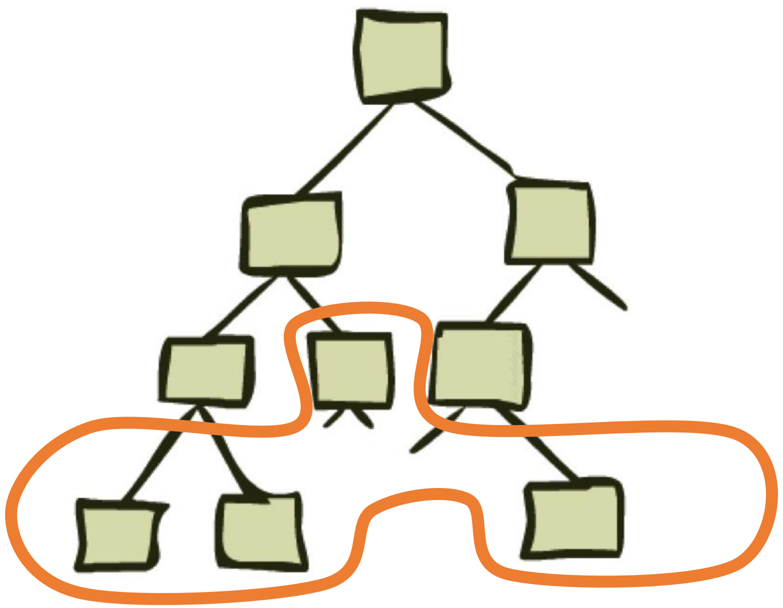
Uninformed Search-II

Depth First Search and its Variants



Previous Lecture

- Breadth First Search (BFS)
- Uniform Cost Search (UCS)

Depth-Limited Search (DLS)

- **DFS with a depth bound**
  - Searching is not permitted beyond the depth bound.

- Works well if we know what is the depth of the solution.

- If the solution is beneath the depth bound, the search cannot find the goal (hence this search algorithm is incomplete).

Depth-Limited Search (DLS)

- **Main idea***:*
  - *Expand node at the deepest level, but limit depth to D.*
- **Implementation**:
  - *Enqueue nodes in LIFO (last-in, first-out) order. But limit depth to D*

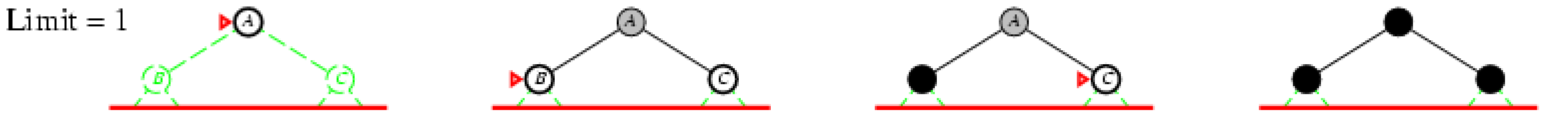
- *Complete?*
  - No
  - Yes: if there is a goal state at a depth less than D
- Optimal?
  - No
- Time Complexity:
  - $O(b^D)$, where D is the cutoff.
- Space Complexity:
  - $O(b^D)$, where D is the cutoff.

Iterative Deepening Search

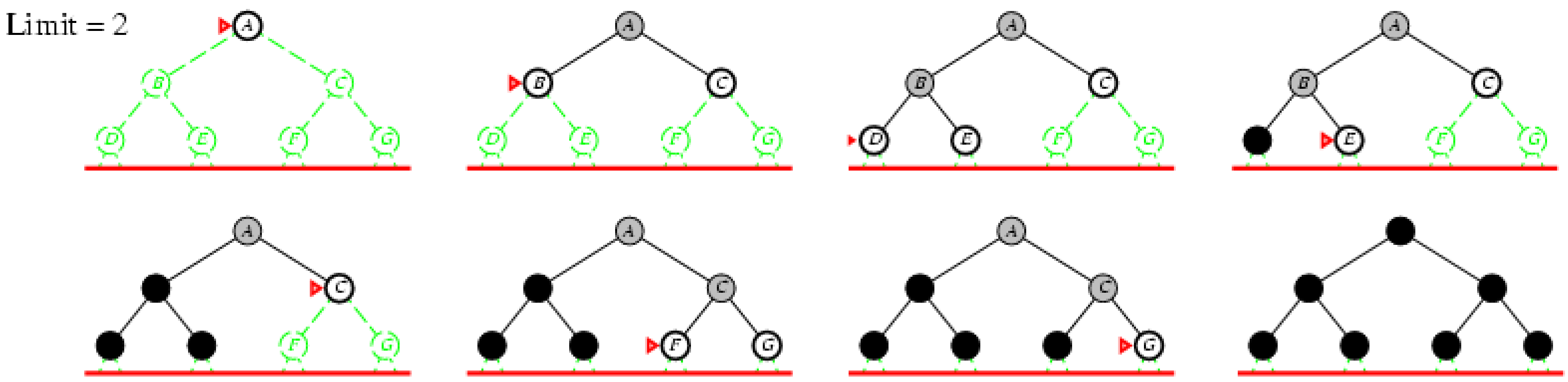
- To avoid the infinite depth problem of DFS:
  - Only search until depth L
  - i.e, don’t expand nodes beyond depth L
  - Depth-Limited Search
- What if solution is deeper than L?
  - Increase depth iteratively
  - Iterative Deepening Search
- IDS
  - Inherits the memory advantage of depth-first search
  - Has the completeness property of breadth-first search

