Recursividade

1

Conceito

- A recursividade em programação ocorre quando uma rotina chama a si mesma em alguma parte de sua lógica
  - A rotina recorre a si mesma em algum momento
- Recursividade é um dos pilares da Teoria da Computação e do pensamento matemático e é suportada por praticamente todas as linguagens de programação importantes

2

Fatorial: versão iterativa

- O fatorial de um inteiro *N* é visto como uma repetição de *N*-1 multiplicações

```
long long int fatorial (unsigned char N)
{   long long int resp = 1;
    unsigned char c;

    if (N > 1)
          for (resp = N, c = N - 1; c > 0; c--)
                resp = resp * c;

    return resp;
}
```


3

Fatorial: versão iterativa

- Uma mesma quantidade de memória é alocada para calcular o fatorial de números naturais menores que 21
  - Naturais superiores a 20 excedem a capacidade do tipo `long long int` da linguagem C
- Cada passagem do *looping* executa uma das *N*-1 multiplicações requeridas pelo cálculo

4

Fatorial: versão recursiva

- Parte da constatação de uma identidade:

5! = 5 * 4 * 3 * 2 * 1 = 5 * 4!
4! = 4 * 3 * 2 * 1 = 4 * 3!
3! = 3 * 2 * 1 = 3 * 2!
2! = 2 * 1 = 2 * 1!
1! = 1 = 1 * 0!
0! = 1 (por definição)

$$N! \begin{cases} 1, \text{ se } N = 0 \text{ (caso base)} \\ N * (N - 1)!, \text{ se } N > 0 \end{cases}$$

5

Fatorial: versão recursiva

```
long long int fatorial(unsigned char N)
{   if (N == 0)
          return 1;
    else
          return N * fatorial(N - 1);
}
```

$$N! \begin{cases} 1, \text{ se } N = 0 \text{ (caso base)} \\ N * (N - 1)!, \text{ se } N > 0 \end{cases}$$

6

Fatorial: versão recursiva

```
fatorial( 5 )
5 * fatorial( 4 )
5 * (4 * fatorial( 3 ) )
5 * (4 * (3 * fatorial( 2 ) ) )
5 * (4 * (3 * (2 * fatorial( 1 ) ) ) )
5 * (4 * (3 * (2 * (1 * fatorial( 0 ) ) ) ) )
5 * (4 * (3 * (2 * (1 * 1) ) ) )
5 * (4 * (3 * (2 * 1) ) )
5 * (4 * (3 * 2) )
5 * (4 * 6)
5 * 24
120
```


7

Fatorial: versão recursiva

- Cada chamada da função aloca uma pequena quantidade de memória, mas para calcular o fatorial de *N*, precisamos de *N*+1 chamadas da função
  - A memória utilizada cresce conforme aumenta o valor de *N*
  - Cada chamada produz um termo da sequência de multiplicações requeridas pelo cálculo

8

MDC

```
int mdc(int n, int m)
{   int r;
    if (n < 0) n = -n;
    if (m < 0) m = -m;

    while (n % m != 0)
    {  r = n % m;
       n = m;
       m = r;
    }

    return m;
}
```

```
int mdc(int n, int m)
{   if (n < 0) n = -n;
    if (m < 0) m = -m;

    if (n % m == 0)
       return m;
    else
       return mdc(m, n % m);
}
```


9

Torre de Hanoi

- É um problema clássico do emprego de raciocínio recursivo
- Foi proposto no Ocidente pela primeira vez em 1883 pelo matemático francês Édouard Lucas
- A proposição original do problema faz referência a uma suposta lenda hindu, embora Hanoi seja uma cidade vietnamita
  - Alguns chamam o problema de Torres de Brahma
  - Existem inúmeras variações do enredo e do problema

10

Torre de Hanoi

- O templo hindu de Kashi Vishwanath possui uma ampla sala onde estão dispostos 3 pinos. Existem 64 discos de ouro, furados no meio, cada um com um diâmetro diferente dos demais. Inicialmente esses discos estavam colocados todos encaixados em um dos pinos, em ordem de seus diâmetros, com os discos maiores embaixo dos discos menores. Sacerdotes brâmanes, obedecendo a uma antiga profecia, passam os dias movendo esses discos, com o objetivo de transferi-los todos para um determinado pino, devendo seguir um pequeno conjunto de regras imutáveis.

