DFS with more info.

- We start by exploring a node
- Finish when we have explored all nodes of the subtree.
- Keep a counter to track the number of steps taken.

**Example:**

```
1 ← 4 ← 6 ← 10
   ↓      ↓      ↓
  2 → 3 → 5 ← 11
   ↓      ↓      ↓
  9 ← 8 ← 7 ← 12
```

```
1 - 1 118
2 - 2 117
3 - 3 116
4 - 4/5
5 - 6 115
6 - 7/8
7 - 9 114
8 - 10/13
9 - 11/12
10 - 19/24
11 - 20/23
12 - 21/22
```
DFS(node)
    if color[node] = white:
        color[node] = grey
        discovery[node] = time
        time += 1
        for u in node:
            if color[u] = white
                DFS(u)
        color[node] = black
        finish[node] = time
        time += 1

This still linear. in (n+m)

This can classify all types of edges.

1) Forward edge a -> b.
   \[ d[a] < d[b] < f[b] < f[a] \]

2) Back edge a -> b.
   \[ d[b] \leq d[a] \leq f[a] \leq f[b] \]

3) Cross edge a -> b
   \[ d[b] < f[b] < d[a] < f[a] \]
Reachability & Connectivity

Strongly connected relation

\[ u \sim v \text{ if } u \text{ reachable from } v \text{ & vice versa.} \]

This is an equivalence relation

\[ \Rightarrow \exists \text{ equivalence classes} \]

These classes are called the Strongly Connected components (SCC's) of \( G \).

\[ \text{E.g.} \]

![Diagram](image_url)
The SCC graph (Component graph)

\[ \text{sgc} = (V, E) \]

The vertices are the distinct equivalence classes of \( \text{sgc} \rightarrow [1], [2], \ldots, [n] \)

- the distinct elements only.

Connect \([i] \rightarrow [j]\) if \(\exists u \in [i] \land v \in [j] \land u \rightarrow v \in E\)

**Claim**: \(\text{sgc}\) is acyclic.

**Proof**: if \([v_1] \rightarrow [v_2] \rightarrow \ldots \rightarrow [v_k] \rightarrow [v_1]\)

is a cycle we can collapse it \(\Rightarrow \notin E\)

**Applications**

1. Given a graph \(G\) with some coins at each vertex, \(c_i\), how many coins can we collect starting from \(S.1\) and ending back at \(S\).1) and ending wherever we want.
First solve the problem assuming $g$ is a DAG.

1) No way to reach back to $s \Rightarrow$ answer = $C_s$.
2) First topologically sort the graph
   
   ![Diagram]

For each vertex $i$, the maximum coins collected can be seen as the max possible depending on the next step.

$\Rightarrow$ maxCoins($i$) = $C_i + \max_{\text{nbhd of } i} \maxCoins(l_i)$

We can solve this using DP.

Now for a general graph $g$

Construct the $g'$.

1) if we want to return to $s$, we cannot leave $[s]$, $\Rightarrow$ answer = $\sum_{i \in [s]} C_i$
2) Do the previous algorithm on $g'$. and treat $DAGr$. 

[2]
Gr. \textit{Def.} $G$ is weakly connected iff.

+ pairs $u, v$ either $u$ is reachable from $v$ or vice versa.

Given a graph $G$, is it weakly connected?

First solve it for a DAG.

Claim \textit{iff} in the topsort ordering of $G$

$v_1, v_2, \ldots, v_n$ if $v_i \rightarrow v_{i+1} \& i < n$

Proof \textit{if} this happens $\Rightarrow$ it is weakly connected

0/w there are two vertices of indegree 0.

$G$ both are unreachable from each other \(\blacksquare\)

Now for a general graph $G$.

Construct $G^{\infty}$ & use the DAG algorithm.

developed above.
Kosaraju - Sharir Algo

Observation: If we reverse the edges of $G$, it has the same SCC's. (transpose of $G$: $G^t$).

Algo:

1) Do a DFS on $G$, to get the DFS forest. Order the vertices in decreasing order of finishing time: $v_1, \ldots, v_n$

2) Do a DFS on $G^t$, but process the nodes for roots in decreasing order $v_1, \ldots, v_n$

E.g. \[ 1 \xleftarrow{1} 116 \rightarrow 2 \xleftarrow{2} 115 \]

\[ 3 \xrightarrow{3} 4 \xleftarrow{4} 413 \rightarrow 5, 7 \xrightarrow{5, 7} 10 \]

\[ 5 \xleftarrow{5} 6 \xrightarrow{6} 7 \xleftarrow{7} 8 \xrightarrow{8} 9 \]

Full tree
\[
\begin{align*}
1 & \rightarrow 2^{2/3} \\
5/103 & \rightarrow 4 \\
6/9 & \rightarrow 7/18 \\
5/13/14 & \rightarrow 8 \\
11/16 & \rightarrow 12/15
\end{align*}
\]

Proof:

**Def.** \( \text{finish}(E[V]) = \max_{u \in E[V]} \text{finish}(u) \)

**Claim** if \([x] \rightarrow [y]\) is an edge in \(G_{\text{sc}}\) then \(f([x]) > f([y])\)

**Proof Case 1.**

During \(BFS\) DFS we first hit an element in \([x]\) before any element in \([y]\).

Then before finishing \([x]\) we will fully explore \([y]\) \( \Rightarrow f([x]) < f([y]) \).
Case 2. we pick an element $u \in [Y]

\[
\begin{array}{c}
\mathcal{G} \\
\mathcal{X} \\
\mathcal{Y}
\end{array}
\]

we will finish exploring $[Y]$ before even touching $[X] \Rightarrow f([X]) > f([Y])$

The SCC of $g_j$, $g_j^T$ will be of the form

\[g_j: \begin{array}{c}
[V_1] \\
[V_2] \\
[V_3] \\
\vdots \\
[V_k]
\end{array} \quad \begin{array}{c}
\rightarrow \\
\rightarrow \\
\rightarrow \\
\rightarrow
\end{array} \quad \begin{array}{c}
[V_1] \\
[V_2] \\
[V_3] \\
\vdots \\
[V_k]
\end{array}\]

\[g_j^T: \begin{array}{c}
[V_1] \\
[V_2] \\
[V_3] \\
\vdots \\
[V_k]
\end{array} \quad \begin{array}{c}
\leftarrow \\
\leftarrow \\
\leftarrow \\
\leftarrow
\end{array} \quad \begin{array}{c}
[V_1] \\
[V_2] \\
[V_3] \\
\vdots \\
[V_k]
\end{array}\]

we will have that

\[f([V_i]) < f([V_j]) \quad \forall \; i > j\]

\Rightarrow \text{when we explore vertices in } g_j^T \text{ we will first explore a vertex in } [V_1].

(\text{as the maximum finish time will be in } [V_1])
which will give the first tree as \([V_1, J]\).

Then the next vertex, that is not black yet, will be in \([V_2]\), as the DFS in \([V_1]\) makes it fully black.

\[\text{[V_1]} \leftarrow \text{[V_2]} \quad \text{[V_3]} \quad \ldots \quad \text{[V_k]}\]

\Rightarrow \text{the second tree in the forest will be [V_2].}

\Rightarrow \text{by induction the trees will be [V_1, V_2, ... , V_k]}.

Giving us the SCC-DAG.