SCC0502 - ALGORITMOS E ESTRUTURAS DE DADOS I

Árvores Binárias de Busca
Remoção e Balanceamento

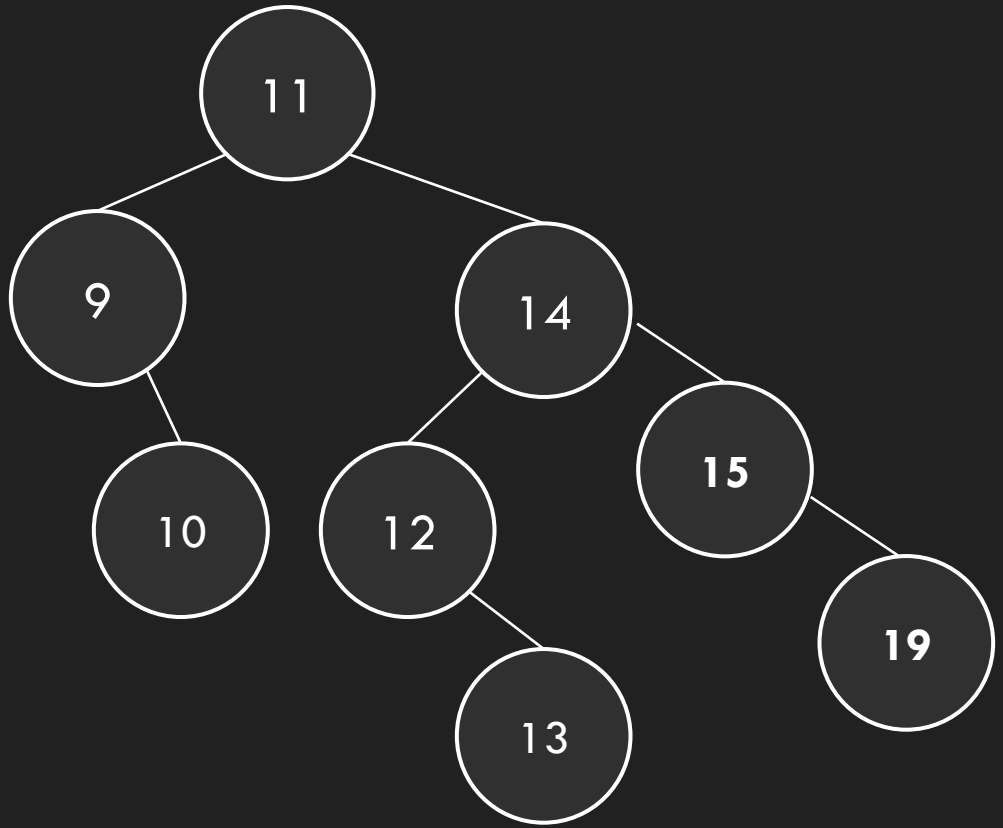
Prof.: Leonardo Tórtoro Pereira
leonardop@usp.br

Baseado nos slides do Prof. Rudinei Goularte

Conteúdo

- ~~Conceitos Introdutórios~~
- Operações
  - ~~Inserção~~
  - ~~Pesquisa~~
  - Remoção
- Conceitos Adicionais

Remoção em ABB



Remoção em ABB

- Casos a serem considerados no algoritmo de remoção de nós de uma ABB
  - Caso 1: o nó é folha
    - O nó pode ser retirado sem problema
  - Caso 2: o nó possui uma sub-árvore (esq/dir)
    - O nó raiz da sub-árvore (esq/dir) “ocupa” o lugar do nó retirado

