Instituto Infnet

TP3 – Projeto de Bloco CC
Prof. Vitor Amadeu Souza

1) O objetivo deste exemplo é implementar um algoritmo de inserção, exclusão e busca em uma árvore binária. Você foi designado para criar um programa em Python que manipule uma árvore binária, carregando o arquivo gerado na TP1 com mais de 10.000 registros na árvore. Sua tarefa é implementar as seguintes operações:

a) Inserção de todos os registros na árvore binária.
b) Exclusão de um registro qualquer que comece com a letra 'M' na árvore binária.
c) Busca por um registro qualquer que comece com a letra 'Z' na árvore binária.
d) Função que imprima a árvore binária até a altura 5.
e) Considerando todos os registros no arquivo, qual seria a melhor escolha para o nó raiz da árvore binária? Por quê?

Instruções:
1. Implemente uma classe Node para representar os nós da árvore binária. Cada nó deve armazenar o nome de um registro e ter referências para seus nós filhos esquerdo e direito (ou ser uma folha).
2. Implemente uma classe BinaryTree que represente a árvore binária. Essa classe deve ter métodos para inserir, deletar e buscar os registros na árvore.
3. Crie um programa principal que permita ao usuário realizar operações de inserção, deleção e busca na árvore binária.
* Para cada solicitação forneça o código exemplo, tabelas de tempo etc.

2) Elabore um código em Python para implementar um algoritmo para estimar o valor de π usando o método de Monte Carlo considerando n=10.000.000.

a) Divida o processo em duas versões: uma utilizando o Numba para otimização com o decorador @numba.jit e outra sem otimização.

b) Elabore uma tabela e anote os tempos de execução de cada versão em ambos os sistemas operacionais Windows e Linux e repita o experimento de cada código pelo menos 6 vezes em cada sistema para obter uma média.

c) Compare os tempos de execução entre as versões otimizada e não otimizada em cada sistema operacional e responda se há diferença significativa no tempo de processamento entre elas. Faça um relatório de pelo menos 10 linhas explicitando os resultados dos testes, colocando observações e o que achar pertinente para a resposta.

Critérios de avaliação:
- Implementação correta do algoritmo de Monte Carlo para estimar π.
- Uso adequado do Numba para otimização com o decorador @numba.jit.
- Medição precisa do tempo de execução em ambos os sistemas operacionais.
- Realização de múltiplas execuções para avaliar a consistência dos resultados.
- Análise e comparação dos tempos de execução entre as versões otimizada e não otimizada em cada sistema operacional.
* Para cada solicitação forneça o código exemplo, tabelas de tempo etc.

3) Elabore um código em Python para implementar um algoritmo para estimar o valor de π usando o método de Monte Carlo considerando n=1.000.000 usando processamento paralelo.

a) Divida o processo em três versões: uma utilizando o Numba, outra usando o concurrent.futures e outra sem otimização.

b) Elabore uma tabela e anote os tempos de execução de cada versão em ambos os sistemas operacionais Windows e Linux e repita o experimento de cada código pelo menos 6 vezes em cada sistema para obter uma média.

c) Compare os tempos de execução entre as versões com e sem processamento paralelo em cada sistema operacional e responda se há diferença significativa no tempo de processamento entre elas. Faça um relatório de pelo menos 10 linhas explicitando os resultados dos testes, colocando observações e o que achar pertinente para a resposta.

d) A proposta desta questão é fazer teste de I/O bound. Salve o resultado de cada valor aleatório encontrado em arquivo texto chamado results.txt, sendo um valor para cada linha. Compare o resultado utilizando o Numba, outra usando o concurrent.futures e outra sem otimização. Houve algum ganho com o uso de processamento paralelo? Justifique medindo e comparando o tempo de cada código solicitado.

Critérios de avaliação:
- Implementação correta do algoritmo de Monte Carlo para estimar π.
- Uso adequado do Numba para processamento paralelo com o decorador @numba.njit(parallel=True).
- Uso adequado do concurrent.futures próprio do Python.
- Medição precisa do tempo de execução em ambos os sistemas operacionais.
- Realização de múltiplas execuções para avaliar a consistência dos resultados.
- Análise e comparação dos tempos de execução entre as versões com e sem processamento paralelo em cada sistema operacional.
* Para cada solicitação forneça o código exemplo, tabelas de tempo etc.

4) Elabore um programa em Python para implementar a busca de prefixos IPv4/IPv6 de maneira otimizada. Para isso, faça o que se pede:

a) Implemente uma classe Node para representar um nó na Árvore Trie em Python. Use o @Numba para otimizar a função.

b) Crie uma classe PrefixTree para representar a Árvore Trie. Esta classe deve ter um atributo root que aponta para o nó raiz da árvore e métodos insert(prefix) para inserir um prefixo na árvore e search(address) para buscar um endereço na árvore e retornar o prefixo correspondente.

c) Implemente um programa principal (main) para criar uma instância da classe PrefixTree, inserir no mínimo 10 prefixos IPv4/IPv6 e realizar a busca de alguns endereços para verificar se pertencem a algum prefixo na árvore.

d) Qual é a complexidade de tempo para inserir um prefixo IPv4/IPv6 na Árvore Trie implementada?

* Para cada solicitação forneça o código exemplo, tabelas de tempo etc.

Obs: Ao término do trabalho com todas as resoluções, forneça um arquivo PDF com todas as respostas incluindo códigos de teste e arquivos auxiliares e envie para avaliação na plataforma.