

Algoritmos de Procura e de Ordenação I

27/09/2023

Sumário

- Recap
- Ordens de complexidade
- Linear Search – Procura sequencial num **array não ordenado**
- Linear Search – Procura sequencial num **array ordenado**
- Binary Search – Procura binária
- Sugestão de leitura

Let's
RECAP




Recapitulação

Best case, Worst case, Average Case

$$B(n) = \min_{I \in D_n} t(I) \quad W(n) = \max_{I \in D_n} t(I)$$

$$A(n) = \sum_{I \in D_n} p(I) \times t(I)$$

Procura do maior elemento

```
int searchMax( int a[], int n ) {  
    int indexMax = 0;   
    for( int i=1; i<n; i++ ) {  
        if( a[i] > a[indexMax] ) {  (n - 1) comps.  
            indexMax = i;   
        }  
    }  
    return indexMax;  
}
```

Atribuições a indexMax:
 $B(n) = 1$
 $W(n) = n$
 $A(n) \approx n/2$

Notações habituais

notation	provides	example	shorthand for	used to
Big Theta	asymptotic order of growth	$\Theta(N^2)$	$\frac{1}{2} N^2$ $10 N^2$ $5 N^2 + 22 N \log N + 3N$ \vdots	classify algorithms
Big Oh	$\Theta(N^2)$ and smaller	$O(N^2)$	$10 N^2$ $100 N$ $22 N \log N + 3 N$ \vdots	develop upper bounds
Big Omega	$\Theta(N^2)$ and larger	$\Omega(N^2)$	$\frac{1}{2} N^2$ N^5 $N^3 + 22 N \log N + 3 N$ \vdots	develop lower bounds

[Sedgewick & Wayne]

Fizeram ?

- $T(n) = 10 n^2 + 100 n - 23$

$$T(n) \text{ ? } O(n^2)$$

$$T(n) \text{ ? } O(n^3)$$

$$T(n) \text{ ? } O(n)$$

$$T(n) \text{ ? } \Omega(n^2)$$

$$T(n) \text{ ? } \Omega(n^3)$$

$$T(n) \text{ ? } \Omega(n)$$

$$T(n) \text{ ? } \Theta(n^2)$$

$$T(n) \text{ ? } \Theta(n^3)$$

$$T(n) \text{ ? } \Theta(n)$$

Ordens de Complexidade

Ordens de Complexidade/Classes de Eficiência

- $O(1)$: constante
 - Que algoritmos?
- $O(\log n)$: logarítmico
 - E.g., **diminuir-para-reinar**
- $O(n)$: linear
 - Processar todos os elementos de um array, uma lista, etc.
- $O(n \log n)$: n-log-n
 - E.g., **dividir-para-reinar**

Ordens de Complexidade/Classes de Eficiência

- $O(n^k)$: polinomial (quadrático, cúbico, etc.)
 - k ciclos encastelados
- $O(2^n)$: exponencial
 - Gerar todos os subconjuntos de um conjunto com n elementos
- $O(n!)$: fatorial
 - Gerar todas as permutações de um conjunto com n elementos

Ordens de Complexidade/Classes de Eficiência

order of growth	name	typical code framework	description	example	$T(2N) / T(N)$
1	constant	<code>a = b + c;</code>	statement	add two numbers	1
$\log N$	logarithmic	<pre>while (N > 1) { N = N / 2; ... }</pre>	divide in half	binary search	~ 1
N	linear	<pre>for (int i = 0; i < N; i++) { ... }</pre>	loop	find the maximum	2
$N \log N$	linearithmic	[see mergesort lecture]	divide and conquer	mergesort	~ 2
N^2	quadratic	<pre>for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) { ... }</pre>	double loop	check all pairs	4
N^3	cubic	<pre>for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) for (int k = 0; k < N; k++) { ... }</pre>	triple loop	check all triples	8
2^N	exponential	[see combinatorial search lecture]	exhaustive search	check all subsets	$T(N)$



[Sedgewick & Wayne]

Exemplo – Contagem de operações

n	1	2	4	8	16	32	64	128	256
$M(n)$	1	3	10	36	136	528	2080	8256	32896

- $M(n)$: número de operações efetuadas
- Ordem de complexidade ?
- Expressão para o número de operações ?

Outro exemplo

n	1	2	3	4	5	6	7	8	9	10
$M(n)$	1	3	7	15	31	63	127	255	511	1023

- $M(n)$: número de operações efetuadas
- Ordem de complexidade ?
- Expressão para o número de operações ?

