Graph Traversals

Graph Traversals

- Two standard techniques
  - Depth-First Search (DFS)
  - Breadth-First Search (BFS)
- Also Topological Order

- Connectivity and other graph problems can be solved using *graph traversal* techniques

Graph Traversals

- Two standard techniques
  - Depth-First Search (DFS)
  - Breadth-First Search (BFS)
- Also Topological Order

- In both BFS and DFS, the nodes of an undirected graph are visited in a systematic manner so that every node is visited exactly once; both traversals can be performed on a directed graph
- Both BFS and DFS consider an embedded tree
  - When a node x is visited, it is labeled as visited, and it is added to the tree
  - If the traversal got to node x from node y, y is viewed as the parent of x, and x a child of y

Depth-First Search

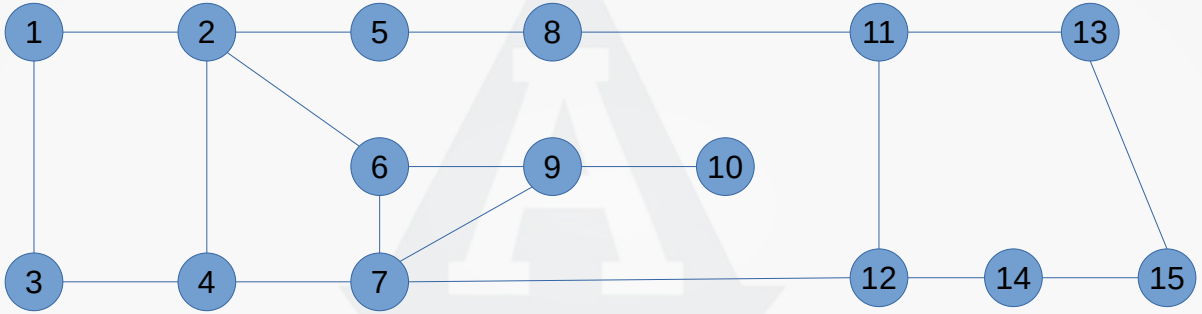
Graph Traversal – Depth-First Search

1. Select an unvisited node x, visit it, and treat as the *current node*
2. Find an unvisited neighbor of the current node, visit it, and make it the new current node
3. If the current node has no unvisited neighbors, backtrack to its parent, then make that parent the new current node
4. Repeat steps 3 and 4 until no more nodes can be visited
5. If there are still unvisited nodes, repeat from step 1
   - This will (potentially) create a forest of trees

Note: For a binary tree (a directed graph), this is what an in-order traversal does

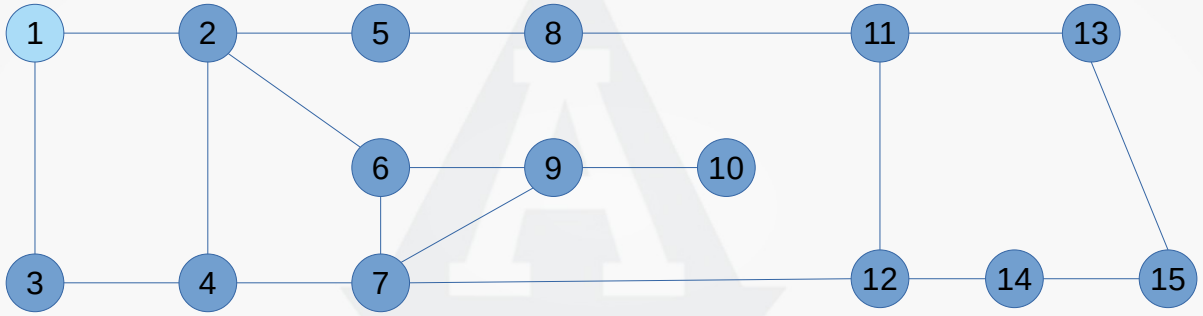
Depth-First Search – Example

- Let’s take a look at this graph and see what a DFS might look like



Depth-First Search – Example

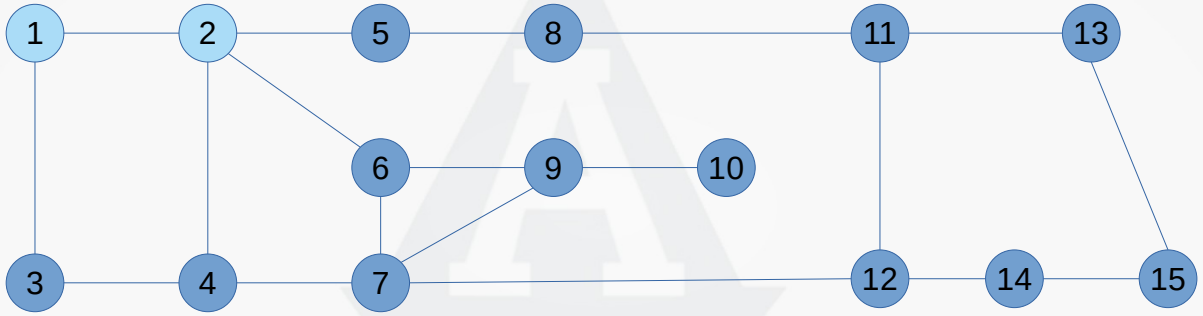
- Let’s take a look at this graph and see what a DFS might look like



- Start with 1

Depth-First Search – Example

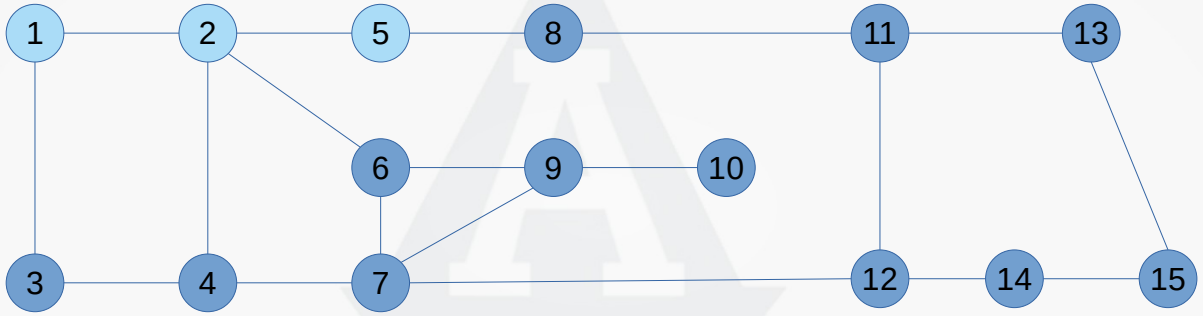
- Let’s take a look at this graph and see what a DFS might look like



- Start with 1, then visit 2

Depth-First Search – Example

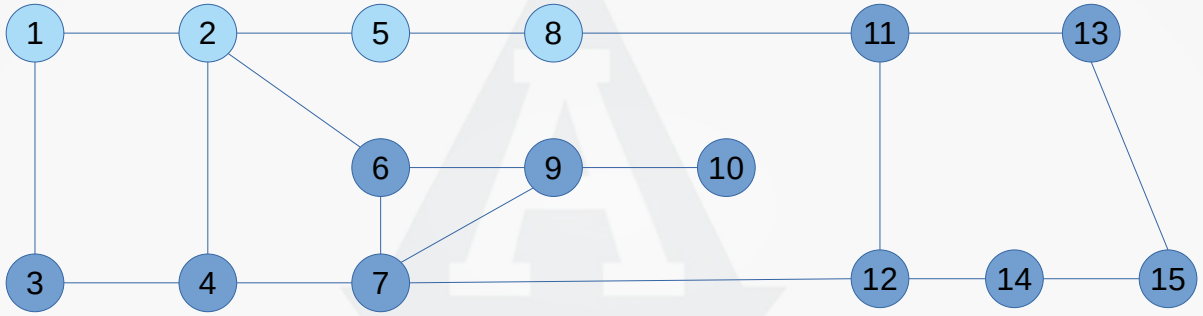
- Let’s take a look at this graph and see what a DFS might look like



