Educação Profissional Paulista

Técnico em **Ciência de Dados**

Secretaria da Educação

SÃO PAULO
GOVERNO DO ESTADO

Lógica de programação e algoritmos

Recursividade

Aula 2

Código da aula: [DADOS]ANO1C3B3S21A2

Mapa da unidade 1 Componente 4

Lógica de programação e algoritmos

semana 10

Prática de estruturas de controle: condições

semana 16

Variáveis e tipos de dados

semana 21

Você está aqui!

Recursividade

semana 23

Busca e ordenação

Lógica de programação e algoritmos

Mapa da unidade 1
Componente 4

Você está aqui!

21

Recursividade

Aula 2

Código da aula: [DADOS]ANO1C3B3S21A2

Objetivos da aula

- Introduzir o conceito de recursão;
- Abordar como esse conceito funciona computacionalmente;
- Diferenciá-lo de alguns outros conceitos, como o de iteração.

Recursos didáticos

- Recurso audiovisual para exibição de vídeos e imagens;
- Acesso ao laboratório de informática e/ou internet.

Duração da aula

50 minutos.

Competências técnicas

- Ser proficiente em linguagens de programação para manipular e analisar grandes conjuntos de dados.

Competências socioemocionais

- Demonstrar resiliência para lidar com pressões e enfrentar novos desafios, bem como saber lidar com as frustrações, por exemplo, quando um projeto de ciência de dados falhar.

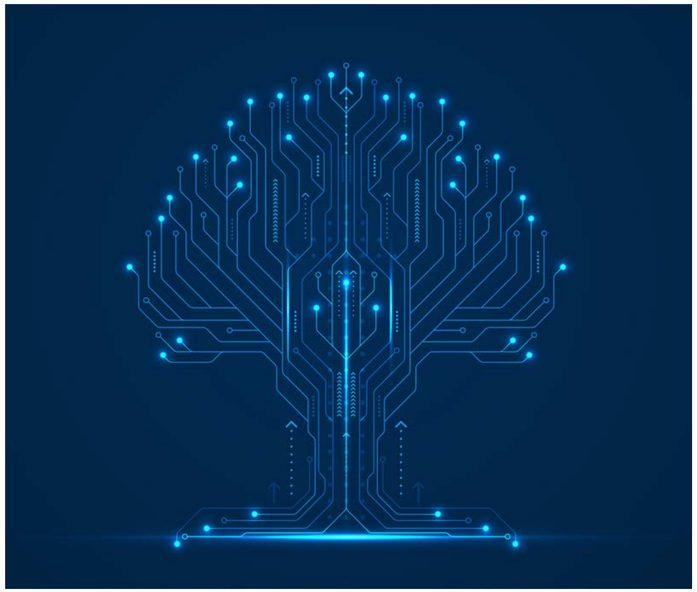
Construindo o **conceito**

Quando usar recursão?

Vimos, na aula anterior, que a recursão é uma ferramenta poderosa, mas que pode ter um custo computacional muito alto.

Dessa forma, a recursão é mais bem utilizada em **problemas** que podem ser divididos em subproblemas menores, de natureza semelhante, como: ordenação de dados, navegação em **estruturas de dados hierárquicas (árvores e grafos)**; e algoritmos de busca.

Vamos conhecer alguns exemplos na prática?



Construindo
o **conceito**

Recursão: método divisão e conquista

Divisão e conquista:

É uma estratégia para a solução de problemas computacionais que consiste em dividir o problema em subproblemas menores, resolver esses subproblemas recursivamente e combinar suas soluções para resolver o problema original.

Esse método é **bastante eficiente** em comparação com abordagens iterativas diretas para alguns problemas.

Conheceremos melhor essa estratégia em uma aula adiante.

Construindo o **conceito**

Exemplo prático de divisão e conquista

Imagine que temos uma **lista ordenada** e queremos encontrar um elemento nela.

Iterativamente, podemos **varrer a lista, elemento a elemento**, consumindo N operações do computador.

Uma outra solução é representada pelo código que você confere a seguir.

Construindo
o **conceito**

Exemplo prático de divisão e conquista

```python
def pesquisa_binaria(lista, item, baixo, alto):
    if baixo > alto:
        return False
    meio = (baixo + alto) // 2  # Usar divisão inteira
    if lista[meio] == item:
        return True
    elif lista[meio] > item:
        return pesquisa_binaria(lista, item, baixo, meio - 1)
    else:
        return pesquisa_binaria(lista, item, meio + 1, alto)

# Exemplo de uso:
lista_ordenada = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]
item_para_buscar = 5
# Lembre-se de iniciar "baixo" como 0 e "alto" como o último índice da lista
resultado = pesquisa_binaria(lista_ordenada, item_para_buscar, 0, len(lista_ordenada) - 1)

if resultado:
    print("Item encontrado.")
else:
    print("Item não encontrado.")
```

```
Item encontrado.
```

Elaborado especialmente para o curso com a ferramenta Jupyter Notebook.

Construindo
o **conceito**

Recursão: método *backtracking*

***Backtracking*:**

É outra estratégia para solução de problemas computacionais. Nele, utilizamos a recursão para resolver problemas de busca e otimização, explorando, sistematicamente, as possibilidades até encontrar a solução desejada ou concluir que não há solução.

