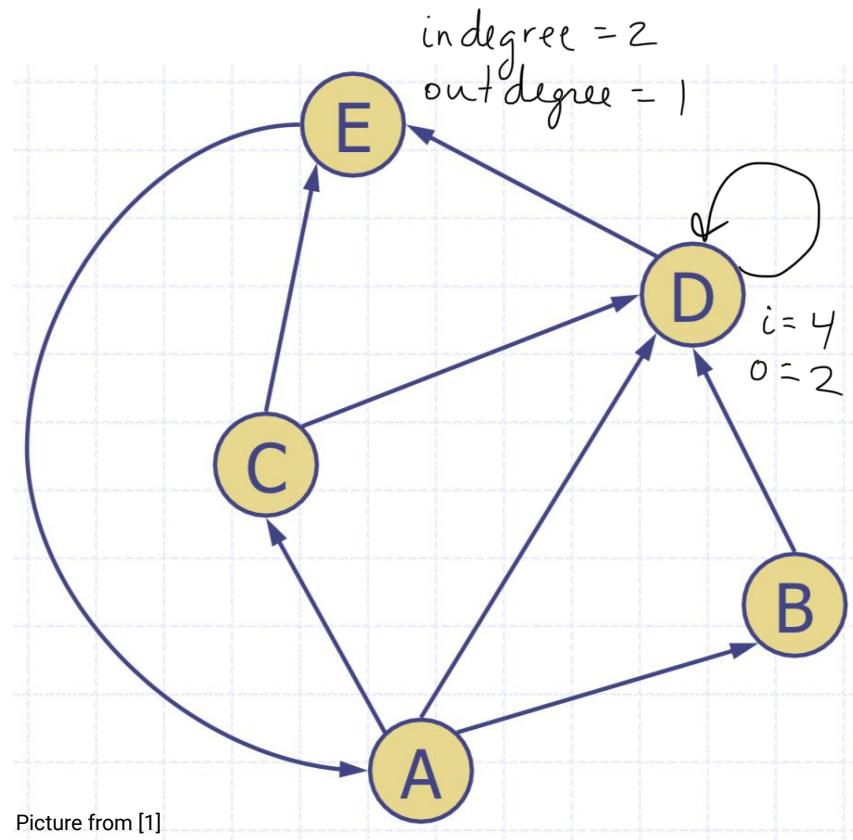


Graphs III

- Digraph Overview
- Directed DFS
- Strong Connectivity
- Transitive Closure
- Floyd-Warshall Algorithm
- Topological Sorting

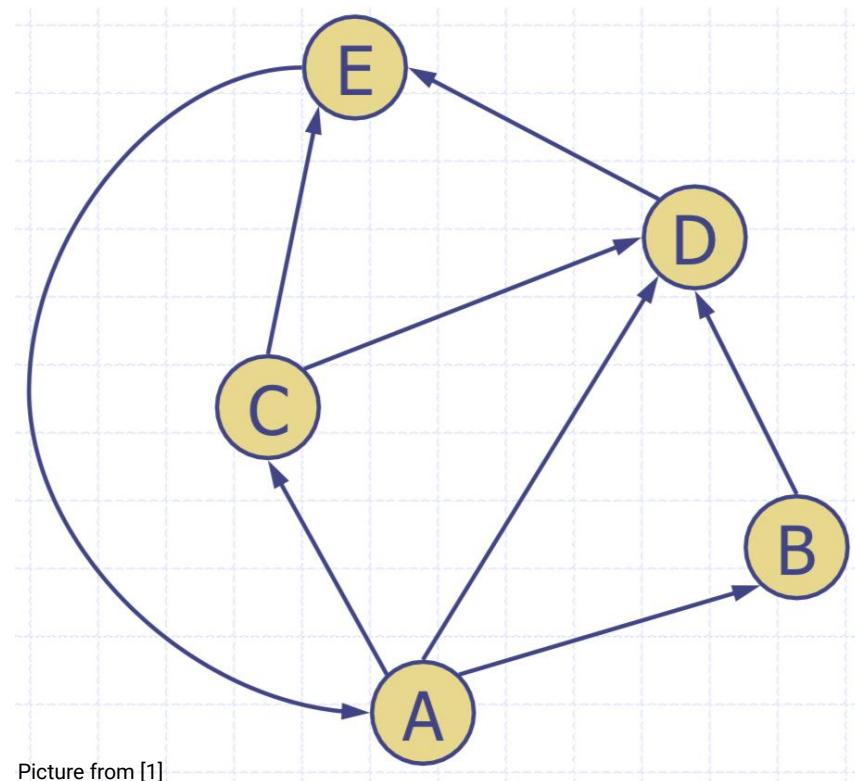
Directed Graph (Digraph)

- A digraph is a graph whose edges are all directed
- The indegree of a vertex is the number of edges pointing to the vertex
- The outdegree of a vertex is the number of edges pointing away from the vertex.



Directed Graph (Digraph)

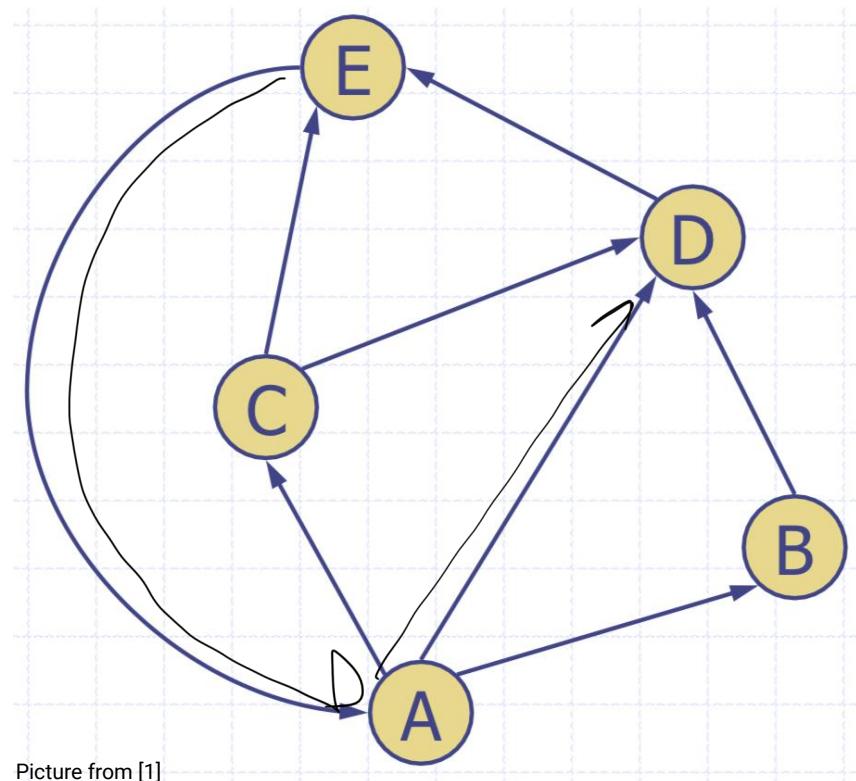
- A directed path is a sequence of vertices in a digraph such that there is a directed edge from each vertex to the next vertex in the sequence (e.g. A-B-D-E)
- A directed Cycle is a directed path whose first and last vertices are the same (e.g. (A-D-E-A))
- Simple: no repeated vertices or edges
- Length: number of edges

