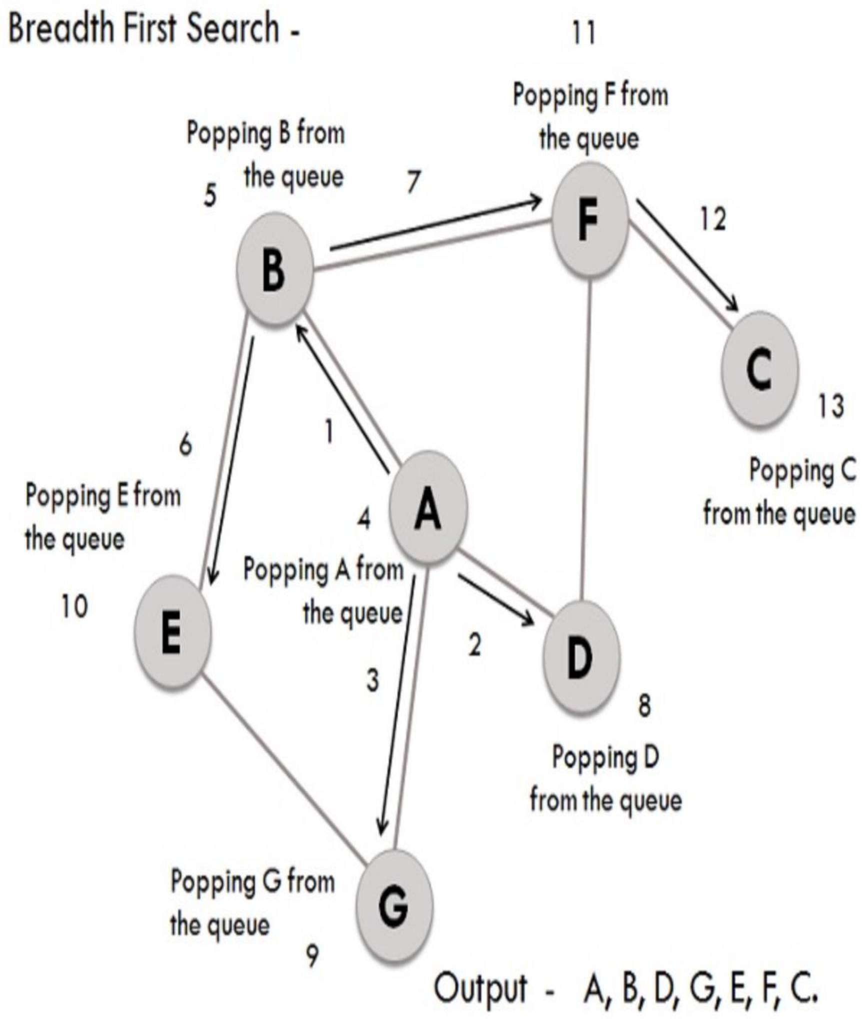
ECE 457A ADAPTIVE COOPERATIVE ALGORITHMS

LOCAL SEARCH

BFS



Example

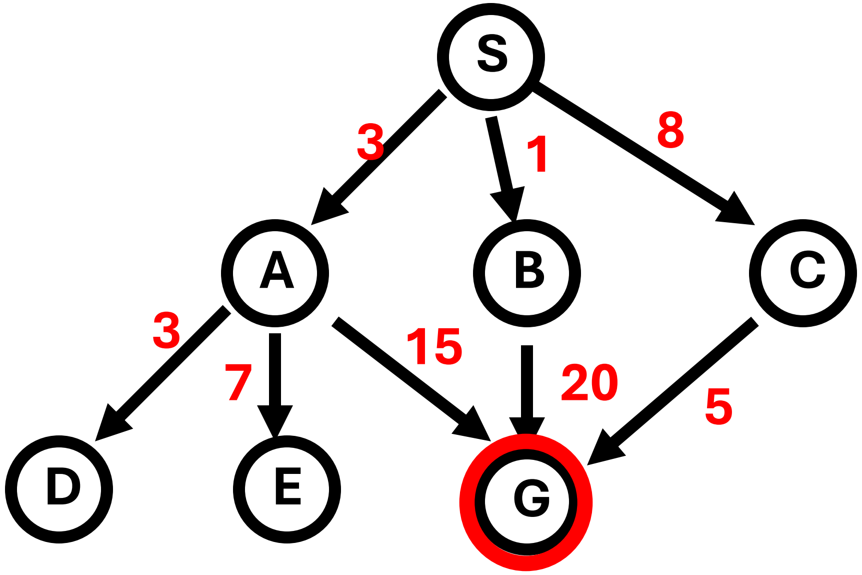
We have a graph whose vertices are A, B, C, D, E, F, G. Considering A as starting point. The steps involved in the process are:

- Vertex A is expanded and stored in the queue.
- Vertices B, D and G successors of A, are expanded and stored in the queue meanwhile Vertex A removed.
- Now B at the front end of the queue is removed along with storing its successor vertices E and F.
- Vertex D is at the front end of the queue is removed, and its connected node F is already visited.
- Vertex G is removed from the queue, and it has successor E which is already visited.
- Now E and F are removed from the queue, and its successor vertex C is traversed and stored in the queue.
- At last C is also removed and the queue is empty which means we are done.
- The generated Output is – A, B, D, G, E, F, C.

Breadth-First Search- Optimality

BFS: Optimal if and only if depth is a reflection of cost

Consider the following example.

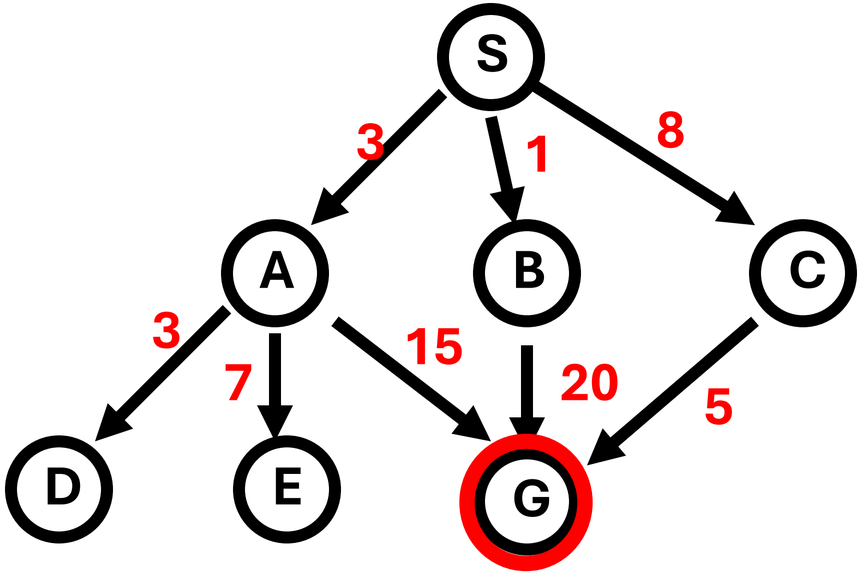


Breadth-First Search Optimality

Expanded node	Queue
	$\{ S^0 \}$
S^0	$\{ A^3\ B^1\ C^8 \}$
A^3	$\{ B^1\ C^8\ D^6\ E^{10}\ G^{18} \}$
B^1	$\{ C^8\ D^6\ E^{10}\ G^{18}\ G^{21} \}$
C^8	$\{ D^6\ E^{10}\ G^{18}\ G^{21}\ G^{13} \}$
D^6	$\{ E^{10}\ G^{18}\ G^{21}\ G^{13} \}$
E^{10}	$\{ G^{18}\ G^{21}\ G^{13} \}$
G^{18}	$\{ G^{21}\ G^{13} \}$

Solution path found is S A G , cost 18

Number of nodes expanded (including goal node) = 7



Uniform Cost Search

- A breadth-first search finds the ***shallowest goal state*** and will therefore be the cheapest solution provided the *path cost is a function of the depth of the solution*.

- But, if this is not the case, then breadth-first search is not guaranteed to find the best (i.e. cheapest solution).

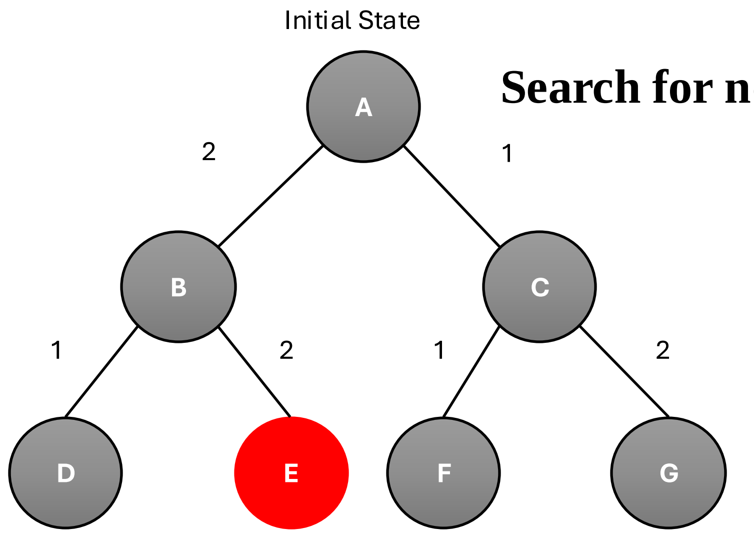
- Uniform cost search remedies this by expanding the ***lowest cost node*** on the fringe, where cost is the path cost, *g(n)*.

- In the following slides, the values that are attached to paths are the cost of using that path.

Uniform Cost Search

Assume the example tree used here with different edge costs, represented by numbers next to the edges.

- Notations:
  - Generated node
  - Expanded node



Search for node E

Uniform Cost Search

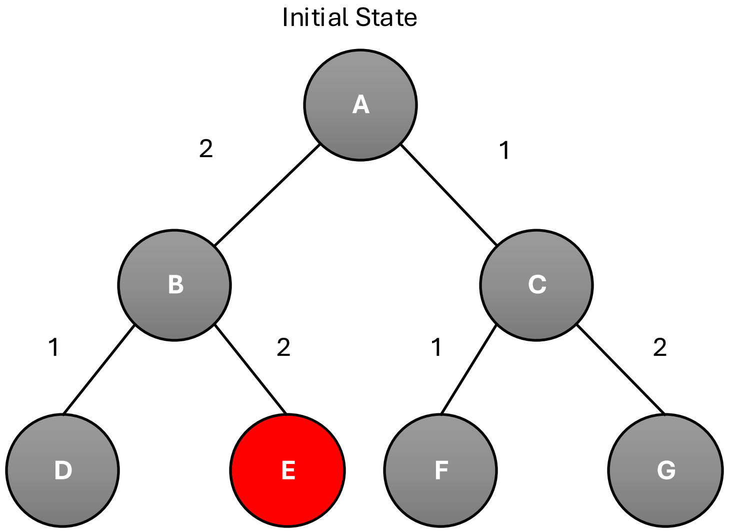
Uniform Cost Strategy:

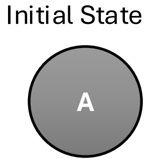
- Start with the initial state.

Open List

A
0

Closed List

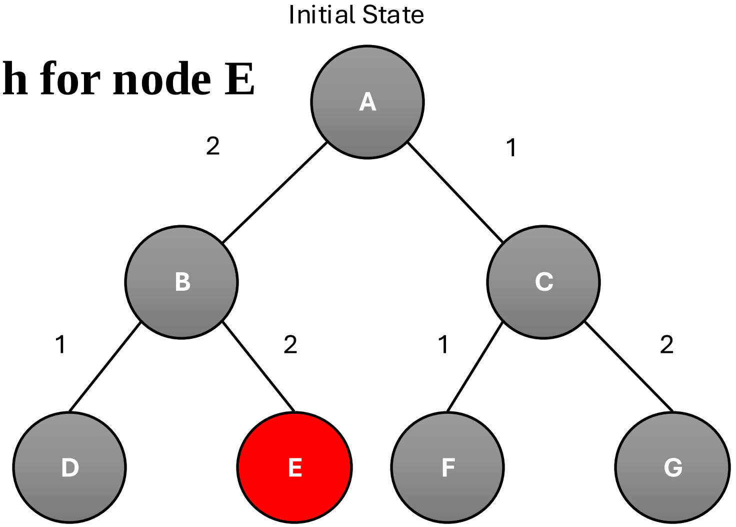


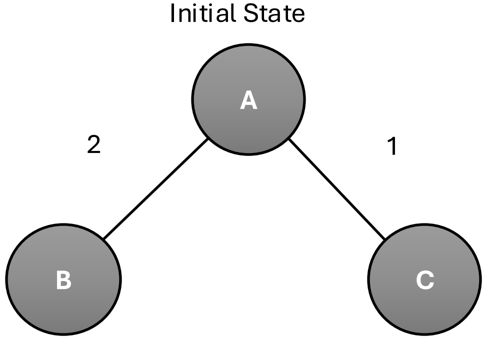


Search for node E

Uniform Cost Search

Search for node E





Uniform Cost Strategy:

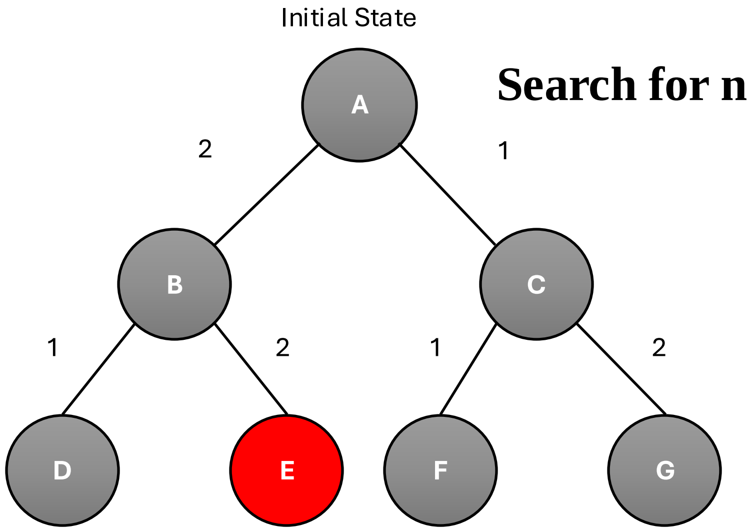
- Look at the node with the lowest cost in queue.
- Is this node the goal state?
- If not, Expand node.

Open List

A
0

Closed List

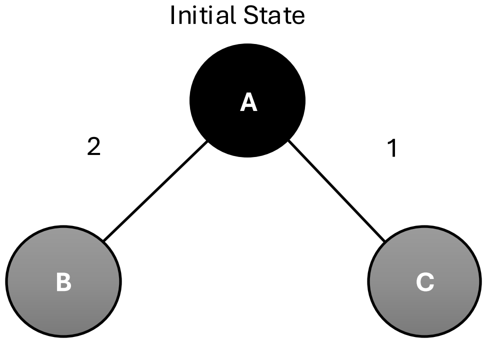
Uniform Cost Search



Search for node E

Uniform Cost Strategy:

- Move the expanded node to Closed list.
- Add the generated nodes to Open list



Open List

B	C
2	1

Closed List

A

Uniform Cost Search

Uniform Cost Strategy:

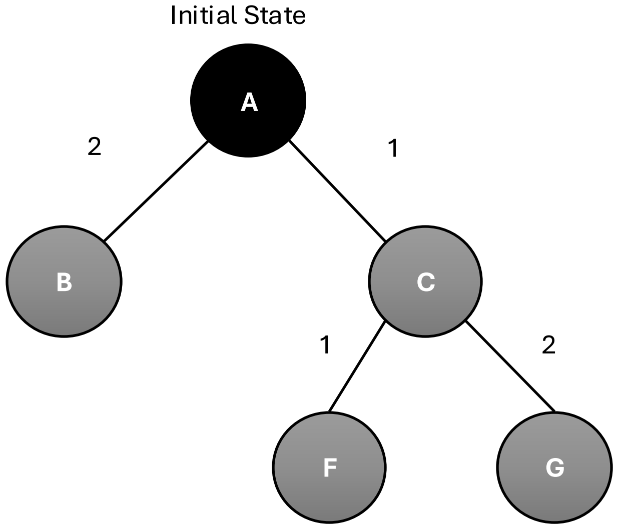
- Look at the node with the lowest cost in queue.
- Is this node the goal state?
- If not, Expand node.

