Recursão

Aula 01

Ivone P. Matsuno Yugoshi
ivone.matsuno@ufms.br

Ronaldo Fiorilo dos Santos
ronaldo.santos@ufms.com

Universidade Federal de Mato Grosso do Sul
Câmpus de Três Lagoas
Bacharelado em Sistemas de Informação

Algoritmos e Programação II

Definição

- Recursão é um conceito fundamental em computação;
- Sua compreensão nos permite construir algoritmos elegantes, curtos e poderosos;
- Em muitos problemas computacionais encontramos a seguinte propriedade: **cada entrada do problema contém uma entrada menor do mesmo problema** (estrutura recursiva);
- Uma função recursiva é aquela que possui uma ou mais chamadas a si mesma (chamada recursiva)

Definição

- Para resolver um problema como esse, usamos em geral a seguinte estratégia:

  se a entrada do problema é pequena então
  resolva-a diretamente;
  senão,
  reduza-a a uma entrada menor do mesmo problema,
  aplique este método à entrada menor
  e volte à entrada original.

- A estratégia acima é dividida em duas partes:
  - parte base (se); e
  - parte recursiva (senão).

Definição

- A parte base possui o critério de parada da recursividade, ou seja, ela garante que o módulo não ficará se chamando infinitamente;
  - Devido sua importância, o critério de parada deve ser a primeira coisa a se providenciar quando está se pensando em um módulo recursivo.
- A parte recursiva é aquela onde se encontram uma ou mais chamadas ao próprio módulo.
  - Todo módulo deve possuir ao menos uma chamada externa a ele. Se todas as chamadas ao módulo são externas, então o módulo é dito iterativo ou não-recursivo.

Exemplos

Problema

Dado um número inteiro $n \geq 0$, computar o fatorial $n!$.

Usamos uma fórmula que nos permite naturalmente escrever uma função recursiva para calcular $n!$:

$$n! = \begin{cases} 1\,, & \text{se } n \leq 1\,, \\ n \times (n-1)!\,, & \text{caso contrário}\,. \end{cases}$$

Exemplos

Uma solução.

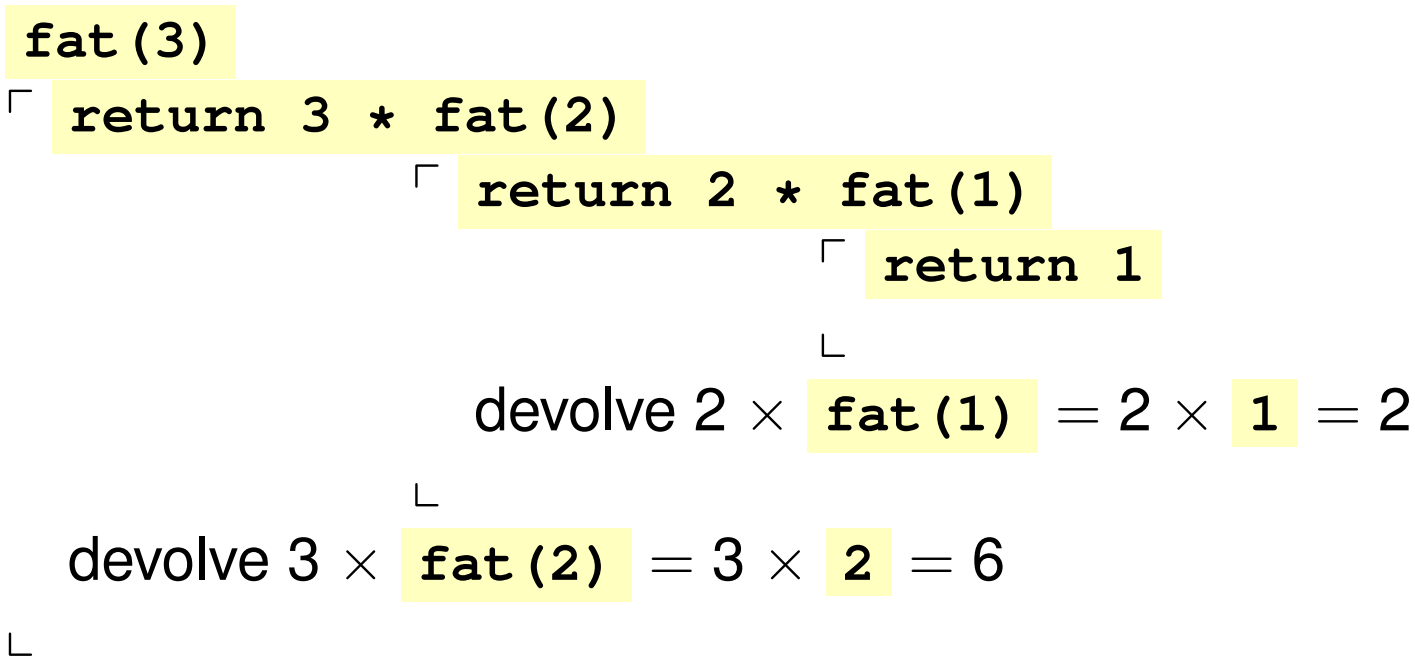
```
/* Recebe um número inteiro n >= 0 e devolve o fatorial de n */
int fat(int n)
{
   int result;

   if (n <= 1)
      result = 1;
   else
      result = n * fat(n-1);

   return result;
}
```

