Algorithm Bootcamp

July 2024

Day 2: Graphs

Dr. Christopher Weyand
Optimization Expert
Fleet Optimization

David Stangl
Software Engineer
Fleet Optimization

Basics

What is a graph?

$G = (V, E)$

V = set of objects

E = connections

Basics

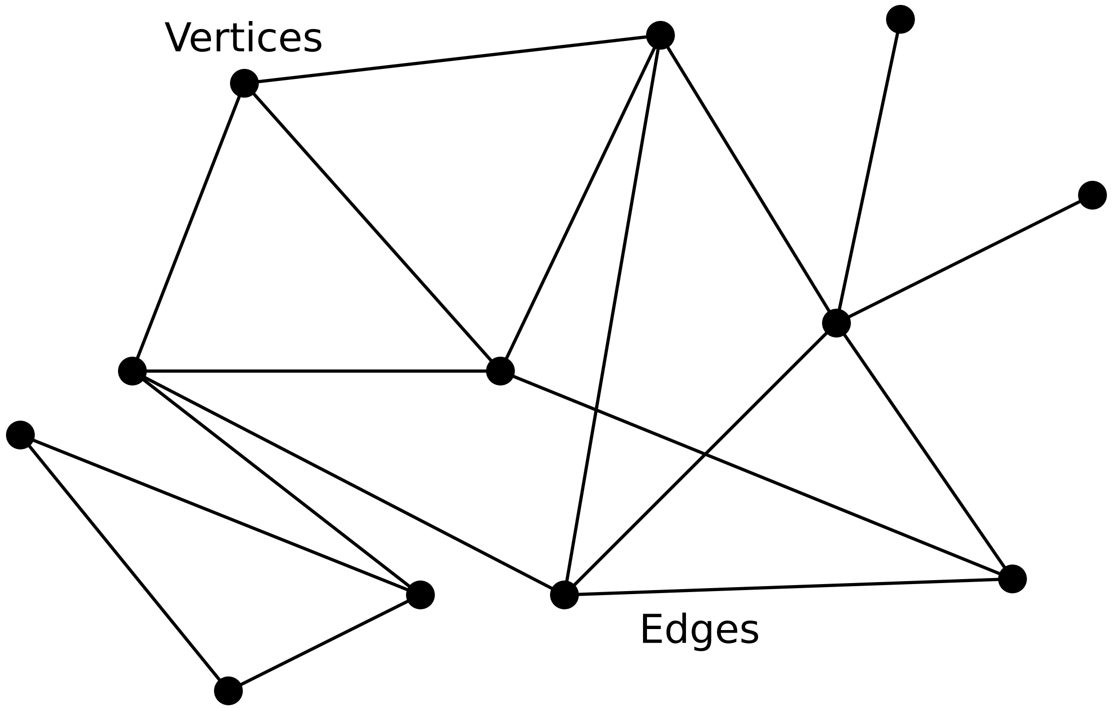
What is a graph?

$G = (V, E)$

V = set of objects

E = connections

undirected $\quad E \subseteq \{\{u, v\} \mid u, v \in V\}$



Basics

What is a graph?

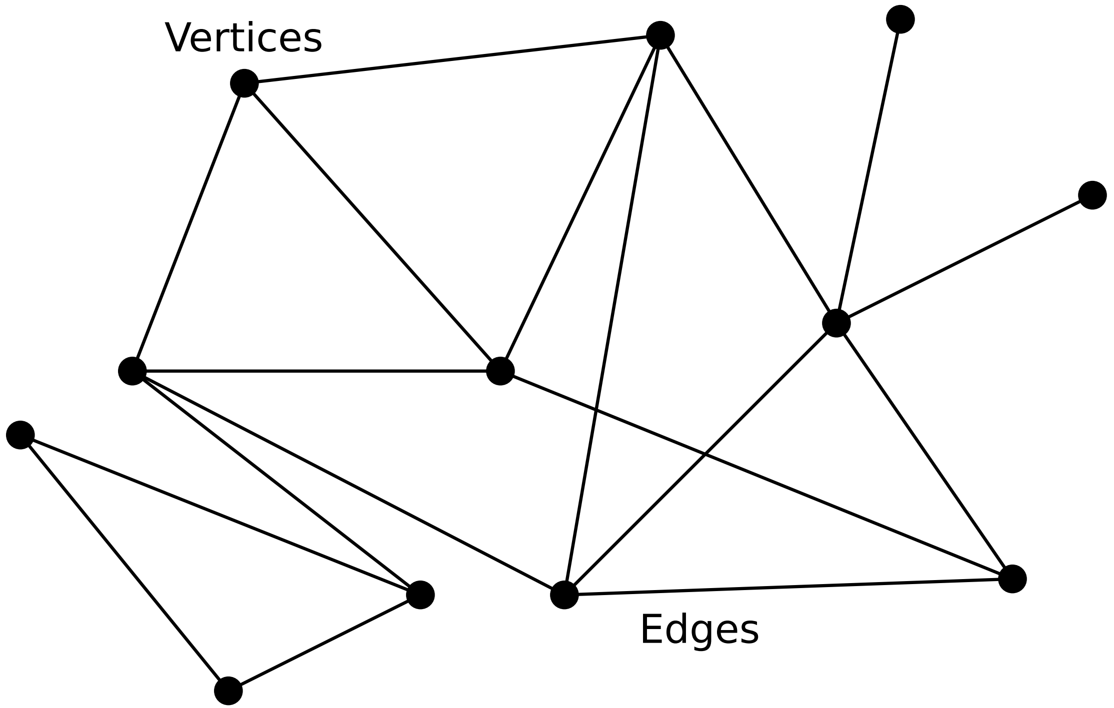
$G = (V, E)$

V = set of objects

E = connections

undirected $\quad E \subseteq \{\{u, v\} \mid u, v \in V\}$

acyclic $\rightarrow$



Basics

What is a graph?

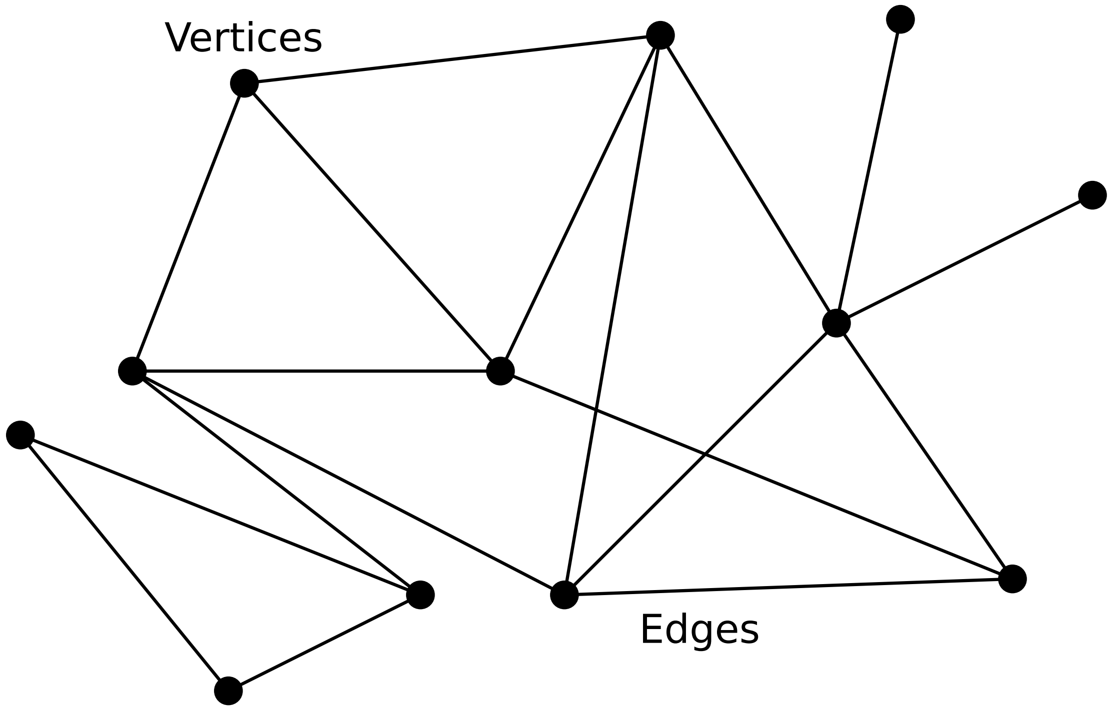
$G = (V, E)$

V = set of objects

E = connections

undirected $E \subseteq \{\{u, v\} \mid u, v \in V\}$

acyclic $\rightarrow$ forest



Basics

What is a graph?

