

APSP

All pairs shortest paths

Goal: Find $P[u, v] \forall (u, v) \in E$, where

$$\omega(P) = \delta(u, v)$$

i.e., P is the shortest path from u to v in the input graph $G = (V, E)$

There are 4 algorithms to solve this problem

① Run Bellman-Ford algorithm n times
($|V|=n$) : $O(n^4)$

② Squaring of the distance matrix:

$$D^0, D^1, D^2, \dots, D^K$$

$$K \geq n-1$$

$$D^0 = \begin{cases} 0 & i=j \\ \infty & i \neq j \end{cases}$$

$$d_{ij}^k = \min_k \left\{ d_{ik}^{k-1} + w_{kj} \right\}$$

③ Relation to matrix multiplication

$$D_{n-1} = D_0 W^{n-1}$$

$$T(n) = O(n^3 \log n)$$

④ Floyd-Warshall's Algorithm

$$T(n) = O(n^3)$$

⑤ Johnson's Algorithm

$$T(n) = \begin{cases} O(n^2 \log n + nm) \\ O(n^3) ; \text{ for } |E|=0 \\ (n^2) \end{cases}$$

Bellman Equations:

Notation: $P[l: i, j]$: Set of all paths from node i to node j with atmost l edges ($l \geq 0$)

$$\omega(P[l: i, j])$$

$$= \delta_{\leq l}(i, j) = [d_{ij}^l]$$

* let $D^l = [d_{ij}^l]$ be a matrix

of order $n \times n$ representing
shortest distance values b/w
every pair of vertices.

∴ Goal : Compute $D^{(n-1)}$

$$\text{i.e., } P[n-1: u, v]$$

$$\text{So, } d_{ij}^l \geq d_{ik}^{l-1} + \omega_{kj} \quad \forall k$$

$$\Rightarrow d_{ij}^l = \min_k \{ d_{ik}^{l-1} + \omega_{kj} \}$$

Operator Overriding : \triangleleft (Semi-Ring matrix multiplication)

* both min and + are associative and commutative

$$\boxed{+ (a, b) = + (b, a)}$$

$$\min (a, b) = \min (b, a)$$

$$+ (+ (a, b), c) = + (a, + (b, c))$$

$$\min (\min (a, b), c) = \min (a, \min (b, c))$$

Overriding $\overset{'+'}{\sim} \triangleleft$
we use $\overset{'+'}{\sim}$ for ' \min '

Overriding $\overset{'\cdot'}{\sim} \triangleleft$
we use $\overset{'\cdot'}{\sim}$ for ' $+$ '
 ↴
 multiplication

$$\text{So, } \min_k \left\{ d_{ik}^{l-1} + w_{kj} \right\}$$

$$= + \left\{ d_{ik}^{l-1} \cdot w_{kj} \right\}$$

$$\Rightarrow D^\ell = D^{\ell-1} \omega$$

$$D^{(1)} = D^{(0)} \omega$$

$$D^{(2)} = D^{(0)} \omega^2$$

$$\vdots$$

$$\Rightarrow D^{(n-1)} = D^{(0)} \omega^{(n-1)}$$

$O(n^4)$
Complexity

* We know that $\omega(P[L:i,j])$
 $= \omega(P[n-1:i,j])$
 $\forall L \geq n-1$

\Rightarrow we can square $\omega^0, \omega^2, \omega^4, \dots, \omega^L$

Such that $L \geq n-1$ i.e., we stop squaring when
 L just becomes equal to or greater than
 $n-1$.

