Principles of Distributed Computing	05/23, 2018
Exercise 13	
Lecturer: Mohsen Ghaffari	

1 Random Edge Identifiers

Consider an n-node graph $G = (V, E)$ and suppose that for each edge $e \in E$, we define an $10 \log n$-bit identifier I_e for e by picking each bit at random.

Exercises

(1a) Prove that with high probability, these are unique edge-identifiers. That is, with probability at least $1 - 1/n$, for each two edges $e, e' \in E$ such that $e \neq e'$, we have $I_e \neq I_{e'}$.

Let $e, e' \in E$ and let $\mathcal{E}_{e,e'}$ be the (bad) event where e and e' are assigned the same identifier. By taking the union bound over $\bigcup_{e,e' \in E} \mathcal{E}_{e,e'}$, we get that an upper bound of $n^4 \cdot 2^{-10 \log n} \leq n^{-6}$ on the probability that any two edges have the same identifier.

(1b) Consider a set $E' \subset E$ of edges with $|E'| \geq 2$. Prove that with probability at least $1 - 1/n$, there is no edge $e \in E$ such that $\oplus_{e' \in E'} I_{e'} = I_e$. That is, with high probability, the bitwise XOR of the identifiers of any non-singleton edge-set is distinguishable from each edge identifier.

In this exercise, a little care has to be taken, since the probability of $\oplus_{e' \in E'} I_{e'} = I_e$ is not (necessarily) independent of I_e in case $e \in E'$. However, it can be the case that $\oplus_{e' \in E'} I_{e'} = I_e$, only if $\oplus_{e' \in E' \setminus \{e\}} I_{e'} = \underline{\mathbf{0}}$ or $\oplus_{e' \in E' \setminus \{e\}} I_{e'} = I_e$, where $\underline{\mathbf{0}}$ stands for the zero vector. Notice that if $e \in E'$, then $\oplus_{e' \in E'} I_{e'} = \underline{\mathbf{0}} \oplus I_e = I_e$. Now, let $\mathcal{E}_{e,E'}$ be the (bad) event that $\oplus_{e' \in E'} I_{e'} = I_e$ if $e \notin E'$ and $\mathcal{E}^*_{e,E'}$ the (bad) even that $\oplus_{e' \in E' \setminus \{e\}} I_{e'} = \underline{\mathbf{0}}$ for the case of $e \in E'$. Similarly to $(1a)$, we can use union bound over E' and all edges to get an upper bound of $2n^2 \cdot 2^{-10 \log n} \leq n^{-7}$.

2 Graph Sketching for Connectivity

Consider an arbitrary n-node graph $G = (V, E)$, where each node in V knows its own edges. Moreover, we assume that the nodes in V have access to a desirably long string *shared randomness*. Each node should send a packet with size B-bits to the referee, who does not know the graph, so that the referee can determine whether the graph G is connected or not, with high probability. In the class, we saw an algorithm which solves this problem with packet size $B = O(\log^4 n)$. We now improve the bound to $B = O(\log^3 n)$.

Exercises

(2a) Suppose that for each phase of Boruvka's algorithm, instead of having $O(\log n)$ sketches for each node — where each sketch is made of $O(\log^2 n)$ bits, as described in the class — we have just one sketch per node. Show that still, for each connected component, we can get one outgoing edge with probability at least $1/40$. The proof follows closely the steps in Lemma 1 in the lecture notes. Consider some connected component A, let $B = V \setminus A$ and let k be the number of edges between A and B. For some estimate $\tilde{k}$ it holds that $\tilde{k}/2 \leq k \leq 2\tilde{k}$. It is known that $1 - x \geq 4^{-x}$, when $0 \leq x \leq 1/2$. In the phase of Boruvka's algorithm, where estimate $\tilde{k}$ is made, the probability of choosing exactly one edge between A and B is **at least**

$$\frac{\tilde{k}}{2} \cdot \frac{1}{\tilde{k}} \left(1 - \frac{1}{k}\right)^{2\tilde{k}} \geq \frac{\tilde{k}}{2} \cdot \frac{1}{\tilde{k}} \left(4^{\frac{1}{k}}\right)^{2\tilde{k}} \geq \frac{1}{40} \ .$$

(2b) Show that $O(\log n)$ phases of the new Boruvka-style algorithm, where per phase we get an outgoing edge from each component with probability at least $1/40$, suffice to determine the connected components, with high probability. In every phase of the algorithm, we remove at least $1/160$ components in expectation. Setting the constant in the $\mathcal{O}$ notation large enough, we can use calculations similar to the ones from the previous exercises (Exercise 12) to obtain the result.

3 Graph Sketching for Testing Bipartiteness

Consider a setting similar to the above problem, where each node v in an arbitrary n-node graph $G = (V, E)$ knows only its own edges. These nodes have access to shared randomness.

Exercise

(3a) Devise an algorithm where each node sends $O(\log^3 n)$ bits to the referee and then the referee can decide whether the given graph $G = (V, E)$ is bipartite or not.

HINT: Think about transforming G into a new graph H such that the number of connected components of H indicates whether G is bipartite or not.

Consider the following graph construction. We replace every node $v \in V$ with two nodes, v_{in} and v_{out} and connect v_{in} to u_{out} for every $\{v, u\} \in E$. Recall now, that a bipartite graph has no odd cycles. Furthermore, if there are only even cycles in the graph, any path from node v_{in} leads back to v_{in}. This follows from the observation that any path has an even amount of steps and every second node on the path is going to be an "in" node. Conversely, a path from v_{in} to v_{out} can be found by following a path starting from v_{in} going around an odd cycle back to v_{out}. Due to the odd amount of steps in this path, the end must be an "out" node. Therefore, we can test bipartiteness by validating that for all $v \in V$, v_{in} is not connected to v_{out}.