# CMP_SC 3050: Connected Components of Directed Graphs

Rohit Chadha

September 18, 2014

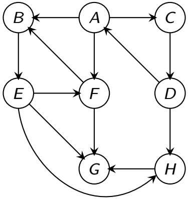
# Strongly connected component of a digraph

- $u$ is said to strongly connected to $v$ if there is a directed path from $u$ to $v$ and from $v$ to $u$.

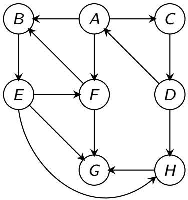
# Strongly connected component of a digraph

- $u$ is said to strongly connected to $v$ if there is a directed path from $u$ to $v$ and from $v$ to $u$.
- The strongly connected component (SCC) of $u$ is the set of all vertices strongly connected to $u$. We shall write the SCC of $u$ as $\mathrm{SCC(u)}$

# Example



# Example




$$
\begin{aligned}
\mathrm{SCC}(A) &= \{A, C, D\} \\
\mathrm{SCC}(B) &= \{B, E, F\} \\
\mathrm{SCC}(C) &= \{A, C, D\} \\
\mathrm{SCC}(D) &= \{A, C, D\} \\
\mathrm{SCC}(E) &= \{B, E, F\} \\
\mathrm{SCC}(F) &= \{B, E, F\} \\
\mathrm{SCC}(G) &= \{G\} \\
\mathrm{SCC}(H) &= \{H\}
\end{aligned}
$$

# Properties of Strongly Connected Components

- For any two vertices $u, v$, either they have the same SCC or their SCCs have no vertex in common

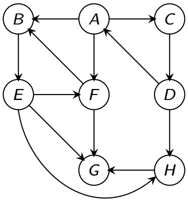
# Properties of Strongly Connected Components

- For any two vertices $u, v$, either they have the same SCC or their SCCs have no vertex in common
- In a dag, for every vertex $u$, $\mathrm{SCC}(u) = \{u\}$.

# Properties of Strongly Connected Components

- For any two vertices $u, v$, either they have the same SCC or their SCCs have no vertex in common
- In a dag, for every vertex $u$, $\mathrm{SCC}(u) = \{u\}$.
- Strongly connected component of a graph $G$ is the set of all SCCs. We shall call this set $\mathrm{SCC(G)}$

# Example




The strongly connected components are $\{B, E, F\}$, $\{A, C, D\}$, $\{G\}$ and $\{H\}$

# Component graph of digraph

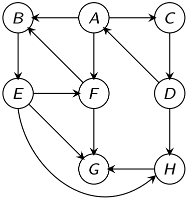



Figure : Graph $G$

Let $C_1, C_2, \ldots C_k$ be the SCCs of $G$. The component graph of $G$ is the graph where

# Component graph of digraph

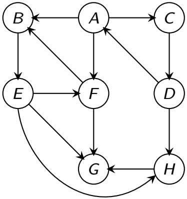



Figure : Graph $G$

Let $C_1, C_2, \ldots C_k$ be the SCCs of $G$. The component graph of $G$ is the graph where

- Vertices are $C_1, C_2, \ldots C_k$

# Component graph of digraph

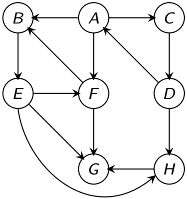
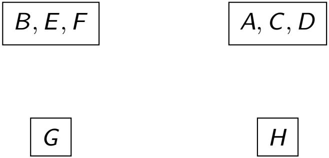



Figure : Graph $G$




Let $C_1, C_2, \ldots C_k$ be the SCCs of $G$. The component graph of $G$ is the graph where

- Vertices are $C_1, C_2, \ldots C_k$

# Component graph of digraph

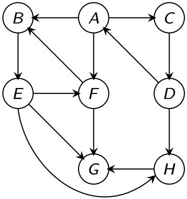
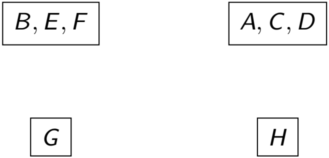



Figure : Graph $G$




Let $C_1, C_2, \ldots C_k$ be the SCCs of $G$. The component graph of $G$ is the graph where

- Vertices are $C_1, C_2, \ldots C_k$
- There is an edge $(C_i, C_j)$ if $i \neq j$ and there is some $u \in C_i$ and $v \in C_j$ such that $(u, v)$ is an edge in $G$.

# Component graph of digraph

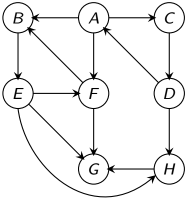



Figure : Graph $G$

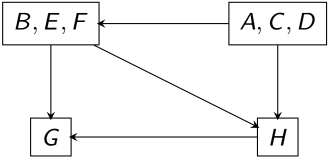



Figure : Component Graph of $G$

Let $C_1, C_2, \ldots C_k$ be the SCCs of $G$. The component graph of $G$ is the graph where

- Vertices are $C_1, C_2, \ldots C_k$
- There is an edge $(C_i, C_j)$ if $i \neq j$ and there is some $u \in C_i$ and $v \in C_j$ such that $(u, v)$ is an edge in $G$.

# Component graph of digraph

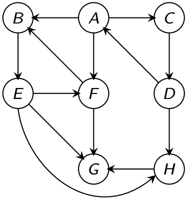



Figure : Graph $G$

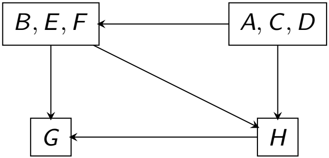



Figure : Component Graph of $G$

Let $C_1, C_2, \ldots C_k$ be the SCCs of $G$. The component graph of $G$ is the graph where

- Vertices are $C_1, C_2, \ldots C_k$
- There is an edge $(C_i, C_j)$ if $i \neq j$ and there is some $u \in C_i$ and $v \in C_j$ such that $(u, v)$ is an edge in $G$.

Component graph is always a dag

# Strongly connected graphs

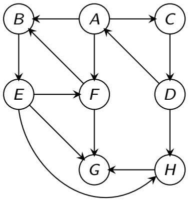



Figure : Not strongly connected

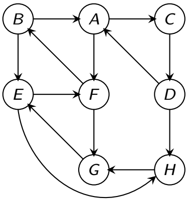



Figure : Strongly connected

A graph is strongly connected if it has exactly one strongly connected component.

# Algorithmic questions

1. Give an algorithm that given a graph $G$, vertices $u$ and $v$ checks if $u$ is connected to $v$

2. Give an algorithm that given a graph $G$ and vertex $u$, outputs $\mathrm{SCC(u)}$ (the strongly connected component of $u$)
3. Give an algorithm that given a graph $G$, outputs $\mathrm{SCC(G)}$

# Algorithmic questions

1. Give an algorithm that given a graph $G$, vertices $u$ and $v$ checks if $u$ is connected to $v$
   (a) Check if $v$ is reachable from $u$ (use BFS or DFS)?
   (b) Check if $u$ is reachable from $v$?
   (c) $u$ is connected to $v$ iff the answers to the questions (a) and (b) is Yes
   (d) Runs in $O(|V| + |E|)$ time
2. Give an algorithm that given a graph $G$ and vertex $u$, outputs $\mathrm{SCC(u)}$ (the strongly connected component of $u$)
3. Give an algorithm that given a graph $G$, outputs $\mathrm{SCC(G)}$

# Transpose of a digraph

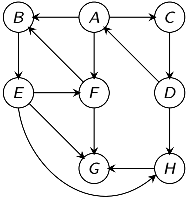



Figure : Graph $G$

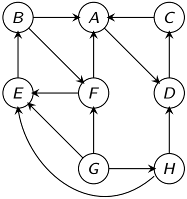



Figure : $G^T$

The transpose of digraph $G$ is the digraph $G^T$ in which all the edges of $G$ are reversed

- The set of vertices of $G^T$ is the same as the set of vertices of $G$
- $u$ is adjacent to $v$ iff $v$ is adjacent to $u$

