Readings

- [Lecture Notes Chapter 20.3: Shortest Path in DAGs]
- [Lecture Notes Chapter 22: Union Find]

Problems

Problem 1

True or False: The shortest path algorithm in an edge weighted DAG works even with negative edge weights.

Solution

True. First, because we are considering a DAG, we do not have to worry about negative weight cycles. Also notice that because we consider vertices in topological order, no ancestor of \( v \) will be relaxed after \( v \) itself is relaxed.

Problem 2

Recall the solution for finding the shortest path on an edge weighted DAG. Now, for a single source \( s \), how can we find the longest (heaviest) path in a DAG that begins at \( s \)?

Solution

First, we negate the edge weights (multiply by -1) and then use the shortest path finding algorithm elucidated earlier. This algorithm also runs in \( O(m+n) \) time as discussed earlier. Intuitively, the shortest path algorithm will return the path that sums to the smallest weight value. This consequently, has the largest absolute value. Eg. -20 is smaller than -10 but has a larger absolute value. As a result, in the original graph, this path would be the heaviest.

Problem 3

Is it guaranteed that a call to \( \text{Find}(v) \) will always return the same result throughout the algorithm? If not, is it possible to modify the algorithm such that it does?

Solution

It is not guaranteed. Whenever a \( \text{Union} \) is performed, one of the subsets will always change its indicator value. It is not possible to prevent this for all vertices, as the nature of the methods requires indicators to change, but it is possible to modify the algorithm such that one particular indicator is maintained.

We can add a condition around our set precedence to state that if the \( \text{Union} \) of two sets is performed, and one of the sets has our desired constant indicator, then force the other set to become a child of the first. Note that this will create inefficiencies, as we can no longer guarantee that the heights of our trees only increase in cases of equal height.
Problem 4
How could one use Union-Find to detect cycles in an undirected graph?

Solution
Perform a DFS on the graph. As the search progresses, do the following whenever a new edge \((u,v)\) is encountered:

1. Compare the result of \texttt{Find} on both \(u\) and \(v\). If they are in the same component, a cycle exists. Otherwise:
   2. Perform \texttt{Union}(u, v).

By the end of the algorithm, if no node was encountered twice within the same component, we may be sure that there are no cycles. Note that this is not quite as efficient as the node-coloring method, but may be useful in some restrictive circumstances.