Introdução à análise de algoritmos

→ **Algoritmo:** sequência finita de passos descritos de forma não ambígua que corretamente resolvem um problema.

→ **Analisar um algoritmo** nos permite prever comportamento e desempenho sem que seja necessário implementá-lo em um dispositivo específico.

Um problema simples

Problema *Busca*

Entrada: $\langle A, n, k \rangle$, onde $A[1..n]$ é um vetor contendo n elementos e k é um número.
Saída: Posição i do elemento de A cuja chave é igual a k, se esse existe, ou -1, caso contrário.

	1	2	3	4	5	6	7	8	9
A	16	9	-3	5	4	27	46	11	-5

K = 27 K = 2

Um algoritmo simples

```
BuscaLinear(A, n, k)
1 i = 1
2 enquanto i ≤ n e A[i].chave ≠ k faça
3     i = i + 1
4 se i ≤ n e A[i].chave == k então
5     devolve i
6 devolve −1
```

Ideia: Percorra o vetor, analisando cada posição i.
Se $A[i]$.chave é igual a k, encontrou e devolve i. Se percorreu o vetor todo e nenhum elemento assim existe, devolva -1.

→ É "fácil" acreditar que BuscaLinear resolve o problema da busca. Vamos deixar assim por ora.

Um problema relacionado

Problema *Busca em Dados Ordenados*

Entrada: $\langle A, n, k\rangle$, onde $A[1..n]$ é um vetor contendo n elementos cujas chaves estão em ordem crescente, i.e., $A[i].\texttt{chave} < A[i+1].\texttt{chave}$ para todo $1 \leq i < n$, e k é um número.

Saída: Posição i do elemento de A cuja chave é igual a k, se esse existe, ou -1, caso contrário.

	1	2	3	4	5	6	7	8	9
A	-6	-1	3	7	10	27	35	37	52

K = 27 K = 2

Como resolver esse novo problema?

→ É fácil acreditar que BuscaLinear resolve o problema.

	1	2	3	4	5	6	7	8	9
A	-6	-1	3	7	10	27	35	37	52

K = 2

→ mas será que dá para fazer algo melhor?

→ Um vetor ordenado nos dá mais informação/estrutura!

Um algoritmo melhor (?)

```
BUSCALINEAREMORDEM(A, n, k)
1 i = 1
2 enquanto i ≤ n e A[i].chave < k faça
3     i = i + 1
4 se i ≤ n e A[i].chave == k então
5     devolve i
6 devolve −1
```

Ideia: Faça como na busca linear, porém pare assim que A[i].chave > K, pois não é possível que o elemento esteja à direita de uma posição assim.

→ É fácil acreditar que BuscaLinearEmOrdem resolve o problema da busca em dados ordenados.

→ Ele resolve o problema Busca?

Outro algoritmo ainda melhor (?)

→ Se as chaves em $A[1..n]$ estão ordenadas, compare k com $A[\frac{n}{2}]$.chave

→ Só pode acontecer um dentre esses três casos:

- $k = A[\frac{n}{2}]$.chave
- $k < A[\frac{n}{2}]$.chave ⇒ se existe elem. com chave k em A, ele está em $A[1..\frac{n}{2}-1]$
- $k > A[\frac{n}{2}]$.chave ⇒ se existe elem. com chave k em A, ele está em $A[\frac{n}{2}+1..n]$

Outro algoritmo ainda melhor (?)

```
BuscaBinaria(A, n, k)
1  esq = 1
2  dir = n
3  enquanto esq < dir faça
4      meio = ⌊(esq + dir)/2⌋
5      se k > A[meio].chave então
6          esq = meio + 1
7      senão
8          dir = meio
9  se A[esq].chave == k então
10     devolve esq
11 devolve −1
```

Solaia: A cada iteração, consideramos que estamos procurando o elemento de chave k em $A[esq..dir]$. Comparamos com a posição do meio desse vetor e atualizamos esq ou dir de acordo.

→ Esse algoritmo resolve o problema da busca em dados ordenados?

→ Ele pode devolver -1 se $k \in A$?

E agora?

→ BuscaLinear, BuscaLinearEmOrdem e BuscaBinaria resolvem (?) o problema da busca em dados ordenados.

