Algoritmos de búsqueda sobre secuencias

Algoritmos y Estructuras de Datos I

Búsqueda lineal

- Recordemos el problema de búsqueda por valor de un elemento en una secuencia.
- proc *contiene*(in $s : seq\langle\mathbb{Z}\rangle$, in $x : \mathbb{Z}$, out *result* : Bool){
  Pre $\{True\}$
  Post $\{result = true \leftrightarrow (\exists i : \mathbb{Z})(0 \leq i < |s| \wedge_L s[i] = x)\}$
  }
- ¿Cómo podemos buscar un elemento en una secuencia?

Búsqueda lineal

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

- ¿Qué invariante de ciclo podemos proponer?

  $I \equiv 0 \leq i \leq |s| \wedge_L$
  $(\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] \neq x)$

- ¿Qué función variante podemos usar?

  $fv = |s| - i$

Búsqueda lineal

- Invariante de ciclo:

$$I \equiv 0 \leq i \leq |s| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] \neq x)$$

- Función variante:

$$fv = |s| - i$$

- ¿Cómo lo podemos implementar en C++?

```
bool contiene(vector<int> &s, int x) {
  int i = 0;
  while( i < s.size() && s[i] != x ) {
    i=i+1;
  }
  return i < s.size();
}
```

- ¿Es la implementación correcto con respecto a la especificación?

Recap: Teorema de corrección de un ciclo

- **Teorema.** Sean un predicado I y una función $fv : \mathbb{V} \to \mathbb{Z}$ (donde $\mathbb{V}$ es el producto cartesiano de los dominios de las variables del programa), y supongamos que $I \Rightarrow \text{def}(B)$. Si
  1. $P_C \Rightarrow I$,
  2. $\{I \wedge B\}\ S\ \{I\}$,
  3. $I \wedge \neg B \Rightarrow Q_C$,
  4. $\{I \wedge B \wedge v_0 = fv\}$ **S** $\{fv < v_0\}$,
  5. $I \wedge fv \leq 0 \Rightarrow \neg B$,

  ... entonces la siguiente tripla de Hoare es válida:

  $\{P_C\}$ **while B do S endwhile** $\{Q_C\}$

Búsqueda lineal

- Para este programa, tenemos:
  - $P_C \equiv i = 0$,
  - $Q_C \equiv (i < |s|) \leftrightarrow (\exists j : \mathbb{Z})(0 \leq j < |s| \wedge_L s[j] = x)$.
  - $B \equiv i < |s| \wedge_L s[i] \neq x$
  - $I \equiv 0 \leq i \leq |s| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] \neq x)$
  - $fv = |s| - i$
- Ahora tenemos que probar que:
  1. $P_C \Rightarrow I$,
  2. $\{I \wedge B\}\ S\ \{I\}$,
  3. $I \wedge \neg B \Rightarrow Q_C$,
  4. $\{I \wedge B \wedge v_0 = fv\}$ **S** $\{fv < v_0\}$,
  5. $I \wedge fv \leq 0 \Rightarrow \neg B$,

Recap: Teorema de corrección de un ciclo

1. $P_C \Rightarrow I$,
2. $\{I \wedge B\}\ S\ \{I\}$,
3. $I \wedge \neg B \Rightarrow Q_C$,
4. $\{I \wedge B \wedge v_0 = fv\}$ **S** $\{fv < v_0\}$,
5. $I \wedge fv \leq 0 \Rightarrow \neg B$,

En otras palabras, hay que mostrar que:

- I es un invariante del ciclo (punto 1. y 2.)
- Se cumple la postcondición del ciclo a la salida del ciclo (punto 3.)
- La función variante es estrictamente decreciente (punto 4.)
- Si la función variante alcanza la cota inferior la guarda se deja de cumplir (punto 5.)

Corrección de búsqueda lineal

¿I es un invariante del ciclo?

$$I \equiv 0 \leq i \leq |s| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] \neq x)$$

- La variable i toma el primer valor 0 y se incrementa por cada iteración hasta llegar a $|s|$.
- $\Rightarrow 0 \leq i \leq |s|$
- En cada iteración, todos los elementos a izquierda de i son distintos de x
- $\Rightarrow (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] \neq x)$

Corrección de búsqueda lineal

¿Se cumple la postcondición del ciclo a la salida del ciclo?

$$I \equiv 0 \leq i \leq |s| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] \neq x)$$

$$Q_C \equiv (i < |s|) \leftrightarrow (\exists i : \mathbb{Z})(0 \leq i < |s| \wedge_L s[i] = x)$$

- Al salir del ciclo, no se cumple la guarda. Entonces no se cumple $i < |s|$ o no se cumple $s[i] \neq x$
  - Si no se cumple $i < |s|$, no existe ninguna posición que contenga x
  - Si no se cumple $s[i] \neq x$, existe al menos una posición que contiene a x

Corrección de búsqueda lineal

¿Es la función variante estrictamente decreciente?

$$fv = |s| - i$$

- En cada iteración, se incremente en 1 el valor de i
- Por lo tanto, en cada iteración se reduce en 1 la función variante.

