Universidad de Alcalá

Uninformed Search

ISG

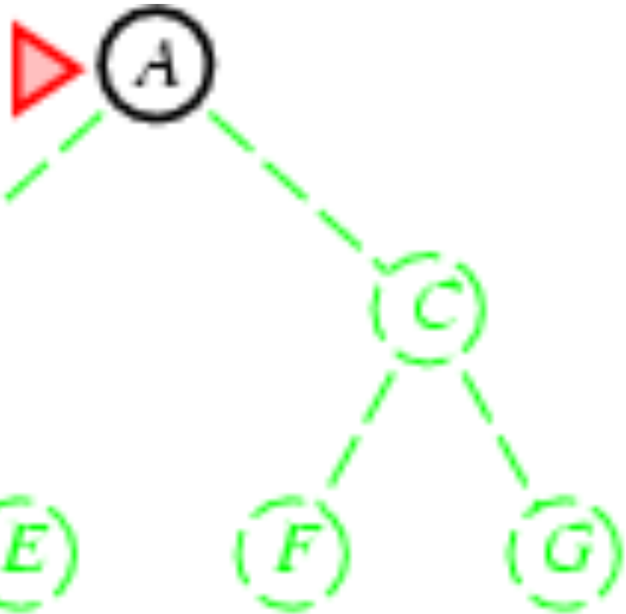
Intelligent Systems Group

Uninformed search strategies

- Uninformed search strategies use only the information available in the problem definition
  - Breadth-first search/ Búsqueda en anchura
  - Uniform-cost search/ Búsqueda de coste uniforme
  - Depth-first search/ Búsqueda en profundidad
  - Depth-limited search/ Búsqueda en profundidad limitada
  - Iterative deepening search/Búsqueda de profundización iterativa

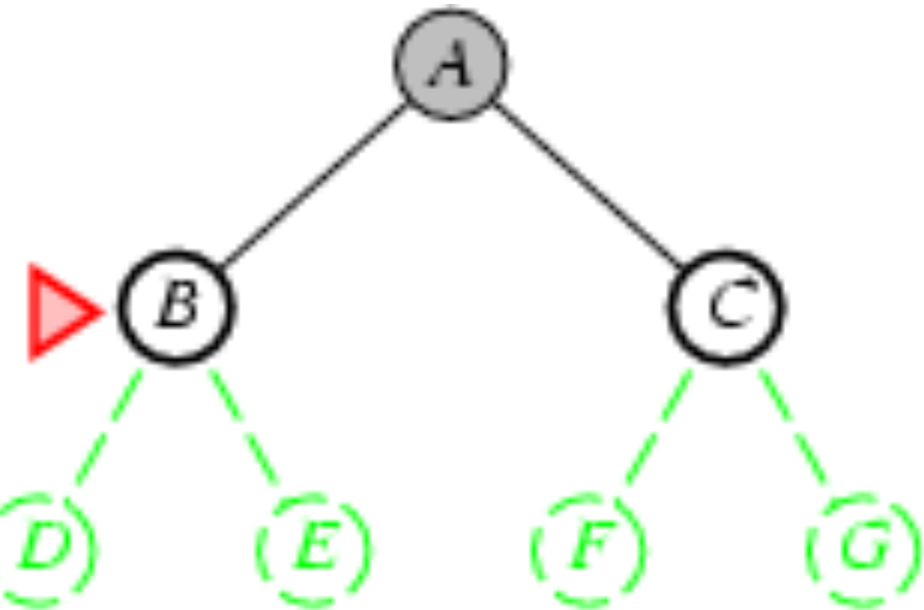
Breadth-first search

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



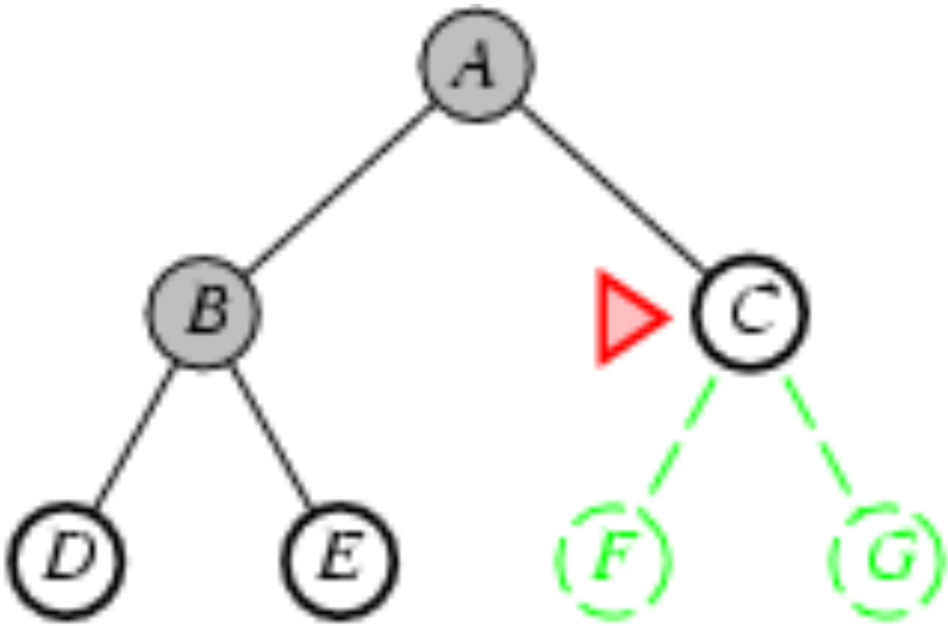
Breadth-first search

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



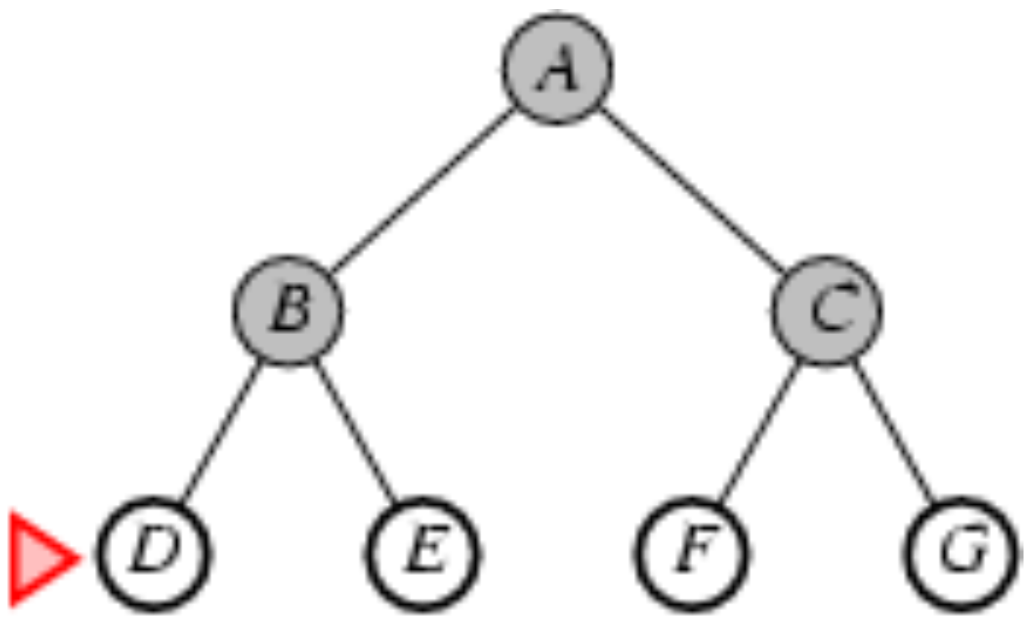
Breadth-first search

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



Breadth-first search

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



Properties of breadth-first search

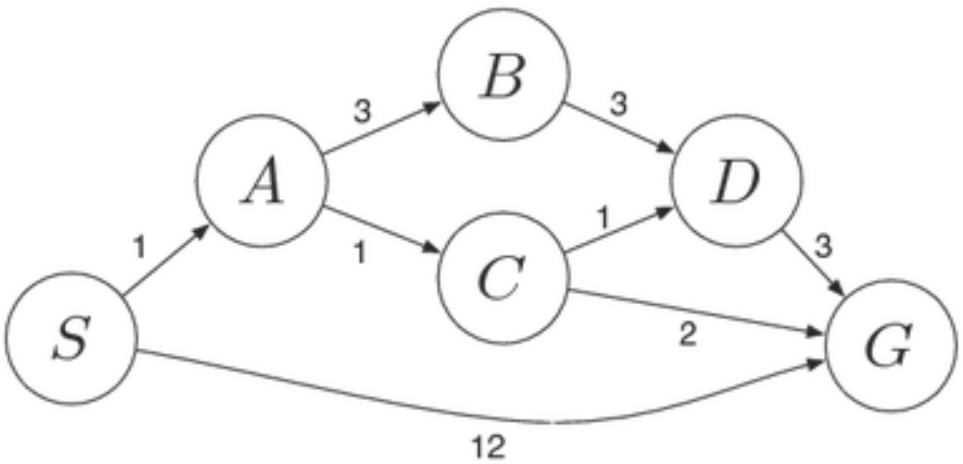
- Complete? Yes (if b is finite)
- Time? $1+b+b^2+b^3+\ldots +b^d + b(b^d-1) = O(b^{d+1})$
- Space? $O(b^{d+1})$ (keeps every node in memory)
- Optimal? Yes (if cost = 1 per step)

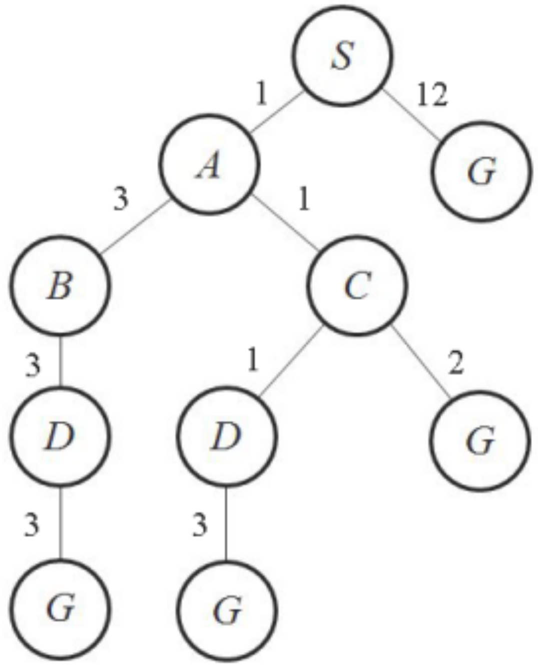
Space is the bigger problem (more than time)

Each state has b successors (branching factor)
d is the shallower depth

Uniform-cost search

- Expand least-cost unexpanded node
- Implementation:
  - *fringe* = queue ordered by path cost
- Equivalent to breadth-first if step costs all equal
- Find the solution with minimum cumulative cost





Uniform-cost search (Solution)

Initialization: { [S , 0] }

Iteration1: { [S->A , 1] , [S->G , 12] }

Iteration2: { [S->A->C , 2] , [S->A->B , 4] , [S->G , 12] }

Iteration3: { [S->A->C->D , 3] , [S->A->C->G , 4] , [S->A->B->D , 7] , [S->G , 12] }

Iteration 4: {[S->A->C->D->G , 6], [S->A->C->G , 4] , [S->A->B->D , 7] , [S->G , 12] }

Iteration 5: {[S->A->C->G , 4], [S->A->C->D->G , 6], [S->A->B->D->G, 10] , [S->G , 12]}

Solution: S->A->C->G.

