Homework #2 Due: September 22–24, 2019

This is an oral presentation assignment. There are three problems. You should work in groups of three. The sign-up sheet will be online soon (details on Piazza) and your group should sign up for a slot. Each person in the group must be able to present every problem. The TA/Professor will select who presents which problem. You are not required to hand anything in at your presentation, however having a written solution is often helpful for doing the presentation.

The bonus problem, B2, is a written problem, and can be turned into gradescope.

The second programming problem will be released at a later time.

1 (30 pts) Amortization

(a) Red and Blue Nodes.

A splay tree is being used to store $n + m$ items; $n$ of them are red and $m$ of them are blue.

Using the standard splay tree potential function (with appropriately chosen weights) prove the following two statements:

- The amortized number of splay steps (as defined in lecture) done when a red item is accessed is at most $4 + 3 \log_2 n$.
- The amortized number of splay steps (as defined in lecture) done when a blue item is accessed is at most $4 + 3 \log_2 m$.

This is curious because if there are very few red items, it seems to become very efficient to access them. However the algorithm itself does not even know which items are red and which are blue! Can you explain this?

(b) A Deep Splay.

i. You have a splay tree of 1,000,000 nodes. Each node has a weight of 1. Using these weights, the usual sizes and ranks of all nodes are computed. You splay a node $x$ that is at depth 1000 in the tree. Consider the 500 splay steps that are done. Call a splay step “scary” if the rank of $x$ does not change during the step. What is the minimum number of scary splay steps that can be occur?

ii. Continuing with the scenario in part i., you correctly reason that the potential ought to decrease. Find the largest value of $k$ you can such that it’s guaranteed that the potential will decrease by at least $k$. 
(c) **Chopping in a Graph.**

Let $G_0$ be the complete undirected graph of four vertices. Now, starting with $G_0$ the graph is updated $n$ times. $G_{i+1}$ is obtained from $G_i$ by one of the following operations:

- **link($u,v$):** Add edge $(u,v)$ to the graph. $u$ and $v$ are distinct and not already an edge in the graph. Cost = 1.

- **chop($u$):** Vertex $u$ (of degree $d(u)$) is deleted from the graph, and replaced by a collection of new vertices $x_1, x_2, \ldots, x_{d(u)}$ connected in a cycle. The edges that used to be connected to $u$ are each connected to a distinct one of these $x_i$s. After this step, each $x_i$ is of degree three. (See the figure below.) The cost of this operation is $d(u)$. (You can think of this as what happens when you chop a corner off of a polyhedron.)

Find a constant $c$ (the smallest possible) and show that any sequence of links and chops has a total cost $\leq cn$.

Structure your proof as follows: Define a potential function. Show that with this potential the amortized cost of chop() and link() are at most $c$. Show that the initial potential is less than the final potential. (It may be useful to view the potential function as placing tokens on nodes of the graph, as in the analysis of splay trees.)

2. (25 pts) **Flow Problems**

(a) **Carpools.**

Say there are $m$ days, and $S_i$ is the set of people that carpool together on day $i$. For each set $S_i$, one of the people $j \in S_i$ must be chosen to be the driver that day. Since people would rather not drive, they want the work of driving to be divided as fairly as possible. Your task in this problem is to give an algorithm to do this efficiently.

The fairness criterion is the following: Say that person $p$ is in some $k$ of the sets, which have sizes $n_1, n_2, \ldots, n_k$, respectively. Person $p$ should really have to drive $\frac{1}{n_1} + \frac{1}{n_2} + \cdots + \frac{1}{n_k}$ times, because this is the amount of resource that this person effectively uses. Of course this number may not be an integer, so let’s round it up to an integer. This quantity $\lceil \frac{1}{n_1} + \frac{1}{n_2} + \cdots + \frac{1}{n_k} \rceil$ is called their *fair cost*. A fair solution is a schedule for who drives on each day, such that each person drives no more than their fair cost.
For example, say that on day 1, Aram and Bob carpool together, and on day 2, Aram, Celia, and Dorothy carpool together. Aram’s fair cost would be $\lceil 1/2 + 1/3 \rceil = 1$. So Aram driving both days would not be fair. Any solution except that one is fair.

Give a polynomial-time algorithm that, given $S_1, S_2, \ldots, S_m$, computes a fair solution. This will also show that there always exists a fair solution.

(b) **Dominos on a Mutilated Chessboard.**

You’re given an $n \times m$ grid of squares, some of which are marked “forbidden”. The goal is to place as many $2 \times 1$ dominos on the chessboard as possible. Each domino must cover two adjacent non-forbidden squares. Give an $O(n^2m^2)$ algorithm to compute this.

3. (20 pts) **Long Path or Large Independent Set**

Here we will see an application of DFS to approximating one of two problems in an undirected graph $G(V,E)$ – longest path and shortest independent set. (An independent set is a set of vertices $U \subseteq V$ with no edge in $E$ connecting any two vertices in $U$.)

(a) Prove that if $(u,v) \in E$, then in any DFS tree of $G$, either $u$ is an ancestor of $v$ or vice versa.

(b) Give an $O(n + m)$-time algorithm which either computes a simple path of length $\lceil \sqrt{n} \rceil$ or an independent set of cardinality $\lceil \sqrt{n} \rceil$.

**Context:** Both longest path and independent set are NP-hard problems, and it is even NP-hard to approximate them within any $n^{1-\epsilon}$ factor, for any constant $\epsilon > 0$. Nonetheless, this exercise shows that for any graph you can approximate (at least) one of these problems to within an $O(n^{1/2})$ factor, in linear time.

B2. (Bonus) **Counting Planar Spanning Trees**

You’re given an undirected connected graph $G = (V,E)$ where $V = \{0, 1, \ldots, n - 1\}$. The vertices are placed in counter-clockwise order around a unit circle with vertex $k$ placed at the point $(\cos(2\pi k/n), \sin(2\pi k/n))$.

Give an $O(n^3)$ algorithm to count the number of planar spanning trees of $G$. That is, a tree satisfies the requirements if no two of its edges cross in the interior of the unit circle, all of its edges are in $E$, and the tree connects all the vertices of $G$.

So, for example, if $G$ is the complete graph on four vertices, then there are 12 planar spanning trees that satisfy the conditions.