Iterative deepening search *l* =0

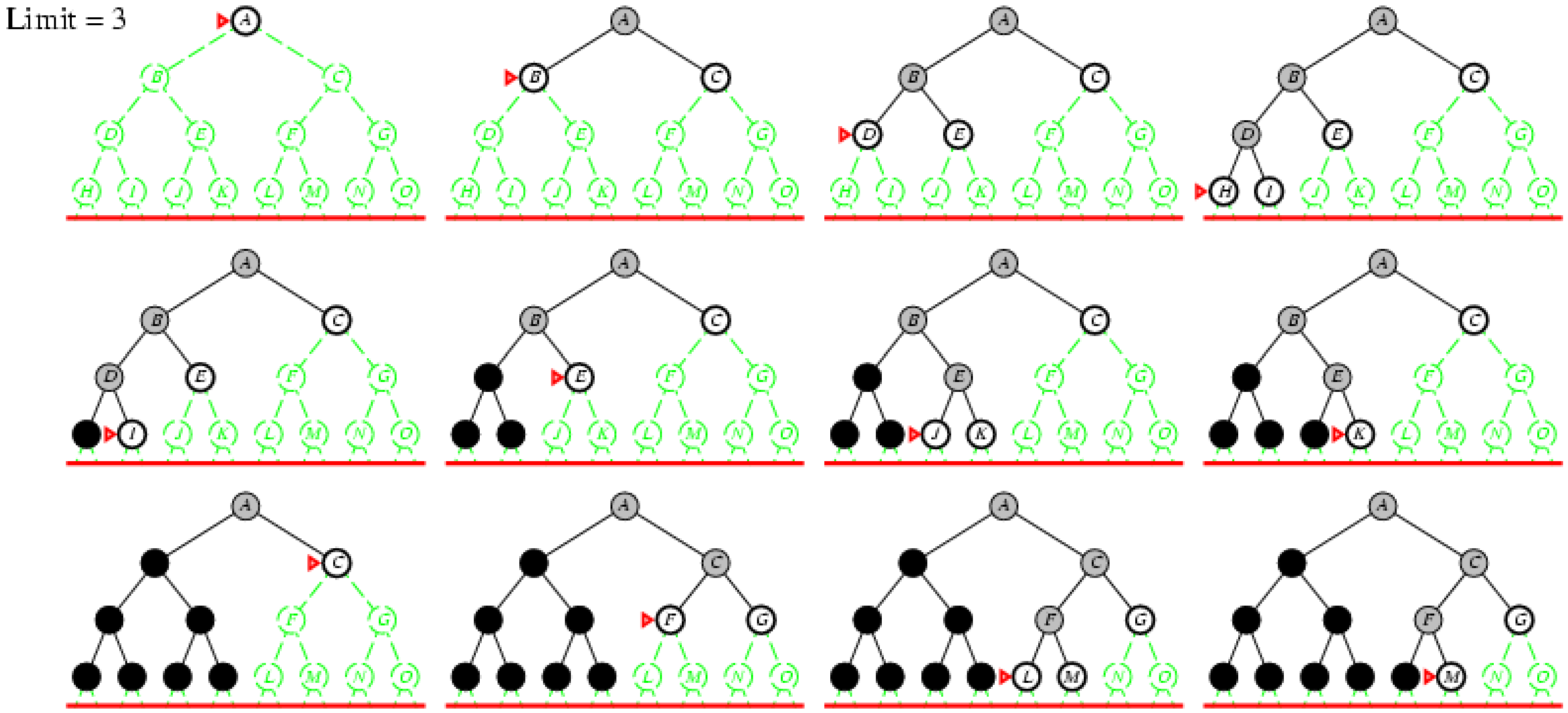
Iterative deepening search l =1



Iterative deepening search l =2

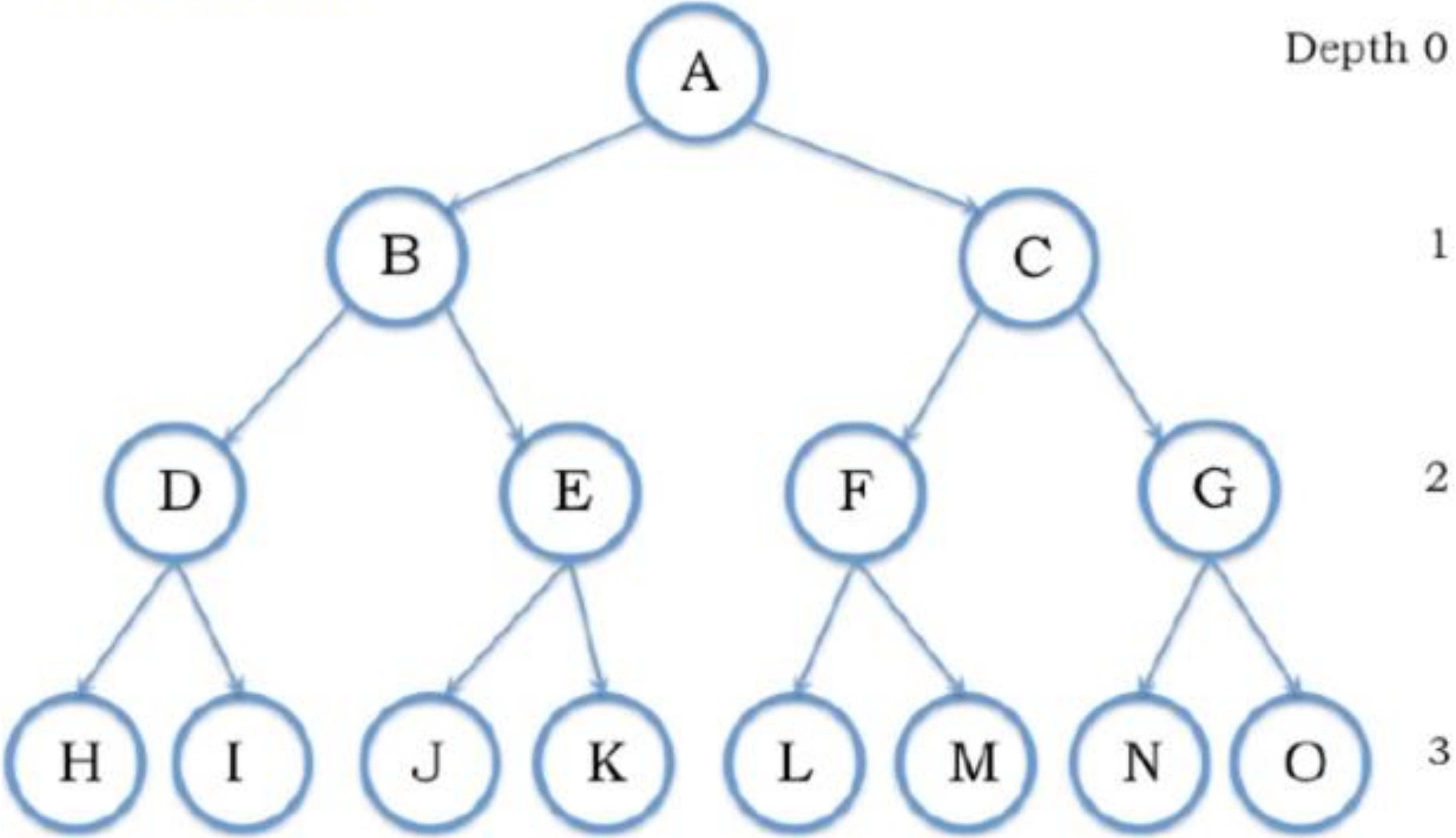


Iterative deepening search *l* =3

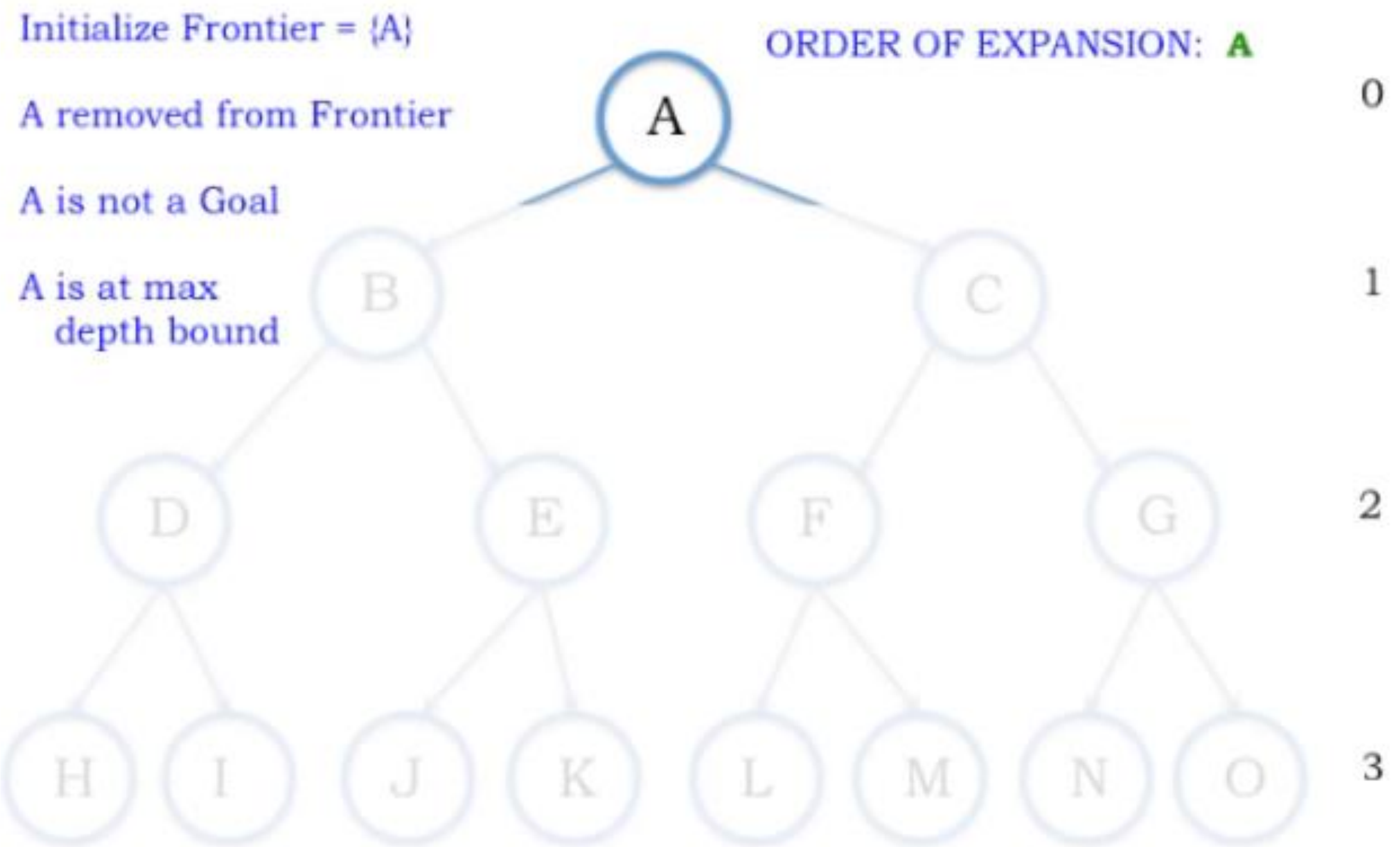


Operators (or actions) are not reversible
A is the start state

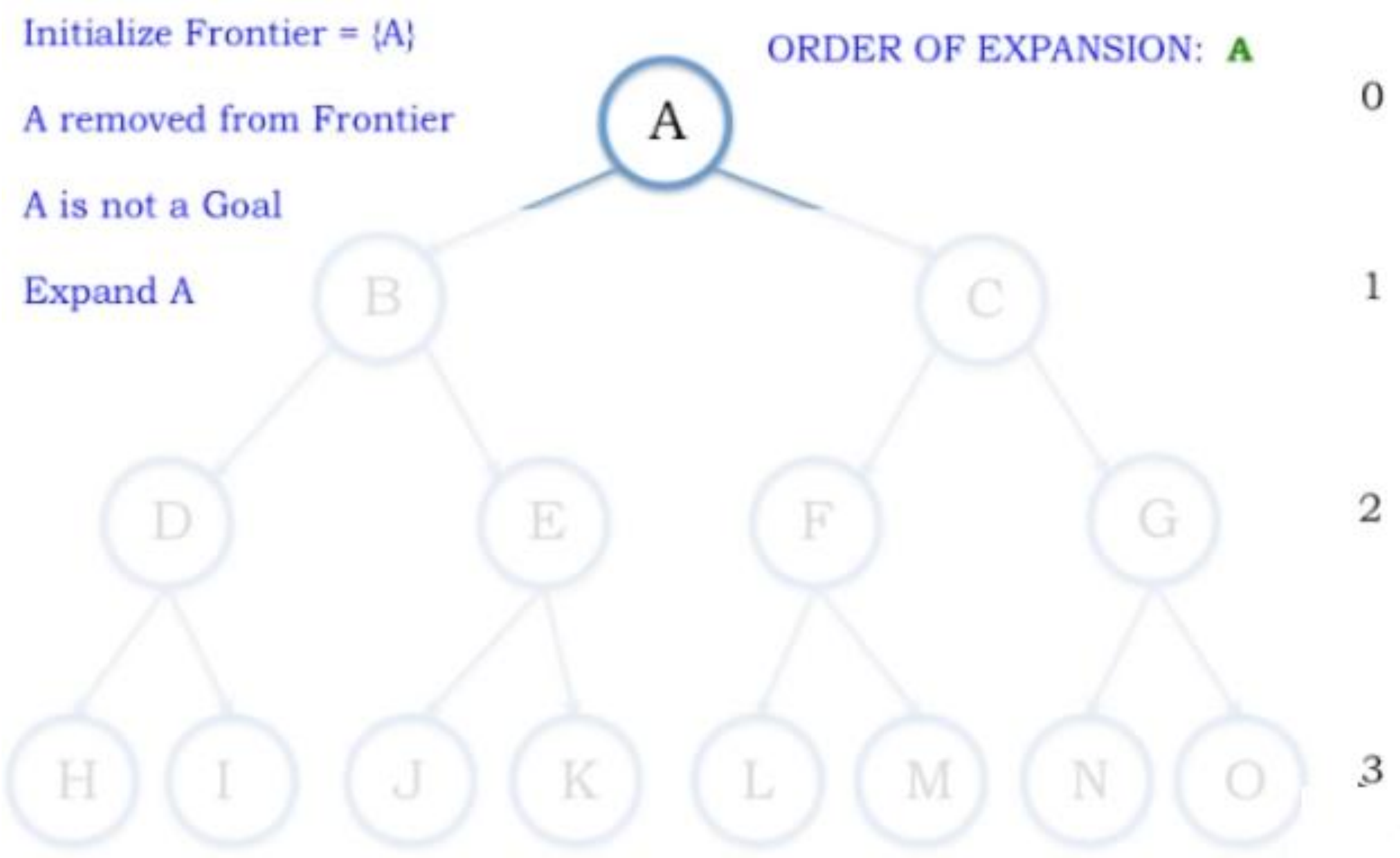
Repeated Depth Bounded searches



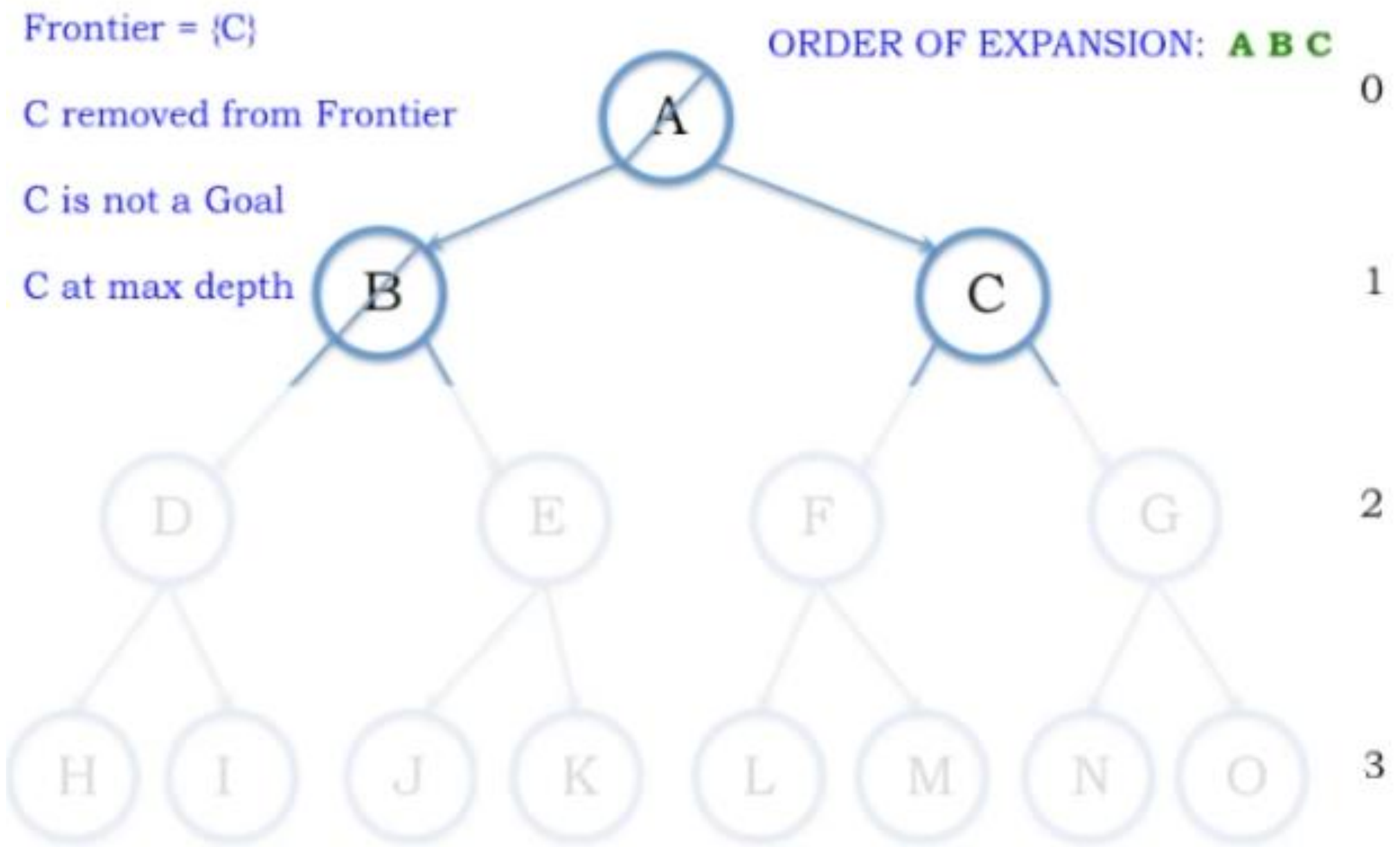
Depth Bound = 0



Depth Bound = 1



Depth Bound = 1



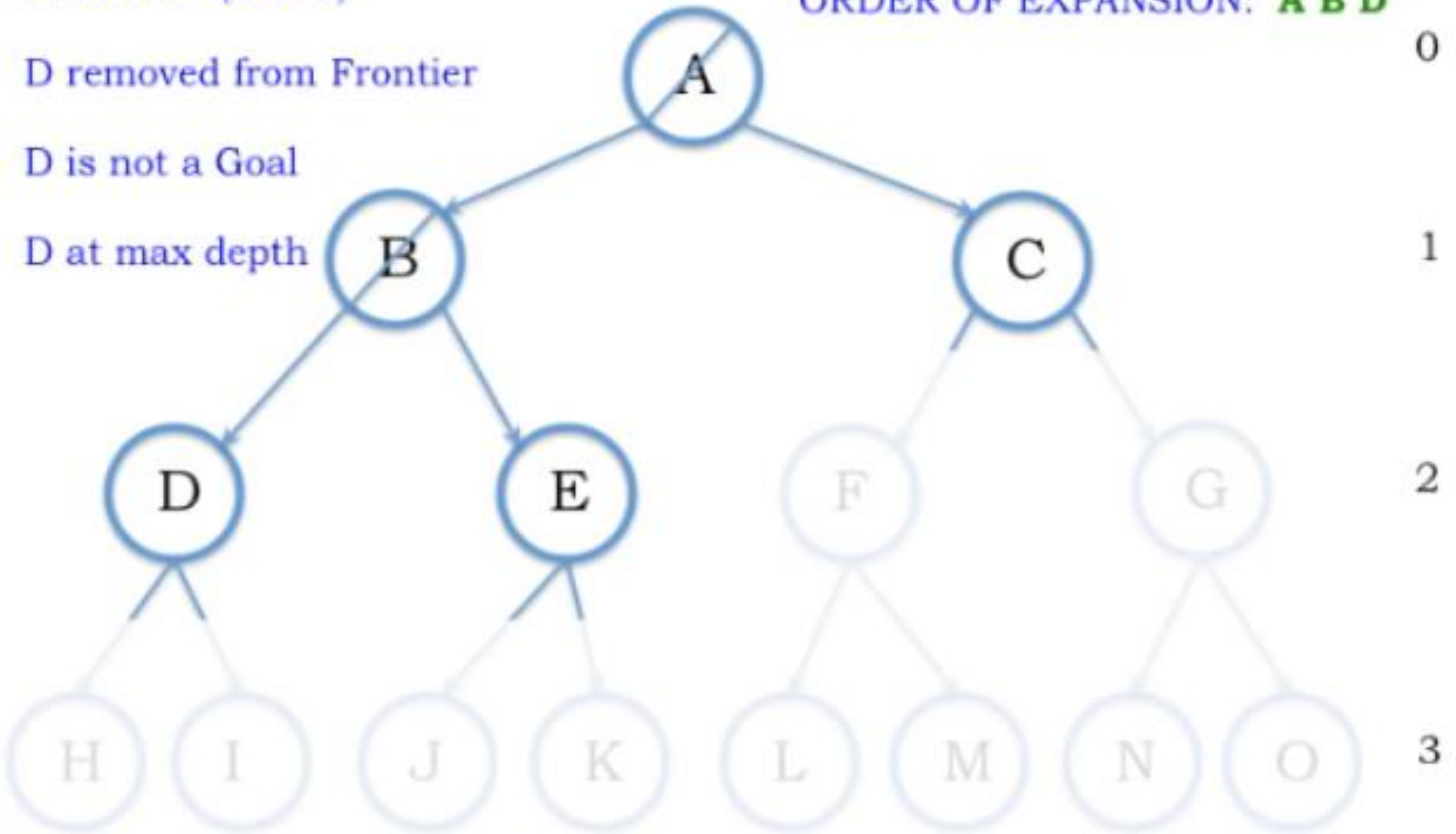
Frontier = {D E C}

D removed from Frontier