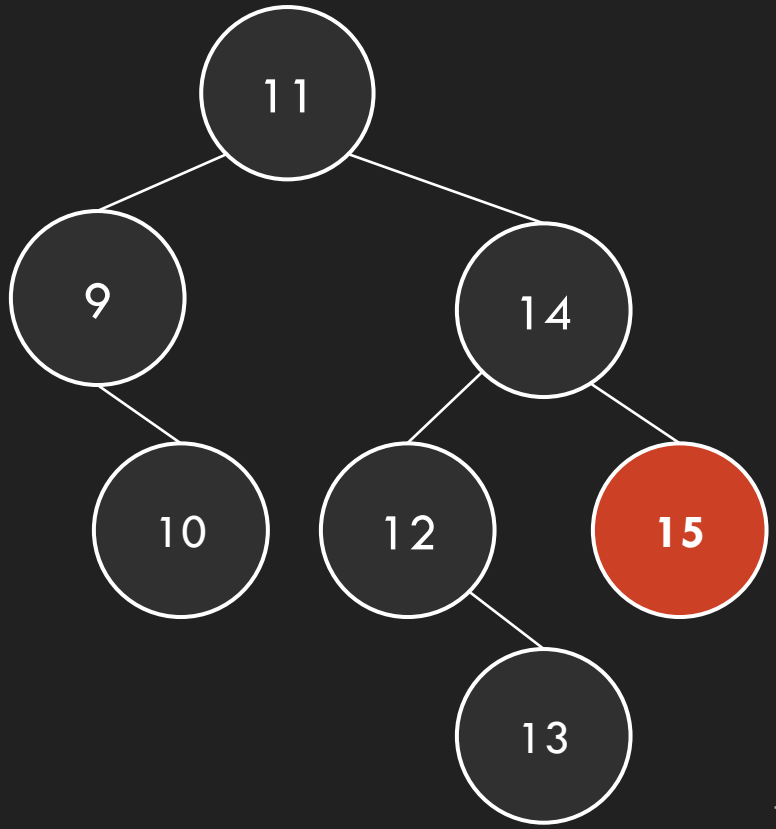
Remoção em ABB

- Casos a serem considerados no algoritmo de remoção de nós de uma ABB
  - Caso 3: o nó possui duas sub-árvores
    - O nó contendo o menor valor da sub-árvore direita pode “ocupar” o lugar
    - Ou o maior valor da sub-árvore esquerda pode “ocupar” o lugar

Remoção - Caso 1

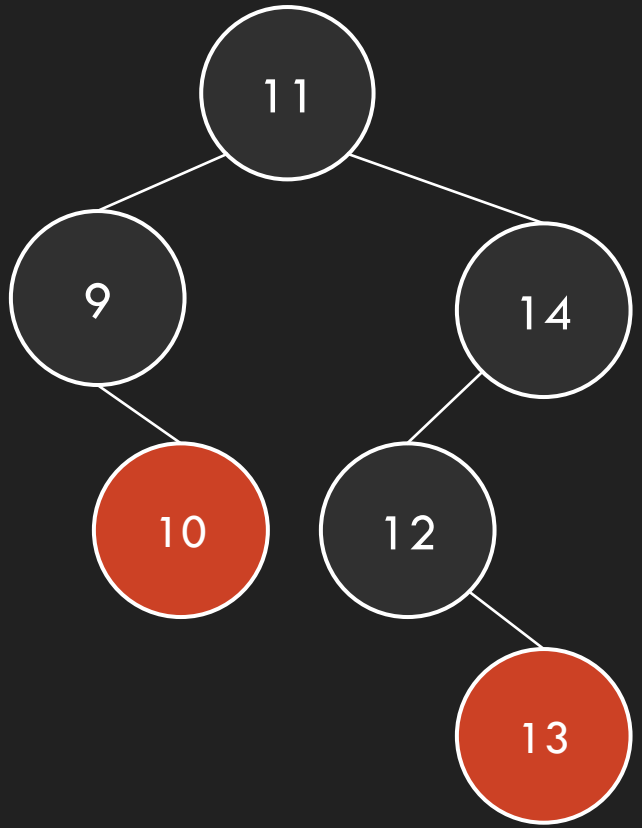
Remoção em ABB - Caso 1

- Caso o valor a ser removido seja o 15
- Pode ser removido sem problema, não requer ajustes posteriores



