

CS 577 - Graphs

Marc Renault

Department of Computer Sciences
University of Wisconsin – Madison

Spring 2021

TopHat Join Code: 524741



GRAPHS

Graphs

A graph G is a pair $G = (V, E)$, where V is a set of vertices/nodes and E is a set of edges/arcs connecting a pair of vertices. That is, $E \subseteq V \times V$.

GRAPHS

Graphs

A graph G is a pair $G = (V, E)$, where V is a set of vertices/nodes and E is a set of edges/arcs connecting a pair of vertices. That is, $E \subseteq V \times V$.

Some Special Graphs

- Complete graph (K_n)

GRAPHS

Graphs

A graph G is a pair $G = (V, E)$, where V is a set of vertices/nodes and E is a set of edges/arcs connecting a pair of vertices. That is, $E \subseteq V \times V$.

Some Special Graphs

- Complete graph (K_n)
- Cycle (C_n)

GRAPHS

Graphs

A graph G is a pair $G = (V, E)$, where V is a set of vertices/nodes and E is a set of edges/arcs connecting a pair of vertices. That is, $E \subseteq V \times V$.

Some Special Graphs

- Complete graph (K_n)
- Cycle (C_n)
- Path (P_n)

GRAPHS

Graphs

A graph G is a pair $G = (V, E)$, where V is a set of vertices/nodes and E is a set of edges/arcs connecting a pair of vertices. That is, $E \subseteq V \times V$.

Some Special Graphs

- Complete graph (K_n)
- Cycle (C_n)
- Path (P_n)
- Trees

GRAPHS

Graphs

A graph G is a pair $G = (V, E)$, where V is a set of vertices/nodes and E is a set of edges/arcs connecting a pair of vertices. That is, $E \in V \times V$.

Some Special Graphs

- Complete graph (K_4)
- Cycle (C_4)
- Path (P_4)
- Trees
- Digraph

GRAPHS

Graphs

A graph G is a pair $G = (V, E)$, where V is a set of vertices/nodes and E is a set of edges/arcs connecting a pair of vertices. That is, $E \in V \times V$.

Some Special Graphs

- Complete graph (K_4)
- Cycle (C_4)
- Path (P_4)
- Trees
- Digraph
- Directed Acyclic Graph (DAG)

GRAPHS

Graphs

A graph G is a pair $G = (V, E)$, where V is a set of vertices/nodes and E is a set of edges/arcs connecting a pair of vertices. That is, $E \subseteq V \times V$.

Some Special Graphs

- Complete graph (K_n)
- Cycle (C_n)
- Path (P_n)
- Trees
- Digraph
- Directed Acyclic Graph (DAG)
- Bipartite

GRAPHS

Graphs

A graph G is a pair $G = (V, E)$, where V is a set of vertices/nodes and E is a set of edges/arcs connecting a pair of vertices. That is, $E \in V \times V$.

Some Special Graphs

- Complete graph (K_4)
- Cycle (C_4)
- Path (P_4)
- Trees
- Digraph
- Directed Acyclic Graph (DAG)
- Bipartite
- Forests

TREES

Definition

- A connected graph without cycles.
- A single node may be designated as the *root* of the tree.
- Any node with degree 1 that is not the root is a *leaf*.

TREES

Definition

- A connected graph without cycles.
- A single node may be designated as the *root* of the tree.
- Any node with degree 1 that is not the root is a *leaf*.

Properties of a tree T

- 1 If $|V| \geq 2$, (unrooted) T has at least 2 leaves.
- 2 For all nodes u and v , there exists one path between them in T .
- 3 $|V| = |E| + 1$ for $|V| \geq 1$.

TREES

Definition

- A connected graph without cycles.
- A single node may be designated as the *root* of the tree.
- Any node with degree 1 that is not the root is a *leaf*.

Properties of a tree T

- 1 If $|V| \geq 2$, (unrooted) T has at least 2 leaves.
- 2 For all nodes u and v , there exists one path between them in T .
- 3 $|V| = |E| + 1$ for $|V| \geq 1$.

TopHat 1

Is P_{10} a tree?

WHAT CAN BE REPRESENTED BY GRAPHS?

- Transportation networks
- Communication networks
- Information networks
- Social networks
- Dependency networks

CONNECTIVITY

GRAPH CONNECTIVITY

Problem: s - t connectivity

Given a graph $G = (V, E)$, and the vertices s and t , is there a path from s to t in G ?

GRAPH CONNECTIVITY

Problem: s - t connectivity

Given a graph $G = (V, E)$, and the vertices s and t , is there a path from s to t in G ?

Connected Graph

If all $(u, v) \in V \times V$ are connected, then G is connected.

GRAPH CONNECTIVITY

Problem: s - t connectivity

Given a graph $G = (V, E)$, and the vertices s and t , is there a path from s to t in G ?

Connected Graph

If all $(u, v) \in V \times V$ are connected, then G is connected.

Connected Components

Let $H \subset G$ be a subgraph of G . If H is connected and there are no edges between H and $G \setminus H$. Then, H is a connected component of G .

GRAPH EXPLORATION/TRAVERSAL

Determining s - t Connectivity

Requires an algorithm that explores or traverses the graph by considering the edges of the graph to find all nodes connected to s .

GRAPH EXPLORATION/TRAVERSAL

Determining s - t Connectivity

Requires an algorithm that explores or traverses the graph by considering the edges of the graph to find all nodes connected to s .

Algorithm: Generalized Exploration

$R = \{s\}$

while \exists an edge (u, v) where $u \in R$ and $v \notin R$ **do**

 | Add v to R

end

return R

GRAPH ENCODINGS AND IMPLEMENTATION

Representations

- **Adjacency matrix:** $|V|$ by $|V|$ matrix with a 1 if nodes are adjacent.
- **Adjacency list:** For each node, list adjacent nodes.
- **Edge list:** List of all node pairs representing the edges (plus list of nodes).
- **Incidence matrix:** $|V|$ by $|E|$ matrix with a 1 if node is incident to the edge.

GRAPH ENCODINGS AND IMPLEMENTATION

Representations

- **Adjacency matrix:** $|V|$ by $|V|$ matrix with a 1 if nodes are adjacent.
- **Adjacency list:** For each node, list adjacent nodes.
- **Edge list:** List of all node pairs representing the edges (plus list of nodes).
- **Incidence matrix:** $|V|$ by $|E|$ matrix with a 1 if node is incident to the edge.

Space

Find (u, v)

List of neighbours

Adjacency matrix

Adjacency list

Edge list

Incidence matrix

GRAPH ENCODINGS AND IMPLEMENTATION

Representations

- **Adjacency matrix:** $|V|$ by $|V|$ matrix with a 1 if nodes are adjacent.
- **Adjacency list:** For each node, list adjacent nodes.
- **Edge list:** List of all node pairs representing the edges (plus list of nodes).
- **Incidence matrix:** $|V|$ by $|E|$ matrix with a 1 if node is incident to the edge.

	Space	Find (u, v)	List of neighbours
Adjacency matrix	$O(V ^2)$		
Adjacency list			
Edge list			
Incidence matrix			

GRAPH ENCODINGS AND IMPLEMENTATION

Representations

- **Adjacency matrix:** $|V|$ by $|V|$ matrix with a 1 if nodes are adjacent.
- **Adjacency list:** For each node, list adjacent nodes.
- **Edge list:** List of all node pairs representing the edges (plus list of nodes).
- **Incidence matrix:** $|V|$ by $|E|$ matrix with a 1 if node is incident to the edge.

	Space	Find (u, v)	List of neighbours
Adjacency matrix	$O(V ^2)$	$O(1)$	
Adjacency list			
Edge list			
Incidence matrix			

GRAPH ENCODINGS AND IMPLEMENTATION

Representations

- **Adjacency matrix:** $|V|$ by $|V|$ matrix with a 1 if nodes are adjacent.
- **Adjacency list:** For each node, list adjacent nodes.
- **Edge list:** List of all node pairs representing the edges (plus list of nodes).
- **Incidence matrix:** $|V|$ by $|E|$ matrix with a 1 if node is incident to the edge.

	Space	Find (u, v)	List of neighbours
Adjacency matrix	$O(V ^2)$	$O(1)$	$O(V)$
Adjacency list			
Edge list			
Incidence matrix			

GRAPH ENCODINGS AND IMPLEMENTATION

Representations

- **Adjacency matrix:** $|V|$ by $|V|$ matrix with a 1 if nodes are adjacent.
- **Adjacency list:** For each node, list adjacent nodes.
- **Edge list:** List of all node pairs representing the edges (plus list of nodes).
- **Incidence matrix:** $|V|$ by $|E|$ matrix with a 1 if node is incident to the edge.

	Space	Find (u, v)	List of neighbours
Adjacency matrix	$O(V ^2)$	$O(1)$	$O(V)$
Adjacency list	$O(V \cdot \min(E , V))$		
Edge list			
Incidence matrix			

GRAPH ENCODINGS AND IMPLEMENTATION

Representations

- **Adjacency matrix:** $|V|$ by $|V|$ matrix with a 1 if nodes are adjacent.
- **Adjacency list:** For each node, list adjacent nodes.
- **Edge list:** List of all node pairs representing the edges (plus list of nodes).
- **Incidence matrix:** $|V|$ by $|E|$ matrix with a 1 if node is incident to the edge.

	Space	Find (u, v)	List of neighbours
Adjacency matrix	$O(V ^2)$	$O(1)$	$O(V)$
Adjacency list	$O(V \cdot \min(E , V))$	$O(\min(V , E))$	
Edge list			
Incidence matrix			

GRAPH ENCODINGS AND IMPLEMENTATION

Representations

- **Adjacency matrix:** $|V|$ by $|V|$ matrix with a 1 if nodes are adjacent.
- **Adjacency list:** For each node, list adjacent nodes.
- **Edge list:** List of all node pairs representing the edges (plus list of nodes).
- **Incidence matrix:** $|V|$ by $|E|$ matrix with a 1 if node is incident to the edge.

	Space	Find (u, v)	List of neighbours
Adjacency matrix	$O(V ^2)$	$O(1)$	$O(V)$
Adjacency list	$O(V \cdot \min(E , V))$	$O(\min(V , E))$	$O(1)$
Edge list			
Incidence matrix			

GRAPH ENCODINGS AND IMPLEMENTATION

Representations

- **Adjacency matrix:** $|V|$ by $|V|$ matrix with a 1 if nodes are adjacent.
- **Adjacency list:** For each node, list adjacent nodes.
- **Edge list:** List of all node pairs representing the edges (plus list of nodes).
- **Incidence matrix:** $|V|$ by $|E|$ matrix with a 1 if node is incident to the edge.

	Space	Find (u, v)	List of neighbours
Adjacency matrix	$O(V ^2)$	$O(1)$	$O(V)$
Adjacency list	$O(V \cdot \min(E , V))$	$O(\min(V , E))$	$O(1)$
Edge list	$O(E + V)$		
Incidence matrix			

GRAPH ENCODINGS AND IMPLEMENTATION

Representations

- **Adjacency matrix:** $|V|$ by $|V|$ matrix with a 1 if nodes are adjacent.
- **Adjacency list:** For each node, list adjacent nodes.
- **Edge list:** List of all node pairs representing the edges (plus list of nodes).
- **Incidence matrix:** $|V|$ by $|E|$ matrix with a 1 if node is incident to the edge.

	Space	Find (u, v)	List of neighbours
Adjacency matrix	$O(V ^2)$	$O(1)$	$O(V)$
Adjacency list	$O(V \cdot \min(E , V))$	$O(\min(V , E))$	$O(1)$
Edge list	$O(E + V)$	$O(E)$	
Incidence matrix			

GRAPH ENCODINGS AND IMPLEMENTATION

Representations

- **Adjacency matrix:** $|V|$ by $|V|$ matrix with a 1 if nodes are adjacent.
- **Adjacency list:** For each node, list adjacent nodes.
- **Edge list:** List of all node pairs representing the edges (plus list of nodes).
- **Incidence matrix:** $|V|$ by $|E|$ matrix with a 1 if node is incident to the edge.

