W2U3:
All Pairs Shortest Path -1
Part 3

C Pandu Rangan

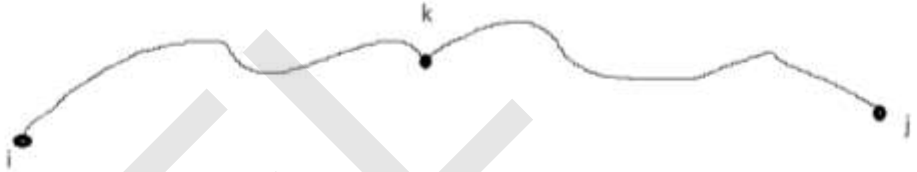
Recap

- Algorithm 1 and 2

Outcome

- Algorithm 3
- k paths
- Algorithm 4 (Floyd-Warshall Algorithm)
- Johnson's Algorithm

Generalised BE 1



- The algorithm described earlier are based on the "last edge of the shortest path" or based on the vertex just before the destination.
- We may generalize this to an arbitrary intermediate vertex of a shortest path.
- It is easy to prove that,
- If k is an intermediate vertex in a shortest path P from i to j, the position of the P from i to k as well as the portion of the path P from k to j are shortest paths (from i to k and k to j respectively).

Generalised BE 2

Here is a kind of converse to the statement given above and this is also easy to prove.

Let Q_{ik} be a shortest path from i to k with at most l edges and R_{kj} be a shortest path from k to j with at most l edges.

Let $W(i,j,k)$ be the walk obtained by concatenating Q_{ik} and R_{kj}.

Let $P(i,j)$ be a shortest walk among $\{W(i,j,k), k = 1,2,\ldots,n\}$.

Then,

1) $P(i,j)$ is a path.
2) $P(i,j)$ is a shortest path from i to j.
3) $P(i,j)$ is a shortest path with $\leq 2l$ edges.

Proof of Generalised BE 1 & BE 2

- $P = Q + R$
- Q and R are concatenated at k.

Proof of Generalised BE 1 & BE 2 (contd)

Proof of Generalised BE 1 & BE 2 (contd)

Squaring for Extended BE

$$D_{ij}^{(2l)} = Min\left\{D_{ik}^{(l)} + D_{kj}^{(l)}\right\}$$

In Matrix Multiplication Notation

$$D^{2l} = D^{l} \cdot D^{l} = \left(D^{(l)}\right)^{2}$$

$$D_{ij}^{(1)} = w_{ij} \text{ or } D^{(1)} = w$$

Thus, by a series of squaring operations we obtain a series of Matrices

$$D^{(1)} \rightarrow D^{(2)} \rightarrow D^{(4)} \rightarrow D^{(8)} \rightarrow \dots D^{(2^{i})} \rightarrow \dots$$

Algorithm 3

We are interested in $D^{(n-1)}$

But $D^{(n-1)} = D^{(l)}$ for all $l \geq (n-1)$,

Since $2^{\log n} \geq n > n-1$,

We conclude that $D^{(n-1)} = D^{\left(2^{\lceil \log n \rceil}\right)}$

Thus, we perform $\lceil \log n \rceil$ squaring operations and output the resulting Matrix.

$$D = W$$

(where W is the extended weight matrix)

For $i = l$ to $\lceil \log n \rceil$

$$D = D \cdot D$$

Return D

$$D = D^{\left(2^{\lceil \log n \rceil}\right)} = D^{(n-1)}$$

The complexity is $O(n^3 logn)$

The Algorithm is due to M. Fisher and A. Meyer).

Thank You

k - Paths

- We will now discuss on $O(n^3)$ algorithm based on a different formulation involving intermediated nodes. (In fact, several researchers have worked with same idea around the same time).
- Call a path from i to j a k-path from i to j if all intermediate nodes are $\leq k$. That is, the path from i to j pass through the set of vertices in $\{1,2,3, \cdots, k\}$.
- k-path is automatically an l-path for all $l > k$. 0-path from i to j is just the edge (i, j), if it exists.
- Note that k is independent of i and j. The nodes i and j are source and destination vertices of the path and upper bound k is applicable only for the intermediate nodes.

k – Paths (contd)

Let $\delta_k(i,j)$ be the weight of shortest k-path from i to j. Since n is the largest vertex label,

$$\delta_k(i,j) = \delta(i,j) \quad \forall\, i,j \in V$$

Note that

$$\delta_o(i,j) = w(i,j) \quad \forall\, i,j \in V$$

Define $A^{(k)} = \left[a_{ij}^{(k)}\right]_{n\times n}$ by $a_{ij}^{(k)} = \delta_k(i,j)$.

