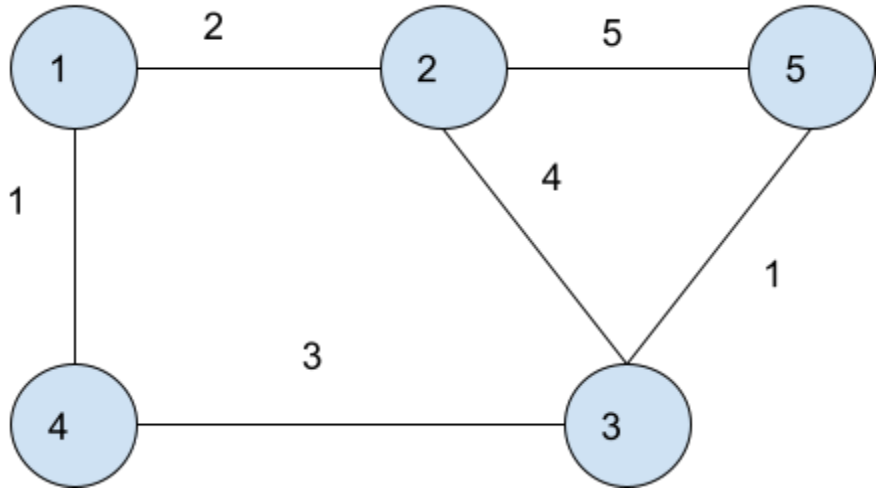
MEDIUM

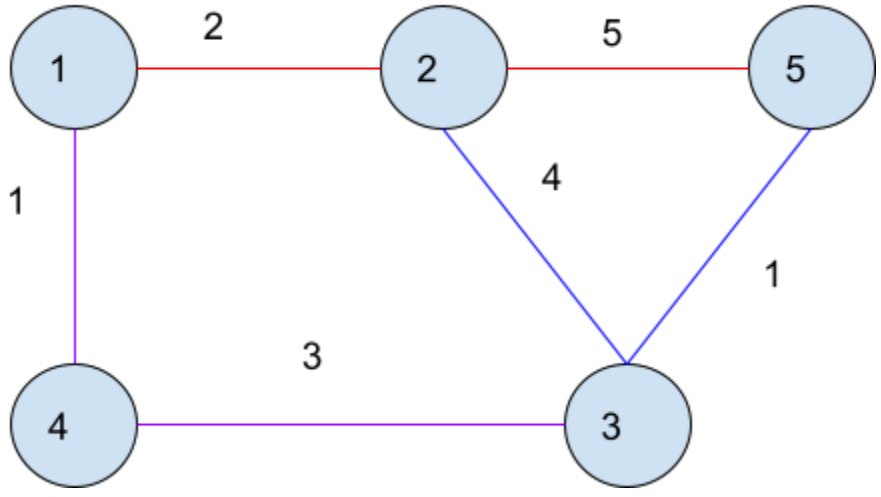
Shortest Path in weighted undirected graph

Intuition



Print the shortest path between the source node which is always 1 and destination is always n (eg. 5)

Possible Paths to reach 5 from 1



Shortest Path is to get to 4 then to 3 and then to 5

Dijkstra tells us the shortest path to 5, but we will do a slight modification to get the path

We will try to remember the path and then go to the destination

We will take three arrays // initial configuration

Parent array :

1	2	3	4	5

Distance array :

0	inf	inf	inf	inf

Priority Queue :
[0,1]

Parent array :

1	1	4	1	3

Distance array :

0	2	4	1	5

Priority Queue :

[0,1]

[2,2]
[1,4]

[2,2]
[4,3]

[4,3]
[7,5]

[5,5]
[7,5]

[7,5]

[ ]

Now parent array looks like :

Parent array :

1	1	4	1	3

5 <- 3 <- 4 <- 1

1 -> 4 -> 3 -> 5 : distance = 5

Approach

- Create an adjacency list
- Create a min-heap ordered priority queue
- Create a distance vector having 1e9 for all elements
- Creating a parent vector initially all elements are set as their own parents
- Making the distance of the source element as 0
- Inserting the source with distance 0 into the priority queue as {0,source}
- Traverse until the queue becomes empty :
    - Extract the first element of the queue
    - Pop the first element of the queue
    - Traverse for the adjacent elements :
        - Get the adjacent element reaching distance from the node
        - Get the adjacent node element
        - Check if distance till now + distance to reach node smaller than current node distance :
            - Update the distance
            - Insert the updated distance with adjacent node into the priority queue
            - Make the parent of the adjacent element as node
- Check if the distance of the destination node 1e9 :
    - Return {-1} because in this case the node is unreachable
- Create a path vector
- Traverse until the parent node element reaches to the source node :
    - Insert the node element to the path vector
    - Replace node with its parent element
- Add the source node element to the path vector
- Reverse the path vector
- Return the path vector

Function Code

```
vector<int> shortestPath(int n, int m, vector<vector<int>>& edges) {
    vector<pair<int, int>> adj[n + 1];
        for (auto it : edges)
        {
            adj[it[0]].push_back({it[1], it[2]});
            adj[it[1]].push_back({it[0], it[2]});
        }
        // Create a priority queue for storing the nodes along with
distances
        // in the form of a pair { dist, node }.
        priority_queue<pair<int, int>, vector<pair<int, int>>,
greater<pair<int,int>>> pq;

        // Create a dist array for storing the updated distances and a
parent array
        //for storing the nodes from where the current nodes represented by
indices of
        // the parent array came from.
        vector<int> dist(n + 1, 1e9), parent(n + 1);
        for (int i = 1; i <= n; i++)
            parent[i] = i;

        dist[1] = 0;

        // Push the source node to the queue.
        pq.push({0, 1});
        while (!pq.empty())
        {
            // Topmost element of the priority queue is with minimum
distance value.
            auto it = pq.top();
            pq.pop();
            int node = it.second;
            int dis = it.first;

            // Iterate through the adjacent nodes of the current popped
node.
            for (auto it : adj[node])
            {
                int adjNode = it.first;
                int edW = it.second;
```

```
                // Check if the previously stored distance value is
                // greater than the current computed value or not,
                // if yes then update the distance value.
                if (dis + edW < dist[adjNode])
                {
                    dist[adjNode] = dis + edW;
                    pq.push({dis + edW, adjNode});

                    // Update the parent of the adjNode to the recent
                    // node where it came from.
                    parent[adjNode] = node;
                }
            }
        }

        // If distance to a node could not be found, return an array
containing -1.
        if (dist[n] == 1e9)
            return {-1};

        // Store the final path in the 'path' array.
        vector<int> path;
        int node = n;

        // Iterate backwards from destination to source through the parent
array.
        while (parent[node] != node)
        {
            path.push_back(node);
            node = parent[node];
        }
        path.push_back(1);

        // Since the path stored is in a reverse order, we reverse the
array
        // to get the final answer and then return the array.
        reverse(path.begin(), path.end());
        return path;
}
```

Time Complexity

O(V^2)