Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital

Estruturas de Dados Clássicas: Árvore B

Bianca Medeiros, Gabriel Carvalho, Marina Medeiros, Vinicius de Lima

Estrutura de Dados Básica II

Trabalho da Terceira Unidade

28 de janeiro de 2025

Sumário

Capítulo 1

Ambiente computacional

Todos os testes de performance nesse trabalho foram realizados no seguinte sistema:

Software	Sistema Operacional	Arch Linux x86_64
	Kernel	Linux 6.12.8-arch1-1
	Gerenciador de Janelas	Hyprland (Wayland)
	Terminal	Ghostty 1.0.1
	Compilador de C++	clang 18.1.8
	Compilador de Rust	rustc 1.82.0
	Versão do Cargo	cargo 1.82.0
Hardware	CPU	AMD Ryzen 5 5500
	GPU	GeForce RTX 4060 Ti
	Driver da GPU	nvidia (proprietário) 565.77
	Memória RAM	31.24 GiB
	Armazenamento	SSD NVMe 2TB

Capítulo 2

Árvore B

Introdução

A Árvore B está inclusa na categoria da árvores autobalanceáveis tal qual as árvores AVL e Rubro-Negra, entretanto o que a diferencia destas são seus nós que podem armazenar mais de um valor chave. Idealizada por Rudolf Bayer e Edward Meyers McCreight, esta tem como fim trabalhar com grandes volumes de dados normalmente armazenados em memória secundária(na época, através de discos rígidos magnéticos).

Devido a isso, sua estratégia conceitual consiste em não carregar todos os dados na memória principal, apenas algumas páginas. E bem como é elucidado na pirâmide de hierarquia de memória, existe uma relação inversamente proporcional entre custo e capacidade de armazenamento de memórias de menor latência.

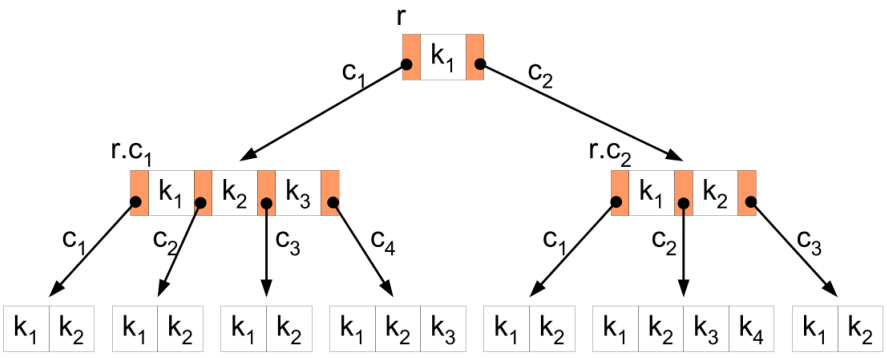
Sendo assim, quando se trabalha com um volume significativo de dados é inviável carregar tudo na memória principal. E é tendo isso em mente que a Árvore B carrega apenas partes específicas dos dados na memória, essas partes também são conhecidas como **páginas**. Sendo assim, essa abordagem visa diminuir a quantidade de operações de entrada e saída da memória secundária.

Ela se assemelha muito com a Árvore Rubro-Negra embora sua altura possa ser significativamente menor devido ao fato dela poder comportar mais de um valor chave por nó. Em suma, toda Árvore B deve ter as seguintes propriedades:

- Todas chaves de um nó devem estar ordenadas entre si

- Seja $t \in \mathbb{Z}$, chamamos `t` de **grau mínimo**
  - A raiz pode ter entre `1` e `2t` chaves
  - Qualquer outro nó deve ter entre `t` e `2t` chaves
  - Cada nó interno(que não é folha nem raiz) tem entre `t + 1` e `2t + 1` filhos
- Todo nó tem `n + 1` ponteiros, em que `n` é o número de chaves

Sintetizando, podemos compreender a Árvore B como algo semelhante à figura abaixo