- Start with 1, then visit 2, then visit 5

Depth-First Search – Example

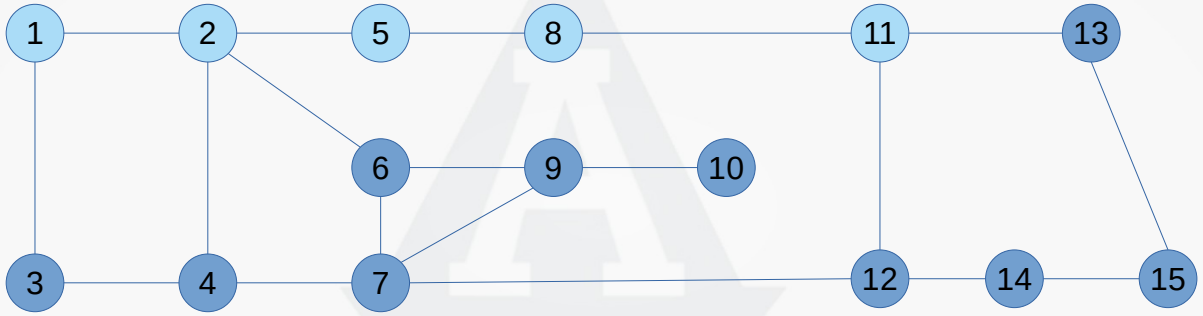
- Let’s take a look at this graph and see what a DFS might look like



- Start with 1, then visit 2, then visit 5, then visit 8

Depth-First Search – Example

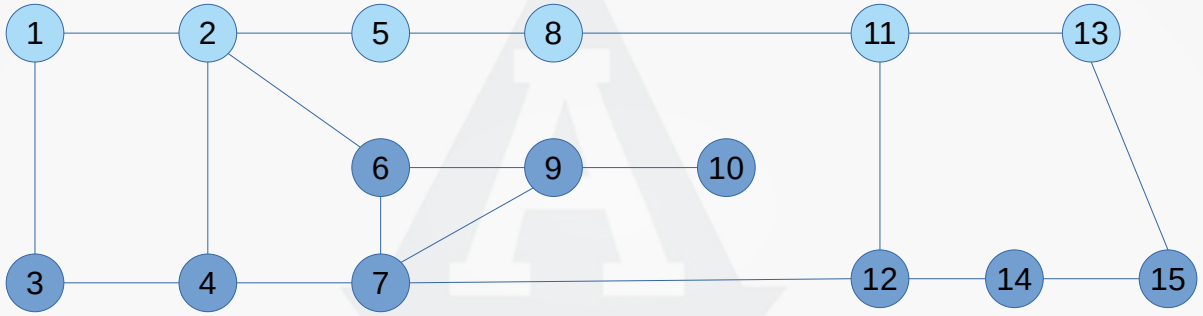
- Let’s take a look at this graph and see what a DFS might look like



- Start with 1, then visit 2, then visit 5, then visit 8, then visit 11

Depth-First Search – Example

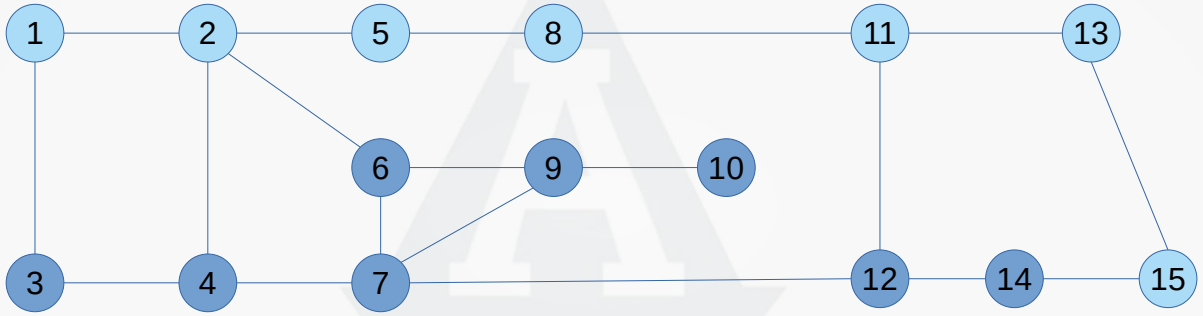
- Let’s take a look at this graph and see what a DFS might look like



- Start with 1, then visit 2, then visit 5, then visit 8, then visit 11, then visit 13

Depth-First Search – Example

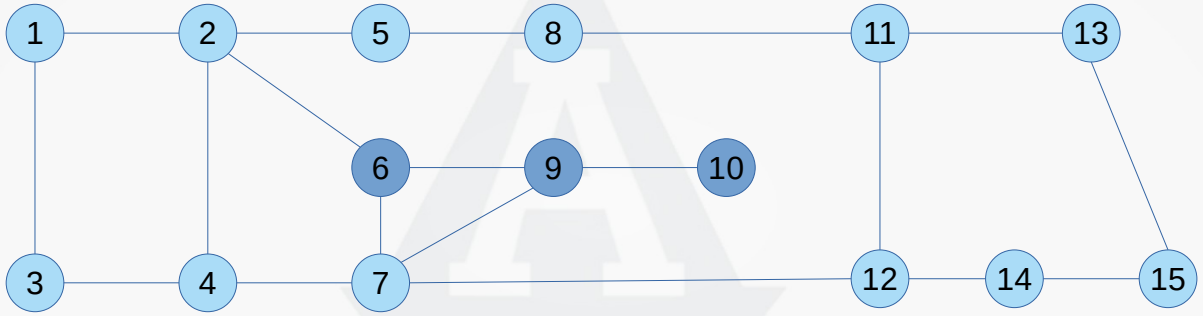
- Let’s take a look at this graph and see what a DFS might look like



- Start with 1, then visit 2, then visit 5, then visit 8, then visit 11, then visit 13, then visit 15

Depth-First Search – Example

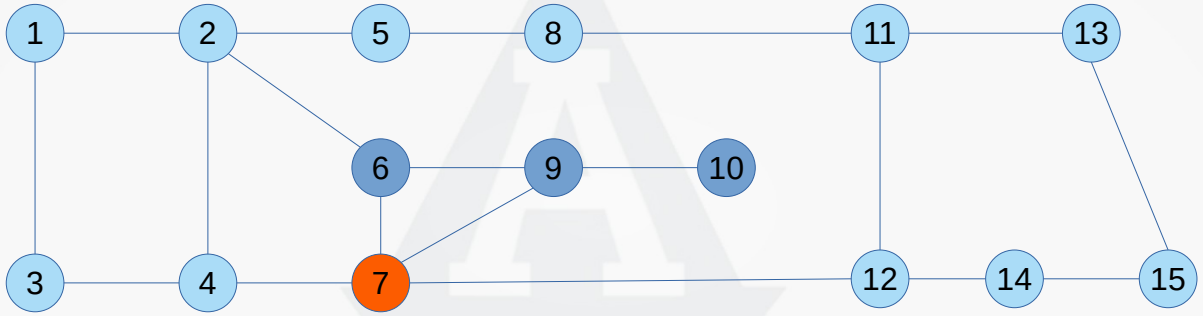
- Let’s take a look at this graph and see what a DFS might look like



