Prim's Algorithm with Time Complexity Analysis

Algorithm: Prim's Minimum Spanning Tree (MST)

Input: A connected, weighted graph represented by an adjacency matrix $\text{cost}[1 \ldots n, 1 \ldots n]$, where $\text{cost}[i, j]$ is the weight of the edge between vertices i and j or ∞ if no edge exists.
Output: A minimum spanning tree stored as a set of edges $\text{t}[1 \ldots n-1, 1 \ldots 2]$, where $\text{t}[i, 1]$ and $\text{t}[i, 2]$ represent the endpoints of the ith edge. The total cost of the MST is also returned.

Algorithm 1 Prim's Algorithm

1: Let (k, l) be the edge of minimum cost in E.
2: mincost $\leftarrow$ cost$[k, l]$
3: t$[1, 1] \leftarrow k$, t$[1, 2] \leftarrow l$
4: **for** $i \leftarrow 1$ to n **do** $\triangleright$ Initialize the array near
5:     **if** cost$[i, k] <$ cost$[i, l]$ **then**
6:         near$[i] \leftarrow k$
7:     **else**
8:         near$[i] \leftarrow l$
9:     **end if**
10: **end for**
11: near$[k] \leftarrow 0$, near$[l] \leftarrow 0$
12: **for** $i \leftarrow 2$ to $n - 1$ **do** $\triangleright$ Add $n - 2$ additional edges
13:     Let j be an index such that near$[j] \neq 0$ and cost$[j,$ near$[j]]$ is minimum.
14:     t$[i, 1] \leftarrow j$, t$[i, 2] \leftarrow$ near$[j]$
15:     mincost $\leftarrow$ mincost + cost$[j,$ near$[j]]$
16:     near$[j] \leftarrow 0$
17:     **for** $k \leftarrow 1$ to n **do** $\triangleright$ Update the array near
18:         **if** near$[k] \neq 0$ and cost$[k,$ near$[k]] >$ cost$[k, j]$ **then**
19:             near$[k] \leftarrow j$
20:         **end if**
21:     **end for**
22: **end for**
23: **return** mincost

Time Complexity Analysis

- **Initialization (lines 3–10):** Initializing the array `near` requires $O(n)$ operations.
- **Outer loop (lines 12–21):** The outer loop runs $n - 2$ times because we add $n - 2$ edges to the MST.
  - Finding the minimum edge (line 13): For each iteration of the outer loop, we check all vertices to find the one with the minimum cost, which takes $O(n)$ time.
  - Updating the array `near` (lines 17–20): For each iteration, updating `near` requires checking all vertices, which takes $O(n)$ time.

  Thus, each iteration of the outer loop requires $O(n) + O(n) = O(2n) = O(n)$ time.

Total Time Complexity:

$$\begin{aligned} O(n) &\quad \text{(initialization)} \\ +(n-2) \cdot O(n) &\quad \text{(outer loop)} \\ &= O(n + n^2 - 2n) = O(n^2) \end{aligned}$$

Therefore, the overall time complexity of Prim's algorithm using an adjacency matrix is $O(n^2)$.