Topological Sort

CSE 2011
Winter 2007

1

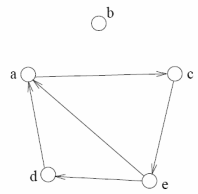
Directed Graphs

- A graph is directed if direction is assigned to each edge.
- A directed edge (arc) is an ordered pair (u,v)
- The adjacency matrix and adjacency list representations can be used for directed graphs.

2

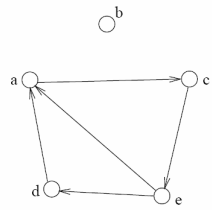
Representations

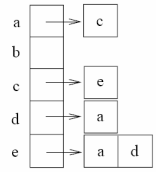
1. *Adjacency Matrix*



	a	b	c	d	e
a	0	0	1	0	0
b	0	0	0	0	0
c	0	0	0	0	1
d	1	0	0	0	0
e	1	0	0	1	0

2. *Adjacency List*





3

Directed Acyclic Graph

- A directed *path* is a sequence of vertices $(v_0, v_1, \ldots, v_k)$ such that (v_i, v_{i+1}) is an *arc*
- A directed *cycle* is a directed path such that the first and last vertices are the same.
- A directed graph is *acyclic* if it does not contain any directed cycles.

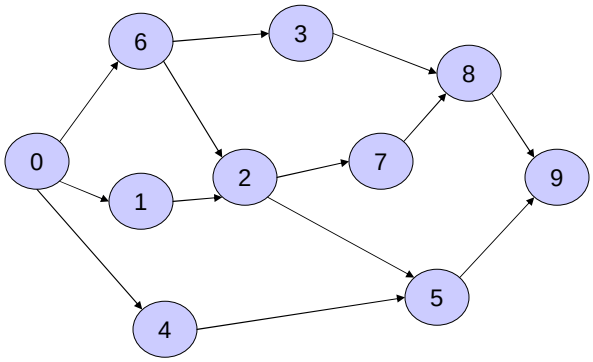
4

Calculating Indegrees and Outdegrees

- Calculating the degree of a vertex in an undirected graph:
  - adjacency matrix O ( )
  - adjacency list O ( )
- Outdegree is simple to compute
  - Scan through the adjacency list of vertex *v* and count the arcs
    - Running time: O( ) for a vertex
    - Running time: O(V + E) for all vertices
  - Note: If we keep the sizes of the lists, then the outdegree of a vertex can be obtained in O(1) time
- Indegree calculation
  - First, initialize indegree[*v*] = 0 for every vertex *v*
  - Scan through the adjacency list of every vertex *v*
    - For each vertex *w* seen, increment indegree[*w*]
    - Running time: O(*V* + *E*)
  - Note: running time for computing indegree of one vertex *v* = O ( )

5

Example



Indeg(2)?

Indeg(8)?

Outdeg(0)?

Num of Edges?

Total OutDeg?

Total Indeg?

6

Directed Graphs

- Directed graphs are often used to represent order-dependent tasks
  - That is we cannot start a task before another task finishes
- We can model this task dependent constraint using *arcs*
- An *arc (i,j)* means *task j* cannot start until *task i* is finished

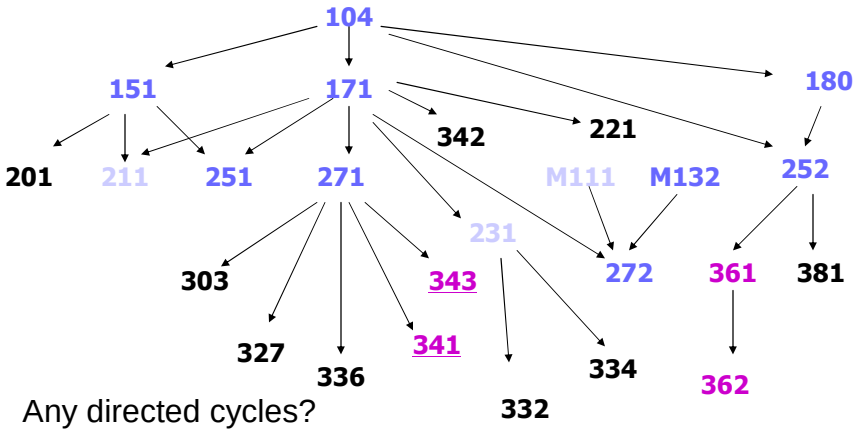
Task j cannot start
until task i is finished