Altura

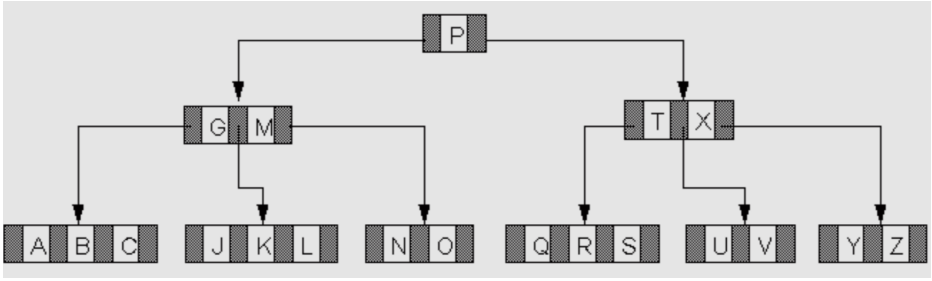
Embora a altura de uma Árvore B seja muito menor do que uma proporcional Rubro-Negra, essa é muito mais larga. E seja `h` sua altura e `n` sua quantidade de elementos, vale que

$$h \leq \log_t \frac{n+1}{2}$$

Busca

Em uma Árvore Binária de Busca qualquer a operação de busca é uma sequência escolhas binária(esquerda e direita) de qual caminho percorrer até que o elemento seja ou não encontrado. Já nas Árvores B, sua capacidade de armazenar mais de um valor chave por nó faz com que essa escolha binária vire uma escolha `n+1`-ária, sendo `n` o número de elementos do nó.

Entretanto, a lógica da busca tem a mesma essência da operação em qualquer BST. Serão feitas apenas as comparações necessárias e quando necessário, a busca seguirá para o nó filho. Ou seja, a busca vai iterar pelo conteúdo do nó(seja valor chave ou ponteiro) em ordem crescente, e sempre vai seguir pelo ponteiro que estiver entre os valores cujo intervalo contém o elemento buscado. Exemplificando:



Dada a árvore acima, caso desejamos buscar a chave **K** o processo seria o seguinte:

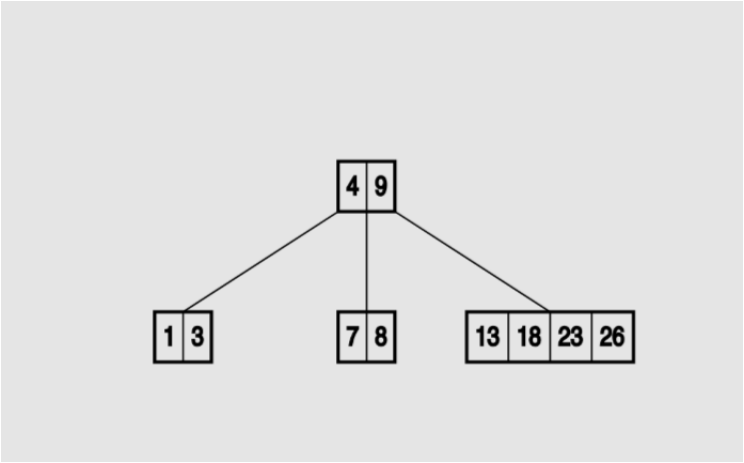
1. $K \leq P$
2. Como $K \in (-\infty; P]$, logo seguimos pelo ponteiro à esquerda de P
3. Atualmente em um ponteiro, mas como $K \in [G; M]$, a busca continuará por ele mesmo
4. $K == K$

Inserção

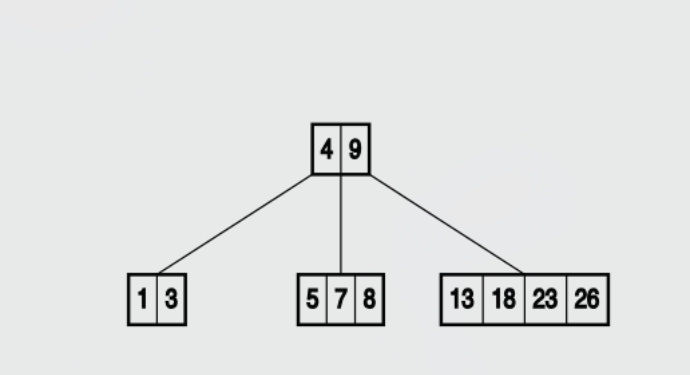
As particularidades da inserção em uma Árvore B acontecem para não violar as propriedades da quantidade de chaves permitidas em um nó. Nos casos em que a inserção causa essa violação, a árvore tem um processo de divisão de nós que ocorre de baixo para cima. Basicamente podemos resumir nos dois exemplos abaixo:

Inserção sem partição

Esse caso se trata de quando o nó em que se deve ser feita a inserção ainda pode comportar elementos. No exemplo a abaixo a árvore tem ordem 3, ou seja, um nó pode ter entre e 4 chaves por nó.

