

# Directed graph algorithms

Data Structures and Algorithms for Computational Linguistics III  
(ISCL-BA-07)

Çağrı Çöltekin

`ccoltekin@sfs.uni-tuebingen.de`

University of Tübingen  
Seminar für Sprachwissenschaft

Winter Semester 2021/22

# Directed graphs

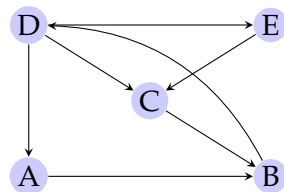
- Directed graphs are graphs with directed edges
- Some operations are more meaningful or challenging in directed graphs
- We will cover some of these operations, and some interesting sub-types of directed graphs
  - Transitive closure
  - Directed acyclic graphs
  - Topological ordering

# Some terminology

- For any pair of nodes  $u$  and  $v$  in a directed graph
  - A directed graph is *strongly connected* if there is a directed path between  $u$  to  $v$  and  $v$  to  $u$
  - A directed graph is *semi-connected* if there is a directed path between  $u$  to  $v$  or  $v$  to  $u$
  - A directed graph is *weakly connected* if the undirected graph obtained by replacing all edges with undirected edges result in a connected graph

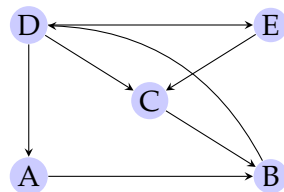
# Checking strong connectivity

- Naive attempt: traverse the graph independently from each node (strongly connected if all traversals visit all nodes)



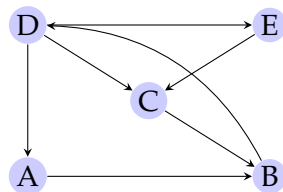
# Checking strong connectivity

- Naive attempt: traverse the graph independently from each node (strongly connected if all traversals visit all nodes)
  - Time complexity:



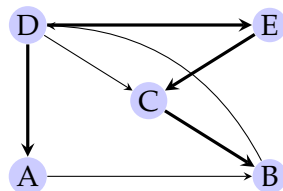
# Checking strong connectivity

- Naive attempt: traverse the graph independently from each node (strongly connected if all traversals visit all nodes)
  - Time complexity:  $O(n(n + m))$



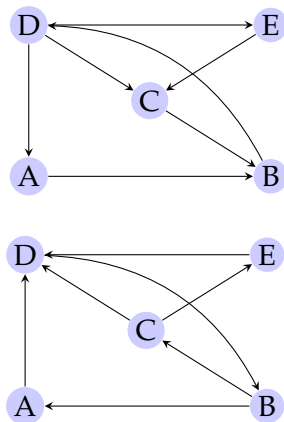
# Checking strong connectivity

- Naive attempt: traverse the graph independently from each node (strongly connected if all traversals visit all nodes)
  - Time complexity:  $O(n(n + m))$
- A better one:
  - traverse the graph from an arbitrary node



# Checking strong connectivity

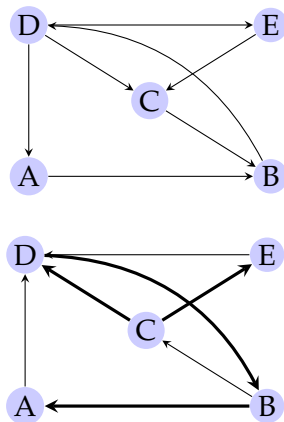
- Naive attempt: traverse the graph independently from each node (strongly connected if all traversals visit all nodes)
  - Time complexity:  $O(n(n + m))$
- A better one:
  - traverse the graph from an arbitrary node
  - reverse all edges, traverse again





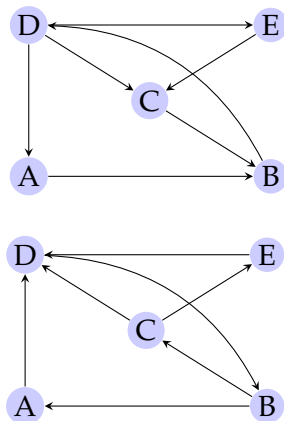
# Checking strong connectivity

- Naive attempt: traverse the graph independently from each node (strongly connected if all traversals visit all nodes)
  - Time complexity:  $O(n(n + m))$
- A better one:
  - traverse the graph from an arbitrary node
  - reverse all edges, traverse again
  - intuition: if there is a reverse path from D to A, then D is reachable from A