	Space	Find (u, v)	List of neighbours
Adjacency matrix	$O(V ^2)$	$O(1)$	$O(V)$
Adjacency list	$O(V \cdot \min(E , V))$	$O(\min(V , E))$	$O(1)$
Edge list	$O(E + V)$	$O(E)$	$O(E)$
Incidence matrix			

GRAPH ENCODINGS AND IMPLEMENTATION

Representations

- **Adjacency matrix:** $|V|$ by $|V|$ matrix with a 1 if nodes are adjacent.
- **Adjacency list:** For each node, list adjacent nodes.
- **Edge list:** List of all node pairs representing the edges (plus list of nodes).
- **Incidence matrix:** $|V|$ by $|E|$ matrix with a 1 if node is incident to the edge.

	Space	Find (u, v)	List of neighbours
Adjacency matrix	$O(V ^2)$	$O(1)$	$O(V)$
Adjacency list	$O(V \cdot \min(E , V))$	$O(\min(V , E))$	$O(1)$
Edge list	$O(E + V)$	$O(E)$	$O(E)$
Incidence matrix	$O(V E)$		

GRAPH ENCODINGS AND IMPLEMENTATION

Representations

- **Adjacency matrix:** $|V|$ by $|V|$ matrix with a 1 if nodes are adjacent.
- **Adjacency list:** For each node, list adjacent nodes.
- **Edge list:** List of all node pairs representing the edges (plus list of nodes).
- **Incidence matrix:** $|V|$ by $|E|$ matrix with a 1 if node is incident to the edge.

	Space	Find (u, v)	List of neighbours
Adjacency matrix	$O(V ^2)$	$O(1)$	$O(V)$
Adjacency list	$O(V \cdot \min(E , V))$	$O(\min(V , E))$	$O(1)$
Edge list	$O(E + V)$	$O(E)$	$O(E)$
Incidence matrix	$O(V E)$	$O(E)$	

GRAPH ENCODINGS AND IMPLEMENTATION

Representations

- **Adjacency matrix:** $|V|$ by $|V|$ matrix with a 1 if nodes are adjacent.
- **Adjacency list:** For each node, list adjacent nodes.
- **Edge list:** List of all node pairs representing the edges (plus list of nodes).
- **Incidence matrix:** $|V|$ by $|E|$ matrix with a 1 if node is incident to the edge.

	Space	Find (u, v)	List of neighbours
Adjacency matrix	$O(V ^2)$	$O(1)$	$O(V)$
Adjacency list	$O(V \cdot \min(E , V))$	$O(\min(V , E))$	$O(1)$
Edge list	$O(E + V)$	$O(E)$	$O(E)$
Incidence matrix	$O(V E)$	$O(E)$	$O(V E)$

GRAPH EXPLORATION/TRAVERSAL

Algorithm: Generalized Exploration

$R = \{s\}$

while \exists an edge (u, v) where $u \in R$ and $v \notin R$ **do**

 | Add v to R

end

return R

TopHat 2

Which graph representation would be best suited?

GRAPH EXPLORATION/TRAVERSAL

Algorithm: Generalized Exploration

$R = \{s\}$

while \exists an edge (u, v) where $u \in R$ and $v \notin R$ **do**

 | Add v to R

end

return R

Rough Running Time

- At step i : $O(|E_i| \cdot (\log |R_i| + \log |R_i|) + \log |R_i|)$, assuming R is a self-balancing BST.

GRAPH EXPLORATION/TRAVERSAL

Algorithm: Generalized Exploration

$R = \{s\}$

while \exists an edge (u, v) where $u \in R$ and $v \notin R$ **do**

 | Add v to R

end

return R

Rough Running Time

- At step i : $O(|E_i| \cdot (\log |R_i| + \log |R_i|) + \log |R_i|)$, assuming R is a self-balancing BST.
- At most $|E|$ steps: $O(|E|^2 \log |V|)$

GRAPH EXPLORATION/TRAVERSAL

Algorithm: Generalized Exploration

$R = \{s\}$

while \exists an edge (u, v) where $u \in R$ and $v \notin R$ **do**

 | Add v to R

end

return R

Rough Running Time

- At step i : $O(|E_i| \cdot (\log |R_i| + \log |R_i|) + \log |R_i|)$, assuming R is a self-balancing BST.
- At most $|E|$ steps: $O(|E|^2 \log |V|)$

What is this algorithm lacking?

