BCC202 - Estruturas de Dados I

Aula 21: Árvores AVL

Pedro Silva

Universidade Federal de Ouro Preto, UFOP
Departamento de Computação, DECOM
Email: **silvap@ufop.edu.br**

2021

Abordagens de pesquisa em Memória Primária

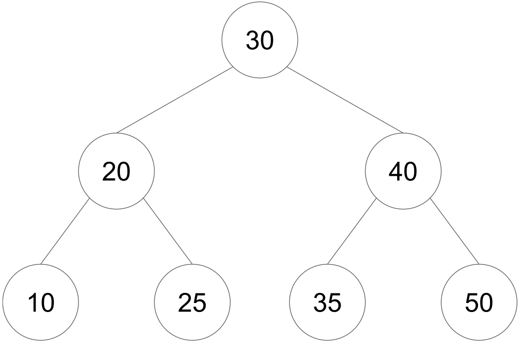
- Pesquisa Sequencial.
- Pesquisa Binária.
- Árvores de Pesquisa:
  - Árvores Binárias de Pesquisa.
  - Árvores AVL.
- Transformação de Chave (*Hashing*):
  - Listas Encadeadas.
  - Endereçamento Aberto.
  - *Hashing* Perfeito.

Conteúdo

Conteúdo

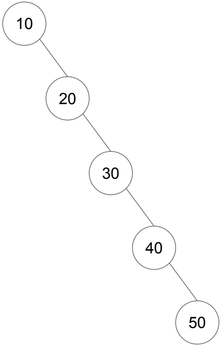
Árvore Binária de Pesquisa:

Inserindo os nós 30, 20, 40, 10, 25, 35 e 50 nesta ordem, teremos:



Árvore Binária de Pesquisa:

Inserindo os nós 10, 20, 30, 40 e 50 nesta ordem, teremos:



Árvores Binárias Balanceadas

- Existem ordens de inserção de nós que conservam o balanceamento de uma árvore binária.
- Na prática é impossível prever essa ordem ou até alterá-la.
  - Algoritmos para balanceamentos.
- A vantagem de uma árvore balanceada com relação a uma degenerada está em sua eficiência.
- Por exemplo:
  - Em uma árvore binária degenerada de 10.000 nós são necessárias, em média, 5.000 comparações (semelhança com arrays ordenados e listas encadeadas).
  - Numa árvore balanceada com o mesmo número de nós essa média reduz-se a 14 comparações.

Definição de Árvores Balanceadas

- Uma árvore binária balanceada é aquela na qual, para cada nó, as alturas de suas sub-árvores esquerda e direita diferem de, no máximo, 1.
- Fator de balanceamento (FB) de um nó é a diferença entre a altura da sub-árvore esquerda em relação à sub-árvore direita.

$$FB(p) = altura(\text{sub-árvore esquerda de p}) - altura(\text{sub-árvore direita de p})$$

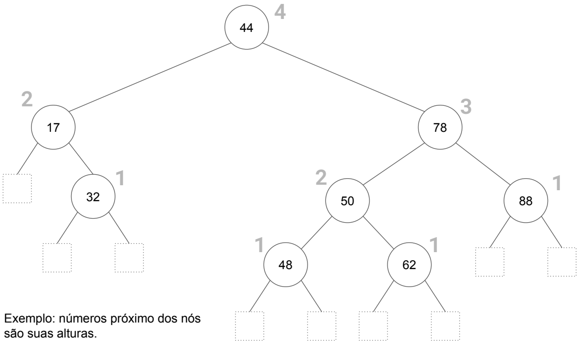
- Em uma árvore binária balanceada todos os FB de todos os nós estão no intervalo $-1 \leq FB \leq 1$

Conteúdo

Árvore AVL

- **Árvore binária de busca** tal que, para qualquer nó interno v, *a diferença das **alturas** dos filhos de v é no máximo 1*.

- Árvores AVL são balanceadas



Exemplo: números próximo dos nós são suas alturas.

AVL

- Algoritmo de balanceamento de árvores binárias.
- A origem da denominação AVL vem dos seus dois criadores:
  Adel'son-**V**el'skii e **L**andis.
- Ano de divulgação: 1962.

TAD - Árvores AVL

```c
typedef long TChave;

typedef struct {
    TChave chave;
    /* outros componentes */
} TRegistro;

typedef struct TNo {
  TRegistro reg;
  TNo *pEsq, *pDir;
} TNo;

typedef TNo * TDicionario;
```

FB e Altura

```c
int FB (TNo* pRaiz) {
    if (pRaiz == NULL)
        return 0;

    return Altura(pRaiz->pEsq)
           - Altura(pRaiz->pDir);
}
```

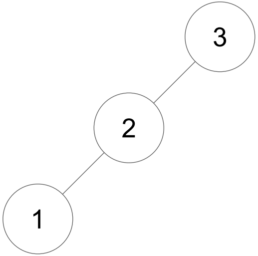
```c
int Altura(TNo* pRaiz) {
    int iEsq, iDir;

    if (pRaiz == NULL)
        return 0;

    iEsq = Altura(pRaiz->pEsq);
    iDir = Altura(pRaiz->pDir);

    if ( iEsq > iDir )
        return iEsq + 1;
    else
        return iDir + 1;
}
```

Inserção em Árvores AVL

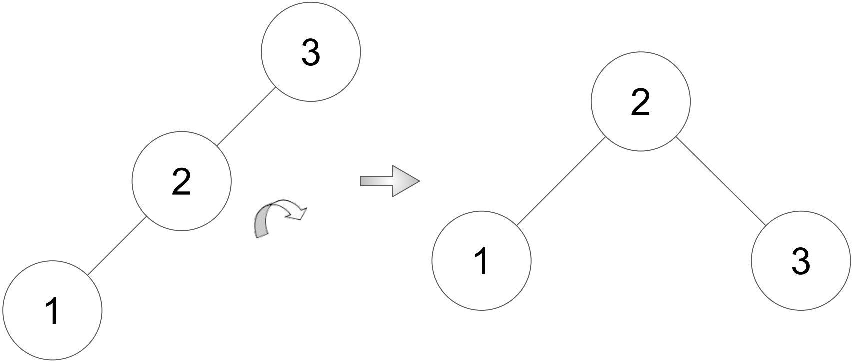
- Inicialmente inserimos um novo nó na árvore normalmente.
- A inserção deste pode degenerar a árvore (desbalancear).
- A restauração do balanceamento é feita por meio de rotações na árvore no nó "**pivô**".
- Nó "**pivô**" é aquele que após a inserção possui **Fator de Balanceamento** fora do intervalo.
- Como resolver? Rotações!

