CSC222 Data Structures & Algorithms II Spring 2014

Depth First Search of a Graph

Depth first search of a graph is similar to a tree traversal, except

1. We must re-start the search if there are nodes we do not reach from our starting point.
2. We must mark the nodes as we go so we do not continue to follow a cycle.

An algorithm for depth-first search including depth-first numbering is given below:

Input: Undirected Graph $G = (V, E)$

```
dfs( G )
    dfnumber = 0 ;
    for all v ∈ V do
        visited[v] = false ;
    for all v ∈ V do
        if ( not visited[v] ) {
            explore( G, v )
        }
```

Input: Undirected Graph $G = (V, E)$ and starting node v

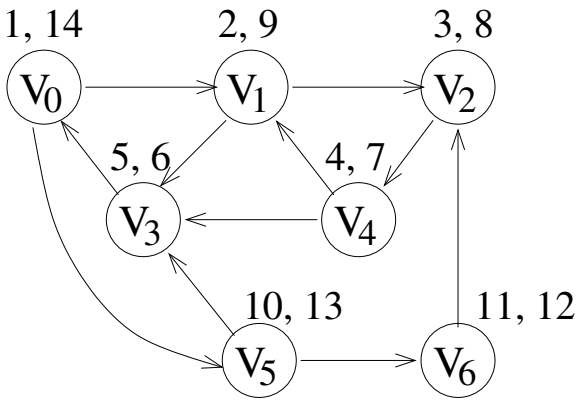
```
explore( G, v )
    visited [ v ] = true ;
    previsit ( v ) ;
    for each edge (v, w) ∈ E do
        if ( not visited[w] ) explore( G, w )
    postvisit ( v ) ;
```

```
previsit( v )
{
    dfnumber = dfnumber + 1 ;
    prenumber[ v ] = dfnumber ;
}
```

```
postvisit( v )
{
    dfnumber = dfnumber + 1 ;
    postnumber[ v ] = dfnumber ;
}
```

Example with Pre- and Post- Numbers

For consistency, we let each node visit its adjacent nodes in numerical order.



A depth first search implicitly defines a **depth first tree**. A depth first search also partitions the edges into different types:

Tree edges – Edges that are followed to nodes that are not previously marked "visited" during the DFS.

Forward edges – Edges that go from an ancestor to a descendent in the DFS tree.

Back edges – Edges that go from an descendent to an ancestor in the DFS tree.

Cross edges – Edges that go from between sibling branches of the DFS tree. I.e., between nodes which are neither a descendant nor an ancestor of each other.

Fill-in the blank for the graph above:

Tree edges _____

Forward edges _____

Back edges _____

Cross edges _____

Classifying Edges by Pre- and Post- Numbers Suppose $(\mathbf{x}, \mathbf{y})$ is an edge in E.

- If $\text{pre}(x) < \text{pre}(y) < \text{post}(y) < \text{post}(x)$ then $(\mathbf{x}, \mathbf{y})$ is a tree edge or a forward edge.
- If $\text{pre}(y) < \text{pre}(x) < \text{post}(x) < \text{post}(y)$ then $(\mathbf{x}, \mathbf{y})$ is a back edge.
- If $\text{pre}(x) < \text{post}(x) < \text{pre}(y) < \text{post}(y)$ then $(\mathbf{x}, \mathbf{y})$ is a cross edge.
- If $\text{pre}(y) < \text{post}(y) < \text{pre}(x) < \text{post}(x)$ then $(\mathbf{x}, \mathbf{y})$ is a cross edge.

Connected Components

Undirected Graphs

We can find the connected components of an undirected graph by a simple modification of the depth first search algorithm.

Input: Undirected Graph $G = (V, E)$

```
dfs( G )
    ccount = 0 ;
    dfnumber = 0 ;
    for all v ∈ V do
        visited[v] = false ;
    for all v ∈ V do
        if ( not visited[v] ) {
            ccount = ccount + 1 ;
            explore( G, v )
        }
```

Input: Undirected Graph $G = (V, E)$ and starting node v

```
explore( G, v )
    visited [ v ] = true ;
    ccnum [ v ] = ccount ;
    previsit ( v ) ;
    for each edge (v, w) ∈ E do
        if ( not visited[w] ) explore( G, w )
    postvisit ( v ) ;
```

```
previsit( v )
{
    dfnumber = dfnumber + 1 ;
    prenumber[ v ] = dfnumber ;
}
```

```
postvisit( v )
{
    dfnumber = dfnumber + 1 ;
    postnumber[ v ] = dfnumber ;
}
```

Directed Graphs

The concept of "connected components" requires some clarification for directed graphs. Two vertices u and v are **strongly connected** if and only if there exists a path from u to v in G, and also a path from v to u. The **strongly connected component** containing vertex v is the set of all vertices u which are strongly connected to v.[1]

Definition: The strongly connected components can be viewed as nodes in a **meta-graph**. Let C and C' be two strongly connected components. If there exists an edge $(u, v) \in E$ such that $u \in C$ and $v \in C'$ then (C, C') is an edge in the meta-graph.

Definition: A node in a directed graph with no incoming edges is called a **source**. A node in a directed graph with no outgoing edges is called a **sink**.

Definition(s): A strongly connected component with no incoming edges in the meta-graph is called a **source component**. A strongly connected component with no outgoing edges in the meta-graph is called a **sink component**.

Properties:

- If the `explore()` function is started at node v then it will visit exactly the set of nodes reachable from v.
- The node that receives the highest post number must lie in a source component.
- If C and C' are strongly connected components and there is an edge from a node in C to a node in C', then the largest post number in C is greater than the largest post number in C'.

