Estrutura de Dados II

Recursividade

(Aula02)

Prof. Rafael Nunes

Relembrando alguns
conceitos ...

O que é *computação*?

Computação

- Busca de uma *solução para um problema*, a partir de entradas, ou cálculo de uma função

- ... através de um *algoritmo*

O que é um *Algoritmo*?

Algoritmo

- Independe de computador
  - Um algoritmo não representa, necessariamente, um programa de computador, e sim os *passos necessários para realizar uma tarefa*
- ... na Computação
  - *Seqüência de Instruções* para a realização de uma tarefa ou a solução de um problema
- ... na Matemática
  - *Conjunto de processos* para efetuar um cálculo

Algoritmo

- O conceito de um algoritmo foi formalizado em 1936
  - Maquina de Turing (Alan Turing)
    - http://ironphoenix.org/tril/tm/
  - Cálculo de Lambda (Alonzo Church)
    - (+ (* 5 6) (* 8 3))

Diferentes Algoritmos podem realizar a *mesma tarefa*?

Diferentes Algoritmos podem realizar a mesma tarefa?

- Sim!

Diferentes algoritmos podem realizar a mesma tarefa usando um *conjunto diferenciado de instruções*, em mais ou menos tempo

Diferentes Algoritmos podem realizar a mesma tarefa?

Algoritmos

- A1:
  - 1º Passo: Vestir a meia
  - 2º Passo: Calçar Sapato
  - 3º Passo: Vestir a Calça
- A2:
  - 1º Passo: Vestir a Calça
  - 2º Passo: Vestir a meia
  - 3º Passo: Calçar Sapato

Recursividade!

O que é Recursão?

... ou recorrência?

... ou recursividade?

O que é Recursão?

- Recursividade, recursão ou recorrência é quando uma determinada função faz chamada a ela mesma ***(com parâmetros diferentes!).***
- O exemplo clássico é o fatorial:
  - Qual o fatorial de cinco (5!)?
  - *É cinco vezes o fatorial de quatro (5 x 4!).*
  - E qual o fatorial de quatro?
  - *É quatro vezes o fatorial de três (4 x 3!).*

Porque devo aprender Recursividade?

Porque devo aprender Recursividade?

- Porque é um método poderoso e comum de simplificação – ***ele divide o problema em subproblemas do mesmo tipo, para poder resolvê-lo***. ***(Chama a ele mesmo!)***

  – Este método é conhecido como ***dividir e conquistar*** e é a chave para a construção de muitos algoritmos importantes, bem como uma parte fundamental da ***programação dinâmica.***

Recursão

(Ciência da Computação)

Recursão

- Em termos de programação a recursividade é *uma técnica* em que uma rotina, no processo de execução de suas tarefas, chama a si mesma...
- ... traduzindo:

Nada mais é ... do que uma ***função que chama a ela mesma*** para resolver um problema.

Como uma função pode chamar a si mesma?

???

Como uma função pode chamar a si mesma?

- Para que a recursividade funcione, é preciso haver um valor base, que funcione como ***parada para a recursão***.
- Exemplo:
  - No caso do fatorial, o valor base é 1! = 1, portanto:

5! = 5 x 4! = 5 x 4 x 3! = 5 x 4 x 3 x 2! = 5 x 4 x 3 x 2 x 1!, mas ***nós sabemos*** que 1! é 1.

5! = 5 x 4 x 3 x 2 x 1 = 120

Como identificar se um problema é recursivo?

???

Como identificar se um problema é recursivo?

- Os problemas que podem ser resolvidos com recursividade têm as seguintes particularidades:
  - Um ou mais *casos de parada* do algoritmo com solução conhecida
  - Casos em que o *problema pode ser minimizado* recursivamente até atingir uma situação de parada.

Exemplo Clássico

FATORIAL DE UM NUMERO N

FATORIAL(N) - Versão Iterativa

- f(n) = 1 x 2 x 3 x ... x (n-2) x (n-1) x n

- Ou seja:
  - f(1) = 1
  - f(2) = 1 x 2 = 2
  - f(3) = 1 x 2 x 3 = 6
  - f(4) = 1 x 2 x 3 x 4 = 24
  - f(5) = 1 x 2 x 3 x 4 x 5 = 120

Implementem a versão Iterativa do Fatorial de um Numero

<< 10 minutos >>

FATORIAL(N) - Versão Iterativa

```
//Função fatorial iterativa
int fatorial (int n)
{
        int i, fat =1;

        for (i=1;i<=n;i++)
                fat = fat * i;

        return (fat);
}
```

Como resolver um problema recursivo?

Como resolver um problema recursivo?

- A solução típica para um problema recursivo é:

Função (...)

se → `caso de base de parada`

então → `resolve o problema`

senão → `divida o problema em partes menores usando uma chamada recursiva à própria função.`

FATORIAL(N)

Versão Recursiva

FATORIAL(N) - Versão Recursiva

$$\texttt{fat(N)} = \begin{cases} \texttt{1, se N = 1} \\ \texttt{N x fat(N-1), se N != 1} \end{cases}$$

$$\texttt{fat(N)} = \begin{cases} \texttt{Condição Base de Parada} \\ \texttt{Chamada Recursiva} \end{cases}$$

- Ou seja:
  - f(1) = 1
  - f(2) = 2 x f(1) = 2
  - f(3) = 3 x f(2) = 6
  - f(4) = 4 x f(3) = 24
  - f(5) = 5 x f(4) = 120

FATORIAL(N) - Versão Recursiva

```
//Função fatorial recursiva
int fat (int n)
{
        if (n==1)
                return(1);
        else
                return (n * fat(n-1) );
}
```

FATORIAL(N) - Versão Recursiva

- Veja o que acontece na memória:

4	`fat(1) = 1`
3	`fat(2) = (2*fat(1))`
2	`fat(3) = (3*fat(2))`
1	`fat(4) = (4*fat(3))`
0	`fat(5) = (5*fat(4))`

Exercício

<< 10 minutos >>

Seqüência de Fibonacci

$$\texttt{fib(N)} = \begin{cases} \texttt{0, se N = 0} \\ \texttt{1, se N = 1} \\ \texttt{fib(N-1)+fib(N-2) se N > 1} \end{cases}$$

- Exercício: Seqüência de Fibonacci
  - $Fib_0 = 0$
  - $Fib_1 = 1$
  - $Fib_n = Fib_{n-1} + Fib_{n-2}$, para n>1

- Ex: 0 1 1 2 3 5 8 13 ...