

Chapter 10

Strongly Connected Components and Topological Sort.

10.1 Topological Sort

Definition 10.1.1. (connectivity)

1. Let $G = (V, E)$ be a non-directed graph. A **connected component** of G is a subset $U \subseteq V$ of maximal size in which there exists a path between every two vertices.
2. A non-directed graph G is said to be a **connected** graph if it only has one connected component.
3. Let $G = (V, E)$ be a directed graph. A **strongly connected component** of G is a subset $U \subseteq V$ of maximal size in which for any pair of vertices $u, v \in U$ there exist both directed path from u to v and a directed path from v to u .

10.1.1 Depth First Search (DFS)

As its name implies, depth-first search searches "deeper" in the graph whenever possible. Depth-first search explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it. Once all of v 's edges have been explored, the search "backtracks" to explore edges leaving the vertex from which v was discovered. This process continues until all vertices that are reachable from the original source vertex have been discovered. If any undiscovered vertices remain, then depth-first search selects one of them as a new source, repeating the search from that source. The algorithm repeats this entire process until it has discovered every vertex.

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_π does indeed form a forest of trees since

```

1 for  $v \in V$  do
2    $v$ .visited  $\leftarrow False$ 
3 end
4 time  $\leftarrow 1$ 
5 for  $v \in V$  do
6   if not  $v$ .visited then
7      $\pi(v) \leftarrow \text{null}$ 
8     Explore( $G, v$ )
9   end
10 end

```

Algorithm 1: DFS(G):

```

1 Previsit( $v$ ) for  $(v, u) \in E$  do
2   if not  $u$ .visited then
3      $\pi(u) \leftarrow v$ 
4     Explore( $G, u$ )
5   end
6 end
7 Postvisit( $v$ )

```

Algorithm 2: Explore(G, v):

```

1 pre( $v$ )  $\leftarrow$  time
2 time  $\leftarrow$  time + 1

```

Algorithm 3: Previsit(v):

```

1 post( $v$ )  $\leftarrow$  time
2 time  $\leftarrow$  time + 1

```

Algorithm 4: Postvisit(v):

the structure of the depth-first trees exactly mirrors the structure of recursive calls of explore-function. That is, $u = \pi(v)$ if and only if $\text{explore}(G, v)$ 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 finish times have a parenthesis structure. If the explore procedure were to print a left parenthesis " $(u$ " when it discovers vertex u and to print a right parenthesis " $)u$ " when it finishes u , then the printed expression would be well-formed in the sense that the parentheses are properly nested.

The following theorem provides another way to characterize the parenthesis structure.

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:

1. the intervals $[\text{pre}(u), \text{post}(u)]$ and $[\text{pre}(v), \text{post}(v)]$ are entirely disjoint, and neither u nor v is a descendant of the other in the depth-first forest.
2. the interval $[\text{pre}(u), \text{post}(u)]$ is contained entirely within the interval $[\text{pre}(v), \text{post}(v)]$, and u is a descendant of v in a depth-first tree, or
3. the interval $[\text{pre}(v), \text{post}(v)]$ is contained entirely within the interval $[\text{pre}(u), \text{post}(u)]$, and v is a descendant of u in a depth-first tree.

Proof. We begin with the case in which $\text{pre}(u) < \text{pre}(v)$. We consider two subcases, according to whether $\text{pre}(v) < \text{post}(u)$. The first subcase occurs when $\text{pre}(v) < \text{post}(u)$, so that v was discovered while u was still gray, which implies that v is a descendant of u . Moreover, since v was discovered after 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 $[\text{pre}(v), \text{post}(v)]$ is entirely contained within the interval $[\text{pre}(u), \text{post}(u)]$. In the other subcase, $\text{post}(u) < \text{pre}(v)$, and by definition, $\text{pre}(u) < \text{post}(u) < \text{pre}(v) < \text{post}(v)$, and thus the intervals $[\text{pre}(u), \text{post}(u)]$ and $[\text{pre}(v), \text{post}(v)]$ 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.

Corollary. 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 $\text{pre}(u) < \text{pre}(v) < \text{post}(v) < \text{post}(u)$.