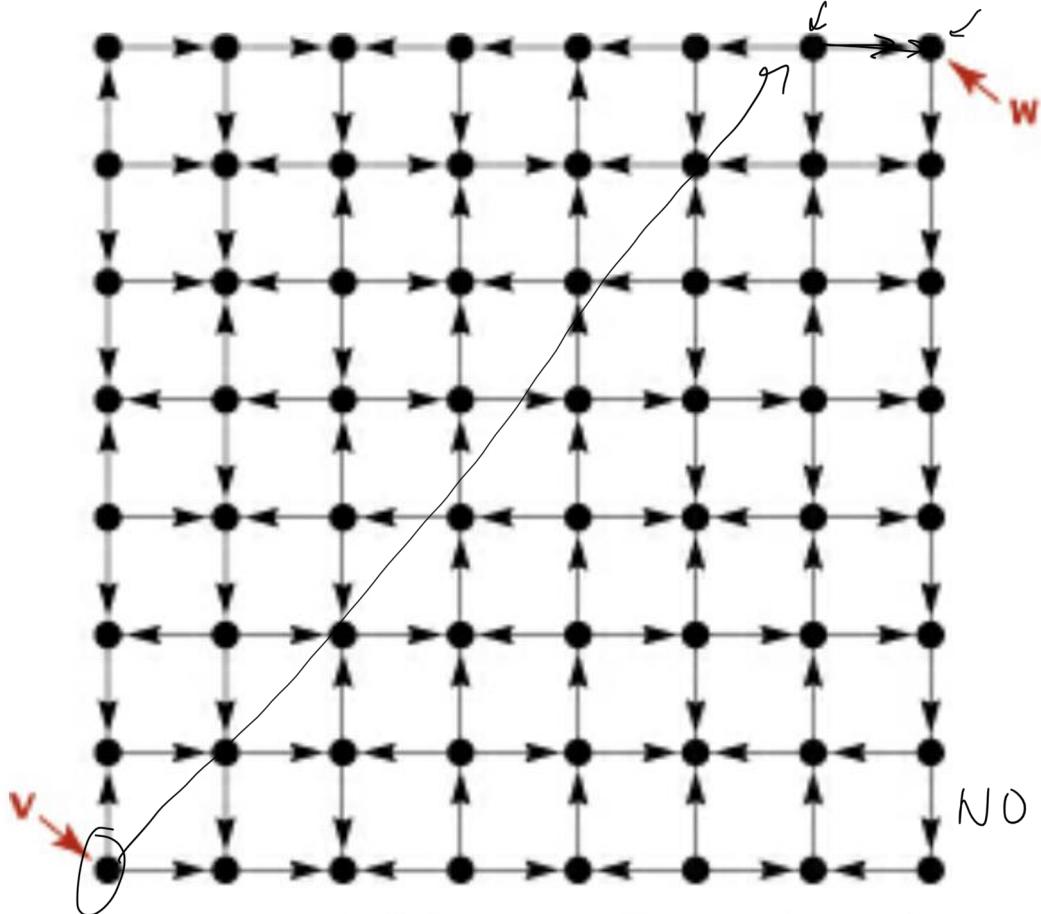


Directed Graph (Digraph)

- The edge (u,v) means that the edge is going out of u and into v
- A vertex u is reachable from vertex v iff there is a directed path from v to u

EXAMPLE: Is **D** reachable from **E**?

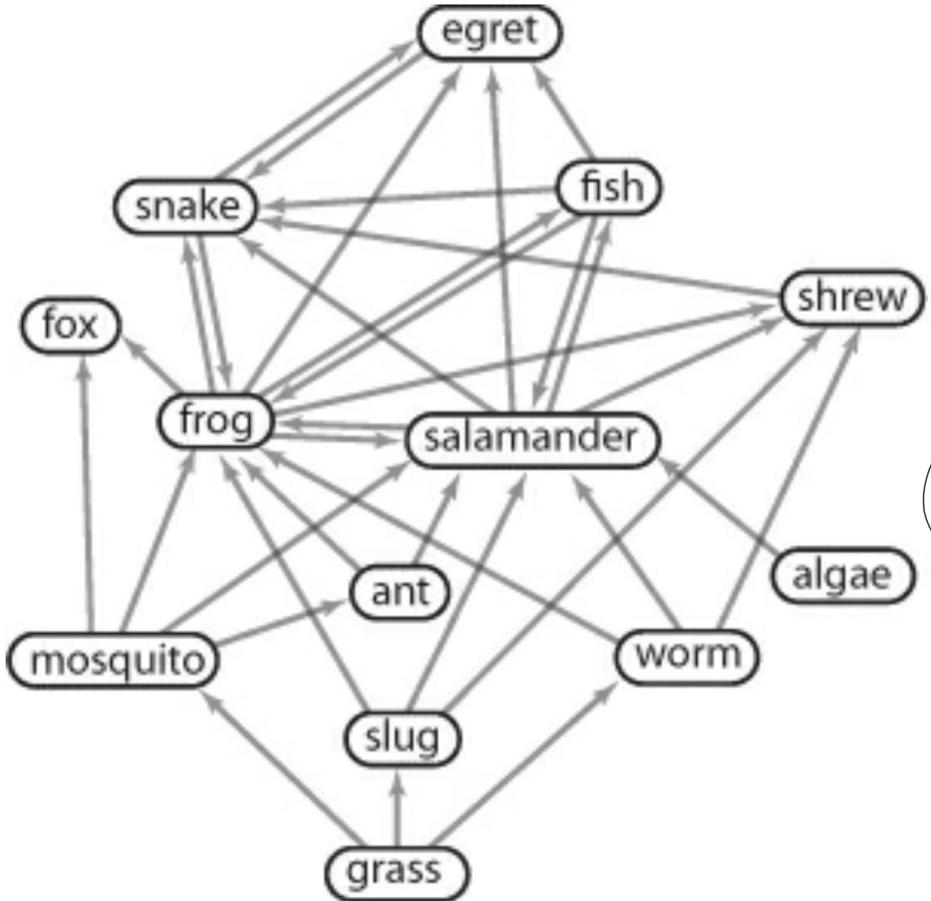




Is w reachable from v in this digraph?

application	vertex	edge
<i>food web</i>	species	predator-prey
<i>internet content</i>	page	hyperlink
<i>program</i>	module	external reference
<i>cellphone</i>	phone	call
<i>scholarship</i>	paper	citation
<i>financial</i>	stock	transaction
<i>internet</i>	machine	connection

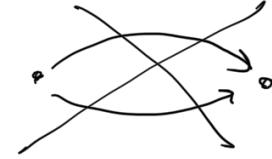
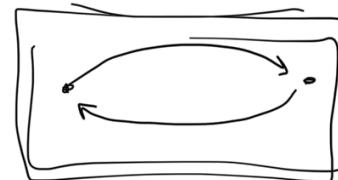
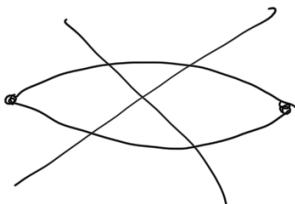
Typical digraph applications



Small subset of food web digraph

- An edge (u, v) means “ u is eaten by v ”
- Edges are directed so (u,v) and (v,u) are not the same edge
- How can we represent digraphs?
 - Edge list
 - Adjacency matrix
 - Adjacency lists
- What would be different about this adjacency matrix and one for an undirected graph?
- What would be the exact space requirement for an adjacency list in terms of $|V|$ and $|E|$?

