Algoritmos y Estructura de Datos I
Taller de Searching

Departamento de Computación, FCEyN, Universidad de Buenos Aires.

4 de Junio de 2018

Algoritmos de Búsqueda

Vamos a analizar algoritmos que resuelvan el siguiente problema (o derivados):

proc *contiene*(in s : $seq\langle\mathbb{Z}\rangle$, in x : $\mathbb{Z}$, out *result* : Bool){
  Pre $\{True\}$
  Post $\{result = true \leftrightarrow (\exists i : \mathbb{Z})(0 \leq i < |s| \land_L s[i] = x)\}$
}

Búsqueda Lineal

- Recorremos la lista de una punta a la otra buscando el elemento.

s[0]	s[1]	s[2]	s[3]	s[4]	...	s[$\|s\| - 1$]

Búsqueda Lineal

- Recorremos la lista de una punta a la otra buscando el elemento.

s[0]	s[1]	s[2]	s[3]	s[4]	$\ldots$	s[$\|s\|-1$]
$= x?$						
$\uparrow$						
i						

Búsqueda Lineal

- Recorremos la lista de una punta a la otra buscando el elemento.

s[0]	s[1]	s[2]	s[3]	s[4]	$\ldots$	s[$\lvert s \rvert - 1$]
$\neq x$	$= x?$					
	$\uparrow$					
	i					

Búsqueda Lineal

- Recorremos la lista de una punta a la otra buscando el elemento.

s[0]	s[1]	s[2]	s[3]	s[4]	$\ldots$	s[$\vert s\vert - 1$]
$\neq x$	$\neq x$	$= x?$				
		$\uparrow$				
		i				

Búsqueda Lineal

- Recorremos la lista de una punta a la otra buscando el elemento.

s[0]	s[1]	s[2]	s[3]	s[4]	...	s[$\|s\|-1$]
$\neq x$	$\neq x$	$\neq x$	$= x?$			
			$\uparrow$			
			i			

Búsqueda Lineal

- Recorremos la lista de una punta a la otra buscando el elemento.

s[0]	s[1]	s[2]	s[3]	s[4]	$\ldots$	s[$\lvert s \rvert - 1$]
$\neq x$	$\neq x$	$\neq x$	$\neq x$			$= x?$
						$\uparrow$
						i

Búsqueda Lineal

- Recorremos la lista de una punta a la otra buscando el elemento.

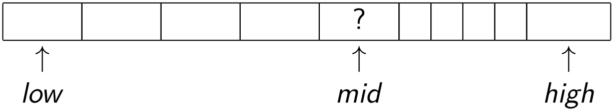
s[0]	s[1]	s[2]	s[3]	s[4]	...	s[$\|s\|-1$]
$\neq x$	$\neq x$	$\neq x$	$\neq x$			$\neq x$

Eficiencia

En el peor caso recorremos toda la lista, es decir, hacemos $|s|$ iteraciones.

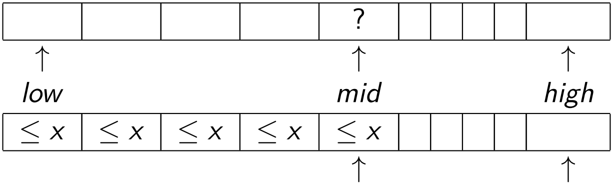
Búsqueda binaria

- Dado un arreglo *ordenado*, comparamos el elemento a buscar con el de la mitad de la secuencia. Si es mayor, buscamos a la derecha. Si es menor, a la izquierda.



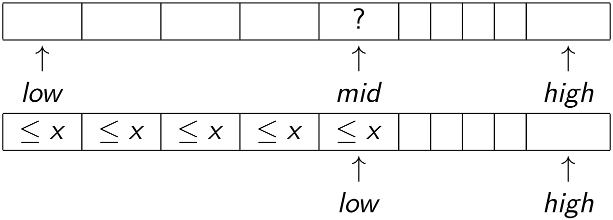
Búsqueda binaria

- Dado un arreglo *ordenado*, comparamos el elemento a buscar con el de la mitad de la secuencia. Si es mayor, buscamos a la derecha. Si es menor, a la izquierda.



