1 The Distance Product

Last time we defined the distance product of $n \times n$ matrices:

$$(A \star B)[i,j] = \min_k \{A(i,k) + B(k,j)\}$$

Theorem 1.1. *Given two $n \times n$ matrices A, B over $\{-M, M\}$, $A \star B$ can be computed in $\tilde{O}(Mn^\omega)$ time.*

2 Oracle for All-Pairs Shortest Paths

Theorem 2.1 (Yuster, Zwick '05)**.** *Let G be a directed graph with edge weights in $\{-M, M\}$ and no negative cycles. Then in $\tilde{O}(Mn^\omega)$ time, we can compute an $n \times n$ matrix D such that for every $u, v \in V$,* ***with high probability****:*

$$(D \star D)[u,v] = d(u,v)$$

It is called distance product because the shortest path from node u to node v can be created by first going on the shortest path from u to an intermediate node k and then from k to v. We are searching for the most convenient node k that will minimize the sum of the two.

Note that this does not immediately imply a fast APSP algorithm, because D may have large entries, making computing $D \star D$ expensive. However, for *single source shortest paths* we have the following corollary:

Corollary 2.1. *Let $G = (V, E)$ be a directed graph with edge weights in $\{-M, M\}$ and no negative cycles. Let $s \in V$. Then single-source shortest path from s can be computed in $\tilde{O}(Mn^\omega)$ time.*

Proof. By Theorem 2.1, we can compute an $n \times n$ matrix D such that $D \star D$ is the correct all-pairs shortest-paths matrix, in $\tilde{O}(Mn^\omega)$ time.

Then for all $v \in V$, we know that:

$$d(s,v) = \min_k D[s,k] + D[k,v]$$

Computing this for all $v \in V$ only takes $O(n^2)$ time. Since $\omega \geq 2$, this entire computation is in $\tilde{O}(Mn^\omega)$ time. □

Similarly, we can show that detecting negative cycles is fast since any negative cycle contains a simple cycle of negative weight, and thus corresponds to a path from i to i for some i of length $\leq n$.

Corollary 2.2. *Let G be a directed graph with edge weights in $\{-M, M\}$. Then negative cycle detection can be computed in $\tilde{O}(Mn^\omega)$ time.*

Note: For notational convenience, suppose that A is an $n \times n$ matrix and that $S, T \subseteq \{1, \ldots, n\}$. Then $A[S,T]$ is the submatrix of A consisting of rows indexed by S and columns indexed by T.

We now prove our main theorem:

The main algorithm uses again randomness and the hitting set lemma but now we do not take freshly random samples every time, but instead we take B_{j+1} to be a random sample from B_j.

Proof of Theorem 2.1. Let $\ell(u,v)$ be the number of nodes on a shortest u to v path.

Algorithm 1: YZ(A)

A is a weighted adjacency matrix;
Set $D \leftarrow A$;
Set $B_0 \leftarrow V$;
for $j = 1, \ldots, \log_{3/2} n$ **do**
  Let D' be D but with all entries larger than $M(3/2)^j$ replaced by ∞;
  Choose B_j to be a random set of B_{j-1} of size $S_j = \frac{c \cdot n}{(3/2)^j} \log n$;
  Compute $D_j \leftarrow D'[V, B_{j-1}] \star D'[B_{j-1}, B_j]$;
  Compute $\overline{D}_j \leftarrow D'[B_j, B_{j-1}] \star D'[B_{j-1}, V]$;
  foreach $u \in V, b \in B_j$ **do**
    Set $D[u,b] = \min(D[u,b], D_j[u,b])$;
    Set $D[b,u] = \min(D[b,u], \overline{D}_j[b,u])$;
return D;

We claim that Algorithm 1 is our desired algorithm. (desired running time and correctness)

Running Time: In iteration j, we multiply an $n \times \tilde{O}\left(\frac{n}{(3/2)^{j-1}}\right)$ matrix by a $\tilde{O}\left(\frac{n}{(3/2)^{j-1}}\right) \times \tilde{O}\left(\frac{n}{(3/2)^j}\right)$ matrix, where all entries are at most $(3/2)^j M$ (we will show iteration j only needs to consider paths with at most $(3/2)^j$ nodes).

