CS 388R: Randomized Algorithms, Fall 2023 October 2nd, 2023

Lecture 12: All Pairs Shortest Path

Prof. Eric Price *Scribe: Jasmeet Kaur, Ronak Ramachandran*

NOTE: THESE NOTES HAVE NOT BEEN EDITED OR CHECKED FOR CORRECTNESS

1 Overview

In the last lecture we explored fingerprinting and examined examples of fingerprinting in action. This included various approaches to verifying a matrix multiplication, polynomial identity testing, string matching, and primality testing.

In this lecture we consider deterministic and randomized approaches to solving the All Pairs Shortest Path (APSP) problem, in which we are given the adjacency matrix of graph and are asked to find the shortest paths between every pair of vertices.

2 But first, a game...

We began class with a simple game: First, all students closed their eyes. Then, Eric named a subset of students—for instance, "all students wearing red." The students won the game if exactly one person in the subset raised their hand. For the record, the students won.

If students are able to remember every other student, then it's always possible to guarantee a win: simply impose an order on the students and have the minimum student in the subset raise their hand. The game only becomes interesting once each student has more limited knowledge. If students only know that k students are in the subset then each student can choose to raise their hand with probability $1/k$, and the expected number of raised hands in the subset will be 1. Under this strategy, the probability that exactly one person in the subset raises their hand is

$$\mathbb{P}[\text{win}] = k \cdot \frac{1}{k}\left(1 - \frac{1}{k}\right)^{k-1} \approx \frac{1}{e},$$

which is a constant winning probability.

What if the students don't know the size of the subset? All students can agree ahead of time (while Eric is not in the room) on some guess $k = 2^i$ for the size of the subset, where each i is chosen uniformly from $[\log n]$. This will succeed with probability $\Omega\left(\frac{1}{\log n}\right)$. This strategy will be relevant in section 4.1.

3 All Pairs Shortest Paths : Computing Distances

Given a graph $G = (V, E)$, the objective is to find the shortest path between every pair of vertex (i, j). Some of the algorithms that achieve this :

- Dijkstra's Algorithm
- Floyd-Warshall Algorithm

Using its Adjacency Matrix A as a representation of the graph, the aim is to compute a distance matrix D, where D_{ij} is the shortest distance between vertex i and j. We can assume that Matrix Multiplication for a matrix of size n happens in n^{ω}.

1. A – Adjacency Matrix of G
2. A^2 – Adjacency Matrix of a graph *G'*, such that each pair of vertex has a path length of at most 2. The value A^2_{ij} denotes the number of paths of length 1 or 2 between i and j in graph *G'*.

Such matrices can be computed upto n in $n \cdot n^{\omega}$ time.

Key Ideas behind the algorithm :

- G is a graph with adjacency matrix A and distance matrix D. *G'* is graph with adjacency matrix $A' = A^2$ and distance *D'*.
- This will be formalised later but it is important to note that if G has a path of length 4, then G' will have a path of length 2. If G has a path of length 5, G' will have a path of length 3.
- *G'* is complete if and only if every vertex in the graph is connected to every other vertex by a path of length at most 2. In such a case, we can compute D' from D using :

$$D = 2A' - A \tag{1}$$

- If *G'* is not complete, we can recursively find D using *D'*.

3.1 Computing D from D'

Lemma : If D_{ij} is even, then $D_{ij} = 2 \cdot D'ij$. If D_{ij} is odd, then $D_{ij} = 2 \cdot D'_{ij} - 1$.

This is easy to prove : If D_{ij} is odd, then D_{ij} is of the form 2l+1. Hence, D'_{ij} can not be more than l+1. If D'_{ij} is l, then path in G at most 2l. Hence if $D'ij \leq$ l, then $D_{ij} \leq$ 2l $<$ 2l+1. Hence, D'_{ij} can not be less than or equal to l. The only value it can take is l+1 and hence, $D_{ij} = 2D'_{ij} - 1$

Hence, given D' and using this lemma we can compute D' if we know that parity of D.

3.2 Computing parity for D

Lemma : If D_{ij} is even, then $D'_{kj} >= D'_{ij}$ for all neighbours k of i. If D_{ij} is odd, then $D'_{kj} <= D'_{ij}$ and there exists $D'_{kj} < D'ij$.

Hence if $N(i)$ denotes the set of neighbors of i and d is the degree.

$\sum_{k \in N(i)} D'_{kj} >= d \cdot D'_{ij}$ if D_{ij} is even. $\sum_{k \in N(i)} D'_{kj} < d \cdot D'_{ij}$ if D_{ij} is odd.

The quantities on the left can be computed as follows :

$S_{ij} = \sum_{k \in N(i)} D'_{kj} = \sum_{k=1}^{n} A_{ik} \cdot D'_{kj}$

if $S_{ij} >= d(i).D'_{ij}$ then D_{ij} is even else if $S_{ij} < d(i).D'_{ij}$ then D_{ij} is odd.

