Prim’s MinimumSpanning Tree Algorithm

Given a weighted, undirected graph \( G = (V, E) \), such that each edge \((u, v)\) has a non-negative weight \( w(u, v) \), and a starting vertex \( s \), find a minimum spanning tree for \( G \) starting from \( s \). A spanning tree is a tree composed of edges of \( G \) that touches every vertex in \( G \). A minimum spanning tree \( T \) minimizes the sum of its edge weights, i.e.

\[
\sum_{(u,v) \in T} w(u, v).
\]

Prim’s algorithm implements the greedy-choice strategy for minimum spanning tree. Starting with an empty tree (one with no edges), the algorithm repeatedly adds the lowest-weight edge \((u, v)\) in \( G \) such that either \( u \) or \( v \), but not both, is already connected to the tree. Put another way, it adds the lowest-weight edge that would connect a new vertex to the tree without forming a cycle. The tricky part is to decide efficiently which among a potentially large number of edges is the best edge to add to the tree at each step.

Prim’s algorithm, like Dijkstra’s, uses a priority queue \( Q \) to keep track of which edge should be added to the tree next. For each vertex \( v \) not already touched by the tree, it tracks the lowest-weight edge that would connect \( v \) to the tree. If \( Q \) is implemented with a standard binary heap, Prim’s algorithm requires time \( O(m \log n) \); if we instead use a Fibonacci heap (which has amortized constant-time insertion and decreaseKey), the time drops to \( O(n \log n + m) \).
\textbf{PRIM}(G, s)
for $u \in V$ do  \\
\hspace{1cm} $u$.distance $\leftarrow \infty$
\hspace{1cm} $u$.conn $\leftarrow$ null
\hspace{1cm} $Q$.insert($u$, $\infty$)
$T \leftarrow \emptyset$

$s$.distance $\leftarrow 0$
$Q$.decreaseKey($s$, 0)

\textbf{while} $Q$ is not empty \textbf{do}
\hspace{1cm} $u \leftarrow Q$.extractMin()
\hspace{1cm} \textbf{if} $u$.distance = $\infty$
\hspace{1cm} \hspace{1cm} \textbf{stop}  \\
\hspace{1cm} $T \leftarrow T \cup (u$.conn, $u$)
\hspace{1cm} \textbf{for} $v \in \text{Adj}[u]$ \textbf{do}
\hspace{1cm} \hspace{1cm} \textbf{if} $Q$.decreaseKey($v$, $w(u, v)$)
\hspace{1cm} \hspace{1cm} \hspace{1cm} $v$.distance $\leftarrow w(u, v)$
\hspace{1cm} \hspace{1cm} \hspace{1cm} $v$.conn $\leftarrow u$