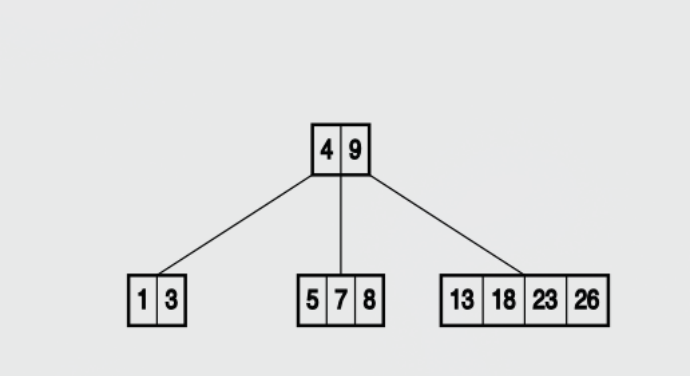


Perceba que após a inserção do 5 nenhuma mudança estrutural significativa aconteceu. E isso ocorre pois o nó que comporta o valor inserido ainda tinha capacidade para adicionar mais um valor nele.

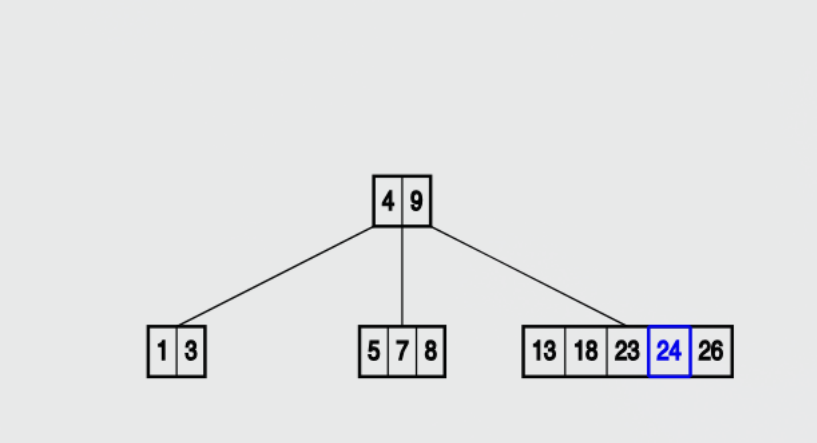


Inserção com partição

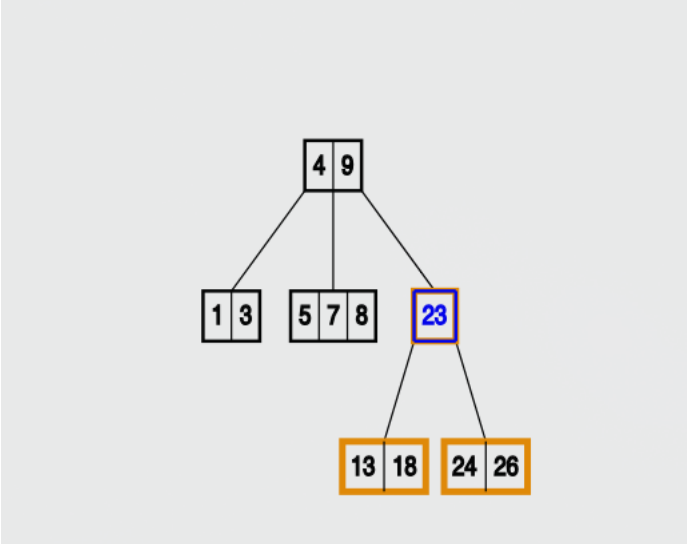
Ainda no mesmo exemplo, agora será feita a inserção de do valor 24, que deve ser alocado no nó mais a direita.