Hence the runtime for iteration j is $\tilde{O}\left(M(3/2)^j (3/2)^j (\frac{n}{(3/2)^j})^\omega\right) = \tilde{O}\left(\frac{Mn^\omega}{(3/2)^{j(\omega-2)}}\right)$. The term $((3/2)^j (\frac{n}{(3/2)^j})^\omega)$ is due to the blocking that we use when computing D_j and $\overline{D}_j$. Over all iterations, the running time is, asymptotically, ignoring polylog factors,

$$Mn^\omega \sum_j ((3/2)^{\omega-2})^j \leq \tilde{O}(Mn^\omega).$$

If $\omega > 2$, one of the log factors in the $\tilde{O}$ can be omitted.

Correctness: We will prove the correctness by proving two claims.

Claim 1: For all $j = 0, \ldots, \log_{3/2} n$, $v \in V$, $b \in B_j$, if $\ell(v,b) < (3/2)^j$ then w.h.p. after iteration j, $D[v,b] = d(v,b)$

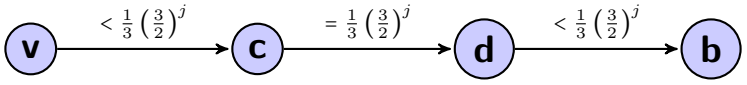
Proof of Claim 1: We will prove it via induction. The base case ($j = 0, \ell(v,b) < (3/2)^0 = 1$) is trivial, since the distance is for one-hop paths is exactly the adjacency matrix. Now, assume the inductive hypothesis is true for $j-1$, that is we have stored correctly $D[u,b] = d[v,b]$ if the shortest path (v,b) has length $\ell(v,b) < (3/2)^{j-1}$. We will show correctness for j. Consider some $v \in V$ and $b \in B_j$. We consider two possible cases depending on how far is node b from v.

Case I: $\ell(v,b) < (3/2)^{j-1}$ (b is near)

But then $b \in B_j \subset B_{j-1}$. By our inductive hypothesis, $D[v,b] = d(v,b)$ w.h.p.!

Case II: $\ell(v,b) \in [(3/2)^{j-1}, (3/2)^j)$ (b is far)

We will need to use our "middle third" technique we saw from last lecture.



We can choose $c, d \in V$ such that:

$$\ell(v,c) < \frac{1}{3}\left(\frac{3}{2}\right)^j$$
$$\ell(d,b) < \frac{1}{3}\left(\frac{3}{2}\right)^j$$
$$\ell(c,d) = \frac{1}{3}\left(\frac{3}{2}\right)^j < \left(\frac{3}{2}\right)^{j-1}$$

By a hitting set argument, if c is a large enough constant, $B_{j-1} \cap$ "middle third" $\neq \varnothing$ (w.h.p. depending on c) since $|B_{j-1}| = c\frac{n}{(3/2)^{j-1}} \log n$.

Let x in $B_{j-1} \cap$ "middle third". Then $\ell(v,x) \leq \ell(v,c) + \ell(c,d) < \frac{2}{3}(\frac{3}{2})^j = (\frac{3}{2})^{j-1}$. Since $x \in B_{j-1}$, by induction $D[v,x] = d(v,x)$ w.h.p. at iteration j. By a similar argument we get that w.h.p. $D[x,b] = d(x,b)$ at iteration j (at the beginning of iteration j).

Hence after this iteration, $D[v,b] \leq D[v,x] + D[x,b] = d(v,b)$.

As a small technical note, we will need to actually remove entries larger than $(3/2)^j M$ from D before multiplying, but they are not needed.

Claim 2: For all $u, v \in V$, w.h.p. $(D \star D)[u,v] = d(u,v)$.

