We can classify the edges in 4 classes, \( \mathcal{C}, \mathcal{B}, \mathcal{F}, \mathcal{T} \):

- **Cross edges**
  - All other edges, i.e., in the subtree and neither \( m \) nor \( m \) is not visited.
  - \( \mathcal{C} \)

- **Backward edges**
  - \( m \) visits \( m \) where \( m \) already visited and \( m \) not visited.
  - \( \mathcal{B} \)

- **Forward edges**
  - \( m \) visits \( m \) where \( m \) already visited and \( m \) not visited.
  - \( \mathcal{F} \)

- **Tree edges**
  - \( \mathcal{T} \)

We can classify these edges in 4 classes, \( \mathcal{C}, \mathcal{B}, \mathcal{F}, \mathcal{T} \):

### DFS for Directed Graphs

#### Procedure DFS (Vertex \( v \)) (cont.)

- **Call DFS** (a) inspects all edges and vertices reachable from \( v \).

#### Procedure DFS (Graph \( G \)) (a)

- If not visited \( |E| \) then DFS

#### Procedure Depth First Search (Graph \( G \)) (a)

- For each \( v \) do

#### Procedure DFSandStronglyConnectedComponentsforDirectedGraphs

- Consider DFS, recursive formulation, for a directed graph \( G = (\mathcal{E}, \mathcal{V}) \), and

#### Procedure DFS (Graph \( G \)) (a)

- If not visited \( |E| \) then DFS

#### Procedure DFSandStronglyConnectedComponentsforDirectedGraphs

- Consider DFS, recursive formulation, for a directed graph \( G = (\mathcal{E}, \mathcal{V}) \), and
Proof: see [Common/Leiserson/Freyd/Real, chapter 23.5]

1. Compute strongly connected components for each A ∈ Λ.
   - Let \( A \) be a strongly connected component.
   - For each \( (a' \in A, a) \), add \( a' \) to \( A \).
   - Repeat until no new vertices are added.

2. Compute the transposed graph \( G^t \).
3. Compute strongly connected components for each \( A \) in \( G^t \).
4. Output the vertices of each tree in the DFS forest of step 3.

Note: \( G \) and \( G^t \) have the same SCCs.

\( G \) (adjacency list) can be computed from \( G \) in time \( O(n + \mid A \mid) \).
\( G^t \) (transpose) can be computed from \( G \) in time \( O(\mid A \mid) \).

The SCCs can be computed in time \( \mid A \mid + O(\mid A \mid) \) by an extension of DFS.

The maximal (wrt. set inclusion) strongly connected subgraphs of \( G \) are the strongly connected components (SCC) of \( G \).

A directed graph \( G = (\Lambda, E) \) is strongly connected iff for all \( A \in \Lambda \), \( \Lambda \rightarrow A \).

An application of DFS.

**Strongly Connected Components of a Directed Graph**

**Lemma 1**: DFS needs time \( O(n + \mid A \mid) \).

**Lemma 2**: Allows algorithmic classification from \( \text{dfsnum} \) and \( \text{compnum} \).

- For all \( (z', a) \) holds: \( \text{dfsnum} > \text{compnum} \).
- For all \( (z', z) \) holds: \( \text{dfsnum} > \text{compnum} \).
- For all \( (z', z) \) holds: \( \text{dfsnum} > \text{compnum} \).
- For all \( (z', z) \) holds: \( \text{dfsnum} > \text{compnum} \).

**Properties of the extended algorithm**

- \( T \) is the original DFS tree.
- \( \text{comp} \) is a partition of \( \text{dfsnum} \).

**Application of SCCs**

- \( \{y\} = \{a\} \)
- \( \{p\} = \{a\} \)
- \( \{^\prime\} = \{a\} \)
- \( \{q\} = \{a\} \)
- \( \{v\} = \{a\} \)

**Low-level code**

Implementing loops in DFS.