Topological Sort
(an application of DFS)

CSC263 Tutorial 9
Topological sort

- We have a **set of tasks** and a **set of dependencies (precedence constraints)** of form "task A must be done before task B"
- **Topological sort**: An ordering of the tasks that conforms with the given dependencies
- **Goal**: Find a topological sort of the tasks or decide that there is no such ordering
Examples

• **Scheduling**: When scheduling *task graphs* in distributed systems, usually we first need to **sort the tasks topologically**
  ...and then assign them to resources (the most efficient scheduling is an NP-complete problem)

• Or during compilation to order modules/libraries
Examples

• **Resolving dependencies**: *apt-get* uses topological sorting to obtain the admissible sequence in which a set of Debian packages can be installed/removed
Topological sort more formally

• Suppose that in a **directed** graph $G = (V, E)$ vertices $V$ represent tasks, and each edge $(u, v) \in E$ means that task $u$ must be done before task $v$

• What is an ordering of vertices $1, ..., |V|$ such that for every edge $(u, v)$, $u$ appears before $v$ in the ordering?

• Such an ordering is called a **topological sort of** $G$
• Note: there can be multiple topological sorts of $G$
Topological sort more formally

• Is it possible to execute all the tasks in $G$ in an order that respects all the precedence requirements given by the graph edges?

• The answer is "yes" if and only if the directed graph $G$ has no cycle! (otherwise we have a deadlock)

• Such a $G$ is called a Directed Acyclic Graph, or just a DAG
Algorithm for TS

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

• Note that the result is just a list of vertices in order of *decreasing* finish times \( f[] \)
Edge classification by DFS

Edge \((u,v)\) of \(G\) is classified as a:

(1) Tree edge iff \(u\) discovers \(v\) during the DFS: \(P[v] = u\)

If \((u,v)\) is NOT a tree edge then it is a:

(2) Forward edge iff \(u\) is an ancestor of \(v\) in the DFS tree
(3) Back edge iff \(u\) is a descendant of \(v\) in the DFS tree
(4) Cross edge iff \(u\) is neither an ancestor nor a descendant of \(v\)
Edge classification by DFS

Tree edges
Forward edges
Back edges
Cross edges

The edge classification depends on the particular DFS tree!
Edge classification by DFS

- Tree edges
- Forward edges
- Back edges
- Cross edges

The edge classification depends on the particular DFS tree!
DAGs and back edges

• Can there be a back edge in a DFS on a DAG?
• NO! Back edges close a cycle!
• A graph $G$ is a DAG $\iff$ there is no back edge classified by $\text{DFS}(G)$
Back to topological sort

• TOPOLOGICAL-SORT(G):
  1) call DFS(G) to compute \textit{finishing} times \( f[v] \) for each vertex \( v \)
  2) as each vertex is finished, insert it onto the \textit{front} of a linked list
  3) return the linked list of vertices
Let’s say we start the DFS from the vertex $c$.

Next we discover the vertex $d$.

1) Call DFS($G$) to compute the finishing times $f[v]$.
Topological sort

1) Call DFS(G) to compute the finishing times f[v]

Let’s say we start the DFS from the vertex c

Next we discover the vertex d

Time = 2

Time = 3

Let’s say we start the DFS from the vertex c

Next we discover the vertex d
Topological sort

1) Call DFS(G) to compute the finishing times f[v]

2) as each vertex is finished, insert it onto the front of a linked list

Next we discover the vertex f

f is done, move back to d
Let's say we start the DFS from the vertex c.

Next we discover the vertex d.

Next we discover the vertex f.

f is done, move back to d.

d is done, move back to c.

1) Call DFS(G) to compute the finishing times f[v].

Topological sort:
Topological sort

Let’s say we start the DFS from the vertex c

Next we discover the vertex d

Next we discover the vertex f

f is done, move back to d

d is done, move back to c

Next we discover the vertex e

1) Call DFS(G) to compute the finishing times \( f[v] \)
Topological sort

1) Call DFS(G) to compute the finishing times f[v]

Let’s say we start the DFS from the vertex c

Next we discover the vertex d

Next we discover the vertex e

Both edges from e are cross edges

d is done, move back to c

Next we discover the vertex e

e is done, move back to c
Let's say we start the DFS from the vertex \( c \). Just a note: If there was \( (c, f) \) edge in the graph, it would be classified as a **forward edge** (in this particular DFS run).

