1 Successor Matrix

So far, we studied how to compute the shortest path distances under various matrix products, but never showed how one can compute the paths corresponding to these shortest distances. These notes will focus on how to find the actual paths, by modifying the algorithms we previously studied.

The only type of matrix product we will use throughout the notes is the $(\min, \odot)$ product, defined as $(A \odot B)[i,j] = \min_k(A[i,k] \odot B[k,j])$ for some operation $\odot : \mathbb{Z} \to \mathbb{Z}$. We will only consider operations $\odot$ that makes this matrix product associative. Note that many problems we considered in previous lectures, such as Boolean matrix multiplication or all-pairs shortest paths on directed, weighted graphs, fall under this framework. Also, in these notes, we will use the notation $i \rightsquigarrow j$ to denote a path from i to j, and whether it represents the shortest path or not should be clear from the context.

To motivate the discussion, suppose that we want to compute shortest paths between *all* pairs of nodes. However, in general, the size of the output can be $\Omega(n^3)$, and as a result, many related problems become uninteresting. Thus, we will look for a *successor matrix* instead, which essentially has the same representational power as the list of all shortest paths, but using only $\tilde{O}(n^2)$ memory.

Definition 1.1. *A successor matrix S is an n-by-n matrix, whose (i,j) entry is the node next to i in a shortest path $i \rightsquigarrow j$.*

This definition immediately gives a procedure to retrieve a shortest path between any given pair of nodes using the successor matrix, assuming that the path is simple[1]; start from a node, proceed to the next one that is given by the successor matrix, and repeat. This shows that finding a successor matrix is basically as useful as having the entire list of shortest paths.

Proposition 1. *Given a successor matrix S, for any i, j, a shortest $i \rightsquigarrow j$ path can be found in time linear in its length.*

In the next section, we will study how a successor matrix can be computed using witness matrix computation, which is another concept that is closely related to successor matrices.

2 Witness Matrix

Witness matrices are defined by multiplication of two matrices.

Definition 2.1. *A witness matrix W of the $(\min, \odot)$ matrix multiplication of A and B is given by $W[i,j] = \operatorname{argmin}_k(A[i,k] \odot B[k,j])$.*

In many cases, witness matrices can be used to construct the successor matrix for the shortest paths.

- *Transitive closure via successive squaring.* Consider computing the transitive closure of a graph G by successively squaring the adjacency matrix A, using the Boolean matrix product. Boolean matrix multiplication (BMM) can actually be viewed as $(\max, \times)$ restricted to $\{0,1\}$ entries, where $\times$ is the ordinary multiplication; by negating the entries of one of the matrices, BMM can also be represented as a $(\min, \odot)$ product.

  If there is a path from i to j, then the (i,j) entry of the successor matrix S should give the node next to i in some simple path $i \rightsquigarrow j$. If there is no path, then $S[i,j]$ can be arbitrary (this is consistent with

[1] The shortest paths problems we consider all have the property that there is always a simple shortest path.

the definition of successor matrices). We show that the successor matrix can be computed along with the transitive closure, using algorithm 1.

Algorithm 1: TransitiveClosure(G, A)

$A^{(0)} \leftarrow A$;
For all i and j, $S^{(0)}[i,j] \leftarrow j$;
for $k = 1, \ldots, \log n$ **do**
  $A^{(k)} \leftarrow (A^{(k-1)})^2$;
  $W^{(k)} \leftarrow$ witness matrix of the product above;
  For all i and j, $S^{(k)}[i,j] \leftarrow S^{(k-1)}[i, W^{(k)}[i,j]]$;
$T \leftarrow A^{(\log n)}$;
$S \leftarrow S^{(\log n)}$;
return (T, S);