- Start with 1, then visit 2, then visit 5, then visit 8, then visit 11, then visit 13, then visit 15
- Then visit 14, then visit 12, then visit 7, then visit 4, then visit 3

Depth-First Search – Example

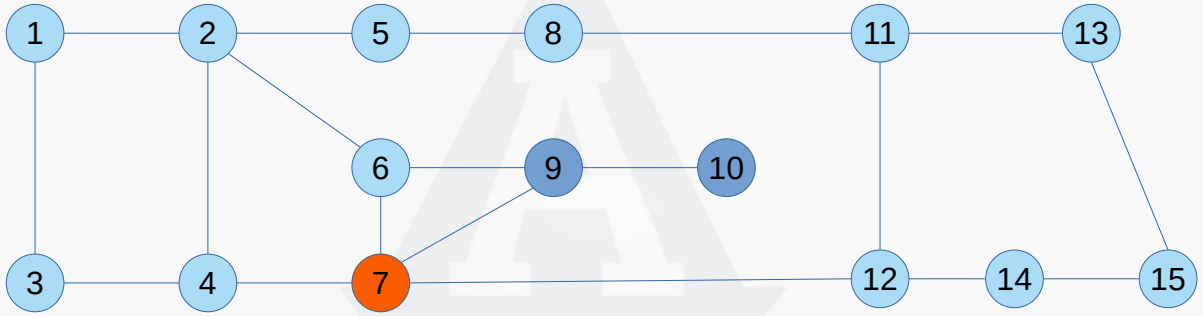
- Let’s take a look at this graph and see what a DFS might look like



- Start with 1, then visit 2, then visit 5, then visit 8, then visit 11, then visit 13, then visit 15
- Then visit 14, then visit 12, then visit 7, then visit 4, then visit 3
- *Backtrack to 7*

Depth-First Search – Example

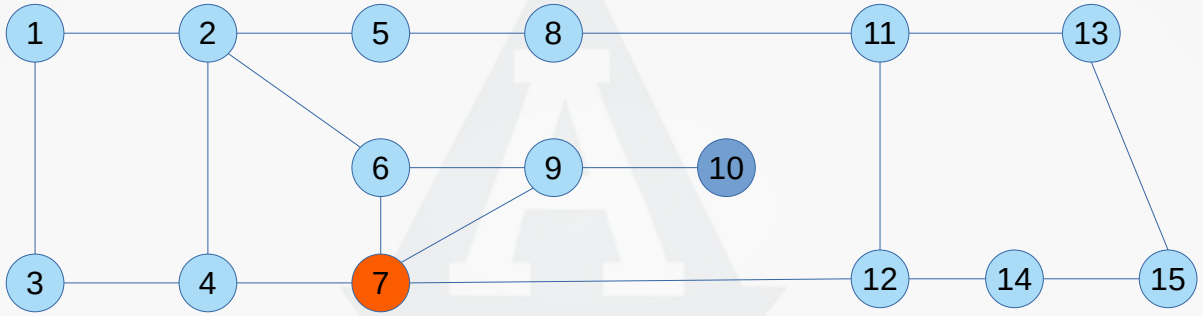
- Let's take a look at this graph and see what a DFS might look like



- Start with 1, then visit 2, then visit 5, then visit 8, then visit 11, then visit 13, then visit 15
- Then visit 14, then visit 12, then visit 7, then visit 4, then visit 3
- *Backtrack to 7*, then visit 6

Depth-First Search – Example

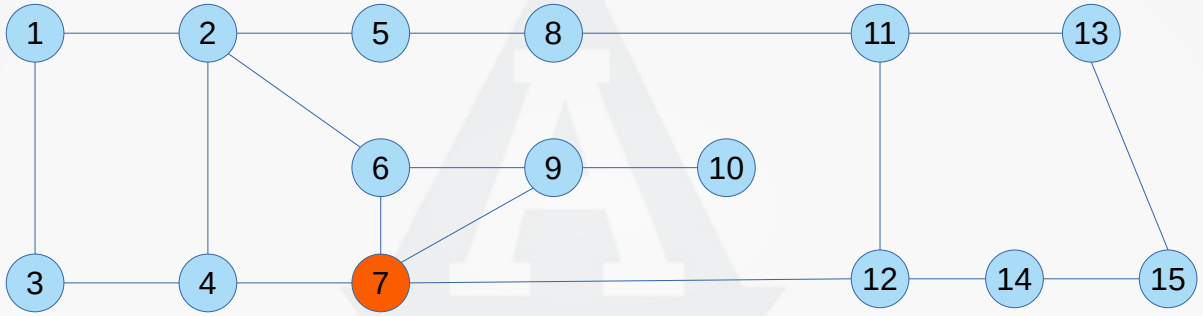
- Let’s take a look at this graph and see what a DFS might look like



- Start with 1, then visit 2, then visit 5, then visit 8, then visit 11, then visit 13, then visit 15
- Then visit 14, then visit 12, then visit 7, then visit 4, then visit 3
- *Backtrack to 7*, then visit 6, then visit 9

Depth-First Search – Example

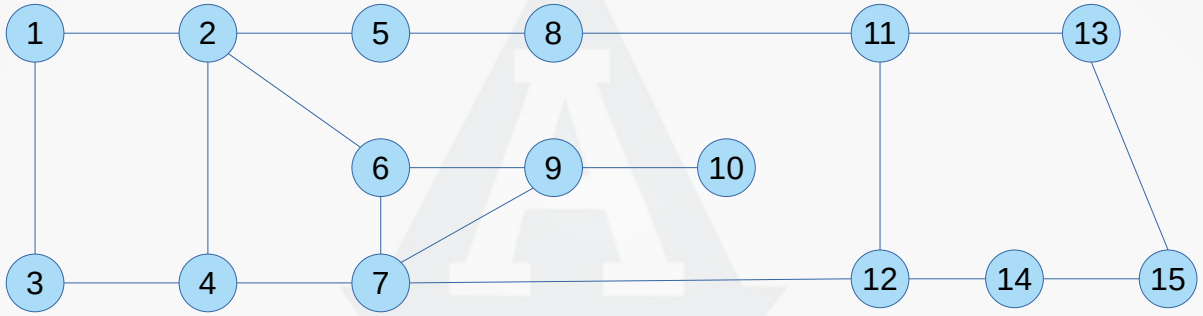
- Let’s take a look at this graph and see what a DFS might look like



- Start with 1, then visit 2, then visit 5, then visit 8, then visit 11, then visit 13, then visit 15
- Then visit 14, then visit 12, then visit 7, then visit 4, then visit 3
- *Backtrack to 7*, then visit 6, then visit 9, then visit 10

Depth-First Search – Example

- Let’s take a look at this graph and see what a DFS might look like



- Start with 1, then visit 2, then visit 5, then visit 8, then visit 11, then visit 13, then visit 15
- Then visit 14, then visit 12, then visit 7, then visit 4, then visit 3
- *Backtrack to 7*, then visit 6, then visit 9, then visit 10
- Backtrack all the way out, all nodes have been visited!