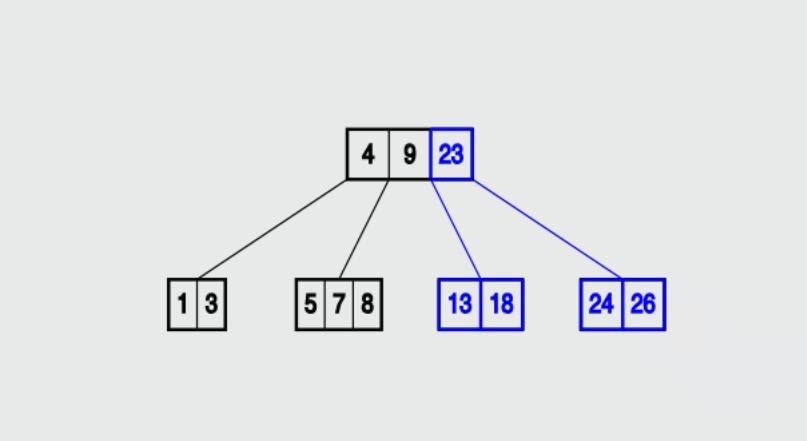
É feita a inserção do valor 24.



Note que o estrutura acima viola as propriedades da Árvore B de ordem 3, sendo assim ela escolhe o valor médio do nó para ser o pai e reparte a página em dois nós filhos.



Entretanto tal estrutura aumenta desnecessariamente a altura da árvore, isso pois o nó pai no "singleton"23 ainda pode comportar elementos. Sendo assim ele é alocado nele.



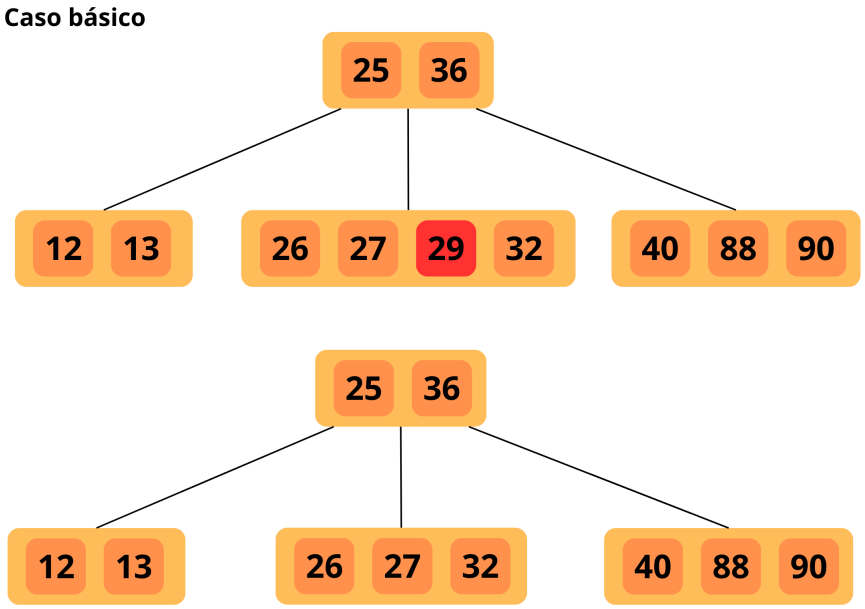
Note que essa inserção causou mudanças estruturais mais significativas na árvore. E vale a pena lembrar que não necessariamente esse processo de para no nó imediatamente acima, tal processo pode se repetir sequencialmente até a raiz da árvore.