D is not a Goal

D at max depth

ORDER OF EXPANSION: **A B D**



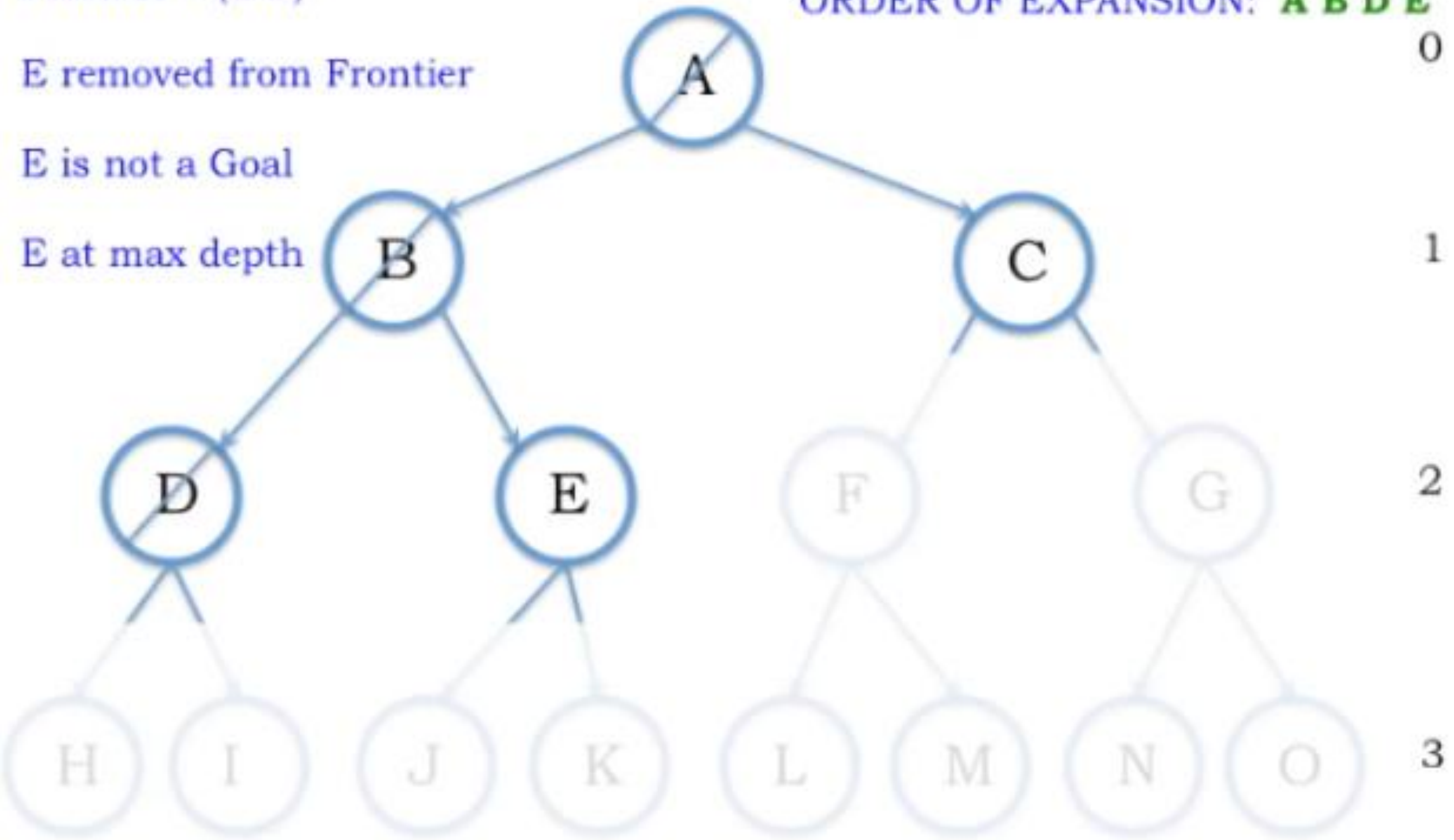
Frontier = {E C}

E removed from Frontier

E is not a Goal

E at max depth

ORDER OF EXPANSION: **A B D E**



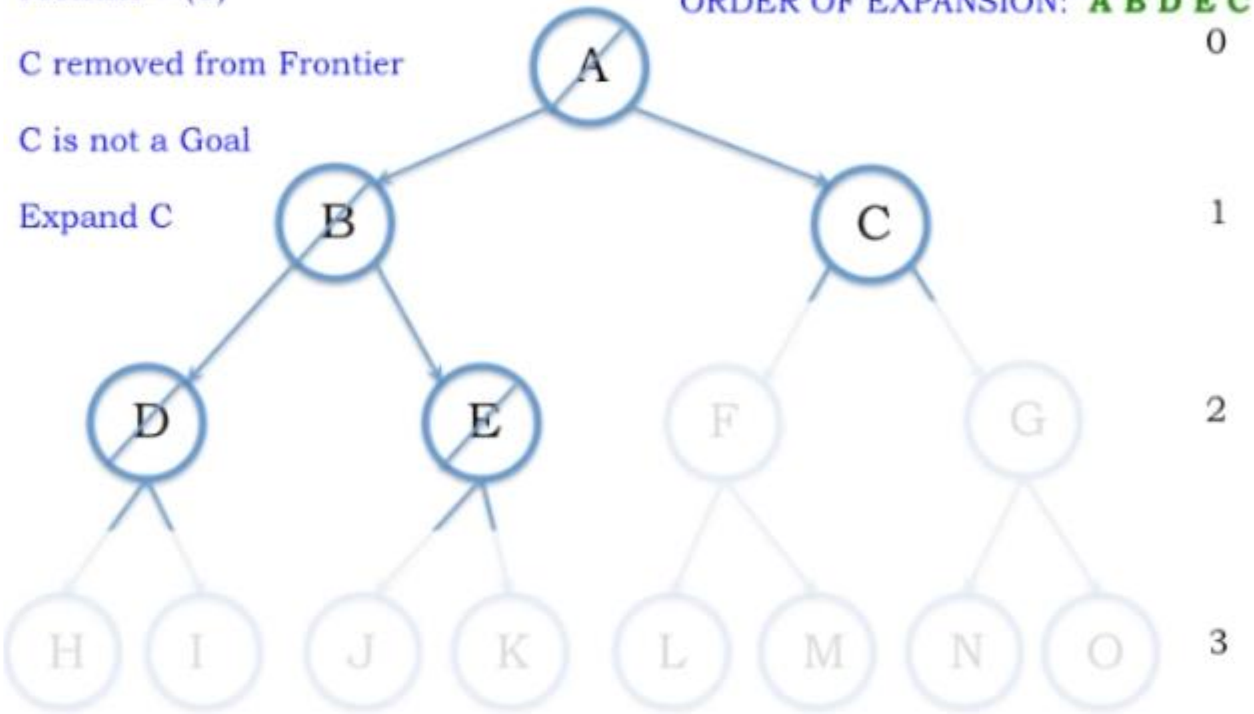
Frontier = {C}

C removed from Frontier

C is not a Goal

Expand C

ORDER OF EXPANSION: **A B D E C**



Frontier = {F G}

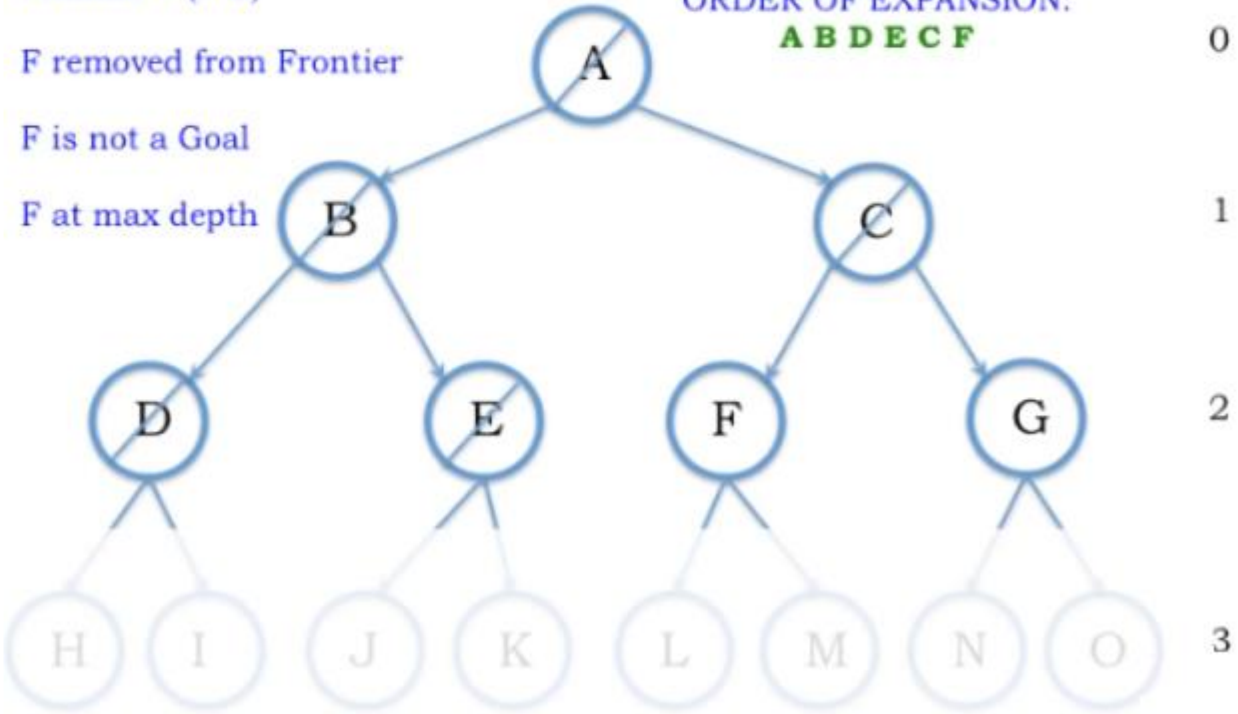
F removed from Frontier

F is not a Goal

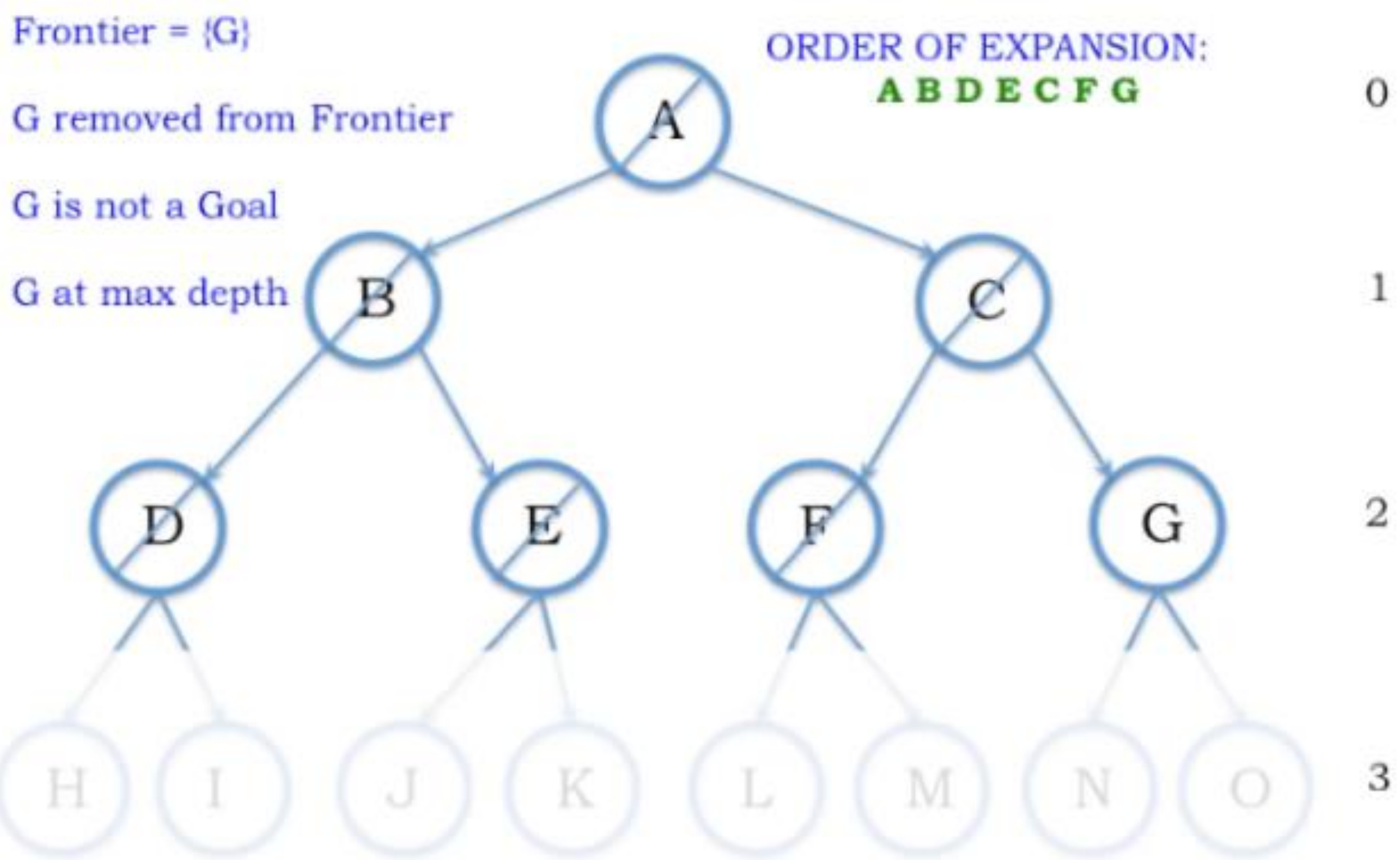
F at max depth

ORDER OF EXPANSION:

A B D E C F

