CS 5800 Assignment 4 Solutions

Course Staff

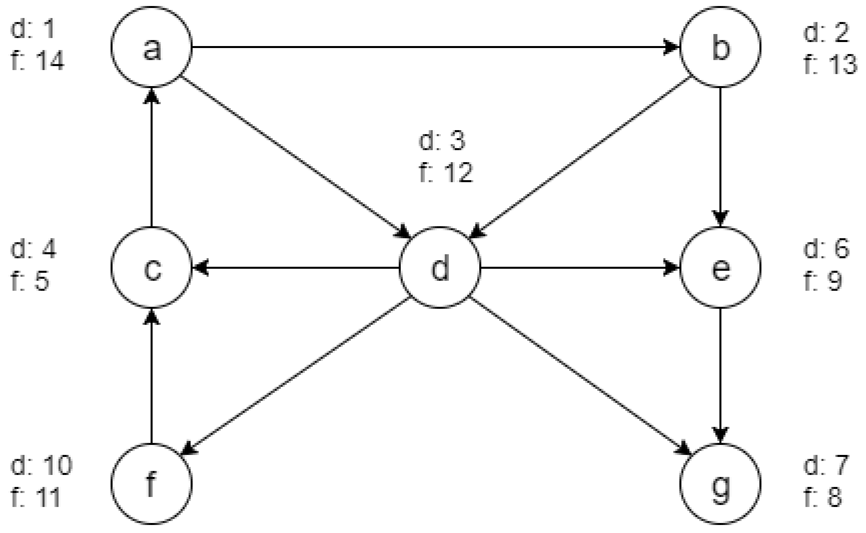
August 9, 2017

1 Strongly Connected Components

We will find strongly connected components of a directed graph $G = (V, E)$ using following algorithm:

1. We will apply DFS algorithm on the graph G and store the finishing time $f(u)$ of the vertices.

   We will apply DFS on the vertices alphabetically (if applied differently it will change the order but the final result of the algorithm won't change). After applying DFS the start and finishing values of the vertices are represented by d and f respectively in the following graph.

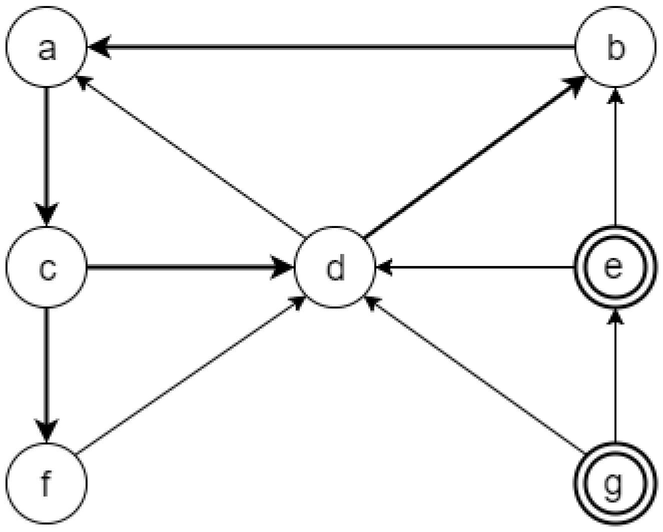
   Arrangement of vertices in order of decreasing finishing time: $\{a, b, d, f, e, g, c\}$

2. Secondly, we will apply DFS on the transpose graph denoted by G^T, considering the vertices in descending order of their finishing times of the DFS

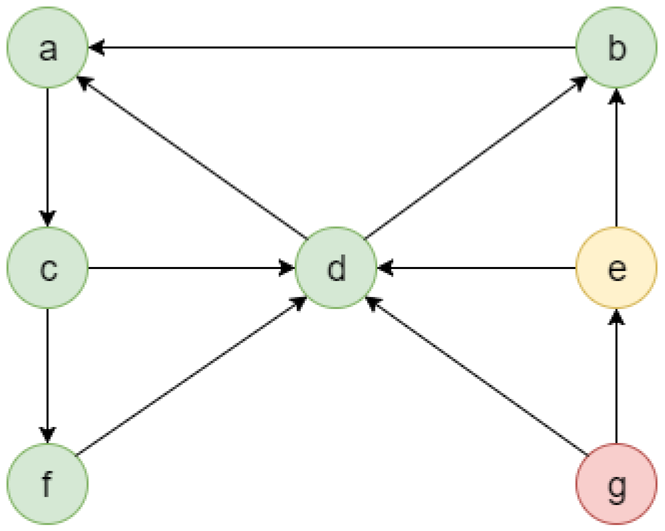
calculated above.

To find transpose of graph G we will reverse direction of all the edges and apply DFS in the following order:
$\{a, b, d, f, e, g, c\}$

We start by applying DFS from the vertex $\{a\}$, the component that we get is $\{a, b, c, d, f\}$. Then we apply on next vertex i.e. vertex $\{e\}$. Since all the vertex are already visited, vertex $\{e\}$ forms a component similarly vertex $\{g\}$ forms a component.



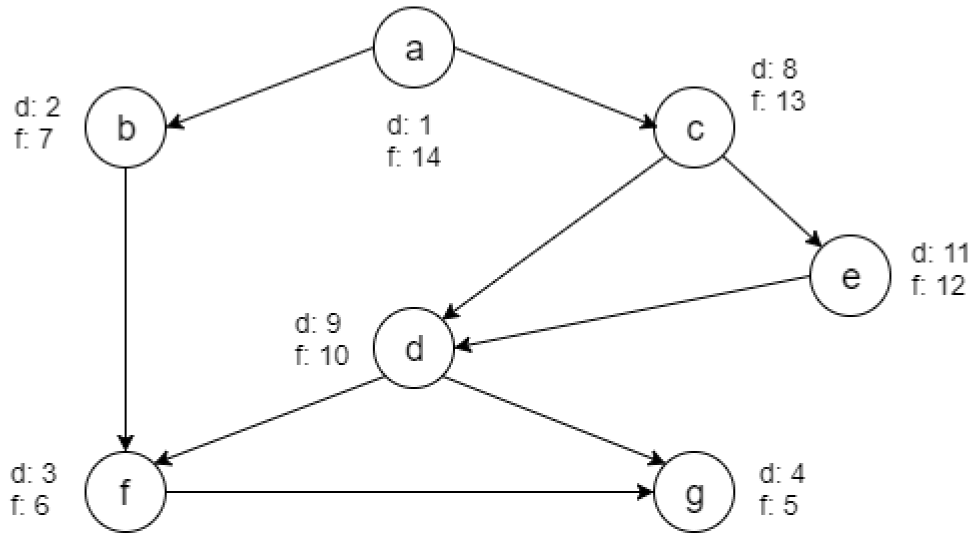
3. The components of G^T are the strongly connected components of G.
$\{a, b, c, d, f\}, \{e\}, \{g\}$



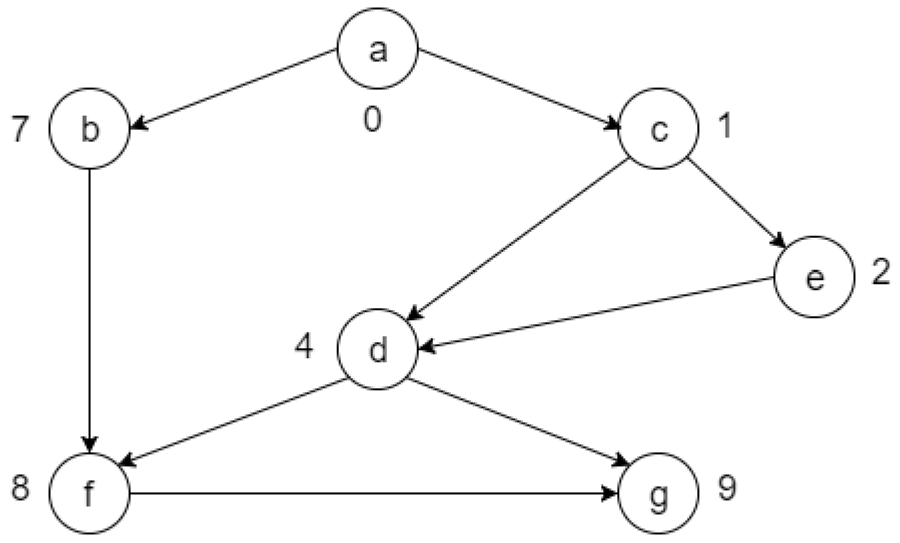
2 Topological Sort

We can find topological sort of Directed Acyclic Graph using following step:

1. We apply DFS algorithm on graph G and store the finishing time $f(u)$ of the vertices. We will apply DFS on vertices alphabetically (if applied differently it will change the order but the final result of the algorithm won't change). After applying DFS the start and finishing value of vertices is represented by d and f respectively in the following graph.