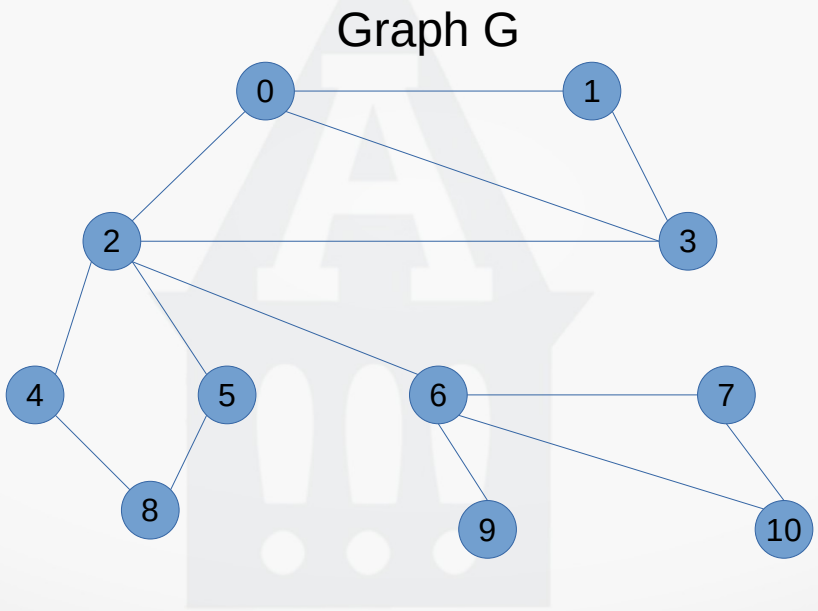
Breadth-First Search

Graph Traversal – Breadth-First Search

1. Select an unvisited node x, visit it, have it be the root in a BFS tree being formed
   - Its level is called the current level
2. Try all one-step extensions of current paths before trying larger extensions
3. Repeat step 2 until no more nodes can be visited
4. If there are still unvisited nodes, repeat from step 1
   - This will (potentially) create a forest of trees

Breadth-First Search – Example

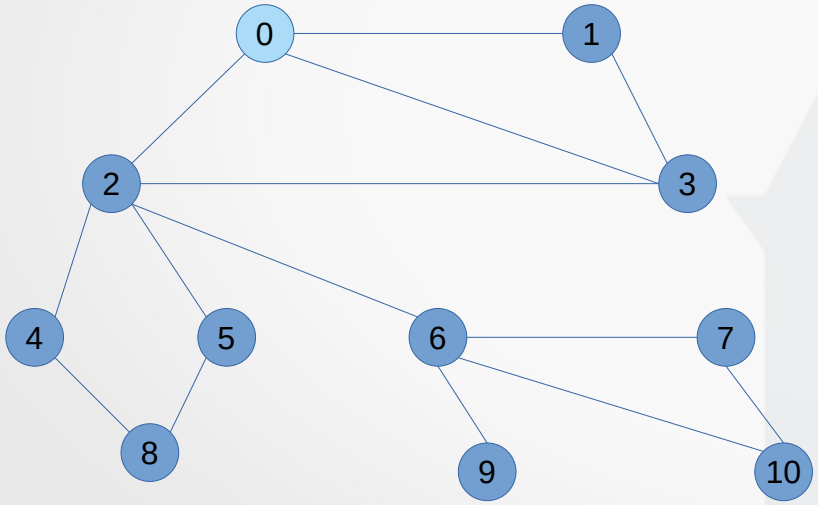
- Given this graph, perform a BFS traversal



