SIN5013 - Análise de Algoritmos e Estruturas de Dados

Árvores Binárias de Busca

Prof. Flávio Luiz Coutinho

Árvores Binárias de Busca

Uma árvore binária de busca é uma especialização de uma árvore binária.

Em uma árvore binária de busca, os elementos armazenados na árvore são organizados de modo que seja possível buscar um elemento de modo bastante eficiente (através de uma busca binária).

Árvores Binárias de Busca

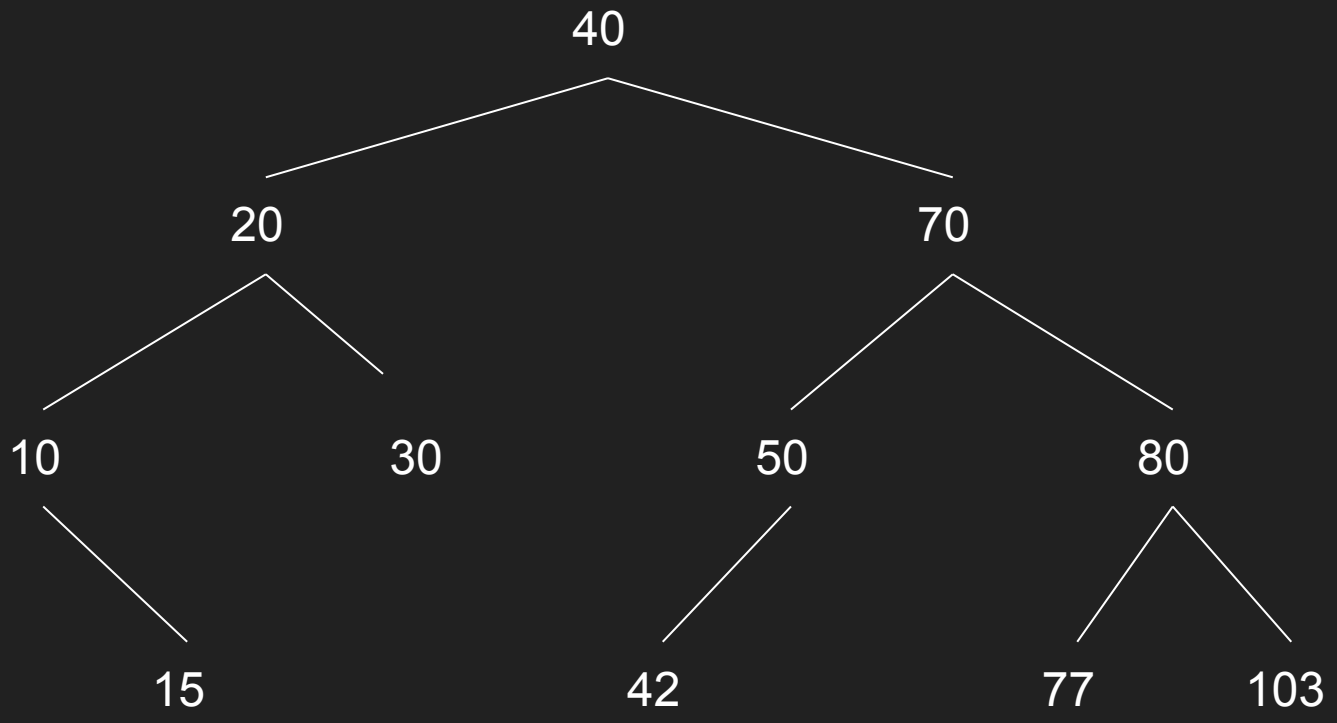
Iremos assumir uma árvore binária de busca em que elementos/valores repetidos não são permitidos. Neste cenário, a seguinte propriedade deve ser válida para todo e qualquer nó:

- Todos os valores presentes na sub-árvore à esquerda do nó devem ser menores do que o valor armazenado no nó.

- Todos os valores presentes na sub-árvore à direita do nó devem ser maiores do que o valor armazenado no nó.

Árvores Binárias de Busca

Exemplo:



Árvores Binárias de Busca: implementação

Em relação à forma como organizamos o conteúdo da árvore em memória, nada muda em relação ao que já vimos para as árvores binárias de propósito geral (usamos exatamente as mesmas *structs*).

Árvores Binárias de Busca: implementação

Já em relação às operações:

- criação — nada muda.
- impressão — nada muda, mas há ressalvas interessantes a fazer.
- busca — modificada para tirar proveito da organização dos valores.
- inserção — elementos novos são inseridos de modo a manter válida a propriedade das árvores de busca binária.
- remoção — idem (remoção de também deve manter a propriedade).

Árvores Binárias de Busca: busca

Quando precisamos buscar um valor em uma árvore de binária de busca, podemos tirar proveito da organização dos elementos na árvore para acelerar esta operação (em comparação com uma busca em que o valor procurado deve ser comparado, na pior das hipóteses, com todos os valores armazenados).

A busca funcionará seguindo praticamente o mesmo princípio de funcionamento da busca binária sobre um vetor (*array*) ordenado que já estudamos.

Árvores Binárias de Busca: busca

Suponha que estejamos procurando pelo valor x em uma árvore. Iniciamos a busca comparando x com o valor que está na raiz da árvore:

- Se o valor da raiz for igual a x, então a busca termina.
- Senão, caso x exista, ele estará em apenas uma das subárvores: na sub-árvore esquerda, caso x seja menor que o valor da raiz, ou na sub-árvore direita, caso x seja maior do que o valor da raiz. Para prosseguir, basta realizar a busca por x de forma recursiva na sub-árvore em que x deve estar caso exista.

Árvores Binárias de Busca: busca

No pior cenário possível para esta versão da busca, ao invés de compararmos x com todos os valores presentes na árvore, iremos comparar x com apenas um valor de cada nível da árvore, até chegarmos em uma folha. Logo, o custo de realizar a busca binária em uma árvore de busca binária é linearmente proporcional à altura h da árvore.

Em um caso favorável, em que n elementos estão distribuídos de forma balanceada em uma árvore binária de busca (ou seja, todos os níveis estão completos, com exceção do último nível), teremos $h = \Theta(\log_2 n)$. Logo, a busca terá complexidade $O(\log_2 n)$ em contraste com uma complexidade $\Theta(n)$ da busca “ingênua”.

Árvores Binárias de Busca: busca

```
No * busca_bin_rec(No * no, Elemento e){
    if(no){
        if(no->valor == e) return no;
        if(e < no->valor) return busca_bin_rec(no->esq, e);
        return busca_bin_rec(no->dir, e);
    }
    return NULL;
}
```

Árvores Binárias de Busca: impressão (percurso)

A Impressão/percurso em uma árvore binária de busca é implementada exatamente da mesma forma que em uma árvore binária de propósito geral.

Árvores Binárias de Busca: impressão (percurso)

A Impressão/percurso em uma árvore binária de busca é implementada exatamente da mesma forma que em uma árvore binária de propósito geral.

Embora não haja nenhuma novidade em termos de implementação, em termos práticos o percurso *in-ordem* tem uma propriedade interessante.

No percurso *in-ordem* passamos primeiro por toda a sub-árvore esquerda, em seguida pela raiz, e posteriormente pela sub-árvore direita (sendo que cada sub-árvore é percorrida usando a mesma regra).

Árvores Binárias de Busca: impressão (percurso)

Logo, se fizermos um percurso *in-ordem* com a finalidade de imprimir os valores armazenados na árvore, será impresso:

- um “bloco” de elementos, todos menores que a raiz.
- a própria raiz.
- outro “bloco” de elementos, todos maiores que a raiz.

Portanto, o percurso *in-ordem* garante a impressão de todos os valores armazenados na árvore em ordem crescente de valor.

Árvores Binárias de Busca: inserção

Em uma árvore binária de busca, não indicamos mais quem será o nó pai do novo elemento a ser inserido (como fazíamos na inserção em uma árvore binária de propósito geral).

É a própria implementação da inserção que define sob qual nó o novo valor será colocado, e isso é feito de modo a manter a propriedade das árvores binárias de busca válida.

Árvores Binárias de Busca: inserção

Podemos pensar na operação de inserção de forma recursiva. Seja x o valor que deve ser inserido na árvore, e assuma que tal valor ainda não existe na árvore. Temos as seguintes possibilidades:

- se a árvore está vazia, um novo nó com x é criado, e este torna-se a raiz.
- se x é menor que o valor na raiz:
  - se a raiz não possui sub-árvore à sua esquerda, cria-se um novo nó contendo x e o colocamos como o filho à esquerda da raiz.
  - caso contrário (sub-árvore à esquerda não vazia), então para inserir x na árvore, devemos inserir x na sub-árvore à esquerda, o que pode ser feito recursivamente.
- se x é maior do que a raiz, seguimos os mesmos passos descritos acima, mas considerando a sub-árvore à direita.

Árvores Binárias de Busca: inserção

Note que o procedimento descrito no slide anterior preserva a propriedade das árvores binárias de busca, uma vez que novos valores menores do que a raiz são sempre inseridos na sub-árvore da esquerda, enquanto que valores maiores do que a raiz são inseridos na sub-árvore à direita.

Árvores Binárias de Busca: inserção

Note que o procedimento descrito no slide anterior preserva a propriedade das árvores binárias de busca, uma vez que novos valores menores do que a raiz são sempre inseridos na sub-árvore da esquerda, enquanto que valores maiores do que a raiz são inseridos na sub-árvore à direita.

Além disso, o custo da operação também é proporcional à altura da árvore que, no cenário ideal de uma árvore balanceada, será $\Theta(\log_2 n)$.

