Pesquisa sequencial e pesquisa binária

Armando Matos

Departamento de Ciência de Computadores
Universidade de Porto

2008

2 problemas importantes...

- Pesquisa: Procurar um valor numa lista
  ou, por exemplo, num ficheiro
  pesquisa `5` em `[8,2,1,5,2,6]` $\longrightarrow$ True
- Ordenação: ordenar uma lista.
  Exemplo, ordem não decrescente:
  `[8,2,1,5,2,6]` $\longrightarrow$ `[1,2,2,5,6,8]`

Outline

Pesquisa sequêncial

Pesquisa sequêncial

Seja

- `x`: o valor que se vai procurar
- *a*: a lista onde se vai procurar `x`;
  índices de `a`: 0 a $n-1$.

`procura(x,a)` $\Rightarrow$ `bool`

Método:

`x` é comparado sucessivamente com `a[0]`, `a[1]`,..., `a[n-1]`.
Se `x` for igual a algum dos `a[i]`, retorna-se `True`.
Senão, retorna-se `False`.

Pesquisa sequêncial, função em python

```
# procura x na lista a
def procura(x,a):
    n=len(a)
    i=0
    while i <= n-1:
        if x==a[i]:
            return True
        i=i+1
    return False
```

Pesquisa sequêncial, método da sentinela

Uma variação do método anterior... função `procura(x,a)`:

- Inicialmente `x` é colocado no fim da lista,
  Ex: `x=9`, `a=[5,10,2,4,2]` $\longrightarrow$ `a=[5,10,2,4,2,9]`
- A comparação é mais simples: `enquanto x != a[i]`
- A parte interna do ciclo é apenas: `i = i+1`
- Seja `n` o valor inicial de `len(a)`.
  Se o ciclo acaba com `i=n`, retorna `False`
  Se o ciclo acaba com `i<n`, retorna `True`

Método da sentinela, função em python

Nesta versão retorna-se `i` tal que `x=a[i]` ou `-1` se `x` não está na lista.

```
1  def procura(x,a):
2      n=len(a)
3      a.append(x)
4      i=0
5      while x!=a[i]:
6          i=i+1
7      if i==n:
8          return -1
9      return i
```

Note-se que o ciclo (onde a função "passa" mais tempo) é constituído apenas pelas linhas 5 e 6.

Eficiência da pesquisa sequêncial

Vamos usar como medida de eficiência no tempo de execução

o número $c(n)$ de comparações entre `x` e `a[i]`

(linha 5 da função anterior)

Eficiência da pesquisa sequêncial, I

Para a função com "sentinela":
Número de comparações, se `x` está na lista

- Melhor caso: $c(n) = 1$
- Pior caso: $c(n) = n$
- Caso médio: $c(n) = (n + 1)/2$

Para o caso médio, admitiu-se que `x` pode ser, com igual probabilidade, um dos `n` elementos da lista.

Eficiência da pesquisa sequêncial, II

Número de comparações, Se `x` não está na lista:

- Melhor caso: $c(n) = n + 1$
- Pior caso: $c(n) = n + 1$
- Caso médio: $c(n) = n + 1$

Pesquisa binária

Pesquisa binária I

Se a lista `a` está ordenada – suponhamos por ordem crescente – há algoritmos muito mais eficientes de procurar `x`:
`procura(x,a)`:

- `x` é procurado entre os índices `i1` e `i2` (extremos incluídos); Seja `n=len(a)`. Inicialmente faz-se `i1=0`, `i2=n-1`.
- Repetidamente, calcula-se o ponto médio `m=(i1+i2)/2` (divisão inteira) e
  - Se `x<a[m]` $\longrightarrow$ `i2=m-1`
  - Se `x>a[m]` $\longrightarrow$ `i1=m+1`
  - Se `x==a[m]` $\longrightarrow$ `retorna m`
- Se o intervalo `[i1,i2]` acaba por ficar vazio (`i1>i2`), `x` não está na lista: $\longrightarrow$ `retorna -1`.

