APSP

All pairs shortest paths

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

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

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:

$$D^0, D^2, D^4, \ldots, D^K$$

$$K \geq n-1$$

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

$$d_{ij}^{\ell} = \min_K \{ d_{iK}^{\ell-1} + w_{Kj} \}$$

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

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

Bellman Equations:

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

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

* 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 $D^{(n-1)}$

i.e, $P[n-1 : u, v]$

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

$\Rightarrow d_{ij}^{l} = \min_{k} \{ d_{ik}^{l-1} + w_{kj} \}$

Operator Overriding :→ (Semi-Ring matrix multiplication)

* both min and + are associative and commutative

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

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

Overriding '+' :→

we use '+' for 'min'

Overriding '.' :→

we use '.' for '+'
↳ addition

↓ multiplication

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

$$= \underset{k}{+} \{ d_{ik}^{l-1} \cdot w_{kj} \}$$

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

$$D^{(1)} = D^{(0)} W$$

$$D^{(2)} = D^{(0)} W^2$$

$$\vdots$$

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

$\Big\}$ $O(n^4)$ Complexity

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

$\forall \; L \geq n-1$

$\Rightarrow$ we can square $\omega^0, \omega^2, \omega^4, \ldots, \omega^L$ such that $L \geq n-1$ i.e, we stop squaring when L just becomes equal to or greater than $n-1$.

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

for $\ell = 1$ to $\lceil \log n \rceil$

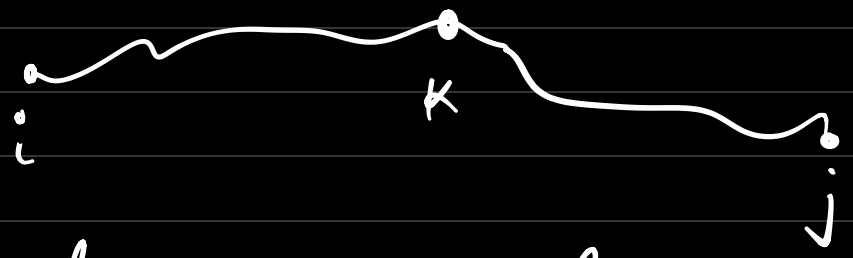
$W = W^2$

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

Return D

Algorithm

Generalized Bellman Edge :1



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

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 atmost ℓ edges

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

P_{ij} : Shortest walk among $W(i,j,k)$ $k=\{1,2,\ldots,n\}$

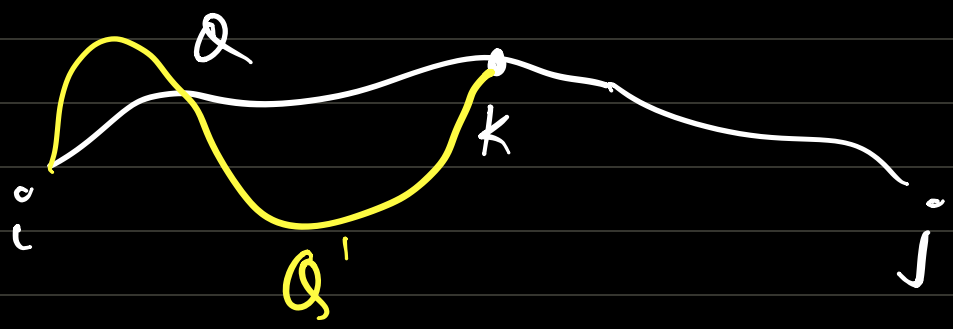
$$W(i,j,k) = Q_{ik} + R_{kj}$$

↓

walk obtained from concatenation of two paths

$\therefore$ G has no $-ve$ or 0 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

$$w(Q') < w(Q)$$

$Q' + R$ may be a walk

and has a path p' Such that

$$\begin{aligned} w(p') &\leq w(Q' + R) \\ &\leq w(Q') + w(R) \\ &\leq w(Q) + w(R) \\ &\leq w(P) \end{aligned}$$

This Contradicts the minimality of P.

$\therefore$ Such a Q' cannot exist.

Similary, R is a Shortest path.

∴ A Shortest path from i to j is a Combination of Shortest path from i to k and Shortest path from k to j.

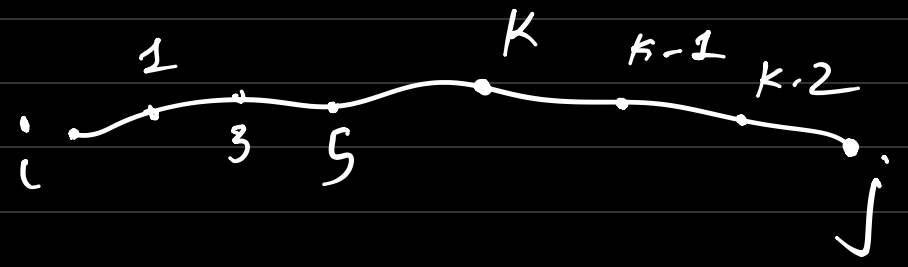
we don't know which k is that. So, we try out every possible k and take min. of the weights of the paths via k.

↓

node in the graph.

Floyd-Warshall's 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 $\ell \geq K$ 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

$\because$ 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 $A^{(k-1)}$

$\star$ 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

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

∵ we defined the matrix $A^{(k)}$ as

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

and also bringing in the operator overriding of '+' and '·' operators, we have

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

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

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

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 APSP 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. ⟶ $O(n^2 m) = O(n^4)$

($|V|$, $|E|$)

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

Weight Transformation

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

h is a mapping from the vertices to +ve 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 $h : V \longrightarrow I$ such that

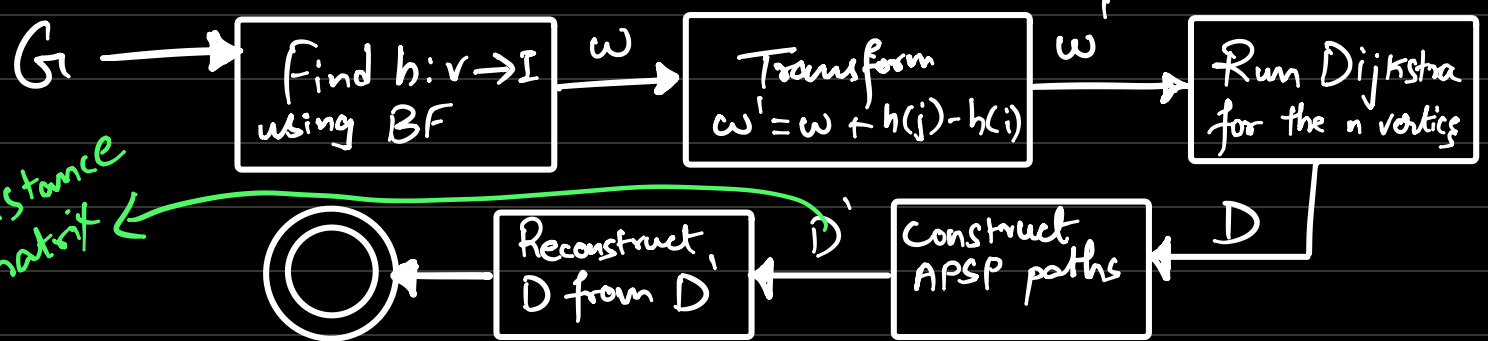
$$w'(e) > 0 \quad \forall e \in E$$

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

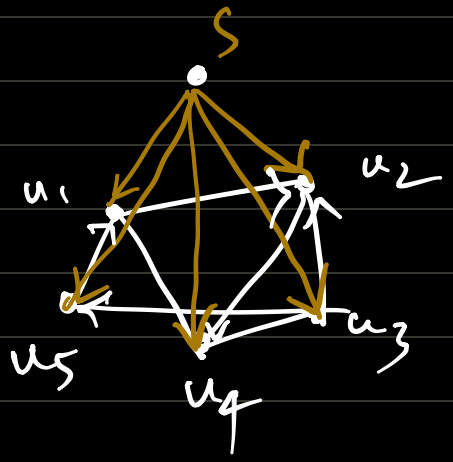
$$w(i,j) = w'(i,j) + h(i) - h(j)$$

↳ $O(n^2)$ time



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

Run BF on the new graph G_1'



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

Algorithm:

① Using Bellman-Ford, determine a mapping $h: V \longrightarrow I$ such that $w'(e) > 0 \quad \forall e \in E$

(I ↓ integers)

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

② For $i, j \in V$, $j \neq i$

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

③ Solve APSP using Dijkstra's algorithm n times on G_1 with weight function w'. Let D' be the shortest path weight matrix obtained.

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

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

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

$+ n^2 + n^2$

$+ n^2$: Transformation of the Original weight function

$+ n^2$: reconstruction in ④

Construction of h:

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

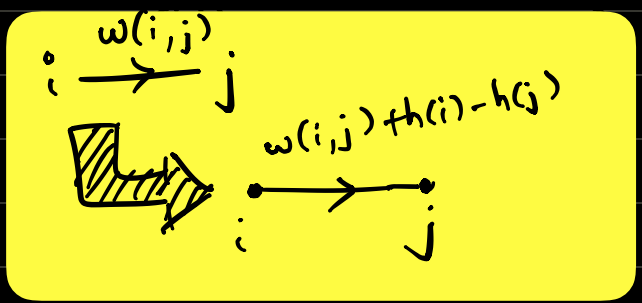
$$\geq 0$$

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

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

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

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



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

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

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

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

SSSP weight from s to j

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

s = 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.