Breadth-First Search – Example

- Start at 0, and visit all the nodes 1 edge away

Graph G

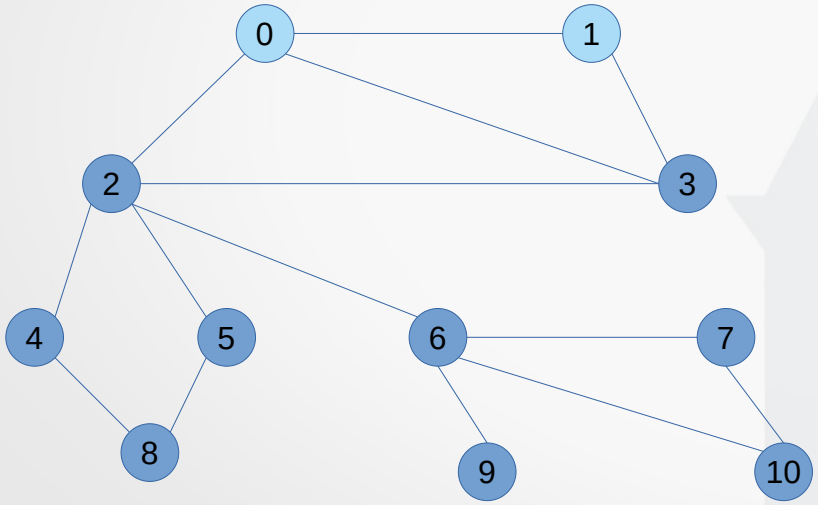


BFS Traversal Tree

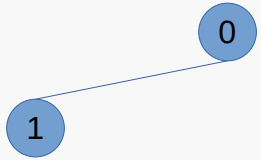
Breadth-First Search – Example

- Start at 0, and visit all the nodes 1 edge away

Graph G



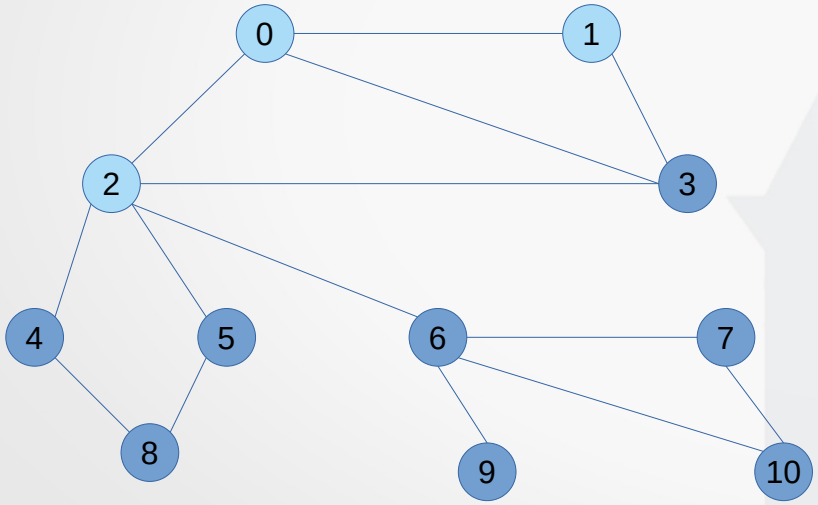
BFS Traversal Tree



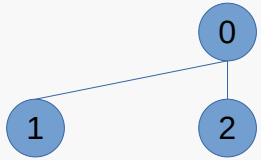
Breadth-First Search – Example

- Start at 0, and visit all the nodes 1 edge away

Graph G



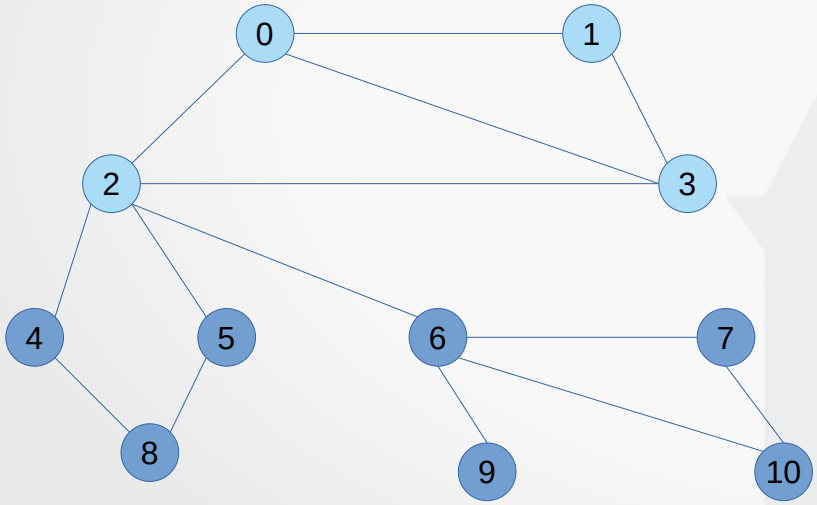
BFS Traversal Tree



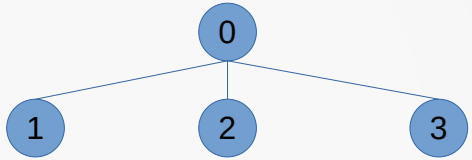
Breadth-First Search – Example

- Start at 0, and visit all the nodes 1 edge away

Graph G



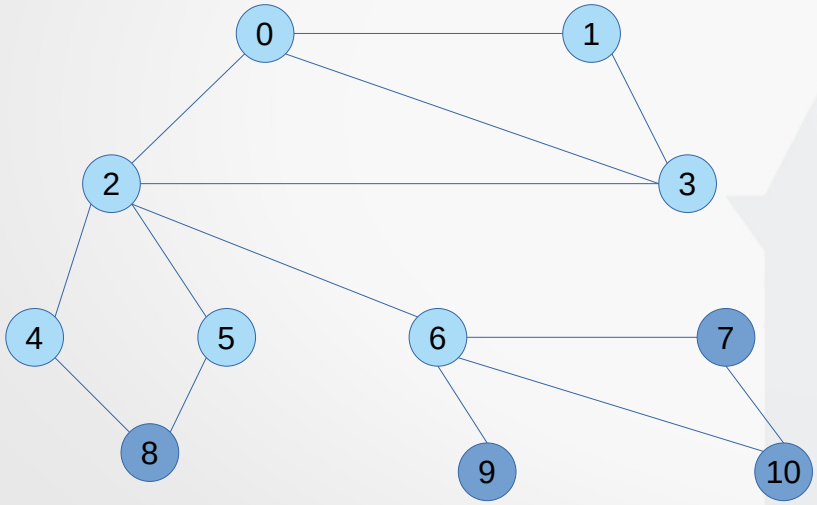
BFS Traversal Tree



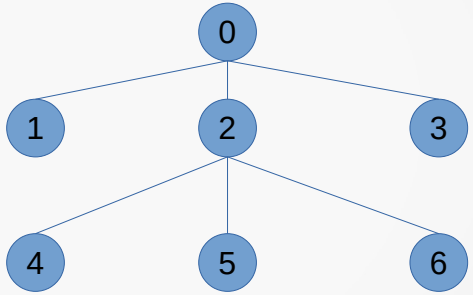
Breadth-First Search – Example

- Start at 0, and visit all the nodes 1 edge away
- Then visit all the nodes 2 edges away

Graph G



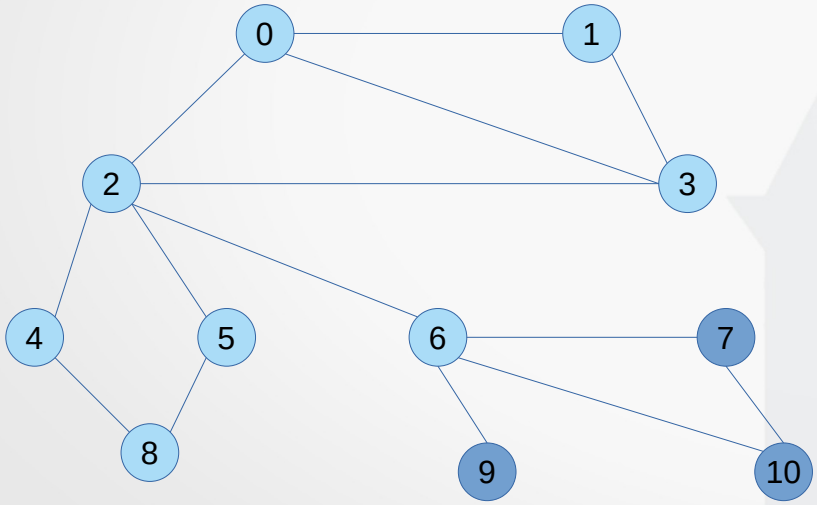
BFS Traversal Tree



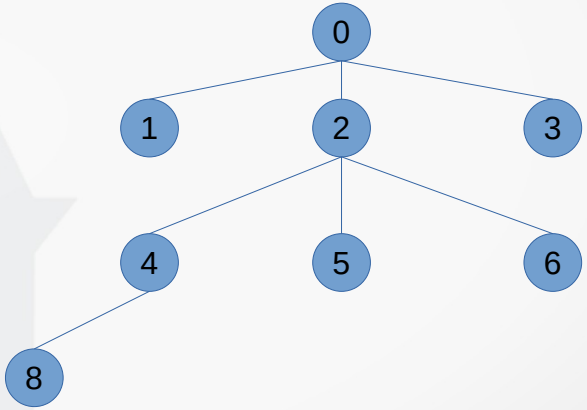
Breadth-First Search – Example

- Start at 0, and visit all the nodes 1 edge away
- Then visit all the nodes 2 edges away
- Then 3...

Graph G



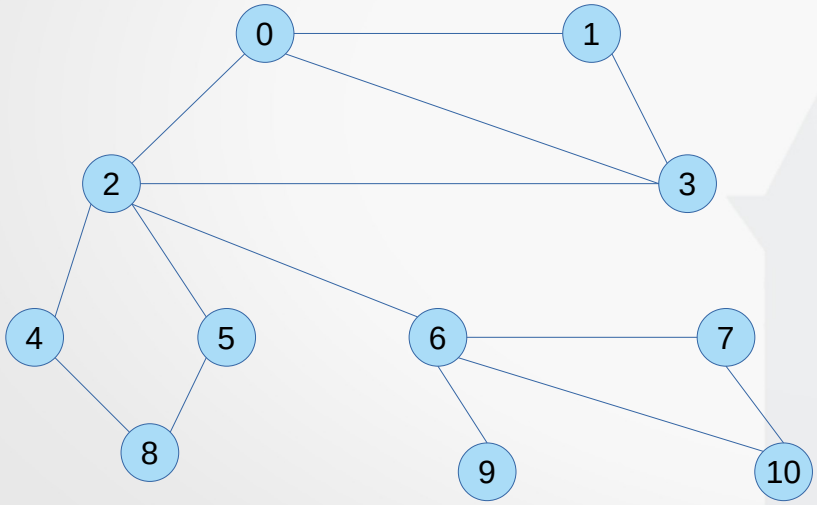
BFS Traversal Tree



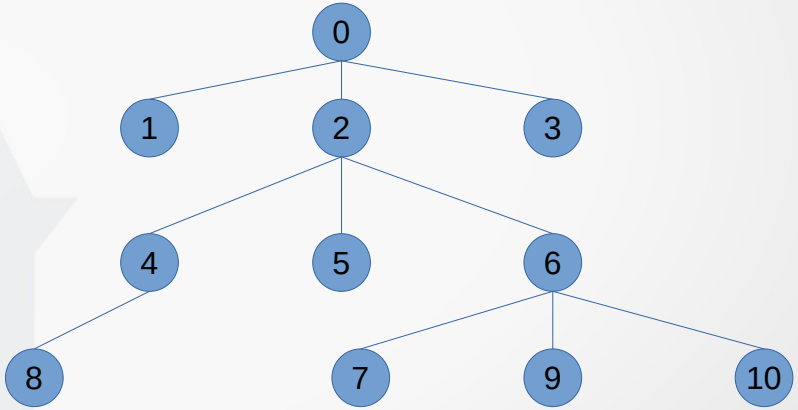
Breadth-First Search – Example

- Start at 0, and visit all the nodes 1 edge away
- Then visit all the nodes 2 edges away
- Then 3...

Graph G

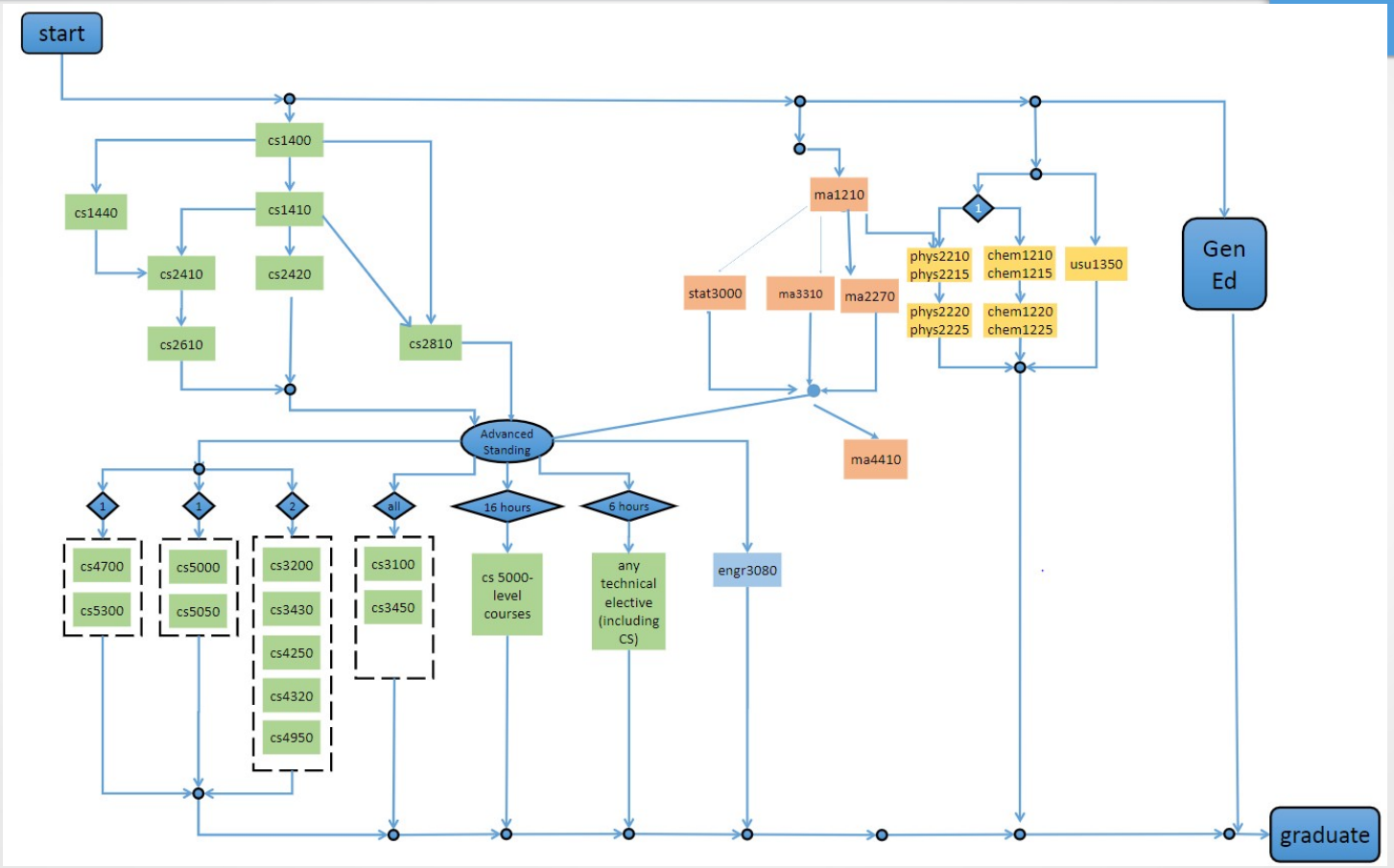


BFS Traversal Tree



Topological Ordering

Topological Ordering



Topological Ordering

- Let G be a directed graph and V(G) = { v_1, v_2, … v_n }, where n > 0
- We assume the graph has no cycles (DAG); because the graph has no cycles
  - There exists a vertex u in G such that u has no predecessor (start)
  - There exists a vertex v in G such that v has no successor (end)

- A ***topological ordering*** of V(G) is a linear ordering v_j, v_k, … v_n of the vertices such that if v_j is a predecessor of v_k, $j \neq k$, then v_j precedes v_k, that is, $j < k$ in this linear ordering

Topological Ordering – Algorithm

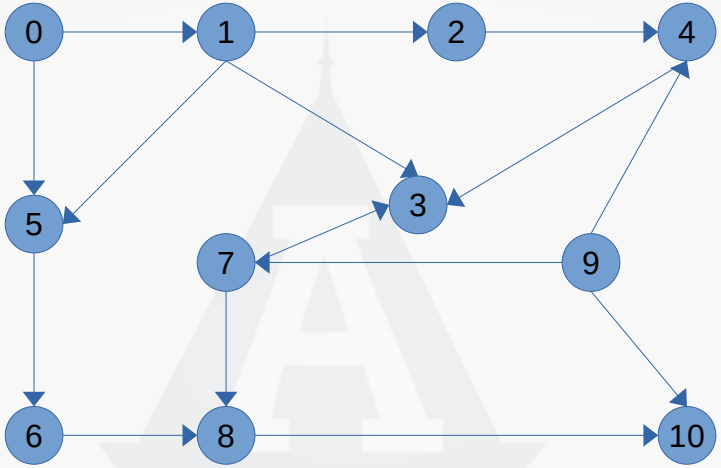
- Find a vertex that has no predecessor and make it first in the topological ordering
- Next, find the vertex v, all of whose predecessors have been placed in the topological ordering and place v next in the topological ordering

- Implementation details
  - To keep track of the number of vertices of a vertex, use an array (e.g., `predecessorCount`)
  - Initially `predecessorCount[j]` is the number of predecessors of the vertex v_j
  - The container (a FIFO queue) used to guide the breadth first traversal is initialized to those vertices v_k, where `predecessorCount[k]` is 0

Topological Ordering – Pseudocode

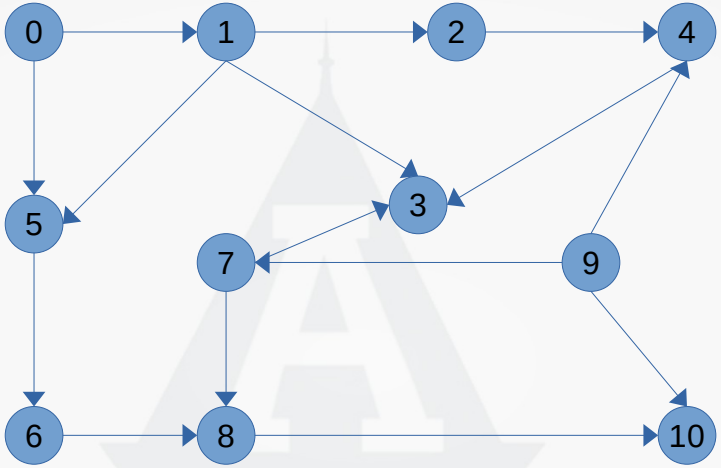
```
Count the number of predecessors of each node
For each node with a count of 0, enqueue the node
While queue is not empty
     current node = dequeue node
     report the node // this prints the nodes in topological order
     For each successor of current node
          subtract one from its predecessor count
          if the predecessor count is 0, enqueue the node
```