Corrección de búsqueda lineal

¿Si la función variante alcanza la cota inferior la guarda se deja de cumplir?

$$fv = |s| - i$$

$$B \equiv i < |s| \wedge_L s[i] \neq x$$

- Si $fv = |s| - i \leq 0$,entonces $i \geq |s|$
- Como siempre pasa que $i \leq |s|$, entonces es cierto que $i = |s|$
- Por lo tanto $i < |s|$ es falso.

Corrección de búsqueda lineal

- Finalmente,ahora que probamos que:
  1. $P_C \Rightarrow I$,
  2. $\{I \wedge B\}\ S\ \{I\}$,
  3. $I \wedge \neg B \Rightarrow Q_C$,
  4. $\{I \wedge B \wedge v_0 = fv\}$ **S** $\{fv < v_0\}$,
  5. $I \wedge fv \leq 0 \Rightarrow \neg B$,
- ...podemos por el teorema concluir que el ciclo termina y es correcto.

Búsqueda lineal

- Implementación:

```
bool contiene(vector<int> &s, int x) {
  int i = 0;
  while( i < s.size() && s[i] != x ) {
    i=i+1;
  }
  return i < s.size();
}
```

- Analicemos cuántas veces itera este programa:

s	x	# iteraciones
$\langle\rangle$	1	0
$\langle 1\rangle$	1	0
$\langle 1,2\rangle$	2	1
$\langle 1,2,3\rangle$	4	3
$\langle 1,2,3,4\rangle$	4	3
$\langle 1,2,3,4,5\rangle$	-1	4

Búsqueda lineal

- ¿Cuántas veces se ejecuta el ciclo? Esto depende de
  - El tamaño de la secuencia
  - Si el valor buscado está o no contenido en la secuencia
- ¿Qué tiene que pasar para que la cantidad de ejecuciones sea máxima?
  - El elemento no debe estar contenido.
- Esto representa el **peor caso** en cantidad de iteraciones, ya que tarda mas
- Dado una secuencia cualquiera, ¿cuál es la cantidad máxima de iteraciones (el peor caso) que puede ejecutar el algortimo? En peor caso se ejecuta $|s|$ veces.

Búsqueda sobre secuencias ordenadas

- Supongamos que la secuencia está ordenada.
- proc *contieneOrdenada*(in $s : seq\langle\mathbb{Z}\rangle$, in $x : \mathbb{Z}$, out $result$: Bool){

  Pre {ordenado(s)}

  Post $\{result = True \leftrightarrow (\exists i : \mathbb{Z})(0 \leq i < |s| \wedge_L s[i] = x)\}$

  }
- ¿Podemos aprovechar que la secuencia está ordenada para crear un programa más eficiente ?

Búsqueda sobre secuencias ordenadas

Podemos interrumpir la búsqueda tan pronto como verificamos que $s[i] \geq x$.

```
bool contieneOrdenada(vector<int> &s, int x) {
  int i = 0;
  while( i < s.size() && s[i] < x ) {
    i=i+1;
  }
  return (i < s.size() && s[i] == x);
}
```

¿Cuántas veces se ejecuta el ciclo en peor caso?

Búsqueda sobre secuencias ordenadas

¿Cuántas veces se ejecuta el ciclo en peor caso?

s	x	# iteraciones (contiene)	# iteraciones (contieneOrdenada)
$\langle\rangle$	1	0	0
$\langle 1\rangle$	10	1	1
$\langle 1,2\rangle$	10	2	2
$\langle 1,2,3\rangle$	10	3	3
$\langle 1,2,3,4\rangle$	10	4	4
$\langle 1,2,3,4,5\rangle$	10	5	5
...	...	...	...
s	$x \notin s$	$\|s\|$	$\|s\|$

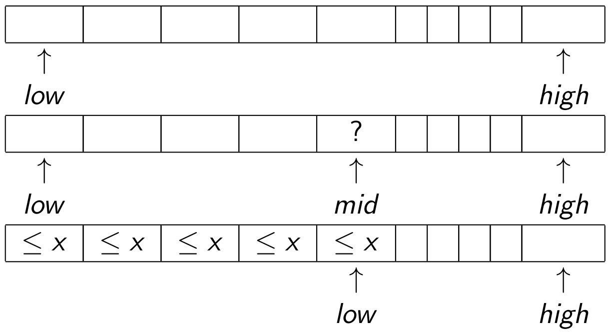
En **peor caso** (cuando el elemento no está) ambos se ejecutan la misma cantidad de veces.