Remoção em ABB - Caso 1

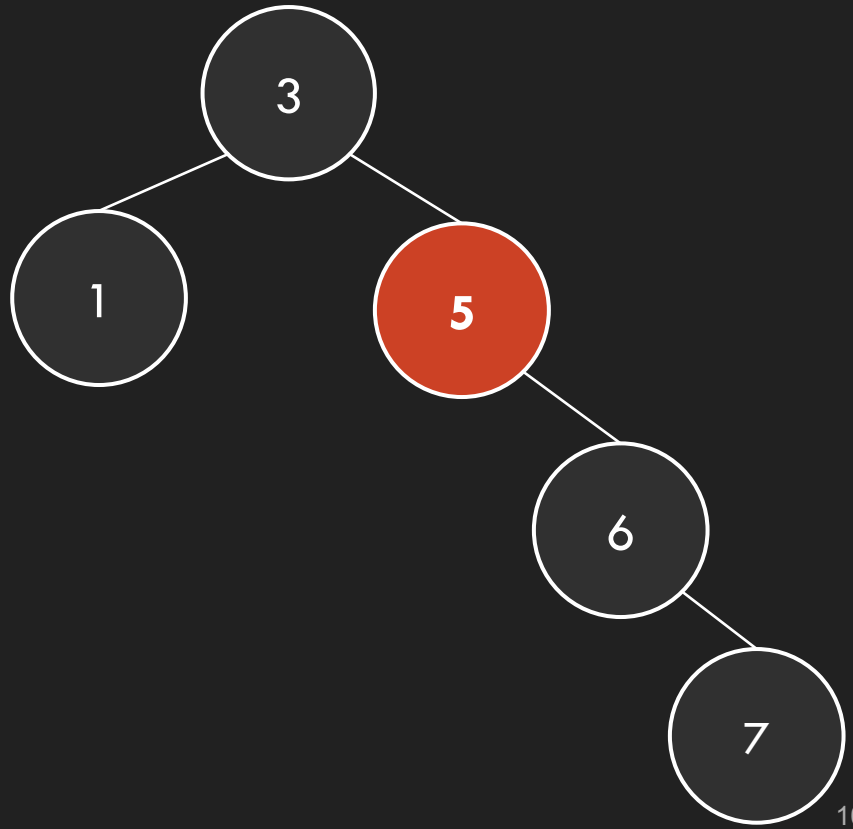
- ➔ Os nós com os valores 10 e 13 também podem ser removidos



Remoção - Caso 2

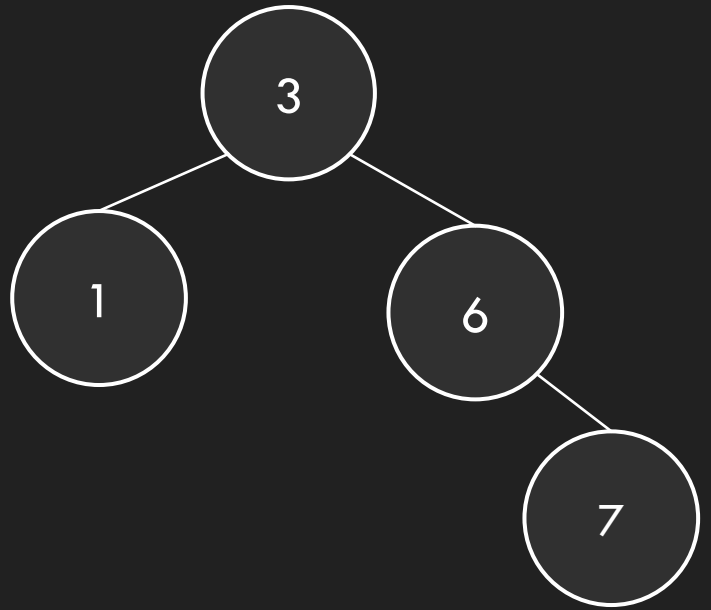
Remoção em ABB - Caso 2

- Removendo-se o nó com o valor 5
- Como ele possui somente a sub-árvore direita, o nó contendo o valor 6 pode “ocupar” o lugar do nó removido