# Properties of transpose of a digraph

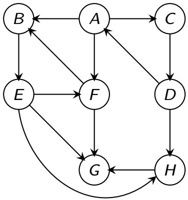



Figure : Graph $G$

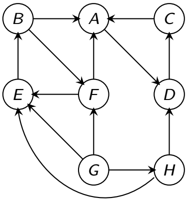



Figure : $G^T$

- If $u$ is reachable from $v$ in $G$ then $v$ is reachable from $u$ in $G^T$

# Properties of transpose of a digraph

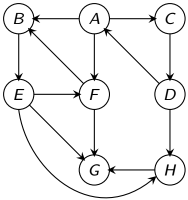



Figure : Graph $G$

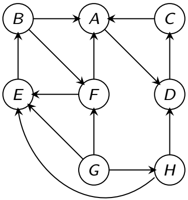



Figure : $G^T$

- If $u$ is reachable from $v$ in $G$ then $v$ is reachable from $u$ in $G^T$
- $u$ is connected to $v$ in $G$ iff $u$ is connected to $v$ in $G^T$

# Properties of transpose of a digraph

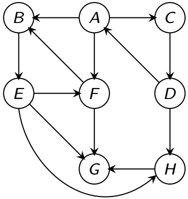



Figure : Graph $G$

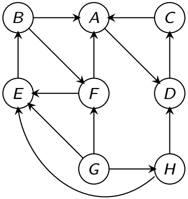



Figure : $G^T$

- If $u$ is reachable from $v$ in $G$ then $v$ is reachable from $u$ in $G^T$
- $u$ is connected to $v$ in $G$ iff $u$ is connected to $v$ in $G^T$
- The connected components of $G$ and $G^T$ are exactly the same

# Properties of transpose of a digraph

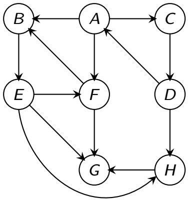



Figure : Graph $G$

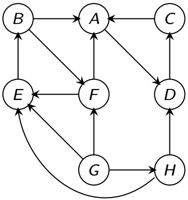



Figure : $G^T$

- If $u$ is reachable from $v$ in $G$ then $v$ is reachable from $u$ in $G^T$
- $u$ is connected to $v$ in $G$ iff $u$ is connected to $v$ in $G^T$
- The connected components of $G$ and $G^T$ are exactly the same
- The component graph of SCCs of $G^T$ is the same as the transpose of component graph of $G$

# Connected component of a vertex $u$ in $G$

## Algorithm

1. Output all vertices reachable from $u$ (run BFS or DFS on source vertex $u$)

# Connected component of a vertex $u$ in $G$

## Algorithm

1. Output all vertices reachable from $u$ (run BFS or DFS on source vertex $u$)
2. Compute the adjacency list representation of $G^T$

# Connected component of a vertex $u$ in $G$

## Algorithm

1. Output all vertices reachable from $u$ (run BFS or DFS on source vertex $u$)
2. Compute the adjacency list representation of $G^T$
3. Output all vertices reachable from $u$ in $G^T$

# Connected component of a vertex $u$ in $G$

Algorithm

1. Output all vertices reachable from $u$ (run BFS or DFS on source vertex $u$)
2. Compute the adjacency list representation of $G^T$
3. Output all vertices reachable from $u$ in $G^T$
4. The vertices that are output in both steps 1 and 3 constitute $\mathrm{SCC}(\mathrm{u})$

# Connected component of a vertex $u$ in $G$

## Algorithm

1. Output all vertices reachable from $u$ (run BFS or DFS on source vertex $u$)
2. Compute the adjacency list representation of $G^T$
3. Output all vertices reachable from $u$ in $G^T$
4. The vertices that are output in both steps 1 and 3 constitute $\mathrm{SCC(u)}$
5. Runs in $\Theta(|V| + |E|)$ time