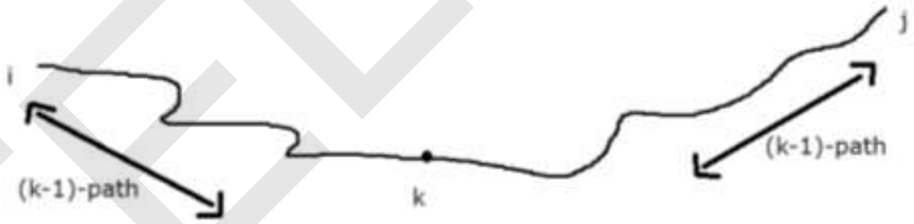
The following observation allows us to write $A^{(k)}$ elements in terms of the elements in $A^{(k-1)}$.

A k-path without k

A k-path from i to j may contain k or may not contain k. If it does not contain k, all its intermediate vertices are $\leq (k-1)$ and hence it is in fact a $(k-1)$ path from i to j.
This is a $(k-1)$ path. In this case
$\delta_k(i,j) = \delta_{k-1}(i,j)$ ---(5)
If the k-path from i to j contain k, then the part of the path from i to k and the part of the path from k to j are both $(k-1)$-path, because k can not occur more than once in any path and rest of the internal nodes are all $\leq (k-1)$.

k-path from i to j of weight $\delta_k(i, j)$

The $(k-1)$-path from i to k and the $(k-1)$-path from k to j are shortest paths (by theorm…)



Hence

$\delta_k(i,j) = \delta_{k-1}(i,j) + \delta_{k-1}(k,j)$ ---(6)

In this case,

From equation (5) and (6)

We conclude that

$\delta_k(i,j) =$ Min $\{\delta_{k-1}(i,j), \delta_{k-1}(i,k) + \delta_{k-1}(k,j)\}$ ---(7)

Complexity for Extension

That is, the computation of $\delta_k(i,j)$ involves referring three elements $\delta_{k-1}(i,j), \delta_{k-1}(i,k)\ and\ \delta_{k-1}(k,j)$ and performing one addition and one comparison and this $O(1)$ computation.

Thus, $A^{(k)}$ matrix values can be determined in $O(n^2)$ time,

if $A^{(k-1)}$ values are available.

Hence, starting from $A^{(0)}$ and computing the sequence of matrices

$A^{(0)} \to A^{(1)} \to A^{(2)} \to \cdots \to A^{(n-1)} \to A^{(n)}$

takes $o(n^3)$ time

Algorithm 4 - Floyd-Warshall Algorithm

Floyd-Warshall (G, W).

$A^{(0)} = W$ for $k = 1$ to n

\\Compute $A^{(k)}$ using $A^{(k-1)}$

For $i = 1$ to n

For $j = 1$ to n

$$a_{ij}^{(k)} = Min\left\{a_{ij}^{(k-1)}, a_{ik}^{(k-1)} + a_{kj}^{(k-1)}\right\}$$

Return $A^{(n)}$ \\ $a_{ij}^{(n)} = \delta(i, j)$

Johnson's Algorithm

- We will now look at yet another algorithm that is faster for sparse graph
- Floyd-Warshall's algorithm is $O(n^3)$ and the complexity is independent of the number of edges of the graph.
- The algorithm by Johnson runs in $O(n^2 \log n + nm)$ time and when the graph is sparse, this is asymptotically better than $O(n^3)$ algorithm.
- For dense graph with $m = O(n^2)$, the complexity is $O(n^3)$, which is same as Warshalls Algorithm. This algorithm uses a clever transformation technique to achieve improvements.

Johnson's Algorithm (contd)

If all weights are positive, we may apply Dijkstra's algorithm n times (once from each vertex as the source) and the complexity for this algorithm would be in

$$O(n[\log n + m]) = O(n^2 \log n + nm)$$

However, this approach is not applicable if G has some negative edges.

