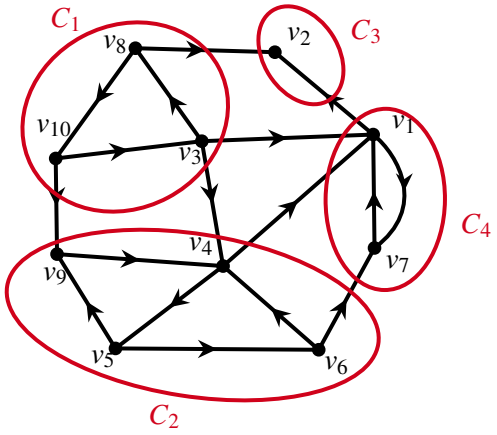
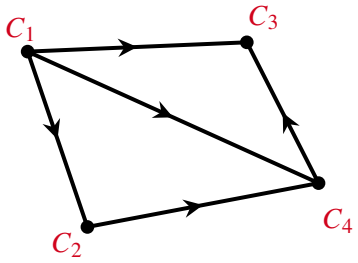
Finding Connected Components of Directed Graphs

We now design algorithms to determine the connected components of directed graphs (i.e., the vertices of its meta-graph). Recall that the DFS algorithm introduced in Lecture 08 can successfully determine all connected components in an undirected graph. Does this algorithm also work for directed graph? And if not, where is the problem? Let's run it on an example to find a clue. See Figure 1. Recall that the (top-layer) of the DFS algorithm is to tranverse all vertices in some ordering, and if the current vertex v is not yet visited, we will explore v. For the example in Figure 1, we run it with the (natrual) ordering $v_1, v_2, \cdots, v_{10}$.





final array of *visited* after running DFS with ordering $(v_1, v_2, \cdots, v_{10})$:

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
1	1	2	2	2	2	1	2	2	2

final array of *visited* after running DFS with a special ordering $(v_2, v_7, v_4, v_6, v_8, v_1, v_3, v_9, v_{10}, v_5)$:

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
2	1	4	3	3	3	2	4	3	4

Figure 1: Run the DFS algorithm with an natural (arbitrary) order and a special order (pseudo-code given below) on above example.

The resulting visited array is given in the figure (please make sure you try it yourself). Unfortunately, it does not give all connected components. In fact, the vertices marked as "1"s are the union of C_3 and C_4, and the vertices marked as "2"s are the union of C_1 and C_2. The reason is quite clear: we start with explore v_1, and during it all vertices reachable from v_1 will be marked as "1" in the visited array. It turns out that v_1 is in connected component C_4, and C_4 can reach C_3 in the meta-graph. Therefore, all vertices in C_4 and C_3 are reachable from v_1; consequently, the visited array is set as "1" for vertices in C_4 and C_3 during explore v_1. Similarly, we can also see why the vertices marked as "2"s are the union of C_1 and C_2. After exploring v_1, $\{v_1, v_2, v_7\}$ are visited; the next unvisited vertex is v_3 according to above natural ordering, then the algorithm will explore v_3. Again, during it all (unvisited) vertices reachable from v_3 will be marked as "2" in the visited array. It turns out that v_3 is in connected component C_1, and C_1 can reach C_2 in the meta-graph. Therefore, all vertices in C_1 and C_2 are reachable from v_2; consequently, the visited array is set as "2" for vertices in C_1 and C_2.

How to fix this issue? The idea is to use a *special ordering* of vertices, instead of an arbitrary ordering, in above DFS. This special ordering should allow us to determine connected component one by one. Hence, the first vertex we explore in DFS, should be one vertex in a *sink* component of the meta-graph. In Figure 1,