AVL - Rotação Simples para a Direita

- FB > 1 (subárvore esquerda maior que subárvore direita).
- E a subárvore esquerda desta subárvore esquerda é maior que a subárvore direita dela.
- Então realizar uma rotação simples para a direita.

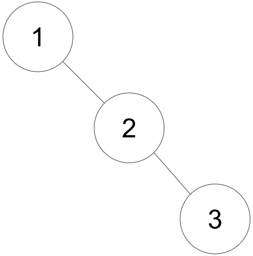


AVL - Rotação Simples para a Direita

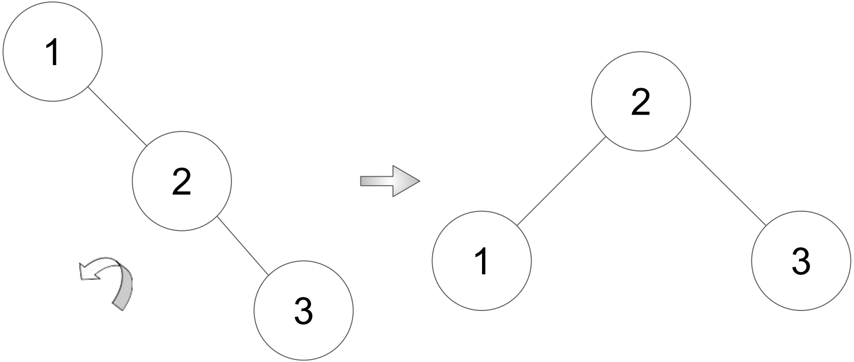


AVL - Rotação Simples para a Esquerda

- FB $<$ -1 (subárvore esquerda menor que subárvore direita).
- E a subárvore direita desta subárvore direita é maior que a subárvore esquerda dela.
- Então realizar uma rotação simples para a esquerda.

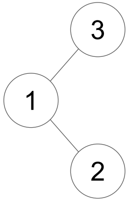


AVL - Rotação Simples para a Esquerda

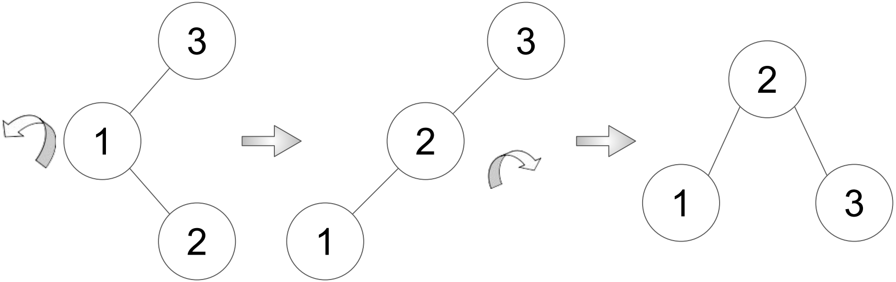


AVL - Rotação Dupla para a Direita

- FB > 1 (subárvore esquerda maior que subárvore direita).
- E a subárvore esquerda desta subárvore esquerda é menor ou igual que a subárvore direita dela.
- Então realizar uma rotação dupla para a direita.

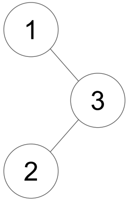


AVL - Rotação Dupla para a Direita

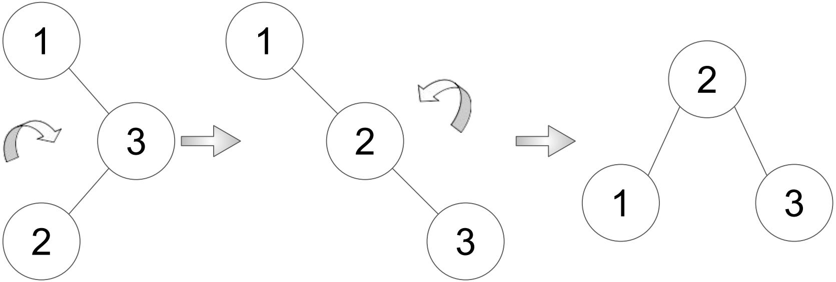


AVL - Rotação Dupla para a Esquerda

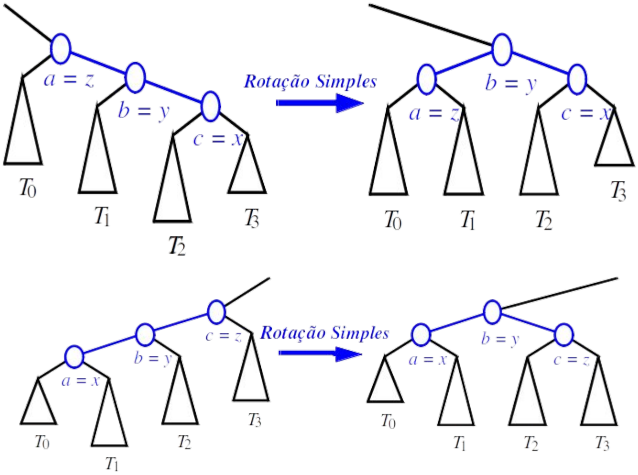
- FB $<$ -1 (subárvore esquerda menor que subárvore direita).
- E a subárvore direita desta subárvore direita é menor que a subárvore esquerda dela.
- Então realizar uma rotação dupla para a esquerda.



