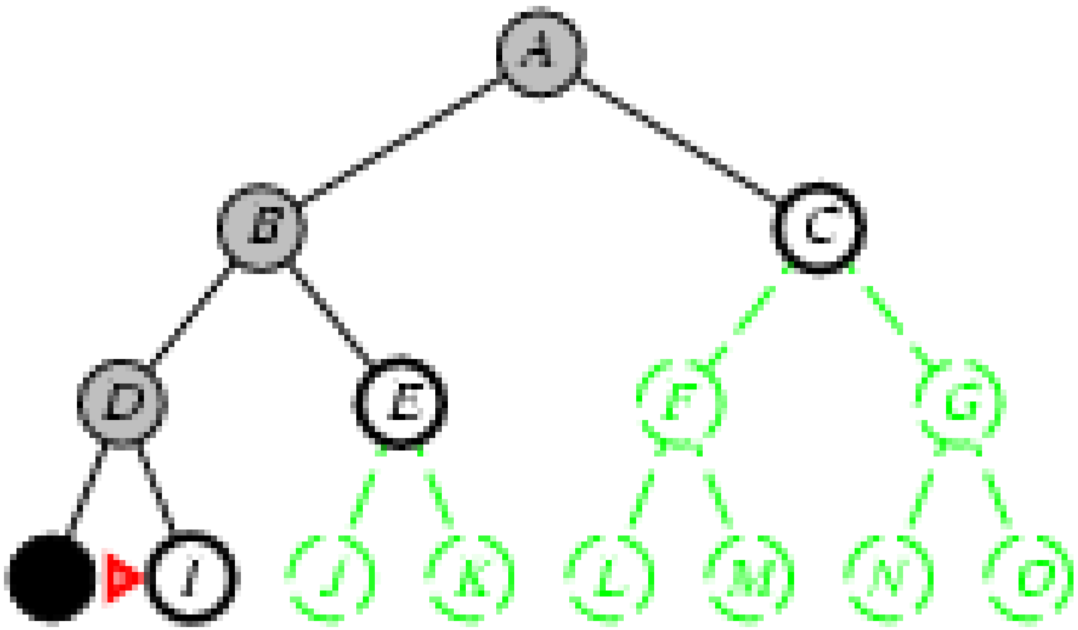
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front



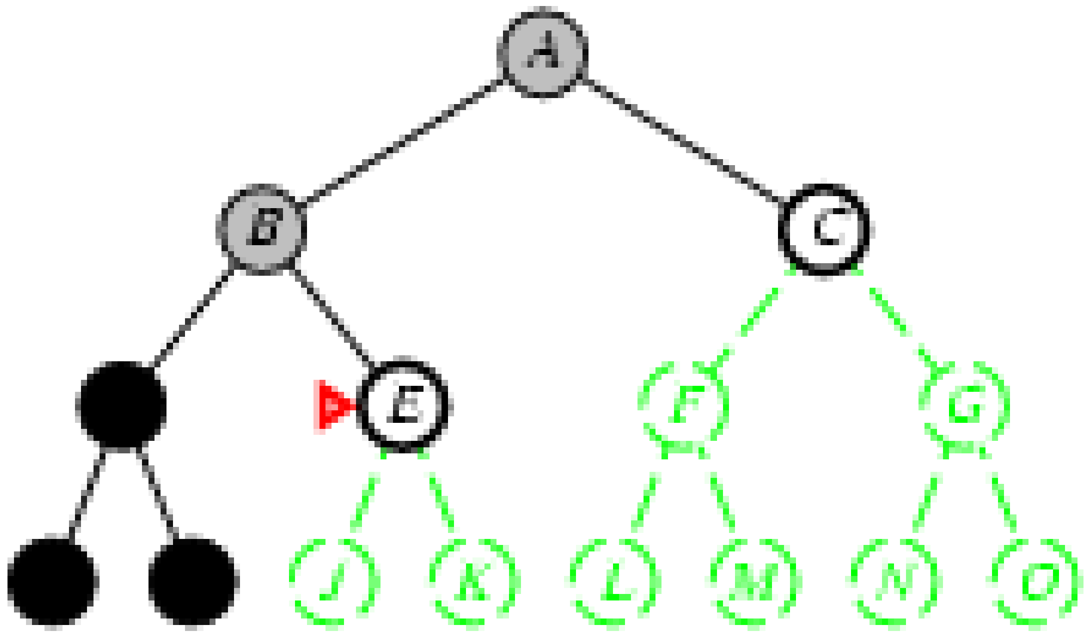
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front