Búsqueda sobre secuencias ordenadas

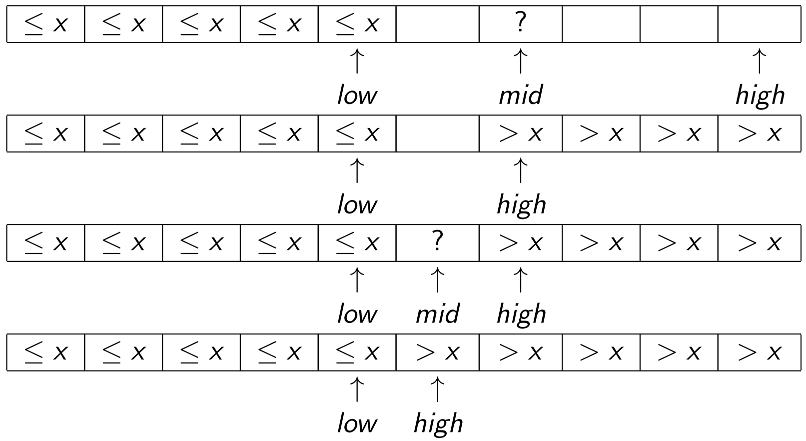
- ¿Cómo podemos aprovechar que la secuencia está ordenada para mejorar el peor caso de ejecución?
  - ¿Necesitamos iterar si $|s| = 0$? Trivialmente, $x \notin s$
  - ¿Necesitamos iterar si $|s| = 1$?Trivialmente, $s[0] == x \leftrightarrow x \in s$
  - ¿Necesitamos iterar si $x < s[0]$? Trivialmente, $x \notin s$
  - ¿Necesitamos iterar si $x > s[|s| - 1]$? Trivialmente, $x \notin s$

Búsqueda sobre secuencias ordenadas

Asumamos por un momento que $|s| > 1 \wedge_L (s[0] \leq x \leq s[|s| - 1])$



Búsqueda sobre secuencias ordenadas



Si $x \in s$, tiene que estar en la posición *low* de la secuencia.

Búsqueda sobre secuencias ordenadas

$\leq x$	$\leq x$	$\leq x$	$\leq x$	$\leq x$	$> x$	$> x$	$> x$	$> x$	$> x$
				$\uparrow$ *low*	$\uparrow$ *high*				

- ¿Qué invariante de ciclo podemos escribir?

$$I \equiv 0 \leq low < high < |s| \wedge_L s[low] \leq x < s[high]$$

- ¿Qué función variante podemos definir?

$$fv = high - low - 1$$

Búsqueda sobre secuencias ordenadas

```
bool contieneOrdenada(vector<int> &s, int x) {
  // casos triviales
  if (s.size()==0 ) {
    return false;
  } else if (s.size()==1) {
    return s[0]==x;
  } else if (x<s[0]) {
    return false;
  } else if (x>=s[s.size()−1]) {
    return s[s.size()−1]==x;
  } else {
    // casos no triviales
    ...
  }
}
```

Búsqueda sobre secuencias ordenadas

```
  } else {
    // casos no triviales
    int low = 0;
    int high = s.size() − 1;
    while( low+1 < high ) {
      int mid = (low+high) / 2;
      if( s[mid] <= x ) {
        low = mid;
      } else {
        high = mid;
      }
    }
    return s[low] == x;
  }
}
```

A este algoritmo se lo denomina búsqueda binaria

Búsqueda binaria

- Veamos ahora que este algoritmo es correcto.

$$\begin{aligned}
P_C &\equiv ordenada(s) \wedge (|s| > 1 \wedge_L s[0] \leq x \leq [|s| - 1]) \\
&\quad \wedge\ low = 0 \wedge high = |s| - 1 \\
Q_C &\equiv (s[low] = x) \leftrightarrow (\exists i : \mathbb{Z})(0 \leq i < |s| \wedge_L s[i] = x) \\
B &\equiv low + 1 < high \\
I &\equiv 0 \leq low < high < |s| \ \wedge_L\ s[low] \leq x < s[high] \\
fv &= high - low - 1
\end{aligned}$$

Corrección de la búsqueda binaria

- ¿Es I un invariante para el ciclo?
  - El valor de *low* es siempre menor estricto que *high*
  - *low* arranca en 0 y sólo se aumenta
  - *high* arranca en $|s| - 1$ y siempre se disminuye
  - Siempre se respecta que $s[low] \leq x$ y que $x < s[high]$
- ¿A la salida del ciclo se cumple la postcondicion Q_C?
  - Al salir, se cumple que $low + 1 = high$
  - Sabemos que $s[high] > x$ y $s[low] <= x$
  - Como s está ordenada, si $x \in s$, entonces $s[low] = x$