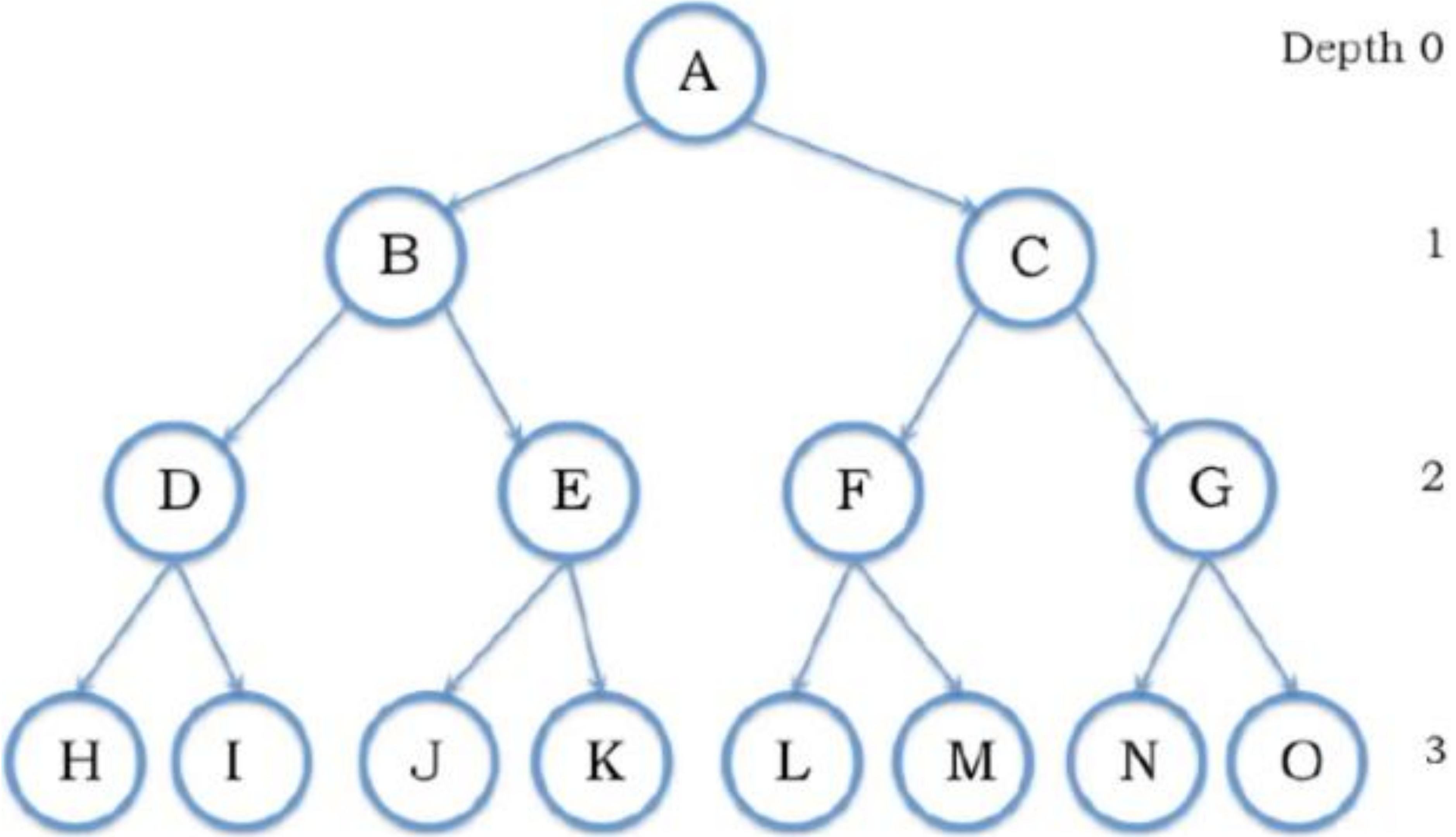


Depth 2


Frontier = {G}
G removed from Frontier
G is not a Goal
G at max depth
ORDER OF EXPANSION:
A B D E C F G
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
0
1
2
3

Over ALL the iterations, from depth bound 0 to 3, the order in which nodes removed from the frontier is:

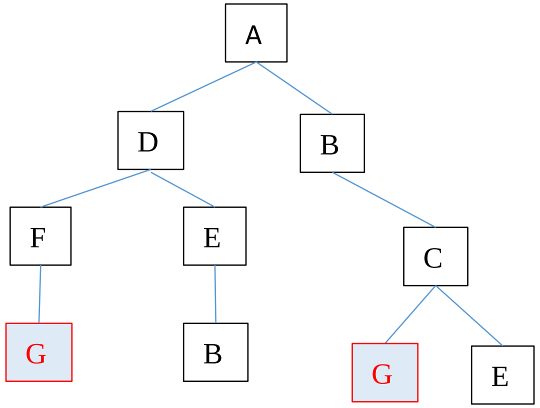
A ABC ABDECFG ABDHIEJKCFLMGNO



Properties of Iterative Deepening Search

- Complete? Yes (in finite spaces)
- Time? $O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, (if step cost = 1 i.e. identical step cost)