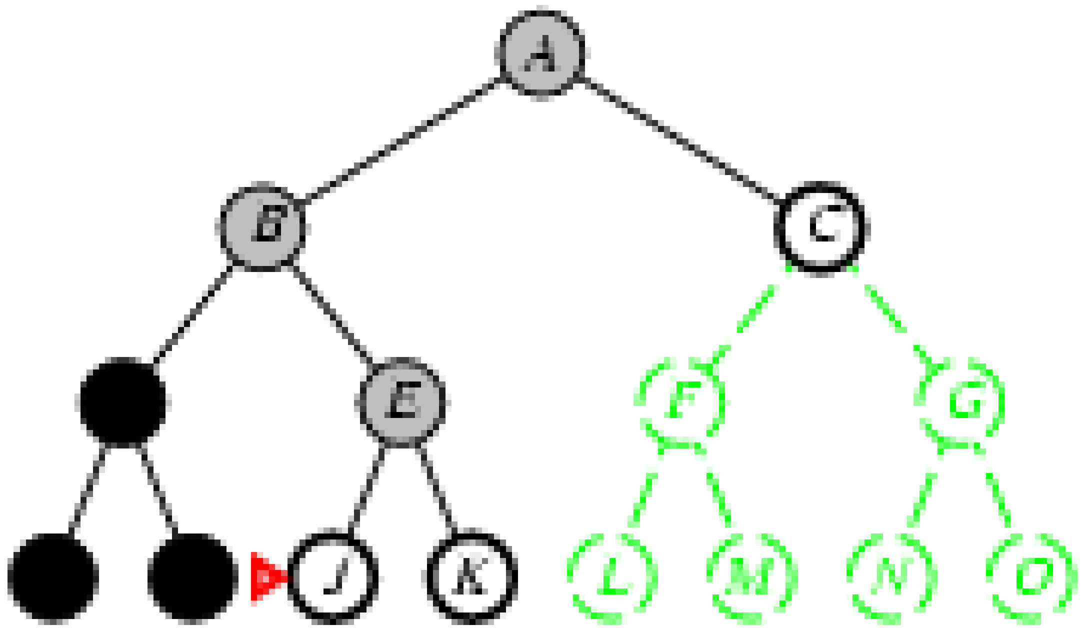
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front
  -



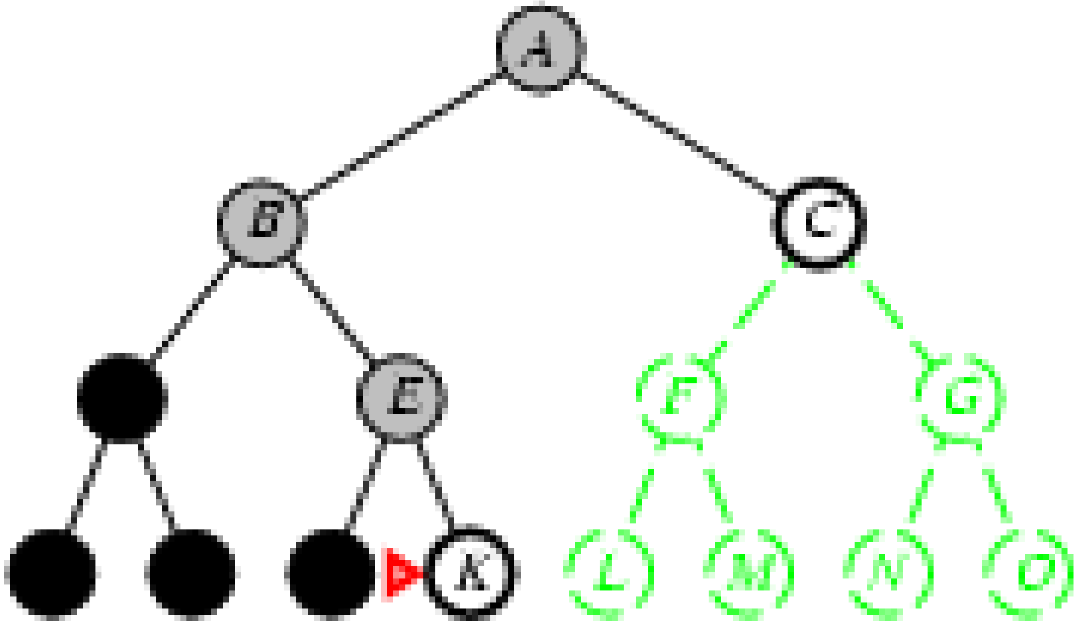
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front