Simple Digraph



- A **simple** digraph has no parallel edges and no self loops.
- In a simple digraph, $E \leq V(V-1)$.
- How is that different from a simple undirected graph and why?

undirected: $E \leq \frac{V(V-1)}{2}$

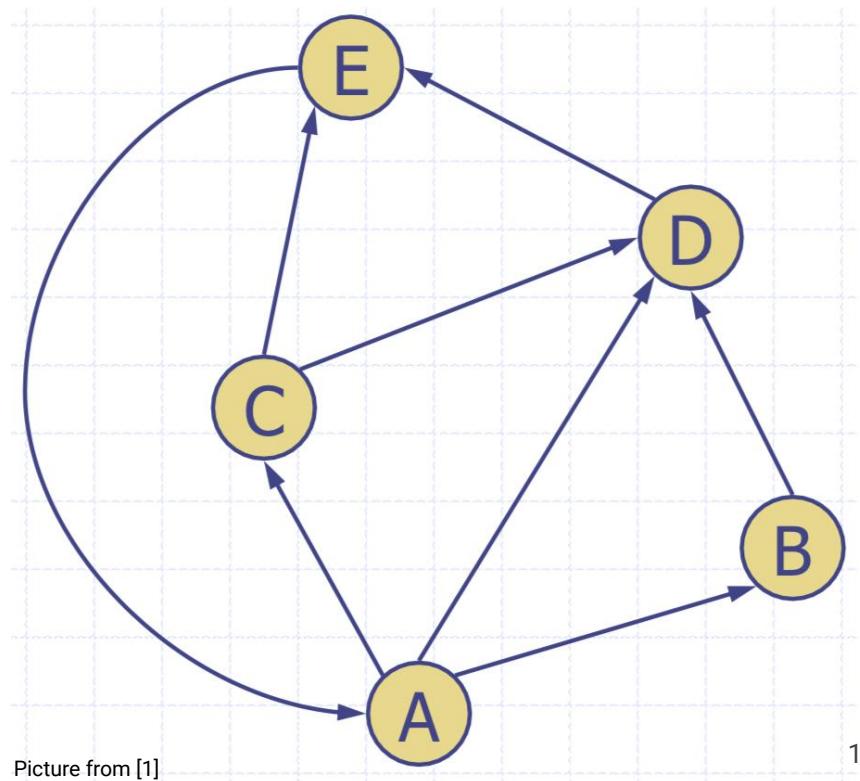
public class Digraph	
Digraph(int V)	<i>create a V-vertex digraph with no edges</i>
Digraph(In in)	<i>read a digraph from input stream in</i>
int V()	<i>number of vertices</i>
int E()	<i>number of edges</i>
void addEdge(int v, int w)	<i>add edge v->w to this digraph</i>
Iterable<Integer> adj(int v)	<i>vertices connected to v by edges pointing from v</i>
Digraph reverse()	<i>reverse of this digraph</i>
String toString()	<i>string representation</i>

API for a digraph

Picture from [2]

Digraph API and Representation

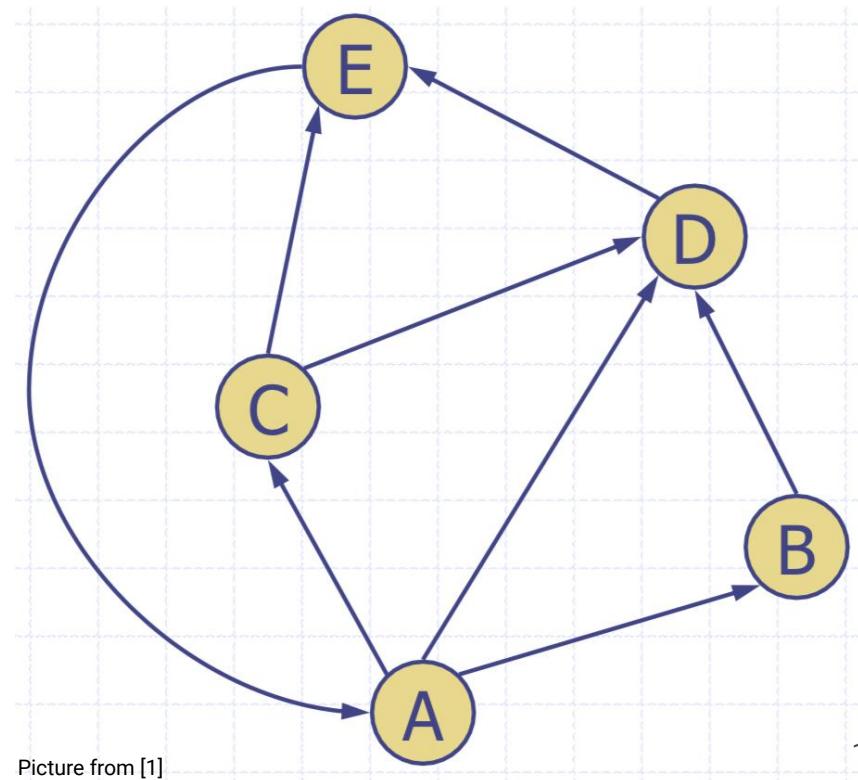
What is a (possible) disadvantage of using a regular adjacency list representation?

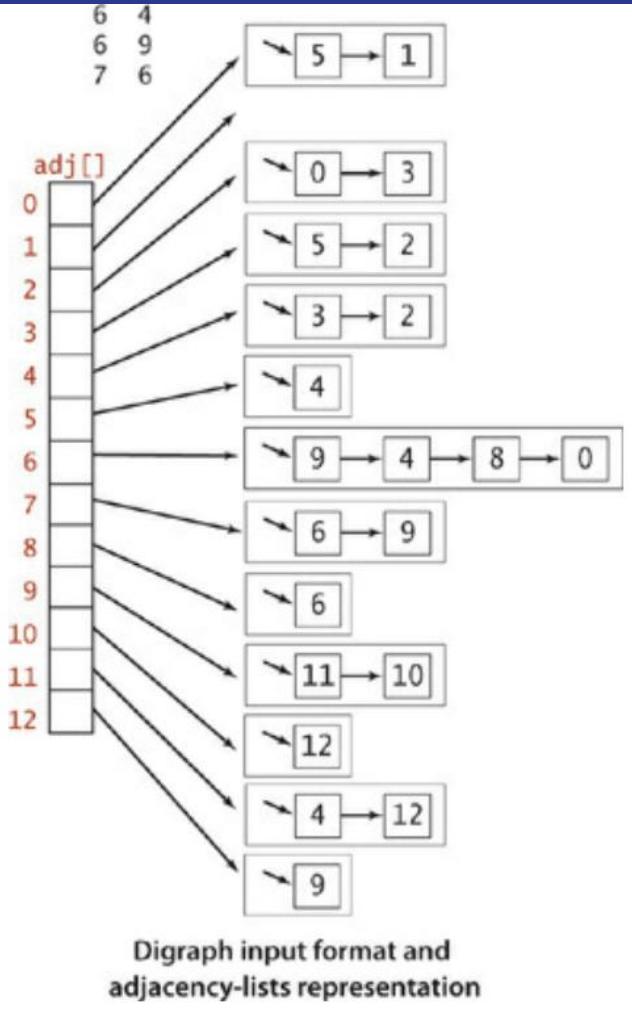


Digraph API and Representation

What is a (possible) disadvantage of using a regular adjacency list representation?

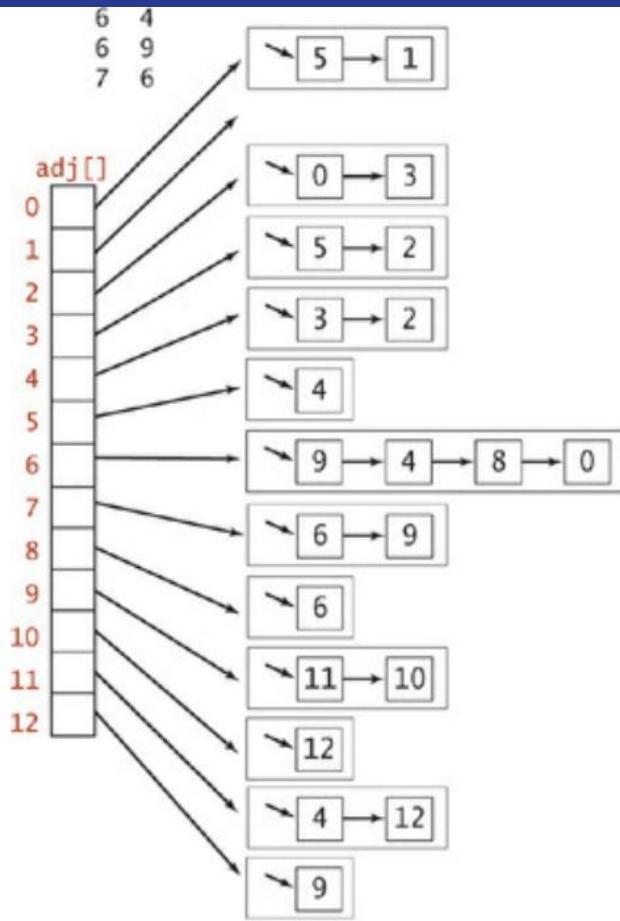
With a regular adjacency list representation, it is easy to get the **outdegree** of a graph but not as easy to get the **indegree** of a graph.



