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.

escrever e não apagar

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

escrever ⇒

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 encontrar = O(n)

pior caso = O(n)

Descobrindo invariantes: que propriedade o alg. encontra/mantém a cada iteração?

Corretude e prova por indução

- O invariante principal do algoritmo é que,
  - no início ~~de cada~~ 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 ~~depois das i - 1 primeiras iterações~~.
- Passo: 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 algoritmo?

Busca binária

Algoritmo iterativo de busca binária em vetor ordenado.

	Contando operações
buscaBin(v[ ], n, x):	
e = -1	1
d = n	1
enquanto (e < d - 1):	2 * p
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
	*5 + 8 * p*

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 13 + 8 lg n

Corretude e prova por indução

- O invariante principal 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 depois das i - 1 primeiras iterações.
- Passo: No início da i-ésima iteração sabemos, pela H.I.,
    - que v[0 .. e] < x <= v[d .. n - 1]
- Como no laço só atualizamos o extremo (e ou d)
    - de acordo com o resultado da comparação v[m] < x,
        - o invariante continua valendo no início da iteração seguinte.
    - Ou seja, sempre descartamos o subvetor correto.

Esse invariante garante que quando o algoritmo sai do laço temos e = d - 1.

- Assim, v[0 .. e] < x <= v[e + 1 .. n - 1].
- Portanto, é a primeira posição em que x pode estar
    - a posição que x deve ocupar para que o vetor permaneça ordenado.

Busca por elemento repetido

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: como resolver esse problema de modo mais eficiente?

Corretude e prova por indução

- O laço interno do algoritmo é uma busca linear, que analisamos antes.
- O invariante principal do laço externo do algoritmo é que,
    - no início de cada iteração do laço,
        - nenhum elemento do vetor v[0 .. i - 1] se repete em v[0 .. n].
- 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 depois das i - 1 primeiras iterações.
- Passo: No início da i-ésima iteração sabemos, pela H.I.,
    - que v[0 .. i - 1] não tem elementos repetidos em v[0 .. n].
- Se a busca linear realizada na i-ésima iteração encontrar o elemento v[i]
    - então o laço é interrompido e i não é mais incrementado.
- Caso contrário, sabemos que v[i] não se repete em v[i+1 .. n]
    - Pela H.I., sabemos que ele também não se repete em v[0 .. i-1].
- 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 condição (i < n),
    - o invariante garante que v[0 .. n - 1] não contém elemento repetido.
- Caso contrário, a violação da segunda condição (j == n)
    - indica que na iteração anterior (i' = i - 1)
        - o laço interno terminou porque encontrou v[j] == v[i'] == v[i - 1]
- Como i' e j sempre são diferentes, encontramos elementos repetidos.