CC4102 - Control 1

Profs. Pablo Barceló y Gonzalo Navarro

18 de Octubre de 2018

P1 (3.0 pt)

1. Considere una función Booleana $f : \{0,1\}^n \to \{0,1\}$ ($n > 0$). Queremos determinar si $f(w) = 1$, para $w \in \{0,1\}^n$, haciendo sólo preguntas de la forma: ¿es cierto que el i-ésimo bit de w es un 1? Decimos que f es *evasiva* si para determinar si $f(w) = 1$ requerimos hacer exactamente n de estas preguntas (es decir, necesitamos preguntar por el valor de todos los bits de w en el peor caso).

   Demuestre que la función $f : \{0,1\}^n \to \{0,1\}$ tal que $f(w) = 1$ ssi w contiene al menos tres 0s consecutivos es evasiva para $n = 4$, pero no es evasiva para $n = 5$.

   Solución: Para $n = 4$ suponga por contradicción que hay un algoritmo que solo verifica el valor de tres posiciones y comprueba si $f(w) = 1$. Supongamos que para contestar $f(w) = 1$ para $w = 0000$ el algoritmo pregunta primero por la posición i, luego por la j, y luego por la k. Si las posiciones en $\{i,j,k\} \subseteq [1,4]$ no son contiguas, entonces hay un w' que es consistente con todas las respuestas entregadas pero no contiene tres 0s consecutivos. Por ejemplo, si $i = 3$, $j = 4$, y $k = 1$, entonces el algoritmo debería responder $f(w') = 1$ para $w' = 0100$, lo que es una contradicción. Entonces las posiciones i, j, k son contiguas, y pueden definir el conjunto $\{1,2,3\}$ o $\{2,3,4\}$. Los dos casos son simétricos, por lo que solo consideramos el primero. Entonces al ingresar la palabra $w' = 1000$ y responder $f(w') = 1$, hay al menos una posición entre $\{2,3,4\}$ que no fue verificada. Existe por tanto un w'' que es consistente con todas las respuestas entregadas en w' que no contiene tres 0s consecutivos. Esto es una contradicción.

   Para $n = 5$ primero preguntamos por el valor de la tercera posición. Si esta es 1 entonces $f(w) = 0$. En caso contrario analizamos la segunda posición. Si esta contiene un 0 entonces pasamos a analizar la cuarta posición. Si esta contiene un 0 entonces $f(w) = 1$. Pero si la cuarta posición contiene un 1 entonces verificamos la primera. Si esta es un 0 entonces $f(w) = 1$, si es un 1 entonces $f(w) = 0$. Por otro lado, si la segunda posición es un 1 pasamos a verificar la cuarta. Si esta vale 1 entonces $f(w) = 0$. En caso contrario, verificamos la quinta posición. Si esta vale 0 entonces $f(w) = 1$; en caso contrario, $f(w) = 0$. Para cualquier entrada w este algoritmo verifica a lo más cuatro posiciones.

2. Una *franja* en el plano es el área infinita que se sitúa entre dos segmentos finitos paralelos. El problema Franjas-cubren-rectángulo toma como entrada un conjunto de n franjas (dadas por los segmentos finitos que las definen) y un rectángulo, y determina si la unión de las franjas contiene al rectángulo. Igualmente se define Franjas-cubren-triángulo. Demuestre que si cualquiera de estos problemas pudiera resolverse en tiempo $O(n^{1,5})$ el otro también lo sería.

   Solución: Suponga primero que podemos resolver Franjas-cubren-triángulo en tiempo $O(n^{1,5})$ y considere una entrada de Franjas-cubren-rectángulo. Divida al rectángulo por su mitad en dos triángulos, y luego resuelva Franjas-cubren-triángulo para cada uno de ellos. Esta reducción toma tiempo lineal, por lo que Franjas-cubren-rectángulo puede resolverse en tiempo $O(n^{1,5})$. Asuma, al contrario que podemos resolver FRANJAS-CUBREN-RECTÁNGULO en tiempo $O(n^{1,5})$ y considere una entrada de FRANJAS-CUBREN-TRIÁNGULO. Inscriba el triángulo en un rectángulo y cubra las tres áreas de diferencia entre el rectángulo y el triángulo con 3 franjas. Resuelva el problema FRANJAS-CUBREN-RECTÁNGULO para este nuevo grupo de franjas y rectángulo. Esta reducción toma tiempo lineal, por lo que FRANJAS-CUBREN-TRIÁNGULO puede resolverse en tiempo $O(n^{1,5})$.