# An Example

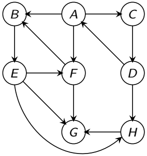



Figure : Graph $G$

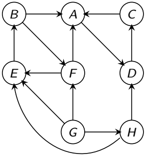



Figure : $G^T$

# An Example

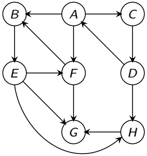



Figure : Graph $G$

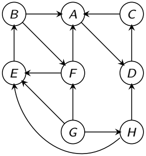



Figure : $G^T$

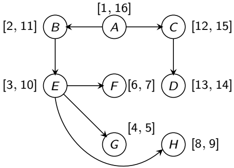



Figure : DFS of $G$ with source $A$

# An Example

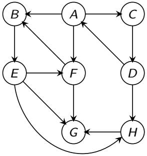



Figure : Graph $G$

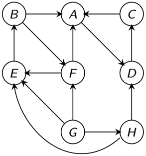



Figure : $G^T$

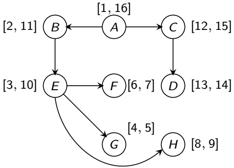



Figure : DFS of $G$ with source $A$

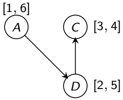



Figure : DFS of $G^T$ with source $A$

# An Example

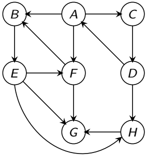



Figure : Graph $G$

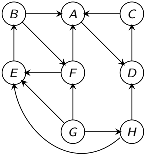



Figure : $G^T$

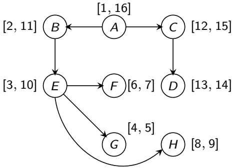



Figure : DFS of $G$ with source $A$

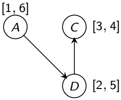



Figure : DFS of $G^T$ with source $A$

$$\mathrm{SCC(A)} = \{\mathrm{A}, \mathrm{C}, \mathrm{D}\}$$

# Computing all Strongly Connected Components

1. Initially the set of all SCCs is empty
2. Pick a vertex $u$ which is not in a previously computed SCC. If there is no such $u$, then we are done
3. Compute $\mathrm{SCC(u)}$ and add it to the set of all SCCs
4. Go back to step 2

# Computing all Strongly Connected Components

1. Initially the set of all SCCs is empty
2. Pick a vertex $u$ which is not in a previously computed SCC. If there is no such $u$, then we are done
3. Compute $\mathrm{SCC}(\mathrm{u})$ and add it to the set of all SCCs
4. Go back to step 2

Running time is $O(|V| \cdot (|V| + |E|))$

# Computing all Strongly Connected Components

1. Initially the set of all SCCs is empty
2. Pick a vertex $u$ which is not in a previously computed SCC. If there is no such $u$, then we are done
3. Compute $\mathrm{SCC(u)}$ and add it to the set of all SCCs
4. Go back to step 2

Running time is $O(|V| \cdot (|V| + |E|))$

There is an algorithm to compute all SCCs using just 2 DFSs!

# Recall: DFS of a graph

We extend DFS to the whole graph!

1. Pick a vertex
2. Perform a DFS on the picked vertex
3. Pick a new vertex which has not been reached as yet and repeat Step 2

# Computing all Strongly Connected Components using 2 DFSs

1. Compute the DFS of the whole graph $G$

# Computing all Strongly Connected Components using 2 DFSs

1. Compute the DFS of the whole graph $G$
2. Store the vertices in a list $L$ with decreasing order of finish times.

# Computing all Strongly Connected Components using 2 DFSs

1. Compute the DFS of the whole graph $G$
2. Store the vertices in a list $L$ with decreasing order of finish times.
3. Compute $G^T$

# Computing all Strongly Connected Components using 2 DFSs

1. Compute the DFS of the whole graph $G$
2. Store the vertices in a list $L$ with decreasing order of finish times.
3. Compute $G^T$
4. Do DFS of $G^T$ but pick vertices in the order of their appearance in list $L$ when picking a new vertex (in step 3 on previous slide)

# Computing all Strongly Connected Components using 2 DFSs

1. Compute the DFS of the whole graph $G$
2. Store the vertices in a list $L$ with decreasing order of finish times.
3. Compute $G^T$
4. Do DFS of $G^T$ but pick vertices in the order of their appearance in list $L$ when picking a new vertex (in step 3 on previous slide)
5. Output the vertices in each tree of the depth-first forest formed in second DFS as a separate SCC

# Computing all Strongly Connected Components using 2 DFSs

1. Compute the DFS of the whole graph $G$
2. Store the vertices in a list $L$ with decreasing order of finish times.
3. Compute $G^T$
4. Do DFS of $G^T$ but pick vertices in the order of their appearance in list $L$ when picking a new vertex (in step 3 on previous slide)
5. Output the vertices in each tree of the depth-first forest formed in second DFS as a separate SCC

