CSE341T/CSE549T **10/15/2014**

More All Pairs Shortest Paths

Lecture 13

1 Johnson's Algorithm Continued

Recall that we proved the following lemma last time:

Lemma 1 *Given a weighted directed graph $G = (V, E, w)$ with weight function $w : E \to \mathcal{R}$, and "potential" function $p : v \to R$, then for a graph $G' = (V, E, w')$ with weights*

$$w'(u, v) = w(u, v) + p(u) - p(v),$$

we have that for every path from s to t,

$$W_{G'}(s, t) = W_G(s, t) + p(s) - p(t)$$

where $W_G(s, t)$ is the weight of the path from s to t in graph G.

This lemma shows that the potential weight transformation does not affect the shortest paths since all paths between the same two vertices will be affected by the same amount (i.e. $p(s) - p(t)$ for vertices s and t).

Now the question is whether we can pick potentials so that they get rid of all negative weight edges. The answer is yes. The trick is to add a source vertex and link it to all other vertices V with an edge weight of zero. Now we find the shortest path from s to all vertices using the original weights. This can be done with a SSSP algorithm that allows negative weights (e.g. Bellman-Ford). Now we simply use the distance from s as the potential.

Exercise 1 *Show that the this potential guarantees that no edge weight will be negative.*

The intuition behind this exercise is as follows: If there are no negative weight cycles in the original graph, then there are none in this new graph either, because the weight of cycles does not change when we reweight the graph. Now consider an edge (u, v) in this new graph. The new weight of this edge is $w'(u, v) = w(u, v) + p(u) - p(v)$. By triangle inequality, we have $p(v) \leq p(u) + w(u, v)$. Therefore, you get the answer.

This together with a parallel application of SSSP with non-negative weights (indicated as SSSP$^+$) gives us our desired algorithm. The work is $O(mn + n^2 \lg n)$ and the critical path length is $O(m + n \lg n)$.

2 Dynamic Programming Using Matrix Multiplication

So far, we have learned the Johnson's algorithm which has work $O((mn+n^2)\lg n)$ if using binary heaps and $O(mn + n^2 \lg n)$ if using fibonacci heaps. The span is $O((m + n)\lg n)$ 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.

We learned some very parallel algorithms to do matrix multiplication. Today, lets 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? $O(n^3 \lg n)$ if you use standard matrix multiply. What is the span? $O(\log^2 n)$ using the triply nested for loop version of matrix multiplication.

3 Floyd Warshall Algorithm

Now lets see if we can reduce the work, still using a form of dynamic programming. In the previous dynamic program, we defined $D^k_{i,j}$ 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^k_{ij} as the shortest path from i to j using only vertices from $1..k$ as intermediate vertices. W^0_{ij} is the edge weight from i to j. Therefore, we can write the recurrence

$$W^k_{ij} = \min\left\{W^{k-1}_{ij}, W^{k-1}_{ik} + W^{k-1}_{kj}\right\}$$

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_ij^k = min{W_ij^(k-1), W_ik^(k-1) + W_kj^(k-1)}
```

The work is $O(n^3)$ and the span is $O(n \lg n)$. The outermost loop can not 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.