Remoção

O processo de remoção é a operação mais complicada, tal complexidade é fruto da não violação das propriedades, bem como na inserção. As árvores usadas para exemplificação tem ordem 3. E a operação pode ser dividida em 3 cenários

Remoção em folha com mais de t elementos

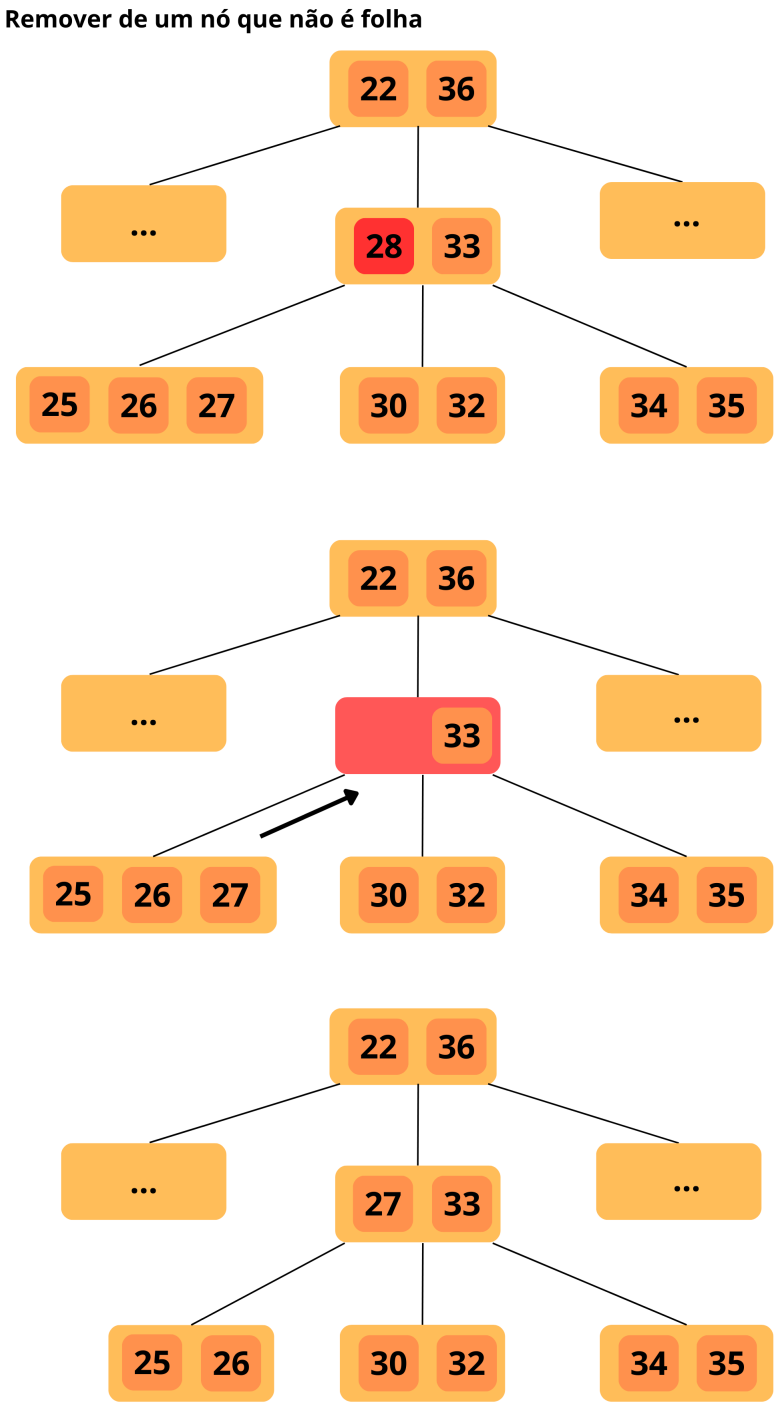
Se trata do melhor cenário, em que nenhuma mudança brusca vai acontecer na estrutura da árvore, isso pois como $n \geq (t+1)$, sendo `n` o número de chaves no nó e `t` a ordem.



Note que o elemento 29 foi excluído sem muito trabalho, isso pois seu nó tinha 4 elementos, e $4 \geq t + 1$.