→ Qual deles usar? Qual é o melhor?

Corretude de algoritmos iterativos

→ Um algoritmo está correto quando devolve uma resposta correta para qualquer instância.
↳ Então ele está incorreto quando devolve uma resposta errada para alguma instância

→ Como convencer alguém de que um algoritmo está correto?

Prova por invariante de laço

DEFINIÇÃO: Uma invariante de laço é uma afirmação que é verdadeira no início de qualquer iteração do laço.

→ Em geral começam com "Antes da t-ésima iteração começar, vale que ..." e envolvem variáveis importantes para o laço.
↳ Notação: $P(t)$

→ São úteis quando nos permitem concluir algo importante após o término do laço.

Como provar que uma frase é uma invariante?

→ Por definição, basta provar que $P(1), P(2), \ldots, P(T+1)$ são verdadeiros, onde T é a quantidade de iterações realizadas.
↳ Qual o problema aqui?

→ Por que provar até $P(T+1)$ e não $P(T)$?
↳ $P(T)$ diz algo válido no início da última iteração, mas no decorrer dela as coisas podem mudar
↳ $P(T+1)$ diz algo válido no início de uma iteração que não existe, ou seja, no fim do laço.

Como provar que uma frase é uma invariante?

→ Por indução!
  ↳ Prove P(1)
  ↳ Considere $t \geq 1$ e suponha que P(t) vale.
  ↳ Prove P(t+1).

→ O valor T não é necessário nos passos acima, porém...
  ... uma vez que provamos que P é invariante, usaremos P(T+1) para dizer algo útil ao fim do laço.

Invariantes de laço: BuscaLinear

P(t) = "Antes da t-ésima iteração começar, $i = t$ e os elementos de $A[1..i-1]$ já foram acessados."
$A[1..t-1]$

R(t) = "Antes da t-ésima iteração começar, $i = t$ e K já foi comparada com $i-1$ chaves de A."
$t-1$

```
BUSCALINEAR(A, n, k)
1 i = 1
2 enquanto i ≤ n e A[i].chave ≠ k faça
3     i = i + 1
4 se i ≤ n e A[i].chave == k então
5     devolve i
6 devolve −1
```

Invariantes de laço: BuscaLinear

→ Para provar que a busca funciona mesmo, precisamos mostrar que "existe um elemento com chave k no vetor se e somente se o algoritmo devolve um índice que contém tal elemento"

→ contrapositiva

→ De forma equivalente, portanto "se existe um elemento com chave k no vetor, então o algoritmo devolve um índice do vetor (que contenha tal elemento) e se não existe um elemento com chave k no vetor, então o algoritmo não devolve um índice do vetor"

Corretude de BuscaLinear

Para provar o resultado, suponha que temos a seguinte invariante:
$P(t)$ = "Antes da t-ésima iteração começar, vale que $i = t$ e o vetor $A[1..i-1]$ não contém um elemento com chave k."

Vamos começar provando α = "Se não existe elem. com chave k em A, então BuscaLinear não devolve um índice válido.".
O laço pode terminar porque $i > n$. Nesse caso, executou n vezes e $P(n+1)$ nos diz que o vetor $A[1..n]$ não contém elem. com chave k.
O teste da linha 4 falha e o alg. devolve -1.
Então de fato α é verdadeira.

```
BUSCALINEAR(A, n, k)
1 i = 1
2 enquanto i ≤ n e A[i].chave ≠ k faça
3     i = i + 1
4 se i ≤ n e A[i].chave == k então
5     devolve i
6 devolve −1
```

Agora vamos provar β = "Se existe elem. com chave k em A, então BuscaLinear devolve seu índice."
O outro único motivo para o laço parar é se $i \leq n$ e $A[i].chave = k$. Mas então existe um elemento com chave k em A. O teste na linha 4 funciona e o algoritmo devolve justamente i.
Então de fato β é verdadeira.

Invariantes de laço: BuscaLinear

Vamos provar que

P(t) = "Antes da t-ésima iteração começar, vale que $i = t$ e o vetor $A[1..i-1]$ não contém um elemento com chave k."

