AULA 13

Busca em vetor ordenado

PF 7.1 a 7.8

http://www.ime.usp.br/~pf/algoritmos/aulas/bub

Busca em vetor ordenado

Um vetor v[0 . . n−1] é **crescente** se

$$v[0] \le v[1] \le v[2] \cdots \le v[n-1].$$

Problema: Dado um número x e um vetor crescente v[0 . . n−1] encontrar um índice m tal que v[m]==x.

Entra: x == 50

	0							7			n−1
v	10	20	25	35	38	40	44	50	55	65	99

Sai: m == 7

Busca em vetor ordenado

Entra: x == 57

	0										n−1
v	10	20	25	35	38	40	44	50	55	65	99

Sai: m == -1 (x não está em v)

Busca sequencial

```
int buscaSequencial(int x, int n, int v[])
 {
   int m = 0;
   while (/*1*/ m < n && v[m] < x) ++m;
   if (m < n && v[m] == x)
       return m;
   return -1;
 }
```

Exemplo

x == 55

	0										10
v	10	20	25	35	38	40	44	50	55	65	99

	m										10
v	10	20	25	35	38	40	44	50	55	65	99

	0	m									10
v	10	20	25	35	38	40	44	50	55	65	99

	0		m								10
v	10	20	25	35	38	40	44	50	55	65	99

	0			m							10
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 55

	0				m						10
v	10	20	25	35	38	40	44	50	55	65	99

	0					m					10
v	10	20	25	35	38	40	44	50	55	65	99

	0						m				10
v	10	20	25	35	38	40	44	50	55	65	99

	0							m			10
v	10	20	25	35	38	40	44	50	55	65	99

	0								m		10
v	10	20	25	35	38	40	44	50	55	65	99

Correção

Relação **invariante** chave:

(i0) em /*1*/ vale que: $v[m-1] < x$. ♡

x == 55

	0							m			101
v	10	20	25	35	38	40	44	50	55	65	99

A relação (i0) vale no começo da primeira iteração se supusermos que $v[-1] = -\infty$.

No início da última iteração $m \geq n$ ou $v[m] \geq x$.

Portanto, se a função devolve -1, então x não está em $v[0\,.\,.\,n-1]$

Consumo de tempo buscaSequencial

Se a execução de cada linha de código consome 1 **unidade** de tempo o consumo total é:

linha	todas as execuções da linha		
1	$=$	1	$= 1$
2	$\leq$	$n+1$	$\approx n$
3	$=$	1	$= 1$
4	$\leq$	1	≤ 1
5	$\leq$	1	≤ 1
total	$\leq$	$n+3$	$= O(n)$

Conclusão

O consumo de tempo do algoritmo buscaSequencial no pior caso é proporcional a n.

O consumo de tempo do algoritmo buscaSequencial é $O(n)$.

Busca binária

```
int buscaBinaria(int x, int n, int v[]) {
   int e, m, d;
1  e = 0; d = n-1;
2  while (/*1*/ e <= d) {
3     m = (e + d)/2;
4     if (v[m] == x) return m;
5     if (v[m] < x) e = m + 1;
6     else d = m - 1;
   }
7  return -1;
}
```

Exemplo

x == 48

	0										10
v	10	20	25	35	38	40	44	50	55	65	99

	e										d
v	10	20	25	35	38	40	44	50	55	65	99

	e					m					d
v	10	20	25	35	38	40	44	50	55	65	99

	0						e				d
v	10	20	25	35	38	40	44	50	55	65	99

	0						e		m		d
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 48

	0						e	d			10
v	10	20	25	35	38	40	44	50	55	65	99

							m				
	0						e	d			10
v	10	20	25	35	38	40	44	50	55	65	99

								e			
	0							d			10
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 48

								m			
								e			
	0							d			10
v	10	20	25	35	38	40	44	50	55	65	99

	0						d	e			10
v	10	20	25	35	38	40	44	50	55	65	99

Correção

Relação **invariante** chave:

(i0) em /*1*/ vale que: $v[e-1] < x < v[d+1]$. ♡

x == 48

	0					e		d			n-1
v	10	20	25	35	38	40	44	50	55	65	99

A relação (i0) vale no começo da primeira iteração se supusermos que $v[-1] = -\infty$ e $v[n] = +\infty$.

Correção

Relação **invariante** chave:

(i0) em /*1*/ vale que: $v[e-1] < x < v[d+1]$. ♡

x == 48

	0					e		d			n-1
v	10	20	25	35	38	40	44	50	55	65	99

No início da última iteração quando $e > d$ nenhum elemento é "$> v[e-1]$" e "$< v[d+1]$", pois o vetor é crescente (!). Logo, x não está em $v[0\,.\,.\,n-1]$ e função devolve -1

Correção

Relação **invariante** chave:

(i0) em /*1*/ vale que: $v[e-1] < x < v[d+1]$. ♡

x == 48

	0					e		d			n-1
v	10	20	25	35	38	40	44	50	55	65	99

O valor de $d - e$ diminui a cada iteração. Portanto, se a função não encontra m tal que `v[m]==x`, então a função para quando $d - e < 0$.