$G = (V, E)$

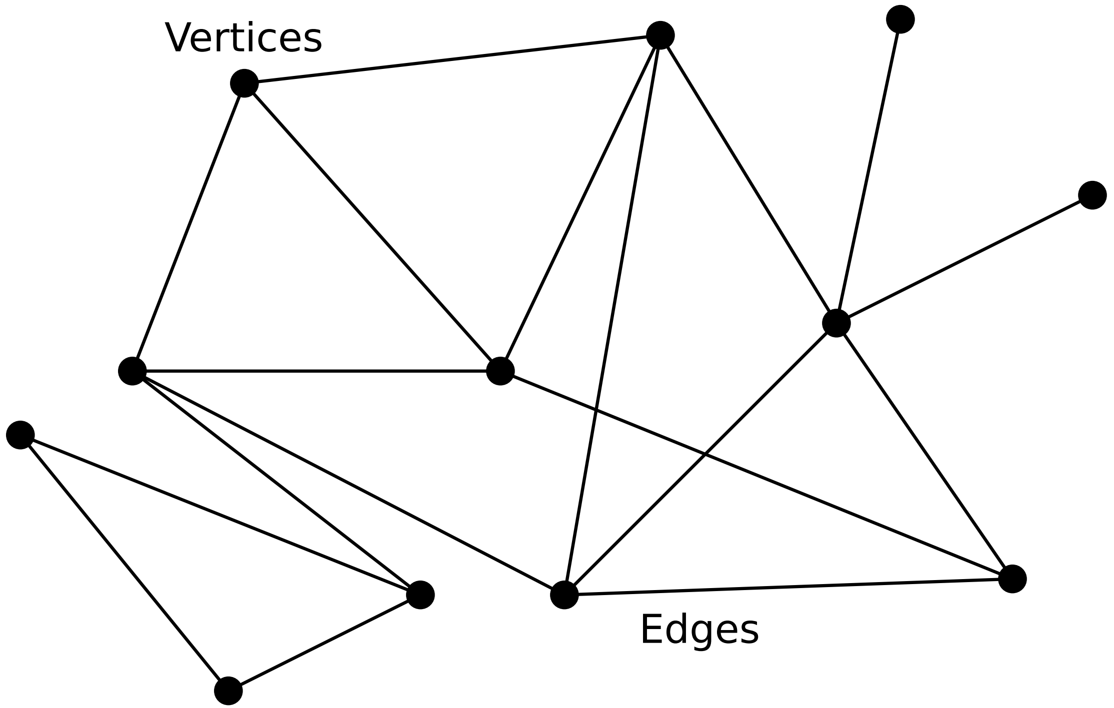
V = set of objects

E = connections

undirected $E \subseteq \{\{u, v\} \mid u, v \in V\}$

acyclic $\rightarrow$ forest

connected $\rightarrow$ tree



Basics

What is a graph?

$G = (V, E)$

V = set of objects

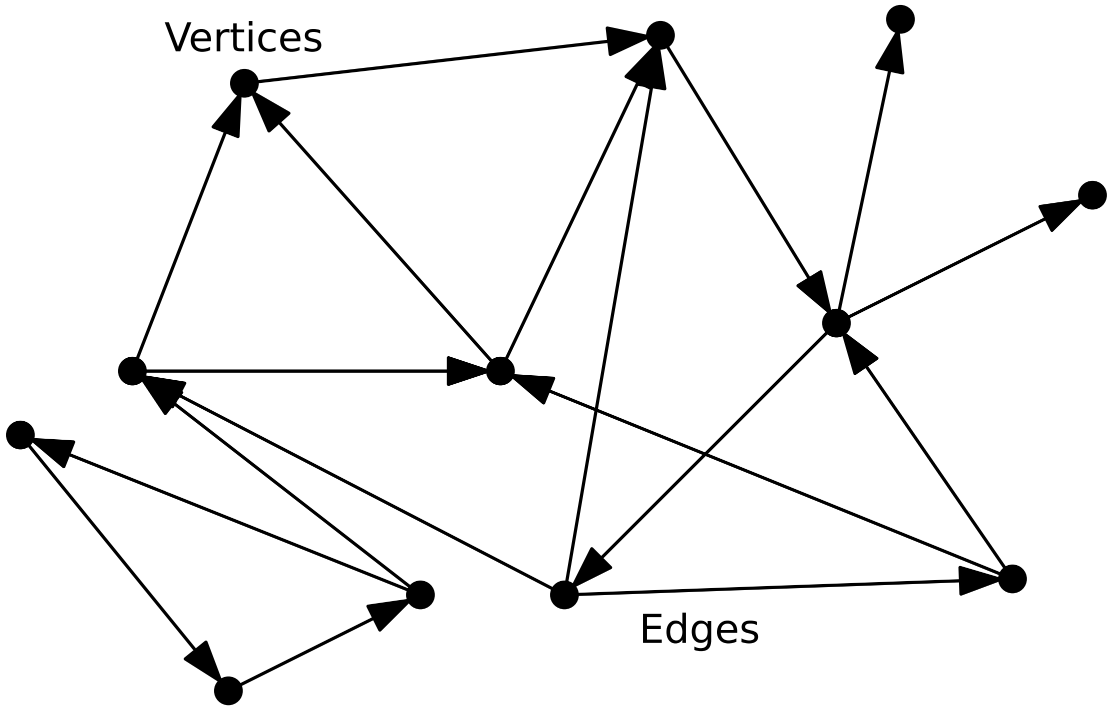
E = connections

undirected $\quad E \subseteq \{\{u, v\} \mid u, v \in V\}$

acyclic $\rightarrow$ forest

connected $\rightarrow$ tree

directed $\quad E \subseteq \{(u, v) \mid u, v \in V\}$



Basics

What is a graph?

$G = (V, E)$

V = set of objects

E = connections

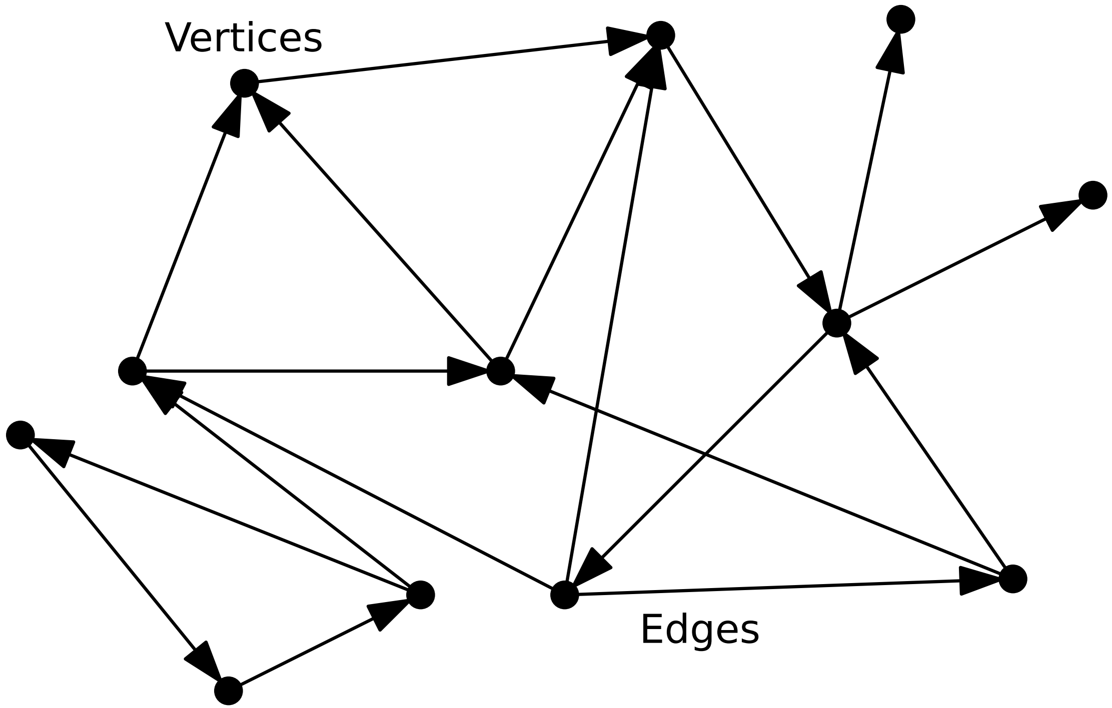
undirected $\quad E \subseteq \{\{u, v\} \mid u, v \in V\}$