Búsqueda binaria

- Dado un arreglo *ordenado*, comparamos el elemento a buscar con el de la mitad de la secuencia. Si es mayor, buscamos a la derecha. Si es menor, a la izquierda.



Eficiencia

- ¿Cuántas iteraciones realiza el ciclo (en peor caso)?

Eficiencia

- ¿Cuántas iteraciones realiza el ciclo (en peor caso)?

Número de iteración	*high* − *low*
0	$\lvert s\rvert - 1$
1	$\cong (\lvert s\rvert - 1)/2$
2	$\cong (\lvert s\rvert - 1)/4$
3	$\cong (\lvert s\rvert - 1)/8$
$\vdots$	$\vdots$
t	$\cong (\lvert s\rvert - 1)/2^t$

Eficiencia

- ¿Cuántas iteraciones realiza el ciclo (en peor caso)?

Número de iteración	*high* − *low*
0	$\|s\| - 1$
1	$\cong (\|s\| - 1)/2$
2	$\cong (\|s\| - 1)/4$
3	$\cong (\|s\| - 1)/8$
$\vdots$	$\vdots$
t	$\cong (\|s\| - 1)/2^t$

- Sea t la cantidad de iteraciones necesarias para llegar a $high - low = 1$.

$1 \cong (|s|-1)/2^t$ entonces $2^t \cong |s|-1$ entonces $t \cong \log_2(|s|-1)$.

Jump Search

Descripción informal

- Dado un arreglo ordenado v, lo recorremos saltando de a bloques de m elementos.
- Paramos cuando el elemento a buscar es menor a $v[k \cdot m]$, para algún k.
- Luego hacemos una búsqueda lineal entre $v[(k-1) \cdot m]$ y $v[k \cdot m]$.

Jump Search

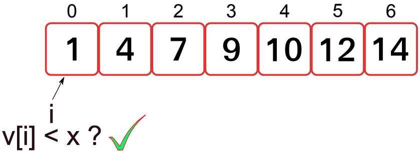
Descripción informal

- Dado un arreglo ordenado v, lo recorremos saltando de a bloques de m elementos.
- Paramos cuando el elemento a buscar es menor a $v[k \cdot m]$, para algún k.
- Luego hacemos una búsqueda lineal entre $v[(k-1) \cdot m]$ y $v[k \cdot m]$.

Ejemplo

Buscar el elemento x=10 en el siguiente array.

Tomamos $m = 3$



Jump Search

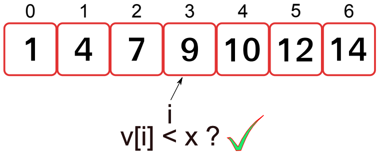
Descripción informal

- Dado un arreglo ordenado v, lo recorremos saltando de a bloques de m elementos.
- Paramos cuando el elemento a buscar es menor a $v[k \cdot m]$, para algún k.
- Luego hacemos una búsqueda lineal entre $v[(k-1) \cdot m]$ y $v[k \cdot m]$.

Ejemplo

Buscar el elemento x=10 en el siguiente array.

Tomamos $m = 3$



Jump Search

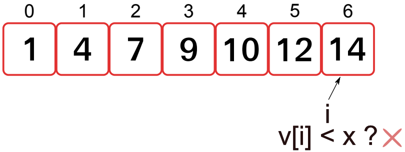
Descripción informal

- Dado un arreglo ordenado v, lo recorremos saltando de a bloques de m elementos.
- Paramos cuando el elemento a buscar es menor a $v[k \cdot m]$, para algún k.
- Luego hacemos una búsqueda lineal entre $v[(k-1) \cdot m]$ y $v[k \cdot m]$.

Ejemplo

Buscar el elemento x=10 en el siguiente array.

