Dijkstra Algorithm

Prepared by:
Monirul Haque

Main idea of Dijkstra is that, we traverse our graph finding our shortest path, during that time we update the paths we have for already known nodes as soon as we find a shorter path to reach it. The Process of updating the paths is called edge relaxation.

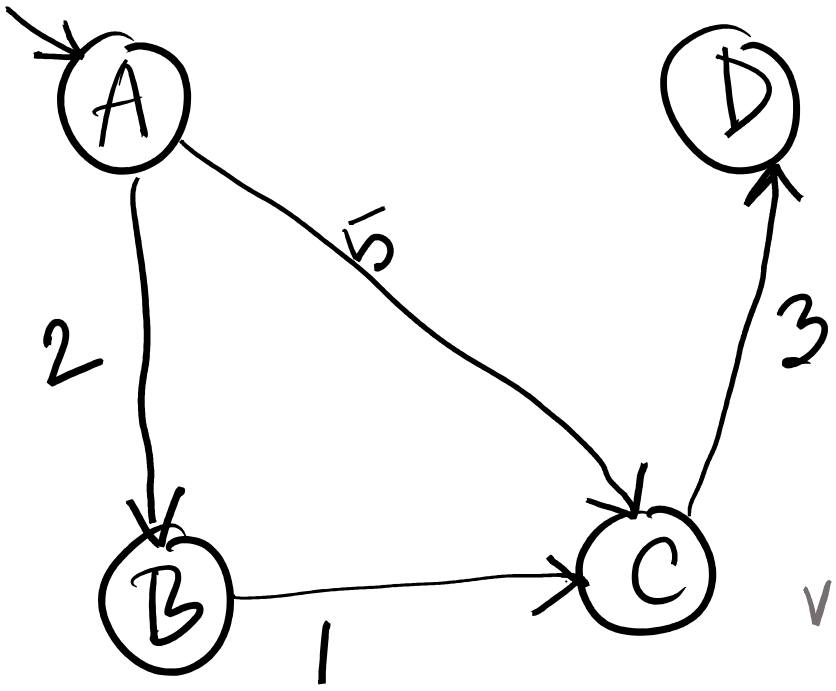
Suppose,

Relaxation equation,

$$d[v] = \min\left(d[v],\ d[u] + w(u,v)\right)$$

Example 1

$$d[v] = \min(d[v],\ d[u] + w(u,v))$$



priority Queue

PQ

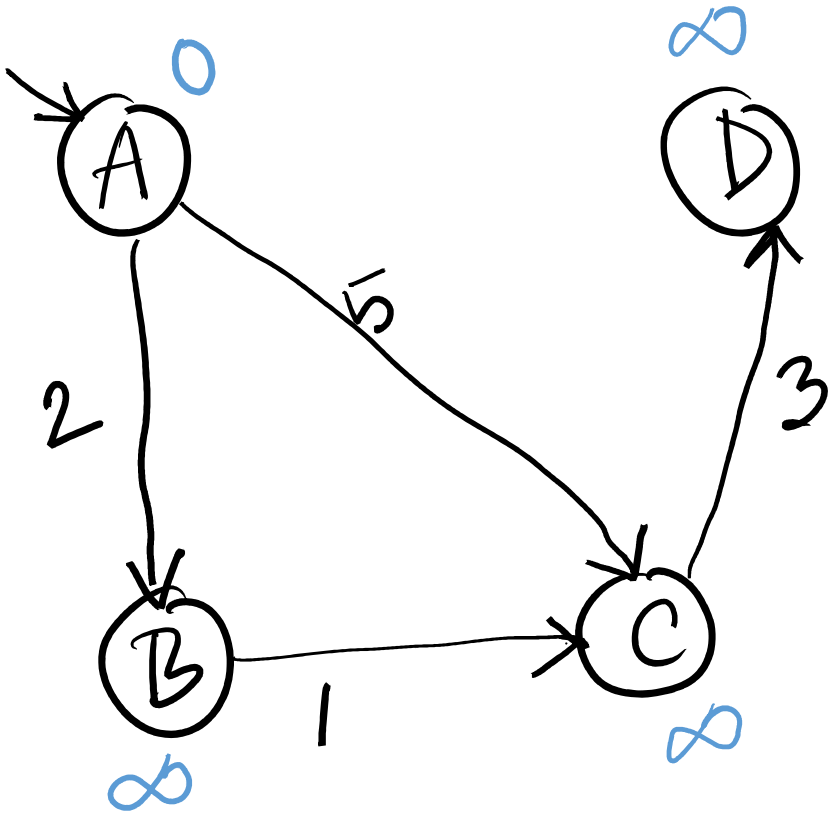
