Árvores Binárias

Enzo Tiriba Appi matricula: 1801130023

Uma árvore é um tipo de estrutura de dados que consiste em nós e arcos.

Ao contrário das árvores naturais, essas árvores são retratadas de cabeça para baixo, com a **raiz na parte superior** e as **folhas (nós terminais)** na parte inferior. (**Figura 1**).

A **raiz** é um nó que não possui **pai (nó hierarquicamente anterior)**, ou seja, pode ter, apenas, **nós filhos (nó hierarquicamente posterior)**.

Folhas, por outro lado, **não têm filhos**, ou melhor, seus filhos são estruturas vazias. (**Figura 1**).

Cada nó deve ser acessível a partir da raiz, através de uma sequência única de **aresta**, chamada de **caminho**. (**Figura 1**).

O número de arestas em um caminho é chamado o **comprimento do caminho**. (**Figura 1**).

O **nível de um nó** é o comprimento do caminho da raiz para o nó mais 1, que é o número de nós no caminho. (**Figura 1**).

A **altura** de um árvore que não está vazia, é o nível máximo de um nó na árvore. Se ela está vazia, sua altura é 0 (por definição). Se um único nó existe em sua estrutura, é uma árvore da altura 1. Obs: Esse é o único caso em que um nó é, ao mesmo tempo, raiz e folha.

O nível de um nó deve estar entre 1 (o nível da raiz) e a altura da árvore, que no caso extremo é o nível da única folha em uma árvore degenerada semelhante a uma lista vinculada.

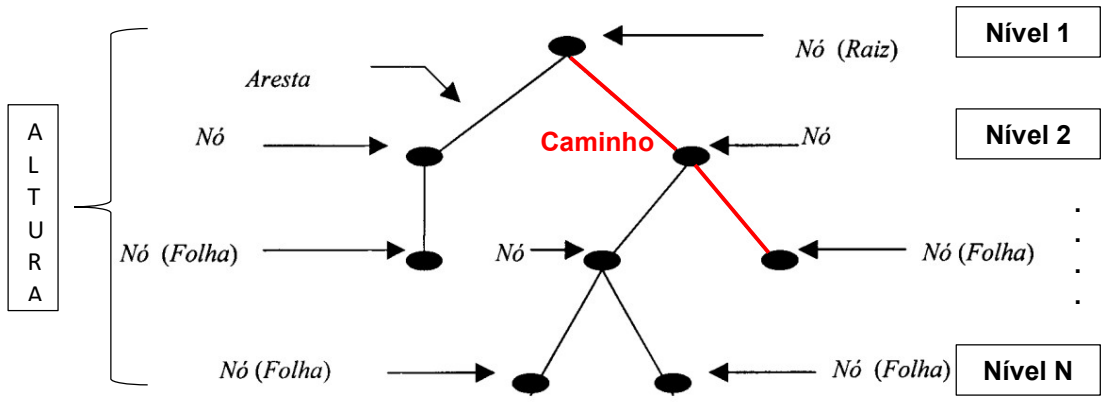

Figura 1: Exemplificação da arquitetura básica da Arvóre Binária.
Fonte: editado de Google imagens

Segundo Drozdek (2013), na estrutura da árvore binária, cada nó pai pode ter, no máximo, dois nós filhos (permite-se, por algumas vezes, vazios). E, cada filho é designado como esquerdo ou direito. (**Figura 1**).

Segundo Drozdek (2013) se na Árvore Binária o número de nós em cada nível (i + 1) for igual a 2^t, ela é referida como uma Árvore Binária Completa.

Podemos entender subárvore como a árvore formada pela estrutura similar à original, naquele trecho, é claro. Desde que o nó raiz dessa estrutura,

seja filho de um nó da árvore original. Podendo ser categorizado como: subárvore esquerda, quando o nó raiz da subárvore é o filho esquerdo do nó antecessor, na árvore original ou subárvore direita, quando o nó raiz dessa estrutura,é o filho direito do nó antecessor, na árvore original. (**Figura 2**)

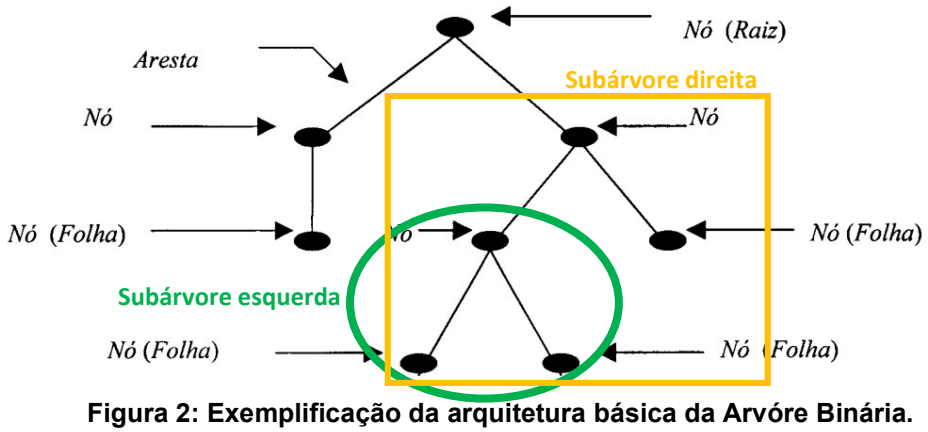

Figura 2: Exemplificação da arquitetura básica da Arvóre Binária.
Fonte: editado de Google imagens

Pode-se afirmar que para todo o nó existente em uma Árvore Binária, existe uma **chave**, que designa o conteúdo armazenado naquele nó.

Para a inserção de dados na estrutura seguimos os seguintes passos: existe um nó hierárquico principal, e, que fica ao centro (raiz); A partir do qual, se inicia a árvore ou subárvore. Exemplificaremos a inserção na estrutura, partindo-se da premissa de que os valores chaves, são números:

- O nó pai é **visitado** afim de que consigamos ver o valor nele armazenado e, sigamos com os próximos procedimentos descritos. * Vale ressaltar que essa visitação aos nós, pode ocorrer uma ou mais de uma vez.*
- Os nós que contenham o valores chave menores que seus antecessores hierárquicos (pai), são “encaminhados”/armazenados para a esquerda, do mesmo antecessor, em um nível hierárquico posterior.
- Os nós que contenham o valores chave maiores que seus antecessores hierárquicos, são “encaminhados”/armazenados para a direita, do mesmo antecessor, em um nível hierárquico posterior.
- Essas designações são realizadas até atingirem-se os nós folhas.

* Percorrer uma árvore visitando cada nó uma única vez gera uma sequência linear de nós, e então passa a ter sentido falar em sucessor e predecessor de um nó segundo um determinado percurso. Há três maneiras recursivas de se percorrer árvores binárias:

- **Travessia em Pré-Ordem**: nessa travessia, deve-se visitar, primeiramente, o nó raiz. Posteriormente, acessamos o filho a esquerda da raiz, ou seja na subárvore da esquerda. Nela, visita-se o seu próprio nó raiz, novamente, e, realizamos o mesmo procedimento de acesso ao nó filho a esquerda dessa subárvore acessada. Continuando, portanto, o caminhamento para as subárvores da

esquerda, em sequência. Faz-se isso até a que atinjamos o nó folha. Nesse momento, visita-se a subárvore da direita de todos os nós visitados, anteriormente.

Ex de trecho de código:

```
void preordem (No* a) {
    if (a!=NULL)){
        printf("%d ",a->info); /* mostra raiz */
        preordem(a->esq); /* mostra sae */
        preordem(a->dir); /* mostra sad */
    }
}
```

Para o trecho de código anterior, se imaginarmos uma Árvore Binária á seguir:

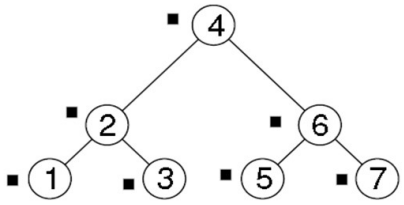

Figura 3: Exemplo de Árvore Binária preenchida.
Fonte: Google imagens

O que será executado em sequência:

1) Executa a função preordem(No* a);
Impressão do valor do nó raiz == 4.
1.1) Executa a função preordem(a->esquerda);
Impressão do valor do nó == 2.
1.1.1) Executa a função preordem(a->esquerda);
Impressão do valor do nó == 1.
1.1.1.1) Executa a função preordem(a->esquerda);
Termina a execução da função previa, pois o *if* não é atendido;
1.1.1.2) Executa a função preordem(a->direita);
Termina a execução da função previa, pois o *if* não é atendido;
1.1.2) Executa a função preordem(a->direita);
Impressão do valor do nó == 3.
1.1.2.1) Executa a função preordem(a->esquerda);
Termina a execução da função previa, pois o *if* não é atendido;
1.1.2.2) Executa a função preordem(a->direita);
Termina a execução da função previa, pois o *if* não é atendido;
1.2) Executa a função preordem(a->direita);
Impressão do valor do nó == 6.
1.2.1) Executa a função preordem(a->esquerda);

Impressão do valor do nó == 5.
1.2.1.1) Executa a função preordem(a->esquerda);
Termina a execução da função previa, pois o *if* não é atendido;
1.2.1.2) Executa a função preordem(a->direita);
Termina a execução da função previa, pois o *if* não é atendido;
1.2.2) Executa a função preordem(a->direita);
Impressão do valor do nó == 7.
1.1.2.1) Executa a função preordem(a->esquerda);
Termina a execução da função previa, pois o *if* não é atendido;
1.1.2.2) Executa a função preordem(a->direita);
Termina a execução da função previa, pois o *if* não é atendido;

2) FINALIZA O ALGORITMO;

Logo, a impressão final, ficou sequênciada como: 4, 2, 1, 3, 6, 5, 7.

- **Travessia em In-Ordem**: nessa travessia, visita-se, primeiramente, a subárvore da esquerda dos nós até até que se alcance o nó folha. Nesse momento, passa-se a re-visitar os nós passados, desvia-se para o caminho à direita. Visitando-se a subárvore da direita até que o nó folha seja alcançado. Finalmente, para cada nó que visitou a subárvore da esquerda volta a visita o nó e desce a sub árvore da direita.

  Ex de trecho de código:

```
void emordem (No* a) {
    if (a != NULL) {
        emordem(a->esq);
        printf("%d ",a->info);
        emordem(a->dir);
    }
}
```

  Para o trecho de código anterior, se imaginarmos uma Árvore Binária á seguir:

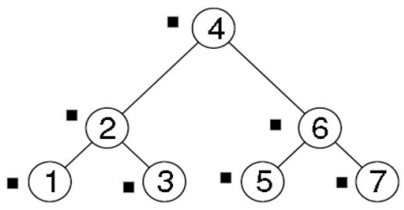

Figura 4: Exemplo de Árvore Binária preenchida.
Fonte: Google imagens

  O que será executado em sequência:

1) Executa a função emordem(No* a);

1.1) Executa a função emordem(a->esquerda);
1.1.1) Executa a função emordem(a->esquerda);
1.1.1.1) Executa a função emordem(a->esquerda);
Termina a execução da função previa, pois o *if* não é atendido;
Impressão do valor do nó == 1.
1.1.1.2) Executa a função emordem(a->direita);
Termina a execução da função previa, pois o *if* não é atendido;
Impressão do valor do nó == 2.
1.1.2) Executa a função emordem(a->direita);
1.1.2.1) Executa a função emordem(a->esquerda);
Termina a execução da função previa, pois o *if* não é atendido;
Impressão do valor do nó == 3.
1.1.2.2) Executa a função emordem(a->direita);
Termina a execução da função previa, pois o *if* não é atendido;
Impressão do valor do nó == 4.
2) Executa a função emordem(a->direita);
1.2.1) Executa a função emordem(a->esquerda);
1.2.1.1) Executa a função emordem(a->esquerda);
Termina a execução da função previa, pois o *if* não é atendido;
Impressão do valor do nó == 5
1.2.1.2) Executa a função emordem(a->direita);
Termina a execução da função previa, pois o *if* não é atendido;
Impressão do valor do nó == 6.
1.2.2) Executa a função emordem(a->direita);
1.1.2.1) Executa a função emordem(a->esquerda);
Termina a execução da função previa, pois o *if* não é atendido;
Impressão do valor do nó == 7.
1.1.2.2) Executa a função emordem(a->direita);
Termina a execução da função previa, pois o *if* não é atendido;
3) FINALIZA O ALGORITMO;

Logo, a impressão final, ficou sequênciada como: 1, 2, 3, 4, 5, 6, 7.

- **Travessia em Pós-Ordem**: nesse tipo de travessia, porcorre-se descendo a subárvore da esquerda dos nós até que o folha seja alcançado. Posteriormente, percorre-se, também,a subárvore da direita dos nós até que o nó folha seja alcançado. Depois de visitarem-se todas as subárvores dos nós, re-visitam-se, os mesmos, fazendo tratamento adequado.

Ex de trecho de código:

```
void posordem (No* a) {
  if (a!=NULL){
    posordem(a->esq); /* mostra sae */
    posordem(a->dir); /* mostra sad */
    printf("%d ",a->info); /* mostra raiz */
  }
}
```

Para o trecho de código anterior, se imaginarmos uma Árvore Binária á seguir:

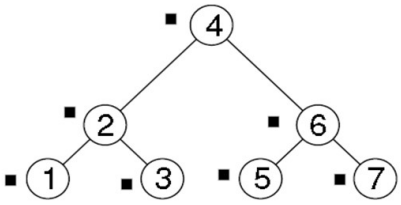

Figura 5: Exemplo de Árvore Binária preenchida.
Fonte: Google imagens

O que será executado em sequência:

1) Executa a função posrdem(No* a);
  1.1) Executa a função preordem(a->esquerda);
    1.1.1) Executa a função preordem(a->esquerda);
      1.1.1.1) Executa a função preordem(a->esquerda);
        Termina a execução da função previa, pois o *if* não é atendido;
      1.1.1.2) Executa a função posordem(a->direita);
        Termina a execução da função previa, pois o *if* não é atendido;
      Impressão do valor do nó == 1.
    1.1.2) Executa a função posordem(a->direita);
      1.1.2.1) Executa a função posordem(a->esquerda);
        Termina a execução da função previa, pois o *if* não é atendido;
      1.1.2.2) Executa a função posordem(a->direita);
        Termina a execução da função previa, pois o *if* não é atendido;
      Impressão do valor do nó == 3.
    Impressão do valor do nó == 2.
2) Executa a função posordem(a->direita);
    1.2.1) Executa a função posordem(a->esquerda);
      1.2.1.1) Executa a função posordem(a->esquerda);
        Termina a execução da função previa, pois o *if* não é atendido;
      1.2.1.2) Executa a função posordem(a->direita);
        Termina a execução da função previa, pois o *if* não é atendido;
      Impressão do valor do nó == 5.
    1.2.2) Executa a função posordem(a->direita);

1.1.2.1) Executa a função posordem(a->esquerda);
Termina a execução da função previa, pois o *if* não é atendido;
1.1.2.2) Executa a função posordem(a->direita);
Termina a execução da função previa, pois o *if* não é atendido;
Impressão do valor do nó == 7.
Impressão do valor do nó == 6.
Impressão do valor do nó == 4.
3) FINALIZA O ALGORITMO;

Logo, a impressão final, ficou sequênciada como: 1, 3, 2, 5, 7, 6, 4.

As Árvores Binárias pode existir de várias formas, mas uma em especial, recebe o nome de Árvore Binária balanceada, e, segundo Drozdek (2013), é assim caracteriza pois a diferença de altura de ambas as subárvores de qualquer nó da árvore é zero ou um. (**Figura 6a**)

Ainda segundo Drozdek (2013), resumidamente, uma Árvore Binária em cujos número de nós totais (n) (folhas, raiz e nós internos) e altura (h) é dita balanceada se: $2^{h-1} \leq n < 2^h$. Caso essa condição não seja satisfeita, é dita desbalanceada. (**Figuras 6b e c**).

Por exemplo: uma árvore binária de altura 4 pode ter de 8 a 15 nós (de 1 a 8 nós folhas) para estar balanceada.

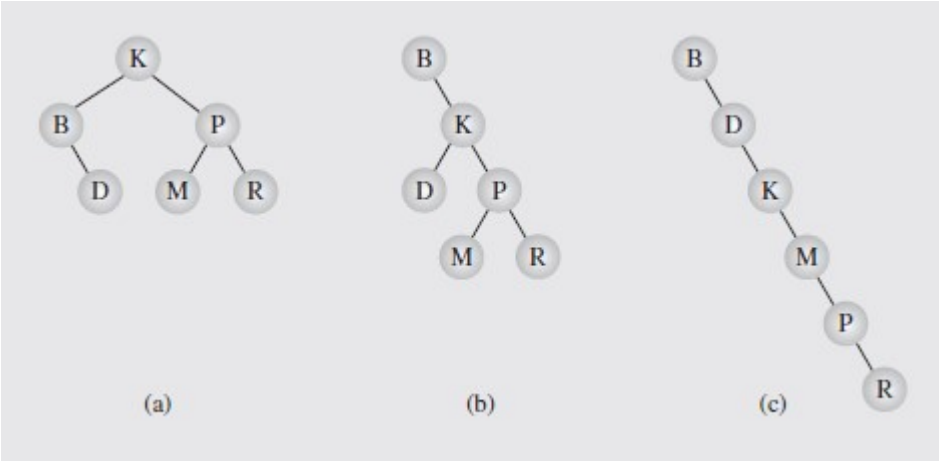

Figura 6: a) Exemplo de Árvore Binária balanceada. b) a) Exemplo de Árvore Binária desbalanceada. d) a) Exemplo de Árvore Binária desbalanceada.
Fonte: Drozdek (2013).

O Balanceamento das árvores binárias objetiva, em teoria, evitar casos degenerados, garantir o tempo logarítmico para todas as operações. Porém, requerer algoritmos mais elaborados para inserção e eliminação.

De modo geral, os nós das árvores balanceadas armazenam mais informações.

Na maioria dos casos, árvores desequilibradas são implementações ruins de estrutura, porque as árvores binárias balanceadas, em contrapartida, são mais compactas e possuem uma distância máxima menor da raiz aos nós das folhas. Entretanto, depende do objetivo, especialmente, de como passa-se do

nó raiz para os nós folha, e, que decisão é tomada ao escolher se deseja ir para a subárvore esquerda ou direita.

O balanceamento de uma árvore binária pode mudar seu significado, portanto, nem sempre é possível equilibrar.

Em algumas aplicações, por exemplo, o balanceamento de uma árvore binária pode ser possível, mas, geralmente, é adiado para um horário mais conveniente, em vez de impor o equilíbrio o tempo todo.

Por exemplo, o balanceamento pode ser feito após n modificações, ou, uma vez por intervalo de tempo, e não após cada modificação.

Existem implementações binárias de árvores que garantem que a árvore seja equilibrada (ou quase equilibrada) o tempo todo, por exemplo, árvores vermelho-pretas ou árvores AVL.

Nas tarefas ususais de manuseamento das Árvores Binárias, geralmente adicionam-se e removem-se elementos que compõem a mesma. Podem ser tecnicamente simples, ou, mais elaboradas, no caso das deleções, à depender da posição do nó-dado, na estrutura. No caso, se ele é um nó folha (sem filhos) ou nó-dado, pai de 1 ou mais filhos.

Ilustrar-se-á os procedimentos de deleção de nós de uma árvore conforme sua complexidade geral.

- **Remoção de um nó folha**: Segundo Drozdek (2013), esse é o caso mais simples, pois o nó em questão não tem filhos. Logo, faz-se com que o ponteiro de seu pai, que aponta ao próprio nó a ser removido, receba o valor nulo, e, assim, o nó folha possa ser descartado por exclusão (liberação da memória alocada para aquele nó). (**Figura 7**)

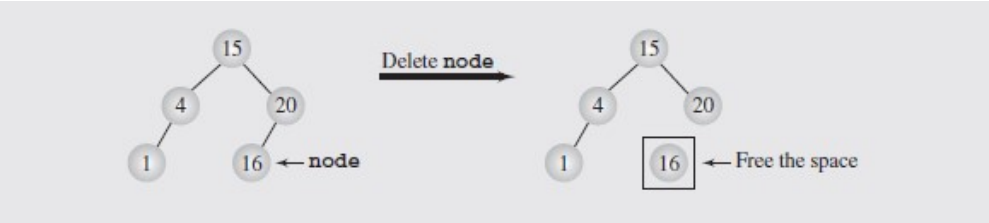

Figura 7: Exemplo de remoção de um nó folha em uma Árvore Binária.
Fonte: Drozdek (2013).

- **Remoção de um nó-dado que tenha 1 nó-filho**: nesse caso, faz-se o ponteiro dos pais do nó-dado a ser removido, apontar para o nó filho do alvo da remoção. Dessa forma, os filhos do nó-dado, a ser removido, são alçados em um nível hierárquico, e, o nó-dado pode ser removido por exclusão (liberação da memória alocada para aquele nó). (**Figura 8**)

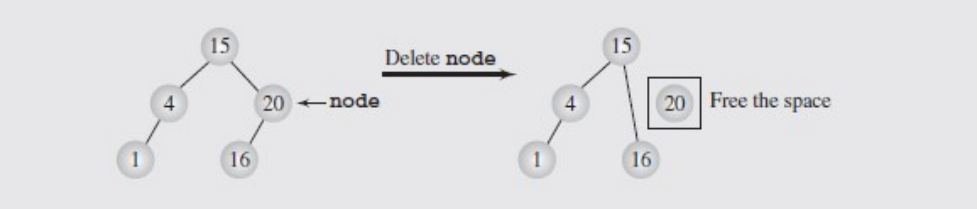

Figura 8: Exemplo de remoção de um nó-dado que tenha um filho, em Árvores Binárias.
Fonte: Drozdek (2013).

Entretanto, quando o nó-dado que deseja-se remover tenha mais de 1 filho apenas, a tarefa aumenta em complexidade. Podendo ser usadas duas técnicas de remoção, exemplificadas, à seguir:

- **Remoção por *Merge*:** Segundo Drozdek (2013), esse tipo de técnica de remoção cria uma árvore, das duas subárvores do nó-dado, e, posteriormente, anexa ao pai do nó-dado.

  Pela estruturação das Árvores Binária, todas as valores-chaves da subárvore à direita são maiores que as da subárvore esquerda. Logo, encontra-se na subárvore esquerda o nó com a maior valor-chave, em nosso exemplo à seguir, é o número 12, e, usamo-no como o nó-pai da subárvore direita, em nosso exemplo o 30 é o nó raiz dessa subárvore, ao nó que deseja-se remover. Fazemos com que o ponteiro do nó folha de valor mais alto da subárvore esquerda (12 em nosso exemplo) aponte para o nó raiz (30 em nosso exemplo) da subárvore direita, podendo-se, por fim, deletar o nó desejado (por liberação da memória alocada pra ele. (**Figura 9 e 10**)

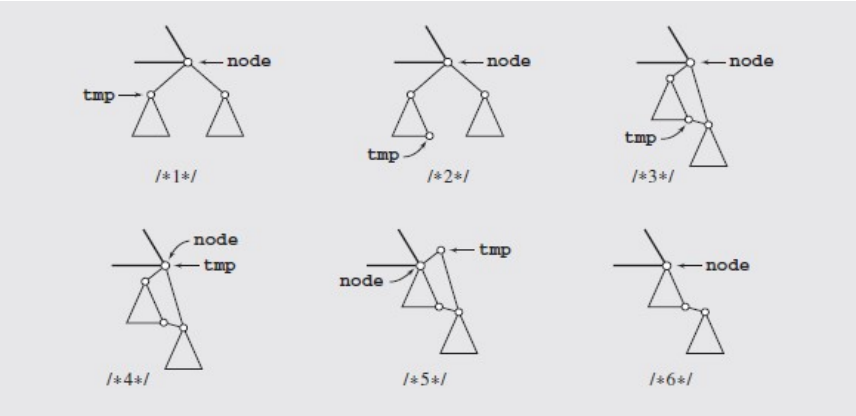

Figura 9: Exemplo dos procedimentos detalhados de Remoção por Merge.
Fonte: Drozdek (2013).