Example



Do it on notebook

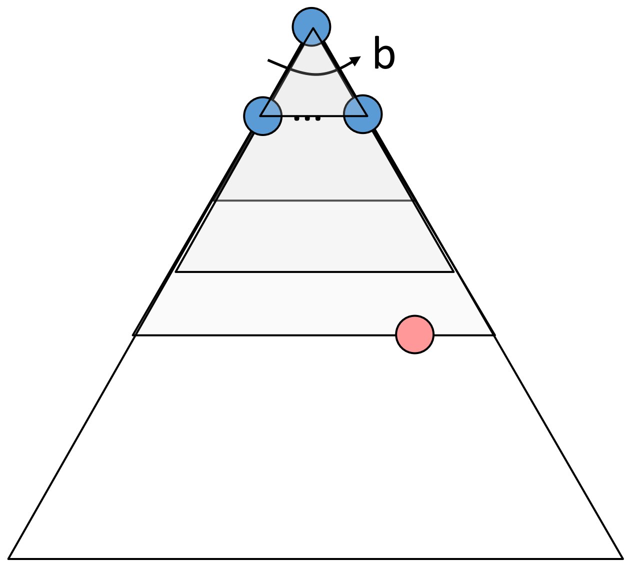
- Start Node: A
- Goal Node: G

Step Frontier Expand[*] Explored: a set of nodes

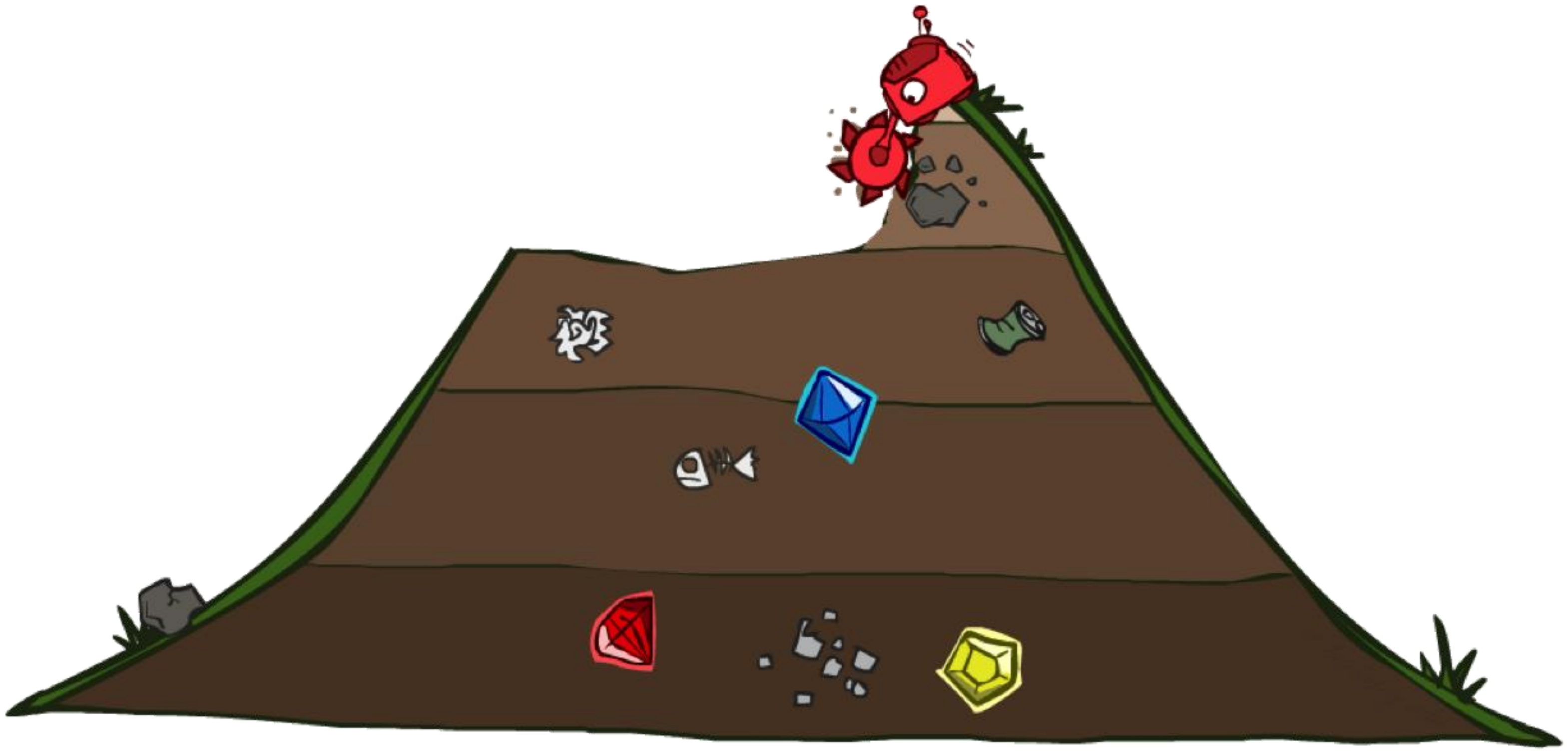
Iterative Deepening Search

- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..

- Isn't that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!



Uniform Cost Search



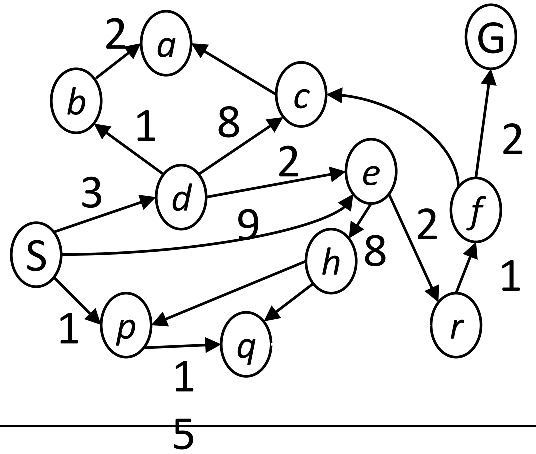
Uniform Cost Search

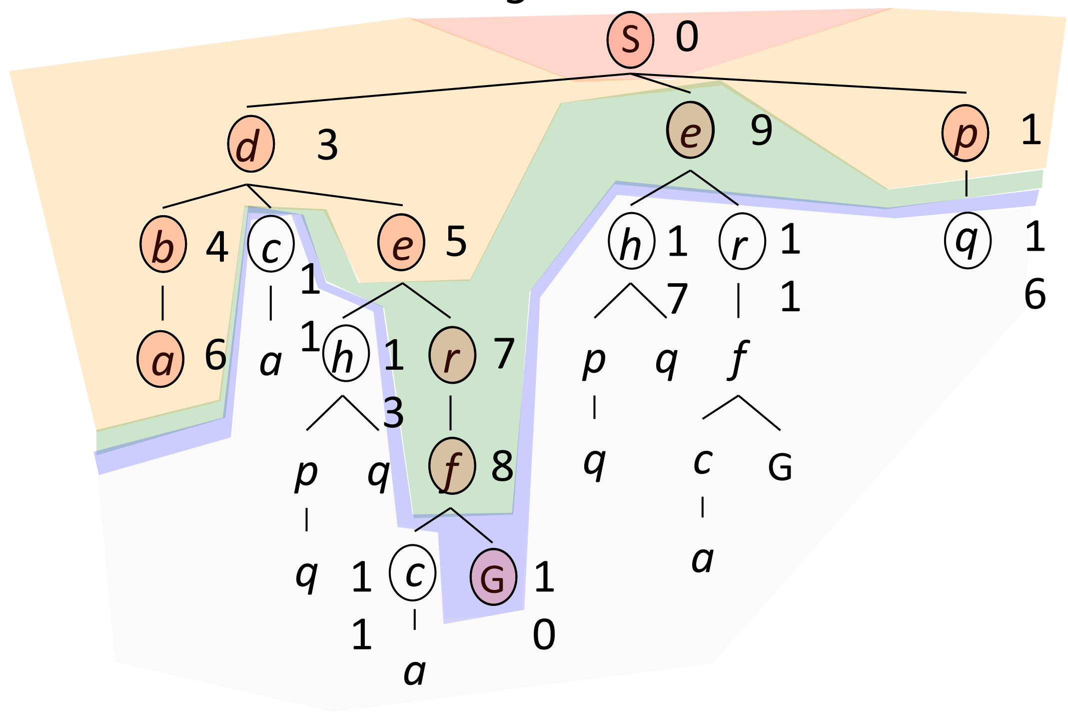
- Breadth-First Search find the shallowest goal state, but this may not always be the least-cost solution.
- Uniform-Cost Search modifies the Breadth-First Search strategy by always expanding the lowest path cost g(n) node on the fringe.
- *Frontier* is a priority queue, i.e., new successors are merged into the queue sorted by g(n).
  - Can remove successors already on queue w/higher g(n).
  - Saves memory, costs time; another space-time trade-off.

Uniform Cost Search

Strategy: expand a cheapest node first:

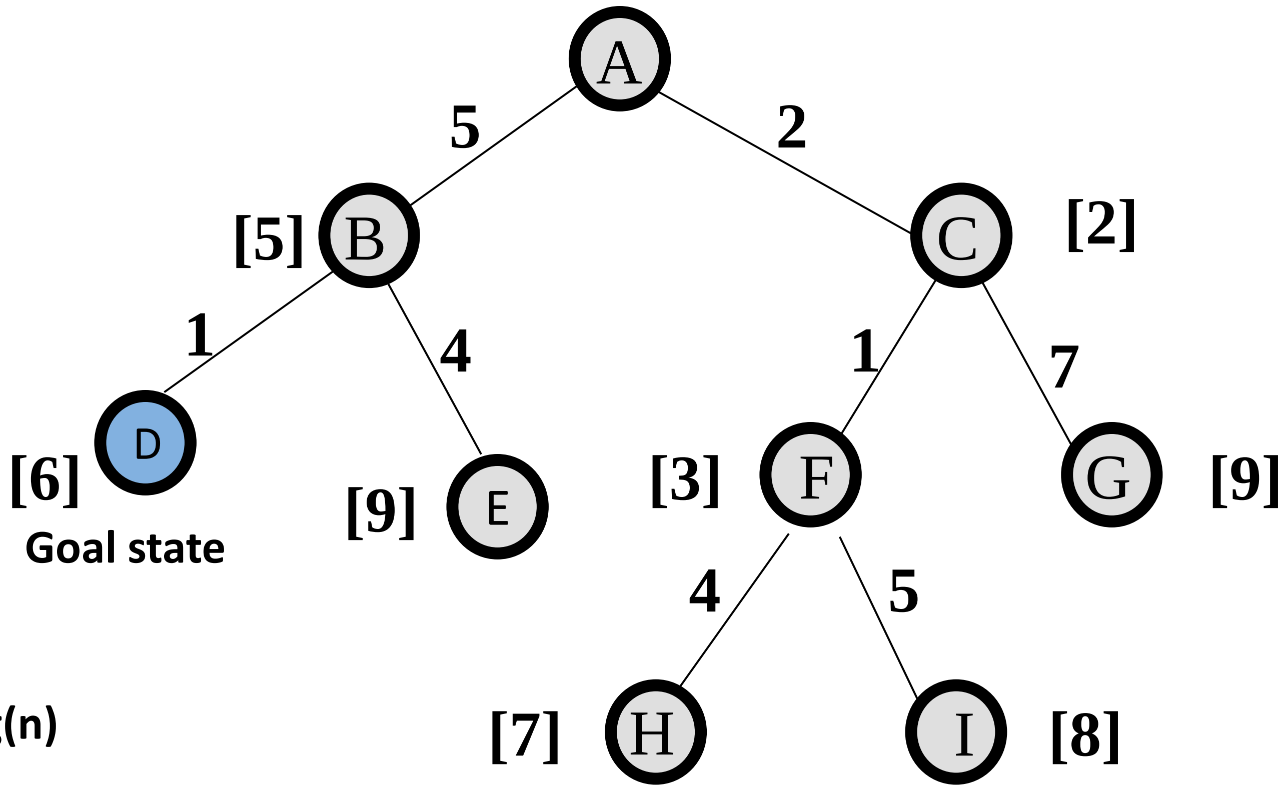
Fringe is a priority queue (priority: cumulative cost)





Cost contours

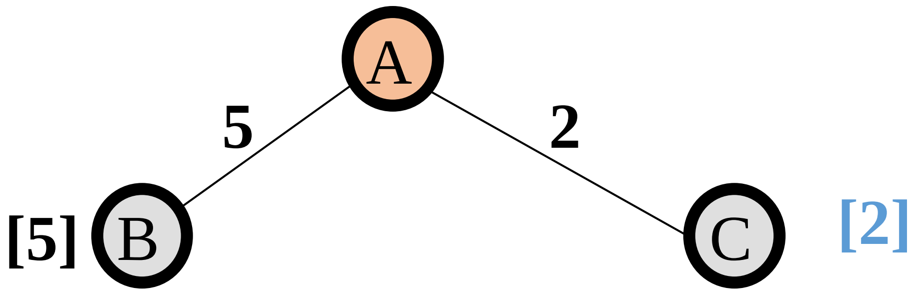
Uniform Cost Search (UCS)



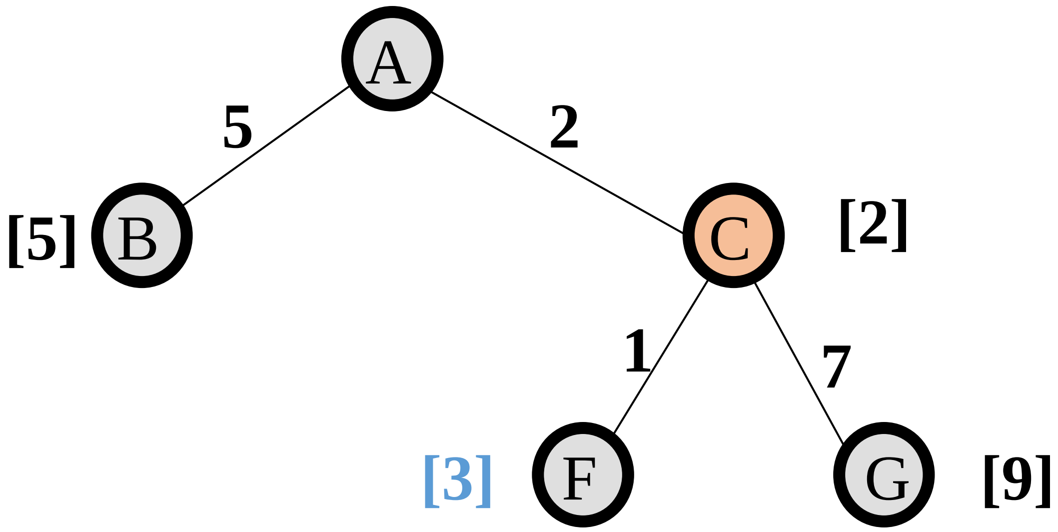
[x] = g(n)

path cost of node n

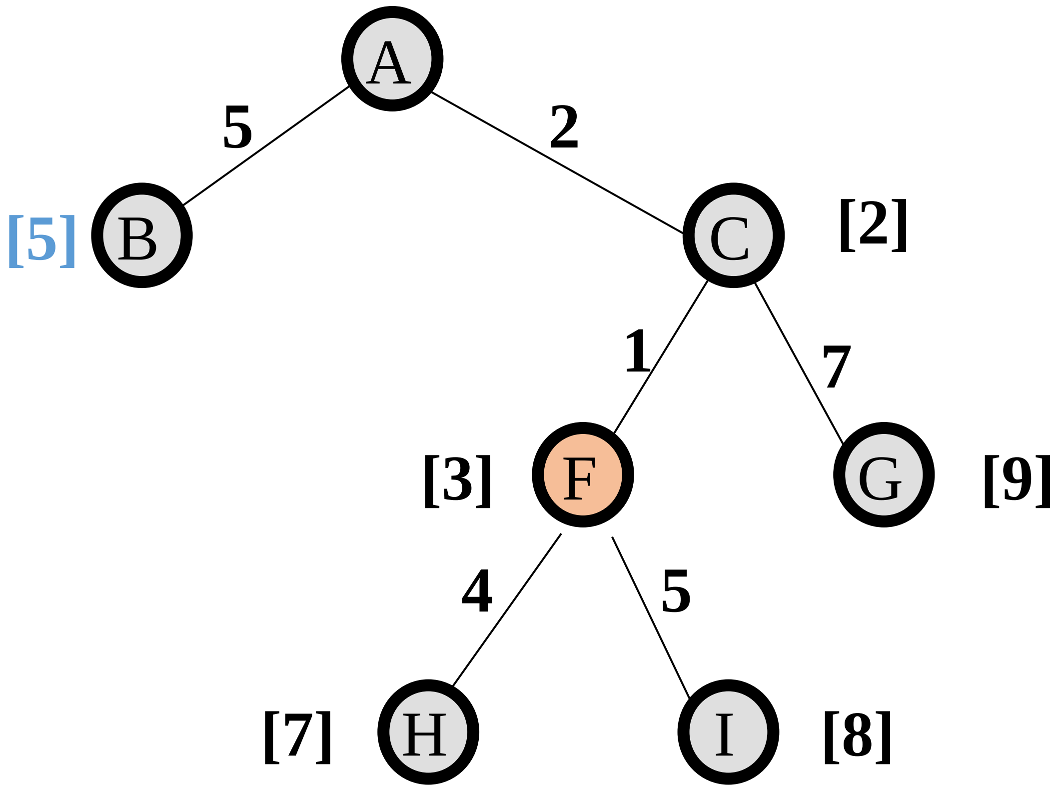
Uniform Cost Search (UCS)



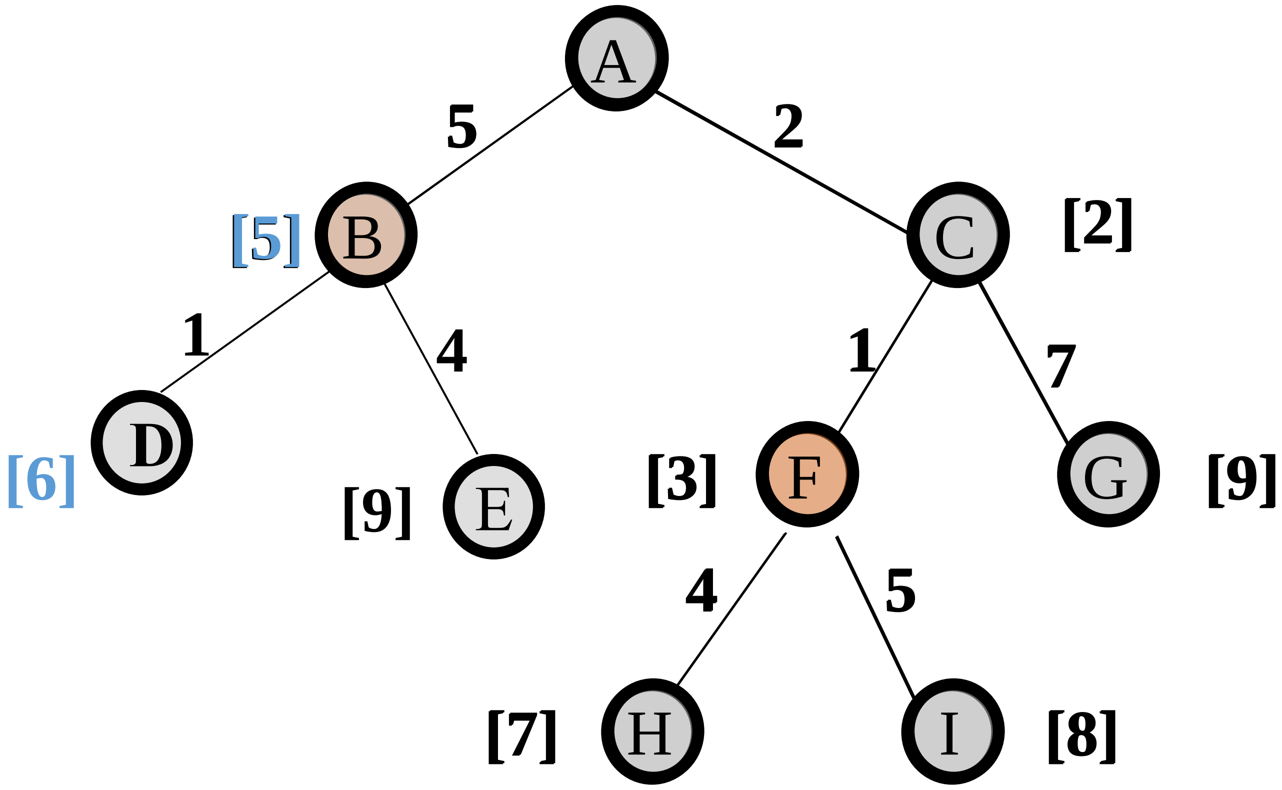
Uniform Cost Search (UCS)



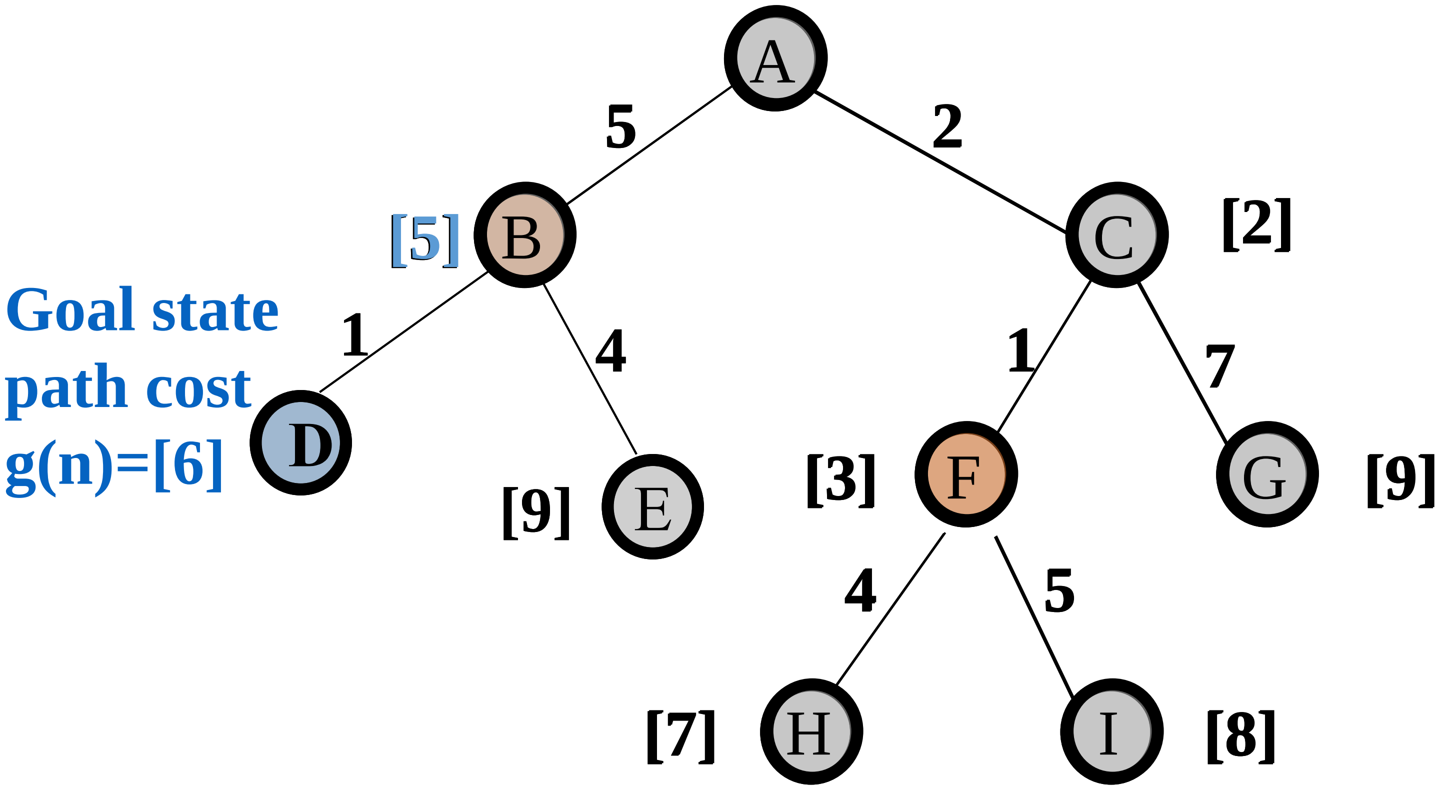
Uniform Cost Search (UCS)