AVL - Rotação Dupla para a Esquerda



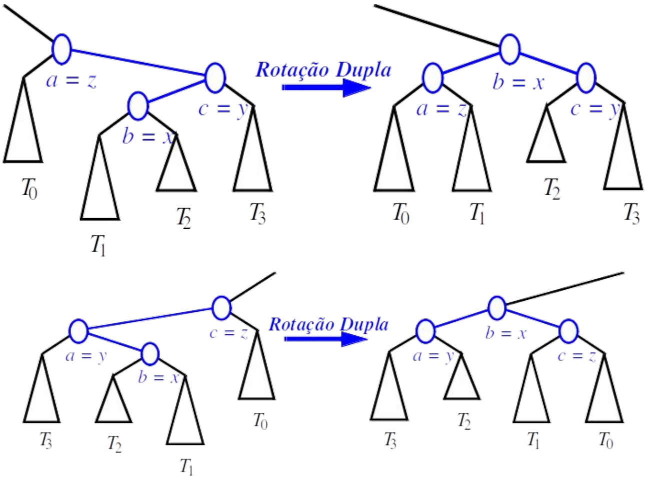
Exemplos de Rotação Simples



Rotação Simples

```c
void RSE(TNo** ppRaiz) {
    TNo *pAux;
    pAux = (*ppRaiz)->pDir;
    (*ppRaiz)->pDir = pAux->pEsq;
    pAux->pEsq = (*ppRaiz);
    (*ppRaiz) = pAux;
}

void RSD(TNo** ppRaiz) {
    TNo *pAux;
    pAux = (*ppRaiz)->pEsq;
    (*ppRaiz)->pEsq = pAux->pDir;
    pAux->pDir = (*ppRaiz);
    (*ppRaiz) = pAux;
}
```

Exemplos de Rotação Dupla



Rotação Dupla

```c
int BalancaEsquerda(TNo** ppRaiz) {
  int fbe = FB ( (*ppRaiz)->pEsq );
  if ( fbe > 0 ) {
    RSD(ppRaiz);
    return 1;
  } else if (fbe < 0 ) {
    /* Rotação Dupla Direita */
    RSE( &((*ppRaiz)->pEsq) );
    RSD( ppRaiz ); /* &(*ppRaiz) */
    return 1;
  }
  return 0;
}
```

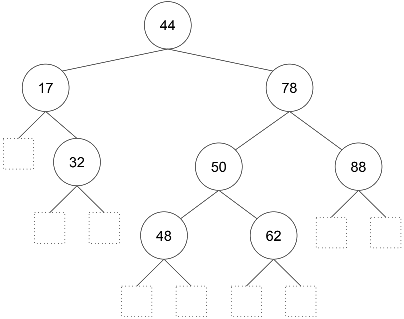
```c
int BalancaDireita(TNo** ppRaiz) {
  int fbd = FB( (*ppRaiz)->pDir);
  if ( fbd < 0 ) {
    RSE (ppRaiz);
    return 1;
  } else if (fbd > 0 ) {
    /* Rotação Dupla Esquerda */
    RSD( &((*ppRaiz)->pDir) );
    RSE( ppRaiz ); /* &(*ppRaiz) */
    return 1;
  }
  return 0;
}
```

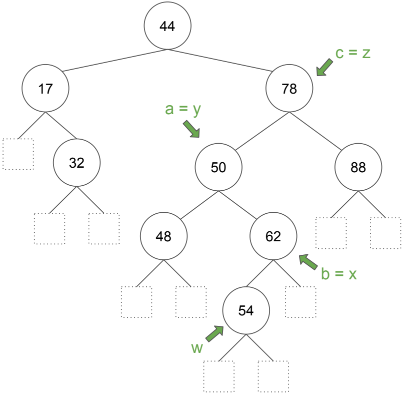
Balanceamento

```
int Balanceamento(TNo** ppRaiz) {
  int fb = FB(*ppRaiz);
  if ( fb > 1)
    return BalancaEsquerda(ppRaiz);
  else if (fb < -1 )
    return BalancaDireita(ppRaiz);
  else
    return 0;
}
```

Inserção em uma Árvore AVL

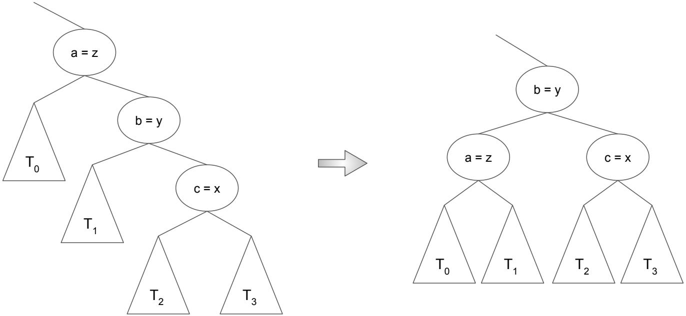
- Inserção como em uma árvore binária de pesquisa.
- Sempre feita expandindo um nó externo.





Reestruturação Trinodo

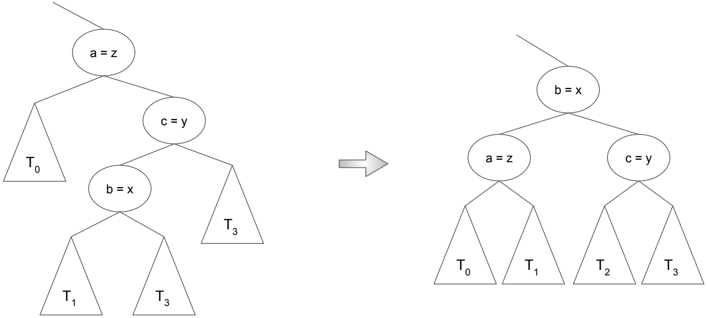
- Os nós x, y, z (*filho*, *pai*, *avô*) renomeados como a, b, c (percurso inter-fixado).
- Rotações levam b para o topo.



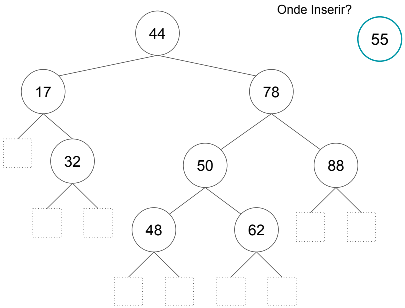
Outros dois casos são simétricos

Reestruturação Trinodo

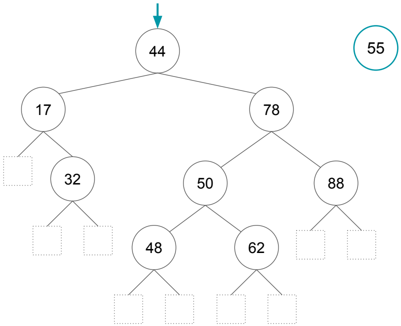
- Os nós x, y, z (*filho*, *pai*, *avô*) renomeados como a, b, c (percurso inter-fixado).
- Rotações levam b para o topo.



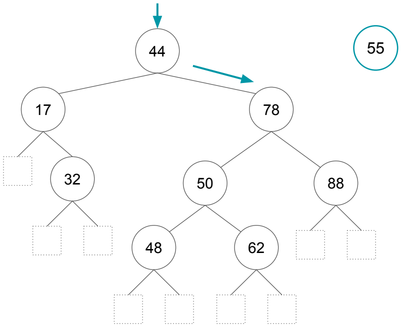
Exemplo de Inserção



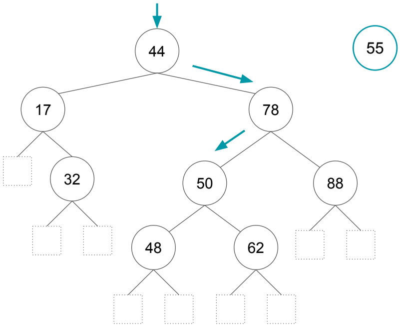
Exemplo de Inserção



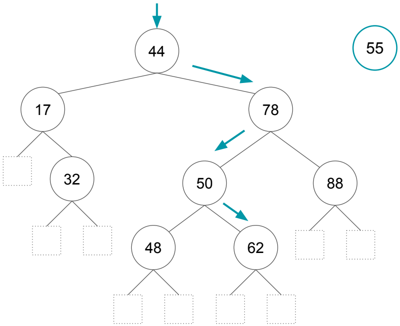
Exemplo de Inserção



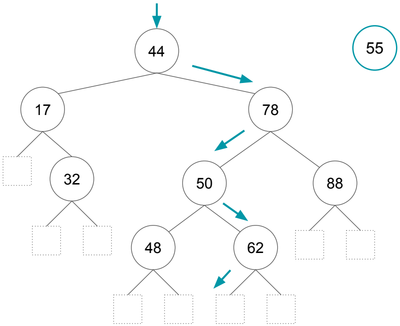
Exemplo de Inserção



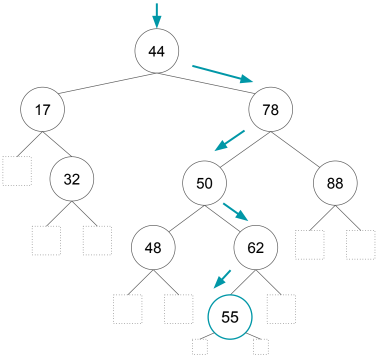
Exemplo de Inserção



Exemplo de Inserção

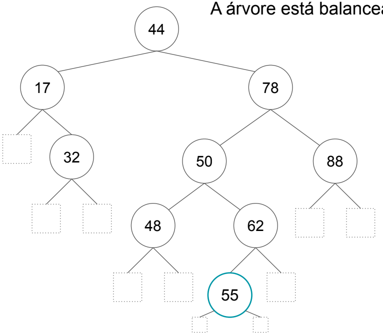


Exemplo de Inserção

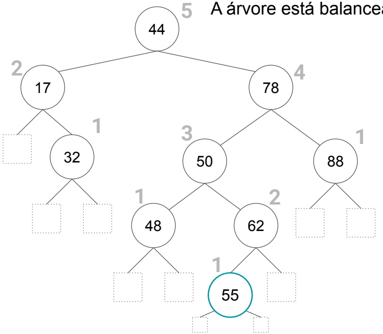


Exemplo de Inserção

A árvore está balanceada?

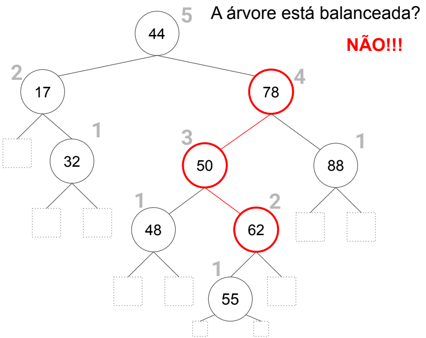


Exemplo de Inserção

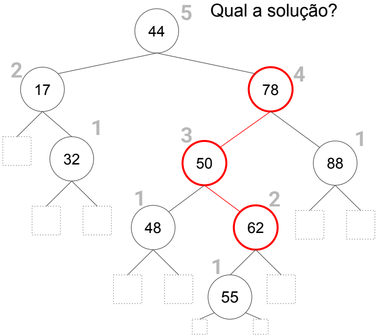


A árvore está balanceada?

