Corretude de Algoritmos

Corretude de Algoritmos

Lembre a definição de fatorial: $n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1$

Corretude de Algoritmos

Lembre a definição de fatorial: $n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1$

Exemplo (Calculando o fatorial)

Considere o código abaixo.

Fatorial

Entrada: x (inteiro positivo)
Função *$Fat(x)$*
    If $x = 1$
        Return 1
    End if
    Return $x \cdot Fat(x-1)$
Fim Função

Mostre que para todo $n \geq 1$, $Fat(n)$ retorna $n!$

```
Função Fat(x)
    If x = 1
        Return 1
    End if
    Return x · Fat(x − 1)
Fim Função
```

Caso Base: $n = 1$**.** Por causa do "**If**", $Fat(1) = 1 = 1!$

```
Função Fat(x)
    If x = 1
        Return 1
    End if
    Return x · Fat(x − 1)
Fim Função
```

Caso Base: $n = 1$**.** Por causa do "**If**", $Fat(1) = 1 = 1!$

Passo Indutivo. Suponha que a propriedade valha para n, ou seja $Fat(n) = n!$. Queremos provar que vale para $n + 1$

```
Função Fat(x)
    If x = 1
        Return 1
    End if
    Return x · Fat(x − 1)
Fim Função
```

Caso Base: $n = 1$**.** Por causa do "**If**", $Fat(1) = 1 = 1!$

Passo Indutivo. Suponha que a propriedade valha para n, ou seja $Fat(n) = n!$. Queremos provar que vale para $n + 1$

Pelo código, $Fat(n + 1) = (n + 1) \cdot Fat(n)$

```
Função Fat(x)
    If x = 1
        Return 1
    End if
    Return x · Fat(x − 1)
Fim Função
```

Caso Base: $n = 1$**.** Por causa do "**If**", $Fat(1) = 1 = 1!$

Passo Indutivo. Suponha que a propriedade valha para n, ou seja $Fat(n) = n!$. Queremos provar que vale para $n + 1$

Pelo código, $Fat(n + 1) = (n + 1) \cdot Fat(n)$

Pela hipótese indutiva $Fat(n) = n!$, então

```
Função Fat(x)
    If x = 1
        Return 1
    End if
    Return x · Fat(x − 1)
Fim Função
```

Caso Base: $n = 1$**.** Por causa do "**If**", $Fat(1) = 1 = 1!$

Passo Indutivo. Suponha que a propriedade valha para n, ou seja $Fat(n) = n!$. Queremos provar que vale para $n + 1$

Pelo código, $Fat(n + 1) = (n + 1) \cdot Fat(n)$

Pela hipótese indutiva $Fat(n) = n!$, então

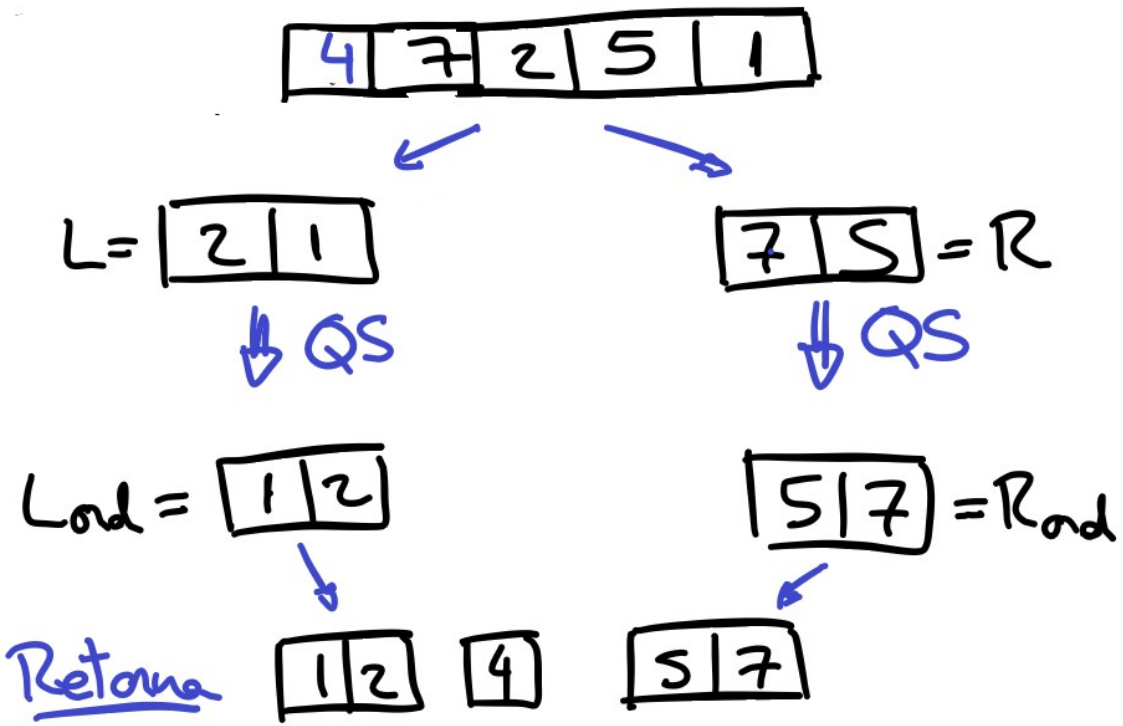
$$\begin{aligned} Fat(n+1) &= (n+1) \cdot Fat(n) \\ &= (n+1) \cdot n! \\ &= (n+1) \cdot n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1 \qquad \text{(Pela definição de } n!) \\ &= (n+1)! \end{aligned}$$

Quicksort

Quicksort é um dos algoritmos mais populares para ordenar lista de números

Quicksort

Quicksort é um dos algoritmos mais populares para ordenar lista de números



Quicksort

Quicksort(A)

If $A.len = 0$ ou $A.len = 1$
    Return A
Else
    $L =$ vetor com números de A menores que $A[1]$ # *pivot*
    $R =$ vetor com números de A maiores que $A[1]$
    $P =$ vetor com números de A iguais a $A[1]$
    $L_{ord} \leftarrow$ Quicksort(L)
    $R_{ord} \leftarrow$ Quicksort(R)
    Return concatenação de L_{ord}, P e R_{ord}
End if

Quicksort

Proposição

Quicksort(A) retorna o vetor A ordenado

Quicksort

Proposição

Quicksort(A) retorna o vetor A ordenado

Prova: Por indução forte no tamanho n do vetor A

Quicksort

Proposição

Quicksort(A) retorna o vetor A ordenado

Prova: Por indução forte no tamanho n do vetor A

Caso base: tamanho $n = 0$ ou $n = 1$. Quicksort(A) retorna A, OK

Quicksort

Proposição

Quicksort(A) retorna o vetor A ordenado

Prova: Por indução forte no tamanho n do vetor A

Caso base: tamanho $n = 0$ ou $n = 1$. Quicksort(A) retorna A, OK

Passo indutivo. Considere $n \geq 1$ e suponha que Quicksort funcione pra todo vetor de tamanho $\leq n$

Queremos provar que funciona para vetores de tamanho $n + 1$

Quicksort

Considere um vetor A de tamanho $n + 1$

Quicksort

Considere um vetor A de tamanho $n + 1$

Note que os vetores L e R construídos têm tamanho estritamente menor que A (ou seja $\leq n$)

Quicksort

Considere um vetor A de tamanho $n + 1$

Note que os vetores L e R construídos têm tamanho estritamente menor que A (ou seja $\leq n$)

Então podemos aplicar a **hipótese indutiva** e obter que L_{ord} e R_{ord} estao ordenados

Quicksort

Considere um vetor A de tamanho $n+1$

Note que os vetores L e R construídos têm tamanho estritamente menor que A (ou seja $\leq n$)

Então podemos aplicar a **hipótese indutiva** e obter que L_{ord} e R_{ord} estao ordenados

Portanto, Quicksort(A) retorna

$$\underbrace{L_{ord}}_{\text{menores que } A[1] \text{ ordenados}} + \underbrace{P}_{\text{iguais a } A[1]} + \underbrace{R_{ord}}_{\text{maiores que } A[1] \text{ ordenados}}$$

Quicksort

Considere um vetor A de tamanho $n+1$

Note que os vetores L e R construídos têm tamanho estritamente menor que A (ou seja $\leq n$)

Então podemos aplicar a **hipótese indutiva** e obter que L_{ord} e R_{ord} estao ordenados

Portanto, Quicksort(A) retorna

$$\underbrace{L_{ord}}_{\text{menores que } A[1] \text{ ordenados}} + \underbrace{P}_{\text{iguais a } A[1]} + \underbrace{R_{ord}}_{\text{maiores que } A[1] \text{ ordenados}}$$

$\Rightarrow$ retorna vetor ordenado

Quicksort

Considere um vetor A de tamanho $n+1$

Note que os vetores L e R construídos têm tamanho estritamente menor que A (ou seja $\leq n$)

Então podemos aplicar a **hipótese indutiva** e obter que L_{ord} e R_{ord} estao ordenados

Portanto, Quicksort(A) retorna

$$\underbrace{L_{ord}}_{\text{menores que } A[1] \text{ ordenados}} + \underbrace{P}_{\text{iguais a } A[1]} + \underbrace{R_{ord}}_{\text{maiores que } A[1] \text{ ordenados}}$$

$\Rightarrow$ retorna vetor ordenado

Conversor binário

Converter número em sua representação binária

Conversor binário

Converter número em sua representação binária

Saída: vetor $v = v_1, v_2, \ldots v_k$ onde v_1 é o digito binário mais significativo, v_2 o segundo mais significativo, etc.

Conversor binário

Converter número em sua representação binária

Saída: vetor $v = v_1, v_2, \ldots v_k$ onde v_1 é o digito binário mais significativo, v_2 o segundo mais significativo, etc.

Ex:

	Saída
$n = 6$	$(v_1, v_2, v_3) = (1, 1, 0)$
$n = 8$	$(v_1, v_2, v_3, v_4) = (1, 0, 0, 0)$
$n = 0$	vetor vazio

Conversor binário

Converter número em sua representação binária

Saída: vetor $v = v_1, v_2, \ldots v_k$ onde v_1 é o digito binário mais significativo, v_2 o segundo mais significativo, etc.

Ex:

	Saída
$n = 6$	$(v_1, v_2, v_3) = (1, 1, 0)$
$n = 8$	$(v_1, v_2, v_3, v_4) = (1, 0, 0, 0)$
$n = 0$	vetor vazio

Q: Qual é a relação entre n e o vetor v?

Conversor binário

Converter número em sua representação binária

Saída: vetor $v = v_1, v_2, \ldots v_k$ onde v_1 é o digito binário mais significativo, v_2 o segundo mais significativo, etc.

Ex:

	Saída
$n = 6$	$(v_1, v_2, v_3) = (1, 1, 0)$
$n = 8$	$(v_1, v_2, v_3, v_4) = (1, 0, 0, 0)$
$n = 0$	vetor vazio

Q: Qual é a relação entre n e o vetor v?

$$n = 2^{k-1}v_1 + 2^{k-2}v_2 + \ldots + 2^0 v_k$$

Conversor binário

Converte(n)

If $n = 0$

    Return vetor vazio

Else if n **é par**

    Vetor $u \leftarrow$ Converte$(\frac{n}{2})$

    Return vetor $(u_1, u_2, \ldots, u_{u.len}, 0)$ [ou seja, u concatenado com 0 no final]

Else if n **é impar**

    Vetor $u \leftarrow$ Converte$(\frac{n-1}{2})$

    Return vetor $(u_1, u_2, \ldots, u_{u.len}, 1)$ [ou seja, u concatenado com 1 no final]

End if

Conversor binário

Converte(n)

If $n = 0$
    Return vetor vazio
Else if n **é par**
    Vetor $u \leftarrow$ Converte$(\frac{n}{2})$
    Return vetor $(u_1, u_2, \ldots, u_{u.len}, 0)$ [ou seja, u concatenado com 0 no final]
Else if n **é impar**
    Vetor $u \leftarrow$ Converte$(\frac{n-1}{2})$
    Return vetor $(u_1, u_2, \ldots, u_{u.len}, 1)$ [ou seja, u concatenado com 1 no final]
End if

Proposição

Para todo n, o vetor v retornado por Converte(n) é a representação binária de n: (k é o tamanho de v)

$$n = 2^{k-1}v_1 + 2^{k-2}v_2 + \ldots + 2^0 v_k$$

Conversor binário

Prova: Por indução forte em n **Caso base:** $n = 0$**.** Converte(0) retorna lista vazia, correto por definição

Conversor binário

Prova: Por indução forte em n **Caso base:** $n = 0$**.** Converte(0) retorna lista vazia, correto por definição

