1 Introduction

In this lecture we will look at Minimum Spanning Trees.

2 Prim’s algorithm for Minimum-Cost Spanning Tree (MST)

Say there is a set of computers in a office or a set of sites for VPN or a set of cities that we want to connect to each other. We can model all these with an undirected graph whose edges represent connections and the weights are the lengths. We want to find a connected subgraph with min sum of edge lengths. Note that the subgraph should be a tree. More formally, the problem is as follows: Given an undirected connected weighted graph \( G = (V, E) \), find a spanning tree that connects every vertex with minimum cost. We now give an algorithm due to Prim for thus problem. Note that this is a greedy algorithm.

\[ \text{Algorithm 1 Prim}(G,v) \]

\begin{enumerate}
    \item \( v_{\text{new}} = \{w\} \) and \( E_{\text{new}} = \{\} \) where \( w \) is any arbitrary vertex;
    \item \textbf{while} \( v_{\text{new}} \neq v \) \textbf{do}
        \item Find an edge \((u,v)\) where \( u \in v_{\text{new}} \) and \( v \in V \setminus v_{\text{new}} \) with min weight (break ties arbitrarily).
        \item \( v_{\text{new}} = v_{\text{new}} + \{v\} \);
        \item \( E_{\text{new}} = E_{\text{new}} + \{(u,v)\} \);
\end{enumerate}

\textbf{Time Complexity and Implementation:} The implementation is very similar (almost identical) to Djikstra (using heap). At each stage we can keep \( SE[v] \) (instead of \( SP[v] \) in Djikstra) which keeps the min edge connectivity \( v_{\text{new}} \).
to this vertex $v$ and we update it each time that we add a new vertex to $v_{new}$.

Identical to Dijkstra, the running time is $O\left(|V| + |E| \log |V|\right)$.

**Proof of Correctness:** This can be shown using induction.

<table>
<thead>
<tr>
<th>Induction Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>There is always a best solution (optimum) that has the first $k$ edges that we add to $E_{new}$</td>
</tr>
</tbody>
</table>

**Proof:** The base is trivial since there is no edge. Note that $v_{new}$ plays the same role of $v_k$ in Dijkstra. Consider the tree OPT which only the first $k$ edges of $E_{new}$. We prove there is another OPT’ with same cost which has the first $k+1$ edges of $E_{new}$. Now let $(u,v)$ be the $(k+1)$-th edge that we add to $E_{new}$ (and thus to the tree) where $u \in v_{new}$ and $v \notin v_{new}$. Now let us add $(u,v)$ to the tree OPT. There should be a cycle and there is an edge $(u',v')$ where $u' \in v_{new}$ and $v' \notin v_{new}$. Since we checked all edges going out of $v_{new}$ we have $w(u',v') \geq w(u,v)$. So we add $(u,v)$ instead of $(u',v')$, preserve connectivity, maybe lower the total weight and still have $k+1$ edges in common with new OPT’.

**References**