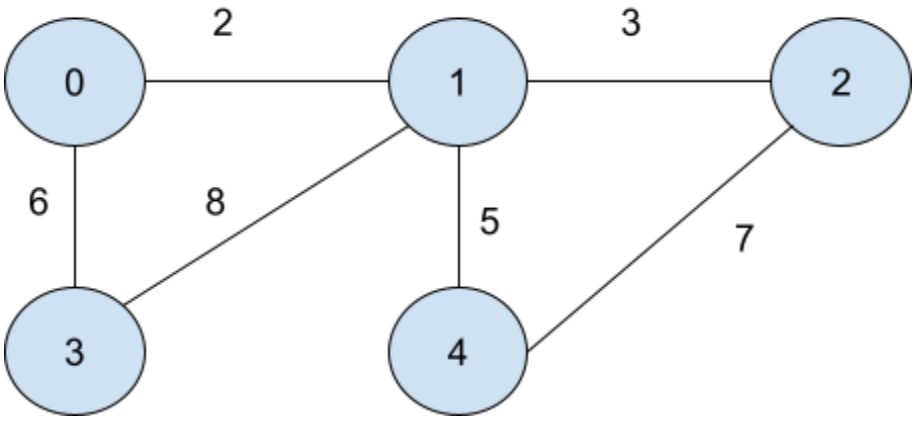
MEDIUM

Prim's Algorithm (Greedy Algorithm)

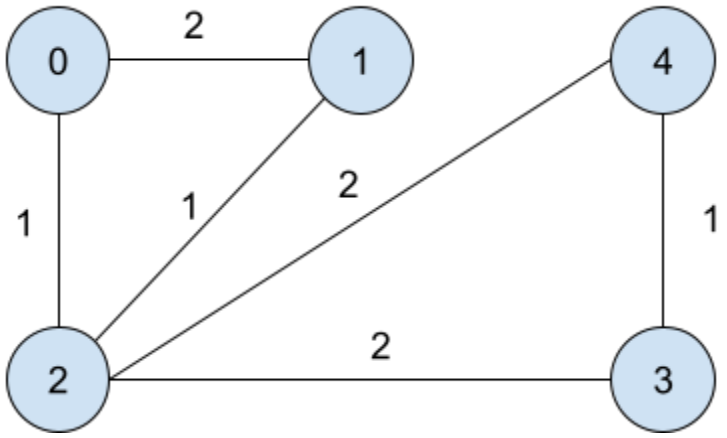
Intuition



Algorithm requires :

- Priority Queue
- Visited Vector

Eg.



Initial Configuration
Priority Queue :
[0,0,-1] // weight node parent
Visited :
0 0 0 0 0

MST :

(0,2) (1,2) (2,3) (3,4)

Visited :

1 1 1 1 1

Priority Queue :

0 0 -1

2 1 0
1 2 0

2 4 2
2 1 0
1 1 2

2 4 2
2 1 0

2 2 3
2 3 2
2 4 2
1 4 3

2 2 3
2 3 2
2 4 2

2 2 3
2 3 2

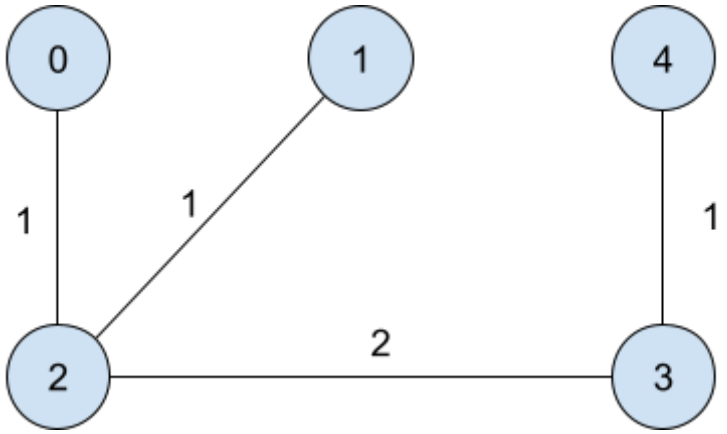
2 2 3

—----------

Now we have got our MST :

(0,2) (1,2) (2,3) (3,4)

MST :



Approach

- Create a min-heap ordered priority queue to store weight and node
- Creating a visited vector initialized with all elements as 0
- Pushing the source element with 0 weight into the queue
- Mark the source node as visited
- Initialize the sum variable with 0
- Traversing until the queue becomes empty :
    - Extracting the first element from the node
    - Popping the first element from the node
    - Checking if the node is already visited :
        - Continue as we don't need to visit it
    - Mark the node as visited
    - Add the node weight to the MST sum
    - Traverse for the adjacent nodes :
        - Check if the adjacent node is not visited :
            - Push the adjacent node to the queue with its weight
- Return MST sum

Function Code

```
int spanningTree(int n, vector<vector<int>> adj[])
    {
        // Creating a priority queue to store weight node we don't need
parent because only require to return the maximum cost
        priority_queue<pair<int, int>,vector<pair<int,
int>>,greater<pair<int, int>>>pq;
        // creating a visited vector
        vector<int> visited(n,0);
```

```
        // inserting first element into the priority queue and mark as
visited
        pq.push({0,0});
        visited[0]=1;
        // creating a sum element to store the sum of the path
        int sum = 0;
        // traversing until the queue becomes empty
        while(!pq.empty())
        {
            // extracting the first node
            auto it = pq.top();
            // popping the first element from the queue
            pq.pop();
            // getting the first element from the node
            int node = it.second;
            int wt = it.first;
            // checking if node is visited then we do not need to go to it
            if(visited[node]==1)
            {
                continue;
            }
            // Add it to the MST and mark it as visited
            visited[node] = 1;
            sum+=wt;
            for(auto it:adj[node])
            {
                int adjnode = it[0];
                int adjweight = it[1];
                if(!visited[adjnode])
                {
                    // pushing the adjacent unvisited node and its weight
into the queue
                    pq.push({adjweight,adjnode});
                }

            }
        }
        // return the sum of the weight
        return sum;
    }
```

Time Complexity

O(Elog(E))