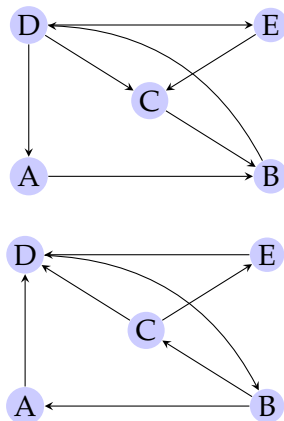
# Checking strong connectivity

- Naive attempt: traverse the graph independently from each node (strongly connected if all traversals visit all nodes)
  - Time complexity:  $O(n(n + m))$
- A better one:
  - traverse the graph from an arbitrary node
  - reverse all edges, traverse again
  - intuition: if there is a reverse path from D to A, then D is reachable from A
- Time complexity:



# Checking strong connectivity

- Naive attempt: traverse the graph independently from each node (strongly connected if all traversals visit all nodes)
  - Time complexity:  $O(n(n + m))$
- A better one:
  - traverse the graph from an arbitrary node
  - reverse all edges, traverse again
  - intuition: if there is a reverse path from D to A, then D is reachable from A
- Time complexity:  $O(n + m)$
- Note: we do not need to copy the graph, we only need to do 'reverse edge' queries



# Transitive closure

- We know that graph traversals answer reachability questions about two nodes efficiently
- Pre-computing all nodes reachable from every other node is beneficial in some applications
- The *transitive closure* of a graph is another graph where
  - The set of nodes are the same as the original graph
  - There is an edge between two nodes  $u$  and  $v$  if  $v$  is reachable from  $u$
- For an undirected graph, transitive closure can be computed by computing the connected components

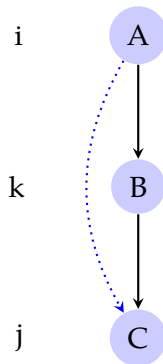
# Computing transitive closure on directed graphs

- A straightforward algorithm:
  - run  $n$  graph traversals, from each node in the graph,
  - add an edge between the start node to any node discovered by the traversal
  - time complexity is  $O(n(n + m))$
- Floyd-Warshall algorithm is another well-known algorithm that runs more efficiently in some settings

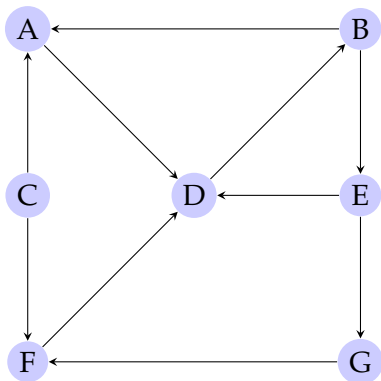
# Floyd-Warshall algorithm

for finding transitive closure

- Remember that transitive closure of a graph is another graph
- Floyd-Warshall algorithm is an iterative algorithm that computes the transitive closure in  $n$  iterations
- The algorithm starts with setting transitive closure to the original graph
- For  $k = 1 \dots n$ 
  - Add a directed edge  $(v_i, v_j)$  to transitive closure if it already contains both  $(v_i, v_k)$  and  $(v_k, v_j)$
- It is efficient if graph is implemented with an adjacency matrix and it is not sparse

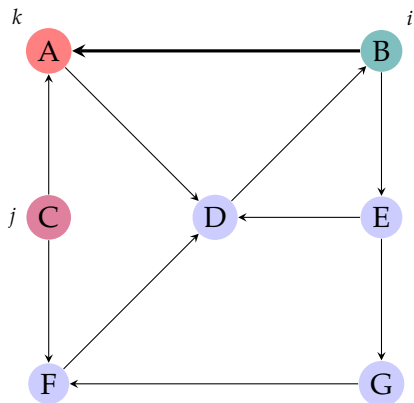


# Floyd-Warshall demonstration



	A	B	C	D	E	F	G
A	F	F	F	T	F	F	F
B	T	F	F	F	T	F	F
C	T	F	F	F	F	T	F
D	F	T	F	F	F	F	F
E	F	F	F	T	F	F	T
F	F	F	F	T	F	F	F
G	F	F	F	F	F	T	F

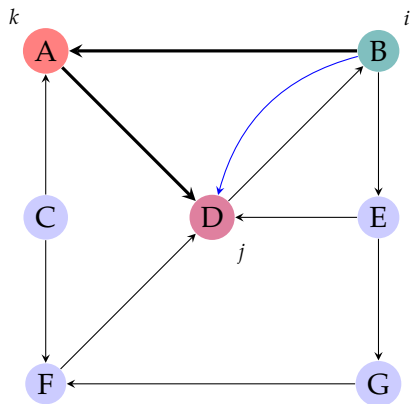
# Floyd-Warshall demonstration



	A	B	C	D	E	F	G
A	F	F	F	T	F	F	F
B	T	F	F	F	T	F	F
C	T	F	F	F	F	T	F
D	F	T	F	F	F	F	F
E	F	F	F	T	F	F	T
F	F	F	F	T	F	F	F
G	F	F	F	F	F	T	F

