BUSCA LINEAR | BUSCA BINÁRIA

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Agenda

Força bruta

Brute force: uma estratégia de desenvolver algoritmos a partir da definição do problema e de conceitos relacionados.

- Encontrar $gcd(m, n)$ com verificação consecutiva de inteiros
- Calcular a^n multiplicando n vezes o valor a
- Encontrar um elemento percorrendo todas as posições do array

Vantagens

- Estratégia com grande aplicabilidade
- Pode ser útil para instâncias pequenas do problema
- Referencial teórico e educacional

Desvantagem: normalmente ineficiente

Busca linear

Algoritmo visto anteriormente:

Algoritmo: SequentialSearch(A[0..n-1], K)

$i \leftarrow 0$;
while $i < n \wedge A[i] \neq K$ **do**
    $i \leftarrow i + 1$;
if $i < n$ **then return** i;
else return -1;

Busca linear

Pequena otimização:

Algoritmo: SequentialSearch2(A[0..n], K)

$A[n] \leftarrow K$;
$i \leftarrow 0$;
while $A[i] \neq K$ **do**
    $i \leftarrow i + 1$;
if $i < n$ **then return** i;
else return -1;

Se a lista estiver ordenada, parar a busca assim que o elemento atual for $\geq$ do que o valor procurado.

Complexidade: $C_{worst}(n) \in O(n)$.

Agenda

Busca binária

Estratégia: decrease-and-conquer (decrease-by-a-constant-factor)

Informalmente (assumindo entrada ordenada):

1. Compara K com o elemento no meio do array $A[m]$. Se encontrou K, para. Caso contrário, segue para o Passo 2.
2. Repete o Passo 1 (recursivamente) para a primeira metade do array se $K < A[m]$ ou para a segunda metade se $K > A[m]$.

$$\begin{array}{c} K \\ \updownarrow \\ \underbrace{A[0] \ldots A[m-1]}_{\substack{\text{search here if} \\ K<A[m]}} \quad A[m] \quad \underbrace{A[m+1] \ldots A[n-1]}_{\substack{\text{search here if} \\ K>A[m]}}. \end{array}$$ [1]

[1] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Busca binária: exemplo

Suponha o seguinte array e que $K = 70$.

3	14	27	31	39	42	55	70	74	81	85	93	98

index	0	1	2	3	4	5	6	7	8	9	10	11	12
value	3	14	27	31	39	42	55	70	74	81	85	93	98
iteration 1	*l*						*m*						*r*
iteration 2								*l*		*m*			*r*
iteration 3								*l,m*	*r*				

[2]

[2] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Busca binária: algoritmo

Observe que o código não é recursivo.

Algoritmo: BinarySearch(A[0..n - 1], K)

```
// Premissa: A deve estar ordenado crescentemente
l ← 0;
r ← n − 1;
while l ≤ r do
    m ← ⌊(l + r)/2⌋;
    if K = A[m] then return m;
    else if K < A[m] then r ← m − 1 ;
    else l ← m + 1 ;
return −1;
```

Busca binária: complexidade

Operação básica: quantidade de comparações envolvendo K

- Para simplificar, após uma comparação K com $A[m]$, já se sabe se $K = A[m]$, $K < A[m]$ ou $K > A[m]$.

$C(n)$ depende não só de n, mas também se K está em A.

- Análise do pior caso, melhor caso e caso médio

Relação de recorrência
$C_{worst}(n) = 1 + C_{worst}(\lfloor n/2 \rfloor)$ para $n > 0$
$C_{worst}(n) = 0$ para $n = 0$

Busca binária: complexidade

Considerando: ambos os lados do array possuem o mesmo tamanho

$$C_{worst}(n) = C_{worst}(\frac{n-1}{2}) + 1$$

$$C_{worst}(\frac{n-1}{2}) = 1 + C_{worst}(\frac{\frac{n-1}{2}-1}{2}) = 1 + C_{worst}(\frac{n-2^0-2^1}{2^2})$$

$$C_{worst}(\frac{n-2^0-2^1}{2^2}) = 1 + C_{worst}(\frac{n-2^0-2^1-2^2}{2^3})$$

$$\vdots$$

$$C_{worst}(\frac{n-2^0-2^1-\cdots-2^j}{2^{j+1}}) = 1 + C_{worst}(\frac{n-2^0-2^1-\cdots-2^{j+1}}{2^{j+2}})$$

A recorrência termina quando o argumento de C_{worst} for 0, logo:

$$C_{worst}(\frac{n-2^0-2^1-\cdots-2^{j+1}}{2^{j+2}}) = C_{worst}(0) = 0$$

Busca binária: complexidade

Por outro lado, temos:

$$\begin{aligned} C_{worst}(n) &= 1 + 1 + 1 + \cdots + 1 + C_{worst}(1) \\ &= 1 + 1 + 1 + \cdots + 1 + 0 \end{aligned}$$

Quando chega em $C_{worst}(0)$, tivemos antes $j + 2$ somas de 1, logo:
$C_{worst}(n) = (j + 2) \cdot 1 + 0 = j + 2$

Para descobrir o valor de j, lembramos que:
$\frac{n-2^0-2^1-\cdots-2^{j+1}}{2^{j+2}} = 0$

$n = 2^0 + 2^1 + \cdots + 2^{j+1} = \sum_{i=0}^{j+1} 2^i = \frac{2^0(2^{j+2}-1)}{2-1} = 2^{j+2} - 1$

$n + 1 = 2^{j+2}$

Busca binária: complexidade

Aplicando *log* em ambos os lados:

$$
\begin{aligned}
log_2(n+1) &= log_2 2^{j+2} \\
&= (j+2) \cdot log_2 2 \\
&= (j+2)
\end{aligned}
$$

Portanto, $j = log_2(n+1) - 2$.

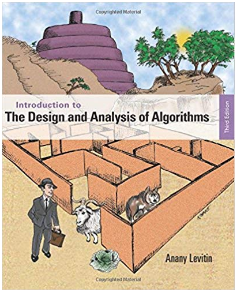
Substituindo em $C_{worst}(n) = (j+2)$, temos:
$C_{worst}(n) = log_2(n+1) - 2 + 2 = log_2(n+1)$

Ignorando constantes, concluimos que: $C_{worst}(n) \in O(log\ n)$.

- Busca binária é extremamente eficiente

Agenda

Bibliografia + leitura recomendada



Capítulo 3 (pp. 97–98, 104)
Capítulo 4 (pp. 150–152)
Anany Levitin.
Introduction to the Design and Analysis of Algorithms.
3a edição. Pearson. 2011.

BUSCA LINEAR | BUSCA BINÁRIA

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil