1 Overview

Under the DMPC model, each machine has memory of $O(\sqrt{N})$ bits, where $N = |V| + |E|$ for a graph $G = (V, E)$. We are interested in maintaining exact Minimum Spanning Trees\(^1\) (MST) in the DMPC model in $O(1)$ rounds per update using $O(\sqrt{N})$ total communication per update. As in the paper, we maintain an Euler tour tree (ET), as a sequence of vertices, for each connected component of $G$. Each vertex $v$ knows the following:

- $id(v)$: ID of the Euler tour tree $T_{id(v)}$ that $v$ belongs to
- $|T_{id(v)}|$: Size of the connected component that $v$ belongs to
- $f(v)$: The first index in $T_{id(v)}$ that $v$ appears in
- $l(v)$: The last index in $T_{id(v)}$ that $v$ appears in

We denote the set of 4 numbers $S(v) = \{id(v), |T_{id(v)}|, f(v), l(v)\}$ as the “side information” of vertex $v$. Each edge $\{u, v\}$ knows $S(u), S(v)$, and whether it is in some Euler tour tree.

1.1 Maintaining an Euler tour tree

For simplicity, we assume there is a coordinating machine (it can be any of the machines). For each vertex $v$, the set $S(v)$ is maintained when we maintain the Euler tour trees. We know that the following Euler tour tree operations can be performed under the DMPC model in $O(1)$ rounds using $O(\sqrt{N})$ total communication by broadcasting $S(\cdot)$ around.

- **Reroot**($T, v$): Returns a ET sequence with $v \in T$ as the root.
- **Cut**($u, v$): Returns 2 ETs, one containing $u \in T$ and the other containing $v \in T$.
- **Join**($u, v$): Joins 2 ETs, one containing $u$ and the other containing $v$.
- **Query**($u, v$): Returns whether $u$ and $v$ are in the same ET.
- **FindEdge**($T_i, T_j$): Returns an edge whose endpoints lie in $T_i$ and $T_j$.

$\text{FindEdge}(T_i, T_j)$ is described in the paper as follows:

- Coordinator broadcasts $T_i$ and $T_j$ to all machines.
  - Note: $T_i$ is just a number representing the ID of the $i^{th}$ Euler tour tree.
- Each machine sends an arbitrary edge whose endpoints lie in $T_i$ and $T_j$.
- The coordinator outputs an arbitrary edge amongst the received edges.

We can define a similar function $\text{FindMinEdge}(T_1, T_2)$ that returns an edge whose endpoints lie in $T_1$ and $T_2$ of the minimum cost. The only change is to require each machine to reply with the minimum cost edge instead of an arbitrary one.

\(^1\)Technically it could be disjoint forests (where each component maintains a separate Euler tour tree) but we write MST instead of MSF.
2 Algorithm

2.1 Edge insertion

Suppose edge \( e = \{a, b\} \) is inserted with weight \( w(e) \). If \( id(a) \neq id(b) \), then we connect the two Euler tour trees via \( \text{Join}(a, b) \). Otherwise, \( id(a) = id(b) \). That is, \( a \) and \( b \) are in the same connected component. Let \( T = T_{id(a)} = T_{id(b)} \) be the Euler tour tree that contains \( a \) and \( b \). We know that adding edge \( e \) into the tree \( T \) forms a cycle \( C \). So, it suffices to argue that we can efficiently find the maximum weight edge in \( C \).

To efficiently find the maximum weight edge in \( C \), the coordinator first broadcast \( S(a) \) and \( S(b) \) to all machines. In the Euler tour tree, any common ancestor \( v \) of \( a \) and \( b \) fulfill the condition:

\[
\min \{f(a), f(b)\} < l(v) < \max \{l(a), l(b)\}
\]

Without loss of generality, suppose \( f(a) < f(b) \). See Fig. 1 for an illustration.

- If \( l(a) < f(b) \), then there is some other lowest common ancestor in the ET. Consider set \( Y = Y_1 \cup Y_2 \), where
  \[
  Y_1 = \{v : f(v) < f(a) \land l(a) < l(v) < l(b)\}
  \]
  \[
  Y_2 = \{v : l(a) < f(v) < f(b) \land l(b) < l(v)\}.
  \]

- If \( l(a) > f(b) \), then \( l(a) > l(b) \) and \( a \) is an ancestor of \( b \) in the ET. Consider set \( X = \{v : f(a) < f(v) < f(b) \land l(b) < l(v) < l(a)\} \).

An edge lies in cycle \( C \) if at least one of its endpoints is in \( X \) or \( Y \). Since all edges know the \( S(\cdot) \) of their endpoints, each edge can self-identify whether it is in the cycle \( C \). Each machine then returns the maximum weight edge amongst all self-identified edges to the coordinator. Let \( e' = \{c, d\} \) be the edge with the maximum weight amongst all \( O(\sqrt{N}) \) edges received by the coordinator. If \( w(e) \geq w(e') \), then we just add the new edge to the system without updating \( T \). On the other hand, if \( w(e) < w(e') \), then the new edge \( e = \{a, b\} \) should replace \( e' = \{c, d\} \) in \( T \). To do so, we perform \( \text{Cut}(c, d) \) and \( \text{Join}(a, b) \).

![Figure 1: Identification of vertices on the cycle if we add edge \( \{a, b\} \)](image-url)
2.2 Edge deletion

Edge deletion is handled similarly to edge deletion in maintaining connected components. The only change is that we use \texttt{FindMinEdge} instead of \texttt{FindEdge}. To be precise, if edge $e = \{a, b\}$ in $T$ is to be removed, we do the following:

- $T_1, T_2 \leftarrow \text{Cut}(a, b)$
- $e' \leftarrow \text{FindMinEdge}(T_1, T_2)$
- If $e' \neq \emptyset$, say $e' = \{u, v\}$. Execute \texttt{Join}(u, v).

3 Analysis

Recall that \texttt{Cut}, \texttt{Join}, \texttt{FindEdge}, and \texttt{FindMinEdge} are operations on the Euler tour trees that run in $O(1)$ rounds using $O(\sqrt{N})$ total communication in the DMPC model. In each communication round, either an $O(1)$-sized message is broadcasted, or each machine sends at most an $O(1)$-sized message to the coordinator.

3.1 Edge insertion

Suppose edge $e = \{a, b\}$ is inserted with weight $w(e)$. If $id(a) \neq id(b)$, a single call to \texttt{Join} is made. If $id(a) = id(b)$, let $e'$ be the maximum weight edge found in $C$. If $w(e) \geq w(e')$, only simple book-keeping is done. If $w(e) < w(e')$, then a call to \texttt{Cut} and \texttt{Join} are made.

3.2 Edge deletion

A call to \texttt{Cut} and \texttt{FindMinEdge} is made. At most one call to \texttt{Join} is made.