If G has negative edges but no negative cycles, we may apply n times the Bellman-Ford algorithm and the complexity would be $O(n.n.m) = O(n^2 m)$. For Dense graph this may go as high as $O(n^4)$.

Johnson's algorithm deploys a transformation of weights that allowed him to use both Dijkstra's and Bellman-Ford algorithms to exploit the best in both methods.

Weight Transformation

Let $G = (V, E)$ be a directed graph and w be the weight function from edge set to integers.

Let $V = \{1,2,\cdots,n\}$ and h be any function from V to integers.

Define a new weight function w' by

$w'(u,v) = w(u,v) + h(u) - h(v)$

1. P is a shortest path from i to j under w if it is a shortest path under w'
2. For any cycle c in $G, w(c) = w'(c)$
3. For any path P from i to j

$w(P) = w'(P) + h(j) - h(i)$

Basic Idea

- Thus, instead of working on G with weight function w, we may work on G with weight function w'.
- If $w(e) > 0 \;\; \forall\, e \in E$, we need not transform the weights. We apply Dijkstra's algorithm n times and obtain as algorithm for APSP with complexity $o(n^2 \log n + nm)$.
- If $w(e)$ is negative for some edges, using Bellman and Ford n times leads to a very inefficient $o(n^2 m)$ algorithm. This is the case that requires a transformation of weights.

Basic Idea (contd)

- The trick is,
  Use Bellman-Ford algorithm ONCE and find a
  $h: V \rightarrow I$ such that $w'(e) > 0 \; \forall \; e \; \in E$.
- Now, Dijkstra's algorithm can be applied n times om G with weight function w' and solve APSP with respect to w'. The same physical paths determined by w' can be used for w, by (1).
- Since $\delta(i,j) = \delta'(i,j) + h(j) - h(i)$ by (3), the APSP weight matrix under w can be constructed from APSP weight matrix under w' in $O(n^2)$ time.

Johnson's Algorithm

The Algorithm at a high level is as follows:

1. Use Bellman-Ford algorithm to determine a $h: V \rightarrow I$ such that $w'(e) > 0\ \forall\ e\ \in E$ where
$w'(i,j) = w(i,j) + h(i) - h(j)$
2. For $i,j\ \ \forall V,\ \ i \neq j.$
$w'(i,i) = w(i,i) = 0 \forall i \in V$
3. Solve APSP problem by using Dijkstra's Algorithm n times on G with weight function w'. Let D' be the shortest path weight Matrix obtained.

Johnson's Algorithm (contd)

4. Construct the shortest path weight Matrix D by using the formula

$$\delta(i,j) = \delta'(i,j) + h(j) - h(i)$$

5. Return D.

The total complexity is

$O(mn) + O(n^2 \log n + nm) + n^2$

which is $O(n^2 \log n + nm)$

Thus, Johnson's Algorithm solves APSP problem for a G with no negative cycles in $O(n^2 \log n + nm)$ time.

Construction of h

We now focus on the construction of h and prove that

$w'(i,j) = w(i,j) + h(i) - h(j) \geq 0$

Let $s \notin V$ and construct G' by adding s to V and adding directed edges

$(s,i)\ \forall\ i \in V$ with $w(s,i) = 0$.

That is $G' = (v', E')$ where $v' = V \cup \{s\}$

$E' = E \cup \{(s,i) | i \epsilon V\}$

Solve SSSP problem on G' with s as a source and define

$h(i) = \delta(s,i)$ in E, (which is also in E').

Weight Transformation

We know that

$$\delta(j) \leq \delta(i) + w(i,j)$$

Hence,

$$w(i,j) + \delta(i) - \delta(j) \geq 0$$

Implying

$$w(i,j) + h(i) - h(j) \geq 0$$

Thus

$w'(i,j) = w(i,j) + h(i) - h(j) \geq 0$ for all $(i,j) \in E$

This completes our discussions on Johnson's Algorithm

Remark

- If G has a cycle with negative weight then G' also will have the same cycle as a negative weight cycle. Thus, if Bellman-Ford algorithm working on G' reports a negative cycle in G', we report G has a negative cycle and simply terminate the algorithm at this point. We will proceed with further steps only when we know that G has no negative cycles.

W3U1

- Minimum Spanning Trees

Thank You