CSI 5165 Combinatorial Algorithms | Winter 2018
Computer Science | University of Ottawa

**Homework Assignment #1** (100 points, weight 15%)
Due: Feb 14, 2018

---

## Generating elementary combinatorial objects

1. (10 points) **Simple practice with combinatorial generation algorithms**
Calculate the result for the following operations. Show your work.

   - Subsets:
   Give the SUCCESSOR and the RANK of 11010110 in the Gray code $G^8$.
   - $k$-subsets:
   Give RANK of $\{3, 6, 7, 9\}$ considered as a 4-subset of $\{1, \ldots, 13\}$ in lexicographic and revolving-door order. What is the SUCCESSOR in each of these orders?
   - Permutations:
   Find the rank and successor of the permutation $[2, 4, 6, 7, 5, 3, 1]$ in lexicographic and Trotter-Johnson order.

     UNRANK the rank $r = 54$ as a permutation of $\{1, 2, 3, 4, 5\}$, using the lexicographic and Trotter-Johnson order.

2. (30 points) Another way to order the subsets of an $n$-set is to order them first in increasing size, and then in lexicographic order for each fixed size. For example, when $n = 3$, this ordering for the subsets of $S = \{1, 2, 3\}$ is:

$$\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}.$$

   Develop **unranking**, **ranking** and **successor** algorithms for the subsets with respect to this ordering.

   Hint: Adapt the ideas developed for the lexicographical order of $k$-subsets of an $n$-set to this situation. Note that efficiency will play a role in the evaluation.

3. (30 points) A *derangement* is a permutation $[\pi[1], \pi[2], \ldots, \pi[n]]$ of the set $\{1, 2, \ldots, n\}$ such that $\pi[i] \neq i$, for all $1 \leq i \leq n$. Let $D_n$ denote the number of derangements of an $n$-element set. Note that $D_1 = 0$ and $D_2 = 1$. To show that $D_n = (n-1)(D_{n-1}+D_{n-2})$, for $n \geq 3$, we can use the following argument:

   We can set $\pi[1]$ in $n - 1$ ways, namely with $i = 2, 3, \ldots n$.
   Once $\pi[1] = i$ there are two possibilities:

   - $\pi[i] = 1$, in which case we list all derangements of $\{1, \ldots, n\} \setminus \{1, i\}$ (there are $D_{n-2}$ of them) in order to complete the current derangement.

- $\pi[i] \neq 1$, in which case we can rename value 1 as $i$ list all derangements of $\{1, 2, \ldots, n\} \setminus \{1\}$ (there are $D_{n-1}$ of them), and then change back $i$ to 1 in each of these derangements.

Use this recurrence relation (and its associate argument) to develop an algorithm to generate all the derangements. Note that you do not need to necessarily come up with a successor algorithm; indeed a recursive algorithm might be the easiest solution. Ideally, you would not store several derangements in main memory at the same time, that is, after a derangement has been generated it can be printed out; this would keep your memory requirements in $O(n)$ rather than exponential. You may have to keep some $n$-arrays in your program in order to deal with current permutations, indexes that are active and possible relabelings. Note that efficiency will play a role in the evaluation.

(a) Provide a **pseudocode** of your algorithm (with similar level of detail as the algorithms given in textbook). Please, also add any comments or extra **explanations** necessary to understand why your pseudocode works.

(b) **Implement** our algorithm, providing a **printout** of the code, as well as **outputs** for $n = 3, 4, 5$

4. (30 points) **Generalized Gray codes**

(a) Let $m_0, m_1, \ldots m_{n-1}$ be integer numbers greater than or equal 2. In this exercise we want to generate all $n$-tuples $(a_{n-1}, a_{n-2}, \ldots, a_1, a_0)$ where $0 \leq a_j < m_j$ for all $j$, $0 \leq j < n$, according to the following minimal change ordering: two successive tuples differ in exactly one component with the absolute value of their difference equals to 1 (i.e. the component is either incremented or decremented by 1). Adapt the binary reflected Gray code successor algorithm to the case of this generalized Gray code. Give your algorithm in pseudocode form.

(b) Given the prime factorization of a number $p_1^{e_1} p_2^{e_2} \cdots p_t^{e_t}$, give an algorithm to run through all divisors of the number, by repeatedly multiplying or dividing by a single prime at each step.
**Hint:** Use the algorithm developed in part 1.