Algoritmos y estructuras de datos

Búsqueda interna

Francisco Javier Zaragoza Martínez

Universidad Autónoma Metropolitana Unidad Azcapotzalco
Departamento de Sistemas

17 de noviembre de 2023

Citas

Eurípides

No dejes ninguna piedra sin remover.

Dichos populares

El que busca, encuentra. Para encontrar, primero debes buscar. Buscar aguja en pajar es naufragar. Cada cosa en su lugar ahorra tiempo al buscar.

Arthur C. Clarke

Todos los exploradores están buscando algo que perdieron. Es raro que lo encuentren, y más raro aún que el encontrarlo les traiga mayor felicidad que la búsqueda.

Búsqueda interna

Definición

Problema abstracto

Dado un arreglo A con n elementos y un dato x, se desea saber si x está en el arreglo y, en ese caso, dónde está.

Problema concreto

Dado un arreglo `int a[MAX]` con `int n` elementos en las posiciones `a[0], ..., a[n-1]` y un dato `int x`, se desea saber si `x` está en el arreglo y, en ese caso, un índice `int i` tal que `a[i]==x`.

Búsqueda lineal

Implementación con arreglos

La búsqueda lineal consiste simplemente de preguntar en cada posición.

```
int  lineal(int n, int a[], int x) {
  for (int i = 0; i < n; i++)
    if (a[i] == x)
      return i;
  return -1;
}
```

En el peor de los casos se hacen n comparaciones (si x es el último o no está).

Búsqueda lineal

Implementación con apuntadores

También lo podemos hacer con apuntadores:

```
int* lineal(int n, int *a, int x) {
  for (int *p = a; p < a+n; p++)
    if (*p == x)
      return p;
  return NULL;
}
```

En el peor de los casos se hacen n comparaciones (si x es el último o no está).

Búsqueda lineal

Ejercicios

1. Escribe una función recursiva `int lineal(int n, int a[], int x)` que lleve a cabo la búsqueda lineal en un arreglo desordenado.
2. Reescribe las dos versiones de `lineal` para comenzar a buscar desde el final del arreglo.
3. Reescribe las dos versiones de `lineal` para que regresen la cantidad de apariciones del elemento `x` en el arreglo.

Problemas de la búsqueda lineal

Estas funciones son muy lentas. En particular, si x no está en el arreglo entonces se requieren n comparaciones para darse cuenta.

¿Qué hacer?

- Si el arreglo no está ordenado no hay nada que hacer. ¿Por qué?
- ¿De qué sirve que el arreglo esté ordenado crecientemente?
- Veamos varias formas de aprovechar el orden.

Búsqueda lineal en un arreglo creciente

Implementación con arreglos

Cambiemos la condición de paro del ciclo:

```
int  lineal(int n, int a[], int x) {
  for (int i = 0; (i < n) && (a[i] <= x); i++)
    if (a[i] == x)
      return i;
  return -1;
}
```

Búsqueda lineal en un arreglo creciente

Implementación con apuntadores

También lo podemos hacer con apuntadores:

```
int* lineal(int n, int *a, int x) {
  for (int *p = a; (p < a+n) && (*p <= x); p++)
    if (*p == x)
      return p;
  return NULL;
}
```

Búsqueda lineal con saltos

Saltos de tamaño fijo

Una posibilidad para acelerar la búsqueda lineal es escoger un salto s y revisar las posiciones $A_0, A_s, A_{2s}, \ldots$ hasta que encontremos un elemento $A_{ks} \geq x$ o salgamos del arreglo. Una vez que esto pase, revisamos una por una las $\leq s$ posiciones que terminan en A_{ks} (o el fin del arreglo).

Búsqueda lineal con saltos

Saltos de tamaño fijo

Una posibilidad para acelerar la búsqueda lineal es escoger un salto s y revisar las posiciones $A_0, A_s, A_{2s}, \ldots$ hasta que encontremos un elemento $A_{ks} \geq x$ o salgamos del arreglo. Una vez que esto pase, revisamos una por una las $\leq s$ posiciones que terminan en A_{ks} (o el fin del arreglo).

