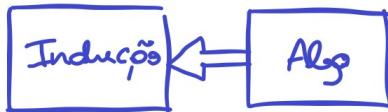


Inducao para desenho de algoritmos

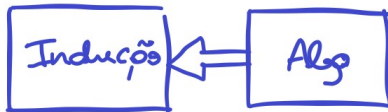
Inducao para desenho de algoritmos

Mostramos como provar corretude de algoritmos usando indução

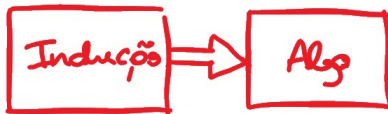


Inducao para desenho de algoritmos

Mostramos como provar corretude de algoritmos usando indução



Agora veremos como usar indução para **desenhar** algoritmos



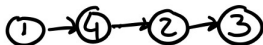
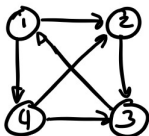
Proposição

Ao final de qualquer campeonato em que todos os $n \geq 2$ participantes jogaram entre si, é possível, independentemente dos resultados, ordenar os participantes como p_1, p_2, \dots, p_n de forma que, para todo $i \in \{1, 2, \dots, n - 1\}$, p_i vence p_{i+1}

Indução para desenho de algoritmos

Proposição

Ao final de qualquer campeonato em que todos os $n \geq 2$ participantes jogaram entre si, é possível, independentemente dos resultados, ordenar os participantes como p_1, p_2, \dots, p_n de forma que, para todo $i \in \{1, 2, \dots, n - 1\}$, p_i vence p_{i+1}



Proposição

Ao final de qualquer campeonato em que todos os $n \geq 2$ participantes jogaram entre si, é possível, independentemente dos resultados, ordenar os participantes como p_1, p_2, \dots, p_n de forma que, para todo $i \in \{1, 2, \dots, n - 1\}$, p_i vence p_{i+1}

Vamos primeiro provar isso por indução fraca no número de jogadores

Proposição

Ao final de qualquer campeonato em que todos os $n \geq 2$ participantes jogaram entre si, é possível, independentemente dos resultados, ordenar os participantes como p_1, p_2, \dots, p_n de forma que, para todo $i \in \{1, 2, \dots, n - 1\}$, p_i vence p_{i+1}

Vamos primeiro provar isso por indução fraca no número de jogadores

Prova: Caso base: $n = 2$. Caso Jogador 1 ganhou do Jogador 2 usamos a ordem

Jogador 1 \rightarrow Jogador 2;

caso contrário usamos

Jogador 2 \rightarrow Jogador 1

Inducao para desenho de algoritmos

Passo indutivo. Seja $n \geq 2$, suponha que propriedade valha para n .
Precisamos provar que vale para $n + 1$

Inducao para desenho de algoritmos

Passo indutivo. Seja $n \geq 2$, suponha que propriedade valha para n .
Precisamos provar que vale para $n + 1$

Considere $n + 1$ jogadores e o resultado dos seus jogos

Indução para desenho de algoritmos

Passo indutivo. Seja $n \geq 2$, suponha que propriedade valha para n .
Precisamos provar que vale para $n + 1$

Considere $n + 1$ jogadores e o resultado dos seus jogos

Pegue um jogador p^* e retire ele do conjunto.

Indução para desenho de algoritmos

Passo indutivo. Seja $n \geq 2$, suponha que propriedade valha para n . Precisamos provar que vale para $n + 1$

Considere $n + 1$ jogadores e o resultado dos seus jogos

Pegue um jogador p^* e retire ele do conjunto. Podemos usar a **hipótese indutiva** nos n jogadores (e seus jogos) restantes para obter uma ordem

$$p_1 \rightarrow p_2 \rightarrow \dots \rightarrow p_n$$

Indução para desenho de algoritmos

Passo indutivo. Seja $n \geq 2$, suponha que propriedade valha para n .
Precisamos provar que vale para $n + 1$

Considere $n + 1$ jogadores e o resultado dos seus jogos

Pegue um jogador p^* e retire ele do conjunto. Podemos usar a **hipótese indutiva** nos n jogadores (e seus jogos) restantes para obter uma ordem

$$p_1 \rightarrow p_2 \rightarrow \dots \rightarrow p_n$$

Aonde inserir p^* nessa ordem?

