ASSIGNMENT 2 SOLUTIONS - DISTRIBUTED ALGORITHMS

(1) Taken from pages 43-44 of 'Distributed Computing, A Locality Sensitive Approach', written by David Peleg.

(2) (a) Denote the identity mapping by Id. For any given rooted tree, (T, rt), and any given set W, define $Set_{T,W} := \{f \mid f$ is a bijection from W to itself, such that $f^2 = Id\} - Id$. For a bijection f, from W to W, define $Sum_P(f) := \sum_{w \in W} P_w(f)$. Let f_{min} satisfy: $Sum_P(f_{min}) := min\{Sum_f \mid f \in Set_{T,W}\}$.

Lemma 0.1. *For each pair of distinct vertices $w, w' \in W$, such that $w' \neq f_{min}(w)$, the paths $P_w(f_{min})$ and $P_{w'}(f_{min})$ are edge-disjoint.*

Proof.
Assume for contradiction that there exist $w1, w2 \in W$, such that $w1 \neq f_{min}(w2)$, and there exists an edge $e = (u, v)$, such that $|P_{w1}(f_{min}) \bigcap P_{w2}(f_{min})| \geq 1$. It is easy to verify that $P_{w1}(f_{min}) \bigcup P_{w2}(f_{min}) - P_{w1}(f_{min}) \bigcap P_{w2}(f_{min})$ is decomposed from the two following disjoint paths: Either from $w1$ to $w2$ and from $f(w1)$ to $f(w2)$, or from $w1$ to $f(w2)$ and from $w2$ to $f(w1)$. Without loss of generality the former decomposition is chosen. Define f'_{min} as follows:

$$f'_{min}(x) = \begin{cases} w2, & \text{x = w1} \\ w1, & \text{x = w2} \\ f_{min}(w2), & \text{x} = f_{min}(w1) \\ f_{min}(w1), & \text{x} = f_{min}(w2) \\ f_{min}(x), & \text{otherwise.} \end{cases}$$

Obviously, $f'_{min} \in Set_{T,W}$. $Sum_P(f'_{min}) = Sum_P(f_{min}) - |P_{w1}(f_{min}) \bigcap P_{w2}(f_{min})| < Sum_P(f_{min}) := min\{Sum_f \mid f \in Set_{T,W}\}$, contradiction.

□

The lemma implies that a mapping that satisfies that conditions of the question exists.

(b) We will describe a distributed algorithm that finds such a bijection. The algorithm starts with the leaves, in a manner similar to $ConvergeCast$. Each leaf $v \in W$ sends a $MATCH(v)$ message, and each leaf $v \in V \setminus W$ sends a $NOMATCH$ message to its parent.
Each vertex $v \in V$ collects the $MATCH$ and $NOMATCH$ messages from its children, and keeps a set of vertices T which is the set of vertices that have sent a $MATCH$ message. After all the children messages have arrived, v start to match vertices.
If $|T|$ is even, v chooses random arbitrary pairs of vertices $w_1, w_2 \in T$ and replies the messages $MATCHED(w_1, w_2)$ to w_1 and $MATCHED(w_2, w_1)$

to w_2. Afterwards, if $v \in W$, it sends a $MATCH(v)$ message to its parent.
If $|T|$ is odd, v chooses random arbitrary pairs of vertices $w_1, w_2 \in T$ and sends the reply messages $MATCHED(w_1, w_2)$ to w_1 and $MATCHED(w_2, w_1)$ to w_2. There will be one vertex $w \in T$ that haven't been paired, since $|T|$ is odd. If $v \in W$, it will send a $MATCHED(w, v)$ message to w and mark w as its paired vertex. Otherwise it'll send a $MATCH(w)$ message to its parent and memoize the unmatched vertex w.
When a node $v \in V$ receives a $MATCHED(w_1, w_2)$ message from its parent, it checks whether $ID(v) = ID(w_1)$. If so, it sets w_2 as its match. Otherwise it sends the $MATCHED(w_1, w_2)$ message to its unmatched child.

