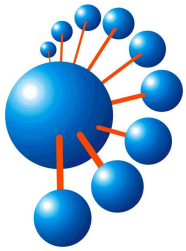
Departamento Ingeniería Informática y Ciencias de la Computación



Universidad de Concepción

Tarea 2
Utilización de Estructuras de Datos Sucintas como Sketches para HyperLogLog

---

Estudiantes: Lucas Kraemer A.
Vicente Lermanda C.
Profesora: Cecilia Hernández

22 de noviembre 2022

Índice

1. Introducción

Con anterioridad se ha tratado el problema de estimación de cardinalidad para una gran cantidad de datos en *streaming*, el cual considera un funcionamiento en linea en el que siempre se podrá obtener el valor de la estimación en un determinado momento, el cual se ira actualizando a medida que se van ingresando nuevos datos.

Para resolver el problema de estimación de cardinalidad se utilizo una representación que consiste de una estructura de datos denominada *sketch*, la cual permite realizar ir agregando nuevos datos y obtener el valor de estimación buscado.

En esta ocasión, y con la motivación de realizar este proceso de estimación con la menor cantidad de recursos posibles, es que se plantea usar la representación del sketch del algoritmo de Hyperloglog, la cual es una estructura de datos que emplea poco espacio.

2. Problema de Estimación de Cardinalidad

Dado un multiset en un Stream de m elementos $S = x_1, x_2, ..., x_m$, con $x_i \in \Sigma$ donde Σ es el alfabeto de S, es decir, el universo de elementos del stream. Se desea encontrar la cantidad de elementos distintos en S, la cual corresponde al valor $n = |\Sigma|$. Notar que $m \geq n \implies |S| \geq |\Sigma|$.

La solución propuesta es el uso del algoritmo HyperLogLog utilizando para su sketch distintas estructuras de datos con el fin de comparar el tiempo y espacio de cada una.

3. HyperLogLog

El algoritmo de HyperLogLog procesa el stream S obteniendo los elementos en tiempo real y actualizando constantemente un *sketch* de M buckets representados por el tipo de dato *unsigned char* (8 bits) y donde en cada uno de estos se almacena el máximo valor de la posición del 1 más a la derecha en $\langle bucket_p \rangle_2$ vista hasta el momento. Esto permite estimar la cantidad de elementos distintos en cualquier momento de la lectura de S.

Para cada elemento $e_i \in S$, con $i = 1 \dots |S|$, se le aplica una función hash h la cual debe garantizar una distribución uniforme en los elementos.

Luego, cada valor hash $h(e_i)$ se divide en 2 partes:

- $\boldsymbol{p}$: corresponde a los primeros $\log M$ bits de $h(e_i)$, los cuales son usados para definir a que bucket corresponde $h(e_i)$ ($buckets[p]$).

- ***b***: son los bits restantes de $h(e_i)$, desde el bit en la posición p hasta la 64. En b se busca la posición del primer bit 1 (count_leading_zeros$(b)+1$), y se actualiza el bucket tal que $bucket[p] = max(bucket[p], count\_leading\_zeros(b)+1)$.

Teniendo el sketch ya con valores, la cardinalidad se estima de la siguiente forma:

- Sea α_M, el factor de corrección en base a la cantidad de buckets M tal que $\alpha_M \approx \frac{0{,}7213}{1+\frac{1{,}079}{M}}$.
- Sea $Z = (\sum_{j=1}^{M} 2^{-bucket[j]})^{-1}$ tal que $Z \cdot M$ corresponde a la media armónica.
- La estimación de cardinalidad inicial E se calcula como $E = \alpha_M \cdot M^2 \cdot Z$.
- Finalmente se aplica una corrección a la estimación considerando a V como la cantidad de buckets iguales a 0. La estimación final E^* se calcula de la siguiente forma:

$$E^* = \begin{cases} M \log \frac{M}{V} & si \quad E < \frac{5}{2}M \text{ y } V \neq 0 \\ E & si \quad \frac{5}{2}M \leq E \leq \frac{2^{64}}{30} \\ -2^{64} \log\left(1 - \frac{E}{2^{64}}\right) & si \quad E > \frac{2^{64}}{30} \end{cases}$$

3.1. Representación de sketch

Se hará uso de algunas estructuras de datos sucintas de la librería SDSL-lite disponible en https://github.com/simongog/sdsl-lite.