Remoção em nó não folha

Este caso requer um pouco de atenção, pois é muito fácil violar $n \geq (t + 1)$ se tratanto de um nó que não é folha. Muitas vezes será necessária uma realocação de elementos. O que na maioria dos casos vai implicar em no maior elemento do nó filho subir para a posição do pai excluído. Repare que isso acontece no exemplo abaixo



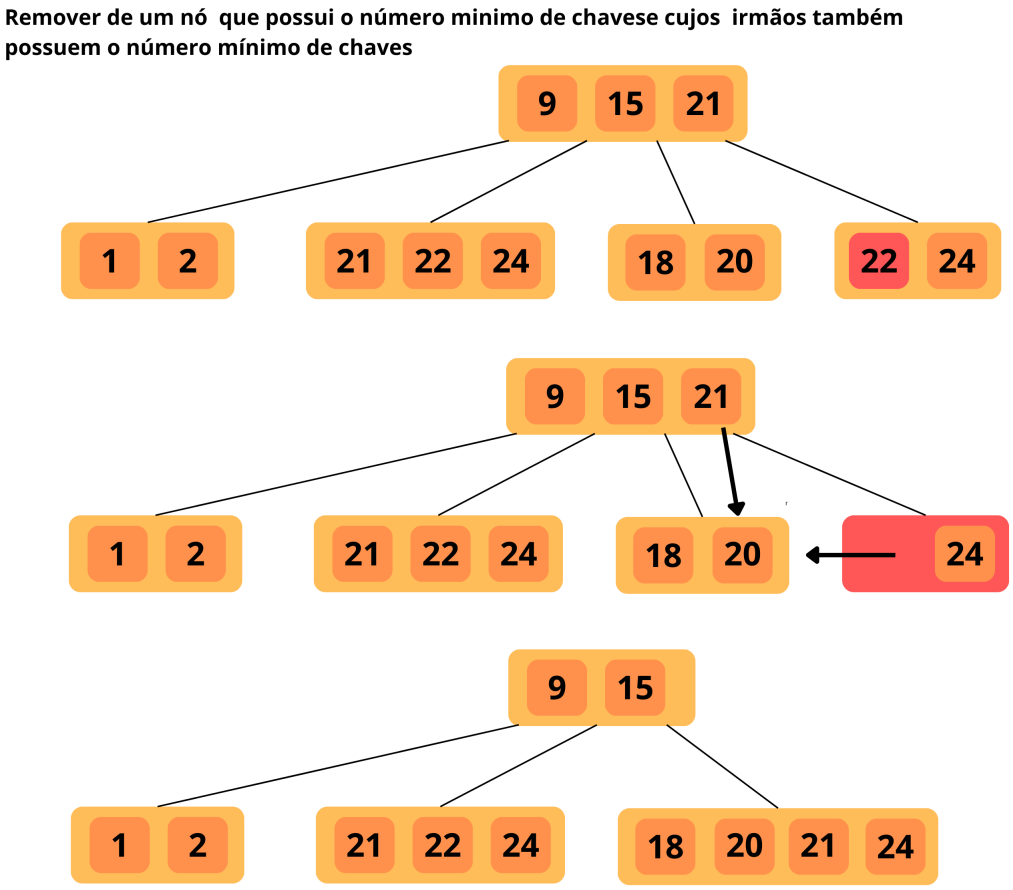
A exclusão do 28 acarreta na violação da propriedade citada, sendo assim se faz necessário a alocação de um elemento para ocupar a posição do 28. O que faz com que o 27(maior valor no nó filho que o ponteiro a esquerda do 28 apontava) tome a posição do elemento excluído no nó pai. Também é importante se atentar que tal cenário não faz com que o nó filho viole a propriedade dado a realocação de um de seus valores chave.

Remoção em folha com t elementos

Este último caso são particularizações do cenário acima, que na verdade dizem respeito a dois casos. Um em que os irmãos adjacentes somam mais de 2t chaves, e outro caso em que somam menos.

Exclusão em um nó cujos irmãos adjacentes somam menos de 2t

Sendo o alvo da exclusão o 22, perceba que o seu nó tem apenas um nó irmão adjacente, e este tem apenas 2 elementos, sendo assim $2 < 2t$.



