Homework 16: Due Friday, May 12

Problem 1: A *derangement* of the set $\{1, 2, 3, \ldots, n\}$ is a permutation of $\{1, 2, 3, \ldots, n\}$ such that the number i does not appear in position i for all $i \in \{1, 2, 3, \ldots, n\}$. For example, $(2, 3, 1)$ is a derangement of $\{1, 2, 3\}$, but $(3, 2, 1)$ is not a derangement of $\{1, 2, 3\}$ (because the 2 is in position 2). Suppose that we code permutations (and hence derangements) of $\{1, 2, 3, \ldots, n\}$ as lists of length n, rather than as n-tuples (so that we use $[2, 3, 1]$ instead of $(2, 3, 1)$ in ML).
a. Write an ML function isDerangement that takes as input a list ps, assumed to be a permutation of the set $\{1, 2, 3, \ldots, \mathsf{length(ps)}\}$, and returns true if ps is a derangement, and false otherwise.
b. Using setFilter, write an ML function derangements that takes as input a natural number $n \in \mathbb{N}$, and returns the set of all derangements of $\{1, 2, 3, \ldots, n\}$.
c. Write an ML function derangementRatio that takes as input a natural number n, and returns the fraction (in decimal form) of permutations of $\{1, 2, 3, \ldots, n\}$ that are derangements. For example, on input 3, your function should return .33333333... because 2 of the 6 permutations of $\{1, 2, 3\}$ are derangements, and $\frac{2}{6} = .33333333\ldots$.
Cultural Aside: As you increase the input n to part c, the outputs appear to be approaching a certain number. It turns out that the outputs converge very rapidly to the number $1/e$.

Problem 2: Let $n, k \in \mathbb{N}$. A sequence of nonnegative integers $(a_1, a_2, \ldots, a_k)$ such that $a_1 + a_2 + \cdots + a_k = n$ is called a *weak composition* of n into k parts. For example $(1, 3, 5, 3)$ is a weak composition of 12 into 4 parts and $(2, 0, 5, 1, 0, 0)$ is a weak composition of 8 into 6 parts. Suppose that, as in Problem 1, we code weak compositions using lists instead of tuples in ML. Write a recursive ML program (i.e. don't just take a big set and filter) that takes two natural numbers n and k as input, and produces the set of all weak compositions of n into k parts. For example, on inputs $n = 3$ and $k = 2$, it should produce

$$[[0, 3], [1, 2], [2, 1], [3, 0]]$$

(although possibly in a different order). If $n = 0$ and $k = 0$, your program should produce [nil] (because the empty sequence is technically a weak composition of 0 into 0 parts). If $n > 0$ and $k = 0$, your program should produce nil.
Hint: Here's a helpful way to think about this recursively: If $k > 0$, then a weak composition of n into k parts is of one of two types: either it starts with a 0 or it does not. In the former case, if we omit the 0, then we obtain a weak composition of n into $k - 1$ parts. In the latter case, if we decrease the first element by 1, then we obtain a weak composition of $n - 1$ into k parts.

Problem 3: In class, we talked about a different way to code directed graphs. The idea is that we will code them as a list of lists, where the first element of each list gives the vertices, and the remaining elements of each list give the vertices that appear as the endpoints of arrows starting with the initial vertex. For example, given our directed graph

$$(\mathsf{intervalSet}(1, 8), [(1, 2), (2, 3), (3, 4), (4, 5), (3, 8), (1, 7), (7, 5), (6, 7)])$$

from class, we would now code it as

$$[[1, 2, 7], [2, 3], [3, 4, 8], [4, 5], [5], [6, 7], [7, 5], [8]].$$

Write an ML function that takes a directed graph represented in the first form, and produces a representation in this latter form.

Problem 4: Let D be a directed graph with vertex set V.

a. In class, we talked about defining a new relation on V by letting $u \sim w$ mean that there exists an u, w-walk in D. Write an ML function that takes as input a directed graph D, and produces this relation. That is, it produces the set of pairs (u, w) such that there is a u, w-walk in D.
b. If D is a graph, then it turns out that the relation $\sim$ in part a is an equivalence relation. The equivalence classes of this relation are called the *connected components of* D. Write an ML function that will take as input a graph D (i.e. you should assume that the edge relation is symmetric), and will produce the number of connected components of D.