Lecture 10

(1) **Cycle**: Graph (V, E)
a sequence $v_1 \dots v_k$ st $(v_i, v_{i+1}) \in E$ for all $1 \le i < k$ and $(v_k, v_1) \in E$
all v_i should be distinct.

(2) **DAG**: A directed graph $G=(V, E)$ is acyclic if there is no cycle in it.

(3) **Topological ordering**: A mapping $T: V \to [1 \dots n]$ st for each edge $(u,v) \in E$ $T(u) < T(v)$ in a DAG. (in $O(n+m)$ time)

"There exists a Topo ordering for every DAG."

- **Single source shortest paths using Topo**: $O(m+n)$
  Topo ordering as follows:
  source (s) (v) _ _ _ (w) _ _ _ _ (t)
  we know that all vertices reachable from s will come after s only. Now, for all vertices not reachable from s, set weight of edge from $s \to v$ as infinity. Now process each vertex from s to end.

i) v is reachable from s ⇒ all nodes which have edge to v, we have their shortest distance from s. So we can take min of (dist + weight of edge) for each such node to calc for v.

ii) w is not reachable from s ⇒ shortest dist for $w = \infty$. For any node reachable from w & not from s, $\min(\infty, \infty + wt) = \infty$

if reachable from s, → val remains same.

- Single Source Longest Paths using Topo: $O(m+n)$
same, just initially all as $-\infty$.
Now, for reachable ones, take max of all possible paths, and unreachable ones:
$\max(-\infty, -\infty + wt) = -\infty$ so no change.

- Count no. of paths from s to d: $O(m+n)$
initialise all $ct(u) \leftarrow 0$. ; $ct(s) = 1$
for each vertex $u \in$ Topo(s) {
for each edge (u, w) {
$ct(w) += ct(u)$; }}

- Every DAG has at least one vertex with indegree $= 0$. Similarly atleast one vertex with outdegree $= 0$.

→ Algo for Topo: $O(m+n)$ ← Create & Fill indeg array.

```
Topo-order (G){
  Create Empty Queue (Q);
  For each u ∈ V { if (indeg[u] = 0)
                     enqueue (u); }

  toponum = 1;
  While (!Q.Empty){
    v ← dequeue (Q);
    topo[v] = toponum++;
    for each edge (v, w) {
        indeg[w]--;
        if (indeg[w] == 0) enqueue (w); }
```

→ General Use of Topo:

If f is a func on V, st $f(y)$ can be computed in terms of only $\{f(x) \mid (x,y) \in E\}$ then f can be computed by processing vertices in inc order of τ.

1 2 3 4 5 6 7 8 9

If $f(y)$ be expressed in terms of only $\{f(z) \mid (y,z) \in E\}$. Then compute f by processing in dec order of τ.

1 2 3 4 5 6 7 8 9

Lecture 11

→ Unique Path Graph

A directed graph $G=(V,E)$ if for each pair $u, v \in V$, there is atmost one path from u to v.

Alternate Topo Sort using DFS.

Strongly connected Components of a graph:

Maximal subset of vertices st $\forall (u,v)$ there is path from u to v & v to u.

Properties of DFS:

① If DFS(x) invokes DFS(y) $\Rightarrow$ after DFS(y) finish, control returns to DFS(x)

(2) If there is no path from x to y, then DFS starting at x will not visit y.

(3) Let U be the set of vertices visited before s. Then DFS(s) is identical to a fresh DFS starting from s in graph $G \setminus U$.

(4) DFS(u) visits all vertices reachable from u.

(5) DFS Tree is not unique for a given graph.

(6) Every Tree is not a DFS Tree

(7) DFS partitions the graph into a forest of DFS Trees.

→ Reachability: see all reachable vertices for each $u \in V$.

→ Discovery time and finish time:

```
DFS(v){
  vis[v] ← 1;   D[v] ← count++;
  for each edge (v, w) {
    if (vis(w) == 0) {
      DFS(w);
    }
  F[v] ← count++;
}
```

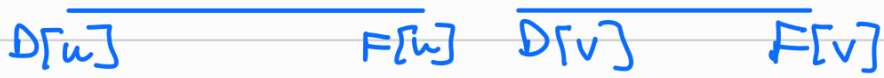
D[v] ← discovery time, DFN number of v: when DFS starts at v

F[v] → Finish number of v ⇒ when DFS at v ends

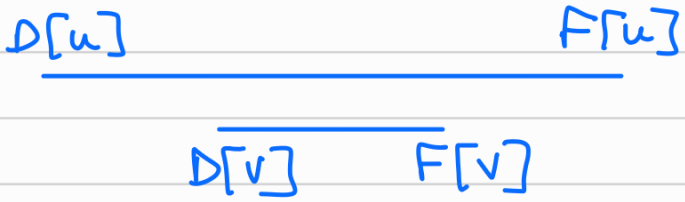
```
DFS-graph(G = (V, E)) {
  for each v∈V  vis[v] = 0;
  ct = 1;
  for each v∈V { if (vis[v] = 0) DFS(v)}
```

