UNIVERSIDADE
FEDERAL DO CEARÁ
UFC-CAMPUS CRATEÚS

Science Computer

DATAS STRUCTURE ADVANCED-CRT0026

Exercises And Resumes

Discente
Victor Gabriel Martins de Oliveira Loiola
Matrícula 473091

Professor
Msc. Luiz Alberto do Carmo Viana

Crateús
2020

Sumário

1 Exercises Resolved

1.1 Binary Search Tree

Exercise 1. O que aconteceria caso remove(leftMaxkey); fosse posta como a última instrução em seu bloco?

Prova

Se deslocamos remove(leftMaxKey) como última instrução do seu bloco resultaria em uma **falha de segmentação**, além do mais estaria também duplicando as chave e valor do leftMaxKey.Assim o conteúdo e a chave do **nó** que deseja-se apagar tem chave e valor do leftMaxKey.

Exercise 2. Por que o nó de leftMaxkey tem ao menos uma sub-árvore vazia?

Prova

Já que leftMaxKey é a chave com maior valor das sub-árvores da esquerda do nó pai, mas perceba que leftMaxKey está restrito somente a ter filho esquerdo ou não,resultando uma sub-árvore vazia que nesse caso seria a da direita.Logo concluir-se que leftMaxKey não deve ter filhos com chaves maiores que a sua, pois se tivesse ele não seria leftMaxKey e sim alguma chave maior na sub-árvore a sua direita.

Portanto leftMaxKey se tiver filho só pode ter-lo na sub-árvore a esquerda, o que resta uma sub-árvore vazia, mas no caso de não possuí filhos terá duas sub-árvores vazias.

Exercise 3. Pesquise sobre move semantics em C++.

Prova

No C ++ 11, as **semânticas de movimento** e as referências rvalue foram adicionadas à linguagem. Além disso, podemos aumentar significativamente o **desempenho**, evitando **cópias desnecessárias de objetos temporários**.

Exercise 4. Prove que, se um nó em uma Árvore Binária de Busca tem dois filhos, então seu sucessor não tem filho esquerdo e seu antecessor não tem filho direito.

Prova

Supondo por absurdo que o sucessor tem filho esquerdo e o antecessor tem filho direito. Por hipótese temos que um nó tem dois filhos, iremos chamar-lo de nó **X**.

Vamos considerar que por definição o **antecessor** é o nó com maior *valor de chave dentre os nós com valores menores* que a que o pai(X).Analogamente o **sucessor** é considerado a por definição que é a *chave com menor valor dentre as chaves com maiores valores* ao nó pai(X).

Temos 2 casos para o antecessor e 2 casos sucessor, à considerar.

2 casos do antecessor.

Caso 1.1: O antecessor é pai do nó **X**, assim o nó X é seu filho direito e X tem sua sub-árvore esquerda vazia.

observe o desenho.

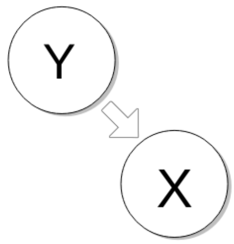

Figura 1: O nó Y é o antecessor do Nó X

Chegamos a um absurdo visto que nesse caso o nó X não pode ter filho esquerdo,pois o nó com valor da maior chave é o seu antecessor(pai) impossibilitando o nó X de possuir uma sub-árvore esquerda, e por hipótese o nó X possui dois filhos.

Caso 1.2: O antecessor de X é um descendente, assim ele possui chave com maior valor dentre os nós com chaves com menores valores que X.Assim a sub-árvore a direita do antecessor possui chave com valor entre o nó X e o antecessor.

$$A < FDA < X$$

onde:

A é o **A**ntecessor(Filho do nó X),
FDA é o **F**ilho **D**ireito do **A**ntecessor,
X é o nó pai.

Supomos que o antecessor de X tem nó direito, o que é uma contradição com a definição de um nó antecessor, pois se o antecessor tiver filho direito, esse nó não pode ser intitulado antecessor, visto que o antecessor é o nó com chave de maior valor dentre os nós de chaves com os menores valores.

2 casos do sucessor.