1) Call DFS(\( G \)) to compute the finishing times \( f[v] \).

\[
\begin{align*}
&d = \infty \\
&f = \infty \\
&d = \infty \\
&f = \infty \\
&d = 2 \\
&f = 5 \\
&d = 3 \\
&f = 4 \\
&d = 1 \\
&f = \infty \\
&d = 6 \\
&f = 7
\end{align*}
\]

Next we discover the vertex \( d \). \( d \) is done, move back to \( c \).

Next we discover the vertex \( e \). \( e \) is done, move back to \( c \).

\( c \) is done as well.
Let’s now call DFS visit from the vertex a

Next we discover the vertex c, but c was already processed => (a,c) is a cross edge

Next we discover the vertex b

1) Call DFS(G) to compute the finishing times f[v]
Topological sort

1) Call DFS(G) to compute the finishing times \( f[v] \)

Let’s now call DFS visit from the vertex \( a \)

Next we discover the vertex \( c \), but \( c \) was already processed

\( => (a,c) \) is a cross edge

Next we discover the vertex \( b \)

\( b \) is done as \( (b,d) \) is a cross edge

\( => \) now move back to \( c \)
Let’s now call DFS visit from the vertex a.

Next we discover the vertex c, but c was already processed => (a,c) is a cross edge.

Next we discover the vertex b. b is done as (b,d) is a cross edge => now move back to c.

a is done as well.
Let's now call DFS visit from the vertex a
d = 6
f = 7
Time = 11
e
d = 1
f = 8
d = 2
f = 5
d = 3
f = 4

1) Call DFS(G) to compute the finishing times f[v]

Next we discover the vertex c, but c was already processed => (a, c) is a cross edge

Time = 13

Next we discover the vertex b
b is done as (b, d) is a cross edge => now move back to c

b is done as well

WE HAVE THE RESULT!

3) return the linked list of vertices

=> (a, c) is a cross edge

a is done as well

Topological sort
Topological sort

The linked list is sorted in **decreasing** order of finishing times \( f[] \)

Try yourself with different vertex order for DFS visit

Note: If you redraw the graph so that all vertices are in a line ordered by a valid topological sort, then all edges point "from left to right"
Time complexity of TS(G)

- Running time of topological sort:
  \[ \Theta(n + m) \]
  where \( n = |V| \) and \( m = |E| \)
- Why? Depth first search takes \( \Theta(n + m) \) time in the worst case, and inserting into the front of a linked list takes \( \Theta(1) \) time
Proof of correctness

• **Theorem**: TOPOLOGICAL-SORT(G) produces a topological sort of a DAG G

• The TOPOLOGICAL-SORT(G) algorithm does a DFS on the DAG G, and it lists the nodes of G in order of decreasing finish times f[]

• We must show that this list satisfies the topological sort property, namely, that for every edge (u,v) of G, u appears before v in the list

• **Claim**: For every edge (u,v) of G: f[v] < f[u] in DFS
Proof of correctness

“For every edge \((u,v)\) of \(G\), \(f[v] < f[u]\) in this DFS”

• The DFS classifies \((u,v)\) as a **tree edge**, a **forward edge** or a **cross-edge** (it cannot be a back-edge since \(G\) has no cycles):
  
  i. If \((u,v)\) is a **tree** or a **forward edge** \(\Rightarrow v\) is a descendant of \(u\) \(\Rightarrow f[v] < f[u]\)
  
  ii. If \((u,v)\) is a **cross-edge**
Proof of correctness

“For every edge \((u,v)\) of \(G\): \(f[v] < f[u]\) in this DFS”

ii. If \((u,v)\) is a cross-edge:
• as \((u,v)\) is a cross-edge, by definition, neither \(u\) is a descendant of \(v\) nor \(v\) is a descendant of \(u\):
  \[d[u] < f[u] < d[v] < f[v]\]
  or
  \[d[v] < f[v] < d[u] < f[u]\]
  since \((u,v)\) is an edge, \(v\) is surely discovered before \(u\)'s exploration completes

Q.E.D. of Claim
Proof of correctness

• TOPOLOGICAL-SORT(G) lists the nodes of G from highest to lowest finishing times

• By the **Claim**, for every edge \((u,v)\) of G: 
  \[ f[v] < f[u] \]

  \[\Rightarrow\] \(u\) will be before \(v\) in the algorithm's list

• Q.E.D of **Theorem**