# Solution 1

## Question 1

a) In the graph $G$ there exists a circle $C = \{v, u\}$. If we remove edge $(v, u)$, then remaining graph will consist of n vertices and n-1 edges, a tree in other words. Lets remove edge $(u, v)$ as well. We achieve an acyclic graph with n-2 edges. Therefore, the graph contains exactly two connected components: one that includes u, and one that includes v.

Next, we need to prove that each component is a rooted tree, with roots u and v. (Separately for each component.)

This can be done by induction on the level: Let's start with the component of u:

Base: all the neighbors of u are oriented towards u since its out-degree is 0 in the component.
Step: Each vertex w in level i has a neighbor z in level i -1. Therefore, by the induction hypothesis z is oriented towards the root (i.e, towards a vertex in level i-2, that is different from w.) Since the out-degree of z is 1, the edge (z,w) cannot be oriented towards w. (Otherwise, the out-degree of z would be at least 2.) Therefore,$(w, z)$ is oriented towards z in level i-1, or in other words, towards the root.

This completes the induction proof. The proof for the other component is similar.

b) A path with n vertices and unique ascending edges.
$G = (V, E); V = \{v_1, v_2, \dots, v_n\};\ E = \{(v_i, v_{i+1})\}; w(v_i, v_{i+1}) = i$
Phase 0 - n fragments, each vertex is a fragment
Phase 1 - a huge fragment made of all vertices from previous phase

c) The runtime of GHS will be at least $\log(n)$, if fragment of each consecutive phase will include exactly two fragments from previous phase.

Such graph can be achieved, for example, as follows. Start with an unweighted n-vertex cycle $\{v_1, v_2, \dots, v_n\}$. Select an edge $e_1 = (v_1, v_2)$, and assign to it the weight 1. Next, skip the edge $e_2 = (v_2, v_3)$, and assign the edge $e_3 = (v_3, v_4)$ the weight 2. Continue in this way untill all the edges with odd index $i$ are assigned the weight $(i + 1)/2$.

Next, go over the cycle again. Each other time you encounter an unweighted edge, assign it the last assigned value plus one. (After the second pass on the cycle, it has 1/4n unweighted edges, where each two unweighted edges are separated by at least 3 weighted edges.) Next perform a third pass on the cycle with assigning the weights 3/4n +1,...,7/8n, to each other unweighted edge. Continue with passes 4,5,...,log n, in the same way, until all edges are assigned a weight

d) The hypothesis is wrong. Lets build a clique with diameter 1, but GHS will have a runtime of $\Omega(n)$.

Assume a clique of n vertices with a lightweight path inside, just like in paragraph b, but this time the path consist of $\frac{n}{2}$ vertices. In addition, assume that rest of the edges have very big weight.

In the first phase a huge fragment of $\frac{n}{2}$ vertices will be created, it will include all the vertices of the lightweight path. The creation of the fragment will take $\Omega(n)$ time. In addition, each CT for MWOE discovery will take $\Omega(n)$ time as well.

## Question 2

We will use a variant of GHS, that (similarly to the standard GHS) works in $O(\log n)$ phases:

- Since rt is connected to each vertex, we can use it as a super representative of all fragments. All messages to rt will include also an id of fragment.
- MWOE lookup will take $O(1)$ time. Every fragment will send their edges to rt in $O(1)$ time, that will calculate the MWOE for each fragment locally.
- Connecting of fragments is done in $O(1)$ time also.
- Broadcasting the identity of the fragment's representative over each fragment is done in $O(1)$ time. In standard GHS it may take up to $O(\mathrm{Diam}(F)) = O(n)$ time, but since rt is connected to each vertex, we can send direct messages to each vertex

v ∈ F in O(1) time units.

Hence : Time (GHS') = O(logn) ∗ O(1) = O(logn)
Comm(GHS') = O(logn) ∗ O(E)

## Question 3

(a) Denote the identity mapping by Id. For any given rooted tree, $(T, rt)$, and any given set $W$, define $Set_{T,W} := \{f \mid f \text{ is a bijection from } W \text{ to itself, such that } f^2 = Id\} - Id$. 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.

□

(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 $Comm(FindMatch) = O(|E|) = O(n)$.