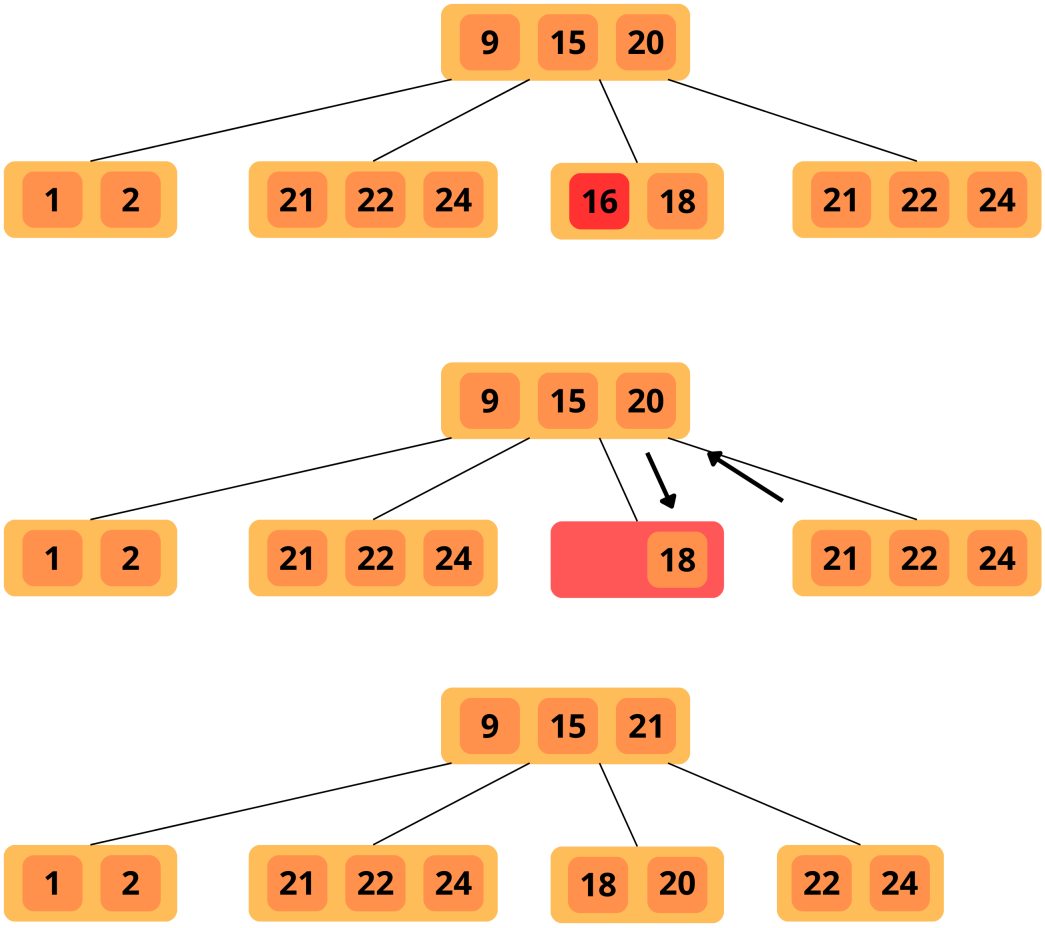
Note que ao excluí-lo, seu nó viola o mínimo necessário de elementos. Sendo assim ele é realocado para o nó vizinho. Entretanto ao fazer isso o nó pai fica com um ponteiro desnecessário, pois só havendo 3 filhos só se faz necessário 2 elementos no nó pai(ou seja, 3 ponteiros). Isso acarreta na descida do 21 para o nó recém-formado, de forma que essa realocação não viola o número máximo de elementos possível.

Exclusão em um nó cujos irmãos adjacentes somam 2t ou mais

Perceba que no caso abaixo o 16 é o alvo da exclusão, entretanto, seus nós irmãos adja-

centes somam 6 elementos e temos que $6 = 2t$. Sendo assim podemos fazer uma realocação razoavelmente simples dos elementos vizinhos para balancear a árvore.

Remover de um nó que possui o número minimo de chaves



Note que quando ocorreu a exclusão do 16 seu nó violou a propriedade do mínimo de elementos necessários. Sendo assim o 20 que estava no nó pai acabou descendo para compor o seu nó, e o 21 que era o menor element do irmão adjacente subiu para a posição onde antes estava o 20.

Referências Bibliográficas

Wengrow, J. (2020), *A Common-Sense Guide to Data Structures and Algorithms, Second Edition*, The Pragmatic Bookshelf, Raleigh, NC-US.