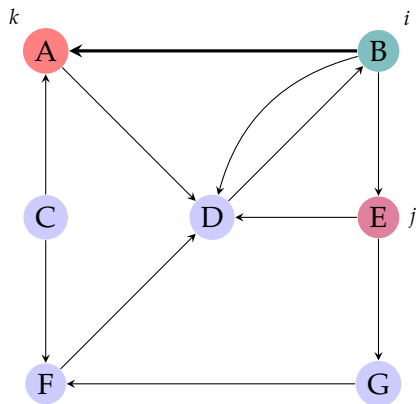


# Floyd-Warshall demonstration



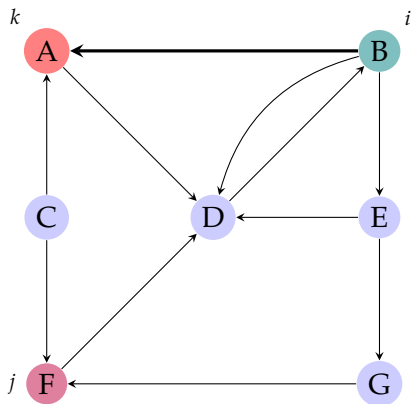
	A	B	C	D	E	F	G
A	F	F	F	T	F	F	F
B	T	F	F	T	T	F	F
C	T	F	F	F	F	T	F
D	F	T	F	F	F	F	F
E	F	F	F	T	F	F	T
F	F	F	F	T	F	F	F
G	F	F	F	F	F	T	F

# Floyd-Warshall demonstration



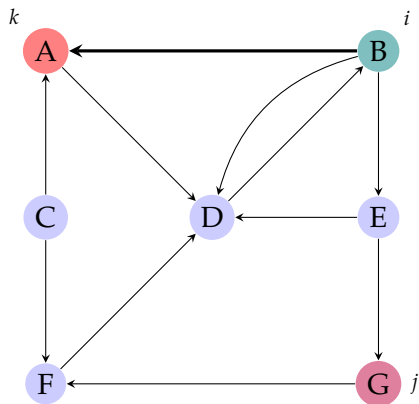
	A	B	C	D	E	F	G
A	F	F	F	T	F	F	F
B	T	F	F	T	T	F	F
C	T	F	F	F	F	T	F
D	F	T	F	F	F	F	F
E	F	F	F	T	F	F	T
F	F	F	F	T	F	F	F
G	F	F	F	F	F	T	F

# Floyd-Warshall demonstration



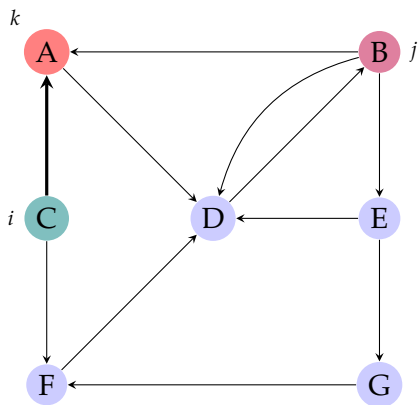
	A	B	C	D	E	F	G
A	F	F	F	T	F	F	F
B	T	F	F	T	T	F	F
C	T	F	F	F	F	T	F
D	F	T	F	F	F	F	F
E	F	F	F	T	F	F	T
F	F	F	F	T	F	F	F
G	F	F	F	F	F	T	F

# Floyd-Warshall demonstration



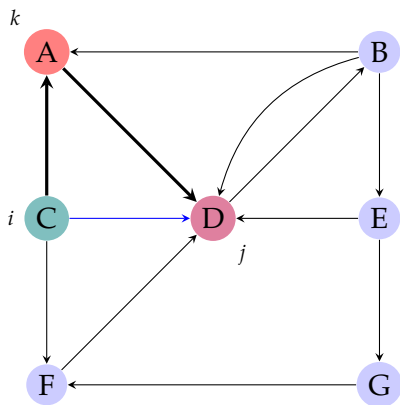
	A	B	C	D	E	F	G
A	F	F	F	T	F	F	F
B	T	F	F	T	T	F	F
C	T	F	F	F	F	T	F
D	F	T	F	F	F	F	F
E	F	F	F	T	F	F	T
F	F	F	F	T	F	F	F
G	F	F	F	F	F	T	F

# Floyd-Warshall demonstration



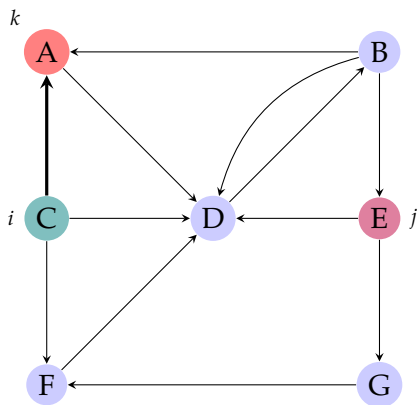
	A	B	C	D	E	F	G
A	F	F	F	T	F	F	F
B	T	F	F	T	T	F	F
C	T	F	F	F	F	T	F
D	F	T	F	F	F	F	F
E	F	F	F	T	F	F	T
F	F	F	F	T	F	F	F
G	F	F	F	F	F	T	F