Tempo de execução – Estimativas

order of growth of time		for a program that takes a few hours for input of size N			
description	function	2x factor	10x factor	predicted time for $10N$	predicted time for $10N$ on a 10x faster computer
<i>linear</i>	N	2	10	a day	a few hours
<i>linearithmic</i>	$N \log N$	2	10	a day	a few hours
<i>quadratic</i>	N^2	4	100	a few weeks	a day
<i>cubic</i>	N^3	8	1,000	several months	a few weeks
<i>exponential</i>	2^N	2^N	2^{9N}	never	never

Predictions on the basis of order-of-growth function

[Sedgewick & Wayne]

Linear Search

– Array não ordenado

Procura sequencial – Comparações ?

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



$B(n) = 1$

$W(n) = n$

$A(n) = ?$

Caso Médio – Abordagem estruturada

- Estabelecer um **cenário**
- Identificar a **operação básica** a contar
- Identificar os **casos/configurações possíveis**
 - **Quantos** são ?
- Contar o **nº de operações** realizadas para cada um dos casos
- Atribuir uma **probabilidade** a cada um dos casos possíveis
- Construir uma **tabela**
- Calcular o **nº de operações para o caso médio**

1 – Valor procurado pertence ao array

- O valor procurado **pertence ao array**
- Contar o **número de comparações** necessárias para o encontrar
- Valor encontrado na **1ª posição**, ou na **2ª posição**, ou ...
- Necessária **1 comparação**, ou **2 comparações**, ou ...
- Situações **equiprováveis**
- Tabela ?

1 – Elemento procurado pertence ao array

Casos possíveis		Nº de comparações	Probabilidade
l_0	É o 1º elemento	1	$1/n$
l_1	É o 2º elemento	2	$1/n$
l_2	É o 3º elemento	3	$1/n$
...
l_{n-1}	É o último elemento	n	$1/n$

$$A(n) = \sum_{i=0}^{n-1} \frac{1}{n} \times (i + 1) = \frac{1}{n} \sum_{i=0}^{n-1} (i + 1) = \frac{1}{n} \times \frac{n \times (n + 1)}{2} = \frac{n + 1}{2} \approx \frac{n}{2}$$

2 – Valor procurado pode não pertencer !

- O valor procurado **pertence** ao array com **probabilidade p**
- O valor procurado **não pertence** ao array com **probabilidade $(1 - p)$**
- Quantos casos são ?
- Contar o **número de comparações** para cada um dos casos
- Valor encontrado na **1ª posição**, ou na **2ª posição**, ou ...
- Necessária **1 comparação**, ou **2 comparações**, ou ...
- Que **probabilidade** atribuir a **cada caso** ?
- Tabela ?

2 – Valor procurado pode não pertencer !

Casos possíveis		Nº de comparações	Probabilidade
l_0	É o 1º elemento	1	p/n
l_1	É o 2º elemento	2	p/n
l_2	É o 3º elemento	3	p/n
...
l_{n-1}	É o último elemento	n	p/n
l_n	Não encontrado !	n	$(1 - p)$

$$A(n) = \sum_{i=0}^{n-1} \frac{p}{n} \times (i + 1) + (1 - p) \times n = \frac{p \times (n + 1)}{2} + (1 - p) \times n$$

2 – Valor procurado pode não pertencer !

$$A(n) = \frac{p \times (n + 1)}{2} + (1 - p) \times n$$

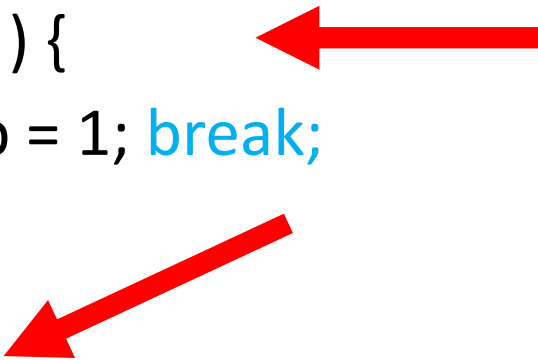
- Se $p = 1$, então $A(n) = (n + 1) / 2 \approx n / 2$
- Se $p = 50\%$, então $A(n) = (n + 1) / 4 + n / 2 \approx 3 \times n / 4$
- Se $p = 25\%$, então $A(n) = (n + 1) / 8 + 3 \times n / 4 \approx 7 \times n / 8$

Linear Search

- Array ordenado

Procura sequencial – Comparações ?

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



// Ordem da conjunção !!

Melhor Caso e Pior Caso

- Valor procurado pode **pertencer** ou **não pertencer** ao array
- $B(n) = 2$ - Quando ?
- $W(n) = (n + 1)$ - Quando ?

Melhor Caso e Pior Caso

- Valor procurado pode **pertencer** ou **não pertencer** ao array
- $B(n) = 2$
 - Valor procurado **é igual ao 1º elemento**
OU
 - Valor procurado **é menor do que o 1º elemento**
- $W(n) = (n + 1)$
 - Valor procurado **é igual ao último elemento**
OU
 - Valor procurado **está entre o penúltimo e o último**

Caso Médio

Casos possíveis		Nº de comparações	Probabilidade
Sucesso 0	É o 1º elemento		
Sucesso 1	É o 2º elemento		
...	...		
Sucesso (n – 1)	É o último elemento		
Insucesso 0	Menor do que o 1º		
Insucesso 1	Entre o 1º e o 2º		
...	...		
Insucesso (n – 1)	Entre o penúltimo e o último		
Insucesso n	Maior do que o último		

- Quantos são os casos possíveis ? Quantas comparações em cada um ?

Caso Médio

Casos possíveis		Nº de comparações	Probabilidade
Sucesso 0	É o 1º elemento	2	
Sucesso 1	É o 2º elemento	3	
...	
Sucesso (n - 1)	É o último elemento	n + 1	
Insucesso 0	Menor do que o 1º	2	
Insucesso 1	Entre o 1º e o 2º	3	
...	
Insucesso (n - 1)	Entre o penúltimo e o último	n + 1	
Insucesso n	Maior do que o último	n	

- Que probabilidade associar a cada caso possível ?

Caso Médio

Casos possíveis		Nº de comparações	Probabilidade
Sucesso 0	É o 1º elemento	2	$1 / (2n + 1)$
Sucesso 1	É o 2º elemento	3	$1 / (2n + 1)$
...
Sucesso (n - 1)	É o último elemento	n + 1	$1 / (2n + 1)$
Insucesso 0	Menor do que o 1º	2	$1 / (2n + 1)$
Insucesso 1	Entre o 1º e o 2º	3	$1 / (2n + 1)$
...
Insucesso (n - 1)	Entre o penúltimo e o último	n + 1	$1 / (2n + 1)$
Insucesso n	Maior do que o último	n	$1 / (2n + 1)$

- $A(n) = ?$

Caso Médio

$$A(n) = \frac{1}{2n+1} \left\{ \left(\sum_{i=0}^{n-1} (i+2) \right) + \left(\sum_{i=0}^{n-1} (i+2) \right) + n \right\}$$

$$A(n) = \frac{1}{2n+1} \left\{ \frac{n \times (n+3)}{2} + \frac{n \times (n+3)}{2} + n \right\}$$

$$A(n) \approx \frac{n}{2}$$

- Comparar com o cenário do array não ordenado
- É melhor ou pior ?

Tarefa 1

- Considerar uma **probabilidade de 25%** de o valor procurado pertencer ao array
- Calcular o **nº de comparações** associadas ao **caso médio** para este cenário

Binary Search

– Array ordenado

Procura binária

0	1	2	3	4	5
2	4	6	8	10	12

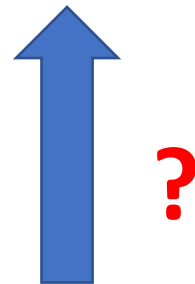
- O valor **2** pertence ao array ?
- Qual é o **primeiro elemento** consultado ?

Procura binária

0	1	2	3	4	5
2	4	6	8	10	12

- O valor **2** pertence ao array ?

0	1	2	3	4	5
2	4	6	8	10	12



Procura binária

0	1	2	3	4	5
2	4	6	8	10	12

- O valor **2** pertence ao array ?

0	1	2	3	4	5
2	4	6	8	10	12

0	1
2	4

Qual é o **próximo elemento** consultado?

Procura binária

0	1	2	3	4	5
2	4	6	8	10	12

- O valor **2** pertence ao array ?

0	1	2	3	4	5
2	4	6	8	10	12

0	1
2	4

Encontrado !!



Procura binária


- O valor **13** pertence ao array ?

0	1	2	3	4	5
2	4	6	8	10	12

Procura binária

- O valor **13** pertence ao array ?

0	1	2	3	4	5
2	4	6	8	10	12



Procura binária

- O valor **13** pertence ao array ?

