CSI 5165 Combinatorial Algorithms
Computer Science

Fall 2003
University of Ottawa

Homework Assignment #1 (100 points, weight 15%)
Due: Tuesday, October 7, at 9:00 a.m. (in lecture)

---

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 01010110 in the Gray code G^8.
   - k-subsets:
     Give RANK of $\{3, 6, 7, 9\}$ considered as a 4-subset of $\{0, 1, \ldots, 12\}$ 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 $r = 56$ as a permutation of $\{1, 2, 3, 4, 5\}$, using the lexicographic and Trotter-Johnson order.

2. (10 points) **Successor algorithm for co-lex ordering of k-subsets of an n-set**
   In Section 2.3.2 of the textbook you can find the definition of co-lex ordering for k-subsets of an n-set; give a SUCCESSOR algorithm for this ordering.

3. (40 points) **Generating k-multisets of an n-set** (Exercise 2.13)
   A *multiset* is a set with (possibly) repeated elements. A k-multiset is one that contains k elements (counting repetitions). Thus, for example, $\{1, 2, 3, 1, 1, 3\}$ is a 6-multiset. The k-multisets of an n-set can be ordered lexicographically, by sorting the elements in each multiset in non-decreasing order and storing the result as a list of length k.

   When writing your algorithms, you will need to consider the following quantity:
   L_k^n = number of k-multisets of an n-set.

   (a) (5 points) Give a recurrence formula for L_k^n.

   (b) (5 points) Give a brief algorithm which computes and stores L_i^m, for all $i \leq k$ and $m \leq n$, on a table.

   (c) (30) Develop successor, ranking and unranking algorithms for the k-multisets of an n-set. You may simply refer to table values calculated in the previous part.

   Hint: Try to adapt the algorithms for lexicographical ordering of k-subsets of an n-set. Note that you will use quantities L_i^m in place of quantities $\binom{p}{s}$.

Example: $k = 3$, $n = 4$, $L_3^4 = 20$

rank	T	$\vec{T}$
0	$\{1,1,1\}$	$[1,1,1]$
1	$\{1,1,2\}$	$[1,1,2]$
2	$\{1,1,3\}$	$[1,1,3]$
3	$\{1,1,4\}$	$[1,1,4]$
4	$\{1,2,2\}$	$[1,2,2]$
5	$\{1,2,3\}$	$[1,2,3]$
6	$\{1,2,4\}$	$[1,2,4]$
7	$\{1,3,3\}$	$[1,3,3]$
8	$\{1,3,4\}$	$[1,3,4]$
9	$\{1,4,4\}$	$[1,4,4]$
10	$\{2,2,2\}$	$[2,2,2]$
11	$\{2,2,3\}$	$[2,2,3]$
12	$\{2,2,4\}$	$[2,2,4]$
13	$\{2,3,3\}$	$[2,3,3]$
14	$\{2,3,4\}$	$[2,3,4]$
15	$\{2,4,4\}$	$[2,4,4]$
16	$\{3,3,3\}$	$[3,3,3]$
17	$\{3,3,4\}$	$[3,3,4]$
18	$\{3,4,4\}$	$[3,4,4]$
19	$\{4,4,4\}$	$[4,4,4]$

4. (40 points) **Backtracking for orthogonal Latin squares**

   A *Latin square* of order n is an n by n array A, whose entries are chosen from $X = \{1, 2, \ldots, n\}$, such that each symbol in X occurs in exactly one cell in each row and in each column of A.

   Two Latin squares $A = (a_{ij})$ and $B = (b_{ij})$ are said to be *orthogonal* if for any pair (k, l) of elements of X, there is a unique position i, j such that $a_{ij} = k$ and $b_{ij} = l$.

   Write a backtracking algorithm that, given a Latin square A of order n, determines the number of Latin squares of order n that are orthogonal to A and have entries $[1, 2, \ldots, n]$ in the first row. Please, hand in the pseudocode, as well as a program implementing your algorithm.

   Please, make your program also print statistics on the number of (recursive) calls to your backtrack algorithm, for each input. Efficiency and clarity count.

   Run your program with the following inputs for $n \leq 4$ and show printouts for your results. Try your luck with $n = 5, 6$ and report any results obtained or difficulties encountered.

$$A_3 = \begin{array}{|ccc|}\hline 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \\ \hline\end{array} \quad A_4 = \begin{array}{|cccc|}\hline 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \\ 3 & 4 & 1 & 2 \\ 4 & 3 & 2 & 1 \\ \hline\end{array} \quad B_4 = \begin{array}{|cccc|}\hline 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \\ 3 & 4 & 2 & 1 \\ 4 & 3 & 1 & 2 \\ \hline\end{array}$$

$$A_5 = \begin{array}{|ccccc|}\hline 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 1 \\ 3 & 4 & 5 & 1 & 2 \\ 4 & 5 & 1 & 2 & 3 \\ 5 & 1 & 2 & 3 & 4 \\ \hline\end{array} \quad A_6 = \begin{array}{|cccccc|}\hline 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 3 & 4 & 5 & 6 & 1 \\ 3 & 4 & 5 & 6 & 1 & 2 \\ 4 & 5 & 6 & 1 & 2 & 3 \\ 5 & 6 & 1 & 2 & 3 & 4 \\ 6 & 1 & 2 & 3 & 4 & 5 \\ \hline\end{array}$$

Some facts: The number of Latin squares of order n which have the first row $[1, 2, \ldots, n]$ for $n = 3, 4, 5, 6$ is 2, 24, 1344, $9408 \times 5!$, respectively. This large number of candidates suggests you should test for orthogonality on the fly. More specifically, it is **not a good idea** to do a backtracking algorithm that generates each of these Latin squares and only then checks for orthogonality with the given Latin square (there would be no hope for $n = 6$). It is advisable to check for orthogonality violations while each Latin square is being generated.