Depth-First Search

If we want to explore a graph, determine whether two nodes are connected, or determine some properties regarding the ordered structure of a directed graph, we use *graph search* algorithms. The two most basic graph search algorithms are *depth-first* and *breadth-first*. Depth-First Search is the algorithm that shoots as far away from a node possible to see if it results in a successful path, and only turns around if it dead ends.

For now, we ignore the calls previsit(v) and postvisit(v).

Algorithm 1 DFS(s, G)

   Input: Graph $G = (V, E)$ and vertex v.
   for each $v \in V$ **do**
      mark v as unexplored
      set v's parent to Null
   end for
   search(s, G)
   return forest F formed by parents

Algorithm 2 search(v, G)

   Input: Graph $G = (V, E)$ and vertex v.
   mark v as explored
   previsit(v)
   for $(v, w) \in E$ **do**
      if w is unexplored **then**
         set w's parent to v
         search(w, G)
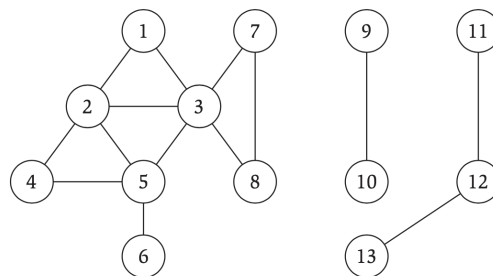   **end if**
   end for
   postvisit(v)

Figure 1: Example graph G. From Kleinberg Tardos.

Exercise.

1. Consider the pseudocode when called on the above example, that is, what happens when we run DFS(1, G) where G is the graph above? Draw the DFS forest as the graph is explored.

2. DFS implicitly uses a stack. Draw the stages of the stack as it runs on the example graph.
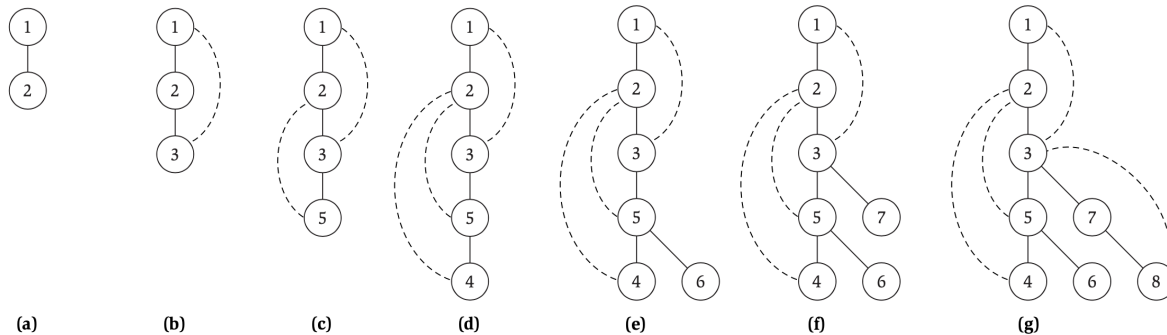
Figure 2: The DFS tree is shown in solid edges as constructed in stages by the above algorithm. The dashed edges are the edges that do not belong to the DFS tree but do belong to G. From Kleinberg Tardos.

Preorder, Postorder, and Tree vs. Non-Tree Edges

Preorder and Postorder: Now, we'll define what happens in $previsit()$ and $postvisit()$. Our idea is to track when nodes are pushed onto the stack and when they are popped off, as this order winds up giving us important properties. Formally, we have the following definitions.

Definition 1. The method previsit(v) assigns a vertex v's *preorder* time, that is, i if it is pushed onto the stack at time i.

Definition 2. The method postvisit(v) assigns a vertex v's *postorder* time, that is, i if it is popped off of the stack at time i.

That is, we set "clock" to 0 when vertices are marked unexplored, and then $previsit(v)$ increments the clock by 1 and sets $v.preorder = clock$ and $postvisit(v)$ increments the clock by 1 and sets $v.postorder = clock$.

Claim 1. For any vertices u and v, either $[preorder(u), postorder(u)]$ and $[preorder(v), postorder(v)]$ are disjoint or nested.

Moreover, the former contains the latter if and only if search(v, G) is called during the execution of search(u, G), or equivalently, if and only if u is an ancestor of v in the DFS tree.
We can partition the edges of G into four types:

- *Tree edges* are edges in the depth-first forest F. Edge (u, v) is a tree edge if v was first discovered by exploring edge (u, v).

- *Back edges* are those edges (u, v) connecting a vertex u to an ancestor v in a depth-first tree. We consider self-loops, which may occur in directed graphs, to be back edges. In undirected graphs, there is no distinction between back edges and forward edges.

- *Forward edges* are those nontree edges (u, v) connecting a vertex u to a descendant in a depth-first tree.
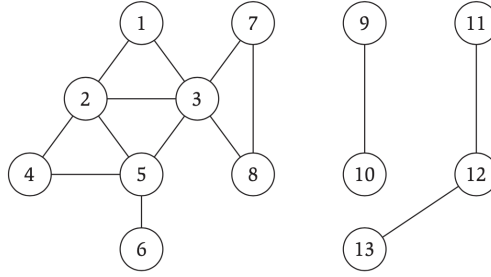
Figure 3: Example graph G. From Kleinberg Tardos.

- *Cross edges* are all other edges. They can go between vertices in the same depth-first tree, as long as one vertex is not an ancestor of the other, or they can go between vertices in different depth-first trees. There are no cross edges in undirected graphs. (Can you explain why?)

Claim 2. If $(u, v) \in E$ then $postorder(u) < postorder(v) \iff (u, v)$ is a back edge.

Proof. If $(u, v) \in E$, then before u is popped off of the stack, we *could* have pushed v onto the stack via (u, v).

($\Leftarrow$) But if (u, v) is a backedge and not a tree edge, it must already be on the stack underneath u, and thus will pop after u.

($\Rightarrow$) Or, if $postorder(u) < postorder(v)$, (u, v) cannot be a tree or forward edge or we'd push and pop v after pushing u, and thus we'd have $postorder(v) < postorder(u)$, a contradiction. Then (u, v) must be a backedge. □

Claim 3. $G = (V, E)$ has a cycle $\iff$ the DFS tree of G yields a back edge.

Proof. ($\Leftarrow$) If (u, v) is a back edge, then (u, v) together with the path from v to u in the DFS forest form a cycle.

($\Rightarrow$) Conversely, for any cycle in $G = (V, E)$, consider the vertex assigned the smallest postorder number. Then the edge leaving this vertex in the cycle must be a back edge by Claim 2, since it goes from a lower postorder number to a higher postorder number. □