Análisis del problema 1B

La idea principal es fijar la posición i y calcular cuántos pares de palabras difieren en i y en el resto de las posiciones son iguales. Si se hace esto para todas las i $(1 \leq i \leq l)$ se resuelve el problema.

Vamos a definir a la función $f(w)$ como la concatenación de todos los caracteres de la palabra w excepto el que está en la posición i.

Entonces dos palabras w_1 y w_2 son similares si se cumple lo siguiente:

- $f(w_1) = f(w_2)$
- $w_{1i} \neq w_{2i}$

Por ejemplo, si $i = 1$:

- $f(\text{“Fax”}) = f(\text{“fax”}) = \text{“ax”}$
- “Fax” $\neq$ “f”

Por lo que “Fax” y “fax” son similares.

Para realizar el conteo es útil agrupar todas las palabras que tengan igual f en alguna estructura. Y para cada f guardar la frecuencia de los caracteres que aparecen en la posición i de todas las palabras que tengan ese valor de f. Luego la cantidad de palabras similares con un valor f se pueden calcular de la siguiente forma:

$$r = \binom{s}{2} - \sum_{j=1}^{n} \binom{v_j}{2} \tag{1}$$

Donde:

- $s = v_1 + v_2 + \ldots + v_k$
- k es el tamaño del alfabeto
- v_j es la frecuencia del j-ésimo caracter del alfabeto (que aparece en palabras con valor f)

Nota que lo que se hizo aquí fue calcular el complemento y restárselo a la cantidad de pares de palabras con valor f. Esto se hace para todos los valores de f y se va sumando a la solución.

Lo que falta es una forma eficiente de representar el valor f para cada palabra. Esto se puede lograr usando hashing. El problema con esto es que hace falta actualizar el hash para cada palabra cuando se pase de i a $i+1$. Esto se puede hacer en $\mathcal{O}(1)$.

Para ilustrar veamos un ejemplo de cómo actualizar el hash. Se tiene la palabra $s = \text{“abcd”}$, su hash es:

$$id(\text{“a”}) \cdot B^0 + id(\text{“b”}) \cdot B^1 + id(\text{“c”}) \cdot B^2 + id(\text{“d”}) \cdot B^3 \mod M \tag{2}$$

Si se quiere el resto de palabra cuando $i = 2$ se resta $id(\text{“v”}) \cdot B^1$ y se divide por B^1 el sufijo $id(\text{“c”}) \cdot B^2 + id(\text{“d”}) \cdot B^3$, luego esto se suma y obtienes el nuevo hash.

En general, si se quiere el resto de palabra para un i se resta $id(s_i) \cdot B^{i-1}$ y se divide por B^{i-1} el sufijo que comienza en la posición i, luego se suma.

En total la complejidad es $\mathcal{O}(n \cdot l \cdot (k + \log_2 n))$. El logaritmo viene de la parte de agrupar por f.

Notas

- Para esta última parte hace falta calcular inverso modular de la base que se elija.
- En mi solución yo uso doble hashing, para disminuir la probabilidad de colisiones, es lo mismo, lo que cada palabra se representa por dos hashings.
- No usen mapa, agrupen los valores f ordenando.