to cause problems in practice, as we can usually use any depth-first search result effectively, with essentially equivalent results.

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 the vertex $u$ on which DFS-VISIT is invoked must be white and the first thing DFS-VISIT does is paint vertex $u$ gray. During an execution of DFS-VISIT($G$, $v$), the loop on lines 4–7 executes $|\text{Adj}[v]|$ times. Since 

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

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 $G_x$ 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, $u = v \pi$ if and only if DFS-VISIT($G$, $v$) 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 a 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). The following theorem provides another way to characterize the parenthesis structure.

**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 $[u.d, u.f]$ and $[v.d, v.f]$ are entirely disjoint, and neither $u$ nor $v$ is a descendant of the other in the depth-first forest,
- the interval $[u.d, u.f]$ is contained entirely within the interval $[v.d, v.f]$, and $u$ is a descendant of $v$ in a depth-first tree, or
- the interval $[v.d, v.f]$ is contained entirely within the interval $[u.d, u.f]$, and $v$ is a descendant of $u$ in a depth-first tree.
The following linear-time (i.e., $O(V + E)$-time) algorithm computes the strongly connected components of a directed graph $G = (V, E)$ using two depth-first searches, one on $G$ and one on $G^T$.

**STRONGLY-CONNECTED-COMPONENTS($G$)**

1. call DFS($G$) to compute finishing times $u.f$ for each vertex $u$
2. compute $G^T$
3. call DFS($G^T$), but in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1
4. output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component

The idea behind this algorithm comes from a key property of the component graph $G^{SCC} = (V_{SCC}, E_{SCC})$, which we define as follows. Suppose that $G$ has strongly connected components $C_1, C_2, \ldots, C_k$. Let $V_{SCC}$ be the set of vertices $v_i$ for each strongly connected component $C_i$ of $G$. There is an edge $(v_i, v_j) \in E_{SCC}$ if $G$ contains a directed edge $(x, y)$ for some $x \in C_i$ and some $y \in C_j$. Looked at another way, by contracting all edges whose incident vertices are within the same strongly connected component of $G$, the resulting graph is $G^{SCC}$. Figure 22.9(c) shows the component graph of the graph in Figure 22.9(a).

The key property is that the component graph is a dag, which the following lemma implies.

**Lemma 22.13**

Let $C$ and $C'$ be distinct strongly connected components in directed graph $G = (V, E)$, let $u, v \in C$, let $u', v' \in C'$, and suppose that $G$ contains a path $u \rightarrow u'$. Then $G$ cannot also contain a path $v' \rightarrow v$.

**Proof**
If $G$ contains a path $v' \rightarrow v$, then it contains paths $u \rightarrow u' \rightarrow v'$ and $v' \rightarrow v \rightarrow u$, thus $u$ and $v'$ are reachable from each other, thereby contradicting the assumption that $C$ and $C'$ are distinct strongly connected components.

Our algorithm for finding strongly connected components of a graph $G = (V, E)$ uses the transpose of $G$, which we defined in Exercise 22.1-3 to be the graph $G^T = (V, E^T)$, where $E^T = \{(v, u) : (v, u) \in E\}$. That is, $E^T$ consists of the edges of $G$ with their directions reversed. Given an adjacency-list representation of $G$, the time to create $G^T$ is $O(V + E)$. It is interesting to observe that $G$ and $G^T$ have exactly the same strongly connected components: $u$ and $v$ are reachable from each other in $G$ if and only if they are reachable from each other in $G^T$. Figure 22.9(b) shows the transpose of the graph in Figure 22.9(a), with the strongly connected components shaded.