What is the running time of DFS? The loops on lines 1–3 and lines 5–7 of DFS take time $\Theta(V)$, exclusive of the time to execute the calls to DFS-VISIT. As we did for breadth-first search, we use aggregate analysis. The procedure DFS-VISIT is called exactly once for each vertex $v \in V$, since DFS-VISIT is invoked only on white vertices and the first thing it does is paint the vertex gray. During an execution of DFS-VISIT$(v)$, the loop on lines 4–7 is executed $|Adj[v]|$ times. Since

$$\sum_{v \in V} |Adj[v]| = \Theta(E) ,$$

the total cost of executing lines 4–7 of DFS-VISIT is $\Theta(E)$. The running time of DFS is therefore $\Theta(V + E)$.

### Properties of depth-first search

Depth-first search yields valuable information about the structure of a graph. Perhaps the most basic property of depth-first search is that the predecessor subgraph $G_\pi$ does indeed form a forest of trees, since the structure of the depth-first trees exactly mirrors the structure of recursive calls of DFS-VISIT. That is, $u = \pi[v]$ if and only if DFS-VISIT$(v)$ was called during a search of $u$'s adjacency list. Additionally, vertex $v$ is a descendant of vertex $u$ in the depth-first forest if and only if $v$ is discovered during the time in which $u$ is gray.

Another important property of depth-first search is that discovery and finishing times have ***parenthesis structure***. If we represent the discovery of vertex $u$ with a left parenthesis "$(u$" and represent its finishing by a right parenthesis "$u)$", then the history of discoveries and finishings makes a well-formed expression in the sense that the parentheses are properly nested. For example, the depth-first search of Figure 22.5(a) corresponds to the parenthesization shown in Figure 22.5(b). Another way of stating the condition of parenthesis structure is given in the following theorem.

***Theorem 22.7 (Parenthesis theorem)***
In any depth-first search of a (directed or undirected) graph $G = (V, E)$, for any two vertices $u$ and $v$, exactly one of the following three conditions holds:

- the intervals $[d[u], f[u]]$ and $[d[v], f[v]]$ are entirely disjoint, and neither $u$ nor $v$ is a descendant of the other in the depth-first forest,
- the interval $[d[u], f[u]]$ is contained entirely within the interval $[d[v], f[v]]$, and $u$ is a descendant of $v$ in a depth-first tree, or
- the interval $[d[v], f[v]]$ is contained entirely within the interval $[d[u], f[u]]$, and $v$ is a descendant of $u$ in a depth-first tree.

***Proof*** We begin with the case in which $d[u] < d[v]$. There are two subcases to consider, according to whether $d[v] < f[u]$ or not. The first subcase occurs when $d[v] < f[u]$, so $v$ was discovered while $u$ was still gray. This implies that $v$ is a descendant of $u$. Moreover, since $v$ was discovered more recently than $u$, all of its outgoing edges are explored, and $v$ is finished, before the search returns to and finishes $u$. In this case, therefore, the interval $[d[v], f[v]]$ is entirely contained within the interval $[d[u], f[u]]$. In the other subcase, $f[u] < d[v]$, and inequality (22.2) implies that the intervals $[d[u], f[u]]$ and $[d[v], f[v]]$ are disjoint. Because the intervals are disjoint, neither vertex was discovered while the other was gray, and so neither vertex is a descendant of the other.

The case in which $d[v] < d[u]$ is similar, with the roles of $u$ and $v$ reversed in the above argument. ■

***Corollary 22.8 (Nesting of descendants' intervals)***
Vertex $v$ is a proper descendant of vertex $u$ in the depth-first forest for a (directed or undirected) graph $G$ if and only if $d[u] < d[v] < f[v] < f[u]$.

***Proof*** Immediate from Theorem 22.7. ■

The next theorem gives another important characterization of when one vertex is a descendant of another in the depth-first forest.

***Theorem 22.9 (White-path theorem)***
In a depth-first forest of a (directed or undirected) graph $G = (V, E)$, vertex $v$ is a descendant of vertex $u$ if and only if at the time $d[u]$ that the search discovers $u$, vertex $v$ can be reached from $u$ along a path consisting entirely of white vertices.

***Proof*** $\Rightarrow$: Assume that $v$ is a descendant of $u$. Let $w$ be any vertex on the path between $u$ and $v$ in the depth-first tree, so that $w$ is a descendant of $u$. By Corollary 22.8, $d[u] < d[w]$, and so $w$ is white at time $d[u]$.