Open List

B	C
2	1

Closed List

A



Uniform Cost Search

Uniform Cost Strategy:

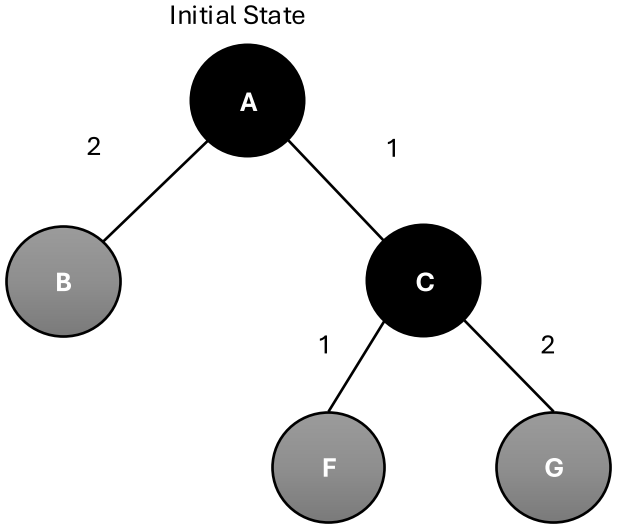
- Move the expanded node to Closed list.
- Add the generated nodes to Open list

Open List

B	F	G
2	2	3

Closed List

A	C



Uniform Cost Search

Uniform Cost Strategy:

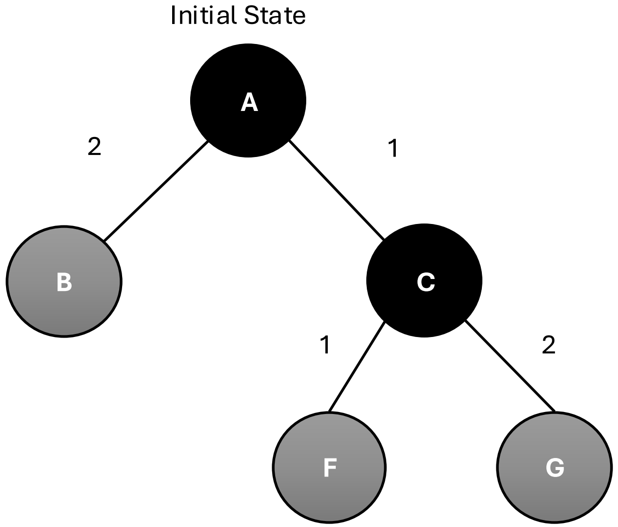
- Look at the node with the lowest cost in queue.
- Is this node the goal state?
- If not, Expand node.

Open List

B	F	G
2	2	3

Closed List

A	C



Uniform Cost Search

Uniform Cost Strategy:

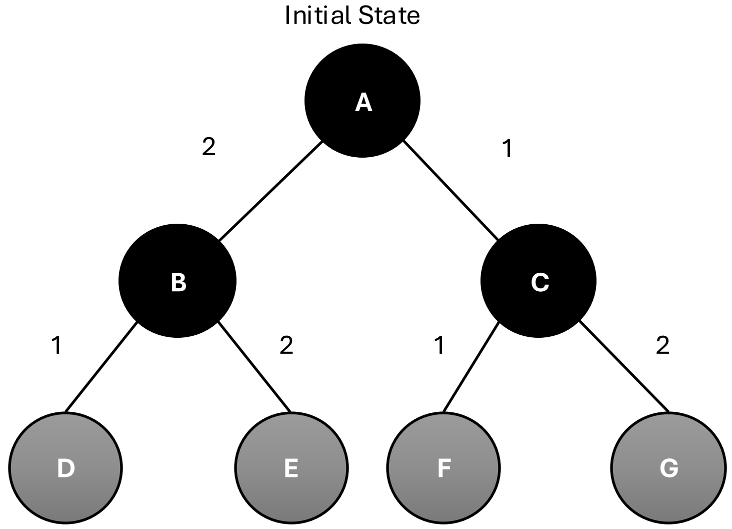
- Move the expanded node to Closed list.
- Add the generated nodes to Open list

Open List

F	G	D	E
2	3	3	4

Closed List

A	C	B



Uniform Cost Search

Uniform Cost Strategy:

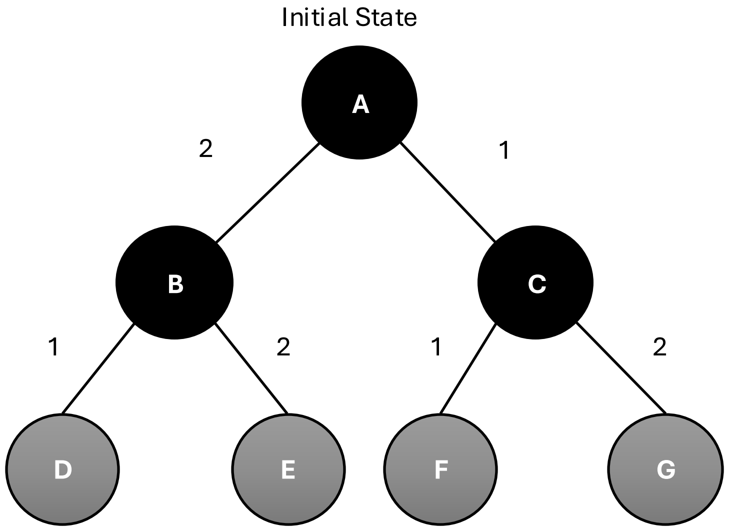
- Look at the node with the lowest cost in queue.
- Is this node the goal state?
- If not, Expand node.

Open List

F	G	D	E
2	3	3	4

Closed List

A	C	B



Uniform Cost Search

Uniform Cost Strategy:

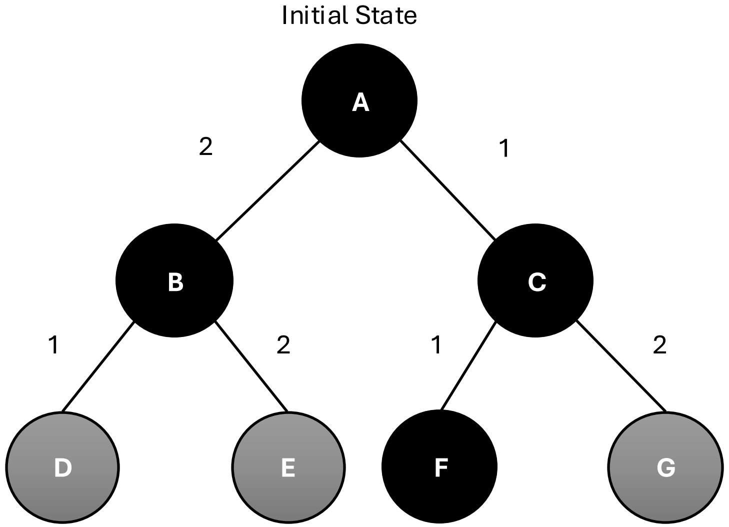
- Move the expanded node to Closed list.
- Add the generated nodes to Open list

Open List

G	D	E
3	3	4

Closed List

A	C	B	F



Uniform Cost Search

Uniform Cost Strategy:

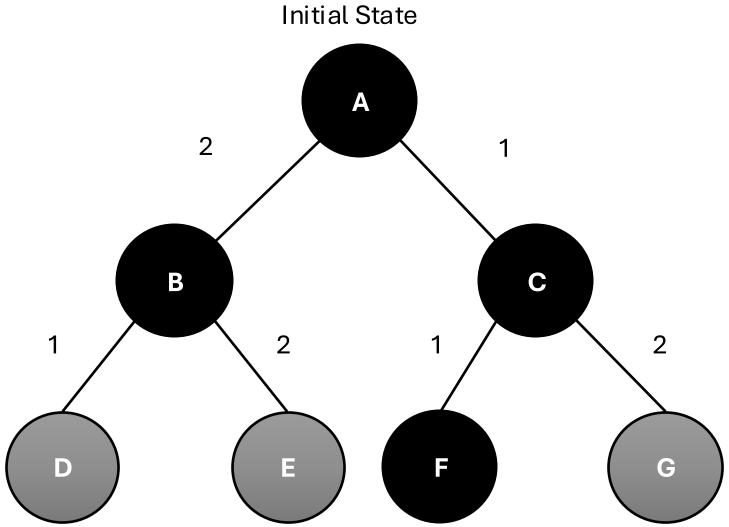
- Look at the node with the lowest cost in queue.
- Is this node the goal state?
- If not, Expand node.

Open List

G	D	E
3	3	4

Closed List

A	C	B	F



Uniform Cost Search

Uniform Cost Strategy:

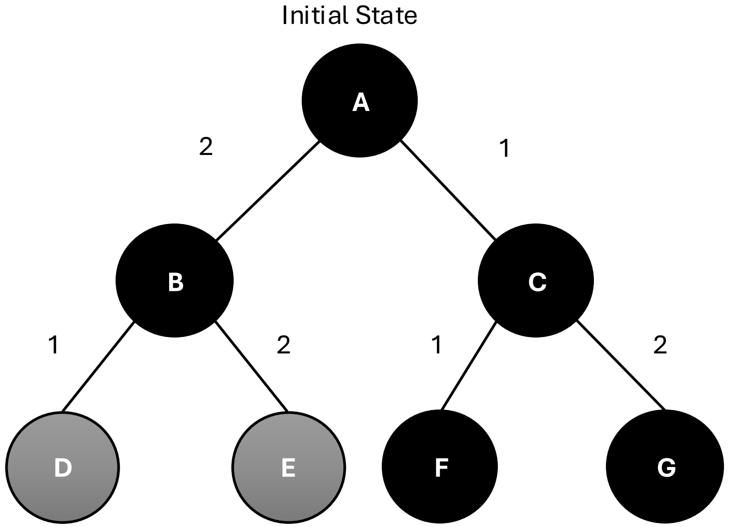
- Move the expanded node to Closed list.
- Add the generated nodes to Open list

Open List

D	E
3	4

Closed List

A	C	B	F	G



Uniform Cost Search

Uniform Cost Strategy:

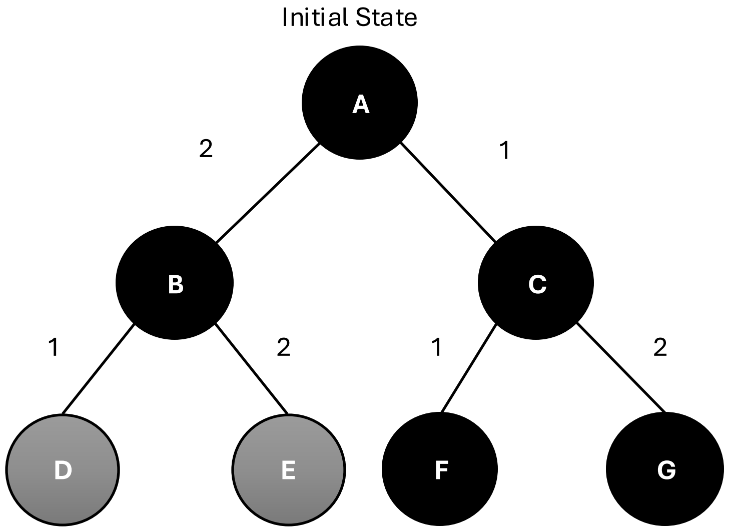
- Look at the node with the lowest cost in queue.
- Is this node the goal state?
- If not, Expand node.

Open List

D	E
3	4

Closed List

A	C	B	F	G



Uniform Cost Search

Uniform Cost Strategy:

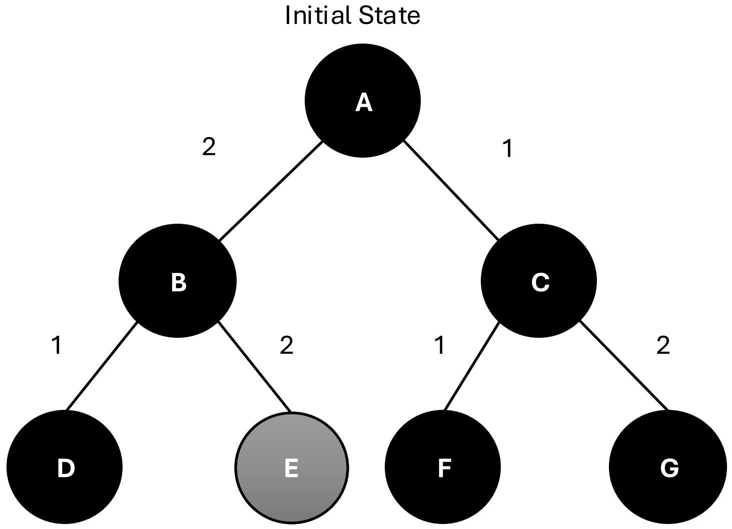
- Move the expanded node to Closed list.
- Add the generated nodes to Open list

Open List

E
4

Closed List

A	C	B	F	G	D



Uniform Cost Search

Uniform Cost Strategy:

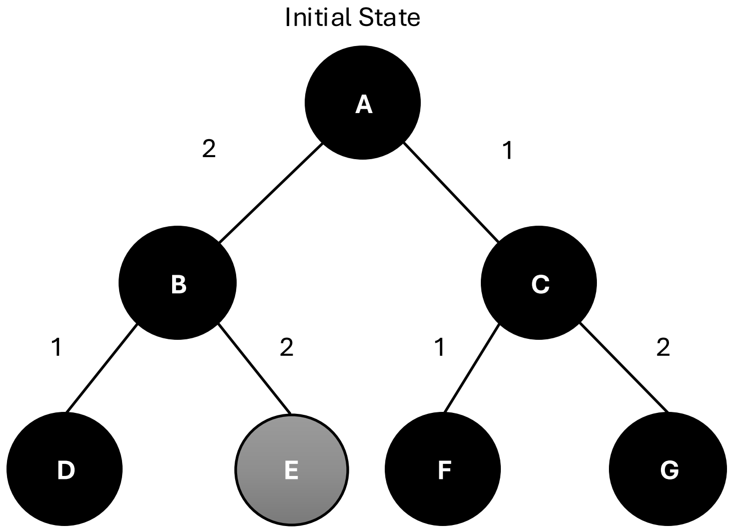
- Look at the node with the lowest cost in queue.
- Is this node the goal state?
- If not, Expand node.

Open List

E
4

Closed List

A	C	B	F	G	D



Uniform Cost Search

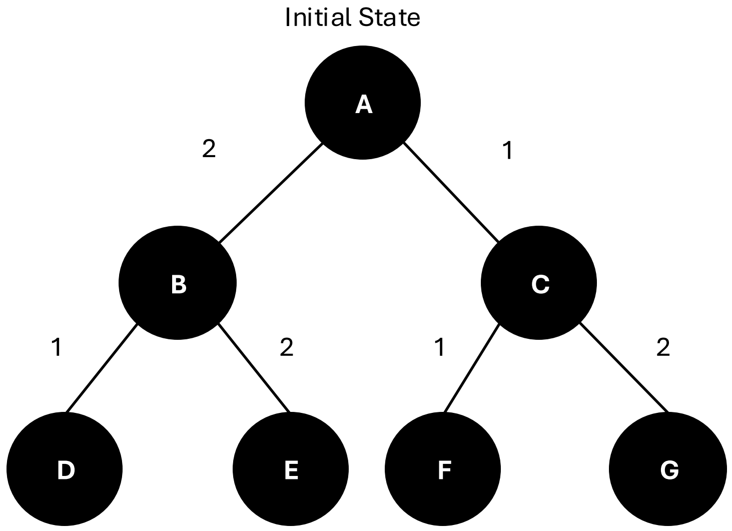
Uniform Cost Strategy:

