Applications of DFS

1. For a undirected graph,
   (a) a DFS produces only Tree and Back edges
Applications of DFS

1. For an undirected graph,
   (a) a DFS produces only **Tree** and **Back** edges
   (b) an **acyclic** (tree) iff a DFS yields **no Back** edges

2. A directed graph is **acyclic** iff a DFS yields **no back** edges, i.e., **DAG (directed acyclic graph)**

3. Topological sort of a DAG

4. Connected components of an undirected graph (see Homework 6)

5. Strongly connected components of a directed graph (see Sec. 22.5 of [CLRS, 3rd ed.])
Applications of DFS

1. For a undirected graph,
   (a) a DFS produces only Tree and Back edges
   (b) acyclic (tree) iff a DFS yields no Back edges

2. A directed graph is acyclic iff a DFS yields no back edges, i.e.,
   DAG (directed acyclic graph) \iff no back edges
Applications of DFS

1. For a undirected graph,
   (a) a DFS produces only Tree and Back edges
   (b) acyclic (tree) iff a DFS yields no Back edges

2. A directed graph is acyclic iff a DFS yields no back edges, i.e.,
   DAG (directed acyclic graph) ⇔ no back edges

3. Topological sort of a DAG – next
Applications of DFS

1. For a undirected graph,
   (a) a DFS produces only **Tree** and **Back** edges
   (b) **acyclic** (tree) iff a DFS yeilds **no Back** edges

2. A directed graph is **acyclic** iff a DFS yields **no back** edges, i.e.,
   
   DAG (directed acyclic graph) ⇔ no back edges

3. Topological sort of a DAG – **next**

4. Connected components of a undirected graph (see Homework 6)
Applications of DFS

1. For a undirected graph,
   (a) a DFS produces only Tree and Back edges
   (b) acyclic (tree) iff a DFS yields no Back edges

2. A directed graph is acyclic iff a DFS yields no back edges, i.e.,
   DAG (directed acyclic graph) ⇔ no back edges

3. Topological sort of a DAG – next

4. Connected components of a undirected graph (see Homework 6)

5. Strongly connected components of a directed graph (see Sec.22.5 of [CLRS,3rd ed.])
A topological sort (TS) of a DAG $G = (V, E)$ is a linear ordering of all its vertices such that if $(u, v) \in E$, then $u$ appears before $v$. A TS is not possible if $G$ has a cycle. The ordering is not necessarily unique.
A topological sort (TS) of a DAG $G = (V, E)$ is a linear ordering of all its vertices such that if $(u, v) \in E$, then $u$ appears before $v$.

Example: given a DAG
A topological sort (TS) of a DAG $G = (V, E)$ is a linear ordering of all its vertices such that if $(u, v) \in E$, then $u$ appears before $v$.

Example: given a DAG

Linear ordering:
A topological sort (TS) of a DAG $G = (V, E)$ is a linear ordering of all its vertices such that if $(u, v) \in E$, then $u$ appears before $v$.

**Example:** given a DAG

![Graph diagram]

Linear ordering:

![Linear ordering diagram]

A TS is not possible if $G$ has a cycle.
A topological sort (TS) of a DAG $G = (V, E)$ is a linear ordering of all its vertices such that if $(u, v) \in E$, then $u$ appears before $v$.

Example: given a DAG

A TS is not possible if $G$ has a cycle.

The ordering is not necessarily unique.
Topological sort

Applications: call-graph
Topological sort

Applications: call-graph
Topological sort

- TS algorithm
  1. run DFS(G) to compute finishing times $f[v]$ for all $v \in V$
  2. output vertices in order of decreasing times

Running time: $\Theta(|V| + |E|)$
Topological sort

- **TS algorithm**
  1. run DFS(G) to compute finishing times $f[v]$ for all $v \in V$
  2. output vertices in order of decreasing times

- Running time: $\Theta(|V| + |E|)$
Topological sort

Example: “Getting-dressed-graph” and DFS

The following simple algorithm topologically sorts a dag:

TOPOLOGICAL-SORT(G)
1 call DFS(G) to compute finishing times f
2 for each vertex v as each vertex is finished, insert it onto the front of a linked list
3 return the linked list of vertices

Figure 22.7(b) shows how the topologically sorted vertices appear in reverse order of their finishing times.

We can perform a topological sort in time $O(V + E)$, since depth-first search takes $O(V + E)$ time and it takes $O(1)$ time to insert each of the $|V|$ vertices onto the front of the linked list.

We prove the correctness of this algorithm using the following key lemma characterizing directed acyclic graphs.
Topological sort

Example: “Getting-dressed-graph” and DFS

The following simple algorithm topologically sorts a dag:

TOPOLOGICAL-SORT $G$
1. call DFS $G$ to compute finishing times $f$
2. for each vertex $v$
3. as each vertex is finished, insert it onto the front of a linked list
4. return the linked list of vertices

Figure 22.7(b) shows how the topologically sorted vertices appear in reverse order of their finishing times.

We can perform a topological sort in time $O(V + E)$, since depth-first search takes $O(V + E)$ time and it takes $O(1)$ time to insert each of the $V$ vertices onto the front of the linked list.

We prove the correctness of this algorithm using the following key lemma characterizing directed acyclic graphs.
Topological sort

**Theorem (correctness of the algorithm):**
TS(G) produces a topological sort of a DAG G.
**Topological sort**

**Theorem (correctness of the algorithm):**

TS(G) produces a topological sort of a DAG G.

Proof: *Just need to show that if* \((u, v) \in E\), *then* \(f[v] < f[u]\).

When we explore edge \((u, v)\), \(u\) is gray, what’s the color of \(v\)?

- **Is \(v\) gray too?**
  
  no, because then \(v\) would be ancestor of \(u\), edge \((u, v)\) is a back edge, a contradiction of a DAG.

- **Is \(v\) white?**
  
  yes, then \(v\) is descendant of \(u\), by DFS, \(d[u] < d[v] < f[v] < f[u]\)

- **Is \(v\) black?**
  
  yes, then \(v\) is already finished. Since we’re exploring \((u, v)\), we have not yet finished \(u\), therefore \(f[v] < f[u]\)