v : p

vertics

priority number

A	B	C	D

$$d[v] = \min(d[v],\ d[u] + w(u,v))$$



PQ

Pop

A : 0

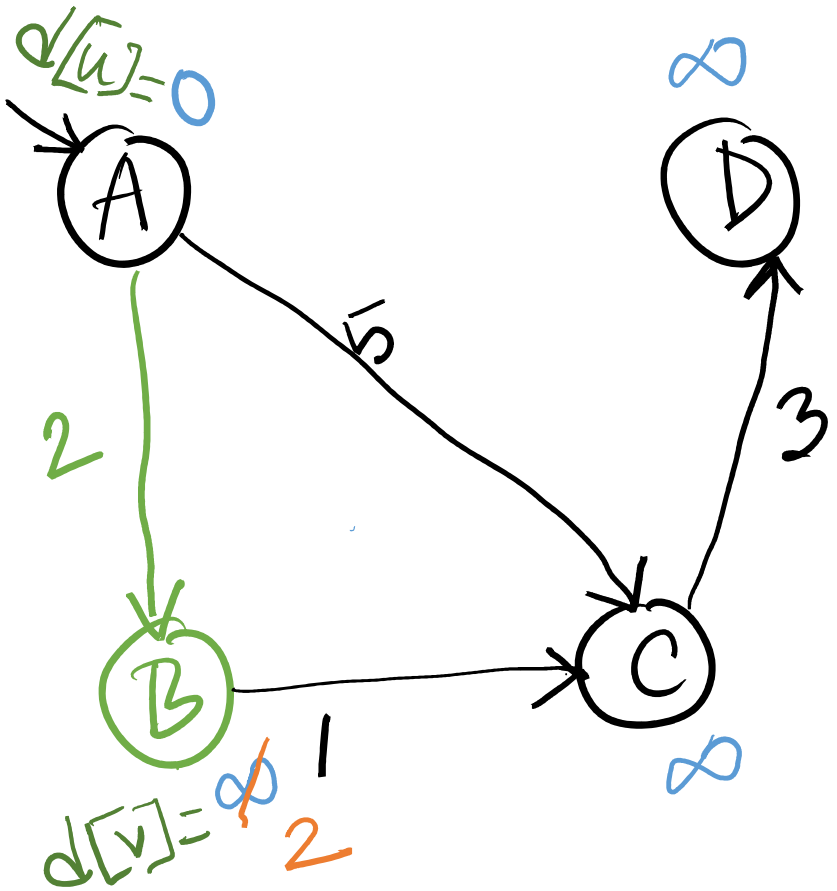
B : ∞

C : ∞

D : ∞

A	B	C	D
0	∞	∞	∞

$$d[v] = \min(d[v],\ d[u] + w(u,v))$$



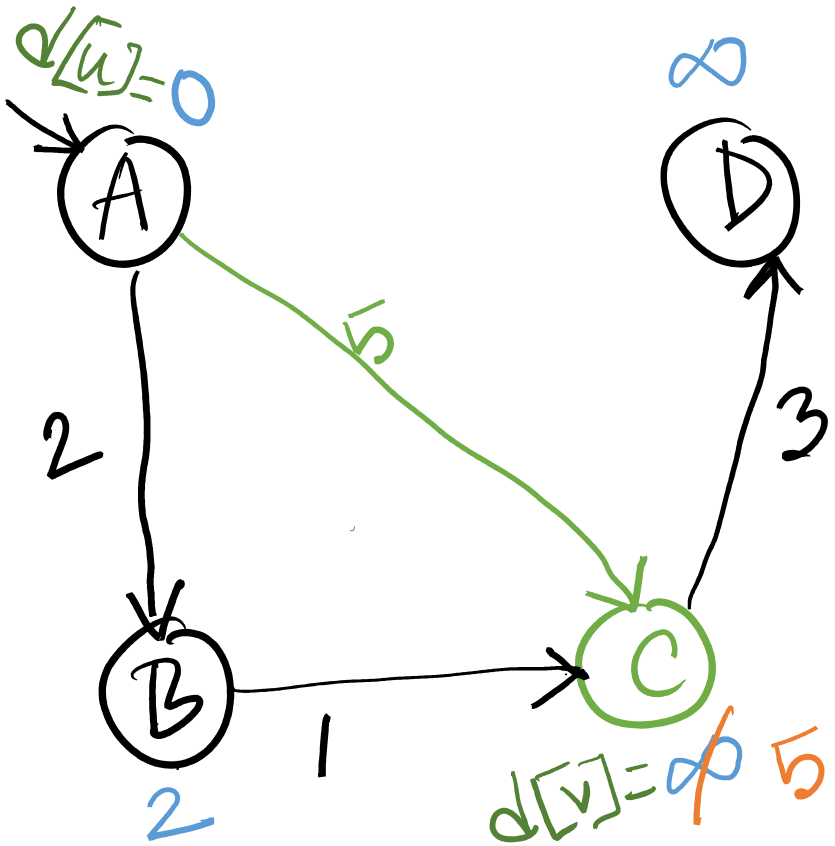
PQ

B : ∞ 2
C : ∞
D : ∞

	A	B	C	D
	0	∞	∞	∞
A	0	2		

S_0, $d[v] = \min(\infty, 0+2) = 2$

$$d[v] = \min(d[v],\ d[u] + w(u,v))$$



PQ

B : 2

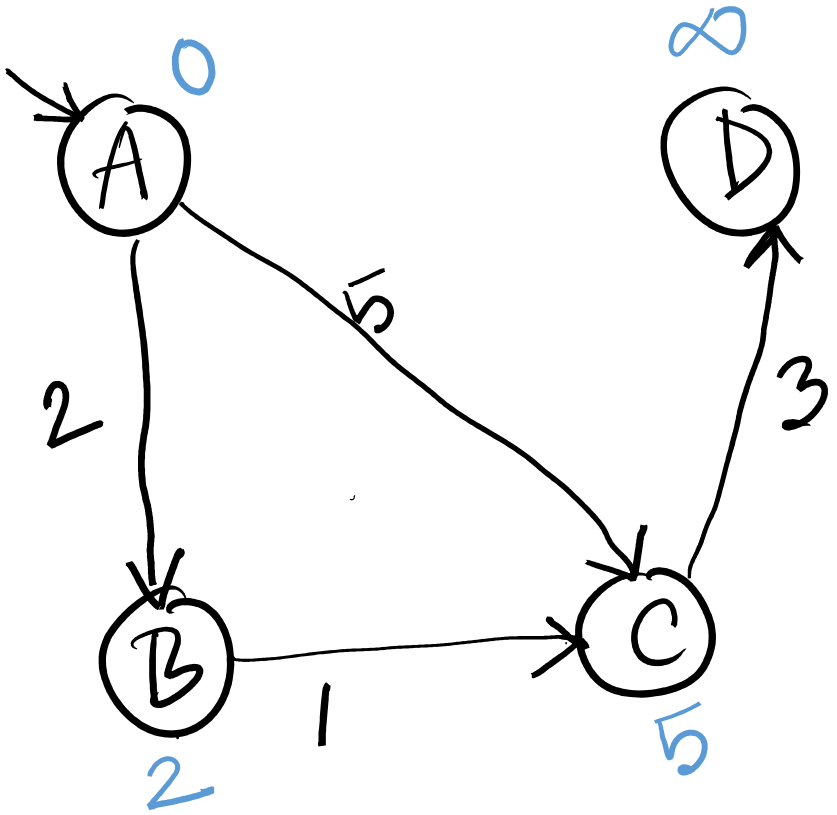
C : ~~∞~~ 5

D : ∞

	A	B	C	D
	0	∞	∞	∞
A	0	2	5	

