Estrutura de Dados e Algoritmos

Recursividade

Prof. Nilton Correia da Silva

Recursividade

"Uma jornada de 1000 quilômetros começa com um simples passo."

- O que este provérbio tem a ver com ciência da computação?
  - Mais comum: Há de se iniciar.
  - Mais fundamental: O primeiro passo é apenas o primeiro de muitos. A jornada como um todo é feita de passos. Depois do primeiro, o destino estará mais próximo. Todo caminho remanescente é repetição do que foi feito para se alcançar o primeiro passo.

Recursividade

"Uma jornada de 1000 quilômetros começa com um simples passo."

- Este provérbio é a base de uma importante técnica de solução de problemas em ciência da computação: A recursividade.
- A recursividade nos permite ter algoritmos curtos para grandes problemas.

Recursividade

- Supondo que um passo tenha em média 40cm, poderíamos reescrever o provérbio:

Uma jornada de 400.000 passos começa com um simples passo.

- No que ficam faltando apenas outros 399.999 passos.

Recursividade

- Poderíamos então fazer a seguinte reescrita do provérbio:

Uma jornada de 400.000 passos começa com um simples passo, restando apenas outros 399.999 para chegar.

- Aparentemente não houve nenhuma melhora, mas esta reescrita é importante.

Uma jornada de 399.999 passos começa com um simples passo, restando apenas outros 399.998 para chegar.

- Este segundo problema possui a mesma forma do primeiro. É apenas um pouco menor.

Uma jornada de 1 passo começa com um simples passo, restando apenas outros 0 para chegar.

- Se não há mais nenhum passo para chegar, é sinal que chegou-se.

Uma jornada de 0 passo não é uma jornada.

Recursividade

- Em um problema finito, podemos continuamente ir reduzindo o tamanho do problema até que se chegue ao objetivo final:

Uma jornada de [qualquer número de passos] começa com um simples passo, seguido por uma jornada que requer um passo a menos.

Recursividade

- Infelizmente nossa última reescrita ainda não se configura como uma instrução no senso computacional. É apenas um roteiro. Todavia ele pode ser traduzido um par de instruções:

  Dê um passo.

  Dê os outros passos.

Recursividade

- Ainda não ficou explícito que os passos remanescentes devem ser dados da mesma forma que foi dado o primeiro:

Dê um passo.

Dê os outros passos da mesma forma.

- Isto pode ser formalizado nomeando-se a forma de se dar o passo:

Algoritmo FaçaUmaJornada

Dê um passo.

FaçaUmaJornada – Com um passo a menos

Recursividade

- Falta ainda formalizar a ideia de chegada. Pois se passarmos do destino não faremos a jornada correta:

Algoritmo FaçaUmaJornada

se ainda não chegou-se ao destino

Dê um passo.

FaçaUmaJornada – Com um passo a menos

Recursividade

- O algoritmo define o que fazer em duas possíveis situações: 1. ainda não chegou-se no destino ou 2. chegou-se no destino. Contudo ele não especifica como dar um passo ou como reconhecer que chegou-se ao destino.

Algoritmo FaçaUmaJornada

se ainda não chegou-se ao destino

Dê um passo.

FaçaUmaJornada – Com um passo a menos

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);
   }
}
```

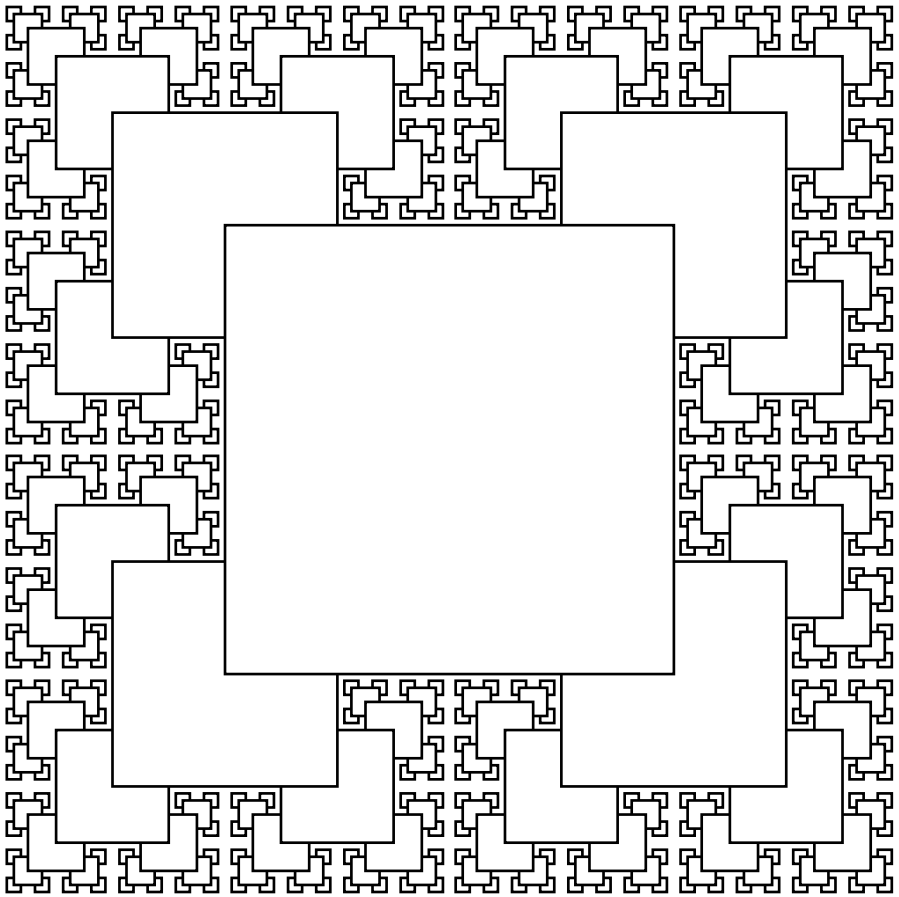
```
int Fat (int n)
{
  int f;
  f = 1;
  while(n > 0)
  {
    f = f * n;
    n = n – 1;
  }
  return f;
}
```

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:

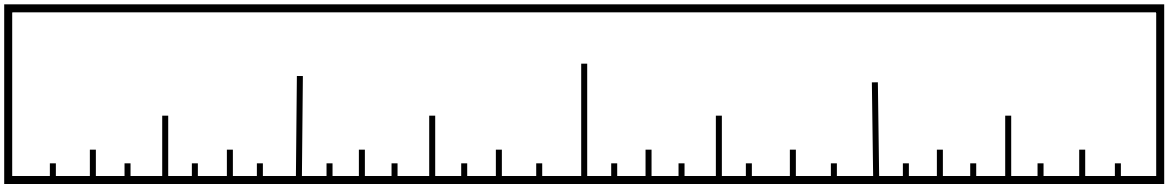


