

ECT2303 - Linguagem de Programação

Funções Recursivas

Carlos Olarte.

3 de Dezembro de 2020

Motivação

Considere as seguintes funções:

$$\bullet \text{ } n! = \begin{cases} 1 & \text{se } n = 0 \\ n \times (n-1)! & \text{se } n > 0 \end{cases}$$

$$\bullet \text{ } \text{pow}(a, n) = \begin{cases} 1 & \text{se } n = 0 \\ a \times \text{pow}(a, n-1) & \text{se } n > 0 \end{cases}$$

$$\bullet \text{ } \text{fib}(n) = \begin{cases} 1 & \text{se } 1 \leq n \leq 2 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{se } n > 2 \end{cases}$$

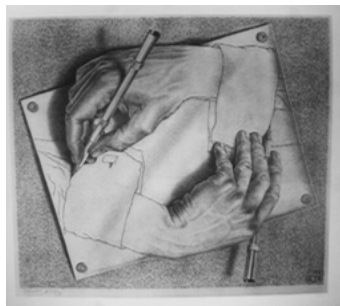
O que elas têm em comum ?

Objetivo da Aula

- Entender o mecanismo de **recursividade** nas linguagens de programação.
- Utilizar o conceito de recursividade para definir funções.

Definições

- Uma função **recursiva** é uma função que se refere a **si própria**.
- Utilizamos a própria função que estamos a definir na sua definição.



Funções Recursivas

Ideia Geral:

- **Caso Base**: o resultado é conhecido (não precisamos calculá-lo).
- **Caso Recursivo**: para resolver um problema de tamanho N , precisamos de uma solução a um problema de tamanho $M < N$ (**subproblemas** do problema inicial).

$$n! = \begin{cases} 1 & \text{se } n = 0 \text{ (caso base)} \\ n \times (n-1)! & \text{se } n > 0 \text{ (caso recursivo)} \end{cases}$$

Exemplo 1

Execução de funções recursivas

Versão não recursiva

```
int fatorial (int num){  
    int prod = 1;  
    int i;  
    for(i=n;i>=1;i--)  
        prod *= i;  
  
    return prod;  
}
```

Versão recursiva

```
int fatorial (int num){  
    if (num == 0)  
        return 1;  
    else  
        return num * fatorial (num-1);  
}
```

Exemplo 2: Sequências geradas recursivamente

1, 1, 2, 3, 5, 8 ...

$$fib(n) = \begin{cases} 1 & \text{se } 1 \leq n \leq 2 \\ fib(n-1) + fib(n-2) & \text{se } n > 2 \end{cases}$$

```
int fib( int n ){  
    if( n <= 2)  
        return 1;  
    else  
        return fib(n-1) + fib(n-2);  
}
```

Funções Recursivas

- Em geral, a todo procedimento recursivo corresponde um outro não recursivo (iterativo).

Vantagens da recursão:

- algoritmos mais concisos;
- simplifica a solução de alguns problemas;
- facilidade de implementação e compreensão;
- estratégia **divisão e conquista**.

Função Ackermann

$$A(m, n) = \begin{cases} n + 1 & \text{se } m = 0 \\ A(m - 1, 1) & \text{se } m > 0 \text{ e } n = 0 \\ A(m - 1, A(m, n - 1)) & \text{se } m, n > 0 \end{cases}$$

Values of $A(m, n)$

$m \backslash n$	0	1	2	3	4	n
0	1	2	3	4	5	$n + 1$
1	2	3	4	5	6	$n + 2 = 2 + (n + 3) - 3$
2	3	5	7	9	11	$2n + 3 = 2 \cdot (n + 3) - 3$
3	5	13	29	61	125	$2^{(n+3)} - 3$
4	13	65533	$2^{65536} - 3$	$2^{2^{65536}} - 3$	$2^{2^{2^{65536}}} - 3$	$\underbrace{2^{2^{\dots^2}}}_{n+3} - 3$

Algoritmo de Euclides:Máximo divisor comum

Alguns exemplos:

- $\text{MDC}(4,2) = ?$
- $\text{MDC}(8,7) = ?$
- $\text{MDC}(12,1) = ?$
- $\text{MDC}(20,15) = ?$
- $\text{MDC}(200,0) = ?$
- $\text{MDC}(X,Y) == \text{MDC}(Y,X) ?$