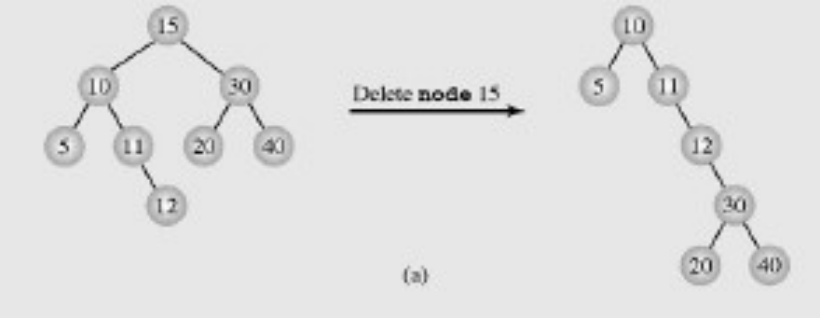

Figura 10: Exemplo prático de remoção por Merge, em Árvores Binárias. Escolhendo-se o maior valor da subárvore esquerda a receber a subárvore direita, do nó a ser deletado.
Fonte: Drozdek (2013).

Simetricamente, o nó-dado com a chave mais baixa pode ser encontrado na subárvore direita, e, portanto, podemos torná-lo pai da subárvore esquerda, do nó que deseja-se remover.

Remoção por Cópia: Segundo Drozdek (2013), essa técnica é realizada fazendo-se a substituição do nó-dado de valor chave que deseja-se exclur, por seu predecessor imediato. Facilmente encontrável, no nó-dado de valor chave mais alto, na subárvore esquerda, é o nó-dado mais à direita na subárvore esquerda.

Em seguida, o valor chave do nó-dado localizado substitui a chave a ser excluída. E, é aí que um dos dois casos mais simples (Rmeção do nó folha ou Remoção de um nó-dado com um filho) se faz útil.

Se o nó-dado mais à direita for um nó-folha, o primeiro caso se aplica. No entanto, se esse nó-dado tem um filho, o segundo caso é utilizado. (**Figura 11**)

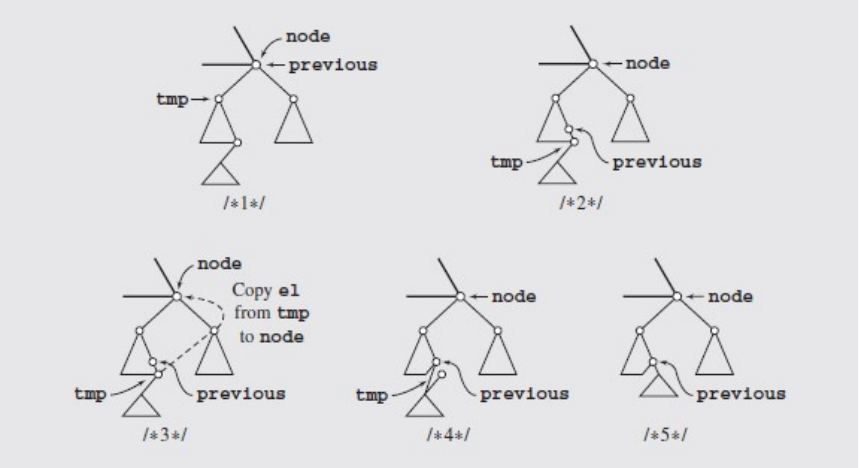

Figura 11: Exemplificação dos passos de utilização da técnica de Remoção por cópia.
Fonte: Drozdek (2013).

Referência Bibliográficas

DROZDEK, Adam. **Data Structure and Algorithms in C++.** 4th edition. ed. Cengage Learning: Boston, 2013.