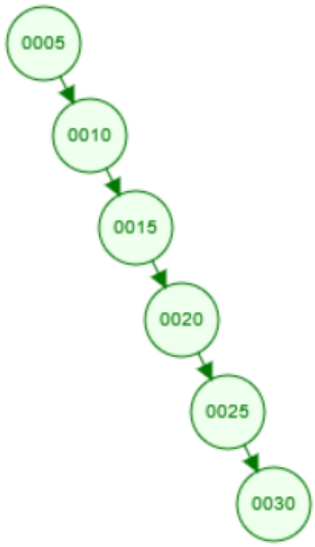
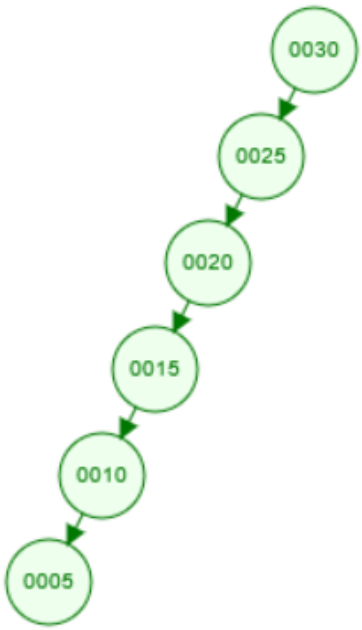
ED2 – 22.2 - RESUMO PROVA – ÁRVORES

OBS: Textos em **roxo** podem ser confusos!

- BST (BINÁRIA DE BUSCA):
  - Cada nó possui uma chave, um ponteiro para o nó a esquerda e um ponteiro para o nó a direita. A chave do nó a esquerda deve ser menor que a do nó corrente, e a da direita deve ser maior.
  - As árvores BST não são balanceadas, o que pode levar a O(n) caso os elementos sejam inseridos em uma ordem que propicie este fato (exemplo: valores em ordem crescente ou decrescente), exemplos abaixo:
    - BST com inserções 5, 10, 15, 20, 25, 30:



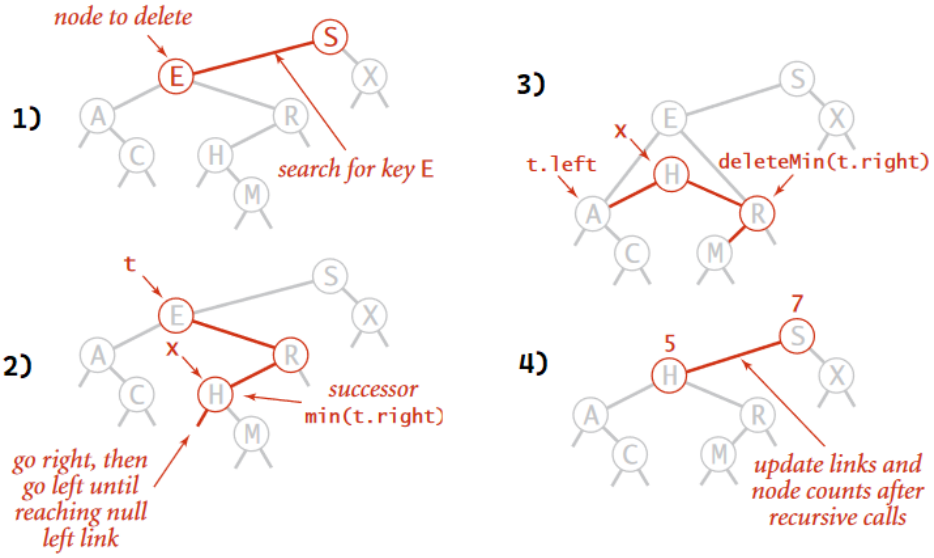
    - BST com inserções 30, 25, 20, 15, 10, 5:



- Busca:
  - Caso o valor seja menor ao do nó comparado, vá a sub-arvore esquerda, caso seja maior, vá a sub-arvore direita, caso seja igual, o nó foi encontrado.

- Inserção:
  - O primeiro elemento inserido é a raiz da árvore;
  - Em uma nova inserção, busca-se a posição a ser inserido a partir das seguintes condições:
    - Se o valor do nó for menor que o valor inserido, vá a sub-arvore esquerda, se for maior, vá a sub-arvore direita.

