```java

class PrimsMST {
  public static class Edge{
      int src;
      int dst;
      int weight;
      Edge(int src,int dst,int weight){
          this.src = src;
          this.dst = dst;
          this.weight = weight;
  }
  }

  static int spanningTree(int V, ArrayList<ArrayList<ArrayList<Integer>>> adj) {
      // store the edges based on thier wait - minheap
      PriorityQueue<Edge> pq = new PriorityQueue<Edge>(new Comparator<Edge>(){
          public int compare(Edge e1,Edge e2){
              return e1.weight-e2.weight;
          }
      });

      boolean visited[] = new boolean[V];

      //traverse the graph with starting at vertex 0 considered as our spannig tree.
      visited[0]=true;
      addEdges(0,pq,adj,0); // to get next edge with min weight;
      int consideredEdges = 0;
      int sumOfWeights=0;

      while(consideredEdges<V-1){
          Edge edge = pq.poll();
          if(visited[edge.dst]){
              //if aready in our spanning tree skip it
              continue;
          }
          visited[edge.dst]=true;
          addEdges(edge.dst,pq,adj,edge.src);
          consideredEdges++;
          sumOfWeights+=edge.weight;
      }

      return sumOfWeights;

  }

  public static void addEdges(int vertex,PriorityQueue<Edge>
pq,ArrayList<ArrayList<ArrayList<Integer>>> adj,int src){
      // for optimising we dont add the edge from parent;
      ArrayList<ArrayList<Integer>> cl = adj.get(vertex);
      int len =  cl.size();
      for(int i=0;i<len;i++){
          int dst = cl.get(i).get(0);
          int weight = cl.get(i).get(1);
          if(dst==src){continue;}
          pq.add(new Edge(vertex,dst,weight));
      }
  }

}
```