Genes

Queremos buscar una subsecuencia de ADN X en una secuencia Y

¿Qué tan costoso es esto según el tamaño de X e Y?

¿Cómo podemos hacer para descartar subsecuencias de Y?

Genes

X = AGCGATGCTATCTTGGGGCTATT

Y =

ACGTGACTGCTCCGCGCGTGAATTTCGATCGCGCGGATCTAGCTAG

CTAGCTGCTAGCTAGCTTCGCTATCGTAGTCGTCAGTATGATGTATAG

AATAATTAATAAAAGCGCCTGCCTAGTCGTGTGTCACGTAGTCATCG

AGCGGGCTCATACGCAGATCAGCGATGCTATCTTGGGGCTATTATG

CTAGCTATCGCCTAGCGCGATATACGCGCGCGGATTCGCTATATGCT

Función de hash

Una función de **hash** para objetos de un dominio D es:

$$h : D \rightarrow \mathbb{N}^0$$

Decimos que la función de hash tiene una **colisión** cuando

$$h(A) = h(B) \ \wedge \ A \neq B$$

Hash Genes

¿Cómo podemos usar una función de hash para saber si $X \in Y$?

¿Qué podríamos usar como función de hash para un *string*?

¿Qué tan rápido podemos resolver el problema ahora?

Hash Incremental

Si Y es una modificación de X, y conocemos $h(X)$

La función h se dice **incremental** si permite calcular $h(Y)$ a partir de $h(X)$ y la modificación que generó Y

El costo de calcularlo es lineal en el número de cambios

Genes Incrementales

¿Y si usamos una función de hash **incremental**?

¿Cuál sería la complejidad entonces?

¿Es posible resolver el problema en menos tiempo?

Sumemos las letras

Si vemos cada letra como un número, h puede ser la suma de cada letra:

$$h(X[i:j]) = x_i + x_{i+1} + \cdots + x_{j-1} + x_j$$

Teniendo $h(X[i:j])$, ¿cómo podemos calcular $h(X[i+1:j+1])$ en $O(1)$?

Interpretación numérica

¿Qué pasa si vemos la secuencia como un número?

Eso significa considerar cada letra como un dígito

¿Podemos calcular el hash de manera incremental?

Interpretación numérica

Para interpretar la secuencia de largo m como un número en base b:

$$h(X[i:j]) = x_i b^m + x_{i+1} b^{m-1} + \cdots + x_{j-1} b^2 + x_j \; b$$

Teniendo $h(X[i:j])$, ¿cómo podemos calcular $h(X[i+1:j+1])$ en $O(1)$?

Muchas colisiones

Mientras más colisiona la función, más lento es el algoritmo

En el peor caso hay que comparar todo: $O\big((n-m)\cdot m\big)$

¿Cómo podríamos garantizar 0 colisiones?

Hash Perfecto

Una función de hash es **perfecta** si no tiene colisiones

Es decir:

$$A = B \leftrightarrow h(A) = h(B)$$

Una función puede ser **perfecta** e **incremental** a la vez

Interpretación numérica

¿Qué valores deben tener las letras y b para que h sea perfecta?

El hash perfecto no es muy práctico en la vida real, ¿por qué?

Diccionarios

Queremos un diccionario en que no nos interesa el orden de los datos

Esto nos debería permitir complejidades menores a $O(\log n)$

¿Podremos guardar los datos en un arreglo? ¿En qué posicion?

Tablas de Hash

Una **tabla de hash** es un diccionario que:

- No tiene noción de orden
- Sus operaciones son $O(1)$ en promedio

Recorrido de la función

Si la tabla de hash es de tamaño m,

¿Qué pasa con los valores de h que se salen de la tabla?

Método de la División

Simplemente, usar el módulo:

$$h'(X) = h(X) \bmod m$$

Si $h(X)$ distribuye bien, entonces $h'(X)$ distribuye bien

Pero si m es potencia de 2, se pierde información sobre $h(X)$

Método de la Multiplicación

Sea A un número entre 0 y 1:

$$h'(X) = \lfloor m \cdot (A \cdot h(X) \bmod 1) \rfloor$$

Si $h(X)$ distribuye bien, entonces $h'(X)$ distribuye bien

Es más costosa, pero no depende del valor de m. Se recomienda $A = \frac{1}{\varphi}$

En resumen

Una **buena** función de hash cumple con lo siguiente:

- Incluye toda información de un objeto
- Es rápida de calcular
- Distribuye de manera uniforme
- Los hash de dos objetos parecidos deben ser muy distintos