CIS 1210—Data Structures and Algorithms—Fall 2024

Topological Sort & Strongly Connected Components—Tuesday, October 22/Wednesday, October 23

Readings

- Lecture Notes Chapter 18: DAGs and Topological Sort
- Lecture Notes Chapter 19: Strongly Connected Components

Review: Topological Sort

A topological sort of a directed acyclic graph (DAG) $G = (V, E)$ is an ordering of the vertices such that for each directed edge $(u, v) \in E$, u appears before v in the ordering. As described in the below algorithms, topologically sorting a DAG only takes $O(m + n)$ time, so **given a DAG, it is helpful to topologically sort it,** since most graph algorithms take $\Omega(m + n)$ time anyway. In other words, topologically sorting a DAG is usually a free step, and if not necessary for your algorithm, it can make reasoning/thinking about the problem easier since it gives you a visual. Below are two algorithms to find a topological sort:

Kahn's Algorithm

Every DAG has a source node, or a node with no incoming edges. Kahn's algorithm relies on this intuition — at a high-level, the algorithm operates by repeatedly finding a source node, putting it next in the topological sort, removing the node and all of the edges incident on it from the graph, and repeating this process.

Kahn's algorithm runs in $O(m+n)$ time. As seen in the pseudocode, the first step to compute the in-degree of each node takes $O(m + n)$ time since for each node, we scan through its neighbors; the second step to populate the queue takes $O(n)$ time since we iterate through all of the vertices. In our while loop, note that we enqueue each node exactly once and scan through each of its neighbors, performing constant work for each, which takes $O(m + n)$ time.

Tarjan's Algorithm

Tarjan's algorithm leverages the finishing times of DFS as shown in the pseudocode by just running DFS and then returning the nodes in decreasing order of finishing times. Thus, it also runs in $O(m + n)$ time.

Review: Kosaraju's Algorithm

Given a directed graph $G = (V, E)$, a **strongly connected component (SCC)** is a maximal set $S \subseteq V$ such that for all $u, v \in S$, there exists a path $u \rightsquigarrow v$ and a path $v \rightsquigarrow u$. Thus, we can decompose a directed graph G into its SCCs, yielding G^{SCC} or our kernel graph. Formally, $G^{SCC} = (V^{SCC}, E^{SCC})$. Each vertex v_i in G^{SCC} represents a single SCC C_i in G, and an edge (v_i, v_j) exists in G^{SCC} if G contains the directed edge (x, y) where x is in SCC C_i and y is in SCC C_j. Observe that G^{SCC} is a DAG, meaning that we can topologically sort it to make the problem easier to think about.

Kosaraju's algorithm is an algorithm that we can use to compute the SCCs of a graph, and by extension, to obtain G^{SCC}. It operates by running two DFS traversals, one on G and another on G^T, the transposed graph obtained by reversing the direction of edges in G; in the latter, we consider vertices in order of decreasing finishing times.

Thus, Kosaraju's runs in $O(m+n)$ time. As seen in the pseudocode, the first step is just DFS, which takes $O(m + n)$ time; the second step is computing G^T, but this can be done in $O(m + n)$ time since we just reverse the direction of edges; and the third step is also just DFS, which takes $O(m + n)$ time.

Problems

Problem 1: True or False

1. Every DAG has exactly one topological sort.
2. If a graph has a topological sort, then a DFS traversal of the graph will not find any back edges.
3. Given a DAG G Tarjan's and Kahn's algorithm will always output the same topological ordering.

Problem 2

How does the number of SCC's of a graph change if a new edge is added?

Problem 3

A graph $G = (V, E)$ is "almost strongly connected" if adding a single edge makes the graph strongly connected. Design an $O(|V| + |E|)$ algorithm to determine whether a graph is almost strongly connected.