Estrutura de Dados e Algoritmos

Recursividade

Prof. Nilton Correia da Silva

Recursividade

- Uma função recursiva é uma função que resolve uma parte pequena de um problema, e que, para resolver o resto do problema, chama ela mesma.
- O conceito é matemático mas se aplica muito bem à programação. Um exemplo vindo da matemática é a definição dos números Naturais (inteiros positivos):
  - 0 é um inteiro natural
  - Se x é um inteiro natural, x+1 é um inteiro natural

Recursividade

- Um exemplo fácil de programar é o fatorial. A definição matemática é:
  - Fatorial(x) = produto dos inteiros de 1 a x
  - Por exemplo, Fatorial(5) = 1 * 2 * 3 * 4 * 5.
- Em outras palavras:
  - Se x = 1, Fatorial(x) = 1
  - Senão, Fatorial(x) = x * Fatorial(x-1)

Recursividade

- Quando uma função chama ela mesma, um novo espaço é reservado na pilha para a execução da nova função, e o ponteiro do programa, que indica qual linha do programa está sendo executado, pula para o início da função.
- O código executado é o mesmo, mas com um conjunto de variáveis e parâmetros novos, num espaço novo na pilha.
- Na hora do return, a função é desempilhada e volta para onde estava, descartando suas variáveis e parâmetros.
- O programa continua onde tinha parado com suas variáveis e parâmetros antigos.

Recursividade

- Partes importantes de uma função recursiva:

```
int Fatorial(int x)
{
   if (x == 1)
   {
      return 1;
   }
   else
   {
      return x * Fatorial(x-1);
   }
}
```

Condição de parada das chamadas recursivas.

Chamada a ela mesma

- Cuidado com Recursões Infinitas:

```
int Fatorial(int x)
{
   if (x == 1)
   {
      return 1;
   }
   else
   {
      return x * Fatorial(x);
   }
}
```

```
int LoopInfinito(int x)
{
   return LoopInfinito(x);
}
```

Recursividade

- Solução Recursiva & Não Recursiva:

```
int Fatorial(int x)
{
   if (x == 1)
   {
      return 1;
   }
   else
   {
      return x * Fatorial(x-1);
   }
}
```

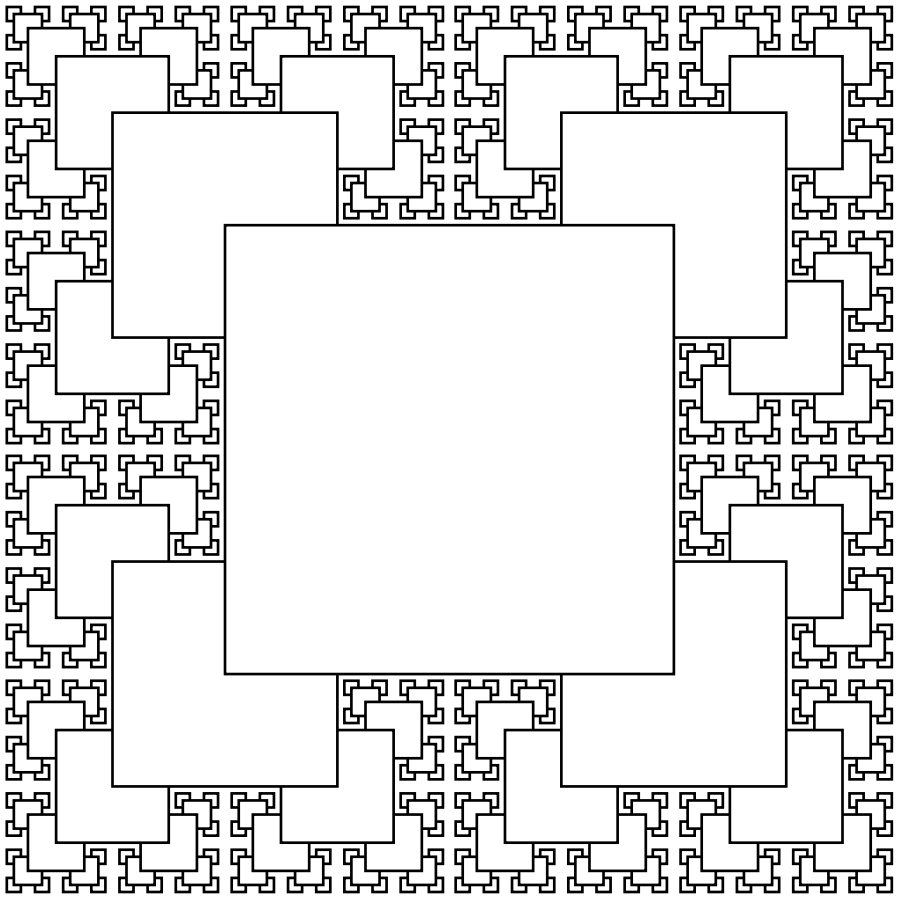
```
int Fat (int n)
{
  int f;
  f = 1;
  while(n > 0)
  {
    f = f * n;
    n = n – 1;
  }
  return f;
}
```