3.3 Algorithm

Algorithm 1: APSP Distance Computation

```
Input : $G(V, E)$
Output : Distance Mqatrix of G
$Z \leftarrow A^2$
Compute A' from Z and A
if $A'_{ij} = 1$ for all i,j $i \neq j$ then
$D = 2 \cdot A' - A$
Return D
Compute the Distance Matrix of A' (D') $S = AD'$
$D_{ij} = 2 \cdot D'_{ij}$ if $(S_{ij} >= D'_{ij}.Z_{ii})$
$D_{ij} = 2 \cdot D'_{ij} - 1$ if $(S_{ij} < D'_{ij}.Z_{ii})$
```

4 Using Randomness for APSP

One goal we might have is to produce a successor matrix, that is, a matrix P such that $P_{ij} = k$ for some k such that $D_{ij} = D_{ik} + D_{kj}$. In other words, we want P_{ij} to store some k included in a shortest path from i to j. We can then recurse (look at P_{ik} and P_{kj} and so on) to reconstruct an entire shortest path from i to j. (A small note: to ensure this recursion halts, we want $k \notin \{i, j\}$, so for adjacent i and j we can say $P_{ij} = -1$ or something.)

4.1 Warmup: Directed Tripartite Graph

Imagine we're given a tripartite graph with an adjacency matrix A specifying the edges from the left to the middle layer and an adjacency matrix B specifying the edges from the middle to the right layer.

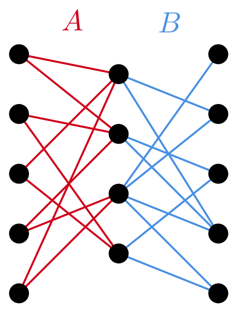

Figure 1: A tripartite graph.

Since $A_{ik}B_{kj} = 1$ iff there is an edge from i to k and an edge from k to j,

$$(AB)_{ij} = \sum_k A_{ik}B_{kj} = (\text{the number of } i \text{ to } j \text{ paths}).$$

Our goal then is to find a successor matrix P such that for all i, j such that $(AB)_{ij} > 0$, $P_{ij} = k$ where $A_{ik}B_{kj} = 1$.

Suppose that $\forall i, j$, only one i to j path exists. In this case, we can define A' such that $A'_{ik} = A_{ik} \cdot k$. Then

$$P := A' \cdot B = \sum_k kA_{ik}B_{kj} = \begin{cases} k & \text{if } i \to k \to j \text{ path exists,} \\ 0 & \text{otherwise.} \end{cases}$$

In general, however, there may be more paths from i to j. Let's say there are q paths from some specific i to some specific j. We have a subset of q possible k, and we want to know just one. The insight here comes from the game we played at the start of class: if we choose $\delta_1, \dots, \delta_n$ to each independently be 1 with probability $\frac{1}{q}$ and then define A' such that $A'_{ij} = A_{ij} \cdot j \cdot \delta_j$, we find that $P := A' \cdot B$ will have P_{ij} equal to a single valid k with probability about $\frac{1}{e}$.

Now what if we don't know q? We can repeatedly try different values of q and set P_{ij} to the minimum non-zero value it took on for all values of q we tried. Guessing the scale of q (in the same way we guessed the size of the subset of students in the game) will succeed with probability $\Omega\left(\frac{1}{\log n}\right)$, so we need to repeat this $O(\log^2 n)$ times to succeed with high probability. This results in an overall running time of $O(MM(n) \cdot \log^2 n)$, where $MM(n)$ is the time complexity of multiplying $n \times n$ matrices.

4.2 Finding paths for general graphs

As a reminder, our goal is to construct some P such that for all i, j, P_{ij} is equal to some k along the shortest path from i to j. We're given the graph's adjacency matrix A and its shortest path distance matrix D.

The intuition is this: If we define R as

$$R_{ij} = \begin{cases} 1 & \text{if } D_{ij} = l - 1 \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

then for nodes with shortest path length l, our problem reduces to the tripartite graph with adjacency matrices A and R. Unfortunately, there may be n unique entries in D, meaning n different l we might need to handle, and constructing n different R would take time $O(n^3)$ time.

With a few modifications, we can resolve this issue. Fix some arbitrary i and j with shortest path of length l. To populate P_{ij} with some k along the path from i to j, we just want some some $k \in N(i)$ such that $D_{kj} = l-1$. Let q be the number of such k. Note that for all $k \in N(i)$, $A_{ik} = 1$ and $D_{kj} \in \{l-1, l, l+1\}$. For this reason, we can just calculate three matrices D^0, D^1, D^2 defined so that $D^a_{ij} = 1$ iff $D_{ij} \equiv a (\text{mod } 3)$. Then, it suffices to find a k such that $A_{ik} \cdot D^{(l-1) \text{mod } 3}_{kj} = 1$.

To summarize the whole algorithm at a glance, for every $a \in \{0, 1, 2\}$,

1. Sample $q = 2^t$ with t drawn uniformly from $[\log n]$.
2. Construct $A'_{ik} = \delta_k \cdot k \cdot A_{ik}$ with δ_k being 1 with probability $\frac{1}{q}$ independently for all $k \in [n]$.
3. Calculate $A' \cdot D^a$.
4. For every (i, j) such that $(D_{ij} - 1) \bmod 3 = a$, set P_{ij} equal to the miniumum non-zero value of $(A' \cdot D^a)_{ij}$ seen so far.
5. Repeat above steps $O(\log^2 n)$ times.

With high probability we will have found the successor matrix P of our graph. As with the tripartite graph case, this takes $O(MM(n) \cdot \log^2 n)$ time.