# Floyd-Warshall demonstration



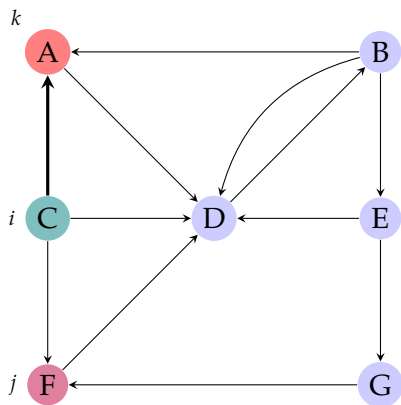
	A	B	C	D	E	F	G
A	F	F	F	T	F	F	F
B	T	F	F	T	T	F	F
C	T	F	F	T	F	T	F
D	F	T	F	F	F	F	F
E	F	F	F	T	F	F	T
F	F	F	F	T	F	F	F
G	F	F	F	F	F	T	F

# Floyd-Warshall demonstration



	A	B	C	D	E	F	G
A	F	F	F	T	F	F	F
B	T	F	F	T	T	F	F
C	T	F	F	T	F	T	F
D	F	T	F	F	F	F	F
E	F	F	F	T	F	F	T
F	F	F	F	T	F	F	F
G	F	F	F	F	F	T	F

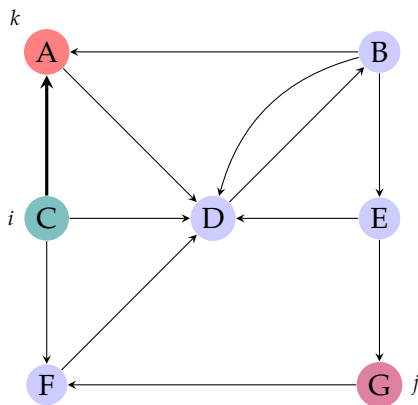
# Floyd-Warshall demonstration



	A	B	C	D	E	F	G
A	F	F	F	T	F	F	F
B	T	F	F	T	T	F	F
C	T	F	F	T	F	T	F
D	F	T	F	F	F	F	F
E	F	F	F	T	F	F	T
F	F	F	F	T	F	F	F
G	F	F	F	F	F	T	F

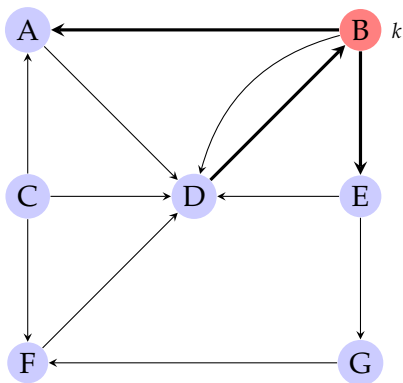


# Floyd-Warshall demonstration



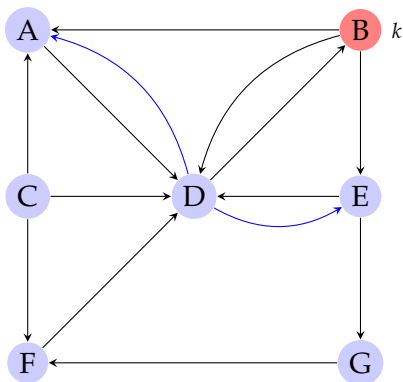
	A	B	C	D	E	F	G
A	F	F	F	T	F	F	F
B	T	F	F	T	T	F	F
C	T	F	F	T	F	T	F
D	F	T	F	F	F	F	F
E	F	F	F	T	F	F	T
F	F	F	F	T	F	F	F
G	F	F	F	F	F	T	F

# Floyd-Warshall demonstration



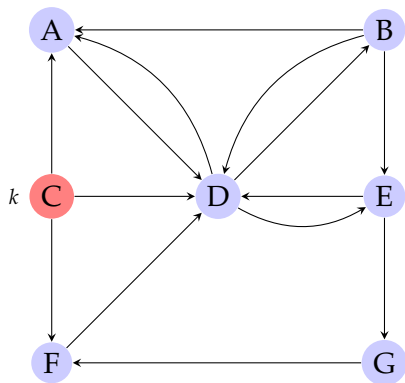
	A	B	C	D	E	F	G
A	F	F	F	T	F	F	F
B	T	F	F	T	T	F	F
C	T	F	F	T	F	T	F
D	F	T	F	F	F	F	F
E	F	F	F	T	F	F	T
F	F	F	F	T	F	F	F
G	F	F	F	F	F	T	F