- Move the expanded node to Closed list.
- Add the generated nodes to Open list

Open List

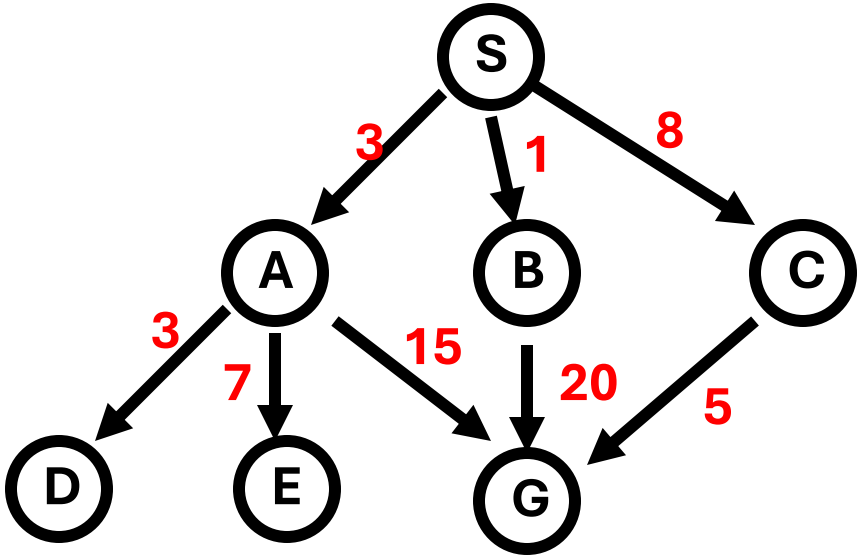
Closed List

A	**C**	**B**	**F**	**G**	**D**	**E**

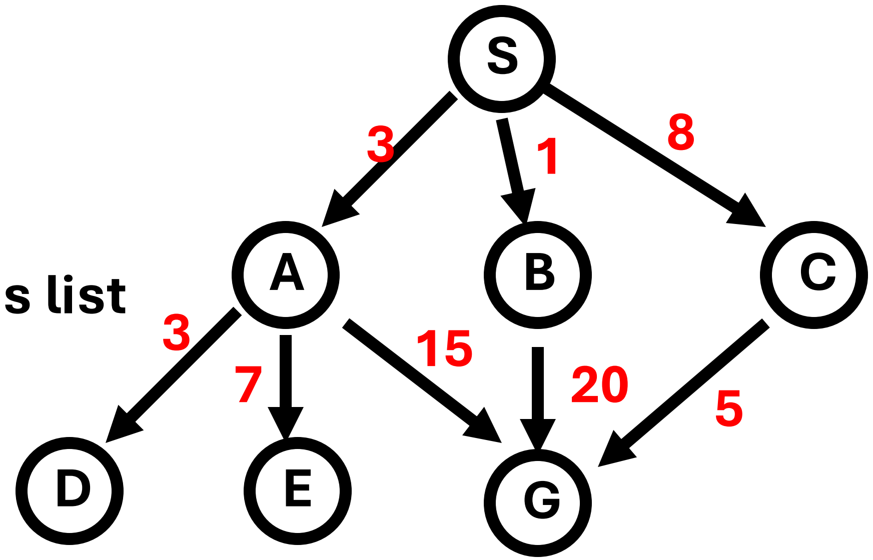


A→ B→E

Uniform Cost Search



Uniform Cost Search Optimality

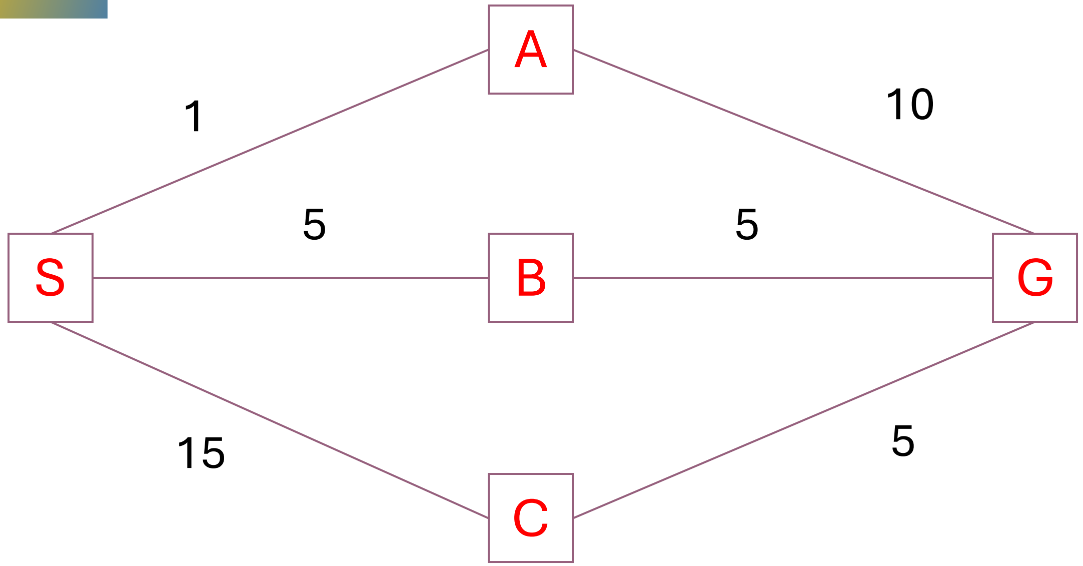


Expanded node	Nodes list
	{ S^0 }
S^0	{ B^1 A^3 C^8 }
B^1	{ A^3 C^8 G^{21} }
A^3	{ D^6 C^8 E^{10} G^{18} G^{21} }
D^6	{ C^8 E^{10} G^{18} G^1 }
C^8	{ E^{10} G^{13} G^{18} G^{21} }
E^{10}	{ G^{13} G^{18} G^{21} }
G^{13}	{ G^{18} G^{21} }

Solution path found is S C G, cost 13

Number of nodes expanded (including goal node) = 7

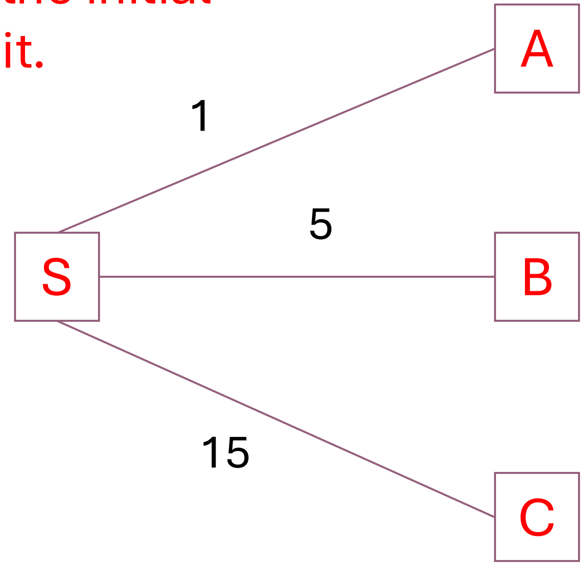
Uniform Cost Search - Example



Find the shortest route from node S to node G; that is, node S is the initial state and node G is the goal state. In terms of path cost, we can clearly see that the route *SBG* is the cheapest route. However, if we let breadth-first search loose on the problem it will find the non-optimal path *SAG*, assuming that A is the first node to be expanded at level 1.

Uniform Cost Search - Example

We will start with the initial state and expand it.



Size of Queue: 1

Queue: S

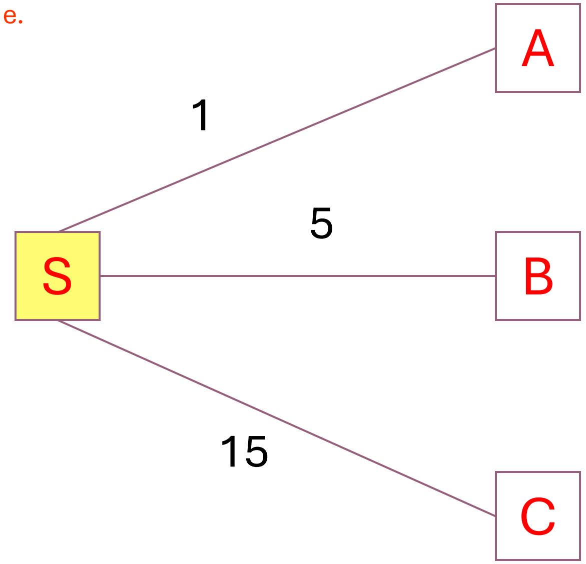
Nodes expanded: 1

Current action: Expanding

Current level: 0

Uniform Cost Search - Example

Node S is removed from the queue and the revealed nodes are added to the queue.



The queue is then sorted on path cost. Nodes with cheaper path cost have priority.

Size of Queue: 3

Queue: A(1), B(5), C(15)

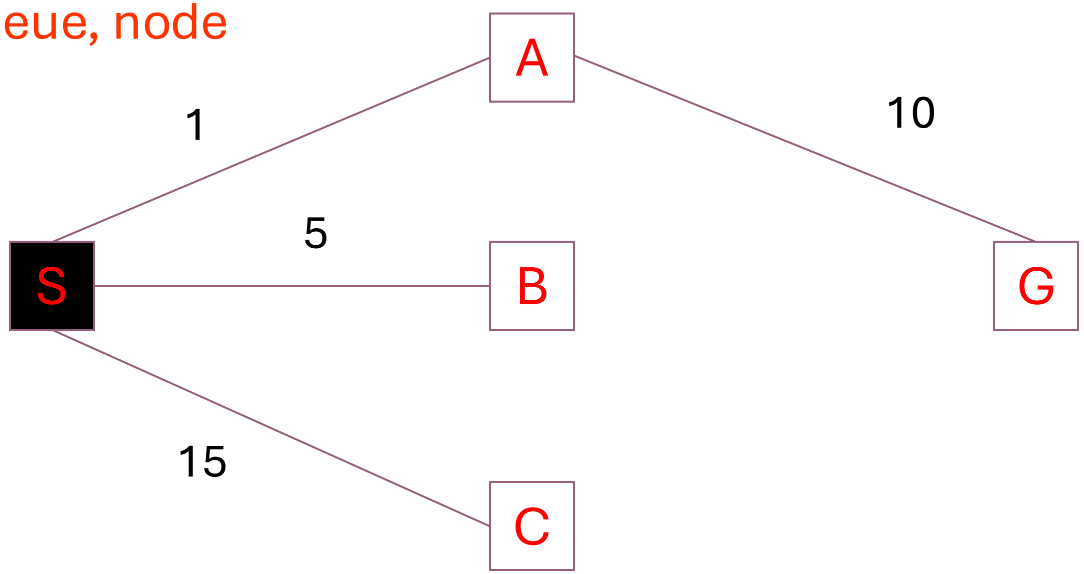
Nodes expanded: 1

Current action: Expanding

Current level: 0

Uniform Cost Search - Example

We now expand the node at the front of the queue, node A.



Size of Queue: 3

Queue: A(1), B(5), C(15)

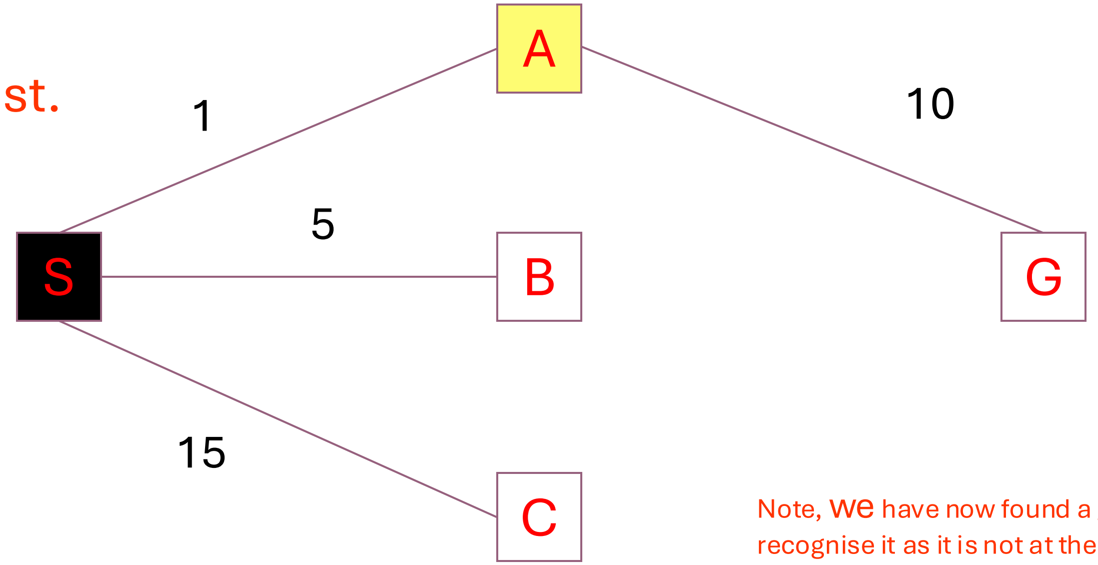
Nodes expanded: 2

Current action: Expanding

Current level: 1

Uniform Cost Search - Example

Node A is removed from the queue and the revealed node (node G) is added to the queue, sorted on path cost.



Note, we have now found a goal state but do not recognise it as it is not at the front of the queue. Node B is the cheaper node.

Size of Queue: 3

Queue: B(5), G(11), C(15)

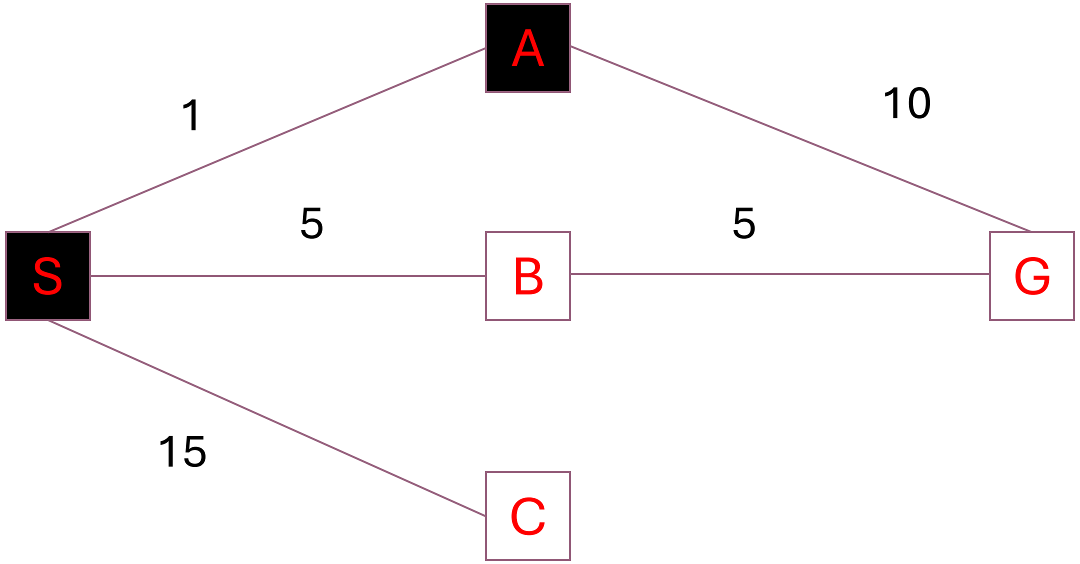
Nodes expanded: 2

Current action: Expanding

Current level: 1

Uniform Cost Search - Example

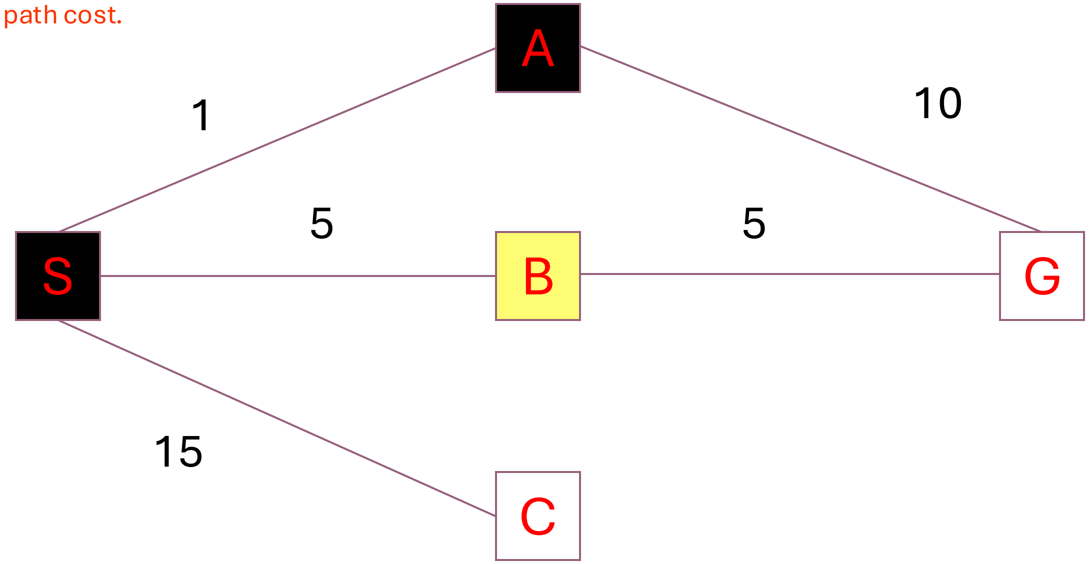
We now expand the node at the front of the queue, node B.



Size of Queue: 3	Queue: B(5), G(11), C(15)
Nodes expanded: 3	Current action: Expanding
	Current level: 1

Uniform Cost Search - Example

Node B is removed from the queue and add the revealed node (node G) is added, sorted by path cost.

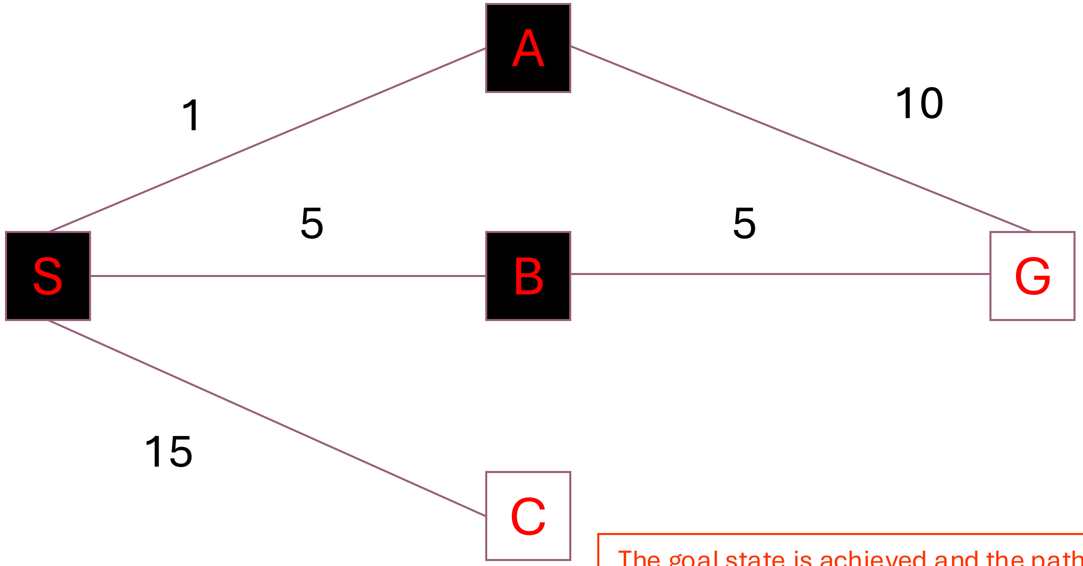


Note, node G now appears in the queue twice.

Size of Queue: 3	**Queue: G(10), G(11), C(15)**	
Nodes expanded: 3	**Current action: Expanding**	**Current level: 1**

Uniform Cost Search - Example

G10 is at the front of the queue, now we proceed to the Goal state



The goal state is achieved and the path S-B-G is returned. In relation to path cost, UCS has found the optimal route.

Size of Queue: 0	**Queue: EMPTY**
Nodes expanded: 3	**Current action: SEARCH COMPLETE**

Current level: 2

Uniform Cost Search

- Expansion of Breadth-First Search

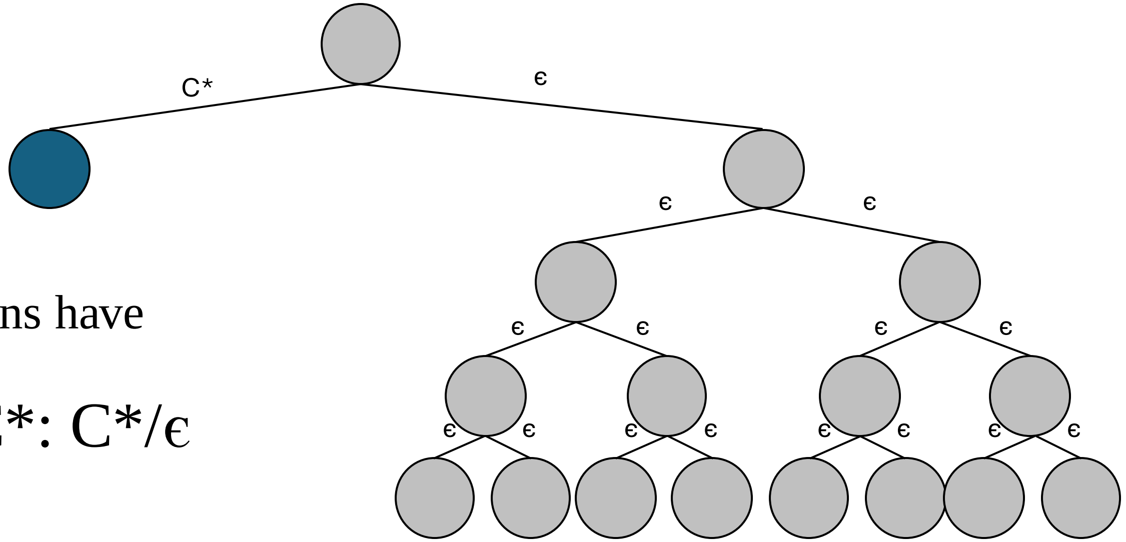
- Explore the cheapest node first (in terms of path cost)

- Condition: No zero-cost or negative-cost edges.
    - Minimum cost is ϵ

- Breadth-First Search is Uniform-Cost Search with constant-cost edges

- Let g(n) be the sum of the edges costs from root to node n. If g(n) is our overall cost function,
    - Then the **Uniform Cost Search** becomes **Dijkstra's single-source-shortest-path algorithm.**

Uniform-Cost Search



- Upper-bound case: goal has path cost C*, all other actions have minimum cost of ϵ
  - Depth explored before taking action C*: C*/ϵ
  - Depth of fringe nodes: C*/ϵ + 1
  - Space & time complexity: all generated nodes: $O(b^{C^*/\epsilon+1})$

Uniform-Cost Search

- Complete given a finite tree
- Optimal
- Time complexity = $O(b^{C^*/\epsilon+1}) \geq O(b^{d+1})$ ← under equal steps
- Space complexity = $O(b^{C^*/\epsilon+1}) \geq O(b^{d+1})$

Uninformed Search Strategies

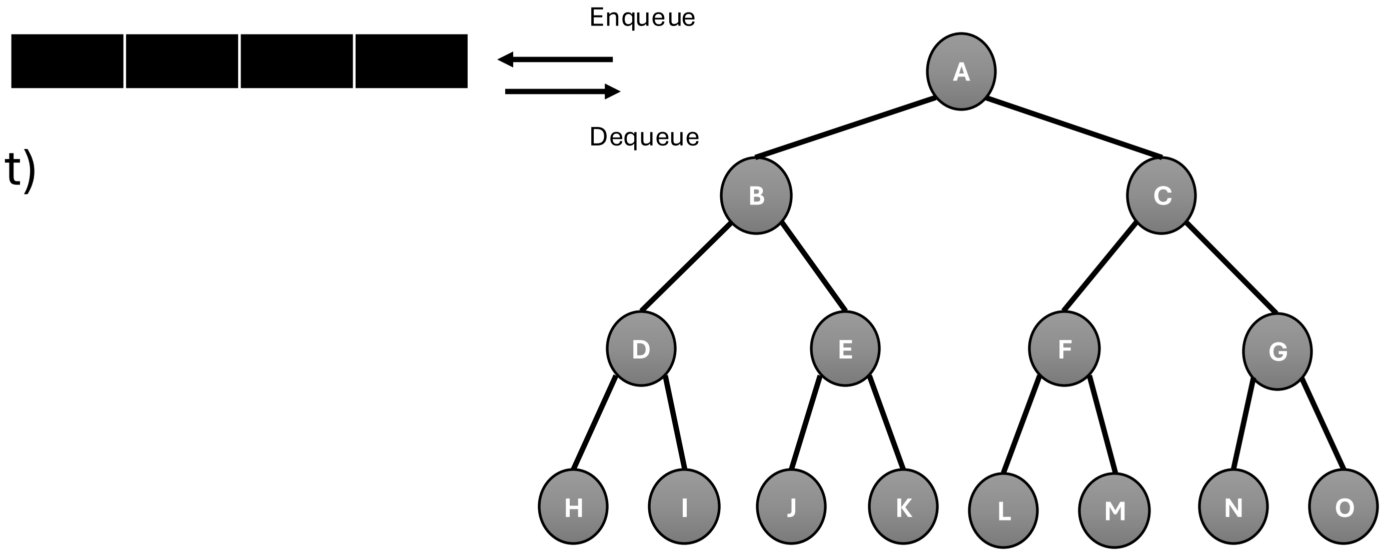
Depth-First Search

Depth First Search

- Expand deepest unexpanded node first
  - The root is examined first; then the left child of the root; then the left child of this node, etc. until a leaf is found. At a leaf, backtrack to the lowest right child and repeat.

- Does not have to keep all nodes on the open list, only retains the children of a single state

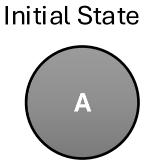
- Fringe is a LIFO queue (Last in, First out)
  - Stack data structure
  - Put successors at front

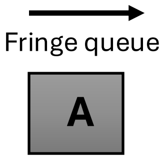


Depth-first search

Depth-first Strategy:

- Start with the initial state.
- Enqueue node

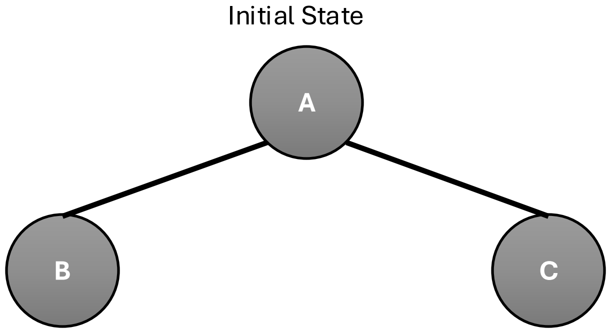




Depth-first search

Depth-first Strategy:

- Look at last node in queue.
- Is this node the goal state?
- If not, Expand node.



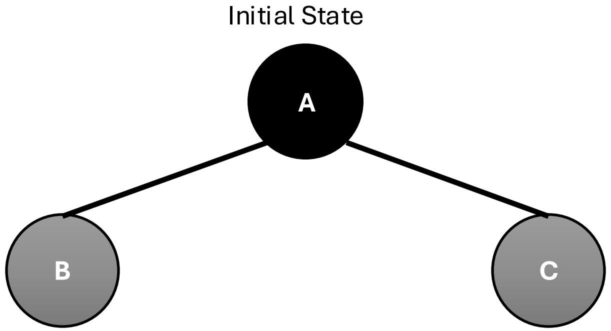
Fringe queue

A

Depth-first search

Depth-first Strategy:

- Dequeue the expanded node.
- Add the generated nodes to queue



Fringe queue

C	B

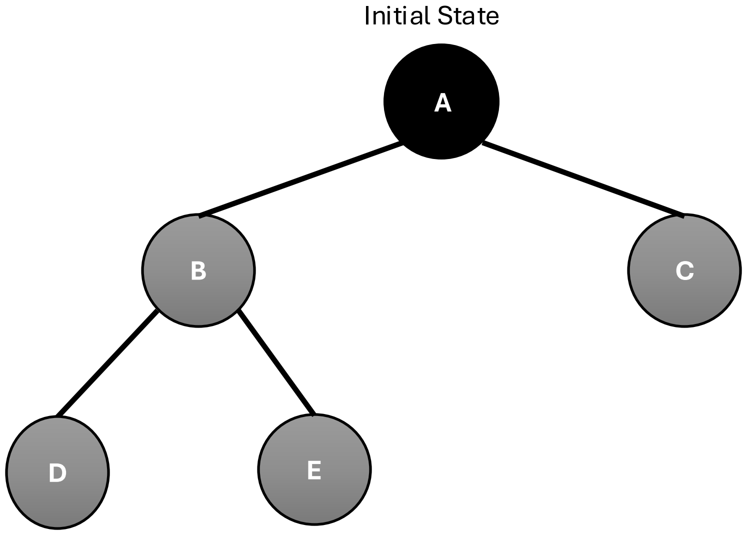
Depth-first search

Depth-first Strategy:

- Look at last node in queue.
- Is this node the goal state?
- If not, Expand node.

Fringe queue

C	B



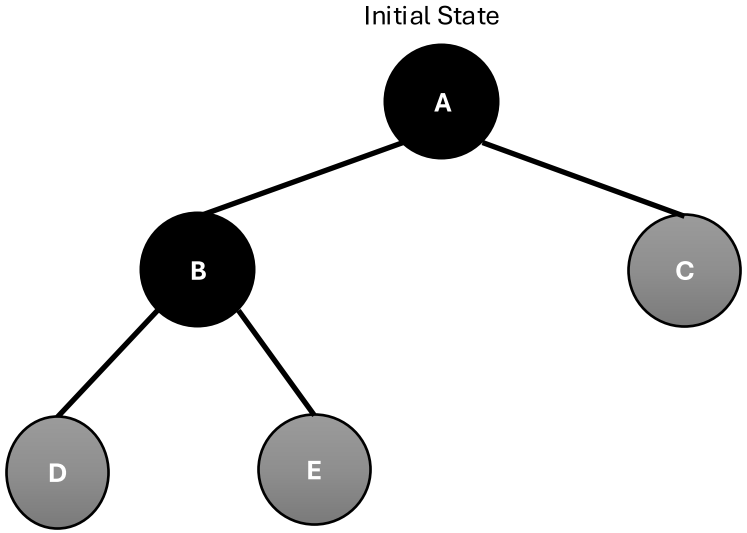
Depth-first search

Depth-first Strategy:

- Dequeue the expanded node.
- Add the generated nodes to queue

Fringe queue

C	E	D



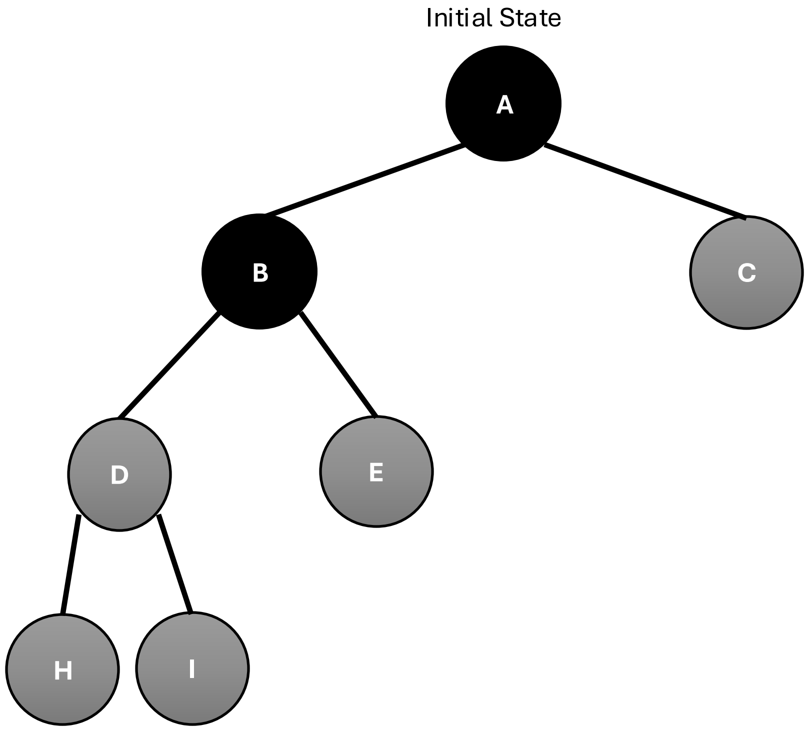
Depth-first search

Depth-first Strategy:

- Look at last node in queue.
- Is this node the goal state?
- If not, Expand node.