2. Now we will find topological sort using function φ given by:
   $\varphi(u) = 2|V| - f(u)$
   Applying above function on graph:



   Hence topological sorting order is: $\{a, c, e, d, b, f, g\}$

3 Question 3

The DFS algorithm essentially finds the required path.
We can find the required path in the following way. We apply the DFS algorithm from a random vertex. When we get to some vertex u, the algorithm go over all of the neighbors of u. If the algorithm finds that the neighbor v had already been visited (it is either gray or black) then we add to our path the 2 edges $\{u, v\}, \{v, u\}$ (The graph is undirected so this is the same edge twice. It means that our path will look ... $\rightarrow u \rightarrow v \rightarrow u \rightarrow$ ...). If the vertex v is white then the DFS algorithm will "move" to v and we take the edge $\{u, v\}$ to our path. When the DFS algorithm backtracks from v to u (this happens when v becomes black) we take the edge $\{v, u\}$ for the second time.
This algorithm will give us a path that uses each edge of G exactly twice.

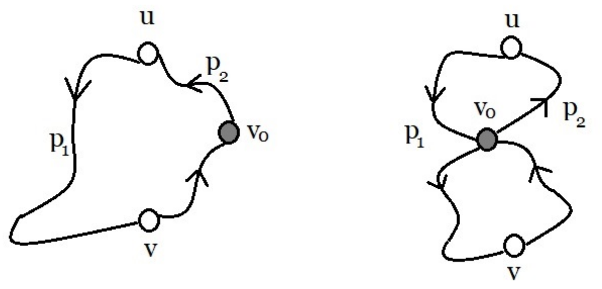
4 Question 4

Let $u, v \in R_G$ be two different roots. The fact that u is a root implies that there is a path $p_1 : u \overset{G}{\rightsquigarrow} v$ from u to v in the graph G. Since v is also a root, there exists a path $p_2 : v \overset{G}{\rightsquigarrow} u$ from v to u in the graph G.

It follows that in the graph G there is a cycle c:

$$c : u \overset{p_1}{\rightsquigarrow} v \overset{p_2}{\rightsquigarrow} u$$

Let v_0 be the vertex whose detection time $d(v_0)$ is minimal among the vertices of the cycle c (v_0 is the first vertex on the cycle c that becomes gray). It follows that on the time $d(v_0)$ there is a white path from v_0 to u and a white path from v_0 to v. It follows from the white path theorem that both v and u are descendants of v_0 in the DFS forest, and hence they are in the same DFS component. Note that both of the following options are possible:



Note that using similar arguments, it is not hard to show that the component containing the roots will always be the last DFS component of the DFS forest.

5 Question 5

5.a

The idea is the following. If a vertex v is on some path from v_0 to u, then there is a path from v_0 to v, and in the graph G^t there is a path from u to v. If we take only those vertices that are reachable from v_0 in G, and are reachable from u in G^t, we get only vertices that are on some path from v_0 to u. It is then enough to see if there is a cycle in the graph that contains only those vertices.

Our input is G = (V, E) the given graph, v_0 the source vertex and u the destination vertex. Let DFS(X, v) be a function that performs DFS on graph X with source vertex v and returns the DFS tree that contains v.

We can apply DFS to G with v_0 as source vertex to get a tree with all the vertices that are reachable from v_0. We consider the subgraph G' of G that is defined by the vertices that are reachable from v_0. That is, $G' = (V'E')$, where $V' = \{v \in V | v \text{ is reachable from } v_0\}$ and $E' = \{(u,v) | (v,u) \in E \text{ and } u, v \in V'\}$. If we apply DFS to $(G')^t$ with u as source vertex, we will get a tree with only those vertices that are reachable from both u in $(G')^t$. Vertices that are reachable from u in $(G')^t$ are vertices in G' that has a path connecting them to u in G'. We now construct another graph $G'' = (V'', E'')$ where $V'' = \{v \in V' | \text{v is reachable from } u \text{ in } (G')^t\}$ and $E'' = \{(u,v) \in E | u, v \in V''\}$. Next we look for cycles in G" using the algorithm from 4 (c) in Homework 3. If there is a cycle in G", then there is a path from v_0 to u with a cycle, otherwise there is no such path.

```
procedure ExistsCyclePath(G, $v_0$, u)
    T = DFS(G, $v_0$)
    $V' = \{v \in V | v$ is reachable from $v_0\}$ // take only vertices in $T$
    $E' = \{(u, v) | (v, u) \in E$ and $u, v \in V'\}$
    $G' = (V', E' \ )$
    $T' = DFS((G')^t, u)$
    $V'' = \{v \in V'|$v is reachable from $u$ in $(G')^t\}$ // take only vertices in $T'$
    $E'' = \{(u, v) \in E | u, v \in V''\}$
    $G'' = (V'', E \ )$
    return IsCyclic($G''$)
```

5.b

We can simply use the algorithm *ExistsCyclePath* for all the vertices in U.