Proof of Claim 2: Fix $u, v \in V$, and let j be such that $\ell(u,v) \in [(3/2)^{j-1}, (3/2)^j)$. Look at a shortest path between u and v. Its middle third hence has a length of $(1/3)(3/2)^j$.

But then w.h.p. B_j hits this path at some $x \in V$ such that $\ell(u,x), \ell(x,v) < (3/2)^{j-1}$. By Claim 1, $D(u,x) = d(u,x)$ and $D(x,b) = d(x,b)$. Hence:

$$d(u,v) \leq (D \star D)[u,v] \leq min_{x \in B_{j-1}} D(u,x) + D(x,v) \leq d(u,v)$$

This completes the proof.

□

3 Node-Weighted All-Pairs Shortest Paths

Now we will see an interesting variant of the APSP which is called node-weighted all pairs shortest paths problem (now the weights are associated with the nodes instead of the edges) for which we can have a truly subcubic solution despite the fact that for APSP no known truly subcubic solution exists. This gap might be inherent since it is difficult to map $\approx n^2$ weight values to only n and still maintain the pairwise shortest paths information.

Here we prove a theorem by Chan [1]

Theorem 3.1. *APSP with node weights can be computed in $O(n^{\frac{9+\omega}{4}})$ or $O(n^{2.84})$ time.*

The idea is to compute long paths ($> s$ hops) via a hitting set argument and running multiple calls to Dijkstra's algorithm, in a running time of $\tilde{O}(\frac{n^3}{s})$. Then, handle short paths ($\leq s$ hops) in $O(sn^{\frac{3+\omega}{2}})$ time via a specialized matrix multiplication.

Let G be a directed graph with node weights $w : V \to Z$. Suppose we just wanted to compute distances over paths of length two.

Let A be the *unweighted* adjacency matrix. Notice that $d_2(u,v) = w(u) + w(v) + \min\{w(j) \mid A[u,j] = A[j,v] = 1\}$ (we are looking for our cheapest neighbour to go through).

Suppose we made two copies of A, and sorted one's columns by $w(j)$ in nondecreasing order, and the others rows by $w(j)$ in nondecreasing order.

Then it would suffice to compute $\min\{j \mid A[i,j] = A[j,k] = 1\}$, or the "minimum witnesses" matrix product. We use an algorithm provided by Kowaluk and Lingas [3]:

Lemma 3.1 (Kowaluk, Lingas '05)**.** *Minimum witnesses of A, B ($n \times n$ matrices) is in $O(n^{2.616})$ or $O(n^{2+\frac{1}{4-\omega}})$ time.*

Note that this algorithm has been improved on by Czumaj, Kowaluk, and Lingas [2]

Proof. Let p be some parameter that we will choose later. Bucket A by columns into buckets of size p. Bucket B by rows into buckets of size p.

For every bucket $b \in \{1, \ldots, \frac{n}{p}\}$, compute $A_b \cdot B_b$ (boolean matrix product). This takes $O((\frac{n}{p})^2 p^\omega)$ time each, or $O(n^2 p^{\omega-2})$ time each. But there are $\frac{n}{p}$ of these, so this takes $O(\frac{n^3}{p^{3-\omega}})$ time total.

Then for all $i, j \in \{1, \ldots, n\}$, do the following. Let b_{ij} be the smallest b such that $(A_b \cdot B_b)[i,j] = 1$. Hence we can just try all the choices of k in bucket b_{ij}, and return the smallest k such that $A_b[i,k]B_b[k,j] = 1$. This is just n^2 exhaustive searches, so this step runs in $O(n^2 p)$ time. The intuition for this is that after the sorting, the minimum witness k for which we have $A_b[i,k]B_b[k,j] = 1$ is the most convenient neighbour of both i and j to go through if we are seeking for the shortest path between i, j.

