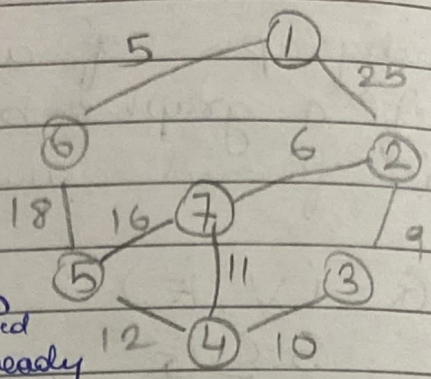
L-10) Prim's Algorithm for minimum cost spanning Tree

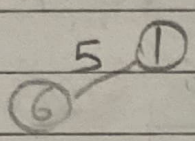
i) Select minimum cost edge from graph (1,6)

ii) Select next minimum cost edge and it should be connected to either of vertices that have been selected already

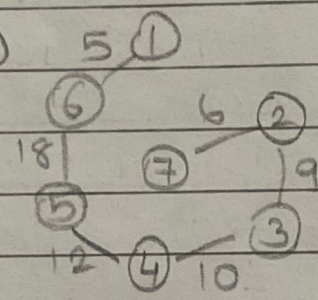
iii) Repeat



i)



ii)



5+18+12+10 +9+6 = 60

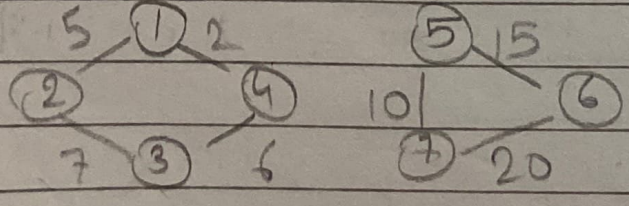
★ Prim's Algorithm tries to keep spanning tree a tree in it's intermediate stages

★ $(|V|-1)\,|E| \Rightarrow n\,e \Rightarrow n*n \Rightarrow O(n^2)$

If we use heap to find minimum cost edge

$(|V|-1)\log|E| \Rightarrow O(n \log n)$

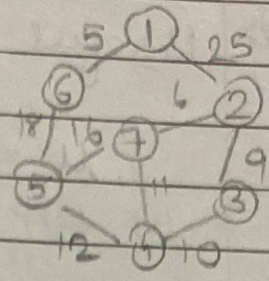
★ If you run prim's algorithm in non connected graph you might get minimum cost spanning tree for one of it's component.



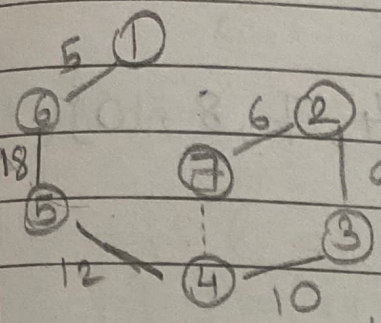
Algorithm will never reach second component

13) Kruskal's Method for finding minimum cost spanning tree.

i) Always select minimum cost edge.



i)



..... → Don't take as it forms cycle, continue

Analysis : $(|V|-1)|E|$ (No. of edges)

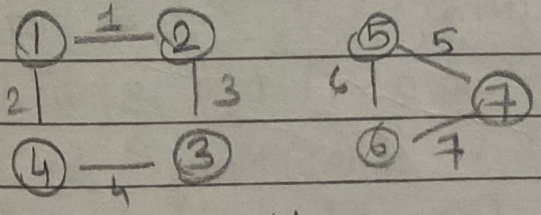
: $ne = n*n$

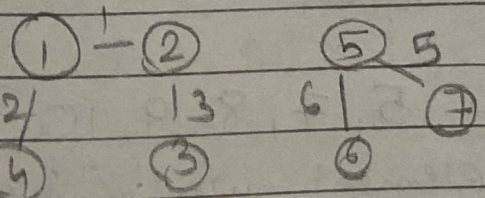
: $O(n^2)$

We can use min heap for finding minimum cost edge, in that case $O(n \log n)$

⇨ Focuses more on finding a minimum tree than finding a tree itself like in Prim's algorithm.

⇨ Useful for non-connected graphs, as it'll find minimum cost spanning tree for individual components but not for entire graph



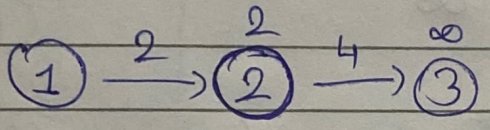


Dijkstra Algorithm

PAGE NO.:

- Single Source Shortest path algorithm
- Based on Greedy Approach
- Minimization problem comes under optimization
- Works on directed as well as non-directed graph

* Working (To understand ⇒ small example)



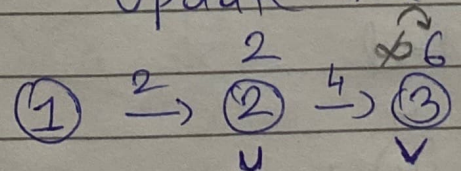
(i) Select the shortest path vertex (Here 2)
(ii) Check if there is a shorter path from selected vertex to other vertices

In example

from 2 → 3 cost = 2 + 4 = 6

Ans 6 < ∞

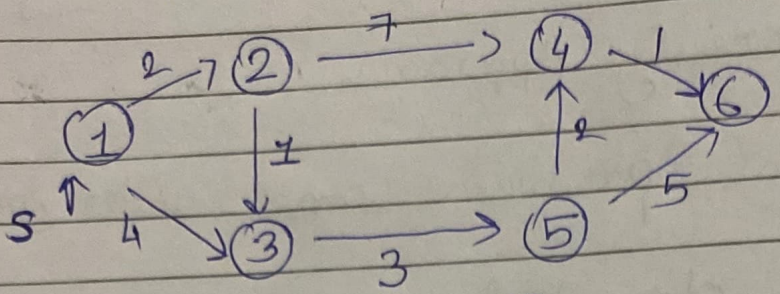
update the path to 6



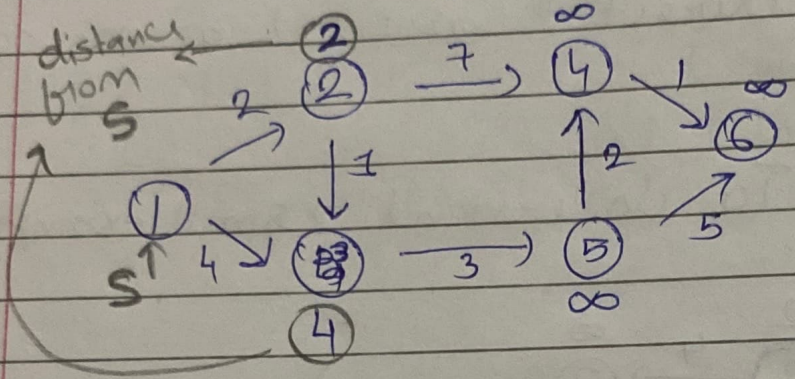
This updation is called relaxation.

```
if ( d[u] + c[u,v] < d[v] ) {
    d[v] = d[u] + c[u,v]
}
```

☆ Example

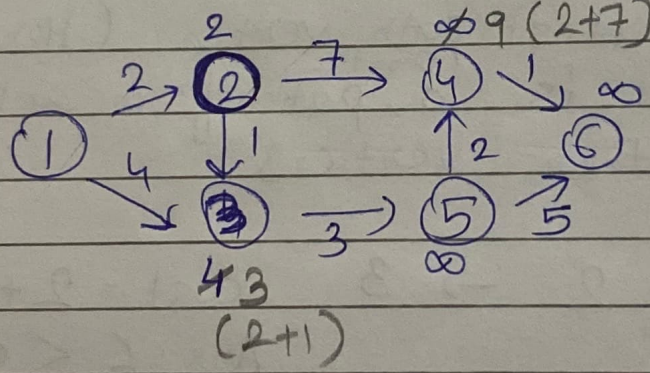


Initial distance from S — S-(1)