So, $d[v] = \min(\infty, 0+5) = 5$

$$d[v] = \min(d[v],\ d[u] + w(u,v))$$



PQ

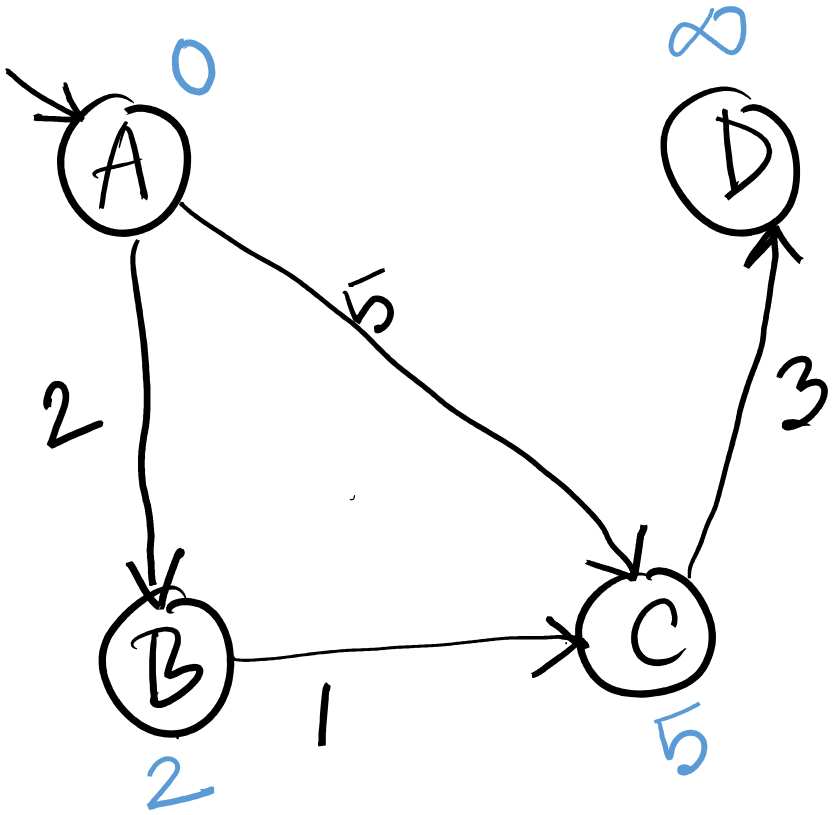
B 2

C 5

D: ∞

	A	B	C	D
	0	∞		
A	0	2		

$$d[v] = \min(d[v],\ d[u] + w(u,v))$$



PQ

B : 2

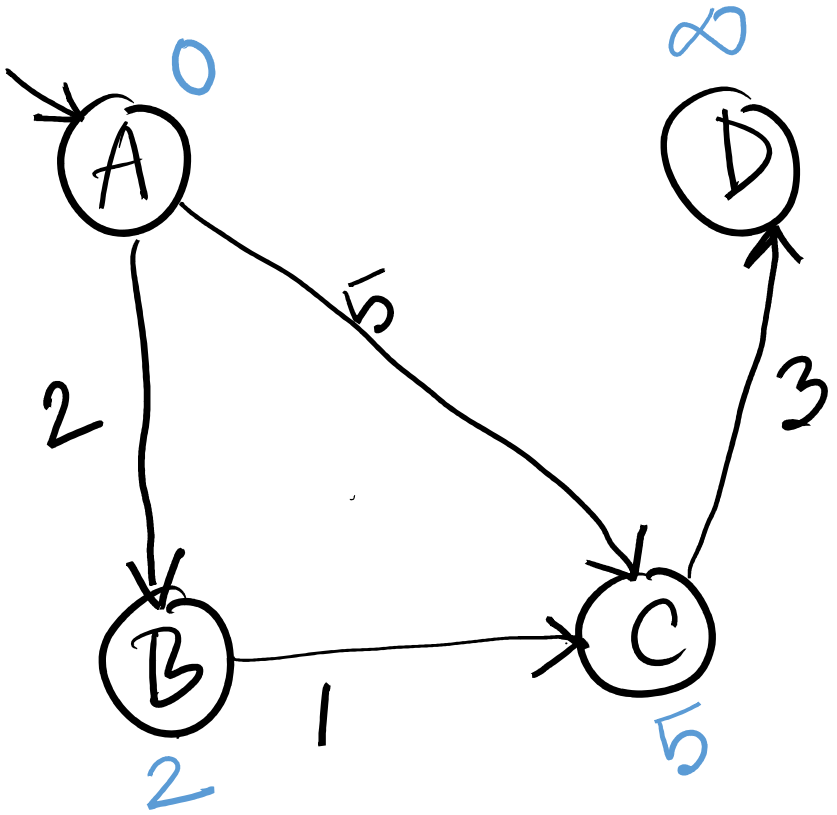
C : 5

D : ∞

Done with A

	A	B	C	D
	0	∞	∞	∞
A	0	2	5	∞

$$d[v] = \min(d[v],\ d[u] + w(u,v))$$



PQ

B : 2

C : 5

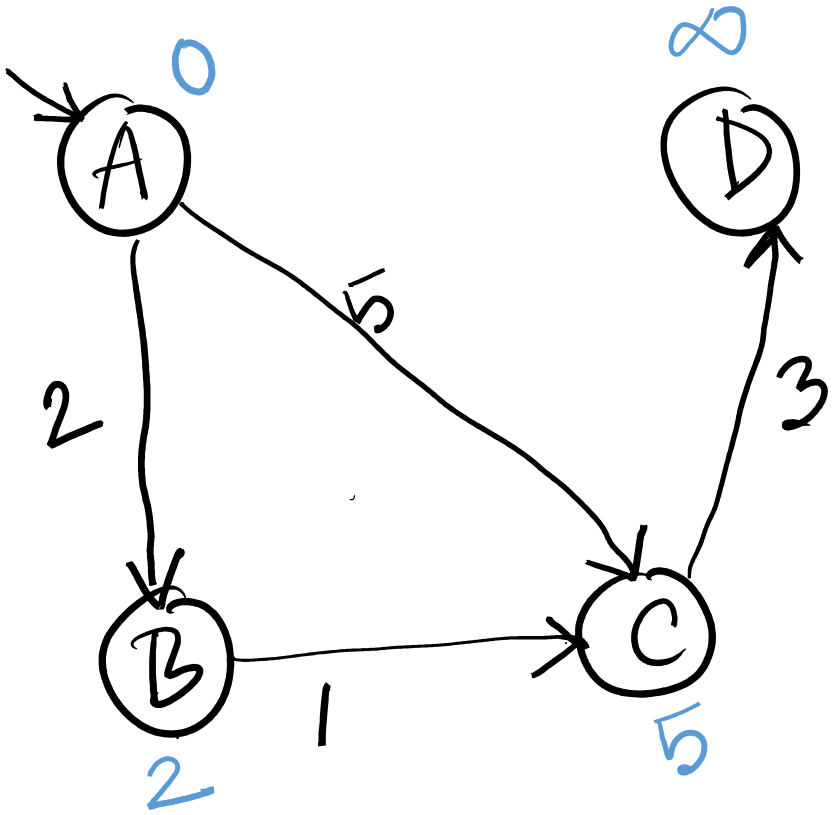
D : ∞

B has the smallest priority num

	A	B	C	D
	0	∞	∞	∞
A	0	2	5	∞

Closest to A is B

$$d[v] = \min(d[v],\ d[u] + w(u,v))$$