$\Leftarrow$: Suppose that vertex $v$ is reachable from $u$ along a path of white vertices at time $d[u]$, but $v$ does not become a descendant of $u$ in the depth-first tree. Without loss of generality, assume that every other vertex along the path becomes a descendant of $u$. (Otherwise, let $v$ be the closest vertex to $u$ along the path that doesn't become a descendant of $u$.) Let $w$ be the predecessor of $v$ in the path, so that $w$ is a descendant of $u$ ($w$ and $u$ may in fact be the same vertex) and, by Corollary 22.8, $f[w] \leq f[u]$. Note that $v$ must be discovered after $u$ is discovered, but before $w$ is finished. Therefore, $d[u] < d[v] < f[w] \leq f[u]$. Theorem 22.7 then implies that the interval $[d[v], f[v]]$ is contained entirely within the interval $[d[u], f[u]]$. By Corollary 22.8, $v$ must after all be a descendant of $u$. ■

### Classification of edges

Another interesting property of depth-first search is that the search can be used to classify the edges of the input graph $G = (V, E)$. This edge classification can be used to glean important information about a graph. For example, in the next section, we shall see that a directed graph is acyclic if and only if a depth-first search yields no "back" edges (Lemma 22.11).

We can define four edge types in terms of the depth-first forest $G_\pi$ produced by a depth-first search on $G$.

1. ***Tree edges*** are edges in the depth-first forest $G_\pi$. Edge $(u, v)$ is a tree edge if $v$ was first discovered by exploring edge $(u, v)$.
2. ***Back edges*** are those edges $(u, v)$ connecting a vertex $u$ to an ancestor $v$ in a depth-first tree. Self-loops, which may occur in directed graphs, are considered to be back edges.
3. ***Forward edges*** are those nontree edges $(u, v)$ connecting a vertex $u$ to a descendant $v$ in a depth-first tree.
4. ***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.

In Figures 22.4 and 22.5, edges are labeled to indicate their type. Figure 22.5(c) also shows how the graph of Figure 22.5(a) can be redrawn so that all tree and forward edges head downward in a depth-first tree and all back edges go up. Any graph can be redrawn in this fashion.

The DFS algorithm can be modified to classify edges as it encounters them. The key idea is that each edge $(u, v)$ can be classified by the color of the vertex $v$ that is reached when the edge is first explored (except that forward and cross edges are not distinguished):

1. WHITE indicates a tree edge,
2. GRAY indicates a back edge, and
3. BLACK indicates a forward or cross edge.

The first case is immediate from the specification of the algorithm. For the second case, observe that the gray vertices always form a linear chain of descendants corresponding to the stack of active DFS-VISIT invocations; the number of gray vertices is one more than the depth in the depth-first forest of the vertex most recently discovered. Exploration always proceeds from the deepest gray vertex, so an edge that reaches another gray vertex reaches an ancestor. The third case handles the remaining possibility; it can be shown that such an edge $(u, v)$ is a forward edge if $d[u] < d[v]$ and a cross edge if $d[u] > d[v]$. (See Exercise 22.3-4.)

In an undirected graph, there may be some ambiguity in the type classification, since $(u, v)$ and $(v, u)$ are really the same edge. In such a case, the edge is classified as the *first* type in the classification list that applies. Equivalently (see Exercise 22.3-5), the edge is classified according to whichever of $(u, v)$ or $(v, u)$ is encountered first during the execution of the algorithm.

We now show that forward and cross edges never occur in a depth-first search of an undirected graph.

***Theorem 22.10***
In a depth-first search of an undirected graph $G$, every edge of $G$ is either a tree edge or a back edge.

***Proof*** Let $(u, v)$ be an arbitrary edge of $G$, and suppose without loss of generality that $d[u] < d[v]$. Then, $v$ must be discovered and finished before we finish $u$ (while $u$ is gray), since $v$ is on $u$'s adjacency list. If the edge $(u, v)$ is explored first in the direction from $u$ to $v$, then $v$ is undiscovered (white) until that time, for otherwise we would have explored this edge already in the direction from $v$ to $u$. Thus, $(u, v)$ becomes a tree edge. If $(u, v)$ is explored first in the direction from $v$ to $u$, then $(u, v)$ is a back edge, since $u$ is still gray at the time the edge is first explored. ■

We shall see several applications of these theorems in the following sections.

## Exercises

***22.3-1***
Make a 3-by-3 chart with row and column labels WHITE, GRAY, and BLACK. In each cell $(i, j)$, indicate whether, at any point during a depth-first search of a directed graph, there can be an edge from a vertex of color $i$ to a vertex of color $j$. For each possible edge, indicate what edge types it can be. Make a second such chart for depth-first search of an undirected graph.

***22.3-2***
Show how depth-first search works on the graph of Figure 22.6. Assume that the **for** loop of lines 5–7 of the DFS procedure considers the vertices in alphabetical order, and assume that each adjacency list is ordered alphabetically. Show the discovery and finishing times for each vertex, and show the classification of each edge.

***22.3-3***
Show the parenthesis structure of the depth-first search shown in Figure 22.4.