Depth-First Search (DFS)

Depth-First Search (DFS)

```
DFS_VISIT(G, u)
    time t = 1
    u.d = time          // d — discover
    u.color = GRAY
    for v in Adj[u]:
        if v.color = WHITE:
            v.parent = u
            DFS_VISIT(G, v)
    u.color = BLACK
    time += 1
    u.f = time          // f — finish
```

Unlike BFS, DFS is not guaranteed to find the shortest path.

We usually use DFS to explore the whole graph, as opposed to what is reachable from a given vertex.

```
DFS(G):
    time = 0
    for u ∈ V:
        u.color = WHITE
        u.parent = NIL
    for u ∈ V:
        if u.color == WHITE:
            DFS-VISIT(G, u)
```

Runtime:

1. $\Theta(|v|)$ in DFS (not counting DFS-VISIT calls).
2. We only call DFS-visit on each vertex once (when it's white and first discovered), spending $\Theta(1) + \Theta(|Adj[v]|)$ time.
3. In total, runtime is $\Theta(\Sigma_{v \in V}(1 + |Adj[v]|)) = \Theta(|V| + |E|)$, same as BFS.

Edge classification

Tree edge: edge used to visit a new vertex

Forward edge: vertex → descendent in tree

Backward edge: vertex → ancestor in tree

Cross edge: between non-ancestor related vertexes (basically everything else)

How can we classify edge (u, v) in DFS?

Tree edge: v is **white** when we when we explore (u, v)

Forward edge: v is **black** when we explore (u, v), **v.d > u.d**

Backward edge: v is **gray** when we explore (u, v)

Cross edge: v is **black** when we explore (u, v), **v.d. < u.d** (also v.f < u.d)

To distinguish forward from cross edges, we add “discovery time” and “finish time” to each vertex.

Undirected Graphs

No forward or cross edges.

In other words, when we first explore an edge $(u, v) \in E$, v cannot be black.

Claim: An undirected graph is acyclic iff there are no back edges.

Proof:

$\leftarrow$ If no back edges, then only tree edges, so acyclic.

$\rightarrow$ If there is back edge, then it closes a cycle.

Back to directed graphs...

White Path Theorem: v is a descendent of u in the DFS forest iff at time $u.d$ there exists a path from u to v with only white vertices.

Proof idea:

$\rightarrow$ Let $u = u_0, u_1, u_2, ..., u_k = v$ be the path in the DFS forest from u to v.

Then u_1 was discovered from u_0, u_2 was discovered from u_1, etc.

Together, $u_k.d > u_{k-1}.d > ... > u_0.d$ and so $u_1, ..., u_k$ were all white at time $u_0.d$.

$\leftarrow$ u_1 is finished before u_0 is, u_2 is finished before u_1, etc.

$u_k.f < u_{k-1}.f < ... < u_0.f$

Therefore, v is detected during the exploration from u, and must have a path from u in the DFS forest.

Lemma: A directed graph is acyclic iff DFS finds no back edges.

A directed graph has cycles iff DFS finds back edges.

Topological Sort

In job scheduling, there is a set of tasks to perform where some tasks must be completed before others. The input is represented as a directed acyclic graph (DAG).

Algorithm:

Input: DAG G = (V, E)

1. Run DFS
2. Output vertices in decreasing order of finish time

Runtime is $O(|V| + |E|)$. In particular, no needs to sort by finish time; we just add vertices to front of linked list as they are finished.

Theorem: The algorithm outputs a topological sort.

Proof: It suffices to show that for all $(u, v) \in E$, $v.f < u.f$. When we explore (u, v) in the DFS, v cannot be gray by Lemma above. Therefore, either black or white.

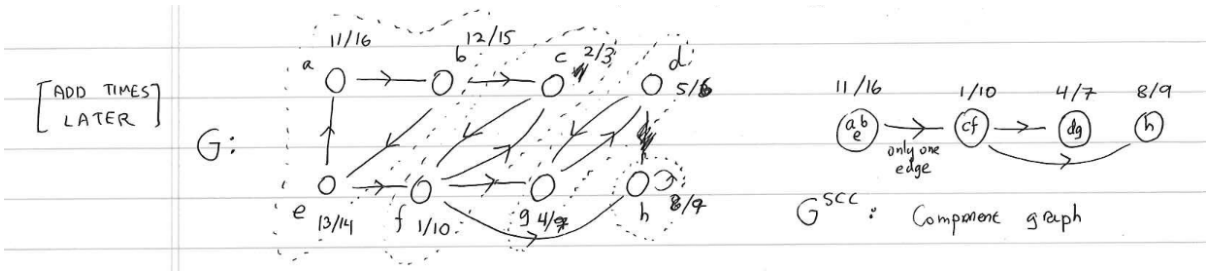
- If black, v.f is already set, but we are still exploring u (it is gray), so u.f is not set yet. We have $v.f < u.f$.

- If white, we will recursively visit v. When recursive call returns, v.f is set, but u.f is not. So again $v.f < u.f$.

If see DAG, think topological sort!

Strongly Connected Components (SCC)

If see a directed graph, think SCC!



Consider DFS on G.

We can pretend that this DFS takes place in G^{SCC}.

- A component is discovered when one of its vertices is discovered.
- A component is finished when all its vertices are finished.
- A component is initially white, then gray, finally black.

Key Lemma: The “virtual walk” on G^{SCC} induced by DFS on G is a valid DFS exploration on G^{SCC}.

As a result, if there is an edge from C to C’ in G^{SCC} (which is a DAC), then $C.f > C'.f$, just like we saw in the topological sort.

Therefore, the component with largest finishing time is first in the topolocal sort of G^{SCC}.

The last vertex to finish the DFS on G must be in the first component of G^{SCC}.

We can identify the entire first component by running DFS-VISIT from that vertex on the transposed G^T. (Reason: can still reach any vertex inside component, but can never leave.)

SCC(G):

1. Call DFS(G) to compute u.f for each $u \in V$.
2. Call DFS(G^T) where in the main outer loop consider in order of decreasing finish time.
3. Output the vertices of each tree as a seperate SCC.

Runtime: $O(|V| + |E|)$.