PQ

pop

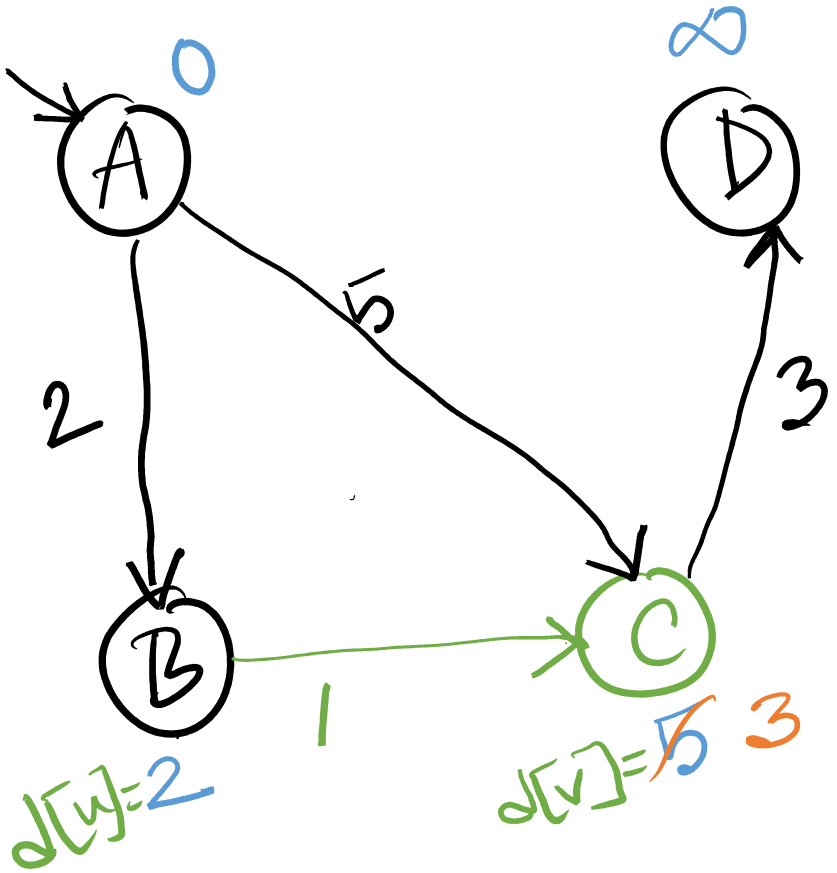
B : 2

C : 5

D : ∞

	A	B	C	D
	0	∞	∞	∞
A	0	2	5	∞

$$d[v] = \min(d[v],\ d[u] + w(u,v))$$



PQ

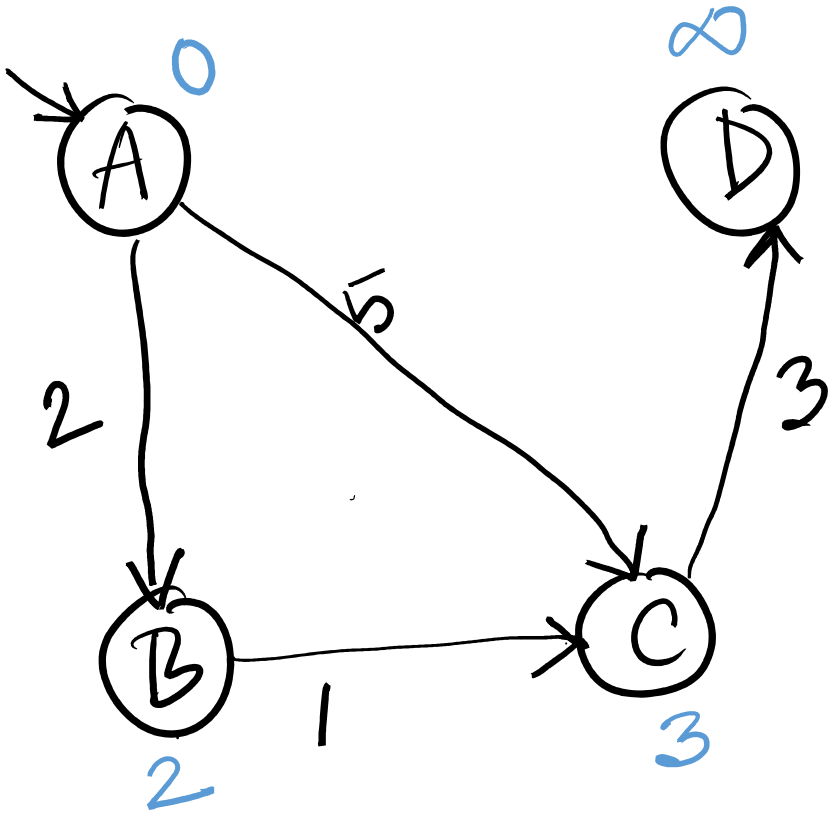
C : ~~5~~ 3

D : ∞

	A	B	C	D
	0	∞	∞	∞
A	0	2	5	∞
B	0	2	3	

So, $d[v] = \min(5, 2+1) = 3$

$$d[v] = \min(d[v],\ d[u] + w(u,v))$$



PQ

pop

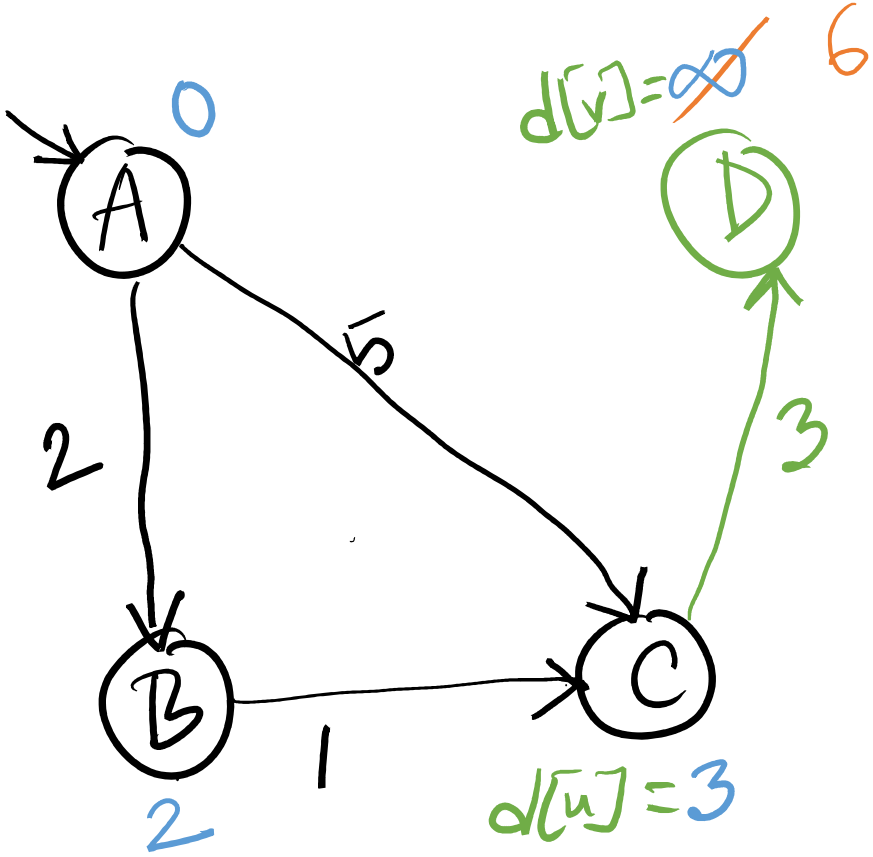
C : 3

D : ∞

	A	B	C	D
	0	∞	∞	∞
A	0	2	5	∞
B	0	2	3	∞

Done with B

$$d[v] = \min(d[v],\ d[u] + w(u,v))$$



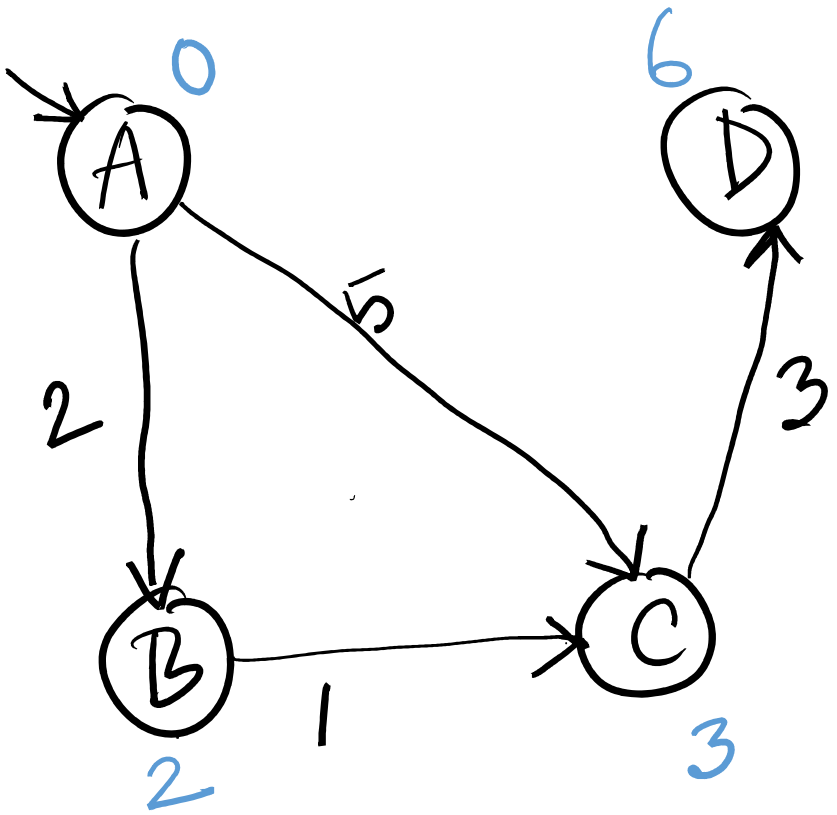
PQ

D: ~~∞~~ 6