# Floyd-Warshall demonstration



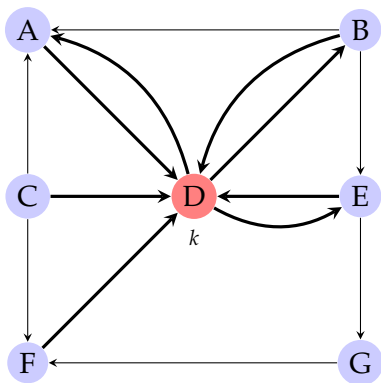
	A	B	C	D	E	F	G
A	F	F	F	T	F	F	F
B	T	F	F	T	T	F	F
C	T	F	F	T	F	T	F
D	T	T	F	F	T	F	F
E	F	F	F	T	F	F	T
F	F	F	F	T	F	F	F
G	F	F	F	F	F	T	F

# Floyd-Warshall demonstration



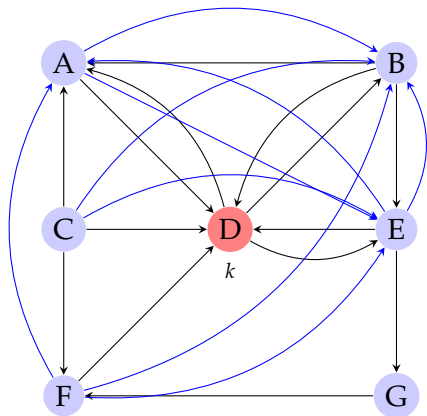
	A	B	C	D	E	F	G
A	F	F	F	T	F	F	F
B	T	F	F	T	T	F	F
C	T	F	F	T	F	T	F
D	T	T	F	F	T	F	F
E	F	F	F	T	F	F	T
F	F	F	F	T	F	F	F
G	F	F	F	F	F	T	F

# Floyd-Warshall demonstration



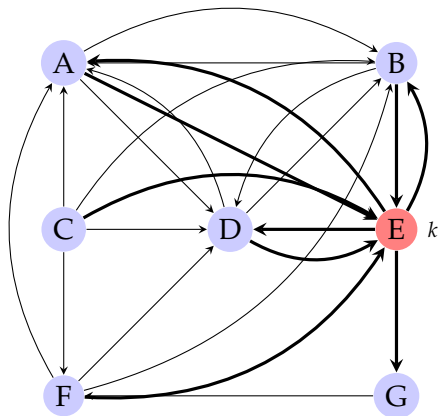
	A	B	C	D	E	F	G
A	F	F	F	T	F	F	F
B	T	F	F	T	T	F	F
C	T	F	F	T	F	T	F
D	T	T	F	F	T	F	F
E	F	F	F	T	F	F	T
F	F	F	F	T	F	F	F
G	F	F	F	F	F	T	F

# Floyd-Warshall demonstration



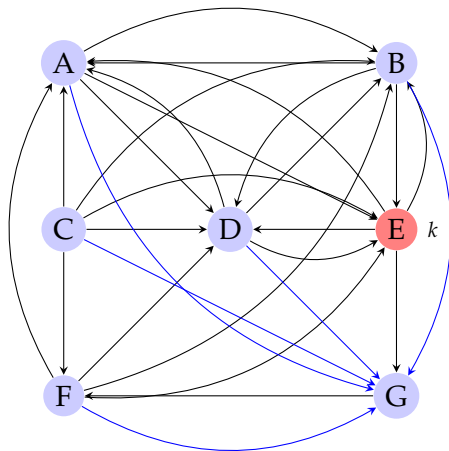
	A	B	C	D	E	F	G
A	F	T	F	T	T	F	F
B	T	F	F	T	T	F	F
C	T	T	F	T	T	T	F
D	T	T	F	F	T	F	F
E	T	T	F	T	F	F	T
F	T	T	F	T	T	F	F
G	F	F	F	F	F	T	F

# Floyd-Warshall demonstration



	A	B	C	D	E	F	G
A	F	T	F	T	T	F	F
B	T	F	F	T	T	F	F
C	T	T	F	T	T	T	F
D	T	T	F	F	T	F	F
E	T	T	F	T	F	F	T
F	T	T	F	T	T	F	F
G	F	F	F	F	F	T	F

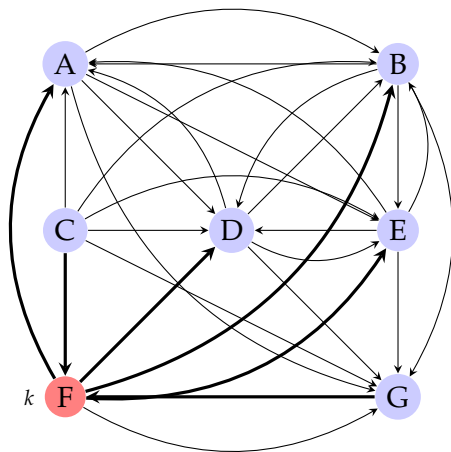
# Floyd-Warshall demonstration



	A	B	C	D	E	F	G
A	F	T	F	T	T	F	T
B	T	F	F	T	T	F	T
C	T	T	F	T	T	T	T
D	T	T	F	F	T	F	T
E	T	T	F	T	F	F	T
F	T	T	F	T	T	F	T
G	F	F	F	F	F	T	F

