Strong Components of a Directed Graph

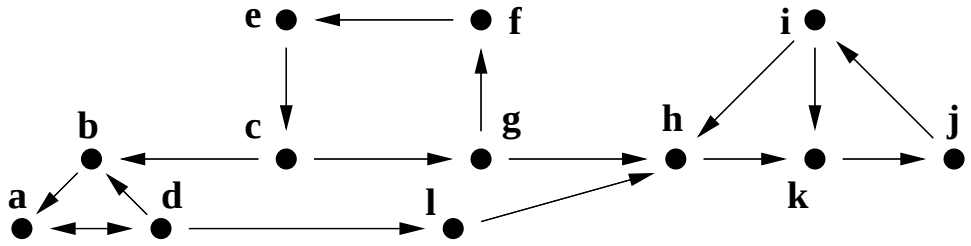
Our textbook, *Algorithms* by Dasgupta, Papadimitriou, and Vazirani, contains what I believe is an important error on page 94, in the description of the algorithm for finding the strong components of a directed graph G. I believe it should read:

1. Run depth-first search on G^R, creating a list of the vertices in order of their `post` numbers.
2. Run depth first search on G, processing the vertices in decreasing order of their `post` numbers from Phase 1.
3. The depth first search in Phase 2 consists of phases. A phase ends when there is no unvisited out-neighbor of the current vertex. The vertices visited during each phase constitute one strong component.

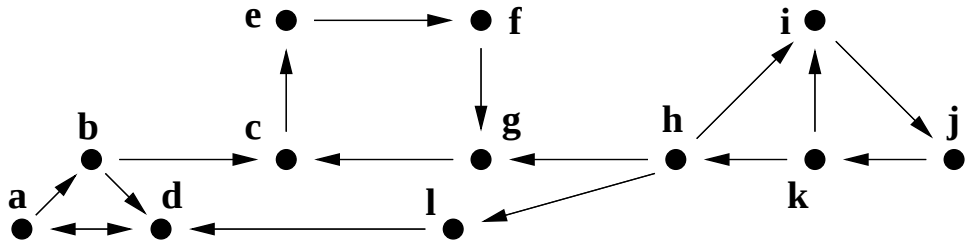
You can reverse these; use G in Phase 1 and G^R in step 2. The strong components are exactly the same, but created in a different order.

An Example

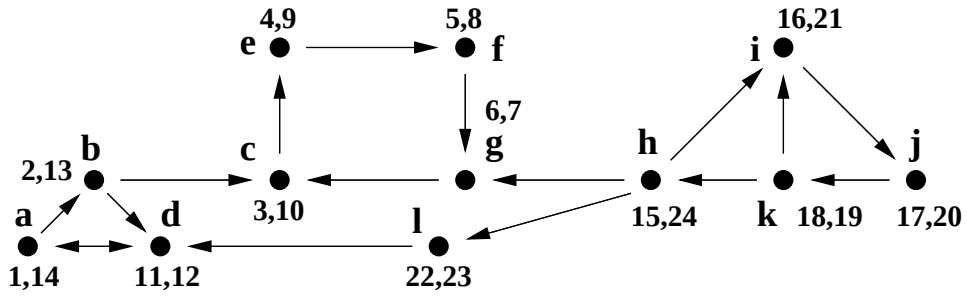
We will step through the algorithm for a directed graph G of twelve vertices shown below. We use lower case letters `a ...l` for the names of the vertices.



The reverse graph G^R:

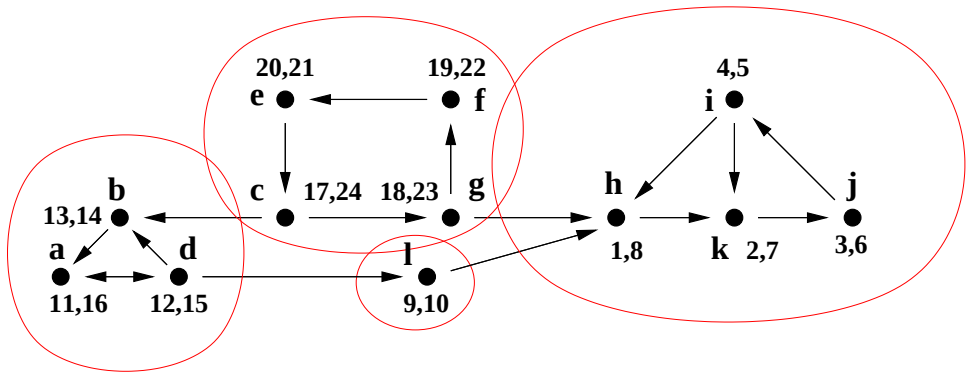


We now execute Phase 1 of the algorithm. Each vertex is labeled with its `pre` amd `post` numbers.



