# CS6033 Lecture 8 Slides/Notes

**Elementary Graph Algorithms: Topological Sorting & Strongly Connected Components (Notes, Ch 20)**

By Prof. Yi-Jen Chiang
CSE Dept., Tandon School of Engineering
New York University

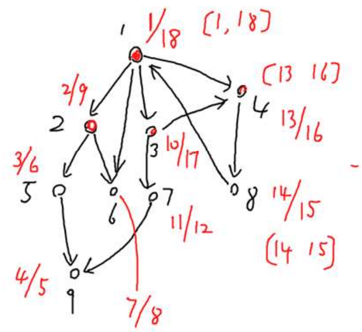
1

## Depth-First Search

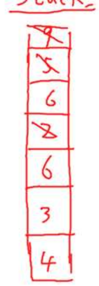
Each vertex is colored:
- white: not yet visited.
- gray: visited but not yet finished
- black: finished.

Use a stack.

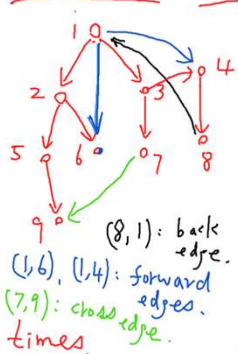
Label each vertex with
- discovery time: time first visited
- finish time: time the vertex is finished.




Stack




DFS-tree: red edges are the tree edges of DFS-tree.




**Lemma**: All intervals are either disjoint or nested.

(disjoint: [ ] [ ]
nested: [ [ ] ])

Pf: $[d_1, f_1]$, $[d_2, f_2]$: $d_1 < d_2$, $f_2 < f_1$, $[d_1 \; [d_2 \; f_2] \; f_1]$, $d_1 < d_2 < f_2 < f_1$

$[d_1\ f_1]$, $[d_2\ f_2]$: $f_1 < d_2$.

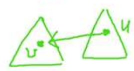
* Each vertex has discovery & finish times, forming an interval [discovery, finish]

2

## Classification of Edges by DFS

u o⟶o v

1. Tree edge: $(u,v) \in$ DFS tree. $v$ is a child of $u$ in DFS tree.
   $v$ is white when $(u,v)$ is explored.
2. Back edge: $v$ is an ancestor of $u$. $v$ is gray when $(u,v)$ is explored.
3. Forward edge: $v$ is a decendent of $u$ but not a child of $u$
   $\Rightarrow$ $v$ is black when $(u,v)$ is explored.
4. Cross edge: $v$ is in a different subtree from $u$. and $v$ is visited before $u$
   $\Rightarrow$ $v$ is black when $(u,v)$ is explored.




Cross edge?

For undirected graphs:

**Claim**: There are only tree edges and back edges

Pf:
Forward edge?


$(v,u)=(u,v)$ is back edge

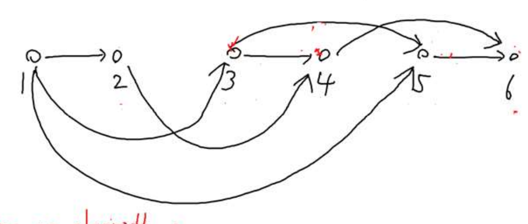
$(v,u)$ is tree edge

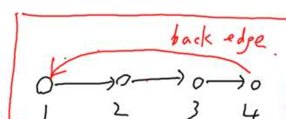
$\Rightarrow$ There is NO forward edge or cross edge.

3

## Topological Sort: Given a directed acyclic graph (DAG) G.

put all vertices into a linear order s.t.
for any edge $(u,v) \in G$, u o⟶o v. $u$ is before $v$ in the linear order.




NOT a DAG (there is a cycle)

* There is No back edge in a DAG.

Deriving an algorithm:

Consider each type of edges in DFS:

① Tree edge: $u$ $[d_u\ f_u]$, $v$ $[d_v\ f_v]$ — $u$ is a parent of $v$. $\boxed{\text{finish time: } f_u > f_v}$

② Forward edge: $u$ $[d_u\ f_u]$, $v$ $[d_v\ f_v]$ — $u$ is an ancestor (non-parent) of $v$. $\boxed{\text{finish time: } f_u > f_v}$

③ Cross edge: $[d_v\ f_v] < [d_u\ f_u]$ $\boxed{\text{finish time: } f_u > f_v}$

④ Back edge: No back edge!

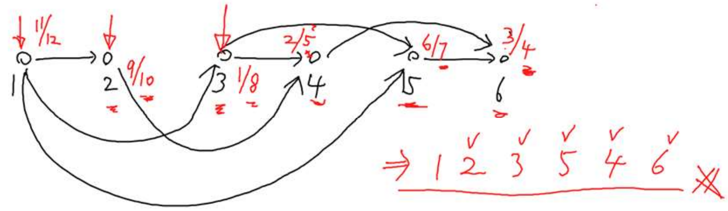
In all cases, $f_u > f_v$

We want: Place $u$ before $v$

4

## Topological Sort

Alg: Perform DFS on G, Put vertices in decreasing order of finish time.




