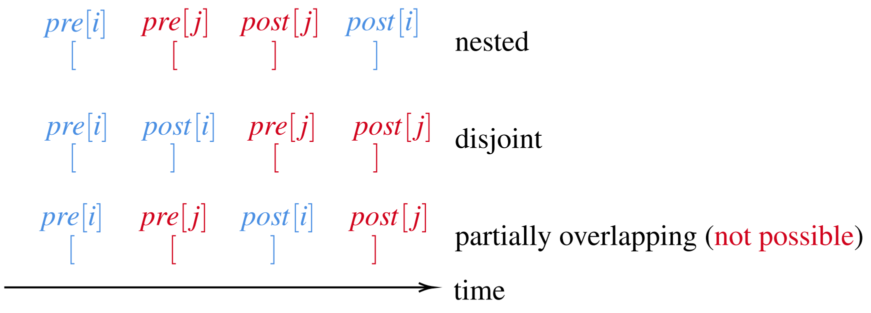
DFS with Timing and Properties

Recall that the DFS-with-timing algorithm gives an interval $[pre, post]$ for each vertex. For two vertices $v_i, v_j \in V$, their corresponding intervals can either be *disjoint*, i.e., the two intervals do not overlap, or *nested*, i.e., one interval is within the other. See Figure 1. But two intervals cannot be *partially overlapping*. Why? This is because the explore funtion is recursive. There are only two possiblities that $pre[i] < pre[j]$. The first one is that explore v_j is *within* explore v_i; in this case the recursive behaviour of explore leads to that $post[j] < post[i]$, as explore v_j must return/terminate first and then explore v_i will return/terminate. This case corresponds to that the two intervals are nested. The second one is that explore v_j starts after explore v_i finishes; this case corresponds to that the two intervals are disjoint.



pre[i] pre[j] post[j] post[i]
nested
pre[i] post[i] pre[j] post[j]
disjoint
pre[i] pre[j] post[i] post[j]
partially overlapping (not possible)
time

Figure 1: Relations between two $[pre, post]$ intervals.

We now show and prove a key claim that relates the post values and the meta-graph.

Claim 1. Let C_i and C_j be two connected components of directed graph $G = (V,E)$, i.e., C_i and C_j are two vertices in its coresponding meta-graph $G_M = (V_M, E_M)$. If we have $(C_i, C_j) \in E_M$ then we must have that $\max_{u \in C_i} post[u] > \max_{v \in C_j} post[v]$.

Intuitively, following an edge in the meta-graph, the largest post value decreases. Before seeing a formal proof, please see an example in Figure 2: the largest post values for C_1, C_2, C_3, and C_4 are 9, 6, 10, and 14, and you may verify that following any edge in the meta-graph, the largest post value always decreases.

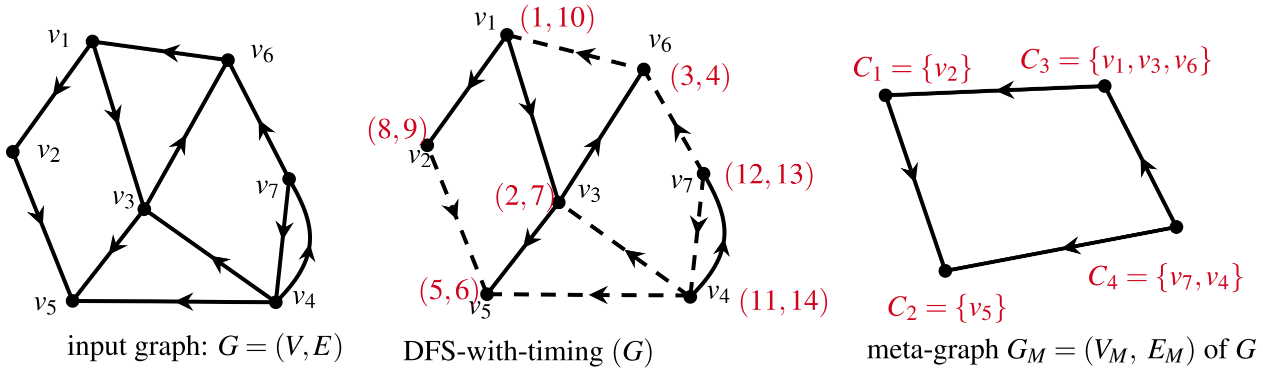


Figure 2: Example of running DFS (with timing) on a directed graph. The $[pre, post]$ interval for each vertex is marked next to each vertex. The $postlist$ for this run is $postlist = (v_4, v_7, v_1, v_2, v_3, v_5, v_6)$.

Proof. Let $u^* := \arg\min_{u \in C_i \cup C_j} pre[u]$, i.e., u^* is the first explored vertex in $C_i \cup C_j$. Consider the two cases.