Indução para desenho de algoritmos

Passo indutivo. Seja $n \geq 2$, suponha que propriedade valha para n . Precisamos provar que vale para $n + 1$

Considere $n + 1$ jogadores e o resultado dos seus jogos

Pegue um jogador p^* e retire ele do conjunto. Podemos usar a **hipótese indutiva** nos n jogadores (e seus jogos) restantes para obter uma ordem

$$p_1 \rightarrow p_2 \rightarrow \dots \rightarrow p_n$$

Caso 1: p^* perdeu todos os jogos

Indução para desenho de algoritmos

Passo indutivo. Seja $n \geq 2$, suponha que propriedade valha para n . Precisamos provar que vale para $n + 1$

Considere $n + 1$ jogadores e o resultado dos seus jogos

Pegue um jogador p^* e retire ele do conjunto. Podemos usar a **hipótese indutiva** nos n jogadores (e seus jogos) restantes para obter uma ordem

$$p_1 \rightarrow p_2 \rightarrow \dots \rightarrow p_n$$

Caso 1: p^* perdeu todos os jogos, coloque no fim:

$$p_1 \rightarrow \dots \rightarrow p_n \rightarrow p^*$$

Indução para desenho de algoritmos

Passo indutivo. Seja $n \geq 2$, suponha que propriedade valha para n . Precisamos provar que vale para $n + 1$

Considere $n + 1$ jogadores e o resultado dos seus jogos

Pegue um jogador p^* e retire ele do conjunto. Podemos usar a **hipótese indutiva** nos n jogadores (e seus jogos) restantes para obter uma ordem

$$p_1 \rightarrow p_2 \rightarrow \dots \rightarrow p_n$$

Caso 1: p^* perdeu todos os jogos, coloque no fim:

$$p_1 \rightarrow \dots \rightarrow p_n \rightarrow p^*$$

Caso 2: Caso contrário, seja p_i o primeiro jogador tal que p^* ganhou de p_i

Indução para desenho de algoritmos

Passo indutivo. Seja $n \geq 2$, suponha que propriedade valha para n . Precisamos provar que vale para $n + 1$

Considere $n + 1$ jogadores e o resultado dos seus jogos

Pegue um jogador p^* e retire ele do conjunto. Podemos usar a **hipótese indutiva** nos n jogadores (e seus jogos) restantes para obter uma ordem

$$p_1 \rightarrow p_2 \rightarrow \dots \rightarrow p_n$$

Caso 1: p^* perdeu todos os jogos, coloque no fim:

$$p_1 \rightarrow \dots \rightarrow p_n \rightarrow p^*$$

Caso 2: Caso contrário, seja p_i o primeiro jogador tal que p^* ganhou de p_i

Coloque p^* logo antes de p_i :

$$p_1 \rightarrow \dots \rightarrow p_{i-1} \rightarrow p^* \rightarrow p_i \rightarrow \dots \rightarrow p_n$$

Note que p^* está numa posição válida: o anterior ganhou dele e ele ganhou do próximo

Indução para desenho de algoritmos

Note que p^* está numa posição válida: o anterior ganhou dele e ele ganhou do próximo

Por definição p^* ganhou de p_i

Note que p^* está numa posição válida: o anterior ganhou dele e ele ganhou do próximo

Por definição p^* ganhou de p_i

E se $i > 1$, então p^* perdeu de p_{i-1} : caso contrário i não seria o menor índice tal que p^* ganhou de i

Indução para desenho de algoritmos

Note que p^* está numa posição válida: o anterior ganhou dele e ele ganhou do próximo

Por definição p^* ganhou de p_i

E se $i > 1$, então p^* perdeu de p_{i-1} : caso contrário i não seria o menor índice tal que p^* ganhou de i

(Note que se $i = 1$ então p^* é o primeiro na ordem, e não precisamos mostrar que perdeu do anterior)

Q: Baseado nessa prova, qual seria um **algoritmo** para **criar** essa ordem, dado n jogadores e seus jogos?

Q: Baseado nessa prova, qual seria um **algoritmo** para **criar** essa ordem, dado n jogadores e seus jogos?