Caso 2.1: O sucessor é pai do nó **X**, assim o nó X é seu filho esquerdo e X tem sua sub-árvore esquerda vazia.

observe o desenho.

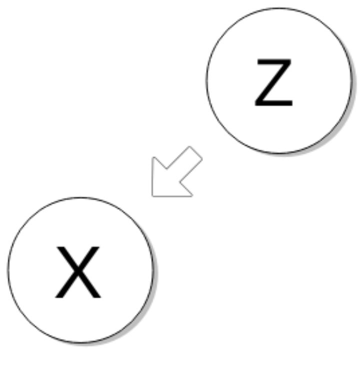

Figura 2: O nó Z é o sucessor do Nó X

Absurdo! Visto que por hipótese o nó X possui dois filhos, pela figura notamos que o nó X não pode ter filho direito, pois o sucessor é o nó com chave de maior valor, e os valores maiores que X ficam na

sub-árvore do sucessor(Y).
Caso 2.2: O sucessor é descendente do nó X.Portanto o sucessor possui chave com menor valor dentre a sub-árvore direita de X.Foi suposto que o sucessor tem filho esquerdo,logo, a chave do filho esquerdo do sucessor tem valor menor que a chave do sucessor e de chave com valor maior ao nó X.

$$S > FES > X$$

onde:

S é o **S**ucessor(Filho do nó X),
FES é o **F**ilho **E**squerdo do **S**ucessor,
X é o nó pai.

Como foi suposto, o sucesso possui filho esquerdo.Chegamos a um absurdo pois há uma quebra de definição de sucessor, perceba se o sucesso tem uma chave menor que a dele, essa chave menor deveria ser o sucessor e não seu pai que antes era sucessor.

Logo provamos que, se um nó em uma Árvore Binária de Busca tem dois filhos, então seu sucessor não tem filho esquerdo e seu antecessor não tem filho direito.

1.2 AVL Tree

Exercise 5. Prove que uma árvore binária de altura h tem no máximo $2^{h+1} - 1$ nós. Dica: tente usar indução.

Prova via indução

Caso Base:
Com altura 0 tempos 1 nó, com altura 1 temos 2 nós e com altura 2 tempos 7 nós.

$$h = 0 \text{ contém 1 nó.}$$
$$h = 1 \text{ contém 3 nós.}$$
$$h = 2 \text{ contém 7 nós.}$$

$$2^0 \text{ tem no máximo 1 nó.}$$
$$2^0 + 2^1 \text{ tem no máximo 3 nós.}$$
$$2^0 + 2^1 + 2^2 \text{ tem no máximo 7 nós.}$$

Agora utilizando a fórmula $2^{h+1} - 1$ para verificarmos o **primeiro caso onde a altura é 0**, como será utilizada o **primeiro princípio da indução**, vulgo indução fraca, verificar até a altura 2 é opcional.Atente-se ao fato que a segunda quantidade máxima(2^1) de nós depende da primeira(2^0), a terceira depende(2^2) da primeira quantidade de nós e da segunda.Assim temos que pegar a 2^h e somar com quantidade de nós máximos anteriores ao nó que deseja-se calcular a quantidade máxima de nós, é válido ressaltar que 2^h com $h \geq 0$, onde h é a altura,é a base para que que possa fazer os cálculos dos determinados níveis.

$$\text{com } h = 0$$
$$\text{pela fórmula } 2^{h+1} - 1 \text{ obtemos: } 2^{0+1} - 1 = 1 \text{ nó}$$
$$\text{com } h = 1$$
$$\text{pela fórmula } 2^{h+1} - 1 \text{ obtemos: } 2^{1+1} - 1 = 3 \text{ nós}$$
$$\text{pela fórmula } 2^{h+1} - 1 \text{ obtemos: } 2^{2+1} - 1 = 7 \text{ nós}$$