Tiempo de ejecución

En la primera etapa se hacen $\leq 1 + \frac{n}{s}$ revisiones. En la segunda etapa se hacen $\leq s - 1$ revisiones adicionales. En total son $r(s) \leq s + \frac{n}{s}$ revisiones.

Búsqueda lineal con saltos

Saltos de tamaño fijo

Una posibilidad para acelerar la búsqueda lineal es escoger un salto s y revisar las posiciones $A_0, A_s, A_{2s}, \ldots$ hasta que encontremos un elemento $A_{ks} \geq x$ o salgamos del arreglo. Una vez que esto pase, revisamos una por una las $\leq s$ posiciones que terminan en A_{ks} (o el fin del arreglo).

Tiempo de ejecución

En la primera etapa se hacen $\leq 1 + \frac{n}{s}$ revisiones. En la segunda etapa se hacen $\leq s - 1$ revisiones adicionales. En total son $r(s) \leq s + \frac{n}{s}$ revisiones.

Mejor valor del salto

Tomando la derivada de $r(s)$ e igualando a cero obtenemos $0 = 1 - \frac{n}{s^2}$, es decir, $s = \sqrt{n}$. En este caso, $r(s) \leq 2\sqrt{n}$. ¡Esto es mucho mejor que lineal!

Búsqueda lineal

Ejercicios

1. Escribe una función recursiva `int lineal(int n, int a[], int x)` que lleve a cabo la búsqueda lineal en un arreglo ordenado.
2. Escribe una función `int creciente(int n, int a[])` que decida si un arreglo cumple `a[0] <= a[1] <= ... <= a[n-1]`.
3. Escribe una función `int creciente(int n, int a[])` que decida si un arreglo cumple `a[0] < a[1] < ... < a[n-1]`.
4. Escribe una función `int pivote(int n, int a[], int p)` que decida si `a[i] <= a[p]` para toda `0 <= i < p` y `a[i] >= a[p]` para toda `p < i < n`.
5. Escribe una función `int salto(int n, int a[], int x, int s)` que implemente la búsqueda lineal con saltos de tamaño `s`.

Búsqueda binaria

Idea principal

Tenemos un arreglo ordenado `a[0] <= a[1] <= ... <= a[n-1]` en el que queremos saber si está `x`. Sea `m = (n-1)/2` y compare `x` con `a[m]`.

1. Si `x == a[m]` ya acabamos.
2. Si `x < a[m]` entonces `x` no está en `a[m]...a[n-1]`.
3. Si `x > a[m]` entonces `x` no está en `a[0]...a[m]`.

Búsqueda binaria

Idea principal

Tenemos un arreglo ordenado `a[0] <= a[1] <= ... <= a[n-1]` en el que queremos saber si está `x`. Sea `m = (n-1)/2` y compare `x` con `a[m]`.

1. Si `x == a[m]` ya acabamos.
2. Si `x < a[m]` entonces `x` no está en `a[m]...a[n-1]`.
3. Si `x > a[m]` entonces `x` no está en `a[0]...a[m]`.

Ejemplo (Buscando 42)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10	11	14	23	26	31	39	40	42	43	51	55	67	70	72	79

Búsqueda binaria

Idea principal

Tenemos un arreglo ordenado `a[0] <= a[1] <= ... <= a[n-1]` en el que queremos saber si está `x`. Sea `m = (n-1)/2` y compare `x` con `a[m]`.

1. Si `x == a[m]` ya acabamos.
2. Si `x < a[m]` entonces `x` no está en `a[m]...a[n-1]`.
3. Si `x > a[m]` entonces `x` no está en `a[0]...a[m]`.

Ejemplo (Buscando 42)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10	11	14	23	26	31	39	40	42	43	51	55	67	70	72	79
								42	43	51	55	67	70	72	79

Búsqueda binaria

Idea principal

Tenemos un arreglo ordenado `a[0] <= a[1] <= ... <= a[n-1]` en el que queremos saber si está `x`. Sea `m = (n-1)/2` y compare `x` con `a[m]`.

1. Si `x == a[m]` ya acabamos.
2. Si `x < a[m]` entonces `x` no está en `a[m]...a[n-1]`.
3. Si `x > a[m]` entonces `x` no está en `a[0]...a[m]`.

