FS 2023 Prof. R. Wattenhofer

Principles of Distributed Computing
Exercise 12: Sample Solution

1 Flow labeling schemes

Question 1 Check that R_k is reflexive, symmetric and transitive.

- reflexive: $\text{flow}(x, x) = \infty$
- symmetric: the graph is undirected, $\text{flow}(x, y) = \text{flow}(y, x)$
- transitive: consider a path $p = (v_1, v_2, \ldots, v_{m_p})$ from x to y in which $v_1 = x$ and $v_{m_p} = y$ and a path $p' = (v'_1, v'_2, \ldots, v'_{m_{p'}})$ from y to z in which $v'_1 = y$ and $v'_{m_{p'}} = z$. Let i be the largest subscript in p' such that $v'_i \in p$. It is easy to check there is a path $x - -v'_i - -z$ where $x - -v'_i$ is a part of p and $v'_i - -z$ is a part of p'.

C_{k+1} is a refinement of C_k.

Question 2

a) Add the depth of each vertex into the label. The depth of the tree is smaller than m, so the added part is of size $O(\log m)$. From the depth of two vertices and the distance between them, SepLevel can be computed.

b) Note that

$$\text{flow}_G(v, w) = \text{SepLevel}_T(t(v), t(w)). \tag{1}$$

The depth of T_G cannot exceed $n\hat{\omega}$ and every level at most has n nodes, hence the total number of nodes in T_G is $O(n^2\hat{\omega})$.

Question 3 Cancel all nodes of degree 2 in T_G, and add appropriate edge weights ($\tilde{T}_G$).

Now, define $\text{SepLevel}_T(x, y)$ as the weighted depth of $z = lca(x, y)$, i.e. its weighted distance from the root. Obtain the SepLevel labeling scheme for weighted trees in the same way as in question 2. For $\tilde{n}$-node trees with maximum weight $\tilde{\omega}$, the labeling size is $O(\log \tilde{n} \log \tilde{\omega} + \log^2 \tilde{n}) + O(\log(\tilde{n}\tilde{\omega})) = O(\log \tilde{n} \log \tilde{\omega} + \log^2 \tilde{n})$.

Again, for two nodes x, y in G, the weighted separation level of the leaves $t(x)$ and $t(y)$ associated with x and y in the tree $\tilde{T}_G$ is related to the flow between the two vertices as in Eq. (1).

Finally, note that as $\tilde{T}_G$ has exactly n leaves, and every non-leaf node in it has at least two children, the total number of nodes in $\tilde{T}_G$ is $\tilde{n} \leq 2n - 1$. The maximum edge weight in $\tilde{T}_G$ is $\tilde{\omega} \leq n\hat{\omega}$. We end up with the label size of $O(\log \tilde{n} \log \tilde{\omega} + \log^2 \tilde{n})$.

For more details, see [1] (Section 2).

2 Labeling Games

Question 1 Alice can encode the whole neighborhood of each vertex in the label. There are at most 1000 vertices and the ID of the current vertex v is also given. She can encode the i-th bit of l_v as 1 if the node with ID i is connected to v and 0 otherwise. Bob can then execute a graph traversal algorithm of his choosing to visit each node. Furthermore, they win all of the 2000 gummybears!

Question 2 Let T be a star graph with center r and π be any ordering of the vertices of T without r. Note that with 20 bits we can encode 2 numbers up to 1023 using 10 bits each. Alice can use the following scheme: For each vertex v she encodes the ID of r as the first number. As the second number she encodes the ID of vertex u following v in π. At vertex r she encodes the ID of the first vertex in π.

Assume Bob starts at r. If not, his first move will be to travel to r (ID of r is saved at every vertex). Then Bob can traverse all vertices by following the ordering of π. At r the first vertex x of π is given and he can take the direct edge to it. At x the next vertex y of π is encoded. He can visit y by going back to r and then taking the edge to y. He repeats this procedure until all vertices have been visited. Afterwards, he can share all 2000 gummybears with Alice!

Question 3 Let T be any graph with at most 1000 vertices. We can use a similar idea as in Question 2, but have to be a bit more careful with traversing through the graph. Pick any arbitrary vertex r and root the tree at r. Furthermore, let π be a preorder tree traversal of the vertices. Recall that we can encode 2 node IDs using 20 bits. At every vertex v we encode the parent of v as the first ID. The second ID will be the ID of vertex u that is after v in π. Therefore, we can decompose $l_v = (parent(v), next(v, \pi))$.

Assume Bob starts at r. If not, his first few moves will be to travel to r. Note that Bob can recognize if he is located at r as he gets id_v and l_v upon visiting a node v. If he is located at the root, id_v will match the first ID encoded in l_v. To get to the root, he always goes to $parent(v)$, the edge towards the parent of the node he currently resides at. Then he can start visiting the nodes (roughly) in the order of π. Upon arriving at a vertex v, he will try to visit $next(v, \pi)$. This will succeed, unless v is a leaf. If v is a leaf, then our request fails and we loose one gummybear. However, because π was constructed as a preorder tree traversal, we now that $next(v, \pi)$ must be connected to one of the ancestors a of v. Furthermore, the whole subtree of a containing v has already been visited. Therefore, we can go back to $parent(v)$ and try to visit $next(v, \pi)$ there. We have to repeat this procedure until we reach a, loosing a gummybear for each failed request until we reach a.

Following these rules, Bob looses one gummybear at every vertex except the root (there is no other ancestor a) and the very last vertex (because he wins the game). Therefore, Alice and Bob can win at least 1002 gummybears!

Question 4 The solution is almost the same as in Question 2. However, we have to be a bit more clever in the beginning. We again root the tree at r and get a preorder traversal π. Instead of assigning the label $l_v = (parent(v), next(v, \pi))$ we assign the bitwise XOR of both IDs and get the label $l_v = parent(v) \oplus next(v, \pi)$. As the starting set S we choose r and the first node of π. Assume Bob starts at r. In this case he can execute the same steps mentioned in Question 3. To get the label $next(v, \pi)$ he just has to xor l_v with $parent(v)$ (which is known because he visits the tree in the same preorder traversal) to get $next(v, \pi)$. Now assume we start in $x \neq r$. We know that we start in one of r's children. We can try all possible IDs from 1 to 1000 as the possible value of the ID of r. It could happen that we end up in $y = next(x, \pi)$ instead of r. However, to distinguish between the two cases, we can xor y with id_x. If we are in the root, then id_y will be equal to $y \oplus id_x = id_r \oplus next(r, \pi) \oplus id_x = id_r$. Otherwise, we can go back to x and go directly to the root r by computing $id_r = l_x \oplus y = parent(x)$. Note that we can have at most 998 wrong requests before getting a valid transition to another node. In the end we are left with at least 4 gummybears to share between Alice and Bob.

References

[1] Katz, Michal, et al., *Labeling schemes for flow and connectivity*, SIAM Journal on Computing 34.1 (2004): 23-40.