Fringe queue

C	E	D



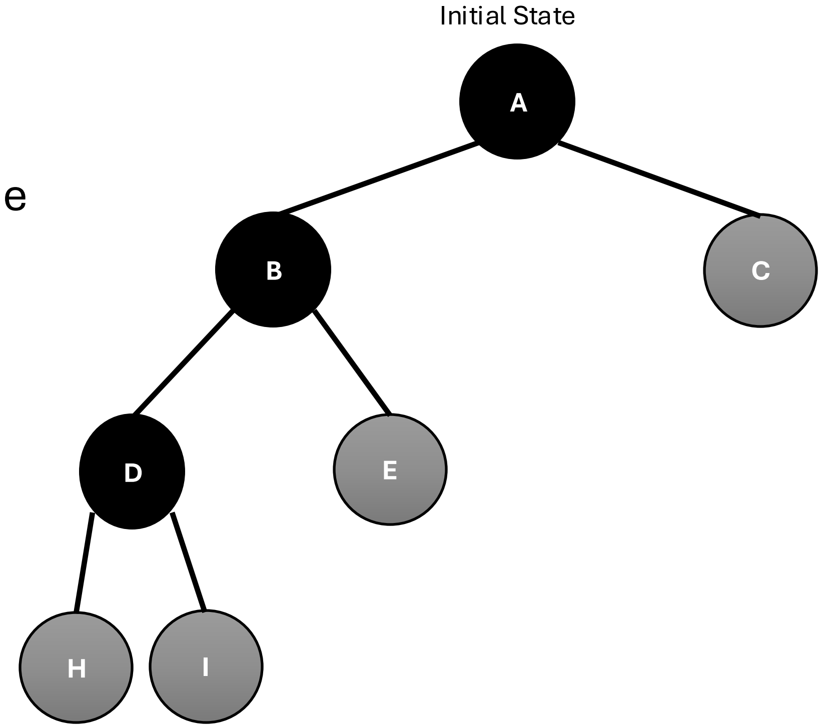
Depth-first search

Depth-first Strategy:

- Dequeue the expanded node.
- Add the generated nodes to queue

Fringe queue

C	E	I	H



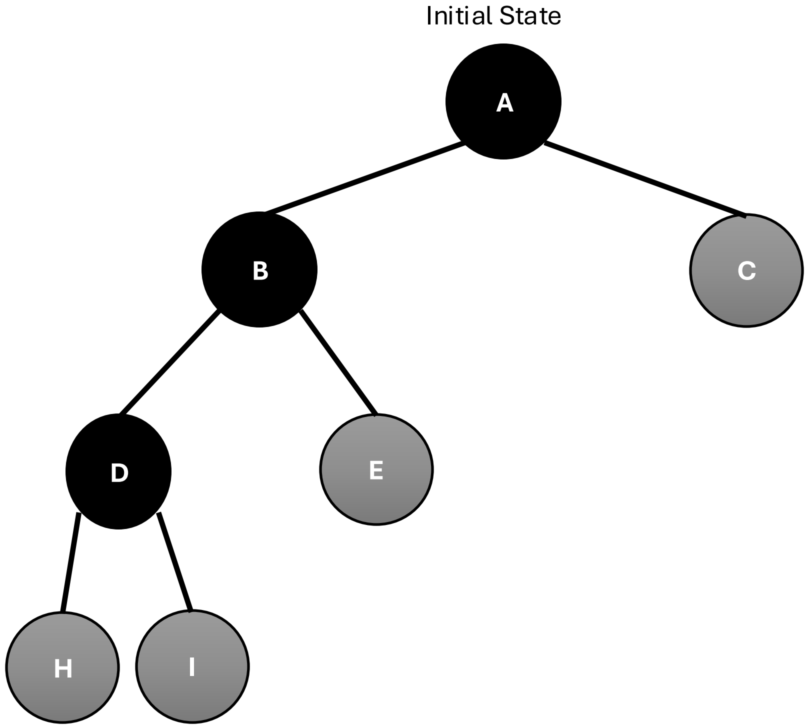
Depth-first search

Depth-first Strategy:

- Look at last node in queue.
- Is this node the goal state?
- If not, Expand node.

Fringe queue

C	E	I	H



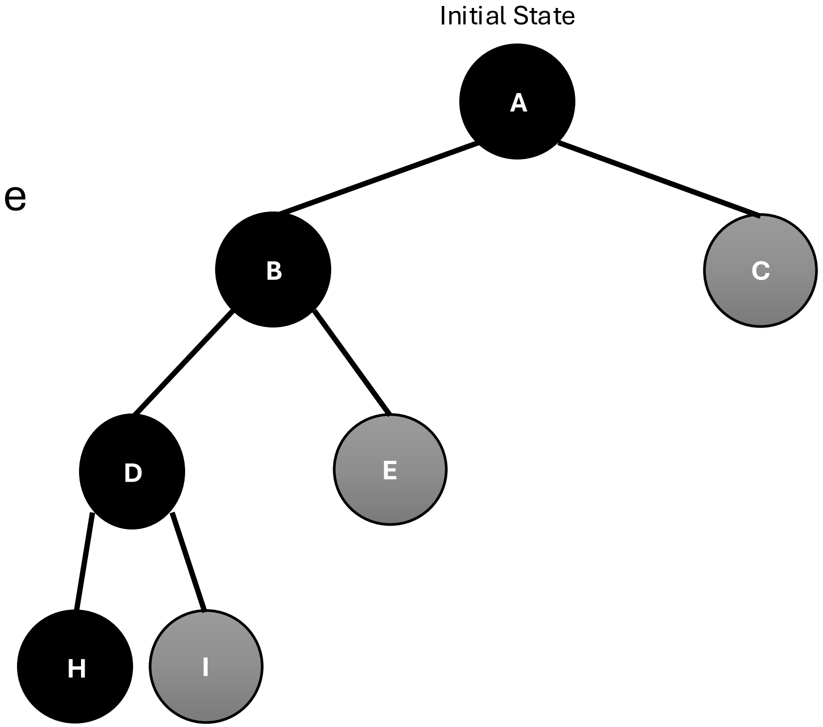
Depth-first search

Depth-first Strategy:

- Dequeue the expanded node.
- Add the generated nodes to queue

Fringe queue

C	E	I



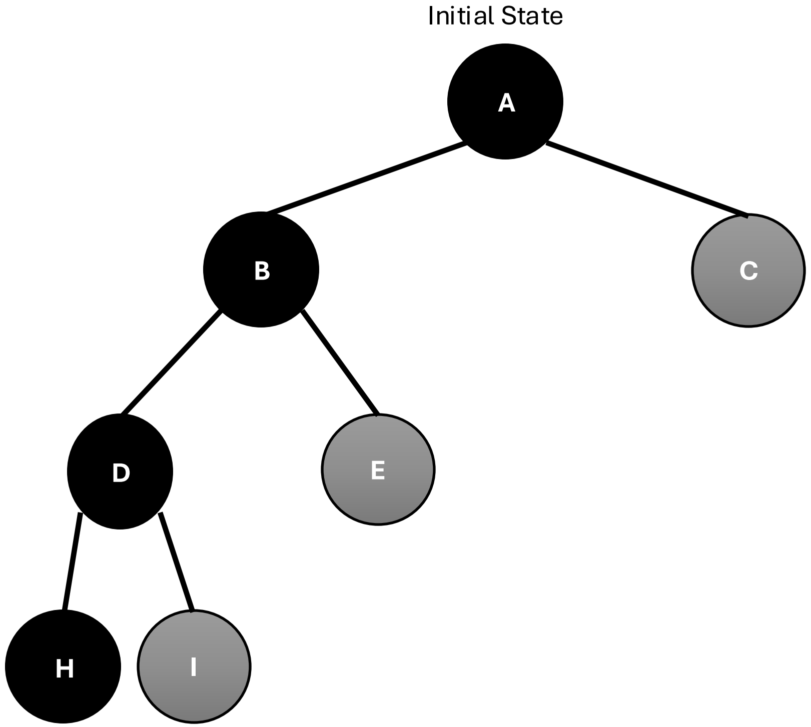
Depth-first search

Depth-first Strategy:

- Look at last node in queue.
- Is this node the goal state?
- If not, Expand node.

Fringe queue

C	E	I



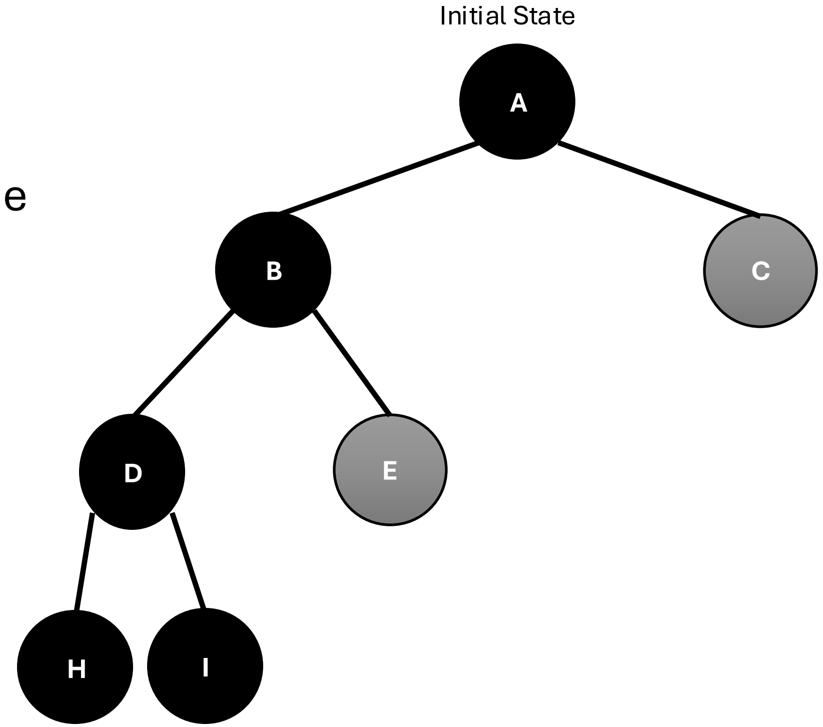
Depth-first search

Depth-first Strategy:

- Dequeue the expanded node.
- Add the generated nodes to queue

Fringe queue

C	E



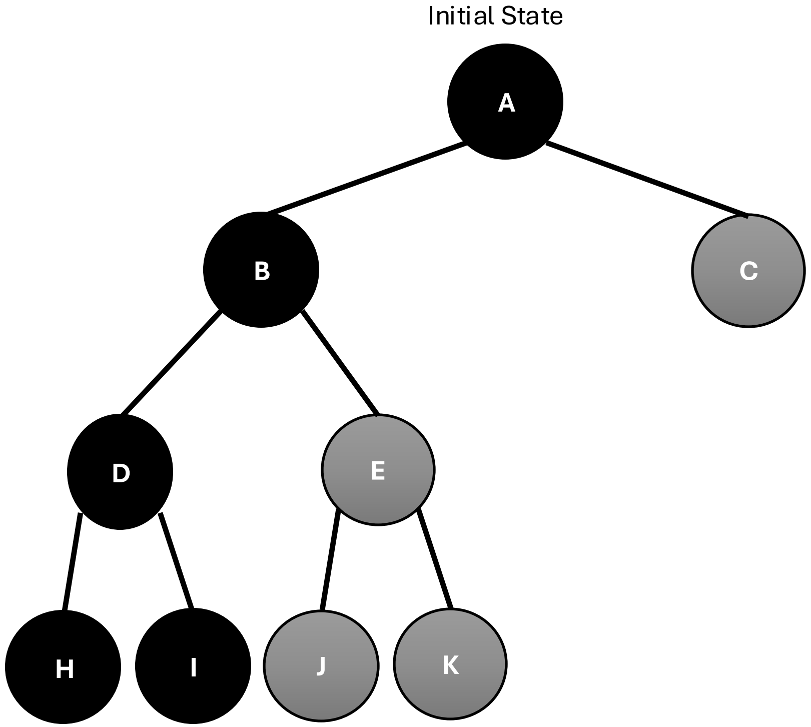
Depth-first search

Depth-first Strategy:

- Look at last node in queue.
- Is this node the goal state?
- If not, Expand node.

Fringe queue

C	E



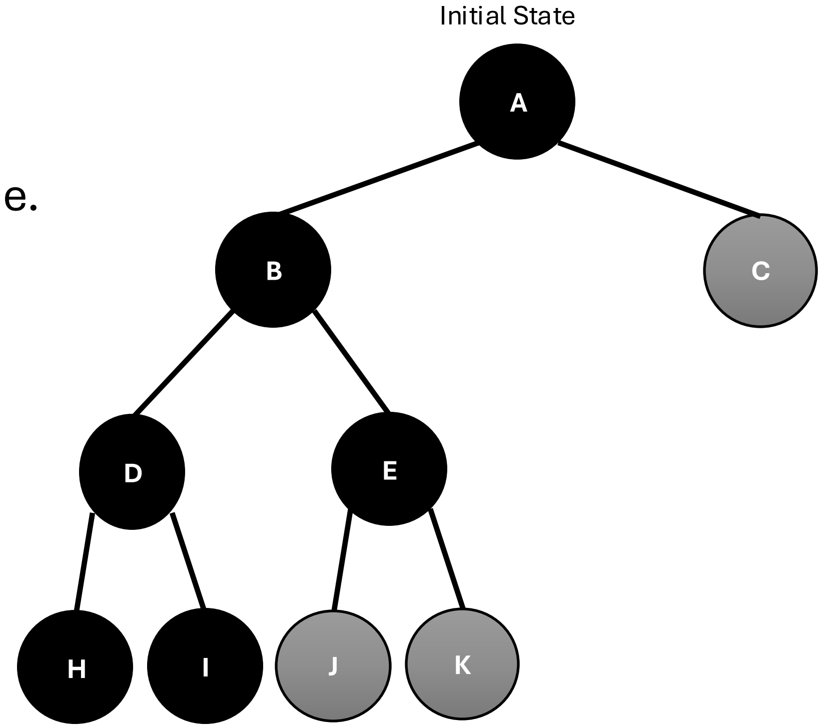
Depth-first search

Depth-first Strategy:

- Dequeue the expanded node.
- Add the generated nodes to queue.

Fringe queue

C	K	J



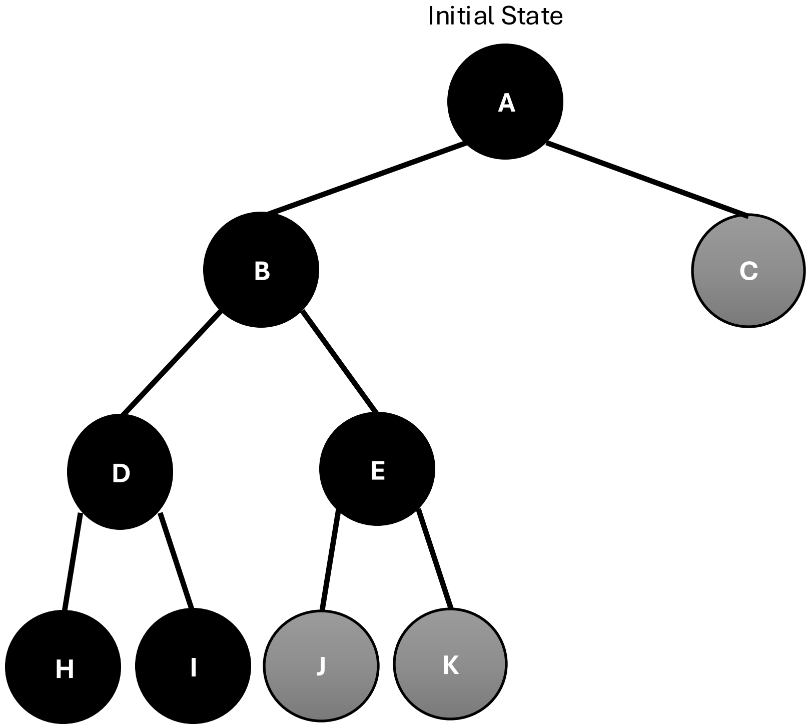
Depth-first search

Depth-first Strategy:

- Look at last node in queue.
- Is this node the goal state?
- If not, Expand node.

Fringe queue

C	K	J



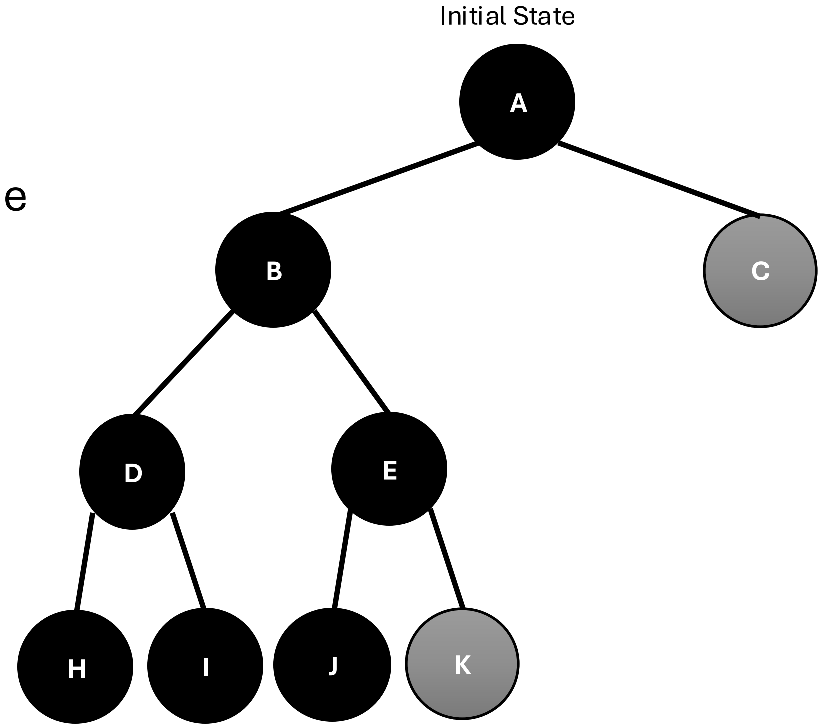
Depth-first search

Depth-first Strategy:

- Dequeue the expanded node.
- Add the generated nodes to queue

Fringe queue

C	K



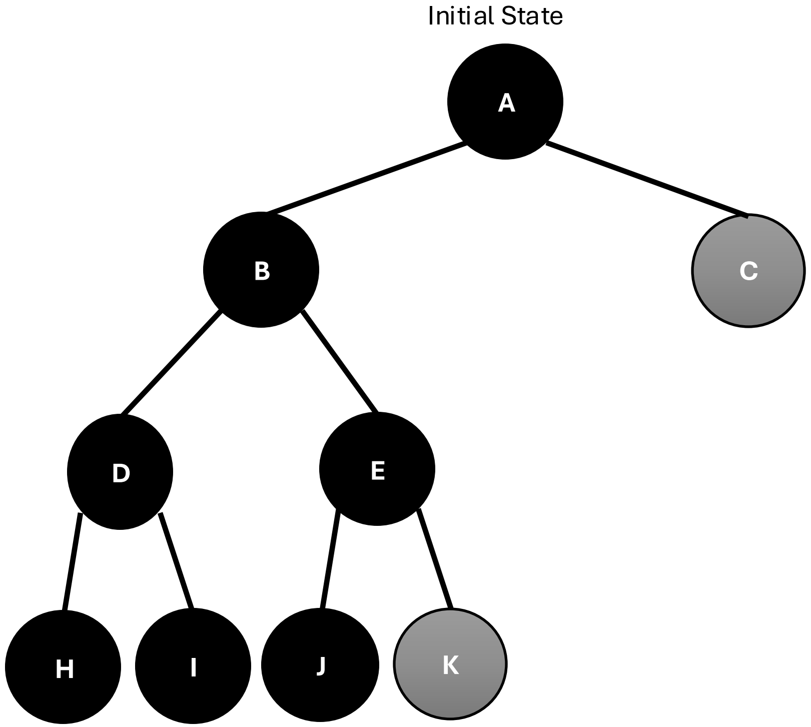
Depth-first search

Depth-first Strategy:

- Look at last node in queue.
- Is this node the goal state?
- If not, Expand node.

