Árvores

HELLO!

I am Fabiano Utiyama

fabianoutiyama@gmail.com

1

Introdução

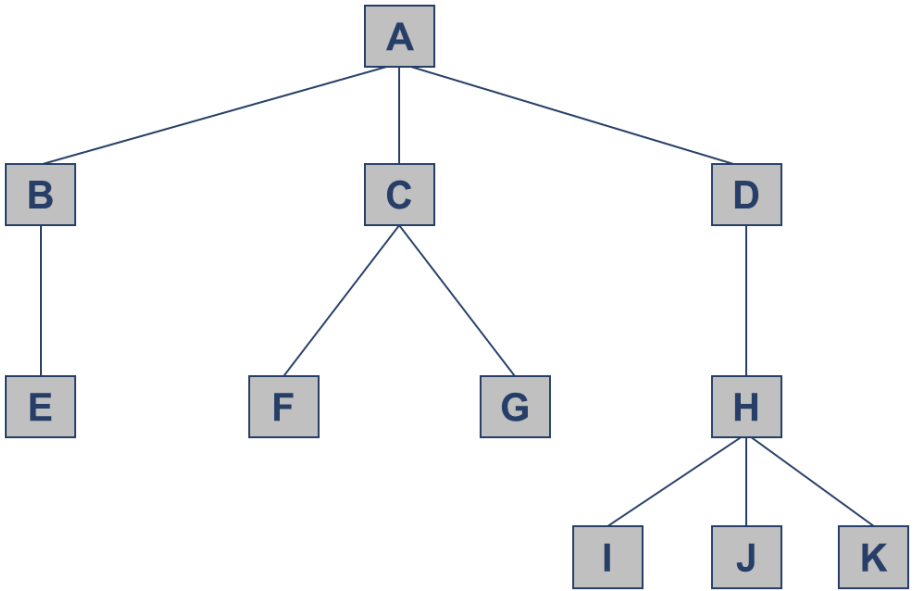
Let’s start!

Árvores são estruturas de dados que caracterizam uma relação entre os dados que a compõem

A relação existente entre os dados (nós) de uma árvore, é uma relação de hierarquia

- Formalmente, uma árvore é um conjunto finito T de um ou mais nós, tais que:
  a. Existe um nó denominado raiz da árvore;
  b. Os demais nós formam $m \geq 0$ conjuntos separados T_1, T_2, ..., T_m, onde cada um destes conjuntos é uma árvore. As árvores T_i $(1 \leq i \leq m)$ recebem a denominação de subárvores.

- Esquematicamente, uma árvore pode ser representada da seguinte forma:

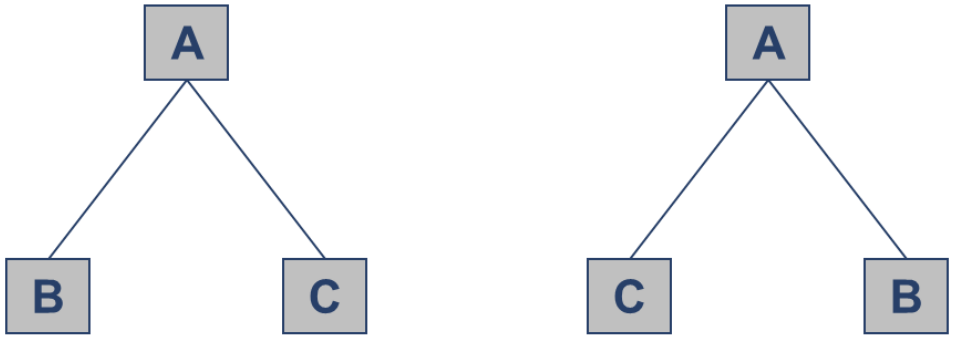


- Terminologia:
  - No exemplo anterior, os quadrados representam os nós da árvore, cuja **raiz** é o nó ‘A’;
  - Pela definição de árvore, cada nó da árvore é raiz de uma subárvore. O número de subárvores de um nó é o **grau** deste nó;
  - O **grau de uma árvore** é igual ao grau do nó de maior grau pertencente à mesma;
  - Um nó de grau zero é chamado de **folha** ou **nó terminal**;
  - O **nível** do nó é definido como sendo o igual ao número de nós que o separam da raiz;
  - A **altura** de uma árvore é definida como sendo o nível mais alto da árvore;
  - Um conjunto de zero ou mais árvores disjuntas é chamado de **floresta**;

