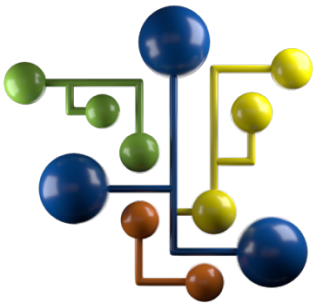

Data Science Academy

www.datascienceacademy.com.br

Introdução à Lógica de Programação

Para concluir a primeira parte do curso de Introdução à Lógica de Programação, um conceito que é considerado a base da teoria moderna da computação: Recursividade!

Em ciência da computação, a recursividade é a definição de uma sub-rotina (função ou método) que pode invocar a si mesma. Um exemplo de aplicação da recursividade pode ser encontrado nos analisadores sintáticos recursivos para linguagens de programação. A grande vantagem da recursão está na possibilidade de usar um programa de computador finito para definir, analisar ou produzir um estoque potencialmente infinito de sentenças, designs ou outros dados.

Praticamente todas as linguagens de programação usadas hoje em dia permitem a especificação direta de funções e procedimentos recursivos.

A recursão está profundamente entranhada na Teoria da computação, uma vez que a equivalência teórica entre as funções recursivas e as máquinas de Turing está na base das ideias sobre a universalidade do computador moderno.

Um algoritmo recursivo deve fazer pelo menos uma chamada a si mesmo, de forma direta (podemos ver o algoritmo sendo chamado dentro dele mesmo) ou indireta (o algoritmo chama um outro algoritmo, que por sua vez invoca uma chamada ao primeiro).

Um algoritmo recursivo deve ter pelo menos uma condição de parada, para que não seja invocado indefinidamente. Esta condição de parada corresponde a instâncias suficientemente pequenas ou simples do problema original, que podem ser resolvidas diretamente.

Para todo algoritmo recursivo existe pelo menos um algoritmo iterativo correspondente e vice-versa. Todavia, muitas vezes pode ser difícil encontrar essa correspondência.

Para quem é mais visual, acesse o endereço abaixo para compreender o conceito através das Bonecas Russas (da mesma imagem que está neste post):

https://pt.khanacademy.org/computing/computer-science/algorithms/recursive-algorithms/a/recursion

E quando usamos a Recursividade?

Muitos problemas têm a seguinte propriedade: cada instância do problema contém uma instância menor do mesmo problema. Dizemos que esses problemas têm estrutura recursiva. Para resolver uma instância de um problema desse tipo, podemos aplicar o seguinte método:

1- Se a instância em questão for pequena, resolva-a diretamente;

2- Senão, reduza-a a uma instância menor do mesmo problema;

3- Aplique o método à instância menor;

4- Volte à instância original;

O programador só precisa mostrar como obter uma solução da instância original a partir de uma solução da instância menor; o computador faz o resto. A aplicação desse método produz um algoritmo recursivo.

Lembra da Aula 4 quando usamos Loops para somar os dígitos de 1 a 5? Será que conseguimos resolver o mesmo problema usado Recursividade em Python? Sim, e é super simples. Observe:

```
# Função
def somaRec(n):
----if n > 0:
--------soma = somaRec(n - 1) + n
----else:
--------soma = 0
----return soma

# Executa a função
somaRec(5)
somaRec(1000)
```

O "segredo" do código acima está no n - 1. Vamos fazendo chamadas à mesma função de forma recursiva, até que a condição n >0 deixe de ser verdadeira.

É como se estivéssemos "andando para trás" até um ponto onde não podemos mais nos mover e então chegamos ao resultado.

Vantagens

Os algoritmos recursivos normalmente são mais compactos, mais legíveis e mais fáceis de serem compreendidos. Algoritmos para resolver problemas de natureza recursiva são fáceis de serem implementados em linguagens de programação de alto nível.

Desvantagens

Por usarem intensivamente a pilha, o que requer alocações e desalocações de memória, os algoritmos recursivos tendem a ser mais lentos que os equivalentes iterativos, porém pode

valer a pena sacrificar a eficiência em benefício da clareza. Algoritmos recursivos são mais difíceis de serem depurados durante a fase de desenvolvimento.

Aplicações

- Nem sempre a natureza recursiva do problema garante que um algoritmo recursivo seja a melhor opção para resolvê-lo. O algoritmo recursivo para obter a Sequência de Fibonacci (que vimos na Aula 1) é um ótimo exemplo disso.

- Algoritmos recursivos são aplicados em diversas situações como em: i) problemas envolvendo manipulações de árvores; ii) analisadores léxicos recursivos de compiladores; e iii) problemas que envolvem tentativa e erro ("Backtracking").

Aqui tem alguns bons exemplos:

https://pt.wikipedia.org/wiki/Recursividade_(ci%C3%AAncia_da_computa%C3%A7%C3%A3o)

Voltamos na segunda-feira com a Aula 6!

#aula05