Passo indutivo. Considere $n \geq 1$. Suponha que o algoritmo funcione pra todo número $\leq n$

Conversor binário

Prova: Por indução forte em n **Caso base:** $n = 0$**.** Converte(0) retorna lista vazia, correto por definição

Passo indutivo. Considere $n \geq 1$. Suponha que o algoritmo funcione pra todo número $\leq n$

Precisamos provar que funciona pro número $n + 1$

Conversor binário

Caso 1: $n+1$ **é par.** Note que $\frac{n+1}{2}$ é estritamente menor que $n+1$, portanto podemos usar a **hipótese indutiva**

Conversor binário

Caso 1: $n+1$ **é par.** Note que $\frac{n+1}{2}$ é estritamente menor que $n+1$, portanto podemos usar a **hipótese indutiva**

Lembre $u = \mathit{Converte}(\frac{n+1}{2})$. Seja k o tamanho de u

Conversor binário

Caso 1: $n+1$ **é par.** Note que $\frac{n+1}{2}$ é estritamente menor que $n+1$, portanto podemos usar a **hipótese indutiva**

Lembre $u = \mathit{Converte}(\frac{n+1}{2})$. Seja k o tamanho de u

Converte$(n+1)$ retorna vetor $(u_1, \ldots, u_k, 0)$ com tamanho $k+1$ que tem valor

$$
\begin{aligned}
& 2^k u_1 + 2^{k-1} u_2 + \ldots + 2^1 u_k + \underbrace{2^0 \cdot 0}_{0} \\
= \; & 2 \cdot \left(2^{k-1} u_1 + 2^{k-2} u_2 + \ldots + 2^0 u_k \right) \\
= \; & 2 \cdot \frac{n+1}{2} && \text{(pela hipótese indutiva)} \\
= \; & n+1
\end{aligned}
$$

Conversor binário

Caso 2: $n+1$ **é impar.** Note que $\frac{(n+1)-1}{2}$ é estritamente menor que $n+1$, portanto podemos usar a **hipótese indutiva**

Conversor binário

Caso 2: $n+1$ **é impar.** Note que $\frac{(n+1)-1}{2}$ é estritamente menor que $n+1$, portanto podemos usar a **hipótese indutiva**

Lembre $u = Converte(\frac{(n+1)-1}{2})$. Seja k o tamanho de u

Conversor binário

Caso 2: $n+1$ **é impar.** Note que $\frac{(n+1)-1}{2}$ é estritamente menor que $n+1$, portanto podemos usar a **hipótese indutiva**

Lembre $u = \mathit{Converte}(\frac{(n+1)-1}{2})$. Seja k o tamanho de u

Converte$(n+1)$ retorna vetor $(u_1, \ldots, u_k, 1)$ com tamanho $k+1$ que tem valor

$$
\begin{aligned}
& 2^k u_1 + 2^{k-1} u_2 + \ldots + 2^1 u_k + 2^0 \cdot 1 \\
= \; & 2 \cdot \left(2^{k-1} u_1 + 2^{k-2} u_2 + \ldots + 2^0 u_k \right) + 1 \\
= \; & 2 \cdot \frac{(n+1)-1}{2} + 1 && \text{(pela hipótese indutiva)} \\
= \; & n+1
\end{aligned}
$$

Exercício

Exercício 2: Prove por indução em $n \geq 0$ que para toda entrada (a, n) o algoritmo abaixo retorna a^n

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then return** 1
else return $a \cdot power(a, n - 1)$

Exercício

Exercício 3: (Avaliação de polinômio)
a) Prove por indução em $n \geq 0$ que para toda entrada (A, x, n) o algoritmo abaixo retorna $A[0] + A[1] \cdot x + A[2] \cdot x^2 + \ldots + A[n] \cdot x^n$

Algo(Vetor de números A, número real x, inteiro n)

If $n = 0$
    Return $A[0]$
Else
    $B = (A[1], A[2], \ldots, A[n])$ # *remove* $A[0]$
    $tmp =$**Algo**$(B,\ x,\ n-1)$
    Return $(tmp \cdot x) + A[0]$
End if

b) Prove por indução que para toda entrada (A, x, n) o número de operações aritméticas ($+$ e $\cdot$) realizadas é menor ou igual a $2n$