Dijkstra in C++

```
#include <bits/stdc++.h>
using namespace std;

class Solution
{
public:
   // Function to find the shortest distance of all
the vertices
   // from the source vertex S.
   vector<int> dijkstra(int V,
vector<vector<int>> adj[], int S)
   {

      // Create a priority queue for storing the
nodes as a pair {dist,node}
      // where dist is the distance from source to
the node.
      priority_queue<pair<int, int>,
vector<pair<int, int>>, greater<pair<int, int>>>
pq;

      // Initialising distTo list with a large
number to
      // indicate the nodes are unvisited initially.
      // This list contains distance from source to
the nodes.
      vector<int> distTo(V, INT_MAX);

      // Source initialised with dist=0.
      distTo[S] = 0;
      pq.push({0, S});

      // Now, pop the minimum distance node
first from the min-heap
      // and traverse for all its adjacent nodes.
      while (!pq.empty())
      {
         int node = pq.top().second;
         int dis = pq.top().first;
         pq.pop();

         // Check for all adjacent nodes of the
popped out
         // element whether the prev dist is larger
than current or not.
         for (auto it : adj[node])
         {
            int v = it[0];
            int w = it[1];
            if (dis + w < distTo[v])
            {
               distTo[v] = dis + w;

               // If current distance is smaller,
               // push it into the queue.
               pq.push({dis + w, v});
            }
         }
      }
      // Return the list containing shortest
distances
```

Graph Setup

Given:

- **Vertices (V):** 3
- **Source (S):** 2
- **Adjacency list (adj):**

adj[0] = {{1, 1}, {2, 6}};
adj[1] = {{2, 3}, {0, 1}};
adj[2] = {{1, 3}, {0, 6}};

This translates to:

From	To	Weight
0	1	1
0	2	6
1	2	3
1	0	1
2	1	3
2	0	6

🔁 Dijkstra's Algorithm

Start from source 2, initialize:

distTo = [∞, ∞, 0]
pq = [(0, 2)]

Now iterate:

Step	Node	Pop (dist,node)	Neighbors	Update Distances	pq After
1	2	(0, 2)	(1,3), (0,6)	dist[1] = 3, dist[0] = 6	(3,1), (6,0)
2	1	(3, 1)	(2,3), (0,1)	dist[0] = min(6, 4) = 4	(4,0), (6,0)
3	0	(4, 0)	(1,1), (2,6)	dist[1] already 3 < 5 → skip	(6,0)
4	0	(6, 0)	-	Already visited with smaller	—

📄 Final Distance Array:

res = [4, 3, 0]

Means:

Vertex	Shortest Distance from Source (2)
0	4

```
        // from source to all the nodes.
        return distTo;
    }
};

int main()
{
    // Driver code.
    int V = 3, E = 3, S = 2;
    vector<vector<int>> adj[V];
    vector<vector<int>> edges;
    vector<int> v1{1, 1}, v2{2, 6}, v3{2, 3}, v4{0, 1},
v5{1, 3}, v6{0, 6};
    int i = 0;
    adj[0].push_back(v1);
    adj[0].push_back(v2);
    adj[1].push_back(v3);
    adj[1].push_back(v4);
    adj[2].push_back(v5);
    adj[2].push_back(v6);

    Solution obj;
    vector<int> res = obj.dijkstra(V, adj, S);

    for (int i = 0; i < V; i++)
    {
        cout << res[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Vertex	Shortest Distance from Source (2)
1	3
2	0 (source itself)

🖳 **Output:**

4 3 0

Output:-
4 3 0