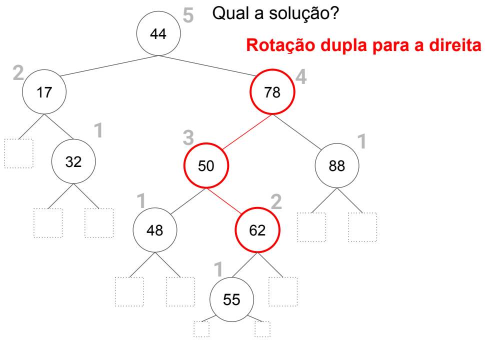
Exemplo de Inserção



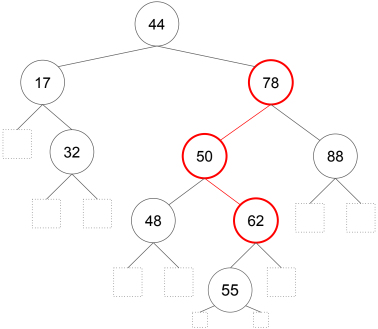
Exemplo de Inserção



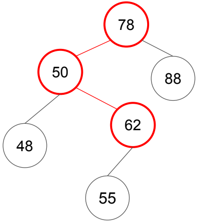
Exemplo de Inserção



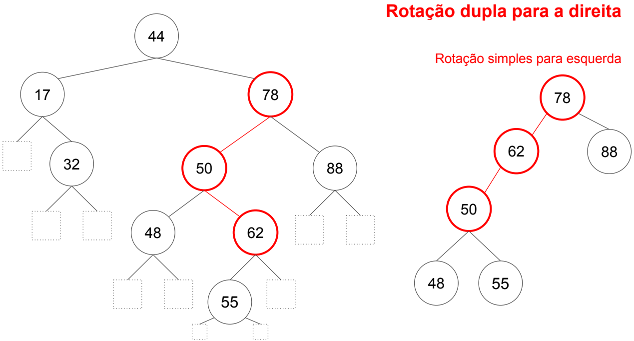
Exemplo de Inserção



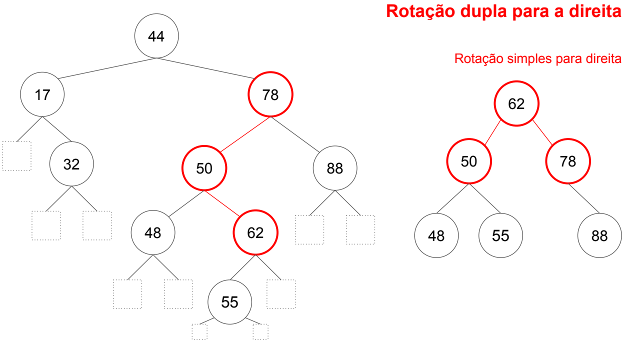
Rotação dupla para a direita



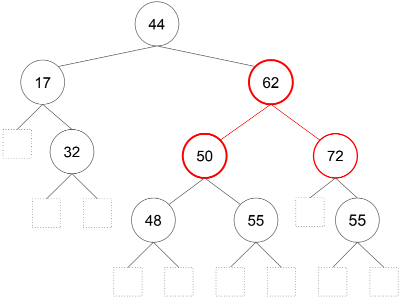
Exemplo de Inserção



Exemplo de Inserção

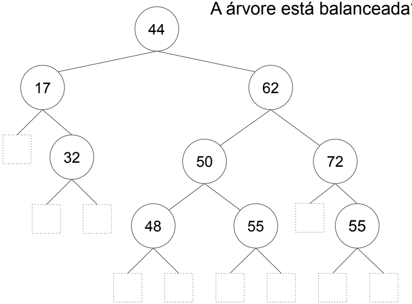


Exemplo de Inserção

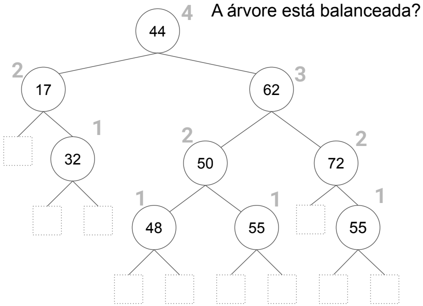


Exemplo de Inserção

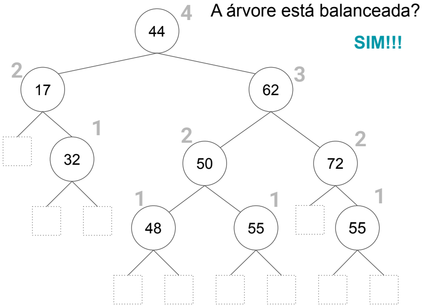
A árvore está balanceada?



Exemplo de Inserção



Exemplo de Inserção



Inserção de um nó em uma Árvore AVL

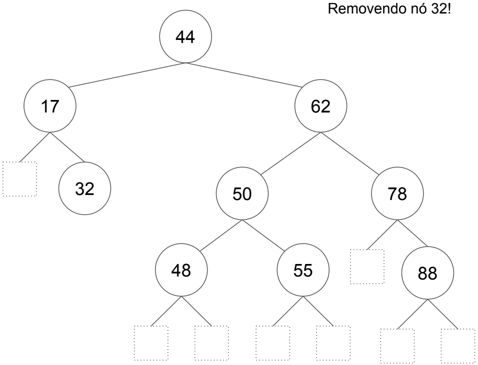
```c
int Insere(TNo** ppRaiz, Registro* x) {
    if (*ppRaiz == NULL) {
        *ppRaiz = (TNo*)malloc(sizeof(TNo));
        (*ppRaiz)->Reg = *x;
        (*ppRaiz)->pEsq = NULL;
        (*ppRaiz)->pDir = NULL;
        return 1;
    } else if ((*ppRaiz)->Reg.chave > x->chave) {
        if (Insere(&(*ppRaiz)->pEsq, x)) {
            if (Balanceamento(ppRaiz))
                return 0;
            else
                return 1;
        }
    } else if ((*ppRaiz)->Reg.chave < x->chave) {
        if (Insere(&(*ppRaiz)->pDir, x)) {
            if (Balanceamento(ppRaiz))
                return 0;
            else
                return 1;
        } else
            return 0;
    } else
        return 0; /* valor jah presente */
}
```

Análise Implementação da Inserção

- Cálculo de fatores de balanceamento:
  - Custo: $O(\log n)$??
- Como melhorar?
  - Cada nó:
    - Fator de balanceamento.
    - Profundidade x Altura.
  - Problema: atualizar dados durante rotações.

