6 Instruction – Recursion

6.1 Binomial

The *binomial coefficient* $C(n,k)$ gives the number of ways to choose k elements from a set of n elements, not considering the order of the chosen elements. An alternative characterisation is as the number of different subsets of k elements of a set of n elements. It has several applications in mathematics.

We come to a recurrence relation as follows. Take an arbitrary element E from the set of n elements. If we choose E, there are $n-1$ elements left to choose from and we have $k-1$ to choose. If we do not choose E, we have again $n-1$ elements left to choose from and we have still k elements to choose. All this under the assumption that we have something to choose from, i.e., $n > 0$, and we don't have to choose more elements than we have, i.e., $k \geq n$. $C(n,0) = 1$, which can be understood if we realize that choosing certain elements is equivalent to not choosing the others, so $C(n,k) = C(n,n-k)$, and hence $C(n,0) = C(n,n) = 1$, since there is exactly one way to choose all elements if order doesn't matter. So we define

$$\begin{array}{ll} C(n,k) = C(n-1,k-1) + C(n-1,k) & \text{for } n > k \geq 0 \\ C(n,0) = 1 & \text{for } n \geq 0 \\ C(n,n) = 1 & \text{for } n \geq 0 \end{array}$$

1. Write a program with a recursive implementation of the binomial coefficient, using the above definition. It will ask repeatedly for an n and a k on input and print the value of $C(n,k)$.

2. This implementation is quite inefficient, since it makes identical recursive calls several times. Keep track of the number of recursive calls of each pair of arguments using a two-dimensional array as an instance variable and print this array as a table.

3. Show that $C(n,k) = \frac{n!}{k!(n-k)!}$ and use this to implement the binomial in a non-recursive way.

4. Apply this function to some values where n is about 20 or 30. Chances are that you get funny results. E.g., do you find that $C(26,2) = 325$?

   Although this implementation doesn't suffer from the many-identical-recursive-calls syndrome, the intermediate factorials can easily exceed the capacity of int (or long). Therefore, find a recurrence relation based on the formula with the factorials, only recurrent in k and not in both n and k, as is the recurrence relation from the definition. This relation should express $C(n,k)$ in $C(n,k-1)$. Again write a recursive implementation of the binomial, now using this recurrence relation.

5. As a third alternative, increase the efficiency of the original recursive implementation by storing (*caching*) the intermediate values of $C(n,k)$ in a 2-dimensional array. Before diving into recursion, consult the array to see whether the desired value has been calculated before. Use the fact that an int array is automatically initialised with the value 0 and that $C(n,k)$ is never 0 in the domain. Optimalists (not to be confused with optimists) may use a triangular array, in order not to waste space to argument pairs where $k > n$, which are outside the domain of our binomials[1].

[1] There exist generalisations of the binomial where n can be any number, even complex; these are not considered here.