Como o caso base é verdadeiro, partiremos para o passo indutivo.
Passo Indutivo: $P(h) \to (h + 1)$.
Temos que a fórmula $2^{h+1}-1$ funciona até 2^h, com $h \geq 0$, sendo assim a hipótese de indução($H.I$).Deseja-ja provar que a fórmula vale para os $h + 1$-ésimos termos.observe a estrutura abaixo:

$$2^0 + 2^1 + 2^2 + ... + 2^h + 2^{h+1}$$

Perceba que fórmula válida para calcular a quantidade de nós máximos em uma árvore de até 2^h que dá liberdade suficiente para utiliza-se a hipótese de indução($H.I$).
Substituindo $H.I$ no somatório e desenvolvendo a equação após a inserção:

$$\begin{aligned} &2^0 + 2^1 + 2^2 + ... + 2^h + 2^{h+1} \\ &= 2^{h+1} - 1 + 2^{h+1} \\ &= 2^{h+1} + 2^{h+1} - 1 \\ &= 2^h(2^1 + 2^1) - 1 \\ &= 2^h \cdot 4 - 1 \\ &= 2^h \cdot 2^2 - 1 \\ &= 2^{h+2} - 1 \end{aligned}$$

Como concluímos os passos base e passo indutivo, sendo ambos verdadeiros, concluímos que $2^{h+1} - 1$ com h$\geq$ 0 é capaz de resultar no número de nós máximo em uma **árvore binária de busca** com altura h.

Exercise 6. Explique por que não é possível, mesmo após uma inserção ou remoção, haver um node com balance(node) $\in \{-2, -1, 0, 1, 2\}$.
Prova

Temos como propriedade da AVL Tree, que, cada nó pode ter o balance(node) $\in \{-1, 0, 1\}$.

***OBS:balance(node) é o mesmo que o fator de balanceamento.**

É válido ressaltar que a cada operação de inserção/remoção são feitas verificações no caminho do nó inserido/removido até a raiz, sempre verificando se há quebra da propriedade do fator de balanceamento em cada nó do caminho.Outro detalhe que apenas um nó por vez é inserido ou removido.Então a cada operação, temos que um nó é inserido ou removido e que só depois as verificações do fator de balanceamento são feitas, ou seja, existe um pequeno intervalo na qual a AVL pode está ou não desbalanceada, na possibilidade da estrutura arbórea está balanceada, seu balance(node) $\in \{-1, 0, 1\}$ por propriedade(invariante).

Analisando a segunda possibilidade, no qual configura-se o desbalanceamento no instante da operação (inserção/remoção) e etapa de verificações do nós do caminho modificado, temos que pelo fato de inserimos um só nó ou remover-lo as manutenções na árvore só serão feitas caso o fator de balanceamento for diferente do esperado pela propriedade, sendo assim, no instante onde verifica-se o balance(node) é -2 ou 2 automaticamente é feito as operações de manutenção da árvore(rotações,atualizações de altura).

Logo como não possível inserir mais de um nó por vez, consequentemente também não é possível ter balance(node) ≤ -3 ou ≥ 3 pelo fato que quando chega a 2 ou -2 já são feitas as manutenções e a árvore volta a ser balanceada.

Portanto só é possível ter o balance(node) $\in \{-2, -1, 0, 1, 2\}$.Observe que os valores de -2 e 2 são casos especiais, anteriormente já mencionados.

Exercise 7. Descreva uma forma de converter uma Árvore Binária de Busca com n nós em uma Árvore AVL em tempo $O(n \ log \ n)$.
Prova
TODO escreve isso quando fizer a questão ou transcreve-la do caderno.

Exercise 8. Em uma árvore binária, um nó é dito filho único sse ele tem pai e seu nó pai tem exatamente um filho. Vamos definir a razão de solidão de uma árvore binária $T(RS(T))$ como o número de filhos únicos em T dividido pelo número de nós de T . Prove que, se T é uma Árvore AVL, então $RS(T) \leq \frac{1}{2}$. Dica: quais nós de uma Árvore AVL podem ser filhos únicos?.
Prova
TODO escreve isso quando fizer a questão ou transcreve-la do caderno.

1.3 Red-Black Tree

TODO escreve o restante das questões quando termina-las ou transcrever do caderno.

2 Resumes

TODO escreve isso quando elaborá uma certa quantidade X de pequenos resumos.