Projeto e Análise de Algoritmos

Prof. Dr. Ednaldo B. Pizzolato

www.dc.ufscar.br/~ednaldo

ednaldo@dc.ufscar.br

INDUÇÃO MATEMÁTICA

Definição

- Indução matemática
  - Método poderoso de demonstração
- Dada proposição P que inclui parâmetro natural *n*
  - Se P é válida, é possível demonstrar que P é verdadeira para todo $n \in N$, através da demonstração das seguintes condições:
    1. P é verdadeira para n = 0;
    2. Por hipótese, assume-se que P é válida até n-1
    3. Considerando a hipótese, demonstra-se que P é valida para n
  - Pelas condições 1, 2 e 3, temos que P é verdadeiro para n=1. Como vale para 1, pelas condições 2 e 3, vale para n=2 e assim por diante..
  - Exemplo do dominó

Definição

- Dada proposição P e parâmetro n
- Formato de uma demonstração
  - **Demonstração de P por <u>indução em n</u>**
  - **Base da indução**: demonstrar que P vale para valor inicial
  - **Hipótese da indução**: assumir que P vale até n
  - **Passo da indução**: demonstrar que P vale para n+1
- Base da indução: pode ser demonstrada par algum valor b ≥ 0
  - Nesse caso, demonstra-se que P vale para qualquer valor $n \geq b$

Definição

- Hipótese de indução
  - Fraca: assume-se que P é verdadeira para n-1 e demonstra-se que P vale para n
  - Forte: assume-se que P é verdadeira para k < n e demonstra-se que P vale para n
- Dependendo do problema a se resolver usa-se indução fraca ou forte
  - Ex.: Somatórias, Grafos

Definição

- Atenção:
  - A indução forte difere da indução fraca (ou simples) apenas na suposição da hipótese;
  - No caso da indução forte, devemos supor que a propriedade vale para todos os casos anteriores, não somente para o anterior.

Exemplos

- Ex.1: Demonstre que, para todos números naturais x e n, x^n-1 é divisível por x-1.
- Prova: Indução em n.
  - Base: n = 1. x^1-1 = x-1, que é divisível por x-1.
  - Hipótese (fraca): x^n-1 é divisível por x-1. Nesse caso, x^n-1=k(x-1).
  - Passo: Provar que x^{n+1}-1 é divisível por x-1.
    - $x^{n+1}-1 = x^{n+}-x+x-1 = x(x^n-1)+(x-1)$. Por hipótese, $x^n-1 = k(x-1)$. Então,
    - $x^{n+1}-1 = x(k(x-1)) + (x-1) = xk(x-1)+(x-1) = (x-1)(xk + 1)$, que é divisível por (x-1).
    - Então, x^{n+1}-1 é divisível por x-1.

Exemplos

- Ex.2: Demonstre que a soma dos n números naturais $S(n)=1+2+3+...+n$ é $n(n+1)/2$.
- Prova: Indução em n.
  - Base: $n = 1$. $S(1) = 1 = 1(1+1)/2$.
  - Hipótese (fraca): $S(n-1)=(n-1)(n-1+1)/2=(n-1)n/2$.
  - Passo: Provar que $S(n)=n(n+1)/2$.
    - $S(n) = 1+2+3+...+n-2+n-1+n = S(n-1)+n$. Por hipótese, $S(n-1)=(n-1)n/2$. Então,
    - $S(n) = (n-1)(n)/2 + n = (n^2-n+2n)/2 = (n^2+n)/2 = n(n+1)/2$.
    - Então, $S(n)=n(n+1)/2$.
  - CQD.

Exemplos - tente

- Ex.3: Demonstre que, para todo natural $n \geq 1$, $S(n) = 1/2 + 1/4 + 1/8 + ... + 1/2^n < 1$.

Exemplos

- Ex.3: Demonstre que, para todo natural $n \geq 1$, $S(n) = 1/2 + 1/4 + 1/8 + \ldots + 1/2^n < 1$.
- Prova: Indução em n.
  - Base: $n = 1$. $S(1) = 1/2 < 1$.
  - Hipótese (fraca): $S(n-1) = 1/2 + 1/4 + 1/8 + \ldots + 1/2^{n-1} < 1$..
  - Passo: Provar que $S(n) < 1$.
    - Tentativa 1:
      - $S(n) = S(n-1) + 1/2^n$. Por hipótese, $S(n-1) < 1$. Então, $S(n-1) + 1/2^n < 1$???
    - Tentativa 2:
      - $S(n) = 1/2 + 1/4 + 1/8 + \ldots + 1/2^n = 1/2 + 1/2(1/2 + 1/4 + \ldots + 1/2^{n-1})$
      - $S(n) = 1/2 + 1/2\, S(n-1)$. Por hipótese, $S(n-1) < 1$, Então
      - $1/2 + 1/2\, S(n-1) < 1/2 + 1/2 \cdot 1 = 1$. Logo, $S(n) < 1$.
  - CQD.

Exemplos - tente

- Ex.4: Considere a sequência definida por $F(1)=F(2)=1$ e $F(n)=F(n-1) + F(n-2)$. Demonstre que sempre é possível calcular o n-ésimo número dessa sequência.

Exemplos

- Ex.4: Considere a sequência definida por $F(1)=F(2)=1$ e $F(n)=F(n-1) + F(n-2)$. Demonstre que sempre é possível calcular o n-ésimo número dessa sequência.
- Prova: Indução em n.
  - Base: $F(1)=1$ e $F(2)=1$, por definição.
  - Hipótese (forte): É possível calcular $F(k)$ para todo $k < n$.
  - Passo: Mostrar que é possível calcular $F(n)$.
    - Por definição, $F(n)=F(n-1) + F(n-2)$. Como, por hipótese, é possível calcular $F(n-1)$ e $F(n-2)$, então também é possível calcular $F(n)$ realizando a soma dos termos anteriores.
  - CQD.