BREADTH-FIRST SEARCH (BFS)

Process

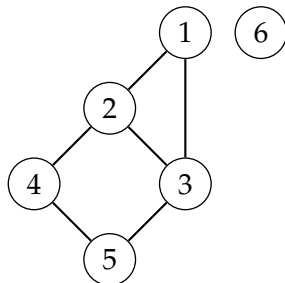
- Also referred to as graph flooding.
- Let L_i be all the neighbours at a distance i from s .
- Starting from $i = 0$, visit all the nodes (not previously visited) in L_i . Increment i and repeat.

BREADTH-FIRST SEARCH (BFS)

Process

- Also referred to as graph flooding.
- Let L_i be all the neighbours at a distance i from s .
- Starting from $i = 0$, visit all the nodes (not previously visited) in L_i . Increment i and repeat.

TopHat 3: This process engenders a BFS tree. Start at 1 and draw such a tree for the following.



DEPTH-FIRST SEARCH (DFS)

Recursive Process starting at s

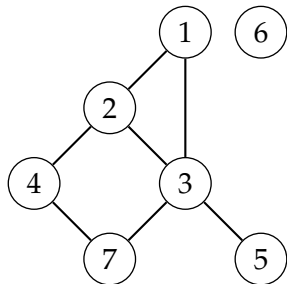
- Mark s as visited.
- For each $(s, u) \in E$ where u has not been visited, do DFS(u).

DEPTH-FIRST SEARCH (DFS)

Recursive Process starting at s

- Mark s as visited.
- For each $(s, u) \in E$ where u has not been visited, do $\text{DFS}(u)$.

TopHat 4: This process engenders a DFS tree. Start at 1 and draw such a tree for the following.



IMPLEMENTING BFS AND DFS

TopHat 5

Which graph representation would be best for BFS and DFS?

IMPLEMENTING BFS AND DFS

TopHat 5

Which graph representation would be best for BFS and DFS?
Why?

IMPLEMENTING BFS AND DFS

BFS Process

- Also referred to as graph flooding.
- Let L_i be all the neighbours at a distance i from s .
- Starting from $i = 0$, visit all the nodes (not previously visited) in L_i . Increment i and repeat.

DFS Recursive Process starting at s

- Mark s as visited.
- For each $(s, u) \in E$ where u has not been visited, do DFS(u).

IMPLEMENTING BFS AND DFS

Algorithm: BFS(S)

Initialize $v[u] = \text{false}$ for all
nodes

Set $v[s] = \text{true}$

Add s to tree T

Add s to queue Q

while Q is not empty **do**

$u = \text{dequeue}(Q)$

foreach neighbour r of u

do

if $\neg v[r]$ **then**

 Add (u, r) to T

 Set $v[r] = \text{true}$

 Enqueue v

end

end

end

return T

IMPLEMENTING BFS AND DFS

Algorithm: BFS(S)

Initialize $v[u] = \text{false}$ for all nodes

Set $v[s] = \text{true}$

Add s to tree T

Add s to queue Q

while Q is not empty **do**

$u = \text{dequeue}(Q)$

foreach neighbour r of u
 do

if $!v[r]$ **then**

 Add (u, r) to T

 Set $v[r] = \text{true}$

 Enqueue v

end

end

end

return T

Algorithm: DFS(S)

Initialize $v[u] = \text{false}$ and

$p[u] = \text{null}$ for all nodes

Push s to stack S

while S is not empty **do**

$u = \text{pop}(S)$

if $!v[u]$ **then**

 Add $(p[u], u)$ to T

 Set $v[u] = \text{true}$

foreach neighbour r
 of u **do**

 Push r to stack S

 Set $p[r] = u$

end

end

end

return T

IMPLEMENTING BFS AND DFS

Algorithm: BFS(S)