Remoção em ABB - Caso 2

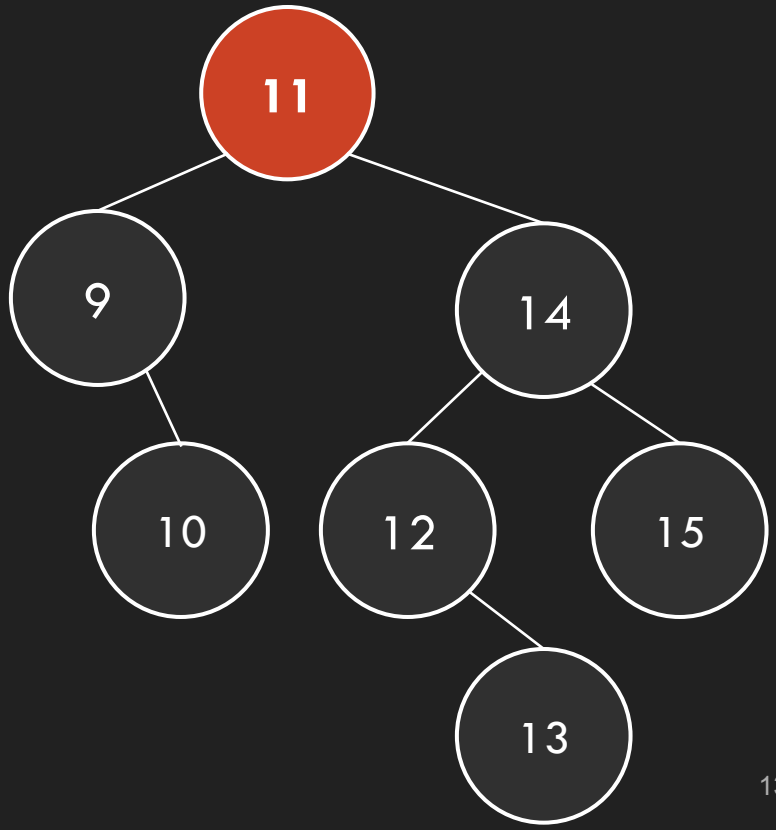
➔ Esse segundo caso é análogo caso existir um nó com sub-árvore esquerda apenas