```
void estrela(int x,int y, int r)
{
  if ( r > 0 )
  {
    estrela(x-r, y+r, r div 2);
    estrela(x+r, y+r, r div 2);
    estrela(x-r, y-r, r div 2);
    estrela(x+r, y-r, r div 2);
    box(x, y, r);
  }
}
```

Recursividade

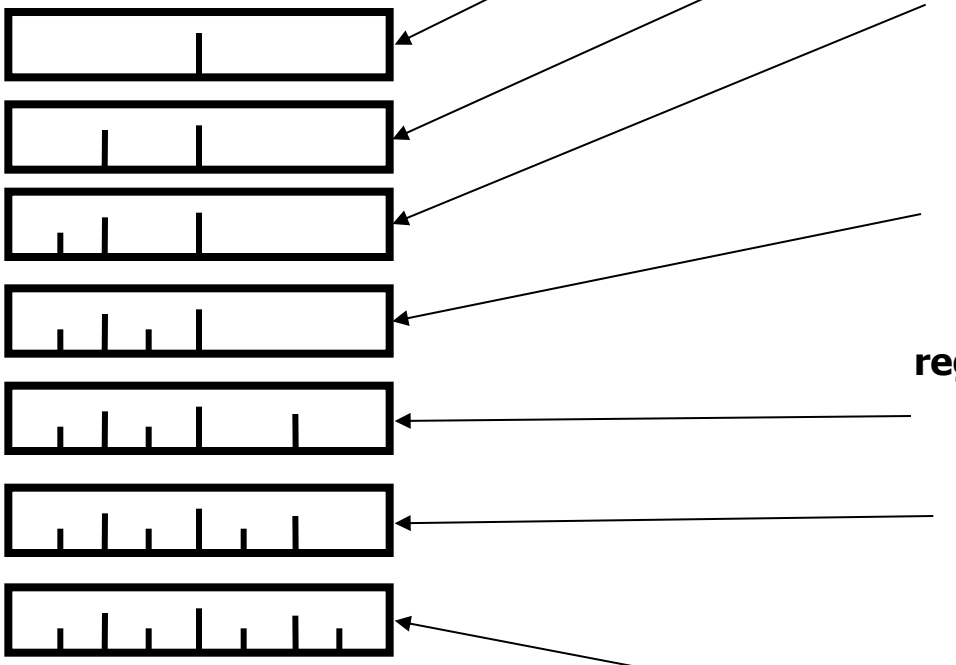
- Outro Exemplo:

```
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)

Exercícios...

- **Implemente uma função recursiva para computar o valor de 2^n.**
- **Implemente uma função recursiva soma(n) que calcula o somatório dos n primeiros números inteiros. Qual é a ordem de complexidade da sua função? Qual seria a ordem de complexidade dessa mesma função implementada sem utilizar recursividade? O que você conclui?**

Exercícios...

- Dada a função X:

```
int X(int n, int m)
{
   if ((n==m) || (n==0))
     return 1;
   else
      return (x(n-1,m)+x(n-1,m+1));
}
```

a) qual o valor de x(5,3) ?
b) quantas chamadas serao feitas na avaliação acima ?

- Dada a função abaixo:

```
int X(int N)
 {
   if (n >= 0) && (n <= 2)
     return (n);
   else
     return (x(n-1)+x(n-2)+x(n-3));
 }
```

a) quantas chamadas serao executadas para avaliar x(6) ?
b) indique a sequencia temporal destas chamadas.