CS 388R: Randomized Algorithms, Fall 2021 October 5th 2021

Lecture 12: All-pairs shortest path

Prof. Eric Price *Scribe: Anirudh Srinivasan, Sai Surya Duvvuri*

1 Overview

In the last lecture we looked at fingerprinting algorithms, which can be used to check if 2 polynomials are equal, if the result of matrix multiplication is correct or if a string contains a particular substring.

In this lecture we will look at shortest path algorithms in graphs and propose a randomized algorithm for all pairs shortest path that is better than deterministic algorithms like Floyd-Warshall [FL62, WA62].

2 Probability game

Consider the following game: n students in the class should close their eyes and decide whether or not to raise their hands. If more than one student raise their hands the whole class loses. Let's say each student raises a hand with probability $1/n$, then

$$\mathbb{P}\left(\text{only one student will raise their hand}\right) = n \cdot (1 - 1/n)^{n-1} \cdot 1/n \simeq 1/e \tag{1}$$

The above algorithm assumes that everyone knows the value of n. If no one knows the actual value and only an upper bound N, then everyone can make a guess k for n then the probability of success will be $n \cdot (1 - 1/k)^{n-1} \cdot 1/k \simeq (n/k) \cdot e^{-n/k}$. if the guess k is of the same order as n then we have a constant probability of success. So we can sample the guess k from the set $\left\{2^0, 2^1, \ldots, 2^{\log N}\right\}$, which will reduce the success probability down by $\log(N)$. So the overall probability of success will be $\Omega\left\{1/\log(N)\right\}$. This randomized algorithm will be used later in the lecture.

3 Deterministic shortest path algorithms

Let $m \overset{\text{def}}{=}$ number of edges in a directed graph, $n \overset{\text{def}}{=}$ number of vertices, the worst case running times for different algorithms solving single source shortest path problem are as follows.

1. For unweighted graphs:

   BFS - $O(n + m)$

2. For weighted graphs:

   Dijkstra [DI59]- $O(m + n\log n)$

   Bellman Ford - $O(mn)$

For all pairs shortest path problem - Floyd-Warshall [FL62, WA62] - $O(n^3)$

4 Shortest distance between all pairs of vertices

Repeated matrix multiplication of adjacency matrix is used to determine the distance between every pair of vertices of a graph. We will consider that the cost of matrix multiplication of 2 (n, n) matrices is $O(MM(n))$.

4.1 Using repeated matrix multiplication

Suppose A is the adjacency matrix of a graph G, where A_{ij} is 1 if there is an edge between vertices i and j, else 0. This graph assumes self loops, i.e diagonal of A are all ones. Let G^2 be the graph we get by using non-zero entries of A^2 as edges. Each entry of the matrix multiplication A^2_{ij} will carry the number of paths of length atmost 2 between vertices i and j in the graph. Multiplying k times would result in entries A^k_{ij} carrying the number of paths of length atmost k between i and j.

Let D be the matrix where D_{ij} is the shortest distance between vertices i and j. Suppose that we want to estimate D', which is a rough estimate of D to a factor of 2, specifically, $D'_{ij} \in [D_{ij}, 2D_{ij}]$. We can compute A, A^2, ... $A^{2^{logn}}$ and determine D'_{ij} as

$$D'_{ij} = \text{minimum } k \text{ such that } A^k_{ij} > 0$$

Since there are $\log(n)$ matrices to compute, this can be done in $O(MM(n)\log(n))$ time, where $MM(n) \overset{\text{def}}{=}$ matrix multiplication time. However, this method only gives us a matrix D' that is rough estimate of D.

4.2 Resolving parity

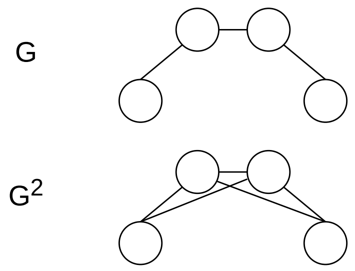

Figure 1: Example of G and G^2

To determine D, a better estimate for D' is first derived. Lets consider a toy graph (like the one in Figure 1) G and it's corresponding G^2. We can compute the values of A and A^2. Let D' be the true distance between distance between 2 vertices in G^2. We can make the following observation about D'

$$D'_{ij} = \lceil \frac{D_{ij}}{2} \rceil$$

When D_{ij} is even, $D'_{ij} = \frac{D_{ij}}{2}$ trivially. When D_{ij} is odd, parity needs to be accounted for during division. To do this, we can look at the neighbours of a vertex j $(N(j))$ to determine the value of D_{ij}. For a vertices s, u, we can observe that

$$\text{if } D_{su} \text{ is even, } \forall v \in N(u)\ D'_{sv} \geq D'_{su}$$
$$\text{if } D_{su} \text{ is odd, } \exists v \in N(u) \text{ such that } D'_{sv} = D'_{su} - 1$$

We can test this condition using matrix multiplication $(D'A)$ to sum up the values of D' over the neighbourhood of a vertex

$$(D'A)_{su} = \sum_{v} D'_{sv} A_{vu} = \sum_{v \in N(u)} D'_{sv}$$

- if D_{ij} is even, $(D'A)_{ij} \geq |N(j)| D'_{ij}$
- if D_{ij} is odd, $(D'A)_{ij} \leq |N(j)| D'_{ij} - 1$

Using this, the value of D_{ij} mod 2 can be estimated, which gives the value of D as

$$D = 2D' - (D \bmod 2)$$

The complexity of this will be $O(MM(n)\log(n))$

5 Randomized Shortest Path Algorithm [SE92]

Given the matrix of true distances D, we define an algorithm to find the path between the nodes. A path matrix P is defined as follows

$$P_{ij} = k \text{ such that } D_{ij} = D_{ik} + D_{kj} \tag{2}$$

Multiple possible k can exist. We will first look at a case where k is unique.