Topological Ordering – Example



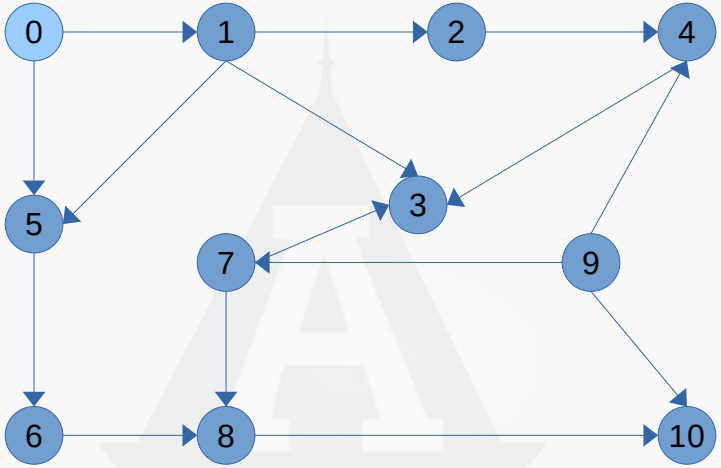
	0	1	2	3	4	5	6	7	8	9	10
`predecessorCount`	0	1	1	3	2	2	1	1	2	0	2
topological order											
queue											

Topological Ordering – Example



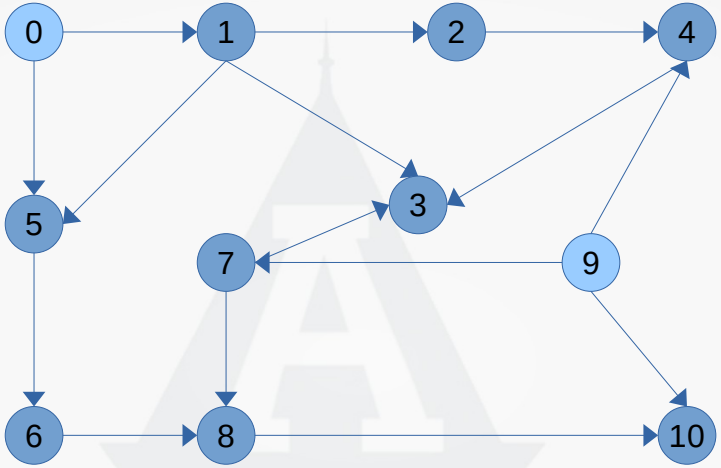
	0	1	2	3	4	5	6	7	8	9	10
`predecessorCount`	0	1	1	3	2	2	1	1	2	0	2
topological order											
queue	0	9									

Topological Ordering – Example



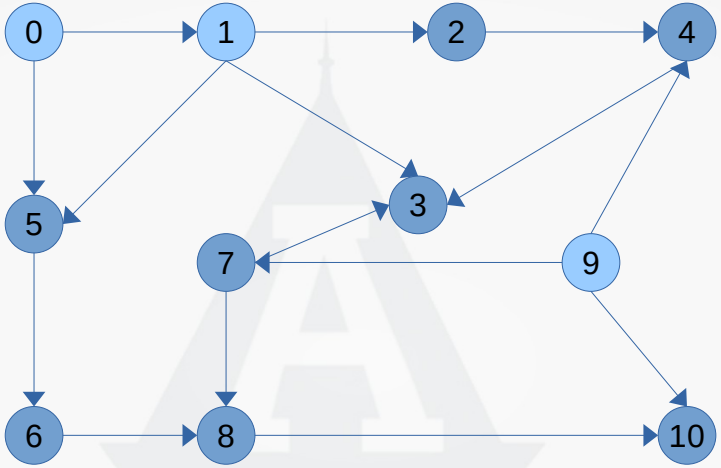
	0	1	2	3	4	5	6	7	8	9	10
`predecessorCount`	0	**0**	1	3	2	**1**	1	1	2	0	2
topological order	0										
queue	9	1									

Topological Ordering – Example



	0	1	2	3	4	5	6	7	8	9	10
`predecessorCount`	0	0	1	3	**1**	1	1	**0**	2	0	**1**
topological order	0	9									
queue	1	7									

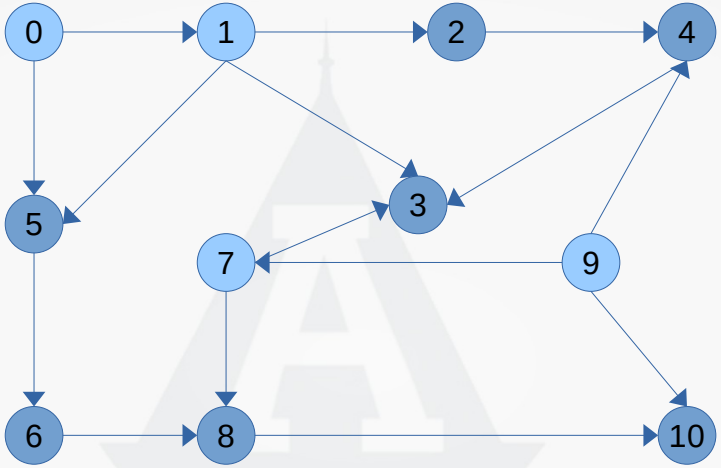
Topological Ordering – Example



`predecessorCount`	0	1	2	3	4	5	6	7	8	9	10
	0	0	**0**	**2**	1	**0**	1	0	2	0	1

topological order	0	9	1								
queue	7	2	5								

Topological Ordering – Example

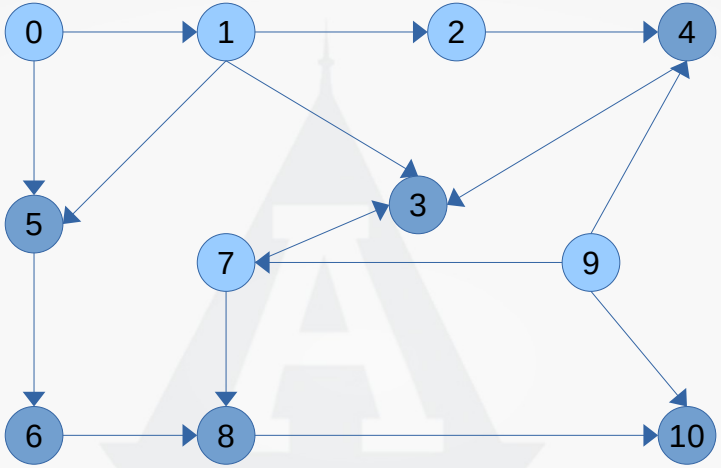


`predecessorCount`	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	**1**	1	0	1	0	**1**	0	1