Corrección de la búsqueda binaria

- ¿Es la función variante estrictamente decreciente?
  - Nunca ocurre que $low = high$
  - Por lo tanto, siempre ocurre que $low < mid < high$
  - De este modo, en cada iteración, o bien high es estrictamente menor, o bien low es estrictamente mayor.
  - Por lo tanto, la expresión $high - low - 1$ siempre es estrictamente menor.
- ¿Si la función variante alcanza la cota inferior la guarda se deja de cumplir?
  - Si $high - low - 1 \leq 0$, entonces $high \leq low + 1$.
  - Por lo tanto, no se cumple $(high > low + 1)$, que es la guarda del ciclo

Búsqueda binaria

- ¿Podemos interrumpir el ciclo si encontramos x antes de finalizar las iteraciones?
- Una posibilidad **no recomendada** (no lo hagan en casa!):
-
```
  ...
  while( low+1 < high) {
    int mid = (low+high) / 2;
    if( s[mid] < x ) {
      low = mid;
    } else if( s[mid] > x ) {
      high = low;
    } else {
      return true; // Argh!
    }
  }
  return s[low] == x;
}
```

Búsqueda binaria

- Una posibilidad **aún peor** (ni lo intenten!):
-
```
  bool salir = false;
  while( low+1 < high && !salir ) {
    int mid = (low+high) / 2;
    if( s[mid] < x ) {
      low = mid;
    } else if( s[mid] > x ) {
      high = mid;
    } else {
      salir = true; // Puaj!
    }
  }

  return s[low] == x || s[(low+high)/2] == x;
}
```

Búsqueda binaria

- Si queremos salir del ciclo, el lugar para decirlo es ... la guarda!
-

```
while( low+1 < high && s[low] != x ) {
  int mid = (low+high) / 2;
  if( s[mid] <= x ) {
    low = mid;
  } else {
    high = mid;
  }
}
return s[low] == x;
}
```

- Usamos fuertemente la condición $s[low] \leq x < s[high]$ del invariante.

Búsqueda binaria

- ¿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	$\cong (\lvert s \rvert - 1)/2^t$

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

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

Búsqueda binaria

- ¿Es mejor un algoritmo que ejecuta una cantidad logarítmica de iteraciones?

$\lvert s \rvert$	Búsqueda Lineal	Búsqueda Binaria
10	10	4
10^2	100	7
10^6	$1,000,000$	21
$2{,}3 \times 10^7$	$23,000,000$	25
7×10^9	$7,000,000,000$	33 (!)

- Sí! Búsqueda binaria es más eficiente que búsqueda lineal
- Pero, requiere que la secuencia esté ya ordenada.

Nearly all binary searches are broken!


Google Research Blog

The latest news from Research at Google

Extra, Extra - Read All About It: Nearly All Binary Searches and Mergesorts are Broken

Posted: Friday, June 02, 2006

G+1 302

Posted by Joshua Bloch, Software Engineer

I remember vividly Jon Bentley's first Algorithms lecture at CMU, where he asked all of us incoming Ph.D. students to write a binary search, and then dissected one of our implementations in front of the class. Of course it was broken, as were most of our implementations. This made a real impression on me, as did the treatment of this material in his wonderful *Programming Pearls* (Addison-Wesley, 1986; Second Edition, 2000). The key lesson was to carefully consider the invariants in your programs.

Fast forward to 2006. I was shocked to learn that the binary search program that Bentley proved correct and subsequently tested in Chapter 5 of *Programming Pearls* contains a bug. Once I tell you what it is, you will understand why it escaped detection for two decades. Lest you think I'm picking on Bentley, let me tell you how I

http://goo.gl/Ww0Cx6

Nearly all binary searches are broken!

- En 2006 comenzaron a reportarse accesos fuera de rango a vectores dentro de la función `binarySearch` implementada en las bibliotecas estándar de Java.
- En la implementación en Java, los enteros tienen precisión finita, con rango $[-2^{31}, 2^{31}-1]$.
- Si `low` y `high` son valores muy grandes, al calcular `k` se produce overflow.
- La falla estuvo *dormida* muchos años y se manifestó sólo cuando el tamaño de los vectores creció a la par de la capacidad de memoria de las computadoras.
- Bugfix: Computar `k` evitando el overflow:

```
int mid = low + (high-low)/2;
```

Conclusiones

- La búsqueda binaria implementada en Java estaba formalmente demostrada ...
- ... pero la demostración suponía enteros de precisión infinita (en la mayoría de los lenguajes imperativos son de precisión finita).
  - En AED1 no nos preocupan los problemas de aritmética de precisión finita (+Info: Orga1).
  - Es importante validar que las hipótesis sobre las que se realizó la demostración valgan en la implementación (aritmética finita, existencia de acceso concurrente, multi-threading, etc.)

Bibliografía

- David Gries - The Science of Programming
  - Chapter 16 - Developing Invariants (Linear Search, Binary Search)