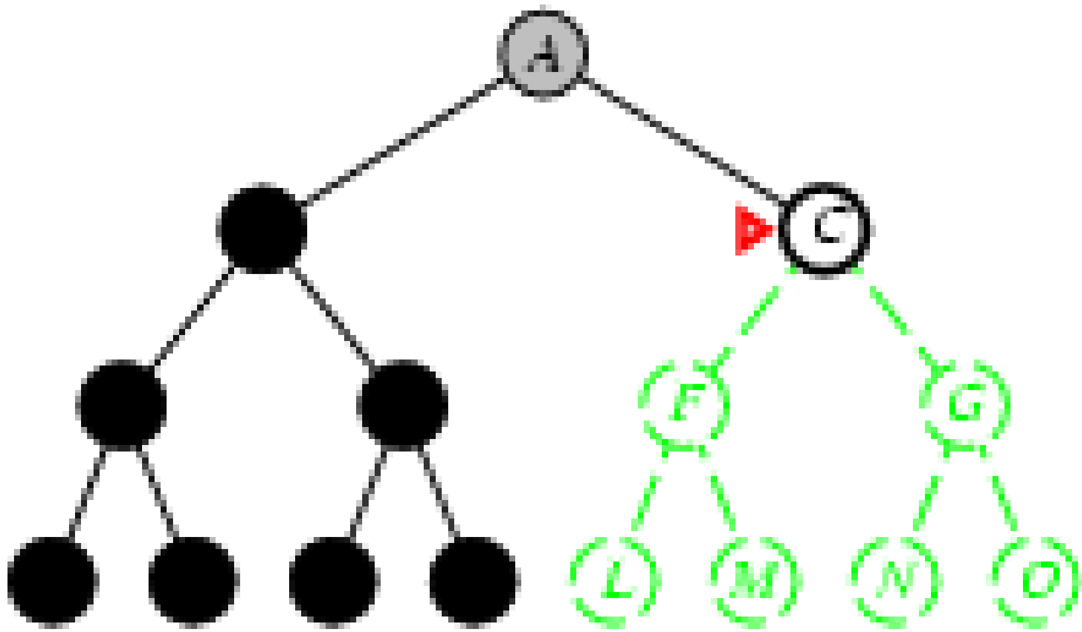
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front