Runs in $O(|V| + |E|)$ time

# An Example

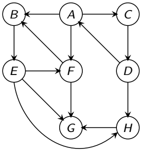



Figure : Graph $G$

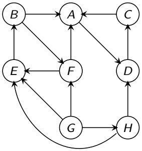



Figure : $G^T$

Components of $G$ are

# An Example

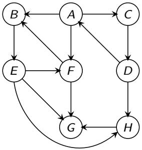



Figure : Graph $G$

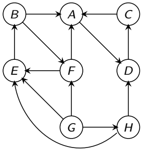



Figure : $G^T$

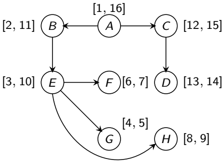



Figure : DFS of $G$

Components of $G$ are

# An Example

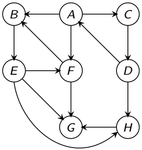



Figure : Graph $G$

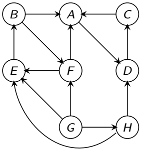



Figure : $G^T$

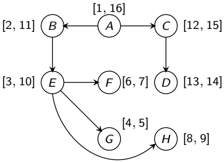



Figure : DFS of $G$

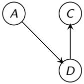



Figure : DFS of $G^T$

Components of $G$ are $\{A, D, C\}$;

# An Example

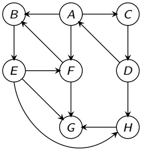



Figure : Graph $G$

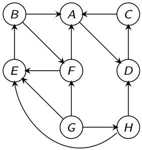



Figure : $G^T$

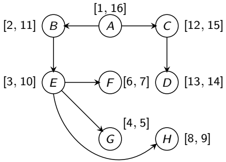



Figure : DFS of $G$

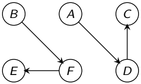



Figure : DFS of $G^T$

Components of $G$ are $\{A, D, C\}$; $\{B, F, E\}$;

# An Example

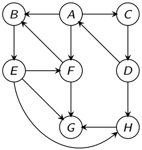



Figure : Graph $G$

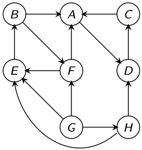



Figure : $G^T$

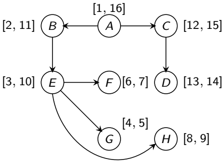



Figure : DFS of $G$

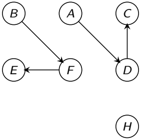



Figure : DFS of $G^T$

Components of $G$ are $\{A, D, C\}$; $\{B, F, E\}$; $\{H\}$;

# An Example

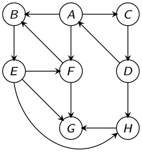



Figure : Graph $G$

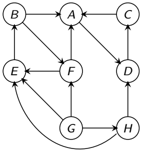



Figure : $G^T$

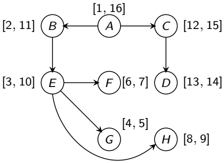



Figure : DFS of $G$

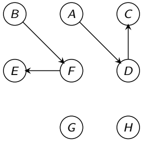



Figure : DFS of $G^T$

Components of $G$ are $\{A, D, C\}$; $\{B, F, E\}$; $\{H\}$; and $\{G\}$

# Why does this algorithm work?

Extend start times and finish times to a component $C$ as follows:

- $C.d$ is the smallest starting time of vertices in $C$ in the DFS of G, and
- $C.f$ is the largest finish time of vertices in $C$ in the DFS of $G$

# Why does this algorithm work?

Extend start times and finish times to a component $C$ as follows:

- $C.d$ is the smallest starting time of vertices in $C$ in the DFS of G, and
- $C.f$ is the largest finish time of vertices in $C$ in the DFS of $G$

Key Idea: By considering vertices in second DFS in decreasing order of finishing times from first DFS, we are visiting vertices of the component graph in topological sort order

# Example

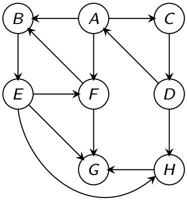



Figure : Graph $G$

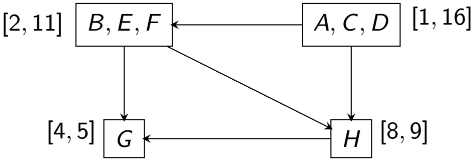



Figure : Component Graph of $G$

In the example, we first found $\{A, C, D\}$ then $\{B, E, F\}$ then $\{H\}$ and finally $G$