- Em relação a árvore descrita anteriormente, pode-se observar:

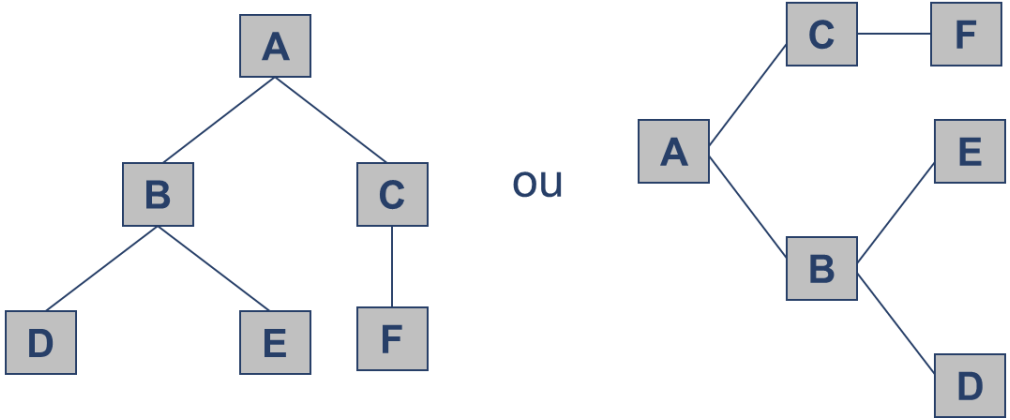
Nodo	Grau	Nível	Observações
A	3	0	Raiz da Árvore
B	1	1	
C	2	1	
D	1	1	
E	0	2	Nó folha
F	0	2	Nó folha
G	0	2	Nó folha
H	3	2	
I	0	3	Nó folha
J	0	3	Nó folha
K	0	3	Nó folha

- Quando a ordem das subárvores é significativa, a árvore é chamada de ordenada. Neste caso, há diferença entre as seguintes árvores:

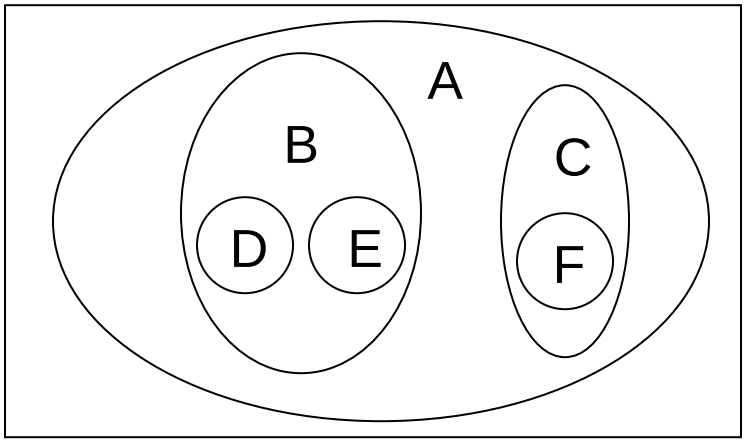


- Entretanto, quando a ordem das subárvores não é relevante, diz-se que a árvore é orientada, uma vez que apenas a orientação dos nós é importante (neste caso as duas árvores mostradas acima são iguais);

- A raiz de uma árvore é chamada de **pai** das raízes de suas subárvores. As raízes das subárvores de um nó são chamadas de **irmãos** que, por sua vez, são **filhos** de seu nó pai.
- Formas de Representação de Árvores:
  - Árvore convencional (grafos)

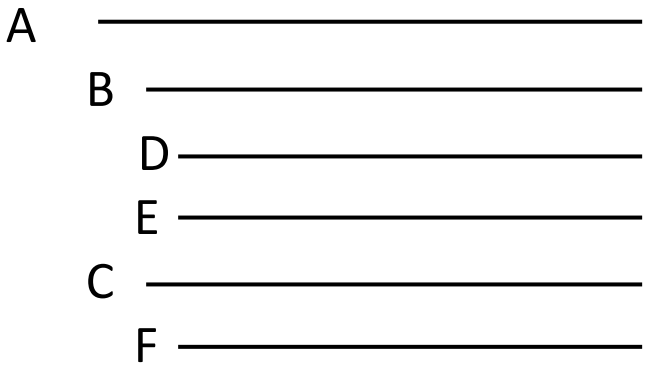


- Conjuntos Aninhados ou Diagramas de Venn:



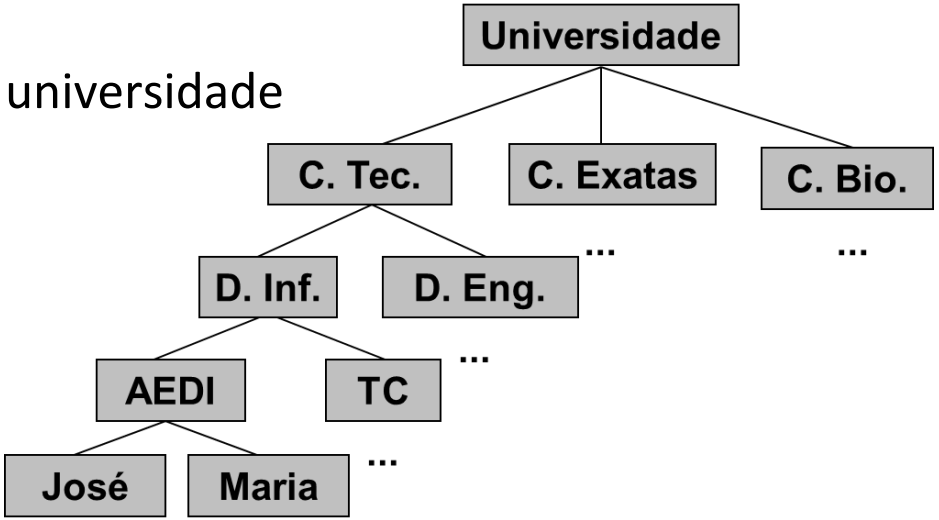
- Parênteses Aninhados:
  (A (B (D) (E)) (C (F)))

- Barramento ou Tabelas:



- Notação Decimal (DEWEY):
  1.A, 1.1.B, 1.1.1.D, 1.1.2.E, 1.2.C, 1.2.1.F.

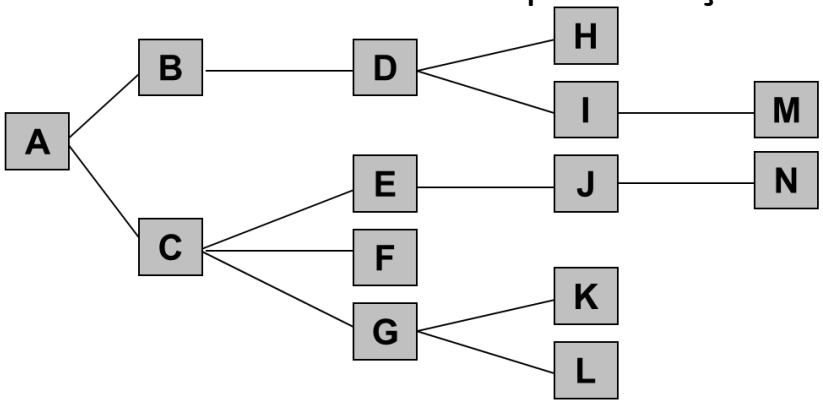
- Aplicações de árvores:
  - Representações genealógicas;
  - Representação de objetos que possuem relação hierárquica.
    - Exemplo: uma universidade



- Aplicações:
  - Índices de arquivos;
  - Árvores genealógicas ou hereditárias;
  - Organização de empresa (organograma);
  - Avaliação de expressões;
  - Árvores de decisão;
  - Busca em tabelas;
  - Algoritmos e Classificação

- Aplicações:
  - Índices de arquivos;
  - Árvores genealógicas ou hereditárias;
  - Organização de empresa (organograma);
  - Avaliação de expressões;
  - Árvores de decisão;
  - Busca em tabelas;
  - Algoritmos e Classificação

- Exercício: dada a seguinte árvore, encontre:
  - A raiz da árvore;
  - Todos os nós folha;
  - O grau e o nível de cada nó;
  - A altura da árvore;
  - Todas as relações entre nós (irmão, pai, filho);
  - Descreva a árvore com todas as representações estudadas.



2

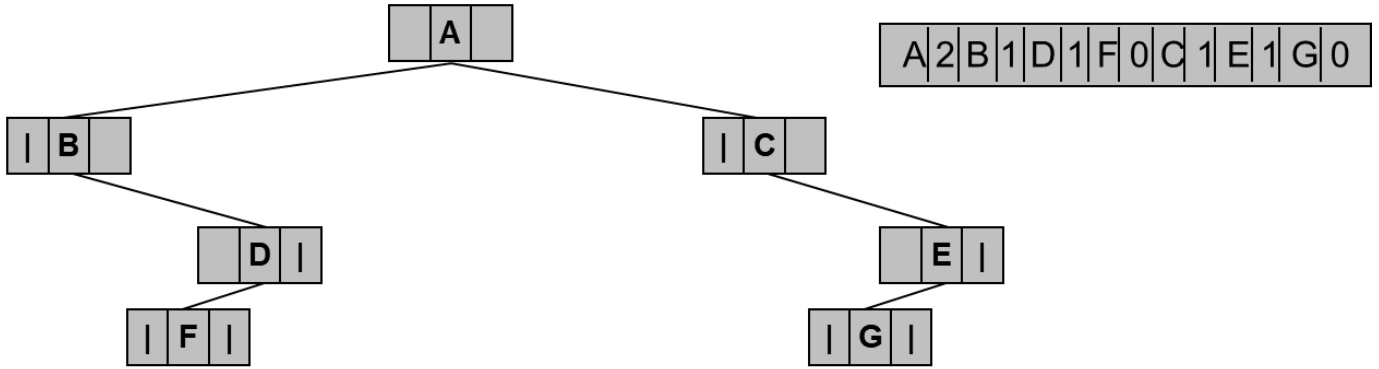
Árvores Binárias

Let’s start!

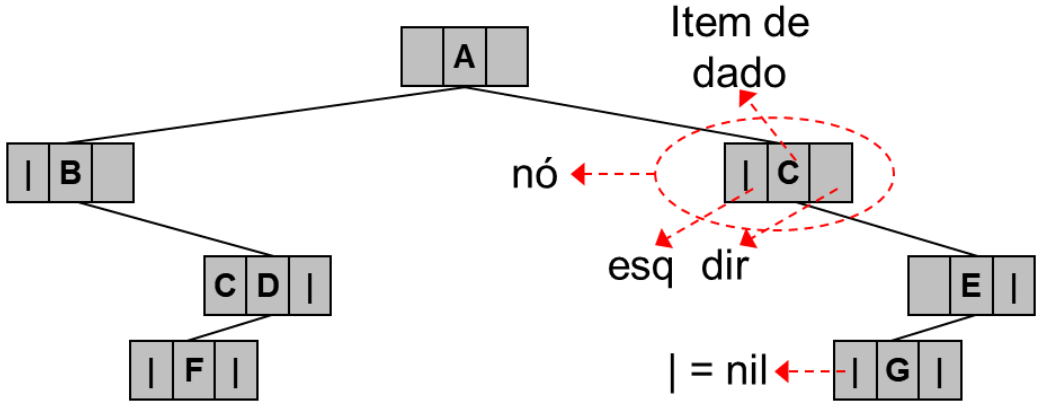
“Conjunto finito de nós que, ou é vazio, ou consiste de uma raiz ligando duas outras árvores binárias.”

