Trees and Forests

**Definition:** A tree is a connected graph without any cycles (disregarding directions of edges).

**Note:** In computing we use rooted trees, i.e., a distinguished vertex is chosen as the root.

**Definition:** A forest is a collection of trees.

**DFS Forests:**
A DFS traversing a graph builds up a forest:
- vertices are all vertices of the graph,
- edges are those traversed during the DFS.

DFS Forests Example

Reminder: Recursive DFS

**Algorithm** $\text{dfs}(G)$
1. Initialise Boolean array $\text{visited}$ by setting all entries to FALSE
2. for all $v \in V$
3. \hspace{1em} if not $\text{visited}[v]$ then
4. \hspace{2em} $\text{dfsFromVertex}(G, v)$

**Algorithm** $\text{dfsFromVertex}(G, v)$
1. $\text{visited}[v] \leftarrow \text{TRUE}$
2. for all $w$ adjacent to $v$
3. \hspace{1em} if not $\text{visited}[w]$ then
4. \hspace{2em} $\text{dfsFromVertex}(G, w)$

**Runtime:** $T(n, m) = \Theta(n + m)$, using Adjacency List representation.
Connected components of an undirected graph

\[ G = (V, E) \text{ undirected graph} \]

**Definition**

- A subset \( C \) of \( V \) is **connected** if for all \( v, w \in C \) there is a path from \( v \) to \( w \) (if \( G \) is directed, say **strongly connected**).
- A **connected component** of \( G \) is a **maximal connected subset** \( C \) of \( V \).
  - **Maximal** means no connected subset \( C' \) of \( V \) strictly contains \( C \).
- \( G \) is **connected** if it only has one connected component, i.e., if for all vertices \( v, w \) there is a path from \( v \) to \( w \).

Algorithm  \( \text{connComp}(G) \)

1. Initialise Boolean array \( \text{visited} \) by setting all entries to \( \text{FALSE} \)
2. for all \( v \in V \) do
3.  
4.      if \( \text{visited}[v] = \text{FALSE} \) then
5.          print “New Component”
6.      ccFromVertex\((G, v)\)

Algorithm  \( \text{ccFromVertex}(G, v) \)

1. \( \text{visited}[v] \leftarrow \text{TRUE} \)
2. print \( v \)
3. for all \( w \) adjacent to \( v \) do
4.      if \( \text{visited}[w] = \text{FALSE} \) then
5.          ccFromVertex\((G, w)\)
Topological Sorting

**Example:**
10 tasks to be carried out. Some of them depend on others.

- Task 0 must be completed before Task 1 can be started.
- Task 1 and Task 2 must be done before Task 3 can start.
- Task 4 must be done before Task 0 or Task 2 can start.
- Task 5 must be done before Task 0 or Task 4 can start.
- Task 6 must be done before Task 4, 5 or 7 can start.
- Task 7 must be done before Task 0 or Task 9 can start.
- Task 8 must be done before Task 7 or Task 9 can start.
- Task 9 must be done before Task 2 or Task 3 can start.

**Example (continued)**

![Diagram of a directed graph](image)

Does this graph have a topological order?

Yes, the topological sort is:

\[ 8 \prec 6 \prec 7 \prec 9 \prec 5 \prec 4 \prec 2 \prec 0 \prec 1 \prec 3. \]

Topological order

**Definition**

Let \( G = (V, E) \) be a directed graph. A topological order of \( G \) is a total order \( \prec \) of the vertex set \( V \) such that for all edges \((v, w) \in E\) we have \( v \prec w \).

**Topological order (continued)**

A digraph that has a cycle does not have a topological order.

**Definition**

A DAG (directed acyclic graph) is a digraph without cycles.

**Theorem**

A digraph has a topological order if and only if it is a DAG.
Classification of vertices during DFS

Let $G$ be a graph and $v$ a vertex of $G$. Consider the moment during the execution of $\text{dfs}(G)$ when $\text{dfsFromVertex}(G, v)$ is started. Then for all vertices $w$ we have:

1. If $w$ is white and reachable from $v$, then $w$ will be black before $v$.
2. If $w$ is grey, then $v$ is reachable from $w$.

Lemma

Let $G$ be a graph and $v$ a vertex of $G$. Consider $\text{dfs}(G)$.

Algorithm $\text{topSort}(G)$

1. Initialise array $\text{state}$ by setting all entries to white.
2. Initialise linked list $L$
3. for all $v \in V$ do
   4. if $\text{state}[v] = \text{white}$ then
       5. $\text{sortFromVertex}(G, v)$
5. print $L$
Algorithm sortFromVertex($G, v$)

1. $\text{state}[v] \leftarrow \text{grey}$
2. for all $w$ adjacent to $v$ do
3. \hspace{1em} if $\text{state}[w] = \text{white}$ then
4. \hspace{2em} sortFromVertex($G, w$)
5. \hspace{1em} else if $\text{state}[w] = \text{grey}$ then
6. \hspace{2em} print “$G$ has a cycle”
7. \hspace{1em} halt
8. $\text{state}[v] \leftarrow \text{black}$
9. $L.\text{insertFirst}(v)$