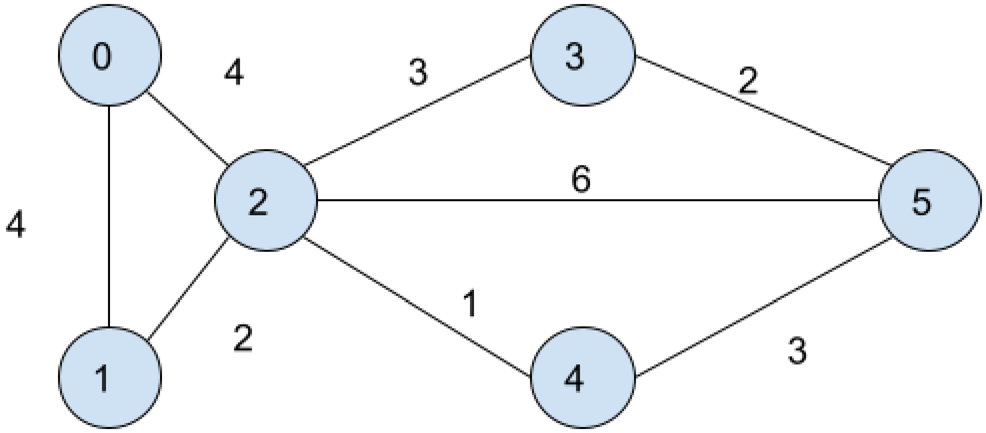
MEDIUM

Dijkstra’s Algorithm | Set

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.

Set : it stores unique value and smallest at the top ie. in ascending order

Distance :

0	inf	inf	inf	inf	inf

Set :

[0,0] // initial configuration

Distance :

0	4	4	7	5	8

Set :

[0,0]

[4,1]
[4,2]

[4,2]

[5,4]
[7,3]
[10,5]

[7,3]
[10,5] // replace it with [8,5]

[8,5]

[ ]

Approach

- Create a set of pairs containing distance, source
- Create a distance vector initialize all with inf value
- Mark the source node to have distance 0 and insert it into set as 0,source
- Traverse until the set becomes empty :

- Get the top most element of the set
- Get the weight of the topmost element till now
- Traverse for the adjacent element of the node :
    - Get the weight of the adjacent element
    - Get the adjacent element node
    - Check if the distance to reach the adjacent element + distance till now is smaller than distance of adjacent node :
        - Check if already present in the set :
            - Remove the element from set
        - Update the distance
        - Insert adjacent element with new distance into the set
- Return the result vector

Function Code

```
vector <int> dijkstra(int n, vector<vector<int>> adj[], int source)
    {
        // Creating a set containing pair of distance and source
        set<pair<int, int>> s;
        // Creating a distance vector
        vector<int> distance(n,1e9);
        // inserting the source element into the set and marking its
distance as 0
        s.insert({0,source});
        distance[source] = 0;

        // traversing until the set becomes empty
        while(!s.empty())
        {
            // extracting the first element of the set
            auto it = *(s.begin());
            int node = it.second;
            int dist = it.first;

            // erasing from set
            s.erase(it);

            // traversing for adjacent elements
            for(auto i : adj[node])
            {
                int adjn = i[0];
                int edgew = i[1];
                if(dist+edgew < distance[adjn])
                {
                    // erase if already exists
                    if(distance[adjn]!=1e9)
                    {
                        s.erase({distance[adjn],adjn});
                    }
                    // update the distance
                    distance[adjn] = dist+edgew;
                    // insert into the set
                    s.insert({distance[adjn],adjn});
                }
            }
        }
        // return the distance vector
        return distance;
    }
```

Time Complexity

E*log(V)