Árvores Binárias de Busca: remoção

Assim como a remoção em uma árvore binária de propósito geral, a remoção de um nó (para o qual já se tem um ponteiro, obtido através de uma busca pelo valor que desejamos remover) em uma árvore binária de busca é uma operação um pouco mais complexa do que as demais.

Podemos implementar a remoção de forma parecida com o que fizemos na árvore binária de propósito geral: “promovendo” algum valor descendente do nó que desejamos remover, e removendo o valor duplicado usando a própria operação de remoção recursivamente.

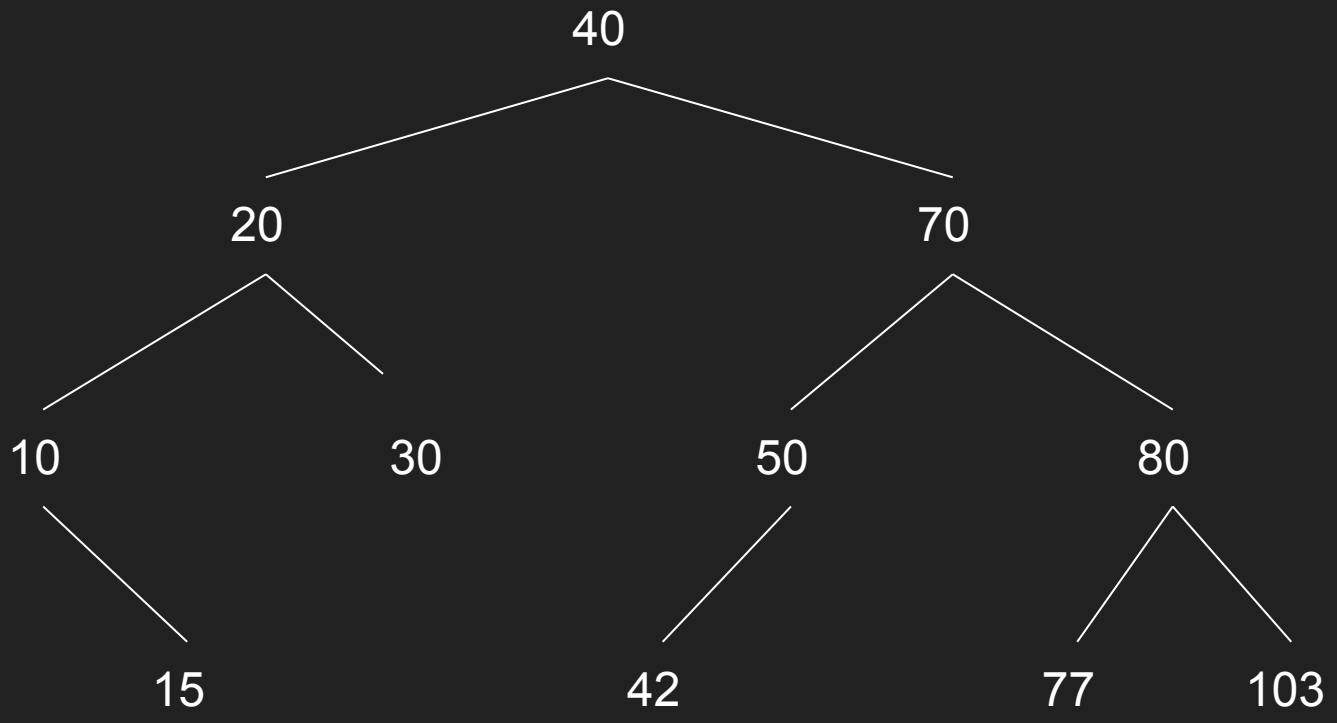
Árvores Binárias de Busca: remoção

A grande diferença é que precisamos ser mais cautelosos na escolha do valor descendente a ser “promovido”. Tomar o valor de um dos filhos do nó sendo removido e promovê-lo pode resultar em uma árvore que viola a propriedade das árvores de binárias de busca.

Veja um exemplo na sequência de slides a seguir de como a “promoção” dos valores nos filhos produz uma árvore binária de busca inválida!

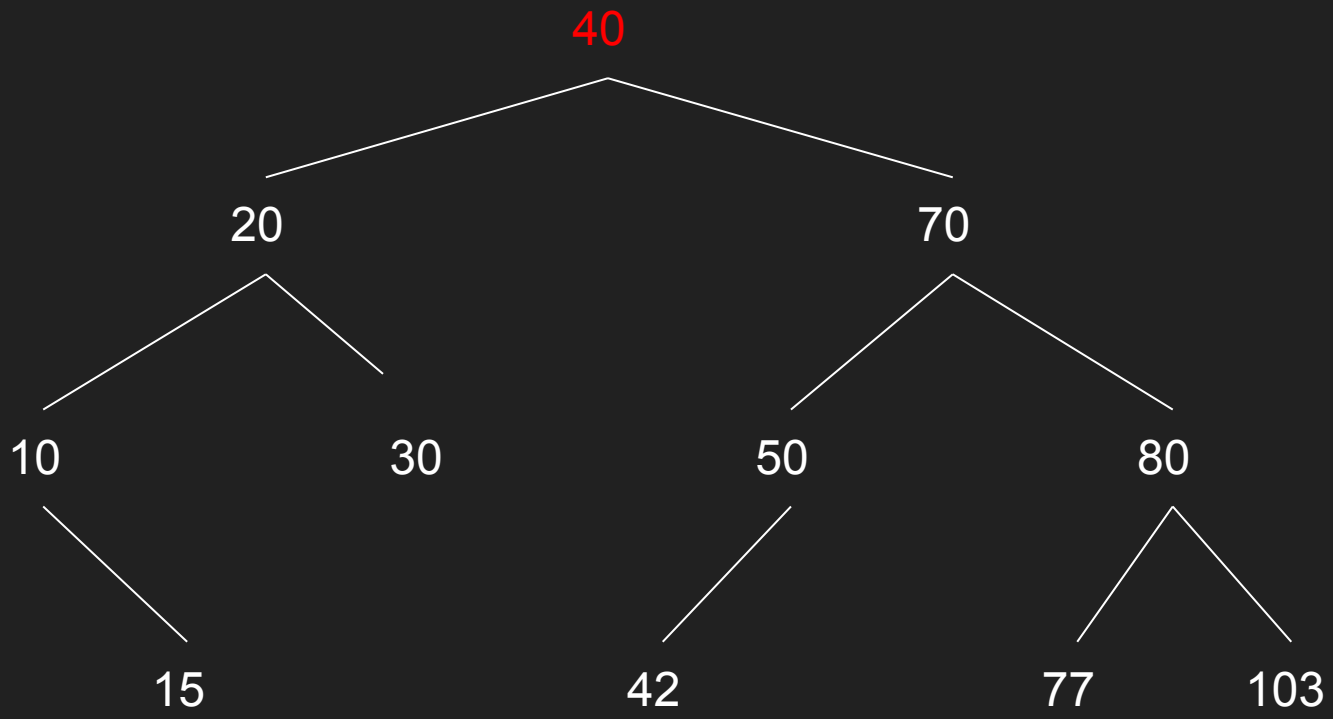
Árvores Binárias de Busca: remoção

Exemplo:



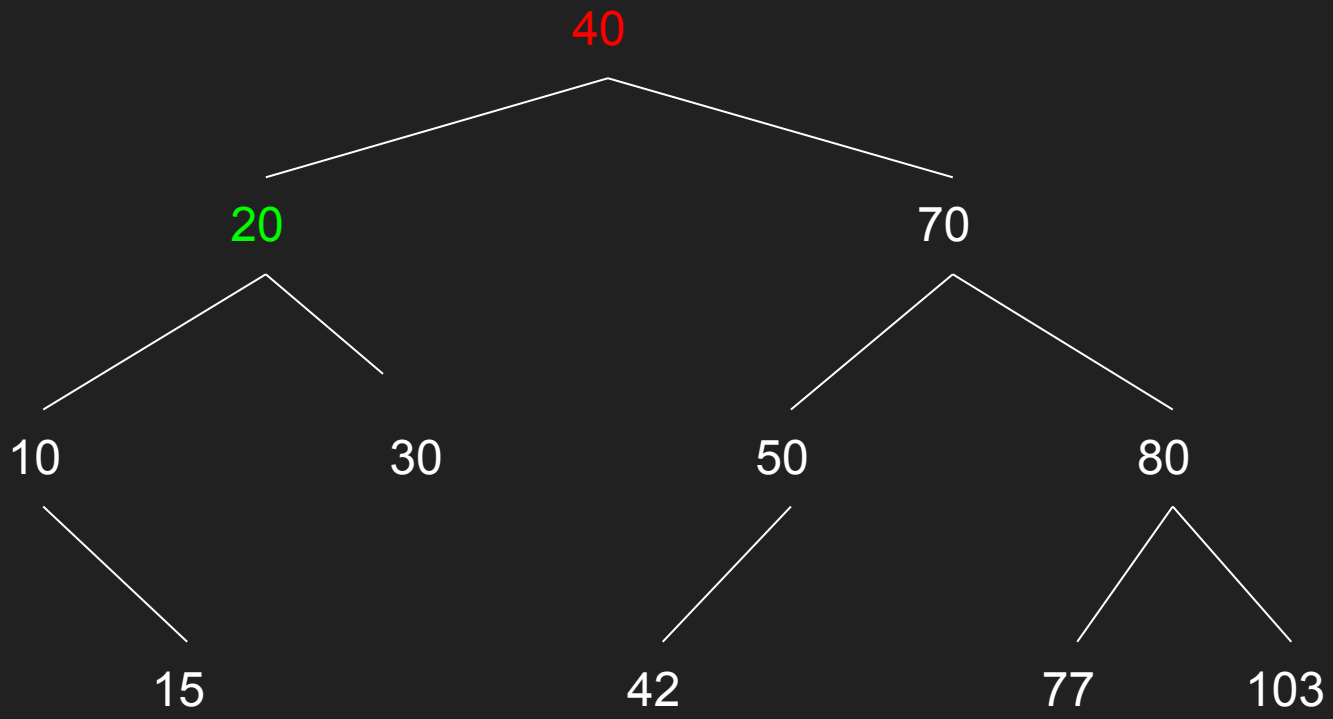
Árvores Binárias de Busca: remoção

Exemplo:



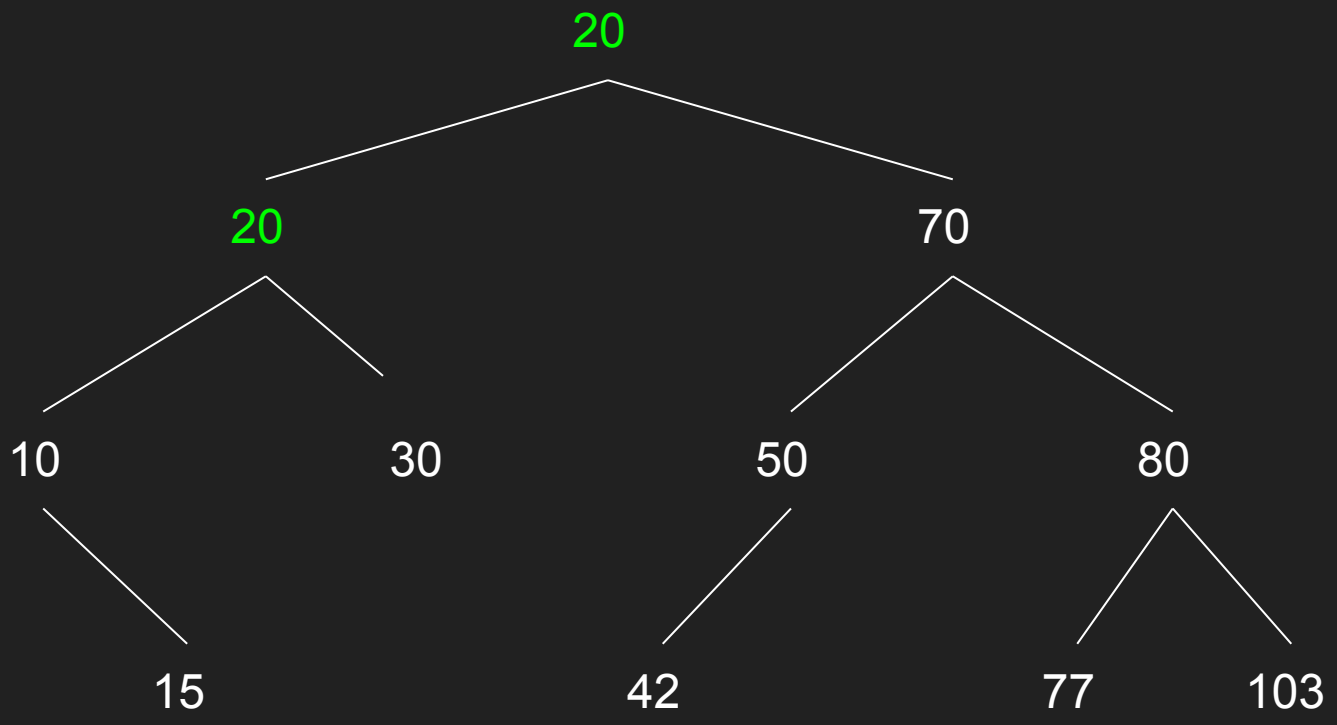
Árvores Binárias de Busca: remoção

Exemplo:



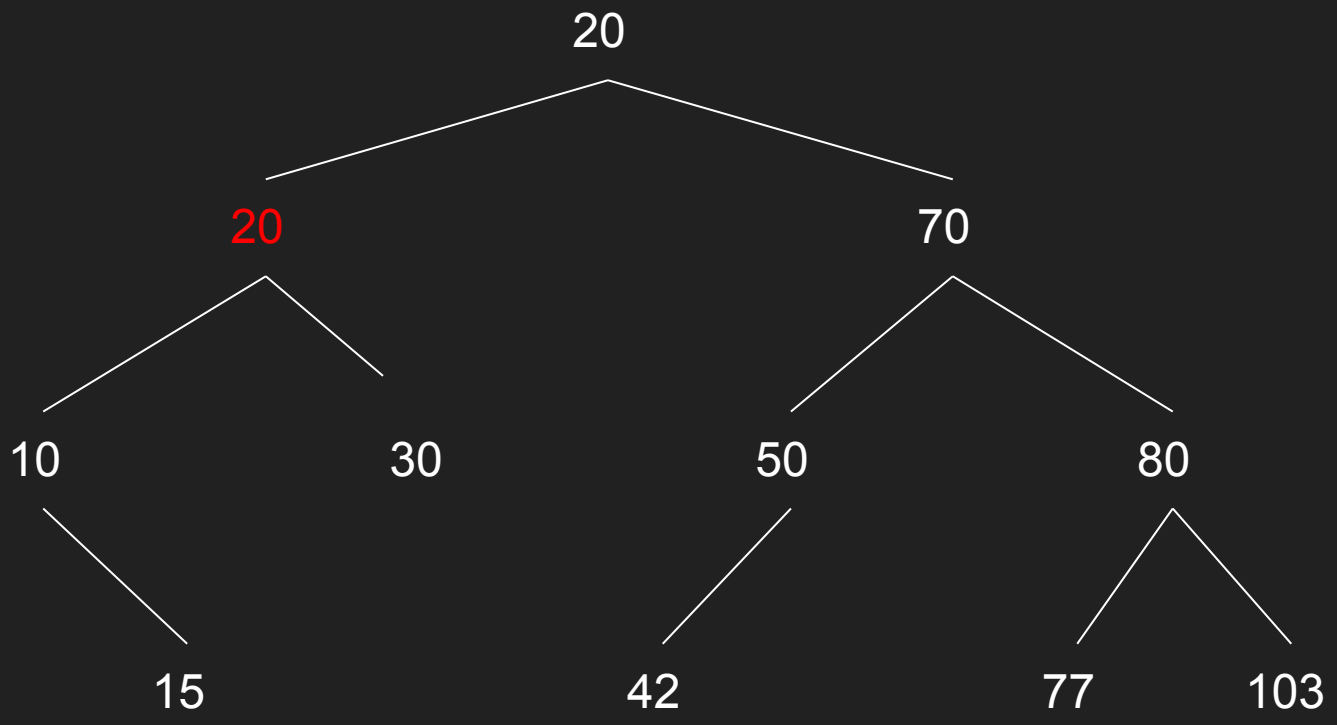
Árvores Binárias de Busca: remoção

Exemplo:



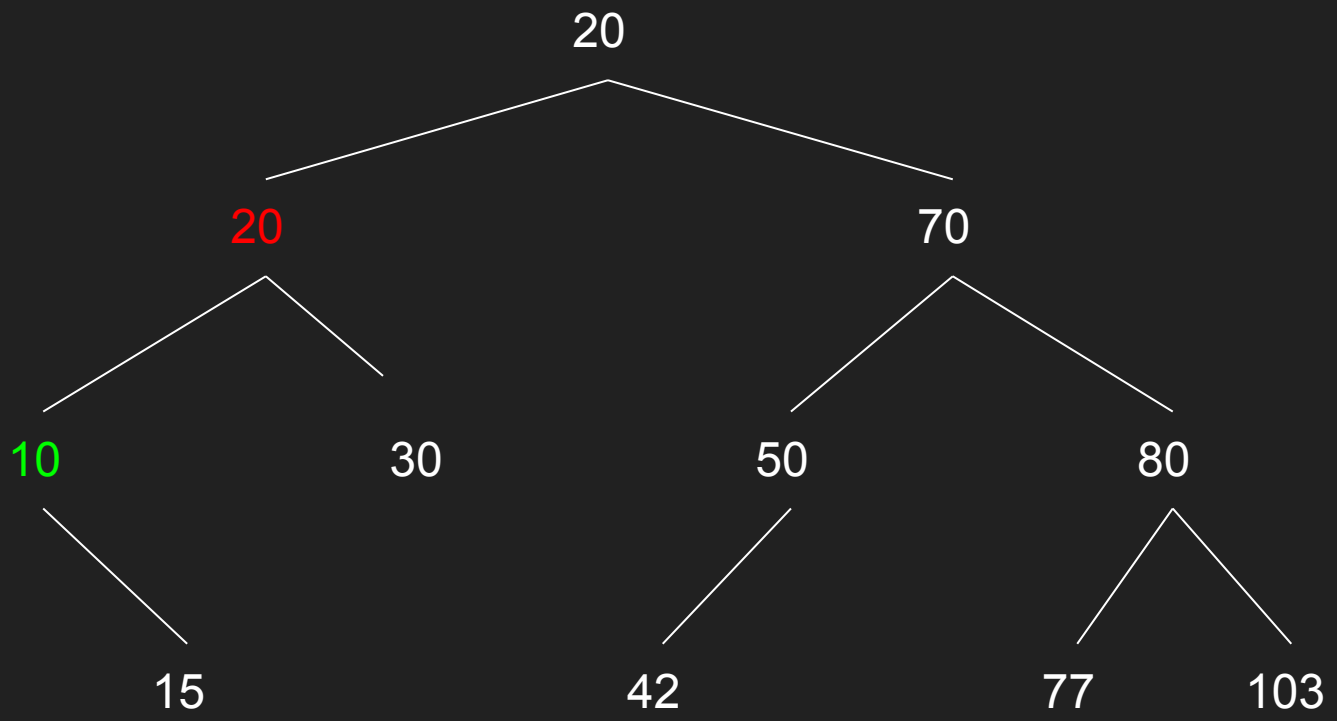
Árvores Binárias de Busca: remoção

Exemplo:



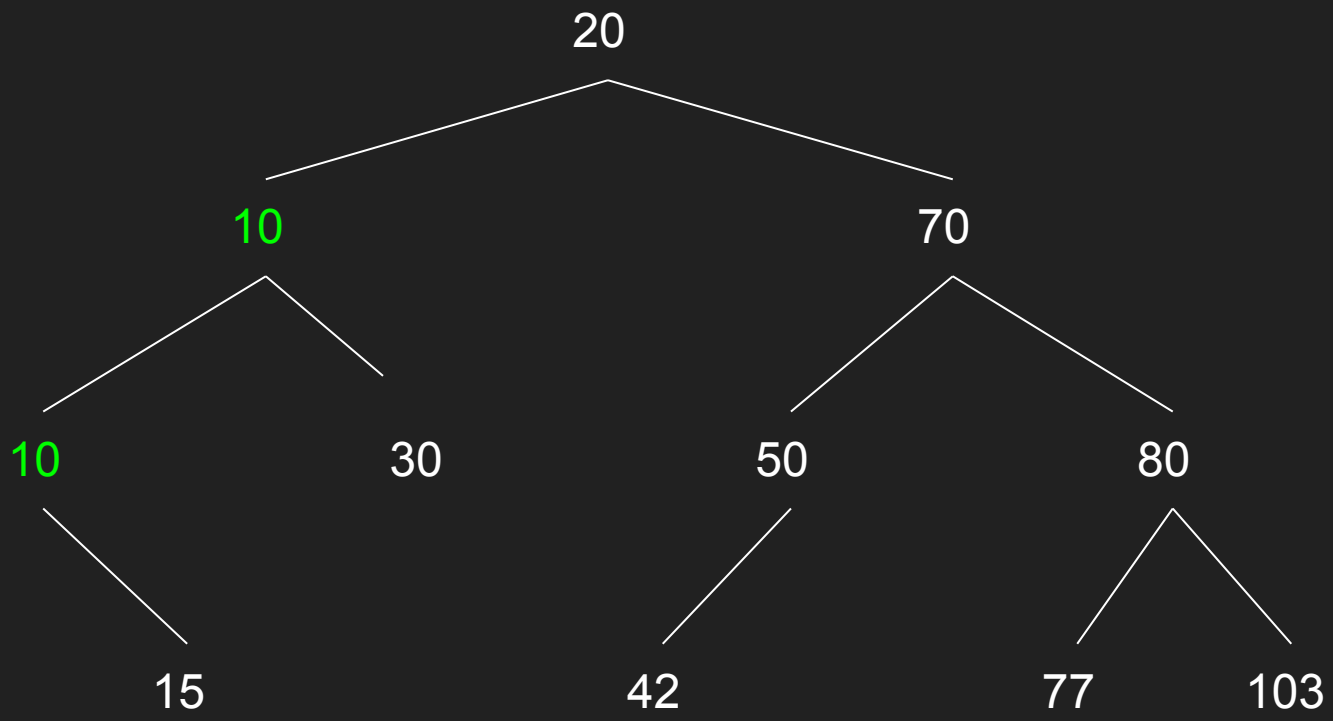
Árvores Binárias de Busca: remoção

Exemplo:



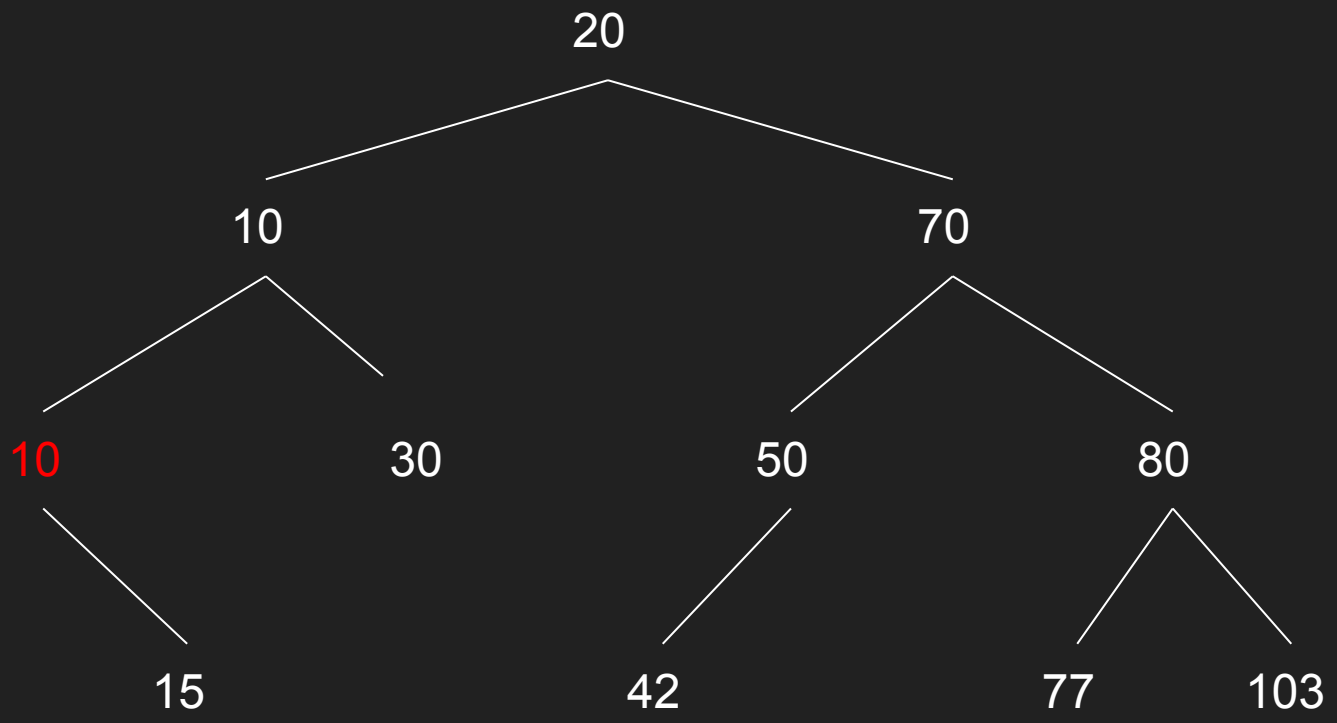
Árvores Binárias de Busca: remoção

Exemplo:



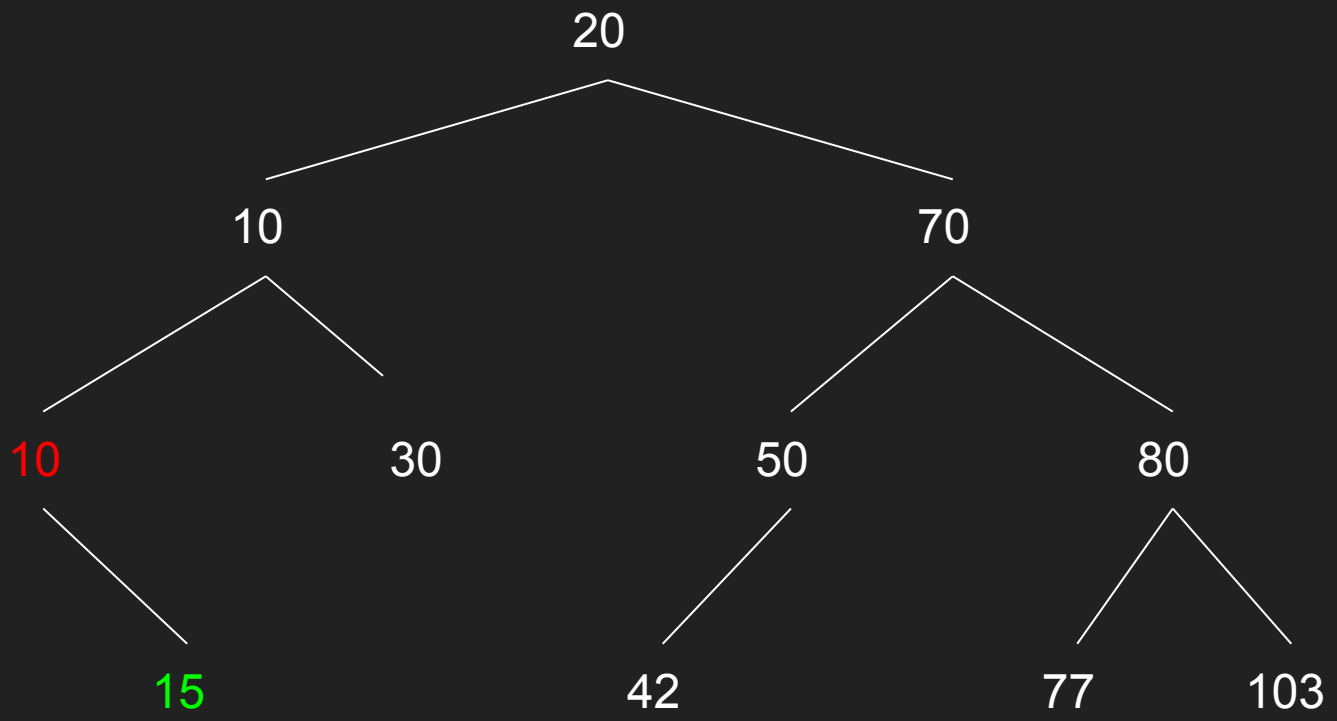
Árvores Binárias de Busca: remoção

Exemplo:



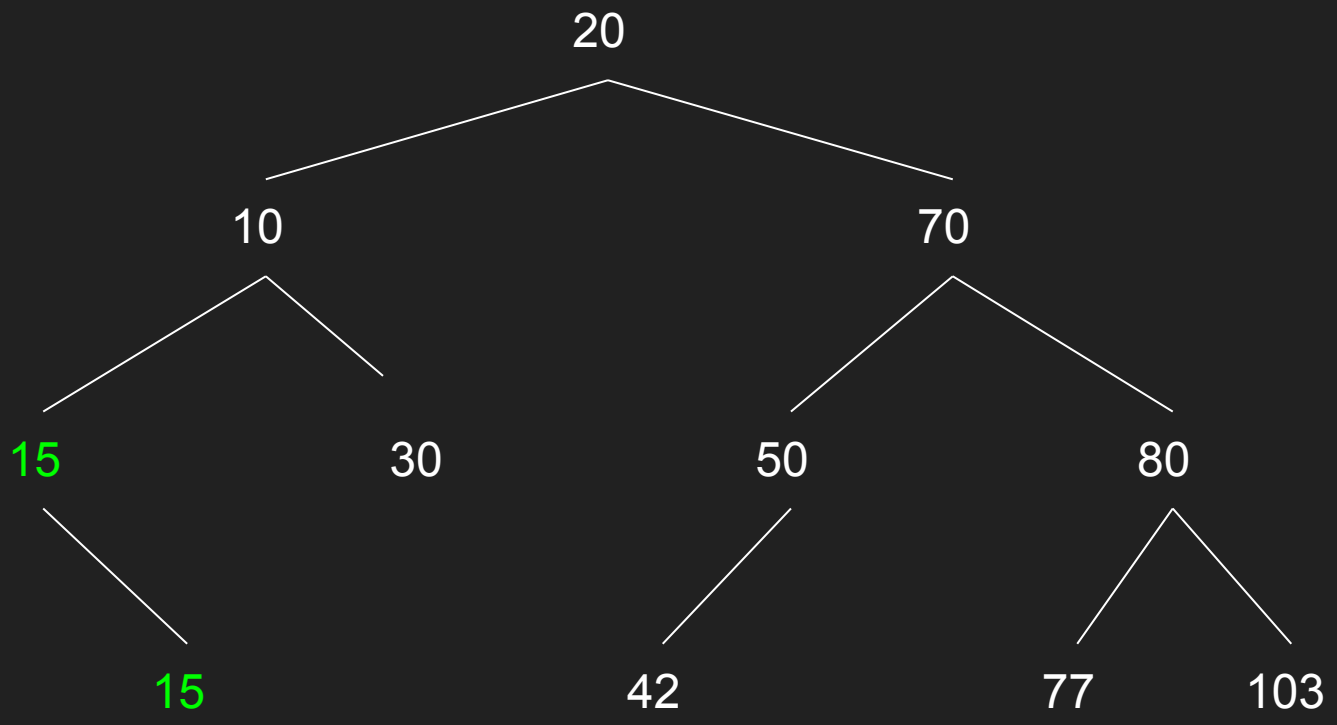
Árvores Binárias de Busca: remoção

Exemplo:



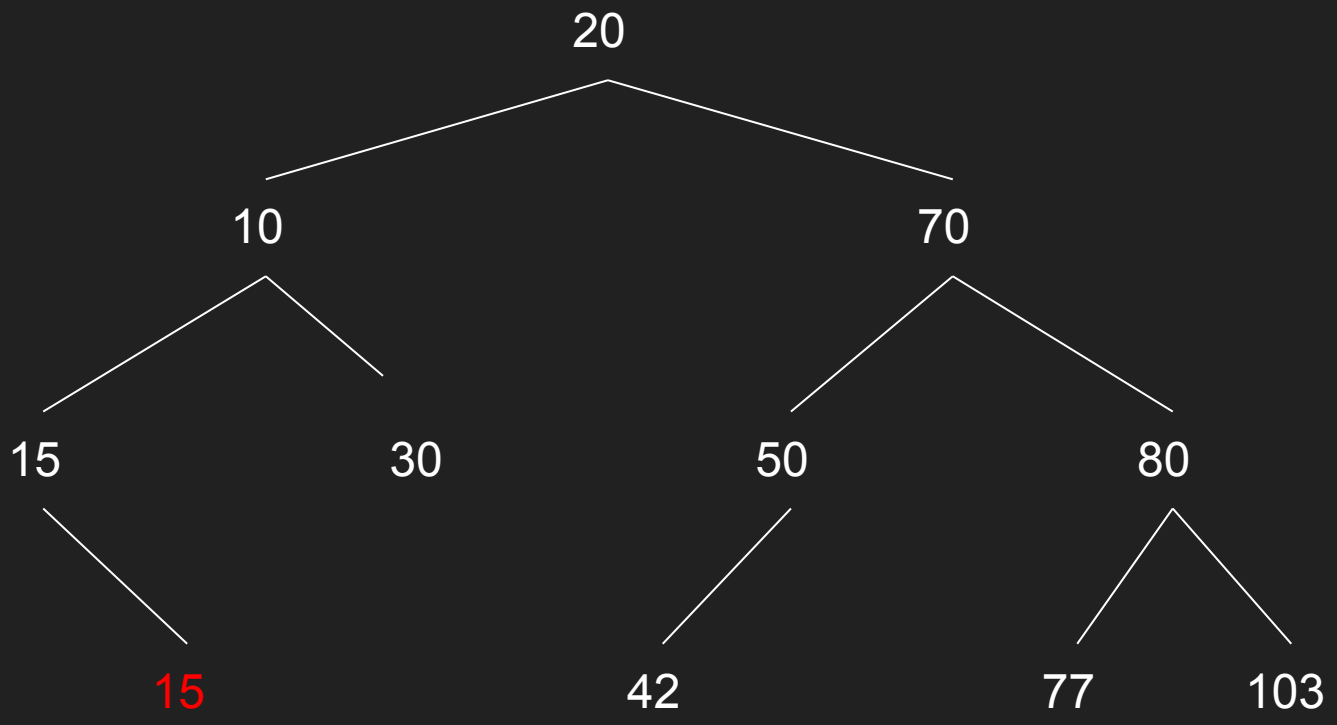
Árvores Binárias de Busca: remoção

Exemplo:



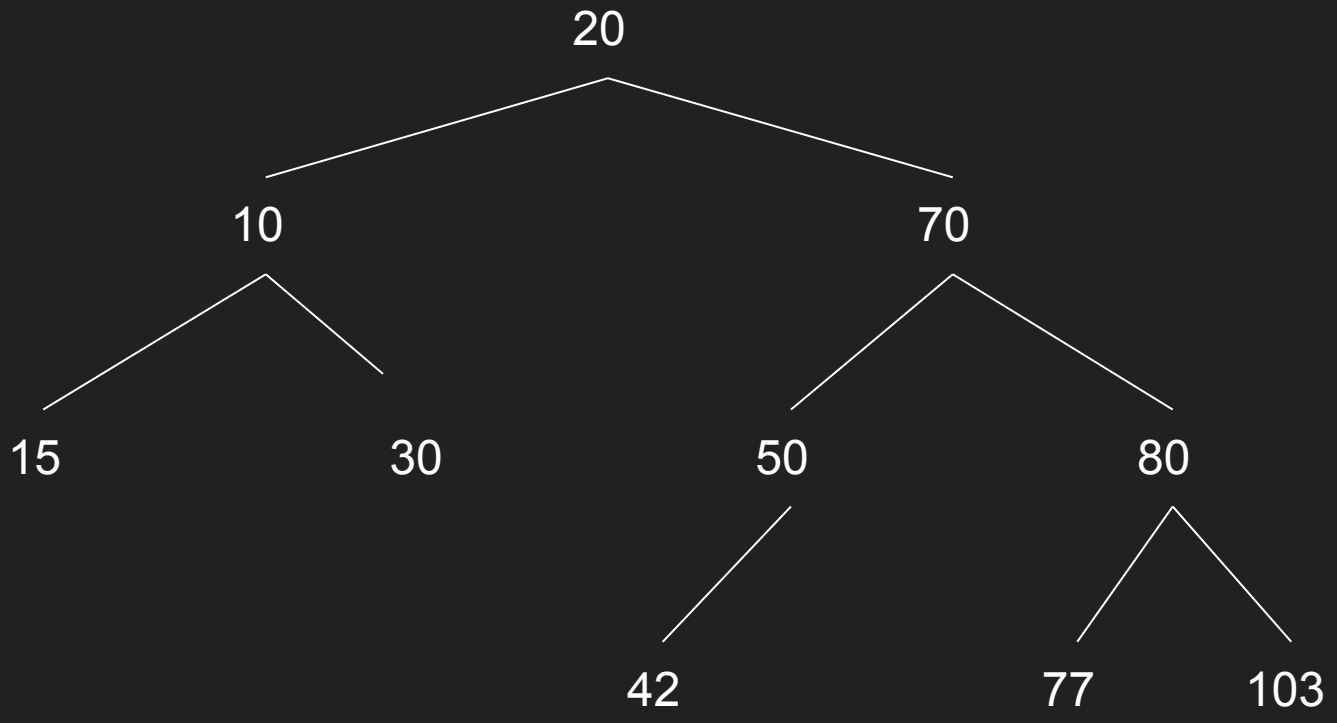
Árvores Binárias de Busca: remoção

Exemplo:



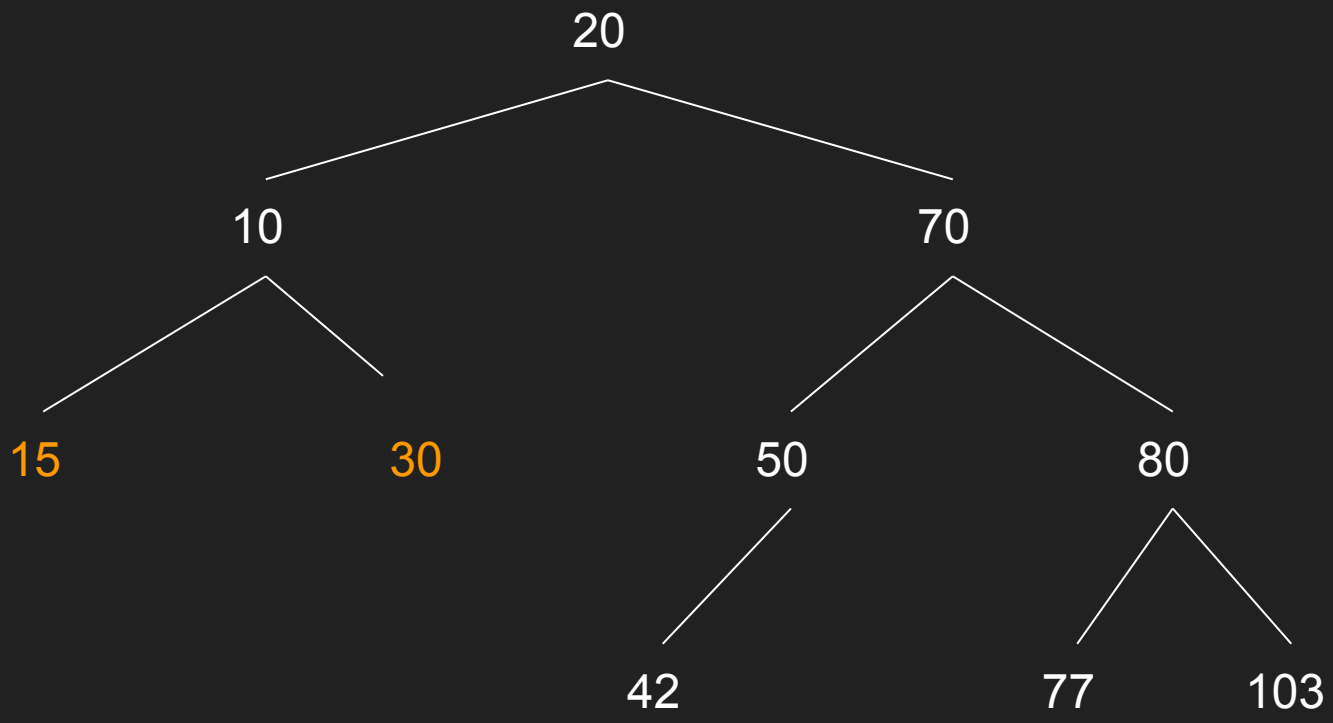
Árvores Binárias de Busca: remoção

Exemplo:



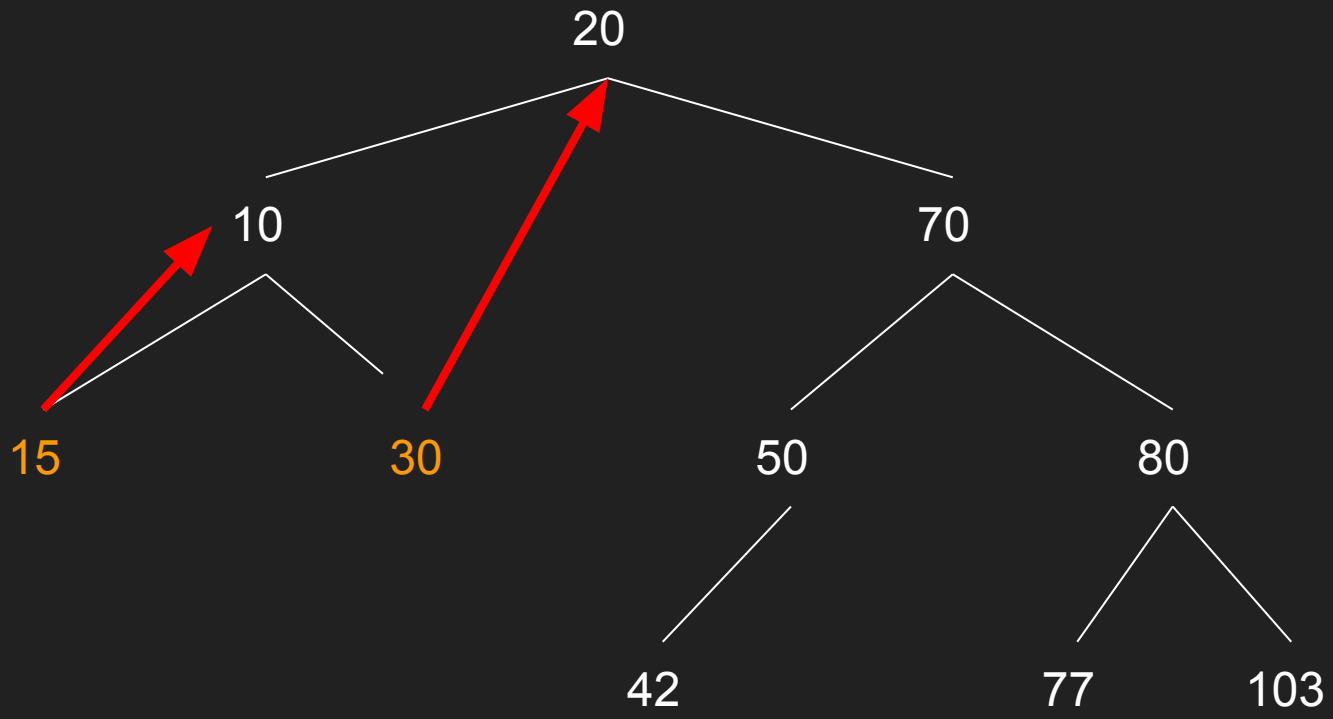
Árvores Binárias de Busca: remoção

Exemplo:



Árvores Binárias de Busca: remoção

Exemplo:



Árvores Binárias de Busca: remoção

A escolha do valor a ser “promovido” não pode violar a propriedade da árvore binária de busca. Como garantir isso?

Escolhendo o menor valor da sub-árvore à direita do nó sendo removido, ou o maior valor da sub-árvore à esquerda.

Qualquer uma destas escolhas garante que a propriedade das árvores binárias de busca continue valendo para o nó que recebeu o valor “promovido”.

Árvores Binárias de Busca: remoção

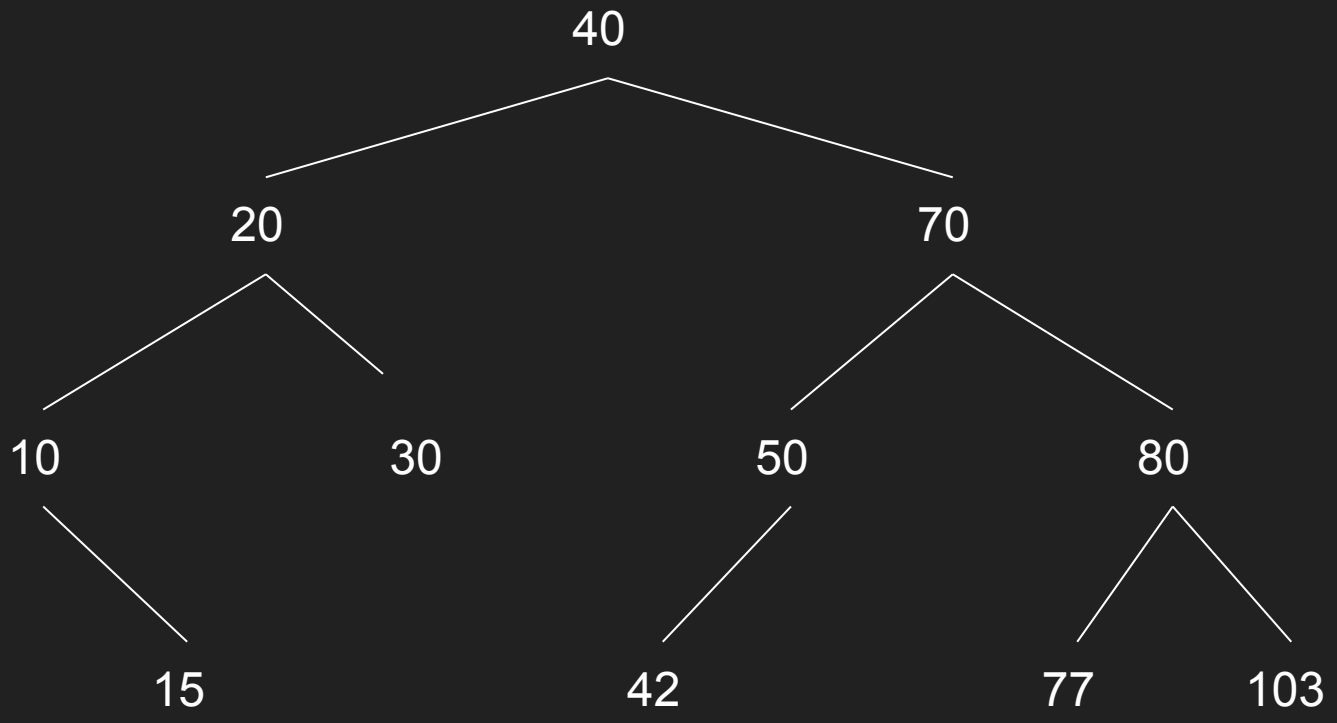
Em seguida, removemos recursivamente o nó que forneceu o valor “promovido” e ficou com o valor duplicado.

