Homework 1 (due March 16)

Teams of up to three people can turn in joint homework solutions. You may use any resource at your disposal—printed, electronic, or human—but cite/acknowledge your sources, just as though you were writing a research paper.

1. After the Great Academic Meltdown of 2010, you get a job as a cook's assistant at Jumpin' Jack's Flapjack Stack Shack, which sells arbitrarily-large stacks of pancakes for just 50 cents each. Jumpin' Jack insists that a stack of pancakes given to one of his customers must be sorted, with smaller pancakes on top of larger pancakes. Also, whenever a pancake goes to a customer, its top side should not be burned.

   The cook provides you with a unsorted stack of n perfectly pancakes, all of different sizes, possibly burned on one or both sides. Your task is to throw out the pancakes that are burned on both sides (and *only* those) and sort the remaining pancakes so that their burned sides (if any) face down. Your only tool is a spatula. You can insert the spatula under any pancake and then either *flip* or *discard* the stack of pancakes above the spatula.

   (a) Describe an algorithm to filter and sort the pancakes using $O(n)$ operations.

   (b) We can represent a stack of pancakes by a sequence of distinct integers between 1 and n, representing the sizes of the pancakes, with each number marked to indicate the burned side(s) of the corresponding pancake. For example, $\underline{1}\overline{4}3\underline{\overline{2}}$ represents a stack of four pancakes: a one-inch pancake burned on the bottom; a four-inch pancake burned on the top; an unburned three-inch pancake, and a two-inch pancake burned on both sides.

   Describe a data structure that supports each of the following operations in $O(\log n)$ amortized time:

   - POSITION(x): Return the position of integer x in the current sequence, or 0 if x is not in the sequence.
   - VALUE(k): Return the kth integer in the current sequence, or 0 if the sequence has no kth element. This is essentially the inverse of POSITION.
   - TOPBURNED(k): Return TRUE if and only if the top side of the kth pancake in the current sequence is burned.
   - FLIP(k): Reverse the order and the burn marks of the first k elements of the sequence. For example, FLIP(3) transforms $\underline{1}\overline{4}3\underline{\overline{2}}$ into $3\underline{4}\overline{\underline{1}\overline{2}}$.
   - DISCARD(k): Remove the first k elements of the sequence.

   Using this data structure and your algorithm from part (a), you can filter and sort n burned integers in $O(n \log n)$ time.

2. [Demaine, Harmon*, and Iacono]

   Let $X = \langle x_1, x_2, \ldots, x_m \rangle \in [n]^m$ be a sequence of m integers in the set $[n] = \{1, 2, \ldots, n\}$. We can visualize this sequence as a set of integer points in the plane, by interpreting each element x_i as the point (x_i, i). The resulting point set, which we can also call X, has exactly one point on each row.

   (a) Let Y be an arbitrary set of integer points in the plane. A pair of points $p, q \in Y$ is *isolated* if (1) they differ in both coordinates and (2) the closed axis-aligned rectangle with p and q in opposite corners contains no other point in Y. If the set Y contains no isolated pairs, we call Y a *commune*.

   Let X be an arbitrary set of points on the $n \times n$ integer grid with exactly one point per row. Show that there is a commune Y that contains X and consists of $O(n \log n)$ points.

   (b) Consider the following model[1] of dynamic binary search trees. For each request x_i in the given sequence X, the search algorithm *arbitrarily reconfigures* some subtree S_i of the current search tree T_i to obtain the next search tree T_{i+1}. The only restriction is that X_i must contain both x_i and the root of T_i. (For example, in a splay tree, S_i is the search path to x_i.) The *cost* of the ith access is the number of nodes in the subtree S_i.

   Prove that the minimum cost of executing an access sequence X in this model is at least the size of the smallest commune containing the corresponding point set X. *[Hint: Lowest common ancestor.]*

   (c) Suppose X is a random permutation of the integers $1, 2, \ldots, n$. Use the lower bound in part (b) to prove that the expected minimum cost of executing X is $\Omega(n \log n)$.

   ★(d) Describe a polynomial-time algorithm to compute (or even approximate up to constant factors) the smallest commune containing a given set X of integer points, with at most one point per row. Alternately, prove that the problem is NP-hard. *[Hint: Tango/multisplay trees imply a fast $O(\log \log n)$-approximation algorithm.]*

[1] This model is slightly more restrictive than the one Sleator and Tarjan originally proposed. Specifically, we are not allowed to perform arbitrary isolated rotations at unit cost

3. Let G be an embedded planar graph with weighted edges, and let s be one of its vertices. Describe a data structure to maintain G under the following operations:

   - SETLENGTH(e, ℓ) — Change the weight of edge e to ℓ, which is always positive.
   - DISTANCE(v) — Return the length of the shortest path from s to v with the current edge weights.

   Assume without loss of generality that at all times, there is a unique shortest path between any pair of vertices in G. Your DISTANCE algorithm should run in $O(\log n)$ amortized time. Your SETLENGTH algorithm should run in amortized time $O((k+1)\log n)$, where k is the number of edges that are in the shortest-path tree rooted at s after the operation but not before.[2]

   [Hint: Let G^* be the dual graph of G, whose vertices correspond to faces of G. The edges in G^* that are not dual to edges in the shortest path tree comprise a spanning tree of G^*. Use appropriate dynamic tree data structures to maintain both the shortest path tree and the dual spanning tree.]

   [Hint: For SETLENGTH*, imagine* ***continuously*** *changing the length of e from its old length to ℓ; when and how does the shortest-path tree change? You may need different algorithms for increasing and decreasing length, and for edges on and off the shortest-path tree.]*

[2] Ideally, SETLENGTH would also run in $O(\log n)$ amortized time, regardless of how the shortest path tree changes, but nobody has a clue how to do that. The best algorithm known to date, due to Fakcharoenphol* and Rao [*Proc. FOCS* 2001], supports both SETLENGTH and DISTANCE in $O(n^{2/3}\log^{7/3} n)$ time. Any $o(n^{2/3})$ running time would be publishable. For outer-planar graphs, Djidjev, Pantziou, and Zaroliagis [*Algorithmica* 2000] describe algorithms to support both SETLENGTH and DISTANCE (between *any* two vertices) in $O(\log n)$ time.