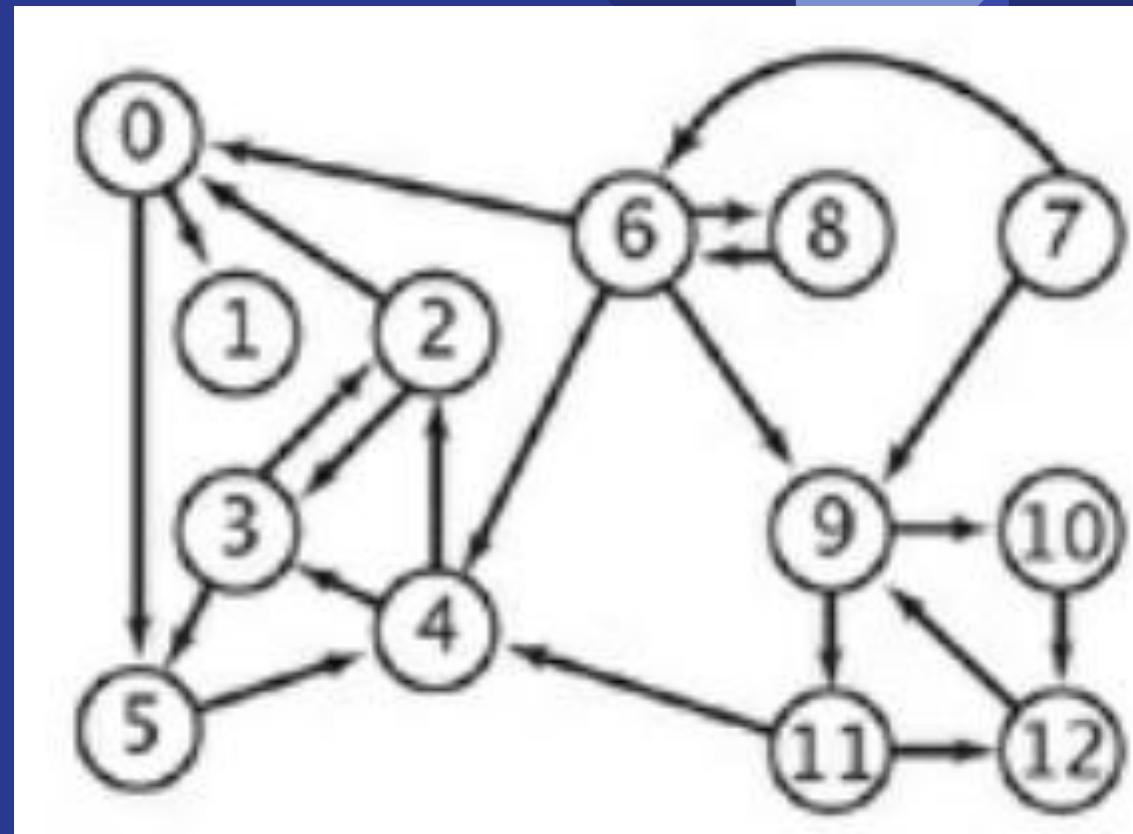


With a regular adjacency list representation, it is easy to get the **outdegree** of a graph but not as easy to get the **indegree** of a graph.

- We could maintain two adjacency lists per vertex, one for the edges going out and one for the edges going in (What would be the space?)
- We could calculate the **indegree** of a vertex by analyzing the **reverse** of the graph



Digraph input format and
adjacency-lists representation

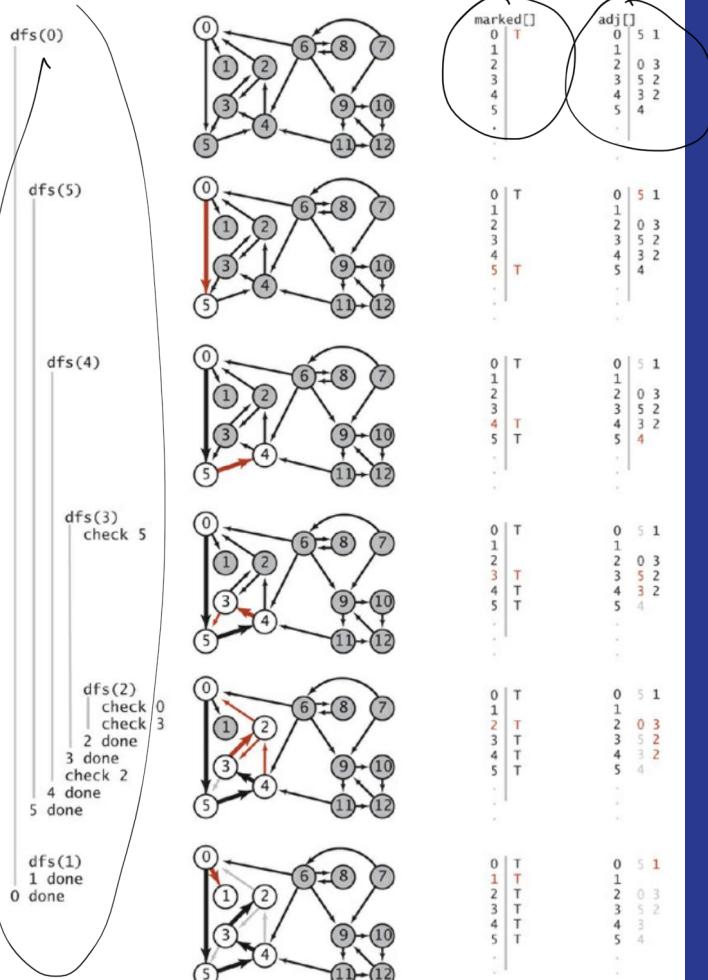


Single-source Reachability

Given a digraph and a source vertex s , determine if there is a directed path from s to a target vertex t .

But how???