Se o nó a ser removido, seja na chamada inicial, ou em alguma chamada recursiva for uma folha, então o nó é efetivamente removido (através de ajuste do ponteiro adequado no nó pai) e desalocado.

Se o nó removido for o único da árvore, precisamos atualizar a raiz da árvore também, para que ela aponte para o endereço nulo.

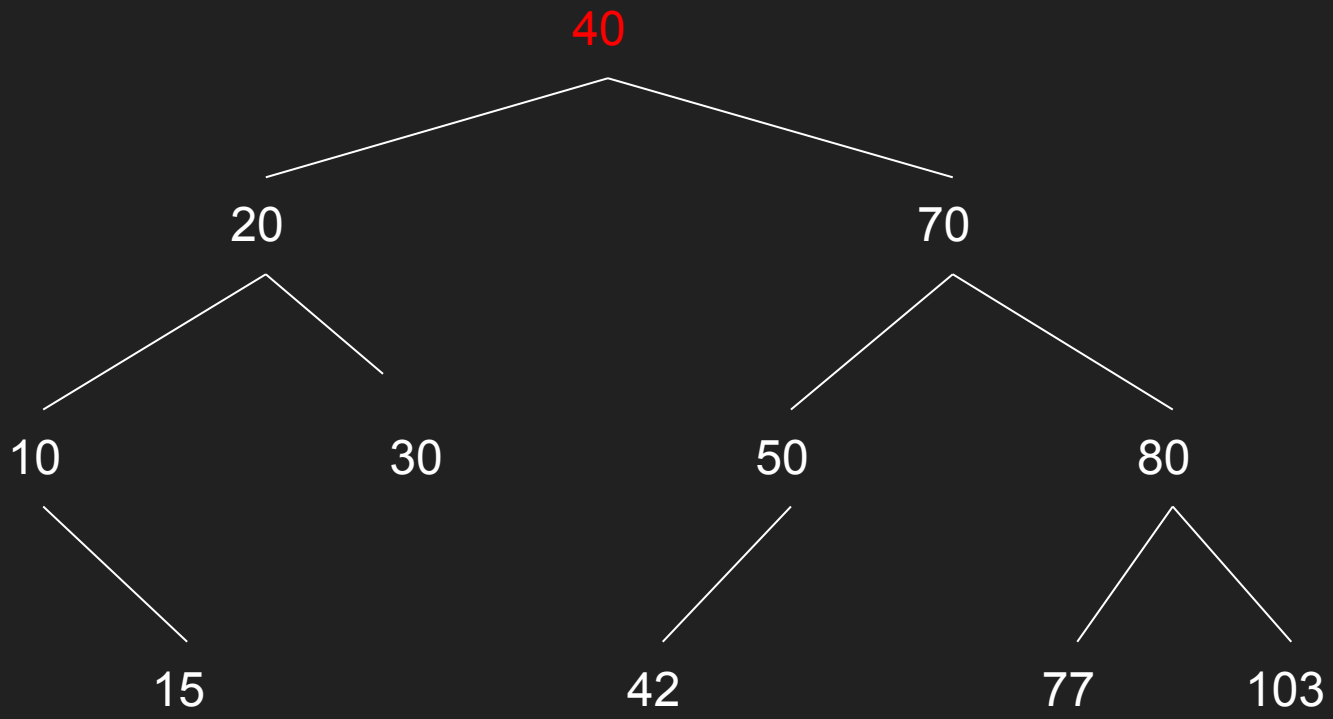
Árvores Binárias de Busca: remoção

Exemplo:



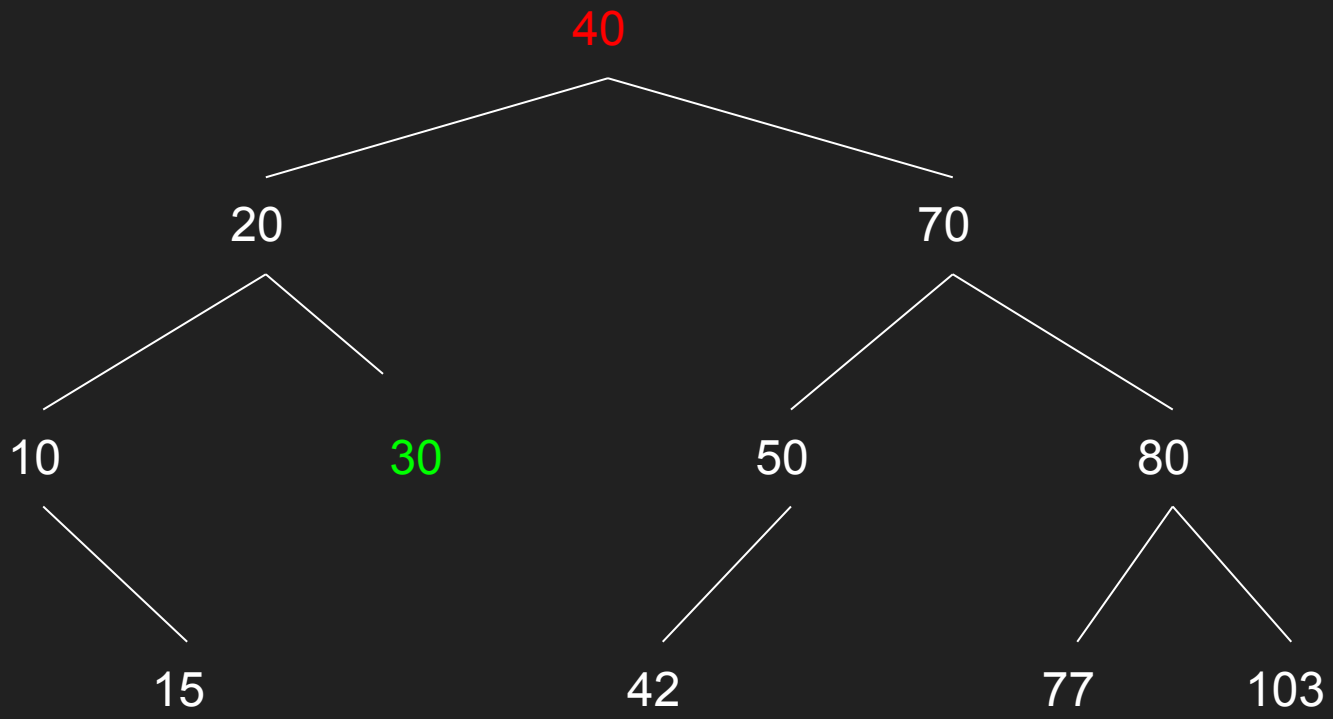
Árvores Binárias de Busca: remoção

Exemplo:



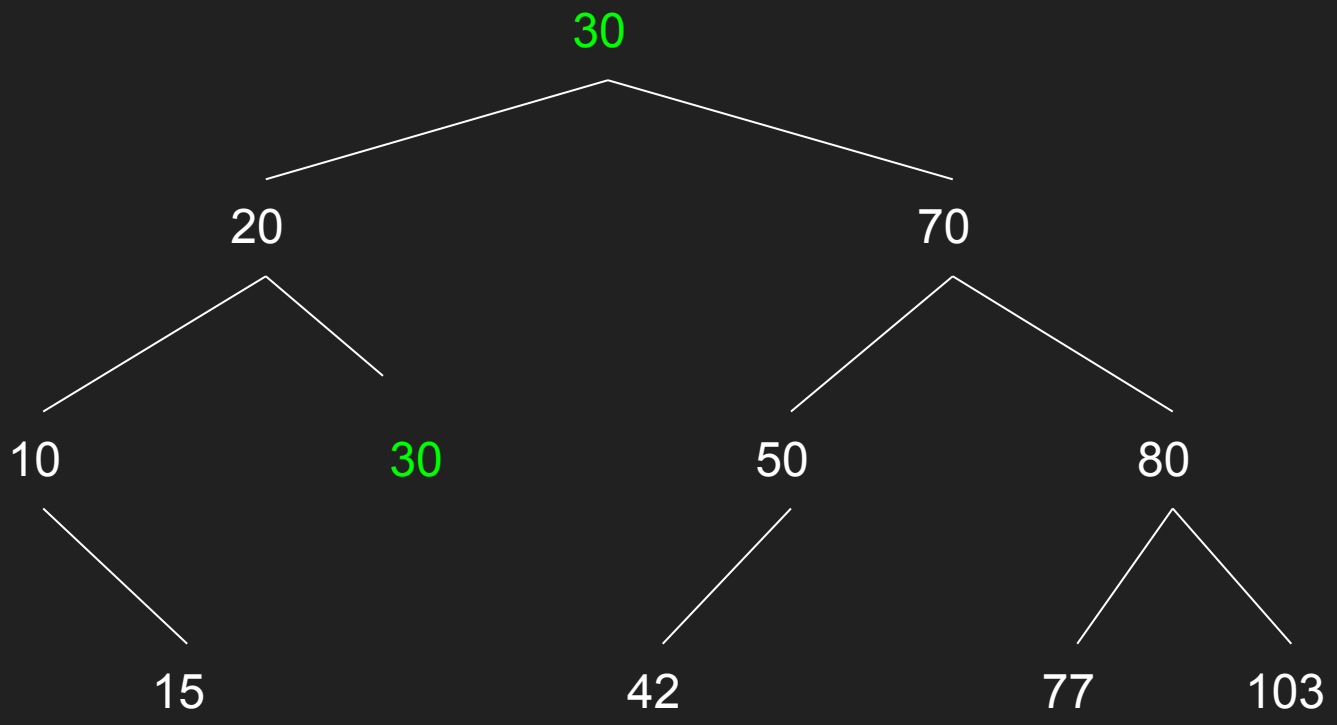
Árvores Binárias de Busca: remoção

Exemplo:



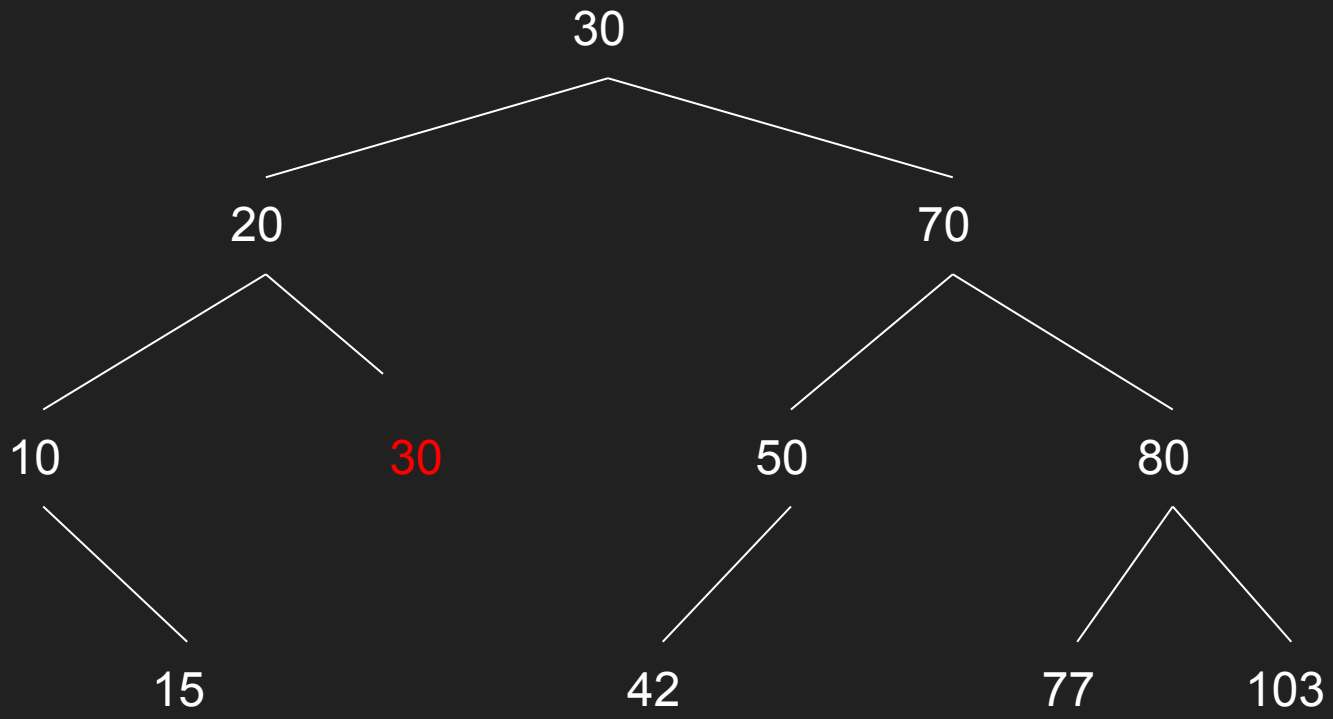
Árvores Binárias de Busca: remoção

Exemplo:



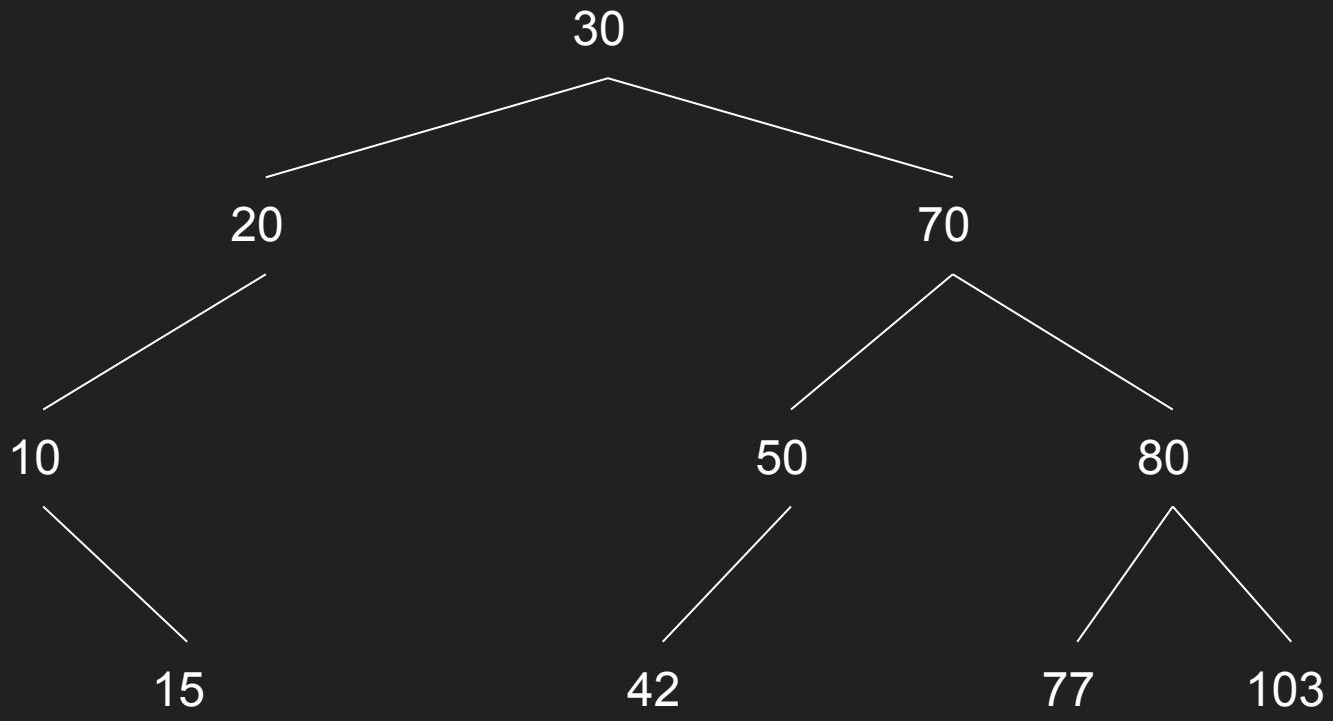
Árvores Binárias de Busca: remoção

Exemplo:



Árvores Binárias de Busca: remoção

Exemplo:



Árvores Binárias de Busca: considerações finais

Árvores binárias de busca são estruturas interessantes pois todas as operações que alteram seu estado (inserção/remoção), assim como a busca são realizadas com complexidade de tempo O(h), onde h é a altura da árvore.

Quando um árvore com n elementos está balanceada, temos $h = \Theta(\log_2 n)$, logo todas estas operações possuem complexidade logarítmica, o que torna a árvore de binária de busca uma estrutura potencialmente mais eficiente do que as demais estudadas para armazenamento, manutenção e recuperação de informações.

Árvores Binárias de Busca: considerações finais

Contudo, em uma árvore com o pior balanceamento possível (ou seja, na prática é uma lista ligada), as operações mencionadas serão O(n). Ou seja, não será uma estrutura melhor do que as listas lineares.

Árvores Binárias de Busca: considerações finais

Contudo, em uma árvore com o pior balanceamento possível (ou seja, na prática é uma lista ligada), as operações mencionadas serão O(n). Ou seja, não será uma estrutura melhor do que as listas lineares.

A boa notícia é que a altura média de uma árvore binária de busca construída de forma aleatória a partir de um conjunto de n valores distintos é $O(\log_2 n)$.

Árvores Binárias de Busca: considerações finais

Contudo, em uma árvore com o pior balanceamento possível (ou seja, na prática é uma lista ligada), as operações mencionadas serão O(n). Ou seja, não será uma estrutura melhor do que as listas lineares.

A boa notícia é que a altura média de uma árvore binária de busca construída de forma aleatória a partir de um conjunto de n valores distintos é $O(\log_2 n)$.

Logo, na média, a complexidade das operações é logarítmica.

Árvores Binárias de Busca: considerações finais

Algumas comparações:

	busca	inserção	remoção
lista sequencial	O(n)	O(n)	O(n)
lista seq. ordenada	$O(\log_2 n)$	O(n)	O(n)
lista ligada	O(n)	O(n) / Θ(1)*	O(n) / Θ(1)*
pilha	-	Θ(1)	Θ(1)
fila	-	Θ(1)	Θ(1)
árvore bin (propósito geral)	O(n)	O(n) / Θ(1)*	O(n) / O(lg n)*
árvore binária de busca*	$O(\log_2 n)$	$\Theta(\log_2 n)$	$\Theta(\log_2 n)$