Algoritmos e Estruturas de Dados II
Remoção em Árvores Binárias de Busca

Bacharelado em Análise de Sistemas, DCT–UFMS, 22/6/2005

Na sala de aula, vimos os procedimentos de busca e inserção em árvores binárias de busca. O procedimento de busca, como para com outras estruturas de dados que já vimos, foi preparado não só para encontrar uma chave na árvore, mas também para prever pontos de inserção de um novo nó na estrutura. No entanto, esse mesmo proecedimento de busca não é igualmente útil para uma remoção.

Na remoção, após encontrar o nó a ser removido na árvore, uma informação essencial é saber quem é o pai deste nó a ser removido, para que apontadores sejam modificados corretamente. Esta informação não é fornecida pelo procedimento de busca, mas uma ligeira modificação permite que esse procedimento devolva também essa informação útil. A seguir, apresentamos o procedimento BUSCA–ABB que foi apresentado em sala, mas com essa modificação considerada.

Os parâmetros do procedimento BUSCA–ABB são dois apontadores *pai* e *apt*, que são passados por referência e que contêm, inicialmente, λ e o endereço da raiz da árvore, respectivamente, a chave x passada por cópia, e um indicador f que, após a execução da busca, conterá um dos seguintes possíveis valores:

$$f \begin{cases} = 0, & \text{se a árvore é vazia,} \\ = 1, & \text{se } x \text{ pertence à árvore, } \textit{apt} \text{ aponta para o nó encontrado e } \textit{pai} \text{ aponta para seu pai,} \\ > 1, & \text{se } x \text{ não pertence à árvore.} \end{cases}$$

Observe que esses valores para o parâmetro f são os mesmos apresentados no procedimento de busca em uma árvore binária de busca visto em sala.

Observe, finalmente, que esse novo procedimento tem o mesmo tempo de execução do procedimento de busca apresentado em sala, já que a modificação essencial foi a inclusão de um apontador *pai* que acompanha o apontador *apt* durante a busca. Portanto, o tempo de execução do procedimento BUSCA–ABB é proporcional à altura da árvore binária de busca. No pior caso, a altura dessa árvore é proporcional à quantidade de elementos nessa estrutura e, portanto, a complexidade de tempo de pior caso de uma busca em uma árvore binária de busca é $O(n)$.

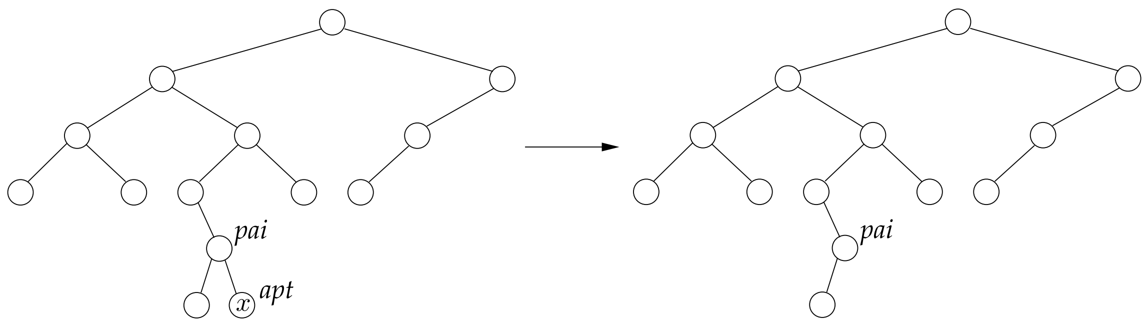
BUSCA–ABB(*pai*, *apt*, x, f): recebe dois apontadores *pai* e *apt*, uma chave x e um indicador f; as variáveis *pai*, *apt* e x são passadas por referência e a variável x é passada por cópia.

```
se apt = λ então
   f ← 0
senão
   se x = apt↑.chave então
      f ← 1
   senão
      se x < apt↑.chave então
         se apt↑.esq = λ então
            f ← 2
         senão
            pai ← apt
            apt ← apt↑.esq
            BUSCA–ABB(pai, apt, x, f)
      senão
         se apt↑.dir = λ então
            f ← 3
         senão
            pai ← apt
            apt ← apt↑.dir
            BUSCA–ABB(pai, apt, x, f)
```

Com a modificação do procedimento BUSCA–ABB, podemos agora pensar apenas na remoção. Então, suponha que após uma busca ter sido realizada, temos as informações que queremos: um apontador para o nó com a chave a ser removida e um apontador para seu nó pai. Temos então três casos principais a serem considerados:

Caso 1. O nó apontado por *apt* é uma folha da árvore.

Este caso é o mais simples. Basta fazer o apontador correto (esquerda ou direita) do *pai* receber λ e liberar o apontador *apt*.

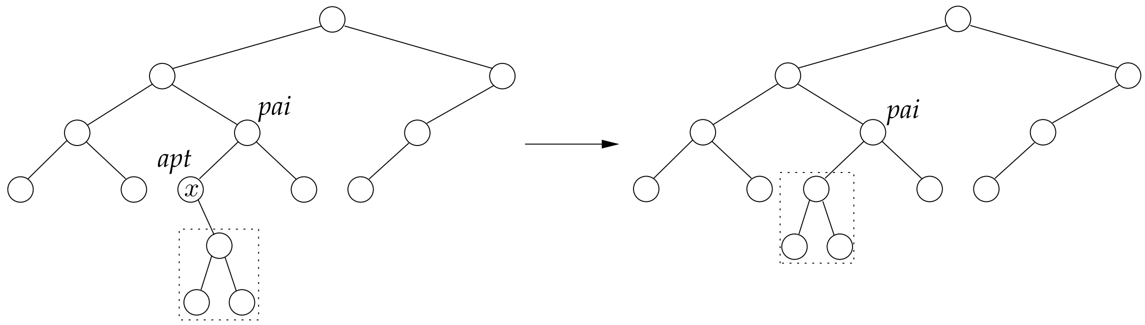


Caso 2. O nó apontado por *apt* tem um único filho.

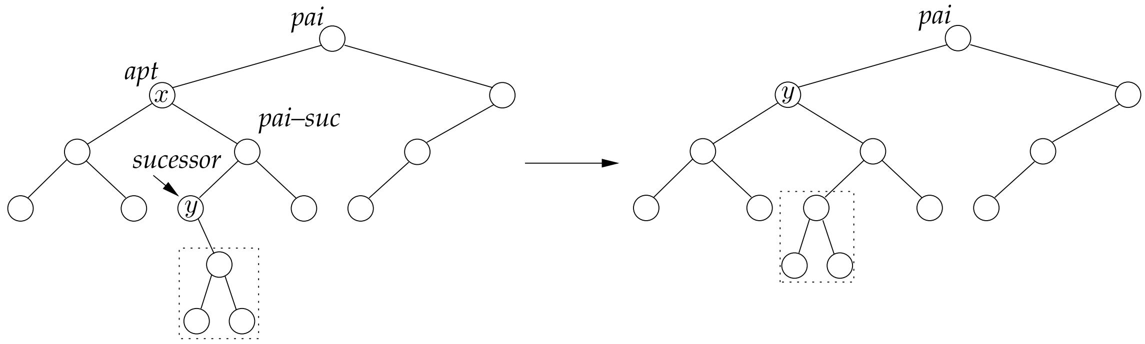
Então, o nó apontado por *apt* é filho esquerdo ou direito do nó apontado por *pai*. Basta fazer o apontador correto (esquerda ou direita) do nó *pai* apontar para o único filho do nó apontado por *apt*.

Caso 3. O nó apontado por *apt* tem dois filhos.