Algoritmo de Euclides

Euclides achou um jeito bem legal de calcular (**recursivamente**) o MDC

X	Y	Observação
9	6	$Y \neq 0, 9 \% 6 == 3$
6	3	$Y \neq 0, 6 \% 3 == 0$
3	0	$Y == 0, FIM$
X	Y	Observação
20	18	$Y \neq 0, 20 \% 18 == 2$
18	2	$Y \neq 0, 18 \% 2 == 0$
2	0	$Y == 0, FIM$

Algoritmo de Euclides

$$MDC(X, Y) = \begin{cases} X & \text{se } Y = 0 \\ MDC(Y, X \% Y) & \text{se } Y > 0 \end{cases}$$

```
int euclides_MDC(int a, int b){  
    if(b==0)  
        return a;  
    else  
        return euclides_MDC(b, a%b);  
}
```

Busca em vetor ordenado

Busca Binária

- Faça uma função que dado um vetor de inteiros v de tamanho n e um número inteiro x , retorne o índice m tal que $v[m] == x$. Se tal m não existe, a função deve retornar -1.

Se o vetor v está ordenado, nossa função poderia ser melhor ?

Exercícios

Defina recursivamente as seguintes funções. Assuma que os parâmetros x e y são inteiros positivos.

- $mult : x \times y$ (utilizando somas)
- $pow : x^y$. (utilizando multiplicações)

Par / Ímpar

Como poderia determinar se um inteiro positivo x é par ou não sem utilizar o resto da divisão ? Dica. Defina uma função recursiva cujos casos base são:

$$\begin{aligned} ehPar(0) &\rightarrow true \\ ehPar(1) &\rightarrow false \end{aligned}$$

Exercícios

- Calcular (recursivamente) o somatório de um vetor de números
- Definir uma função recursiva para determinar se uma palavra é um palíndromo.

Merge Sort

Dividir e Conquistar:

- 1 Em lugar de ordenar um vetor de n elementos, ordenamos dois subvetores (de $n/2$ elementos cada).
- 2 Após ordenar os subvetores, “unimos as soluções”

[5 , 3 , 2 , 10] ==> [5 , 3] , [2 , 10] (dividir)

[5 , 3] ==> [5] , [3] (dividir)

[5] + [3] ==> [3 , 5] (conquistar)

[2 , 10] ==> [2] , [10] (dividir)

[2] , [10] ==> [2 , 10] (conquistar)

[3 , 5] + [2,10] = [2,3,5,10] (conquistar)

Merge Sort

A parte recursiva... bem simples!

```
// Merge Sort
void mergeSort(int v[], int n){
    mergeRec(v,0, n-1);
}

// Ordenar o subvetor v[ini..fim]
void mergeRec(int v[], int ini, int fim){
    if (fim > ini) { // Condição de parada
        int m = (fim + ini) / 2; // metade
        mergeRec(v, ini, m); // subv a esquerda
        mergeRec(v, m+1, fim); // subv a direita
        intercalar(v, ini, m+1, fim); // Unir
    }
}
```

Merge Sort

Juntar os 2 subvetores já foi implementado antes:

```
// Unir v[ini..meio-1] + v[meio ..fim]
// Os subvetores devem estar ordenados
void interc(int v[], int ini, int meio, int fim){
    int aux[TAM];
    int i=ini, j=meio, k=0;
    while(i < meio && j <= fim){
        if(v[i] < v[j]) // copiar da esquerda
            aux[k++] = v[i++];
        else // copiar da direita
            aux[k++] = v[j++];
    }
    // Armazenar os valores que faltaram
    while(i < meio) aux[k++] = v[i++];
    while(j <= fim) aux[k++] = v[j++];
    // Copiar aux no vetor original
    k=0;
    for(i=ini ; i <= fim ; i++) v[i] = aux[k++];
}
```

Merge Sort

Como ordenamos de forma decrescente?
O que deve ser modificado?

Merge Sort

Só precisamos redefinir a função de intercalação.

Antes:

```
if (v[i] < v[j])  
    aux[k++] = v[i++];  
else  
    aux[k++] = v[j++];
```

Depois:

```
if (v[i] > v[j])  
    aux[k++] = v[i++];  
else  
    aux[k++] = v[j++];
```

Merge Sort

Note que isso funciona para os outros algoritmos de ordenação:

Antes:

```
void bubble(int v[], int n){  
    ...  
    if(v[j] > v[j+1])  
        swap(v, j, j+1);  
    ...  
}
```

Depois:

```
...  
if(v[j] < v[j+1])  
    swap(v, j, j+1);  
...
```

Teste!