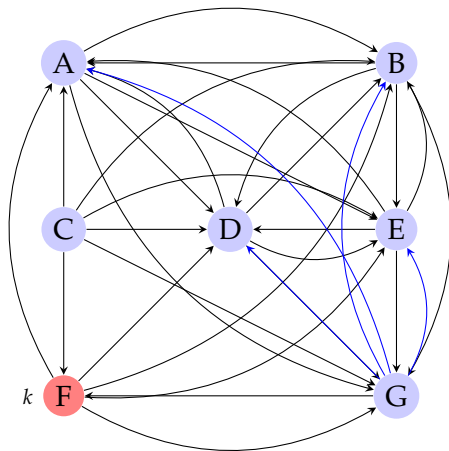


# Floyd-Warshall demonstration



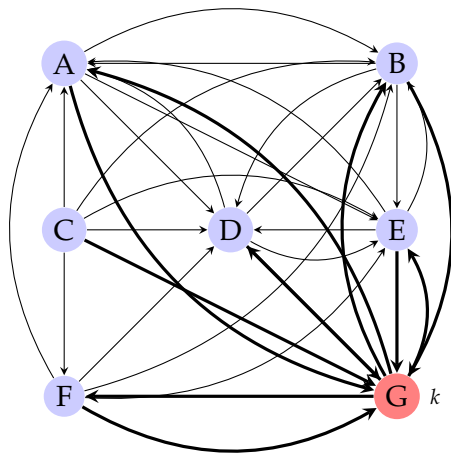
	A	B	C	D	E	F	G
A	F	T	F	T	T	F	T
B	T	F	F	T	T	F	T
C	T	T	F	T	T	T	T
D	T	T	F	F	T	F	T
E	T	T	F	T	F	F	T
F	T	T	F	T	T	F	T
G	F	F	F	F	F	T	F

# Floyd-Warshall demonstration



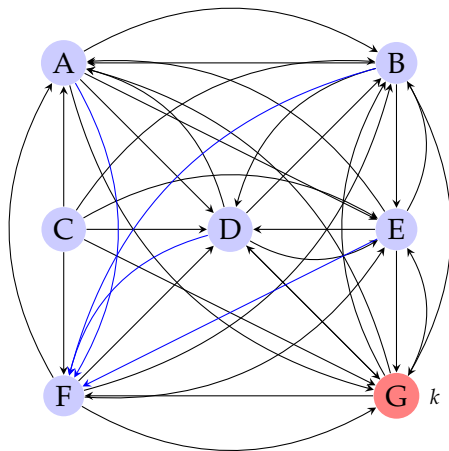
	A	B	C	D	E	F	G
A	F	T	F	T	T	F	T
B	T	F	F	T	T	F	T
C	T	T	F	T	T	T	T
D	T	T	F	F	T	F	T
E	T	T	F	T	F	F	T
F	T	T	F	T	T	F	T
G	T	T	F	T	T	T	F

# Floyd-Warshall demonstration



	A	B	C	D	E	F	G
A	F	T	F	T	T	F	T
B	T	F	F	T	T	F	T
C	T	T	F	T	T	T	T
D	T	T	F	F	T	F	T
E	T	T	F	T	F	F	T
F	T	T	F	T	T	F	T
G	T	T	F	T	T	T	F

# Floyd-Warshall demonstration



	A	B	C	D	E	F	G
A	F	T	F	T	T	T	T
B	T	F	F	T	T	T	T
C	T	T	F	T	T	T	T
D	T	T	F	F	T	T	T
E	T	T	F	T	F	T	T
F	T	T	F	T	T	F	T
G	T	T	F	T	T	T	F

# Floyd-Warshall algorithm

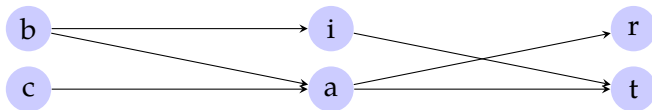
adjacency matrix implementation

```
T = [row[:] for row in G]
for k in range(n):
    for i in range(n):
        if i == k: continue
        for j in range(n):
            if j == i or j == k:
                continue
            T[i][j] = T[i][j] or \
                T[i][k] and T[k][j]
```

