The goal of this lecture is to show algorithms for APSP in directed graphs, where the edge weights are integers in $\{-M, \ldots, M\}$ for an integer $M \geq 1$. In the previous lecture we gave an $O(n^\omega \log n)$ time algorithm for APSP in undirected unweighted graphs (by Seidel). Shoshan and Zwick showed how to extend Seidel's algorithm to obtain an $\tilde{O}(Mn^\omega)$ time algorithm for undirected graphs with integer weights in $\{-M, \ldots, M\}$.

Let $G = (V, E)$ be a given directed graph with weights $w(\cdot) \in \{-M, \ldots, M\}$ on its edges. We assume that the graph does not have negative cycles.

In the first part of the lecture, we give an algorithm that runs in $\tilde{O}(\sqrt{M}n^{(3+\omega)/2})$ time. For the second part, we will show Zwick's algorithm that improves the previous algorithm. The running time of Zwick's algorithm is $\tilde{O}(M^{\frac{1}{4-\omega}}n^{2+\frac{1}{4-\omega}})$, which is faster than $\tilde{O}(\sqrt{M}n^{(3+\omega)/2})$ when $M = O(1)$ and $\omega > 2$. Zwick's algorithm can be improved to $\tilde{O}(M^{0.752}n^{2.5286})$, using rectangular matrix multiplication, but we won't cover the rectangular matrix multiplication part.

1 An $\tilde{O}(\sqrt{M}n^{(3+\omega)/2})$ Algorithm

Given a parameter k, we call a path *short* if it uses at most k vertices, and *long* otherwise. At a high level, the algorithm we present considers considers long shortest paths and short shortest paths separately. For short paths, it uses matrix multiplication; for long paths, it uses sampling and single source shortest paths (SSSP).

Throughout this lecture, we use $d(i, j)$ to denote the distance from i to j, and use $\ell(i, j)$ to denote the number of vertices on a fixed shortest path from i to j. If there are multiple shortest paths from i to j, then we pick one of the shortest paths $P_{i,j}$ and we let $\ell(i, j)$ be the number of nodes on $P_{i,j}$.

1.1 Handling Long Paths

Recall that for every pair of vertices i, j we have picked a representative shortest path $P_{i,j}$, and $\ell(i, j)$ is the number of nodes on $P_{i,j}$. Here we consider all $P_{i,j}$ with $\ell(i, j) > k$.

We use the following "Hitting Set Lemma":

Lemma 1.1. *Let $S = \{S_1, \ldots, S_N\}$ be a collection of N sets where for every $i \in [N]$, we have $S_i \subseteq [L]$ for an integer L and $|S_i| > k$. Then a uniformly random subset T of $[L]$ of size at least $C \cdot (L/k) \ln N$, with probability at least $1 - 1/N^{C-1}$ will have $S_i \cap T \neq \emptyset$ for every $i \in [N]$.*

Let's apply the Hitting Set Lemma where $N = n^2$, and S is the set of $\leq n^2$ paths $P_{i,j}$ with $\ell(i, j) > k$. We think of the paths as subsets of the vertex set V which we associate with $[n]$. From the lemma, we know that if $T \subseteq V$ is a uniformly random subset of $\Theta(\frac{n}{k} \log n)$ nodes, then $T \cap P_{i,j} \neq \emptyset$ for every pair of i, j, with high probability.

Thus, after picking a random T, we know that it contains a node on every long shortest path (with high probability). We can run SSSP to and from every vertex $s \in T$ to compute $d(i, s)$ and $d(s, i)$ for every $i \in V$. Then for every pair $i, j \in V$ with a long shortest path $P_{i,j}$, we have $d(i, j) = \min_{s \in T} d(i, s) + d(s, j)$. Thus, for long paths, it suffices to perform $O(|T|)$ SSSP calls, and use $O(n^2|T|)$ time to use the SSSP results to compute $d(i, j)$ where $\ell(i, j) \geq k$. It remains to discuss how to perform SSSP.