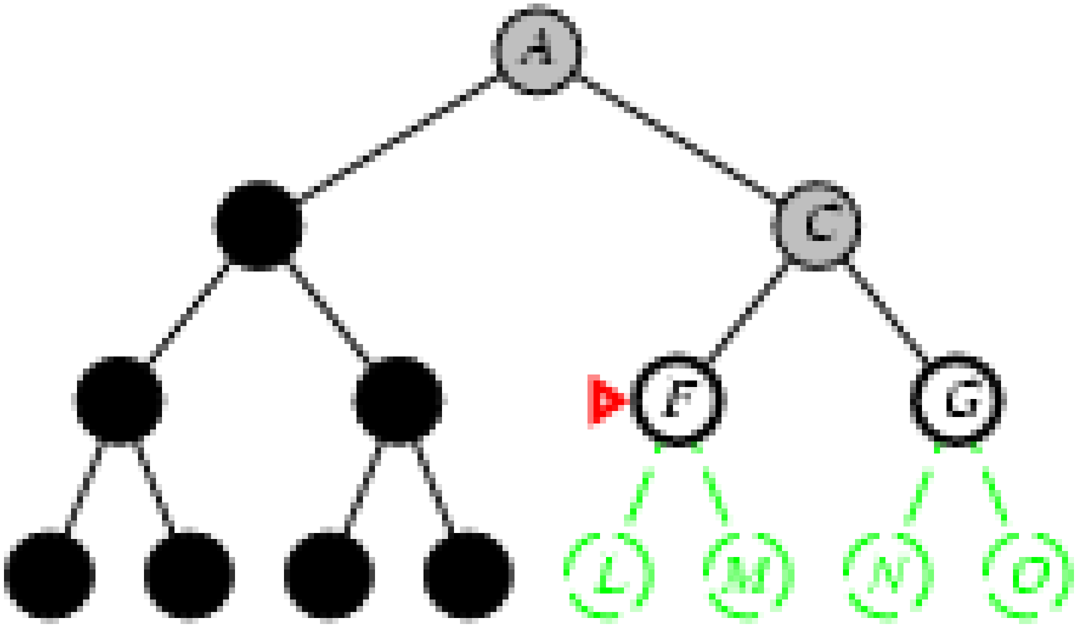
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front



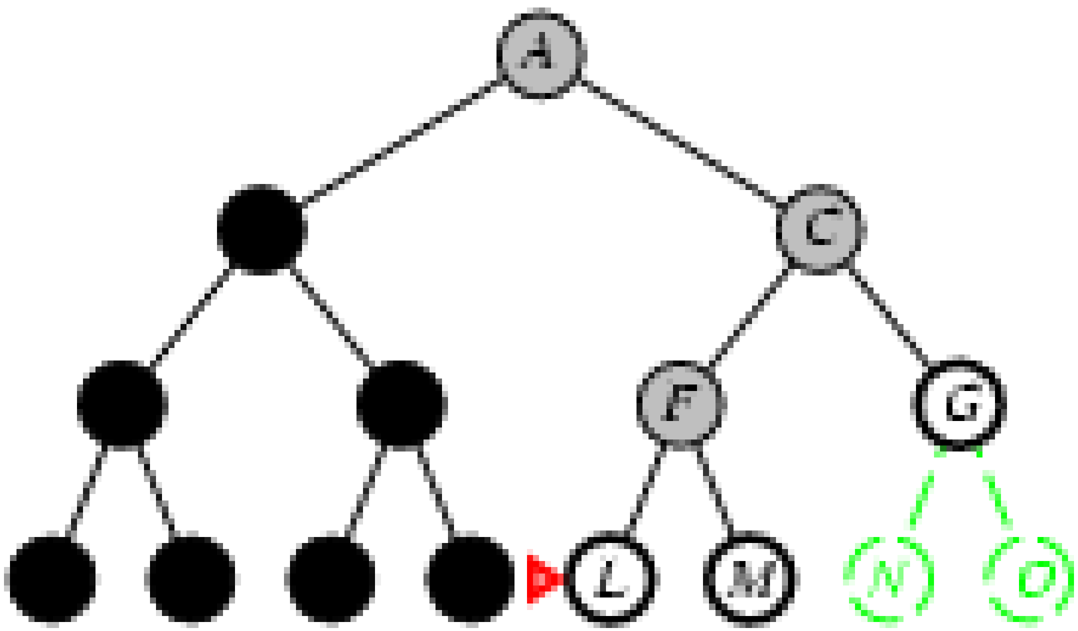
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front



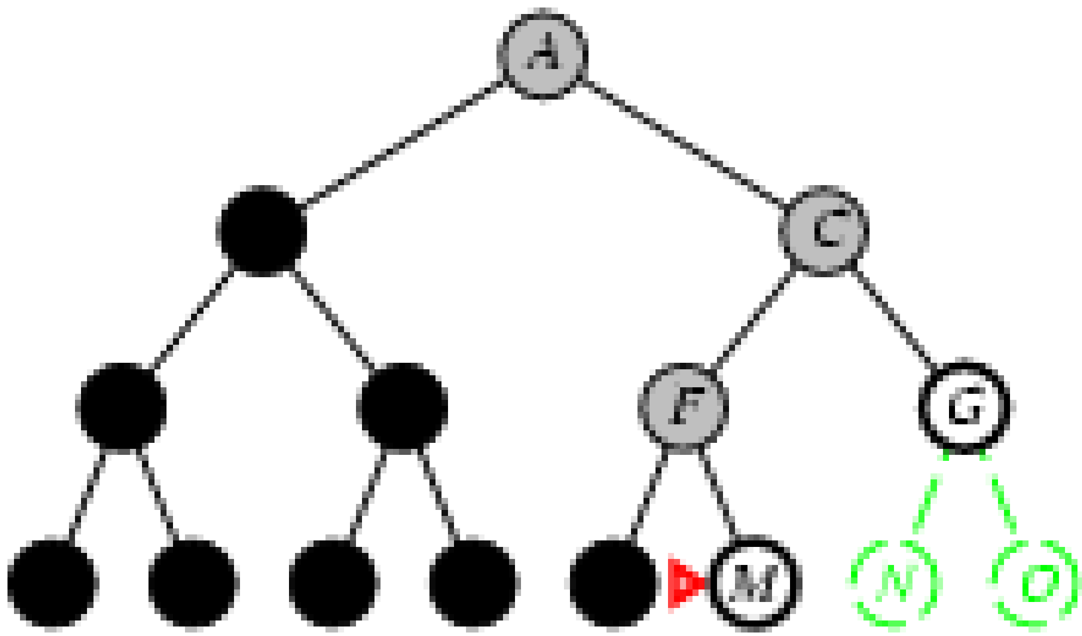
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front



Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front



Properties of depth-first search

- Complete?

- Time

- Space?
- Optimal?

Properties of depth-first search

- <u>Complete?</u> No: fails in infinite-depth spaces, spaces with loops
  - Modify to avoid repeated states along path
    - → complete in finite spaces
- <u>Time?</u> $O(b^m)$: terrible if m is much larger than d
  - but if solutions are dense, may be much faster than breadth-first
- <u>Space?</u> $O(bm)$, i.e., linear space!
- <u>Optimal?</u> 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/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred? ← false
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    else if DEPTH[node] = limit then return cutoff
    else for each successor in EXPAND(node, problem) do
        result ← RECURSIVE-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred? ← true
        else if result ≠ failure then return result
    if cutoff-occurred? then return cutoff else return failure
```

Depth limited search in Python

```python
def depth_limited_search(problem, limit=50):
    "[Fig. 3.17]"
    def recursive_dls(node, problem, limit):
        if problem.goal_test(node.state):
            return node
        elif node.depth == limit:
            return 'cutoff'
        else:
            cutoff_occurred = False
            for child in node.expand(problem):
                result = recursive_dls(child, problem, limit)
                if result == 'cutoff':
                    cutoff_occurred = True
                elif result is not None:
                    return result
            return if_(cutoff_occurred, 'cutoff', None)

    # Body of depth_limited_search:
    return recursive_dls(Node(problem.initial), problem, limit)
```

Iterative deepening search

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution, or fail-ure
  inputs: *problem*, a problem

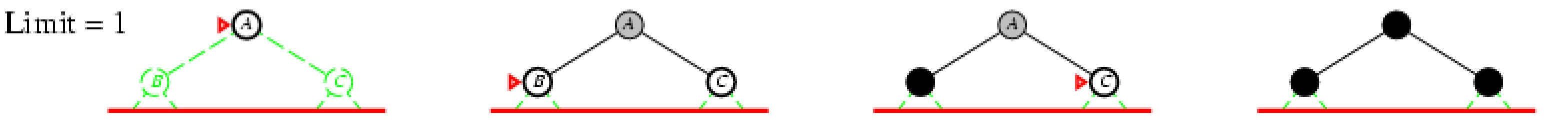
  for *depth* ← 0 **to** ∞ **do**
    result ← DEPTH-LIMITED-SEARCH(*problem*, *depth*)
    if *result* ≠ cutoff **then return** *result*