	A	B	C	D
	0	∞	∞	∞
A	0	2	5	∞
B	0	2	3	∞
C	0	2	3	6

$$d[v] = \min(\infty,\ 3+3) = 6$$

$$d[v] = \min(d[v],\ d[u] + w(u,v))$$



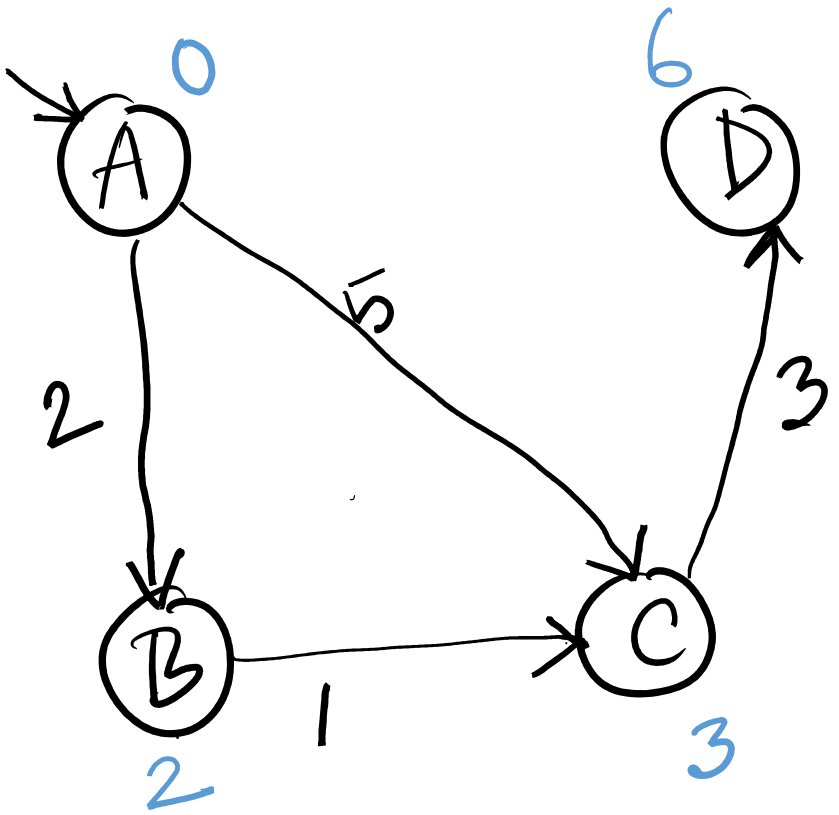
PQ

POP

D:6

	A	B	C	D
	0	∞	∞	∞
A	0	2	5	∞
B	0	2	3	∞
C	0	2	3	6

$$d[v] = \min(d[v],\ d[u] + w(u,v))$$



PQ

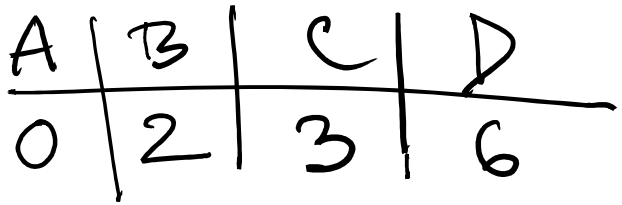
	A	B	C	D
	0	∞	∞	∞
A	0	2	5	∞
B	0	2	3	∞
C	0	2	3	6
D	0	2	3	6

Finally, we have shortest path of all the node from source

Notice

- Dijkstra is a **single source shortest path** algorithm

From the previous example, the end result we got was,

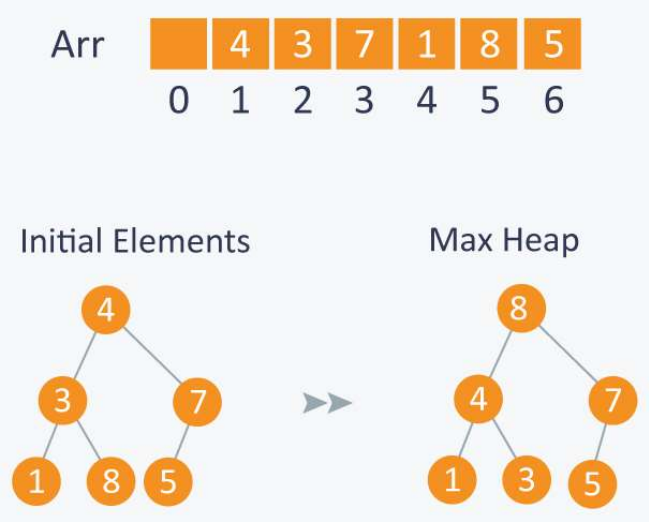


The arrays has distance of all the nodes from the source node.
So, if we want to know the shortest distance of B to D, we need to run the algorithm again, keeping B as the source node.

Notice

- Dijkstra uses a **min heap Priority Queue**

A simple example of heapifying process a max heap priority queue, where at first the values less than the initial root node goes to left and greater than the root node goes to right node.



