CC4102 - Examen

Prof. Gonzalo Navarro

5 de Diciembre de 2022

Memoria

P1 (2 pt)

El problema de los *heavy hitters* es, dada una secuencia de n valores y enteros $k, f > 0$, encontrar k elementos que aparezcan f veces o más en la secuencia, y además reportar esas frecuencias. Si hay menos de k elementos con frecuencia $\geq f$, reportar todos esos elementos. Considere el caso en que n es demasiado grande para almacenar todos los elementos en memoria principal.

1. (0.8 pt) Diseñe y analice un algoritmo eficiente de memoria secundaria para encontrar k heavy hitters, suponiendo que tiene $M = O(k)$ espacio de memoria principal.# M espacio disponible para mem. volatil
En mem. principal no hay costo, en secundaria si.

   Solución: Algo fácil es ordenar la secuencia en disco y luego pasar por ella secuencialmente. Los elementos iguales estarán todos juntos, por lo que es fácil ir obteniendo cada uno con su multiplicidad. En RAM mantenemos los k elementos encontrados que tienen suficiente frecuencia. Esto cuesta $O(Sort(n))$ I/Os.# Ordenar la secuencia en disco (secundaria) cuesta Sort(n) I/Os

2. (1.2 pt) Suponga ahora que debe resolver el problema en modo *streaming*, es decir, sólo hay tiempo para leer los datos de disco una sola vez. Se propone un algoritmo probabilístico/aleatorizado que contabiliza sólo algunos elementos. Para cada nuevo elemento, si ya está en memoria, se incrementa su contador. Si no, con probabilidad p se lo agrega al conjunto en memoria, con su contador en 1. Se pide calcular las siguientes medidas del error de este esquema.

   (b) E(c) considera f pues representa la frecuencia real del elemento x. Son contabilizadas todas las apariciones posteriores a la primera dada en 1/p => E(c) = f -1/p

   a) Calcule la probabilidad de ignorar (es decir, ni siquiera tener en memoria) a un elemento de frecuencia f.

   b) Calcule la esperanza $E(c)$ del contador asociado a un elemento x que se contabiliza, siendo que su frecuencia real es f. Indique cómo se puede entregar una estimación de la frecuencia, cuya esperanza sea f.

   c) Calcule el uso esperado de memoria si la secuencia de largo n tiene m elementos distintos, cada uno de frecuencia n/m.

   d) Indique por qué convendría usar un p mayor y por qué un p menor.

   Solución: (a) Es necesario que las f veces decida no contabilizarlo, lo que ocurre con probabilidad $(1-p)^f$, es decir, se hace exponencialmente más difícil ignorar a los elementos más frecuentes. (b) En promedio un elemento aparece $1/p$ veces hasta que se lo empieza a contabilizar, y desde ese momento siempre se lo contabiliza. Por ello, $E(c) = f - 1/p$, y la forma de entregar un estimador con esperanza f es entregar $c + 1/p$. (c) Un elemento que ocurre n/m veces se almacena en memoria con probabilidad $1-(1-p)^{n/m}$, por lo que la esperanza de la memoria usada por los m elementos es $m(1-(1-p)^{n/m})$. (d) Un p mayor reduce la probabilidad de ignorar elementos frecuentes (y la varianza del contador, aunque no importa que no digan esto). Pero a la vez incrementa la cantidad de memoria que se usa en promedio.

   (c) Tenemos que la frecuencia es f=n/m. Luego, las veces que no se contabiliza es (1-p)^f = (1-p)^n/m
   Como queremos ver el uso esperado de memoria (cuando se cuenta) la probabilidad será 1-(1n-p)^n/m
   Esta probabilidad es para cada elemento m ∃> Esperanza de la memoria usada es m(1-(1-p)^n/m)

P2 (2 pt)

La inversa π^{-1} de una permutación π de $[n]$ cumple que $\pi^{-1}(i) = j$ sii $\pi(j) = i$. Tenemos una permutación π guardada en un arreglo $P[1..n]$, con $P[i] = \pi(i)$. El siguiente algoritmo invierte la permutación, dejándola en el mismo $P[i] = \pi^{-1}(i)$, usando sólo n bits extra.

P es un arreglo de permutaciones, en donde en P[i] se tiene pi(i)

```
INVERT(P, n)
 1  Crear un bitmap B[1..n]
 2  for i ← 1 to n do  B[i] ← 0 ;
 3  for i ← 1 to n do
 4  |  if B[i] = 0 then
 5  |  |  B[i] ← 1;
 6  |  |  k ← i;
 7  |  |  j ← P[k];
 8  |  |  while j ≠ i do
 9  |  |  |  B[j] ← 1;
10  |  |  |  ℓ ← P[j];
11  |  |  |  P[j] ← k;
12  |  |  |  k ← j;
13  |  |  |  j ← ℓ;
14  |  |  end
15  |  |  P[i] ← k;
16  |  end
17 end
```

Bitmap: En cada posición i se almacena un bit (0 o 1)

Se pide:

1. (0.5 pt) Explique la lógica del algoritmo y muestre que invierte P correctamente.
2. (1.5 pt) Demuestre que el algoritmo es de tiempo $O(n)$ usando alguna técnica de análisis amortizado.

Solución: (1) El algoritmo recorre los ciclos de la permutación invirtiendo las flechas. Recorre el arreglo y, cada nuevo elemento que encuentra, recorre su ciclo invirtiendo las flechas hasta volver a encontrar el elemento original. Marca en B los elementos de los ciclos ya procesados. (2) Por ejemplo, pueden argumentar que se trabaja a lo sumo 2 veces por cada elemento i, una vez al llegar por el for de la línea 3, y una vez cuando les llegamos por el ciclo de la línea 8. El bitmap B garantiza que sólo se recorre una vez cada elemento por ese while.

Se tiene O(n) por el costo del for, y O(n) por el costo del while (gracias al bitmap solo ejecuta una vez cada i. Con esto el algoritmo da O(n)+O(n)=O(2n)=O(n). Por tanto hacemos costo amortizado y solo cobramos O(n) al for.

P3 (2 pt)

El problema de *programación de intervalos* consiste de un conjunto de n pedidos de intervalos de tiempo $[s_i, t_i]$, $1 \leq i \leq n$, para usar un determinado recurso exclusivo (ej. una cabaña de veraneo). Se desea satisfacer la mayor cantidad de pedidos posible, pero no se puede permitir que traslapen, es decir, se busca un conjunto maximal de intervalos disjunto.

Muestre que la estrategia de elegir el intervalo más corto, eliminar a todos los que traslapan con él, e iterar, entrega una 2-aproximación.

Solución: Como es el más corto, no contiene a otros. Puede traslapar con varios que terminen/empiecen dentro de él, de los cuales el óptimo sólo podrá elegir a uno (pues todos coinciden en su s_i/t_i). Por

Tenemos los intervalos, al elegir el más corto se eliminan los que se solapan, con esto puede ocurrir que se elimina un intervalo solapado optimo (que comenzaba al terminar el primero), en el peor caso esto ocurre para cada intervalo que se añade, lo que provoca que del total de intervalos la mitad sean borrados, es decir, de cada dos intervalos se deja uno, obteniendo una 2-aproximación.

lo tanto, no elegirlo sólo puede eliminar a dos segmentos que estén en la solución óptima, dejando un segmento a cambio.

Tiempo: 3 horas