If all edge weights are nonnegative, we can run Dijkstra's algorithm to and from every vertex in T, which will take $O(n^2|T|)$ time. In order to handle any negative weight edges in the graph, we can use *Johnson's trick.*

Claim 1. *For every $i, j \in V$, $w'(i, j) \geq 0$.*

Proof. By the triangle inequality, $d(q, j) \leq d(q, i) + w(i, j)$, so $w'(i, j) = w(i, j) + d(q, i) - d(q, j) \geq 0$. □

Algorithm 1: Johnson's trick, $G = (V, E)$, with edge weights $w : E \to \{-M, \ldots, M\}$

Add a new node q.
Add an edge with weight 0 from q to every vertex $v \in V$.
Compute SSSP from q (Look for an $\tilde{O}(Mn^{\omega})$ time algorithm in the next lecture).
foreach $(i, j) \in E$ **do**
  $w'(i, j) := w(i, j) + d(q, i) - d(q, j)$

Let $d'(i, j)$ be the distance from i to j using weights w'.

Claim 2. *For every $i, j \in V$, $d'(i, j) = d(i, j) + d(q, i) - d(q, j)$.*

Proof. For any path $v_1 \to \cdots \to v_l$, we have

$$w'(P_{i,j}) = \sum_{k=1}^{l-1} w'(v_k, v_{k+1}) = \sum_{k=1}^{l-1} w(v_k, v_{k+1}) + d(q, v_k) - d(q, v_{k+1}) = w(P_{i,j}) + d(q, v_1) - d(q, v_l).$$

This means that for any path, its weight under w' only depends on the start, end, and its weight under w, so the shortest path between i and j remains the same. Thus, $d'(i, j) = d(i, j) + d(q, i) - d(q, j)$. $\square$

Claim 2 suggests that we can compute SSSP under w' and then recover d from d', and Claim 1 suggests that we can use Dijkstra's algorithm to compute SSSP under w'. It takes $\tilde{O}(Mn^{\omega})$ time to perform Johnson's trick using the algorithm you will see in the next lecture, and $\tilde{O}(n^2|T|)$ time to run Dijkstra's algorithm to and from every vertex in T.

Overall, it takes $\tilde{O}(Mn^{\omega} + n^2|T|)$ time to handle long paths. Recall that $|T| = \Theta(\frac{n}{k} \log n)$, so the running time becomes $\tilde{O}(Mn^{\omega} + \frac{n^3}{k})$.

1.2 Handling Short Paths

For short paths, we want to compute $d(i, j)$ for $i, j \in V$ where $\ell(i, j) < k$. For this purpose, we define the $(\min, +)$-product (a.k.a distance-product or funny product).

Definition 1.1. *For two n by n matrices A, B, the $(\min, +)$-product $C = A \star B$ is defined by*

$$C(i, j) = \min_k \{A(i, k) + B(k, j)\}, \forall i, j \in [n].$$

Although we will not prove it, a theorem of Fischer and Meyer'1971 states that $(\min, +)$-product is asymptotically equivalent to APSP: if $A \star B$ can be computed in $T(n)$ time, then APSP in weighted graphs can be done in $O(T(n))$ time, and vice-versa. It is not hard to show that APSP can be used to solve $(\min, +)$-product, and that APSP can be solved using $(\min, +)$-product with successive squaring, at a cost of a logarithmic factor. Fisher and Meyer's result removes the logarithmic overhead.

It turns out that $(\min, +)$-product can be computed relatively quickly when the matrix entries are integers with small absolute values.

Theorem 1.1. *If A, B are $n \times n$ matrices with entries in $\{-M, \ldots, M\} \cup \{\infty\}$, then $A \star B$ can be computed in $\widetilde{O}(Mn^{\omega})$ time.*

Proof. First note that we can assume that there are no infinite entries - replace each ∞ with $3M + 1$. These entries can never be used in a $(\min, +)$-product entry (unless that entry is ∞ itself) since any finite $(\min, +)$-product entry is at most $2M$ and even if one uses a $-M$ entry together with the $3M + 1$, one would get $2M + 1 > 2M$.

Now assume that the matrix entries of A and B are in $\{-M, \ldots, M\}$. We will work in the word-RAM model of computation with $O(\log n)$ bit words.

