Lab Report-2

Dijkstra: Dijkstra's Algorithm is a method used in computer science to find the **shortest path** from a starting node (point) to all other nodes in a network (graph). Its primary goal is to determine the most efficient route by minimizing the total "cost" accumulated along the path. This algorithm is specifically designed to work on **weighted graphs**.

Graph Search Example:

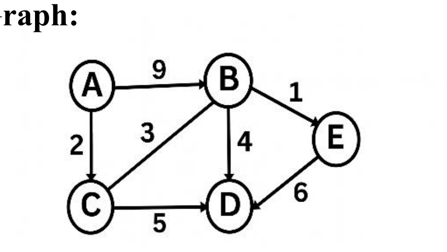
Relaxation Formula: It's the mechanism that checks if a shorter path to a neighbor has been found.

Formula: $if(d[u]+c(u,v)<d[v]) \implies d[v]=d[u]+c(u,v)$

Meaning of Each Symbol

- **u** : The *current node* (the node we are exploring now).
- **v** : A *neighbor node* of uuu.
- **d[u]** : The shortest distance from the start node to u (known so far).
- **d[v]** : The shortest distance from the start node to v (recorded so far).
- **c(u,v)** : The cost/weight of the edge from u to v.

Graph:



Find the path from A to E, where A is the source Node.

Initialization:

	A	**B**	**C**	**D**	**E**
A	$\boxed{0}$	∞	∞	∞	∞

Node	A	B	C	D	E
Parent	-1	-1	-1	-1	-1

Step-1:

	A	**B**	**C**	**D**	**E**
A	$\boxed{0}$	∞	∞	∞	∞
C	$\boxed{0}$	9	$\boxed{2}$	∞	∞

Check Condition for B:

d[u] + c(u,v) < d[v] ⇒ 0 + 9 < ∞
Condition is **true**, so we **relax**:
d[v] = d[u] + c(u,v) = 0+9 = 9

Check Condition for C:

d[u] + c(u,v) < d[v] ⇒ 0 + 2< ∞
Condition is **true**, so we **relax**:
d[v] = d[u] + c(u,v) = 0+2 = 2

Node	A	B	C	D	E
Parent	-1	A	A	-1	-1

Step-2:

	A	**B**	**C**	**D**	**E**
A	$\boxed{0}$	∞	∞	∞	∞
C	$\boxed{0}$	9	$\boxed{2}$	∞	∞
B	$\boxed{0}$	$\boxed{5}$	$\boxed{2}$	7	∞

Check Condition for B:

d[u] + c(u,v) < d[v] ⇒ 2 + 3 < 9
Condition is **true**, so we **relax**:
d[v] = d[u] + c(u,v) = 2+3 = 5

Check Condition for D:

d[u] + c(u,v) < d[v] ⇒ 2 +5 < ∞
Condition is **true**, so we **relax**:
d[v] = d[u] + c(u,v) = 2 + 5 = 7

Node	A	B	C	D	E
Parent	-1	C	A	B	-1

Step-3:

	A	**B**	**C**	**D**	**E**
A	$\boxed{0}$	∞	∞	∞	∞
C	$\boxed{0}$	9	$\boxed{2}$	∞	∞
B	$\boxed{0}$	$\boxed{5}$	$\boxed{2}$	7	∞
E	$\boxed{0}$	$\boxed{5}$	$\boxed{2}$	7	$\boxed{6}$

Check Condition for D:

d[u] + c(u,v) < d[v] ⇒ 5+4 > 7
Condition is **false**, so:
d[v] = 7

Check Condition for E:

d[u] + c(u,v) < d[v] ⇒ 5+1 < ∞
Condition is **true**, so we **relax**:
d[v] = d[u] + c(u,v) = 5 +1 = 6

Node	A	B	C	D	E
Parent	-1	C	A	7	B

Step-4:

	A	**B**	**C**	**D**	**E**
A	$\boxed{0}$	∞	∞	∞	∞
C	$\boxed{0}$	9	$\boxed{2}$	∞	∞
B	$\boxed{0}$	$\boxed{5}$	$\boxed{2}$	7	∞
E	$\boxed{0}$	$\boxed{5}$	$\boxed{2}$	7	$\boxed{6}$
D	$\boxed{0}$	$\boxed{5}$	$\boxed{2}$	$\boxed{7}$	$\boxed{6}$

Check Condition for D:

d[u] + c(u,v) < d[v] ⇒ 6+6 > 7
Condition is **false**, so:
d[v] = 7

Node	A	B	C	D	E
Parent	-1	C	A	7	B

- **The Shortest path from A to E Node:**
  A → C → B → E

Pseudocode:

```
Dijkstra(G, source) {
    for each vertex u in G {                    // initialization
        dist[u] = infinity
        parent[u] = null
    }
    dist[source] = 0                            // initialize source
    PQ = priority_queue()                       // min-priority queue
    PQ.push( (0, source) )                      // (distance, vertex)
    while (PQ is not empty) {
        (du, u) = PQ.pop()                      // extract node with min distance
        for each v in Adj[u] {                  // explore all neighbors
            w = weight(u, v)
            if (dist[u] + w < dist[v]) {        // relaxation
                dist[v] = dist[u] + w
                parent[v] = u
                PQ.push( (dist[v], v) )         // push updated distance
            }
        }
    }
    for each vertex u in G {
        print dist[u]
    }   }
```