HOMEWORK 1: Solutions

October 30, 2012

Don't regard these as the unique way to solve the problems in the homework.

1 Double counting applied to permutation identities

In the seminar we saw that one could prove identities using combinations by counting words in several ways.

Using a similar method show that when a, b are integers greater or equal than two and $0 \leq k \leq a + b$ we have

$$C_{a+b}^{k} = \sum_{i=0}^{k} C_{a}^{i} \cdot C_{b}^{k-i}$$

(Hint: what kind of words does C_n^k count ?)

Solution: C_{a+b}^{k} counts the number of *binary words of length* $a + b$ having *exactly* k *ones*. Let i be the number of ones of such a word in the first a positions of the word. There are C_a^i ways to choose these positions. Correspondingly, there are $k - i$ positions in the last b ones that have to be 1, and C_b^{k-i} ways to choose them. Summing up we get the formula.

2 Paths in the plane

You are in the plane at point P=(0,0) of the integer grid (think of a math notebook). You want to reach a point with coordinates $Q = (m, n)$, where m, n are integers ≥ 1. You want to go from P to Q. At each step you either

- move one position up, or
- move one position to the right

(that is you "follow the grid" and never go left or down).

How many such paths do exist ? Justify your answer (a simple guess is not good enough).

Solution: Any such path is composed of exactly $m+n$ segments of length 1. We can encode such a path by a word of length $m+n$ over the alphabet R, U. R means "we go one step right", U means "move one position up". Any such word has exactly m letters of type R, and is uniquely determined by their choice. So the number of paths is C_{m+n}^{m}.

3 Pseudocode: elementary school method for addition

Write the method we learned in school for adding two natural numbers in pseudocode.

```
program add(int first[n], int second[n])

// we assume for simplicity that first and second number have
// the same dimension. Otherwise we have to pad the smallest
// number with zeros.
// also we assume that least significant digit is in the first position.

int result [n+1];

// result may have one digit more.

int index = 1;
int carry = 0;

int partial = 0;

while (index <= n) do
     partial = first[index]+second[index]+carry
     if (partial > 10)
          carry = 1
          partial = partial - 10
     else
          carry = 0
     endif

     result[index]=partial
     index++;
endwhile

// might have extra digit
// when carry is positive

if (carry > 0)
```

```
result[n+1]=carry
```

4 Coin weighing: two fake coins

You have nine coins. *Two* of them are fake, and are lighter than the rest. You have a scale with two sides. You can put an arbitrary number of coins on each sides and observe if the two sides have equal weight or not, and which side is lighter.

Devise at least one algorithm and write it in pseudocode for detecting the two fake coins using at most *four* comparisons.

SOLUTION:

First we present two procedures we will use. Each of them achieves its goal by making *exactly one comparison.*

```
1. Procedure for identifying one fake coin out of three:

compare two coins, leaving one coin aside.

if the weighing shows different weights
   then
     the lighter coin is the fake one,
   else
     the remaining coin is fake
```

The second procedure is very similar. In fact it is essentially the first procedure with "fake" and "genuine" reversed, and fake coins being heavier than genuine ones.

```
2. Procedure for identifying two fake coins out of three:

compare two coins, leaving one coin aside.

if the weighing shows different weights
   then
     the lighter coin and the remaining coin
     are the fake ones
   else
     both coins are fake
```

We use these procedures to solve the main problem as follows:

- First, we divide the coins into three groups A,B,C of three coins each, and compare the first two groups A,B.

Case 1. One of the groups (say B) is heavier.

Then B has to consist of good coins only. By comparing the remaining group A against C we can distinguish between the following two situations:

- **A and C have equal weight.** Then A and C contain one fake coin each. We can apply the first procedure to locate them.
- **A and C have different weights.** Then the lighter of the two groups contains two fake coins. We apply the second procedure to identify them.

Case 2. A and B have identical weight.

Then either A and B have one fake coin each, or both fake coins are in C. One extra comparison (between A and C) tells us which of this two alternatives applies.

```
if (weight(A)<weight(C))
   then
     // A and B contain one fake coin each
     apply first procedure above to both A and B
   else
     // C contains two fake coins
     apply second procedure above to C
endif
```

5 Computing frequencies

An array contains n numbers (not necessarily distinct) in increasing order: $x[1] \leq x[2] \leq \ldots \leq x[n]$.

Give a pseudocode that will compute:

- the number of different numbers in the array.
- for each of them the number of times it appears in the array.

```
function count (int x[n])

// number of distinct elements found.
// also index of last element.
int elem = 1;

// array of frequencies.
int times[n]
// array of distinct elements
int distinct[n]

int index = 1;
```

```
times[1]=1
distinct[1]=x[1]

for (index = 2 to n)
   if (x[index]>x[index-1])
        // new element
        elem = elem+1;
        distinct[elem]=x[index]
        times[elem]=1;
   else
        // same element
        times[elem]=times[elem]+1
   endif
endfor

for (int i= 1 to elem)
  print "element " distinct[i] "occurs " times[i] "times\n"
```