Algoritmos e Estruturas de Dados

Igor Montagner

qua 18 dez 2024 19:00:21 -03

Vamos definir nossa linguagem para trabalhar com árvores nesta aula. Dado um nó x de nossa árvore,

- $x.left$ é o filho esquerdo de x. Se ele não existir seu valor é NIL
- $x.right$ é o filho direito de x. Se ele não existir seu valor é NIL
- $x.key$ é o valor que x representa na árvore.

Árvores são estruturas naturalmente recursivas. Se tormarmos um nó x qualquer de uma árvore podemos falar em uma *subárvore enraizada em x*. Assim, podemos definir uma árvore como:

- um nó raiz r com uma chave $r.key$
- um apontador para uma subárvore esquerda $r.left$
- um apontador para uma subárvore direita $r.right$

A figura (fonte abaixo é um bom resumo do vocabulário usado em árvores.

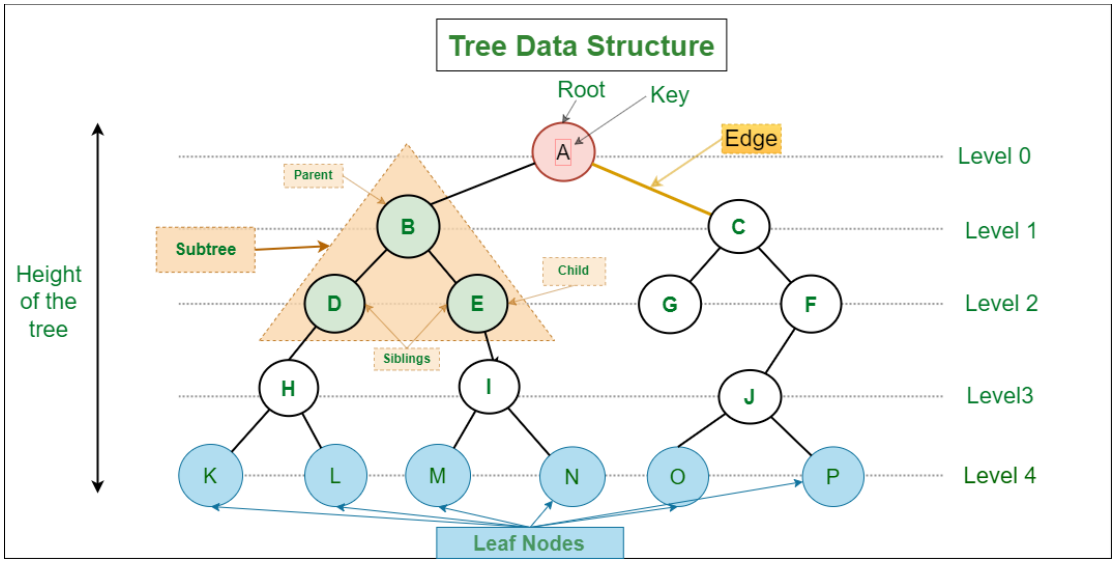


Figure 1: Fonte: https://www.geeksforgeeks.org/introduction-to-tree-data-structure-and-algorithm-tutorials/

Uma *Árvore de Busca Binária* é umaárvore especial que guarda seus elementos em ordem crescente de *key*. Para isso ela tem a seguinte propriedade:

Propriedade básica da ABB

Para todo nó x em uma ABB:

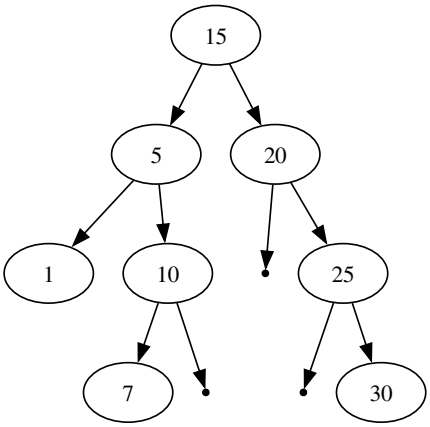
- $x.key \geq l.key$ para todo nó l na subárvore esquerda de x
- $x.key \leq r.key$ para todo nó r na subárvore direita de x

Se uma árvore não respeitar as propriedades acima então ela não é uma ABB.
Isso facilita muito checarmos se um elemento está na árvore.

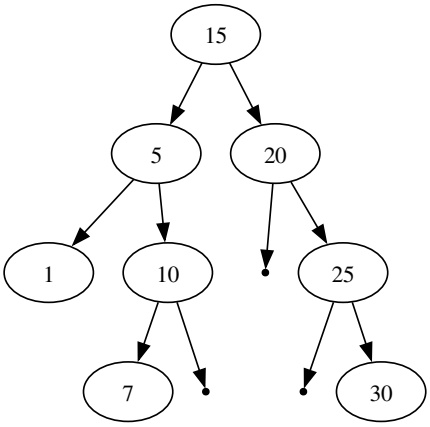
Buscando por elementos

A busca (*query*) é a operação mais básica de uma ABB e envolve dizer se existe um elemento com chave k na árvore. É também um ótimo lugar para começarmos a entender como aplicar a *Propriedade básica da ABB*.

