Recursividade

Recursividade

- Um objeto é dito recursivo se ele consistir parcialmente ou for definido em termos de si próprio. Nesse contexto, um tipo especial de algoritmo será utilizado, algumas vezes, para a solução de alguns problemas. Esse procedimento é denominado recursivo.
- Em programação, uma função que chama a si mesma, é denominada de função recursiva.
- Toda função, recursiva ou não, deve possuir pelo menos uma chamada proveniente de um local exterior a ela. Essa chamada é denominada externa. Uma função não recursiva é, pois, aquela em que todas as chamadas são externas.

Recursividade

- O algoritmo recursivo precisa ter uma condição de parada. Da mesma forma que, quando utilizamos uma estrutura de repetição, precisamos de um condição que delimite o número de repetições a serem executadas, em um algoritmo recursivo precisamos de uma condição que nos diga quantas vezes o algoritmo será executado recursivamente.

Recursividade

A recursão pode ser direta e indireta:

a) Direta

São as chamadas feitas diretamente na própria função

b) Indireta

A recursão pode ser chamada de dentro de outras funções, do tipo:

f( ) -> f1( ) -> f2( ) ... -> fn( ) -> f( )

Recursividade

Para resolver um problema recursivo podemos aplicar o seguinte método:

- se a instância em questão é pequena,
  - resolva-a diretamente ;
- senão,
  - *reduza-a* a uma instância menor do mesmo problema,
  - aplique o método à instância menor e
  - volte à instância original.

Recursividade

Sabe-se que o cálculo de fatorial de um número n inteiro é a multiplicação de n pelos seus inteiros antecessores até o número inteiro 1. Se atribui a n o valor 5 (n = 5), o fatorial de n (n!) será:

5! = 5 * 4 * 3 * 2 * 1

ou

n! = n * (n-1) * (n-2) * (n-3) * (n-4)

Ressalva se faz para o número 0 (zero), em que o fatorial de 0 é sempre 1.

Recursividade

Podemos determinar o fatorial de 5 da seguinte forma:

5! = 5 * 4!

4! = 4 * 3!

3! = 3 * 2!

2! = 2 * 1!

1! = 1 * 0!

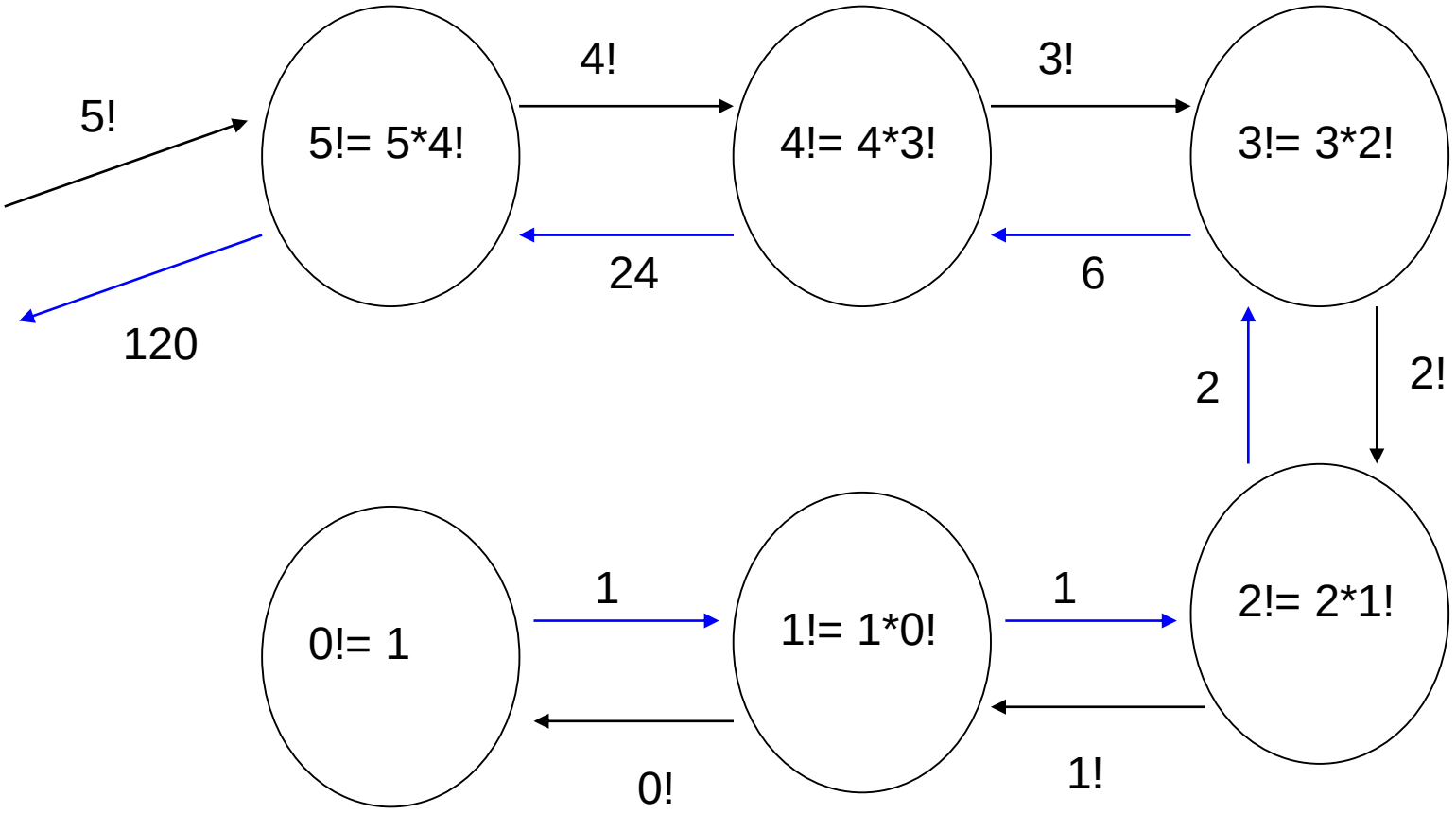
0! = 1

fatorial (0) = 1

fatorial (n) = n * fatorial (n-1)

```
def fatorial (n):
    if (n==0):
        return 1
    else :
        return n* fatorial (n-1)
```

Recursividade



Série de Fibonacci:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 ...

Fibonacci (n) =

- 0, se n= 1
- 1, se n= 2
- Fibonacci (n-1) + Fibonacci(n-2), se n>2

```
def fibonacci (n):
    if n==1:
        return 0
    elif n==2:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

Recursividade

- A recursão é tratada como qualquer outra chamada de função. Isso envolve guardar o estado atual do processamento de maneira que ela possa continuar de onde parou, quando a função estiver terminada. Guardar o estado de um processamento consome tempo e memória, por isso a recursão é usualmente tida como menos eficiente que a iteração (repetição).

Exercícios

1. Escrever uma função recursiva para mostrar a representação binária de um número inteiro.
2. Escrever uma função recursiva para mostrar o nro de dígitos de um número inteiro
3. Escrever uma função recursiva para inverter um número inteiro