Grupo 1 – Algoritmos de Busca e Manipulação em Estruturas de Árvore

Exercício 1.1 – Implementar uma Árvore Binária de Busca com Inserção e Percursos

Descrição:
Crie uma classe de árvore binária de busca (BST) que permita inserir elementos e realizar percursos em pré-ordem, in-ordem e pós-ordem.
Exemplo:
Inserindo os elementos `[50, 30, 70, 20, 40, 60, 80]`, os percursos gerados serão:

- In-order: 20 30 40 50 60 70 80
- Pre-order: 50 30 20 40 70 60 80
- Post-order: 20 40 30 60 80 70 50

Exercício 1.2 – Implementar Deleção de um Nó na Árvore Binária de Busca

Descrição:
Implemente a remoção de um nó na BST, considerando os três casos: nó sem filhos, com um filho e com dois filhos (usando o sucessor in-order).
Exemplo:
Após inserir `[50, 30, 70, 20, 40, 60, 80]`, remova os nós 20, 30 e 50, observando a mudança do percurso in-order.

Exercício 1.3 – Implementar Busca de um Valor na Árvore Binária de Busca

Descrição:
Implemente uma função que, dado um valor, procure o nó correspondente na BST.
Exemplo:
Após inserir os elementos, busque o valor 40 e informe se o elemento foi encontrado.

Exercício 1.4 – Verificar se uma Árvore é uma BST Válida

Descrição:
Implemente uma função que verifica se uma árvore binária qualquer satisfaz a propriedade de BST, utilizando os limites mínimo e máximo permitidos para cada nó.
Exemplo:
Construa uma árvore e teste a função antes e depois de alterar manualmente algum nó para invalidar a propriedade BST.

Grupo 2 – Otimização de Desempenho com Técnicas de Paralelismo

Observação: Apesar do OpenMP ser uma API para C/C++/Fortran, em Python utilizaremos módulos como `multiprocessing` e `concurrent.futures` para explorar o paralelismo e otimizar a execução.

Exercício 2.1 – Soma Paralela de uma Lista Grande

Descrição:
Divida uma lista de números em partes e utilize o processamento paralelo para calcular a soma de cada parte, combinando os resultados para obter a soma total.
Exemplo:
Some os números de 1 até 10 milhões usando todos os núcleos disponíveis.

Exercício 2.2 – Multiplicação de Matrizes com Paralelismo

Descrição:
Utilize o módulo `multiprocessing` para paralelizar a multiplicação de duas matrizes pequenas, dividindo o cálculo por linhas.
Exemplo:
Multiplique duas matrizes 3×3 e exiba a matriz resultante.

Exercício 2.3 – Paralelismo para Processamento de Tarefas Pesadas (Contagem de Primos)

Descrição:
Implemente uma função que verifica se um número é primo e, em seguida, divida um intervalo grande entre vários processos para contar quantos números primos existem.
Exemplo:
Conte os números primos entre 1 e 100000 utilizando todos os núcleos disponíveis.

Exercício 2.4 – Comparação de Desempenho: Execução Sequencial vs. Paralela

Descrição:
Utilize a função de contagem de primos para comparar o tempo de execução entre uma abordagem sequencial e uma paralela, mensurando o desempenho de cada uma.
Exemplo:
Exiba o tempo gasto em cada método para o intervalo de 1 a 100000.

Grupo 3 – Soluções Otimizadas com Computação Paralela e Algoritmos de Busca em Árvore

Exercício 3.1 – Busca Paralela em Árvore Binária

Descrição:
Implemente uma função de busca em árvore que, utilizando processamento paralelo (com threads), procure um elemento simultaneamente nas subárvores esquerda e direita.
Exemplo:
Dada a árvore com raiz 50 e subárvores, procure o valor 60 de forma paralela.

Exercício 3.2 – Busca em Profundidade (DFS) Paralela com Retorno de Caminho

Descrição:
Utilize paralelismo para implementar uma busca em profundidade que retorne o caminho (lista de nós) desde a raiz até o elemento buscado.
Exemplo:
Para uma árvore simples com raiz 1, retorne o caminho até o nó 5.

Exercício 3.3 – Busca Paralela em Árvore para Encontrar o Valor Máximo

Descrição:
Implemente uma função que percorra a árvore de forma paralela para encontrar o maior valor armazenado.
Exemplo:
Em uma árvore com nós [15, 10, 20, 8, 12, 17, 25], o valor máximo deve ser 25.

Grupo 4 – Busca Otimizada de Prefixos IPv4/IPv6

Exercício 4.1 – Validação e Busca de Prefixo em IPv4

Descrição:
Utilize o módulo `ipaddress` para validar se um dado endereço IPv4 está contido em um prefixo (por exemplo, "192.168.1.0/24").
Exemplo:
Verifique se o IP `"192.168.1.5"` está dentro do prefixo `"192.168.1.0/24"`.

Exercício 4.2 – Implementar uma Trie para Prefixos IPv4

Descrição:
Crie uma estrutura Trie para armazenar prefixos IPv4 e implemente a busca pelo "longest prefix match" para um endereço dado.
Exemplo:
Insira os prefixos `["192.168.0.0/16", "192.168.1.0/24", "10.0.0.0/8"]` e, para o IP `"192.168.1.100"`, retorne o prefixo mais específico encontrado.

Exercício 4.3 – Implementar Busca de Prefixos para IPv6 utilizando Trie

Descrição:
Adapte a estrutura Trie para trabalhar com endereços IPv6, convertendo-os em uma string binária de 128 bits, e implemente a busca pelo prefixo mais longo.
Exemplo:
Insira os prefixos `["2001:db8::/32", "2001:db8:1234::/48"]` e, para o IP `"2001:db8:1234:5678::1"`, retorne o prefixo correspondente.

Exercício 4.4 – Comparação de Desempenho: Busca Linear vs. Trie para Prefixos IPv4

Descrição:
Implemente duas abordagens para encontrar o prefixo mais longo correspondente a um endereço IPv4: uma busca linear (iterando sobre uma lista de prefixos) e outra utilizando a estrutura Trie. Em seguida, compare os tempos de execução de cada método.
Exemplo:
Utilize uma lista de 1000 prefixos e compare o tempo gasto para encontrar o prefixo correspondente ao IP `"192.168.1.55"`.