CC4102 - Examen

Profs. Pablo Barceló y Gonzalo Navarro

22 de Diciembre de 2018

P1 (1.5 pts)

Dado un conjunto de números $X = \{x_1, x_2, \ldots, x_m\}$ con $x_i \in [1..u]$ y $x_i < x_{i+1}$ para todo $1 \leq i < m$, se define el *predecesor* de y como $pred(y) = \max\{x_i \in X, x_i \leq y\}$. El *problema del predecesor* es el de preprocesar X para responder consultas *pred* eficientemente.

Podemos calcular *pred* en tiempo constante guardando todas las respuestas en un arreglo $P[1..u]$, pero en muchas aplicaciones u es demasiado grande para los m valores x_i que realmente se necesita almacenar. Si exigimos que el espacio total que usamos sea polinomial en m, entonces existen varias cotas inferiores para el problema del predecesor, del estilo $\Omega(\log\log u)$ (son más complicadas, pero la tomaremos así por simplicidad).

Sabiendo esto, considere el problema de calcular *rank*. Dado un vector de bits $B[1..n]$, con k 1s, se desea preprocesarlo para responder eficientemente la consulta $rank(B, i) = |\{1 \leq j \leq i, B[j] = 1\}|$. Se puede calcular *rank* en tiempo constante usando $O(n)$ espacio.

1. Demuestre que, si queremos usar $O(k)$ espacio, es decir proporcional a la cantidad de 1s del arreglo, entonces *rank* no se puede resolver en tiempo menor que $\Omega(\log\log n)$.
2. Demuestre que, si se quiere calcular $rank(B, i)$ solamente para las posiciones i donde $B[i] = 1$, entonces sí es posible hacerlo en tiempo constante con espacio $O(k)$.

Solución (sketch):

1. Deben reducir *pred* a *rank*, guardando en un arreglo $X[1..m]$ los valores x_i para poder responder $pred(y) = X[rank(B, y)]$. Lo importante es que reduzcan correctamente.
2. Pueden conseguirlo usando hashing perfecto sobre los valores x_i.

P2 (1.5 pts)

Se quiere implementar una cola común, con las operaciones *enqueue* y *dequeue*, más la operación $findmin$, que en cualquier momento indica el mínimo de los elementos que están en la cola. Se propone la siguiente estructura:

1. Una cola normal Q, que implementa *enqueue* y *dequeue*.
2. Una lista doblemente enlazada M, que contiene los *mínimos de izquierda a derecha* de la cola. Es decir, si la cola actual tiene los elementos $x_1, \ldots, x_n$ (donde x_1 es el elemento insertado más recientemente), entonces x_i estará en M si $x_i < x_j$ para todo $1 \leq j < i$. La lista M contiene los elementos x_i así escogidos, con i creciente (y valores decrecientes) de izquierda a derecha: $m_1, m_2, \ldots$.

Por ejemplo, si la cola en un momento contiene $\langle x_1, \ldots, x_7\rangle = \langle 8, 6, 9, 3, 5, 9, 4\rangle$, entonces M contiene $\langle m_1, m_2, m_3\rangle = \langle 8, 6, 3\rangle$. Las operaciones se realizan de la siguiente manera:

- *enqueue*(x) agrega x a la cola, como un nuevo elemento x_0 anterior a x_1. Luego, se actualiza el invariante de M: se eliminan todos los elementos m_j tal que $x_0 < m_j$. Finalmente, se agrega x_0 al comienzo de M.
- *dequeue*( ) elimina el último elemento de la cola, y si coincide con el último elemento de M, también lo elimina de M.
- $findmin$( ) entrega el último elemento de M.

Se pide:

1. Demuestre que $findmin$ entrega correctamente el mínimo actual de la cola.
2. Analice el costo amortizado de las operaciones, si se parte de una cola vacía.