Recursividade

- A recursividade nem sempre é a melhor solução, mesmo quando a definição matemática do problema é feita em termos recursivos;
- Recursividade vale a pena para Algoritmos complexos, cuja a implementação iterativa é complexa e normalmente requer o uso explícito de uma pilha. Exemplos:
  - Dividir para Conquistar (Ex. Quicksort)
  - Caminhamento em Árvores (pesquisa, backtracking)

Recursividade

- Outro Exemplo:

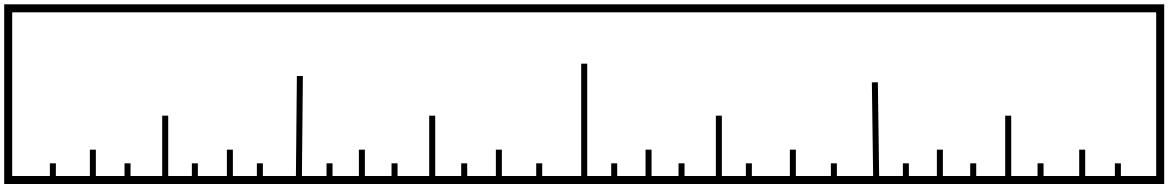


```
void estrela(int x,int y, int r)
{
  if ( r > 0 )
  {
    estrela(x-r, y+r, r div 2);
    estrela(x+r, y+r, r div 2);
    estrela(x-r, y-r, r div 2);
    estrela(x+r, y-r, r div 2);
    box(x, y, r);
  }
}
```

Recursividade

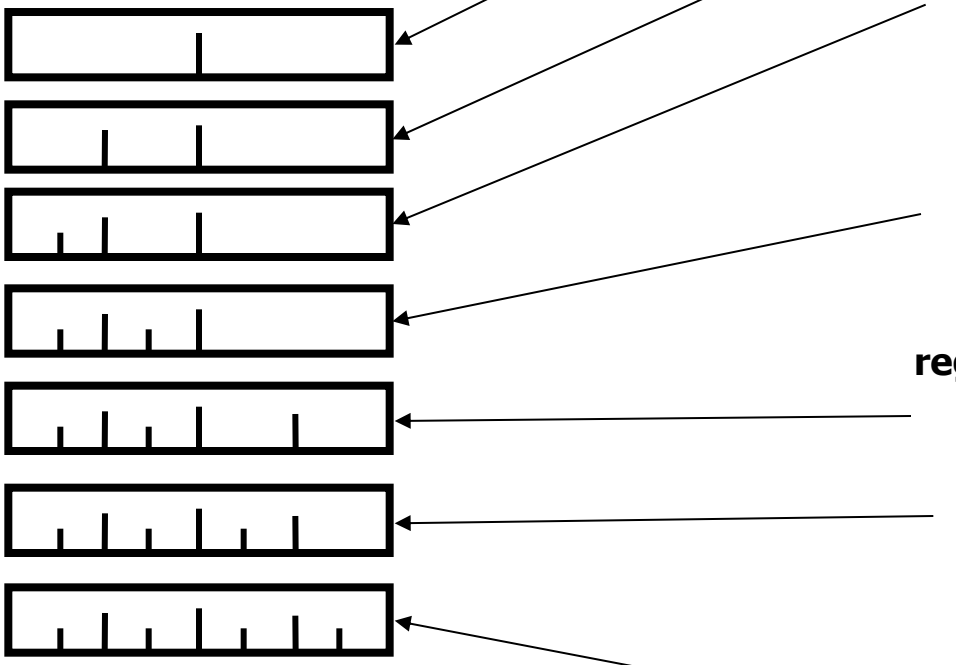
- Outro Exemplo:

```
int regua(int l,int r,int h)
{
  int m;
  if ( h > 0 )
  {
    m = (l + r) / 2;
    marca(m, h);
    regua(l, m, h - 1);
    regua(m, r, h - 1);
  }
}
```



Recursividade

- Outro Exemplo:



```
regua(0, 8, 3)
  marca(4, 3)
  regua(0, 4, 2)
    marca(2, 2)
    regua(0, 2, 1)
      marca(1, 1)
      regua(0, 1, 0)
      regua(1, 2, 0)
    regua(2, 4, 1)
      marca(3, 1)
      regua(2, 3, 0)
      regua(3, 4, 0)
regua(4, 8, 2)
    marca(6, 2)
    regua(4, 6, 1)
      marca(5, 1)
      regua(4, 5, 0)
      regua(5, 6, 0)
    regua(6, 8, 1)
      marca(7, 1)
      regua(6, 7, 0)
      regua(7, 8, 0)
```

Recursividade

- Outro Exemplo: Crie uma função recursiva que calcula a potência de um número

```
int pot(int base, int exp)
{
  if (!exp)
    return 1;

  /* else */
  return (base*pot(base, exp-1));
}
```

O(n)