Exemplos

Outra solução.

```
/* Recebe um número inteiro n >= 0 e devolve o fatorial de n */
int fat(int n)
{
   if (n <= 1)
      return 1;
   else
      return n * fat(n-1);
}
```

Exemplos


fat(3)
return 3 * fat(2)
return 2 * fat(1)
return 1
devolve 2 × fat(1) = 2 × 1 = 2
devolve 3 × fat(2) = 3 × 2 = 6

Exemplos

Problema

Dado um número inteiro $n > 0$ e uma sequência de n números inteiros armazenados em um vetor v, determinar um valor máximo em v.

Exemplos

```
/* Recebe um número inteiro n > 0 e um vetor v de números in-
   teiros com n elementos e devolve um elemento máximo de v */
int maximo(int n, int v[MAX])
{
   int aux;

   if (n == 1)
      return v[0];
   else {
      aux = maximo(n-1, v);
      if (aux > v[n-1])
         return aux;
      else
         return v[n-1];
   }
}
```

Definição

- Em geral, todo módulo recursivo corresponde a um outro iterativo que executa exatamente a mesma computação.
- Muitas vezes o equivalente iterativo é muito mais difícil de escrever, ler e compreender que a solução recursiva, devido em especial à estrutura recursiva intrínseca do problema.
- Uma solução recursiva tem como principal desvantagem maior consumo de memória, já que durante seu processo de execução muitas informações devem ser guardadas na sua pilha de execução.

Correção

Como verificar que uma função recursiva está correta?

- Passo 1: escreva *o que* a função deve fazer;
- Passo 2: verifique se a função de fato faz o que deveria fazer quando a entrada é pequena;
- Passo 3: imagine que a entrada é grande e suponha que a função fará a coisa certa para entradas menores; sob essa hipótese, verifique que a função faz o que dela se espera.

Correção

Proposição

A função `maximo` encontra um maior elemento em um vetor v com $n \geq 1$ números inteiros.

Correção

Prova

Indução na quantidade n de elementos do vetor v.
Se $n = 1$ é fácil.
Suponha que para qualquer valor inteiro positivo $m < n$ a função compute corretamente `maximo(m, v)`.
Suponha agora que temos um vetor v contendo $n > 1$ números inteiros.
Chamada externa `maximo(n, v)`, com $n > 1$. A função executa:

```
aux = maximo(n-1, v);
```

Por hipótese de indução, `aux` contém um valor máximo para os $n - 1$ primeiros valores do vetor v. Então, a função decide quem é maior: `aux` ou `v[n-1]`.

Exercícios

1. O que faz a função abaixo?

```
void imprime_alguma_coisa(int n)
{
   if (n != 0) {
      imprime_alguma_coisa(n / 2);
      printf("%c", '0' + n % 2);
   }
}
```

2. (a) Escreva uma função iterativa com a seguinte interface:

   `int pot(int x, int n)`

   que receba dois números inteiros x e n e calcule e devolva x^n.

   (b) Escreva uma função recursiva com a seguinte interface:

   `int potR(int x, int n)`

   que receba dois números inteiros x e n e calcule e devolva x^n.

Exercícios

3. A **sequência de Fibonacci** é uma sequência de números inteiros positivos dada pela seguinte fórmula:

$$\begin{cases} F_1 &= 1\,, \\ F_2 &= 1\,, \\ F_i &= F_{i-1} + F_{i-2}\,, \quad \text{para } i \geq 3. \end{cases}$$

   Escreva uma função recursiva com a seguinte interface:

   ```
   int Fib(int i)
   ```

   que receba um número inteiro positivo i e devolva o i-ésimo termo da sequência de Fibonacci, isto é, F_i.

4. Escreva uma função recursiva com a seguinte interface:

   ```
   int soma_digitos(int n)
   ```

   que receba um número inteiro positivo n e devolva a soma de seus dígitos.

Desafio

- A **torre de Hanói** é um quebra-cabeça que consiste em uma base contendo três pinos, em um dos quais são dispostos *n* discos uns sobre os outros, em ordem crescente de diâmetro, de cima para baixo.

- O problema consiste em passar todos os discos de um pino para outro qualquer, usando um dos pinos como auxiliar, de maneira que um disco maior nunca fique em cima de outro menor em nenhuma situação.
- Escreva um algoritmo que, dado a quantidade *n* de discos, liste quais os movimentos devem ser feitos para mover os *n* discos do pino A para o pino C.