acyclic $\rightarrow$ forest

connected $\rightarrow$ tree

directed $\quad E \subseteq \{(u, v) \mid u, v \in V\}$

acyclic $\rightarrow$ DAG



Basics

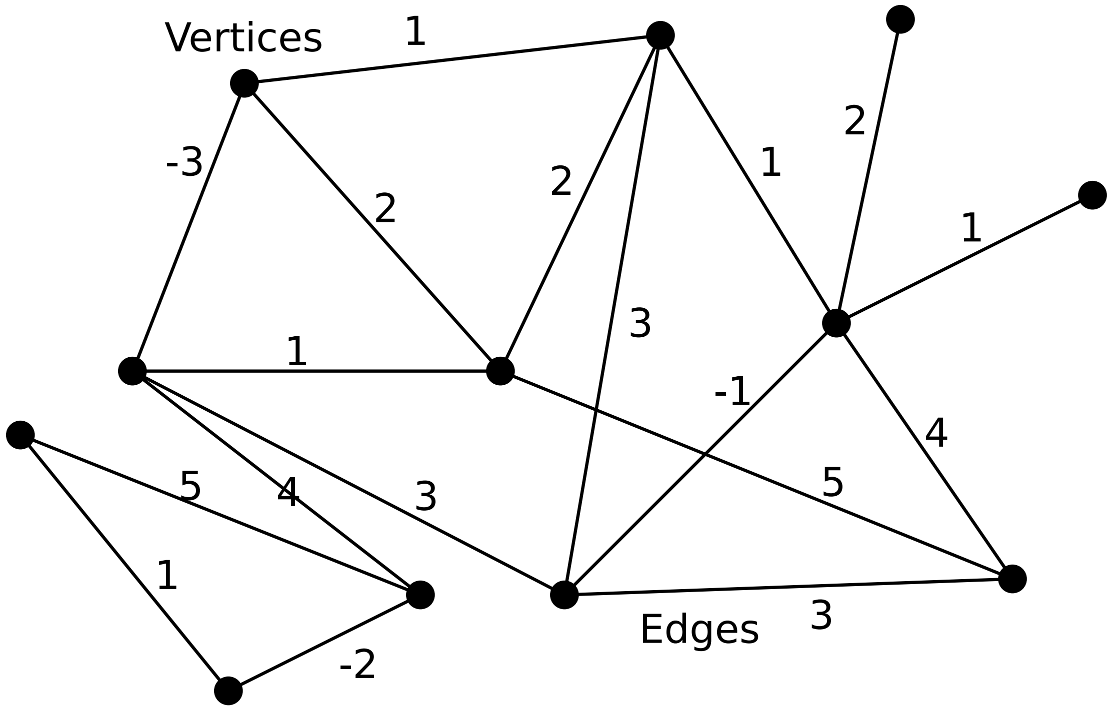
What is a graph?

$G = (V, E)$

V = set of objects

E = connections

undirected	$E \subseteq \{\{u, v\} \mid u, v \in V\}$
acyclic $\rightarrow$	forest
connected $\rightarrow$	tree
directed	$E \subseteq \{(u, v) \mid u, v \in V\}$
acyclic $\rightarrow$	DAG
weighted	$w : E \mapsto \mathbb{Z}$



Basics

What is a graph?

$G = (V, E)$

V = set of objects

E = connections

undirected $\quad E \subseteq \{\{u, v\} \mid u, v \in V\}$

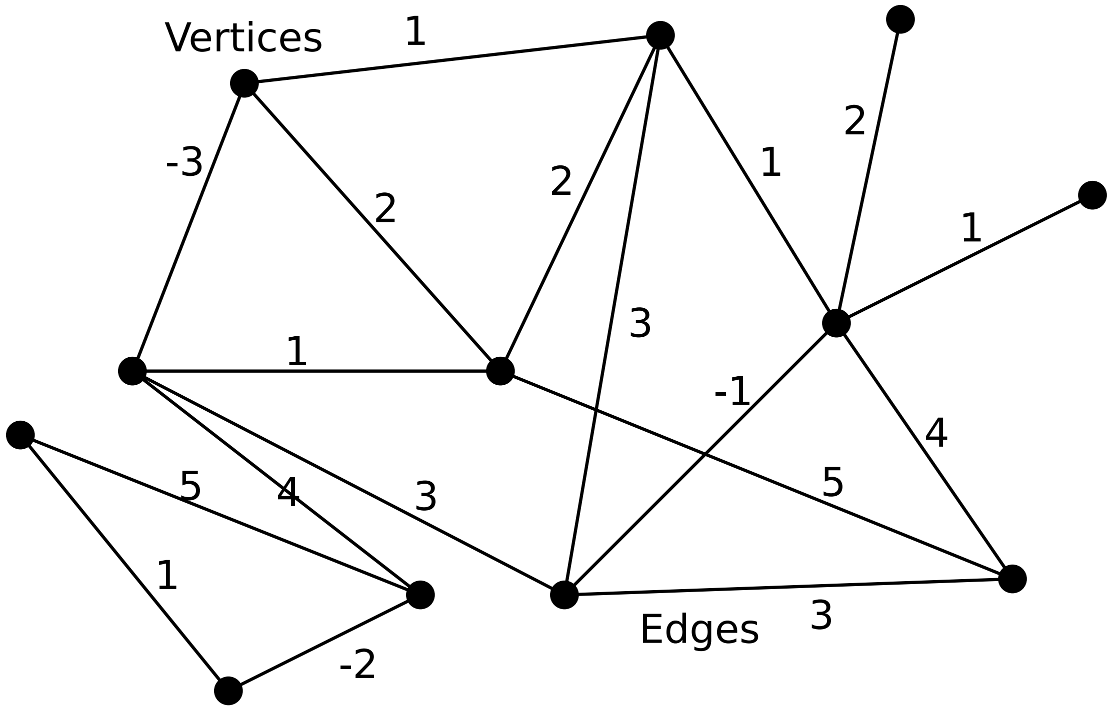
acyclic $\rightarrow$ forest

connected $\rightarrow$ tree

directed $\quad E \subseteq \{(u, v) \mid u, v \in V\}$

acyclic $\rightarrow$ DAG

weighted $\quad w : E \mapsto \mathbb{Z}$



usually: $|V| = n \quad |E| = m$

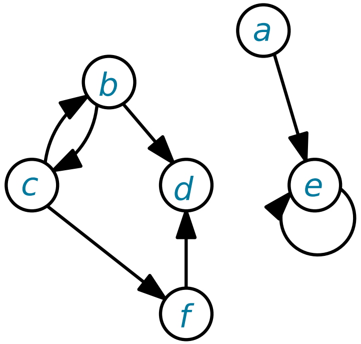
Exceptions

tree with cycle

connected forest (Pando)