DFS

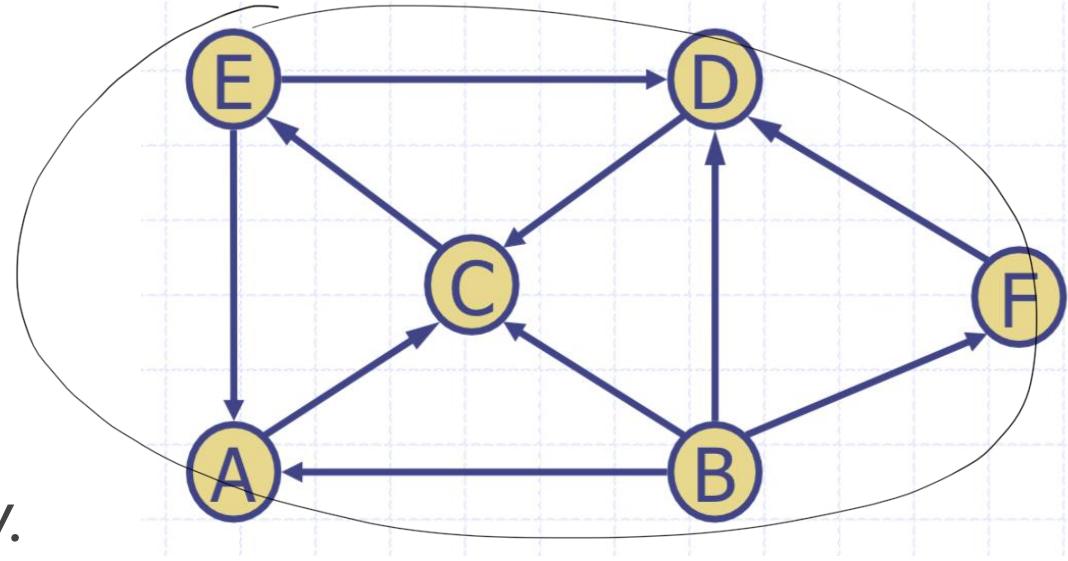
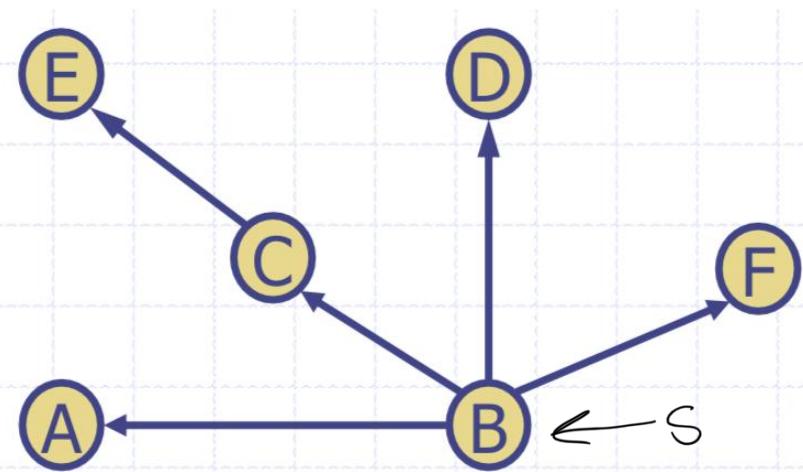
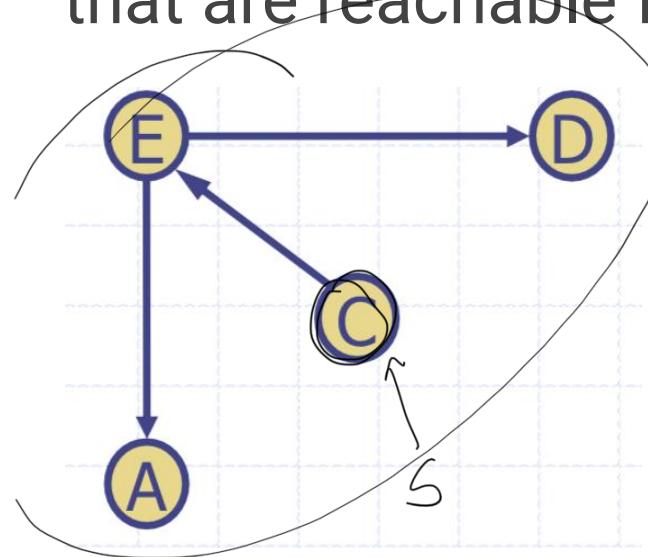


Directed DFS

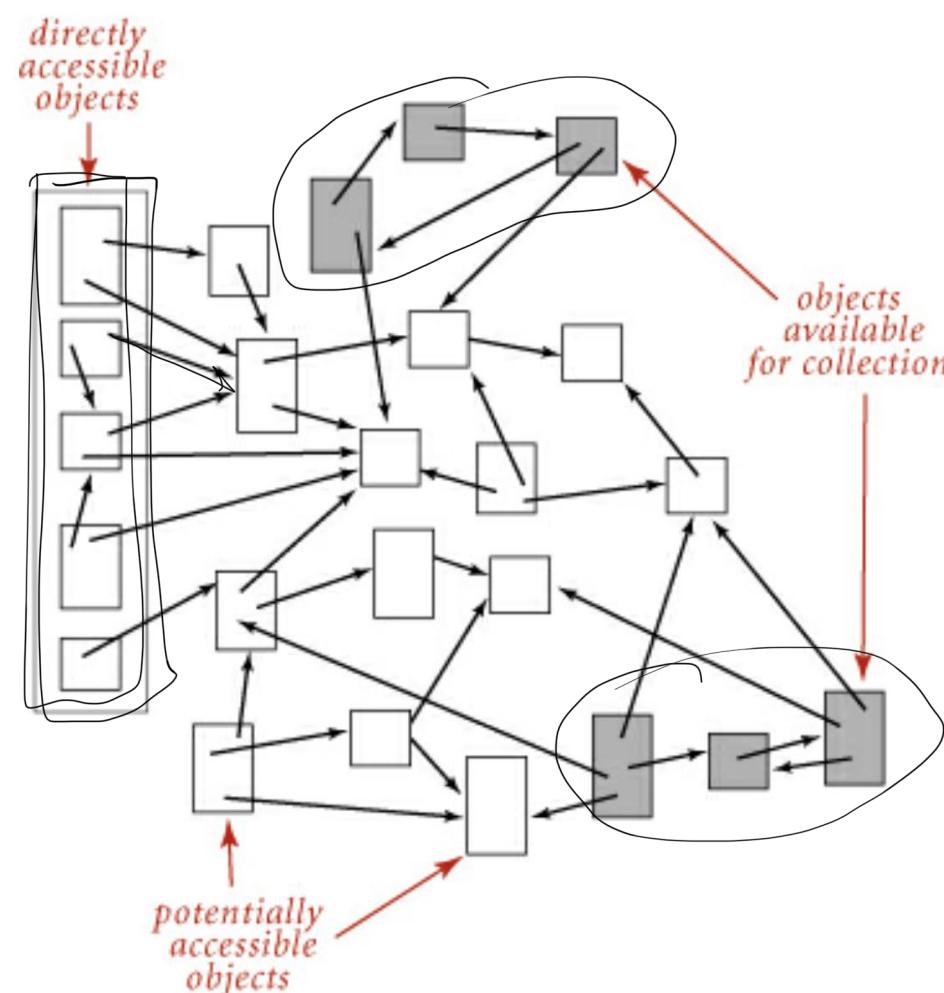
Note that with a directed graph, we would benefit from a new version of the notion of “connectedness” because *connected* no longer implies *reachability*.

Directed DFS

Given a source v , we can discover all the vertices that are reachable from v .