Define matrices A' and B' with entries

$$A'(i,j) = (n+1)^{M-A(i,j)},$$
$$B'(i,j) = (n+1)^{M-B(i,j)}.$$

Computing the integer product of A' and B' we obtain C' with entries

$$C'(i,j) = \sum_k (n+1)^{2M-(A(i,k)+B(k,j))}.$$

Observe that $(n+1)^{2M-C(i,j)} \leq C'(i,j)$ because $(n+1)^{2M-C(i,j)}$ is a summand in $C'(i,j)$. At the same time, $C'(i,j) \leq (n+1)^{2M-C(i,j)} \cdot n$ because $(n+1)^{2M-C(i,j)}$ is the largest summand in $C'(i,j)$ and $C'(i,j)$ has only n summands. Therefore, we can set $C(i,j)$ to be the unique integer L such that $(n+1)^{2M-L} \leq C'(i,j) \leq n \cdot (n+1)^{2M-L}$.

Note that we are dealing with integers having $O(M \log n)$ bits in C', for which arithmetic operations take $\widetilde{O}(M)$ time (both additions and multiplications). Bearing this in mind, it is straightforward to see that the above algorithm computes $A \star B$ in $\widetilde{O}(Mn^\omega)$ time. $\square$

To handle the short paths, we define the weighted adjacency matrix A of the graph as follows

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

We want to compute A^k where the powering is under $(\min, +)$-product. Assume k is a power of 2, we can successively square the matrix A to get $A^{2^i} = A^{2^{i-1}} \star A^{2^{i-1}}$. The running time depends on how large the entries of $A^{2^{i-1}}$ could be. Since $A^{2^{i-1}}$ represents the distance matrix for paths up to length 2^{i-1}, the absolute values of the entries are bounded by $2^{i-1}M$. Thus, by Theorem 1.1, the running time is

$$\sum_{i=1}^{\log k} 2^{i-1} M n^\omega = O(2^{\log k} M n^\omega) = O(kMn^\omega).$$

1.3 Combining the Long/Short Path Algorithms

The overall running time is $\tilde{O}(Mn^\omega + \frac{n^3}{k} + Mkn^\omega)$. The Mn^ω term is dominated by the Mkn^ω term, so we can ignore it. To balance the remaining two terms, we set $k = \frac{n^{(3-\omega)/2}}{\sqrt{M}}$, which gives an $\tilde{O}(\sqrt{M} n^{(3+\omega)/2})$ time algorithm.

Note that if $\omega = 2$, the above algorithm runs in time $\tilde{O}(\sqrt{M} n^{2.5})$. When $M = O(n^{1-\epsilon})$ for some constant $\epsilon > 0$, the running time is $\tilde{O}(n^{3-\epsilon/2})$. It is an open problem whether we can achieve a truly sub-cubic time ($O(n^{3-\delta})$ for positive δ) algorithm for directed APSP when $M = \Theta(n)$.

2 Zwick's Algorithm

In this section, we describe Zwick's Algorithm.

Theorem 2.1. *All-Pairs Shortest Paths (APSP) on directed graphs, where edge weights are integers in* $\{-M, \ldots, M\}$ *can be solved in* $\widetilde{O}(M^{1/(4-\omega)} n^{2+1/(4-\omega)})$ *time.*

Similar to the previous algorithm, Zwick's Algorithm handles paths that use at least k nodes, and paths that use fewer than k nodes separately. For long paths, the running time is the same as the previous algorithm, which is $\tilde{O}(Mn^\omega + \frac{n^3}{k})$ time. Zwick's Algorithm improves on the short paths.

In order to handle shortest paths of length less than k, we combine fast computations of $(\min, +)$ products with the idea of a hitting set argument.

Proposition 1. *Let G be a directed graph, where edge weights are integers in $\{-M, \ldots, M\}$, and k be a fixed parameter. We can compute $d(u, v)$ for every pair (u, v) where $\ell(u, v) \leq k$ in time*

$$\widetilde{O}\left(k^{3-\omega} M n^{\omega}\right).$$

Proof. We will have $\lceil \log_{3/2} P \rceil$ stages. Let V_j be the set of pairs of vertices (u, v) such that $\ell(u, v) \in ((3/2)^{j-1}, (3/2)^j]$, and let $V_{\leq j}$ denote $\cup_{i=1}^{j} V_i$. In stage j, we will compute $d(u, v)$ for every $(u, v) \in V_{\leq j}$. More specifically, we will compute a matrix D_j such that with high probability,

$$D_j(x, y) \begin{cases} = d(x, y) & \text{if } (x, y) \in V_{\leq j} \\ \geq d(x, y) & \text{otherwise.} \end{cases}$$

Note that D_1 can easily be obtained from the edge weights of G.