→ DFS Numbering property

① Disjoint



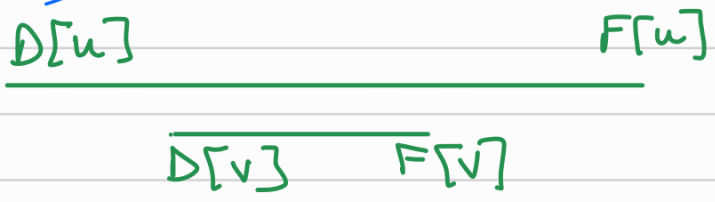
② One enclosing the other



→ Classification of non-tree edges

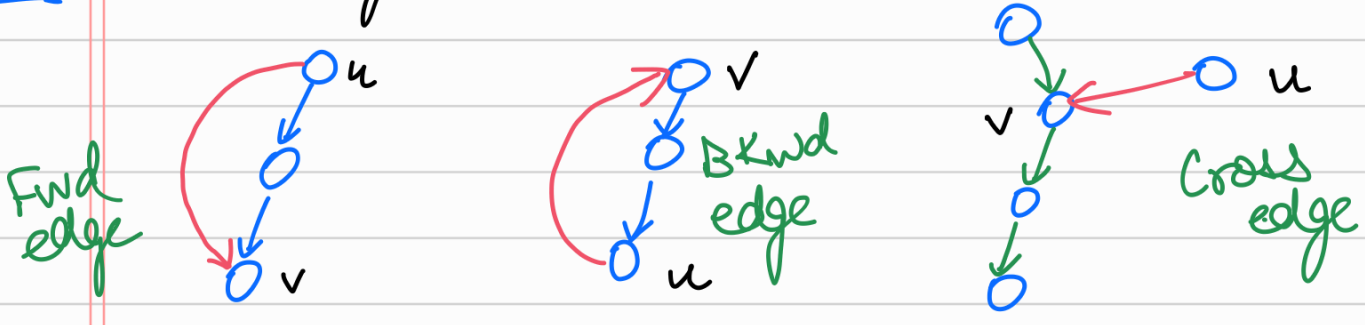
Consider an edge (u, v)

Case 1 Tree / Forward edge: ancestor to descendant



Case 2 Backward edge: descendant to ancestor

Case 3 Cross edge



→ Order (mn) Algo

2MST ≥ MST

For each vertex $u \in V$ {
  DFS(u);
  if any forward or cross edge appears in the DFS tree(u) ⇒ u has multiple paths to some vertex. ✗

One backward edge ⇒ no problem for u

any backward edge ⇒ not possible for u.)

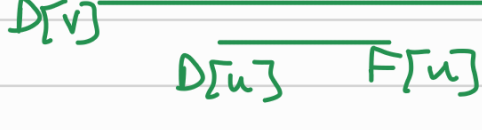
→ Algo for Unique Paths from any vertex u:

$O(n^2)$
for every vertex, we visit almost $O(n)$ vertices: $O(n)$ tree edges, $O(n)$ back edges for all
⇒ $O(n^2)$

```
DFS(u) {
  vis[u] = 1 ; Backedges = 0;
  DFN[u] ← cont++;
  for every edge (u, w) {
    if (vis[w] == 0) DFS(w);
    else { if (finished(w) == 1) ret false;
      else { backedges[u]++;
        if (backedges[u] > 1) ret false; }
  finished[u] ← 1;
  F[u] ← cont++; }
```

→ Alt Algo for Topo Sort

In any DAG, we cannot have any cycle.
Cycle → Backward non-tree edges. ✗



(u → v) edge
if backward
⇒ F[v] > F[u] ✗
→ for all edges (u → v) F[u] > F[v]

Vertices arranged in decreasing order of their finish time during DFS is a valid Toposort.

Algo: "O(n+m) algo"

Apply DFS graph traversal. Now, as any vertex : finished → true, push it into the stack. Topo order is the stack top → bottom.

Lecture 13

→ **Strongly Connected Components:**
Two vertices u and v are said to be strongly connected if $u \rightsquigarrow v$ and $v \rightsquigarrow u$.

Strongly conn comp: a maximal subset of "strongly conn. vertices"

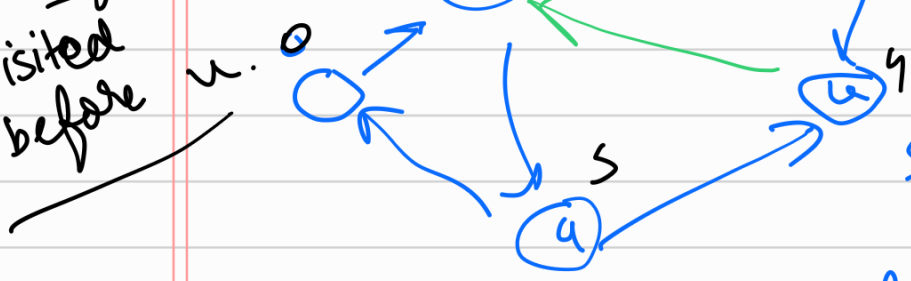
- An SCC will appear in exactly one tree in the forest of DFS trees!
(Kisi ek hi tree me aayega, thoda isme thoda usme nai ho skta.