An application: garbage collection



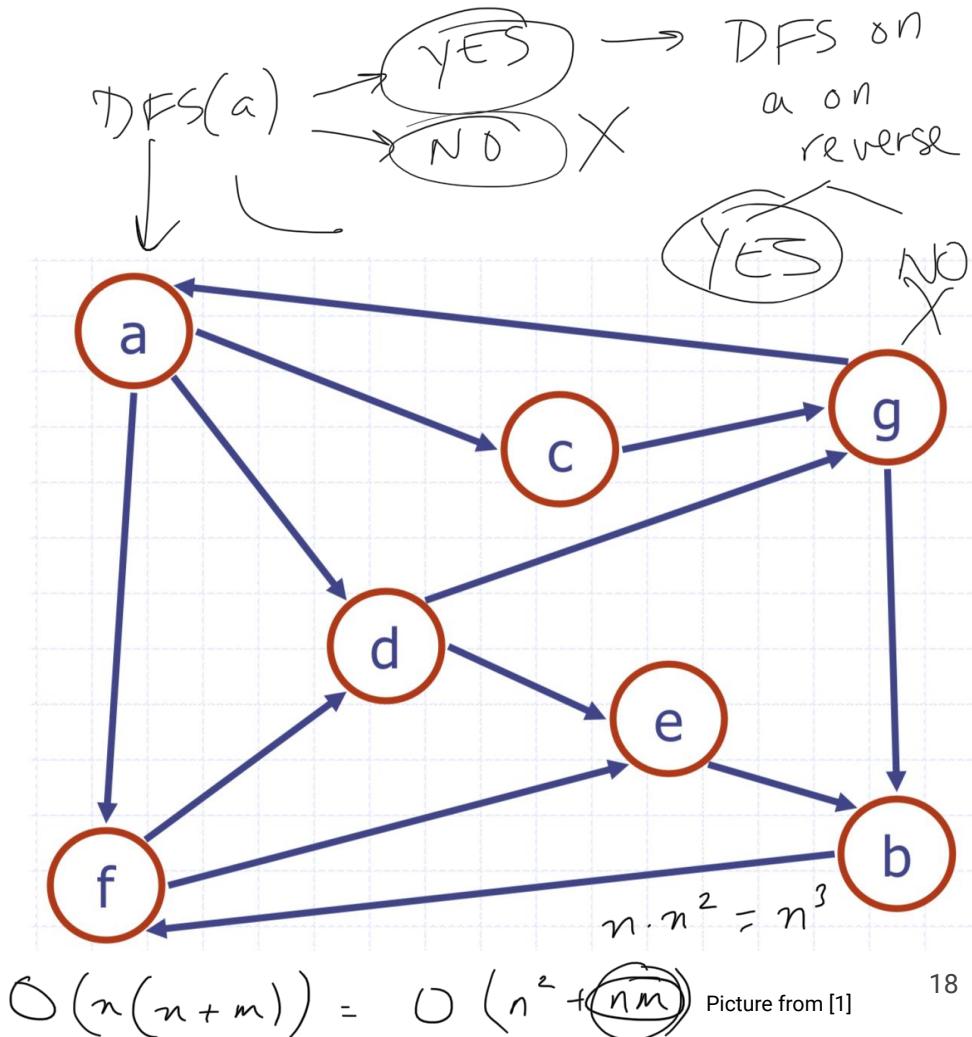
Strong Connectivity

A digraph is strongly connected if every vertex is reachable from every other vertex (similar to the idea of connectivity in an undirected graph.)

How can we determine if a graph is strongly connected or not?

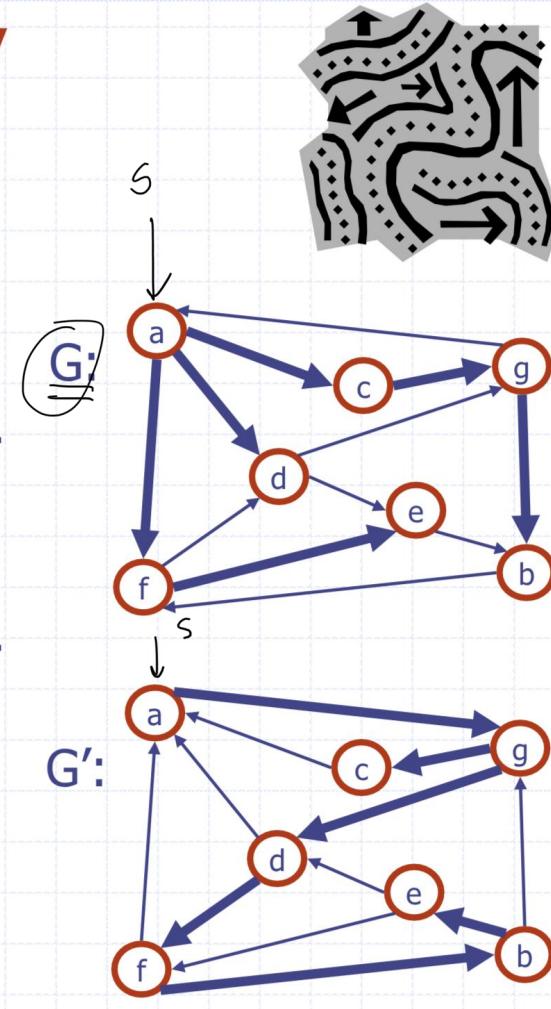
DFS : $O(n + m)$

DFS on every vertex :



Strong Connectivity Algorithm

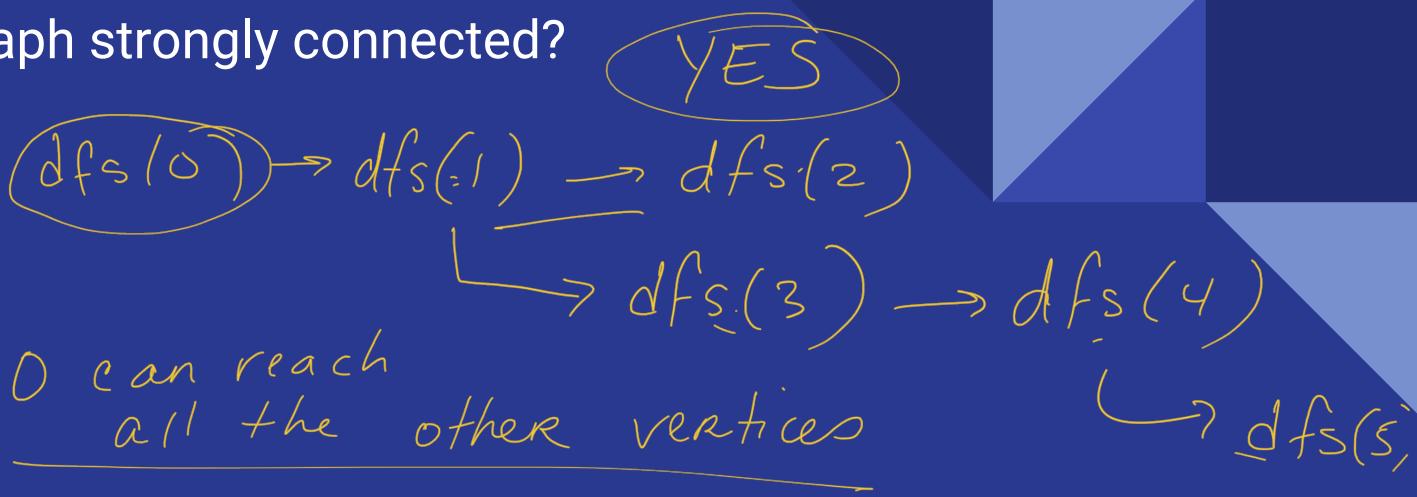
- ◆ Pick a vertex v in G .
- ◆ Perform a DFS from v in G .
 - If there's a w not visited, print "no".
- ◆ Let G' be G with edges reversed.
- ◆ Perform a DFS from v in G' .
 - If there's a w not visited, print "no".
 - Else, print "yes".
- ◆ Running time: $O(n+m)$.



Example. Is this graph strongly connected?

YES

0	1, 3
1	2, 3, 4
2	0, 1
3	4, 5
4	2, 5
5	0, 1



G' :

0	2, 5
1	0, 2, 5
2	1, 4
3	0, 1
4	1, 3
5	3, 7

dfs(0) → dfs(2) → dfs(1) → dfs(5)

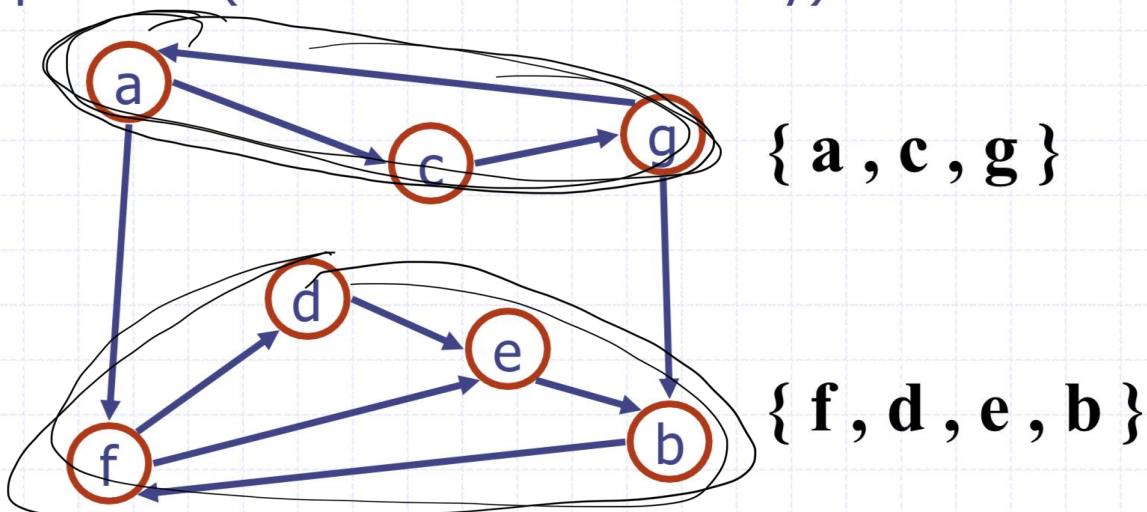
0 can be reached by all other vertices

dfs(3) ↘
↓
dfs(4)