Ejemplo (Buscando 42)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10	11	14	23	26	31	39	40	42	43	51	55	67	70	72	79
								42	43	51	55	67	70	72	79
								42	43	51					

Búsqueda binaria

Idea principal

Tenemos un arreglo ordenado `a[0] <= a[1] <= ... <= a[n-1]` en el que queremos saber si está `x`. Sea `m = (n-1)/2` y compare `x` con `a[m]`.

1. Si `x == a[m]` ya acabamos.
2. Si `x < a[m]` entonces `x` no está en `a[m]...a[n-1]`.
3. Si `x > a[m]` entonces `x` no está en `a[0]...a[m]`.

Ejemplo (Buscando 42)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10	11	14	23	26	31	39	40	42	43	51	55	67	70	72	79
								42	43	51	55	67	70	72	79
								42	43	51					
								42							

Búsqueda binaria

Idea más general

Tenemos un arreglo ordenado `a[i] <= a[i+1] <= ... <= a[j]` en el que queremos saber si está `x`. Sea `m = (i+j)/2` y compare `x` con `a[m]`.

1. Si `x == a[m]` ya acabamos.
2. Si `x < a[m]` entonces `x` no está en `a[m]...a[j]`.
3. Si `x > a[m]` entonces `x` no está en `a[i]...a[m]`.

En los últimos dos casos, la búsqueda continúa en el intervalo no descartado.

Búsqueda binaria

Idea más general

Tenemos un arreglo ordenado `a[i] <= a[i+1] <= ... <= a[j]` en el que queremos saber si está `x`. Sea `m = (i+j)/2` y compare `x` con `a[m]`.

1. Si `x == a[m]` ya acabamos.
2. Si `x < a[m]` entonces `x` no está en `a[m]...a[j]`.
3. Si `x > a[m]` entonces `x` no está en `a[i]...a[m]`.

En los últimos dos casos, la búsqueda continúa en el intervalo no descartado.

Tiempo de ejecución

Sea $T(n)$ el número de comparaciones que hacemos en el peor de los casos. Entonces $T(1) = 1$ y $T(n) = T(\lfloor \frac{n}{2} \rfloor) + 1$ si $n \geq 2$. Finalmente, $T(n) = \lceil \log_2 n \rceil + 1$.

Búsqueda binaria

Implementación iterativa con arreglos

```
int  binaria(int i, int j, int a[], int x) {
  int  m;
  while (i <= j) {
    m = i + (j-i)/2; // m = (i+j)/2
    if (x == a[m])
      return m;
    if (x < a[m])
      j = m-1;
    else
      i = m+1;
  }
  return -1;
}
```

Búsqueda binaria

Implementación iterativa con apuntadores

```
int* binaria(int *p, int *q, int x) {
  int *m;
  while (p <= q) {
    m = p + (q-p)/2; // resta de apuntadores
    if (x == *m)
      return m;
    if (x < *m)
      q = m-1;
    else
      p = m+1;
  }
  return NULL;
}
```

Búsqueda binaria

Recursivamente

Podemos hacer búsqueda binaria de x en el arreglo $(a_i, \ldots, a_j)$ así:

$$f(i,j) = \begin{cases} -1 & \text{si } i > j \\ m & \text{si } x = a_m \\ f(i, m-1) & \text{si } x < a_m \\ f(m+1, j) & \text{si } x > a_m \end{cases}$$

donde $m = \lfloor \frac{i+j}{2} \rfloor$.

Búsqueda binaria

Ejemplo: buscando 42

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10	11	14	23	26	31	39	40	42	43	51	55	67	70	72	79
								42	43	51	55	67	70	72	79
								42	43	51					
								42							

$$
\begin{aligned}
f(0,15) &= f(\lfloor \tfrac{0+15}{2} \rfloor + 1, 15) \\
&= f(8,15) \\
&= f(8, \lfloor \tfrac{8+15}{2} \rfloor - 1) \\
&= f(8,10) \\
&= f(8, \lfloor \tfrac{8+10}{2} \rfloor - 1) \\
&= f(8,8) \\
&= 8
\end{aligned}
$$