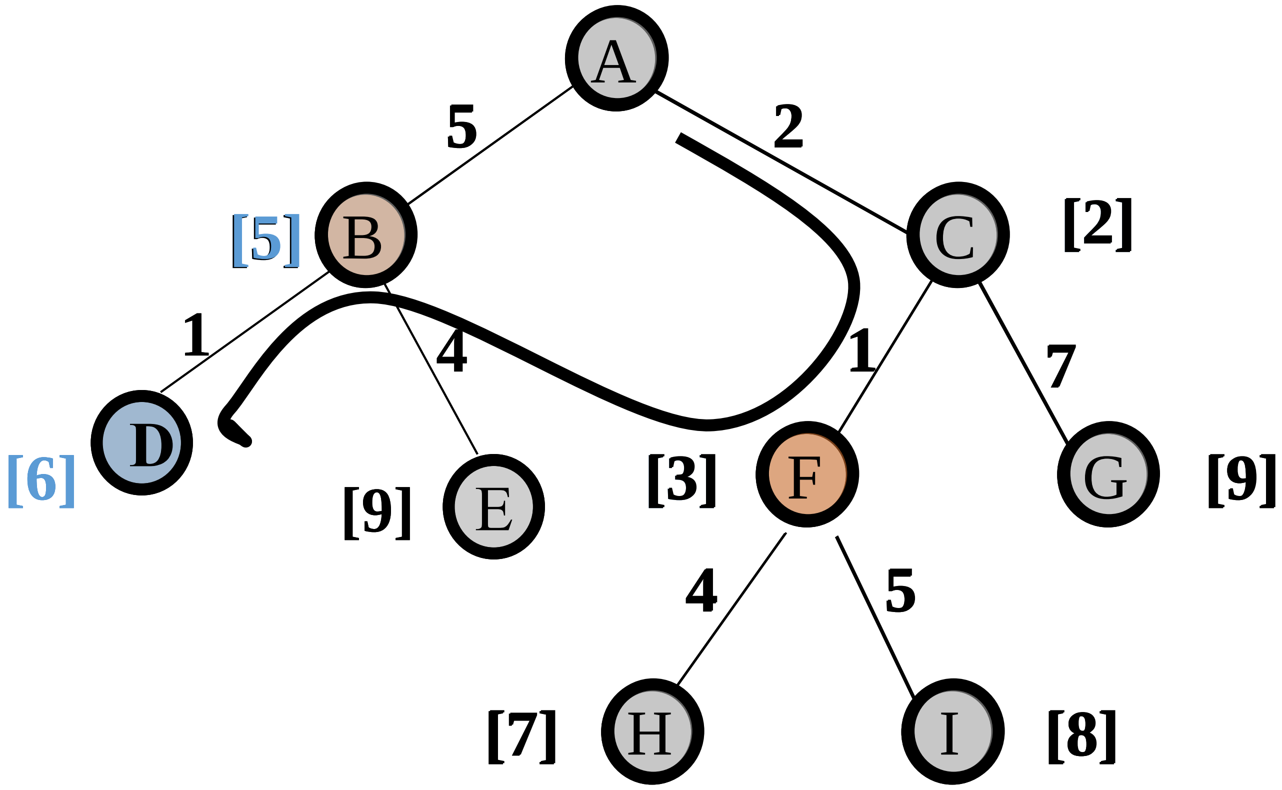
Uniform Cost Search (UCS)

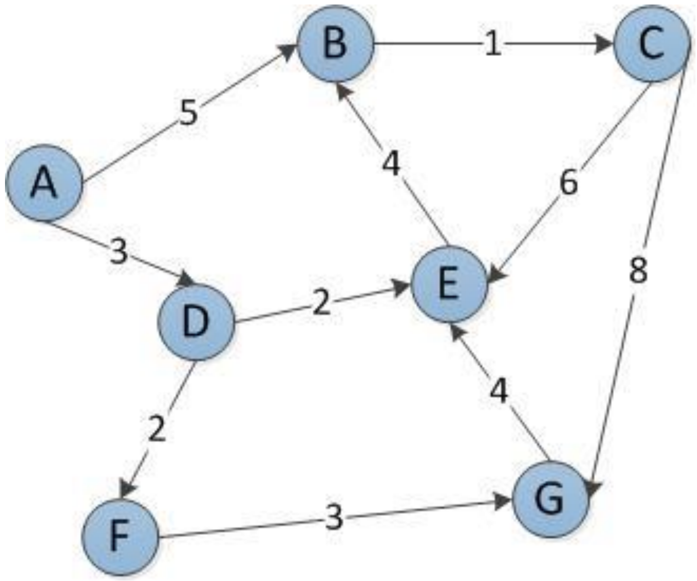


Uniform Cost Search (UCS)



Uniform Cost Search (UCS)





- Start Node: A
- Goal Node: G

Step	Frontier	Expand[*]	Explored: a set of nodes
1	{(A,0)}	A	Ø
2	{(A-D,3),(A-B,5)}	D	{A}
3	{(A-B,5),(A-D-E,5),(A-D-F,5)}	B	{A,D}
4	{(A-D-E,5),(A-D-F,5),(A-B-C,6)}	E	{A,D,B}
5	{(A-D-F,5),(A-B-C,6)}[*]	F	{A,D,B,E}
6	{(A-B-C,6),(A-D-F-G,8)}	C	{A,D,B,E,F}
7	(A-D-F-G,8)}	G	{A,D,B,E,F,C}
8	Ø		

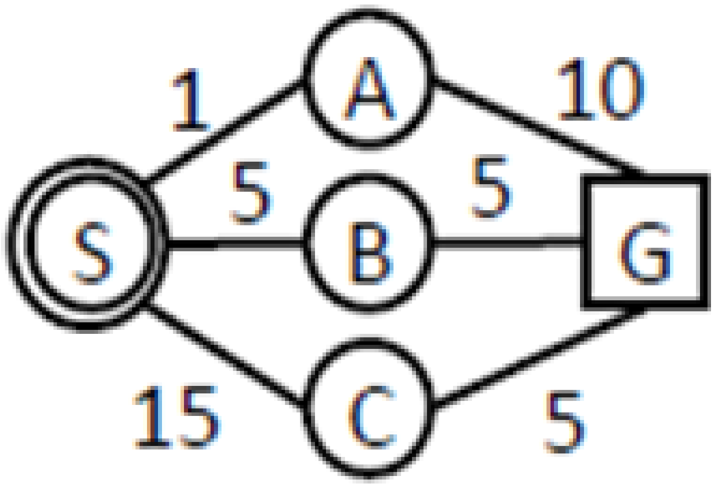
- Found the path: A -> D -> F -> G.
- *B is not added to the frontier because it is found in the explored set.

Uniform Cost Search

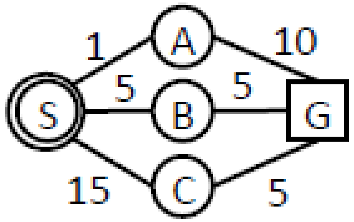
```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)   /* chooses the lowest-cost node in frontier */
     if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           frontier ← INSERT(child, frontier)
        else if child.STATE is in frontier with higher PATH-COST then
           replace that frontier node with child
```

Figure 3.14 Uniform-cost search on a graph. The algorithm is identical to the general graph search algorithm in Figure 3.7, except for the use of a priority queue and the addition of an extra check in case a shorter path to a frontier state is discovered. The data structure for *frontier* needs to support efficient membership testing, so it should combine the capabilities of a priority queue and a hash table.

Uniform Cost Search



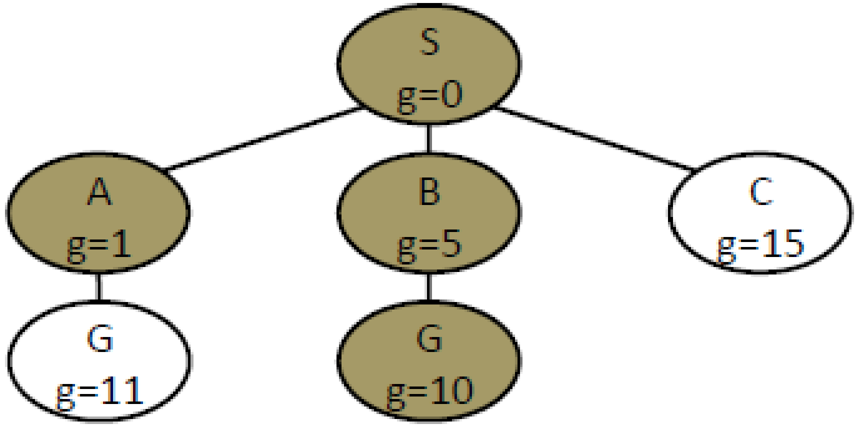
Uniform Cost Search



Route finding problem.
Steps labeled w/cost.

Order of node expansion: S A B G

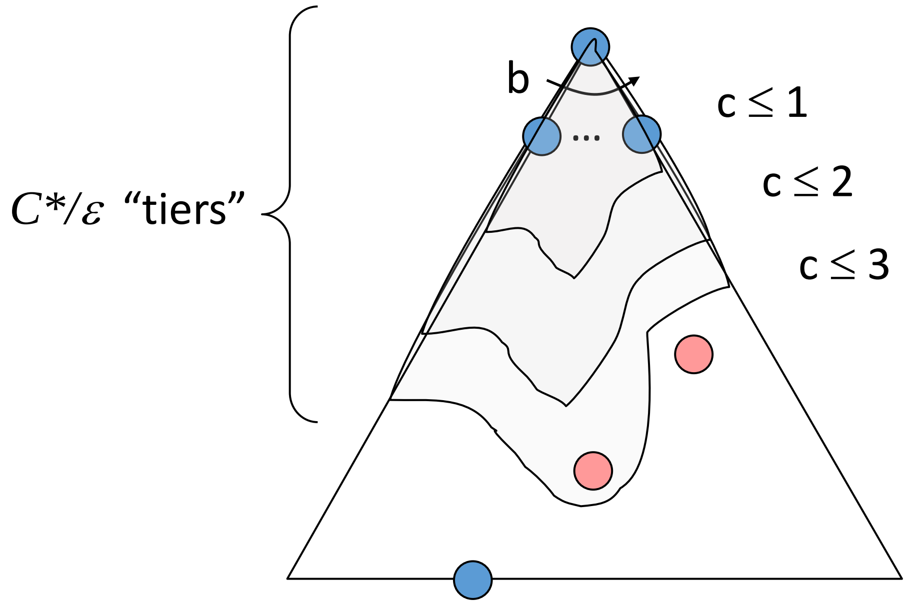
Path found: S B G Cost of path found: 10



Technically, the goal node is not really expanded, because we do not generate the children of a goal node. It is listed in "Order of node expansion" only for your convenience, to see explicitly where it was found.