Initialize $v[u] = \text{false}$ for all nodes

Set $v[s] = \text{true}$

Add s to tree T

Add s to queue Q

while Q is not empty **do**

$u = \text{dequeue}(Q)$

foreach neighbour r of u
 do

if $!v[r]$ **then**

 Add (u, r) to T

 Set $v[r] = \text{true}$

 Enqueue v

end

end

end

return T

Algorithm: DFS(S)

Initialize $v[u] = \text{false}$ and

$p[u] = \text{null}$ for all nodes

Push s to stack S

while S is not empty **do**

$u = \text{pop}(S)$

if $!v[u]$ **then**

 Add $(p[u], u)$ to T

 Set $v[u] = \text{true}$

foreach neighbour r
 of u **do**

 Push r to stack S

 Set $p[r] = u$

end

end

end

return T

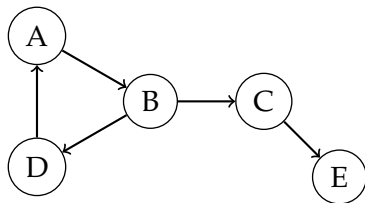
Runtime: $O(|E| + |V|)$

STRONGLY CONNECTED COMPONENTS

DIRECTED GRAPHS

Directed Graph

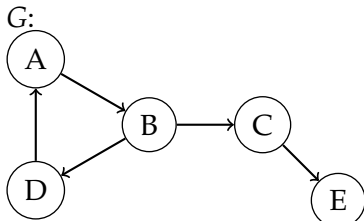
- In a directed graph, the edges have a direction and are often called *arcs*.
- I.e. (u, v) is different than (v, u) .



STRONG CONNECTIVITY

Mutually Reachable

- A pair of nodes (u, v) in a directed graph are *mutually reachable* if there is a path from u to v , and from v to u .
- Note: This property is transitive: if (u, v) and (v, w) are both mutually reachable, then u, w is mutually reachable.



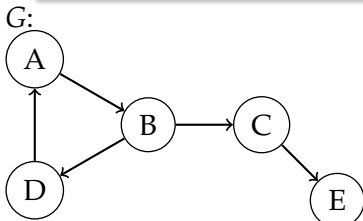
STRONG CONNECTIVITY

Mutually Reachable

- A pair of nodes (u, v) in a directed graph are *mutually reachable* if there is a path from u to v , and from v to u .
- Note: This property is transitive: if (u, v) and (v, w) are both mutually reachable, then u, w is mutually reachable.

Strongly Connected

A directed graph is *strongly connected* if, for every pair of nodes (u, v) , u and v are mutually reachable.



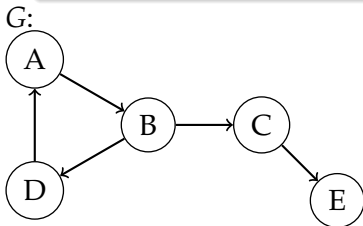
STRONG CONNECTIVITY

Mutually Reachable

- A pair of nodes (u, v) in a directed graph are *mutually reachable* if there is a path from u to v , and from v to u .
- Note: This property is transitive: if (u, v) and (v, w) are both mutually reachable, then u, w is mutually reachable.

Testing for Mutually Reachable

How might we check if (u, v) is mutually reachable?



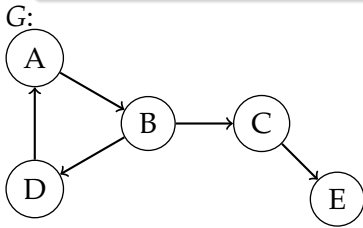
STRONG CONNECTIVITY

Mutually Reachable

- A pair of nodes (u, v) in a directed graph are *mutually reachable* if there is a path from u to v , and from v to u .
- Note: This property is transitive: if (u, v) and (v, w) are both mutually reachable, then u, w is mutually reachable.

Testing for Mutually Reachable

Check if DFS/BFS from u reach v , and DFS/BFS from v reaches u .