Búsqueda binaria

Ejemplo

$$
\begin{aligned}
f(0,15) &= f(\lfloor \tfrac{0+15}{2} \rfloor + 1, 15) \\
&= f(8,15) \\
&= f(8, \lfloor \tfrac{8+15}{2} \rfloor - 1) \\
&= f(8,10) \\
&= f(8, \lfloor \tfrac{8+10}{2} \rfloor - 1) \\
&= f(8,8) \\
&= 8
\end{aligned}
$$

Árbol de recursión

$$
\begin{array}{c}
f(0,15) = f(8,15) \\
\downarrow \\
f(8,15) = f(8,10) \\
\downarrow \\
f(8,10) = f(8,8) \\
\downarrow \\
f(8,8) = 8
\end{array}
$$

Árbol de recursión

$$
\begin{array}{c}
(0,15) \\
\downarrow \\
(8,15) \\
\downarrow \\
(8,10) \\
\downarrow \\
(8,8)
\end{array}
$$

Búsqueda binaria

Implementación recursiva con arreglos

```
int  binaria(int i, int j, int a[], int x) {
  if (i <= j) {
    int  m = i + (j-i)/2;
    if (x == a[m])
      return m;
    if (x < a[m])
      return binaria(i, m-1, a, x);
    else
      return binaria(m+1, j, a, x);
  }
  return -1;
}
```

Búsqueda binaria

Implementación recursiva con apuntadores

```
int* binaria(int *p, int *q, int x) {
  if (p <= q) {
    int *m = p + (q-p)/2;
    if (x == *m)
      return m;
    if (x < *m)
      return binaria(p, m-1, a, x);
    else
      return binaria(m+1, q, a, x);
  }
  return NULL;
}
```

Búsqueda por interpolación

Regresando a la idea de los saltos

Hasta ahora hemos usado saltos independientes de los datos.

Búsqueda por interpolación

Regresando a la idea de los saltos

Hasta ahora hemos usado saltos independientes de los datos.

Usemos los datos

Si los elementos del arreglo `a` son numéricos entonces podemos hacer cálculos con ellos. En particular, si `i < j` y `a[i] <= x <= a[j]` podemos usar una línea recta para estimar la posición `m` en la que debería estar `x`:

$$\frac{\texttt{m - i}}{\texttt{j - i}} = \frac{\texttt{x - a[i]}}{\texttt{a[j] - a[i]}}$$

de donde `m = i + (j-i)*(x-a[i])/(a[j]-a[i])`. ¿Qué pasa si `a[i] = a[j]`?

Búsqueda por interpolación

Regresando a la idea de los saltos

Hasta ahora hemos usado saltos independientes de los datos.

Usemos los datos

Si los elementos del arreglo `a` son numéricos entonces podemos hacer cálculos con ellos. En particular, si `i < j` y `a[i] <= x <= a[j]` podemos usar una línea recta para estimar la posición `m` en la que debería estar `x`:

$$\frac{\texttt{m - i}}{\texttt{j - i}} = \frac{\texttt{x - a[i]}}{\texttt{a[j] - a[i]}}$$

de donde `m = i + (j-i)*(x-a[i])/(a[j]-a[i])`. ¿Qué pasa si `a[i] = a[j]`?

Cantidad de comparaciones

Si los datos están uniformemente distribuidos entonces la búsqueda basada en esta idea hace $\approx \log_2 \log_2 n$ comparaciones en promedio.

Búsqueda binaria

Ejercicios

1. Escribe una función `int inferior(int i, int j, int a[], int x)` que regrese el menor índice `k` tal que `a[k] >= x` o bien `j`+1 si tal índice no existe.
2. Escribe una función `int superior(int i, int j, int a[], int x)` que regrese el menor índice `k` tal que `a[k] > x` o bien `j`+1 si tal índice no existe.
3. Reescribe `inferior` y `superior` con apuntadores.
4. Estoy pensando en un entero positivo x arbitrariamente grande y tú quieres encontrarlo. Para ello puedes hacerme preguntas del estilo ¿cómo es tu número comparado con z? y yo te responderé si son iguales o cuál es mayor. ¿Cómo encontrarías x rápidamente?
5. Implementa la búsqueda por interpolación.