Algoritmos, Recursividade

Prof. Ricardo Reis
Universidade Federal do Ceará
Campus de Quixadá

17 de março de 2013

1 Introdução

Recursividade ou **recorrência** é uma técnica utilizada na elaboração de algoritmos que se caracteriza pela presença de uma ou mais chamadas ao próprio algoritmo. Muitos algoritmos são mais facilmente escritos utilizando-se recursividade. Algoritmos recursivos, com menor ou maior esforço, podem ser reescritos sem uso de recursividade (versão *iterativa*). Na prática a recursão é computacionalmente mais custosa que a iteração e sempre que possível é aconselhável substituir um algoritmo recursivo por sua versão iterativa.

Considere um algoritmo implementado na forma de uma função f. Se f faz uma chamada a si mesma, então diz-se que essa chamada em particular é uma *chamada recursiva* e que a instância de f onde ocorreu a chamada recursiva é uma *função requisitante*. Um *processo recursivo* constitui-se de um encadeamento de funções requisitantes que se atrelam por chamadas recursivas gerando uma cadeia de dependências, ou seja, se f faz uma chamada recursiva então esta chamada poderá ser função requisitante de uma nova chamada recursiva e assim sucessivamente.

Esquematicamente um processo recursivo é constituído de uma sequência de instâncias de f, ou seja, $f_1, f_2, f_3, \cdots, f_m$, onde a finalização de uma dada chamada f_k, com $k \in \{1, 2, 3...m\}$, depende da finalização de f_{k+1}. A primeira chamada do processo recursivo, f_1, é externa, ou seja, é requisitada fora de f para onde é devolvido o controle depois de encerrada. O valor m, denotado por *profundidade da recursividade*, representa o comprimento da sequência de funções requisitantes. Se m é finito então a chamada recursiva em f_m não é executada o que naturalmente provocará um desenlace sucessivo de dependências em ordem regressiva desde f_m até f_1. Esta finitude é normalmente programada por um teste que falhará no m-ésimo estágio do processo impedido uma nova chamada recursiva. Este teste é denominado *condição de parada da recursão.*

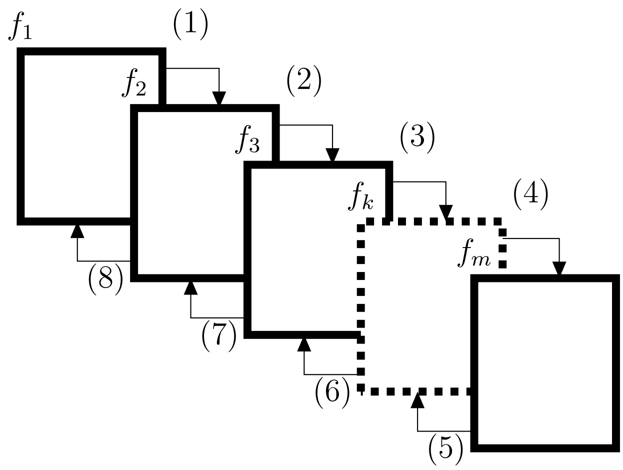

Figura 1: Esquema de um processo recursivo.

Na figura-1 é ilustrado o esquema de um processo recursivo. Os retângulos representam as instâncias da função f. As setas na parte superior denotam as chamadas recursivas e partem dos respectivos requisitantes. As setas na parte inferior denotam a devolução de controle de uma dada instância de f ao seu respectivo requisitante.

A sequência em que ocorrem chamadas e retornos é marcada numericamente.

2 Leis de Recorrência

Uma Lei de Recorrência é uma equação matemática expressa utilizando-se recursividade. Em geral é denotada por T e pode possuir uma ou mais variáveis independentes. São úteis para expressar algoritmos e suas respectivas complexidades [1].

Estudaremos nesta sessão, através de ilustrações, como construir algoritmos recursivos a partir de suas respectivas leis de recorrência.

ILUSTRAÇÃO-1 (Fatoração de Naturais)
Como exemplo inicial citamos o cálculo do fatorial de números naturais. O *fatorial* de um número natural é definido como sendo o produto entre este número e seus antecessores positivos. O fatorial de zero é definido como sendo 1. Denotando por $T(n)$ o fatorial de um número natural n então podemos escrever seu valor na forma da seguinte lei de recorrência,

$$T(n) = \begin{cases} n \cdot T(n-1) & n \geq 2 \\ 1 & n < 2 \end{cases} \tag{1}$$

A primeira parte da equação-(1) é aplicada a valores de n maiores ou iguais a 2 e corresponde a componente recorrente propriamente dita pois necessita de T para ser definida. Sua interpretação é a seguinte, *O fatorial de um número é igual ao produto entre ele e o fatorial de seu antecessor* (você pode provar isso por indução). A segunda parte da lei de recorrência corresponde a condição de parada da recursão e trata especificamente dos fatoriais de 0 e 1 (que valem ambos, 1).

O algoritmo-1 representa a implementação da equação-(1). A função FAT recebe um inteiro n e calcula recursivamente seu fatorial. As recorrências ocorrem na linha-3 enquanto o teste na linha-2 permitir. O desenlace de dependências principia quando este teste falha e a linha-5 é executada.

Algoritmo 1 Fatorial de um número natural

1: **Função** FAT(n)
2:     **Se** $n > 1$ **então**
3:         **Retorne** $n \cdot \text{FAT}(n-1)$
4:     **senão**
5:         **Retorne** 1

ILUSTRAÇÃO-2 (Série de Fibonacci)
Uma recorrência poderá possuir mais de uma dependência, ou seja, mais de uma chamada recursiva poderá ser necessária para expressar um algoritmo. O exemplo clássico é o cálculo no n-ésimo termo da série de Fibonacci. Esta série é infinita, possui os dois primeiros termos iguais a 1 e os demais são calculados pela soma dos dois termos imediatamente anteriores. A lei de recorrência, $T(n)$, que calcula o n-ésimo termo da série de Fibonacci é dada por,

$$T(n) = \begin{cases} T(n-1) + T(n-2) & n \geq 3 \\ 1 & n < 3 \end{cases} \tag{2}$$

onde n é um inteiro positivo. A primeira parte da equação-2 contém duas recorrências, $T(n-1)$ e $T(n-2)$, que calculam os dois termos imediatamente anteriores ao n-ésimo termo. A segunda parte representa a condição de parada e trata dos dois primeiros termos da série (que valem ambos, 1).

Algoritmo 2 n-ésimo termo da série de Fibonacci

1: **Função** FIBO(n)
2:     **Se** $n < 3$ **então**
3:         **Retorne** 1
4:     **senão**
5:         **Retorne** $\text{FIBO}(n-1) + \text{FIBO}(n-2)$

No Algoritmo-2 a função FIBO implementa a lei de recorrência da equação-2. As recorrências se procedem na linha-5 e a parada na linha-3 conforme teste da linha-2.

ILUSTRAÇÃO-3 (Máximo Divisor Comum)
Outro exemplo clássico de recorrência é o

[1] assunto abordado em outro artigo

a	b	$a \bmod b$
29029	70091	29029
70091	29029	12033
29029	12033	4963
12033	4963	2107
4963	2107	749
2107	749	609
749	609	140
609	140	49
140	49	42
49	42	7
42	7	0
7	0	

Figura 2: Etapas do cálculo do máximo divisor comum dos números 29029 e 70091 conforme algoritmo de Euclides.

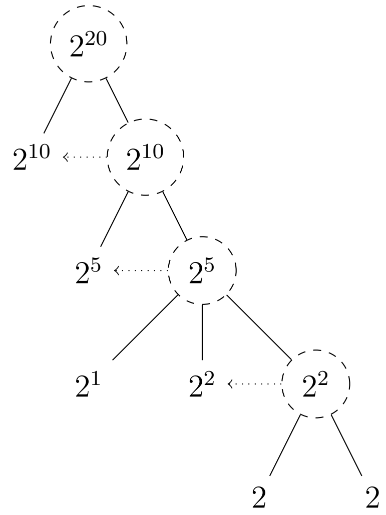

Figura 3: Cálculo recursivo da potência 2^{20} (potência rápida).

cálculo do máximo divisor comum de dois números naturais. O máximo divisor comum de dois naturais a e b, ou $mdc(a, b)$, é definido como o maior número natural pelo qual a e b podem ser divididos (ou que dividem a e b, pela notação de teoria dos números).

Um algoritmo para determinação de $mdc(a, b)$, conhecido como *algoritmo de Euclides*, é descrito a seguir. Inicialmente calcula-se o resto da divisão de a por b, se b for não nulo. Em seguida atribui-se a a o valor de b e a b o valor do resto de divisão encontrado. Repete-se este processo até que b se anule. O último valor assumido por a será o valor de mdc procurado.

A figura-2 ilustra as etapas do algoritmo de Euclides quando aplicado aos números $a = 29029$ e $b = 70091$. As três colunas apresentam respectivamente os valores de a, b e resto de divisão de a por b, durante o processo. As setas indicam as redefinições de a e b a partir de valores do passo imediatamente anterior. O valor 7 é obtido como resposta.

Pelas considerações feitas tem-se que a lei de recorrência para o algoritmo de Euclides é dada por,

$$T(a,b) = \begin{cases} T(b, a \bmod b) & b \neq 0 \\ a & a \neq 0 \quad b = 0 \end{cases} \tag{3}$$

onde $T(a,b)$ representa o mdc de a e b e $\bmod$ denota o operador resto de divisão. Note que não existe mdc quando a e b são iguais a zero. O Algoritmo-3 implementa o método de Euclides.

Algoritmo 3 Algoritmo de Euclides para cálculo de mdc

```
Função MDC(a, b)
    Se b ≠ 0 então
        Retorne MDC(b, a mod b)
    senão
        Retorne a
```

ILUSTRAÇÃO-4 (Potência Rápida)

Uma forma recorrente e eficiente de determinar o valor da potência a^n, com $a \in \mathbb{R}$ e $n \in \mathbb{N}$, é descrita a seguir. Quando n é par então $a^n = a^{\lfloor n/2 \rfloor} \cdot a^{\lfloor n/2 \rfloor}$ e quando n é impar então $a^n = a^{\lfloor n/2 \rfloor} \cdot a^{\lfloor n/2 \rfloor} \cdot a$. Em ambos os casos o sub-resultado $a^{\lfloor n/2 \rfloor}$ é utilizado duas vezes, ou seja, se for calculado uma vez pode ser reaproveitado. Isto diminui consideravelmente, ao final, o total de multiplicações efetuadas em relação ao procedimento aberto $a \cdot a \cdot a \cdot \cdots \cdot a$ que efetua $n - 1$ multiplicações.

A figura-3 ilustra os passos do algoritmo descrito para o caso da potência 2^{20}. As arestas indicam desmembramentos nas formas $a^n \Rightarrow a^{\lfloor n/2 \rfloor} \cdot a^{\lfloor n/2 \rfloor}$ ou $a^n \Rightarrow a \cdot a^{\lfloor n/2 \rfloor} \cdot a^{\lfloor n/2 \rfloor}$. Os elementos circulados são determinados recursivamente pelo produto de seus desmembramentos. As setas indicam reuso, ou seja, o valor na posição de onde partem é reutilizado na posição aonde chegam.

Matematicamente este processo de determinação de potência é descrito pela seguinte lei de recorrência,

$$T(n) = \begin{cases} \left[T\left(\left\lfloor \frac{n}{2} \right\rfloor\right)\right]^2 \cdot Corr(n) & n > 1 \\ a & n = 1 \\ 1 & n = 0 \end{cases} \tag{4}$$

$$Corr(n) = \begin{cases} 1 & \texttt{se } n \texttt{ é par} \\ a & \texttt{se } n \texttt{ é ímpar} \end{cases} \tag{5}$$

onde $T(n)$, na equação-4, denota a potência a^n e $corr(n)$, equação-5, a correção de desmembramento de potência quando o expoente é ímpar. O quadrado na primeira parte da equação-4 indica o reuso do valor $T\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$ obtido recursivamente. As outras duas partes ($n < 2$) indicam condições de parada, ou seja, quando desmembramentos não são mais possíveis. De fato $a^1 = a$ e $a^0 = 1$ (se $a \neq 0$). Note que não é tratada a indeterminação 0^0.

Algoritmo 4 Potência Rápida

1: **Função** QPOT(a, n)
2: **Se** $n > 1$ **então**
3: $x \leftarrow$ QPOT($a, \lfloor n/2 \rfloor$)
4: **Se** $n \bmod 2 = 0$ **então**
5: **Retorne** $x \cdot x$
6: **senão**
7: **Retorne** $x \cdot x \cdot a$
8: **senão** $\triangleright$ $n = 0$ ou $n = 1$
9: **Se** $n = 1$ **então**
10: **Retorne** a
11: **senão**
12: **Retorne** 1

O Algoritmo-4 implementa a equação-4. A recorrência ocorre na linha-3. A variável auxiliar x é responsável por manter o cálculo parcial da potência e possibilitar o reuso, ou seja, $x \cdot x$ na linha-5 e $x \cdot x \cdot a$ na linha-7 (potência corrigida, conforme equação-5). As condições de parada ocorrem na linha-10 e linha-12.

3 Recursividade e Vetores

Um algoritmo recursivo que recebe um vetor como argumento normalmente o faz utilizando referências para permitir que todas as iterações do processo de fato se refiram sempre ao mesmo vetor sem necessidade de cópia de dados. A grande vantagem disso é a possibilidade de processar recursivamente partes distintas de um mesmo vetor como se fossem vetores independentes gerando uma associação de processamentos que ao final afetam o vetor como um todo. As ilustrações a seguir tratam da construção de algoritmos recursivos envolvendo vetores.

ILUSTRAÇÃO-5 (Soma de Elementos de um Vetor)

Consideremos a soma dos elementos de um vetor de números. Podemos imaginar este problema de forma recorrente da seguinte maneira: a soma dos elementos de um vetor é igual a soma do primeiro elemento com a soma dos elementos do sub-vetor dos elementos restantes. Matematicamente temos,

$$T(p, q) = \begin{cases} M[p] + T(p+1, q) & p \leq q \\ 0 & p > q \end{cases} \tag{6}$$

onde M representa um vetor, p e q índices válidos de M e $T(p, q)$ a soma dos elementos de M entre os índices p e q (inclusiva). A expressão recorrente $M[p] + T(p+1, q)$ traduz a soma do elemento de posição p, $M[p]$, com a soma dos elementos do sub-vetor de M formado entre as posições $p+1$ e q, ou seja, $T(p+1, q)$. A condição de parada são faixas de elementos de comprimento nulo, ou seja, $p > q$ (segunda parte da equação-6).

Algoritmo 5 Soma recursiva das chaves de um vetor de inteiro

Função VSOMA(ref M, p, q)
Se $p > q$ **então**
Retorne 0
senão
Retorne $M[p] +$ VSOMA($M, p+1, q$)

O Algoritmo-5 implementa a equação-6. Numa chamada externa à função VSOMA do Algoritmo-5 que repassa um vetor M de comprimento n então os valores repassados a p e q deverão ser respectivamente 1 e n caso seja procurada a soma de todos os elementos de M. Exemplo,

$M \leftarrow [2, 7, 13, 21]$
$x \leftarrow$ **VSoma**$(M, 1, 4)$
Escreva x ▷ Imprime 43

ILUSTRAÇÃO-6 (Inversão de um Vetor)
Inverter um vetor significa dispor no próprio vetor suas chaves na ordem inversa em que se encontram. O processo é facilmente solucionado por iteração, mas sugerimos aqui o seguinte algoritmo de inversão recursivo: trocam-se a primeira com a última chave e depois inverte-se recursivamente o sub-vetor formado entre a segunda e a penúltima posição. Matematicamente,

$$T(p, q) = \{M[p] \Leftrightarrow M[q]\} \oplus T(p+1, q-1) \qquad p < q \tag{7}$$

onde $T(p, q)$ representa a inversão do vetor M entre as posições de índices p e q. O operador $\oplus$ associa a troca dos elementos das posições p e q, com notação $M[p] \Leftrightarrow M[q]$, à inversão recorrente do sub-vetor restante, ou seja, $T(p + 1, q - 1)$. A parada da recursão se dá quando $p \geq q$ (note que não existe uma condição de parada explícita na equação-7 em virtude de a inversão ser um processo *in locus*)

A função INVERTER no Algoritmo-6 implementa a equação-7. A troca entre o primeiro e último elementos ocorre na linha-3, a recorrência na linha-4 e a condição de continuidade da recursão na linha-2.

Algoritmo 6 Inversão recursiva de um vetor de inteiros

1: **Função** INVERTER(M, p, q)
2:     **Se** $p < q$ **então**
3:         $M[p] \Leftrightarrow M[q]$
4:         INVERTER$(M, p + 1, q - 1)$

ILUSTRAÇÃO-7 (Valor Máximo de um Vetor)
Um algoritmo recorrente para o problema de determinação da chave de valor máximo em um vetor de números inteiros é descrito a seguir. Divide-se o vetor em duas partes e determina-se recursivamente o valor máximo dos sub-vetores obtidos. O valor procurado é o maior entre estes dois valores. Matematicamente,

$$T(p, q) = \begin{cases} \text{MÁX}\,[T(p, r), T(r + 1, q)] & p < q \\ M[p] & p = q \end{cases} \tag{8}$$

onde,

$$r = \left\lfloor \frac{p + q}{2} \right\rfloor \tag{9}$$

e,

$$\text{MÁX}(a, b) = \begin{cases} a & a \geq b \\ b & a < b \end{cases} \tag{10}$$

Na equação-8, $T(p, q)$ representa o elemento de valor máximo no vetor M no intervalo entre as posições de índices p e q. O parâmetro r, equação-9, representa o índice médio entre p e q o qual impõe a divisão da faixa aproximadamente ao meio (note a presença do operador piso [2]). A função MÁX , equação-10, determina o maior entre os dois valores que recebe como argumento. Os intervalos dados às chamadas recursivas são disjuntos, ou seja, a chamada $T(p, r)$ inclui a posição r, mas a chamada $T(r + 1, q)$ não. A parada da recursão, denotada pela segunda parte da equação-8, se dá quando a faixa avaliada tem comprimento 1 (o maior elemento é o único da faixa, ou seja, máximo $= M[p] = M[q]$).

[2] O *piso* de um número real x, ou $\lfloor x \rfloor$, é o maior número inteiro menor ou igual a x. Já o *teto* de um valor real x, ou $\lceil x \rceil$, é o menor inteiro maior ou igual a x. Uma propriedade importante destes operadores é,

$$\left\lfloor \frac{n}{2} \right\rfloor + \left\lceil \frac{n}{2} \right\rceil = n \tag{11}$$

onde n é um número inteiro.

O Algoritmo-7 implementa a equação-8. Os parâmetros x e y representam respectivamente os valores máximos, obtidos recursivamente, referentes as primeira e segunda metades avaliadas na faixa $p \cdots q$ de M.

Algoritmo 7 Determinação recursiva do elemento máximo num vetor.

```
Função VMAX(M, p, q)
    Se p = q então
        Retorne M[p]
    senão
        r ← ⌊(p + q)/2⌋
        x ← VMAX(M, p, r)
        y ← VMAX(M, r + 1, q)
        Se x ≥ y então
            Retorne x
        senão
            Retorne y
```

4 Resolvendo Leis de Recorrência

Nesta sessão exploraremos um método que permite solucionar uma lei de recorrência, ou seja, determinar o valor de $T(n)$ em função apenas de n.

O método consiste em expandir a parte recorrente da lei de recorrência por substituições diretas da própria lei de recorrência por um total de passos que se permita observar um padrão de formação. É importante nesse processo não efetuar simplificações algébricas que ocultem o padrão. Uma vez determinado o padrão escreve-se a lei de recorrência como função de um passo k arbitrário de recursão. Por fim, utilizando-se condições de parada, eliminam-se os termos recorrentes que devem se apresentar como funções de k. Os exemplos a seguir ilustram o procedimento.

ILUSTRAÇÃO-8 (Fatorial de Números Naturais)

Consideremos inicialmente a equação-(1) que modela a fatoração de números naturais. O desenvolvimento a seguir mostra a expansão da parte recorrente desta equação por vários passos até que se defina um padrão num passo arbitrário k,

$$\begin{aligned} T(n) &= n \cdot T(n-1) \\ &= n \cdot (n-1) \cdot T(n-2) \\ &= n \cdot (n-1) \cdot (n-2) \cdot T(n-3) \\ &\cdots \\ &= n \cdot (n-1) \cdot (n-2) \cdots (n-(k-1)) \cdot T(n-k) \\ &= \left[\prod_{i=0}^{k-1} (n-i) \right] \cdot T(n-k) \end{aligned} \tag{12}$$

A equação-(12) representa o padrão num passo arbitrário k. Fazendo $k = n$ ela se torna,

$$T(n) = \left[\prod_{i=0}^{n-1} (n-i) \right] \cdot T(0)$$

Da equação-(1) sabe-se que $T(0) = 1$ e logo tem-se,

$$T(n) = \left[\prod_{i=0}^{n-1} (n-i) \right]$$

Fazendo a substituição de variável $j = n - i$ então este produto se torna,

$$\begin{aligned} T(n) &= \left[\prod_{j=1}^{n} j \right] \\ &= n! \end{aligned}$$

que é naturalmente o resultado esperado.

ILUSTRAÇÃO-9

Solucionemos agora a seguinte lei de recorrência,

$$T(n) = \begin{cases} 3 \cdot T(n-1) + 1 & n > 0 \\ 0 & n = 0 \end{cases} \tag{13}$$

Efetuando-se a expansão temos,

$$\begin{aligned}
T(n) &= 3 \cdot T(n-1) + 1 \\
&= 3 \cdot [3T(n-2) + 1] + 1 \\
&\Rightarrow 3^2 T(n-2) + 3 + 1 \\
&= 3^2 \cdot [3T(n-3) + 1] + 3 + 1 \\
&\Rightarrow 3^3 T(n-3) + 3^2 + 3 + 1 \\
&= 3^3 \cdot [3T(n-4) + 1] + 3^2 + 3 + 1 \\
&\Rightarrow 3^4 T(n-4) + 3^3 + 3^2 + 3 + 1 \\
&\dots \\
&= 3^k \cdot T(n-k) + \sum_{i=0}^{k-1} 3^i \qquad (14)
\end{aligned}$$

onde a equação-(14) é o padrão da lei de recorrência na equação-(13) num passo arbitrário k. Fazendo $n = k$ na equação-(14) obtém-se,

$$T(n) = 3^n \cdot T(0) + \sum_{i=0}^{n-1} 3^i$$

Da equação-(13) tem-se que $T(0) = 0$ e logo a última equação pode ser reduzida para,

$$T(n) = \sum_{i=0}^{n-1} 3^i$$

Utilizando a fórmula de soma dos termos de uma série de potências [3] temos,

$$T(n) = \frac{3^n - 1}{2}$$

ILUSTRAÇÃO-10

Resolver a seguinte lei de recorrência,

$$T(n) = \begin{cases} 2 \cdot T(n/2) + n & n > 1 \\ 0 & n = 1 \end{cases} \qquad (16)$$

Efetuando-se a expansão da equação-(16) temos,

$$\begin{aligned}
T(n) &= 2 \cdot T(n/2) + n \\
&= 2 \cdot [2 \cdot T(n/2^2) + n/2] + n \\
&\Rightarrow 2^2 \cdot T(n/2^2) + 2n \\
&= 2^2 \cdot [2 \cdot T(n/2^3) + n/2^2] + 2n \\
&\Rightarrow 2^3 \cdot T(n/2^3) + 3n \\
&\dots \\
&= 2^k \cdot T(n/2^k) + kn \qquad (17)
\end{aligned}$$

onde a equação-(17) é o padrão de recorrência da equação-(16). Fazendo $\frac{n}{2^k} = 1$ temos $n = 2^k \Rightarrow k = \log_2 n$. Utilizando este valor de k na equação-(17) encontramos,

$$\begin{aligned}
T(n) &= 2^{\log_2 n} \cdot T(n/2^{\log_2 n}) + n \cdot \log_2 n \\
&= n \cdot T(1) + n \cdot \log_2 n
\end{aligned}$$

Da equação-(16) sabe-se que $T(1) = 0$ e a equação se torna,

$$T(n) = n \cdot \log_2 n$$

5 O Problema da Torre de Hanói

O problema clássico *Torre de Hanói* consiste em movimentar uma dada quantidade de discos empilhados em um pino A para um pino B utilizando um pino C como auxiliar. Os discos possuem diâmetros diferentes e as movimentações devem seguir duas regras,

i Só pode ser movimentado um disco de cada vez.

ii Um disco só pode ser colocado sobre um de diâmetro maior ou num pino sem discos.

No estado inicial o pino A deve conter todos os discos empilhados em ordem decrescente de diâmetro de baixo para cima e os pinos B e C devem estar vazios. A figura-4 ilustra os estados inicial e final do problema da Torre de Hanói.

[3] Soma dos termos de uma série de potências,

$$\sum_{i=a}^{b} c^i = \frac{c^{b+1} - c^a}{c - 1} \qquad (15)$$

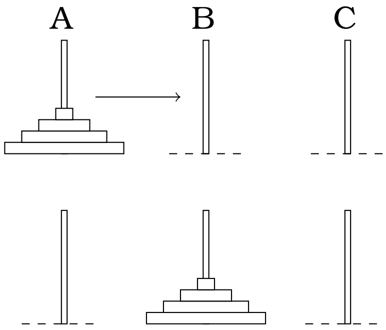

Figura 4: Esquema do problema da Torre de Hanói.

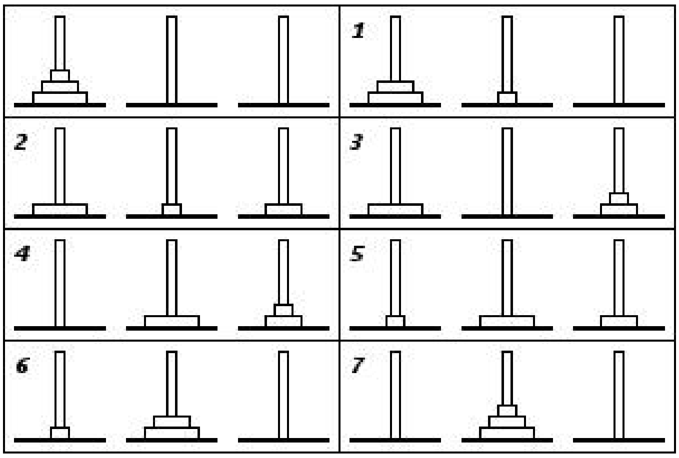

Figura 5: Estágios do problema Torre de Hanói com 3 pinos e 3 discos.

A figura-5 ilustra passo a passo os estágios da resolução do problema de Torre de Hanói com 3 pinos e 3 discos. São executados 7 movimentações numeradas na figura por estágio.

O algoritmo para resolução do problema Torre de Hanói com 3 pinos e n discos é mais facilmente construído com recursão. Sua construção é feita a seguir a partir de observações da figura-5. Inicialmente deve-se considerar a movimentação que leva do estágio 3 para o estágio 4. Esta é a primeira movimentação definitiva do processo, ou seja, no estágio 4 a posição do disco (o de maior diâmetro) é definitiva (não sofrerá movimentações futuras). Outro aspecto importante do estágio 4 é que o novo layout de distribuição dos discos recorre novamente ao problema original, ou seja, a torre em C deve ser movida para B utilizando A como auxiliar. Notemos que nessa recorrência o primeiro disco posto em B é desconsiderado tanto por já estar em sua posição definitiva quanto pelo fato de não impor restrições a movimentações dos demais discos (haja vista ter diâmetro maior que o deles). De forma geral se n discos estão empilhados em A então deve-se primeiramente mover seus $n-1$ discos para C, em seguida mover o disco que ficou em A para B e por fim mover os $n-1$ em C para B encerrando o processo. As movimentações intermediárias resolvem-se recursivamente pelo mesmo princípio mudando apenas entre os pinos as funções de destino, origem e auxiliar.

Algoritmo 8 Solução do problema Torre de Hanói com 3 pinos e n discos

```
1: Função HANÓI(n, A, B, C)
2:     Se n > 0 então
3:         HANÓI(n − 1, A, C, B)
4:         A → B    ▷ Movimentação de A para B
5:         HANÓI(n − 1, C, B, A)
```

A função HANÓI no Algoritmo-8 implementa o procedimento descrito anteriormente. O número de discos no pino origem é repassado por n e os três argumentos A, B e C denotam respectivamente os pinos origem, destino e auxiliar. Nas linhas 3 e 5 ocorrem as chamadas recursivas que respectivamente movimentam os $n-1$ discos de A para C usando B como auxiliar e os $n-1$ discos que chegaram a C para B usando A como auxiliar. Na linha-4 ocorrem as movimentações propriamente ditas. A condição de parada ocorre na linha-2 permitindo recorrências somente quando houverem discos a serem movidos, ou seja, $n > 0$.

Matematicamente o problema Torre de Hanói com 3 pinos e n discos pode ser representado pela seguinte lei de recorrência,

$$T(n) = \begin{cases} 2 \cdot T(n-1) + 1 & n > 0 \\ 0 & n = 0 \end{cases} \tag{18}$$

onde $T(n)$ representa o número de movimentações envolvendo n discos. A expressão $2 \cdot T(n-1)$ contabiliza os dois conjuntos de movimentações dos $n-1$ discos, o primeiro entre A e C e o segundo entre C e B. O "$+1$" adicional representa a movimentação que sucede entre estes dois conjuntos de movimentações. A segunda parte da equação-18, que representa a condição de parada da recorrência, é trivial.

Efetuando-se a expansão da equação-(18) temos,

$$\begin{aligned} T(n) &= 2\cdot T(n-1)+1 \\ &= 2\cdot[2\cdot T(n-2)+1]+1 \\ &\Rightarrow 2^2\cdot T(n-2)+2+1 \\ &= 2^2\cdot[2\cdot T(n-3)+1]+2+1 \\ &\Rightarrow 2^3\cdot T(n-3)+2^2+2+1 \\ &\ldots \\ &= 2^k\cdot T(n-k)+\sum_{i=0}^{k-1}2^i \end{aligned} \quad (19)$$

Fazendo $n-k=0$ temos que $k=n$ cuja substituição na equação-(19) a torna,

$$T(n)=2^n\cdot T(0)+\sum_{i=0}^{n-1}2^i$$

utilizando nesta equação a restrição $T(0)=0$ da equação-(18) obtém-se,

$$T(n)=\sum_{i=0}^{n-1}2^i=2^n-1 \quad (20)$$

De fato para $n=3$, $T(3)=2^3-1=7$ como verificado na figura-8.

6 Exercícios

Determine o valor das somatórias a seguir,

1. $\sum_{i=1}^{n} i$
2. $\sum_{i=1}^{n} a^i$
3. $\sum_{i=1}^{n} ia^i$
4. $\sum_{i=1}^{k} 2^{k-i}i^2$
5. $\sum_{i=0}^{n} \binom{n}{i}$
6. $\sum_{i=1}^{n} i\binom{n}{i}$
7. $\sum_{i=m}^{n} a_i-a_{i-1}$
8. $1+\frac{1}{7}+\frac{1}{49}+\ldots\frac{1}{7^n}$
9. $\sum_{i=1}^{n}\log_2 i$
10. $\sum_{i=1}^{n} i2^{-i}$

Resolva as leis de recorrência a seguir,

11. $T(n)=\begin{cases} T(n-1)+c & n>1 \\ 0 & n=1 \end{cases}$
12. $T(n)=\begin{cases} T(n-1)+n-1 & n>1 \\ 0 & n=1 \end{cases}$
13. $T(n)=\begin{cases} 2T(n/2)+n-1 & n>1 \\ 0 & n=1 \end{cases}$
14. $T(n)=\begin{cases} T(n-1)+2^n & n>0 \\ 1 & n=0 \end{cases}$
15. $T(n)=\begin{cases} cT(n-1) & n>0 \\ k & n=0 \end{cases}$
16. $T(n)=\begin{cases} 3T(n/2)+n & n>1 \\ 1 & n=1 \end{cases}$
17. $T(n)=\begin{cases} 3T(n-1)-2T(n-2) & n>1 \\ 1 & n=1 \\ 0 & n=0 \end{cases}$
18. $T(n)=\begin{cases} 1+T(n-1)+T(n-2) & n>1 \\ 1 & n=1 \\ 0 & n=0 \end{cases}$

Transforme as leis de recorrência a seguir em algoritmos,

19. $T(n)=\begin{cases} n\cdot T(2n-1) & n>0 \\ 1 & n=0 \end{cases}$
20. $T(n)=\begin{cases} 3T(n-1)+n & n>0 \\ 0 & n=0 \end{cases}$

21. $T(a,b) = \left[T(a, \lfloor \frac{b}{2} \rfloor)\right]^2 \cdot \begin{cases} 1 & \text{se b é par} \\ a & \text{se b é ímpar} \end{cases}$

Escreva versões iterativas de algoritmos cujas as saídas são descritas a seguir,

22. Fatorial de um número natural.

23. n-ésimo termo da série de Fibonacci.

24. Máximo Divisor Comum (mdc) de dois números inteiros positivos.

Dado um vetor de entrada, U, de números inteiros, construir algoritmos recursivos a seguir,

25. Que imprime as chaves de U em sua ordem de armazenamento.

26. Que imprime as chaves de U na ordem inversa a de armazenamento.

27. Que determine se U é ou não um palíndromo (uma sequência palíndroma é aquela cuja ordem de elementos é a mesma que a sequência inversa de elementos, por exemplo, 12321).

28. Que determina a soma dos quadrados dos elementos de U.

29. Que determina da soma de elementos ímpares subtraída da soma de elementos pares de U.

30. Que determina o mdc de todos os valores de U.

31. Que determina o m-ésimo maior valor de U.

Seja o problema Torre de Hanói com n discos e 4 pinos, A, B, C e D, onde A é origem, B é destino e C e D auxiliares. Determine,

32. Um algoritmo de resolução que realize menos movimentos que o caso de 3 pinos.

33. Uma função C que implemente o algoritmo da questão-32.

34. A lei de recorrência que descreve o número de movimentações.

35. A resolução da lei de recorrência determinada na questão-34.

Resolver os problemas a seguir,

36. Escrever uma versão recursiva do algoritmo que imprime a versão binária de um número natural n dado como entrada. Não utilizar laços.

37. Um número inteiro é divisível por 7 se o dobro do último algarismo, subtraído do número sem o último algarismo, resultar um número divisível por 7. Utilizando esta propriedade, escreva algoritmo que testa se um número natural é divisível por 7.

38. Escrever um algoritmo recursivo que calcule a somatória dos dígitos de um número natural n dado como entrada. Não use vetores nem laços.

39. O reverso de um número natural é o número obtido pela inversão de sequência de seus dígitos. Por exemplo, o reverso de 45987 é 78954. Escrever algoritmo recursivo que receba um valor n e determine o seu reverso. Não use vetores nem laços.

40. Para calcular a raiz n-ésima de um número A de entrada, ou seja, $\sqrt[n]{A}$, é proposto a seguir um algoritmo. Estipula-se inicialmente um valor x_0 para a raiz (sugerimos $A/2$). Utilizando-se a equação,

$$x_{k+1} = \frac{1}{n}\left[x_k(n-1) - \frac{A}{x_k}\right]$$

com $k \in \mathbb{N}$, determina-se x_1 (note que quando $k = 0$ tem-se x_0 - já conhecido - no lado direito da equação e x_1 do lado esquerdo - a determinar). A partir de x_1, e utilizando a mesma equação, estima-se x_2 e assim por diante (x_3, x_4, ...). Quando for atingido um valor x_i tal que $|x_i - x_{i-1}| < tol$ (tol é definido como a *tolerância* e equivale numericamente a um valor positivo próximo de zero, por exemplo, 10^{-5}) então x_i representará o valor procurado. Escrever algoritmo recursivo que tem como entrada A e n e como saída $\sqrt[n]{A}$.

41. Um vetor de frequências de um dado vetor M é o vetor F de mesmo comprimento que M e que contém na k-ésima posição o número de vezes que a chave $M[k]$ ocorre em M. Por exemplo, se $M = \{1, 3, 1, 1, 3, 5\}$ então $F = \{3, 2, 3, 3, 2, 1\}$. Construa função recursiva que receba M e gere o F correspondente.

42. Seja A uma matriz quadrada de ordem n. O *menor complementar*, M_{ij}, de um elemento a_{ij} da matriz A é definido como o determinante da matriz quadrada de ordem $n-1$ obtida a partir da matriz A excluindo-se a linha i e a coluna j. O *co-fator*, α_{ij}, de A é definido como,
$$\alpha_{ij} = (-1)^{i+j} \cdot M_{ij}$$
O determinante de A pode ser calculado usando-se os co-fatores de uma linha arbitrária, k, conforme equação recorrente,
$$det(A) = det\left(\sum_{j=1}^{n} \alpha_{kj}\right)$$
Construir algoritmo para calcular o determinante de uma matriz A de ordem n.