Consumo de tempo buscaBinaria

O consumo de tempo da função `buscaBinaria` é proporcional ao número k de iterações do `while`.

No início da 1a. iteração tem-se que $d - e = n - 1 \approx n$.

Sejam

$$(e_0, d_0), (e_1, d_1), \ldots, (e_k, d_k),$$

os valores das variáveis e e d no início de cada uma das iterações.
Assim, $d_{k-1} - e_{k-1} \geq 0$ e $d_k - e_k < 0$

Número iterações

Estimaremos o valor de k em função de $d - e$.

Note que $d_{i+1} - e_{i+1} \leq (d_i - e_i)/2$ para $i=1,2,\ldots,k-1$.

Desta forma tem-se que

$$\begin{array}{lclcll}
d_0 - e_0 & = & n-1 & < & n & \\
d_1 - e_1 & \leq & (d_0 - e_0)/2 & < & n/2 & \\
d_2 - e_2 & \leq & (d_1 - e_1)/2 & < & (n/2)/2 & = n/2^2 \\
d_3 - e_3 & \leq & (d_2 - e_2)/2 & < & (n/2^2)/2 & = n/2^3 \\
d_4 - e_4 & \leq & (d_3 - e_3)/2 & < & (n/2^3)/2 & = n/2^4 \\
\vdots & & \vdots & & \vdots & \quad \vdots
\end{array}$$

Número iterações

Percebe-se que depois de cada iteração o valor de $d - e$ é reduzido pela metade.
Seja t o número inteiro tal que

$$2^t \leq n < 2^{t+1}$$

Da primeira desigualdade temos que

$$t \leq \lg n,$$

onde $\lg n$ denota o logaritmo de n na base 2.

Número iterações

Da desigualde estrita, concluímos que

$$0 \leq (d_{k-1} - e_{k-1})/2^{k-1} < \underline{n}/2^{k-1} < \underline{2^{t+1}}/2^{k-1}.$$

Assim, em particular temos que

$$1 \leq 2^{t+1}/2^{k-1}$$

ou, em outras palavras

$$k \leq t + 2.$$

Portanto , o número k de iterações é não superior a

$$t + 2 \leq \lg n + 2.$$

Conclusão

O consumo de tempo do algoritmo `buscaSequencial` no pior caso é proporcional a $\lg n$.

O consumo de tempo do algoritmo `buscaSequencial` é $O(\lg n)$.

Número de iterações

`buscaSequencial`	`buscaBinaria`
n	$\lg n$
256	8
512	9
1024	10
2048	11
4096	12
8192	13
16384	14
32768	15
65536	16
262144	18
1048576	20
⋮	⋮
4294967296	32

Versão recursiva da busca binária

Para formular uma versão recursiva é necessário generalizar um pouco o problema trocando $v[0\,.\,.\,n-1]$ por $v[e\,.\,.\,d]$.

```
int buscaBinaria(int x, int n, int v[])
 {
   return buscaBinariaR(x,0,n-1,v);
 }
```

Versão recursiva da busca binária

Recebe um vetor crescente v[e..d] e devolve um índice m tal que v[m] == 1. Se tal m não existe, devolve -1.

```
int
buscaBinariaR(int x,int e,int d,int v[]) {
   int m;
   if (d < e) return -1;
   m = (e + d)/2;
   if (v[m] == x) return m;
   if (v[m] < x)
      return buscaBinariaR(x,m+1,d,v);
   return buscaBinariaR(x,e,d-1,v);
}
```

Outra versão recursiva

A função abaixo não resolve o problema...

Por quê? Como consertar?

```
int
buscaBinariaR(int x,int n, int *v) {
  int m;
  if (n == 0) return -1;
  m = n/2;
  if (v[m] == x) return m;
  if (v[m] < x)
     return buscaBinariaR(x,n-m-1,&v[m+1]);
  return buscaBinariaR(x,m,v);
}
```

Outra versão recursiva

Observações:

- As declarações `int v[]` e `int *v` no protótipos de funções são equivalentes. Abaixo escolhemos `int *v` apenas para deixar **mais explicito** que em ambos os casos o que está sendo passado como parâmetro é um **endereço**(!).
- As expressões "`&v[m+1]`" e "`v+m+1`" são equivalentes (=tem o memo valor =representam o mesmo endereço).
- Tem um problema ...