$$\therefore T(n) = O(n^3 \log n)$$

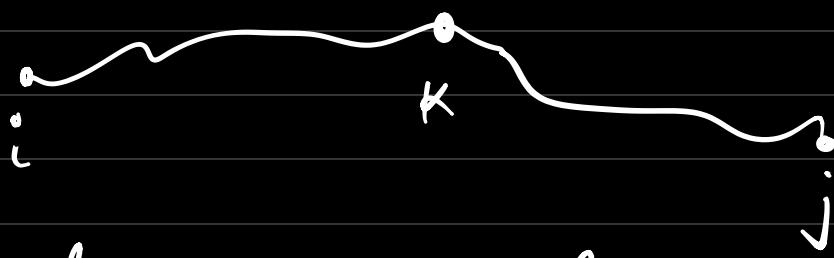
for $\ell=1$ to $\lceil \log n \rceil$
 $w = w^2$

$$D = D^{(0)} W$$

Algorithm

Return D

Generalized Bellman Edge :



$$d_{ij}^l = \min_k \{ d_{ik}^{l-1} + w_{kj} \}$$

is based on the last edge in the path.

we now generalize the same on an arbitrary vertex in the path (k is the arbitrary vertex in the path from i to j)

Q_{ik} : shortest path from i to k with almost l edges

R_{kj} : shortest path from k to j with almost l edges

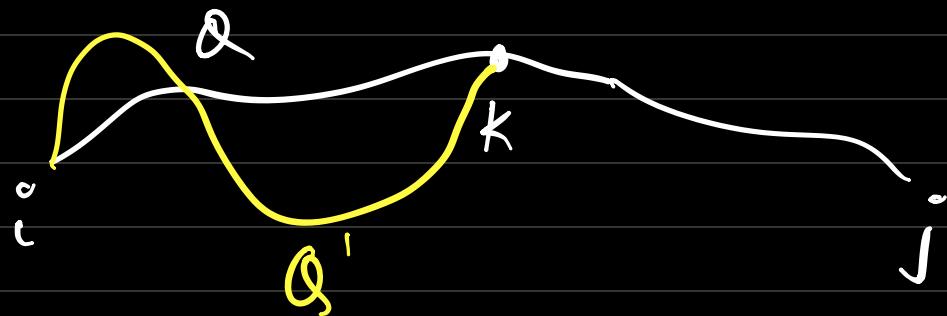
P_{ij} : shortest walk among $K = \{1, 2, \dots, n\}$ $W(i, j, K)$

$$W(i, j, K) = Q_{ik} + R_{kj}$$

walk obtained from concatenation of two paths

$\therefore G$ has no -ve or o cycle, the
Shortest path = Shortest walk.

$P(i, j)$ is a shortest path with atmost
 $2l$ edges.



$$P = Q + R$$

Q and R are concatenated at k

Let Q' be the shortest path from i to k

$$\omega(Q') < \omega(Q)$$

$Q' + R$ may be a walk

and has a path P' such that

$$\omega(P') \leq \omega(Q' + R)$$

$$\leq \omega(Q') + \omega(R)$$

$$\leq \omega(Q) + \omega(R)$$

$$\leq \omega(P)$$

This contradicts the minimality of P .

\therefore Such a Q' cannot exist.

Similarly, R is a shortest path.

\therefore A shortest path from i to j is a combination of shortest path from i to k and shortest path from k to j .

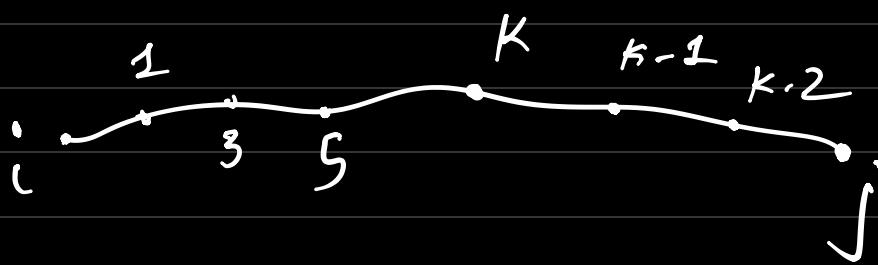
We don't know which k is that. So, we try out every possible k and take min. of the weights of the paths via k .
↓
node in the graph.

