Exercise 3. Let W be a walk from x and y. Prove that W contains a path P from x and y. Furthermore, if the edge weights are nonnegative, prove that P's total cost is at most W's total cost.

- We can reduce the walk W to be the path P
- So, with nonnegative edge weights, P's total cost is at most W, both traversing the same set of edges and nodes.

Let W be a walk from x to y. We consider two cases:

[Case 1: W does not repeat any vertices] Since W does not repeat any vertices for this case, then it is already a path P.

[Case 2: W repeats vertices] Suppose some vertex v appears at least twice in W. Consider the first two occurrences of v. The portion of W between these occurrences forms a cycle. Removing this cycle yields another walk W' from x to y.

Repeat this process whenever a vertex appears more than once. Each removal shortens the walk, so the process terminates. The resulting walk has no repeated vertices and is therefore a path P from x to y.

<u>Problem 4. Single Source Shortest Path. Given a connected weighted digraph, and a node x, what is the length of the shortest path from x to y, for every node y?</u>

<u>Problem 5. Single Destination Shortest Path. Given a connected weighted digraph, and a node x, what is the length of the shortest path from y to x, for every node y?</u>

<u>Problem 6. All-Pair Shortest Path. Given a connected weighted digraph, what is the length of the shortest path from x to y, for every node x and every node y?</u>

Exercise 7. Reduce Problem 4 to Problem 5 and vice versa. That is, find an algorithm for Problem 4 assuming you have a magic oracle that can solve instances of Problem 5, and vice versa.

We can solve SSSP via an SDSP oracle by reversing the graph, then computing the distance to x, which will be equal to the distance to x.

To solve SDSP via an SSSP oracle, we do the same: reverse the graph, then compute the distance to every node y.

Exercise 8. Reduce Problem 6 to Problem 4 have an algorithm that solves Problem 4 with only a $\Theta(n)$ overhead. That is, assuming you an algorithm that solve Problem 4 in $\mathcal{O}(f(n))$ time, find an algorithm that solves Problem 6 in $\mathcal{O}(n \cdot f(n))$ time.

We run the SSSP algorithm once per vertex. The i-th run computes the distances from vertex i to all others, so running it for all n nodes would solve Problem 6 since it would yield the distances for all pairs. Since we perform n runs, the algorithm would run in O(n * f(n)) time.

Why do we drop the k variable in Floyd-Warshall?

The invariant for Floyd-Warshall's algorithm is: "At the start of iteration k, d[i][j] equals the length of the shortest path from i to j whose immediate vertices are {0, 1, …, k - 1}. Then {0, 1, …, k} after iteration k finishes."

We can update the matrix in place because during iteration k, we only introduce vertex k as a new allowed intermediate. Since distances only decrease and don't use vertices beyond k, the DP invariant remains valid.

How can we detect cycles in Floyd-Warshall?

We can detect cycles in Floyd-Warshall by checking whether any diagonal entry is negative. The length of the shortest path from a vertex to itself is 0, so if d[i][i] < 0, then a negative cycle is reachable from the vertex i.