Setting the above running times equal ($\frac{n^3}{p^{3-\omega}} = n^2 p$) and balancing, we get that we should set $p = n^{\frac{1}{4-\omega}}$ to make the overall time $O(n^{2+\frac{1}{4-\omega}})$. □

Intuition: The blocking we do to our matrices after the sorting has the following interpretation: On the sorted adjacency matrix, when we do blocking is like grouping together the nodes according to their weight values. So if we had 2 blocks then we would have 2 node groups: cheap nodes, expensive nodes. Then by the multiplication process we are trying to figure out which nodes we can reach passing through the different node groups.

Now that we saw how to deal with distance 2 we can proceed with longer paths. How can we compute distances for paths that are longer than two hops?

We will have two parameters p, s that we will choose later so that we minimise the runtime. Parameter s is for distinguishing the "short" paths from the "long" paths. Parameter p is again related to the blocking of the matrices that is convenient. For each $\ell \leq s$, we want to compute D_ℓ such that:

$$D_\ell[u,v] = d(u,v) - w(u) - w(v) \text{ if } \ell(u,v) = \ell$$
$$D_\ell[u,v] = \min_{j \in N(u)} \{w(j) + D_{\ell-1}[j,v]\}$$

This gives rise to a new matrix product! Suppose we are given $D_{\ell-1}$. Let $\overline{D}_{\ell-1}[u,v] = w(u) + D_{\ell-1}[u,v]$. Then we are interested in $(A \odot \overline{D}_{\ell-1})[u,v] = \min\{\overline{D}_{\ell-1}[j,v] \mid A[u,j] = 1\}$.

We can compute this product as follows. Again, let p be a parameter that we will choose later. Sort the columns of $\overline{D}_{\ell-1}$, using $O(n^2 \log n)$ time. Then partition each column into blocks of length p.

Let $D_b[u,v] = 1$ if $\overline{D}_{\ell-1}[u,v]$ is between the $\left(b\frac{n}{p}\right)^{th}$ and the $\left((b+1)\frac{n}{p}\right)^{th}$ element of column v.

Compute the boolean matrix product of A and D_b for all b. Notice that $(A \cdot D_b)[u,v] = 1$ iff there exists an x such that $A[u,x] = 1$ and $\overline{D}_{\ell-1}[x,v]$ is among the b^{th} block of p elements in the sorted order of the v^{th} column. We can finish via an exhaustive search, trying all j such that $D_{\ell-1}[j,v]$ is in the b^{th} block of column v.

This takes $O(\frac{n}{p} n^\omega)$ time for multiplications, and $O(n^2 p)$ time for the exhaustive search. This yields $O(n^{\frac{3+\omega}{2}})$ time after balancing. However, we need to do this s times.

The overall runtime is hence $O(n^{\frac{3+\omega}{2}} s + n^3/s)$, which becomes $O(n^{\frac{9+\omega}{4}})$ time after balancing.

Today we will present and solve two variants of All Pairs Shortest Paths (APSP) in $O(n^{3-\delta})$ time for some constant $\delta > 0$. In doing so, we will also introduce two more matrix products, namely the $(\min, \leq)$ product and the dominance product.

4 Earliest Arrivals

The first variant of APSP we will study is the Earliest Arrivals problem. We are given a set V consisting of n airports and a set F of n flights. Each flight $f \in F$ consists of a a source airport $s \in V$, a destination airport $t \in V$, a departure time, and an arrival time.

Definition 4.1. *A valid itinerary from s to t is a sequence of flights $f_1, \dots f_k$ such that, for all $i \in \{1, \dots, k\}$, $source(f_{i+1}) = destination(f_i)$ and $departure(f_{i+1}) \geq arrival(f_i)$.*

The Earliest Arrivals problem is to compute, for all airports $u, v \in V$, the earliest arrival time over all valid itineraries. This problem has a natural graph interpretation. Consider a bipartite graph $G = (V \cup F, E)$. For each flight $f \in F$, we add a directed edge $(source(f), f)$ to E with weight $departure(f)$. Then, we add another directed edge $(f, destination(f))$ with weight $arrival(f)$.

