So far, we have learned the Johnson’s algorithm which has work $O((mn + n^2) \log n)$ if using binary heaps and $O(mn + n^2 \log n)$ if using fibonacci heaps. The span is $O((m+n) \log n)$ when using binary heaps. We consider dense graphs where we will learn an algorithm that has the same work as Johnson’s with binary heap, but can have a smaller span. It is also much easier to implement.

1 Dynamic Programming Using Matrix Multiplication

We learned some very parallel algorithms to do matrix multiplication. Today, let’s see how to leverage these algorithms to find shortest paths in graphs.

Now instead of full all-pairs shortest paths, what if I asked you, how to compute the shortest path between all vertices if you only want to use at most 1 edge? The answer is already in the adjacency matrix.

$$D^1_{i,j} = W_{i,j}$$

What if you want to use at most 2 edges?

$$D^2_{u,v} = \min \left\{ W_{u,v}, \min_{w \in V} \{ W_{u,w} + W_{w,v} \} \right\}$$

Now if I represent these as an adjacency matrix:

$$D^2_{i,j} = \min \left\{ D^1_{i,j}, \min_{k=1}^{n} \{ D^1_{i,k} + D^1_{k,j} \} \right\}$$

Therefore, $D^2 = D^1 \times D^1$

This looks very much like a matrix multiplication operation. How about if we wanted to use at most 3 edges?

$$D^3_{i,j} = \min \left\{ D^2_{i,j}, \min_{k=1}^{n} \{ D^2_{i,k} + D^1_{k,j} \} \right\}$$

Therefore, $D^3 = D^2 \times D^1 = (D^1)^3 = W^3$

What is the shortest path that uses at most $n$ edges? $W^n$. This is the absolute shortest path. What is the work of computing $W^n$? $O(n^3 \log n)$ if you use standard matrix multiply. What is the span? $O(\log^2 n)$ using the triply nested for loop version of matrix multiplication.
2 Floyd Warshall Algorithm

Now let's see if we can reduce the work, still using a form of dynamic programming. In the previous dynamic program, we defined $D_{i,j}^k$ as the shortest distance from $i$ to $j$ using at most $k$ edges. Now we will define a slightly different version.

We can number all vertices from 1..$n$. Now we define $W_{i,j}^k$ as the shortest path from $i$ to $j$ using only vertices from 1..$k$ as intermediate vertices. $W_{i,j}^0$ is the edge weight from $i$ to $j$. Therefore, we can write the recurrence

$$W_{i,j}^k = \min \{ W_{i,j}^{k-1}, W_{i,k}^{k-1} + W_{k,j}^{k-1} \}$$

This allows us to compute all pairs shortest paths with three nested for loops as follows:

```
1   for k ← 1 to n
2    do parallel_for i ← 1 to n
3      do parallel_for j ← 1 to n
4         do $W_{i,j}^k = \min \{ W_{i,j}^{k-1}, W_{i,k}^{k-1} + W_{k,j}^{k-1} \}$
```

The work is $O(n^3)$ and the span is $O(n \log n)$. The outermost loop cannot be parallelized each iteration depends on the previous one.

How can you compute this? You keep two $n \times n$ matrices — initialize the first one with the adjacency matrix and compute $D^2$ and place the second one using the results from the first. After you are done computing the second one, you can use the first one again for $D^3$. Therefore, you will only use a total of $2n^2$ space ever.