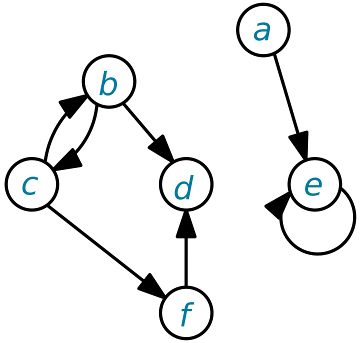
Representation and Storage

directed Graph

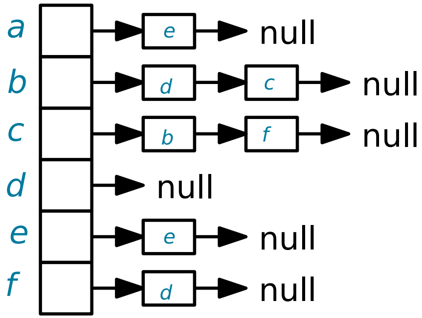


Representation and Storage

directed Graph

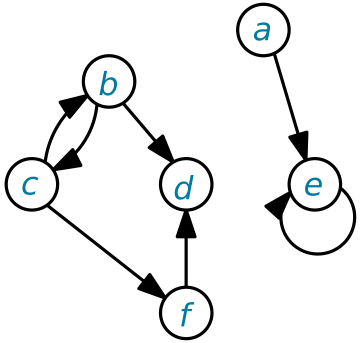


adjacency list

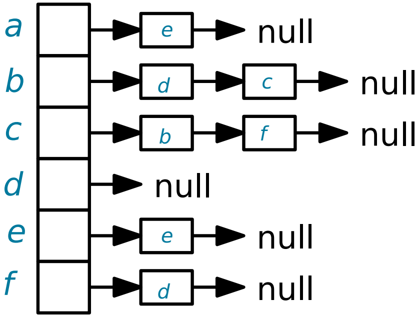


Representation and Storage

directed Graph



adjacency list

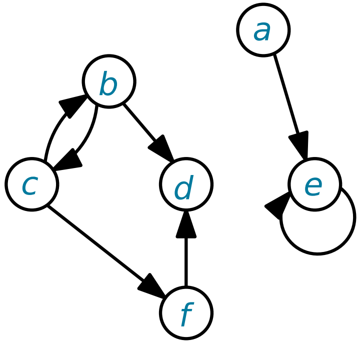


adjacency matrix

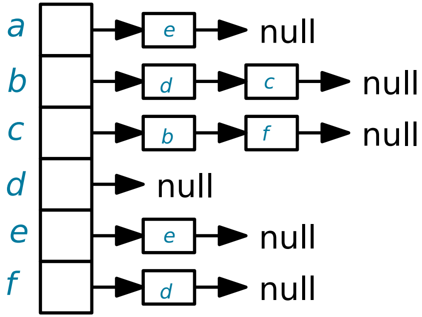
$$
\begin{array}{c|cccccc}
 & a & b & c & d & e & f \\
\hline
a & 0 & 0 & 0 & 0 & 1 & 0 \\
b & 0 & 0 & 1 & 1 & 0 & 0 \\
c & 0 & 1 & 0 & 0 & 0 & 1 \\
d & 0 & 0 & 0 & 0 & 0 & 0 \\
e & 0 & 0 & 0 & 0 & 1 & 0 \\
f & 0 & 0 & 0 & 1 & 0 & 0
\end{array}
$$

Representation and Storage

directed Graph



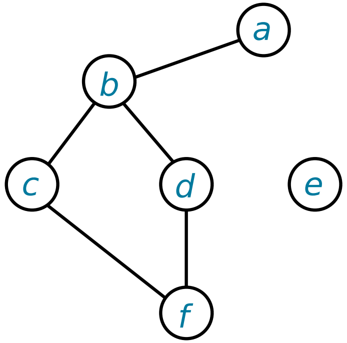
adjacency list



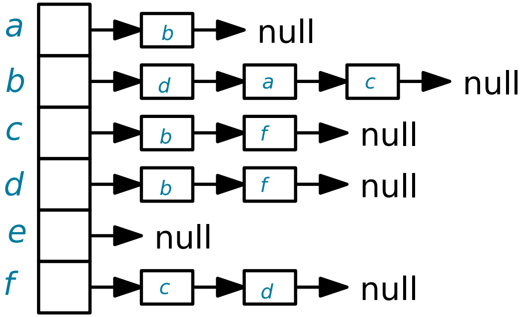
adjacency matrix

$$\begin{array}{cc} & \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array} & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

undirected Graph



adjacency list

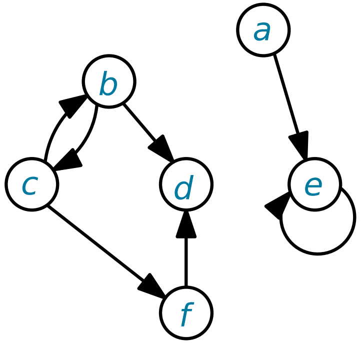


adjacency matrix

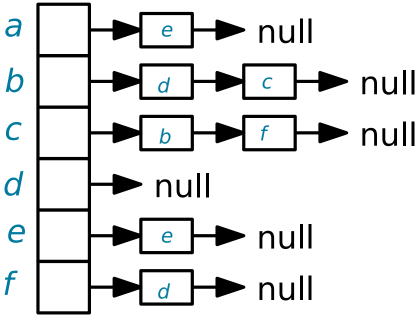
$$\begin{array}{cc} & \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array} & \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Representation and Storage

directed Graph



adjacency list

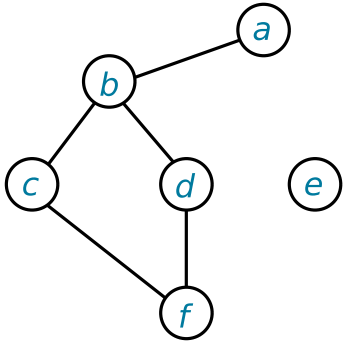


adjacency matrix

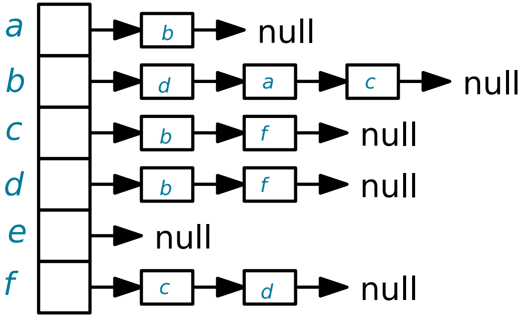
$$\begin{array}{c|cccccc} & a & b & c & d & e & f \\ \hline a & 0 & 0 & 0 & 0 & 1 & 0 \\ b & 0 & 0 & 1 & 1 & 0 & 0 \\ c & 0 & 1 & 0 & 0 & 0 & 1 \\ d & 0 & 0 & 0 & 0 & 0 & 0 \\ e & 0 & 0 & 0 & 0 & 1 & 0 \\ f & 0 & 0 & 0 & 1 & 0 & 0 \end{array}$$

What about weighted graphs?

undirected Graph



adjacency list

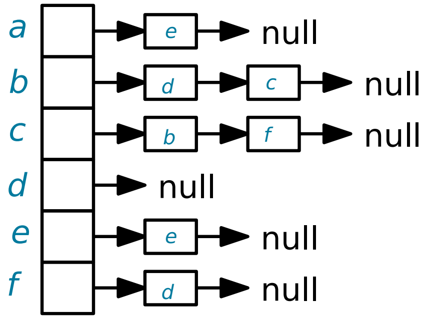


adjacency matrix

$$\begin{array}{c|cccccc} & a & b & c & d & e & f \\ \hline a & 0 & 1 & 0 & 0 & 0 & 0 \\ b & 1 & 0 & 1 & 1 & 0 & 0 \\ c & 0 & 1 & 0 & 0 & 0 & 1 \\ d & 0 & 1 & 0 & 0 & 0 & 1 \\ e & 0 & 0 & 0 & 0 & 0 & 0 \\ f & 0 & 0 & 1 & 1 & 0 & 0 \end{array}$$

Pros & Cons

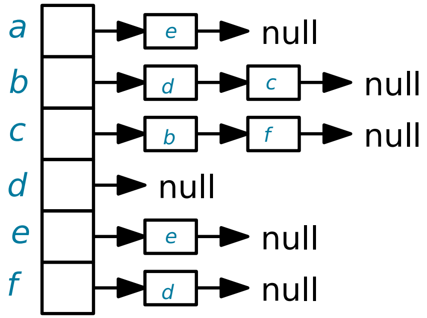
adjacency list | adjacency matrix



$$\begin{array}{c|c} & \begin{matrix} a & b & c & d & e & f \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \end{matrix} & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Pros & Cons

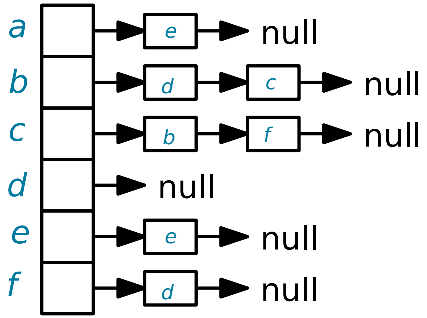
	adjacency list	adjacency matrix
memory		



$$
\begin{array}{c|c}
 & \begin{matrix} a & b & c & d & e & f \end{matrix} \\
\begin{matrix} a \\ b \\ c \\ d \\ e \\ f \end{matrix} &
\begin{pmatrix}
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0
\end{pmatrix}
\end{array}
$$

Pros & Cons

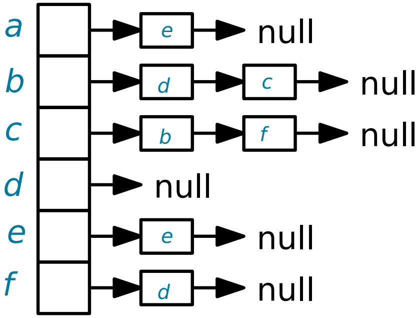
	adjacency list	adjacency matrix
memory	$O(n+m)$	$O(n^2)$



$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array}\begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Pros & Cons

	adjacency list	adjacency matrix
memory	$O(n + m)$	$O(n^2)$
find outgoing edges		

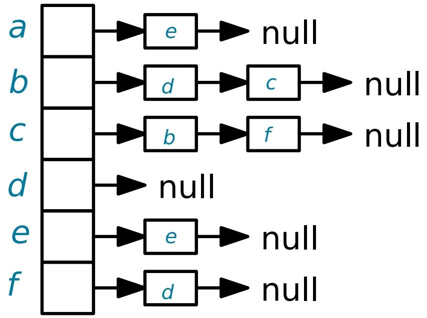


$$\begin{array}{c|c} & \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array} & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

$$N(v) = \{u \mid (v, u) \in E\}$$

Pros & Cons

	adjacency list	adjacency matrix
memory	$O(n+m)$	$O(n^2)$
find outgoing edges	$O(\|N(v)\|)$	$O(n)$

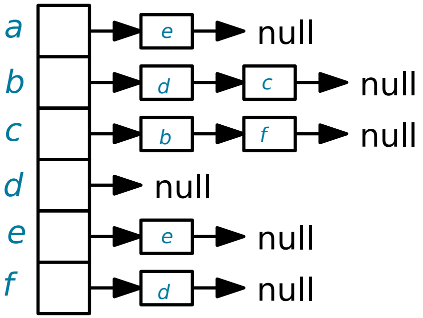


$$\begin{array}{c c} & \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array} & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

$$N(v) = \{u \mid (v,u) \in E\}$$

Pros & Cons

	adjacency list	adjacency matrix
memory	$O(n+m)$	$O(n^2)$
find outgoing edges	$O(\|N(v)\|)$	$O(n)$
find incoming edges		

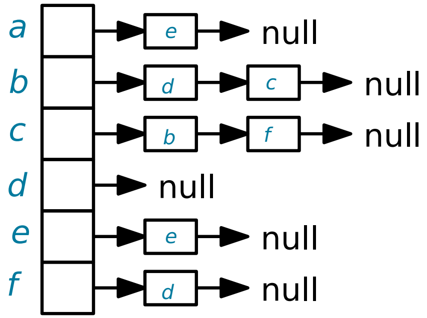


$$\begin{array}{c|c} & \begin{matrix} a & b & c & d & e & f \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \end{matrix} & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

$$N(v) = \{u \mid (v,u) \in E\}$$

Pros & Cons

	adjacency list	adjacency matrix
memory	$O(n+m)$	$O(n^2)$
find outgoing edges	$O(\|N(v)\|)$	$O(n)$
find incoming edges	$O(n+m)$	$O(n)$

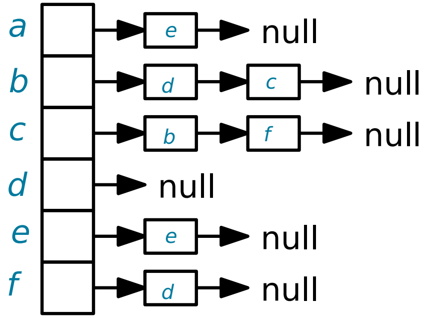


$$
\begin{array}{c|cccccc}
 & a & b & c & d & e & f \\
\hline
\begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array} &
\multicolumn{6}{c}{\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}}
\end{array}
$$

$$N(v) = \{u \mid (v,u) \in E\}$$

Pros & Cons

	adjacency list	adjacency matrix
memory	$O(n+m)$	$O(n^2)$
find outgoing edges	$O(\|N(v)\|)$	$O(n)$
find incoming edges	$O(n+m)$ $\boxed{\text{Improve!}}$	$O(n)$

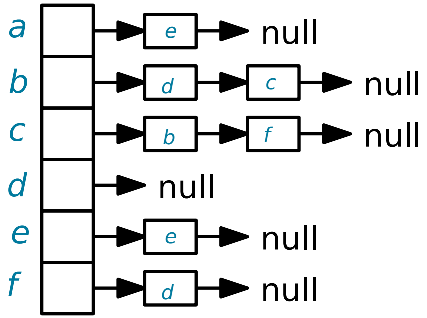


$$
\begin{array}{c c}
 & \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\
\begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array} & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}
\end{array}
$$

$N(v) = \{u \mid (v,u) \in E\}$

Pros & Cons

	adjacency list		adjacency matrix
memory	$O(n+m)$		$O(n^2)$
find outgoing edges	$O(\|N(v)\|)$		$O(n)$
find incoming edges	$O(n+m)$	Improve!	$O(n)$
test edge			

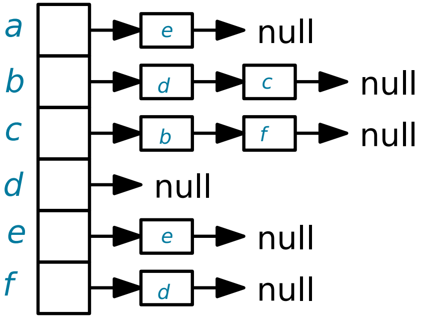


$$\begin{array}{c|c} & \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array} & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

$$N(v) = \{u \mid (v,u) \in E\}$$

Pros & Cons

	adjacency list	adjacency matrix
memory	$O(n+m)$	$O(n^2)$
find outgoing edges	$O(\|N(v)\|)$	$O(n)$
find incoming edges	$O(n+m)$ Improve!	$O(n)$
test edge	$O(\|N(v)\|)$	$O(1)$

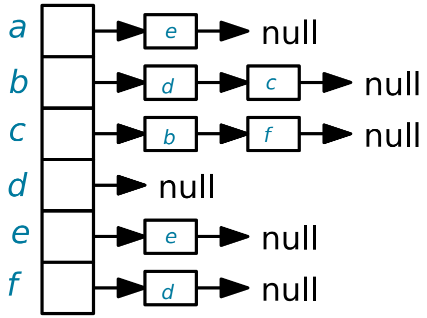


$$\begin{array}{c|c} & \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array} & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

$$N(v) = \{u \mid (v,u) \in E\}$$

Pros & Cons

	adjacency list		adjacency matrix
memory	$O(n+m)$		$O(n^2)$
find outgoing edges	$O(\|N(v)\|)$		$O(n)$
find incoming edges	$O(n+m)$	Improve!	$O(n)$
test edge	$O(\|N(v)\|)$	Improve!	$O(1)$

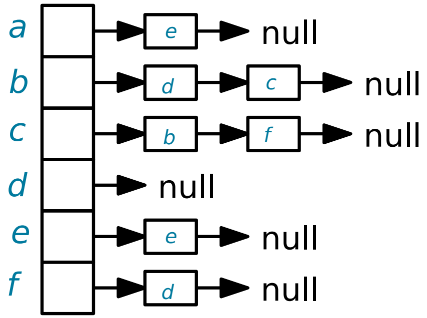


$$\begin{array}{c|cccccc} & a & b & c & d & e & f \\ \hline a & \\ b & \\ c & \\ d & \\ e & \\ f & \end{array} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$N(v) = \{u \mid (v,u) \in E\}$$

Pros & Cons

	adjacency list		adjacency matrix
memory	$O(n+m)$		$O(n^2)$
find outgoing edges	$O(\lvert N(v)\rvert)$		$O(n)$
find incoming edges	$O(n+m)$	Improve!	$O(n)$
test edge	$O(\lvert N(v)\rvert)$	Improve!	$O(1)$
delete edge			

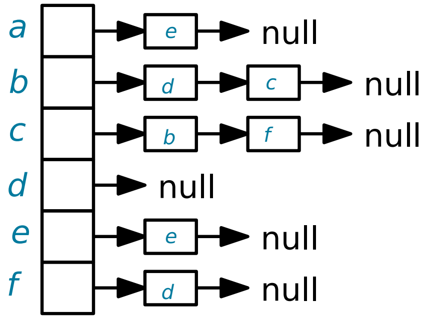


$$
\begin{array}{c}
\\ a \\ b \\ c \\ d \\ e \\ f
\end{array}
\begin{array}{c}
\begin{array}{cccccc} a & b & c & d & e & f \end{array} \\
\begin{pmatrix}
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0
\end{pmatrix}
\end{array}
$$

$$N(v) = \{u \mid (v,u) \in E\}$$

Pros & Cons

	adjacency list		adjacency matrix
memory	$O(n+m)$		$O(n^2)$
find outgoing edges	$O(\|N(v)\|)$		$O(n)$
find incoming edges	$O(n+m)$	Improve!	$O(n)$
test edge	$O(\|N(v)\|)$	Improve!	$O(1)$
delete edge	$O(\|N(v)\|)$		$O(1)$

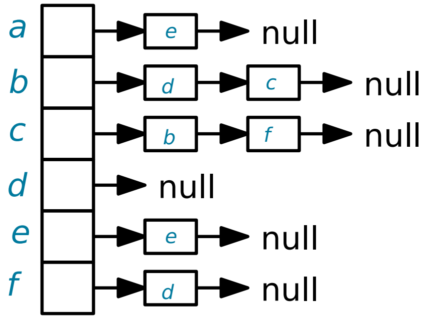


$$\begin{array}{c c} & \begin{matrix} a & b & c & d & e & f \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \end{matrix} & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

$N(v) = \{u \mid (v,u) \in E\}$

Pros & Cons

	adjacency list		adjacency matrix
memory	$O(n+m)$		$O(n^2)$
find outgoing edges	$O(\lvert N(v)\rvert)$		$O(n)$
find incoming edges	$O(n+m)$	Improve!	$O(n)$
test edge	$O(\lvert N(v)\rvert)$	Improve!	$O(1)$
delete edge	$O(\lvert N(v)\rvert)$	Improve!	$O(1)$



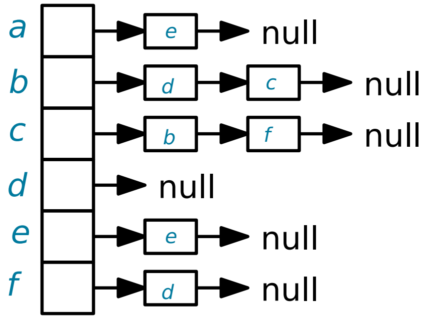
$$\begin{array}{c c} & \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array} & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

$$N(v) = \{u \mid (v,u) \in E\}$$

Pros & Cons

	adjacency list		adjacency matrix
memory	$O(n+m)$		$O(n^2)$
find outgoing edges	$O(\|N(v)\|)$		$O(n)$
find incoming edges	$O(n+m)$	Improve!	$O(n)$
test edge	$O(\|N(v)\|)$	Improve!	$O(1)$
delete edge	$O(\|N(v)\|)$	Improve!	$O(1)$

Choose representation based on situation!



$$\begin{array}{c c} & \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array} & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

$$N(v) = \{u \mid (v,u) \in E\}$$

Graph Traversal

Input: Undirected connected graph $G = (V, E)$.

Task: Traverse G, such that each vertex is visited at least once.

Graph Traversal

Input: Undirected connected graph $G = (V, E)$.
Task: Traverse G, such that each vertex is visited at least once.

- standard solutions:

Graph Traversal

Input: Undirected connected graph $G = (V, E)$.

Task: Traverse G, such that each vertex is visited at least once.

- standard solutions:

Breadth-First Search (BFS)

Idea: Start at arbitrary vertex v, visit all its neighbors, then all their neighbors, etc.

Graph Traversal

Input: Undirected connected graph $G = (V, E)$.

Task: Traverse G, such that each vertex is visited at least once.

- standard solutions:

Breadth-First Search (BFS)

Idea: Start at arbitrary vertex v, visit all its neighbors, then all their neighbors, etc.

Graph Traversal

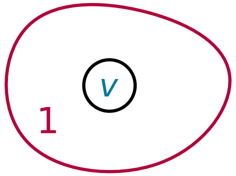
Input: Undirected connected graph $G = (V, E)$.

Task: Traverse G, such that each vertex is visited at least once.

- standard solutions:

Breadth-First Search (BFS)

Idea: Start at arbitrary vertex v, visit all its neighbors, then all their neighbors, etc.



Graph Traversal

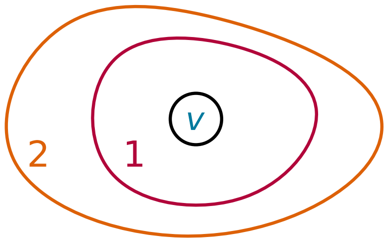
Input: Undirected connected graph $G = (V, E)$.

Task: Traverse G, such that each vertex is visited at least once.

- standard solutions:

Breadth-First Search (BFS)

Idea: Start at arbitrary vertex v, visit all its neighbors, then all their neighbors, etc.



Graph Traversal

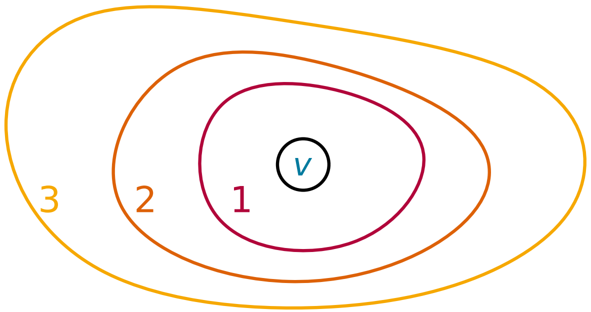
Input: Undirected connected graph $G = (V, E)$.

Task: Traverse G, such that each vertex is visited at least once.

- standard solutions:

Breadth-First Search (BFS)

Idea: Start at arbitrary vertex v, visit all its neighbors, then all their neighbors, etc.



Graph Traversal

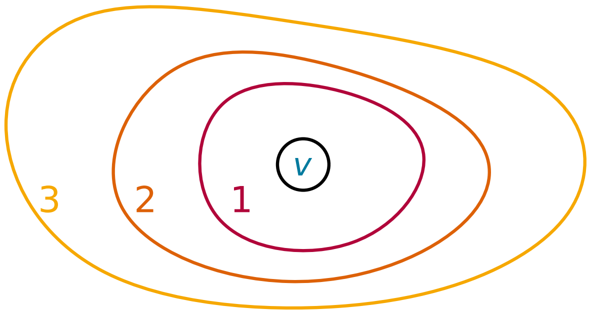
Input: Undirected connected graph $G = (V, E)$.

Task: Traverse G, such that each vertex is visited at least once.

- standard solutions:

Breadth-First Search (BFS)

Idea: Start at arbitrary vertex v, visit all its neighbors, then all their neighbors, etc.



Depth-First Search (DFS)

Idea: Start at arbitrary vertex v. Go to any neighbor. From there, proceed to any unvisited neighbor, etc. If stuck, backtrack.

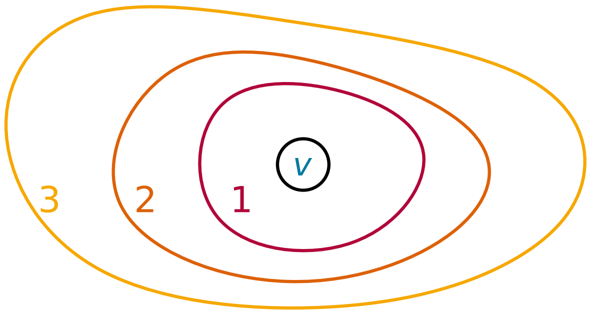
Graph Traversal

Input: Undirected connected graph $G = (V, E)$.
Task: Traverse G, such that each vertex is visited at least once.

- standard solutions:

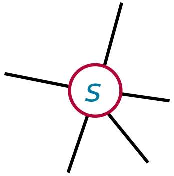
Breadth-First Search (BFS)

Idea: Start at arbitrary vertex v, visit all its neighbors, then all their neighbors, etc.



Depth-First Search (DFS)

Idea: Start at arbitrary vertex v. Go to any neighbor. From there, proceed to any unvisited neighbor, etc. If stuck, backtrack.



Graph Traversal

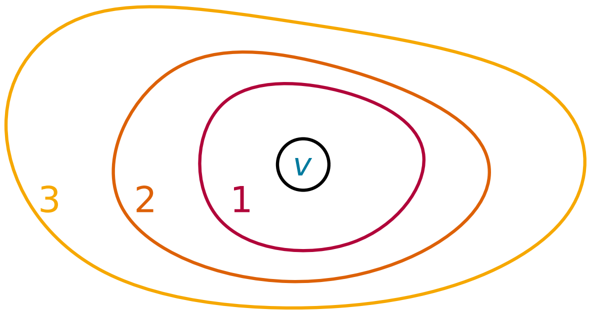
Input: Undirected connected graph $G = (V, E)$.

Task: Traverse G, such that each vertex is visited at least once.

- standard solutions:

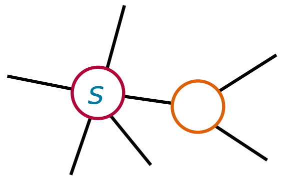
Breadth-First Search (BFS)

Idea: Start at arbitrary vertex v, visit all its neighbors, then all their neighbors, etc.



Depth-First Search (DFS)

Idea: Start at arbitrary vertex v. Go to any neighbor. From there, proceed to any unvisited neighbor, etc. If stuck, backtrack.



Graph Traversal

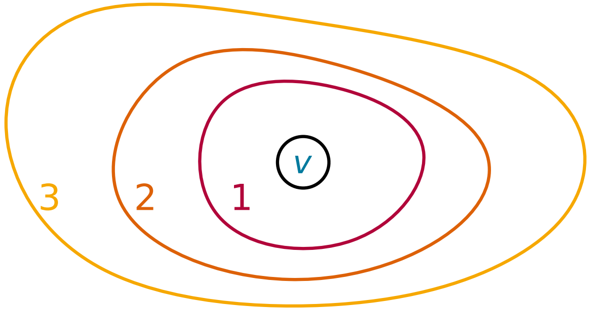
Input: Undirected connected graph $G = (V, E)$.

Task: Traverse G, such that each vertex is visited at least once.

- standard solutions:

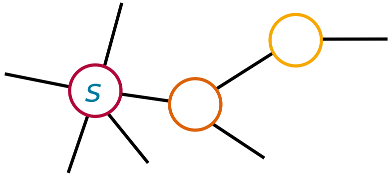
Breadth-First Search (BFS)

Idea: Start at arbitrary vertex v, visit all its neighbors, then all their neighbors, etc.



Depth-First Search (DFS)

Idea: Start at arbitrary vertex v. Go to any neighbor. From there, proceed to any unvisited neighbor, etc. If stuck, backtrack.



Graph Traversal

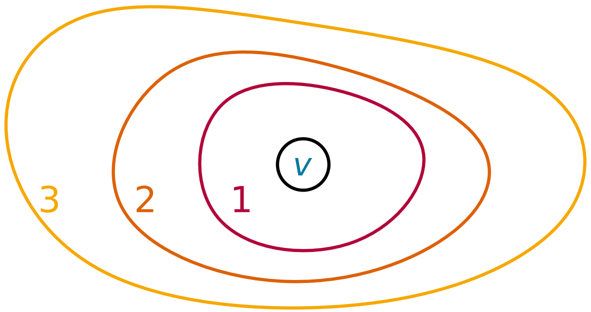
Input: Undirected connected graph $G = (V, E)$.

Task: Traverse G, such that each vertex is visited at least once.

- standard solutions:

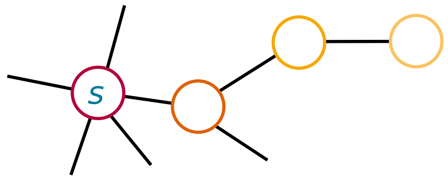
Breadth-First Search (BFS)

Idea: Start at arbitrary vertex v, visit all its neighbors, then all their neighbors, etc.



Depth-First Search (DFS)

Idea: Start at arbitrary vertex v. Go to any neighbor. From there, proceed to any unvisited neighbor, etc. If stuck, backtrack.



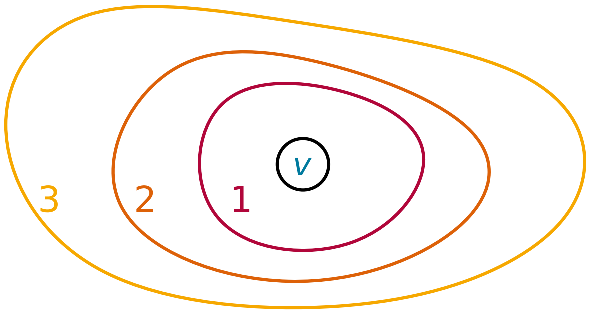
Graph Traversal

Input: Undirected connected graph $G = (V, E)$.
Task: Traverse G, such that each vertex is visited at least once.

- standard solutions:

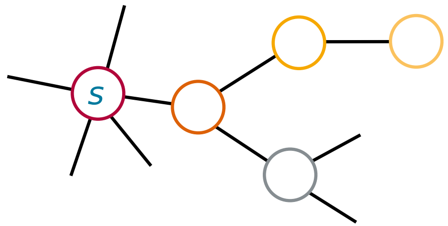
Breadth-First Search (BFS)

Idea: Start at arbitrary vertex v, visit all its neighbors, then all their neighbors, etc.



Depth-First Search (DFS)

Idea: Start at arbitrary vertex v. Go to any neighbor. From there, proceed to any unvisited neighbor, etc. If stuck, backtrack.



Graph Traversal

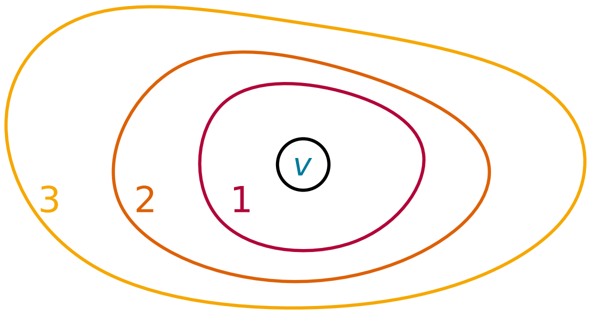
Input: Undirected connected graph $G = (V, E)$.

Task: Traverse G, such that each vertex is visited at least once.

- standard solutions:

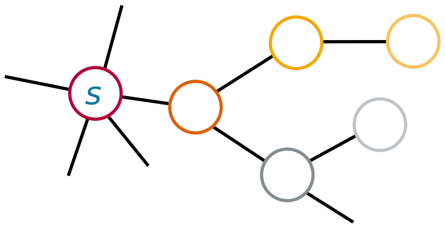
Breadth-First Search (BFS)

Idea: Start at arbitrary vertex v, visit all its neighbors, then all their neighbors, etc.



Depth-First Search (DFS)

Idea: Start at arbitrary vertex v. Go to any neighbor. From there, proceed to any unvisited neighbor, etc. If stuck, backtrack.



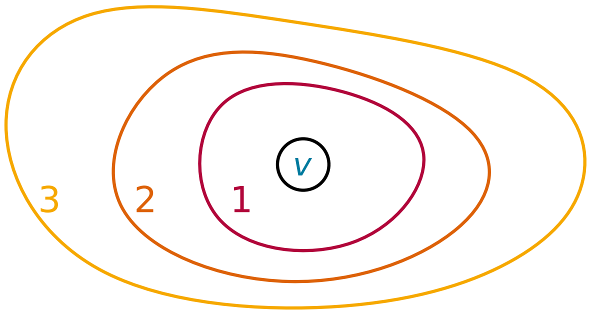
Graph Traversal

Input: Undirected connected graph $G = (V, E)$.
Task: Traverse G, such that each vertex is visited at least once.

- standard solutions:

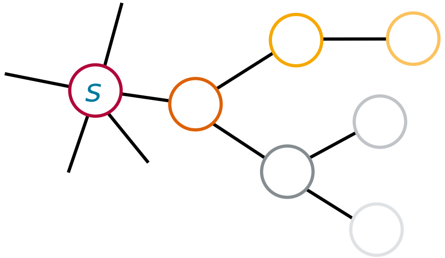
Breadth-First Search (BFS)

Idea: Start at arbitrary vertex v, visit all its neighbors, then all their neighbors, etc.



Depth-First Search (DFS)

Idea: Start at arbitrary vertex v. Go to any neighbor. From there, proceed to any unvisited neighbor, etc. If stuck, backtrack.



Graph Traversal

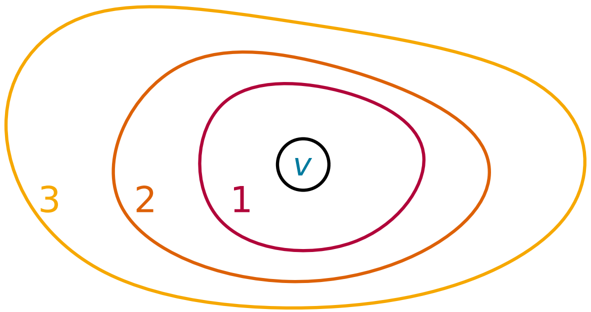
Input: Undirected connected graph $G = (V, E)$.

Task: Traverse G, such that each vertex is visited at least once.

- standard solutions:

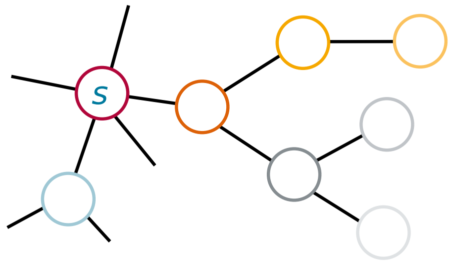
Breadth-First Search (BFS)

Idea: Start at arbitrary vertex v, visit all its neighbors, then all their neighbors, etc.