Strongly Connected Components



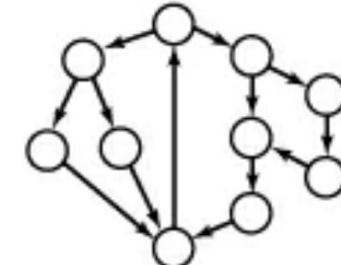
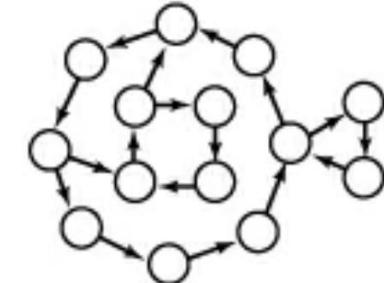
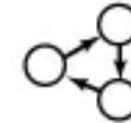
- ◆ Maximal subgraphs such that each vertex can reach all other vertices in the subgraph
- ◆ Can also be done in $O(n+m)$ time using DFS, but is more complicated (similar to biconnectivity).



Equivalence Relation

Properties of Strong Connectivity

- **Reflexive:** Every vertex is strongly connected to itself
- **Symmetric:** If v is strongly connected to u , then u is strongly connected to v
- **Transitive:** If v is strongly connected to u and u is strongly connected to w , then v is strongly connected to w



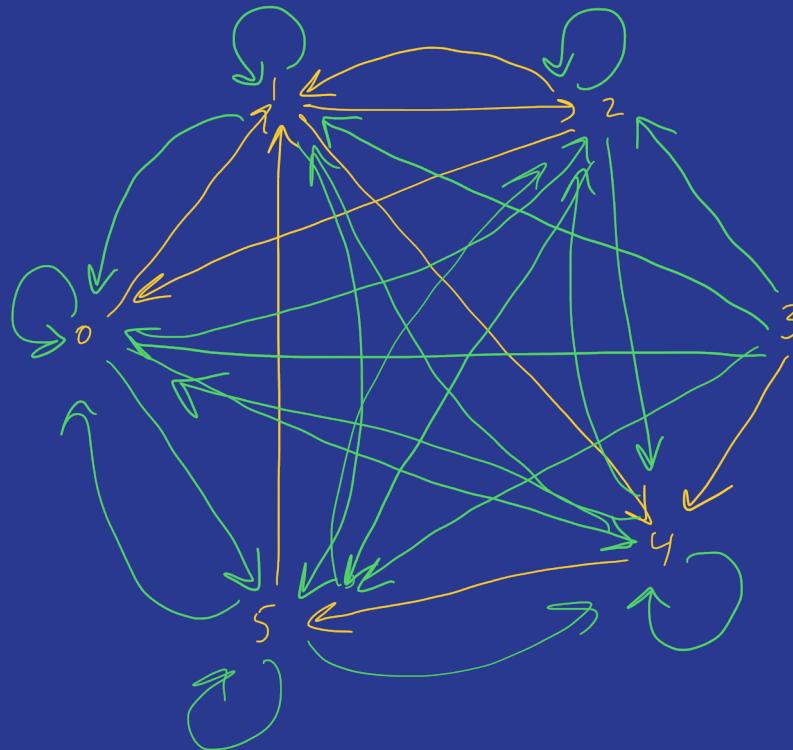
Transitive Closure

G^* is the *transitive closure* of a directed graph G if:

- G^* has the same vertices as G
- G^* contains a directed edge (u, v) if and only if there is a directed path from u to v in G

Example. Draw the transitive closure of this graph.

0	1
1	2, 4
2	0, 1
3	4
4	5
5	1



Transitive Closure and self-loops...

- Variation 1 (Sedgewick & Wayne): Automatically add 1's down the diagonal because every vertex is reachable from itself (i.e. every vertex has a self-loop in the transitive closure.)
- Variation 2 (Goodrich & Tamassia): Set the diagonal to all 0's unless a self-loop exists in the original graph.
- Variation 3 (Warshall): The transitive closure will have 1's in the diagonal (i.e. self-loops) if:
 - there is a self-loop in the original graph
 - there is a cycle in the original graph containing that vertex

In this class, we use Variation 3.

How can we compute the transitive closure of a graph?

DFS on each node:

$$O(n(n+m))$$

↓

$$O(n^3)$$

How can we compute the transitive closure of a graph?

Run DFS on all vertices:
 $O(n(n+m)) \rightarrow O(n^3)$ for dense graphs...

Floyd-Warshall Transitive Closure

Algorithm FloydWarshall(M)

Input: M , the adjacency matrix of a graph G

Output: The transitive closure G^* of the graph G

$R^{(0)} \leftarrow M$

for $k \leftarrow 1$ to n do

 for $i \leftarrow 1$ to n do

 for $j \leftarrow 1$ to n do

 return $R^{(n)}$

already has from i to j
an edge

$R^{(k)}[i, j] \leftarrow R^{(k-1)}[i, j]$ or $(R^{(k-1)}[i, k] \text{ and } R^{(k-1)}[k, j])$

can i reach j through k ?

iterating through matrix

Example. Compute the transitive closure of the graph below using the Floyd-Warshall Algorithm.

$M:$

	1	2	3
1	0	1	1
2	0	0	0
3	1	1	0

$R^1:$

\downarrow

	1	2	3
1	0	1	1
2	0	0	0
3	1	1	1

$R^2:$

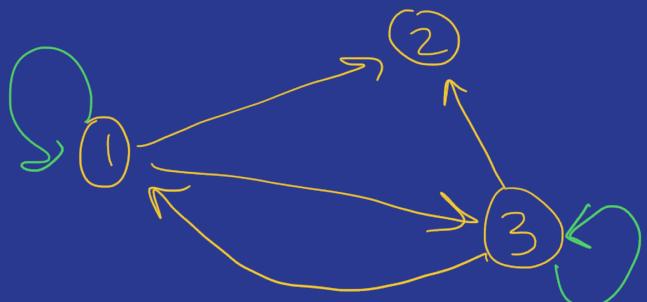
\downarrow

	1	2	3
1	0	1	1
2	0	0	0
3	1	1	1

$R^3:$

\downarrow

	1	2	3
1	1	1	1
2	0	0	0
3	1	1	1



Floyd-Warshall Algorithm Discussion

- Assuming an adjacency matrix representation, what is the space requirement for the algorithm?

$$O(V^2)$$

- What is the runtime of the algorithm?

$$O(V^3)$$

- Assuming no self-loops in the original graph, what does a self-loop (a 1 in the diagonal of the matrix) in the transitive closure graph tell us?

They are in a cycle.

- What does it mean if the resulting matrix is all 1's?

It is strongly connected.

Floyd-Warshall Algorithm Discussion

- Assuming an adjacency matrix representation, what is the space requirement for the algorithm?
 $O(V^2)$
- What is the runtime of the algorithm?
 $O(V^3)$
- Assuming no self-loops in the original graph, what does a self-loop (a 1 in the diagonal of the matrix) in the transitive closure graph tell us?
There is a cycle in the original graph starting and ending at that vertex.
- What does it mean if the resulting matrix is all 1's?
The original graph is strongly connected.