Pesquisa binária, função em python, versão iterativa

```
 1  def pb(x,a):
 2      n = len(a)
 3      i1 = 0
 4      i2 = n-1
 5  (*) while i1 <= i2:
 6          m = (i1+i2)/2
 7          if x < a[m]:
 8              i2 = m-1
 9          elif x > a[m]:
10              i1 = m+1
11          else:
12              # verifica-se x == a[m]
13              return m
14      return -1
```

Notas sobre a correcção de algoritmos

Exercício: Escreva um invariante de ciclo apropriado à demonstração dacorrecção do algoritmo e válido antes do teste do ciclo - linha (*).
Exercício: Escreva uma versão recursiva do mesmo algoritmo.

Notas sobre a correcção de algoritmos

Muitas vezes é vantajoso dividir as demonstrações de correcção em 2 partes:

- Correcção parcial de um algoritmo: se o algoritmo terminar, a resposta está correcta
- Terminação de um algoritmo: o algoritmo termina (ver La Palisse, obras completas).

$$\text{correcção} = \text{correcção parcial} + \text{terminação}$$

Pesquisa binária – correcção da função

Notas sobre a correcção da função de pesquisa binária:
Invariante de ciclo: se `x` está na lista está entre os índices `i1` e `i2`, extremos incluídos.

- Início: o invariante de ciclo fica verdadeiro com as atribuições das linhas 2-4.
- O invariante de ciclo mantém-se verdadeiro se o ciclo continuar, isto é, quando se verificam as condições das linhas 7 ou 9.
- Logo: no teste da linha 5 verifica-se o invariante de ciclo.
- Se a linha 13 for executada, `x` está na lista, na posição `m`
- Se o ciclo acabar (`i1>i2`), pela manutenção do invariante do ciclo, `x` não está na lista.

Pesquisa binária – terminação da função

Notas sobre a terminação da função:
Os índices em que `x` pode "estar" são `i1`, `i1+1`,. . . , `i2`.
Seja `num=i2-i1+1` o número desses índices.
A prova de terminação é baseada no facto de `num` diminuir de cada vez que o teste das linhas 7 e 9 é efectuado e isso prova-se com os seguintes factos

- $\texttt{i1} \leq \texttt{m} \leq \texttt{i2}$ onde `m` resulta da atribuição `m=(i1+i2)/2`.
- A atribuição `i2=m-1` (linha 8) reduz `num`, pois, com o valor inicial da `i2`, $\texttt{m} - 1 < \texttt{m} \leq \texttt{i2}$
- A atribuição `i1=m+1` (linha 10) reduz `num`, pois, com o valor inicial da `i1`, $\texttt{i1} \leq \texttt{m} < \texttt{m} + 1$

Eficiência da pesquisa binária

Vamos usar como medida de eficiência no tempo de execução

o número $c(n)$ de comparações entre `x` e `a[i]`, contando as linhas 7 e 9 como uma só comparação, (comparação entre `x` e `a[m]`)

Vamos determinar o número de comparações no pior caso (quando `x` não está na lista)

Eficiência da pesquisa binária

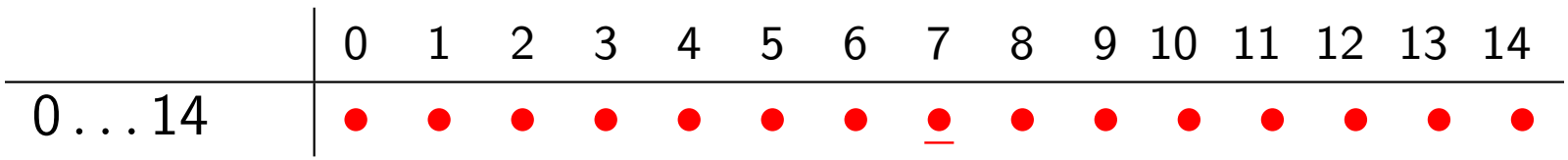
Para simplificar, supomos que `n=len(a)` é da forma $\mathtt{n} = 2^p - 1$.
Após uma divisão a "meio", o número de índices em que `x` pode estar passa de $\mathtt{n} = 2^p - 1$ a $\mathtt{n} = 2^{p-1} - 1$.
Por exemplo,