```
def iterative_deepening_search(problem):
    "[Fig. 3.18]"
    for depth in xrange(sys.maxint):
        result = depth_limited_search(problem, depth)
        if result != 'cutoff':
            return result
```

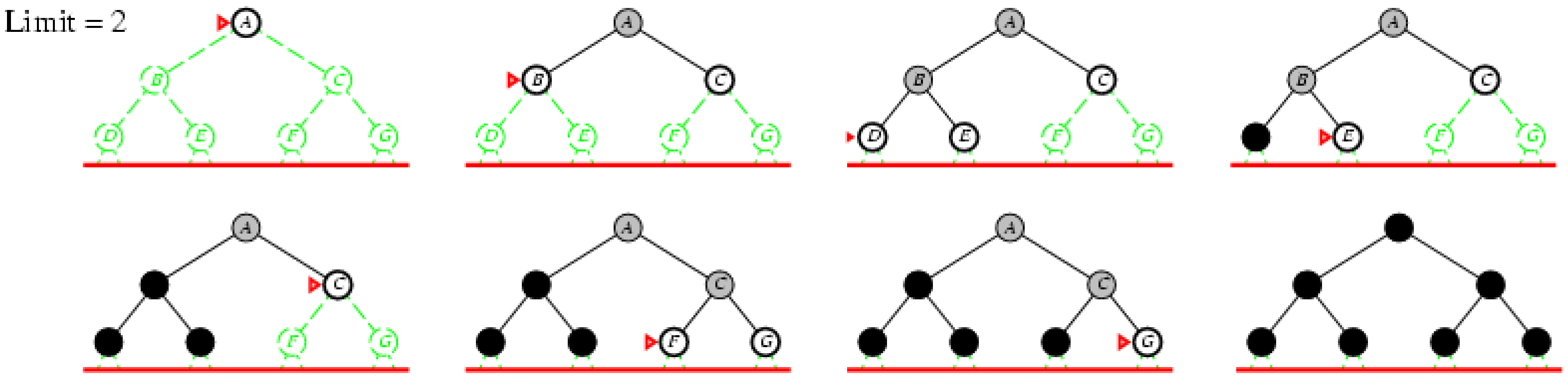
Iterative deepening search *l* =0

Limit = 0

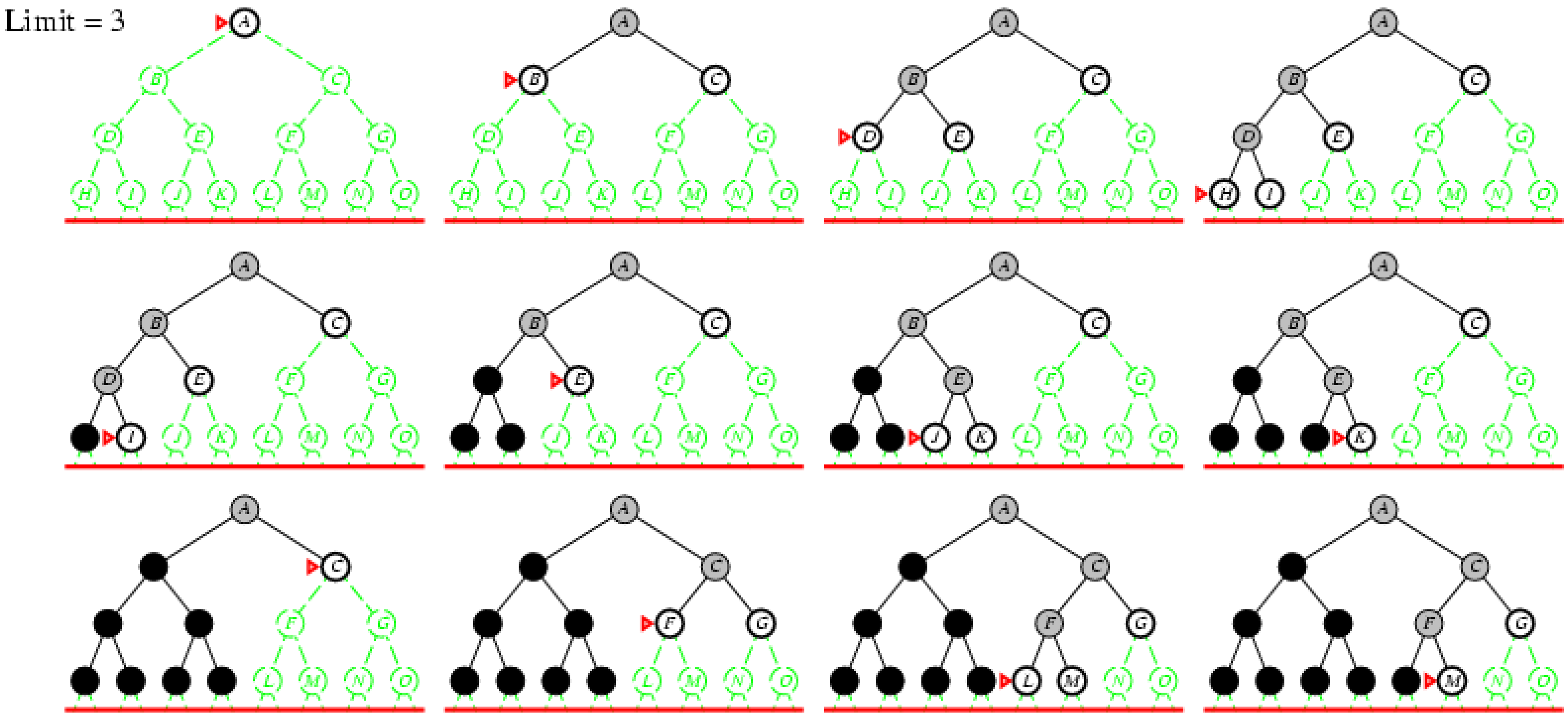
Iterative deepening search *l* =1



Iterative deepening search *l* =2



Iterative deepening search *l* =3



Iterative deepening search

- Number of nodes generated in a depth-limited search to depth *d* with branching factor *b*:

$$N_{DLS} = b^0 + b^1 + b^2 + \dots + b^{d-2} + b^{d-1} + b^d$$

- Number of nodes generated in an iterative deepening search to depth *d* with branching factor *b*:

$$N_{IDS} = (d+1)b^0 + d\, b^1 + (d-1)b^2 + \dots + 3b^{d-2} + 2b^{d-1} + 1b^d$$

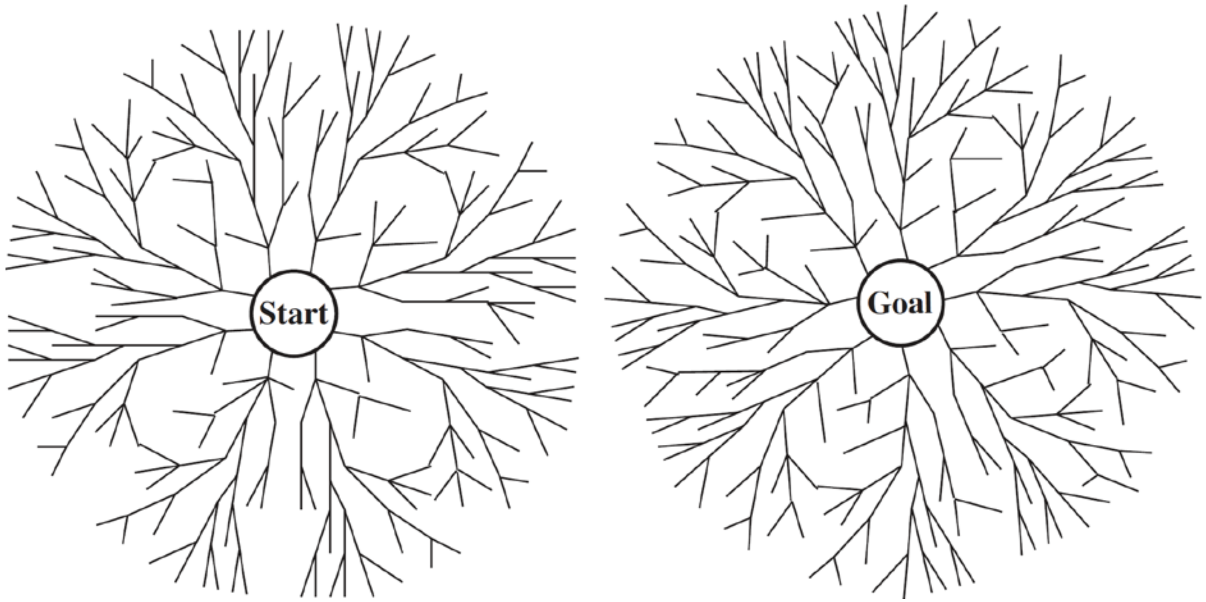
- For *b = 10*, *d = 5*,
-
  - N_{DLS} = 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111
  -
  - N_{IDS} = 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456
  -

- Overhead = (123,456 - 111,111)/111,111 = 11%

Properties of iterative deepening search

- Complete? Yes
- Time? $(d+1)b^0 + d\, b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1

Bidirectional search



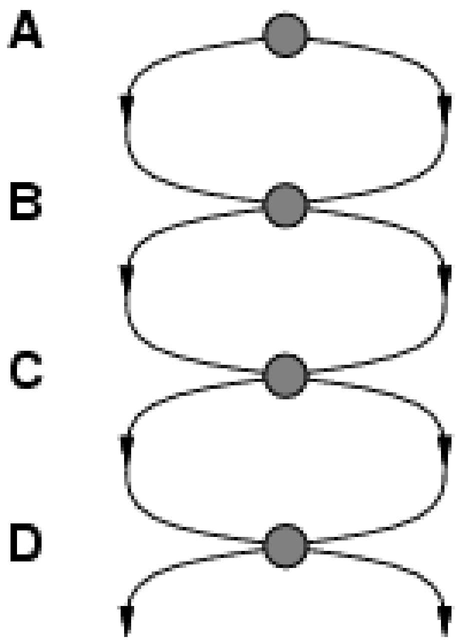
- Run two simultaneous searches in parallel
- Ideally $b^{d/2} + b^{d/2} << b^d$
  – But there has to be an **intersection check** if the frontiers intersect.