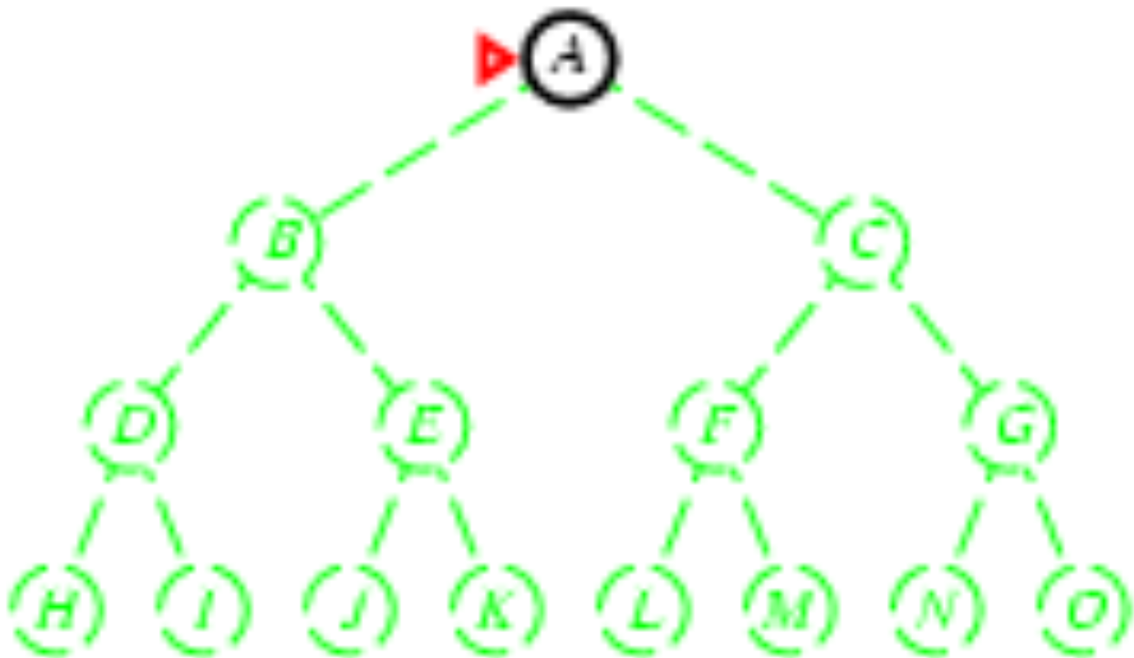
Uniform-cost search

- 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)*

If all costs are equal → $O(b^d)$

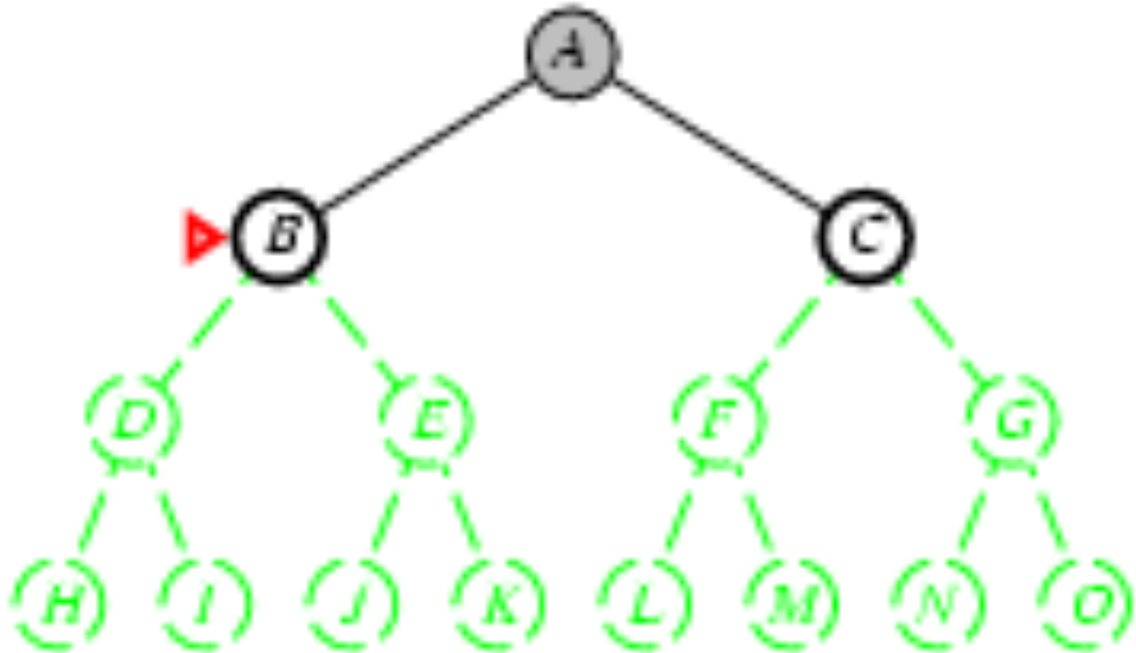
Depth-first search

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



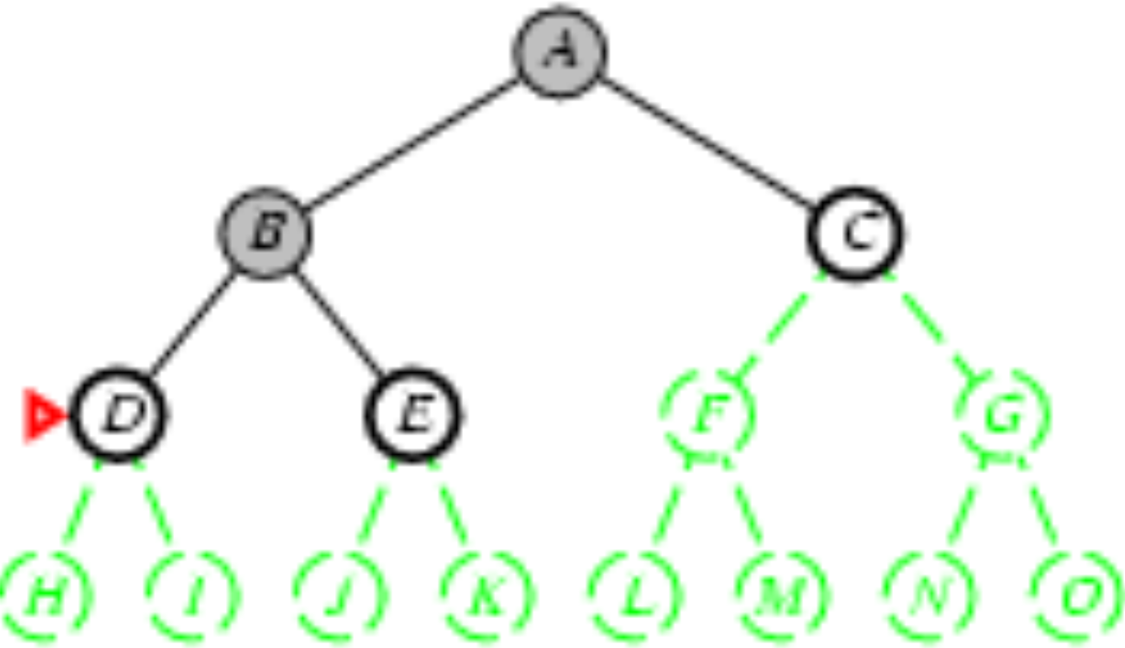
Depth-first search

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