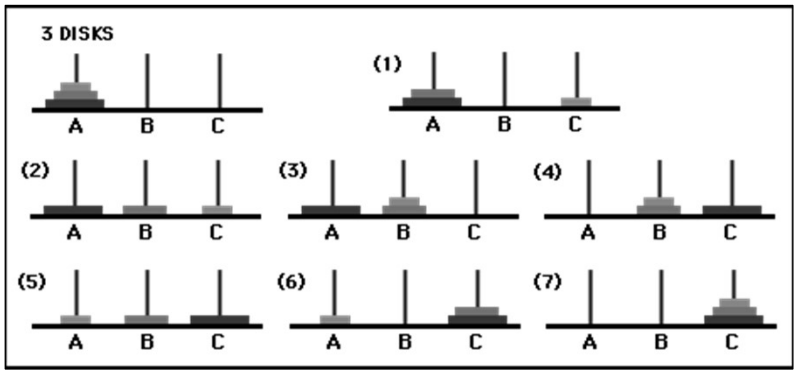
11

Torre de Hanoi

- Regras:
  - Só se pode mover um disco de cada vez
  - Pode-se usar os 3 pinos para fazer os movimentos
    - Um é o pino de origem
    - Um é o pino de destino
    - O pino restante é o pino auxiliar
  - Não é permitido em momento algum colocar um disco maior sobre um disco menor
- Deseja-se saber o menor número de movimentos para mover todos os discos para o pino de destino

12

Idéia básica



Com esses 3 discos, foram necessários 7 movimentos

13

Alguns resultados ...

Qtde de discos	Qtde de movimentos
1	1
2	3
3	7
4	15
5	31
6	63

Qual é o número mínimo de movimentos se tivermos 7 discos?

Resposta certa: 127

A quantidade mínima é dada por $2^N - 1$, onde N indica quantos discos existem

14

Ainda o problema original

- Supondo então que cada sacerdote do templo realize um movimento por segundo, serão necessários $2^{64} - 1$ movimentos, ou seja, 18.446.744.073.709.551.615 segundos (já que temos um movimento por segundo) o que corresponde a cerca de 585 bilhões de anos
  - Ainda bem, pois segundo a lenda, quando todos os discos estiverem em seu destino final, o mundo acabará

15

Recursividade das Torres

- Para mover um determinado disco para o seu destino, primeiro precisamos mover todos os que estão acima dele para o pino auxiliar
  - Para isso usamos o pino de destino final como pino auxiliar provisório
- Com o disco desejado livre, podemos move-lo para o seu destino
- Encerramos movendo os discos do pino auxiliar para o destino final
  - E para isso usamos o pino de origem como pino auxiliar

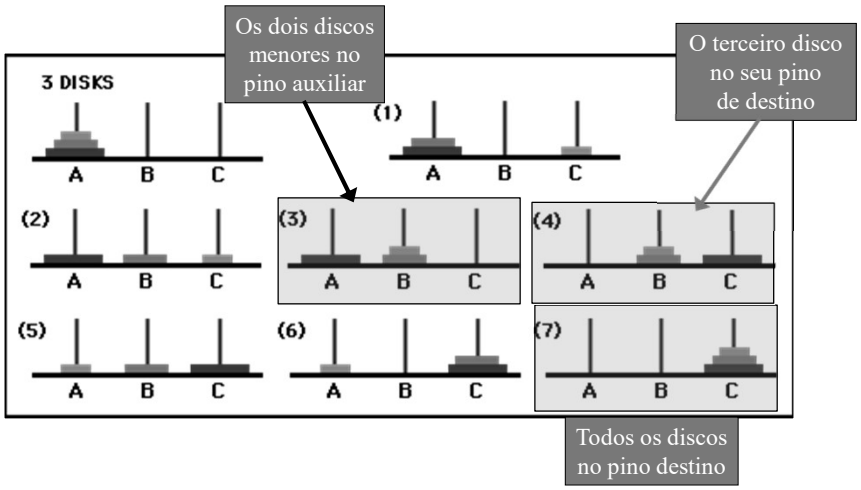
16

Recursividade das Torres

- Com 3 discos:
  - Precisamos mover o terceiro e último disco para o destino
  - Então, primeiro, movemos os dois discos acima dele para o pino auxiliar
  - Movemos o terceiro disco para o seu destino
  - Movemos os dois discos que estão no pino auxiliar para o pino de destino

17

Idéia básica



18

Recursividade das Torres

```
procedimento Hanoi(N, Orig, Dest, Temp)
  se N = 1 então
    mover o menor disco do pino Orig para o pino Dest;
  senão
    Hanoi(N-1, Orig, Temp, Dest);
    mover o N-ésimo menor disco do pino Orig para o pino Dest;
    Hanoi(N-1, Temp, Dest, Orig);
  fim-se
fim
```


Complementos

- Toda rotina recursiva possui uma versão iterativa equivalente
  - A chamada da rotina a si mesma é substituída por uma estrutura de repetição
  - Existem linguagens, principalmente do tipo funcional, que não possuem *loopings* e então a recursividade é usada para suprir essa lacuna

Complementos

- Para linguagens imperativas a recursividade é apenas uma opção
  - Soluções recursivas para problemas recursivos são simples e elegantes, e permitem que o código evolua de maneira natural
  - Soluções iterativas para problemas recursivos costumam ser extensas, trabalhosas e muitas vezes são assimétricas, criando dificuldades para a adaptação do código a novos requisitos