- Remoção (de Hibbard):
  - Se o nó for uma folha, remova o ponteiro do nó pai que direciona a ele;
  - Se o nó tiver um filho, remova o ponteiro do nó pai que direciona a ele e direcione ao filho;
  - Se o nó tiver dois filhos:
    - 1) Busque o seu sucessor (o nó a mais extrema esquerda da sub-arvore direita – este nó não deve possuir filho a esquerda);
    - 2) Após encontrá-lo, guarde seu valor e remova o nó da sub-arvore direita;
    - 3) Coloque o valor do sucessor no lugar do nó a ser deletado e ajuste os ponteiros:



- Percursos/Travessias:
  - ***Pré-ordem:*** Os filhos são processados após o nó. (Cima pra baixo);
  - ***Pós-ordem***: Os filhos são processados antes do nó. (Baixo pra cima);
  - ***Em-ordem***: São processados o filho à esquerda, o nó e o filho a direita.

- OBSERVAÇÕES IMPORTANTES:
  - *Existe alguma razão para evitarmos algoritmos de busca em árvore recursivos (memória, complexidade, etc)? Justifique.*
    - Sim, pois quando essa busca acontece no pior dos casos, ou seja, a árvore é degenerada, a busca irá percorrer todos os nós de um lado, apenas para depois desempilhar os valores na pilha de recursão e retornar nulo quando chegar na raiz, caso não encontre o valor. Isso faz com que a complexidade do algoritmo aumente, sendo ele O($\log n$), consequentemente aumentando seu tempo de execução.
      E ainda existem outras formas de busca, como a busca iterativa, que também possui loops mas que não utilizam recursão, e sim repetição, resultando numa complexidade e tempo de execução menores.

- AVL:
  - Árvore na qual as alturas das subárvores esquerda e direita de cada nó diferem no máximo por uma unidade.
  - Altamente balanceada, isto é: ao se inserir ou remover, executa-se uma rotina de balanceamento tal qual as alturas das sub-arvores esquerda e sub-árvores direita tenham alturas bem próximas;
  - Idealmente deve ser razoavelmente equilibrada e a sua altura será dada (no caso de estar completa) por $h = log2(n + 1)$
  - Possui complexidade O($\log n$) para todas as operações e ocupa espaço n, onde n é o número de nós na árvore.

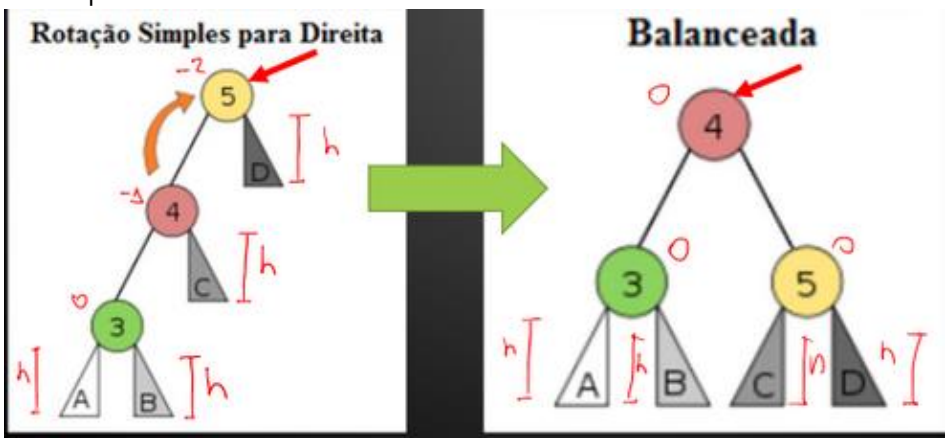
Complexidade da árvore AVL em notação O

	Média	Pior caso
Espaço	$O(n)$	$O(n)$
Busca	$O(\log n)$	$O(\log n)$
Inserção	$O(\log n)$	$O(\log n)$
Deleção	$O(\log n)$	$O(\log n)$