First, assume that $u^* \in C_i$. Then u^* can reach all vertices in $C_i \cup C_j \setminus \{u^*\}$. Hence, all vertices in $C_i \cup C_j \setminus$

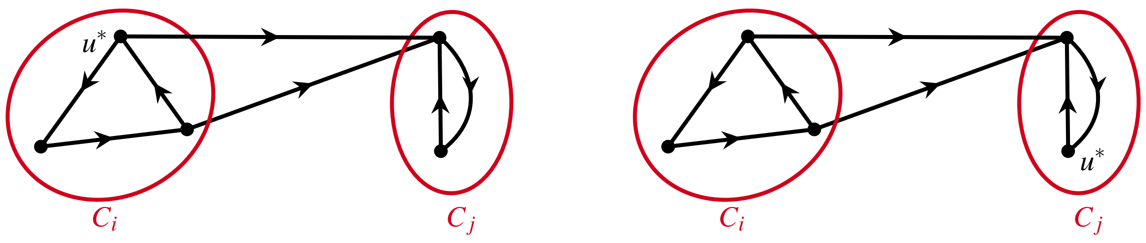


Figure 3: Two cases in proving above claim.

$\{u^*\}$ will be explored *within* exploring u^*. In other words, for any vertex $v \in C_i \cup C_j \setminus \{u^*\}$, the interval $[pre[v], post[v]]$ is a subset of $[pre[u^*], post[u^*]]$. This results in two facts: $\max_{u \in C_i} post[u] = post[u^*]$ and $\max_{v \in C_j} post[v] < post[u^*]$. Combined, we have that $\max_{u \in C_i} post[u] > \max_{v \in C_j} post[v]$.

Second, assume that $u^* \in C_j$. Then u^* can *not* reach any vertex in C_i; otherwise $C_i \cup C_j$ form a single connected component, conflicting to the fact that any connected component must be maximal. Hence, all vertices in C_i will remain unexplored after exploring u^*. In other words, for any vertex $v \in C_i$, the interval $[pre[v], post[v]]$ locates after (and disjoint with) $[pre[u^*], post[u^*]]$. This gives that $\max_{u \in C_i} post[u] > post[u^*]$. Besides, we also have $\max_{v \in C_j} post[v] = post[u^*]$ as all vertices in $C_j \setminus \{u^*\}$ will be examined within exploring u^*. Combined, we have that $\max_{u \in C_i} post[u] > \max_{v \in C_j} post[v]$. □

The above key claim directly suggest a property about the structure of the meta-graph, given below.

Corollary 1. The list of all connected components in V_M sorted in decreasing order of $\max_{u \in C_i} post[u]$, $1 \le i \le k$, is a linearization of G_M.

Verify this is the case in Figure 2. Answer: such list is (C_4, C_3, C_1, C_2), and it is a linearization of G_M.

Above property also suggests us to consider an *ordering of vertices* in descending post values. Specifically, we define *postlist* as such ordering of vertices. Notice that postlist can be stored in an array and can be efficiently constructed in DFS-with-time. Please refer to Lecture A10 for its pseudo-code. See Figure 2 for an example.

As a direct consequence of Corollary 1 and the definition of postlist, we have the following property. This is true as "the first appearance" exactly gives the "largest post value" since the postlist is sorted in descending order of post values.

Corollary 2. The list of all connected components in V_M sorted by their first appearance of in the postlist is a linearization of G_M.

Verify this is the case in Figure 2. Answer: following *postlist* the corresponding list of connected components is $(C_4, C_4, C_3, C_1, C_3, C_2, C_3)$. The first appearance of each component is (C_4, C_3, C_1, C_2), which is a linearization of G_M and exactly the one that is sorted by their largest post value.

To summarize, now we can build a postlist that satisfies above Corollary 2. This almost achieves the desired property of the special ordering: "the ordering of connected components sorted by their first appearance in the special ordering should form a reverse-linearization of the meta-graph." The only difference is that postlist implies a *linearization* of the meta-graph while special ordering requires a *reverse-linearization* of the meta-graph. Below, we introduce *reverse graph* to fill this gap.

Reverse Graph

Definition 1. Let $G = (V,E)$ be a directed graph. The *reverse graph* of G, denoted as $G_R = (V,E_R)$, has the same set of vertices and edges with reversed direction, i.e., $(u,v) \in E$ if and only if $(v,u) \in E_R$.