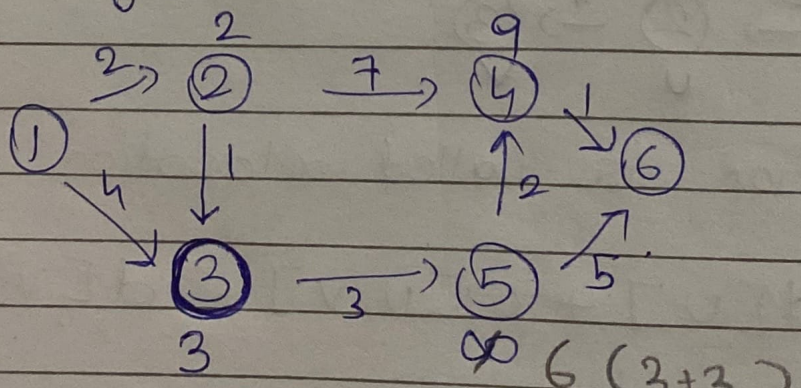


Repeating steps

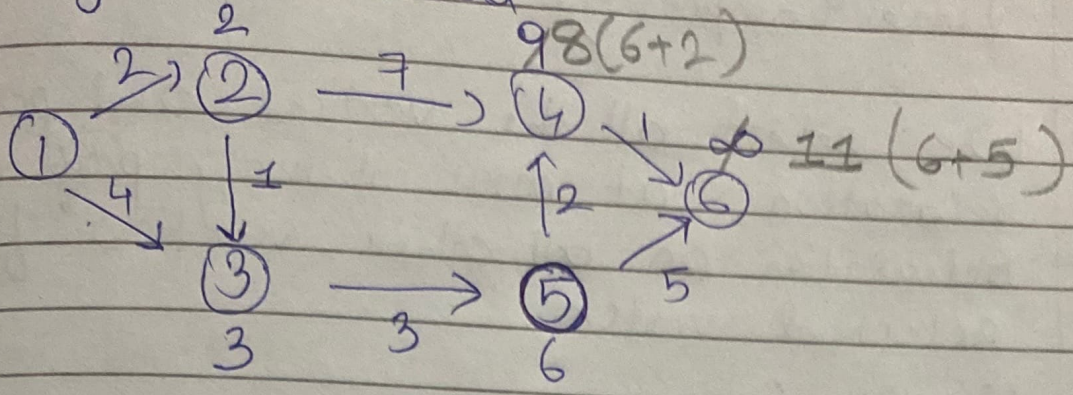
S-2 Select shortest path vertex (Here 2)



