começar pela contagem de iteração de laços mais limpos!!!

Análises de Corretude e Eficiência (Contagem de Operações)

Dados um vetor v de tamanho n e um elemento x,

- temos de devolver a posição de x em v ou -1 se não encontrarmos.

Busca sequencial / linear

- A ideia é percorrer o vetor verificando se x é o elemento de cada posição.

Contando operações se	*encontrar x*	*não encontrar*
buscaSeq(v[ ], n, x):		
i = 0	1	1
enquanto (i < n E v[i] != x):	4 * pi	4 * (n + 1)
i = i + 1	2 * (pi - 1)	2 * n
se (i < n):	1	1
devolva i		
devolva -1	1	1
pi (núm. de iter. até encontrar x)	*1 + 6 * pi*	*7 + 6 * n*
1	7	
n/2	1 + 3 * n	
n	1 + 6 * n	

melhor caso O(1)

caso médio O(n)

Pior caso O(n)

Corretude e prova por indução

- Descobrindo invariantes: que propriedade o alg. mantém a cada iteração?
- O invariante principal do algoritmo é que,
  - no início da i-ésima iteração do laço, o vetor v[0 .. i - 1] não contém x.
- Caso base: No início o invariante vale trivialmente,
  - pois i = 0 e v[0 .. i - 1] é vazio.
- H.I.: Supomos que o invariante vale no início da i-ésima iteração.
- Passo: Seja i' = i + 1 o valor de i ao final da iteração atual.
  - Queremos mostrar que P(i') vale ao final da iteração atual.
- No início da i-ésima iteração sabemos, pela H.I., que x != v[0 .. i - 1]
- Se v[i] == x o laço é interrompido e i não é mais incrementado.
- Caso contrário, sabemos que x != v[i]. Junto da H.I., temos que x != v[0 .. i].
  - No fim da iteração, i é incrementado e o invariante restaurado.

Esse invariante garante que o algoritmo está correto pois,

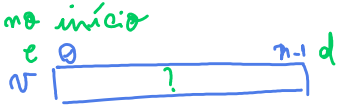
- se o algoritmo sair do laço por violar a primeira condição (i < n),
  - o invariante garante que v[0 .. n - 1] não contém x.
- Caso contrário, a violação da segunda condição (v[i] != x)
  - garante que o algoritmo devolve a posição do primeiro x encontrado.

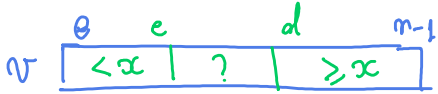
Quiz1: se fosse um algoritmo que encontra o máximo,

  - qual seria o invariante principal?

- Quiz2: e qual seria a diferença entre o melhor e o pior caso deste algoritmo?

Busca binária





Algoritmo iterativo de busca binária em vetor ordenado.

Seja p o # de iter. do laço principal

	Contando operações
buscaBin(v[ ], n, x):	
e = -1	1
d = n	1
enquanto (e < d - 1):	2 * (p + 1)
m = (e + d) / 2	3 * p
se v[m] < x:	2 * p
e = m	
senão /* v[m] >= x */ d = m	1 * p
se v[d] == x: devolva d	
devolva -1	3
	*7 + 8 * p*

# iter	|?|
0	n
1	n/2
2	n/4
3	n/8
⋮	⋮
p-1	$n/2^{p-1} = 1$
p	$\lfloor n/2^p \rfloor = 0$

$2^{p-1} = n$

$p - 1 = \lg n$

Quanto vale p? Note que, a cada iteração,

- o tamanho do subvetor entre e e d diminui pela metade.

- Assim, depois de p iterações temos
  - n / 2^(p - 1) = 1 → p = 1 + lg n
- Portanto, número de operações é 15 + 8 lg n

como a divisão é inteira na penúltima iter. |?| = d - e - 1 = 0

Corretude e prova por indução

- O invariante principal P(l) é que no início de cada iteração
  - v[0 .. e] < x <= v[d .. n - 1]
- Caso base: o invariante vale no início da primeira iteração,
  - pois os subvetores começam vazios.
- H.I.: Supomos que o invariante vale no início da l-ésima iteração,
  - i.e., P(l) é verdadeira por hipótese.
- Passo: Seja e' e d', respectivamente,
  - os valores de e e d no início da iteração l+1.
    - Queremos mostrar que P(l+1) vale.
- No início da i-ésima iteração sabemos, pela H.I.,
  - que v[0 .. e] < x <= v[d .. n - 1]
- Na iteração atual temos dois casos:
  - **a)** v[m] < x => e' = m, d' = d; **b)** v[m] >= x => e' = e, d' = m.
- Se ocorreu a) temos que a própria condição junto de v[] estar ordenado garante que x > v[0 .. m] = v[0 .. e'] e a H.I. garante que x <= v[d' .. n - 1].
- Se ocorreu b) condição + ordenação garantem que x <= v[m .. n-1] = v[d' .. n-1] e a H.I. garante que x > v[0 .. e'].