At each postvisit, we append the name of the vertex to a list. The list of vertices in order of their Phase 1 `post` number is `g,f,e,c,d,b,a,k,j,i,l,h`.

We now execute Phase 2, processing vertices in the reverse order of our list. (We do not actually use the Phase 2 `pre` and `post` numbers, shown just to aid comprehension.)

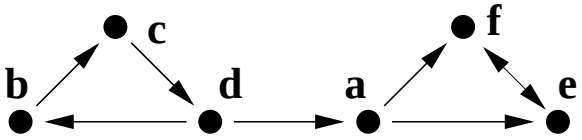


We show the stack at each step of Phase 2, where \$ indicates the bottom of the stack. A component is defined whenever the stack becomes empty, after which `explore` begins at the unvisited vertex with the largest Phase 1 `post` number. Strong components are indicated in the figure.

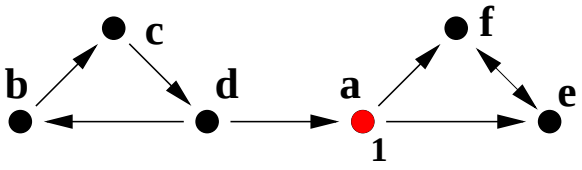
```
$
$h
$hk
$hk
$hkj
$hkji
$hkj
$hk
$h
$       {h,k,j,i} is a strong component
$l
$       {l} is a strong component
$a
$ad
$adb
$ad
$a
$       {a,d,b} is a strong component
$c
$cg
$cgf
$cgfe
$cgf
$cg
$c
$       {c,g,f,e} is a strong component
```

An Example in Greater Detail

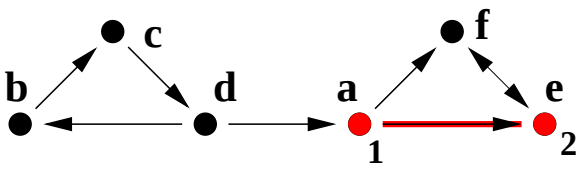
We now walk through a smaller example showing more steps. At each step the DFS stack is shown in red. We use the alphabetic tie-breaker rule. Initially, the stack is empty.



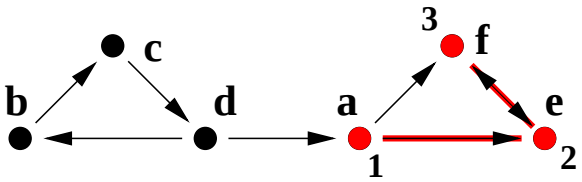
Initial digraph $G = (V, A)$. Vertices are labeled **a,b,c,d,e,f**. When a vertex is pushed onto the stack, it is given a pre-number, and when it is popped it is given a post-number.



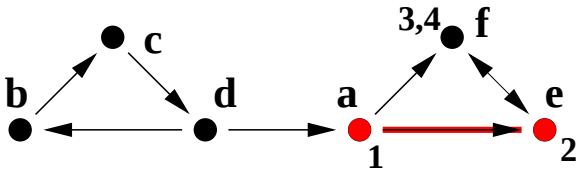
The DFS stack is initialized at **a**, using the alphabetic rule. The vertex is labeled with the pre-number 1.



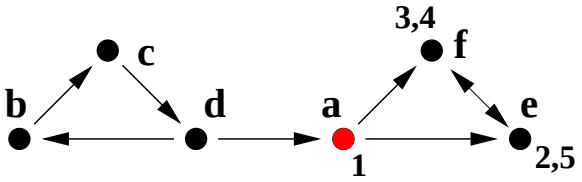
e, the alphabetically first out-neighbor of **a**, is pushed, and labeled with the pre-number 2. The stack is now **ae**.



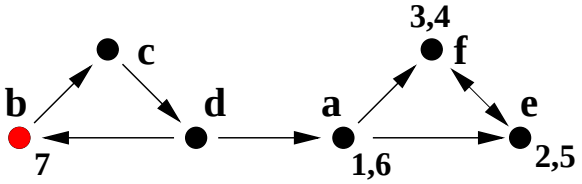
f, the alphabetically first out-neighbor of **e**, is pushed, and labeled with the pre-number 2. The stack is now **aef**



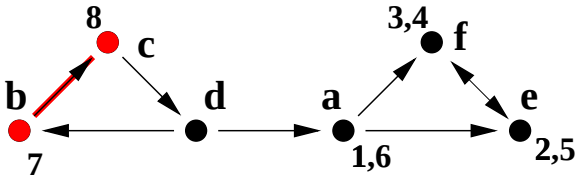
The top vertex **f** has no unvisited out-neighbors, and hence is popped and given the post-number 4.