Tomamos $m = 3$



Jump Search

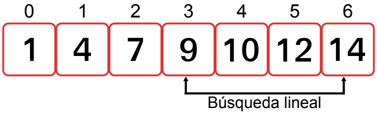
Descripción informal

- Dado un arreglo ordenado v, lo recorremos saltando de a bloques de m elementos.
- Paramos cuando el elemento a buscar es menor a $v[k \cdot m]$, para algún k.
- Luego hacemos una búsqueda lineal entre $v[(k-1) \cdot m]$ y $v[k \cdot m]$.

Ejemplo

Buscar el elemento x=10 en el siguiente array.

Tomamos $m = 3$



Jump Search

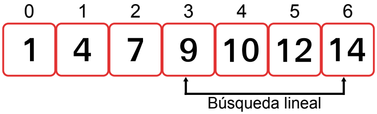
Descripción informal

- Dado un arreglo ordenado v, lo recorremos saltando de a bloques de m elementos.
- Paramos cuando el elemento a buscar es menor a $v[k \cdot m]$, para algún k.
- Luego hacemos una búsqueda lineal entre $v[(k-1) \cdot m]$ y $v[k \cdot m]$.

Ejemplo

Buscar el elemento x=10 en el siguiente array.

Tomamos $m = 3$



Código

Ustedes!

Eficiencia (1)

- ¿Cuántas iteraciones vamos a hacer en el peor caso?

Eficiencia (1)

- ¿Cuántas iteraciones vamos a hacer en el peor caso?
- En la iésima iteración hicimos $i \cdot m$ cantidad de iteraciones. En el peor caso, hacemos $\frac{n}{m}$ iteraciones.

Eficiencia (1)

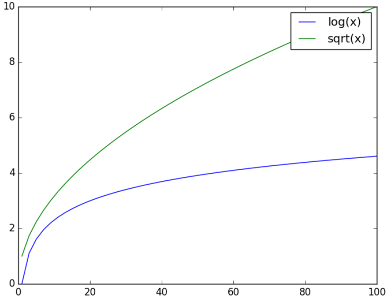
- ¿Cuántas iteraciones vamos a hacer en el peor caso?
- En la iésima iteración hicimos $i \cdot m$ cantidad de iteraciones. En el peor caso, hacemos $\frac{n}{m}$ iteraciones.
- Por último, tenemos $m - 1$ iteraciones más provenientes de la búsqueda lineal dentro del bloque.
- Con $m = \sqrt{n}$ se obtiene el mínimo valor de la función $\frac{n}{m} + m$. Por lo tanto, $m = \sqrt{n}$ es el tamaño de bloque óptimo.
- De esta manera, la cantidad de iteraciones nos queda *del orden de* $\sqrt{n}$

Eficiencia (2)

- Entonces, ¿ganamos o perdemos contra búsqueda binaria?

Eficiencia (2)

- Entonces, ¿ganamos o perdemos contra búsqueda binaria?



Con este algoritmo, a diferencia de búsqueda binaria, hacemos menos retrocesos. Puede ser más útil en aplicaciones dosde sea costoso ir para atrás.

Búsqueda exponencial

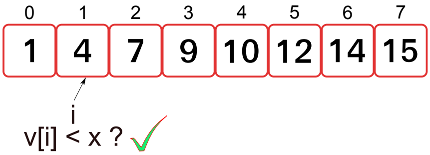
Descripción informal

- Dado un arreglo *ordenado*, lo recorremos de a potencias de 2 mientras el elemento actual sea menor al elemento a buscar.
- Cuando deja de cumplirse la condición, hacemos búsqueda binaria entre $\frac{i}{2}$ e i.

Ejemplo

Buscar el elemento x=14 en el siguiente array.

Empezamos con $i = 1$.



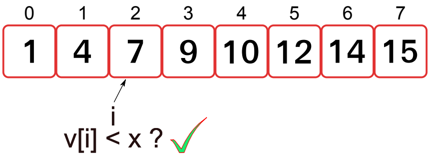
Búsqueda exponencial

Descripción informal

- Dado un arreglo *ordenado*, lo recorremos de a potencias de 2 mientras el elemento actual sea menor al elemento a buscar.
- Cuando deja de cumplirse la condición, hacemos búsqueda binaria entre $\frac{i}{2}$ e i.

Ejemplo

Buscar el elemento x=14 en el siguiente array.



