Directed $G = (V, E)$.

**Strong component**: maximal subset of vertices s.t. $u$ can reach $v$ for each pair in subset.

Contract each component to get **strong component graph** $\text{sc}(G)$ is a DAG.
source: in-degree = 0
sink: out-degree = 0

If we dfs from a vertex of a sink component, we'll mark exactly the vertices of that component.

**Algorithm idea:**

Somehow find a vertex $v$ in a sink component $C$.

dfs($v$) to mark $C$ "delete" $C$ & recurse

$O(|V| + |E|)$ time in addition
to time to find each vertex v.
Kosaraju & Sharir.

Claim: Last vertex of a postorder lies in a source component.

But we want $v$ in a sink...

$$\text{scc(rev}(G)) = \text{rev(scc}(G))$$

So last vertex in postorder of $\text{rev}(G)$ is in a sink of $\text{scc}(G)$. 
Stronger claim: a reverse postorder of $G$ first touch strong components in top. order of $scc(G)$.

Final algorithm:
Compute postorder of $\text{rev}(G)$, putting vertices in a stack.

Do another dfsAll of $G$, popping from stack for the outer loop.
Each top level call to
strong component. $O(M+|E|)$ time
Problem: Have $n$ strings over an alphabet $\Sigma$.

Want to store them and support insert, remove, contains, predecessor balanced.

One solution: BST using the strings as the elements.

$\mathcal{O}(\log n)$ comparisons per operation.

But comparisons take $\mathcal{O}(\min \{r,s,3\})$ time for strings of length $r+s$. 
M: max string length.
Each operation takes \(O(M \log n)\) time.

Trie (short for re-trie-val)

A rooted tree.
Each node has a boolean \(t\) and an array of links to children indexed by members of \(\Sigma\).
Each node \(x\) corresponds to a string \(w\) equal to links you
use to reach x.

Boolean is true if w is a member of collection.
If $|E| = O(1)$, a search takes $O(1w)$ time.

**insert:** Do a search adding nodes not already there. Set Boolean to true at last node.

**remove:** Set Boolean to false. While current node is false and has no children, remove node & set current to parent.
Has \( N = M_n \) nodes + uses \( \Theta(N, 1) = O(M_n, 1) \) space.

Bad example.
Patricia trio (radix tries):

If a node has one child, merge node with its parent.

Links come with the substring that got merged in.

Still $O(Mn)$ space for strings, but now $\leq 2n-1$ nodes (all non-leaves have $\geq 2$ children)
$O(M_n + n|\Sigma|)$ space

(inserts may split edge strings)
String matching:

Given a string $T$ and a non-empty pattern string $P$.

Goal: Find all positions where $P$ appears in $T$.

One solution: For each character $T_i$, check if $P$ starts there. $O(|P| \cdot |T|)$ time.

Knuth-Morris-Pratt:

Build a data structure of size $O(|P|)$ in $O(|P|)$ time. Can use structure to
search $T$ in $O(|T|)$ additional time.

$O(|P| + |T|)$ total
Multi-string searching:

Given $T$ + $k$ patterns $P_1, \ldots, P_k$.

Want to know if any appear in $T$.

$m = |T|$

$n := |P_1| + \ldots + |P_k|$

Assume $|E| = O(1)$

"obvious": run KMP once per pattern.

$O(km + n)$ time

scan $\frac{n}{k}$ times \[ \text{build data structures} \]
Goal for Monday: $O(m+n)$ time.