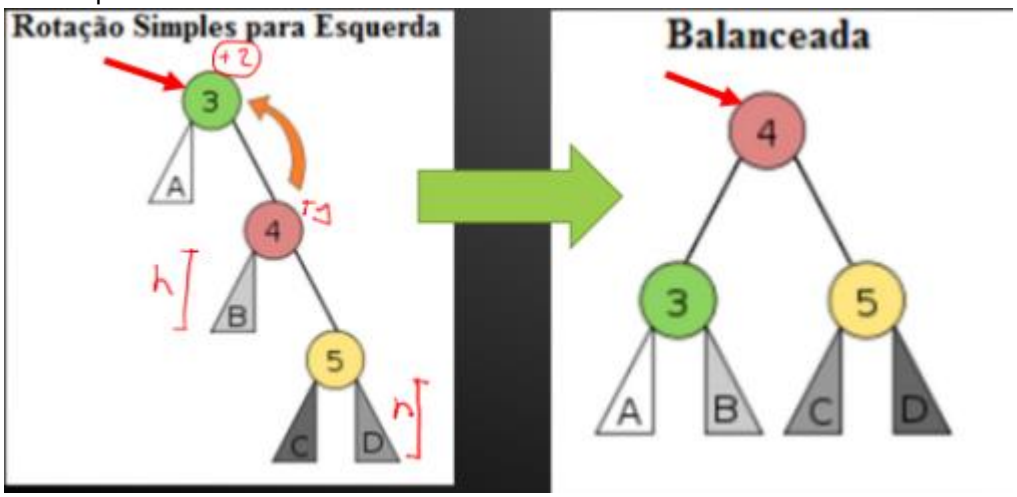
  - Busca:
    - Igual a da BST, caso o valor seja menor ao do nó comparado, vá a sub-arvore esquerda, caso seja maior, vá a sub-arvore direita, caso seja igual, o nó foi encontrado.

  - Balanceamento:
    - O fator de balanceamento de um nó é a diferença da altura da sub-árvore direita e esquerda: $FB = h_{direita} - h_{esquerda}$
    - Um nó balanceado deve possuir $-1 \leq FB \geq 1$

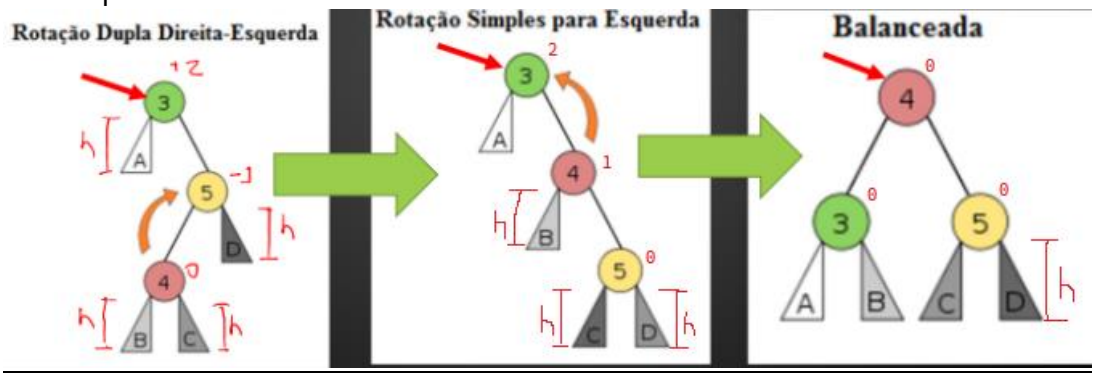
- *Rotação para a direita:*
  - Quando um nó possuir $FB = -2$, significa que a altura da sub-árvore esquerda está maior que a sub-árvore direita e deve ser balanceada. Esta rotação simples para a direita deve ser feita se o nó a esquerda possuir $FB = -1$ ou $FB = 0$.
  - Nesta rotação, o ponteiro direito do nó a sua esquerda deve receber este nó, e este nó deve receber em seu ponteiro esquerdo a direita do nó a esquerda.
  - Exemplo:



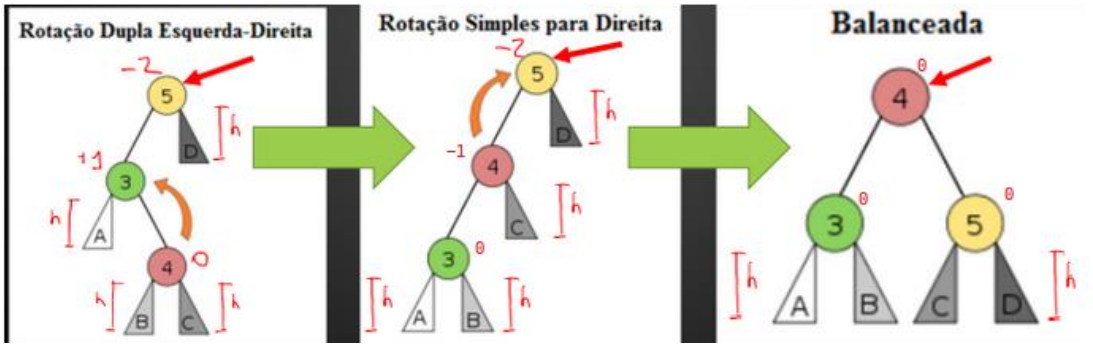
- *Rotação para a esquerda:*
  - Quando um nó possuir $FB = 2$, significa que a altura da sub-árvore direita está maior que a sub-árvore esquerda. Esta rotação simples para a esquerda deve ser feita se o nó a direita possuir $FB = 1$ ou $FB = 0$.
  - Nesta rotação, o ponteiro esquerdo do nó a sua direita deve receber este nó, e este nó deve receber em seu ponteiro direito a esquerda do nó a direita.
  - Exemplo:



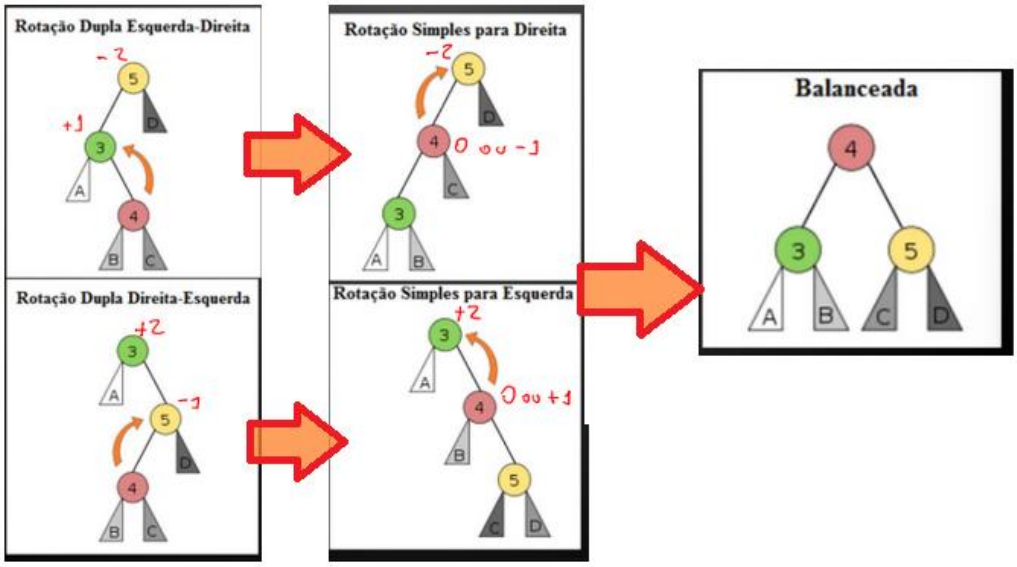
- *Rotação dupla Direita-Esquerda:*
  - Quando um nó possuir $FB = 2$, significa que a altura da sub-árvore direita está maior que a sub-árvore esquerda. Esta rotação deve ser feita e se o nó a direita possuir $FB = -1$.
  - Nesta rotação, é feita uma rotação para a direita no nó a direita e depois desta rotação, pegar o novo nó a direita e fazer uma rotação a esquerda.
  - Exemplo:



- *Rotação dupla Esquerda-Direita:*
  - Quando um nó possuir $FB = -2$, significa que a altura da sub-árvore direita está maior que a sub-árvore esquerda. Esta rotação deve ser feita e se o nó a direita possuir $FB = 1$.
  - Nesta rotação, é feita uma rotação para a esquerda no nó a esquerda e depois desta rotação, pegar o novo nó a esquerda e fazer uma rotação a direita.
  - Exemplo:



- TODAS AS ROTAÇÕES:



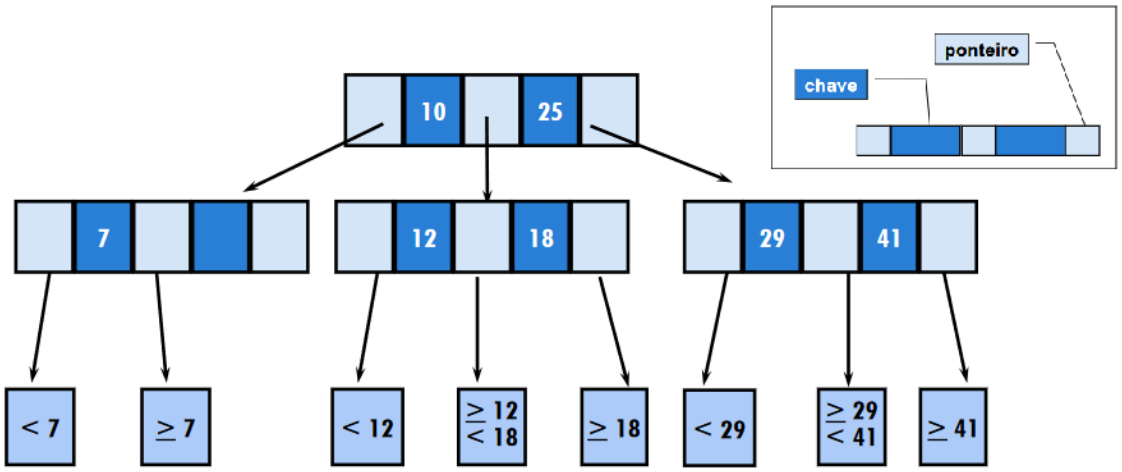
- Inserção:
  - Busca-se a posição a ser inserido a partir das seguintes condições: se o valor do nó for menor que o valor inserido, vá a sub-arvore esquerda, se for maior, vá a sub-arvore direita.
  - Após a inserção do nó, retorne o caminho até a raiz verificando o balanceamento dos nós do caminho:

- Remoção:
  - Busca-se o nó a ser removido a partir das seguintes condições: se o valor do nó for menor que o valor inserido, vá a sub-arvore esquerda, se for maior, vá a sub-arvore direita, caso seja igual, retorne o nó. Se chegou a um nó nulo, o nó não existe na árvore e não pode ser removido.
  - Caso o nó seja encontrado e seja uma folha, remova o ponteiro do nó pai que direciona a ele e volte o caminho a raiz da árvore rebalanceando-a se necessário;
  - Se o nó tiver um filho, remova o ponteiro do nó pai que direciona a ele e direcione ao filho e volte o caminho a raiz da árvore rebalanceando-a se necessário;
  - Se o nó tiver dois filhos:
    - 1) Busque o seu sucessor (o nó a mais extrema esquerda da sub-arvore direita – este nó não deve possuir filho a esquerda) ou o seu predecessor;
    - 2) Após encontrá-lo, guarde seu valor e remova o nó da sub-arvore direita;
    - 3) Coloque o valor do sucessor no lugar do nó a ser deletado e ajuste os ponteiros;
    - 4) Volte o caminho a raiz da árvore rebalanceando-a se necessário;

- Árvore B:
  - Árvore na qual ao invés de se armazenarem chaves em nós individuais, utiliza-se um bloco de chaves (página) para armazenar vários valores. Cada nó de uma árvore B é uma página.
  - Diferente das outras árvores, pode ter mais de 2 filhos e é muito utilizada para armazenamento e recuperação de dados.
  - Possuem ponteiros que apontam para os múltiplos caminhos e possui autobalanceamento. Ela costuma não ter uma altura tão grande comparada à outras árvores, pelo fato de armazenar blocos de chaves (as páginas).
  - Cada árvore B possui uma ordem, que determina as características dela. Uma Árvore B de ordem d deve ter as seguintes propriedades:
    - A raiz da árvore OU é uma folha, ou seja, é o único nó da árvore, OU possui no mínimo 2 filhos;
    - Cada nó interno possui no mínimo $d + 1$ filhos;
    - Cada nó possui no máximo $2d + 1$ filhos;
    - As chaves das páginas devem ser ordenadas do menor para o maior;
    - Cada página possui entre d e $2d$ chaves, exceto o nó raiz, que possui entre 1 e $2d$ chaves;
    - A quantidade de ponteiros de uma página é a quantidade de chaves + 1.
    - Folhas estão sempre no mesmo nível;