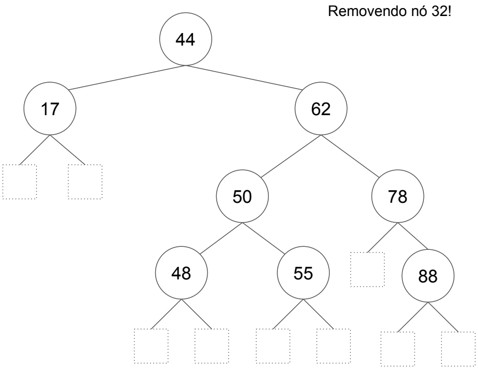
Remoção em uma Árvore AVL

- Remoção começa como em uma árvore binária de busca.
  $\Rightarrow$ pode causar desbalanceamento.



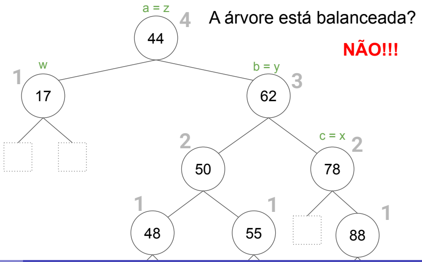
Remoção em uma Árvore AVL

- Remoção começa como em uma árvore binária de busca.
  - $\Rightarrow$ pode causar desbalanceamento.

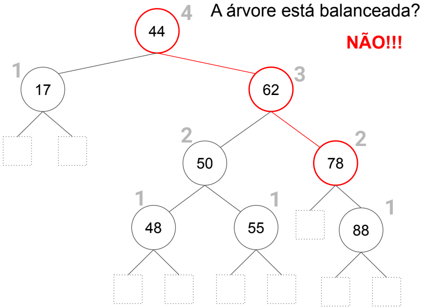


Rebalanceamento Após uma Remoção

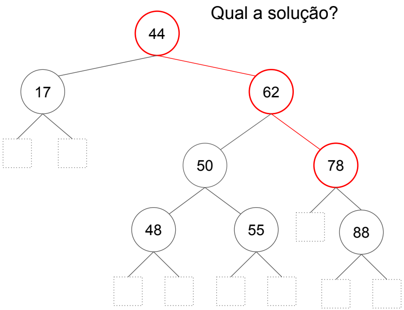
- Seja z o primeiro nó desbalanceado encontrado acima de w.
- Seja y o filho de z com maior altura, e x o filho de y com maior altura.
- Executar *Balanceamento*(x) para rebalancear z.
- Pode ocorrer desbalanceamento de outro nó acima ⇒ continuar verificação de balanceamento até à raiz.



