Final Exam Topics:

1. Disjoint Sets
2. Binary Search Tree (except delete)
3. Graphs - BFS / DFS / MST
4. MST (choose kruskal/prim)
5. SSSP (choose dijkstra/bellman-ford)
6. APSP (Floyd-Warshall)
7. One Question from Midterm

Floyd-Warshall (All Pairs Shortest Path)

Calculates shortest path for all nodes to all other nodes.

It uses B-F ($O(VE)$) where $E \approx V^2$. Therefore, BF is actually $O(V^3)$.

If we have to find the shortest path for all nodes, we must do BF V times. This brings us to the time complexity of $O(V^4)$! But, can we run it faster than $O(V^4)$?

- Yes! We are wasting lots of extra calculations.

We can actually do this with $O(V^3)$ time complexity using *dynamic programming*.

Dynamic Programming:

- Divide and conquer (recursive)
- Optimal structure
- Overlapping subproblems

We use recursive functions without the performance downsides of recursion via *memoization*! With *memoization*, we store the data that we have already calculated.

Recursion:

- "intermediate nodes" by the ID of the node

Given nodes n from $1...n$. Consider nodes $1...k$ where $(k \leq n)$ meaning we are allowed to use nodes 1 through k in our shortest path. Also consider the shortest path from node i to node j ("assuming" connected).

This gives us two possibilities:

1. k is somewhere between i and j
2. k is not between i and j

If k is not on the path then the path from i to j using k is equivalent to the path from i to j using $k-1$.

How Floyd-Warshall Works:

We use 2D matrices to execute this algorithm.

NOTE: We only need the current and previous layers of the matrix

The two matrices on top show you the length of the path, but not the path itself.

We hold one matrix to keep track of distance and one to hold the parents.

Here's an example initial matrix: $D^{(0)}$

$$\begin{bmatrix} 0 & 3 & 8 & - & -4 \\ - & 0 & - & 1 & 7 \\ - & 4 & 0 & - & - \\ 2 & - & -5 & 0 & - \\ - & - & - & 6 & 0 \end{bmatrix}$$

NOTE: The initial matrix is just the adjacency matrix. All paths are null/infinite if not adjacent (no reachability info).

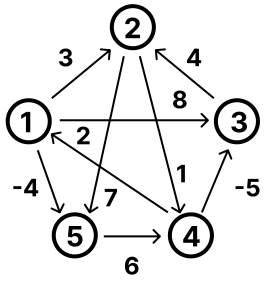
Here's the final matrix (result): $D^{(5)}$

$$\begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

We got D_5 from D_4 which we got from D_3...

The highest node will be the end ie node of 5 will have a matrix of D_5

Given this sample graph:



We then use the functions:

$$d_{ij}^{(k)} = \begin{cases} \text{weight(i,j) if k == 0 (base case)} \\ \min(\ d_{ij}^{(k-1)},\ d_{ik}^{(k-1)},\ d_{kj}^{(k-1)}\ ) \text{ for k >= 1 (recursive case)} \end{cases}$$

$$P_{ij}^{(0)} = \begin{cases} \text{null if i == j or weight (i,j) = INF (no edge)} \\ \text{i if i != j and weight (i,j) < INF} \end{cases}$$

$$P_{ij}^{(0)} = \begin{cases} P_{ij}^{(k-1)} \text{ if } d_{ij}^{(k-1)} <= d_{jk}^{(k-1)} + d_{kj}^{(k-1)} \\ P_{kj}^{(k-1)} \text{ if } d_{ij}^{(k-1)} > d_{jk}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

Looking at the initial parent/predecessor matrix: $P^{(0)}$

$$\begin{bmatrix} - & 1 & 1 & - & 1 \\ - & - & - & 2 & 2 \\ - & 3 & - & - & 3 \\ 4 & - & 4 & - & - \\ - & - & - & 5 & - \end{bmatrix}$$

With a final of: $P^{(6)}$

$$\begin{bmatrix} - & 3 & 4 & 5 & 1 \\ 4 & - & 4 & 2 & 1 \\ 4 & 3 & - & 2 & 1 \\ 4 & 3 & 4 & - & 1 \\ 4 & 3 & 4 & 5 & - \end{bmatrix}$$