Here is an implementation of Prim's algorithm in Java to determine the Minimum Spanning Tree (MST) for the given graph. The implementation will include the graph data structure and the algorithm to compute the MST. The total cost of spanning the tree will be outputted.

**Graph Data Structure and Prim's Algorithm in Java**

import java.util.\*;

class Graph {

private int V; // Number of vertices

private LinkedList<Edge> adj[]; // Adjacency list

class Edge {

int v; // Destination vertex

int weight; // Edge weight

Edge(int v, int weight) {

this.v = v;

this.weight = weight;

}

}

Graph(int V) {

this.V = V;

adj = new LinkedList[V];

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

adj[i] = new LinkedList<>();

}

}

void addEdge(int u, int v, int weight) {

adj[u].add(new Edge(v, weight));

adj[v].add(new Edge(u, weight));

}

// Prim's Algorithm

void primMST() {

// Array to store constructed MST

int parent[] = new int[V];

// Key values used to pick minimum weight edge in cut

int key[] = new int[V];

// To represent set of vertices included in MST

Boolean mstSet[] = new Boolean[V];

// Initialize all keys as INFINITE

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

key[i] = Integer.MAX\_VALUE;

mstSet[i] = false;

}

// Always include first 1st vertex in MST.

key[0] = 0; // Make key 0 so that this vertex is picked as first vertex

parent[0] = -1; // First node is always root of MST

// The MST will have V vertices

for (int count = 0; count < V - 1; count++) {

// Pick the minimum key vertex from the set of vertices not yet included in MST

int u = minKey(key, mstSet);

// Add the picked vertex to the MST Set

mstSet[u] = true;

// Update key value and parent index of the adjacent vertices of the picked vertex.

// Consider only those vertices which are not yet included in MST

for (Edge edge : adj[u]) {

int v = edge.v;

int weight = edge.weight;

if (!mstSet[v] && weight < key[v]) {

parent[v] = u;

key[v] = weight;

}

}

}

// Print the constructed MST

printMST(parent);

}

// A utility function to find the vertex with the minimum key value, from the set of vertices not yet included in MST

int minKey(int key[], Boolean mstSet[]) {

// Initialize min value

int min = Integer.MAX\_VALUE, min\_index = -1;

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

if (!mstSet[v] && key[v] < min) {

min = key[v];

min\_index = v;

}

}

return min\_index;

}

// A utility function to print the constructed MST stored in parent[]

void printMST(int parent[]) {

int totalCost = 0;

System.out.println("Edge \tWeight");

for (int i = 1; i < V; i++) {

for (Edge edge : adj[i]) {

if (edge.v == parent[i]) {

System.out.println(parent[i] + " - " + i + "\t" + edge.weight);

totalCost += edge.weight;

}

}

}

System.out.println("Total cost of spanning tree: " + totalCost);

}

public static void main(String[] args) {

int V = 8; // Number of vertices in graph

Graph g = new Graph(V);

// Adding edges and their respective weights

g.addEdge(0, 1, 5);

g.addEdge(0, 3, 4);

g.addEdge(1, 2, 2);

g.addEdge(1, 3, 3);

g.addEdge(2, 3, 2);

g.addEdge(3, 4, 3);

g.addEdge(3, 5, 4);

g.addEdge(4, 5, 2);

g.addEdge(4, 7, 6);

g.addEdge(5, 6, 1);

g.addEdge(6, 7, 8);

g.addEdge(7, 8, 2);

g.primMST();

}

}

**Output**

The output of the program will be the edges included in the MST and their respective weights, followed by the total cost of the spanning tree.

**Asymptotic Analysis**

* **Time Complexity:** The time complexity of Prim's algorithm using an adjacency list and a min-heap is O(E log V), where EEE is the number of edges and V is the number of vertices.
* **Space Complexity:** The space complexity is O(V+E) due to the adjacency list representation of the graph.

**References**

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

Weiss, M. A. (2012). *Data Structures and Algorithm Analysis in Java* (3rd ed.). Pearson.