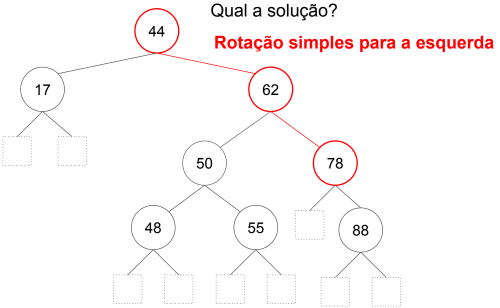
Exemplo de Remoção



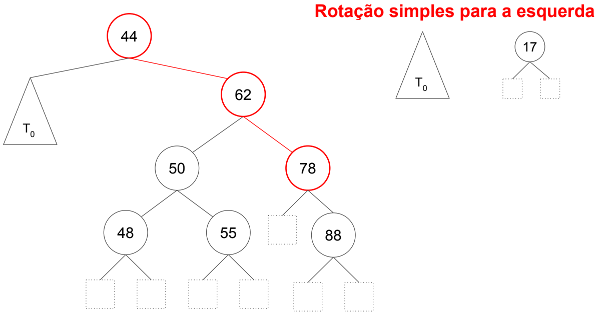
Exemplo de Remoção



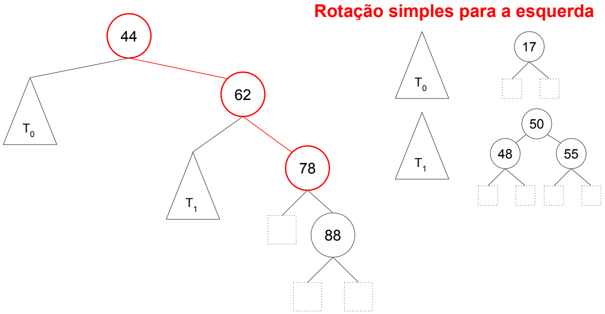
Exemplo de Remoção



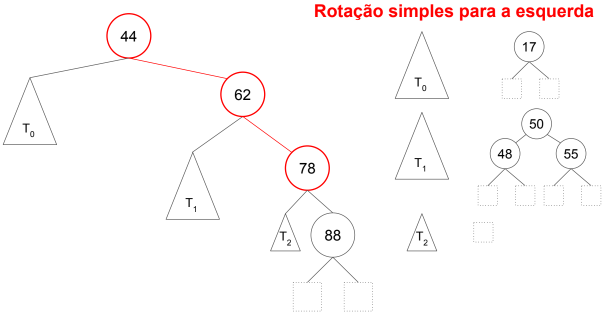
Exemplo de Remoção



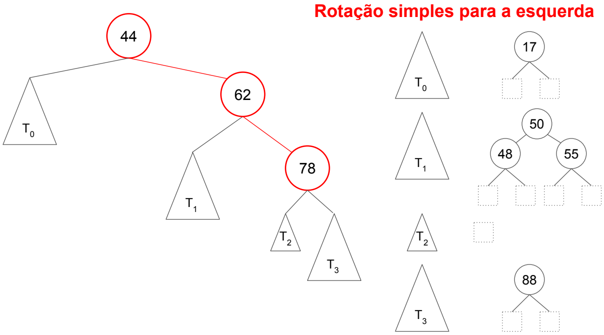
Exemplo de Remoção



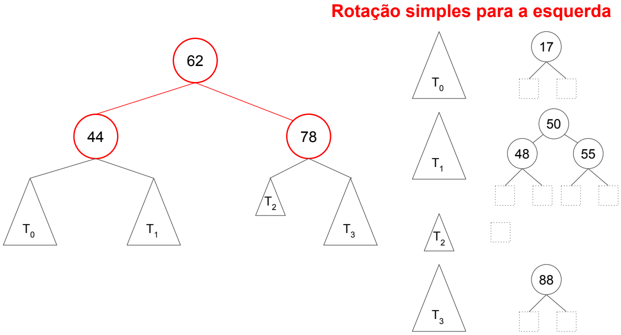
Exemplo de Remoção



Exemplo de Remoção

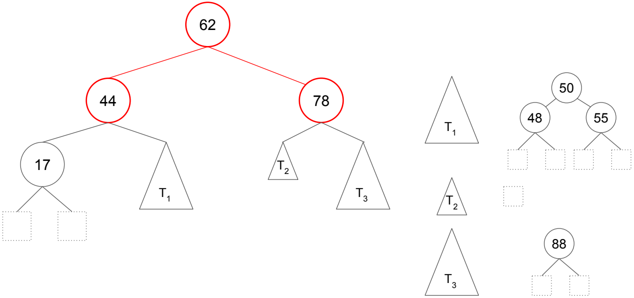


Exemplo de Remoção

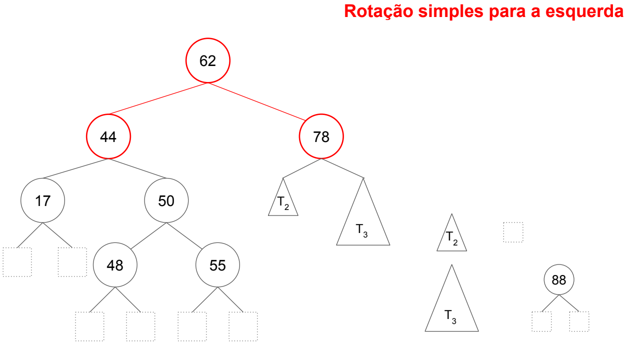


Exemplo de Remoção

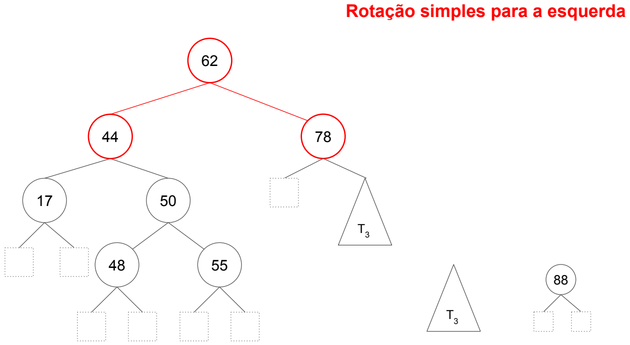
Rotação simples para a esquerda



Exemplo de Remoção

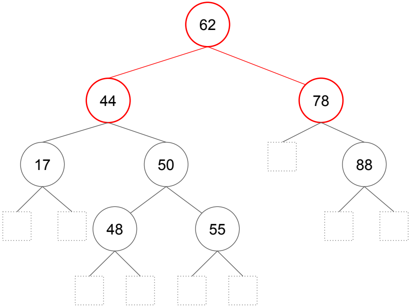


Exemplo de Remoção



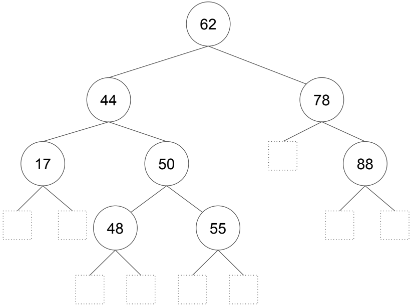
Exemplo de Remoção

Rotação simples para a esquerda



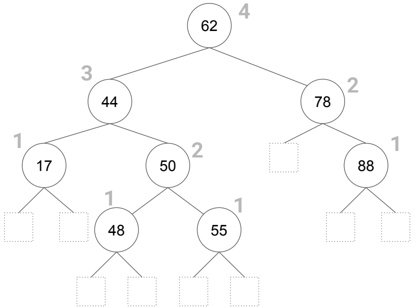
Exemplo de Remoção

A árvore está balanceada?



Exemplo de Remoção

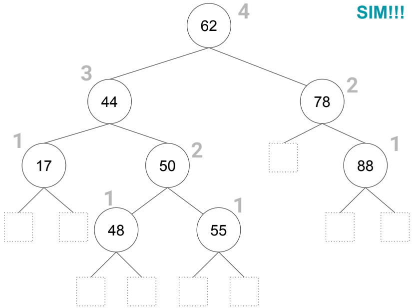
A árvore está balanceada?



Exemplo de Remoção

A árvore está balanceada?

SIM!!!



Remoção de um nó em uma Árvore AVL

```c
int Remove (TNo** ppRaiz,Registro* pX) {
  if (*ppRaiz == NULL)
    return 0;
  else if ( (*ppRaiz)->Reg.chave = pX->chave) {
    *pX = (*ppRaiz)->Reg;
    Antecessor(ppRaiz,&((*ppRaiz)->pEsq));
    Balanceamento(ppRaiz);
    return 1;
  } else if ( (*ppRaiz)->Reg.chave > pX->chave ) {
    if (Remove((*ppRaiz)->pEsq,pX)) {
      Balanceamento(ppRaiz);
      return 1;
    } else
      return 0;
  } else {
     /* código para sub-árvore direita */
  }
}
```

Conteúdo

Complexidade de Tempo para Árvores AVL

- Uma única reestruturação é $O(1)$.
  - Usando uma árvore binária implementada com estrutura ligada.
- Pesquisa é $O(\log n)$.
  - Altura de árvore é $O(\log n)$, não necessita reestruturação.
- Inserir é $O(\log n)$.
  - Busca inicial é $O(\log n)$.
  - Reestruturação para manter balanceamento é $O(\log n)$.
- Remove é $O(\log n)$.
  - Busca inicial é $O(\log n)$.
  - Reestruturação para manter balanceamento é $O(\log n)$.