(Imaginamos que cada jogador é uma estrutura que guarda de quem ele ganhou e quem ganhou dele)

```
Função Torneio( $J_1, J_2, \dots, J_n$ )    # conjunto de  $n$  jogadores
If  $n = 2$ 
    If  $J_1$  ganhou de  $J_2$ 
        Return sequencia  $J_1, J_2$ 
    Else
        Return sequencia  $J_2, J_1$ 
Else
    seq  $\leftarrow$  Torneio( $J_1, J_2, \dots, J_{n-1}$ )    # retirou jogador  $J_n$ 
    If  $J_n$  não ganhou de nenhum jogador
        Insira  $J_n$  no fim da sequencia seq
    Else
         $p \leftarrow$  primeiro jogador em seq tal que  $J_n$  ganhou de  $p_i$ 
        Insira  $J_n$  logo antes de  $p$  na sequencia seq
    Return seq
```

Exercício: a) Considere uma festa com n pessoas onde cada uma conheça no máximo 2 outras

Prove que é possível separar as pessoas em 3 grupos de modo que quaisquer duas pessoas do mesmo grupo não se conheçam

(I.e., podemos pintar as pessoas com 3 cores tal que se a conhece b , então tem cores diferentes)

b) Transforme sua prova em um **algoritmo** recursivo para pintar as pessoas da forma desejada.

Pode assumir que a entrada são objetos $P[1], P[2], \dots, P[n]$, com propriedades:

- $P[i].amigo1$: primeiro conhecido da pessoa $P[i]$ (null se não existe)
- $P[i].amigo2$: segundo conhecido da pessoa $P[i]$ (null se não existe)
- $P[i].cor$: a cor que voce atribuirá a à pessoa $P[i]$

Corretude de algoritmos iterativos

Podemos também usar indução para provar corretude de algoritmos **iterativos** (não-recursivos)

Podemos também usar indução para provar corretude de algoritmos **iterativos** (não-recursivos)

hipótese indutiva \equiv invariante do **For**

Podemos também usar indução para provar corretude de algoritmos **iterativos** (não-recursivos)

hipótese indutiva \equiv invariante do **For**

Vamos só fazer alguns exemplos simples

(o livro-texto do Rosen vê isso em muito mais profundidade)

Algoritmos iterativos

Função $\text{foo}(n)$

$x \leftarrow 0$

For $i = 1$ **to** n

$x \leftarrow x + 3$ (*)

End for

Return x

Algoritmos iterativos

Função $\text{foo}(n)$

$x \leftarrow 0$

For $i = 1$ **to** n

$x \leftarrow x + 3$ (*)

End for

Return x

Proposição (1)

Para todo inteiro $n \geq 1$, $\text{foo}(n)$ retorna o número $3n$.

Função $\text{foo}(n)$

$x \leftarrow 0$

For $i = 1$ **to** n

$x \leftarrow x + 3$ (*)

End for

Return x

Proposição (1)

Para todo inteiro $n \geq 1$, $\text{foo}(n)$ retorna o número $3n$.

Proposição (Invariante do **For**)

*Para todo $i \geq 1$, ao fim da i -ésima iteração do **For** (após executar (*)), $x = 3i$*

Algoritmos iterativos

Função $foo(n)$

$x \leftarrow 0$

For $i = 1$ **to** n

$x \leftarrow x + 3$ (*)

End for

Return x

Proposição (1)

Para todo inteiro $n \geq 1$, $foo(n)$ retorna o número $3n$.

Proposição (Invariante do **For**)

*Para todo $i \geq 1$, ao fim da i -ésima iteração do **For** (após executar (*)), $x = 3i$*

Proposição 2 implica na Proposição 1, pois $foo(n)$ retorna valor de x logo após a n -ésima iteração do **For**

Proposição (Invariante do **For**)

*Para todo $i \geq 1$, ao fim da i -ésima iteração do **For** (após executar $(*)$), $x = 3i$ (caso essa iteração exista)*

Prova: Por indução fraca em i , número da iteração

Proposição (Invariante do **For**)

*Para todo $i \geq 1$, ao fim da i -ésima iteração do **For** (após executar $(*)$), $x = 3i$ (caso essa iteração exista)*

Prova: Por indução fraca em i , número da iteração

Caso base: $i = 1$. Como x inicia como 0, verificamos que ao fim da primeira iteração do **For** $x = 3$

Proposição (Invariante do **For**)

Para todo $i \geq 1$, ao fim da i -ésima iteração do **For** (após executar $(*)$), $x = 3i$ (caso essa iteração exista)

Prova: Por indução fraca em i , número da iteração

Caso base: $i = 1$. Como x inicia como 0, verificamos que ao fim da primeira iteração do **For** $x = 3$

Passo indutivo. Seja $i \geq 2$, suponha verdade para a $i - 1$ -ésima iteração. Precisamos provar para a iteração i

Proposição (Invariante do **For**)

Para todo $i \geq 1$, ao fim da i -ésima iteração do **For** (após executar $(*)$), $x = 3i$ (caso essa iteração exista)

Prova: Por indução fraca em i , número da iteração

Caso base: $i = 1$. Como x inicia como 0, verificamos que ao fim da primeira iteração do **For** $x = 3$

Passo indutivo. Seja $i \geq 2$, suponha verdade para a $i - 1$ -ésima iteração. Precisamos provar para a iteração i

Devido ao algoritmo

$$(x \text{ ao fim da iteração } i) = (x \text{ ao fim da iteração } i - 1) + 3$$

Proposição (Invariante do **For**)

Para todo $i \geq 1$, ao fim da i -ésima iteração do **For** (após executar $(*)$), $x = 3i$ (caso essa iteração exista)

Prova: Por indução fraca em i , número da iteração

Caso base: $i = 1$. Como x inicia como 0, verificamos que ao fim da primeira iteração do **For** $x = 3$

Passo indutivo. Seja $i \geq 2$, suponha verdade para a $i - 1$ -ésima iteração. Precisamos provar para a iteração i

Devido ao algoritmo

$$\begin{aligned}(x \text{ ao fim da iteração } i) &= (x \text{ ao fim da iteração } i - 1) + 3 \\ &\stackrel{\text{hip ind}}{=} 3(i - 1) + 3 = 3i\end{aligned}$$

Isso conclui o passo indutivo. Fim de prova

Uma coisa interessante dessa análise é que precisamos criar uma **proposição auxiliar** para o **For**

Definição

A sequência de Fibonacci F_0, F_1, \dots , é definida assim:

- $F_0 = 0, F_1 = 1$
- $F_i = F_{i-1} + F_{i-2}$ para $i \geq 2$

Considere o seguinte algoritmo para calcular F_n

Função Fib(n)

$F \leftarrow$ vetor com posição $F[0]$ até $F[n]$

$F[0] \leftarrow 0, F[1] \leftarrow 1$

For $i = 2$ to n

$F[i] = F[i - 1] + F[i - 2]$

End For

Return $F[n]$

Função $\text{Fib}(n)$ $F \leftarrow$ vetor com posição $F[0]$ até $F[n]$ $F[0] \leftarrow 0, \quad F[1] \leftarrow 1$ **For** $i = 2$ to n $F[i] = F[i - 1] + F[i - 2]$ **End For**Return $F[n]$

Proposição (1)

Para todo $n \geq 0$, $\text{Fib}(n)$ retorna o número de Fibonacci F_n

Função $\text{Fib}(n)$ $F \leftarrow$ vetor com posição $F[0]$ até $F[n]$ $F[0] \leftarrow 0, \quad F[1] \leftarrow 1$ **For** $i = 2$ to n $F[i] = F[i - 1] + F[i - 2]$ **End For**Return $F[n]$

Proposição (1)

Para todo $n \geq 0$, $\text{Fib}(n)$ retorna o número de Fibonacci F_n

Q: Qual é o invariante do **For** que devemos provar?

Função $\text{Fib}(n)$ $F \leftarrow$ vetor com posição $F[0]$ até $F[n]$ $F[0] \leftarrow 0, \quad F[1] \leftarrow 1$ **For** $i = 2$ to n $F[i] = F[i - 1] + F[i - 2]$ **End For**Return $F[n]$

Proposição (1)

Para todo $n \geq 0$, $\text{Fib}(n)$ retorna o número de Fibonacci F_n

Q: Qual é o invariante do **For** que devemos provar?

