Code Jam 2010 - Round 1B

Analysis: Your Rank is Pure

Foreword

This is a very "mathematical" problem, and solving it required thinking in very formal terms. So please bear with a lot of formulas and little text in the below explanation.

Initial approach

Let's study the process described in the problem statement. Suppose the rank of number **N** with respect to set **S** is **K**. Since **N** is the largest number in **S**, that just means the number of elements in **S** is **K**.

Then let's consider the set **S'** = **S** ∩ {1, 2, ..., **K**}. From the definition of a pure number, **K** is now pure with respect to **S'**.

Have we got a Dynamic Programming solution yet?

Does that mean that we've managed to reduce the problem for **N** to a smaller problem for **K**? Not yet: suppose we know the number of possible sets **S'** for which **K** is pure. How do we find the number of sets **S** that contain this set (and for which **N** is pure and that have **K** elements)?

In order to do that, we need to know how many numbers are there in **S'**. Suppose there are **K'** numbers in **S'**. Then the number of ways to extend this set **S'** back to **S** is the number of ways to choose **K**-**K'**-1 numbers from the set {**K+1**, **K+2**, ..., **N-1**}.

Now we have a Dynamic Programming solution!

Let's define ***Count[N, K]*** to be the number of sets **S** that are subsets of {2, 3, ..., **N**}, have **K** elements, contain number **N** and for which number **N** is pure.

The above discussion proves that **Count**[**N**, **K**] is equal to the sum over all **K'** of **Count**[**K**, **K'**] times **C**[**N**-**K**-1, **K**-**K'**-1], where ***C[A, B]*** is the number of ways to choose **B** items out of **A** (so-called combination number).

We can calculate **Count** values in increasing order of **N**. That will give us O($\mathbf{N}^2$) values to calculate, each requiring O(**N**) operations, for the total running time of O($\mathbf{N}^3$). That seems to be too slow for **N**=500 and 100 testcases.

Final observation

However, one can notice that the above algorithm calculates the answer for smaller values of **N** as well. That means we can run it just once for **N**=500, and get the answers for all testcases at once, so the total runtime will be just O($\mathbf{N}^3$), which is okay.