One could easily obtain a valid D_j from D_{j-1} by simply computing $D_{j-1} \star D_{j-1}$. However, it won't give the running time we desire. Instead, we will take advantage of the hitting set lemma.

For every $(u, v) \in V_j$, consider a shortest path $P_{u,v}$ from u to v. The *middle third* of $P_{u,v}$ is a set of $\lfloor (3/2)^{j-1} \rfloor$ nodes appearing consecutively in $P_{u,v}$ such that at most $(3/2)^{j-1}$ nodes precede them, and at most $(3/2)^{j-1}$ nodes follow them.

At stage j, we take a random $S_j \subseteq V$ with $|S_j| = \Theta(\frac{n}{(3/2)^{j-1}} \log n)$ so that with high probability, V hits a node $s_{u,v}$ in the middle third of $P_{u,v}$ for every $(u, v) \in V_j$. Observe that because $s_{u,v}$ is in the middle third of $P_{u,v}$, we get that $(u, s_{u,v}), (s_{u,v}, v) \in D_{\leq j-1}$.

It follows that with high probability, for all $(u, v) \in V_j$,

$$d(u, v) = \min_{s \in S_j} \{D_{j-1}(u, s) + D_{j-1}(s, v)\}.$$

Thus we can compute $D_j(u, v)$ via

$$D_j(u, v) = \min \left\{ D_{j-1}(u, v), \min_{s \in S_j} \{D_{j-1}(u, s) + D_{j-1}(s, v)\} \right\}.$$

This is easy to do in $O(n^2)$ time once we have already computed $\min_{s \in S} \{D_{j-1}(u, s) + D_{j-1}(s, v)\}$ for every (u, v). It can be obtained by computing the product $X \star Y$ where X contains the columns in D_{j-1} corresponding to the elements of S_j, and Y contains the rows in D_{j-1} corresponding to the elements of S_j. In other words, by selecting a hitting set S_j, we are able to use the $(\min, +)$-product of matrices much smaller than D_{j-1} in order to compute D_j.

Breaking X and Y into square blocks of side-length approximately $n/(3/2)^j$, so that there are approximately $(3/2)^j$ blocks in X and Y. We can use the $(\min, +)$-products of all $(3/2)^{2j}$ pairs of blocks to easily recover $X \star Y$. By theorem 1.1, since D_j has entries in $\{-(3/2)^j M, \ldots, (3/2)^j M\} \cup \{\infty\}$, this takes time

$$\widetilde{\mathrm{O}}\left((3/2)^{2j} \cdot (3/2)^j \cdot M \cdot \left(\frac{n}{(3/2)^j} \right)^{\omega} \right) = \widetilde{\mathrm{O}}\left(((3/2)^{3-\omega})^j M n^{\omega} \right).$$

Summing over the $\lceil \log_{3/2} k \rceil$ stages, we get a running time of

$$\widetilde{\mathrm{O}}\left(n^{\omega} M \sum_{j:(3/2)^j < k} ((3/2)^j)^{3-\omega} \right) = \widetilde{\mathrm{O}}\left(k^{3-\omega} M n^{\omega}\right).$$

□

We are now in a position to complete the proof of Zwick's Theorem. Indeed, combining the long distance algorithm and Proposition 1 and optimizing for k at $k = \frac{n^{(3-\omega)/(4-\omega)}}{M^{1/(4-\omega)}}$, we get a total running time of $\widetilde{\mathrm{O}}(M^{1/(4-\omega)} n^{2+1/(4-\omega)})$. Observe that both the algorithm for long paths and for short paths compute either the correct distances or overestimate for distances between pairs of nodes; thus minimizing the outputted distances of the two, one can obtain the exact $d(u, v)$ for all $u, v \in V$.