Project 1: Binomial Coefficients

CSE4081
Author: Eric Pereira

September 29, 2017

Algorithm / Code

```
long long binomCoeff(long long n, long long k) {
  if (k==0)
    return 1;

  return ((n*binomCoeff(n-1, k-1))/k);
}
```

Figure 1: Recursive binomial coefficient function written in C++

This is an example of a binomial coefficients done recursively, due to it being done recursively the time complexity is dependent on k, as there are k calls to the binomCoeff(`long long, long long`) function. This function also had to use long long values as that is the largest primitive type available in C++.

There are no good BigInteger objects or types available for C++, they tend to not be easy to use, give inaccurate results, or not support the correct operators to do this function (especially division); therefore, `long long`[1] seemed the best route.

Implementation

The way the program was implemented was in a recursive way, where it simply called upon itself many times until it reaches k=0.

The best way to view this is in the form of an equation:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \quad \textit{for } n > k > 0$$

Figure 2: Mathematical definition for recursive binomial coefficient equation[2]

In this way the recursion is very clearly seen. It is important to note that n and k are both non-zero whole numbers, and that n is always greater than k. This method of calculating binomial coefficients is linear, as it depends on when k reaches 0.

Due to k being less than n, and both numbers being subtracted at the same rate, k is always going to reach 1 first, and the function will run until such standards are met. Such a result means that the time complexity, expected, from this specific function will be k.

This is one of the quickest ways to run this type of equation, and actually serves as a useful example of how quickly a computer could make such a seemingly impossible task, this could be considered the ‘good’[3] way to recursively solve the binomial coefficient problem.

Test

Initially a test was done with small values to make sure it worked, the first values tested on this was 13 choose 4. The program did this correctly, as tested[4] by a binomial coefficient calculator. However this calculation did not put the Code to the test, the variables were assigned to the types `long long` after all.

It was time to see the limits for this test, and so the numbers 15,000 choose 7,500 were tested. Sure enough the program answered this incorrectly, the number was just way too large for the type `long long` to store (`long long` can only store a 64 byte number which is far too small for this test). Output was received, but the output was an inaccurate value as the number was so large it lead to integer overflow.

However, it is possible to still do tests, as cases such as 10,000 choose 10,000, or 10,000 choose 9,999, can get accurate results (the results would be n in the case that the binomial coefficient is $\binom{n}{n-1}$ for all n > 1, and 1 if the case is $\binom{n}{n}$). Utilizing this, the program could be tested accurately knowing it was still yielding correct results.

The largest number that was possible to hit, without any errors occurring, was a k value 261,911. Any value larger than this resulted in an error due to a lack of memory in the system.

Profile

This program was very fast. So fast in fact that it was not possible to catch it on the Gprof C++ profiler[5]. The profiler yielded such results when k value was 261,911:

```
 %   cumulative   self              self     total
time   seconds   seconds    calls  Ts/call  Ts/call  name
 0.00      0.00     0.00        1     0.00     0.00  _GLOBAL__sub_I__Z6choosexx
 0.00      0.00     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
 0.00      0.00     0.00        1     0.00     0.00  binomCoeff(long long, long long)
```

Figure 3: Table with time of functions and data in a C++ program created by the Gprof C++ profiler.

The Program was way too fast to be calculated. The quickest that the Gprof profiler can see is $\frac{1}{100}$ of a second. Due to the program being so quick, it is impossible to calculate the coefficient of determination, so no chart of such can be provided.

This result seems to match up with the result that was expected from this. It was expected that there would be k calls, which is only 261,911 calls to the function. This can be seen in another piece of information given from the profiler.

Evaluation

```
-----------------------------------------------
                          260000          binomCoeff(long long, long long) [10]
              0.00    0.00       1/1          main [6]
[10]     0.0    0.00    0.00       1+260000  binomCoeff(long long, long long) [10]
                          260000          binomCoeff(long long, long long) [10]
-----------------------------------------------
```

Figure 4: Table from Gprof with information on calls to function

The profiler caught 260,000 calls to the binomCoeff(`long long, long long)` function, which is truncated from the 261,911 calls expected. Moreover, following more on chart 3 the expected result and the actual result also resulted in numbers both less than $\frac{1}{100}$ of a second. A result this low was to be expected, this program ran in linear time, and it ran at a small amount of calls.

Bibliography

Soulié, Juan. “Variables and Types.” *Cplusplus.com*, 2017, www.cplusplus.com/doc/tutorial/variables/.[1]

Weisstein, Eric W. "Choose." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/Choose.html[2]

Rolfe, Timothy. “Binomial Coefficient Recursion: The Good, and The Bad and Ugly .” June 2001.[3]

Ohrt, Monte. “Binomial Coefficient Calculator.” *Binomial Coefficient Calculator*, 26 Oct. 2015, www.ohrt.com/odds/binomial.php.[4]

Engelen, Arnout. “Optimizing C/C++ Programs Using the GProf Profile.” *lf371, SoftwareDevelopment: Optimizing C/C++ Programs Using the GProf Profiler*, 1 Apr. 2004, www.linuxfocus.org/English/March2005/article371.shtml.[5]