é invariante, por indução no nº t de vezes que o teste do laço executa

```
BUSCALINEAR(A, n, k)
1 i = 1
2 enquanto i ≤ n e A[i].chave ≠ k faça
3     i = i + 1
4 se i ≤ n e A[i].chave == k então
5     devolve i
6 devolve −1
```

CASO BASE: $t = 1$. O teste executou uma única vez. Claramente temos $i = 1$. Além disso, $A[1..0]$ é vazio e certamente não contém um elemento com chave k. Logo, P(1) vale.

PASSO: $t \geq 1$. Suponha que P(t) vale e vamos provar P(t+1). Como o t-ésimo teste foi executado, pois o laço começou, sabemos que $A[t].chave \neq k$ (pois $i = t$ já que P(t) vale). Juntando com o fato de $A[1..t-1]$ não ter elementos com chave k (pois P(t) vale), sabemos que $A[1..t]$ não tem elemento com chave k.
O laço ainda incrementa i, fazendo $i = t+1$.
Acabamos de provar, portanto, P(t+1).
Logo, P é invariante.

RESUMO DO PASSO (válido em qualquer prova):
- → Estamos no início da t-ésima iteração.
- → O teste executou e deu verdadeiro (pois a iteração começou).
- → Sabemos: a info. do teste $\underline{\underline{e}}$ P(t).
- → Queremos: P(t+1).
- → Analise o que ocorre durante $\underline{\underline{esta}}$ iteração.
- → Conclua P(t+1).

Invariante de laço: BuscaBinaria

```
BUSCABINARIA(A, n, k)
1  esq = 1
2  dir = n
3  enquanto esq < dir faça
4      meio = ⌊(esq + dir)/2⌋
5      se k > A[meio].chave então
6          esq = meio + 1
7      senão
8          dir = meio
9  se A[esq].chave == k então
10     devolve esq
11 devolve −1
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19=n
-4	3	6	7	10	12	16	17	25	29	33	35	39	43	49	57	79	93	99

k = 25

Invariante de laço: BuscaBinaria

Vamos começar provando que P(t) = "Antes da t-ésima iteração começar, esq ≤ dir e A[1..esq-1] e A[dir+1..n] não contém um elemento com chave K." é uma invariante, por indução no nº t de vezes que o teste é executado.

CASO BASE: t = 1. O teste executou uma vez e fizemos esq = 1 e dir = n. Claramente, não há elem. com chave K em A[1..0] e em A[n+1..n]. Então P(1) vale.


```
BUSCABINARIA(A, n, k)
1  esq = 1
2  dir = n
3  enquanto esq < dir faça
4      meio = ⌊(esq + dir)/2⌋
5      se k > A[meio].chave então
6          esq = meio + 1
7      senão
8          dir = meio
9  se A[esq].chave == k então
10     devolve esq
11 devolve −1
```

Invariante de laço: BuscaBinaria

↳ Poderia usar esq_t, dir_t ao invés de x e y ☺

$P(t)$ = "Antes da t-ésima iteração começar, $esq \leq dir$ e $A[1..esq-1]$ e $A[dir+1..n]$ não contém um elemento com chave k."

```
BUSCABINARIA(A, n, k)
1  esq = 1
2  dir = n
3  enquanto esq < dir faça
4  |  meio = ⌊(esq + dir)/2⌋
5  |  se k > A[meio].chave entã
6  |  |  esq = meio + 1
7  |  senão
8  |  |  dir = meio
9  se A[esq].chave == k então
10 |  devolve esq
11 devolve −1
```

PASSO: Seja $t \geq 1$ e suponha que $P(t)$ vale.
Sejam x e y os valores de esq e dir no início dessa t-ésima iteração. Logo, $x \leq y$.
Como a iteração começou, o teste nos dá $x < y$.
A iteração começa fazendo $meio = \lfloor (x+y)/2 \rfloor$.
Como $\lfloor (x+y)/2 \rfloor > (x+y)/2 - 1 > \frac{2x}{2} - 1 = x - 1$ e $\lfloor (x+y)/2 \rfloor \leq (x+y)/2 < \frac{2y}{2} = y$, então vale que $x - 1 < meio < y$, ou, $x \leq meio < y$.

Há duas possibilidades: $A[meio].chave < k$ ou $A[meio].chave \geq k$.
Se $A[meio].chave < k$, o algoritmo atualiza esq para $meio+1$ e mantém $dir = y$. Como $meio < y$, vale que $meio+1 \leq y$, o que mantém $esq \leq dir$.
Além disso, como A está com as chaves ordenadas, $A[meio].chave < k$ indica que não há elem. com chave k em $A[1..meio] = A[1..esq-1]$. Por $P(t)$ já sabíamos que não há elem. com chave k em $A[y+1..n] = A[dir+1..n]$, então provamos $P(t+1)$ neste caso.