- Time complexity is  $O(n^3)$
- Compare with repeated traversal:  
 $O(n(n + m))$ 
  - Note that in a dense graph  $m$  is  $O(n^2)$
- A version of this algorithm is also used for finding shortest paths in weighted graphs (later in the course)

# Directed acyclic graphs

- *Directed acyclic graphs* (DAGs) are directed graphs without cycles
- DAGs have many practical applications (mainly, dependency graphs)
  - Prerequisites between courses in a study program
  - Class inheritance in an object-oriented program
  - Scheduling constraints over tasks in a project
  - Dependency parser output (generally trees, but can also be more general DAGs)
  - A compact representation of a list of words:



# Directed acyclic graphs

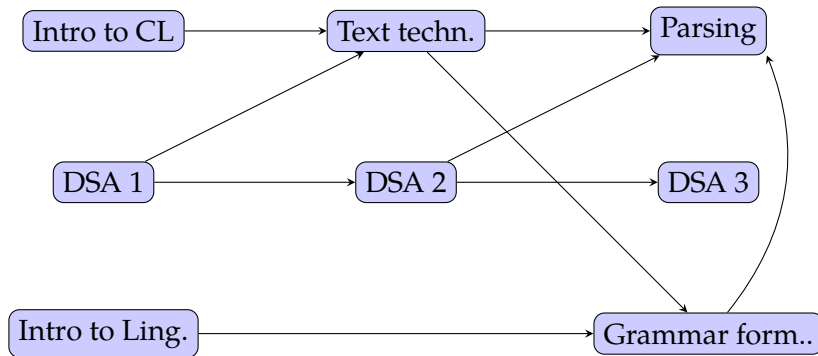
PAGE 3

DEPARTMENT	COURSE	DESCRIPTION	PREREQS
COMPUTER SCIENCE	CPSC 432	INTERMEDIATE COMPILER DESIGN, WITH A FOCUS ON DEPENDENCY RESOLUTION.	CPSC 432

<https://www.xkcd.com/754/>

# DAG example

a (hypothetical) course prerequisite graph



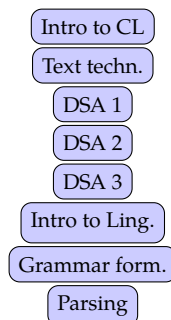
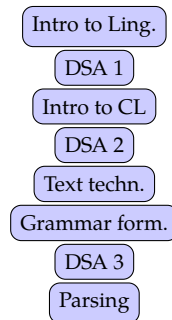
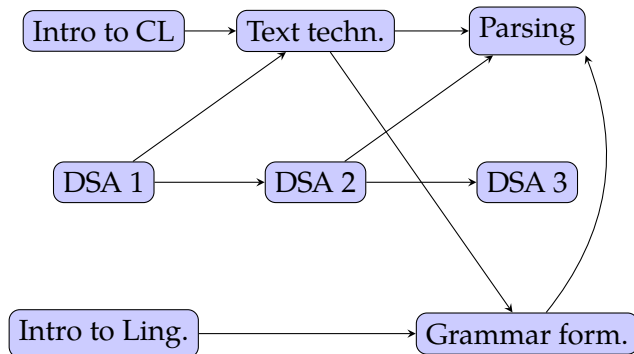


# Topological order

- A *topological ordering* of a directed graph is a sequence of nodes such that for every directed edge  $(u, v)$   $u$  is listed before  $v$
- A topological ordering lists 'prerequisites' of a node before listing the node itself
- There may be multiple topological orderings
- In the course prerequisite example, a topological ordering lists any acceptable order that the courses can be taken

# Topological order example

course prerequisites – two alternative topological orders



# Topological sort

## algorithm

```

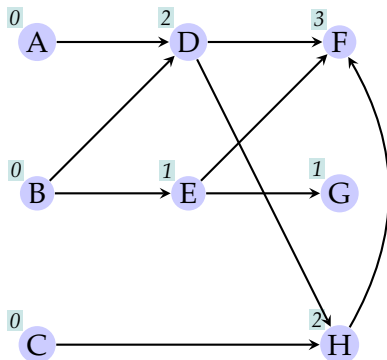
topo, ready = [], []
incount = {}
for u in nodes:
    incount[u] = u.outdegree()
    if incount[u] == 0:
        ready.append(u)
while len(ready) > 0:
    u = ready.pop()
    topo.append(u)
    for v in u.neighbors():
        incount[v] -= 1
    if incount[v] == 0:
        ready.append(v)

```

- Keep record of number of incoming edges
- A node is ready to be placed in the sorted list if there no unprocessed incoming edges
- Running time is  $O(n + m)$
- If the topological ordering does not contain all the edges, the graph includes a cycle