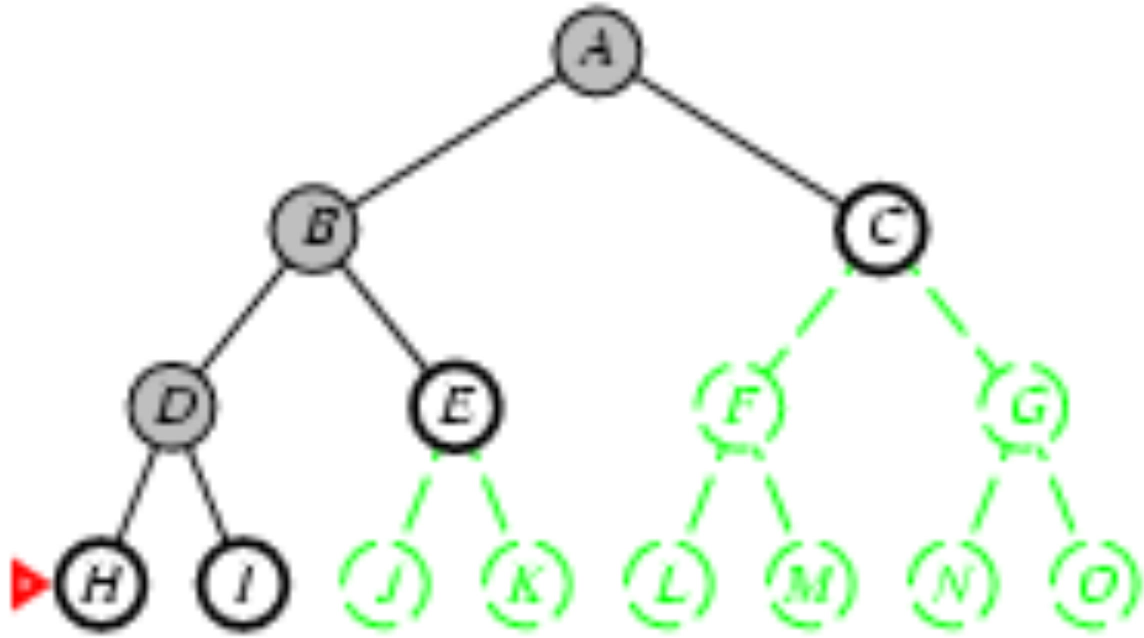
Depth-first search

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



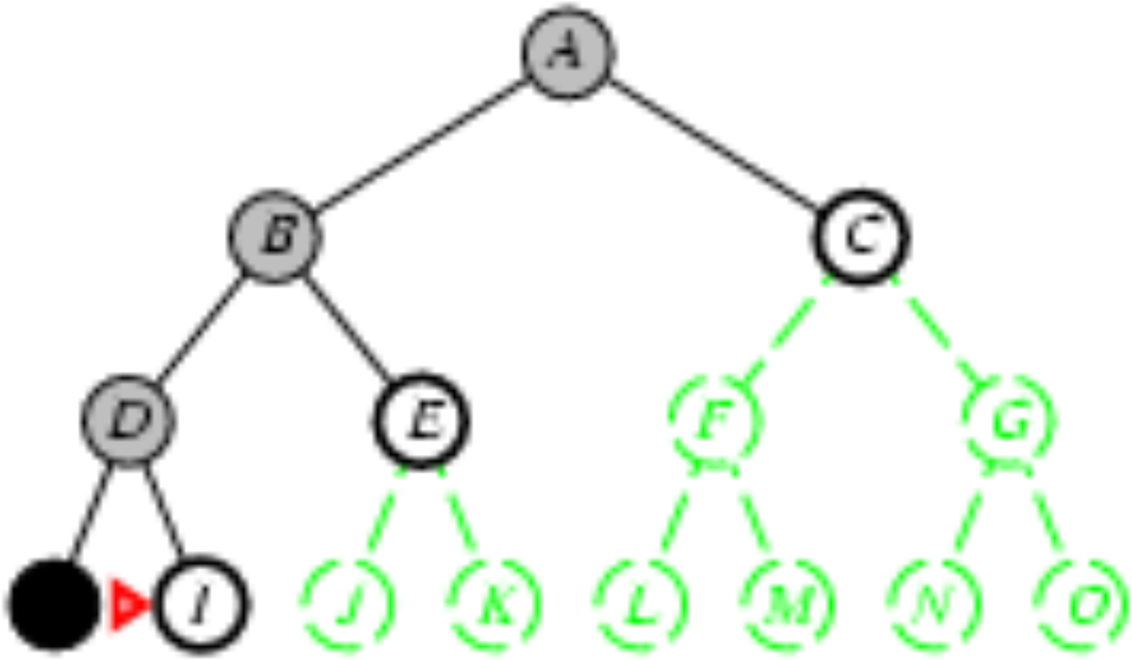
Depth-first search

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



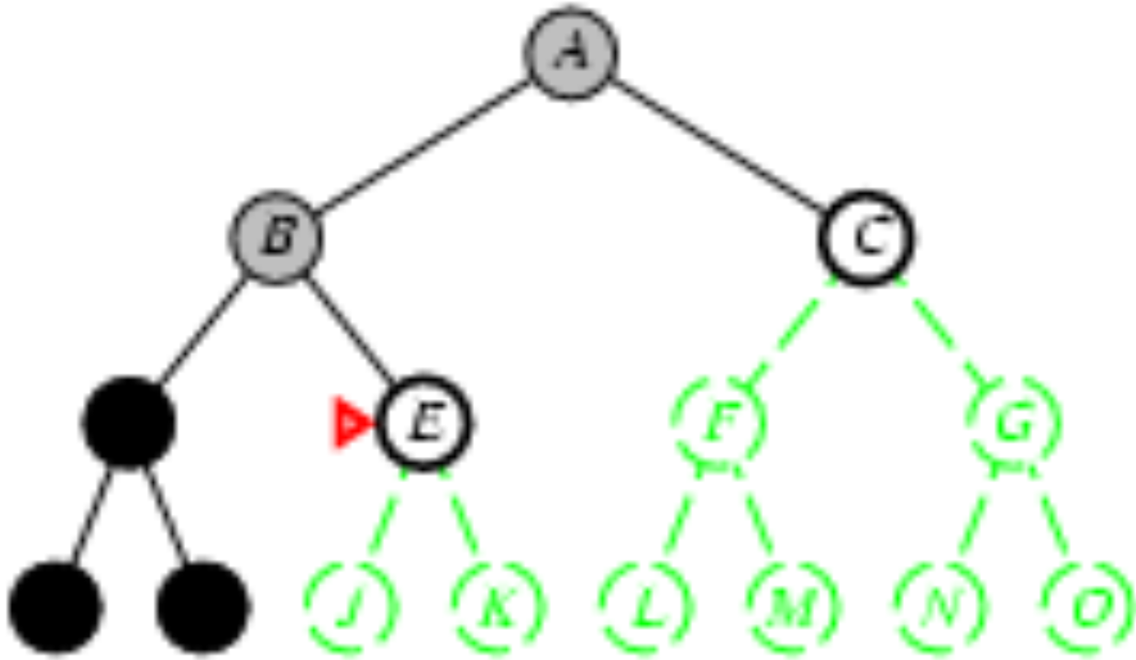
Depth-first search

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



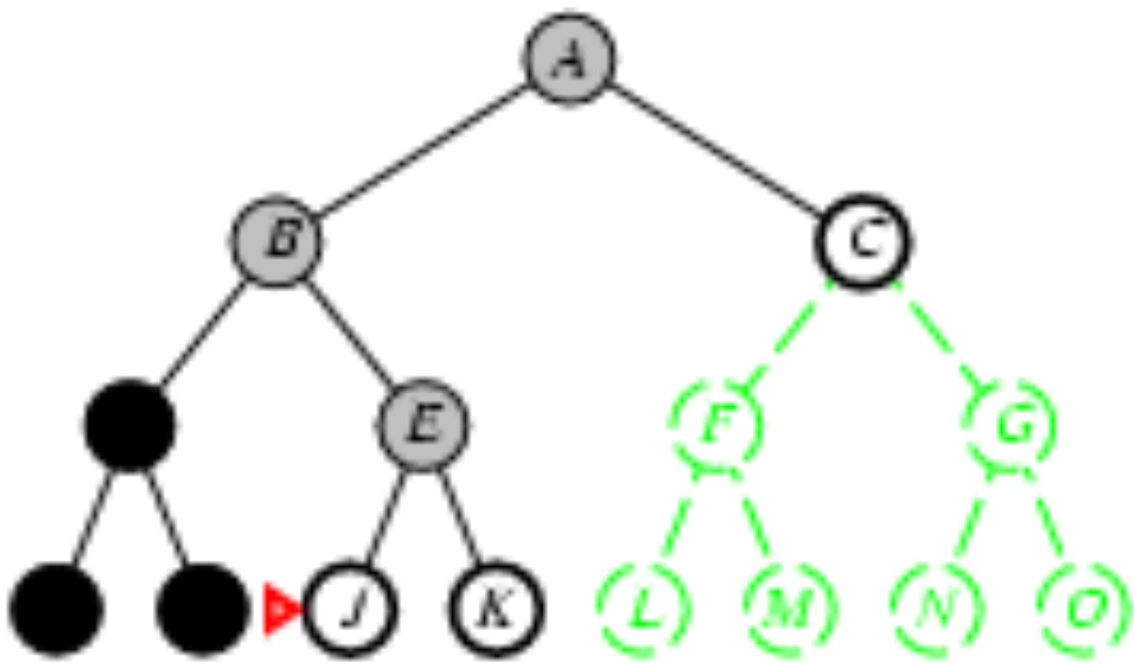
Depth-first search

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



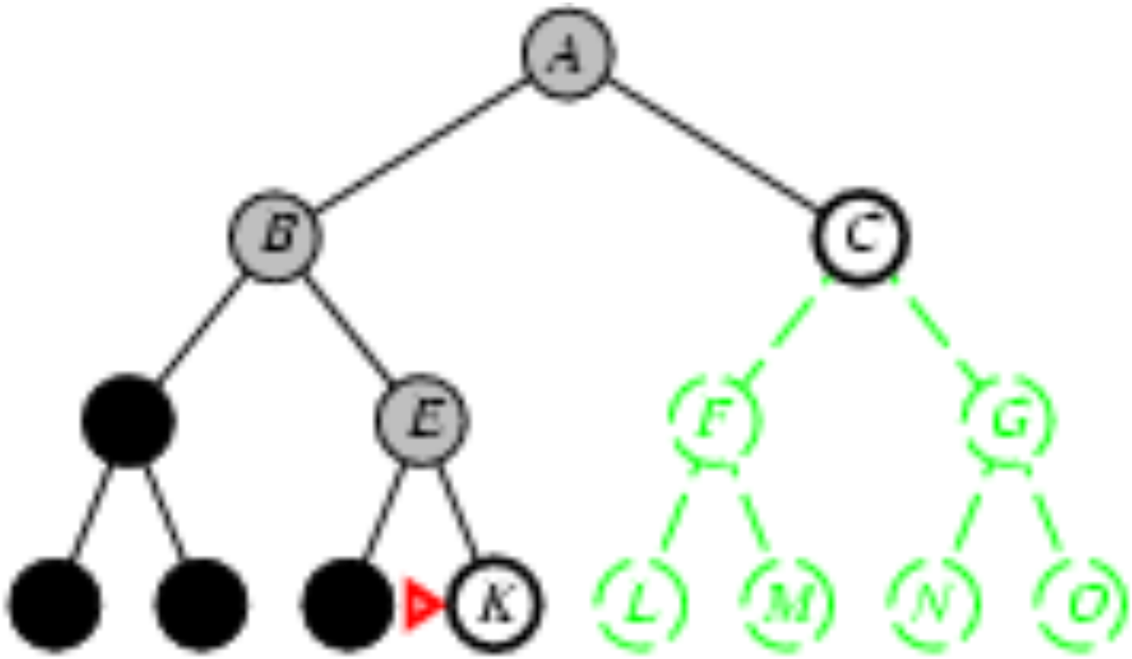
Depth-first search

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



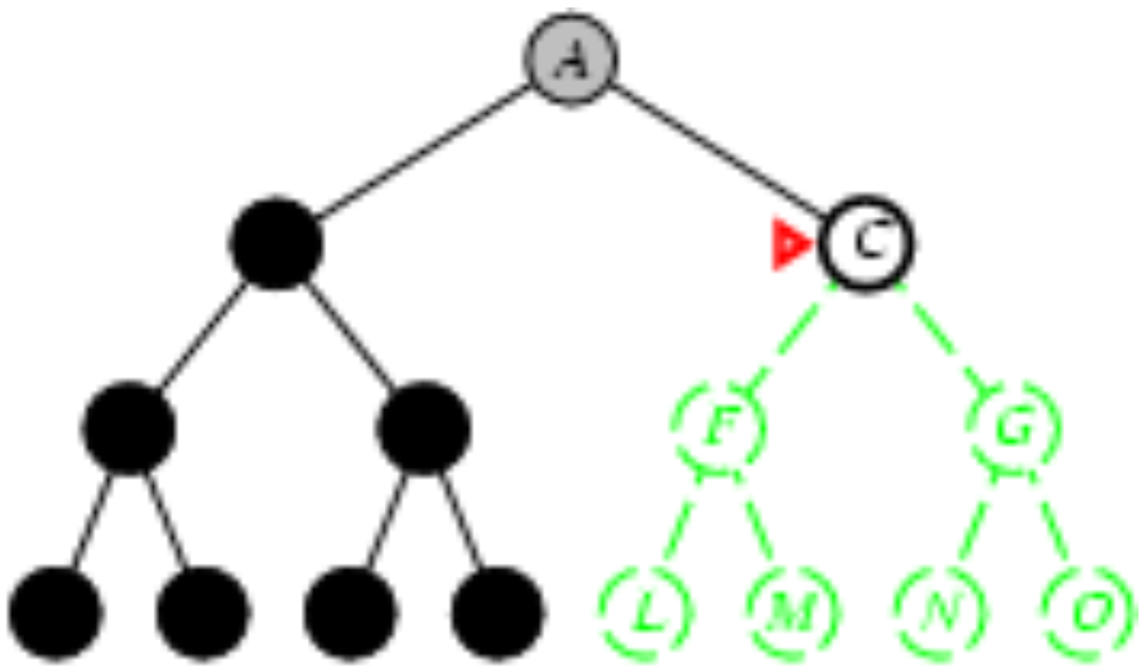
Depth-first search

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



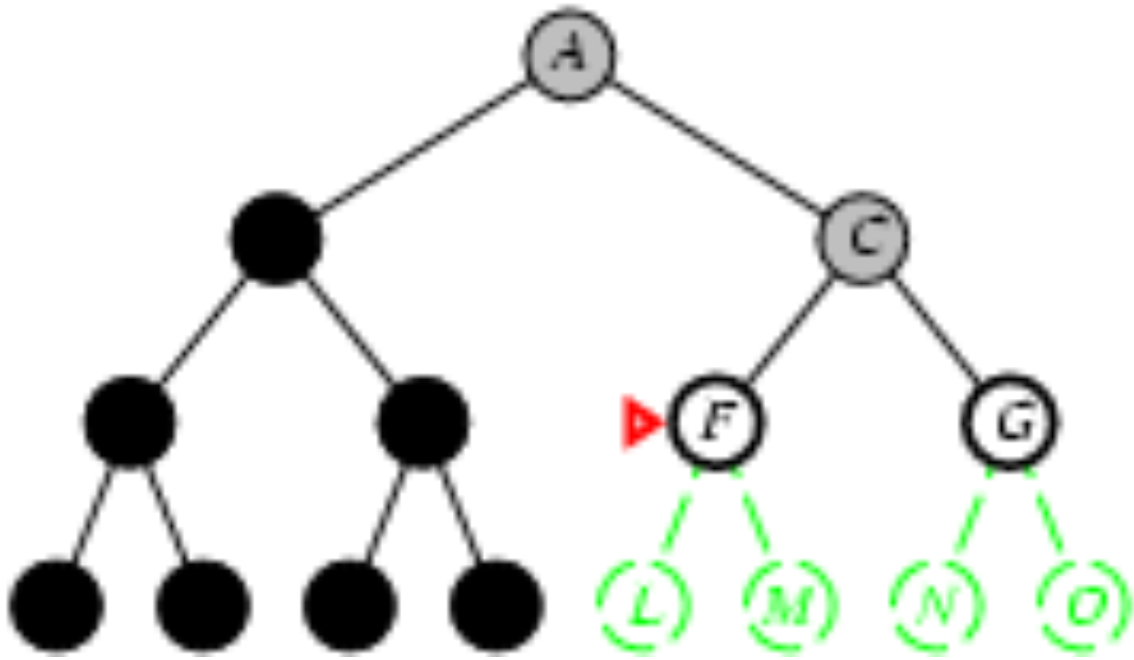
Depth-first search

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