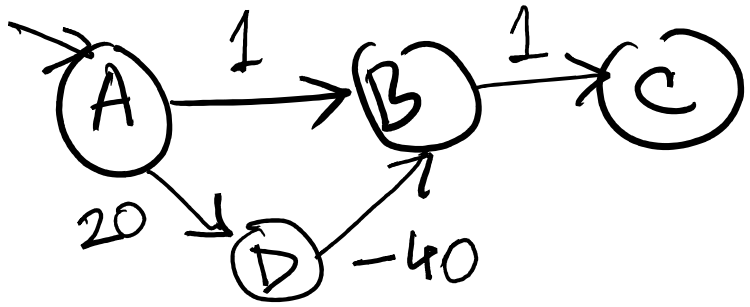
Min heap priority queue works exactly same, just the opposite way.

Picture source: https://www.hackerearth.com/practice/notes/heaps-and-priority-queues/

Notice

- Dijkstra can produce wrong output if any negative edge is present

For example, look at the following graph,

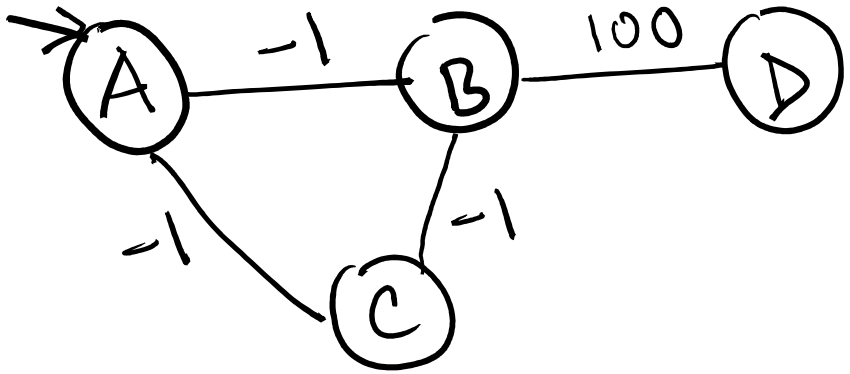


For this graph, according to Dijkstra algorithm, the shortest path distance from A to B would be 1. However, the correct path should be A → D → B and the distance should be -20.

