Departamento de Sistemas de Computação
Universidade de São Paulo
SSC 501 – Introdução a Ciência de Computação I

Recursão II
Continuação

Seiji Isotani, Rafaela V. Rocha

sisotani@icmc.usp.br

rafaela.vilela@gmail.com

PAE: Armando M. Toda, Geiser Chalco

armando.toda@gmail.com

geiser.gcc@gmail.com

Recursão

- ✓ Em computação temos recursividade quando um dos passos de um algoritmo é repetir o **mesmo algoritmo** para uma parte menor do problema

- ✓ 3 passos para definir a função recursiva:
- Verificar qual a condição de parada (caso base) onde o algoritmo termina.
- Pensar em como dividir o problema em problemas menores do mesmo tipo.
- Encontrar uma maneira de chamar recursivamente a função passando para ela um parâmetro do problema menor.

Sequência de Fibonacci

0 1 2 3

0,1, **1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, ...**

- Versão Recursiva

```
int fibonacciR(int n)
{
   if(n < 2) //0, 1
   {
      return n;
   }
   else
   {
      return fibonacciR(n-1) + fibonacciR(n-2);
   }
}
```

condição de parada

problemas menores

chamadas recursivas

Recursividade

- Um programa recursivo é **mais elegante** e **menor** que a sua versão iterativa, além de exibir com maior clareza o processo utilizado, desde que o problema ou os dados sejam naturalmente **definidos por meio de recorrência**. Por outro lado, um programa recursivo exige **mais espaço de memória** e é, na grande maioria dos casos, **mais lento do que a versão iterativa**

Quando usar recursividade

- O problema é **naturalmente recursivo** e a versão recursiva do algoritmo não gera ineficiência evidente se comparado com a versão iterativa do mesmo algoritmo.
- O algoritmo se torna compacto sem perda de clareza ou generalidade.

Quando não usar recursividade

- A solução recursiva causa **ineficiência** se comparada com uma versão iterativa.
- **Parâmetros** consideravelmente **grandes** têm que ser passados por **valor**.

Recursividade

- Funções que recebem vetor como parâmetro.
- Tente criar a função abaixo que imprime um vetor recursivamente, sem utilizar comandos de repetição:

```
void imprimeVetorR(int v[],int ini, int fim)
```

Recursividade

```
void imprimeVetorR(int v[],int ini, int fim)
{
   if(ini == fim)
   {
      printf("%d ",v[ini]);
   }
   else
   {
      printf("%d ",v[ini]);
      imprimeVetorR(v,ini+1,fim);
   }
}
```

O que acontece se invertemos a linha do printf com a chamada recursiva?

```
#include <stdio.h>

void imprimeVetorR(int v[],int ini, int fim) {
   if(ini == fim)
   {
      printf("%d ",v[ini]);
   }
   else
   {
      printf("%d ",v[ini]);
      imprimeVetorR(v,ini+1,fim);
   }
}

int main(void) {
      int n[8] = {10,21,42,13,34,55,16,27};
      imprimeVetorR(n, 3, 7);
      return 0;
}
```

Recursividade

Encontrar o maior valor em um vetor:

```
int maiorVetorI(int v[],int tam)
{
    int i,max=v[0];
    for(i=1;i<tam;i++)
    {
        if(v[i] > max)
        {
            max = v[i];
        }
    }
    return max;
}
```

Faça a versão Recursiva.

Recursividade

```
int maiorVetorR(int v[],int tam)
{
   if(tam == 1)
          return v[0];
    else
   {
      int M = maiorVetorR(v,tam-1);
      if(M > v[tam-1])
            return M;
        else
            return v[tam-1];
   }
}
```

```
#include <stdio.h>

int maiorVetorR(int v[],int tam) {
   if(tam == 1)
          return v[0];
     else
    {
       int M = maiorVetorR(v,tam-1);
       if(M > v[tam-1])
            return M;
         else
            return v[tam-1];
     }
}

int main(void) {
      int n[5] = {10,21,42,13,34};
      int resp = maiorVetorR(n,5)
      printf("%d ", resp);
      return 0;
}
```