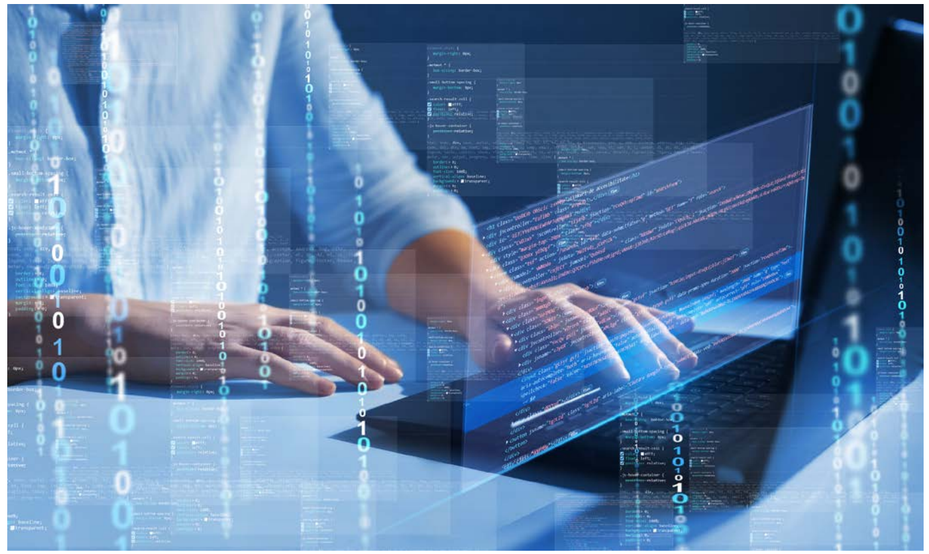
Construindo
o **conceito**

Exemplo de recursão com *backtracking*

Imagine que tenhamos um conjunto de números e **queremos verificar se um certo valor pode ser obtido** somando alguns deles.

Podemos **testar possibilidades** com o *backtracking*.

Vamos ver a implementação!



Construindo o **conceito**

Exemplo de recursão com *backtracking*

```python
def is_subset_sum(numbers, n, sum_target):
    # Caso base: se a soma alvo é 0, um subconjunto foi encontrado
    if sum_target == 0:
        return True
    # Se não há mais elementos para processar e a soma alvo não é alcançada
    if n == 0:
        return False

    # Se o último elemento é maior que a soma, ignora ele
    if numbers[n-1] > sum_target:
        return is_subset_sum(numbers, n-1, sum_target)

    # Caso contrário, verifica duas possibilidades:
    # 1. incluir o último elemento na soma
    # 2. não incluir o último elemento
    return is_subset_sum(numbers, n-1, sum_target) or is_subset_sum(numbers, n-1, sum_target-numbers[n-1])

# Exemplo de uso
numbers = [3, 34, 4, 12, 5, 2]
sum_target = 9
n = len(numbers)

if is_subset_sum(numbers, n, sum_target):
    print("Existe um subconjunto com soma dada.")
else:
    print("Não existe nenhum subconjunto com soma dada.")

```

```
Existe um subconjunto com soma dada.
```

Elaborado especialmente para o curso com a ferramenta Jupyter Notebook.

Registro

Colocando em **prática**

Esse é um bom cenário para uma recursão?

A partir dos estudos sobre recursão, analise as seguintes situações e responda:

Quais dos cenários abaixo são favoráveis para o uso de recursão e quais não são, e por quê?

1. Gerar uma sequência de números na qual os termos seguintes dependem dos anteriores.
2. Em uma aplicação de processamento de imagens, fatiar a imagem em imagens menores para processar cada pedaço por vez.
3. Varrer uma lista desordenada em busca de um número.

Ao finalizar a atividade, registre a sua resposta e envie pelo AVA.

Durante a aula.

Grupos de 4 pessoas

O que nós **aprendemos hoje?**

Nuvem de palavras

© Getty Images

O que nós **aprendemos hoje?**

Então ficamos assim...

1. Podemos utilizar recursão para implementar estratégias clássicas de resolução de problemas em computação.

2. Divisão e conquista é uma técnica de resolução de problemas através da qual problemas complexos podem ser divididos em subproblemas mais simples.

3. *Backtracking* é outra técnica de resolução de problemas, utilizada para explorar, sistematicamente, todas as possíveis combinações para encontrar uma solução desejada.

Saiba mais

Curtiu conhecer métodos de desenvolvimento de algoritmos? Que tal conhecer outros?

Neste artigo da Wikipédia, temos outros algoritmos clássicos para você conhecer! Confira:

WIKIPÉDIA. *Técnicas de projeto de algoritmos*, 14 jul. 2022. Disponível em: https://pt.wikipedia.org/wiki/T%C3%A9cnicas_de_projeto_de_algoritmos. Acesso em: 13 jun. 2024.

Referências da aula

FORBELLONE, A. L. V.; EBERSPÄCHER, H. F. *Lógica de programação*: a construção de algoritmos e estruturas de dados com aplicações em Python. Porto Alegre: Bookman, 2022.

Educação Profissional Paulista

Técnico em **Ciência de Dados**

Secretaria da Educação

SÃO PAULO
GOVERNO DO ESTADO