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

Adj list:-
adj[0] = {{1, 1}, {2, 6}}
adj[1] = {{2, 3}, {0, 1}}

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

Initialization

- distTo array (stores the shortest distance to each vertex):

  ```
  distTo = [INT_MAX, INT_MAX, 0]
  // Source vertex S=2 distance
  initialized to 0
  ```

- Priority queue pq (min-heap):

  ```
  pq = {(0, 2)}  // {distance,
  node}
  ```

Iteration 1: Process Node 2

- Pop (0, 2) from pq.
- For adjacent nodes of 2:
  - **Node 1** (weight = 3):

    ```
    plaintext
    Copy code
    distTo[1] = min(INT_MAX,
    0 + 3) = 3
    pq = {(3, 1)}
    ```

  - **Node 0** (weight = 6):

    ```
    plaintext
    Copy code
    distTo[0] = min(INT_MAX,
    0 + 6) = 6
    pq = {(3, 1), (6, 0)}
    ```

Iteration 2: Process Node 1

- Pop (3, 1) from pq.
- For adjacent nodes of 1:
  - **Node 2** (weight = 3):

    ```
    plaintext
    Copy code
    distTo[2] = min(0, 3 + 3)
    = 0  // No update,
    already shorter
    pq = {(6, 0)}
    ```

  - **Node 0** (weight = 1):

    ```
    distTo[0] = min(6, 3 + 1)
    ```

```
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;
}
```

```
= 4
pq = {(4, 0), (6, 0)}
```

Iteration 3: Process Node 0

- Pop `(4, 0)` from `pq`.
- For adjacent nodes of `0`:
  - **Node 1** (`weight = 1`):

    ```
    distTo[1] = min(3, 4 + 1)
    = 3  // No update,
    already shorter
    ```

  - **Node 2** (`weight = 6`):

    ```
    distTo[2] = min(0, 4 + 6)
    = 0  // No update,
    already shorter
    ```

Final State

- `distTo` array:

  ```
  distTo = [4, 3, 0]
  ```

Output

The shortest distances from source vertex `S = 2` to all vertices are:

```
4 3 0
```

Output:-
4 3 0