**Recall**

**Depth First Search** — a bold search where we go as far as possible until there’s nothing new to explore, and then back up to find something new.

**Adjacency lists**
- 1: 2, 3, 4
- 2: 5, 7, 4, 3, 1
- 3: 1, 4, 2
- 4: 3, 1, 2
- 5: 2, 6, 7
- 6: 5
- 7: 5, 2

**DFS tree**

Order in which vertices are discovered:
1, 2, 5, 6, 7, 4, 3

Order in which vertices are finished:
6, 7, 5, 3, 4, 2, 1

There are tree edges and **back** edges (from descendant to ancestor).
DFS for a directed graph

Note: we’re using letters for the vertex names just for our convenience.
(Recall that vertices are always 1 . . n.)
DFS for a directed graph

There are 4 kinds of edges $(v,u)$:
- **tree edge**
- **back edge** - $u$ is an ancestor of $v$ in the DFS tree
- **forward edge** - $u$ is a descendant (not a child)
- **cross edge** - none of the above

How can we test?
- $u$ not discovered: $u$ is not finished
- $u$ is finished: $d(u) > d(v)$
- $u$ is finished: $d(u) < d(v)$
DFS for a directed graph

d(v) = discover time  f(v) = finish time

\[
\text{DFS}(v) \\
\text{mark}(v) := \text{discovered} \\
d(v) := \text{time}; \text{time} := \text{time} + 1 \\
\text{for each vertex } u \text{ in } \text{Adjacency}(v) \\
\text{if } u \text{ is undiscovered then} \\
\quad (v, u) \text{ is a tree edge} \\
\quad \text{DFS}(u) \\
\text{else} * \\
\text{mark}(v) := \text{finished} \\
f(v) := \text{time}; \text{time} := \text{time} + 1
\]

the blue part is new for directed graphs

*  # label back, forward, cross edges
if u not finished then (v, u) is a back edge
else if d(u) > d(v) then (v, u) is a forward edge
else if d(u) < d(v) then (v, u) is a cross edge

DFS  # to handle disconnected graphs
initialize: mark every vertex undiscovered, unfinished
initialize time := 1
for each vertex v
\quad if v is undiscovered then \\
\quad \# start a new tree rooted at v \\
\quad \text{DFS}(v)

Note that the result depends on a vertex order.

DFS takes O(n+m) time.
Applications of DFS (for directed graphs)

1. detecting cycles in a directed graph

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

**Proof.**

\[\begin{array}{c}
\text{a back edge gives a cycle} \\
\Rightarrow \\
\text{Suppose \textbf{I} [directed] cycle} \\
\text{Let } v_i \text{ be first discovered vertex} \\
\text{Number vertices of cycle} \\
\quad v_i, v_2, \ldots, v_k \\
\text{Claim edge } (v_k, v_i) \text{ is a back edge.} \\
\text{Pf. we discover (and call DFS on) all } v_i \text{ before we finish } v_i. \\
\text{When we test edge } (v_5, v_i), v_i \text{ is discovered, not finished} \\
\text{so label } (v_5, v_i) \text{ as back edge.}
\end{array}\]
Applications of DFS (for directed graphs)

2. topological sort of a directed acyclic graph

**acyclic** means no directed cycle.

What is a **topological sort**?

an ordering of the vertices of the graph such that if (u,v) is an edge then u comes before v.

Such an ordering exists iff the graph has no cycle.

Useful for, e.g., job scheduling, precedence constraints.

**Example.**

One way to find a topological sort: Find a vertex v with no in-edges, put v next in the ordering, delete v, repeat.

**Exercise.** Do this in $O(n+m)$ time.
Applications of DFS (for directed graphs)

2. topological sort of a directed acyclic graph

Finding a topological sort using DFS
Use reverse of finish order! O(n+m)

Example.

This is the first example but minus the back edge so it’s acyclic

Why does this work?

Claim. Every edge (v,u) goes from high finish to low finish, i.e. f(v) > f(u).
Example.

This is the first example but minus the back edge so it's acyclic

Why does this work?

Claim. Every edge \((v,u)\) goes from high finish to low finish, i.e. \(f(v) > f(u)\).

Proof. Consider this edge when we label it

- \((v,u)\) is tree edge - ✓
- forward edge ? by rule for non-tree edges,
- cross edge ? \(u\) is finished, \(v\) is not
- back edge - there aren't any!
Applications of DFS (for directed graphs)

3. finding strongly connected components in a directed graph

**strongly connected** = for all vertices u, v, there is a path u to v and a path v to u.

**Testing strong connectivity in linear time** $O(n+m)$

Idea 1: we don’t need to test all pairs u,v:

**Claim.** Let $s$ be a vertex. $G$ is strongly connected iff for all vertices $v$, there is a path from $s$ to $v$ and a path from $v$ to $s$.

**Proof.**

$$
\begin{align*}
\Rightarrow & \quad \checkmark \\
\Leftarrow & \quad \text{to get from } u \text{ to } v: \\
& \quad \text{path } u \text{ to } s + \text{path } s \text{ to } v.
\end{align*}
$$

How do we test if there’s a path from $s$ to $v$ for all $v$?

*do* $\text{DFS}(s)$

How do we test if there’s a path from $v$ to $s$ for all $v$?

*Reverse all edge directions and DFS(s).* Neat!
Applications of DFS (for directed graphs)

3. finding strongly connected components in a directed graph

The structure of a digraph

Contracting the strongly connected components gives an acyclic graph

Find strongly connected components in linear time $O(n+m)$
Find strongly connected components in linear time $O(n+m)$

Run DFS (use vertex ordering $1,2,\ldots,n$ to iterate through vertices)

Let finish order be $f_1, f_2, \ldots, f_n$

Let $G^R = G$ with all edge directions reversed

Run DFS on $G^R$ using vertex order $f_n,\ldots, f_2, f_1$ to iterate through vertices

Example

If first DFS started at vertex 2

1st DFS

start

last finished

So 2nd DFS will behave as here and find same 2 strongly connected components.
Find strongly connected components in linear time $O(n+m)$

Run DFS (use vertex ordering $1, 2, \ldots, n$ to iterate through vertices)
Let finish order be $f_1, f_2, \ldots, f_n$
Let $G^R = G$ with all edge directions reversed
Run DFS on $G^R$ using vertex order $f_n, \ldots, f_2, f_1$ to iterate through vertices

Pseudocode can be found in CLRS.
Runtime is $O(n+m)$.

**Lemma.** The trees found by the second DFS are exactly the strongly connected components.

**Proof.** Just the idea.

1. If first DFS starts in $C_1$, then DFS reaches $C_1 \cup C_2$ and $f_n$ lies in $C_1$ ($f_n =$ start vertex)
2. So 2nd DFS in $G^R$ starts in $C_1$ and does not reach $C_2$. So we find $C_1$ and $C_2$ as 2 components.

2. If first DFS starts in $C_2$ then it reaches $C_2$ and we start a new tree for $C_1$, so $f_n$ lies in $C_1$. So ✗ again.
Summary of Lecture 14

- Depth First Search for directed graphs, back/forward/cross edges
- DFS to test acyclic, find topological order, find strongly connected components

What you should know from Lecture 14:

- how to do DFS on a directed graph, and identify types of edges
- applications of DFS on directed graphs
  - cycles, topological order (and how to find it)
  - how to test strongly connected
  - appreciate strongly connected components

Next:

- more graph algorithms: minimum spanning trees and shortest paths