- São árvores onde o grau de cada nó é menor ou igual a dois;
- As subárvores de cada nó são chamadas subárvore esquerda e subárvore direita;
- Assim, se um nó possuir apenas uma subárvore, ela deve ser estabelecida como direita ou esquerda;
- Uma árvore binária pode ser vazia, isto é, não possuir nenhum nó;

- Alocação:
  - Por adjacência:
    - Nós representados seqüencialmente na memória;
    - Esta alocação não é conveniente na maioria dos casos, pois dificulta a manipulação da estrutura;



- Alocação:
  - Encadeada:
    - Mais adequada;
    - Permite melhor manipulação dos dados com diversas ordens de acesso aos nós;
    - Os nós são alocados dinamicamente;

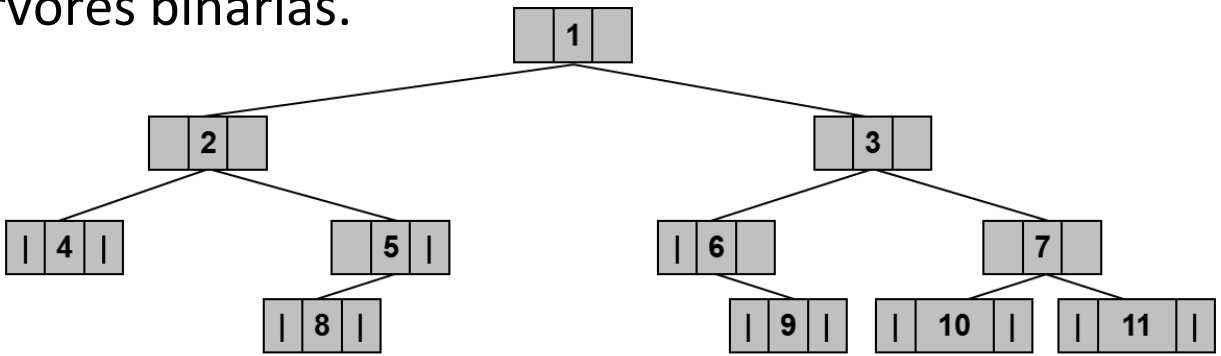


- Caminhamento em Árvores Binárias:
  - É a forma de percorrer todos os nós da árvore, com o objetivo de consultar ou alterar suas informações;
  - Existem vários métodos de tal forma que cada nó seja “visitado” apenas uma vez;
  - Um completo percurso da árvore nos dá um arranjo linear sobre os nós;
  - São três os principais caminhos utilizados para percorrer uma árvore binária: visitar os nós em ordem **pré-fixada**, **pós-fixada**, ou **in-fixa** (in-ordem).

- Formas de caminhar:
  - Pré-ordem: RED
    - Visitar a raiz;
    - Percorrer a subárvore esquerda;
    - Percorrer a subárvore direita;
  - In-ordem: ERD (percorre as chaves em ordem crescente)
    - Percorrer a subárvore esquerda;
    - Visitar a raiz;
    - Percorrer a subárvore direita;
  - Pós-ordem: EDR (ou forma polonesa)
    - Percorrer a subárvore esquerda;
    - Percorrer a subárvore direita;
    - Visitar a raiz;

Obs.: o termo visita indica alguma manipulação sobre o nó.

- Exemplo: dada a seguinte árvore, verifique a seqüência de nos percorridos segundo as três formas de caminhar sobre árvores binárias.



Pré-ordem: 1, 2, 4, 5, 8, 3, 6, 9, 7, 10, 11
In-ordem: 4, 2, 8, 5, 1, 6, 9, 3, 10, 7, 11
Pós-ordem: 4, 8, 5, 2, 9, 6, 10, 11, 7, 3, 1 (polonesa)

- Algoritmos de travessia em árvores binárias
  - observe que os procedimentos são recursivos, devido à natureza recursiva da estrutura

```
void pre_ordem (Apontador T)
{
 if (T!=NULL)
  {
   printf("Item: %d",T->Reg.chave);
   pre_ordem (T->Esq);
   pre_ordem (T->Dir);
  }
}
```

```
void in_ordem (Apontador T)
{
 if (T!=NULL)
  {
   in_ordem (T->Esq);
   printf("Item: %d",T->Reg.chave);
   in_ordem (T->Dir);
  }
}
```