On this graph, a valid itinerary is a $s \to t$ path such that all of the edges form a nondecreasing sequence, and the arrival time is given by the last edge weight. Therefore, Earliest Arrivals is equivalent to finding, $\forall s, t \in V$, the minimum last edge weight over all nondecreasing $s \to t$ paths.

Definition 4.2. *Let A, B be $n \times n$ matrices. The $(\min, \leq)$ product of A and B, denoted $A \circledcirc B$ is given by*

$$(A \circledcirc B)(i, j) = \min_k \{B(k, j) \mid A(i, k) \leq B(k, j)\}$$

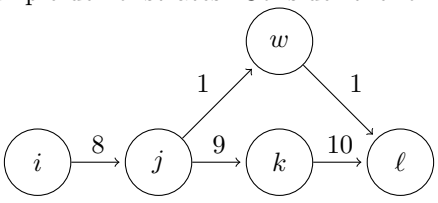
or ∞ if no such k exists.

If we define the adjacency matrix A of G in the natural way,

$$A(i, j) = \begin{cases} w(i, j) & (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

(here we assume that all weights are positive) then we find that $(A \circledcirc A)(i, j)$ is the minimum last edge weight over paths of length 2. Iterating this relationship, we find that $\underbrace{(A \circledcirc \cdots \circledcirc A)}_{\ell - 1 \text{ times}} \circledcirc A(i, j)$ is the minimum last edge over all paths of length ℓ.

We must be careful, however, because the $(\min, \leq)$ product is not associative in general, as the following example demonstrates. Consider the following graph.



Observe that $(A \circledcirc A) \circledcirc A(i, \ell) = 10$, but $A \circledcirc (A \circledcirc A)(i, \ell) = \infty$. Consequently, we cannot simply use successive squaring to solve the Earliest Arrivals problem. Instead, our approach will be to compute Earliest Arrivals for "short paths" and use the random sampling technique developed in previous lectures to handle "long paths." The rough idea is as follows.

Algorithm 2: Earliest Arrivals(G)

Form adjacency matrix A
Set $D = A$
for $i := 1$ **to** s **do**
  Compute $D = D \otimes A$
end for
Compute a random sample, S, of size $c * \frac{n}{s} \log n$
for all $x \in S$ **do**
  Compute All Pairs Earliest Arrivals for paths through x
end for
for all $i, j \in V$ **do**
  $EA(i,j) = \min_{x \in S}$ min last edge weight over valid itineraries of the form $i \to x \to j$
  $EA(i,j) = \min\{EA(i,j), D(i,j)\}$
end for
Return EA

It is left as a homework exercise to show that, for any node $x \in S$ we can compute all pairs earliest arrival for paths through x in $O(n^2 \log n)$ time.

Lemma 4.1. *If the* $(\min, \leq)$ *product of* $n \times n$ *matrices can be computed in* $O(n^c)$ *time, then we can solve Earliest Arrivals in* $O(n^{\frac{3+c}{2}})$ *time.*

Proof of Lemma 4.1. Using the algorithm sketched above, we obtain a runtime of $O(\frac{n^3}{s} + s(n^c))$. Optimizing over s, we set $s = n^{\frac{3-c}{2}}$ and obtain a total runtime of $O(n^{\frac{3+c}{2}})$, as required. □

5 All Pairs Bottleneck Paths

(we skipped that in class but you should still study it)

Let graph $G = (V, E)$ be a graph with edge weights given by $w : E \to \mathbb{Z}$.

Definition 5.1. *Given a path* p *in* G*, its bottleneck edge is the edge of minimum weight.*

Definition 5.2. *The All Pairs Bottleneck Paths problem (APBP) is to find, for all pairs* $u, v \in V$*, the maximum bottleneck weight over all* $u \to v$ *paths.*

In order to tackle this problem, we need to define another matrix product.