Exercício

- Mostre por indução matemática que

$$1^3 + 2^3 + 3^3 + ... + n^3 = (1 + 2 + 3 + ... + n)^2, n \geq 1$$

Exercício

- Mostre por indução matemática que

$$1^3 + 2^3 + 3^3 + ... + n^3 = (1 + 2 + 3 + ... + n)^2,\ n \geq 1$$

dica $(1 + 2 + ... + n) = \dfrac{n.(n+1)}{2}$

Exercício

- Mostre por indução matemática que

$$1 + 3 + 5 + 7 + \ldots (2n-1) = n^2,\ n \geq 1$$

Exercício

- Valide a fórmula abaixo por indução matemática

$$1^2 + 2^2 + 3^2 + \ldots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Exercício

- Valide a fórmula abaixo por indução matemática

$$\frac{1}{1.2}+\frac{1}{2.3}+\ldots+\frac{1}{n(n+1)}=\frac{n}{n+1}$$

Erros comuns

- 1 – Verificar se a hipótese de indução pode ser usada
  - Ex.: Indução no número de vértices de um grafo G <u>conexo</u>.
    - Base: ...
    - Hipótese (fraca): Para G conexo com n-1 vértices a proposição P é verdadeira.
    - Passo: Mostrar que, para G conexo com n vértices, P é verdadeira.
      - Seja G conexo com n vértices. Escolha um vértice v de G e retire esse vértice de G. Agora tem-se G´ com n-1 vértices para o qual a hipótese vale.
      - ERRO: ao retirar um vértice qualquer, G pode se tornar desconexo e, portanto, a hipótese não pode ser aplicada.

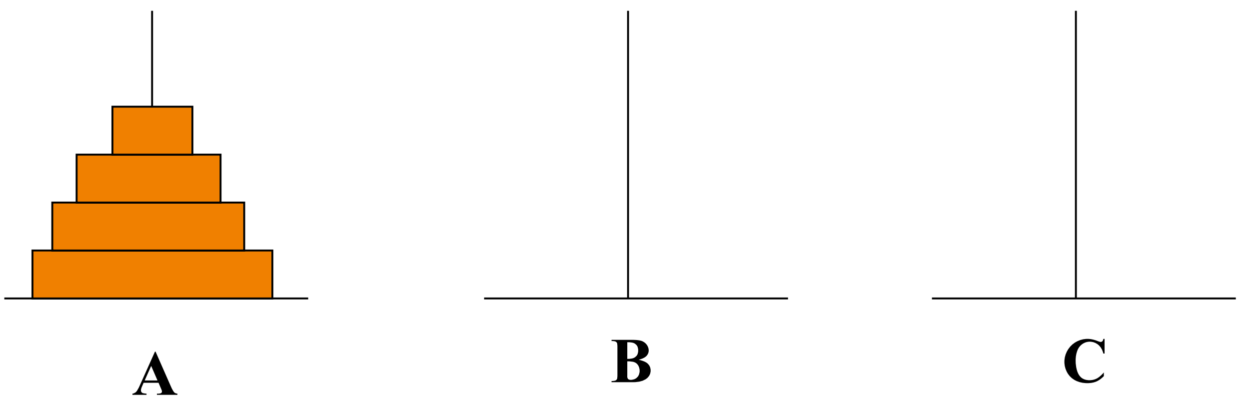
Erros comuns

- 2 – Descuido no passo da indução no tratamento de algum caso particular
  - Ex. 1: Teorema: para todo $n \geq 2$, dadas n retas não paralelas entre si, existe sempre um ponto em comum a todas elas.
  - Prova: indução em n.
    - Base: $n=2$. $P = r_1 \cap r_2$ existe, pois r_1 e r_2 são não paralelas.
    - Hipótese (fraca): Para qualquer conjunto de n-1 retas não paralelas entre si, existe um ponto P comum a todas as retas.
    - Passo: Mostrar que, dadas n retas não paralelas, existe um ponto em comum entre elas.
      - Sejam as retas $r_1, r_2, \ldots, r_n$ não paralelas entre si
      - Por hipótese: $r_1, r_2, \ldots, r_{n-1}$ têm um ponto em comum P_1
      - Por hipótese: $r_2, r_3, \ldots, r_n$ têm um ponto em comum P_2

