WORKSHEET, WEEK08: GRAPH TRAVERSALS

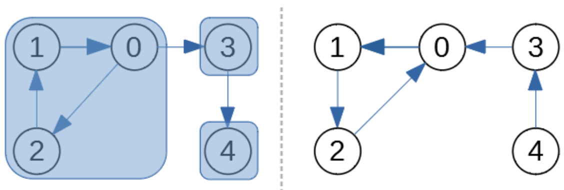
8.1 Strongly Connected Components

One of the multiple practical applications of a DFS traversal of a directed graph is finding strongly connected components (strongly connected graphs are defined in (Goodrich2011, p.626)), the relevant algorithm is known as Kosaraju's algorithm. https://bit.ly/3lI20ec, https://bit.ly/3mNU2la.

Definition: A subset of vertices in a directed graph $S \subseteq G.V$ makes a strongly connected component, iff for any two distinct vertices u, v there is a path $u \rightsquigarrow v$ (one or more and also another path $v \rightsquigarrow u$ that goes back from v to u.

If you can travel only in one direction (say, from u to v), but cannot return, then u, v should be in different strongly connected components. (Same thing, if u and v are mutually unreachable.) Moreover, every vertex is strongly connected to itself – so even in the worst case a graph with n vertices would have at most n strongly connected components (containing one vertex each).

Figure shows an example of a graph with $n = 5$ vertices having 3 strongly connected components. Next to that graph is the *transposed graph* G^T where all the edges are reversed.



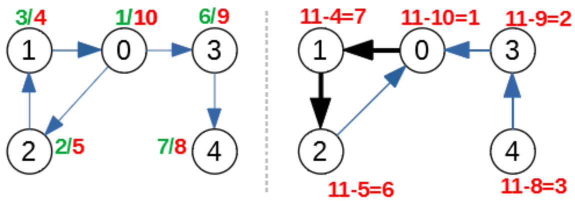
8.2 Kosaraju's algorithm

There is a way to find strongly connected components in an arbitrary graph by running DFS twice (i.e. it works in linear time $O(n + m)$).

STRONGLY_CONNECTED(G)
(compute all finishing times $u.f$)
call DFS(G)
(G^T is transposed G, all edges reversed)
compute G^T
(visit vertices in decreasing $u.f$ order)
call DFS(G^T)
for each tree T in the forest DFS(G^T)
    Output T as a component

To see how this works, we can run it on the example graph shown earlier. After the DFS on graph G is run, we get the finishing times for the vertices $0, 1, 2, 3, 4$ (all shown in red on the left side of Figure below). After that we replace G

by G^T (to the right side of the same figure), and assign priorities in the decreasing sequence of $u.f$ (the finishing times when running DFS(G)).



To make this reverse order obvious, we assign new priorities to the vertices in G^T. The new priorities in G^T are the following:

- Vertex `0` has priority $11 - 10 = 1$.
- Vertex `1` has priority $11 - 4 = 7$.
- Vertex `2` has priority $11 - 5 = 6$.
- Vertex `3` has priority $11 - 9 = 2$.
- Vertex `4` has priority $11 - 8 = 3$.

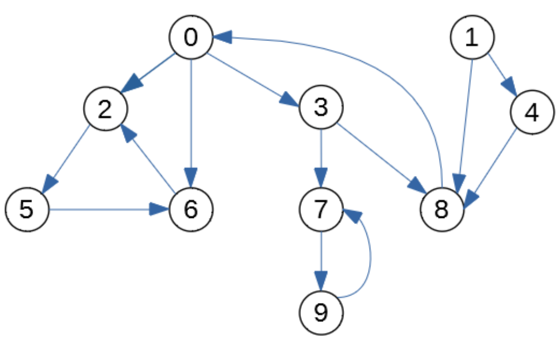
Now run DFS(G^T). It turns out that the DFS algorithm starts in the vertex `"0"` once again (since it was finished last in DFS(G)). But unlike the DFS algorithm in G itself (it produced just one DFS tree), we get a DFS forest with 3 components (tree/discovery edges shown bold and black in the previous Figure).

- $\{0, 1, 2\}$ (DFS tree has root "0").
- $\{3\}$ (DFS tree has root "3").
- $\{4\}$ (DFS tree has root "4").

They represent the strongly connected components in G (they are also strongly connected in G^T).

8.3 Problem

We start with the graph shown in Figure below.



(A) Run the DFS traversal algorithm on the graph G. Mark each vertex with the pair of numbers `d/f`, where the first number `d` is the discovery time, and the second number `f` is the finishing time.

(B) Draw the transposed directed graph (same vertices, but each arrow points in the opposite direction). Run the DFS traversal algorithm on G^T. Make sure that the DFS outer loop visits the vertices in the reverse order by $u.f$ (the finishing time for the DFS algorithm in step **(A)**). In this case you do not produce the discovery/finishing times once again, just draw the discovery edges used by the DFS on G^T – you can highlight them (show them in bold or use a different color).

(C) List all the strongly connected components (they are the separate pieces in the forest obtained by running DFS on G^T).