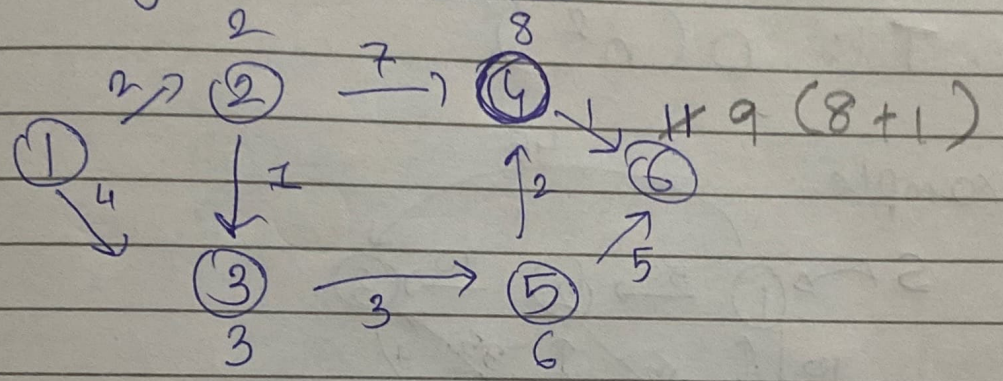
S-3 3 gets selected



5-4 5 gets selected

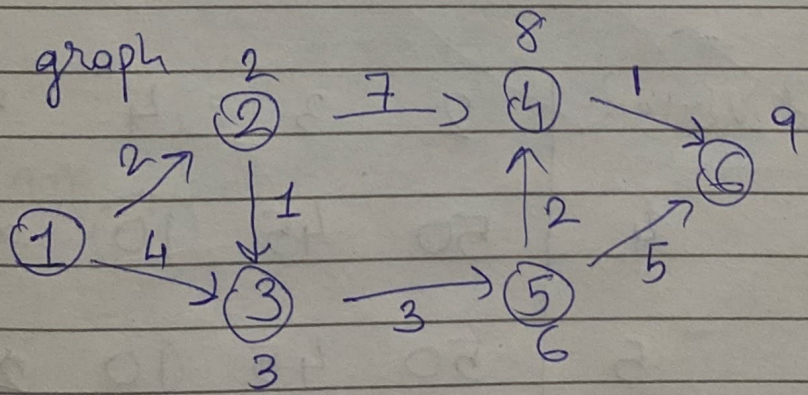


5-5 4 gets selected



5-6 6 gets selected but no other edges remain

Final graph



V	d[v]
2	2
3	3
4	8
5	6
6	9

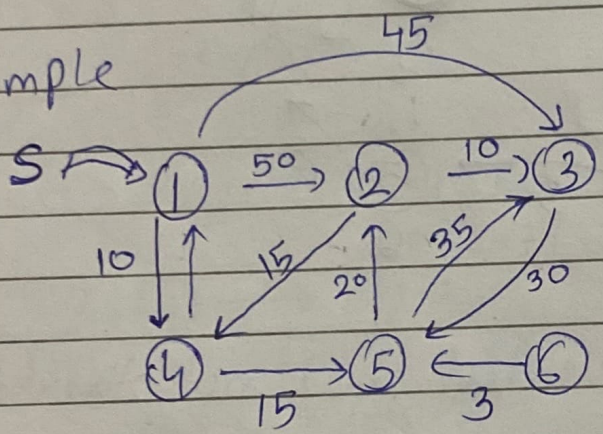
Time Complexity

We select all the vertices and perform relaxation, at most it may perform relaxation on all other vertices from selected vertex.

$$n = |V|\ |V| = n * n$$

$$T : O(n^2)$$

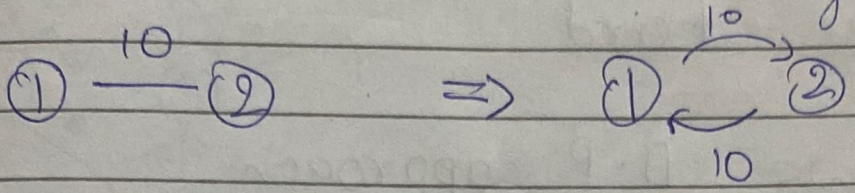
Example



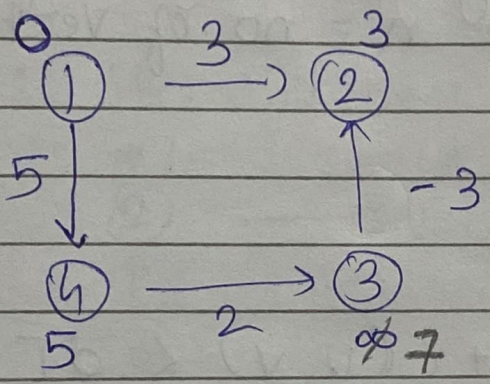
Selected Vertex	2	3	4	5	6
				min	
4	50	45	10	∞	∞
5	50	45	10	25	∞
2	45	45	10	25	∞
3	45	45	10	25	∞
6	45	45	10	25	∞

