Przemysław Uznański April 25, 2017

Advanced Data Structures

Spring Semester 2017

Exercise Set 9

Recall from the lecture:

Definition. *Given* U *and* m*, a family of sets* $\mathcal{S} \subseteq 2^U$ *is a* separator family*, if for any disjoint* $A, B \subseteq U$ *such that* $|A|, |B| \leq m$*, there is* $C \in \mathcal{S}$ *such that* $A \subseteq C$ *and* $B \subseteq U \setminus C$*.*

Exercise 1:
Prove that there exist separator family $\mathcal{S}$ such that $|S| \leq 2^{\mathcal{O}(m+\log\log|U|)}$.

Hint: Use probabilistic method. There are at least two proof strategies:

- Pick $C \in 2^U$ at random. What is the probability that it separates any given pair A, B?
- Pick at random hash fuction $h : U \to [4m]$. What is the probability, that for any given set M of size $2m$, h is injective on M?[1] If you fix, $M = A \cup B$, then h separates A from B.

Exercise 2:
Finish the proof from the lecture (that is, in the *worst case* connectivity takes $\Omega(\log n)$ steps per operation). Recall: T is the number of leaves under given node, R, W are the memory cells, either read in the right subtree or written in the left subtree, respectively. By applying Exercise 1 to the simulation argument, we get (*how?*):

$$|R \cap W| \cdot \mathcal{O}(\log n) + \mathcal{O}(|R| + |W| + \log\log n) = \Omega(T\sqrt{n}\log n)$$

Show that it suffices (consider two cases, either $|R| + |W|$ is small or large).

Exercise 3:
Show $\Omega(\log n)$ lower bound for the dynamic testing of connectivity of the whole graph.

Hint: Consider G_1, graph from connectivity lower bound. Let $G_2 = G_1$ plus extra vertex s, where s will be connected to all vertices from first column. Updates remain the same. Show how to perform connectivity queries in G_1 by testing connectectivity of the whole G_2 (you can afford to add and remove constant number of edges before and after each query).

Exercise 4:
Show $\Omega(\log n)$ lower bound for *dynamic minimal spanning forest* problem.

Hint: It holds even if every edge has weight 1.

[1] see: *perfect hash family*