Recap of this week’s lectures:

- DFS: topological sort, cycle detection, biconnected and strongly connected components

### SCC Problem

We are given a directed graph $G$ with $N$ nodes and $M$ edges. Each node $v$ has a treasure chest containing $c_v > 0$ value. You are allowed to start and end anywhere in the graph, and are allowed to visit each node and traverse each edge multiple times. At any point in time, you can loot the treasure chest of your current node, but each chest can be looted at most once! Design an $O(N + M)$ algorithm to calculate the maximum value you can obtain.

#### Solution:

Note that if we are at some vertex $v$, it is optimal to visit, at some point, all vertices in the SCC of $v$. So compress the graph into a graph $G'$ of its SCCs. For each SCC in the graph, create a node in $G'$ with value equal to the total value of nodes in it, and for all pair of SCCs $A$ and $B$, create an edge between $A$ and $B$ in $G'$ if and only if there exists $u \in A$ and $v \in B$ such that there exists an edge between $u$ and $v$ in the original graph. Now we still want to find the maximum value possible in $G'$, under the same rules, except this time the graph is a directed acyclic graph. The runtime of the compression $O(N + M)$, as we can calculate all SCCs in $O(N + M)$ time and we can build $G'$ in $O(M)$ time (for every edge in $G$, if it connects two nodes of different SCCs, add it appropriately into $G'$). It’s fine if $G'$ has multiedges.

Let $DP[u]$ be the maximum value of a path starting at node $u$. $DP[u] = c_u + \max_v(DP[v])$ where $v$ goes over all neighbors of $u$ (i.e. there exists an edge from $u$ to $v$). Since the graph is acyclic, there are no circular dependencies. The runtime is $O(N + M)$ because there are $O(N)$ states, and $O(M)$ transitions over all states. If you wanted to do bottom up DP, you would need to first topologically sort the graph, then calculate all $DP$ values in reverse topological order. (Tarjan’s SCC algorithm from lecture outputs the SCCs in reverse topological order.)
Ear Decomposition
Throughout these notes let $G = (V, E)$ be a simple, no multiple edges, undirected graph with at least three vertices. Let $n$ be the number of vertices and $m$ the number of edges.

In lecture we said that $G$ is bi-connected if it does not contain any articulation points. This definition was carefully chosen so that $G$ would have a partition of the edge into bi-connected components. Thus a single edge is bi-connected.

**Lemma 1** $G$ is connected and bi-connected and contains at least three vertices if for every pair of vertices there are two vertex-disjoint (except at the endpoints) paths between them.

**Solution:** Suppose that for every pair of vertices in $G$, there exists two vertex-disjoint paths between them. It is trivial to show that $G$ is connected and contains at least 3 vertices, so it remains to show that $G$ is biconnected. So assume for the sake of contradiction that $G$ has an articulation point $a$ such that when $a$ is removed, vertices $u$ and $v$ are disconnected. However since there are two vertex-disjoint paths between $u$ and $v$, at least one doesn’t use $a$, so $u$ and $v$ are still connected after removing $a$, a contradiction. Thus $G$ contains no articulation points and is biconnected.

**Definition 1** An ear decomposition $ED$ of $G$ is a partition of the edges of $G$ into simple paths $P_1, \ldots, P_k$.

1. $P_1$ is a simple cycle.
2. $P_2, \ldots, P_k$ are open paths, their endpoints are distinct.
3. The end points of the path $P_{i+1}$, its attachments, belong to the vertices of $P_1 \cup \cdots \cup P_i$ for $1 \leq i < k$.

The following figure shows a graph and an ear decomposition of it. The cycle $P_1$ is shown in blue, the first path $P_2$ is shown in red, the second path $P_3$ is shown in green, and the last path (just an edge) $P_4$ is shown in purple.

![Graph and Ear Decomposition Diagram](image)

Each $P_i$ is called an ear.

**Lemma 2** The number of ears of a $ED$ is $m - n + 1$. 

2
Solution: We show with induction that for all $i$, the collection $P_1, P_2, \ldots, P_i$ in the graph (all missing one edge), the graph is a tree.

For $i = 1$, the $P_1$ is a single cycle with one edge removed, which is a tree.

For $i > 1$, assuming the collection $P_1, P_2, \ldots, P_{i-1}$ is a tree. Then adding $P_i$ with one edge removed simply adds two lines off the graph, so it is still a tree.

Now since a tree has $n - 1$ edges, the total number of edges removed is $m - n + 1$. Since one edge is removed per ear, it follows that the number of ears is $m - n + 1$.

Here’s an alternate proof. Consider the number of edges minus the number of vertices at all stages of the ear graph construction. In the beginning, since the graph is just a single cycle, this quantity is 0, and at the end it has value $m - n$. It is easy to see that each addition of an ear increases the quantity by 1. Thus $m - n$ ears are added, and including the initial cycle, this gives $m - n + 1$ ears.

Lemma 3 The number of back-edges of every DFS is $m - n + 1$.

Solution: The DFS tree has $n - 1$ tree edges. Since an undirected graph contains no cross edges, every other edge is a back edge, so there are $m - n + 1$ back edges.
Programming Problem Hints:

Your algorithm must make a new ear from each back-edge (why?). Thus the main subgoal of your algorithm is to assign each tree edge to a back-edge/ear.

Recall that if $T$ is a spanning tree and $e$ is a non-tree edge then there exists a unique cycle consisting of $e$ and edges from $T$. We denote this cycle by $C_e$. All edges in the ear belonging to back edge $e$ must come from $C_e$.

The next issue is that if there are several cycles all using a given tree edge, we need to decide which backedge/ear this tree edge belongs to. What criteria should we use? Consider the case when the DFS tree is a line.