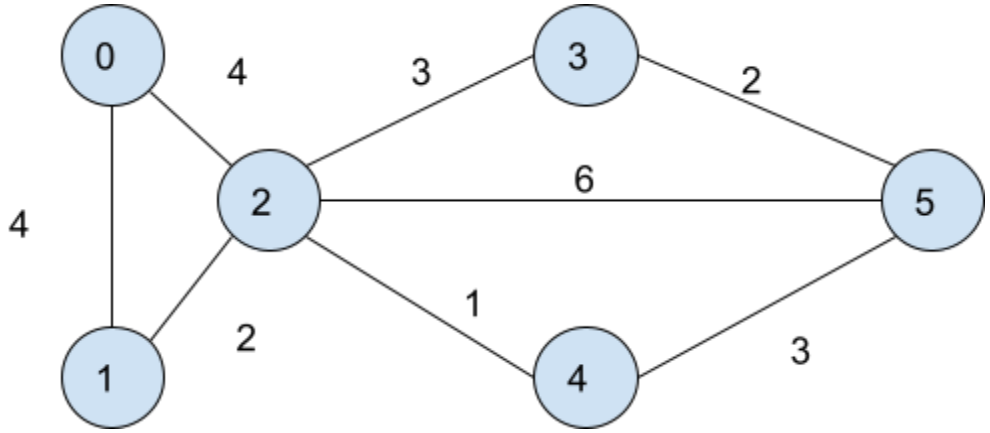
MEDIUM

Dijkstra's Algorithm | Priority Queue

Intuition



Adjacency List :
0 - [1,4],[2,4]
1 - [0,4],[2,2]
2 - [0,4],[1,2],[3,3],[4,1],[5,6]
3 - [2,3],[5,2]
4 - [2,1],[5,3]
5 - [2,6],[3,2],[4,3]

Source = 0

In order to reach 0 we need 0
0 to 1 we will take either 4 (going directly)
0 to 1 we can also take 6 (going from 2 and 1)
We have many multiple path where shortest is 4

If 0-2 had 1 in case then 2 will be shortest distance

Dijkstra works for single source shortest path

We can use :

- Queue
- Priority Queue
- Set

To implement the Dijkstra's Algorithm

We will be storing weight and the node in the adjacency list.

We will always have a min heap data structure that will store a distance and a node in it

We will also take distance array

Distance :

0	1	2	3	4	5
0	inf	inf	inf	inf	inf

Min-heap / priority queue :

[0,0] // initial configuration

We will mark all the other nodes initially infinite

0	1	2	3	4	5
0	4	4	7	5	8

[0,0]
[4,1] [4,2]
[4,2]
[7,3] [5,4] [10,5]
[7,3], [10,5] [8,5]
[10,5] [8,5]
[10,5]
[ ] // traversal ends

Now we have the distance array that stores all the shortest distances from 0 to all nodes. It can't weight for the graphs having negative weights.

Approach

- Create a min-heap (priority queue) containing pair of element and distance
- Create a distance vector and initialize all elements with inf
- Mark the source node to have a distance 0
- Push the source element into the min-heap with distance 0 as pq.push({0,source})
- Traverse until queue becomes empty :
    - Get the distance of the node at top
    - Get the node number at top
    - Pop the top element of the queue
    - Traverse for the adjacent element of the top :

        - Get the distance to reach the adjacent element from node
        - Get the number of the adjacent element
        - Check if distance to reach node + current distance from pq is smaller than the distance of the adjacent node :
            - Update the distance in the distance vector
            - Push the adjacent element to the priority queue with updated distance as ({distance, adjacent_element})
- Return distance vector

Function Code

```
vector <int> dijkstra(int n, vector<vector<int>> adj[], int source)
    {
        // we implement a min-heap using priority queue


priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>>
pq;

        // Creating a distance vector
        vector<int> distance(n,1e9);

        // marking source as distance 0 and inserting it into the initial
queue
        distance[source] = 0;
        pq.push({0,source});
        // traversing until the queue becomes empty
        while(!pq.empty())
        {
            // extracting the element with the maximum priority ie. with
minimum distance
            int dis = pq.top().first;
            int node = pq.top().second;
            // popping it from the queue
            pq.pop();
            // traversing for the adjacent elements
            for(auto it:adj[node])
            {
                // getting the weight to reach adjacent element
                int edgew = it[1];
                // getting which element are we going to reach
                int adjn = it[0];
                // checking if distance until now + distance to reach node
```

```
better than current distance
                if(dis+edgew < distance[adjn])
                {
                    // updating distance and inserting node with new
distance into the queue
                    distance[adjn] = dis+edgew;
                    pq.push({distance[adjn],adjn});
                }
            }
        }

        // returning the distance vector
        return distance;


    }
```

Time Complexity

E*log(V)