- An SCC will appear intact within its DFS tree also.
Kisi SCC ke all vertices form a tree within the DFS tree. ek se dusre vertex pe jana ho to saari SCC ki hi vertices aayengi, koi non SCC nai aa skti beech me.

- Let x and y belong to the same SCC in a graph. Let P be any path from x to y in the graph. Then each vertex of P also belongs to the same SCC as of x and y.

Notation (root of an SCC):
The vertex of the SCC which is visited first during the DFS, in a given DFS tree.

Case 1: u, v ∈ same SCC



$F[u] > F[v]$

and $F[u] < F[v]$ st=4

also possible. both possible

If u before v : $F[u] > F[v]$

Case2: u, v ∈ diff SCC $(u \to v)$

i) v before u: since no path $(v \to u)$ this means DFS[v] ends then DFS(u) starts.

$D[v] < F[v] < D[u] < F[u]$

ii) u before v: v will be a descendant of u in DFS Tree. $D[u] < D[v] < F[v] < F[u]$

$\Rightarrow$ both $F[v] < F[u]$

Lemma: If (u, v) is an edge and u & v belong to diff SCCs, then $F[u] > F[v]$.

$$F[\text{root}(S_2)] > F[\text{root}(S_1)]$$

Proof: There is an edge $u \to v$. $u \in S_2$ $v \in S_1$

i) If root(u) vis before root(v)

$\Rightarrow u \to v \Rightarrow v$ in its dfs trav.

Now we claim that $v = \text{root}(S_2)$, since now from here, we would visit all vertices in S_2.

$\Rightarrow F[\text{root}(u)] > F[u] > F[v] = F[\text{root}(v)]$

ii) root(v) before root(u) $\Rightarrow$

dfs trav ends in S_2 and then begins in u.

So $F[\text{root}(S_1)] < F[\text{root}(S_2)]$

Theorem: If $F[root(S_2)] > F[root(S_1)]$ then there can be edge from a vertex u in S_2 to a vertex v in S_1 but not vice versa.

$u(S_2) \longrightarrow v(S_1)$ $\quad F[S_2] > F[S_1]$

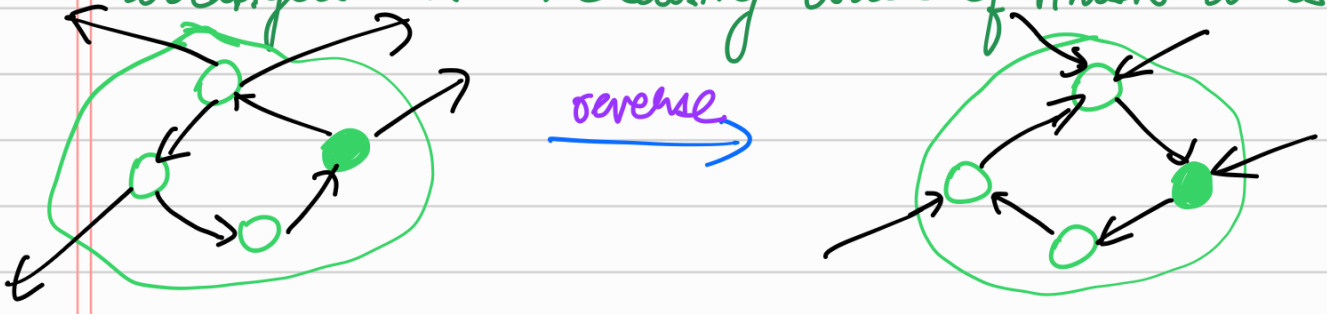
But no edge from S_1 to S_2 vertex.

Consider an increasing order of F_i's:

So for the last vertex in a topo sort, it must be the root since it will have the largest finish time for that component.

Since green > red ⇒ no edge from red component to green component.

Now reverse all the edges in the graph and process vertices from the end of the array arranged in increasing order of Finish times.



- Strongly connected vertices remain Str conn.
- There are only incoming edges, NO outgoing edges from SCC(green)

So we find SCC (green) from its root, in right → left order of F's array

→ **Algo**

i) DFS (G) → compute Fin[ ] of all $v \in V$.

ii) L → vertices in dec order of Fin[ ]

iii) G^r → graph G ko reverse edge directions.
iv) SCC-found = 0 ∀ v ∈ V.
v) num ← 1.

```
for each v in L {
    if (SCC-found[v] == 0){
        do BFS/DFS from v on G^r;
        A ← set of reachable vertices;
        For all x ∈ A {
            SCC-found[x] = 1;
            SCC-num[x] = num;
            remove x from G^r along with its edges.
        }} num++; }
```

SCC Graph

Collapsing all SCC components into one single vertex. Such a graph is always a DAG.

For a given directed graph G, transform each SCC into a single vertex. Remove multiple edges.

If some function $f: V \to R$ st
f takes the same value on all vertices of a SCC, it suffices to compute f on SCC graph.