0	1	2	3	4	5
2	4	6	8	10	12

3	4	5
8	10	12

Procura binária

- O valor **13** pertence ao array ?

0	1	2	3	4	5
2	4	6	8	10	12

3	4	5
8	10	12



Procura binária

- O valor **13** pertence ao array ?

0	1	2	3	4	5
2	4	6	8	10	12

3	4	5
8	10	12

5
12



Procura binária

- O valor **13** pertence ao array ?

0	1	2	3	4	5
2	4	6	8	10	12

3	4	5
8	10	12

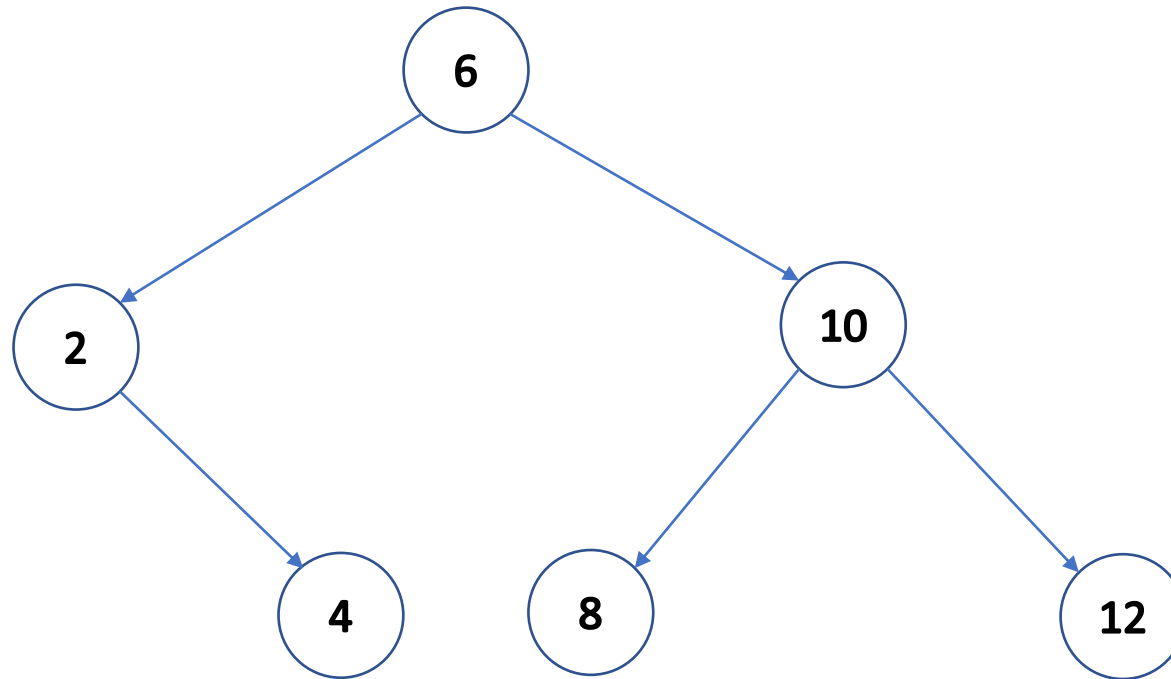
5
12

Não encontrado !!






Árvore binária

- A **representação gráfica como árvore binária** auxilia a compreensão do funcionamento do algoritmo



Procura binária – Nº de iterações do ciclo ?

```
int binSearch( int a[], int n, int x ) {  
    int left = 0; int right = n - 1;  
    while( left <= right ) {  
        int middle = (left + right) / 2;  
        if(a[middle] == x) return middle;  
        if(a[middle] > x) right = middle - 1;  
        else left = middle + 1;  
    }  
    return -1;  
}
```



Tarefa 2

- **Simule** o funcionamento do algoritmo para arrays ordenados com um **maior número de elementos**
- Por exemplo, para **$n = 7$** e **$n = 15$**

Tarefa 3

- Habitualmente expressa-se o **esforço computacional** do algoritmo de procura binária através do **número de iterações do ciclo**
- **Melhor caso : Quando ?** Quantas **iteraões** ? Quantas **comparaões** ?
- **Pior caso : Quando ?** Quantas **iteraões** ? Quantas **comparaões** ?

Sugestão de leitura

Sugestão de leitura

- J. J. McConnell, Analysis of Algorithms, 1st Edition, 2001
 - Capítulo 2: secções 2.1, 2.2