- Clearly, for the system not to hang, the graph must be acyclic

7

An Example

- Course structure

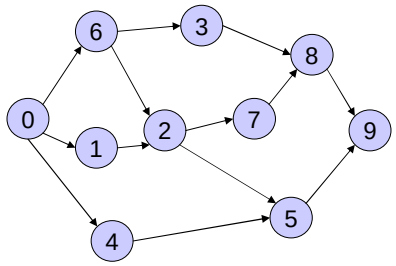


Any directed cycles?
How many indeg(171)?
How many outdeg(171)?

8

Topological Sort

- Topological sort is an algorithm for directed acyclic graphs
- Ordering the vertices so that the linear order respects the ordering relations implied by the edges



For example:

0, 1, 2, 5, 9
0, 4, 5, 9
0, 6, 3, 7 ?

9

Topological Sort Algorithm

- Observations
  - Starting point must have zero indegree
  - If it doesn't exist, the graph would not be acyclic
- Algorithm

1. A vertex with zero *indegree* is a task that can start right away. So we can output it first in the linear order
2. If a vertex *i* is output, then its outgoing edges *(i, j)* are no longer useful, since tasks *j* do not need to wait for *i* anymore. So remove all *i*'s outgoing arcs, and decrement the indegrees of tasks *j.*
3. With vertex *i* removed, the new graph is still a directed acyclic graph. So, repeat steps 1 and 2 until no vertex is left.

10

Topological Sort

Algorithm $TSort(G)$
Input: a directed acyclic graph G
Output: a topological ordering of vertices

```
initialize Q to be an empty queue;
for each vertex v
    do if indegree(v) = 0
          then enqueue(Q, v);
while Q is non-empty
   do v := dequeue(Q);
      output v;
      for each arc (v, w)
          do indegree(w) = indegree(w) − 1;
             if indegree(w) = 0
                then enqueue(w)
```

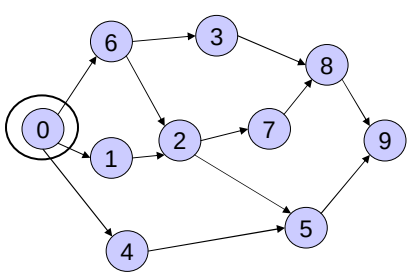
Find all starting points

Reduce indegree(w)

Place new start vertices on the Q

The running time is $O(n + m)$.

Example



0	→ 6 1 4
1	→ 2
2	→ 7 5
3	→ 8
4	→ 5
5	→ 9
6	→ 3 2
7	→ 8
8	→ 9
9	

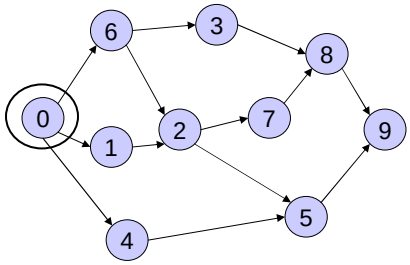
Indegree

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

start

Q = { 0 }

OUTPUT:



0	6 → 1 → 4
1	2
2	7 → 5
3	8
4	5
5	9
6	3 → 2
7	8
8	9
9	

Indegree

0	0	
1	1	-1
2	2	
3	1	
4	1	-1
5	2	
6	1	-1
7	1	
8	2	
9	2	

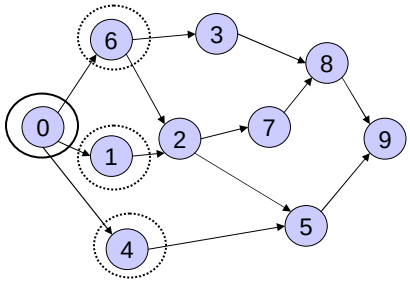
Dequeue 0 Q = { }

⇒ remove 0's arcs – adjust indegrees of neighbors

Decrement 0's neighbors

OUTPUT: **0**

13



0	6 → 1 → 4
1	2
2	7 → 5
3	8
4	5
5	9
6	3 → 2
7	8
8	9
9	

Indegree

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

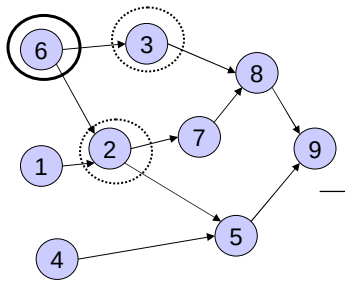
Q = { 6, 1, 4 }

Enqueue all starting points

Enqueue all new start points

OUTPUT: 0

14



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

Indegree

0	0	
1	0	
2	2	-1
3	1	-1
4	0	
5	2	
6	0	
7	1	
8	2	
9	2	

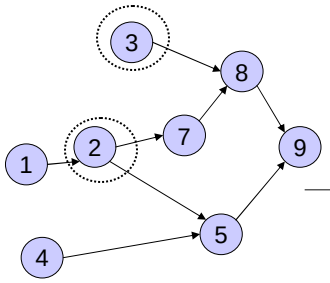
Dequeue 6 Q = { 1, 4 }

Remove arcs . Adjust indegrees of neighbors

Adjust neighbors' indegrees

OUTPUT: 0 6

15



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

Indegree

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

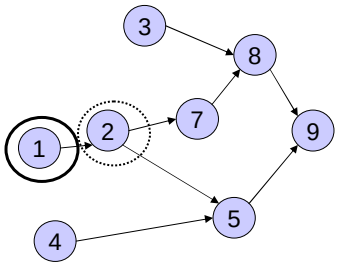
Q = { 1, 4, 3 }

Enqueue 3

Enqueue new start point

OUTPUT: 0 6

16



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

Indegree

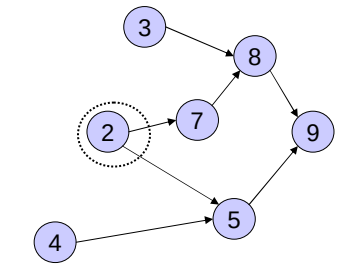
0	0	
1	0	
2	1	-1
3	0	
4	0	
5	2	
6	0	
7	1	
8	2	
9	2	

Dequeue 1 Q = { 4, 3 }
Adjust indegrees of neighbors

Adjust neighbors of 1

OUTPUT: 0 6 1

17



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

Indegree

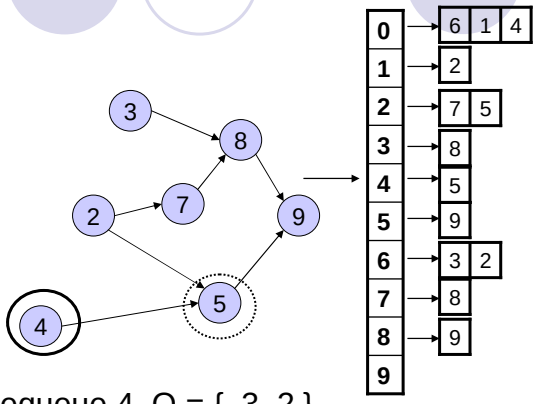
0	0
1	0
2	0
3	0
4	0
5	2
6	0
7	1
8	2
9	2

Dequeue 1 Q = { 4, 3, 2 }
Enqueue 2

Enqueue new starting points

OUTPUT: 0 6 1

18



Indegree

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

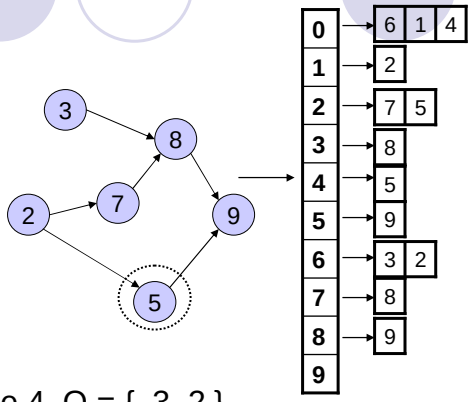
-1

Dequeue 4 Q = { 3, 2 }
Adjust indegrees of neighbors

Adjust 4's neighbors

OUTPUT: 0 6 1 4

19



Indegree

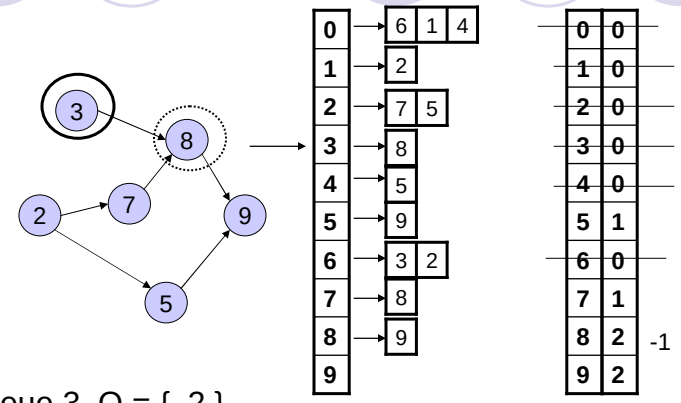
0	0
1	0
2	0
3	0
4	0
5	1
6	0
7	1
8	2
9	2

Dequeue 4 Q = { 3, 2 }
No new start points found

NO new start points

OUTPUT: 0 6 1 4

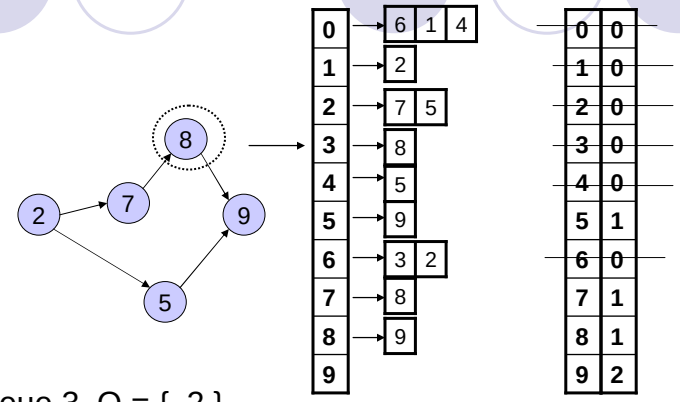
20



Dequeue 3 Q = { 2 }