Floyd

Warshall

APSP Algorithm

K - paths : <



we call a path from i to j a K -path
iff all the nodes in the path are $\leq K$
(labeling/indexing)

- * If $l \geq K$, then automatically every K -path is an l -path.
- * 0-path is just the edge (i, j) , if it exists.
- * The upper bound K is applicable only to the intermediate vertices in the path from i to j .

i : Source vertex ; j : destination vertex

in $P[i, j]$: path from i to j . Rest are
are intermediate vertices that belong to
the path $P[i, j]$

$\delta_K(i, j)$: weight of the shortest K -path
from i to j

$\therefore n$ is the largest label a vertex can be labeled with,

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

NOTE:

$$\delta_0(i, j) = w[i, j] \quad \forall i, j \in V$$

Define $A^{(K)} = [a_{ij}^{(K)}]$ not power!

$$a_{ij}^{(K)} = \delta_K(i, j)$$

Observation:

we can write $A^{(K)}$ in terms of the elements of $A^{(k-1)}$

* A K -path may or may not contain the vertex labeled as K .

i.e,

Cases / Scenarios of
a K -path

Contains K

Does not
contain K

This is consistent with
our definition of a k -path.

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

$$\delta_K(i, j) = \delta_{k-1}(i, j)$$

we'll take the min. of both cases and hence

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

" " we defined the matrix $A^{(k)}$ as

$$A^{(k)} = [a_{i,j}^{(k)}] = [\delta_k(i, j)]$$

and also bringing in the operator overriding of '+' and '...' operators,

we have

$$\delta_k(i, j) = A^{(k-1)}[i][k] \cdot A^{(k-1)}[k][j] + A^{(k-2)}[i][j]$$

$$A^{(k)}[i][j] = A^{(k-1)}[i][k] \cdot A^{(k-1)}[k][j] + A^{(k-1)}[i][j]$$

" " Compute $A^{(0)} \rightarrow A^{(1)} \rightarrow \dots \rightarrow A^{(n-1)} \rightarrow A^{(n)}$
return $A^{(n)}$ //

This is Floyd-Warshall's Algorithm

Time Complexity: $O(n^3)$ and the complexity is independent of the no. of edges in the graph

Johnson's
Algorithm

FW has a drawback of Computation where the Computation has to be carried out regardless of the no. of edges in the input graph.

∴ For a Sparse graph, there will be unnecessary Computations.

Johnson's Algorithm alleviates this problem by a clever transformation technique to achieve improvements.

NOTE :

① The Scenario when there are negative edges :

for APSP. Simply applying Dijkstra's algo. n times won't work.

BF algo. will do.



Apply BF n times. $\rightarrow O(n^2m) = O(n^4)$

$|V|$ $|E|$

Johnson's algo. will use a mix of FW and BF to exploit the best of both the methods.

Weight Transformation

$$h: V \rightarrow \mathbb{R}^+$$

h is a mapping from the vertices to
tve real numbers

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

↓ is the transformation technique used.

