Single Source Shortest Path Algorithm.

Dijkstra's Algorithm

Pseudo Code

Dijkstras(v_s)

```
for (i = 1 to n)
    visited[i] = 0 ;
    dist[i] = ∞ ;
endfor
dist[s] = 0
for (i = 1 to n)
    select j such that visited[j] != 1 (that means v_j is not visited) and
    dist[j] is minimum
    visited[j] = 1
    for each edge (j,k) ∈ E such that visited[k] != 1
        if dist[k] > dist[j] + W[j][k]
            dist[k] ← dist[j] + W[j][k]
        endif
    endfor each
endfor
```

O(n) — if visited array is used.

or

O(log n) if priority Queue (Binary heaps are used)

if Priority Queue is used then checking visited[k] != 1 requires O(log n) time.

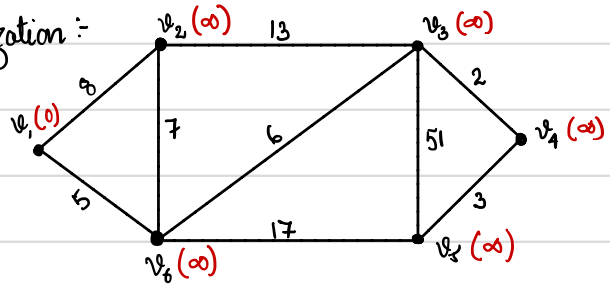
OR $dist[k] = \min\{dist[k], dist[j] + W[j][k]\}$

$O(n^2)$ even if the adjacency list representation is employed for sparse graph.

Complexity of Dijkstra's algorithm is $O(n \log n + m \log n)$ if binary heap data structure is used. and its complexity is $O(n + m \log n)$ if fibonacci heap is used.

Example :-

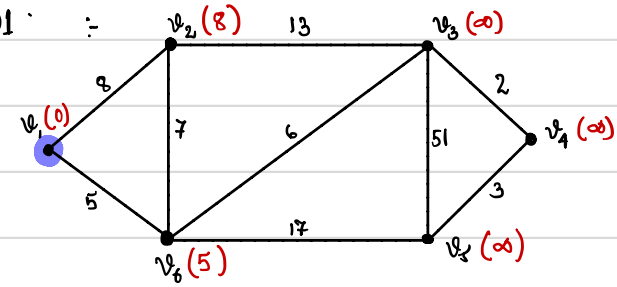
Initialization :-



visited = | 0 | 0 | 0 | 0 | 0 | 0 |

dist = | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |

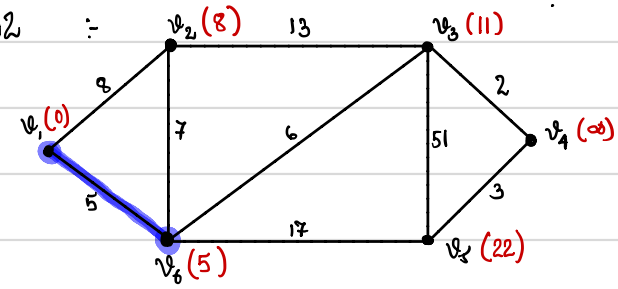
Iteration 01 :-



visited = | 1 | 0 | 0 | 0 | 0 | 0 |

dist = | 0 | 8 | ∞ | ∞ | ∞ | 5 |

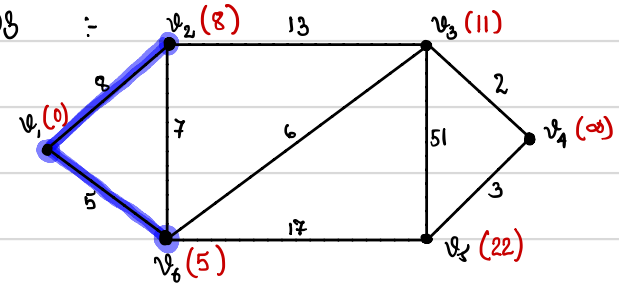
Iteration 02 :-



visited = | 1 | 0 | 0 | 0 | 0 | 1 |

dist = | 0 | 8 | 11 | ∞ | 22 | 5 |

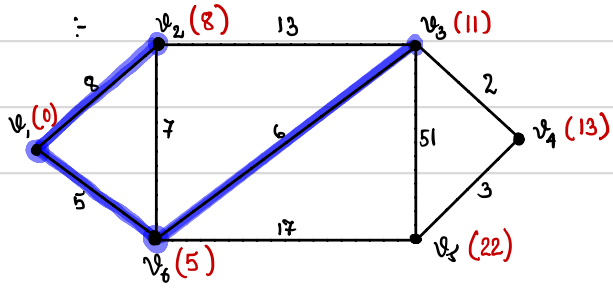
Iteration 03 :-



visited = | 1 | 1 | 0 | 0 | 0 | 1 |

dist = | 0 | 8 | 11 | ∞ | 22 | 5 |

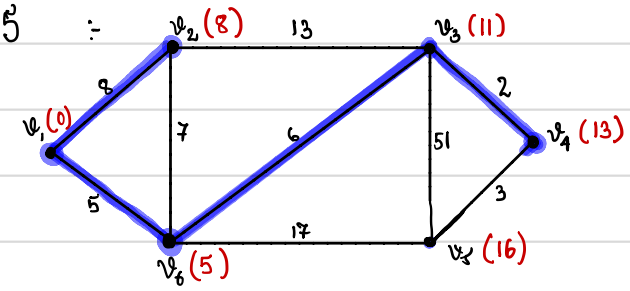
Iteration 04 :-



visited = | 1 | 1 | 1 | 0 | 0 | 1 |

dist = | 0 | 8 | 11 | 13 | 22 | 5 |

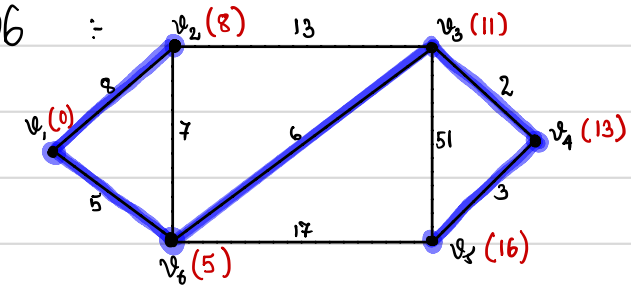
Iteration 05 :-



visited =	1	1	1	1	0	1

dist =	0	8	11	13	16	5

Iteration 06 :-



visited =	1	1	1	1	1	1

dist =	0	8	11	13	16	5