PAGE NO.:

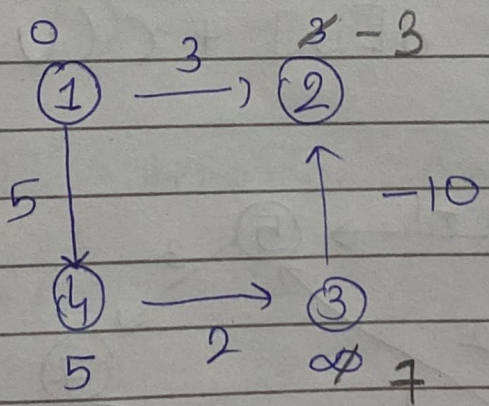
* For non-directed graph convert non directed edge to bi directional edge



* Drawback



Here dijkstra works



Here dijkstra fails

Hence it may or maynot work on negative edges

✡ Single - Source Shortest Path

✡ Bellman - Ford

⇒ Follows D·P approach

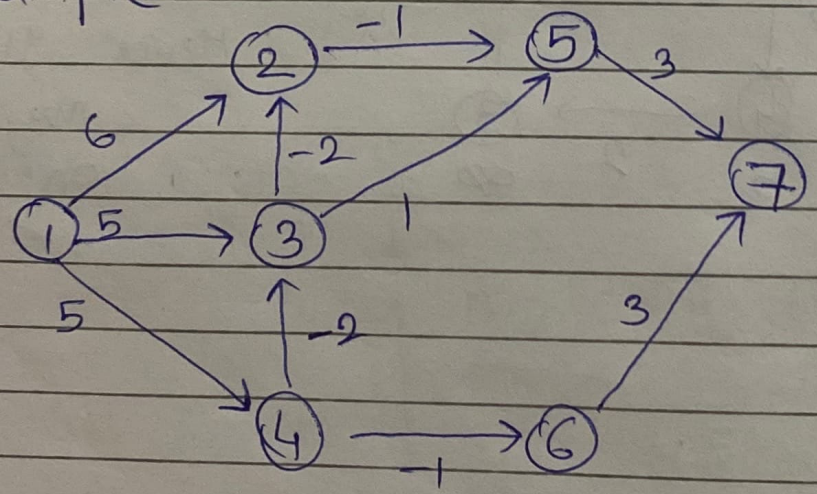
⇒ Go on relaxing all edges for n-1 times where n = no. of vertices.

✡ ~~Relax~~ Relaxation.
(u, v)

if (d[u] + c(u, v) < d[v])

d[v] = d[u] + c(u, v)

