Topological Sort

dfs的递归实现, 把一个dfs的postorder反过来就是topological order
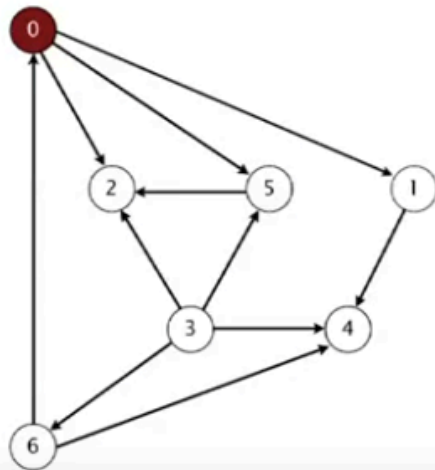
Depth-first search order

```java
public class DepthFirstOrder
{
    private boolean[] marked;
    private Stack<Integer> reversePost;

    public DepthFirstOrder(Digraph G)
    {
        reversePost = new Stack<Integer>();
        marked = new boolean[G.V()];
        for (int v = 0; v < G.V(); v++)
            if (!marked[v]) dfs(G, v);
    }

    private void dfs(Digraph G, int v)
    {
        marked[v] = true;
        for (int w : G.adj(v))
            if (!marked[w]) dfs(G, w);
        reversePost.push(v);
    }

    public Iterable<Integer> reversePost()
    {   return reversePost;  }
}
```

returns all vertices in
"reverse DFS postorder"

Leetcode 207

Strong Component 强连通分量

link

G'是G的反向图，通过dfs求G'的拓扑排序(reverse post dfs)，然后根据拓扑排序，的顺序每次dfs的路径都是强联通分量

link

connected component

```java
public class CC
{
    private boolean marked[];
    private int[] id;
    private int count;

    public CC(Graph G)
    {
        marked = new boolean[G.V()];
        id = new int[G.V()];

        for (int v = 0; v < G.V(); v++)
        {
            if (!marked[v])
            {
                dfs(G, v);
                count++;
            }
        }
    }

    private void dfs(Graph G, int v)
    {
        marked[v] = true;
        id[v] = count;
        for (int w : G.adj(v))
            if (!marked[w])
                dfs(G, w);
    }

    public boolean connected(int v, int w)
    {  return id[v] == id[w];  }
}
```

strong component

```java
public class KosarajuSharirSCC
{
    private boolean marked[];
    private int[] id;
    private int count;

    public KosarajuSharirSCC(Digraph G)
    {
        marked = new boolean[G.V()];
        id = new int[G.V()];
        DepthFirstOrder dfs = new DepthFirstOrder(G.reverse());
        for (int v : dfs.reversePost())
        {
            if (!marked[v])
            {
                dfs(G, v);
                count++;
            }
        }
    }

    private void dfs(Digraph G, int v)
    {
        marked[v] = true;
        id[v] = count;
        for (int w : G.adj(v))
            if (!marked[w])
                dfs(G, w);
    }

    public boolean stronglyConnected(int v, int w)
    {  return id[v] == id[w];  }
}
```