Proposição (Invariante do **For**)

Para todo $2 \leq i \leq n$, ao fim da iteração i do **For** temos que $F[0], F[1], \dots, F[i]$ são os números de Fibonacci apropriados

Função $\text{Fib}(n)$ $F \leftarrow$ vetor com posição $F[0]$ até $F[n]$ $F[0] \leftarrow 0, \quad F[1] \leftarrow 1$ **For** $i = 2$ to n $F[i] = F[i - 1] + F[i - 2]$ **End For**Return $F[n]$ Proposição (Invariante do **For**)

*Para todo $2 \leq i \leq n$, ao fim da iteração i do **For** temos que $F[0], F[1], \dots, F[i]$ são os números de Fibonacci apropriados*

Prova: Por indução em i

Função $\text{Fib}(n)$ $F \leftarrow$ vetor com posição $F[0]$ até $F[n]$ $F[0] \leftarrow 0, \quad F[1] \leftarrow 1$ **For** $i = 2$ to n $F[i] = F[i - 1] + F[i - 2]$ **End For**Return $F[n]$ Proposição (Invariante do **For**)

*Para todo $2 \leq i \leq n$, ao fim da iteração i do **For** temos que $F[0], F[1], \dots, F[i]$ são os números de Fibonacci apropriados*

Prova: Por indução em i

Caso base: $i = 2$. $F[0]$ e $F[1]$ são inicializados com os números de Fibonacci apropriados, então ao fim da iteração $i = 2$ temos $F[2]$ correto

Função $\text{Fib}(n)$

$F \leftarrow$ vetor com posição $F[0]$ até $F[n]$

$F[0] \leftarrow 0, F[1] \leftarrow 1$

For $i = 2$ to n

$F[i] = F[i - 1] + F[i - 2]$

End For

Return $F[n]$

Proposição (Invariante do **For**)

*Para todo $2 \leq i \leq n$, ao fim da iteração i do **For** temos que $F[0], F[1], \dots, F[i]$ são os números de Fibonacci apropriados*

Passo indutivo. Suponha que valha pra $i - 1$, prove para i

Função $\text{Fib}(n)$

$F \leftarrow$ vetor com posição $F[0]$ até $F[n]$

$F[0] \leftarrow 0, F[1] \leftarrow 1$

For $i = 2$ to n

$F[i] = F[i - 1] + F[i - 2]$

End For

Return $F[n]$

Proposição (Invariante do **For**)

*Para todo $2 \leq i \leq n$, ao fim da iteração i do **For** temos que $F[0], F[1], \dots, F[i]$ são os números de Fibonacci apropriados*

Passo indutivo. Suponha que valha pra $i - 1$, prove para i

Pela hipótese indutiva, ao fim da iteração $i - 1$ temos $F[0], \dots, F[i - 1]$ corretos...

Função $\text{Fib}(n)$

$F \leftarrow$ vetor com posição $F[0]$ até $F[n]$

$F[0] \leftarrow 0, F[1] \leftarrow 1$

For $i = 2$ to n

$F[i] = F[i - 1] + F[i - 2]$

End For

Return $F[n]$

Proposição (Invariante do **For**)

*Para todo $2 \leq i \leq n$, ao fim da iteração i do **For** temos que $F[0], F[1], \dots, F[i]$ são os números de Fibonacci apropriados*

Passo indutivo. Suponha que valha pra $i - 1$, prove para i

Pela hipótese indutiva, ao fim da iteração $i - 1$ temos $F[0], \dots, F[i - 1]$ corretos. continua correto no início da iteração i

Função Fib(n)

$F \leftarrow$ vetor com posição $F[0]$ até $F[n]$

$F[0] \leftarrow 0, F[1] \leftarrow 1$

For $i = 2$ to n

$F[i] = F[i - 1] + F[i - 2]$

End For

Return $F[n]$

Proposição (Invariante do **For**)

*Para todo $2 \leq i \leq n$, ao fim da iteração i do **For** temos que $F[0], F[1], \dots, F[i]$ são os números de Fibonacci apropriados*

Passo indutivo. Suponha que valha pra $i - 1$, prove para i

Pela hipótese indutiva, ao **fim da iteração $i - 1$** temos $F[0], \dots, F[i - 1]$ corretos. continua correto no **início da iteração i**

Nesse iteração computamos

$$\begin{aligned} F[i] &= F[i - 1] + F[i - 2] \\ &= F_{i-1} + F_{i-2} \\ &= F_i \quad \Leftarrow \text{correto!} \end{aligned}$$