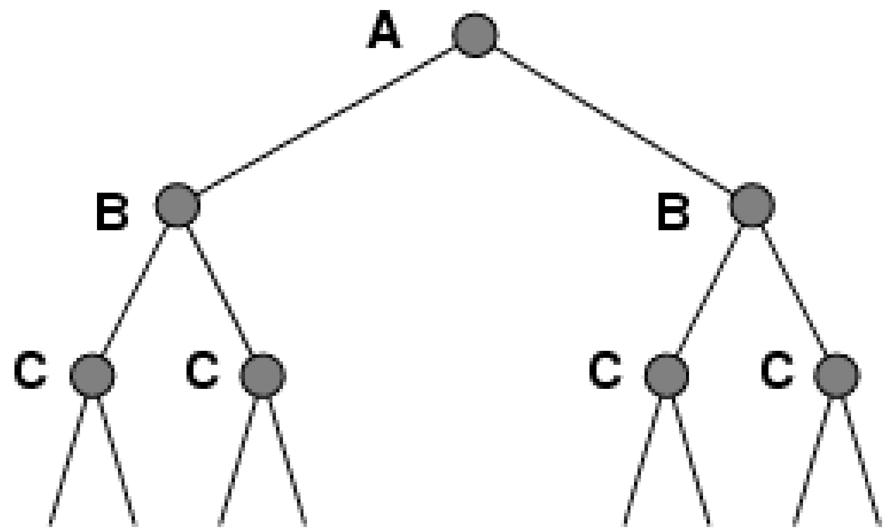
Summary of algorithms

Criterion	Breadth-first	Uniform-cost	Depth-first	Depth-limited	Iterative deepening	Bidirectional (if applicable)
Complete?	Yes	Yes	No	No	Yes	Yes
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	O(bm)	O(bl)	O(bd)	$O(b^{d/2})$
Optimal?	Yes	Yes	No	No	No	Yes

Repeated states

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





Summary

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored
- Variety of uninformed search strategies
- Iterative deepening search uses only linear space and not much more time than other uninformed algorithms

Acknowledgements

- This set of slides contains several prepared by Hwee Tou Ng and Stuart Russell, available from the AIMA pages.