|  |  |
| --- | --- |
| **Prim in C++** | |
| #include <bits/stdc++.h>  using namespace std;  class Solution  {  public:  //Function to find sum of weights of edges of the Minimum Spanning Tree.  int spanningTree(int V, vector<vector<int>> adj[])  {  priority\_queue<pair<int, int>,  vector<pair<int, int> >, greater<pair<int, int>>> pq;  vector<int> vis(V, 0);  // {wt, node}  pq.push({0, 0});  int sum = 0;  while (!pq.empty()) {  auto it = pq.top();  pq.pop();  int node = it.second;  int wt = it.first;  if (vis[node] == 1) continue;  // add it to the mst  vis[node] = 1;  sum += wt;  for (auto it : adj[node]) {  int adjNode = it[0];  int edW = it[1];  if (!vis[adjNode]) {  pq.push({edW, adjNode});  }  }  }  return sum;  }  };  int main() {  int V = 5;  vector<vector<int>> edges = {{0, 1, 2}, {0, 2, 1}, {1, 2, 1}, {2, 3, 2}, {3, 4, 1}, {4, 2, 2}};  vector<vector<int>> adj[V];  for (auto it : edges) {  vector<int> tmp(2);  tmp[0] = it[1];  tmp[1] = it[2];  adj[it[0]].push\_back(tmp);  tmp[0] = it[0];  tmp[1] = it[2];  adj[it[1]].push\_back(tmp);  }  Solution obj;  int sum = obj.spanningTree(V, adj);  cout << "The sum of all the edge weights: " << sum << endl;  return 0;  } | **Input:**  We have 5 vertices (V = 5) and the edges:  edges = [ {0, 1, 2}, {0, 2, 1}, {1, 2, 1}, {2, 3, 2}, {3, 4, 1}, {4, 2, 2}]  **Graph Representation (Adjacency List):**  adj[0] = {{1, 2}, {2, 1}}  adj[1] = {{0, 2}, {2, 1}}  adj[2] = {{0, 1}, {1, 1}, {3, 2}, {4, 2}}  adj[3] = {{2, 2}, {4, 1}}  adj[4] = {{3, 1}, {2, 2}}  **Prim's Algorithm Process**   1. **Initialization:**    * Use a **priority queue** pq to process edges in increasing weight order. The queue stores {weight, node}.    * Use a vis array to track visited nodes: vis = [0, 0, 0, 0, 0].    * Start with node 0: push {0, 0} to pq.   **Iteration 1:**   * **Priority Queue:** pq = {{0, 0}} * **Pop the top element:** {0, 0} → node = 0, weight = 0. * **Check if node is visited:** It's not, so mark node 0 as visited: vis = [1, 0, 0, 0, 0]. * **Add weight to sum:** sum = 0 + 0 = 0. * **Push adjacent edges to pq:**   + From adj[0] = {{1, 2}, {2, 1}}:     - Push {2, 1} (edge to node 1 with weight 2).     - Push {1, 2} (edge to node 2 with weight 1). * **Updated Priority Queue:** pq = {{1, 2}, {2, 1}}.   **Iteration 2:**   * **Priority Queue:** pq = {{1, 2}, {2, 1}} * **Pop the top element:** {1, 2} → node = 2, weight = 1. * **Check if node is visited:** It's not, so mark node 2 as visited: vis = [1, 0, 1, 0, 0]. * **Add weight to sum:** sum = 0 + 1 = 1. * **Push adjacent edges to pq:**   + From adj[2] = {{0, 1}, {1, 1}, {3, 2}, {4, 2}}:     - Skip {0, 1} (node 0 is already visited).     - Push {1, 1} (edge to node 1 with weight 1).     - Push {2, 3} (edge to node 3 with weight 2).     - Push {2, 4} (edge to node 4 with weight 2). * **Updated Priority Queue:** pq = {{1, 1}, {2, 1}, {2, 3}, {2, 4}}.   **Iteration 3:**   * **Priority Queue:** pq = {{1, 1}, {2, 1}, {2, 3}, {2, 4}} * **Pop the top element:** {1, 1} → node = 1, weight = 1. * **Check if node is visited:** It's not, so mark node 1 as visited: vis = [1, 1, 1, 0, 0]. * **Add weight to sum:** sum = 1 + 1 = 2. * **Push adjacent edges to pq:**   + From adj[1] = {{0, 2}, {2, 1}}:     - Skip {0, 2} and {2, 1} (nodes 0 and 2 are already visited). * **Updated Priority Queue:** pq = {{2, 1}, {2, 3}, {2, 4}}.   **Iteration 4:**   * **Priority Queue:** pq = {{2, 3}, {2, 4}} * **Pop the top element:** {2, 3} → node = 3, weight = 2. * **Check if node is visited:** It's not, so mark node 3 as visited: vis = [1, 1, 1, 1, 0]. * **Add weight to sum:** sum = 2 + 2 = 4. * **Push adjacent edges to pq:**   + From adj[3] = {{2, 2}, {4, 1}}:     - Skip {2, 2} (node 2 is already visited).     - Push {1, 4} (edge to node 4 with weight 1). * **Updated Priority Queue:** pq = {{1, 4}, {2, 4}}.   **Iteration 5:**   * **Priority Queue:** pq = {{1, 4}, {2, 4}} * **Pop the top element:** {1, 4} → node = 4, weight = 1. * **Check if node is visited:** It's not, so mark node 4 as visited: vis = [1, 1, 1, 1, 1]. * **Add weight to sum:** sum = 4 + 1 = 5. * **Push adjacent edges to pq:**   + From adj[4] = {{3, 1}, {2, 2}}:     - Skip {3, 1} and {2, 2} (nodes 3 and 2 are already visited). * **Updated Priority Queue:** pq = {{2, 4}}.   **Iteration 6:**   * **Priority Queue:** pq = {{2, 4}} * **Pop the top element:** {2, 4} → node = 4, weight = 2. * **Check if node is visited:** It is already visited, so skip this iteration.   **Final Output:**   * **Sum of Weights of MST:** 5. * **Visited Array:** vis = [1, 1, 1, 1, 1] (all nodes visited). |
| **Output:-**  The sum of all the edge weights: 5 | |