Strongly Connected Components

Decomposing a directed graph into its strongly connected components is a classic application of DFS. Two vertices of directed graph are in the same component if and only if they are reachable from each other. For example, consider the following directed graph

Diagram

The above directed graph has four strongly connected components, namely $\{a, b, e\}$, $\{c, d\}$, $\{f, g\}$ and $\{h\}$.

From any vertex v, one can visit to any other vertex in the same component as *v* and then return back v; if one visits a vertex in a different component the return to *v* is impossible.

The component graph for the above directed graph is

Diagram

The above directed graph has 4 strongly connected components: c_1, c_2, c_3 and c_4. If G has an edge from some vertex in c_i to some vertex in c_j where $i \neq j$, then one can reach any vertex in c_j from any vertex in c_i but not return. In the example, one can reach any vertex in c_3 from any vertex in c_1 but cannot return to c_1 from c_3.

If G = (V, E) is a directed graph, its transpose, $G^T = (V, E^T)$ is the same as G with all arrows reversed.

For example, given directed graph G = (V, E)

Diagram

The transpose of G = (V, E) is $G^T = (V, E^T)$

Diagram

From above example it is apparent that edge set E^T contains edge (u, *v*) iff edge set E contains (u, *v*). This observation implies that G^T has same strongly components as G and the strongly components of G are transposes of strongly components of G^T.

ALGORITHM

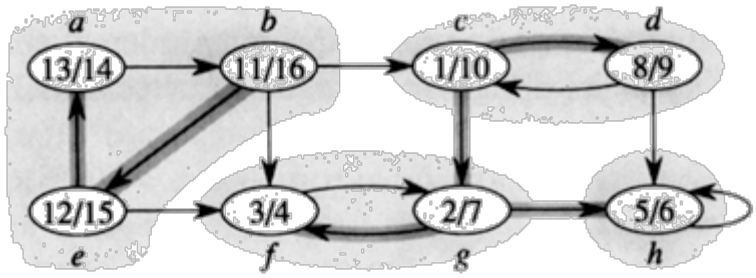
A DFS(G) produces a forst of DFS-trees. Let C be any strongly connected component of G, let *v* be the first vertex on C discovered by the DFS and let T be the DFS-tree containing v when DFS-visit(*v*) is called all vertices in C are reachable from *v* along paths containing visible vertices; DFS-visit(v) will visit every vertex in C, add it to T as a descendant of *v*.

STRONGLY-CONNECTED-COMPONENTS(G)

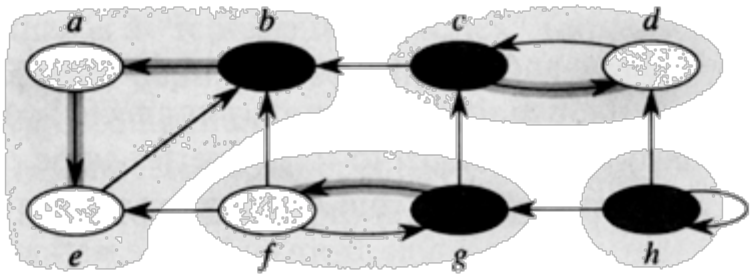
1. Call DFS(G) to compute finishing time for each vertex.
2. Compute transpose of G i.e., G^T.
3. Call DFS(G^T) but this time consider the vertices in order of decreasing finish time.
4. Out the vertices of each tree in DFS-forest.

Example (CLR) Consider a graph G = (V, E)

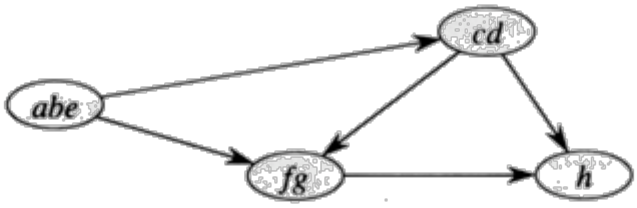
1. Call DFS(G)



2. Compute G^T



3. Call DFS(G^T) but this time consider the vertices in order of decreasing finish time.



First 16
Start with 10
Start with 7

4. Output the vertices of each tree in the DFS-forest as a separate strongly connected component.

$$\{a, b, e\}, \{c, d\}, \{f, g\} \text{ and } \{h\}$$

The algorithm computes the strongly connected components of a directed graph G = (V, E) using two depth searches, one on G and one on G^T. Thus, the total running time is linear i.e., $\Theta(V + E)$.

Before leaving strongly connected components, lets prove that component graph of

G(V, E) is a directed acylic graph.

Proof (by contradiction)

Suppose component graph of $G = (V, E)$ was not a DAG and G comprised of a cycle consisting of vertices $v_1, v_2, \ldots, v_n$. Each v_i corresponds to a strongly connected component (SCC) of component graph G. If $v_1, v_2, \ldots, v_n$ themselves form a cycle then each v_i (i runs from 1 to n) should have been included in the SCC corresponding to v_j (j runs from 1 to n and $i \neq j$). But each of the vertices is a vertex from a difference SCC of G. Hence, we have a contradiction! Therefore, SCC of G is a directed acylic graph.

Back