$$15 \longrightarrow 7 \longrightarrow 3 \longrightarrow 1 \longrightarrow 0$$

Eficiência da pesquisa binária

Para simplificar, supomos que `n=len(a)` é da forma $\mathtt{n} = 2^p - 1$.
Após uma divisão a "meio", o número de índices em que `x` pode estar passa de $\mathtt{n} = 2^p - 1$ a $\mathtt{n} = 2^{p-1} - 1$.
Por exemplo,

$$15 \longrightarrow 7 \longrightarrow 3 \longrightarrow 1 \longrightarrow 0$$



Eficiência da pesquisa binária

Para simplificar, supomos que `n=len(a)` é da forma $n = 2^p - 1$.
Após uma divisão a "meio", o número de índices em que `x` pode estar passa de $n = 2^p - 1$ a $n = 2^{p-1} - 1$.
Por exemplo,

$$15 \longrightarrow 7 \longrightarrow 3 \longrightarrow 1 \longrightarrow 0$$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0...14	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
8...14	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

Eficiência da pesquisa binária

Para simplificar, supomos que `n=len(a)` é da forma $\mathtt{n} = 2^p - 1$.
Após uma divisão a "meio", o número de índices em que `x` pode estar passa de $\mathtt{n} = 2^p - 1$ a $\mathtt{n} = 2^{p-1} - 1$.
Por exemplo,

$$15 \longrightarrow 7 \longrightarrow 3 \longrightarrow 1 \longrightarrow 0$$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0...14	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
8...14	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
8...10	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

Eficiência da pesquisa binária

Para simplificar, supomos que `n=len(a)` é da forma $\mathtt{n} = 2^p - 1$.
Após uma divisão a "meio", o número de índices em que `x` pode estar passa de $\mathtt{n} = 2^p - 1$ a $\mathtt{n} = 2^{p-1} - 1$.
Por exemplo,

$$15 \longrightarrow 7 \longrightarrow 3 \longrightarrow 1 \longrightarrow 0$$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0...14	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
8...14	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
8...10	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
9...9	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

Eficiência da pesquisa binária

Para simplificar, supomos que `n=len(a)` é da forma $\mathtt{n} = 2^p - 1$.
Após uma divisão a "meio", o número de índices em que `x` pode estar passa de $\mathtt{n} = 2^p - 1$ a $\mathtt{n} = 2^{p-1} - 1$.
Por exemplo,

$$15 \longrightarrow 7 \longrightarrow 3 \longrightarrow 1 \longrightarrow 0$$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0...14	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
8...14	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
8...10	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
9...9	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
9...8 = ∅	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

Eficiência da pesquisa binária

Para simplificar, supomos que `n=len(a)` é da forma $\mathtt{n} = 2^p - 1$.
Após uma divisão a "meio", o número de índices em que `x` pode estar passa de $\mathtt{n} = 2^p - 1$ a $\mathtt{n} = 2^{p-1} - 1$.
Por exemplo,

$$15 \longrightarrow 7 \longrightarrow 3 \longrightarrow 1 \longrightarrow 0$$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0...14	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
8...14	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
8...10	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
9...9	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
9...8 = ∅	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
0...14															

O número de divisões a "meio", que é igual ao número de comparações, é, neste caso, igual a $4 = \log(\mathtt{n} + 1)$.
No caso geral, o número de comparações não excede $\log(\mathtt{n}) + 1$.

Eficiência da pesquisa binária

Número de comparações no pior caso.
Exemplo para $\mathtt{n} = 10^8$, $t = 1\mu$seg (tempo associado a cada comparação)

- Pesquisa sequêncial: $c(\mathtt{n}) = \mathtt{n}$, $t = 100$seg, quase 2 minutos.
- Pesquisa binária: $c(n) \approx \log(\mathtt{n})$, $t \approx 27\mu$seg!

Conclusão: a pesquisa binária é muito mais rápida que a pesquisa sequêncial.

fim