Tutor: Julien Dallot

Solution Sheet 3

Tree Algorithms

Exercise 1 (Lightest Edges)**.**

1. Clearly, the execution of this algorithm cannot take more than n rounds. Let the $(n-1)$ lightest edges form two stars of the same size and the n^{th} lightest edge connect the two centers of the stars. We are not interested in the distribution of the other weights. In this scenario it takes $\lceil n/2 \rceil$ rounds until the two center nodes announce the edge between the two centers of the stars. Since it is necessary to know this edge, the algorithm cannot terminate earlier and the time complexity of this algorithm is $\Omega(n)$.

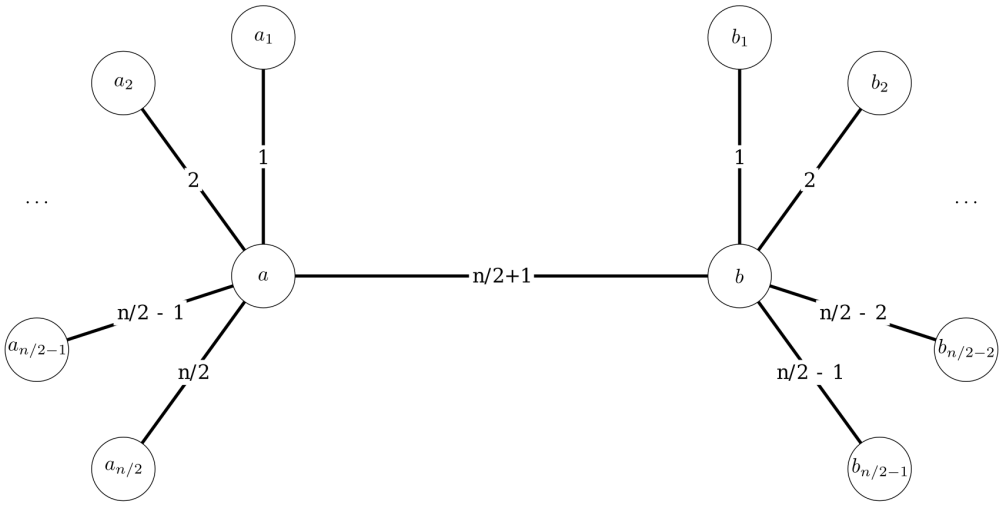

Figure 1: The two centers of the star are nodes a an b. The number in the middle of the edge is the weight of the edge. All the other non-represented edges have weight greater than $n/2$, say n.

2. We first prove that the time complexity is upper bounded by $\lceil \sqrt{2n} \rceil \in O(\sqrt{n})$. After $\lceil \sqrt{2n} \rceil$ rounds, all nodes with at most $\sqrt{2n}$ edges among the n lightest edges have broadcast all relevant edges known to them. That means, after $\lceil \sqrt{2n} \rceil$ rounds, there can only be missing edges between nodes that initially had at least $\sqrt{2n}+1$ lightest edges leading to nodes that are also incident to at least $\sqrt{2n}$ lightest edges. Assume there is such a node. Since each edge connects two nodes, initially we must have had at least $(\sqrt{2n}+1)^2/2 > n$ lightest edges, a contradiction.

   We now construct a worst-case graph example G with n nodes. We weight the edges of G as follows. Take a subset of $\sqrt{2n}$ nodes in G, we call the *subclique* the subgraph induced by those nodes (that is, the $\sqrt{2n}$ nodes and all the edges between them). Weight the edges of G such that the optimal solution is to take all the edges of the subclique.

Let c be the maximum number of node identifiers that can travel through one edge during one round — c exists since we are in the congest model. We now prove the following claim: let $u < v \leq \sqrt{n}$, at the u-th round no edge with weight v have been added into the solution.

3. Node v can send the n^{th} smallest edge weight to all nodes. Every node v_i can now determine how many among its edges (v_i, v_j), where $i < j$, belong to the n lightest edges and send this value N_i to all nodes. Now, the nodes know to which node they have to send their edge weights such that they can be distributed in the next round without contention: Node v_i sends its smallest weight to the node v_k, where $k = 1 + \sum_{j=1}^{i-1} N_j$, the next one to v_{k+1}, etc. Thus, every node receives exactly one edge weight to forward to all nodes. This procedure takes four rounds, i.e., the time complexity is $O(1)$.

Exercise 2 (License to Match)**.** 1. We use a variant of the Echo algorithm (Algorithm 12). A node (i.e. an agent in the hierarchy) matches up all (except for at most one) of its children. If one participating child remains and the node itself also participates, it matches itself with that child. If either the node or a one of its children remain, then the node sends a request to "match" upwards in the hierarchy. Otherwise, it sends a "no match" and that subtree is done. We give an asynchronous, uniform matching algorithm below.

Algorithm 1 Edge-Disjoint Matching

```
1: wait until received message from all children
2: while at least 2 requests remain (including myself) do
3:     match any two requests
4: if exists leftover request then
5:     send "match" to parent (= superior)
6: else
7:     send "no match" to parent
```

When a node v sends a "match" request to its parent u, then the edge $\{u, v\}$ will be used only once since there will be only one request in the subtree rooted at v. Along with the messages of the algorithm, the required path information is sent; we left this out in the pseudocode to improve readability.

2. Let T be the tree with n nodes. Assuming each message takes at most 1 time unit, then the time complexity of Algorithm 1 is in $O(\mathit{depth}(T))$ since all the requests travel to the root (and back down if we inform the agents of their assigned partners). On each link, there are at most 2 messages: 1 that informs the parent whether a match is needed and optionally 1 more to be informed by the parent of the match partner. So there are a total of at most $2(n-1)$ messages.

Exercise 3 (License to Distribute)**.** 1. Again we apply an echo-style algorithm where a node locally balances the documents as much as possible. For each node v we can define

$$\mathit{balance}(v) := \mathit{have}(v) - \mathit{need}(v)$$

where $\mathit{have}(v)$ is the total number of documents and $\mathit{need}(v)$ is the number of nodes in the subtree rooted at v. Each node then computes and sends this balance to its parent. Again the algorithm is asynchronous and uniform. Abstractly we refer to a document as a token. When we gather the balance information from the children, they also send along any extra tokens they might have.

2. The time complexity is in $O(\mathit{depth}(T))$ analogous to Exercise 2. Again, there is one message upwards for each link and optionally one downwards with the missing tokens. Thus there are at most $2(n-1)$ messages.

Algorithm 2 Token Distribution for node v

```
1: wait until received balance from all children
2: balance(v) := 0
3: for each child c do
4:     balance(v) := balance(v) + balance(c)
5: balance(v) := balance(v) + tokens(v) - 1
6: send up balance(v)
7: if balance(v) < 0 then
8:     wait to receive needed tokens from parent
9: redistribute tokens among children
```