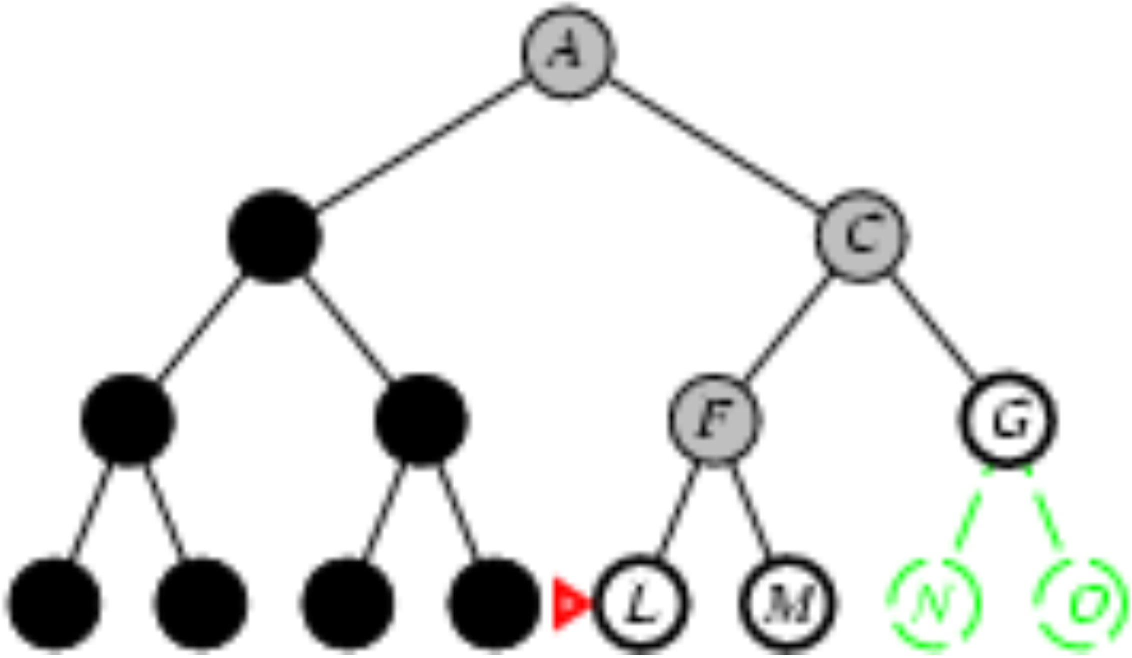
Depth-first search

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



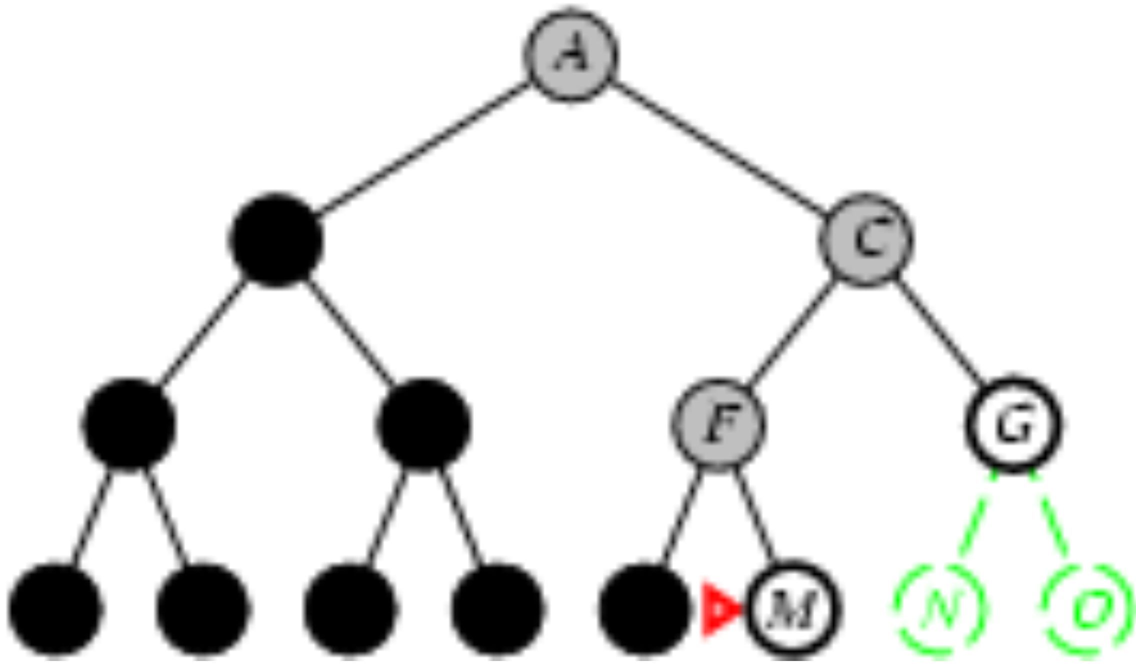
Depth-first search

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



Depth-first search

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



Properties of depth-first search

- Complete? No: fails in infinite-depth spaces, spaces with loops