Remoção - Caso 3

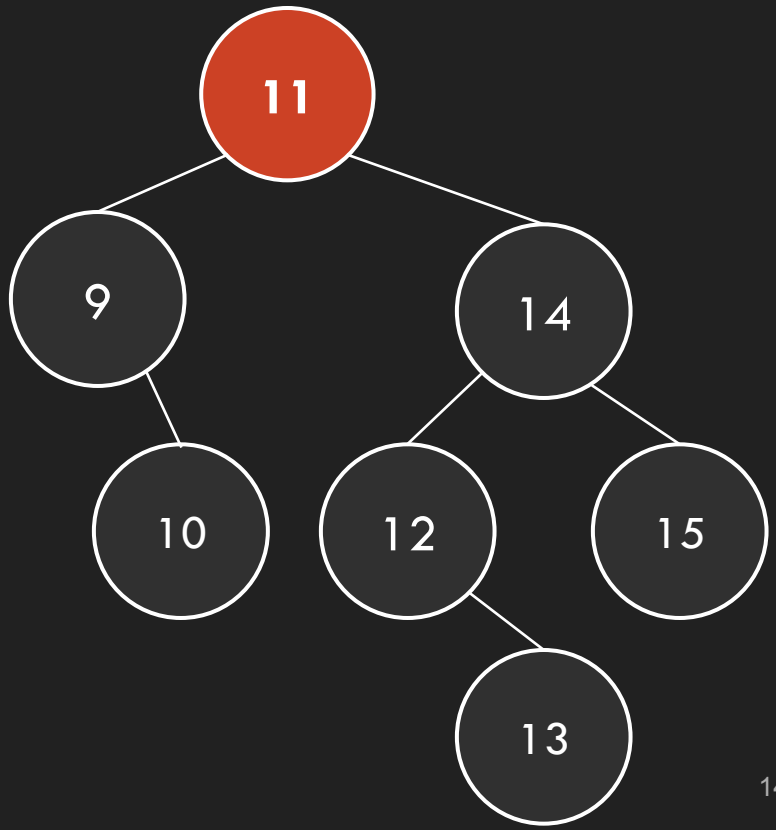
Remoção em ABB - Caso 3

➔ Eliminando-se o nó de chave 11



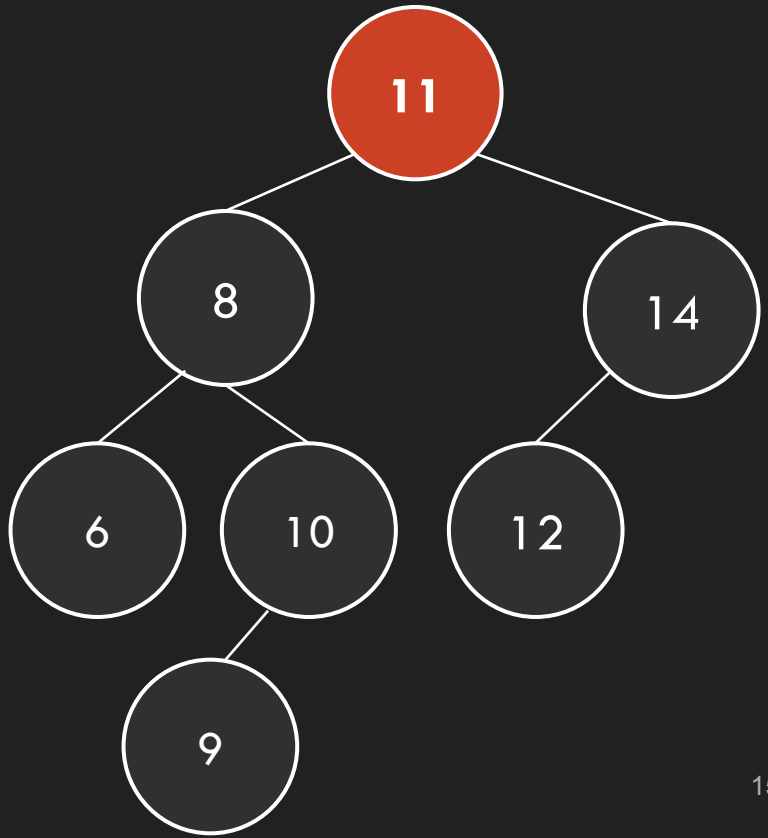
Remoção em ABB - Caso 3

- Neste caso, existem 2 opções
- O nó com chave 10 pode “ocupar” o lugar do nó-raiz, ou
- O nó com chave 12 pode “ocupar” o lugar do nó-raiz



Remoção em ABB - Caso 3

- Outro Exemplo:
- Eliminando o nó 11
- O nó 10 ocupa seu lugar do nó 11



Remoção em ABB

```
boolean abb_remover_aux (NO **raiz, int chave){
      NO *p;
      if(*raiz == NULL)
            return (FALSE);
      if(chave == item_get_chave((*raiz)->item))
      {
            if ((*raiz)->esq == NULL|| (*raiz)->dir == NULL)
            {/*Caso 1 se resume ao caso 2: há um filho ou nenhum*/
                  p = *raiz;
                  if((*raiz)->esq == NULL)
                        *raiz = (*raiz)->dir;
                  else
                        *raiz = (*raiz)->esq;
                  free(p);
                  p = NULL;
            }
            else /*Caso 3: há ambos os filhos*/
                  troca_max_esq((*raiz)->esq, (*raiz), (*raiz));
            return(TRUE);
      }
      …
```

Remoção em ABB

```
boolean abb_remover_aux (NO **raiz, int chave){
     …
     else
          if(chave < item_get_chave((*raiz)->item))
               return abb_remover_aux (&(*raiz)->esq, chave);
          else
               return abb_remover_aux (&(*raiz)->dir, chave);
}

boolean abb_remover(ABB *T, int chave){
     if (T != NULL)
          return (abb_remover_aux(&T->raiz, chave));
     return (FALSE);
}
```