5.1 Simpler Case: Directed Tripartite Graph

Consider the case where the graph is a **directed tripartite graph** (Figure 2). This graph has 3 sets of vertices X, Y and Z, with edges running only from X to Y and from Y to Z. Let A and B be the adjacency matrices for these 2 sets of edges. This fits in neatly with our definition from P from above, as all the intermediate nodes k will lie only in Y. To find whether a path exists between 2 vertices i and j in X and Z respectively, the product AB can be computed. AB_{ij} being 1 means that a path is present between i and j and 0 means that a path is absent. To find the actual path between the 2 vertices, a new matrix B' is defined as follows

$$B'_{kj} = kB_{kj}$$

AB'_{ij}, gives the index of the node in Y that connects i and j.

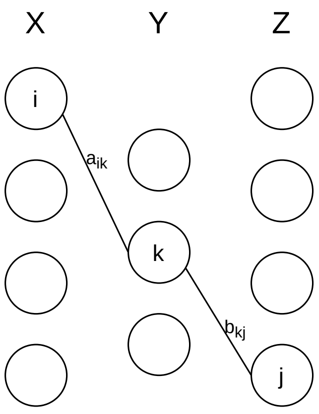

Figure 2: Directed Tripartite Graph

If k there are r possible values of k, a small modification is made to B' so that we get the index of one of the solutions of k.

$$B'_{kj} = kB_{kj}Z_k$$

where $\mathbb{P}[Z_k = 1] = \frac{1}{r}$. This algorithm will have a constant chance of finding the solution. Similar to the discussion in Section 2, if we do not know the number of solutions r, we can keep guessing the value of r and repeat this algorithm $logn$ times to get the correct intermediate node.

The overall complexity for these graphs will be $O(MM(n)\log^2(n))$, because the matrix exponentiation step has a $O(\frac{1}{\log(n)})$ chance of correctness and the intermediate node finding step has a $O(\frac{1}{\log(n)})$ chance of correctness.

5.2 Generalized Solution

The original problem on a dense graph can be reduced to the tripartite graph problem (Section 5.1), by splitting the path finding into 2 parts. If we are trying to find the path between 2 vertices i and j at distance L ($D_{ij} = L$), we can split this into 2 parts, one of length $L-1$ and one of length 1.

$$D_{ij} = D_{ik} + A_{kj}$$

where $D_{ik} = L-1$ and $A_{kj} = 1$. The algorithm from Section 5.1 can be used to solve this case, with the second transition matrix B from that algorithm being replaced by $B^{(L-1)}$. This is a binary matrix with $B_{ij}^{(L-1)}$ being 1 if there exists a path of length $L-1$ between i and j. However, to compute $B^{(L-1)}$, we will need to recursively compute $B^{(L-2)}$. This will involve recursion of depth $O(n)$, and due to this, the complexity of this method will increase by a factor of n compared to the tripartite graph problem.

Now we'll come up with a better solution. Since we're considering nodes k on the i to j path that are a distance 1 away from j, we call k as the successor of (i, j). We define a matrix S as the successor matrix such that S_{ij} contains the successor k of the i to j path. We are effectively solving for the successor matrix S. We observe that all neighbours k of j will have D_{ik} belonging to $\{D_{ij} - 1, D_{ij}, D_{ij} + 1\}$. Also, all nodes k that can be potential successors must have $D_{ik} \equiv D_{ij} - 1 \bmod 3$.

Based on this intuition, all we need to do to find the successor matrix S is to run the algorithm from Section 5.1 3 times, each time only considering transitions that have the same value of D_{ik} modulo 3. For each of the 3 runs ($s = 0, 1, 2$), we define the B matrix as follows

$$B_{ij}^{s} = \begin{cases} 1 & D_{ij} \bmod 3 = s \\ 0 & otherwise \end{cases}$$

and run the algorithm from Section 5.1 with A and B^s. This way, we no longer have a dependency on the length L of the path between the 2 nodes and are still able to consider all the possible transitions between vertices to determine the successor matrix. Since this is just a constant factor, the complexity remains the same as Section 5.1 $O(MM(n)\log^2(n))$.

References

[SE92] Seidel, R. On the all-pairs-shortest-path problem. *Proceedings Of The Twenty-fourth Annual ACM Symposium On Theory Of Computing.* pp. 745-749 (1992)

[MR95] Motwani, R. & Raghavan, P. Randomized algorithms. (Cambridge university press,1995)

[DI59] Dijkstra, E. & Others A note on two problems in connexion with graphs. *Numerische Mathematik.* **1**, 269-271 (1959)

[FL62] Floyd, R. Algorithm 97: shortest path. *Communications Of The ACM.* **5**, 345 (1962)

[WA62] Warshall, S. A theorem on boolean matrices. *Journal Of The ACM (JACM).* **9**, 11-12 (1962)