⇒ 1 2 3 5 4 6 ※

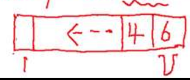
Q: Should we sort by finish time? A: Sorting integers is OK → linear time. but we can do better! (simpler!)

O(V+E) worst-case time

$\boxed{\text{No sorting (simpler):}}$ During DFS, when we finish a vertex u, put u to the front of the current list

1 2 3 5 4 6 ←

* Use an array of size V as the list. Fill up from end to start






## Pf of Correctness: (follows how we derive the alg.)

Consider each type of edges from DFS.

Show: For each edge $(u,v) \in G$, u is put before v in the linear order.

i.e. finish time of u > finish time of v

(u,v) is

① tree edge — u [ $f_u$ ], v [ $f_v$ ]: u is a parent of v in DFS tree. $f_u > f_v$ ✓.

② forward edge — u [ $f_u$ ], v [ $f_v$ ]: u is an ancestor of v. $f_u > f_v$ ✓

③ cross edge →


⇒ $f_u > f_v$ ✓

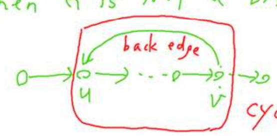
$[d_v\ f_v] < [d_u\ f_u]$ disjoint

④ back edge ×
DAG has No back edge. OK.

Lemma: If G is a DAG then there is No back edge in G.

Pf: Use contrapositive
$(p \Rightarrow q \equiv \underbrace{\sim q \Rightarrow \sim p})$

If there is a back edge in G then G is NOT a DAG.




cycle

G is NOT a DAG.

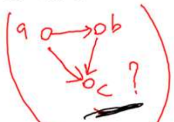
## Strongly Connected Components:

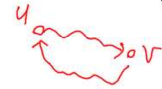
Def: Let $G=(V,E)$ be a directed graph.

vertices $u, v$ are in the same strongly connected component (SCC) $C$

if $u$ can reach $v$ ($\exists$ directed path $u \rightsquigarrow v$)

and $v$ 〃 $u$ ($\exists$ 〃 $v \rightsquigarrow u$)



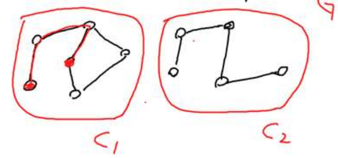




undirected graph:

connected component $C$:

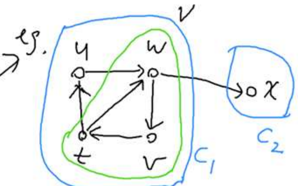
$u, v \in C$ if there is a path $u \rightsquigarrow v$ in the graph

e.g.

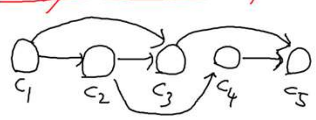



Goal: Given a directed graph $G=(V,E)$ decompose $G$ into maximal strongly connected components

e.g.

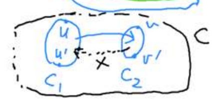



Component Graph is a DAG:



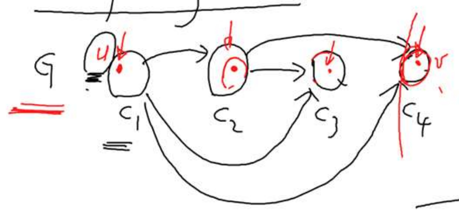

Component Graph

$C_1 \longrightarrow C_2.$




7

## Computing SCCs

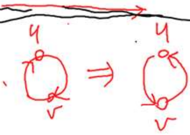


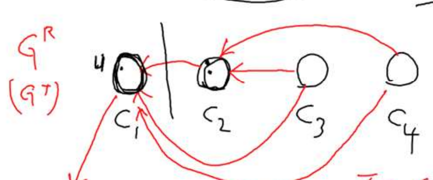

DFS.

$G=(V,E)$

Top. sort:

$C_1\ C_2\ C_3\ C_4$




last strongly CC in $G^R$

($u \in C_1$)

Top. sort

$\longleftarrow$

$C_1\ C_2\ C_3\ C_4$

$\boxed{O(V+E)}$ worst-case time

Reverse graph $G^R$ | Transpose graph $= G^T$

$G^R = (V, E^R)$  $E^R = E^T$

$E^R = \{(u,v) \mid (v,u) \in E\}$

$v \longrightarrow u \in E$

$\Downarrow$

$v \longleftarrow u \in E^R$

$i \quad j$  $A_{ij} = 0 = A_{ji}$

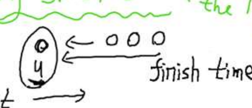
$i \to j \Rightarrow i \leftarrow j$  $A_{ij}=1 \Rightarrow (A^T)_{ji}=1$

Adjacency Matrix $A_{ij}=1 \iff (i,j) \in E.$

$(A)^T_{ij} = A_{ji}$  $(A^T)_{ij}=1 \iff (j,i) \in E.$

Alg: 1. DFS on $G$. Put vertices in the order of decreasing finish time. (first vertex $u$ has the largest finish time)

2. DFS on $G^R$ in the order of step 1.




8