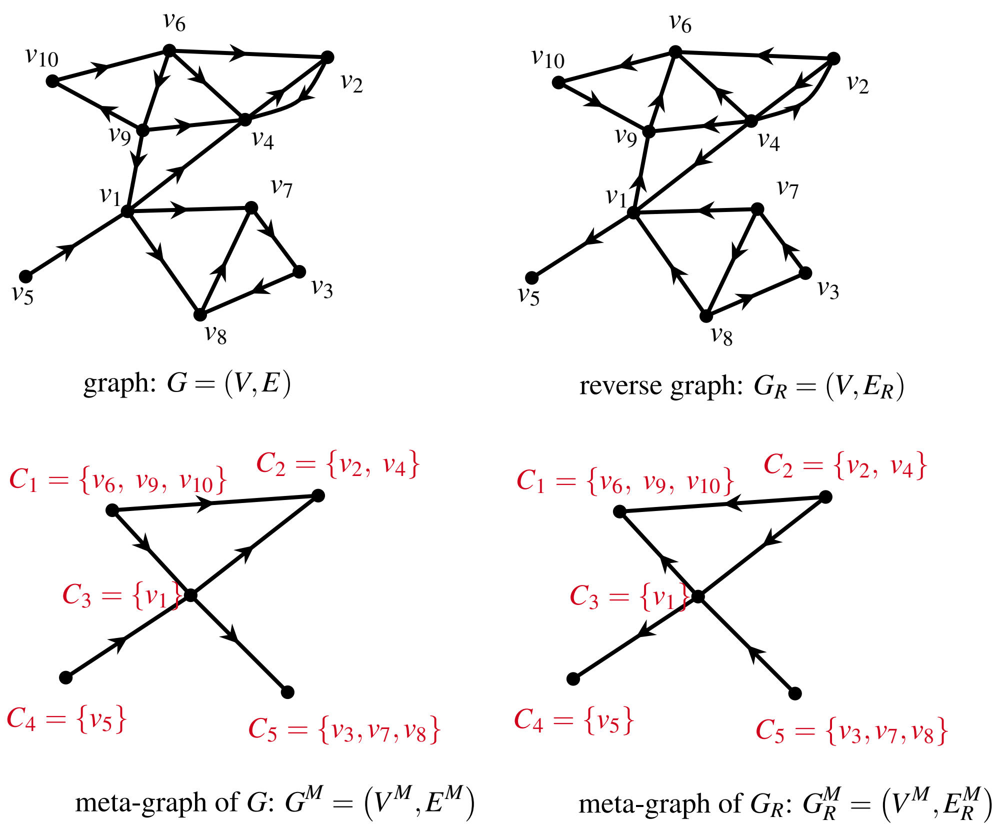

v_{10}
v_6
v_2
v_9
v_4
v_1
v_7
v_5
v_3
v_8
graph: $G = (V,E)$
reverse graph: $G_R = (V,E_R)$
$C_1 = \{v_6,\ v_9,\ v_{10}\}$
$C_2 = \{v_2,\ v_4\}$
$C_3 = \{v_1\}$
$C_4 = \{v_5\}$
$C_5 = \{v_3, v_7, v_8\}$
meta-graph of G: $G^M = \left(V^M, E^M\right)$
meta-graph of G_R: $G_R^M = \left(V^M, E_R^M\right)$

Figure 4: Graph and its reverse graph, and the corresponding meta-graphs.

Following properties can be easily proved using above definition.

Property 1. There is a path from u to v in G if and only if there is a path from v to u in G_R. In other words, u can reach v in G if and only if u can be reached from v in G_R.

Property 2. $(G_R)_R = G$.

Property 3. A vertex v of DAG G is a source vertex if and only if v is a sink vertex of G_R.

Property 4. G and G_R has the same set of connected components.

Property 5. The meta-graph of G_R is the reverse graph of the meta-graph of G. Formally, $(G_R)^M = (G^M)_R$.

Property 6. X is a linearization of DAG G if and only if the reverse of X is a linearization of G_R (or, X is a reverse-linearization of G_R). In particular, since any meta-graph is a DAG, we have that X is linearization of G^M if and only if X is a reverse-linearization of $(G^M)_R = (G_R)^M$, and that X is a linearization of $(G_R)^M = (G^M)_R$ if and only if X is a reverse-linearization of G^M.

Algorithm to Determine Connected Components in Directed Graphs

Combining above DFS-with-timing and reverse graph, here is the pseudo-code we use to obtain a special ordering. The key is that we run DFS-with-timing on the reverse graph G_R, instead of on the given graph G.

```
Algorithm to determine the specific order
    build $G_R$ of $G$;
    run DFS with timing on $G_R$ to get $postlist$;
    return $postlist$;
end algorithm;
```

We now show that, above *postlist*, i.e., the *postlist* obtained by running DFS-with-timing on G_R, satisfies the condition that "the ordering of connected components sorted by their first appearance in above *postlist* forms a reverse-linearization of the meta-graph G_M". In fact, since above *postlist* is obtained by running DFS-with-timing on G_R, according to Corollary 2, the list of all connected components sorted by their first appearance of in above *postlist* is a linearization of $(G_R)^M$. According to Property 6 of reverse graph, we know that a linearization of $(G_R)^M$ is a reverse-linearization of G_M.

Above *postlist* can then be used by the DFS algorithm as a special order to obtain all connected components, with pseudo-code copied below.

```
function DFS $(G = (V,E))$
    num-cc = 0;
    for $v_i$ in the $postlist$ obtained above
        if $(visited[i] = 0)$
            num-cc = num-cc + 1;
            explore $(G, v_j)$;
        end if;
    end for;
end algorithm;
```

```
function explore $(G = (V,E), v_i \in V)$
    $visited[i] =$ num-cc;
    for any edge $(v_i, v_j) \in E$
        if $(visited[j] = 0)$: explore $(G, v_j)$;
    end for;
end algorithm;
```

The entire algorithm runs in $\Theta(|V| + |E|)$ time.