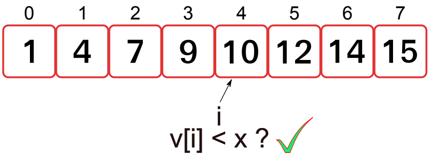
Búsqueda exponencial

Descripción informal

- Dado un arreglo *ordenado*, lo recorremos de a potencias de 2 mientras el elemento actual sea menor al elemento a buscar.
- Cuando deja de cumplirse la condición, hacemos búsqueda binaria entre $\frac{i}{2}$ e i.

Ejemplo

Buscar el elemento x=14 en el siguiente array.



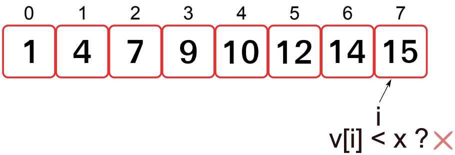
Búsqueda exponencial

Descripción informal

- Dado un arreglo *ordenado*, lo recorremos de a potencias de 2 mientras el elemento actual sea menor al elemento a buscar.
- Cuando deja de cumplirse la condición, hacemos búsqueda binaria entre $\frac{i}{2}$ e i.

Ejemplo

Buscar el elemento x=14 en el siguiente array.



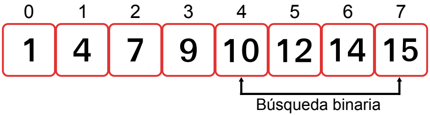
Búsqueda exponencial

Descripción informal

- Dado un arreglo *ordenado*, lo recorremos de a potencias de 2 mientras el elemento actual sea menor al elemento a buscar.
- Cuando deja de cumplirse la condición, hacemos búsqueda binaria entre $\frac{i}{2}$ e i.

Ejemplo

Buscar el elemento x=14 en el siguiente array.



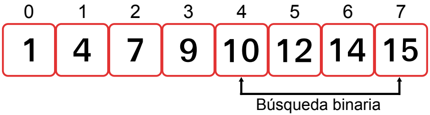
Búsqueda exponencial

Descripción informal

- Dado un arreglo *ordenado*, lo recorremos de a potencias de 2 mientras el elemento actual sea menor al elemento a buscar.
- Cuando deja de cumplirse la condición, hacemos búsqueda binaria entre $\frac{i}{2}$ e i.

Ejemplo

Buscar el elemento x=14 en el siguiente array.



Código

Ustedes!

Eficiencia

- Para llegar al *i-ésimo* índice vamos a hacer $log_2(i)$ iteraciones.

Eficiencia

- Para llegar al *i-ésimo* índice vamos a hacer $log_2(i)$ iteraciones.
- Una vez allí, debemos hacer búsqueda binaria entre i y $i/2$. La cantidad de iteraciones va a ser algo como $log_2(i) + log_2(i - i/2)$. Es decir, anda por el orden de $log(i)$.

Eficiencia

- Para llegar al *i-ésimo* índice vamos a hacer $log_2(i)$ iteraciones.
- Una vez allí, debemos hacer búsqueda binaria entre i y $i/2$. La cantidad de iteraciones va a ser algo como $log_2(i) + log_2(i - i/2)$. Es decir, anda por el orden de $log(i)$.
- En el peor caso, vamos a hacer aproximadamente $log(n)$ iteraciones. Entonces, ¿Qué ganamos con este algoritmo con respecto a la búsqueda binaria?

Eficiencia

- Para llegar al *i-ésimo* índice vamos a hacer $log_2(i)$ iteraciones.
- Una vez allí, debemos hacer búsqueda binaria entre i y $i/2$. La cantidad de iteraciones va a ser algo como $log_2(i) + log_2(i - i/2)$. Es decir, anda por el orden de $log(i)$.
- En el peor caso, vamos a hacer aproximadamente $log(n)$ iteraciones. Entonces, ¿Qué ganamos con este algoritmo con respecto a la búsqueda binaria?
- Si el array es muuuy grande y de alguna manera sabemos que el número a buscar lo vamos a encontrar más bien al principio.