# Component graph of $G$ and finish times

Fact: If $C_1$ and $C_2$ are distinct SCCs in $G$ and there is an edge from $C_1$ to $C_2$ in the component graph of $G$ then $C_1.f > C_2.f$

# Back to Example

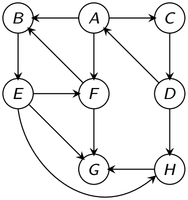



Figure : Graph $G$

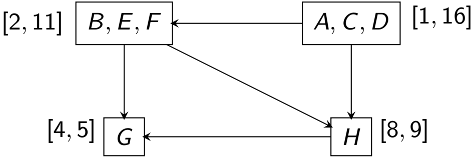



Figure : Component Graph of $G$

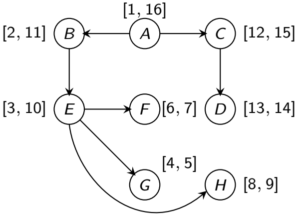



Figure : DFS of $G$

# Component graph of $G$ and finish times

Fact: If $C_1$ and $C_2$ are distinct SCCs in $G$ and there is an edge from $C_1$ to $C_2$ in the component graph of $G$ then $C_1.f > C_2.f$

Why is this fact true?

Let $u$ be first vertex amongst those in $C_1$ or $C_2$ that is visited.

- If $u \in C_1$ then all of $C_2$ will be explored before $\texttt{DFS}(u)$ completes
- If $u \in C_2$ then all of $C_2$ will be explored before any of $C_1$

# Why does the algorithm for connected components work?

From the previous fact, we get a new fact:

If $C_1$ and $C_2$ are distinct SCCs in $G$ such that $C_1.f > C_2.f$ then there cannot be an edge from $C_1$ to $C_2$ in $G^T$

# Why does the algorithm for connected components work?

From the previous fact, we get a new fact:

If $C_1$ and $C_2$ are distinct SCCs in $G$ such that $C_1.f > C_2.f$ then there cannot be an edge from $C_1$ to $C_2$ in $G^T$

- When we do the second DFS on $G^T$, we start with SCC $C$ such that $C.f$ is the maximum finish time amongst all components

# Why does the algorithm for connected components work?

From the previous fact, we get a new fact:
If $C_1$ and $C_2$ are distinct SCCs in $G$ such that $C_1.f > C_2.f$ then there cannot be an edge from $C_1$ to $C_2$ in $G^T$

- When we do the second DFS on $G^T$, we start with SCC $C$ such that $C.f$ is the maximum finish time amongst all components
- The second DFS starts from some vertex $u$ of $C$ and it visits all vertices in $C$

# Why does the algorithm for connected components work?

From the previous fact, we get a new fact:
If $C_1$ and $C_2$ are distinct SCCs in $G$ such that $C_1.f > C_2.f$ then there cannot be an edge from $C_1$ to $C_2$ in $G^T$

- When we do the second DFS on $G^T$, we start with SCC $C$ such that $C.f$ is the maximum finish time amongst all components
- The second DFS starts from some vertex $u$ of $C$ and it visits all vertices in $C$
- From the new fact above, there is no edge from $C$ to a different $C'$ in $G^T$

# Why does the algorithm for connected components work?

From the previous fact, we get a new fact:
If $C_1$ and $C_2$ are distinct SCCs in $G$ such that $C_1.f > C_2.f$ then there cannot be an edge from $C_1$ to $C_2$ in $G^T$

- When we do the second DFS on $G^T$, we start with SCC $C$ such that $C.f$ is the maximum finish time amongst all components
- The second DFS starts from some vertex $u$ of $C$ and it visits all vertices in $C$
- From the new fact above, there is no edge from $C$ to a different $C'$ in $G^T$
- So DFS of $u$ in $G^T$ only visits the vertices of $C$

# Why does the algorithm for connected components work?

From the previous fact, we get a new fact:
If $C_1$ and $C_2$ are distinct SCCs in $G$ such that $C_1.f > C_2.f$ then there cannot be an edge from $C_1$ to $C_2$ in $G^T$