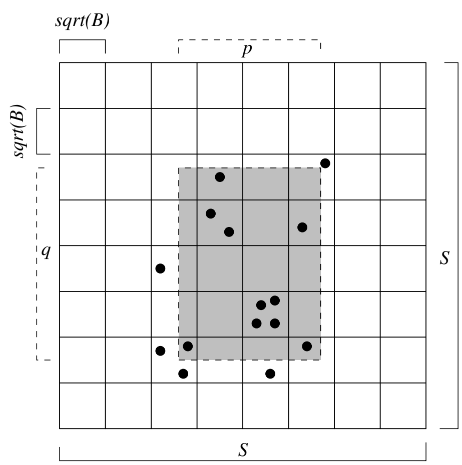
P2 (3.0 pt)

Considere la siguiente estructura estática para almacenar N puntos en una grilla de $[1..S]\times[1..S]$ en memoria secundaria (cada celda contiene a lo sumo un punto). La grilla se divide en grupos de largo $\sqrt{B}$ en ambas dimensiones, formando cuadrados de B celdas de la grilla (es decir, los cuadrados son de $\sqrt{B}\times\sqrt{B}$).

Las coordenadas (x, y) de los N puntos se guardan contiguas en un archivo P, con todos los puntos de cada cuadrado juntos, y consecutivos a los del cuadrado de la izquierda (es decir, primero todos los puntos del cuadrado $[1..\sqrt{B}]\times[1..\sqrt{B}]$, luego los de $[\sqrt{B}+1..2\sqrt{B}]\times[1..\sqrt{B}]$, y así hasta completar la primera fila de cuadrados, luego los de la segunda fila, partiendo por $[1..\sqrt{B}]\times[\sqrt{B}+1..2\sqrt{B}]$, etc.). El archivo P ocupa $\lceil N/B\rceil$ páginas de disco.

Asimismo, en un archivo C se guarda, por cada cuadrado, la posición de P donde comienza la lista de los puntos de ese cuadrado. El archivo C ocupa $\lceil S^2/B\rceil$ páginas de disco.

Sobre esta estructura haremos consultas, que serán rectángulos $[x, x+p]\times[y, y+q]$, y querremos recuperar todos los puntos dentro de él. Vea la figura, donde la consulta es el rectángulo gris.



1. Muestre cómo listar todos los *occ* puntos que caen dentro de un rectángulo $[x, x+p]\times[y, y+q]$ en $O(1+(p+q)/\sqrt{B}+occ/B)$ I/Os.

2. Muestre cómo contar esos puntos en tiempo $O(1+(p+q)/\sqrt{B})$ I/Os.

3. Simplifique la solución para almacenar N enteros en $[1..S]$ de manera de encontrar todos los puntos en un rango $[x..x+p]$ en tiempo $O(p/B)$. Esto es $O(1)$ si $p = O(B)$. ¿Por qué podemos romper la cota inferior de $\Omega(\log_B N)$? ¿Cuál es el costo que, en la práctica, tendría usar esta solución para almacenar enteros? ¿Qué se podría hacer para mitigar este costo?

Solución:

1. Se escanean los puntos de las celdas que contienen al perímetro de la consulta. Estas son a lo más $2(p+q)/\sqrt{B}+O(1)$. Cada celda se procesa en $O(1)$ I/Os leyendo su puntero de C, yendo a P, y leyendo sus puntos, pues éstos son a lo más B y por lo tanto caben en $O(1)$ páginas de disco. El costo total es entonces $O(1+(p+q)/\sqrt{B})$. Las celdas internas, en cambio, se distribuyen en $O(q/\sqrt{B})$ filas. Para cada fila i se toman de C el puntero inicial y final en P, y los occ_i puntos se leen de P en $O(1+occ_i/B)$ I/Os, pues están todos contiguos. Eso suma la segunda parte del costo, $O(q/\sqrt{B}+occ/B)$.

2. Similar, sólo que los puntos internos no necesitan escanearse de P sino que basta contar cuántos son restando los punteros inicial y final a P de cada fila de celdas interna al área de la consulta. Desaparece entonces la componente $O(occ/B)$.

3. Se corta el arreglo en buckets de largo B, de modo que toda query $[x..x+p]$ cae en $O(p/B)$ buckets consecutivos. Como los puntos de cada bucket caben en $O(1)$ páginas, el costo es $O(p/B)$. Bastaría incluso que tuvieran un arreglo de S elementos, cada uno indicando el punto que tiene o nulo si no lo tiene. Rompemos la cota inferior porque no usamos comparaciones. El problema es que usamos $O(S)$ espacio en disco en vez del $O(N)$ que basta para almacenar los N enteros, y podría ser $N \ll S$. Una forma de reducir el espacio sería usar un hash que mapeara los valores $[1..S]$ a $[1..O(N)]$: no podríamos reportar $O(B)$ puntos en $O(1)$ I/Os pero sí al menos uno, lo cual aún rompe la cota. Aquí pueden ser bien creativos y lo que importa es que muestren que saben lo que están haciendo.

Tiempo: 2.0 horas | Con una hoja de apuntes | Responder en hojas separadas