Adjust 3's neighbors

OUTPUT: 0 6 1 4 3

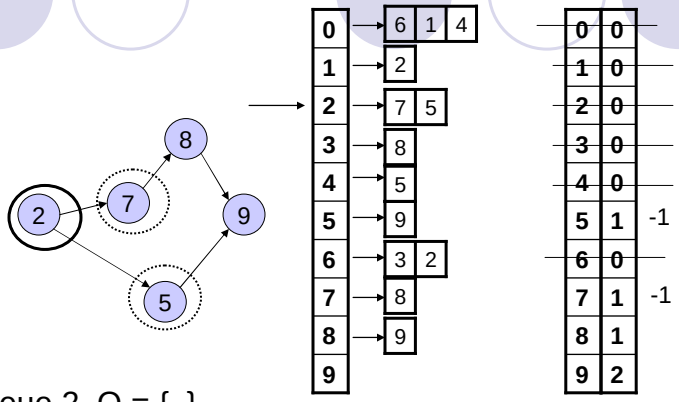
21



Dequeue 3 Q = { 2 }
No new start points found

OUTPUT: 0 6 1 4 3

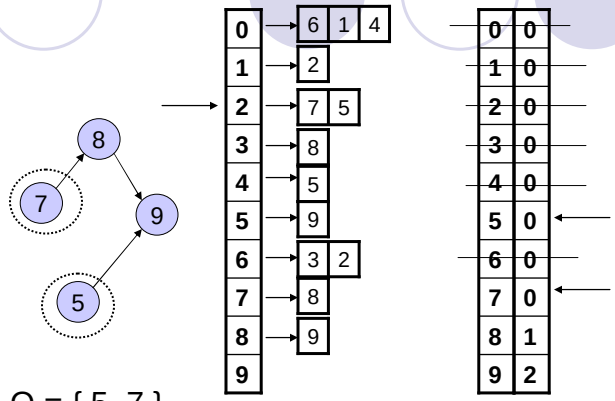
22



Dequeue 2 Q = { }
Adjust 2's neighbors

OUTPUT: 0 6 1 4 3 2

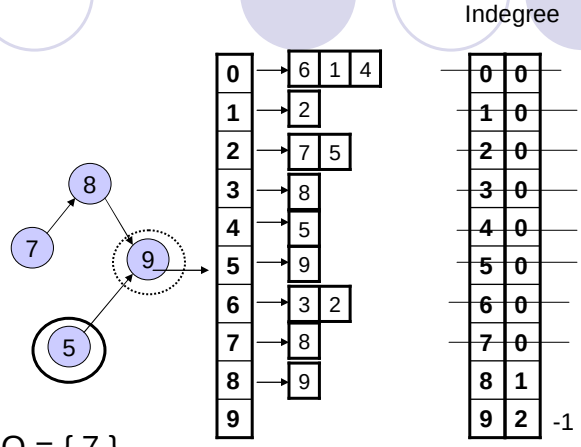
23



Dequeue 2 Q = { 5, 7 }
Enqueue 5, 7

OUTPUT: 0 6 1 4 3 2

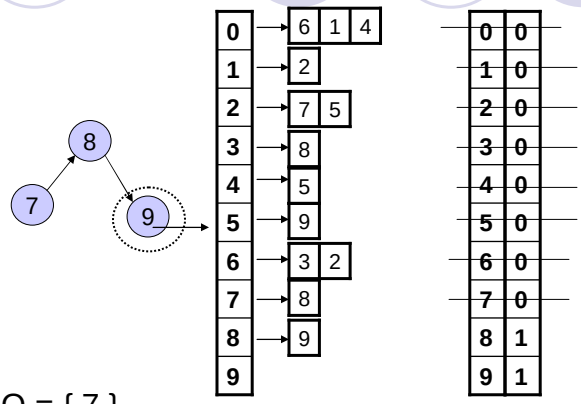
24



Dequeue 5 Q = { 7 }
Adjust neighbors

OUTPUT: 0 6 1 4 3 2 5

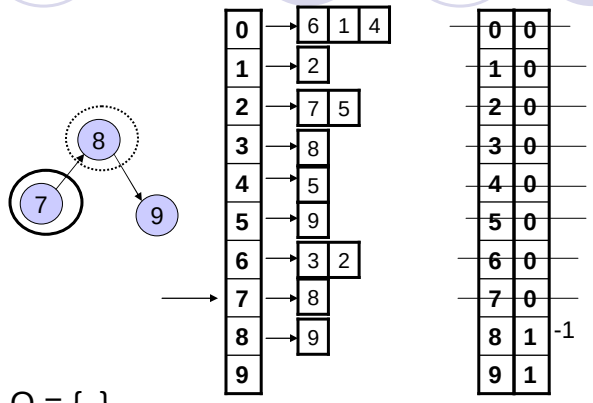
25



Dequeue 5 Q = { 7 }
No new start points

OUTPUT: 0 6 1 4 3 2 5

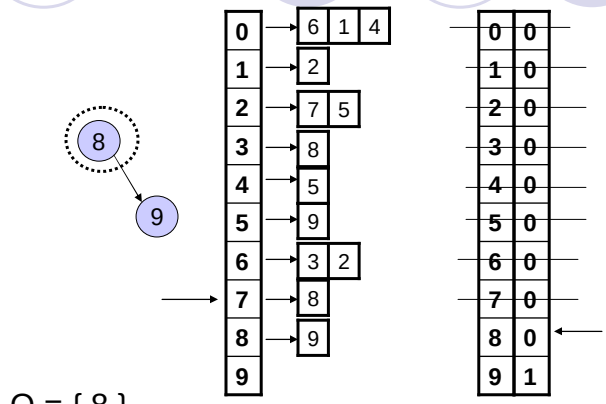
26



Dequeue 7 Q = { }
Adjust neighbors

OUTPUT: 0 6 1 4 3 2 5 7

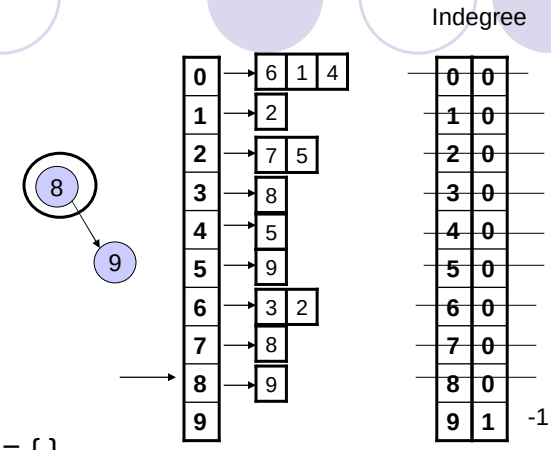
27



Dequeue 7 Q = { 8 }
Enqueue 8

OUTPUT: 0 6 1 4 3 2 5 7

28

