Kick Start 2014 - Round B

Analysis: Password Attacker

For small dataset, a simple brute-force algorithm will do. Also, since $M < 10$ and $N \leq 7$, you can simply use 32-bit integer to enumerate all potential passwords. Pseudo-code:

```
answer = 0
for password in (0 .. 10^N-1):
  condition1 = every digit in password < M
  condition2 = every element in {0 .. M-1} occurs at least once in password
  if condition1 == true and condition2 == true:
    answer = answer + 1
print(answer)
```

However, the algorithm above is not fast enough for large dataset. Assume that f(M,N) be the answer. If we ignore the condition that "All M characters should occurs in the password at least once", then the answer will be very simple -- M^N.

But we can't ignore that. f(M,N) = M^N takes some invalid passwords into count. What are they? They're the passwords formed with less than M characters. More precisely, we need to deduct all f(i,N) from M^N where $1 \leq i < M$. Is that enough?

Not yet. Note that we need to choose a set of i characters first. Consider character set S={3,5,7} with N=4. If we want to remove all i=1-character passwords, we need to remove 3333, 5555 and 7777 -- we need to pick up a subset of S with i elements. That's a classical combinatorial problem.

By putting everything together, we have f(M,N) = M^N - sigma($1 \leq i < M$)C(M,i)*f(i,N) and f(1,N) = 1. We can easily solve it by using dynamic programming.