

**CMSI 282 Problem Set #4**  
**Due March 19, 2009**

Directions: Compute #'s 1-3 (by hand), and do at least two more (via computer program).

- (1) Set Partitions: (a) Enumerate the partitions of  $\{0,1,2,3,4\}$ . (b) How many partitions of  $\{0,1,2,3,4,5,6\}$  are there?
- (2) Integer Partitions: (a) Enumerate the partitions of 6. (b) How many partitions of 7 are there?
- (3) Ramsey Partitions: (a) Enumerate the Ramsey partitions of 9, with respect to 5 and 4. (b) Give three distinct Ramsey partitions of 13 with respect to 8 and 5. (c) Is  $\{3,3,3,2,2,1,1,1\}$  a Ramsey partition of 16 with respect to 11 and 5?

- (4) Subsequences: Find the longest common subsequence of the strings

*accbbbacaccacdebaebcddcabcbbaaecbcdebdcdebd*      and

*bacbacebdcbbcbdcdeabadeadcbbacedaebcddebd*

- (5) Edit Distance: If  $s$  and  $t$  are different strings, then we can transform  $s$  to  $t$  via some sequence of these operations:
- change some character of  $s$  to a different character;
  - delete some character of  $s$ ;
  - insert some new character into  $s$ .

For example, one (not very efficient) way of transforming *chorsex* into *houses* is to delete the  $c$ , then change the  $r$  to a  $u$ , then insert an  $s$  before the  $x$ , and then delete the  $x$ . Furthermore, if we suppose that each operation has unit cost, then the cost of this particular transformation is four. Define the *edit distance* between two strings as the minimum-cost way of transforming one string to another (zero, if they're identical).

Make *EditDistance.java*, a program for computing the edit distance between arbitrary strings. A typical invocation of your program will look like this:

*java EditDistance chorsex houses*

(6) Permutations:

- (a) Make *public class Permutation* for constructing and manipulating permutation objects, which consist of the integers  $0, 1, 2, \dots, (n-1)$ , for arbitrary  $n$ . For example, here are some permutation objects, represented as strings:

`<0,1,2,3,4>` the first 5-permutation;  
`<3,1,4,0,2>` another example of a 5-permutation;  
`<5,4,3,2,1,0>` the last 6-permutation.

Your class will need a constructor:

```
new Permutation (5); // constructs the initial 5-permutation
```

and it will need some methods like:

```
p.advance(); // transforms p to the next permutation in lex order  
p.isLast(); // returns true iff p is the last perm in lex order  
p.toString(); // returns a string representation of permutation p  
p.intAt(i); // returns the ith component of permutation p
```

and so on.

- (b) Make *PentagonSolver.java*, a brute-force program that uses this class to enumerate all solutions to the pentagon problem that was stated in class.

- (7) Integer Knapsack: A thief breaks into a house and spots (indivisible) items weighing 16, 14, 18, 15, 7, 21, 45, 15, 5, 9, 13, 18 pounds, which he values at 14, 13, 22, 20, 12, 25, 51, 18, 9, 13, 17, 24 dollars, respectively. Unfortunately (for him), his knapsack only holds 60 pounds of stuff. (a) Which items should he take? (b) If his knapsack were able to hold 73 pounds, which items should he take?