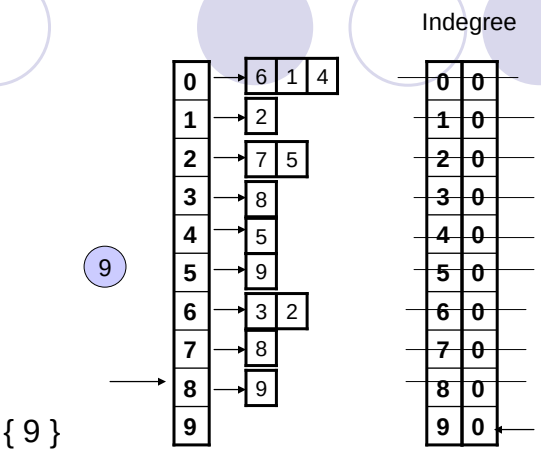


Dequeue 8 Q = { }

Adjust indegrees of neighbors

OUTPUT: 0 6 1 4 3 2 5 7 8

29



Dequeue 8 Q = { 9 }

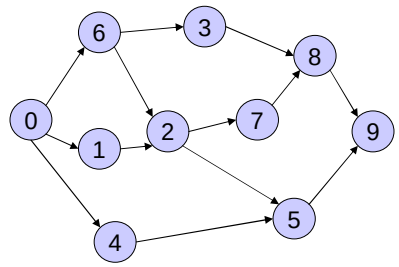
Enqueue 9

Dequeue 9 Q = { }

STOP – no neighbors

OUTPUT: 0 6 1 4 3 2 5 7 8 9

30



OUTPUT: 0 6 1 4 3 2 5 7 8 9

Is output topologically correct?

31

Running Time

- We never visited a vertex more than one time
- For each vertex, we had to examine all outgoing edges
  - Σ *outdegree(v)* = *E*
  - This is summed over all vertices, not per vertex
- So, our running time is exactly O(V + E)

32

Next time …

- Graph traversal
  - Breadth first search
  - Depth first search

- Note: Level-order traversal for trees uses an algorithm similar to (but much simpler than) that of topological sort for DAGs.

33