- When we do the second DFS on $G^T$, we start with SCC $C$ such that $C.f$ is the maximum finish time amongst all components
- The second DFS starts from some vertex $u$ of $C$ and it visits all vertices in $C$
- From the new fact above, there is no edge from $C$ to a different $C'$ in $G^T$
- So DFS of $u$ in $G^T$ only visits the vertices of $C$
- The next root chosen in the second DFS is in SCC $C_1$ such that $C_1.f$ is maximum over all SCCs other than $C$

# Why does the algorithm for connected components work?

From the previous fact, we get a new fact:
If $C_1$ and $C_2$ are distinct SCCs in $G$ such that $C_1.f > C_2.f$ then there cannot be an edge from $C_1$ to $C_2$ in $G^T$

- When we do the second DFS on $G^T$, we start with SCC $C$ such that $C.f$ is the maximum finish time amongst all components
- The second DFS starts from some vertex $u$ of $C$ and it visits all vertices in $C$
- From the new fact above, there is no edge from $C$ to a different $C'$ in $G^T$
- So DFS of $u$ in $G^T$ only visits the vertices of $C$
- The next root chosen in the second DFS is in SCC $C_1$ such that $C_1.f$ is maximum over all SCCs other than $C$
- Thus DFS now visits all vertices in $C_1$

# Why does the algorithm for connected components work?

From the previous fact, we get a new fact:
If $C_1$ and $C_2$ are distinct SCCs in $G$ such that $C_1.f > C_2.f$ then there cannot be an edge from $C_1$ to $C_2$ in $G^T$

- When we do the second DFS on $G^T$, we start with SCC $C$ such that $C.f$ is the maximum finish time amongst all components
- The second DFS starts from some vertex $u$ of $C$ and it visits all vertices in $C$
- From the new fact above, there is no edge from $C$ to a different $C'$ in $G^T$
- So DFS of $u$ in $G^T$ only visits the vertices of $C$
- The next root chosen in the second DFS is in SCC $C_1$ such that $C_1.f$ is maximum over all SCCs other than $C$
- Thus DFS now visits all vertices in $C_1$
- The only edges out of $C_1$ in $G^T$ go to $C$ which we have already visited

# Why does the algorithm for connected components work?

From the previous fact, we get a new fact:

If $C_1$ and $C_2$ are distinct SCCs in $G$ such that $C_1.f > C_2.f$ then there cannot be an edge from $C_1$ to $C_2$ in $G^T$

- When we do the second DFS on $G^T$, we start with SCC $C$ such that $C.f$ is the maximum finish time amongst all components
- The second DFS starts from some vertex $u$ of $C$ and it visits all vertices in $C$
- From the new fact above, there is no edge from $C$ to a different $C'$ in $G^T$
- So DFS of $u$ in $G^T$ only visits the vertices of $C$
- The next root chosen in the second DFS is in SCC $C_1$ such that $C_1.f$ is maximum over all SCCs other than $C$
- Thus DFS now visits all vertices in $C_1$
- The only edges out of $C_1$ in $G^T$ go to $C$ which we have already visited
- Therefore, tree edges will be only to vertices in $C_1$

# Why does the algorithm for connected components work?

From the previous fact, we get a new fact:
If $C_1$ and $C_2$ are distinct SCCs in $G$ such that $C_1.f > C_2.f$ then there cannot be an edge from $C_1$ to $C_2$ in $G^T$

- When we do the second DFS on $G^T$, we start with SCC $C$ such that $C.f$ is the maximum finish time amongst all components
- The second DFS starts from some vertex $u$ of $C$ and it visits all vertices in $C$
- From the new fact above, there is no edge from $C$ to a different $C'$ in $G^T$
- So DFS of $u$ in $G^T$ only visits the vertices of $C$
- The next root chosen in the second DFS is in SCC $C_1$ such that $C_1.f$ is maximum over all SCCs other than $C$
- Thus DFS now visits all vertices in $C_1$
- The only edges out of $C_1$ in $G^T$ go to $C$ which we have already visited
- Therefore, tree edges will be only to vertices in $C_1$
- The process continues

# Summary

- We have several studied fundamental graph algorithms
- DFS and BFS are the two basic strategies for traversing a graph
- BFS was used to find distances from a vertex in a graph
- We have seen applications of DFS to finding topological sort of dag and also to find connected components in a graph
- We have finished Chapter 22 of the book. We shall start Sections 16.1 and 16.2 next week before coming back to graphs