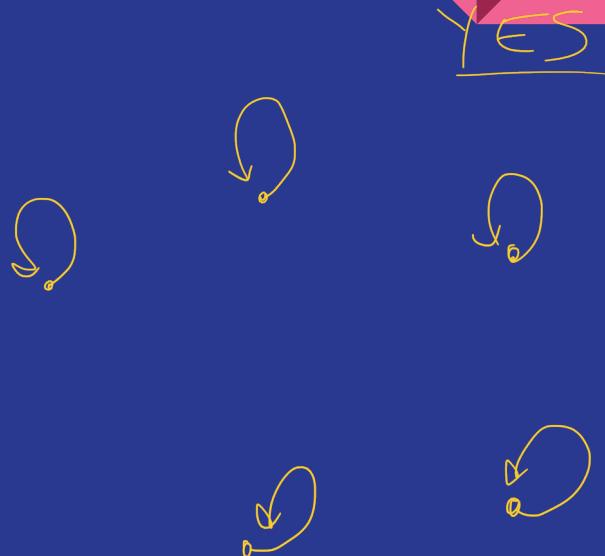
Is this a possible transitive closure of a directed graph?

0	1	1	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

NOT with our variation

Is this a possible transitive closure of a directed graph?

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1



YES

Is this a possible transitive closure of a directed graph?

1	1	1	1	1
0	1	1	1	1
0	0	1	1	1
0	0	0	1	1
0	0	0	0	1



YES

Is this a possible transitive closure of a directed graph?

1	0	1	0	1
0	0	1	0	1
1	1	1	0	1
0	0	0	0	1
1	1	1	1	1

No



DAGs and Topological Ordering

- ◆ A directed acyclic graph (DAG) is a digraph that has no directed cycles
- ◆ A topological ordering of a digraph is a numbering

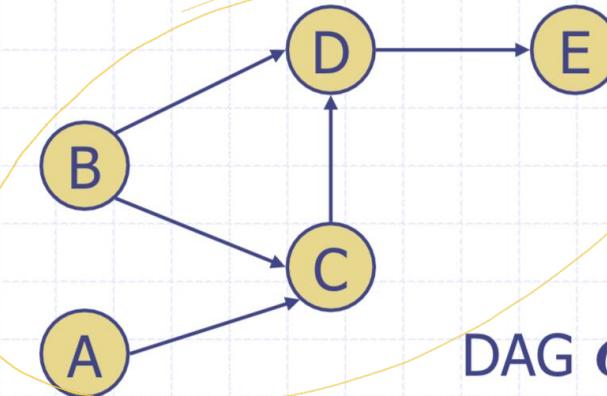
v_1, \dots, v_n

of the vertices such that for every edge (v_i, v_j) , we have $i < j$

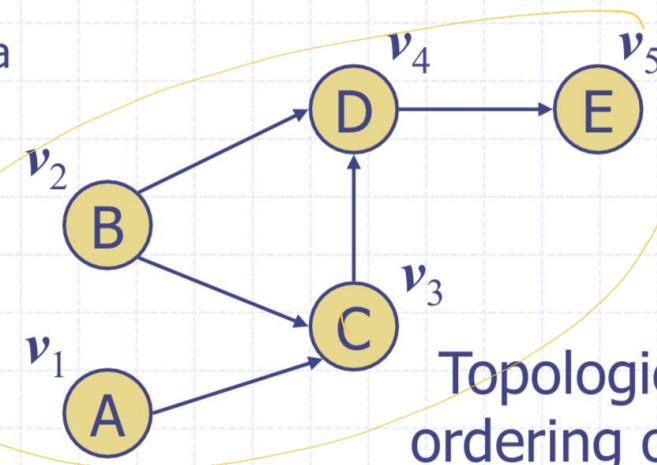
- ◆ Example: in a task scheduling digraph, a topological ordering of a task sequence satisfies the precedence constraints

Theorem

A digraph admits a topological ordering if and only if it is a DAG



DAG G

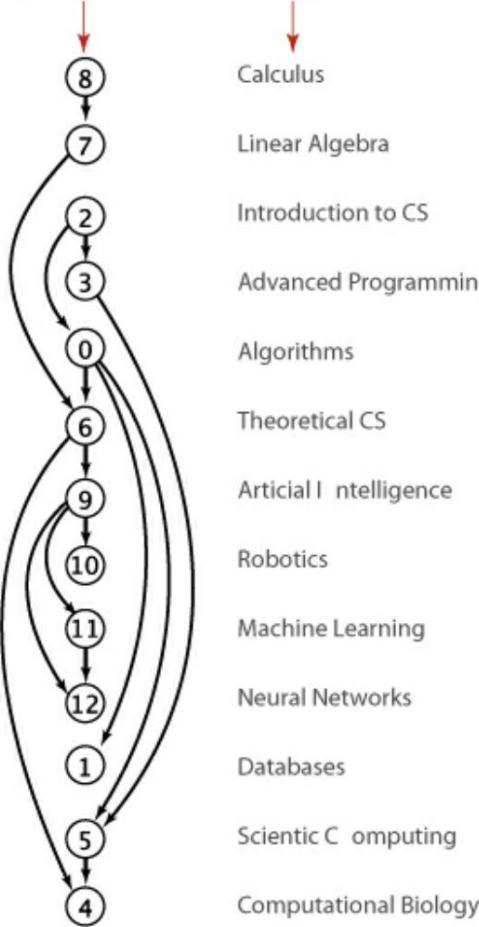


Topological ordering of G



*edges all
point down*

*prerequisites
all satisfied*



Topological Ordering is especially useful in scheduling problems...

Topological Sorting

Method TopologicalSort(G)

$H \leftarrow G$ // Temporary copy of G

$n \leftarrow G.\text{numVertices}()$

while H is not empty **do**

 Let v be a vertex with no outgoing edges

 Label $v \leftarrow n$

$n \leftarrow n - 1$

 Remove v from H

Example. Find a topological ordering of the graph below.

0	1, 3
1	2, 3, 4
2	0, 1
3	4, 5
4	2, 5
5	0, 1

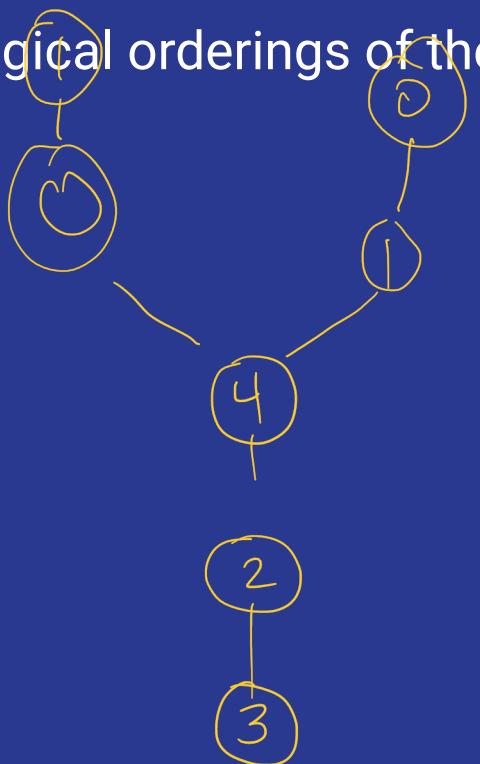
No.

Example. Find a topological ordering of the graph below.

0	
1	3
2	1, 3
3	
4	0, 3
5	2

Example. Find all the topological orderings of the graph below.

0	2, 3, 4
1	2, 3, 4
2	3
3	
4	



Topological Sorting

Method `TopologicalSort(G)`

$H \leftarrow G$ // Temporary copy of G

$n \leftarrow G.\text{numVertices}()$

while H is not empty **do**

 Let v be a vertex with no outgoing edges

 Label $v \leftarrow n$

$n \leftarrow n - 1$

 Remove v from H

Runtime: $O(V + E)$

How do we actually implement it given an adjacency list representation?

DFS

Picture from [1]

Topological Sorting with DFS

- Run DFS using a Stack to keep track of the current path (this will also determine if there are any cycles.)
- Return vertices in the opposite order of the vertices that are “done” first--done meaning the recursive call is completed (i.e. there are no outgoing edges to consider.)
- May need to run DFS again on any unvisited vertices, but this is a separate run, so the runtime is added not multiplied.
- Total runtime: $O(V + E)$

References

- [1] Tamassia and Goodrich
- [2] Sedgewick and Wayne