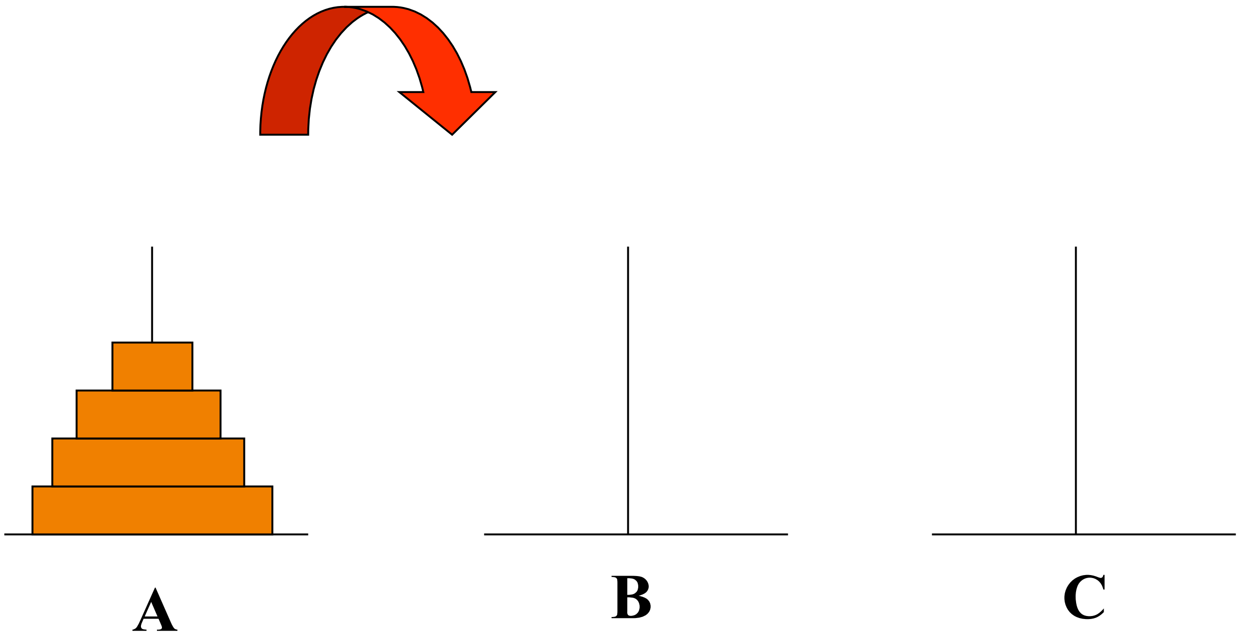
Erros comuns

- (cont.)
  - $P_1 = P_2$ pois, caso contrário, $r_2,r_3,...,r_{n-1}$ possuiriam 2 pontos distintos em comum e, nesse caso, seriam paralelas entre si (por hipótese elas não são paralelas).
  - Como $P_1 = P_2$, então $r_1,r_2,...,r_{n-1},r_n$ têm um ponto em comum $P = P_1 = P_2$.
  - CDQ?
  - ERRO: a técnica de demonstração empregada no passo indutivo assume que as retas distintas $r_1,...,r_n$ são divididas em 2 conjuntos: um com as retas $r_1,...,r_{n-1}$ e outro com as retas $r_2,...,r_n$. Nesse caso, deveríamos ter pelo menos 3 retas distintas para formar esses conjuntos: r_1,r_2 e r_3. Como a técnica empregada no passo precisa de pelo menos 3 retas, a base deveria ser demonstrada para pelo menos 3, o que não ocorreu. De fato, contra-exemplos mostram que não é possível demonstrar essa base de indução.

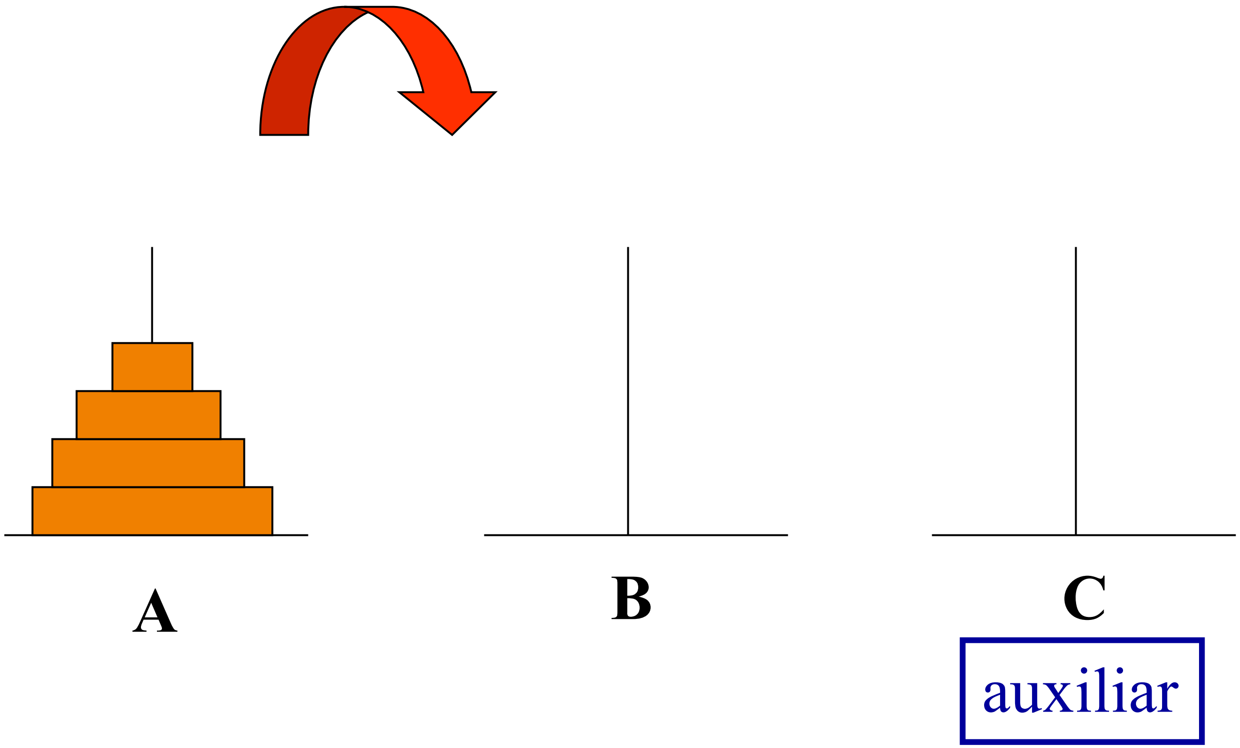
Torre de Hanói



Torre de Hanói



Torre de Hanói



Torre de Hanói - Algoritmo

Algoritmo HANOI(n, de A para B usando C)
[Torre de Hanoi, n Discos]

Torre de Hanói - Algoritmo

Algoritmo HANOI(n, de A para B usando C)
[Torre de Hanoi, n Discos]

Início

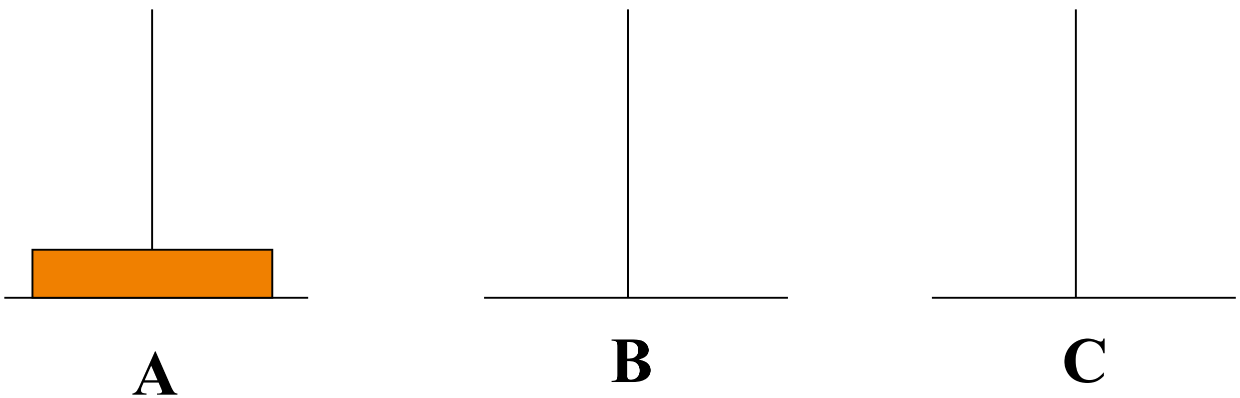
[1] **Se** n=1 **então** A $\rightarrow$ B

Torre de Hanói - Algoritmo

Algoritmo HANOI(n, de A para B usando C)
[Torre de Hanoi, n Discos]

Início

[1] **Se** n=1 **então** A $\rightarrow$ B

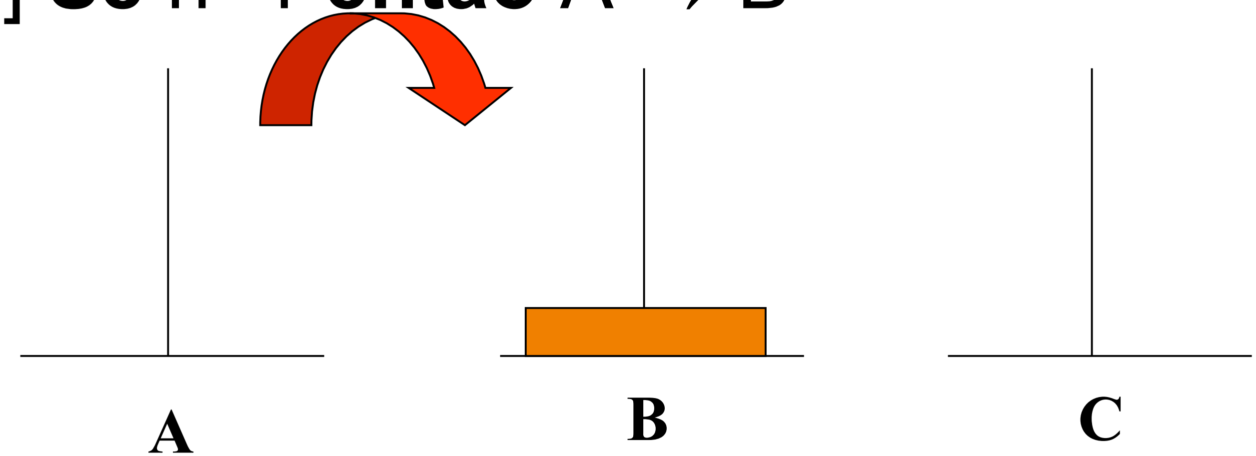


Torre de Hanói - Algoritmo

Algoritmo HANOI(n, de A para B usando C)
[Torre de Hanoi, n Discos]

Início

[1] **Se** n=1 **então** A $\rightarrow$ B



Torre de Hanói - Algoritmo

Algoritmo HANOI(n, de A para B usando C)
[Torre de Hanoi, n Discos]

Início

[1] **Se** n=1 **então** A $\rightarrow$ B

Caso contrário

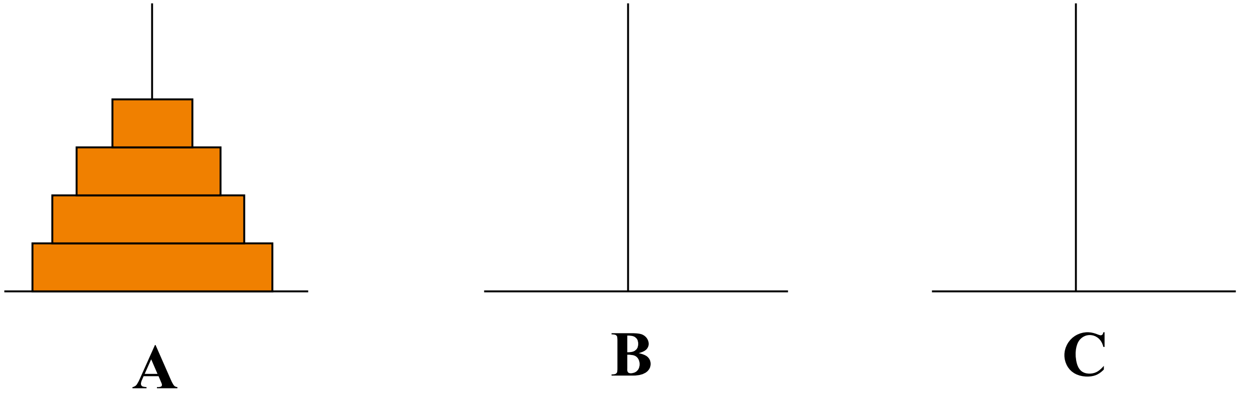
Torre de Hanói - Algoritmo

Algoritmo HANOI(n, de A para B usando C)
[Torre de Hanoi, n Discos]

Início

[1] **Se** n=1 **então** A $\rightarrow$ B

Caso contrário



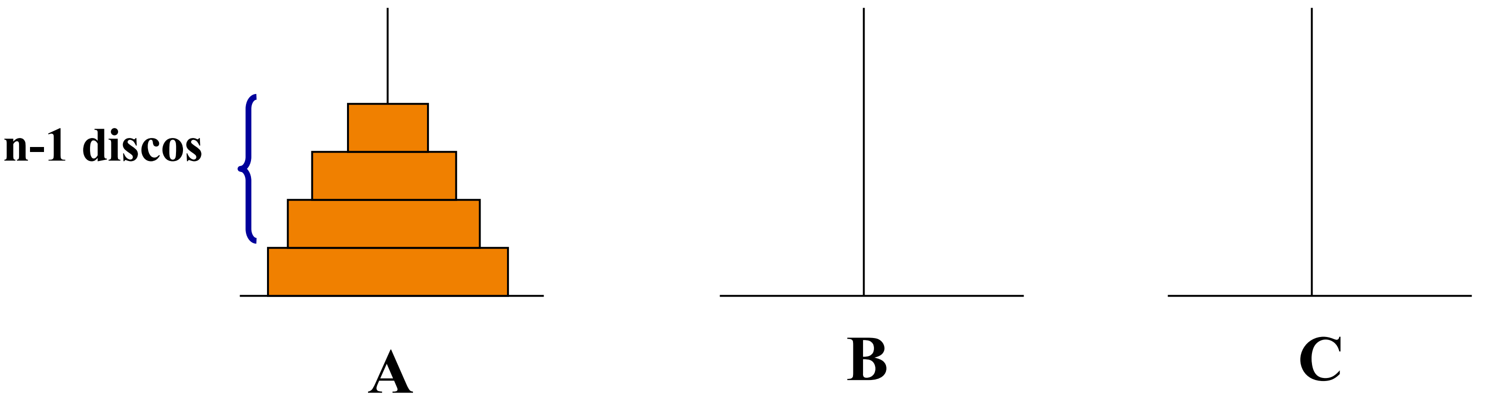
Torre de Hanói - Algoritmo

Algoritmo HANOI(n, de A para B usando C)
[Torre de Hanoi, n Discos]

Início

[1] **Se** n=1 **então** A $\rightarrow$ B

Caso contrário



Torre de Hanói - Algoritmo

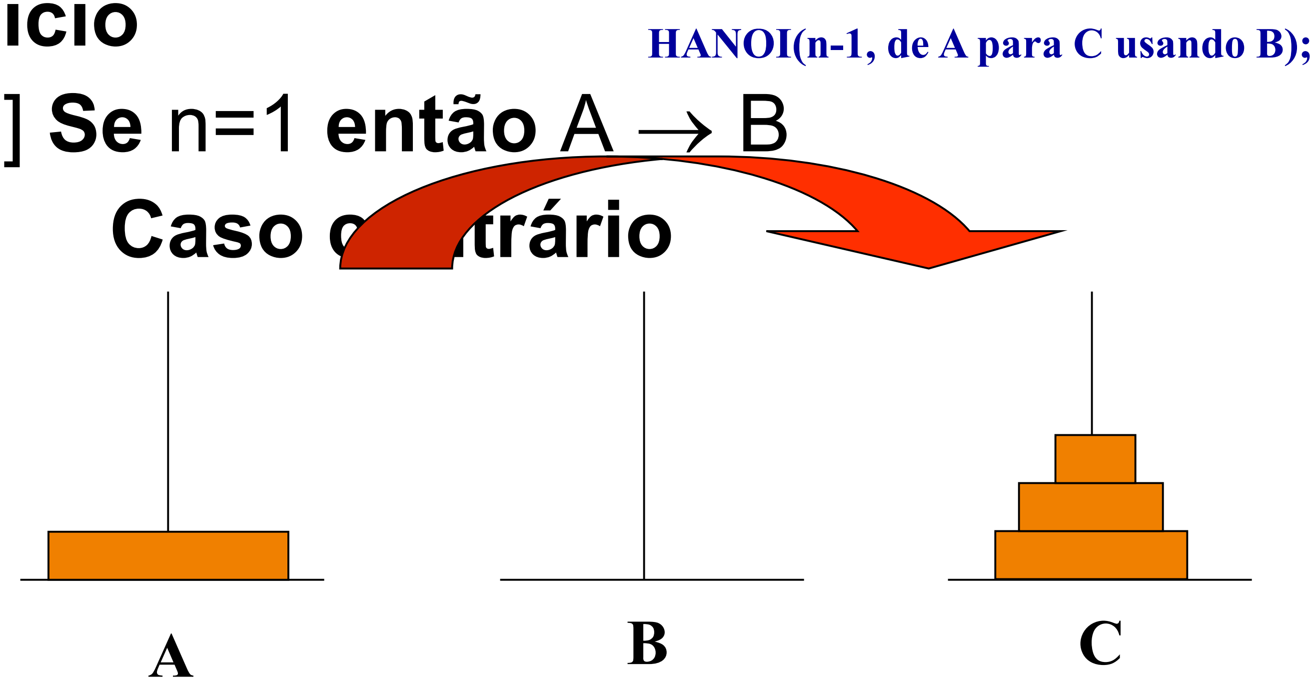
Algoritmo HANOI(n, de A para B usando C)

[Torre de Hanoi, n Discos]

Início

[1] **Se** n=1 **então** A → B

Caso contrário



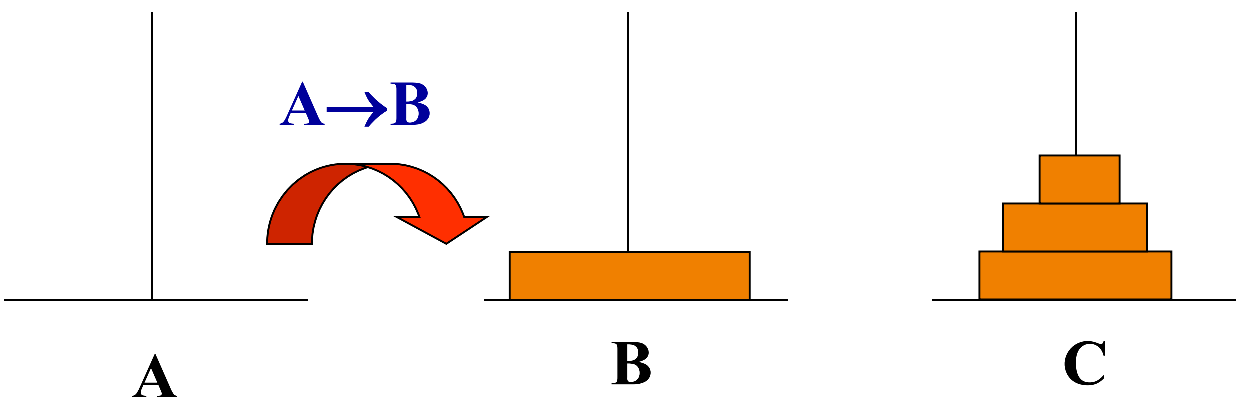
Torre de Hanói - Algoritmo

Algoritmo HANOI(n, de A para B usando C)
[Torre de Hanoi, n Discos]

Início

[1] **Se** n=1 **então** A → B

Caso contrário



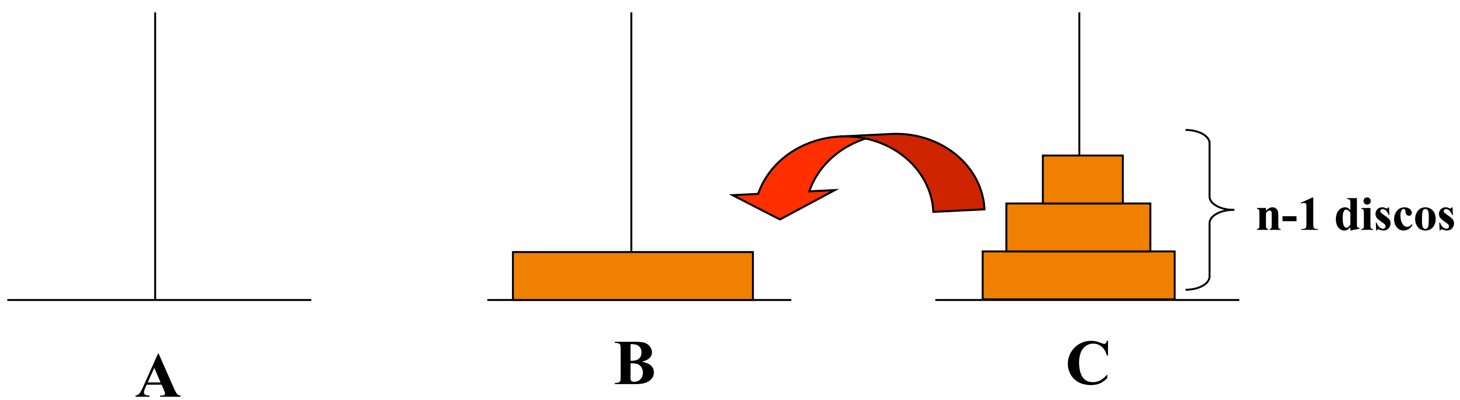
Torre de Hanói - Algoritmo

Algoritmo HANOI(n, de A para B usando C)
[Torre de Hanoi, n Discos]

Início

[1] **Se** n=1 **então** A $\rightarrow$ B

Caso contrário



Torre de Hanói - Algoritmo

Algoritmo HANOI(n, de A para B usando C)

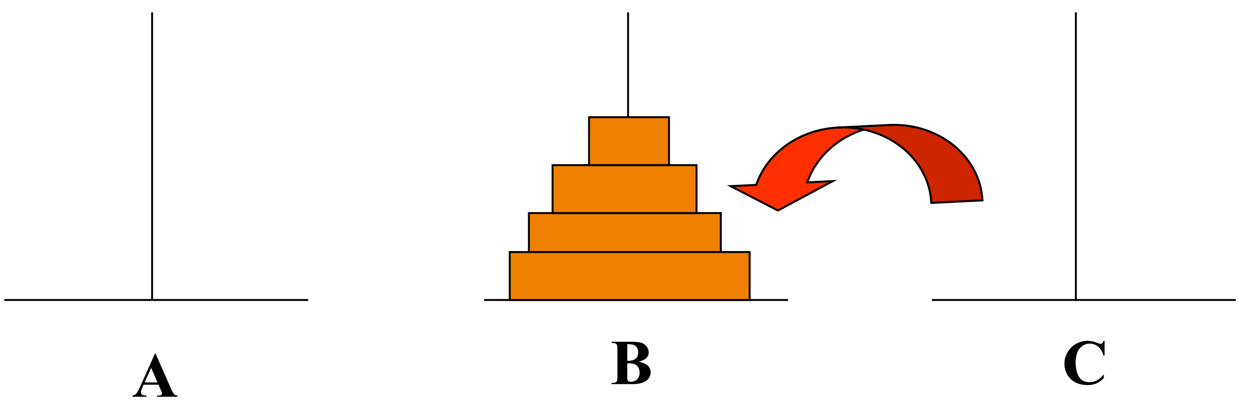
[Torre de Hanoi, n Discos]

Início

HANOI(n-1, de C para B usando A);

[1] **Se** n=1 **então** A $\rightarrow$ B

Caso contrário



Torre de Hanói - Algoritmo

Algoritmo HANOI(n, de A para B usando C)
[Torre de Hanoi, n Discos]

Início
[1] **Se** n=1 **então** A $\rightarrow$ B
Caso contrário
Início
[2] **HANOI**(n-1, de A para C usando B);
[3] A $\rightarrow$ B;
[4] **HANOI**(N-1, DE C PARA B USANDO A)
Fim;
Fim.

O Algoritmo recursivo HANOI resolve o problema das Torres de Hanoi com n discos

- Prova (por indução)
- Base da indução
- Hipótese Indutiva
- Passo da Indução

- Prova (por indução)
- Base da indução $n=1$ (um único disco)
- Hipótese Indutiva
- Passo da Indução

- Prova (por indução)
- Base da indução n=1 (um único disco)

Algoritmo HANOI(n, de A para B usando C)
[Torre de Hanoi, n Discos]

Início

[1] **Se** n=1 **então** A $\rightarrow$ B

Caso contrário

Início

[2] **HANOI**(n-1, de A para C usando B);

[3] A $\rightarrow$ B;

[4] **HANOI**(N-1, DE C PARA B USANDO A)

Fim;

Fim.

- Prova (por indução)
- Base da indução n=1 (um único disco) OK!!
- Hipótese Indutiva
- Passo da Indução

- Prova (por indução)
- Base da indução n=1 (um único disco) OK!!

- Hipótese Indutiva: O algoritmo funciona corretamente para n-1 discos (os n-1 discos são movidos corretamente de A para Z)
- Passo da Indução:

- Prova (por indução)
- Base da indução n=1 (um único disco) OK!!

- Hipótese Indutiva: f(n-1) é válida, ou seja, o algoritmo funciona corretamente para n-1 discos (os n-1 discos são movidos corretamente de X para Z)
- Passo da Indução:

- Prova (por indução)
- Base da indução n=1 (um único disco) OK!!

- Hipótese Indutiva: f(n-1) é válida, ou seja, o algoritmo funciona corretamente para n-1 discos (os n-1 discos são movidos corretamente de X para Z usando Y)

n = 2 move o menor de X para Y

move o maior de X para Z

move o menor de Y para Z

- Prova (por indução)
- Passo da Indução:provar que f(n) é válida, ou seja, que é possível mover n discos de X para Z.
- Mover n discos de X para Z significa:
  - mover n-1 discos de X para Y (hipótese)
  - mover 1 disco de X para Z (base)
  - mover n-1 discos de Y para Z (hipótese)

Recorrência

- Quando um algoritmo contém uma chamada recursiva a ele próprio, seu tempo de execução pode frequentemente ser descrito por uma recorrência.

Torre de Hanói

```
Algoritmo HANOI(n, de A para B usando C)
                                [Torre de Hanoi, n Discos]
    Início
    [1]        Se n=1 então A → B
               Caso contrário
               Início
    [2]                HANOI(n-1, de A para C usando B);
    [3]                A → B;
    [4]                HANOI(N-1, DE C PARA B USANDO A)
               Fim;
    Fim
```

Torre de Hanói

```
Algoritmo HANOI(n, de A para B usando C)
                              [Torre de Hanoi, n Discos]
    Início
    [1]  (1)  Se n=1 então A → B
              Caso contrário
              Início
    [2]               HANOI(n-1, de A para C usando B);
    [3]               A → B;
    [4]               HANOI(N-1, DE C PARA B USANDO A)
              Fim;
    Fim
```

Torre de Hanói

Algoritmo HANOI(n, de A para B usando C)
[Torre de Hanoi, n Discos]

Início

[1] (1) **Se** n=1 **então** A $\rightarrow$ B

Caso contrário

Início

[2] T(n-1) **HANOI**(n-1, de A para C usando B);

[3] A $\rightarrow$ B;

[4] **HANOI**(N-1, DE C PARA B USANDO A)

Fim;

Fim

Torre de Hanói

Algoritmo HANOI(n, de A para B usando C)
[Torre de Hanoi, n Discos]
Início
[1] (1) **Se** n=1 **então** A $\rightarrow$ B
Caso contrário
Início
[2] T(n-1) **HANOI**(n-1, de A para C usando B);
[3] (1) A $\rightarrow$ B;
[4] **HANOI**(N-1, DE C PARA B USANDO A)
Fim;
Fim

Torre de Hanói

Algoritmo HANOI(n, de A para B usando C)
[Torre de Hanoi, n Discos]

Início

[1] (1) **Se** n=1 **então** A $\rightarrow$ B

Caso contrário

Início

[2] T(n-1) **HANOI**(n-1, de A para C usando B);

[3] (1) A $\rightarrow$ B;

[4] T(n-1) **HANOI**(N-1, DE C PARA B USANDO A)

Fim;

Fim

Torre de Hanói – quebra cabeça para casa

$$T(n) = \begin{cases} 1, & \text{se } n = 1 \\ 2T(n-1) + 1, & \text{se } n > 1 \end{cases}$$

Torre de Hanói – quebra cabeça para casa

$$T(n) = \begin{cases} 1, & \text{se } n = 1 \\ 2T(n-1) + 1, & \text{se } n > 1 \end{cases}$$

Lembre-se que $\sum_{j=1}^{n} a^j = \dfrac{a^{n+1} - 1}{n-1}$

Análise de Algoritmos

Framework

Eficiencia de tempo e de espaço

- Complexidade temporal
- Complexidade espacial

- Evolução dos computadores, a questão espacial foi bem atenuada.

Tamanho da entrada

- Algoritmos demoram mais quando têm que trabalhar com maiores quantidades de dados.
- Como medir o tamanho da entrada?
  - Matriz n x n
    - Ordem n
    - Tamanho N = n x n
  - Verificação ortográfica
    - Número de palavras?
    - Número de letras?

Tamanho da entrada

- Conhecer um pouco melhor o problema permitirá avaliar o tamanho da entrada
  - Número primo

Tamanho da entrada

- Conhecer um pouco melhor o problema permitirá avaliar o tamanho da entrada
  - Número primo

    O que importa neste caso é a magnitude do número.

    Uma possível medida é o número de bits em sua representação binária.

Comparações

- Benchmark (prática)
  - Quando se compara 2 ou mais programas projetados para resolver a mesma tarefa, normalmente um conjunto padrão de dados é reservado para avaliar o desempenho da solução. Isso significa que este conjunto deve ser representativo do universo de dados aos quais o programa estará exposto e pode servir como referência de teste para outras soluções.
- **Análise teórica**
- Exemplos : reconhecimento de face, fala...

Comparações

- Benchmark (tempo)
- Análise teórica - Operações básicas
  - Uma possibilidade: contar quantas vezes cada operação básica é executada → muito detalhista
  - Normalmente os maiores custos estão nos loops mais internos

Regra 90-10

- Muitos programas executam um pequeno conjunto de instruções muitas vezes (90% do tempo de execução em 10% do código).

O que é complexidade?

A complexidade de um algoritmo consiste na quantidade de "trabalho" necessária para a sua execução, expressa em função das operações fundamentais, as quais variam de acordo com o algoritmo, e em função do volume de dados.

O que é complexidade?

- Ou seja, normalmente, programas demoram mais para executar de acordo com sua estruturação de instruções e na medida em que se aumenta a quantidade de dados de entrada.
- Esta “demora” pode ser linearmente proporcional, quadrática...
- Em alguns casos, o tempo de execução não depende somente da quantidade de dados, mas sim de sua forma.

Para que?

Para permitir a comparação teórica de soluções diferentes para o mesmo problema e identificar as melhores soluções (eficiência).

Unidades para medir tempo de execução

- $T(n) \approx c_{op}.C(n)$
- c_{op} é o custo da operação op
- $C(n)$ é o número de vezes que ela é repetida
- $T(n)$ é a estimativa do tempo de execução

Ordem de Crescimento

- A diferença em tempo de execução para entradas pequenas não é o que irá distinguir um algoritmo eficiente de um ineficiente.

entrada	Log n	N	N log N	N^2	N^3	2^n	N!
10	3.3	10	33	10^2	10^3	10^3	$3.6\ 10^6$
10^2	6.6	10^2	660	10^4	10^6	$1.3\ 10^{30}$	$9.3\ 10^{157}$
10^3	10	10^3	10^4	10^6	10^9		
10^4	13	10^4	$1.3\ 10^5$	10^8	10^2		

Ordem de Crescimento

- Algoritmos que requerem um número exponencial de operações são práticos para resolver apenas problemas de pequeno porte.

Perspectivas de análise

- Pior caso
- Caso médio
- Melhor caso

Perspectivas de análise

Pior caso

Este método é normalmente representado por **O ( ).** Por exemplo, se dissermos que um determinado algoritmo é representado por g(x) e a sua complexidade no pior caso é n, será representada por g(x) = **O**(n).

Consiste, basicamente, em assumir o pior dos casos que podem acontecer, sendo muito usado e sendo normalmente o mais fácil de se determinar.

Perspectivas de análise

Caso médio

Representa-se por θ().

Este método é, dentre os três, o mais difícil de se determinar pois necessita de análise estatística (muitos testes). No entanto, é muito usado pois é também o que representa mais corretamente a complexidade do algoritmo.

Perspectivas de análise

Melhor caso

Representa-se por Ω ( )

Método que consiste em assumir que vai acontecer o melhor. Pouco usado. Tem aplicação em poucos casos.

Limite Superior

Seja dado um problema, por exemplo, multiplicação de duas matrizes quadradas de ordem n (n*n). Conhecemos um algoritmo para resolver este problema(pelo método trivial) de complexidade $O(n^3)$. Sabemos assim que a complexidade deste problema não deve superar $O(n^3)$, uma vez que existe um algoritmo desta complexidade que o resolve. Um limite superior (upper bound) deste problema é $O(n^3)$. O limite superior de um algoritmo pode mudar se alguém descobrir um algoritmo melhor. Isso de fato aconteceu com o algoritmo de Strassen que é de $O(n^{\log 7})$. Assim o limite superior do problema de multiplicação de matrizes passou a ser $O(n^{\log 7}) \approx O(n^{2.807})$. Outros pesquisadores melhoraram ainda mais este resultado. Atualmente, o melhor resultado é o de Coppersmith e Winograd:

$$O(n^{2.376}).$$

Limite Superior

O limite superior de um algoritmo é parecido com o record mundial de uma modalidade de atletismo. Ele é estabelecido pelo melhor atleta (algoritmo) do momento. Assim como o record mundial o limite superior pode ser melhorado por um algoritmo (atleta) mais veloz.

Limite Inferior

Às vezes é possível demonstrar que para um dado problema, qualquer que seja o algoritmo a ser usado o problema requer pelo menos um certo número de operações. Essa complexidade é chamada Limite inferior (Lower Bound). Veja que o limite inferior dependo do problema mas não do particular algoritmo. Usamos a letra Ω em lugar de O para denotar um limite inferior.

Para o problema de multiplicação de matrizes de ordem *n*, apenas para ler os elementos das duas matrizes de entrada leva $O(n^2)$. Assim uma cota inferior trivial é $\Omega(n^2)$.

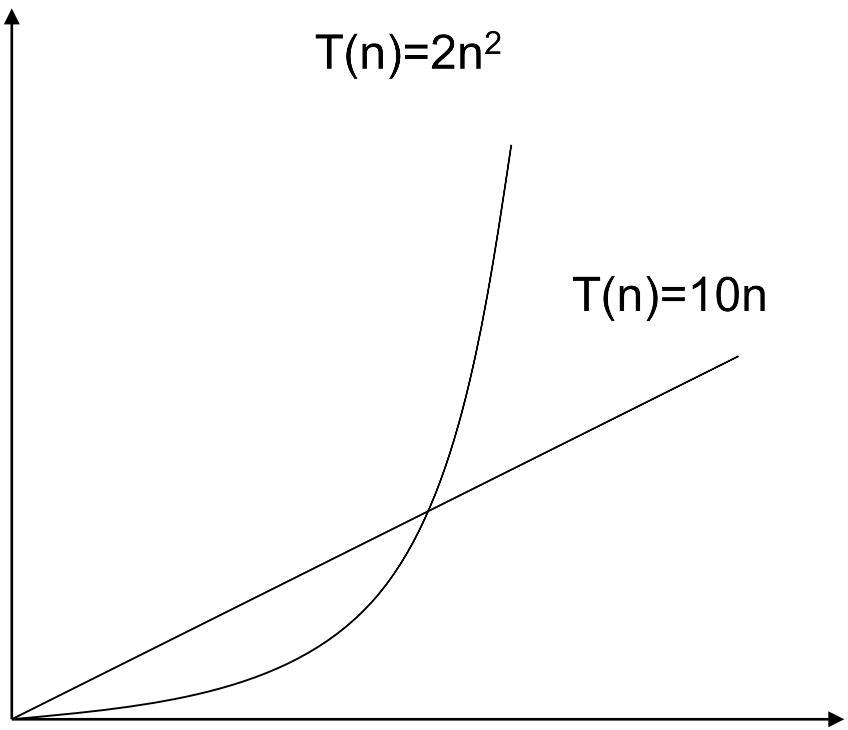
Limite Inferior

Na analogia anterior, um limite inferior de uma modalidade de atletismo não dependeria mais do atleta. Seria algum tempo mínimo que a modalidade exige, qualquer que seja o atleta. Um limite inferior trivial para os 100 metros seria o tempo que a velocidade da luz leva a percorrer 100 metros no vácuo.

Se um algoritmo tem uma complexidade que é igual ao limite inferior do problema então o algoritmo é ótimo.

O algoritmo de CopperSmith e Winograd é de $O(n^{2.376})$ mas o limite inferior é de $\Omega(n^2)$. Portante não é ótimo. É possível que este limite superior possa ainda ser melhorado.

Comparando



Tempo	Tam. A	Tam. B
1s	10	22
10s	100	70
100s	1.000	223
1.000s	10.000	707

Continua…

- Continua na próxima aula

THE END