Definition 5.3. *Let* A *and* B *be* $n \times n$ *matrices. The* $(\max, \min)$ *product of* A *and* B*, denoted* $A \circledast B$ *is given by*

$$(A \circledast B)(i,j) = \max_k \min(A(i,k), B(k,j))$$

Observe that that $(\max, \min)$ product is precisely the bottleneck path problem in graphs with diameter 2. It is left as an exercise to verify that $\circledast$ is associative. Since $\circledast$ is associative and $A \circledast A$ gives the maximum bottleneck for length 2 paths, we can solve All Pairs Bottleneck Paths using successive squaring. This gives us the following lemma.

Lemma 5.1. *If the* $(\max, \min)$ *product of two* $n \times n$ *matrices can be computed in* $\tilde{O}(n^c)$ *time, then we can solve All Pairs Bottleneck Paths in* $\tilde{O}(n^c)$ *time.*

In fact, we will show that computing $\circledast$ is equivalent to two $\otimes$ product computations.

Lemma 5.2. *If there is an* $O(n^c)$ *algorithm for computing* $(\min, \leq)$ *products, there is an* $O(n^c)$ *algorithm for computing* $(\max, \min)$ *products.*

Proof. Consider the matrix product defined by $(A \circledcirc B)(i,j) = \max_k\{A(i,k) \mid A(i,k) \le B(i,k)\}$. Note that this product is in fact a $(\min, \le)$ product. In particular, it is the product $-B \circledcirc -A$ obtained by negating all of the entries $a_{i,j}$ in A and $b_{i,j}$ in B and then swapping matrices A and B, i.e. $(A \circledcirc B)(i,j) = -(-B \circledcirc -A)(i,j)$. Using this product, we can compute

$$(A \circledast B)(i,j) = \max\{(A \circledcirc B)(i,j), (B \circledcirc A)(i,j)\}.$$

Therefore, we can compute $A \circledast B$ using two $(\min, \le)$ computations, as required. □

By the above discussion, we can solve both the All Pairs Earliest Arrivals problem and the All Pairs Bottleneck Path problem with a fast algorithm for computing $(\min, \le)$ products. The rest of this writeup is dedicated to finding such an algorithm.

6 A Fast Algorithm for Computing $(\min, \le)$ Products

We will use another special matrix product in our algorithm for computing $(\min, \le)$.

Definition 6.1. *The dominance product of $n \times n$ matrices A and B, denoted $A \odot B$, is given by*

$$(A \odot B)(i,j) = |\{k \mid A(i,k) \le B(k,j)\}|$$

Theorem 6.1. *(Matousek) The dominance product of two $n \times n$ matrices can be computed in $O(n^{\frac{3+\omega}{2}})$ time.*

Theorem 6.2. *If dominance product can be computed in $O(n^d)$ time, then the $(\min, \le)$ product can be computed in $O(n^{\frac{3+d}{2}})$ time.*

Assuming 6.1, we first prove 6.2.

Proof of Theorem 6.2. Let A, B be two $n \times n$ matrices. We will compute $A \circledcirc B$ as follows.

1. Sort each column j of matrix B
2. Fix parameter p. Partition each sorted column into $\frac{n}{p}$ consecutive buckets of p elements each. Name the buckets so that for all buckets $b \le b'$, $\forall B(i,j)$ in bucket b of column j, and $\forall B(\ell,j)$ in bucket b' of j, we have $B(i,j) \le B(\ell,j)$.
3. For each $b \in \{1, \ldots, \frac{n}{p}\}$, create an $n \times n$ matrix B_b such that
$$B_b(i,j) = \begin{cases} B(i,j) & \text{if } B(i,j) \text{ in bucket } b \text{ of column } j \\ -\infty & \text{otherwise} \end{cases}$$
4. Compute for all buckets b, $A \odot B_b$, which is
$$(A \odot B_b)(i,j) = \begin{cases} \neq 0 & \text{if } \exists k \text{ such that } B_b(k,j) \neq -\infty \text{ and } A(i,k) \le B(k,j) \\ 0 & \text{otherwise} \end{cases}$$
5. For all i,j determine $b_{i,j}$ = smallest b such that $(A \odot B_b)(i,j) \neq 0$. This is equivalent to
$$\min\{B[k,j] \mid B(k,j) \text{ in bucket } b(i,j) \text{ and } A(i,k) \le B(k,j)\}.$$
Therefore, we can use brute force, as follows. For all i,j examine each $B(k,j)$ in bucket $b_{i,j}$ of j, compare it with $A(i,k)$ and output the minimum $B(k,j)$ for which $A(i,k) \le B(k,j)$. Observe that this is $(A \circledcirc B)(i,j)$.