Esse invariante garante que quando o algoritmo sai do laço temos e = d - 1.
  - Assim, v[0 .. d-1] < x <= v[d .. n - 1].
- Portanto, a posição em que x pode estar, dado que v[] está ordenado, é d.

Busca por elemento repetido

Deixar como exercício por falta de Tempo !!.

Quiz3: o que faz o seguinte algoritmo?

Contando operações se	*não encontrar*
incognito(v[ ], n):	
j = n	1
para (i = 0; i < n E j == n; i++):	1 + 4 * (n + 1)
para (j = i + 1; j < n E v[j] != v[i]; j++)	2n + sum_k=1..n 6 (n - k)
	= 2n + 6 n (n - 1) / 2
se j < n:	1
devolva (i - 1, j)	
devolva -1	2

Quiz4: Qual a eficiência deste algoritmo?

Quiz5: Como resolver esse problema de modo mais eficiente?

Quiz6: Qual o principal invariante do laço externo?

$(n-1)-0+1 = n$ iter.

Para $i = 0$ até $n-1$:
⋮ $v[i] = i+1$

Tempo — contagem e corretude via invariantes c/ esses exemplos/análises!!!

$$c\underbrace{(1+1+1+1+\dots+1)}_{n \text{ vezes}} = cn = O(n)$$

```
aux = 0; i = n
while (i ≥ 0):
⋮   aux += i
⋮   i -= 1
```

$n-0+1 = n+1$ iter.

$$c\underbrace{(1+1+1+\dots+1)}_{n+1 \text{ vezes}} = c(n+1) = cn+c = O(n)$$

$-0+1 = n+1$ iter

```
para i = 0 até n:
⋮   para j = 1 até n:
⋮   ⋮   v[i][j] = 0
```

$n-1+1 = n$ iter.

$$c\underbrace{(n+n+n+\dots+n)}_{n+1 \text{ vezes}} = c\cdot(n+1)n = cn^2+cn = O$$

$0+1 = n$

```
para i = 0 até n-1:
⋮   para j = i+1 até n-1:
⋮   ⋮   v[i][j] = 1
```

Quiz!: O que acontece na última iteração do laço externo?

$n-1-(i+1)+1$
$= n-i-1$

$$c\underbrace{((n-1)+(n-2)+(n-3)+\dots+1+0)}_{n \text{ parcelas}}$$
$$= c(n-1)n/2 = c\cdot\frac{n^2}{2} - c\frac{n}{2} = O(n^2)$$

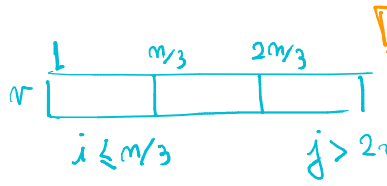
Análise simplificada:

$$\sum_{i=0}^{n-1}\sum_{j=i+1}^{n-1} c \geq \sum_{i=0}^{n/2-1}\sum_{j=n/2}^{n-1} c = \sum_{i=0}^{n/2-1} \frac{n}{2}\cdot c = \frac{n}{2}\cdot\frac{n}{2}\cdot c = \frac{c}{4}\cdot n^2 = \Omega(n^2)$$

Note que limitar superiormente por $O(n^2)$ é simples

considere apenas $i < n/2$ e $j \geq n/2$

```
aux = 0
para i = 1 até n:
⋮   para j = i+1 até n:
⋮   ⋮   para k = i+1 até j:
⋮   ⋮   ⋮   aux += v[k]
```



Quiz 2: resolva o caso c/ 3 laços aninhados usando a análise simplificada!!!

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\sum_{k=i+1}^{j} c \geq \sum_{i=1}^{n/3}\sum_{j=\frac{2n}{3}+1}^{n}\sum_{k=n/3+1}^{2n/3} c$$

Note que limitar superiormente por $O(n^3)$ é simples

$$= \frac{n}{3}\cdot\frac{n}{3}\cdot\frac{n}{3}\cdot c = \frac{c}{27}\cdot n^3 = \Omega(n^3)$$