Depth-First Search (DFS)

Idea: Start at arbitrary vertex v. Go to any neighbor. From there, proceed to any unvisited neighbor, etc. If stuck, backtrack.



Topological Sort

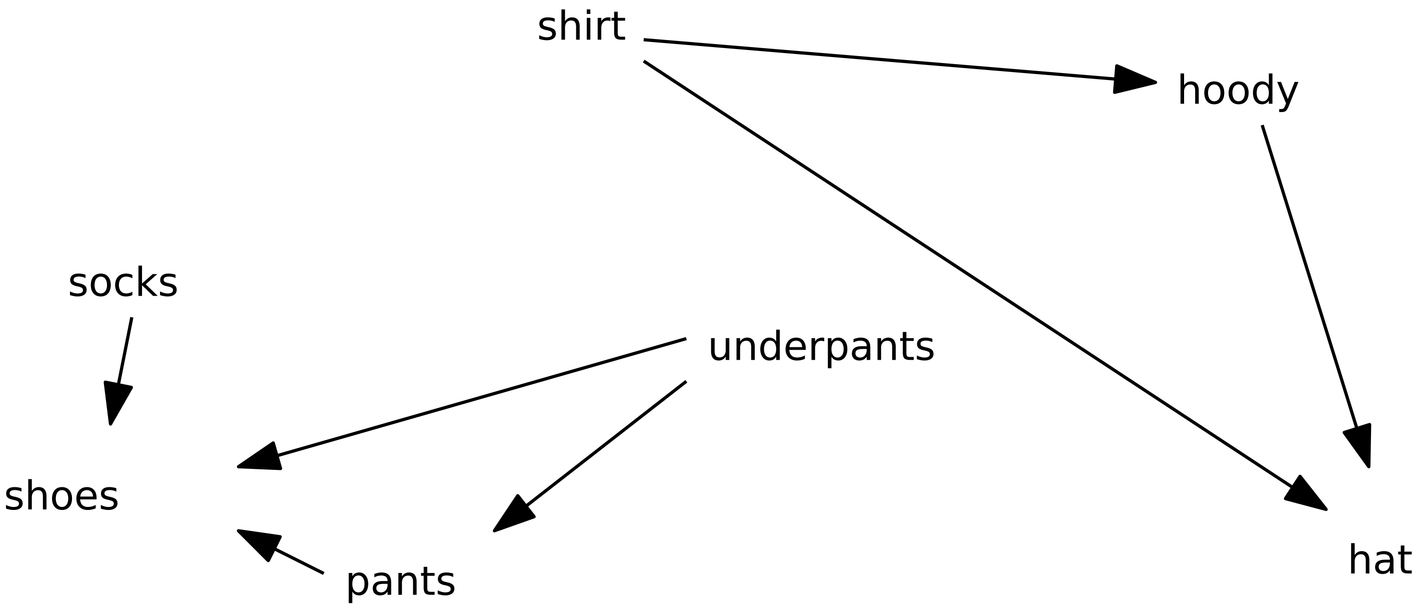
Input: Directed Graph $G = (V, E)$.

Task: Compute a *topological ordering* of G, that is, a linear ordering of vertices s.t. for every directed edge uv, u comes before v in the ordering.

Topological Sort

Input: Directed Graph $G = (V, E)$.

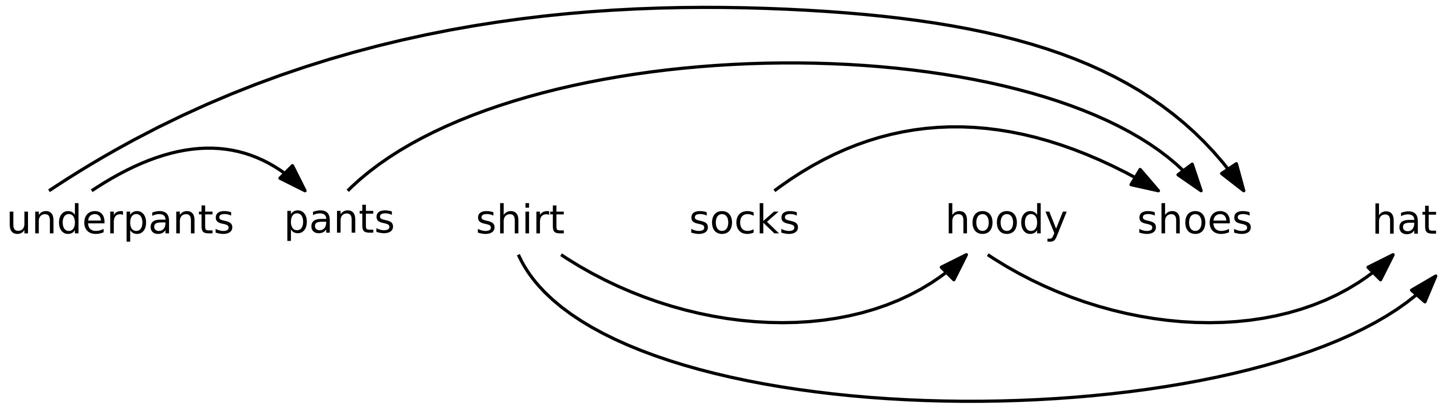
Task: Compute a *topological ordering* of G, that is, a linear ordering of vertices s.t. for every directed edge uv, u comes before v in the ordering.



Topological Sort

Input: Directed Graph $G = (V, E)$.

Task: Compute a *topological ordering* of G, that is, a linear ordering of vertices s.t. for every directed edge uv, u comes before v in the ordering.



Topological Sort

Input: Directed Graph $G = (V, E)$.

Task: Compute a *topological ordering* of G, that is, a linear ordering of vertices s.t. for every directed edge uv, u comes before v in the ordering.

Is this always possible?

Topological Sort

Input: Directed Graph $G = (V, E)$.

Task: Compute a *topological ordering* of G, that is, a linear ordering of vertices s.t. for every directed edge uv, u comes before v in the ordering.

Is this always possible?

No!

Only possible if G is a directed acyclic graph (DAG), i.e., contains no cycles!

Topological Sort

Input: Directed Graph $G = (V, E)$.

Task: Compute a *topological ordering* of G, that is, a linear ordering of vertices s.t. for every directed edge uv, u comes before v in the ordering.

Is this always possible?

No!

Only possible if G is a directed acyclic graph (DAG), i.e., contains no cycles!

Greedy Idea: Repeatedly remove some vertex w/o incoming edges and add it to the ordering.

Topological Sort

Input: Directed Graph $G = (V, E)$.

Task: Compute a *topological ordering* of G, that is, a linear ordering of vertices s.t. for every directed edge uv, u comes before v in the ordering.

Topological Sorting(G)

Input: directed graph G
Output: topological order or error

```
initialize empty list L
while V ≠ ∅ do
    if G has vertex v without incoming edge then
        append v to L
        delete v and adjacent edges from G
    else
        return Error
    end if
end while
return L
```

Is this always possible?

No!

Only possible if G is a directed acyclic graph (DAG), i.e., contains no cycles!

Greedy Idea: Repeatedly remove some vertex w/o incoming edges and add it to the ordering.

Topological Sort

Input: Directed Graph $G = (V, E)$.

Task: Compute a *topological ordering* of G, that is, a linear ordering of vertices s.t. for every directed edge uv, u comes before v in the ordering.

Topological Sorting(G)

Input: directed graph G
Output: topological order or error

```
initialize empty list L
while V ≠ ∅ do
    if G has vertex v without incoming edge then
        append v to L
        delete v and adjacent edges from G
    else
        return Error
    end if
end while
return L
```

Is this always possible?

No!

Only possible if G is a directed acyclic graph (DAG), i.e., contains no cycles!

Greedy Idea: Repeatedly remove some vertex w/o incoming edges and add it to the ordering.

How would you implement this?

Talk to your Neighbor

Given a directed graph. Each edge has a difficulty w, meaning you need at least w skill to reverse the edge. How skilled do you have to be to remove all cycles?

(Your skill is **not** consumed by operations)

Talk to your Neighbor

Given a directed graph. Each edge has a difficulty w, meaning you need at least w skill to reverse the edge. How skilled do you have to be to remove all cycles?

(Your skill is **not** consumed by operations)

Bonus: Determine the set of edges that will be reversed.

Talk to your Neighbor

Given a directed graph. Each edge has a difficulty w, meaning you need at least w skill to reverse the edge. How skilled do you have to be to remove all cycles?

(Your skill is **not** consumed by operations)

Bonus: Determine the set of edges that will be reversed.

http://codeforces.com/problemset/problem/1100/E