Neste caso, vamos transferir a tarefa da remoção para outro nó da árvore. Isto é, vamos encontrar o sucessor da chave x na árvore binária de busca, copiá-lo sobre a chave x e remover o nó



contendo esse sucessor. Note que o sucessor de x é a menor chave maior que x e pode ser obtido percorrendo a subárvore direita do nó contendo x, de sua raiz até o nó mais à esquerda. Esse nó pode ser uma folha ou pode ter um único filho. Nesse casos, usamos o Caso 1 ou o Caso 2 acima, respectivamente, para remover esse nó.



O procedimento de remoção em uma árvore de busca binária é apresentado a seguir. Esse procedimento, além dos casos previstos acima, considera o caso em que a remoção deve ser realizada na raiz da árvore e, assim, o nó apontado por *apt* não tem um pai ($pai = \lambda$). As instruções para processar esse caso em que a raiz deve ser removida estão diluídas nos casos 1, 2 e 3 descritos acima.

Outro detalhe importante no procedimento de remoção em uma árvore binária de busca é a utilização de um procedimento que busca por um nó que contém a chave sucessora da chave x a ser removida. Esse procedimento chama-se SUCESSOR e recebe como parâmetros uma cópia do apontador *apt* para o nó que contém a chave x, e duas referências para apontadores *sucessor* e *pai-suc* que, ao final desse procedimento, conterão o endereço do nó que contém a chave sucessora da chave x na árvore e o endereço no pai desse nó, respectivamente. Esse procedimento é simples e é descrito a seguir.

SUCESSOR(*apt*, *sucessor*, *pai-suc*): recebe um apontador *apt* para a raiz de uma subárvore binária de busca e devolve os apontadores *sucessor* e *pai-suc* que contêm o endereço do nó que tem a chave sucessora da chave contida no nó apontado por *apt* e o endereço do nó pai desse nó, respectivamente.

pai-suc $\leftarrow$ *apt*
sucessor $\leftarrow$ *apt*$\uparrow$.*dir*
enquanto *sucessor*$\uparrow$.*esq* $\neq \lambda$ **faça**
    pai-suc $\leftarrow$ *sucessor*
    sucessor $\leftarrow$ *sucessor*$\uparrow$.*esq*

Finalmente, observe que o tempo de execução do procedimento é dado pela operação de busca da chave x na árvore binária de busca. O restante das operações realizadas são todas de tempo constante (comparações e atribuições). Portanto, a complexidade de tempo de pior caso do procedimento REMOVE–ABB é $O(n)$.

REMOVE–ABB($raiz, x$): recebe um apontador *raiz* para a raiz da árvore e uma chave x e remove, se possível, o nó contendo a chave x.

```
pai ← λ
apt ← raiz
BUSCA–ABB(pai, apt, x, f)
se f = 0 então
  underflow
senão
  se f > 1 então
    escreva "chave não se encontra na árvore"
  senão
    se apt↑.esq = λ e apt↑.dir = λ então
      se apt = raiz então
        raiz ← λ
      senão
        se pai↑.esq = apt então
          pai↑.esq ← λ
        senão
          pai↑.dir ← λ
      LIBERAR(apt)
    senão
      se apt↑.esq = λ ou apt↑.dir = λ então
        se apt = raiz então
          se apt↑.esq ≠ λ então
            raiz ← apt↑.esq
          senão
            raiz ← apt↑.dir
        senão
          se pai↑.esq = apt então
            se apt↑.esq = λ então
              pai↑.esq ← apt↑.dir
            senão
              pai↑.esq ← apt↑.esq
          senão
            se apt↑.esq = λ então
              pai↑.esq ← apt↑.esq
            senão
              pai↑.esq ← apt↑.dir
        LIBERAR(apt)
      senão
        SUCESSOR(apt, sucessor, pai-suc)
        apt↑.chave ← sucessor↑.chave
        se sucessor↑.esq = λ e sucessor↑.esq = λ então
          pai-suc↑.esq ← λ
        senão
          pai-suc↑.esq ← sucessor↑.dir
        LIBERAR(sucessor)
```