The goal of this lecture is to show algorithms for APSP in directed graphs, where the edge weights are integers in \{-M, \ldots, M\}. We assume that the graph does not have negative cycles. In the first part, we show an algorithm that runs in \(\tilde{O}(\sqrt{M}n^{(3+\omega)/2})\). This algorithm uses the idea of the Hitting Set Algorithm last lecture. For the second part, we will show Zwick’s algorithm that improves the previous algorithm. The running time of Zwick’s algorithm is \(\tilde{O}(M^{\frac{3}{2}}n^{\frac{3+\omega}{2}})\), 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 \(O(M^{0.752}n^{2.5286})\), using rectangular matrix multiplication, but we won’t cover the rectangular matrix multiplication part.

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

Recall from last lecture, we showed a Hitting Set Algorithm that solves APSP in unweighted directed graphs in \(O(n^{(3+\omega)/2})\) time. In a high level, it considers long paths and short paths separately. For short paths, it uses matrix multiplication; for long paths, it uses sampling and single source shortest paths (SSSP). The \(\tilde{O}(\sqrt{M}n^{(3+\omega)/2})\) algorithm in this lecture has a similar structure.

Let \(k\) be a parameter that serves as a threshold of long and short paths. We call paths that use at least \(k\) nodes long paths, and paths that use fewer than \(k\) nodes short paths. 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 the shortest path from \(i\) to \(j\). If there are multiple shortest paths, let \(\ell(i, j)\) be the number of vertices of any of these paths.

1.1 Handle Long Paths

For every pair of vertices \(i, j\), if \(\ell(i, j) \geq k\), then we pick one shortest path \(P_{i,j}\) from \(i\) to \(j\) that has length at least \(k\).

Using the hitting set lemma, we know that if \(T \subseteq V\) is a uniformly random subset of \(\Theta(n^2 \log n)\) nodes, then \(T \cap P_{i,j} \neq \emptyset\) for every pair of \(i, j\), 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 \(i, j\) 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 the negative edges in the graph, we can use Johnson’s trick.

**Algorithm 1:** Johnson’s trick, \(G = (V, E), \) with edge weights \(w : E \rightarrow \{-M, \ldots, M\}\)

1. Add a new node \(q\).
2. Add an edge with weight 0 from \(q\) to every vertex \(v \in V\).
3. Compute SSSP from \(q\) (Look for a \(\tilde{O}(Mn^2)\) time algorithm in the next lecture).
4. foreach \((i, j) \in E\) do
5. \(\quad w'(i, j) = w(i, j) + d(q, i) - d(q, j)\)

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

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

Let \(d'(i, j)\) be the distance from \(i\) to \(j\) using weights \(w'\).
Claim 2. For every \(i, j, d'(i, j) = d(i, j) + d(q, i) - d(q, j)\).

Proof. For any path \(v_1 \rightarrow \cdots \rightarrow v_t\), we have

\[
w'(P_{v,v}) = \sum_{k=1}^{t-1} w'(v_k, v_{k+1}) = \sum_{k=1}^{t-1} w(v_k, v_{k+1}) + d(q, v_k) - d(q, v_{k+1}) = w(P_{i,j}) + d(q, v_1) - d(q, v_t).
\]

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 \(j\) preserves. 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 \(\tilde{O}(Mn^2)\) time to perform Johnson’s trick, and \(\tilde{O}(n^2|T|)\) time to run Dijkstra’s algorithm to and from every vertex in \(T\).

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

Note that Johnson’s trick does not help with short paths, since it blows up the weights: \(d'(i, j)\) can be as large as \(Mn\) in magnitude.

1.2 Handle Short Paths

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

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

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

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 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 \(\tilde{O}(Mn^2)\) time.

Proof. First note that we can assume that there are no infinite entries - replace each \(\infty\) with \(3M + 1\). These entries can never be used in a \((\min, +)-product\) entry (unless that entry is \(\infty\) 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 \(\{-M, \ldots, M\}\). 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 $\tilde{O}(M)$ time (both additions and multiplications). Bearing this to mind, it is straightforward to see that the above algorithm computes $A \ast B$ in $O(Mn^\omega)$ time. \hfill $\square$

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 $(\min,+)\text{-product}$. Assume $k$ is a power of 2, we can successively square the matrix $A$ to get $A^2 = A^{2-1} \ast A^{2-1}$. The running time depends on how large the entries of $A^{2^{-1}}$ could be. Since $A^{2^{-1}}$ 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

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

1.3 Combining the Long/Short Path Algorithms

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

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 $\tilde{O}(M^{1/(4-\omega)}n^{2+1/(4-\omega)})$ 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 $(\min,+)\text{-products}$ 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

$$\tilde{O}(k^{3-\omega}Mn^\omega).$$

**Proof.** We will have $[\log_{3/2} P]$ 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 $(u,v) \in V_j$. 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 $D_{j-1} \ast D_{j-1}$. 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 $|S_j| = \Theta(\frac{n}{(3/2)^j \log n})$ 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 $(u, s_{u,v}), (s_{u,v}, v) \in D_{j-1}$.

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 $\min_{s \in S_j} \{D_{j-1}(u, s) + D_{j-1}(s, v)\}$ for every $(u, v)$. It can be obtained by computing the product $X \ast Y$ 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 $X \ast Y$. By theorem 1.1, since $D_j$ has entries in $\{-3(3/2)^j M, \ldots, (3/2)^j M\} \cup \{\infty\}$, this takes time

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

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

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

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 $\widetilde{O}(M^{1/(4-\omega)} n^{2+1/(4-\omega)})$. 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 $u, v \in V$. 

\[\square\]