Distributed computing

Question 2:

Each IP can be transformed to a number that can be compare so each two different ids either IP1 > IP2 or IP2 > IP1, also assume they can be transformed into natural number.

Then we will add wight to each edge to be the minimum of the two IPS, so for any edge u,v with the IPS, u\_ip and v\_ip (wolg) if u\_ip < v\_ip then E(u,v)=u\_ip + 1/port\_number .

The reason we add the 1/port number is in order to create different and unique weights.

Now we run the GHS algorithm that will take O(n \* log n)

After that we will run the following function starting from the root.

* Start from the root and start going down the path

Def split\_into\_circle(node, end=false, root\_ip)

* If node.children=1:
* - split\_into\_circle(node[0], end, root\_ip)
* If node.children=0 and end:
* - node.next=root
* For child, index in node.children:
* - if index == 0:
* - - split\_into\_circle(node[0], false, root\_ip)
* - - continue
* - # this will set only to the last child to be a child node
* - split\_into\_circle(node[index],index == len(node.children) , node[index - 1].ip)

This function will move the pointer in each step to create a cycle and at the end will create a cycle.

This will take O(n) rounds at max at the end of the GHS algorithm.

So the time complexity is still O(n)

b) addition: assume a node x is join the network, then the node x send a message to it lowest ip address (or they send to him and he choose the lowest ip address).

And he send a join request to the node.

Then the node send it child to follow the new x node as a parent and the node changes his child to be x.

Deletion: when a node is deleted that mean that there is a node with no next [x], and node with no back [y] (assume that when a node gets disconnect then the other nodes “knows” about that). Then x start move the message backward, and y send the message forward until they meet in O(n/2)=O(n), and then the node that get the two messages at the same time send a message to x that his next his y and to y that his back his x.

Question 2

Each IP address can be transformed into a numerical value that allows comparison, meaning that for any two different IPs, either **IP1 > IP2** or **IP2 > IP1**. Additionally, assume that IPs can be mapped to natural numbers.

To construct the graph, we assign a weight to each edge as the **minimum of the two IP addresses**. Specifically, for any edge **(u, v)** with IPs **u\_ip** and **v\_ip** (without loss of generality, assume **u\_ip < v\_ip**), we define the edge weight as:

E(u,v)=u\_ip+1/port\_number ​

The purpose of adding **1/port number​** is to ensure that each edge weight remains **unique and differentiable**.

Next, we run the **GHS (Gallager-Humblet-Spira) algorithm**, which constructs a **minimum spanning tree (MST)**. The time complexity of this step is: O(n \* log n)

Once the MST is formed, we execute the following function, starting from the root, to transform the tree into a **Hamiltonian cycle**:

**Algorithm: Transforming MST into a Hamiltonian Cycle**

python

CopyEdit

def split\_into\_circle(node, end=False, root\_ip):

if len(node.children) == 1:

split\_into\_circle(node.children[0], end, root\_ip)

if len(node.children) == 0 and end:

node.next = root\_ip # Connect the last node back to the root

for index, child in enumerate(node.children):

if index == 0:

split\_into\_circle(child, False, root\_ip)

continue

# Set the last child as the connecting node

split\_into\_circle(child, index == len(node.children) - 1, node.children[index - 1].ip)

This function:

* Traverses the MST.
* Adjusts the node pointers in each step to form a **cycle**.
* Ensures that the final result is a **Hamiltonian cycle**.

The function runs in **O(n) rounds** at most, following the completion of the GHS algorithm. Since GHS runs in **O(n log n)** and the transformation runs in **O(n)**, the overall time complexity remains: O(nlogn)

**(b) Handling Node Addition and Deletion**

**Node Addition:**

1. Suppose a new node **x** joins the network.
2. Node **x** sends a message to the **lowest IP address** it can reach (or the existing nodes send their IPs to **x**, and **x** selects the lowest one).
3. **x** sends a **join request** to the selected node.
4. The selected node:
   * Reassigns one of its children to follow **x** as a parent.
   * Updates its child pointer to **x**.

Time O(1) messges O(1)

**Node Deletion:**

1. If a node is deleted, it creates a **gap** in the cycle.
2. This results in:
   * A node **x** with no successor (next pointer missing).
   * A node **y** with no predecessor (back pointer missing).
   * (Assume nodes detect when a neighbor disconnects.)
3. **Recovery Process:**
   * **x** starts sending a **backward message**.
   * **y** starts sending a **forward message**.
   * The messages propagate until **x and y meet** (this takes **O(n/2) = O(n)** rounds).
   * The meeting node updates both **x** and **y**:
     + **x’s next** is set to **y**.
     + **y’s previous** is set to **x**.

**time and messsage complexity : O(n)**

**c)**

### **Final Answer for c**

* **For node addition:** **c = O(1)** (constant time).
* **For node deletion:** **c = O(n)** (in the worst case, messages traverse half the cycle).

This means that our algorithm **self-stabilizes in O(n) rounds in the worst case** but can be optimized further if nodes store additional pointers or use faster lookup mechanisms.

Question 3:

Because the graph is connected then each node have one or more neighbors, meaning that in the first round each node at least one node. We assume that the network stays connected, so each round we either discover that neighbors at distance of r, or we get connected to node that give us more information about his k-1 neighbors (at least one new one). That mean that we will not stop running until we finish to discover information (cause at each round we get either one or more information from node) on all the nodes.