Depth First Search on Directed Graphs

**order of exploration**
- back edge - from vertex to ancestor
- forward edge - from vertex to descendant
- cross edges

\[ d(v), d(z), d(w), d(w), f(w), f(w), d(y), f(y), f(z), f(r), d(s), \ldots \]

**DFS**

- mark \( v \) \(<\) discovered
- \( d(v) \leftarrow \) time; \( \text{time} \leftarrow \text{time} + 1 \) \(*\) discover time
- for each neighbour \( u \) of \( v \) do
  - if \( u \) is undiscovered then
    - DFS\((u); \ (v, u) \) is a tree edge.
  - else
  - mark \( u \) \(<\) finished
- \( f(v) \leftarrow \) time; \( \text{time} \leftarrow \text{time} + 1 \) \(*\) finish time

\[ \bigcirc \] label back, forward, cross edges
- if \( u \) not finished then \((v, u)\) is back edge
- else if \( d(u) > d(v) \) then \((v, u)\) is forward edge
- else if \( d(u) < d(v) \) then \((v, u)\) is cross edge

DFS takes \( O(n + m) \)

Note that result depends on vertex ordering.
Applications of DFS

1. detecting cycles in directed graphs

Lemma. A directed graph has a [directed] cycle iff DFS has a back edge

Proof

\[ \iff \quad \text{back edge gives directed cycle} \]

Suppose there is a directed cycle

Let \( v_1 \) be first vertex discovered in DFS.

Number vertices of cycle \( v_1 \ldots v_k \)

Claim \((v_k, v_1)\) is a back edge

If Because we must discover & explore all \( v_i \) before we finish \( v_i \)

when we test edge \((v_k, v_1)\)

we label it a back edge.
Applications of DFS

2. Topological sort of directed acyclic graph

Edge \((a,b)\) means \(a\) must come before \(b\) (e.g. job scheduling)

Find a linear order of vertices satisfying all edges. (possible iff no directed cycle).

Example

\[\begin{array}{ccc}
& b & \\
\text{c} & \rightarrow & a \\
& d & \\
\end{array}\]

Topological sort: \(b, c, a, d\) or \(c, d, b, a\) or \ldots

One solution: find vertex \(v\) with no in-edge (= source)
Remove \(v\) and repeat.

Ex: Do this in \(O(n+m)\) time.

Solution using DFS: (also \(O(n+m)\)
Use reverse of finish order.

Example

(first ex, minus back edge)

Finish order

Reverse finish order \(s, w, z, r, x, y, u\).
This is a topological order.
Claim. For every directed edge \((u, v)\),
\[\text{finish}(v) < \text{finish}(u)\]

Proof. Case 1. \(u\) discovered before \(v\)
Then \(v\) is discovered and finished before \(u\) is finished.

Case 2. \(v\) discovered before \(u\)
Because \(G\) has no directed cycle, we can't reach \(u\) in DFS\((v)\). So \(v\) finished before \(u\) is discovered and finished.
3. Finding strongly connected components in a directed graph.

strongly connected = for all vertices \( u, v \),
there is a path \( u \rightarrow v \) and a path \( v \rightarrow u \).

Easy to test if \( G \) is strongly connected because we don't need to test all pairs \( u, v \).

Let \( s \) be a vertex.

Claim. \( G \) is strongly connected iff for all vertices \( v \),
there is a path \( s \rightarrow v \) and a path \( v \rightarrow s \).

pf. \( \equiv \) clear

\[ \iff \text{ to get from } u \rightarrow u \text{ and } u \rightarrow s \rightarrow v \]

To test if there's a path \( s \rightarrow v \) \& \( v \rightarrow u \) -- do DFS \( (s) \).

How can we test if there's a path \( v \rightarrow s \) \& \( u \) ?
Reverse edge directions and do DFS \( (s) \).

More generally, structure of a digraph is

![Diagram of strongly connected components]

Contracting strongly conn. components gives acyclic graph
(think about the reason)
How to find strongly connected components.

History: Tarjan '72, Kosaraju '80's, Gabow '99

All linear time and all simple, but different.

We'll do Kosaraju's method.

Idea: Vertices 1...n

Run DFS (vertex ordering resolves what vertex comes next)

Let finish order be \( f_1 \ f_2 \ldots \ f_n \)

\( G^R = G \) with all edges reversed

run DFS on \( G^R \) with vertex order \( f_n \ f_{n-1} \ldots \ f_1 \)

Lemma: Trees in 2nd DFS are exactly the strongly connected components.

For pseudocode, see CLR.

Run time is \( O(n+m) \)

Example

If first DFS started at 2

last finished vertex still on left. So same 2 trees.
Idea of why this works:

If DFS of $G$ starts at $v$ in $C_1$ then it reaches $C_1 \cup C_2$ and $v$ is last finished.

If DFS of $G$ starts at $v$ in $C_2$ then it reaches all of $C_2$ and new DFS tree starts at $C_1$. So last finished vertex is in $C_1$.

In $G^R$:

DFS starts in $C_1$ — so it does not reach $C_2$ — need a new tree for $C_2$. 
**Formal Proof of Lemma**

Must prove: vertices \( u, v \) are strongly connected iff they are in the same tree of DFS of \( G^R \)

\( \Rightarrow \) Suppose without loss of generality \( u \) discovered first in DFS of \( G^R \). Then, since there is a \( u \rightarrow v \) path in \( G^R \), \( v \) is discovered before \( u \) is finished.

\( \Leftarrow \) Suppose \( u, v \) in same tree of DFS \( G^R \). Let \( r = \text{root} \).

Claim: \( r \) and \( u \) are strongly connected.

Then, by same argument, \( r \) and \( v \) are strongly connected. Therefore \( u \) and \( v \) are too.

Pf. of claim: \( r \) is root of tree containing \( u \).

So \( \exists \) path \( r \rightarrow u \) in \( G^R \), i.e. a path \( u \rightarrow r \) in \( G \).

We must show \( \exists \) path \( r \rightarrow u \) in \( G \),

When we started the tree rooted at \( r \), \( u \) was undiscovered too. Why did we pick \( r \)? It had a higher finish time in DFS of \( G \),

\[ u \xrightarrow{r} \text{ finished later} \]

In DFS of \( G \):

If \( u \) discovered before \( r \), then \( r \) is discovered and finished before \( u \) is finished. Contra.

So \( r \) is discovered before \( u \) but finished later.

Then \( u \) is in \( r \)'s tree so \( \exists \) path \( r \rightarrow u \) \( \square \)