

Auxiliar 3 - “Jamais Vu”

Profesor: Gonzalo Navarro

Auxiliar: ~~Manuel~~ Ariel Cáceres Reyes

28 de Agosto del 2017

Cotas inferiores

P1. Mediana

Considere el problema de encontrar la mediana de un arreglo desordenado, $A[1 \dots n]$ (n impar). En un modelo basado en comparaciones:

- a) Muestre que el problema es $\mathcal{O}(n)$
- b) Muestre que $\frac{3(n-1)}{2}$ es una cota inferior del problema

Memoria Externa

P2. Lista Enlazada

Cree una implementación de una *Lista Enlazada* que permita realizar las operaciones de:

- Inserción
- Borrado
- Concatenación

en $\mathcal{O}(1)$ operaciones I/O. Además debe permitir búsqueda de un elemento en $\mathcal{O}(n)$

P3. BFS

Considere un almacenamiento en disco para grafos no dirigidos $G(V, E)$, con $|V| = n$ y $|E| = e$, usando un archivo ordenado para la lista de adyacencia de cada nodo. Escriba el pseudocódigo de un algoritmo que implemente *recorrido en amplitud* (o BFS) sobre el grafo y haga $\mathcal{O}(n + \frac{e}{B} \log_m \frac{e}{B})$ operaciones I/O.

“There’s an opposite to Deja vu. They call it jamais vu. It’s when you meet the same people or visit places, again and again, but each time is the first. Everybody is a stranger. Nothing is ever familiar”

Chuck Palahniuk

P1. a) Un algoritmo con peor caso lineal es el algoritmo *Mediana de medianas*. Este particiona el arreglo a partir de un pivote “bien escogido”

```

1 Funcion select( $A, k$ )
2   Dividir  $A$  en  $\frac{n}{5}$  grupos de 5 elementos
3   Ordenar cada grupo en  $\mathcal{O}(1)$  y todos en  $\mathcal{O}(n)$ 
4    $C \leftarrow \{\text{Medianas de los grupos}\}$ , en  $\mathcal{O}(n)$ 
5    $x \leftarrow \text{select}(C, n/10)$ , que sería la mediana de las medianas
6    $i \leftarrow \text{particionar}(A, x)$ , en  $\mathcal{O}(n)$ 
7   if  $i == k$  then
8     | return  $A[i]$ 
9   end
10  else if  $i > k$  then
11    | return select( $A[1 : i - 1], k$ )
12  end
13  else
14    | return select( $A[i + 1 : n], k - i$ )
15  end

```

La gracia de escoger nuestro pivote x como la mediana de las medianas de los grupos es, que puedo asegurar:

- x es mayor a al menos $3(n/10 - 2)$ elementos, pues como es la mediana de un grupo de tamaño $n/5$, es mayor que $n/10 - 1$ elementos los cuales son medianas de sus grupos y por lo tanto cada uno mayor a 2 elementos.
- x es menor a al menos $3(n/10 - 2)$ elementos, por un argumento análogo al anterior.

Y por lo tanto el segundo llamado recursivo que se hace se puede asegurar que tendrá tamaño de a lo más $7n/10 + 6$. Con la ecuación de recurrencia que mide el costo del algoritmo queda:

$$T(n) = \begin{cases} \mathcal{O}(1) & n \leq n_0 \\ \mathcal{O}(n) + T(n/5) + T(7n/10 + 6) & \text{si no} \end{cases}$$

que puede ser demostrado por inducción que es $\mathcal{O}(n)$

b) Nombremos m como la mediana del conjunto.

Definición 1. Diremos que la comparación de un elemento x con otro y fue crucial para x si:

- Es la primera vez que se comparan
- Sabemos que $y \leq m$ y el resultado de la comparación fue $x < y$
- Sabemos que $y \geq m$ y el resultado de la comparación fue $x > y$

“There’s an opposite to Deja vu. They call it jamais vu. It’s when you meet the same people or visit places, again and again, but each time is the first. Everybody is a stranger. Nothing is ever familiar”

Chuck Palahniuk

Veamos primero que todo algoritmo debe hacer al menos $n - 1$ comparaciones cruciales, pues ni no fuese así podríamos encontrar un elemento x distinto de la mediana con al cual no se le realizó una comparación crucial y por lo tanto no podemos saber si este es menor o mayor a la mediana, alterando la verdadera mediana a nuestro parecer.

Por otro lado, definamos un adversario que responde a las comparaciones solicitadas por el algoritmo según el análisis de las siguientes cantidades:

- a : número de elementos nunca comparados
- b : número de elementos comparados, a los cuales se les asignó un valor mayor a m .
- c : número de elementos comparados, a los cuales se les asignó un valor mayor a m .

El adversario entonces, va asignando valores a los elementos a medida que el algoritmo le va preguntando comparaciones según la siguiente tabla.

	a	b	c
a	$(a - 2, b + 1, c + 1)$	$(a - 1, b, c + 1)$	$(a - 1, b + 1, c)$
b		(a, b, c)	(a, b, c)
c			(a, b, c)

Es decir, cuando se quiere comparar un elemento nunca antes comparado:

- Con otro nunca antes comparados, uno de ellos se hace mayor y otro menor que la mediana.
- Con uno mayor que la mediana, este se hace menor.
- Con uno menor que la mediana, este se hace mayor.

De este modo el adversario puede responder consistentemente a las comparaciones excepto cuando b (o c) alcanza $\frac{n-1}{2}$ pues en tal caso el adversario no puede seguir asignando a los elementos, valores mayores (menores) a la mediana. En este caso el adversario simplemente le asigna a todos los elementos restantes valores menores (mayores) a la mediana y responde según ellos.

