12.1 Types of Edges

Given a graph \( G = (V, E) \), we can use depth-first search to construct a tree on \( G \). An edge \((u, v) \in E\) is in the tree if DFS finds either vertex \( u \) or \( v \) for the first time when exploring \((u, v)\). In addition to these tree edges, there are three other edge types that are determined by a DFS tree: forward edges, cross edges, and back edges. A forward edge is a non-tree edge from a vertex to one of its descendants. A cross edge is an edge from a vertex \( u \) to a vertex \( v \) such that the subtrees rooted at \( u \) and \( v \) are distinct. A back edge is an edge from a vertex to one of its ancestors. The graphic below depicts the four types of edges for a DFS tree that was initialized from vertex \( s \). Solid lines indicate tree edges.

![Figure 12.1: The Four Edge Types](image)

For DFS trees, edges can also be classified using the pre-order and post-order of their vertices. Recall that in DFS, the pre-order of a vertex is when it is pushed into the stack, and the post-order is when it is popped off the stack. For a given edge \((u, v)\), we have the following pre/post-orders for each type:

<table>
<thead>
<tr>
<th>Edge Type ((u, v))</th>
<th>Pre/Post-Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree/forward</td>
<td>pre((u)) &lt; pre((v)) &lt; post((v)) &lt; post((u))</td>
</tr>
<tr>
<td>Back</td>
<td>pre((v)) &lt; pre((u)) &lt; post((u)) &lt; post((v))</td>
</tr>
<tr>
<td>Cross</td>
<td>pre((v)) &lt; post((v)) &lt; pre((u)) &lt; post((u))</td>
</tr>
</tbody>
</table>

We will now show two applications of DFS: cycle-finding and topological sort.
12.2 Cycle Finding

Definition 12.1 A graph $G$ contains a cycle if there is a path in $G$ such that a vertex is reachable from itself. In other words, there is some some path $v_0, v_1, \cdots, v_k, v_0$ in $G$.

Claim 12.2 A graph $G$ has a cycle if and only if it has a back edge with respect to a DFS tree.

Proof: First, suppose that graph $G$ has a back edge $(u, v)$ with respect to a DFS tree on $G$. Then, by the definition of a back edge, we know that $v$ is an ancestor of $u$ in the DFS tree. Thus, there is a path of tree edges given by $v, v_1, \cdots, v_n, u$. We therefore have a path in $G$ given by $v, v_1, \cdots, v_n, u, v$, which is a cycle.

To prove the opposite direction is true, suppose that graph $G$ contains a cycle $v_1, \cdots, v_n, v_1$. Let $v_i$ be the first vertex that is visited by DFS on $G$. Then when $v_{i-1}$ is reached, $v_i$ will still be in the stack, so $(v_{i-1}, v_i)$ will be a back edge.

Altogether, we see that given a graph $G$, we can determine whether $G$ contains a cycle by running a slightly modified version of DFS. This algorithm will run in the same time as DFS, i.e. $O(n + m)$, where $|V| = n, |E| = m$.

Algorithm 1 DFS Cycle-Finding

Require: $G = (V, E)$ is a graph.
Ensure: Return True if $G$ contains a cycle, False otherwise

function FIND-CYCLE(G)
    for $u \in V$ do
        if DFS-Cycle($u, G$) then
            return True
        end if
    end for
    return False
end function

function DFS-Cycle($u, G$)
    Mark $u$ visited
    Mark $u$ in stack
    for $v \mid (u, v) \in E$ do
        if $v$ is in stack then
            return True
        end if
        if $v$ is not visited then
            if DFS-Cycle($v, G$) then
                return True
            end if
        end if
    end for
    Mark $u$ as not in stack
    return False
end function
12.3 Topological Sort

**Definition 12.3** Given a directed acyclic graph $G$, a topological sort on the vertices is an ordering such that all edges go from an earlier vertex to a later vertex.

**Claim 12.4** The inverse of the post-order values of DFS on $G$ will give a topological sort.

**Proof:** Recall that the post-order of DFS marks the vertices as they are popped from the stack. A vertex $v$ is only popped from the stack once all of its descendant vertices have been visited. Thus, if $v$ is an ancestor of $u$, it will be popped from the stack after $u$, and will thus have a higher post-order. So reversing the post-order will ensure ancestor vertices come before descendant vertices, so all edges lead from earlier vertices to later vertices.

Thus, to give a topological sort on graph $G$, simply run DFS, sort the vertices by their post-order values, and reverse them.

12.4 Breadth First Search

Breadth first search (BFS) is another possible way to traverse a graph. In BFS, upon visiting a vertex $v$, we visit all the neighbors of $v$ before we visit any other vertices. BFS can be implemented in a similar manner to DFS, but with use of a queue rather than a stack. Since vertices leave the queue in the same order that they enter it, there is no longer a distinct pre-order and post-order. Instead, the order that the vertices enter/leave the queue is referred to as the BFS order. Like DFS, we have to explore all edges and vertices in the graph and so BFS will run in $O(n + m)$ time. Pseudocode for BFS is below:

**Algorithm 2** Breadth-First Search

**Require:** $G = (V, E)$ is a graph.

**function** BFS($G$)

for $u \in V$ do

BFS-Visit($u, G$)

end for

end function

**function** BFS-Visit($u, G$)

Mark $u$ visited
Add $u$ to queue $Q$
while $Q$ is not empty do

$v \leftarrow$ head of $Q$
for $w \mid (v, w) \in E$ do

if $w$ not visited then

Mark $w$ visited
Add $w$ to $Q$
end if

end for

Remove $v$ from $Q$
end while

end function
As we did with DFS, we can use BFS to construct a tree on $G$. An edge $(u, v) \in E$ is in the tree if BFS finds either vertex $u$ or $v$ for the first time when exploring $(u, v)$. We now show that BFS can be used to find the shortest distance between two vertices in an unweighted graph.

**Claim 12.5** Given a vertex $u$ in unweighted graph $G$, a BFS tree rooted at $u$ contains the shortest path to any other vertex $v \in G$

**Proof:** We will prove this claim by induction. The inductive hypothesis is that BFS from $u$ visits all vertices of distance less than or equal to $t$ before it visits any vertices of distance at least $t + 1$. We see that for the base case $t = 1$, this is certainly true, as all of the immediate neighbors of $u$ added to queue in first step before any further vertices are explored.

Now, assume that the inductive hypothesis is true for $t = 1, 2, \ldots, k$, we wish to show it’s true for $t = k + 1$. Thus, we want to prove that BFS visits all vertices at distance $k + 1$ before visiting any at $k + 2$. Consider the time that the last vertex of distance $k$ is removed from the queue. If $v$ has a distance of $k + 1$, then there exists a vertex $w$ such that $(w, v)$ is an edge and the distance of $w$ is $k$. Since all vertices of distance $k$ have been processed, it must therefore be true that $v$ is in the queue. Similarly, by the inductive hypothesis, no vertices of distance $k + 1$ have been processed yet, so there can be no vertices of $k + 2$ in the queue. Thus BFS visits all vertices of distance $k + 1$ before $k + 2$, completing the proof by induction. ■