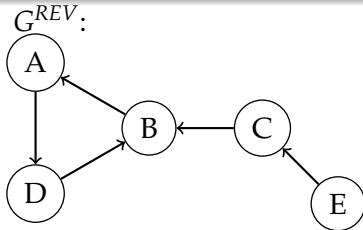
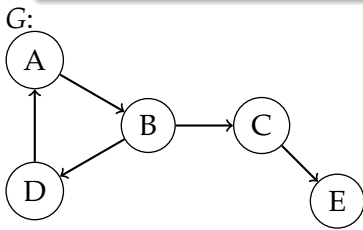
STRONG CONNECTIVITY

Mutually Reachable

- A pair of nodes (u, v) in a directed graph are *mutually reachable* if there is a path from u to v , and from v to u .
- Note: This property is transitive: if (u, v) and (v, w) are both mutually reachable, then u, w is mutually reachable.

Testing for Mutually Reachable

Check if DFS/BFS from u in G reaches v , and DFS/BFS from u in G^{REV} reaches v .



STRONGLY CONNECTED COMPONENTS

Strongly Connected Component (SCC)

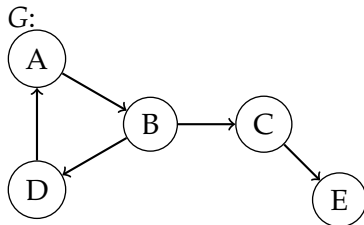
A maximal strongly connected subgraph.

STRONGLY CONNECTED COMPONENTS

Strongly Connected Component (SCC)

A maximal strongly connected subgraph.

TopHat 6: How many SCC in G ?

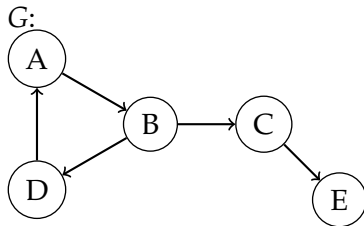


STRONGLY CONNECTED COMPONENTS

Strongly Connected Component (SCC)

A maximal strongly connected subgraph.

TopHat 6: How many SCC in G ? 3



STRONGLY CONNECTED COMPONENTS

Problem

Find the SCCs in a digraph G .

STRONGLY CONNECTED COMPONENTS

Problem

Find the SCCs in a digraph G .

Kosaraju's Algorithm

- ❶ Populate a stack S with a DFS on G .
- ❷ Build G^{REV} for G , and set all nodes to unvisited.
- ❸ While S is not empty:
 - ❶ Pop node v from S .
 - ❷ If v is unvisited, run DFS on G^{REV} from v to extract an SCC.

STRONGLY CONNECTED COMPONENTS

Problem

Find the SCCs in a digraph G .

Kosaraju's Algorithm

- ❶ Populate a stack S with a DFS on G .
- ❷ Build G^{REV} for G , and set all nodes to unvisited.
- ❸ While S is not empty:
 - ❶ Pop node v from S .
 - ❷ If v is unvisited, run DFS on G^{REV} from v to extract an SCC.

TopHat 7: What is the time complexity of Kosaraju's Algorithm?

STRONGLY CONNECTED COMPONENTS

Problem

Find the SCCs in a digraph G .

Kosaraju's Algorithm

- ❶ Populate a stack S with a DFS on G .
- ❷ Build G^{REV} for G , and set all nodes to unvisited.
- ❸ While S is not empty:
 - ❶ Pop node v from S .
 - ❷ If v is unvisited, run DFS on G^{REV} from v to extract an SCC.

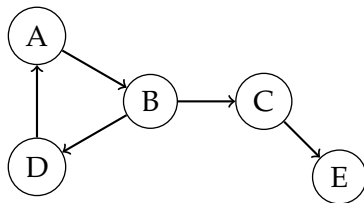
TopHat 7: What is the time complexity of Kosaraju's Algorithm? $O(|E| + |V|)$

TOPOLOGICAL ORDERING

DIRECTED GRAPHS

Directed Graph

- In a directed graph, the edges have a direction and are often called *arcs*.
- I.e. (u, v) is different than (v, u) .



DIRECTED GRAPHS

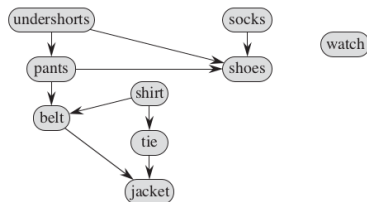
Directed Graph

- In a directed graph, the edges have a direction and are often called *arcs*.
- I.e. (u, v) is different than (v, u) .