Fringe queue

C	K



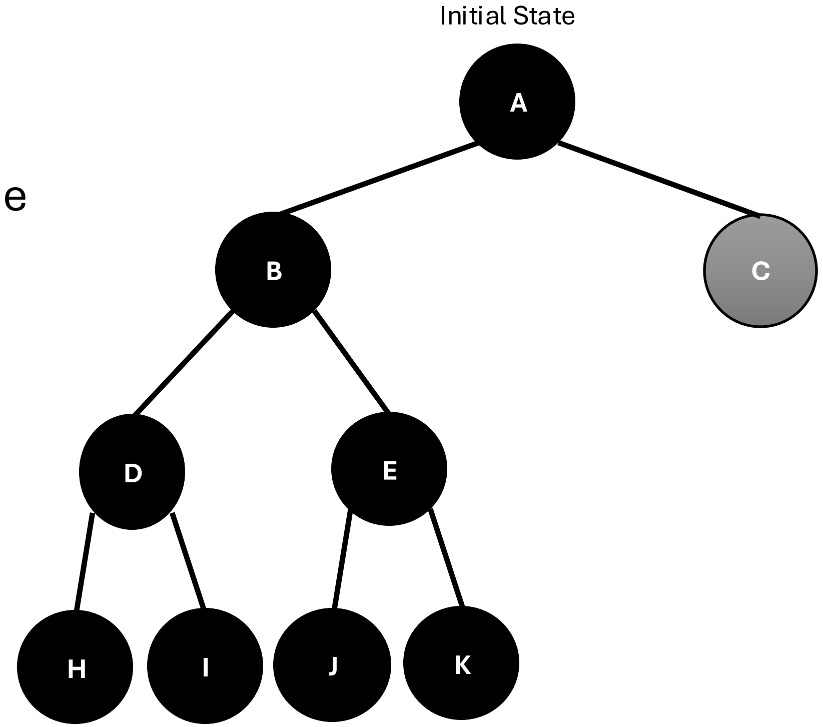
Depth-first search

Depth-first Strategy:

- Dequeue the expanded node.
- Add the generated nodes to queue

Fringe queue

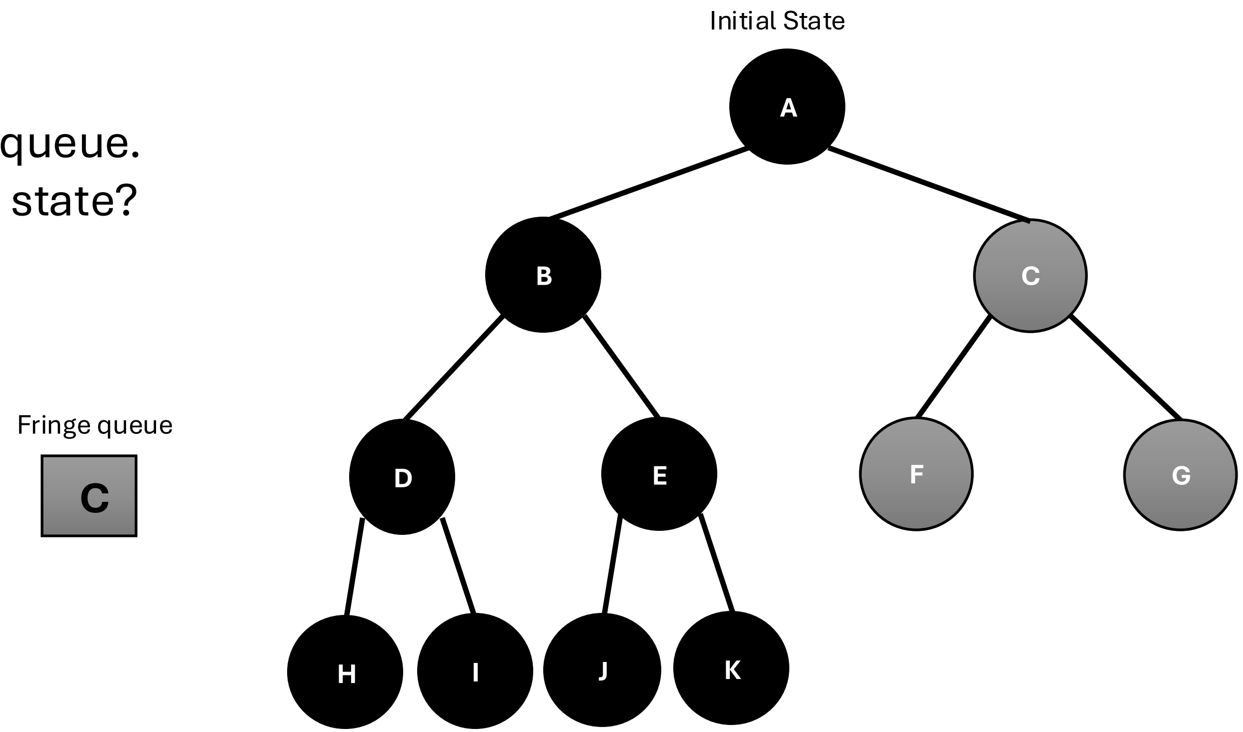
C



Depth-first search

Depth-first Strategy:

- Look at last node in queue.
- Is this node the goal state?
- If not, Expand node.



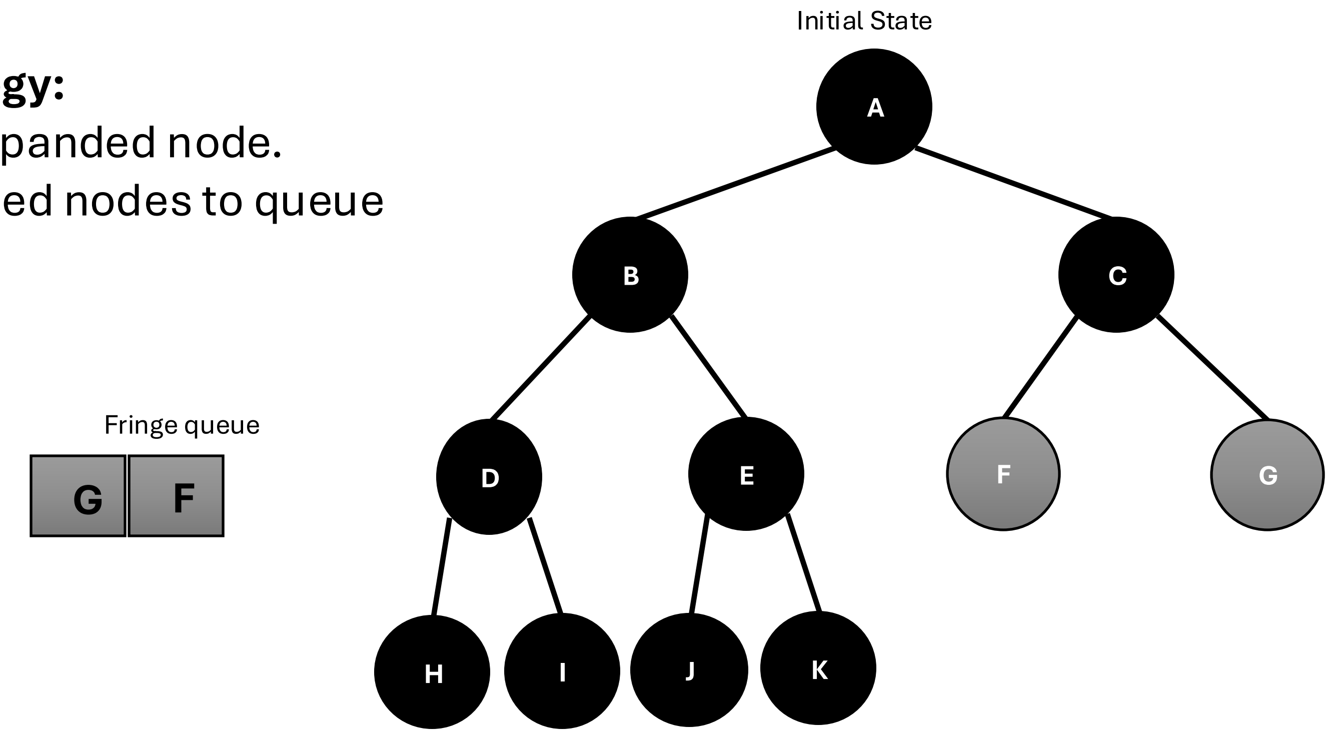
Fringe queue

C

Depth-first search

Depth-first Strategy:

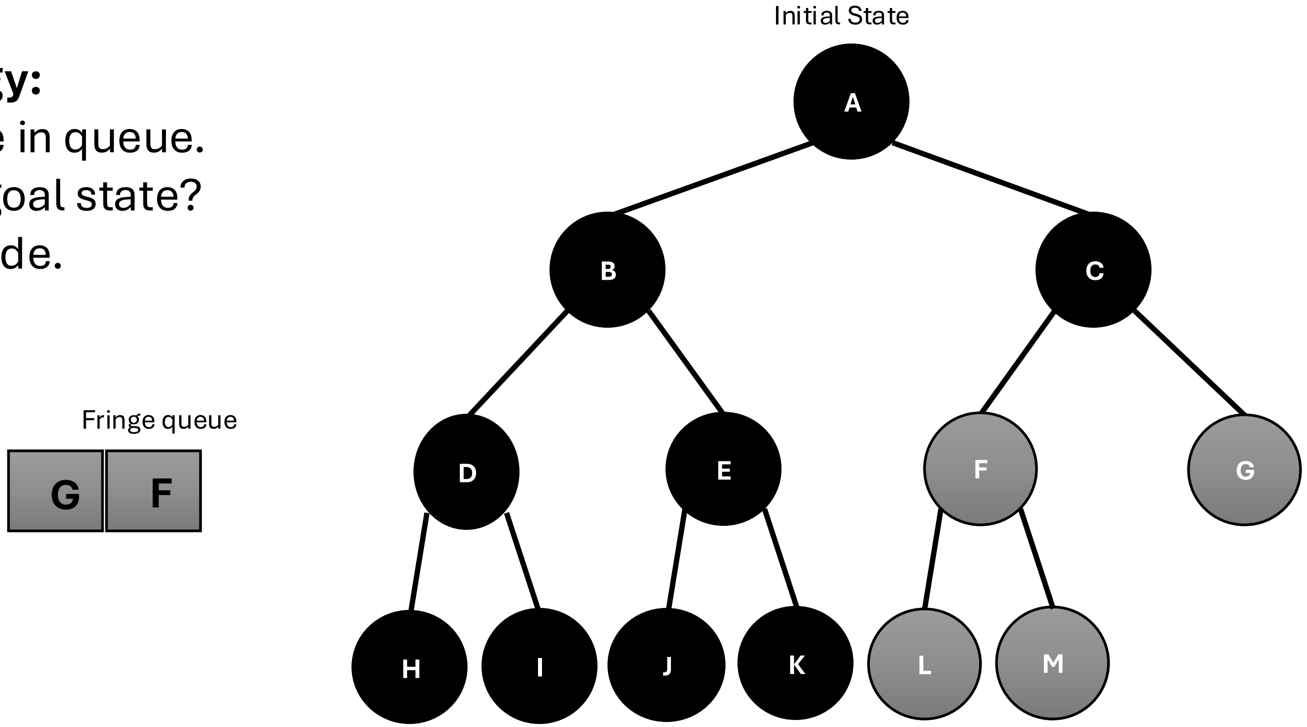
- Dequeue the expanded node.
- Add the generated nodes to queue



Depth-first search

Depth-first Strategy:

- Look at last node in queue.
- Is this node the goal state?
- If not, Expand node.


Initial State

Fringe queue

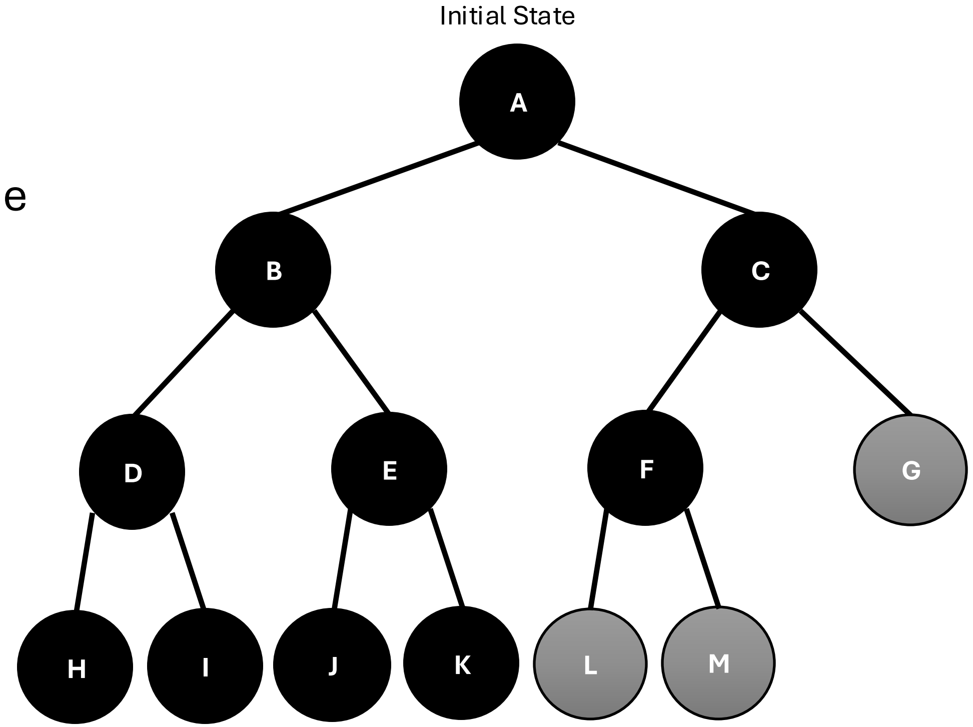
Depth-first search

Depth-first Strategy:

- Dequeue the expanded node.
- Add the generated nodes to queue

Fringe queue

G	M	L



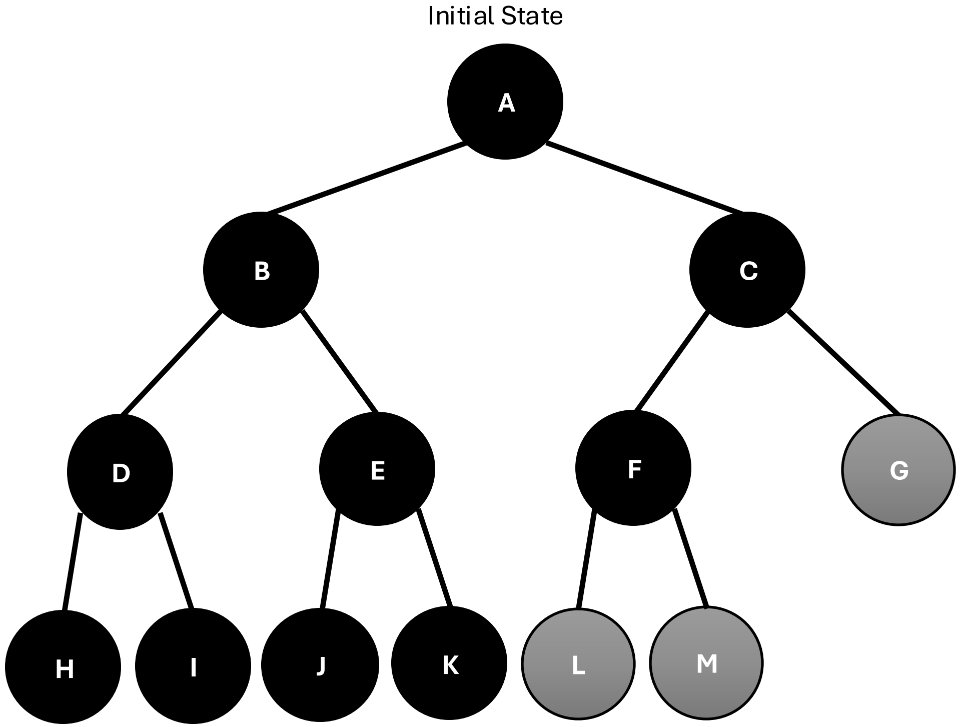
Depth-first search

Depth-first Strategy:

- Look at last node in queue.
- Is this node the goal state?
- If not, Expand node.

