DIJKSTRA'S ALGORITHM

We need to find the shortest distance from the source to all the other nodes in the graph. It can be done over directed or undirected graph.

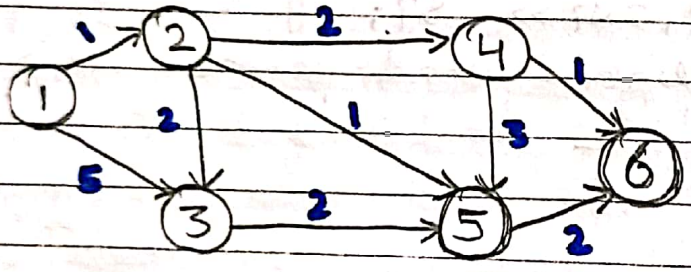
First we need to maintain a

Page No.
Date

priority queue, set or multiset, then we ~~just~~ need to perform BFS over every child we visit and update the current distance to the minimum possible one or make no changes. Then simply do this for all the nodes, until all the nodes are marked visited.

At the end we have the shortest distance, from source to all the other nodes in the graph.

e.g.:



Pq	DIST	VIS
~~(0,1)~~	1 - ∞ 0	1
VISITED (5,3)	2 - ∞ 1	1
~~(1,2)~~	3 - ∞ ~~5~~ 3	1
~~(3,3)~~	4 - ∞ 3	1
~~(2,5)~~	5 - ∞ 2	1
~~(3,4)~~	6 - ∞ 4	1
~~(4,6)~~		

CODE

```
const int N = 1e5+10;
const int INF = 1e9+10;

-> Pair of vector to store node and its
   Height together
vector<pair<int,int>> g[N];
-> Visited array
vector<int> vis(N);
-> Distance array initialized with INF
vector<int> dis(N, INF);

void dijkstra(int source)
{
   set<pair<int,int>> s; // set of pair
   to store weight and node resp.
   (sorted according to weight as weight
   is first)

   -> Inserting source
   s.insert({0, source});
   -> setting source distance to 0
   dis[source] = 0;

   -> Run until set is empty, just like BFS
   while(s.size() > 0)
   {
      auto node = *s.begin(); // getting
      1st element out of set
      int v = node.s;
      int v-dis = node.f;
      s.erase(s.begin()); // removing
      first element from set.
```

```
        IF(VIS[V]) CONTINUE;
        VIS[V] = 1;
        FOR (AUTO CHILD : g[V])
        {
            INT CHILD_V = CHILD.F;
            INT WT = CHILD.S;
            IF (DIST[V] + WT < DIST[CHILD_V])
            {
                DIST[CHILD_V] = DIST[V] + WT;
                ST.INSERT({DIST[CHILD_V], CHILD_V
                          });
            }
        }
    }

    // T.C = O(V + E log(V))
}

INT MAIN()
{
    INT n, m;
    CIN >> n >> m;

    ~~g[x].PB({y, w~~
    FOR (INT i=0; i<m; i++)
    {
        INT x, y, WT;
        CIN >> x >> y >> WT;
        -> Directed weighted graph
        g[x].PB({y, WT});
    }
```

-> Running Dijkstra from source 1

```
DIJKSTRA(1);
```

-> Printing shortest distance of all nodes from source node

```
FOR (INT i=1; i <= n; i++)
{
    COUT << "DIST OF NODE " << i <<
    " FROM SOURCE 1: " << DIST[i]
     << "\n";
}

    RETURN 0;
}
```

INPUT :

```
6 9   -> no. of vertex and edge
1 2 1   -> node, node, wt
1 3 5
2 3 2
2 4 2
2 5 1
3 5 2
4 5 3
4 6 1
5 6 2
```

OUTPUT :

```
Dist of node 1 from source 1 : 0
```

Page No.
Date

" " " 2 " " : 1
" " " 3 " " " : 3
" " " 4 " " " : 3
" " " 5 " " " : 2
" " " 6 " " " : 4