Verifica se uma árvore é AVL

```c
int EhArvoreArvl(TNo* pRaiz) {
    int fb;
    if (pRaiz == NULL)
        return 1;
    if (!EhArvoreArvl(pRaiz->pEsq))
        return 0;
    if (!EhArvoreArvl(pRaiz->pDir))
        return 0;
    fb = FB (pRaiz);
    if ((fb > 1) || (fb < -1))
        return 0;
    else
        return 1;
}
```

Conteúdo

Aplicações

- Para que servem as Árvores Binárias?
- Exemplos de aplicações:
  - Redes de Comunicação de Dados:
    - Envio de pacotes ordenados e/ou redundantes.
  - Codificação de Huffman.
    - Compressão e Descompressão de arquivos.

Redes de Comunicação

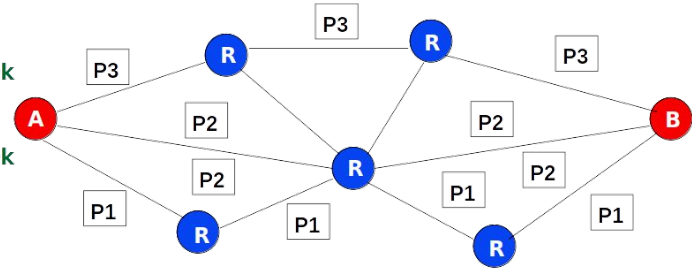
- A maioria dos protocolos de comunicação fragmenta as mensagens em pacotes que são numerados e enviados através da rede.
- Não há garantia da chegada em ordem dos pacotes.
- Perdas de pacotes geram novos envios e estes podem causar duplicatas dos mesmos

Reconstrução da Mensagem

- Como reconstruir a mensagem corretamente?
  - Descartar os pacotes repetidos.
  - Ordenar os pacotes.
- Como implementar tal algoritmo?
  - Utilizando Árvores Binárias

Exemplo

P1 Ok
P2 ?
P3 Ok



Ordem de Chegada:

P3 P1 P2

Confirmação de envio: P1 e P3.

Reenvio de P2.

Problemas: ordens e redundância dos pacotes

Redes de Comunicação - Algoritmo

- O primeiro pacote é colocado na raiz da árvore. Cada pacote sucessivo é comparado com o da raiz.
- Se for igual, descarta-se a réplica. Se for menor ou maior, percorre-se os lados esquerdo ou direito da árvore.
- Sub-árvore vazia implica inserção do novo pacote.
- Sub-árvore não vazia implica comparação dos pacotes com a mesma.

Redes de Comunicação - Problemas resolvidos?

Problema da ordenação.

- A ordenação dos pacotes pode ser feita trivialmente com apenas uma chamada ao método *inOrder*() da árvore binária.

Problema da redundância.

- Solucionado com o algoritmo de inserção na árvore, visto que o pacote, antes de ser inserido, é comparado com os demais que já se encontram na árvore binária.

Codificação de Huffman

- Algoritmo utilizado para comprimir arquivos.
- Todo o algoritmo é baseado na criação de uma Árvore Binária.
- Programas como **Winzip** e **WinRAR** utilizam este algoritmo.
- Criado por David Huffman em 1952.

Códigos e Caracteres

- Caracteres são letras, números e símbolos.
- Códigos são sequências de bits que podem representar de maneira ÚNICA um caractere.
- **b** bits para representar **c** caracteres: $c = 2^b$.

ASCII (7 bits)	Extended ASCII (8 bits)
$2^7 = 128$ caracteres	$2^8 = 256$ caracteres

Codificação de Huffman - Como comprimir arquivos?

- No código ASCII, todos os caracteres têm um número fixo de bits.
- Números variáveis de bits implica menor capacidade de armazenamento.
- Associações com bits variáveis podem comprimir consideravelmente o arquivo.
- Como comprimir arquivos desta maneira?
- Utilizando a Codificação de Huffman!

Codificação de Huffman - Exemplo

- Considere o arquivo com o seguinte texto:

  $\boxed{\text{AAAAAAAAAABBBBBBBBCCCCCCDDDDDEE}}$

- Frequências: $A = 10$; $B = 8$; $C = 6$; $D = 5$; $E = 2$.
- Construção da Árvore Binária.
- Comparação do número de bits
  - Tamanho Fixo (8 bits) $\Rightarrow$ Total $= 248$ bits.
  - Tamanho Variável $\Rightarrow$ Total $= 69$ bits

Codificação de Huffman - Compressão

- Depois da geração da árvore, o arquivo é percorrido novamente e cada caractere do arquivo é substituído pelo código binário contido na árvore, gerando uma cadeia de bits.
- Criação da tabela de caracteres e códigos binários.
- O que é armazenado?
  - Cadeia de bits gerada.
  - Tabela de caracteres e códigos.

Codificação de Huffman - Descompressão

- Regeneração da árvore binária através da tabela de caracteres e códigos.
- A cadeia de bits é percorrida e, à medida que uma sub-cadeia é encontrada na tabela de caracteres e códigos, a mesma é substituída pelo caractere correspondente.

Conteúdo

Conclusão

- As árvores binárias são uma das estruturas de dados mais importantes devido a grande aplicabilidade das mesmas.
- A maioria dos algoritmos das árvores binárias são de simples entendimento, facilitando sobremaneira o desenvolvimento de sistemas.

Tabela Hash

Conteúdo

Bibliografia

Os conteúdos deste material, incluindo figuras, textos e códigos, foram extraídos ou adaptados de:

Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Clifford.
Introduction to Algorithms.
The MIT Press, 2011.

Exercício

- Dada a sequência de números: `10, 20, 5, 8, 12, 22, 23, 24, 11, 13, 18`.
- Mostre (desenhe) uma árvore AVL após a inserção de cada um dos elementos acima.
- Mostre como ficará a árvore criada após a remoção dos seguintes elementos na seguinte ordem: `22, 11, 5, 10`.