the only sink component is C_3, and since there is only one vertex in C_3, we should start with exploring v_2. Clearly, exploring v_7 will exactly determine the single connected component C_2. Next, after C_3 is determined, the next component we can determine is again a sink component after removing C_3 from the meta-graph. It is C_4. So, the next vertex the DFS should explore must be v_1 or v_7. It does matter which one we pick; let's assume we pick v_7 (and it does not matter where v_1 is in the special ordering as long as it is after v_2 and v_7). Clearly, exploring v_2 will exactly determine the single connected component C_4—see the visited array. The next component we can determine is again a sink component after removing C_3 and C_4 from the meta-graph. It is C_2. So, the next vertex the DFS should explore must be in $\{v_4, v_5, v_6, v_9\}$. And it does matter which one we pick neither the ordering of remaining ones as long as they are behind the one we pick. Let's say we pick v_4. Clearly, exploring v_4 will exactly determine the single connected component C_2. Now the only component remaining is C_1 after removing C_2, C_3 and C_4 from the meta-graph. So, the next vertex the DFS should explore must be in $\{v_3, v_8, v_{10}\}$. Let's say we pick v_8. Clearly, exploring v_8 will exactly determine the last connected component C_1. In Figure 1, we give such a special order that follows above analysis. You should try it—run DFS following this special order correctly identifies all connected components, as shown in the Figure.

Let's summarize above observation/analysis, aiming for characterizing/constructing the special order. First, it should be clear now that, the identified connected components, following the special order, forms a reverse-linearization of the meta-graph. (Verify this with the Figure: the revealed connected components is (C_3, C_4, C_2, C_1) which is a reverse-linearization of the meta-graph.) Second, the first appearance of a vertex in a connectec component matters, as exploring it identifies the entire connected component. Combined, the special ordering of vertices should satisfy this condition: *the first appearance of connected components in the special ordering should form a reverse-linearization of the meta-graph*. Again see Figure 1, the ordering $(v_2, v_7, v_4, v_6, v_8, v_1, v_3, v_9, v_{10}, v_5)$ satisfies above condition. To see that, the corresponding list of connected components is $(C_3, C_4, C_2, C_2, C_1, C_4, C_1, C_2, C_1, C_2)$, and hence the ordering of their first appearance is (C_3, C_4, C_2, C_1) which is indeed a reviers-linearization of the meta-graph.

If the special ordering satisfies this condition, the DFS algorithm will work—it will identify all connected components. See the pseudo-code below. It is the same with that for undirected graphs except a single line of difference (marked blue).

```
function DFS (G = (V,E)) it identifies connected components for directed graphs
    num-cc = 0;
    for v_i in a specific order
        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 ∈ V)
    visited[i] = num-cc;
    for any edge (v_i, v_j) ∈ E
        if (visited[j] = 0): explore (G, v_j);
    end for;
end algorithm;
```

The remaining question is therefore how to find an ordering of vertices that satisfy above condition. Recall the key property we proved in Lecture 10: for any two connected components C_i and C_j, if $(C_i, C_j) \in E_M$ then $\max_{u \in C_i} post[u] > \max_{v \in C_j} post[v]$. Consider the postlist, i.e., the list of vertices in descending order of post-values. The *first* appearance of (a vertex in) connected component C_i in the postlist is the vertex with *largest* post-value in C_i, simply because postlist sorts vertices in descending order of post-values. According to the key property, if $(C_i, C_j) \in E_M$, then the first appearance of C_i is before the first appearance of C_j. This implies the following fact.

Claim 1. Let G be a directed graph. After running DFS-with-timing on G, the first appearance of connected components in the resulting postlist forms a linearization of G_M.

We are close. The desired special order requires "the first appearance of connected components in the special ordering form a reverse-linearization of the meta-graph", while the postlist yields "the first appearance of connected components in the postlist form a linearization of the meta-graph", How to close this last gap? The answer is the reverse-graph! We can build the reverse graph G_R of the given directed graph G, and then run DFS-with-timing on G_R to get a postlist. This postlist will be the desired, magic special order! See the pseudo-code below.

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

We now show that, above $postlist_R$, i.e., the one obtained by running DFS-with-timing on G_R, satisfies the condition that "the first appearance of connected components in this $postlist_R$ forms a reverse-linearization of the meta-graph G_M". In fact, since this $postlist_R$ is obtained by running DFS-with-timing on G_R, applying Claim **??**, we know that the first appearance of connected components in $postlist_R$ forms a linearization of meta-graph of G_R, which is $(G_R)_M$. According to the properties of reverse graph, we know that $(G_R)_M = (G_M)_R$. We also know that a linearization of $(G_M)_R$ is a reverse-linearization of G_M. Hence, we have that, the first appearance of connected components in $postlist_R$ forms a reverse-linearization G_M, proving that $postlist_R$ is the desired special order.