Algorithm 1 returns the transitive closure T, along with the successor matrix S. To see its correctness, we argue that $S^{(k)}$ is a correct successor matrix for all pairs of nodes i, j such that j is reachable from i via a path of length $\leq 2^k$. If there is a path $i \rightsquigarrow j$ of length at most 2^k, then the corresponding entry of $W^{(k)}$ gives a midpoint node w so that there are paths $i \rightsquigarrow w$ and $w \rightsquigarrow j$, both of which have length at most 2^{k-1}. The successor of path $i \rightsquigarrow j$ can then be retrieved from the successor of path $i \rightsquigarrow w$, which is already computed in $S^{(k-1)}$. The base case $k = 0$ is obtained directly from the adjacency matrix. This shows that $S^{(k)}$ is indeed a correct successor matrix.

- *All-pairs shortest paths for weighted graphs via Zwick's algorithm.* The idea in the previous example can readily be applied to Zwick's algorithm. Recall, from lecture 4, that Zwick's algorithm does a matrix multiplication in each of its $\log_{3/2} n$ iterations, computing the distances between nodes connected via paths of length $\leq (3/2)^j$ in iteration j. For each of these matrix multiplications, we can find the associated witness matrices. The witness matrix for iteration j finds a midpoint of each shortest path of length in $[(3/2)^j, (3/2)^j)$. The midpoint w of any such $i \rightsquigarrow j$ path lies in the "middle third" of the path, and hence similar to algorithm 1, we can find the successor of i on the path by setting it to the successor of i on the $i \rightsquigarrow w$ path computed in the previous iteration.

- *All-pairs shortest paths for unweighted undirected graphs.* We presented a nice algorithm by Seidel for APSP in undirected graphs. For Seidel's algorithm, however, the above approach for computing the successor matrix does not apply as Seidel's algorithm runs recursively on a new graph and it computes the distances in a special way, using the counting power of integer matrix multiplication. Nevertheless, we can show that for undirected graphs, just being given the matrix of pairwise distances is sufficient to obtain the successor matrix in $O(n^\omega)$ time.

  One of the main ideas in Seidel's algorithm was that for all i, j, and a neighbor k of i, $d[i,j]$ and $d[k,j]$ differ by at most 1, where d is the shortest distance matrix. Moreover, any k that satisfies $d[k,j] = d[i,j] - 1$ is a valid successor of i in path $i \rightsquigarrow j$. Now, because $d[k,j]$ can be only 1 away from $d[i,j]$, we know that $d[k,j] \equiv d[i,j] - 1 \pmod 3$ implies that k is a successor. This fact can be used to compute the successor matrix, given the matrix D of distances $D[i,j] = d[i,j]$, as follows.

  For $s = 0, 1, 2$, construct matrices $D^{(s)}$ as below:

$$D^{(s)}[k,j] = \begin{cases} 1 & \text{if } D[k,j] \equiv s - 1 \pmod 3 \\ 0 & \text{otherwise.} \end{cases}$$

  Then, compute the Boolean matrix product $AD^{(s)}$, and the corresponding witness matrix $W^{(s)}$. Now, for each i and j, set $S[i,j] = W^{(s_{ij})}[i,j]$, where $s_{ij} = D[i,j] \mod 3$. To see why this is correct, fix i

and j, and consider the (i,j) entry of the product $A \cdot D^{(s_{ij})}$. The index k that contributes to this entry satisfies $A[i,k] = D^{(s_{ij})}[k,j] = 1$. By construction, this means that $D[k,j] \equiv D[i,j] - 1 \pmod 3$, and from the previous observation, we can conclude that k is a correct successor of i in path $i \rightsquigarrow j$.

3 Computing Witness Matrices

In this section, we show an algorithm for computing witness matrices associated with the matrix product $A \odot B$, so that the successor matrix can be computed as described in the previous part.

Before handling the general case, we first focus on an easier variant of the problem, in which the witnesses are unique.

