**Minimum Spanning Tree**

**Medium**

Given a weighted, undirected and connected graph of **V** vertices and **E** edges. The task is to find the sum of weights of the edges of the Minimum Spanning Tree.

**Example 1:**

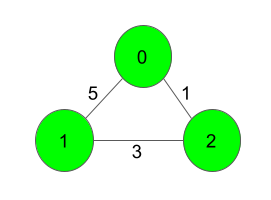
**Input:**

3 3

0 1 5

1 2 3

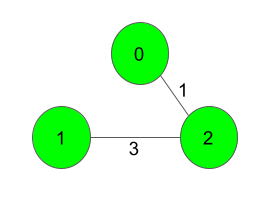
0 2 1



**Output:**

4

**Explanation**:



The Spanning Tree resulting in a weight

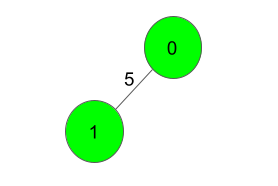
of 4 is shown above.

**Example 2:**

**Input:**

2 1

0 1 5



**Output:**

5

**Explanation**:

Only one Spanning Tree is possible

which has a weight of 5.

**Expected Time Complexity:**O(ElogV).  
**Expected Auxiliary Space:**O(V2).

**Constraints:**  
2 ≤ V ≤ 1000  
V-1 ≤ E ≤ (V\*(V-1))/2  
1 ≤ w ≤ 1000  
The graph is connected and doesn't contain self-loops & multiple edges.

**Company Tags**

[**Flipkart**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Flipkart)[**Amazon**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Amazon)[**Microsoft**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Microsoft)[**Samsung**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Samsung)[**Cisco**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Cisco)

//{ Driver Code Starts

import java.util.\*;

import java.io.\*;

import java.lang.\*;

public class Main{

static BufferedReader br;

static PrintWriter ot;

public static void main(String args[]) throws IOException {

br = new BufferedReader(new InputStreamReader(System.in));

ot = new PrintWriter(System.out);

int t = Integer.parseInt(br.readLine().trim());

while(t-- > 0){

String s[] = br.readLine().trim().split(" ");

int V = Integer.parseInt(s[0]);

int E = Integer.parseInt(s[1]);

int edges[][] = new int[E][3];

for(int i = 0; i < E; i++){

s = br.readLine().trim().split(" ");

edges[i][0] = Integer.parseInt(s[0]);

edges[i][1] = Integer.parseInt(s[1]);

edges[i][2] = Integer.parseInt(s[2]);

}

ot.println(new Solution().spanningTree(V, E, edges));

}

ot.close();

}

}

// } Driver Code Ends

// User function Template for Java

class Pair{

int node ;

int dist;

public Pair(int node, int dist){

this.node=node;

this.dist=dist;

}

}

class Solution{

static int spanningTree(int V, int E, int edges[][]){

List<List<Pair>> adj=new ArrayList<>();

for(int i=0;i<V;i++){

adj.add(new ArrayList<>());

}

for(int i=0;i<E;i++){

int u=edges[i][0];

int v=edges[i][1];

int wt=edges[i][2];

adj.get(u).add(new Pair(v, wt));

adj.get(v).add(new Pair(u, wt));

}

boolean[] visited=new boolean[V];

//Arrays.fill(visited, false);

//apna primes method.....

PriorityQueue<Pair> pq=new PriorityQueue<Pair>((x, y)->x.dist-y.dist);

int ans=0;

pq.add(new Pair(0,0));

while(!pq.isEmpty()){

Pair pair=pq.poll();

int u=pair.node;

int wt=pair.dist;

if(visited[u])

continue;

ans+=wt;

visited[u]=true;

for(Pair edge : adj.get(u)){

int nextnode=edge.node;

int weight=edge.dist;

if(!visited[nextnode]){

pq.add(new Pair(nextnode, weight));

}

}

}

return ans;

}

}