```
BUSCABINARIA(A, n, k)
1  esq = 1
2  dir = n
3  enquanto esq < dir faça
4  |  meio = ⌊(esq + dir)/2⌋
5  |  se k > A[meio].chave então
6  |  |  esq = meio + 1
7  |  senão
8  |  |  dir = meio
9  se A[esq].chave == k então
10 |  devolve esq
11 devolve −1
```

Agora, se $A[meio].chave \geq k$, então o algoritmo atualiza dir para meio e mantém $esq = x$. Como $x \leq meio$, mantemos $esq \leq dir$.
Como A está com as chaves ordenadas, sabemos agora que não há elem. com chave k em $A[meio+1..n] = A[dir+1..n]$. Por $P(t)$ já sabíamos isso sobre $A[1..x-1] = A[1..esq-1]$. Logo, $P(t+1)$ é verdade nesse caso também.

Então P é invariante.

```
BuscaBinaria(A, n, k)
1  esq = 1
2  dir = n
3  enquanto esq < dir faça
4      meio = ⌊(esq + dir)/2⌋
5      se k > A[meio].chave então
6          esq = meio + 1
7      senão
8          dir = meio
9  se A[esq].chave == k então
10     devolve esq
11 devolve −1
```

Agora sejam r e t os valores de esq e dir quando o laço termina, após T iterações.
$P(T+1)$ nos diz que $r \leq t$ e como o laço terminou, $r \geq t$. Então só pode ser o caso de $r = t$. $P(T+1)$ também nos diz (já usando $r=t$) que $A[1..r-1]$ e $A[r+1..n]$ não contêm elem. com chave K.
Na linha 9, verificamos se $K = A[r].chave$.
Se for, o elemento procurado existe em A e o algoritmo devolve sua posição.
Se não for, o elemento não existe em A e o alg. devolve -1.
Logo, o algoritmo funciona corretamente.

Mais exemplos

```
CONVERTE BASE 2 (n)
  seja B[1..n] um vetor
  i = 1
  m = n
  enquanto m > 0
      B[i] = resto divisão m por 2
      i = i + 1
      m = ⌊m/2⌋
  devolve B[1..i-1]
```

P(t) = "Antes da t-ésima iteração começar, $i = t$ e o vetor $B[1..i-1]$ representa um inteiro z tal que $z = n - m \cdot 2^{i-1}$"

Seja r a qtd. de execuções do laço. $P(r+1)$ nos diz que $i = r+1$ e $B[1..r]$ representa um inteiro z tal que $z = n - 0 \cdot 2^r = n$.

```
MAXIMO (A, n)
  maxc = A[1]
  para i = 2 até n, inc. 1
      se A[i] > maxc
          maxc = A[i]
  devolve maxc
```

P(t) = "Antes da t-ésima iteração começar, $i = t+1$ e maxc tem o maior nº do vetor $A[1..i-1]$"

O laço executa $n-1$ vezes. $P(n)$ nos diz que $i = n+1$ e maxc tem o maior nº do vetor $A[1..n]$.

```
MAXIMO (A, n)
  maxc = A[n]
  para i = n-1 até 1, dec. 1
      se A[i] > maxc
          maxc = A[i]
  devolve maxc
```

P(t) = "Antes da t-ésima iteração começar, $i = n-t$ e maxc tem o maior nº do vetor $A[i+1..n]$"

$P(n) \Rightarrow i = n-t = 0$ e $A[i+1..n] = A[1..n]$.