Unidad 4: Algoritmos para flujos de datos

Conteo

Gibran Fuentes-Pineda
Diapositivas basadas en las de la M. en C. Blanca Vázquez

Mayo 2021

Contando elementos distintos (1)

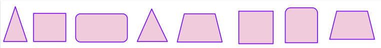
- Objetivo: contar el número de elementos distintos que han ocurrido en un flujo desde algún punto específico en el tiempo
    - Dado un flujo de elementos $x_1, x_2, ..., x_n$, encontrar el número de elementos distintos n, donde $n = |\{x_1, x_2, ..., x_n\}|$
    - También conocido como el problema de estimación de la cardinalidad
- Ocurren cuando se desea encontrar el número de elementos únicos
    - Direcciones IP que pasan a través de un router, visitantes a un sitio web, secuencias de ADN, dispositivos IoT, etc.
- Ejemplo
    - Dado el flujo a, b, a, c, d, b, d, entonces $n = |\{a, b, c, d\}| = 4$

Contando elementos distintos (2)

- Una solución simple es guardar los elementos que vayan llegando en una tabla *hash* o árbol de búsqueda.
- Cuando el número de elementos distintos es demasiado grande, sería necesario usar múltiples máquinas o guardar la estructura en disco.
- Una forma más eficiente es estimar el número de elementos distintos, evitando a través de algoritmos como el de Flajolet-Martin

Contando elementos distintos (2)

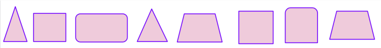
- Supongamos que tenemos un flujo de datos con m elementos



¿Cuántos elementos distintos tenemos?

Contando elementos distintos (3)

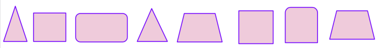
- Supongamos que tenemos un flujo de datos con m elementos



¿Cuántos elementos distintos tenemos?

- Si m es pequeña:
    - Solución: generar un diccionario
    - Memoria: $O(m)$
    - Costo computacional: $O(log(m))$ para almacenamiento y para búsqueda

- Supongamos que tenemos un flujo de datos con m elementos



¿Cuántos elementos distintos tenemos?

- Si m es grande:
  - Almacenamiento de todos los elementos sería inviable
  - Requerimientos de memoria y costo computacional muy altos
  - Sería necesario usar múltiples máquinas o guardar la estructura en disco.

Algoritmo de Flajolet–Martin (1)

- Es un algoritmo para aproximar el número de elementos distintos en un flujo de datos[1]
    - Utiliza múltiples funciones *hash* para mapear los elementos del conjunto universal a una cadena de bits
        - La cadena debe ser suficientemente grande como para que haya más posibles valores *hash* que elementos en el conjunto universal
    - Intuición: entre más elementos distintos haya en el flujo, mayor número de valores *hash* distintos deberían ocurrir yes más probable que uno de estos valores sea inusual (que termine en muchos ceros).

[1] Philippe Flajolet and G. Nigel Martin. *Probabilistic Counting Algorithms for Data Base Applications, 1985*

Algoritmo de Flajolet–Martin (2)

- La propiedad de uniformidad de las función *hash* permite estimar que la mitad de los valores de los elementos terminarán en 0, una cuarta parte de los elementos terminarán en 00, una octava parte terminarán en 000 y en general $\frac{1}{2^k}$ terminarán en k ceros
- Por lo tanto, si una función *hash* genera un valor que termina en k ceros, una estimación del número de elementos distintos es $\frac{2^k}{\phi}$, donde $\phi \approx 0.77351$ es un factor de corrección
- Se usan múltiples funciones *hash* para obtener varias estimaciones, las cuales se combinan

- Se pueden promediar estimaciones de múltiples funciones *hash*, pero la estimación es muy sensible a valores atípicos: un 0 de más y se duplica.
- La mediana es menos sensible pero solo obtiene estimaciones que son potencias de 2.
- Una mejor estrategia es agrupar las estimaciones, obtener la mediana de cada grupo y posteriormente promediar las medianas.

- Sea R el número de 0s al final de la cadena correspondiente al valor *hash* de algún elemento, se inicializa un arreglo de bits de tamaño L con ceros y se pone a 1 el bit de la posición R de cada elemento del flujo.

- Sea r la primera posición del arreglo de bits cuyo valor es cero, un estimador del número de elementos en el flujo de datos es $\frac{2^r}{\phi}$, donde $\phi \approx 0.77351$ es un factor de corrección.

Análisis de las estimaciones (1)

- La probabilidad de que un elemento dato tenga un valor *hash* que termine en al menos r 0s es 2^{-r}
- Si tenemos m elementos distintos en el flujo, la probabilidad de que ninguno tenga al menos r 0s es $(1-2^{-r})^m = ((1-2^{-r})^{2^r})^{m \cdot 2^{-r}}$. Cuando r es suficientemente grande este valor se aproxima a $(1-\epsilon)^{1/\epsilon} = \frac{1}{e}$

Análisis de las estimaciones (2)

- La probabilidad de que ninguno de los valores *hash* de los m elementos distintos termine en al menos r 0s es $e^{-m \cdot 2^{-r}}$, por lo que
  1. Si m es mucho más grande que 2^r, la probabilidad de que alguno de los valores termine en al menos r 0s se aproxima a 1
  2. Si m es mucho más pequeño que 2^r, la probabilidad de que alguno de los valores termine en al menos r 0s se aproxima a 0

- Ejemplo: dado el flujo $3, 1, 4, 1, 5, 9, 2, 6, 5$ y las funciones *hash* $h_1(x) = (2 \cdot x + 1) \mod 32$, $h_2(x) = (3 \cdot x + 7) \mod 32$ y $h_3(x) = 4 \cdot x \mod 32$, determina el número de 0s en los que termina el valor de cada elemento y estima el número de elementos distintos usando 5 bits

- Usa menos cantidad de memoria para aproximar el número de elementos únicos

- Una de las desventajas del algoritmo de Flajolet–Martin es la suposición de la generación de claves *hash* totalmente aleatorias y uniformes