  - OBS: Determinação do número de ordem de uma árvore – há dois conceitos:
    - O conceito de Rudolf Bayer e Edward McCreight, 1972, criadores da árvore B, constata que a ordem de uma árvore é a metade da capacidade de chaves que sua página pode armazenar. Dessa forma, sendo uma árvore com ordem 3, suas páginas podem armazenar entre 3 e 6 chaves.
    - Já o conceito de Donald Knuth, 1978, constata que a ordem uma árvore B é a quantidade máxima de filhos que uma página da árvore pode ter, e que cada página contém, sendo d a ordem dessa árvore, entre $(2d - 1 / 3)$ e $d - 1$ chaves. Então, sendo uma árvore com ordem 5, suas páginas podem armazenar entre 2 e 4 chaves, e pode ter no máximo 5 filhos. Esta variação de Knuth é chamada de Árvore B*.

- Exemplo:



- Complexidade:

Time complexity in big O notation

Algorithm	Average	Worst case
Space	O(*n*)	O(*n*)
Search	O(log *n*)	O(log *n*)
Insert	O(log *n*)	O(log *n*)
Delete	O(log *n*)	O(log *n*)

- Busca:
  - Carrega o primeiro nó e verifica se o valor está antes ou depois do nó (direita ou esquerda) com base no valor se for maior ou menor, e vai percorrendo os nós da árvore.
  - Para cada nó não-folha visitado:
    - Se o nó tem a chave, retorna a chave.
    - Se não, desce para o filho apropriado, baseado no valor da chave (se for menor ou maior que os das chaves do nó atual).
  - Se chegar num nó folha e não for encontrado na posição que deveria estar, retorna *NULL*.

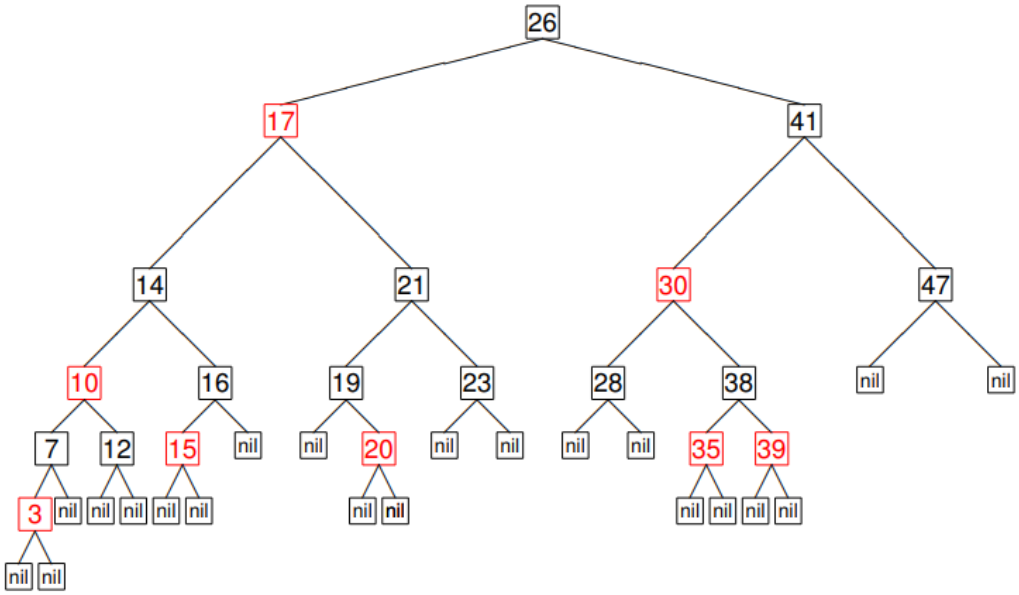
- Inserção:
  - Vai ordenando os valores conforme for inserindo, ou seja, mudando de posição caso entre um valor menor ou valor que o que estava. De forma que a árvore fique balanceada e que a página fique ordenada da menor para a maior chave.
  - Como é feita:
    - Executa algoritmo de busca, que identifica a posição em que a chave deverá ser inserida. Se a inserção for válida, insere a chave no nó adequado;
    - Em caso de página cheia:
      - Quando o último campo é ocupado, faz o split:

        - Divide o nó na metade e pega o último elemento da primeira metade.
        - Sobe ele para um nó acima, criando uma nova raiz;
        - Os elementos que sobraram são redivididos em novos nós abaixo do que subiu;

- <u>Remoção:</u>
    - Se a chave está em um nó folha, somente remova a chave.
    - Se a chave não estiver num nó folha, pegar a chave imediatamente maior (a primeira chave do nó que é referenciado no ponteiro a sua direita) e substituir.
        - Caso após a remoção desta chave o nó fique com menos chaves que o necessário, há 2 soluções:
            - Concatenação: duas páginas podem ser unidas, concatenadas, se elas forem irmãs adjacentes e se juntas possuírem menos de $2d$ chaves (número máximo de chaves). Páginas são irmãs adjacentes se têm o mesmo pai e são apontadas por ponteiros adjacentes desse pai.
                - 1) Seja página X pai das páginas Y e Z, que são irmãs adjacentes;
                - 2) Z teve uma chave removida e que ficou com menos de d chaves;
                - 3) Transferir chaves de Z para Y;
                - 4) Transferir a chave de X que separava os ponteiros de Y e Z para Y;
                - 5) Eliminar página Z e ponteiro.
            - Redistribuição: ocorre quando a soma de chaves de páginas irmãs adjacentes é maior que $2d$.
                - 1) Seja página X pai das páginas Y e Z, que são irmãs adjacentes;
                - 2) Colocar em Y d chaves;
                - 3) Colocar em X a chave $d + 1$;
                - 4) Colocar em Z as chaves restantes;
            - OBS: Se ambas opções forem possíveis, optar pela redistribuição, que é menos custosa, não se propaga e evita que o nó fique cheio, deixando espaço para futuras inserções.

- Rubro-Negro:
  - Árvore binária de busca auto balanceada, projetada para busca de dados na memória principal (RAM), em que cada nó possui os atributos abaixo:
    - cor (1 bit): pode ser **vermelho** ou **preto**.
    - key (e.g. inteiro): indica o valor de uma chave.
    - left, right: ponteiros que apontam para a sub-árvore esquerda e direita.
    - pai: ponteiro que aponta para o nó pai. O campo pai do nó raiz aponta para NIL.
  - Quando um nó não possui um filho (esquerdo ou direito), então vamos supor que ao invés de apontar para NIL, ele aponta para um nó fictício, que será uma folha da árvore. Assim, todos os nós internos contêm chaves e todas as folhas são nós fictícios.
  - As propriedades da árvore rubro-negra são:
    - Todo nó da árvore ou é **vermelho** ou é **preto**.
    - A raiz é sempre **preta**;
    - Toda folha (NIL - leaf) é **preta**;
    - Se um nó é **vermelho**, então ambos os filhos são **pretos**.
    - Para todo nó, todos os caminhos do nó até as folhas descendentes contêm o mesmo número de nós **pretos**;
    - É uma BST auto balanceada.
  - A altura h de uma árvore rubro-negra de n chaves ou nós internos é no máximo $2\ log(n\ +\ 1)$;

  - Exemplo:



- Busca:
  - 1) Começando pelo nó raiz da árvore, verificar se o valor do nó a ser buscado é maior ou menor do que o nó atual;
  - 2) Se for menor, o nó atual passa a ser o filho esquerdo do nó antigo (nesse caso, o raiz);
  - 3) Se for maior, o nó atual passa a ser o filho direito do nó antigo;
  - 4) Realizar esse procedimento até encontrar (ou não) o nó desejado;
  - 5) Retornar o valor booleano da busca.

- Inserção:
  - 1) Seja x e y nós raiz e folha da árvore, respectivamente;
  - 2) Checar se o nó raiz está vazio ou não. Se sim, o nó inserido será adicionado como nó raiz de cor preta;
  - 3) Se não, comparar o nó raiz com o novo nó. Se o novo nó for maior que a raiz, percorrer pela sub-árvore direita. Se não, pela esquerda;
  - 4) Repetir passo 3 até chegar na folha;
  - 5) Fazer o pai do nó raiz também ser pai do novo nó;
  - 6) Se o valor do nó folha for menor que o do novo nó, novo nó será filho esquerdo, se for maior, filho direito;
  - 7) Fazer filhos do novo nó como sendo nulos;
  - 8) Novo nó será **vermelho**;
  - 9) Restaurar as propriedades da árvore performando rotações ou mudando as cores dos nós.