Una estructura de datos sucinta es un estructura de datos que utiliza un espacio cercano a su límite teórico. Usan un espacio de $Z + o(Z)$ donde Z es el límite dado por la teoría de la información (entropía).

Con la descripción anterior podemos intuir que reduciremos aún más el espacio requerido por HLL para funcionar. Se presenta a continuación una lista con las estructuras a utilizar:

- **int_vector**: corresponde a un contenedor de enteros con cantidad fija de bits l.
- **enc_vector**: contenedor de enteros representados de manera comprimida aprovechando un orden ascendente de los datos.
- **rrr_vector**: contenedor de bits comprimido basado en entropía de orden 0.
- **sd_vector**: corresponde a un bit vector esparso. Mientras más 0 haya, menos espacio ocupa.
- **wt_int**: Wavelet Tree balanceado con representación de huffman donde se almacenan enteros.

- **wm_int**: Wavelet Tree balanceado representado de forma matricial donde se almacenan enteros.
- **wt_huff**: Wavelet Tree con representación de Huffman.

Se utiliza la estructura de *int_vector* como sketch inicial a la hora de actualizar, y en base a este, se construyen las representaciones de todas las demás estructuras.

4. Experimentación

4.1. Datos

Se utilizan datos de genómas provistos. De la larga lista de archivos se utilizan 5 para hacer la pruebas.

Cada uno de estos archivos se preprocesa de tal manera que el alfabeto de los archivos queda $\{A, C, T, G\}$.

4.1.1. Pregunta a): Entropía de orden 0 de los genómas

Entropía de orden 0 calculada como: $-\sum_{m}^{M} \frac{f_m}{M} \cdot \log \frac{f_m}{M}$ donde f_m es la frecuencia del valor del bucket m y M es la cantidad de buckets.

Genoma	Entropía$_0$	Tamaño kb
GCF_001522075.1_ASM152207v1	0.215801	7830
GCF_001522455.1_ASM152245v1	0.226562	7576
GCF_001522925.1_ASM152292v1	0.226562	8651
GCF_001522965.1_ASM152296v1	0.215801	8608
GCF_001527385.1_ASM152738v1	0.215801	8172

4.2. Estimación de HLL

A continuación se muestra la tabla con el experimento realizado. Por simplicidad a cada archivo de genóma se le asigna un valor numérico y se muestra el tamaño de cada representación junto al tiempo de ejecución de HLL y la estimación de cardinalidad.

Para mejorar la visibilidad de la tabla, las columnas que representan las estructuras son representadas numéricamente de la siguiente forma:

- **1**: *int_vector*.
- **2**: *enc_vector*.
- **3**: *wt_int usando rrr_vector*.

- **4**: *wt_huff usando sd_vector.*
- **5**: *wm_int.*

Genoma	1	2	3	4	5	[ms]	$\|\Sigma\|$
1	0.000161171	0.0014286	0.000243187	0.0065155	0.000572205	217	5080977
2	0.00016117	0.00138283	0.00024318	0.00634766	0.00057220	210	4938524
3	0.000161171	0.00145912	0.000243187	0.00673294	0.000572205	236	5614664
4	0.000161171	0.00145912	0.000243187	0.0067749	0.000572205	233	5074333
5	0.00016117	0.00144386	0.00023555	0.00685883	0.00057220	221	4765318

5. Conclusión

Se pudo notar durante la experimentación lo que teóricamente estaba planteado. Utilizar representaciones basadas en estructuras de datos compactas o sucintas reducen el tamaño requerido de cualquier esquema donde se implementen. En el caso de Hyperloglog se logra apreciar una disminución de espacio requerido, lo que es beneficial dado el contexto de grandes volúmenes de datos donde se utilizan los algoritmos de streaming.

Dado lo anterior, es importante a la hora de diseñar algoritmos considerar el uso de estas estructuras utilizadas como alternativa a las tradicionales, ya que de esta forma se apoya y se fomenta mayor investigación en el área, y software y algoritmos escalables para una gran cantidad de datos.

Como trabajo futuro se puede implementar de mejor manera algoritmos tradicionales de streaming para que estos funcionen con estructuras reducidas.