topological order											
	0	9	1	7							

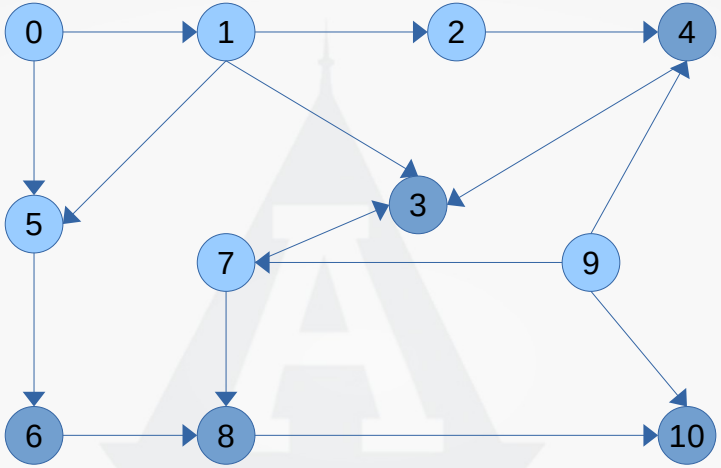
queue											
	2	5									

Topological Ordering – Example



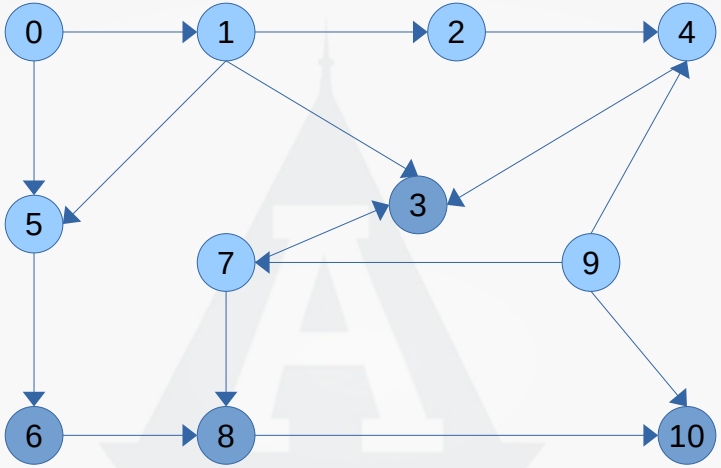
	0	1	2	3	4	5	6	7	8	9	10
`predecessorCount`	0	0	0	1	**0**	0	1	0	1	0	1
topological order	0	9	1	7	2						
queue	5	4									

Topological Ordering – Example



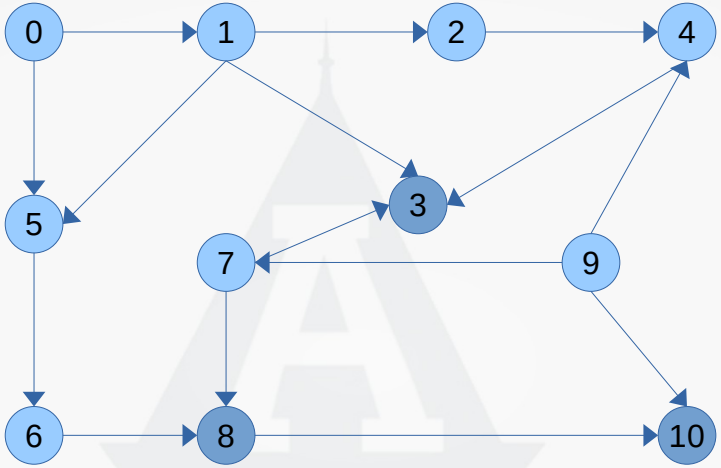
	0	1	2	3	4	5	6	7	8	9	10
`predecessorCount`	0	0	0	1	0	0	**0**	0	1	0	1
topological order	0	9	1	7	2	5					
queue	4	6									

Topological Ordering – Example



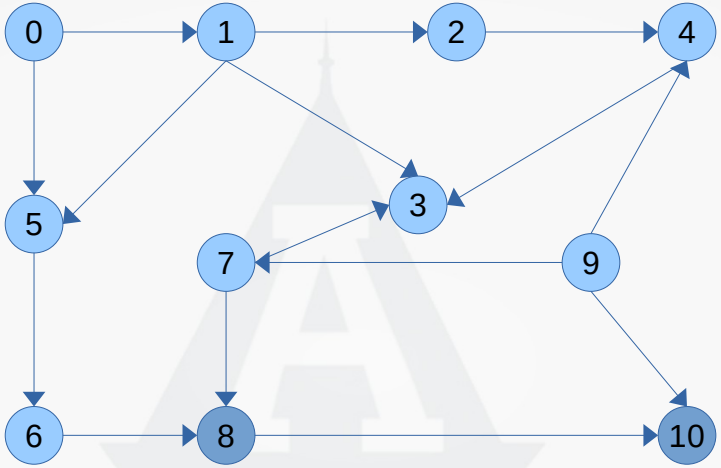
	0	1	2	3	4	5	6	7	8	9	10
`predecessorCount`	0	0	0	**0**	0	0	0	0	1	0	1
topological order	0	9	1	7	2	5	4				
queue	6	3									

Topological Ordering – Example



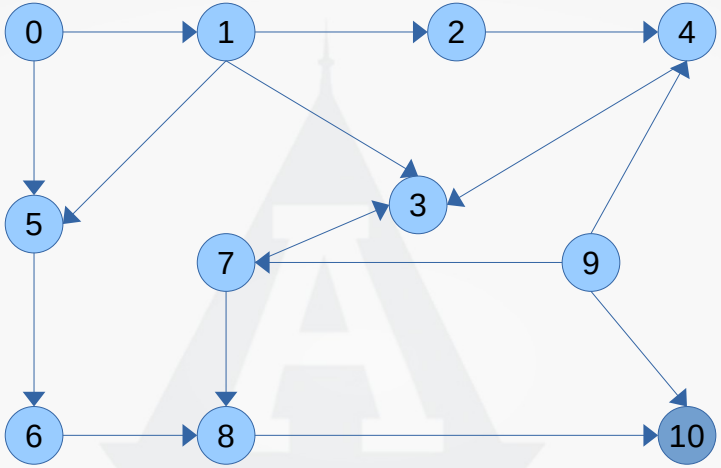
`predecessorCount`	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	**0**	0	1
topological order	0	9	1	7	2	5	4	6			
queue	3	8									

Topological Ordering – Example



	0	1	2	3	4	5	6	7	8	9	10
`predecessorCount`	0	0	0	0	0	0	0	0	0	0	1
topological order	0	9	1	7	2	5	4	6	3		
queue	8										

Topological Ordering – Example

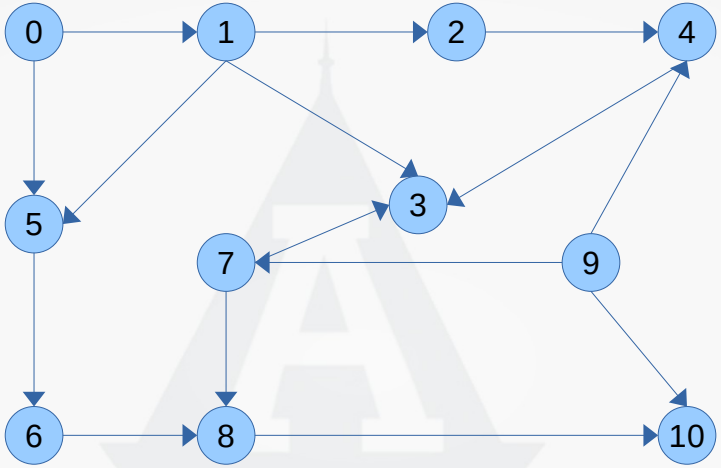


`predecessorCount`	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	**0**

topological order	0	9	1	7	2	5	4	6	3	8	

queue	10										

Topological Ordering – Example



	0	1	2	3	4	5	6	7	8	9	10
`predecessorCount`	0	0	0	0	0	0	0	0	0	0	0
topological order	0	9	1	7	2	5	4	6	3	8	10
queue											