weight transformation

Trick: Use Bellman-Ford algo. ONCE and find a $h: V \rightarrow I$ such that

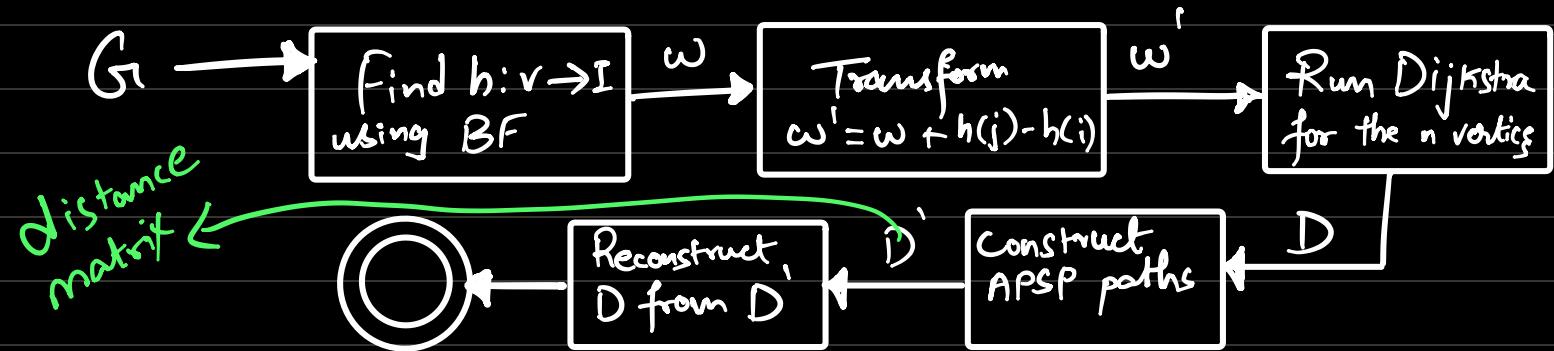
$$w'(e) > 0 \quad \forall e \in E$$

Run Dijkstra's SSSP n times on G with weight function w' and solve APSP w.r.t w' .

The same physical paths can be used for w

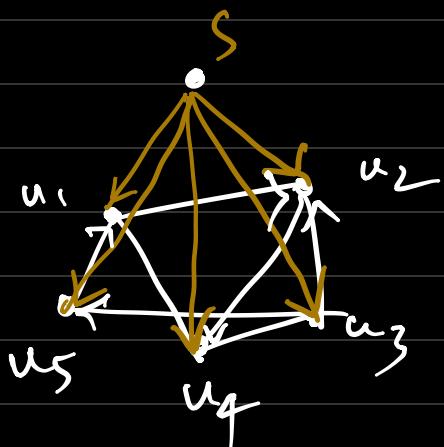
$$w(i, j) = w'(i, j) + h(i) - h(j)$$

↳ $O(n^2)$ time



$h(u)$ if $u \in V$ can be found by introducing a new vertex S which is connected to all other vertices with $w(S, u) = 0 \forall u \in V$

Run BF on the new graph G'



$h(u) = \delta(S, u)$
will be the function h for all the vertices.

Algorithm:

- Using Bellman - Ford, determine a mapping $h: V \rightarrow I$ such that \downarrow
 $w'(e) > 0 \forall e \in E$
integers

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

- For $i, j \in V, j \neq i$

$$w'(i, i) = w(i, i) = 0 \quad \forall i \in V$$

- Solve APSP using Dijkstra's algorithm n times on G' with weight function w' .
Let D be the shortest path weight matrix obtained.

- Construct the shortest path weight matrix D by using the formula
 $\delta(i, j) = \delta'(i, j) + h(j) - h(i)$

⑤ Return D

Time Complexity : $T(n) = O(mn) +$

BF

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

$+ n^2 + n^2$

\downarrow

n times

Dijkstra

Transformation & reconstruction
of the in ④

Original weight function

Construction of h :

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

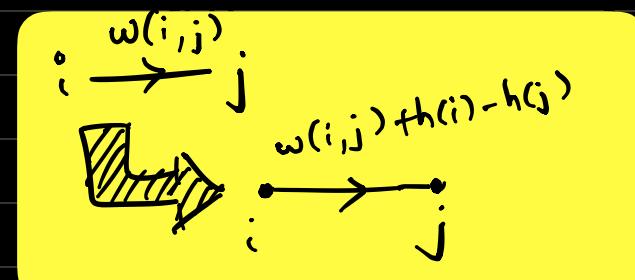
$$\geq 0$$

$$s \notin V \Rightarrow V' = V \cup \{s\}$$

$$h(u) = \delta(s, u)$$

$$E' = E \cup \{(s, u) \mid u \in V\}$$

$$\Rightarrow G' = (V', E', \omega')$$



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

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

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

$$\Rightarrow \omega'(i,j) \geq 0 \quad \forall (i,j) \in E$$

SSSP weight
from s to j

$$\delta(i) = \delta(s, i)$$

\downarrow
source

Summary

① Bellman - Ford's algo. can detect
-ve Cycle and Dijkstra's algo.
Cannot detect -ve cycle.

-ve cycle detection ?



Use Bellman-Ford algorithm (SSSP)

② So, we will use this as a preprocess
to assert the input graph does not
contain -ve cycle. ✓