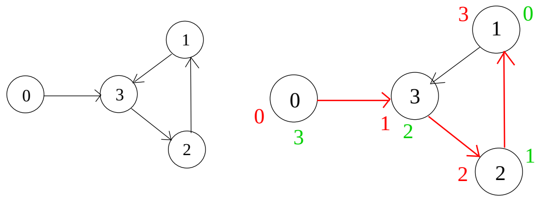


Figure 1: Illustrating dfs of the digraph mentioned below. Red edges are tree edges, red numbers are preorder ranks, and green numbers are postorder ranks.

17 Depth-first search

Depth-first search of a directed graph is a way to traverse the graph which reveals many important properties of the digraph. For example, it can be used to topologically sort the digraph (and detect cycles).

There is a routine

```
void full_dfs ( u ) // the real version needs more info
{
  int u;
  for (u=0; u < digraph->n; ++u)
    if ( u has not already been visited )
      dfs(u);
}
```

And

```
dfs (u) // more like pseudocode.  Pre_count etcetera
        // are passed in a ‘digraph_extra.’  See
        // the sample code.
{
  pre_rank[u] = pre_count; ++ pre_count;
  for all out-edges (u,v)
  {
    if pre_rank[v] < 0   // if v not already visited
    {
      parent[v] = u;
      dfs ( v );
    }
  }
  post_rank[u] = post_count; ++ post_count;
}
```

`Full_dfs()` constructs a forest. Vertices with parent -1 are the roots of the trees in the forest. If a vertex v has a parent u, then (u, v) is an edge of the digraph.

This is called a *depth-first spanning forest.*

The Figure shows the results of a depth-first search on the following digraph:

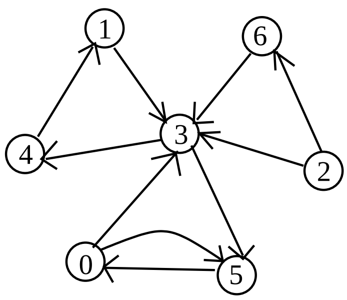
```
% a.out < cyclic-2
4 4
0 1 3
1 1 3
2 1 1
3 1 2
   digraph extras, 4 vertices
   vertex    parent  pre_rank post_rank
        0        -1         0         3
        1         2         3         0
        2         3         2         1
        3         0         1         2
%
```

17.1 Nested subroutine calls.

Here is another example, for which depth-first search produces the given result.

```
7 10
0 2 3 5
1 1 3
2 2 3 6
3 2 4 5
4 1 1
5 1 0
6 1 3
   digraph extras, 7 vertices
   vertex    parent  pre_rank post_rank
        0        -1         0         3
        1         4         3         0
        2        -1         4         4
        3         0         1         2
        4         3         2         1
        5        -1         5         5
        6        -1         6         6
```

Figure:



Here is a trace of the full dfs of the above graph; it gives the sequence in which each call to `dfs` begins and ends. Observe the nesting property![1]

```
7 10
0 2 3 5
1 1 3
2 2 3 6
3 2 4 5
4 1 1
5 1 0
6 1 3
```

```
  dfs(0) begins
    dfs(3) begins
      dfs(4) begins
        dfs(1) begins
        dfs(1) ends
      dfs(4) ends
      dfs(5) begins
      dfs(5) ends
    dfs(3) ends
  dfs(0) ends
  dfs(2) begins
    dfs(6) begins
    dfs(6) ends
  dfs(2) ends
```

```
   digraph extras, 7 vertices
   vertex    parent  pre_rank post_rank
        0        -1         0         4
        1         4         3         0
        2        -1         5         6
        3         0         1         3
        4         3         2         1
        5         3         4         2
        6         2         6         5
```

- Speaking of the sequence of actions developing over time, for every vertex u, `dfs(`u`)` is 'active' over a time-span (t_0, t_1) where it begins and ends, respectively.
- If `dfs(u)` spans (t_0, t_1), $t_0 < t_1$, and `dfs(v)` spans another interval (t_2, t_3), $t_2 < t_3$, then these intervals cannot 'interlace.' That is, only the following are possible:

[1] This is a corrected version.

- Disjoint time-spans

$$\text{(i)} t_0 < t_1 < t_2 < t_3 \quad \text{or}$$
$$\text{(ii)} t_2 < t_3 < t_0 < t_1$$

and nested time-spans

$$\text{(iii)} t_0 < t_2 < t_3 < t_1 \quad \text{or}$$
$$\text{(iv)} t_2 < t_0 < t_1 < t_3$$

(17.1) Lemma *These nesting relations determine the structure of the depth-first forest as follows.*

(i) *dfs(u) ends before dfs (v) begins. We would think of u being to the left of v in the depth-first forest; that could be made precise.*

(ii) *Or v is to the left of u.*

(iii) *The time-span of dfs(u) encloses that of dfs(v).* Important: u is an ancestor of v in the depth-first forest.

(iv) *or v is ancestor of u.*

(v) *The preorder ranks follow the time sequence in which* `dfs()` *is initiated. That is,* `pre_rank[u] < pre_rank[v]` *if and only if* `dfs(u)` *begins before* `dfs(v)`.

(vi). The postorder ranks follow the time sequence in which `dfs()` *ends*, so `post_rank[u] < post_rank[v]` if and only if `dfs(u)` ends before `dfs(v)` ends.

(vii). **Ancestorhood.** `u` is an ancestor of `v` in the depth-first forest if and only if both (a) `pre_rank(u) < pre_rank(v)` and `post_rank(u) > post_rank(v)`. ∎

The most important result is the following

(17.2) Theorem *In a directed graph G, the vertices which are descendants of a vertex u in the depth-first spanning forest are precisely those vertices v which can be reached from u in the deleted graph*

$$G \backslash \{u_1, \ldots, u_k\}$$

where $u_1, \ldots, u_k$ are the vertices preceding u in preorder. ∎

Here is a corollary.

(17.3) Lemma *If G is an acyclic directed graph subjected to a full* `dfs`*, then after completion, reverse postorder is a topological order on G.*

Proof. Suppose otherwise, that is, G is acyclic, but there exists an edge (u, v) where u precedes v in postorder.

Claim that u cannot precede v in preorder. This is because the vertex v is certainly reachable from u in the deleted graph — because (u, v) is an edge — and v must then be a descendant of u. In that case u follows v in postorder, which contradicts the assumption.

So u follows v in preorder and u precedes v in postorder. This makes u a descendant of v, so there is a path from v to u. But the edge (u, v) completes a cycle and G is not acyclic, contradicting the assumption. ∎