  - Modify to avoid repeated states along path

    → complete in finite spaces
- Time? $O(b^m)$: terrible if m is much larger than d
  - but if solutions are dense, may be much faster than breadth-first
- Space? $O(bm)$
- Optimal? No

Depth-limited search

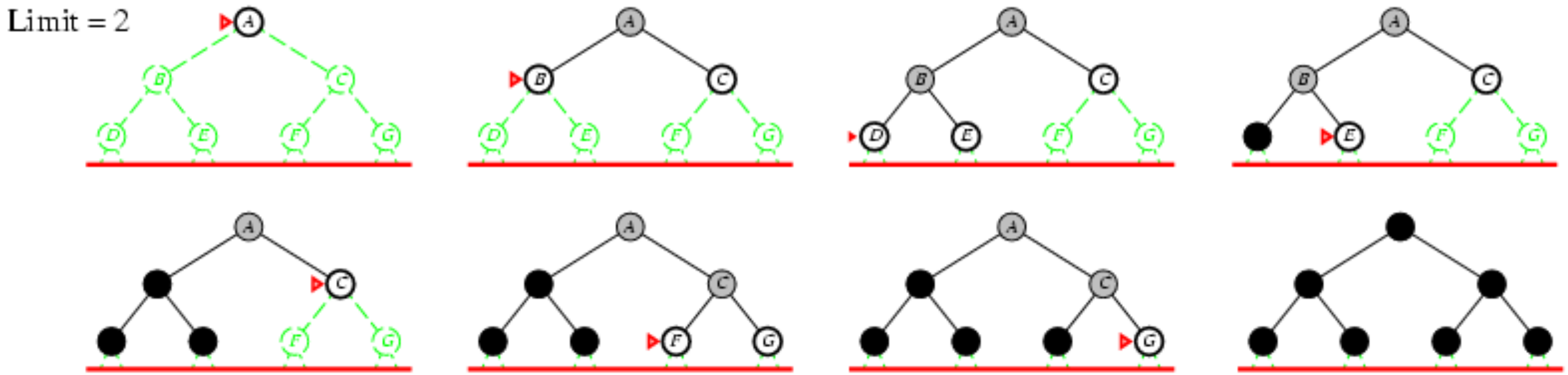
- = depth-first search with depth limit L
- i.e., nodes at depth L have no successor

Iterative deepening search *L* =0

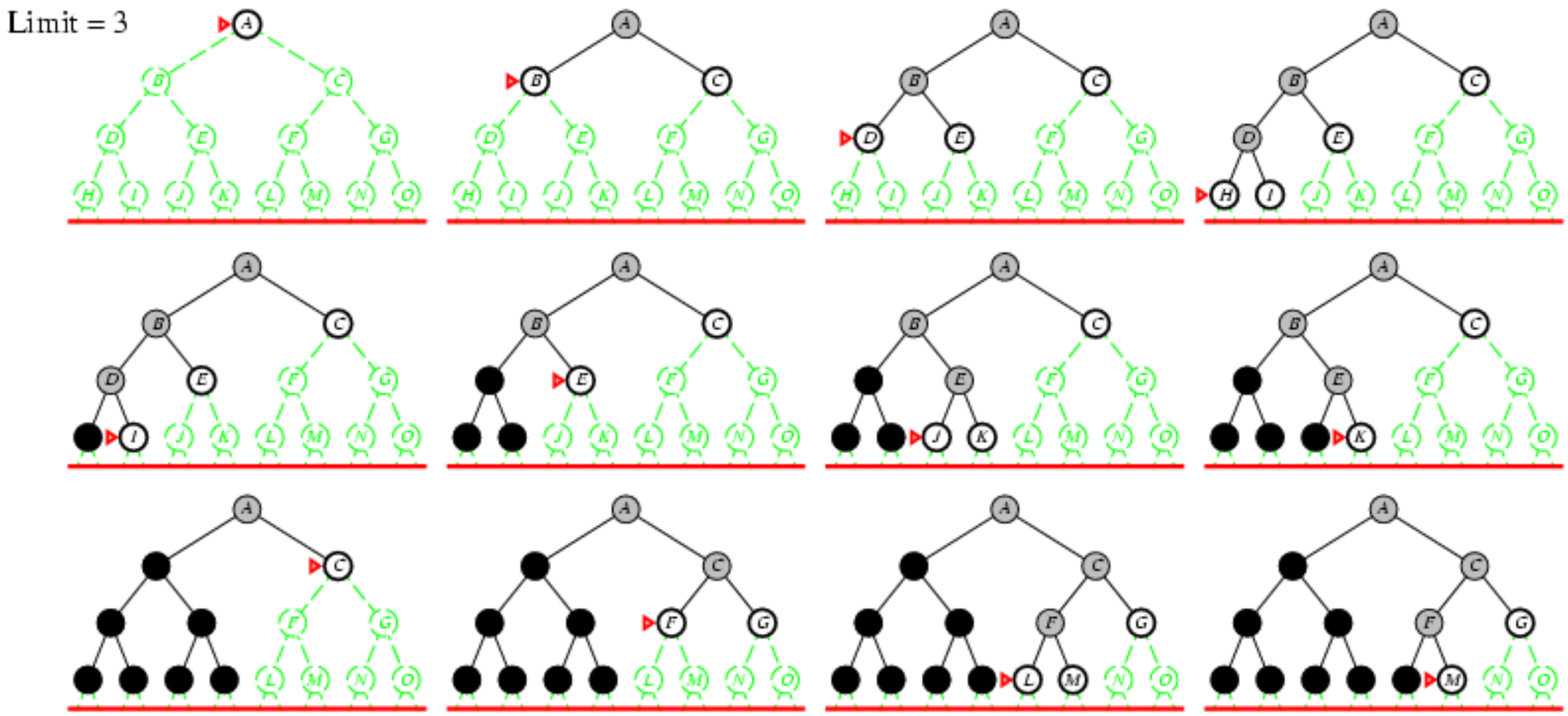
Limit = 0

Iterative deepening search L =1

Iterative deepening search L =2



Iterative deepening search L =3

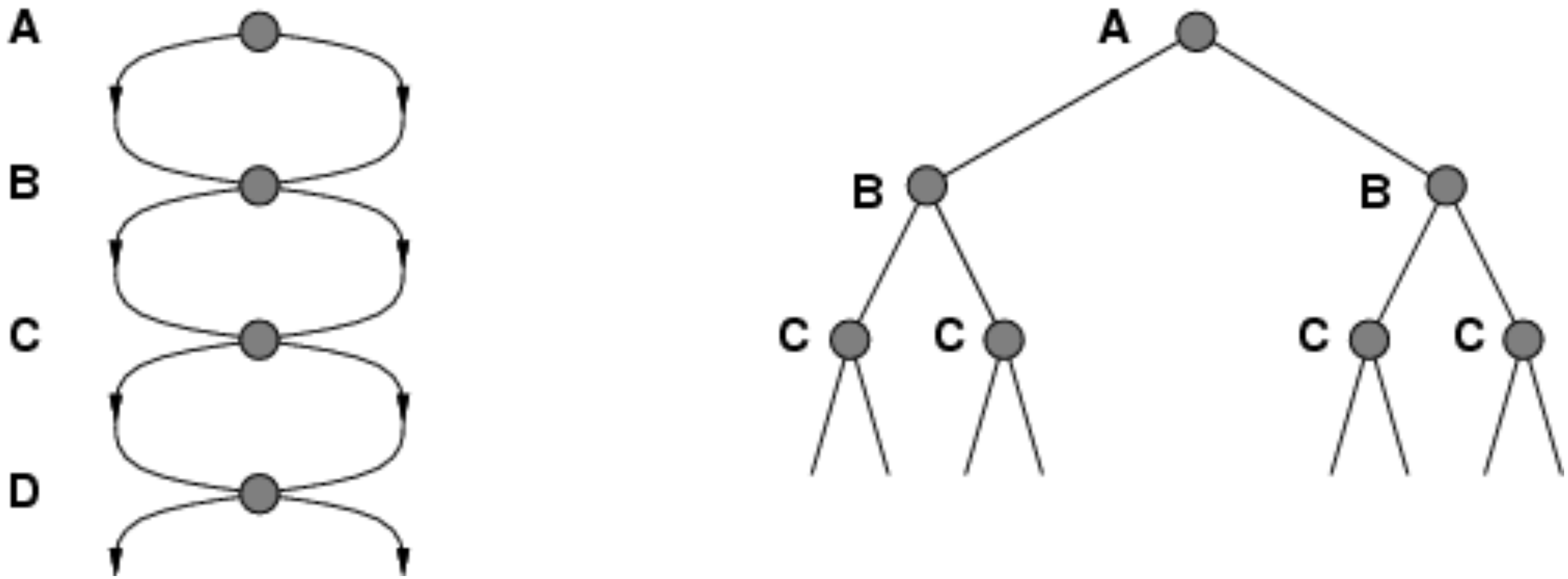


Properties of iterative deepening search

- 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

Repeated states

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



Summary of 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

Uninformed Search

Iterative deepening search

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

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

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

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

- 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%