16 All-Pair Shortest Paths (March 6)

16.1 The Problem

In the last lecture, we saw algorithms to find the shortest path from a source vertex s to a target vertex t in a directed graph. As it turns out, the best algorithms for this problem actually find the shortest path from s to every possible target (or from every possible source to t) by constructing a shortest path tree. The shortest path tree specifies two pieces of information for each node v in the graph

- $\text{dist}(v)$ is the length of the shortest path (if any) from s to v.
- $\text{pred}(v)$ is the second-to-last vertex (if any) the shortest path (if any) from s to v.

In this lecture, we want to generalize the shortest path problem even further. In the *all pairs shortest path* problem, we want to find the shortest path from *every* possible source to *every* possible destination. Specifically, for every pair of vertices u and v, we need to compute the following information:

- $\text{dist}(u, v)$ is the length of the shortest path (if any) from u to v.
- $\text{pred}(u, v)$ is the second-to-last vertex (if any) on the shortest path (if any) from u to v.

For example, for any vertex v, we have $\text{dist}(v, v) = 0$ and $\text{pred}(v, v) = \text{NULL}$. If the shortest path from u to v is only one edge long, then $\text{dist}(u, v) = w(u \to v)$ and $\text{pred}(u, v) = u$. If there is *no* shortest path from u to v—either because there's no path at all, or because there's a negative cycle—then $\text{dist}(u, v) = \infty$ and $\text{pred}(v, v) = \text{NULL}$.

The output of our shortest path algorithms will be a pair of $V \times V$ arrays encoding all V^2 distances and predecessors. Many maps include a distance matrix—to find the distance from (say) Champaign to (say) Columbus, you would look in the row labeled 'Champaign' and the column labeled 'Columbus'. In these notes, I'll focus almost exclusively on computing the distance array. The predecessor array, from which you would compute the actual shortest paths, can be computed with only minor additions to the algorithms I'll describe (hint, hint).

16.2 Lots of Single Sources

The most obvious solution to the all pairs shortest path problem is just to run a single-source shortest path algorithm V times, once for every possible source vertex! Specifically, to fill in the one-dimensional subarray $\text{dist}[s][\,]$, we invoke either Dijkstra's or Moore's algorithm starting at the source vertex s.

```
OBVIOUSAPSP(V, E, w):
  for every vertex s
      dist[s][ ] ← SSSP(V, E, w, s)
```

The running time of this algorithm depends on which single source algorithm we use. If we use Moore's algorithm, the overall running time is $\Theta(V^2E) = O(V^4)$. If all the edge weights are positive, we can use Dijkstra's algorithm instead, which decreases the running time to $\Theta(VE + V^2 \log V) = O(V^3)$. For graphs with negative edge weights, Dijkstra's algorithm can take exponential time, so we can't get this improvement directly.