Remoção em ABB

```
void troca_max_esq(NO *troca, NO *raiz, NO *ant)
{
     if(troca->dir != NULL)
     {
          troca_max_esq(troca->dir, raiz, troca);
          return;
     }
     if(raiz == ant)
          ant->esq = troca->esq;
     else
          ant->dir = troca->esq;
     raiz->item = troca->item;
     free(troca); troca = NULL;
}
```

Custo da Remoção em ABB

➔ A remoção requer uma busca pela chave do nó a ser removido, portanto, com custo de uma busca qualquer (tempo proporcional à altura da árvore).

Custo da Remoção em ABB

- O custo da remoção, após a localização do nó dependerá de 2 fatores:
  - Do caso em que se enquadra a remoção: se o nó tem 0, 1 ou 2 sub-árvores; se 0 ou 1 filho, custo é constante.
  - De sua posição na árvore, caso tenha 2 sub-árvores (quanto mais próximo do último nível, menor esse custo)

Custo da Remoção em ABB

- ➔ Repare que um maior custo na busca implica num menor custo na remoção pp. dita; e vice-versa.
- ➔ Logo, tem complexidade dependente da altura da árvore.
- ➔ Chamadas à “Troca_max_esq” requerem localizar o maior elemento da sub-árvore esquerda. Mas o número de operações é sempre menor que a altura da árvore.

Consequências das operações de inserção e eliminação

Consequências da inserção e da eliminação

- ➔ Uma ABB balanceada ou perfeitamente balanceada tem a organização ideal para buscas.
- ➔ Inserções e eliminações podem desbalancear uma ABB, tornando futuras buscas ineficientes.

Consequências da inserção e da eliminação

- ➔ Possível solução:
  - Construir uma ABB inicialmente perfeitamente balanceada (algoritmo a seguir)
  - Após várias inserções/eliminações, aplicamos um processo de rebalanceamento (algoritmo a seguir)

Algoritmo para criar uma ABB Perfeitamente Balanceada

Criar uma ABB Perfeitamente Balanceada

1. Ordenar num array os registros em ordem crescente das chaves;
2. O registro do meio é inserido na ABB vazia (como raiz);
3. Tome a metade esquerda do array e repita o passo 2 para a sub-árvore esquerda;
4. Idem para a metade direita e sub-árvore direita;
5. Repita o processo até não poder dividir mais.

Algoritmo de Rebalanceamento

Algoritmo de Rebalanceamento

1. Percorra em Em-ordem a árvore para obter uma sequência ordenada em array.
2. Repita os passos 2 a 5 do algoritmo de criação de ABB PB.

Resumo

ABB: Resumo

- Boa opção como ED para aplicações de pesquisa (busca) de chaves, SE árvore balanceada: O(log2 n)
- Inserções (como folhas) e Eliminações (mais complexas) causam desbalanceamento.
- Inserções: melhor se em ordem aleatória de chaves, para evitar linearização (se ordenadas)
- Para manter o balanceamento, 2 opções:
  - Como descrito anteriormente
  - Árvores AVL

Exercícios

- Escreva uma função que verifique se uma árvore binária está perfeitamente balanceada
  - O número de nós de suas sub-árvores esquerda e direita difere em, no máximo, 1

Exercícios

- ➔ Quais sequências de inserções criam uma ABB degenerada? Quais sequências criam uma ABB balanceada?
- ➔ Implemente um TAD para árvores binárias de busca com as operações discutidas em aula
- ➔ Implemente uma versão iterativa do algoritmo de remoção em ABBs

Referências

- Material baseado no originais produzidos pelo professor Rudinei Gularte
- SZWARCFITER, J. L.; MARKENZON, L. Estruturas de Dados e seus Algoritmos, Livros Técnicos e Científicos, 1994.
- TENEMBAUM, A.M., e outros Data Structures Using C, Prentice-Hall, 1990.
- ZIVIANI, N. Projeto de Algoritmos, Thomson, 2a. Edição, 2004.