Directed Acyclic Graph (DAG)

- A directed graph with no directed cycles.
- Precedence relationships.

Getting dressed:



TOPOLOGICAL ORDERING

Definition

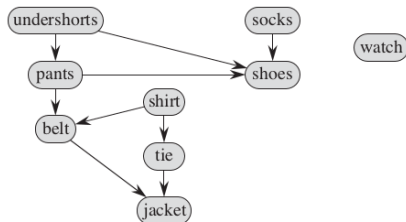
An ordering of the nodes of a DAG which respected the precedence relations.

TOPOLOGICAL ORDERING

Definition

An ordering of the nodes of a DAG which respected the precedence relations.

Getting dressed DAG:

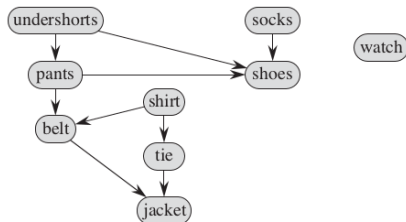


TOPOLOGICAL ORDERING

Definition

An ordering of the nodes of a DAG which respected the precedence relations.

Getting dressed DAG:



Topological ordering:



DAGs AND TOPOLOGICAL ORDERING

Observation 1

If G has a topological ordering, then G is a DAG.

DAGs AND TOPOLOGICAL ORDERING

Observation 1

If G has a topological ordering, then G is a DAG.

Key Property

In every DAG G , there is a node v with no incoming edges.

DAGs AND TOPOLOGICAL ORDERING

Observation 1

If G has a topological ordering, then G is a DAG.

Key Property

In every DAG G , there is a node v with no incoming edges.

Proof (Exercise)

DAGs AND TOPOLOGICAL ORDERING

Observation 1

If G has a topological ordering, then G is a DAG.

Key Property

In every DAG G , there is a node v with no incoming edges.

Proof (Exercise)

- By way of contradiction, assume all nodes in G have an incoming edge.

DAGs AND TOPOLOGICAL ORDERING

Observation 1

If G has a topological ordering, then G is a DAG.

Key Property

In every DAG G , there is a node v with no incoming edges.

Proof (Exercise)

- By way of contradiction, assume all nodes in G have an incoming edge.
- Pick an arbitrary node u and follow the incoming node back to v . Since all nodes have an incoming edge, when can repeat this infinitely.

DAGs AND TOPOLOGICAL ORDERING

Observation 1

If G has a topological ordering, then G is a DAG.

Key Property

In every DAG G , there is a node v with no incoming edges.

Proof (Exercise)

- By way of contradiction, assume all nodes in G have an incoming edge.
- Pick an arbitrary node u and follow the incoming node back to v . Since all nodes have an incoming edge, when can repeat this infinitely.
- After visiting $|V| + 1$ nodes, by the Pigeon Hole principle, we have visited some node w twice $\implies G$ contains a cycle.

DAGs AND TOPOLOGICAL ORDERING

Observation 1

If G has a topological ordering, then G is a DAG.

Key Property

In every DAG G , there is a node v with no incoming edges.

- The Key Property allows us to show that all DAGs have a topological ordering.

DAGs AND TOPOLOGICAL ORDERING

Observation 1

If G has a topological ordering, then G is a DAG.

Key Property

In every DAG G , there is a node v with no incoming edges.

- The Key Property allows us to show that all DAGs have a topological ordering.
- Prove it by induction.

DAGs AND TOPOLOGICAL ORDERING

Observation 1

If G has a topological ordering, then G is a DAG.

Key Property

In every DAG G , there is a node v with no incoming edges.

- The Key Property allows us to show that all DAGs have a topological ordering.
- Prove it by induction.
- Does the inductive proof imply an algorithm to build a topological ordering from a DAG? If so, what is it?

APPENDIX

REFERENCES

IMAGE SOURCES I



WISCONSIN
UNIVERSITY OF WISCONSIN-MADISON

<https://brand.wisc.edu/web/logos/>