  - Algoritmo para manutenção das propriedades da árvore:

    1. Performar os passos até que o pai p do nó inserido seja **vermelho**;
    2. Se p for filho esquerdo do nó avô do nó inserido:
       2.1. Caso 1
       - Quando a cor do filho direito do nó avô do nó inserido for **vermelho**, transforme a cor de ambos os filhos do avô para **preto** e faça avô ficar **vermelho**;
       - Designar o nó avô para o nó inserido.

       2.2. Caso 2
       - Se não, se o nó inserido for o filho direito do pai, designar p para o nó inserido e realizar rotação esquerda;

       2.3. Caso 3
       - Transformar pai para **preto** e avô para **vermelho** e performar rotação direita para o nó inserido;
    3. Se p não for filho esquerdo do nó avô:
       3.1. Se o filho esquerdo do avô for **vermelho**, transformar ambos os filhos do avô em **preto** e avô em **vermelho**;
       3.2. Designar avô para o nó inserido;

3.3. Se não, se o nó inserido for filho esquerdo do pai, designar nó pai para o filho e performar rotação direita;
3.4. Transformar nó pai para **preto** e avô para **vermelho**;
3.5. Realizar rotação esquerda para nó avô;

4. Fazer nó raiz **preto**.

- Remoção:

1. Realizar a deleção padrão de árvore binária de busca. Fazendo isso, sempre é deletado um nó que ou é folha ou só tem um filho (se for interno, copia-se o sucessor e recursivamente chama a remoção para o sucessor, sendo o sucessor sempre folha ou nó com um filho). Então só é necessário cuidar de casos em que o nó é folha ou só tem um filho. Seja v o nó a ser deletado e u o filho que o substitui (u será nulo quando v for folha, e nulo é sempre **preto**);

2. A) Caso simples - Se ou u ou v são **vermelhos**, deleta-se v e marca-se u como **preto**;
   B) Se ambos u e v forem **preto**:
   - B.1. Colorir u como **double black**. Agora deve-se converter **double black** em apenas **black**;
   - B.2. Fazer os seguintes passos enquanto u for **double black** e não for raiz. Seja s o irmão de u;
   - B.2.1 Se s for **preto** e um de seus filhos for **vermelho**, realizar rotação. Seja r o filho **vermelho** de s. Existem quatro possíveis alternativas dependendo das posições:
     a) Caso esquerda esquerda - s é filho esquerdo de seu pai e r é filho direito de s ou ambos os filhos de s são **vermelhos**. Aqui, o pai p substitui o nó deletado, nó irmão s se torna novo pai e o filho esquerdo r substitui s;
     b) Caso esquerda direita - s é filho esquerdo de seu pai e r é filho direito de s. Nessa situação, filho r substitui s e s se torna filho esquerdo de r. Após isso, r se torna pai de p e s.
     c) Caso direita direita - s é filho direito de seu pai e r é filho direito de s ou ambos os filhos de s são **vermelhos**. Nesse caso, o pai p substitui o nó deletado, nó irmão s se torna novo pai e o filho direito r substitui s;
     d) Caso direita esquerda - s é filho direito de seu pai e r é filho esquerdo de s. Nesse caso, filho r substitui s e s se torna filho direito de r. Após isso, r se torna pai de p e s.
   - B.2.2 Se s for **preto** e ambos seus filhos são **preto**, recolorir e recorrer ao pai se ele for **preto**;
   - B.2.3 Se s for **vermelho**, performar rotação para levantar o antigo irmão, recolorir ele e o pai. O novo irmão é sempre **preto**. Existem duas alternativas para esse caso:
     a) Caso esquerda - s é filho esquerdo do pai. Rotacionar para direita o pai p;

b) Caso direita - s é filho direito do pai. Rotacionar para esquerda o pai p;

- B.3. Se u for raiz, colorí-lo para apenas **black** e retornar (altura de **pretos** da árvore reduz em 1).