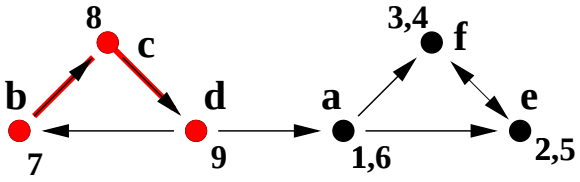
The top vertex **e** has no unvisited out-neighbors, and hence is popped



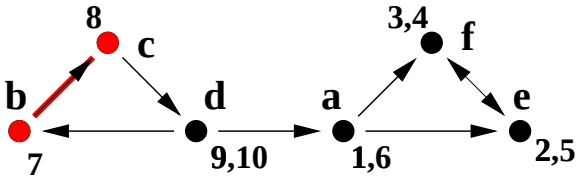
The top vertex **a** has no unvisited out-neighbors, and hence is popped and given the post-number 6. The stack is empty. The alphabetically first unvisited vertex, **b** is pushed and given the pre-number 7. The stack is now **b**



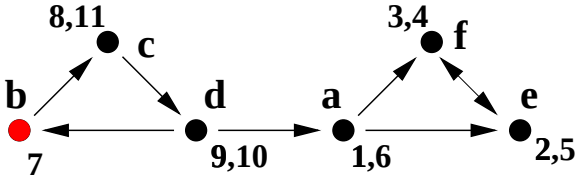
The alphabetically first unvisited out-neighbor of the top vertex **b** is pushed. The stack is now **bc**.



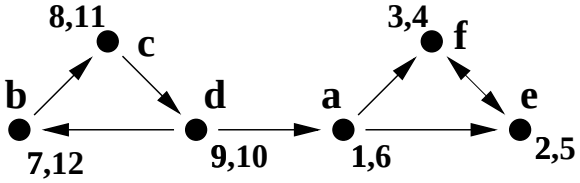
The alphabetically first unvisited out-neighbor **d** of the top vertex **c** is pushed and given the pre-number 9. The stack is now **bcd**.



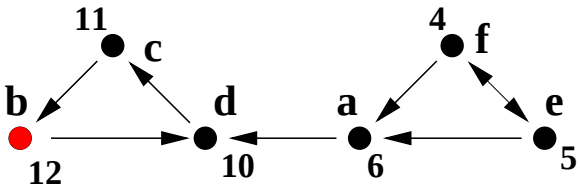
The top vertex **d** has no unvisited out-neighbor, hence is popped and given the post-number **10**. The stack is now **bc**.



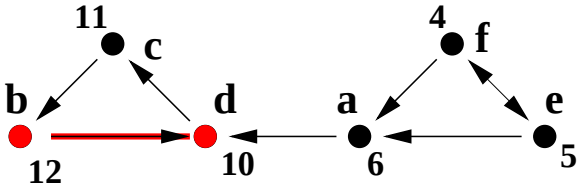
The top vertex **c** has no unvisited out-neighbor, hence is popped and given the post-number **11**.



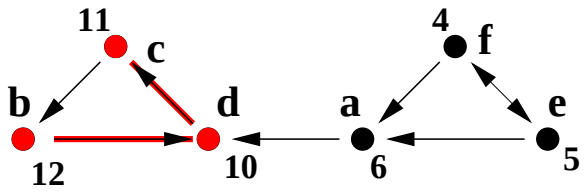
The top vertex **b** has no unvisited out-neighbor, hence is popped and given the post-number **12**. The stack is now empty.



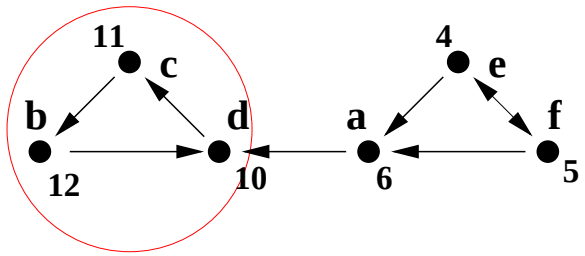
All arcs are inverted, and pre-numbers are deleted. The second DFS phase begins. Henceforth, "unvisited" means unvisited during the second phase. The vertex of highest post-number **b** is pushed.