De cualquier modo, el algoritmo necesitará hacer al menos $\frac{n-1}{2}$ comparaciones del tipo de la primera fila de la tabla, todas las cuales son no cruciales.

P2. Tendremos nodos consecutivos en un bloque de memoria externa y además el último nodo de un bloque será el anterior del primero de su bloque consecutivo.
Mantendremos el siguiente invariante:

Hay más de $\frac{2}{3}B$ elementos en cada par de bloques consecutivos.

Cada vez que el invariante sea roto lo **reparamos** haciendo *merge* del par de bloques que lo rompen.


De este modo las operaciones nos quedan como sigue:

“There’s an opposite to Deja vu. They call it jamais vu. It’s when you meet the same people or visit places, again and again, but each time is the first. Everybody is a stranger. Nothing is ever familiar”

Chuck Palahniuk

- (a) **Inserción:** Simplemente vamos al bloque correspondiente y lo ponemos ahí a costo $\mathcal{O}(1)$. En caso que el bloque donde queremos ponerlo este lleno, revisamos si alguno de sus hermanos tiene espacio disponible, si es así hacemos un *shift* de la lista hacia ese hermano a costo $\mathcal{O}(1)$. En caso que los tres bloques estén llenos hacemos *split* del bloque de inserción en 2 bloques ocupados hasta $\frac{B}{2}$ a costo $\mathcal{O}(1)$.
 - (b) **Eliminación:** Vamos al bloque y eliminamos el nodo correspondiente, en el caso que rompamos el invariantes lo **reparamos**.
 - (c) **Concatenación:** Asociamos el último bloque de una de las listas con el primero de la otra, en el caso que el par *primero/último* rompa el invariante lo **reparamos**.
 - (d) **Búsqueda:** Hacemos una búsqueda secuencial, que en el peor caso nos requerirá buscar en todos los bloques de la lista. Dado el invariante el número de bloques de la lista será siempre $\leq N/(B/3) = \mathcal{O}(n)$
- P3.** Si s es el vértice de origen de BFS, definimos $FRONT(t)$ como el conjunto de los vértices que están a distancia t de s en el grafo.

En BFS en memoria principal vamos obteniendo $FRONT(t)$ a partir de los vecinos $FRONT(t-1)$ que no han sido marcados como visitados. El trasladar esta idea a memoria secundaria resulta en un algoritmo con $\mathcal{O}(n + e)$ llamadas I/O.

Idea.  Los vértices de $FRONT(t)$ son vecinos de aquellos en $FRONT(t-1)$, pero que no están en $FRONT(t-1)$ ni en $FRONT(t-2)$. Esto pues, para saber si un vecino de $FRONT(t-1)$ ha sido visitado solo basta mirar que no se encuentre en $FRONT(t-1) \cup FRONT(t-2)$, pues, por definición, los vecinos de $FRONT(t-1)$ no pueden estar, por ejemplo, en $FRONT(t-3)$.

El siguiente pseudocódigo ejecuta BFS con esta implementando la idea anterior.

Comentarios importantes:

- Paso 1 : Creación de la lista a costo $\mathcal{O}(n/B)$
- Paso 4 : El BFS continúa mientras se no se haya terminado de recorrer una componente conexa o mientras hayan componentes conexas por recorrer
- Pasos 5-6: En caso de visitar una nueva componente conexa se establece un origen de los posibles que quedan en U
- Paso 9: *Vecinos* une las listas de adyacencia de los vértices a costo $\mathcal{O}(|FRONT(t-1)| + |\mathcal{N}(FRONT(t-1))|/B)$ y luego de todas las iteraciones a costo total $\mathcal{O}(n + e/B)$

“There’s an opposite to Deja vu. They call it jamais vu. It’s when you meet the same people or visit places, again and again, but each time is the first. Everybody is a stranger. Nothing is ever familiar”

Chuck Palahniuk

```

1  Crear Lista Enlazada  $U \leftarrow (1 \rightarrow 2 \rightarrow \dots \rightarrow n)$ 
2   $FRONT(-2) \leftarrow FRONT(-1) \leftarrow \emptyset$ 
3   $t \leftarrow 0$ 
4  while  $FRONT(t-1) \neq \emptyset \vee U \neq \emptyset$  do
5      if  $FRONT(t-1) == \emptyset$  then
6          |  $FRONT(t) \leftarrow \{U.removeFirst\}$ 
7      end
8      else
9          |  $A(t) \leftarrow Vecinos(FRONT(t-1))$ 
10         |  $A(t).removeDuplicates$ 
11         |  $FRONT(t) \leftarrow A(t) \setminus (FRONT(t-1) \cup FRONT(t-2))$ 
12     end
13     for  $v \in FRONT(t)$  do
14         |  $U.remove(v)$ 
15     end
16      $++ t$ 
17 end
    
```

- Paso 10: Esto lo podemos hacer ordenando $A(t)$ y luego haciendo un escaneo sobre la lista eliminando los duplicados todo esto a costo $\mathcal{O}(|A(t)|/B \log_m |A(t)|/B)$ y luego de todas las iteraciones podemos acotar el costo total por $\mathcal{O}(\frac{e}{B} \log_m \frac{e}{B})$
- Paso 11: Esto lo podemos hacer con un escaneo paralelo de las tres listas a costo total (de todas las iteraciones) $\mathcal{O}(e/B)$
- Paso 13: Este loop incurre en un costo total de $\mathcal{O}(n)$

Con lo que el costo del algoritmo es $\mathcal{O}(n + \frac{e}{B} \log_m \frac{e}{B})$

“There’s an opposite to Deja vu. They call it jamais vu. It’s when you meet the same people or visit places, again and again, but each time is the first. Everybody is a stranger. Nothing is ever familiar”

Chuck Palahniuk