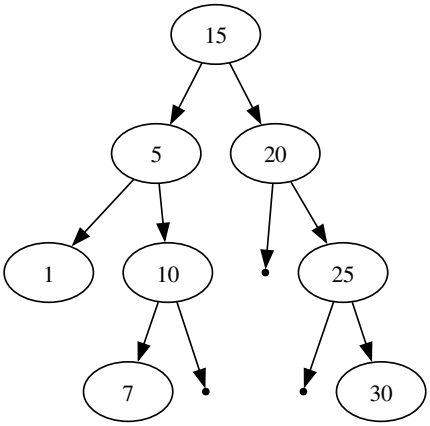
Faremos a busca do valor **7** na árvore abaixo. Desenhe em cima da árvore as decisões tomadas em cada nó, começando na raiz e descendo até encontrar o valor **7**.



Agora faça a busca pelo valor **27**. Desenhe em cima da árvore as decisões tomadas em cada nó.
O que acontece quando chegamos em um nó raiz?



Agora faça a busca pelo valor **13**. Desenhe em cima da árvore as decisões tomadas em cada nó.
Como descobrimos que esse valor **não** está na árvore?



Vamos agora formalizar essas simulações no algoritmo `QUERY(R, K)` que busca o valor `K` na árvore enraizada em `R`.

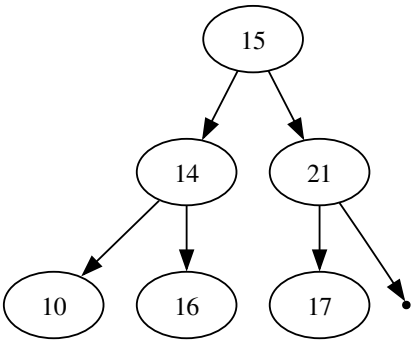
Versão iterativa:

Versão recursiva:

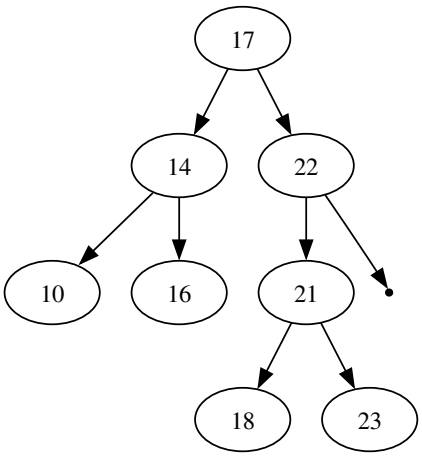
Validar uma *ABB*

Vamos começar vendo alguns exemplos de árvores que não são *ABB*. Para cada árvore abaixo, indique os nós em que **propriedade básica da ABB** não vale. Se não lembra a propriedade, consulte a página 1 :)

Árvore 1



Árvore 2



Anote nas margens como fez para checar se qual nó não cumpre a propriedade.

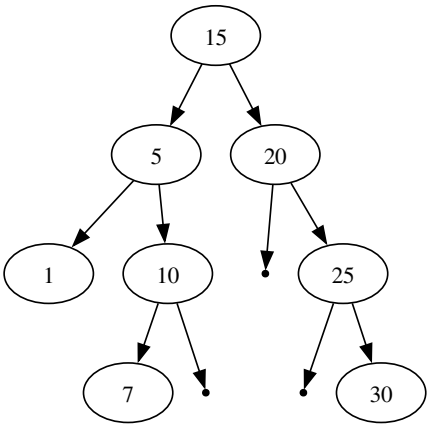
Percebemos que um algoritmo para isso precisa de várias partes!

1. precisamos checar se a propriedade é válida para todos os nós
2. para checar a propriedade precisamos percorrer toda a subárvore esquerda e toda a subárvore direita, novamente percorrendo todos os nós.

Em árvores temos 3 maneiras "clássicas" de percorrer todos os nós:

1. **pré-ordem**: analisa primeiro o nó raiz, depois a subárvore esquerda e por último a subárvore direita.
2. **pós-ordem**: analisa primeiro a subárvore esquerda, depois a direita e por último o nó raiz.
3. **em-ordem**: analisa primeiro a subárvore esquerda, depois o nó raiz e por último a subárvore direita.

Note que as ordens induzem a criação de algoritmos recursivos: passar por todos os nós de uma árvore enraizada em `R` implica passar por todos os nós de outras duas árvores (seus ramos esquerdo e direito). Use a árvore abaixo para simular essas três ordens de visitação.



- **pré-ordem**:
- **pós-ordem**:
- **em-ordem**:

Formalize agora abaixo as três ordens de visitação. Indique que o nó raiz está sendo processado chamando a função auxiliar `FAZ_ALGO(R)`.

Pré-ordem:

Pós-ordem:

Em-ordem:

Atividades Práticas e Implementação

Agora é hora de exercitar os conceitos vistos nessa atividade. Acesse o PrairieLearn e faça os exercícios deste handout. Nos exercícios de código, use os algoritmos escritos no handout para prosseguir. Para o exercício de validação, pense em como os pseudo-códigos de ordem de visitação podem ajudar a criar um algoritmo para resolver este problema.

Todos exercícios de código do PrairieLearn usam o seguinte `struct`. Ele é basicamente idêntico à maneira que escrevemos nosso pseudo-código, então portar os algoritmos para *C* deve ser algo relativamente fácil/rápido.

```
typedef struct _TreeNode {
    struct _TreeNode *left, *right;
    int key;
} TreeNode;
```