COMPUTAÇÃO
U F C G

Revisão de Java

Recursão

Teoria da Recursão

- Dedeking [1888] – axiomatização da aritmética, estudo de isomorfismos entre estruturas. Isomorfismos satisfazendo os axiomas eram definidos por recursão primitiva.
- Kroneker[1887], Poincaré[1903,1913], Lebesgue[1905] – matemática (Teoria dos conjutnos de Cantor)
- Hilbert[1904] – Logica de primeira ordem
- Ciência da computação – equivalência com máquinas de Turing (base da idéia da universalidade dos computadores)

Teoria da Recursão

- **Recursion** in computer science is a method where the solution to a problem depends on solutions to smaller instances of the same problem.
  - "The power of recursion evidently lies in the possibility of defining an infinite set of objects by a finite statement. In the same manner, an infinite number of computations can be described by a finite recursive program, even if this program contains no explicit repetitions."

Teoria da Recursão

- https://en.wikipedia.org/wiki/Recursion_(computer_science)
- https://pt.wikipedia.org/wiki/Recursividade_(ci%C3%AAncia_da_computa%C3%A7%C3%A3o)
- http://www.techiedelight.com/recursion-practice-problems-with-solutions/

Teoria da Recursão

- Funções e algoritmos recursivos (template geral)
  - Caso(s) base(s) – situações triviais (terminais)
  - Caso indutivo/recursivo – dependem da aplicação da mesma função a casos mais simples.
- Tipos de dados recursivos
  - Expr = Numero | Expr * Expr | Expr + Expr | Expr – Expr | Expr / Expr
  - List<T> = Empty | Cons T List<T>
  - Natural = 0 | Natural + 1

Teoria da Recursão

- Tipos de recursão
  - **Recursão simples e recursão multipla**

```
int factorial(int n) {
   if (n == 0)
      return 1;
   else
      return (n * factorial(n-1));
}
```

```
procedure DFS(G,v):
  label v as discovered
  for all edges from v to w in G.adjacentEdges(v)
    if vertex w is not labeled as discovered then
      DFS(G,w)
```

Teoria da Recursão

- Tipos de recursão
  - **Indireta (ou mútua)**

```
fib2(n) { return fib(n-2); }

fib1(n) { return fib(n-1); }

fib(n){
  if (n>1)
    return fib1(n) + fib2(n);
  else
    return 1;
}
```

- Tipos de recursão
  - **Estrutural**

```
length []     = 0
length (h:t)  = 1 + length t
```

Exemplos

$$\text{fact}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot \text{fact}(n-1) & \text{if } n > 0 \end{cases}$$

```
procurar xi em [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11]
```

```
[x1,x2,x3,...xn] tal que x1 < x2 < x3 < ... < xn
binSearch(int[] array, int key, int left, int right) {
  middle = (left + right)/2
  if (key == array[middle])
      return true;
   else
      if(key < array[middle])
         return binSearch(array,key,left,middle-1)
      else
        return binSearch(array,key,middle+1,right)
}
```

Teoria da Recursão

- Questões de implementação
  - Função auxiliar

```
int factorialSquare (int n) {
    return factorialRecursivo(n*n);
}
```

```
int factorialRecursivo(int n) {
    if (n == 0)
        return 1;
    else
        return (n * factorialRecursivo(n-1));
}
```

Teoria da Recursão

- Recursão versus Iteração

```
function recursive(n)
    if n==base
        return x_base
    else
        return f(n, recursive(n-1))
```

```
function iterative(n)
    x = x_base
    for i = n downto base
        x = f(i, x)
    return x
```

```
int recursive(int n) {
   if (n == 0)
      return 1;
   else
      return (n * recursive(n-1));
}
```

```
int iterative(int n) {
  int product = 1;
  for (i = n; i > 0;i--) {
    product = product * i;
  }
  return product;
}
```

Teoria da Recursão

- **Poder expressivo**
  - Recursão e iteração possuem o mesmo poder e podem ser convertidos um no outro
- **Questões de performance**
  - Depende do compilador usado. Normalmente a versão iterativa é mais eficiente que a recursiva (manipulação da pilha de recursão). Em linguagem funcional, essa diferença é insignificante

Teoria da Recursão

- Tail Recursion (recursão em cauda)
  - A ultima ação de uma função recursiva é a chamada recursiva

Tail recursion:

```
//INPUT: Integers x, y such that x >= y and y > 0
int gcd(int x, int y)
{
  if (y == 0)
     return x;
  else
     return gcd(y, x % y);
}
```

Augmenting recursion:

```
//INPUT: n is an Integer such that n >= 0
int fact(int n)
{
   if (n == 0)
      return 1;
   else
      return n * fact(n - 1);
}
```

Recursão

- Importância da ordem de execução

```c
void recursiveFunction(int num) {
    printf("%d\n", num);
    if (num < 4)
        recursiveFunction(num + 1);
}
```

```c
void recursiveFunction(int num) {
    if (num < 4)
        recursiveFunction(num + 1);
    printf("%d\n", num);
}
```

```
1  recursiveFunction(0)
2  printf(0)
3          recursiveFunction(0+1)
4          printf(1)
5                  recursiveFunction(1+1)
6                  printf(2)
7                          recursiveFunction(2+1)
8                          printf(3)
9                                  recursiveFunction(3+1)
10                                 printf(4)
```

```
1  recursiveFunction(0)
2          recursiveFunction(0+1)
3                  recursiveFunction(1+1)
4                          recursiveFunction(2+1)
5                                  recursiveFunction(3+1)
6                                  printf(4)
7                          printf(3)
8                  printf(2)
9          printf(1)
10 printf(0)
```