Solución (sketch):

1. Deben notar que el último mínimo de izquierda a derecha es el mínimo total, y luego argumentar que las operaciones mantienen correctamente la lista M.
2. Lo único no constante es la eliminación que se hace en *enqueue*, pero como esos elementos alguna vez se insertaron en M, le podemos cobrar 2 a la inserción de x_0 para que después, cuando lo saquemos, sea gratis. O podemos usar función potencial con $\Phi = |M|$.

P3 (1.5 pts)

Construya un algoritmo paralelo del tipo CRCW que compute el máximo de n elementos con $n^{3/2}$ procesadores en un número constante de pasos.

Solución (sketch):

De clases sabemos que en un número constante de paso podemos computar el máximo de n elementos en un número constante de pasos si tenemos n^2 procesadores. Podemos entonces dividir la entrada en subarreglos de tamaño $\sqrt{n}$, computar el máximo de cada uno de estos en paralelo con n procesadores para cada subarreglo en tiempo $O(1)$ (y, por tanto, con $n^{3/2}$ procesadores en total), y luego computar el máximo de los $\sqrt{n}$ máximos en tiempo constante con n procesadores.

P4 (1.5 pts)

Suponga que tenemos un vector de bits generado al azar de largo $n > 3$. Es decir, el vector puede verse como una secuencia $X_1X_2 \dots X_n$ de variables aleatorias independientemente generadas, las que pueden tomar valor 0 o 1 con igual probabilidad ($n > 3$). Queremos ahora ordenar este vector en paralelo siguiendo una secuencia de pasos $i = 0, \dots, t$ que reorganizan sus elementos. Definimos entonces X_j^i, para $0 \leq i \leq t$ y $1 \leq j < n$, como el símbolo en la posición j en el paso i, asumiendo que $X_j^0 = X_j$. En el paso i se realiza lo siguiente para cada $1 \leq j < n-1$: Si $X_j^i = 1$ y $X_{j+1}^i = 0$, entonces realizamos un *swap*, es decir, definimos $X_j^{i+1} = 0$ y $X_{j+1}^{i+1} = 1$.

Sea Y_i el número de swaps realizados en el paso i, para $0 \leq i \leq t$. Determine $E(Y_0)$ y $E(Y_1)$.

Solución (sketch):

Partimos con Y_0. Tenemos que $Y_0 = \sum_{1 \leq j < n} Z_j^0$, donde Z_j^0 es variable indicatoria que señala si hubo swap en posición j en el primer paso. Por tanto, $E(Y_0) = (n-1)Pr(Z_1^0 = 1) = (n-1)/4$.

Para Y_1 hay que ser más cuidadosos, ya que ahora hay más casos que analizar. Como antes, tenemos que $Y_1 = \sum_{1 \leq j < n} Z_j^1$, donde Z_j^1 es variable indicatoria que señala si hubo swap en posición j en el segundo paso. Pero ahora $Pr(Z_j^1 = 1)$ depende de j. Analizamos entonces $Pr(Z_j^1 = 1)$ por casos:

- $1 < j < n-1$. Entonces hay swap en posición j en el segundo paso si $X_{j-1}X_jX_{j+1}X_{j+2}$ es uno de los siguientes strings: 0110, 1000, 1001, 1010, y 1110. Por tanto, $Pr(Z_j^1 = 1)$, para $1 < j < n-1$, es $5/16$.
- $j = 1$. Hay un swap en en posición 1 en el segundo paso ssi $X_1X_2X_3 = 110$, lo que ocurre con probabilidad $1/8$.
- $j = n-1$. Hay un swap en en posición $n-1$ en el segundo paso ssi $X_{n-3}X_{n-2}X_{n-1}X_n$ es 0100, 0110, 1000, 1010, 1001, 1110, y 1110, lo que ocurre con probabilidad $7/16$.

En consecuencia, $E(Y_1) = 5(n-3)/16 + 1/8 + 7/16 = 5n/3 - 3/8$.

Tiempo: 3.0 horas Con una hoja de apuntes Responder en hojas separadas