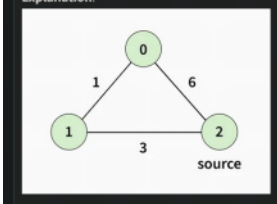
Djikstra Algorithm :-

Given an undirected, weighted graph with **V** vertices numbered from 0 to V-1 and **E** edges, represented by 2d array **edges[][]**, where edges[i]=[u, v, w] represents the **edge** between the nodes u and v having w **edge weight**.
You have to find the **shortest distance** of all the vertices from the source vertex **src**, and return an array of integers where the **ith** element denotes the shortest distance between **ith** node and source vertex **src**.

Note: The Graph is connected and doesn't contain any negative weight edge.

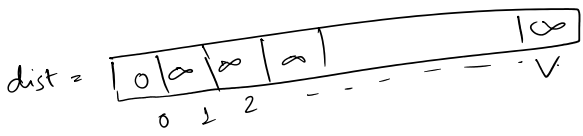
Examples:

Input: V = 3, edges[][] = [[0, 1, 1], [1, 2, 3], [0, 2, 6]], src = 2
Output: [4, 3, 0]
Explanation:



Shortest Paths:
For 2 to 0 minimum distance will be 4. By following path 2 -> 1 -> 0
For 2 to 1 minimum distance will be 3. By following path 2 -> 1
For 2 to 2 minimum distance will be 0. By following path 2 -> 2

Constraints:
$1 \leq V \leq 10^5$
$1 \leq E = edges.size() \leq 10^5$
$0 \leq edges[i][j] \leq 10^4$
$0 \leq src < V$




```
class Solution{
public:
vector<int> djikstra(int V, vector<vector<int>>& edges,
 int src){
 vector<vector<pair<int, int>>> graph(V);
 vector<int> dist(V, INT_MAX);
for(int i=0; i< edges.size(); i++){
graph[edges[i][0]].push_back(make_pair(edges[i][1],
  edges[i][2]));
graph[edges[i][1]].push_back(make_pair(edges[i][0],
 edges[i][2]));}

priority_queue<pair<int, int>, vector<pair<int, int>>,
 compare> q;
 q.push(make_pair(src, 0));
 while(!q.empty()){
      pair<int, int> p=q.top();
       q.pop();
      if(dist[p.first]>p.second) dist[p.first]=p.second;
```

```
        q.pop();
if(dist[p.first] > p.second) dist[p.first] = p.second;
else continue;

for(int i=0; i<graph[p.first].size(); i++){
 int x = p.second + graph[p.first][i].second;
 q.push(make_pair(graph[p.first][i].first, x));

  }}
  return dist; }

 struct compare{
   bool operator()(const pair<int,int>& a,
    const pair<int,int>& b){

    return a.second > b.second;

   }
 };
};
```

T.C. $O(E \log V)$

S.C. $O(V)$