Árvore Binária Aleatória de Busca e Treap

Total de pontos 90/90

O e-mail do participante (**martinson.freitas@gmail.com**) foi registrado durante o envio deste formulário.

1. O que é uma Árvore Binária Aleatória de Busca (ABBA) e como ela difere de uma Árvore Binária de Busca tradicional em sua construção? *10/0

Uma Árvore Binária de Busca construída inserindo elementos de uma permutação aleatória, com cada permutação sendo igualmente provável. Diferente de árvores tradicionais que podem ser desbalanceadas por ordem de inserção específica, a ABBA busca o balanceamento via aleatoriedade na construção.

✓ 2. Explique por que a inserção de uma permutação aleatória de elementos é mais provável de produzir uma ABBA balanceada do que uma desbalanceada. *10/10

A inserção de uma permutação aleatória é mais provável de produzir uma ABBA balanceada porque a quantidade de sequências que geram árvores balanceadas é vastamente maior. Para 15 nós, existem milhões de sequências para árvores balanceadas contra apenas uma para a árvore totalmente desbalanceada. ✗

✓ 3. Como a combinação das propriedades de busca e heap garante que o formato de uma Treap esteja completamente determinado uma vez que a chave e a prioridade de cada nó são definidas? *10/10

A Treap combina as propriedades de árvore binária de busca e de heap: cada nó tem um dado x e uma prioridade p. A propriedade de heap garante que o nó de menor prioridade é a raiz, e a de busca direciona elementos menores/maiores para subárvores esquerda/direita. Essa combinação determina recursivamente a estrutura da Treap, fixando seu formato. ✗

✓ 4. Qual é o tempo esperado para uma operação find(x) em uma ABBA de tamanho *n*, tanto para elementos presentes quanto ausentes? *10/10

O tempo esperado para find(x) em uma ABBA de tamanho n é O(logn). Para elementos presentes ou ausentes, o comprimento esperado do caminho de busca é, no máximo, 2lnn + O(1). Esta expectativa é para qualquer x, não dependendo de sua escolha aleatória. ✗

✓ 5. Quais são as duas principais propriedades que um nó em uma Treap deve satisfazer? *10/10

1) Propriedade de Árvore Binária de Busca: para cada nó u, chaves menores que u.x estão na subárvore esquerda, e maiores na direita. 2) Propriedade de Heap: para cada nó u (exceto raiz), a prioridade do seu pai (u.pai.p) é menor que a sua própria (u.p). ✗

✓ 6. Como as Treaps superam a limitação das ABBA de não serem dinâmicas? *10/10

Treaps superam a limitação das ABBAs de não serem dinâmicas (não suportam add/remove). Elas utilizam rotações de árvore para manter a propriedade de heap durante as operações de inserção (bubble_up) e remoção (trickle_down). Isso permite que a estrutura seja modificada de forma eficiente enquanto mantém o desempenho esperado de O(logn). ✗

✗ 7. Descreva brevemente o conceito de uma "rotação" em uma Treap e seu propósito principal. *.../10

Seu propósito principal em uma Treap é manter a propriedade de heap após add(x) ou remove(x). Rotações (esquerda/direita) permitem que um nó se mova para cima ou para baixo para satisfazer as prioridades sem violar a ordem das chaves. ✗

✓ 8. Como a operação add(x) em uma Treap garante que a propriedade de heap seja mantida após a inserção de um novo nó? *10/10

Na add(x), um novo nó u com prioridade aleatória é inserido como folha, satisfazendo a propriedade de busca. Se u.pai.p > u.p, a função bubble_up(u) é chamada, que executa rotações no pai de u. Este processo eleva u na árvore até que u.pai.p < u.p ou u se torne a raiz, restabelecendo a propriedade de heap. ✗

✓ 9. Explique por que o tempo esperado da operação remove(x) em uma Treap é O(logn). *10/10

Isso se deve ao fato de que remove(x) é o inverso de add(x): se o elemento fosse reinserido com a mesma prioridade, a add(x) realizaria o mesmo número de rotações para retornar ao estado original. Portanto, como o tempo esperado de add(x) é O(logn), remove(x) também tem essa complexidade esperada. ✗

✓ 10. Em comparação com uma Skiplist, quais são as vantagens de desempenho das Treaps em termos de comprimento do caminho de busca esperado? *10/10

Em Treaps, o comprimento esperado do caminho de busca é 2lnn + O(1) (aproximadamente 1.386logn + O(1)). Em uma SkiplistSSet, é 2logn + O(1), ou e lnn + O(1) (aprox. 1.884logn + O(1)) com otimização. Assim, as Treaps possuem caminhos de busca esperados consideravelmente menores. ✗