Applications of DFS

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

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

$$\text{DAG (directed acyclic graph)} \quad \Leftrightarrow \quad \text{no back edges}$$

Applications of DFS

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

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

   $$\text{DAG (directed acyclic graph)} \quad \Leftrightarrow \quad \text{no back edges}$$

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.,

   $$\text{DAG (directed acyclic graph)} \quad \Leftrightarrow \quad \text{no back edges}$$

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 yeilds **no Back** edges
2. A directed graph is **acyclic** iff a DFS yields **no back** edges, i.e.,

   $$\text{DAG (directed acyclic graph)} \quad \Leftrightarrow \quad \text{no back edges}$$
3. Topological sort of a DAG – next
4. Connected components of a undirected graph (see Homework 6)
5. Strongly connected components of a drected graph (see Sec.22.5 of [CLRS,3rd ed.])

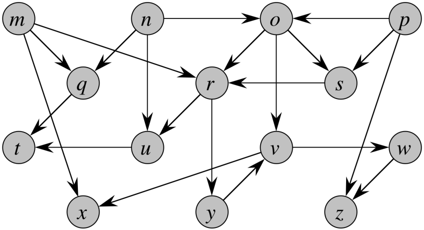
Topological sort

- 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.

Topological sort

- 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.

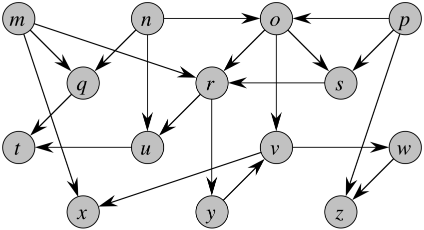
  Example: given a DAG



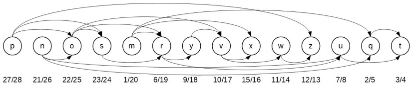
Topological sort

- 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



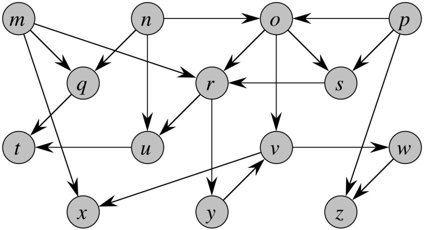
  Linear ordering:



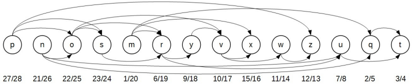
Topological sort

- 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:

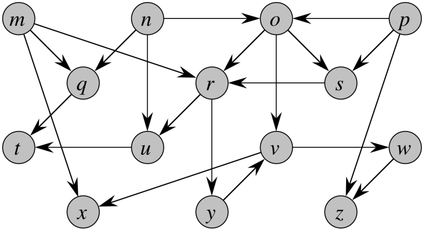


- A TS is not possible if G has a cycle.

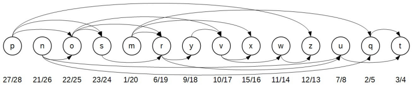
Topological sort

- 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



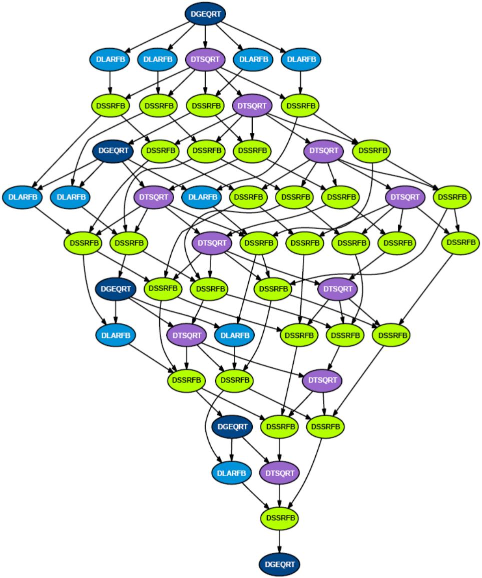
  Linear ordering:



- A TS is not possible if G has a cycle.
- The ordering is not necessarily unique.

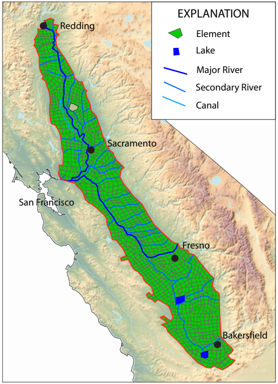
Topological sort

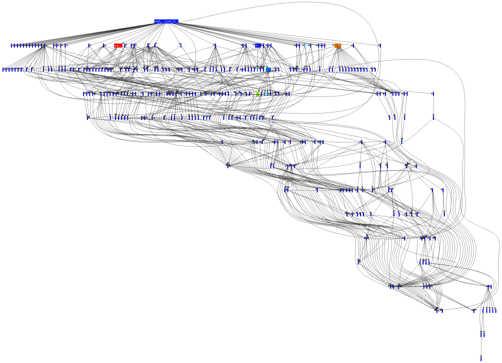
Applications: call-graph



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

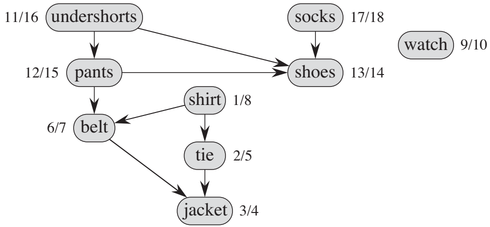
Topological sort

- 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: $\Theta(|V| + |E|)$

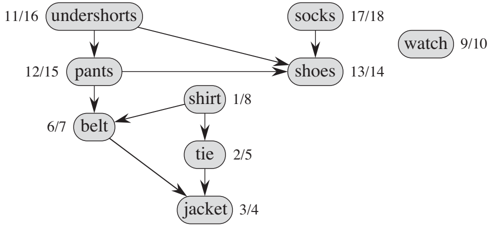
Topological sort

Example: "Getting-dressed-graph" and DFS

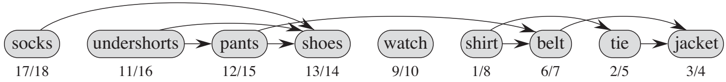


Topological sort

Example: "Getting-dressed-graph" and DFS



Topologically sorted



Topological sort

Theorem (correctness of the algorithm):
TS(G) produces a toplogical sort of a DAG G.

Topological sort

Theorem (correctness of the algorithm):
TS(G) produces a toplogical 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 $f[v] < f[u]$