# Topological sort

demonstration



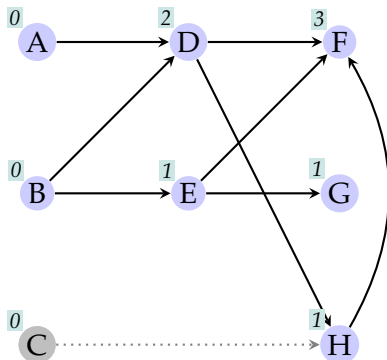
ready

A
B
C

sorted


# Topological sort

## demonstration



ready

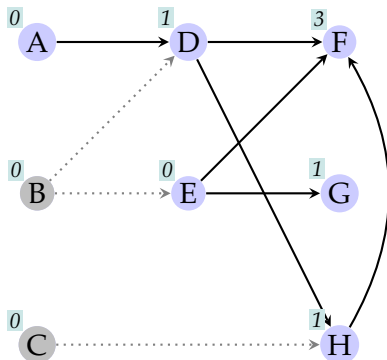
A  
B

sorted

C

# Topological sort

## demonstration



ready

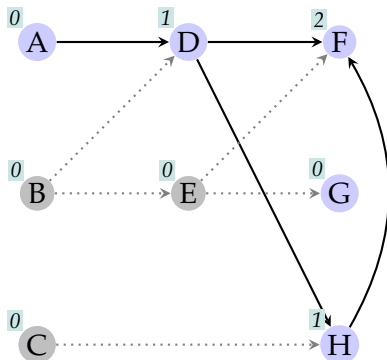
A
E

sorted

C
B

# Topological sort

## demonstration



ready

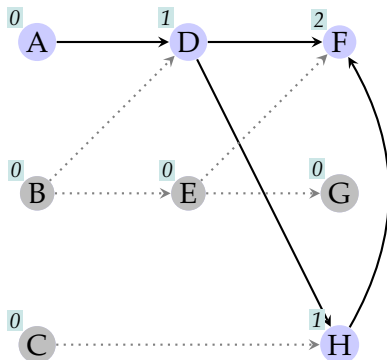
A  
G

sorted

C  
B  
E

# Topological sort

## demonstration



ready  
A

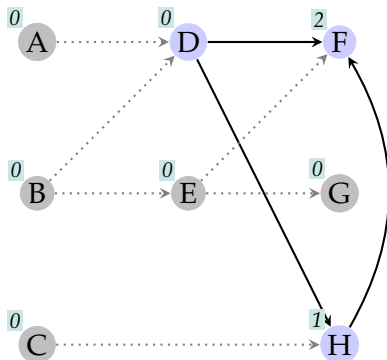
sorted

C
B
E
G



# Topological sort

## demonstration



ready

D

sorted

C

B

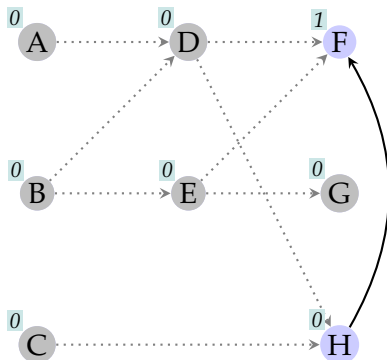
E

G

A

# Topological sort

## demonstration



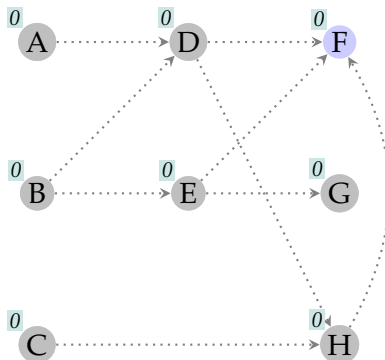
ready  
H

sorted

C
B
E
G
A
D

# Topological sort

## demonstration



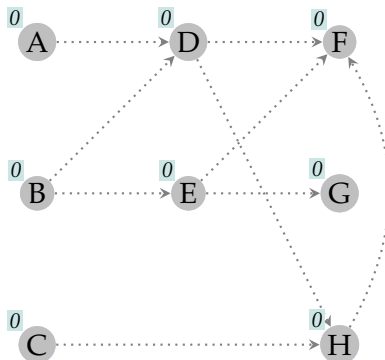
ready  
F

sorted

C
B
E
G
A
D
H

# Topological sort

## demonstration



ready

sorted

C
B
E
G
A
D
H
F

# Summary

- Some operations on directed graphs are more challenging,
- We covered
  - Finding strongly connected components
  - Finding the transitive closure of a digraph
  - DAGs and topological ordering
- Reading on graphs: Goodrich, Tamassia, and Goldwasser (2013, chapter 14)

Next:

- More on graphs: shortest paths, minimum spanning trees

# Acknowledgments, credits, references



Goodrich, Michael T., Roberto Tamassia, and Michael H. Goldwasser (2013).  
*Data Structures and Algorithms in Python*. John Wiley & Sons, Incorporated. ISBN:  
9781118476734.