Definition 3.1. *A* unique witness *matrix U for the $(\min, \odot)$-product of A and B is defined as $U[i,j] = k$ whenever k is the unique minimizer of $A[i,k] \odot B[k,j]$, and $U[i,j]$ is arbitrary otherwise.*

In the case of Boolean matrix multiplication, computing a unique witnesses matrix is easy; let A' be a matrix defined by $A'[i,k] = kA[i,k]$. Then, the integer matrix product $A'B$ directly gives the unique witness matrix. Even under general matrix products, computing a unique witness matrix isn't much harder than computing the matrix product itself.

Lemma 3.1. *If the $(\min, \odot)$-product takes $T(n)$ time, then computing a unique witness matrix takes $O(T(n) \log n)$ time.*

Proof. We use a procedure that determines the witnesses bit by bit. For a subset S of $\{1, \ldots, n\}$, let's write $A[\cdot, S]$ to denote the submatrix of A, formed by taking the columns that are indexed by S. Define $A[S, \cdot]$ similarly. Now, for each $b = 1, \ldots, \log n$, let $S_b = \{k : b\text{th bit of } k \text{ is } 1\}$, and compute $C_b \leftarrow A[\cdot, S_b] \odot B[S_b, \cdot]$. For every i and j, if $C_b[i,j] = C[i,j]$, we set the bth bit of $W[i,j]$ to 1, and 0 otherwise. This procedure computes the $(\min, \odot)$ product $\log n$ times, so the total running time is $O(T(n) \log n)$. It is also easy to see why this algorithm is correct; fix i and j that have a unique witness k. Then at every b, the bth bit of $W[i,j]$ will be set correctly . □

Finally, we show how to compute the more general witness matrix, given an algorithm for computing unique witnesses. To do this, we do random samplings and use the algorithm above. First, we claim the following:

Claim 1. *Assume that the pair (i,j) has c witnesses, where $n/2^{s+1} \le c \le n/2^s$. That is, the number of indices k such that $A[i,k] \odot B[k,j]$ is c. Let S be a random subset of $\{1, \ldots, n\}$ of size 2^s. Then, S contains a unique witness for i, j with at least some constant probability.*

Proof. Let W be the set of witnesses of i, j, and let W have size c. Then,

$$\mathbf{Prob}[|S \cap W| = 1] \approx 2^s \cdot \frac{c}{n} \cdot \left(1 - \frac{c}{n}\right)^{2^s - 1} \ge 2^s \cdot \frac{1}{2^{s+1}} \cdot \left(1 - \frac{1}{2^s}\right)^{2^s - 1} \ge \frac{1}{2e}.$$

□

This fact can be used to design a randomized algorithm that outputs a correct witness matrix. We do not know the number of witness for each individual (i,j) pair, but we know that it has to be contained in the intervals $[n/2^{s+1}, n/2^s]$ for some $s = 0, 1, \ldots, \log n$. The idea is to loop over all possible values of s, and determine the witnesses for all the (i,j) index pairs for which the number of witnesses fall into the right interval. Formally, the algorithm can be written as below.

For every outermost iteration, each $W[i,j]$ is filled with a correct witness with some constant probability, and once it is determined, it never changes. There are n^2 entries to be determined, so the expected number of outermost iterations is $O(\log n)$. We conclude by this final result.

Theorem 3.1. *If the $(\min, \odot)$-product is computable in $T(n)$ time, then finding the witness matrix takes $O(T(n) \log^3 n)$ time.*

Algorithm 2: WitnessMatrix(A, B)

```
C ← A ⊙ B;
repeat
    for s = 0, 1, ..., log n do
        S ← random subset of {1, ..., n} of size 2^s;
        Attempt to find the unique witness matrix W' of the matrix product A[·, S] ⊙ B[S, ·];
        for i, j = 1, ..., n do
            if A[i, W'[i, j]] ⊙ B[W'[i, j], j] = C[i, j] then
                W[i, j] ← W'[i, j];
until all elements of W are determined;
return W;
```