Fringe queue

G	M	L



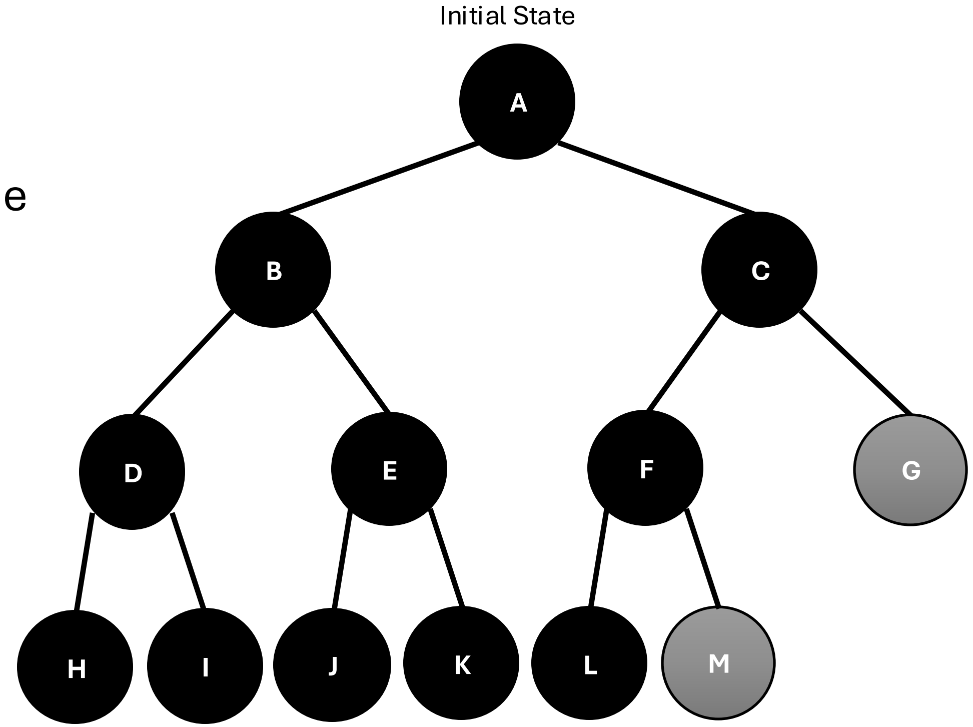
Depth-first search

Depth-first Strategy:

- Dequeue the expanded node.
- Add the generated nodes to queue

Fringe queue

G	M



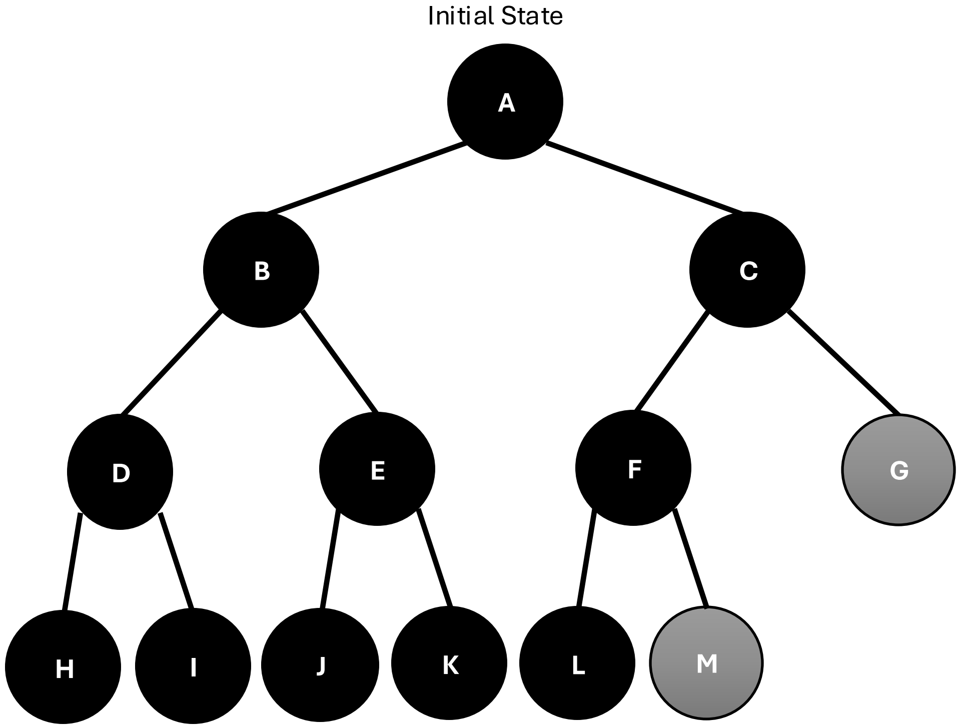
Depth-first search

Depth-first Strategy:

- Look at last node in queue.
- Is this node the goal state?
- If not, Expand node.

Fringe queue

G	M



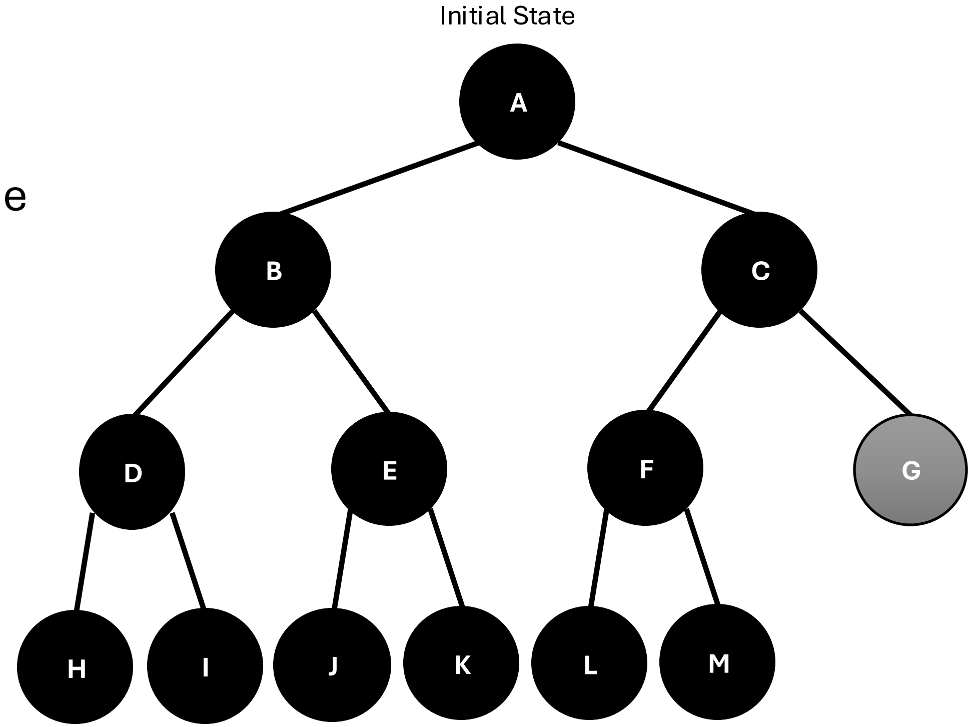
Depth-first search

Depth-first Strategy:

- Dequeue the expanded node.
- Add the generated nodes to queue

Fringe queue

G



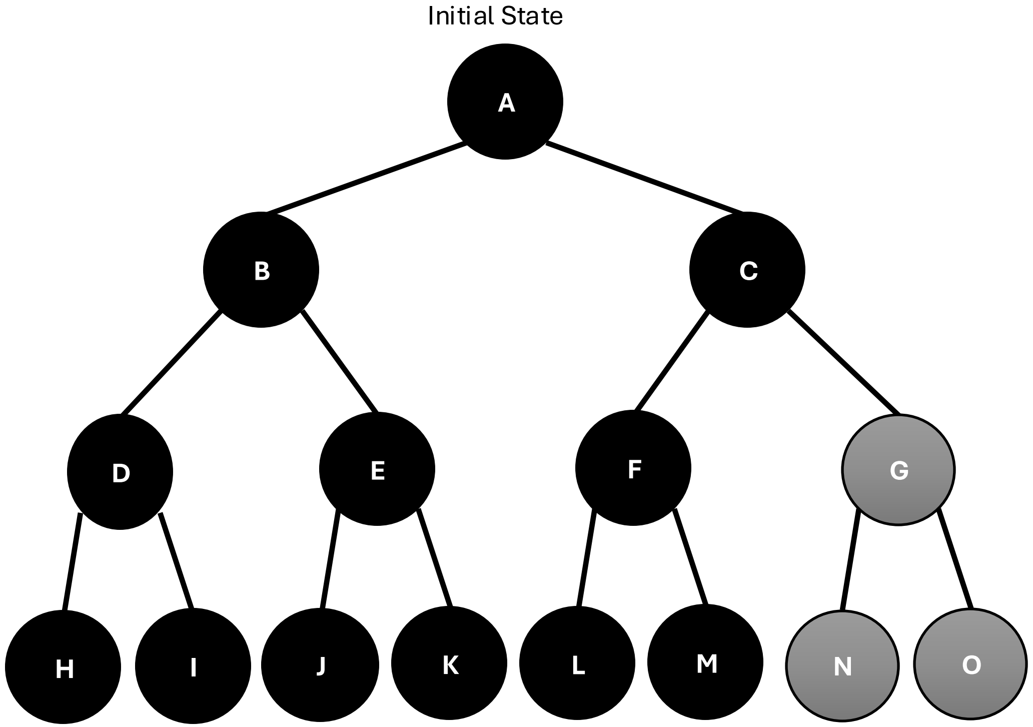
Depth-first search

Depth-first Strategy:

- Look at last node in queue.
- Is this node the goal state?
- If not, Expand node.

Fringe queue

G



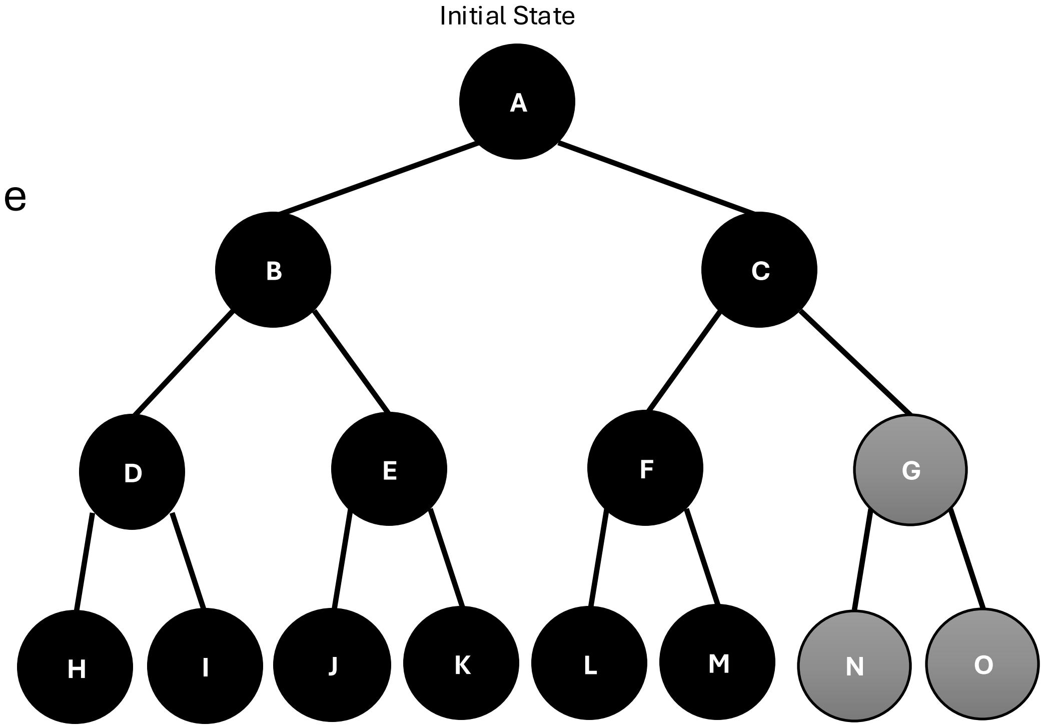
Depth-first search

Depth-first Strategy:

- Dequeue the expanded node.
- Add the generated nodes to queue

Fringe queue

G



Depth-first Search

- **Complete:** No: fails in infinite-depth spaces, spaces with loops
    - Complete in finite spaces
    - Complete only if m is finite
        - m is maximum depth of any node

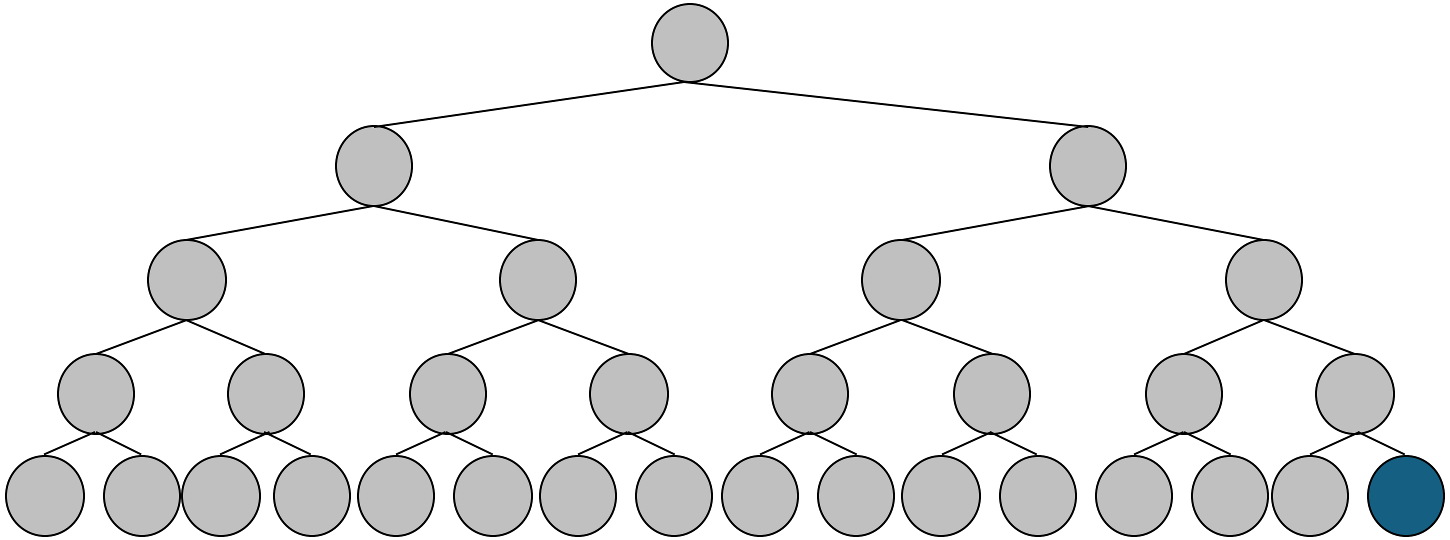
- **Optimal:** No

- **Time:** O(b^m): terrible if m is much larger than d,
    - d is depth of the least-cost solution
    - but if solutions are dense, may be much faster than breadth-first

- **Space:** O(bm), i.e., linear space
    - where b is maximum branching factor of the search tree