The running time of this algorithm is dominated by computing the dominance product in step 4 and brute force in step 5. Using 6.1, we can compute dominance product in $O(n^d)$ time. Therefore, it takes $O(\frac{n^{d+1}}{p})$ time to compute the required $\frac{n}{p}$ dominance products. The brute force step takes $O(n^2 p)$ time. Choosing $p = n^{\frac{d-1}{2}}$, we obtain a total runtime of $O(n^{\frac{3+d}{2}})$, as desired. □

It remains to prove 6.1

Proof of Theorem 6.1. Let A, B be $n \times n$ matrices. We compute $A \odot B$ as follows.

1. For all j, sort the set of entries of column j of A and row j of B together. This produces a list of $2n$ elements.
2. Partition this list into buckets of p elements each. There are $\frac{2n}{p}$ buckets for each j.
3. For all $b \in \{1, \ldots, \frac{2n}{p}\}$, create $n \times n$ matrices

$$A_b(i,j) = \begin{cases} 1 & \text{if } A(i,j) \text{ is in bucket } b \text{ of } j \\ 0 & \text{otherwise} \end{cases}$$

$$B_b(j,k) = \begin{cases} 1 & \text{if } \exists b' > b \text{ such that } B(j,k) \text{ is in bucket } b' \text{ of } j \\ 0 & \text{otherwise} \end{cases}$$

4. For distinct buckets b, compute the integer matrix product

$$(A_b B_b)(i,j) = |\{k \mid A(i,k) \in b, A(i,k) \leq B(k,j), \text{ and } B(k,j) \notin b\}$$

   We handle identical buckets b using brute force search. For all i, j and buckets b, compare $A(i,k)$ with all $B(k,j)$ in the same bucket as $A(i,k)$ and update the sum in the output.

The brute force step requires $O(n^2 p)$ time. Then, we require $\frac{n}{p} n^\omega$ time to perform matrix multiplications. We minimize p by taking $p = n^{\frac{3-\omega}{2}}$ to obtain a final running time of $O(n^{\frac{3+\omega}{2}})$, as desired. □

7 Conclusions

In this lecture we saw many different matrix products that are useful for many applications. The distance product is useful for APSP oracles and a slight variant of it is useful in the Node-Weighted APSP problem (NW-APSP). The $(\min, \leq)$ product (which is not associative) is useful when searching for non-decreasing paths and has applications in the All Pairs Earliest Arrivals problem (APEA). The (max,min) product is used when searching for All Pairs Bottleneck Paths (APBP). Finally, we defined the dominance product. Using all these, we concluded that NW-APSP, APEA, APBP are truly subcubic, having $O(n^{2.84})$,$O(n^{2.9})$,$O(n^{2.8})$ time algorithms respectively, if we use the current value of ω. However, it remains a major open question to find a truly subcubic algorithm for general APSP.

References

[1] T.M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.*, 39(5):20752089, 2010.

[2] Artur Czumaj, Miroslaw Kowaluk, and Andrzej Lingas. Faster algorithms for finding lowest common ancestors in directed acyclic graphs. *Theoretical Computer Science*, 380(1):3746, 2007.

[3] Miroslaw Kowaluk and Andrzej Lingas. Lca queries in directed acyclic graphs. In *Automata, Languages and Programming*, pages 241248. Springer, 2005.

[4] Raphael Yuster and Uri Zwick. Answering distance queries in directed graphs using fast matrix multiplication. In *FOCS*, pages 389396, 2005.