Code Jam 2009 - Round 3

Analysis: Alphabetomials

1. Product of two sets

The problem was created during a short walk in Manhattan in a summer evening. At first there came the picture of an N by M matrix. Let *A* be a set of N words $\{A_i \mid 1 \le i \le N\}$, and *B* be a set of M words $\{B_j \mid 1 \le j \le M\}$. For any word X, let us denote X(*c*) the number of occurrences of character *c* in X. In the (i,j)-entry of the matrix, we write, say, the product A_i('a')B_j('a')B_j('b'). What is the sum of all the entries in the matrix?
One can compute the NM terms one by one. But it is easy to see, as you can sum them row by row, as well as column by column, it is nothing but the full expansion of the following expression

$$(1) \quad \sum_{X \in A,\, Y \in B} X('a')Y('a')Y('b') = \sum_{X \in A} X('a') \sum_{Y \in B} Y('a')Y('b').$$

i.e., you can compute the sum on A and B separately, then compute their product.

What if, as in our problem, instead of writing A_i('a')B_j('a')B_j('b') for the (i,j)-entry, we write Z('a')^{2}Z('b'), where Z is the concatenation of A_i and B_j? i.e., we do not care which 'a's are from the first set, and which are from the second. In this case we do not have something as simple as (1). But, we can reduce it if we *do care* which of the 'a's are from the first set. We use a somehow simplified notation (which is closer to our problem statement) -- the symbol *a* now also represent the number of 'a's in a string. For a fixed string Z that is a concatenation of words from *A* and *B*, let a_1 denote the number of 'a's contributed by the first set, and a_2 the number of 'a's contributed by the second set. For each entry in the matrix, we write a^2b, this is equal to $(a_1+a_2)^2(b_1+b_2)$. Expand it, we have

$$(2) \quad \sum_{X \in A,\, Y \in B} a^2b = \sum_{X \in A,\, Y \in B} (a_1+a_2)^2(b_1+b_2)$$
$$= \sum_{X \in A,\, Y \in B} (a_1^2b_1 + a_1^2b_2 + 2a_1a_2b_1 + 2a_1a_2b_2 + a_2^2b_1 + a_2^2b_2)$$
$$= \sum_{A,B} (a_1^2b_1) + \sum_{A,B} (a_1^2b_2) + 2\sum_{A,B} (a_1a_2b_1) +$$
$$2\sum_{A,B} (a_1a_2b_2) + \sum_{A,B} (a_2^2b_1) + \sum_{A,B} (a_2^2b_2)$$
$$= \sum_A a^2b \sum_B 1 + \sum_A a^2 \sum_B b + 2\sum_A ab \sum_B a +$$
$$2\sum_A a \sum_B ab + \sum_A b \sum_B a^2 + \sum_A 1 \sum_B a^2b$$

In the last step, for each summand, since the characters from the first set and those from the second set are nicely separated, we can apply the same reason for (1).

2. Product of ten sets

Our problem asks to compute the sum of expressions such like a^2b over the product of a dictionary set by itself up to 10 times. (Or more generally, we can think of the product of 10 sets.) The size of the product set is the huge number n^{10}. But if we use the same trick as in the previous section, write a^2b as

$$(a_1 + a_2 + ... + a_{10})^2 (b_1 + b_2 + ... + b_{10})$$

and expand it. Then we reduce the problem to at most K^D easier problems, where D is the maximal degree ($K \le 10$ and $D \le 4$ in our problem). Each of the easier problems can be solved similar as in (1).

This gives one solution to our problem. Focus on one monomial a time. We are basically choosing the origin (one of the ten sets) for each letter in the monomial, so each set gets a sub-monomial, which can be expressed as a subset of the indices of the letters in the original monomial.
We pre-compute $\sum_{X \in A} q(X)$ for each sub-monomial *q*, there are at most 16 of these. Then for each of the K^D "easier problems", the computation involves only multiplying K of the pre-computed numbers.

3. Speedups

The solution above can be easily sped up to solve for much bigger K and slightly bigger D, if one compute the summations incrementally, do the product of two sets in each step (you call it dynamic programming or not).
First, we compute the sum for A × A, not only for the monomial, but for all its sub-monomials. Then we compute the sums over A^3. Now we can view A^3 as A^2 × A. Each sub-monomial can be expanded to at most 2^D terms, and every one of them submit to the simple trick in (1). Because we already have the summation of A^2 on any sub-monomial, each of the 2^D problems can be solved in constant time. The running time of this solution is mostly dominated by $O(3^D K)$ for each monomial form the input.

Observe the last line in (2), notice that we only have one dictionary set in our problems, so we can ignore the subscripts -- A, B, etc are the same set. There are other speedups available along this line by exploring the symmetry in indices. We omit the details.

4. The theory behind it

In case you are familiar with discrete probability, you may well know that probability theory provides some powerful abstract machinery for counting. What we went through so far are just some easy exercises in probability. And solving this problem with the abstract theory in mind certainly speeds up your thinking and boosts your confidence in the solution.
Especially, we went through the important fact that the expectation of the product of K random variables equals to the product of the expectation of each of the variables, given that the K random variables are mutually independent. The interested readers may carry the formal correspondences out.