Observe: If we start the `explore()` function in a vertex belonging to a sink component, it will find exactly that sink component. The question is: How do we find a node that belongs to a sink component ?

Definition: Let $G = (V, E)$ be a directed graph. We define the **reverse graph** to be $G^R = (V, E^R)$, where

$$E^R = \{ (u, v) \mid (v, u) \in E \}$$

Observe:

- Both G and G^R have the same strongly connected components.
- A source component in G is a sink component in G^R.
- A sink component in G is a source component in G^R.

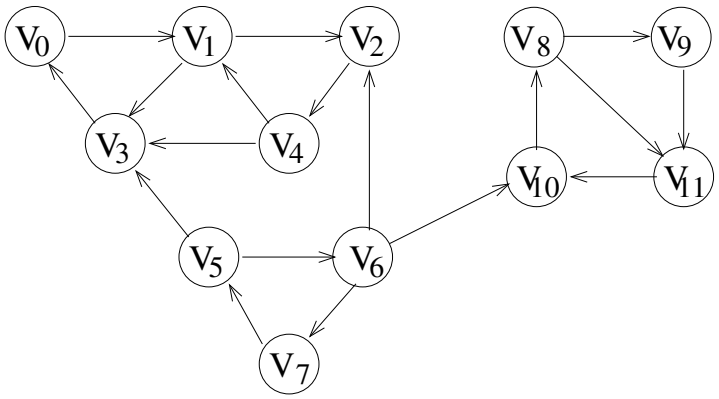
We can combine definitions and observations to construct an efficient algorithm for finding strongly connected components.

[1] This definition implies that every node within the strongly connected component is strongly connected to every other node in that component.

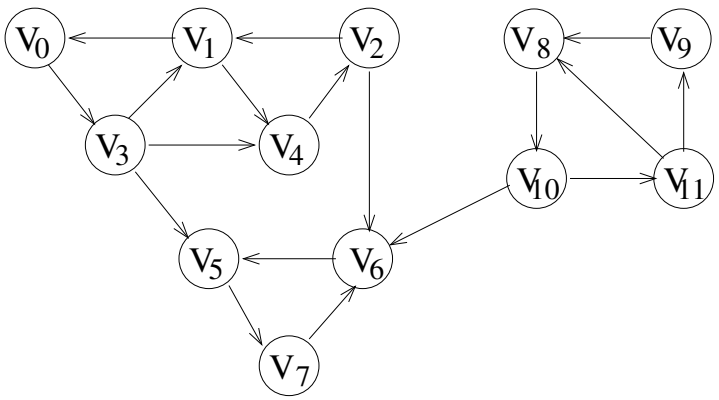
Input: A directed Graph $G = (V, E)$
Output: Strongly connected components of G)

strongly_connected_components($G = (V, E)$)
    construct G^R
    dfs(G^R)
    Sort nodes by decreasing post number in G^R // $\mathcal{O}(n)$ by bucket sort
    Let `nodes[j]` denote the array of sorted nodes.
    ccount = 0 ;
    for all $v \in V$ do
        G.visited[v] = false ;
    while there exists unvisited nodes in G do
    {
        Select an unvisited node v in G with the highest post-number in G^R.
        // Implement search for v by scanning the `nodes[]` array.
        ccount = ccount + 1 ;
        explore(G, v) ;
    }

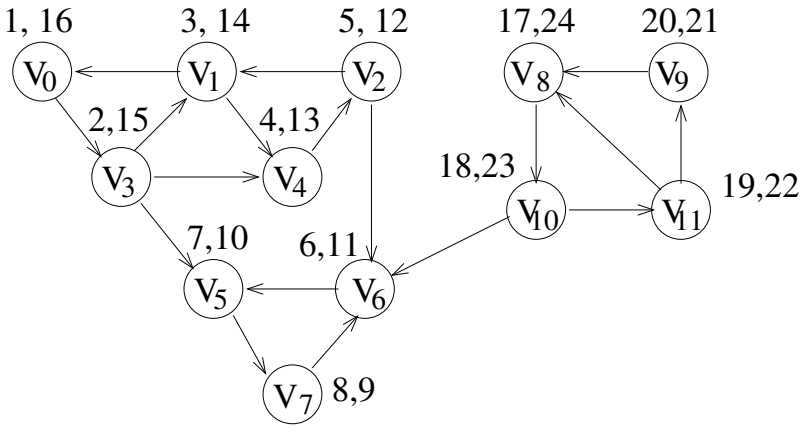
Example Graph G:



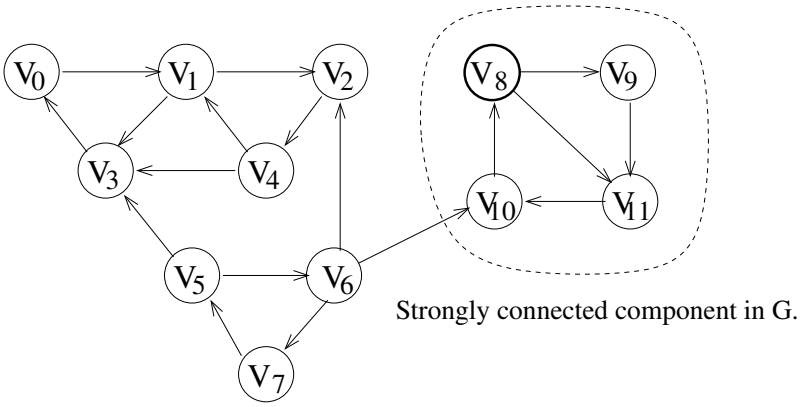
Example Reverse Graph G^R:



Example Reverse Graph with DFS Numbers:



Example Strongly Connected Component in Graph G:



Next Strongly Connected Component in Graph G: