A note on the lecture of Tuesday, November 18

On Tuesday we used a bijection to prove that the number of $k$-subsets of $[n]$ is

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

if $0 \leq k \leq n$. The bijection is

$$f : \binom{[n]}{k} \rightarrow \binom{[n-1]}{k} \cup \binom{[n-1]}{k-1}$$

defined

$$f(S) = S \setminus \{n\}.$$  

For $0 < k \leq n$, the $k$-subsets of $[n]$ that lack $n$ as an element are

$$\binom{[n-1]}{k},$$

while the $k$-subsets of $[n]$ that contain $n$ are

$$\{\{n\} \cup S \mid S \in \binom{[n-1]}{k-1}\}.$$ 

A recursive algorithm for enumerating the $k$-subsets of $[n]$ is apparent. Here is an implementation in Haskell.

```haskell
comb n k | k < 0 || n < k = [] | k == 0 = [[]] | otherwise = comb (n-1) k ++ map (++[n]) (comb (n-1) (k-1))
```

This defines a function called `comb` with parameters $n$ and $k$. The last 3 lines are cases in what Java and C programmers call a `switch` statement.

In case $k < 0$ or $n < k$, there are no $k$-subsets of $[n]$, so the desired enumeration is an empty list ($[]$). In the remaining cases we may assume $0 \leq k \leq n$.

In case $k = 0$, the sole $k$-subset of $[n]$ is $\emptyset$.

Finally, if $k > 0$, then recursively call `comb` twice. The first invocation,

$$\text{comb} \ (n-1) \ k,$$
returns an enumeration of $\binom{n-1}{k}$. The second,

$$\text{comb } (n-1) \ (k-1),$$

returns an enumeration of $\binom{n-1}{k-1}$, and then element $n$ is appended (++) to each set. In Haskell, applying function $f$ to each element of list is done by

```
map f list.
```

Finally, the two enumerations are concatenated (++).

To evaluate the function, use the Glasgow Haskell Compiler.

http://haskell.org/ghc/

With the code in file lec24.hs, you can have the following interaction with the GHC interpreter at a command prompt.

```bash
$ ghci
GHCi, version 6.8.2: http://www.haskell.org/ghc/ :? for help
Loading package base ... linking ... done.
Prelude> :load lec24.hs
[1 of 1] Compiling Main ( lec24.hs, interpreted )
Ok, modules loaded: Main.
*Main> comb 3 5
[]
*Main> comb 5 3
[[1,2,3],[1,2,4],[1,3,4],[2,3,4],[1,2,5],[1,3,5],[2,3,5],[1,4,5],[2,4,5],[3,4,5]]
*Main> comb 5 0
[]
*Main> comb 0 0
[]
*Main> comb (-1) 0
[]
*Main> length $ comb 5 3
10
*Main> length $ comb 7 3
35
*Main> length $ comb 10 4
210
```