In the worst case, $MATCH$ messages will go from a leaf to the tree root, and the $MATCHED$ replies will go down back to the leaves. Since this communication is done concurrently, $Time(FindMatch) = Depth(T) + Depth(T) = O(Depth(T))$.
We can observe that each edges passes one $MATCH$ or $NOMATCH$ message and at most one $MATCHED$ message, hence
$Comm(FindMatch) = O(|E|) = O(n)$.

(3) Denote the Maximum Weight Spanning Tree of a graph G by $M_X ST(G)$.

Lemma 0.2. *Given the graphs $G = (V, E, \omega)$ and $G' = (V, E, -\omega)$, it holds that $MST(G) = M_X ST(G')$. In other words, the Minimum Weight Spanning Tree of a graph G is the Maximum Weight Spanning Tree of the same graph with the negative weights function.*

Proof. Assume towards contradiction that there exist some graphs $G = (V, E, \omega)$ and $G' = (V, E, -\omega)$ for which $T \equiv MST(G) \neq M_X ST(G')$; Hence there exists a spanning tree $T' = (V, E')$ for which $W_{G'}(T') > W_{G'}(T)$.
$-\sum_{e \in E'} \omega(e) = \sum_{e \in E'} -\omega(e) = W_{G'}(T') > \cdots$
$\cdots > W_{G'}(T) = \sum_{e \in E} -\omega(e) = -\sum_{e \in E} \omega(e)$, so:
$W_G(T') = \sum_{e \in E'} \omega(e) < \sum_{e \in E} \omega(e) = W_G(T)$, in contradiction to T being $MST(G)$. □

Using this lemma, we can take any existing distributed algorithm for MST construction, such as *Prim*, *GHS* or *Pipelined MST*, and let it operate on the negative weights graph $G' = (V, E, -\omega)$.

(4) Since the graph $G = (V, E, \omega)$ is a cycle, we can use the *Red Rule* and exclude the heaviest edge to create $MST(G)$.
Here is a simple distributed algorithm:
- Choose an arbitrary vertex $v_0 \in V$ as the initiator, which will also act as the tree's root.

- v_0 discovers the weight of the heaviest edge. It sends a $MAX_EDGE(x)$ message to one of its neighbours w, where $x = \omega((v_0, w))$. Each vertex v, once received a $MAX_EDGE(x)$ message from its neighbour, sends a $MAX_EDGE(x')$ message to its other neighbour, where $x' = max\{x, \omega((w, v))\}$. When v_0 receives a MAX_EDGE message from its other neighbour, it knows the weight of the heaviest edge in the graph.
- v_0 now issues a $REMOVE(x)$ message to one of its neighbours. When a vertex v receieves a $REMOVE(x)$ message from its neighbour w, it checks whether $\omega((w, v)) = x$; if so, it removes it from the resulting tree. Otherwise it sends this message to its other neighbour.

Remark: This is a schematic description. You were supposed to describe the tree construction process, in which each vertex knows its children and parent.

(5) The naive approach to perform MST on a clique is to use one of the existing MST distributed algorithms. This takes $\Omega(n \cdot logn)$ time units. Here we will take advantage of the clique's connectivity to perform an MST construction in $O(logn)$ time units.

We will use a variant of the *GHS* algorithm, that (similarly to the standard *GHS*) works in $O(logn)$ phases, each consisting of three parts:

- Finding the $MWOE(F)$ of each fragment F. This may take up to $O(Diam(F)) = O(n)$ time units on the standard *GHS*, but since each fragment F is a clique, we can send direct messages to the fragment's representative in $O(1)$ time units.
- Connecting the fragments. This is done in $O(1)$ time units and there is no need to modify it.
- Broadcasting the new identity of the fragment's representative over each fragment. This may take up to $O(Diam(F)) = O(n)$ on the standard *GHS*, but since each fragment F is a clique, we can send direct messages to each vertex $v \in F$ in $O(1)$ time units.

Hence: $Time(CliqueGHS) = O(logn) \cdot O(1) = O(logn)$.

Note that at the end of each phase, each vertex should send its fragment identity to each of its neighbours (required for the $MWOE$ finding phase). This action alone takes $O(|E|)$ messages for each phase, hence: $Comm(CliqueGHS) = O(logn) \cdot O(|E|) = O(|E|logn) = O(n^2 logn)$.