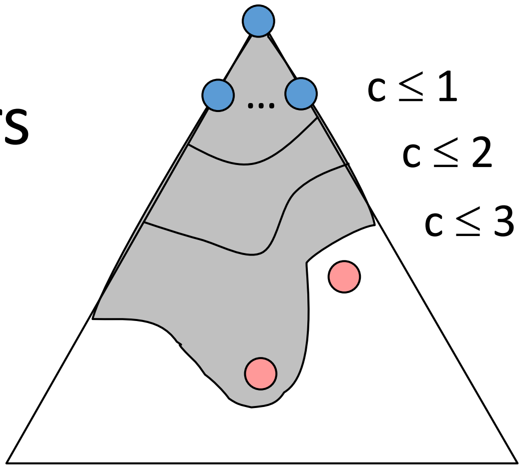
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs $C*$ and arcs cost at least ε, then the "effective depth" is roughly $C*/\varepsilon$
  - Takes time O(b$^{C*/\varepsilon}$) (exponential in effective depth)
- How much space does the fringe take?
  - Has roughly the last tier, so O(b$^{C*/\varepsilon}$)
- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!
- Is it optimal?
  - Yes! (Proof next lecture via A*)



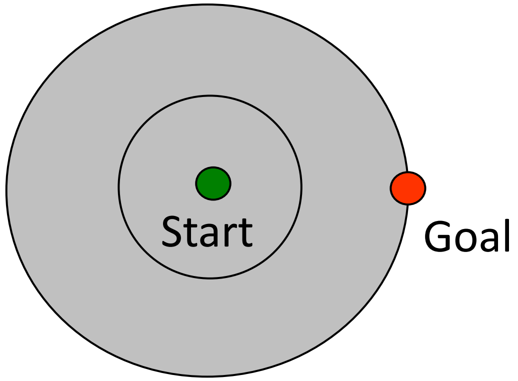
Uniform Cost Issues

- Remember: UCS explores increasing cost contours



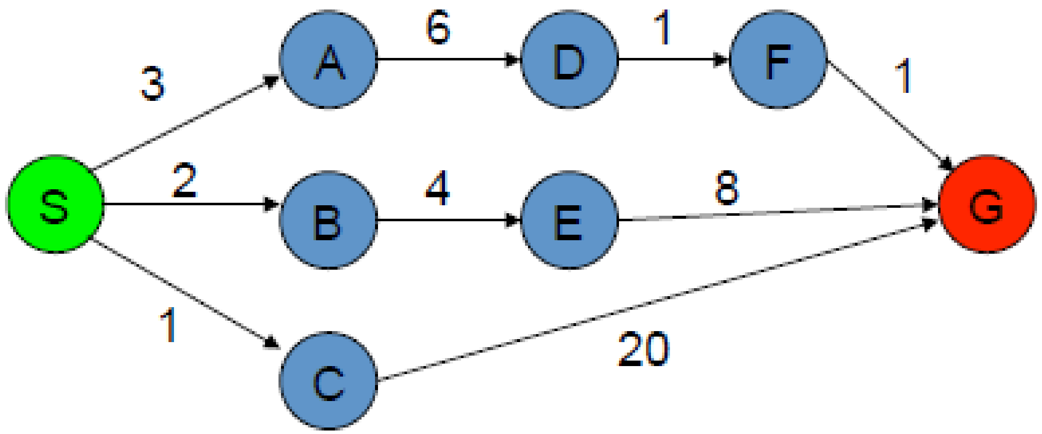
- The good: UCS is complete and optimal!

- The bad:
  - Explores options in every “direction”
  - No information about goal location

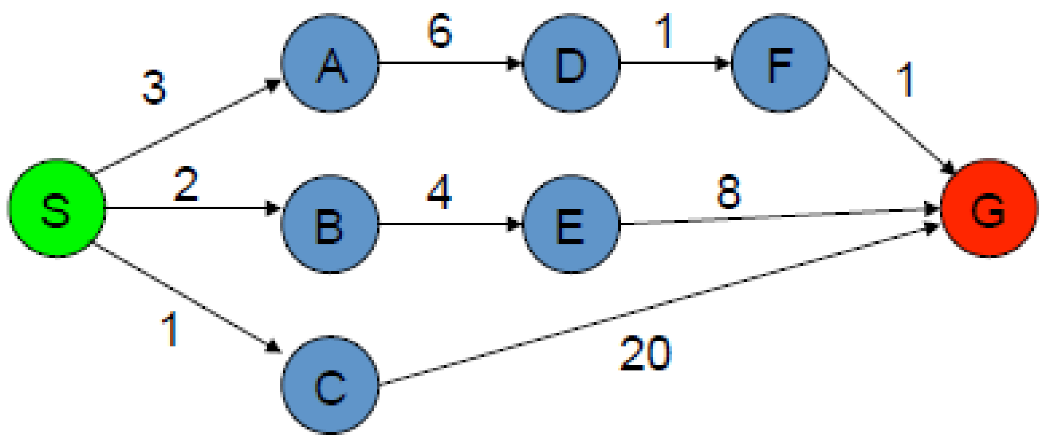


- We’ll fix that soon!

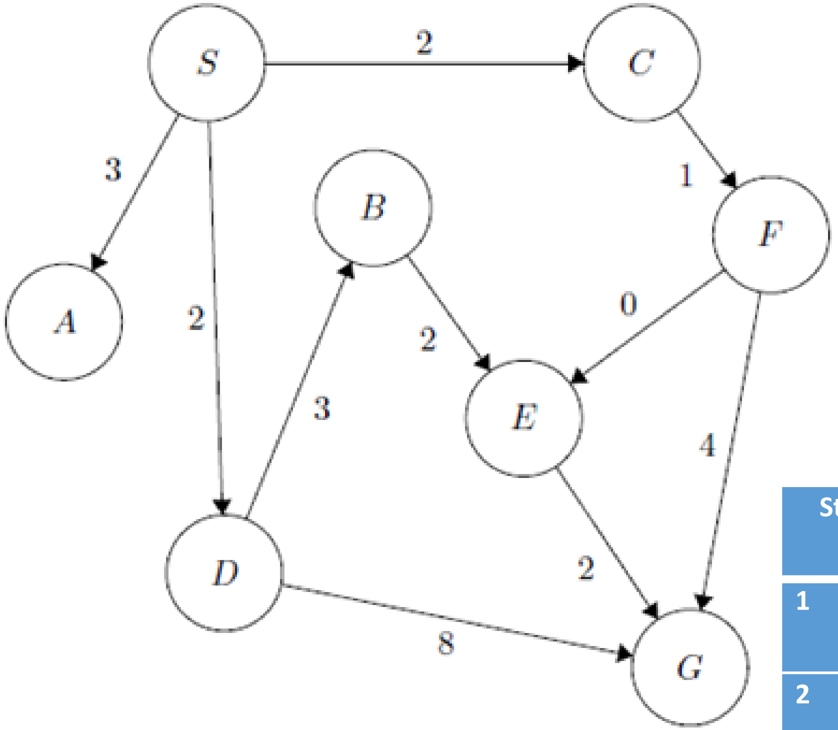
Uniform Cost Search



- The graph above shows the step-costs for different paths going from the start (S) to the goal (G).
- Use uniform cost search to find the optimal path to the goal.



	Frontier	Expand	Explored
1	S	S	Empty
2	(S-C,1) (S-B,2)(S-A,3)	C	S
3	(S-B,2)(S-A,3)(S-C-G,21)	B	S,C
4	(S-A,3)(S-C-G,21)(S-B-E,6)	A	S,C,B
5	(S-C-G,21)(S-B-E,6)(S-A-D,9)	E	S,C,B,A
6	(S-C-G,21) (S-A-D,9) (S-B-E-G,14)	D	S,C,B,A,E
7	(S-C-G,21) (S-B-E-G,14) (S-A-D-F,10)	F	S,C,B,A,E,D
8	(S-C-G,21) (S-B-E-G,14) (S-A-D-F-G,11)		
9	S-A-D-F-G,11 Goal Found		S,C,B,A,E,D,F,G



Step	Frontier	Expand	Explored: a set of Nodes
1	(S,0)	S	Ø
2	(S-C,2) (S-D,2) (S-A,3)	C	S
3	(S-D,2) (S-A,3)(S-C-F,3)	D	S,C
4	(S-A,3)(S-C-F,3)(S-D-B,5)(S-D-G,10)	A	S,C,D
5	(S-C-F,3)(S-D-B,5)(S-D-G,10)	F	S,C,D,A
6	(S-D-B,5)(S-D-G,10)(S-C-F-E,3)(S-C-F-G,7)	E	S,C,D,A,F
7	(S-D-B,5)(S-D-G,10) (S-C-F-G,7) (S-C-F-E-G,5)	B	S,C,D,A,F,E
8	(S-D-G,10) (S-C-F-G,7) (S-C-F-E-G,5)~~(S-D-B-E,7)~~	G	S,C,D,A,F,E,B
9	(S-C-F-E-G,5) Goal Found		

Bidirectional Search

- Idea
  - simultaneously search forward from S and backwards from G
  - stop when both “meet in the middle”
  - need to keep track of the intersection of 2 open sets of nodes
- What does searching backwards from G mean
  - need a way to specify the predecessors of G
  - this can be difficult,
  - e.g., predecessors of checkmate in chess?
- what if there are multiple goal states?
  - what if there is only a goal test, no explicit list?
- Complexity
  - Time complexity is best: $O\left(2b^{(d/2)}\right) = O\left(b^{(d/2)}\right)$
  - memory complexity is the same

Bidirectional Search

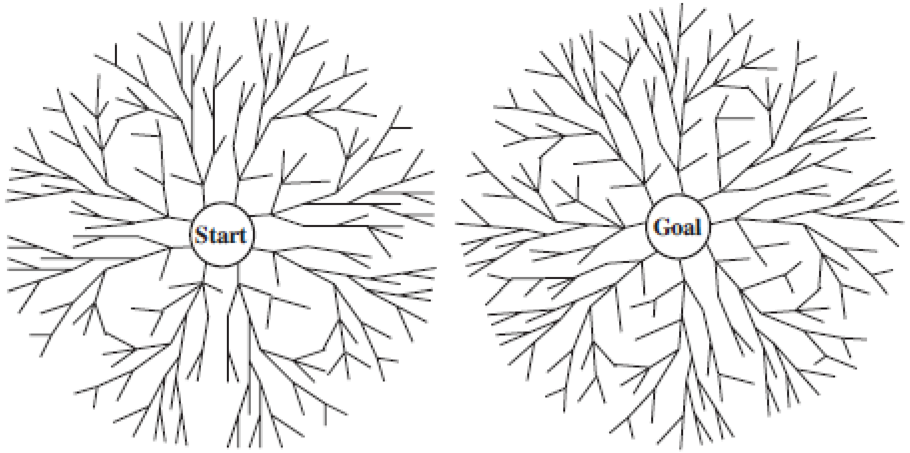


Figure 3.20 A schematic view of a bidirectional search that is about to succeed when a branch from the start node meets a branch from the goal node.

Summary of Algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$
Optimal?	Yes[c]	Yes	No	No	Yes[c]

b branching factor

d depth of the shallowest solution

m maximum depth of the search tree

l depth limit

Superscripts:

a complete if b is finite

b complete if step costs ≥ epsilon for +ve epsilon

c optimal if step costs are all identical

Summary of Algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^{a}	Yes^{a,b}	No	No	Yes^{a}	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^{c}	Yes	No	No	Yes^{c}	Yes^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.