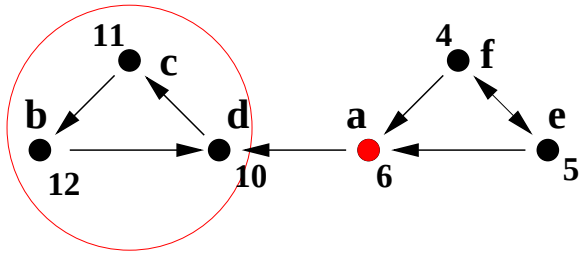
The out-neighbor **d** of the top vertex **b** of highest post-number is pushed. The stack is now **bd**.



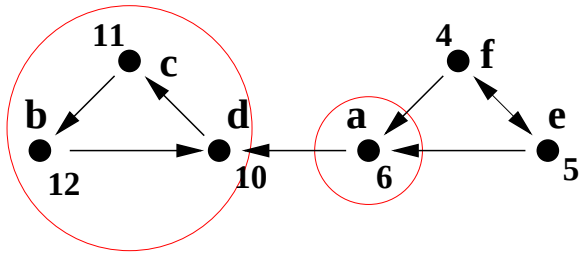
The out-neighbor **c** of the top vertex **d** of highest post-number is pushed. The stack is now **bdc**.



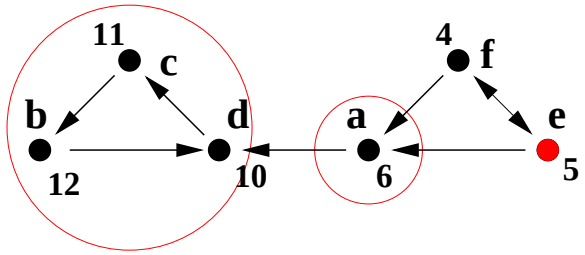
The top vertex **c** has no unvisited out-neighbor. All vertices pushed onto the stack since the last time it was empty constitute a strong component, namely **b**, **c**, and **d**. The stack is cleared, and is now empty.



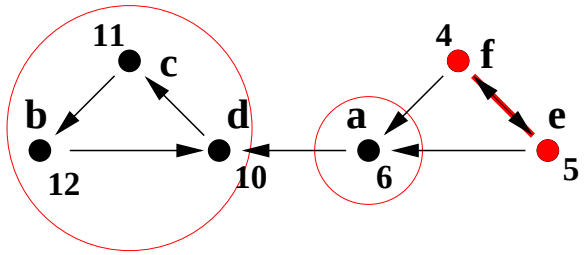
The unvisited vertex of highest post-number, namely **a**, is pushed. The stack is now **a**



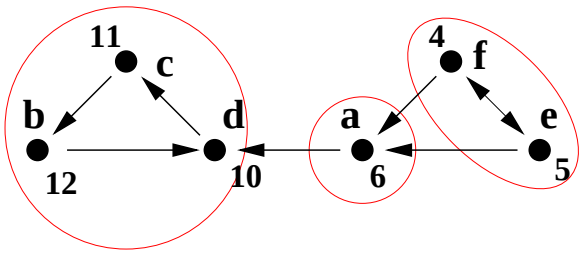
The top vertex **a** has no unvisited out-neighbor. Only one vertex, namely **a** was pushed since the last time the stack was empty, so **a** is the sole element of a strong component. The stack is emptied.



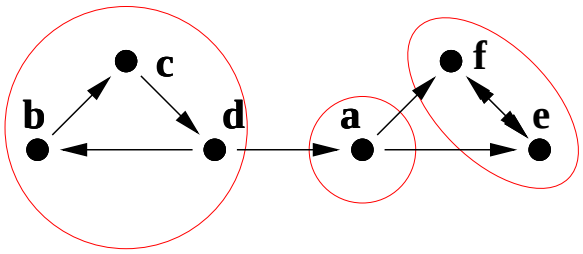
The stack is restarted by pusing the unvisited node of heighest post-number, namely **e**. The stack is now **e**.



The out-neighbor **f** of the top vertex **e** of highest post-number is pushed. The stack is now **ef**.



The top vertex **f** has no unvisited out-neighbor. The vertices **e** and **f** were pushed since the last time the stack was empty, so they constitute the final strong component. The stack is emptied.



The arcs are inverted again. We now show the original digraph with its strong components.