✡ Example

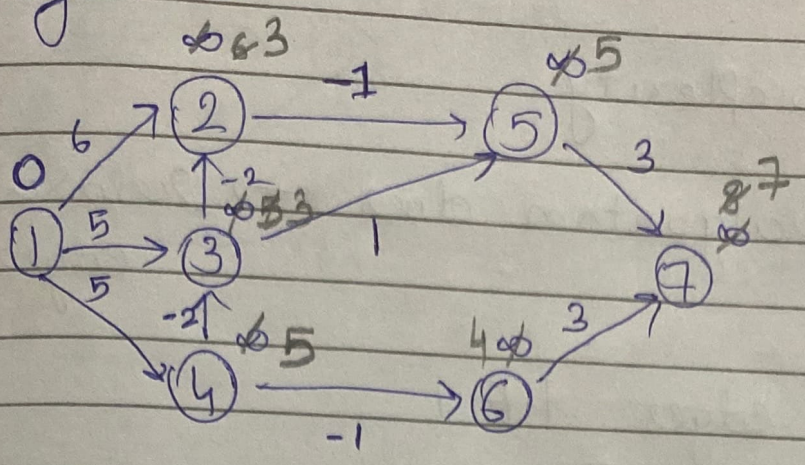


edge list

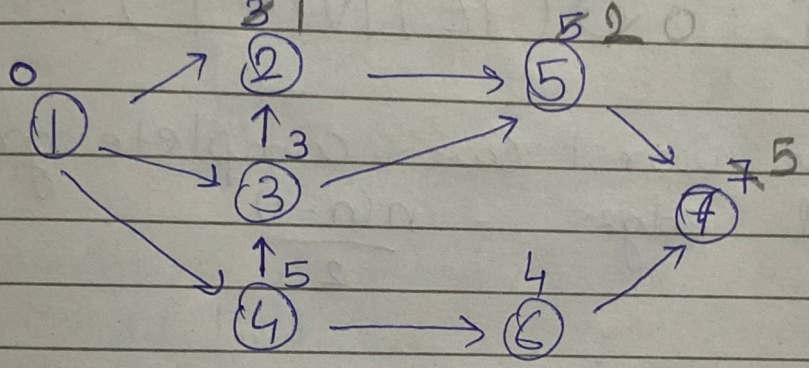
(1,2), (1,3), (1,4), (2,5), (3,2), (3,5), (4,3)
(4,6), (5,7), (6,7)

x ← distance

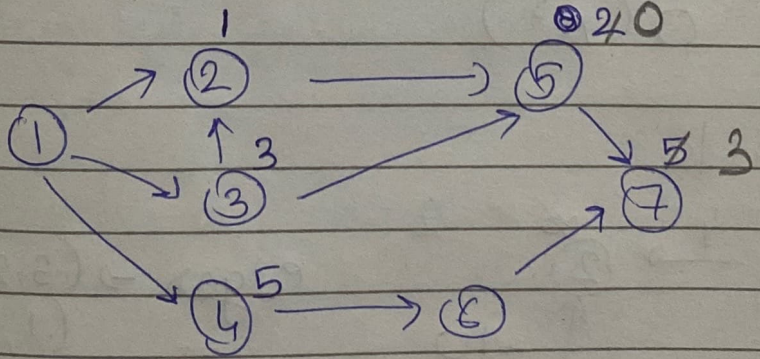
Initially



After 1 iteration over 10 edges



After 2nd iteration over 10 edges



After 3rd iteration
for 4, 5, 6 there is no change

1 – 0	5 – 0
2 – 1	6 – 4
3 – 3	7 – 3
4 – 5	

PAGE NO.:

Time complexity

What algorithm does → relaxation of edges

Total edges $|E|$

How many times it is relaxed $|V|-1$

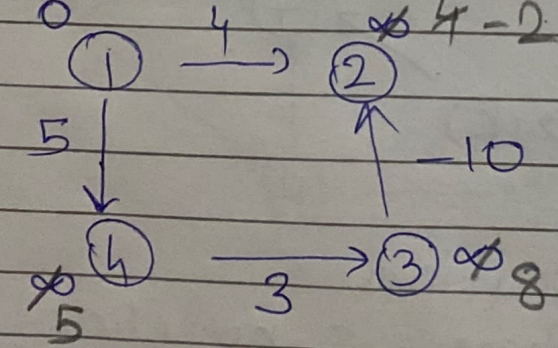
$$T : O(|E||V|-1)$$

In the worst case complete graph total edges $= \frac{n(n-1)}{2}$

$$T : \frac{n(n-1)}{2} \times (n-1)$$

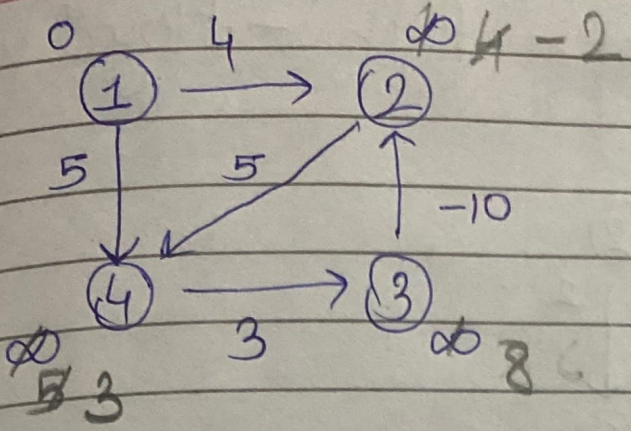
$$O(n^3)$$

★ Drawback



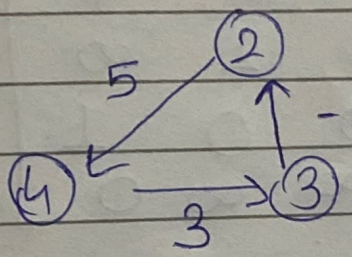
edges → (3,2), (4,3), (1,4), (1,2)

After 3rd iteration if we try 4th iteration nothing gets relaxed



After 3 iteration 4th iteration if we try relaxing in 4th we see
(4, 3) 3+3 = 6 < 8
one more relaxation which should not happen. Similarly for more iteration relaxation occurs

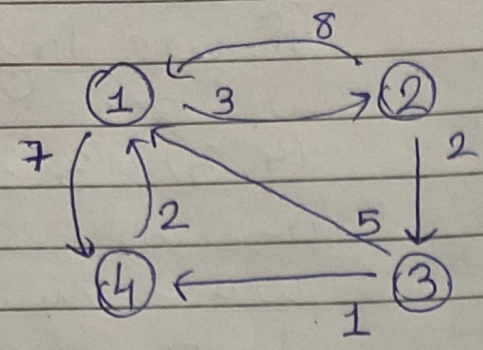
Reason



5+3 ~~8~~ -10
= -2

Negative weighted cycle, Every time we would find a relaxation.
Bellman-ford fails here.
But it can detect if we have negative weighted cycle.

☆ All pairs Shortest Path - Floyd Warshall



$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 2 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

Approach

Shortest path can be direct or via intermediate vertex hence we consider a vertices to be that intermediate vertex and calculate shortest path

$$A' = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 8 & \infty & 0 \end{bmatrix} \end{array}$$

A' denotes finding shortest path via intermediate vertex as 1.
Hence all paths directly connected to 1 stay same.

$A^0[2,3]$ $\quad$ $A^0[2,1] + A^0[1,3]$

2 $\quad <$ $\quad 8 + \infty$

Keep it 2

$A^0[2,4]$ $\quad$ $A^0[2,1] + A^0[1,4]$

$\infty \quad >$ $\quad 8 + 7 = 15$

Update to 15

$A^0[3,2]$ $\quad$ $A^0[3,1] + A^0[1,2]$

$\infty \quad >$ $\quad 5 + 3$

Now we find shortest path and keep 2 as intermediate vertex. from A'

$$A^2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ 1 & 0 & 3 & 5 & 7 \\ 2 & 8 & 0 & 2 & 15 \\ 3 & 5 & 8 & 0 & 1 \\ 4 & 2 & 8 & 7 & 0 \end{array}$$

$A^2[1,3] = A'[1,3]$ $\quad$ $A'[1,2] + A'[2,3]$

$\infty \quad >$ $\quad 3 + 2$

Update to 5

$A^2[1,4] = A'[1,4]$ $\quad$ $A'[1,2] + A'[2,4]$

$7 \quad <$ $\quad 3 + 15$

Keep 7

Similarly we get A^3 from A^2

$$A^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 5 & 6 \\ 7 & 0 & 2 & 3 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix} \end{array}$$

$$A^3[1,2] = A^2[1,2] \quad A^2[1,3] + A^2[3,2]$$

$$3 \quad < \quad 5 + 8$$

$$A^4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 5 & 6 \\ 5 & 0 & 2 & 3 \\ 3 & 8 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix} \end{array}$$

Recurrence formula

$$A^k(i,j) = \min\{A^{k-1}(i,j), A^{k-1}(i,k) + A^{k-1}(k,j)\}$$

Time complexity : $O(n^3)$