Notice

- Dijkstra falls in infinite loop when a negative Cycle is present in the graph

For example, look at the following graph,



Notice

- When weights of all the edges of the nodes or vertices are 1. The graph acts like an un-directed graph and Dijkstra works like BFS. Also the time complexity becomes **O(V+E)**

Pseudo-code

```
Dijkstra(G,s):                          # G denoting graph, s denoting source
    for each v in G:                    # v denoting vertices/nodes
        d[v] ← ∞                        # d to keep track of the distance from source
        p[v] ← None                     # p to keep track of the parent node
    d[s] = 0
    PQ ← (s, d[s])                      # pushing source node in Priority Queue
    while PQ not empty:
        u ← PQ.pop()                    # extract minimum
        for each vertex v ∈ Adj[u]:
            if d[v] > d[u] + w(u, v):           # edge relaxation process
                d[v] ← d[u] + w(u, v)           # updating the distance
                p[v] ← u                        # updating the parent node
                PQ ← (v, d[v])
```

Time Complexity

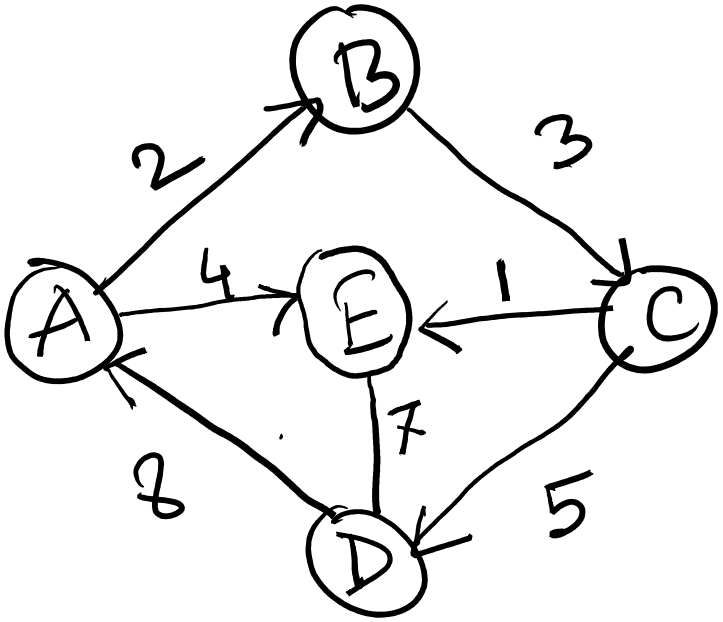
So, T.C = $O(E \log V)$

```
Dijkstra(G,s):
    for each v in G:
        d[v] ← ∞
        p[v] ← None
    d[s] = 0
    PQ ← (s, d[s])
    while PQ not empty:
        u ← PQ.pop()
        for each vertex v ∈ Adj[u]:
            if d[v] > d[u] + w(u, v):
                d[v] ← d[u] + w(u, v)
                p[v] ← u
                PQ ← (v, d[v])
```

Just traversing all the edges here $O(E)$

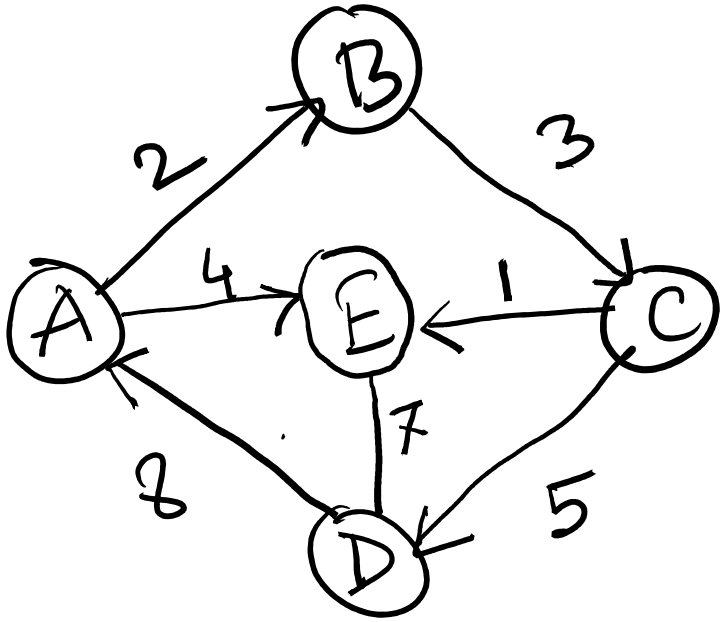
For every insertion Priority Queue takes $O(\log V)$ time to sort

Practice Simulation 1



*Notice D to E edge is undirectional

Practice Simulation 1 (Solution)



	A	B	C	D	E
	0	∞	∞	∞	∞
A	0	2	∞	∞	4
B	0	2	5	∞	4
E	0	2	5	11	4
C	0	2	5	10	4
D	0	2	5	10	4

References

- Slides of Mumit Khan Sir
- http://www.shafaetsplanet.com/
- Algorithms in a Nutshell by *George T. Heinmen*
- Grokking Algorithm by *Aditya* Y. Bhargava
- https://github.com/jwasham/coding-interview-university
- https://github.com/64bitpandas/cs61b-notes/blob/master/algorithms/shortest-paths/dijkstras-algorithm.md
- https://www.hackerearth.com/practice/notes/heaps-and-priority-queues/
- https://books.goalkicker.com/AlgorithmsBook/