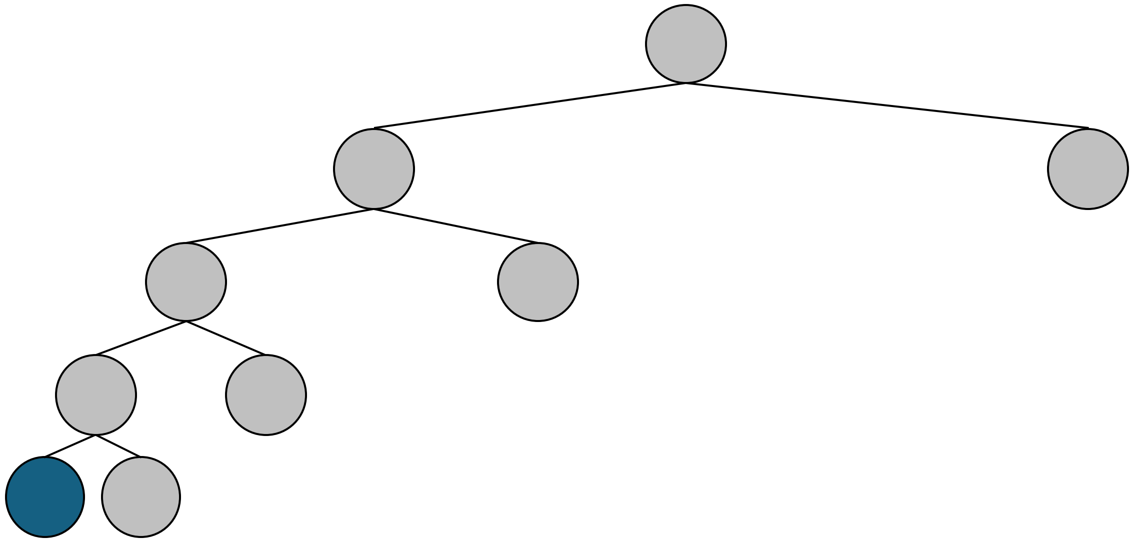
Depth-First Search

- Upper-bound case for time: goal is last node of last branch
  - Number of nodes generated: b nodes for each node of m levels (entire tree)
  - Time complexity: all generated nodes O(b^m)

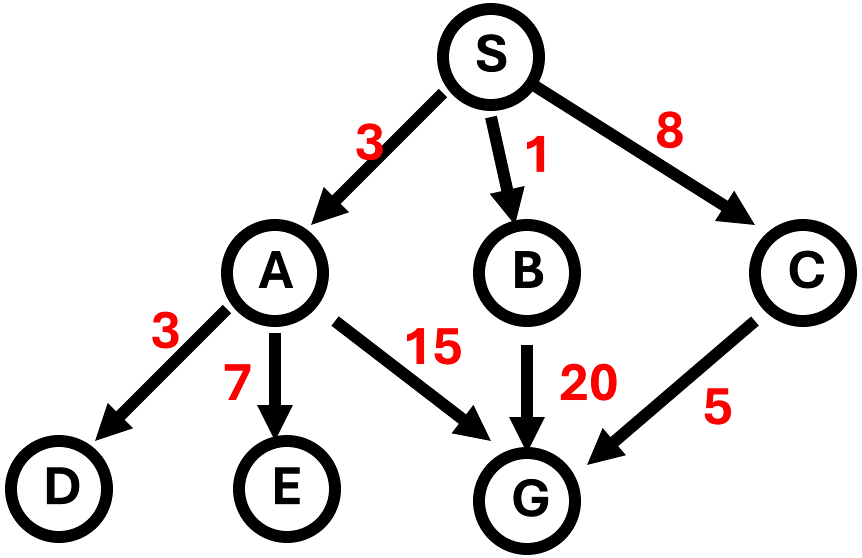


Depth-First Search

- Upper-bound case for space: goal is last node of first branch
  - After that, we start deleting nodes
  - Number of generated nodes: b nodes at each of m levels
  - Space complexity: all generated nodes = O(bm)

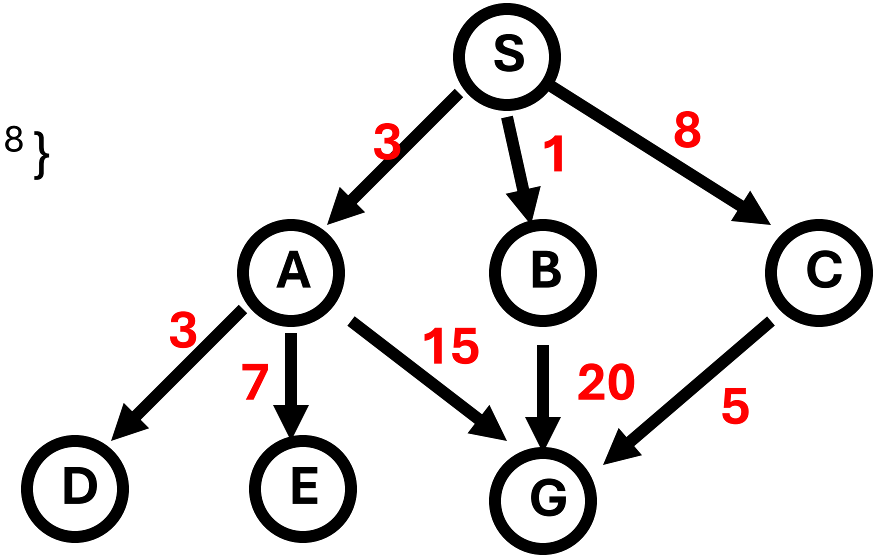


Depth-First Search example



Depth-First Search Example

Expanded node	Nodes list
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ D^6 E^{10} G^{18} B^1 C^8 }
D^6	{ E^{10} G^{18} B^1 C^8 }
E^{10}	{ G^{18} B^1 C^8 }
G^{18}	{ B^1 C^8 }



Solution path found is S A G, cost 18

Number of nodes expanded (including goal node) = 5

Uninformed Search Strategies

Depth-Limited Search

Depth-Limited Search

- Like the normal depth-first search, depth-limited search is an uninformed search.

- It works exactly like depth-first search, but avoids its drawbacks regarding completeness by imposing a maximum limit on the depth of the search.

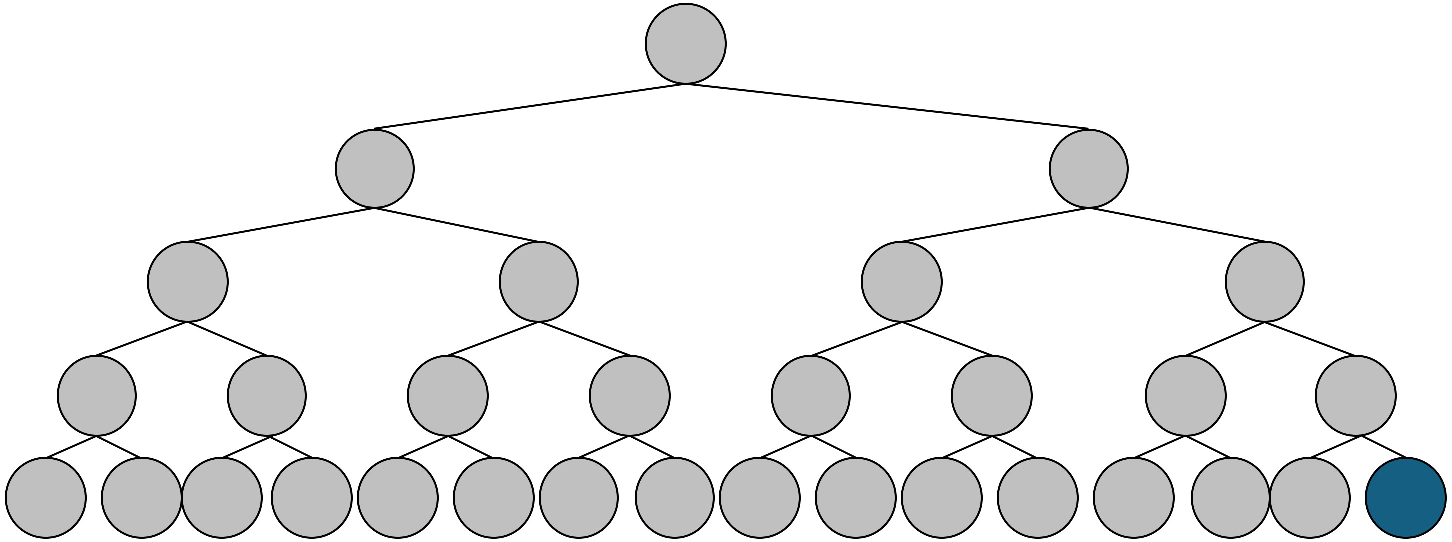
- Even if the search could still expand a vertex beyond that depth, it will not do so and thereby it will not follow infinitely deep paths or get stuck in cycles.

- Therefore depth-limited search will find a solution if it is within the depth limit, which guarantees at least completeness on all graphs

Depth-Limited Search

- Complete if there is a solution within depth bound d<=l
  - If l < d, i.e., the shallowest solution is deeper than the bound, DLS will fail even though a solution exists
  - Always terminates
- Optimal: No
- Space complexity is O(bl) where b is the branching factor and l is the depth bound.
- Time complexity is $O(b^l)$
- If l>>d, this can significantly increase the cost of the search compared to breadth-first search.

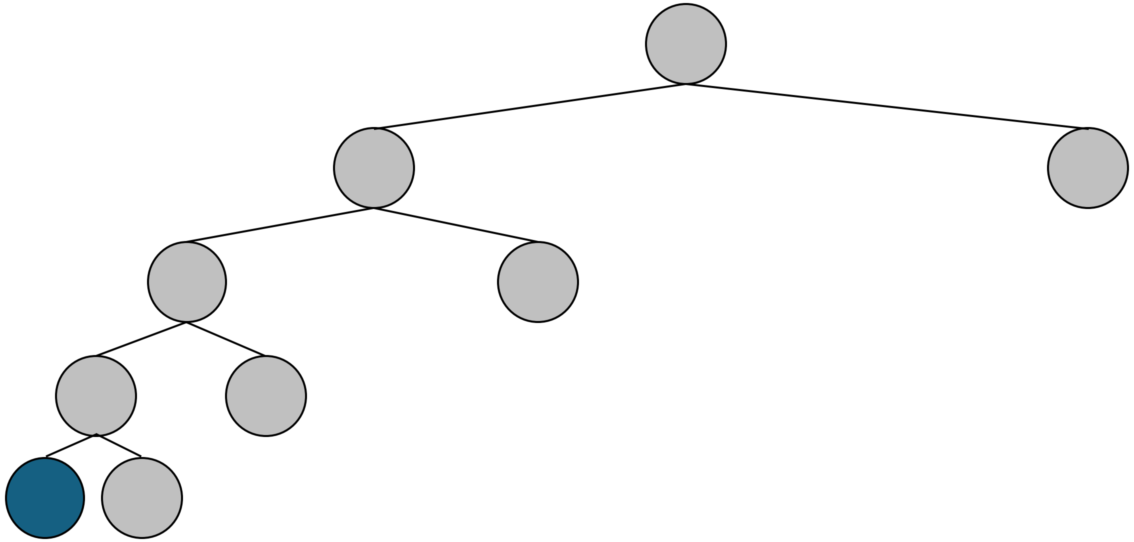
Depth-Limited Search



- Upper-bound case for time: goal is last node of last branch
  - Number of nodes generated:
    b nodes for each node of l levels (entire tree to depth l)
  - Time complexity: all generated nodes $O(b^l)$

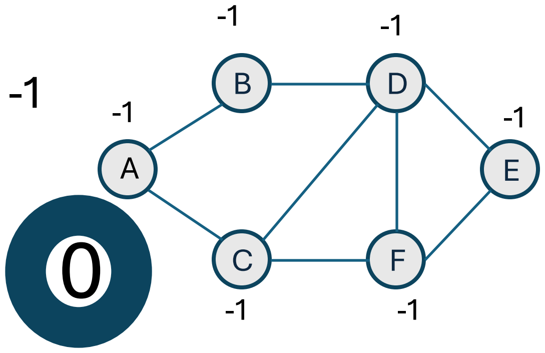
Depth-Limited Search

- Upper-bound case for space: goal is last node of first branch
  - After that, we start deleting nodes
  - Number of generated nodes: b nodes at each of l levels
  - Space complexity: all generated nodes = O(bl)

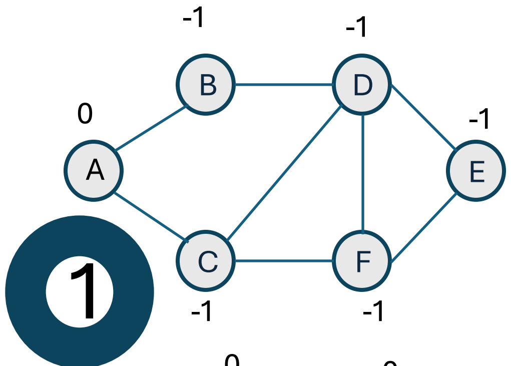


Enqueue a node only of status is -1

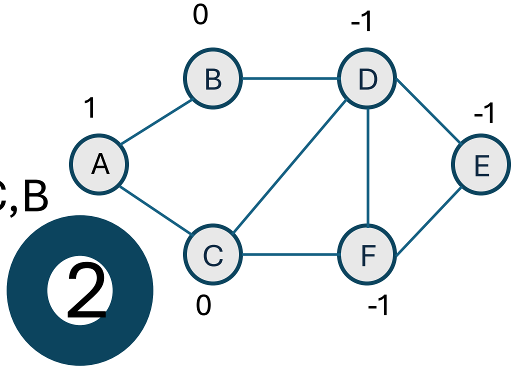
	A
A	CB
C	BFD
B	FD
F	DE
D	E
E	

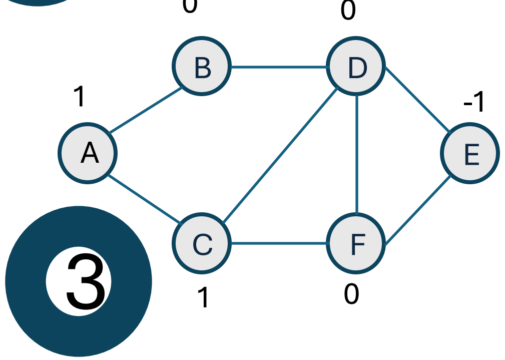


Enqueue A



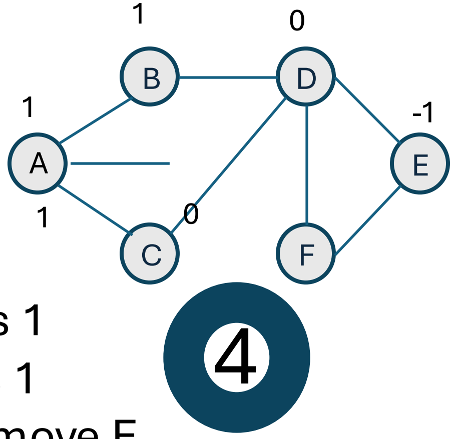
Enqueue C,B
Remove A

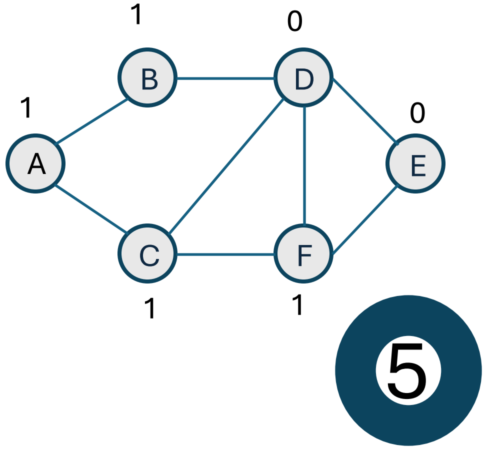




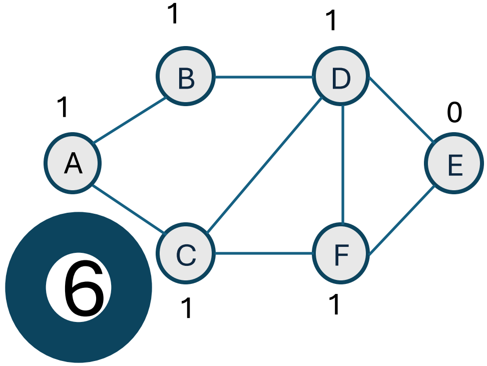
Enqueue FD
Remove C

D is zero
Remove B

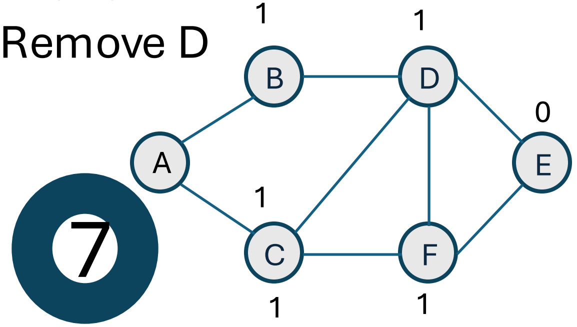




Enqueue E
D is zero
Remove F



F is 1
E is zero
Remove D



D is 1
F is 1
Remove E