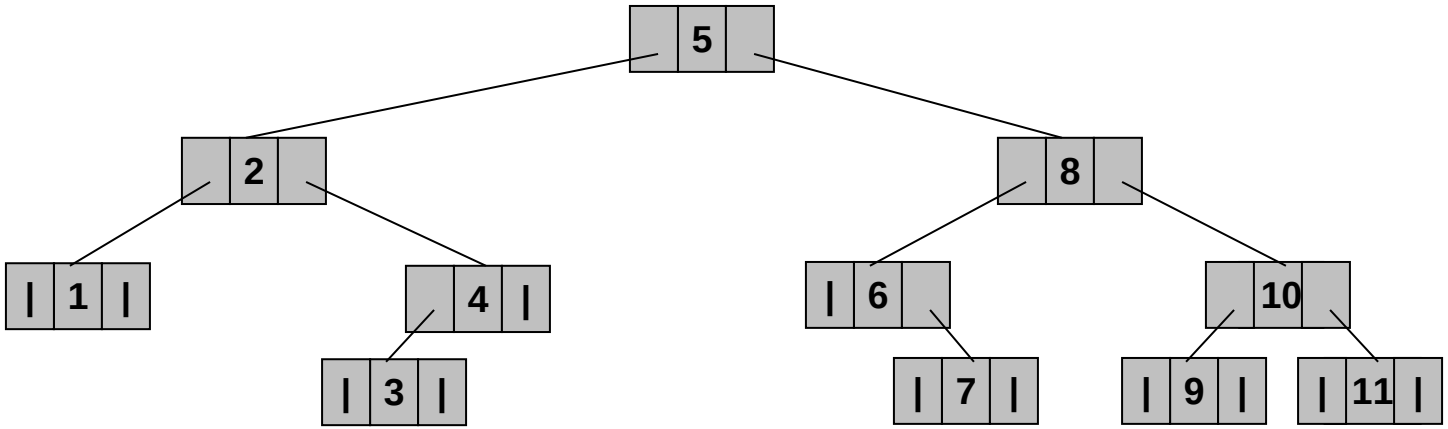
```
void pos_ordem (Apontador T)
 {
  if (T!=NULL)
   {
    pos_ordem (T->Esq);
    pos_ordem (T->Dir);
    printf("Item: %d",T->Reg.chave);
   }
 }
```

- Algoritmos de travessia em árvores binárias
  - observe que os procedimentos são recursivos, devido à natureza recursiva da estrutura

- Árvore Binária de Busca ou Árvore de Pesquisa:
  - Uma ABB para um subconjunto S é uma árvore binária com rótulos no qual cada vértice v está rotulado com elementos $e(v) \in S$ |:
    1. Para cada vértice μ na subárvore Esq de $v \Rightarrow e(\mu) < e(v)$;
    2. Para cada vértice μ na subárvore Dir de $v \Rightarrow e(\mu) > e(v)$;
    3. Para cada elemento $a \in S$, existe exatamente um vértice $v | e(a)=v$.

- Em resumo, uma árvore binária de pesquisa é uma árvore binária onde cada nó interno possui um registro, tal que:
  - todo registro alocado na sua subárvore esquerda é menor do que o nó pai;
  - e todo registro alocado na subárvore direita é maior do que o nó pai.

- Exemplo:



- Procedimentos para uma árvore binária de busca:
  - Inicialização, inserção e remoção;
- Para isto, é preciso utilizar os processos recursivos de busca da árvore;
  - Procurasse um elemento Y na raiz, se ele não for encontrado deve-se procurá-lo na subárvore esquerda caso ele seja menor que a raiz, ou na subárvore direita se ele for maior que a raiz;
- Nos procedimentos de alteração, remoção e consulta a busca deve ter sucesso, nos procedimentos de inserção a busca deve fracassar;

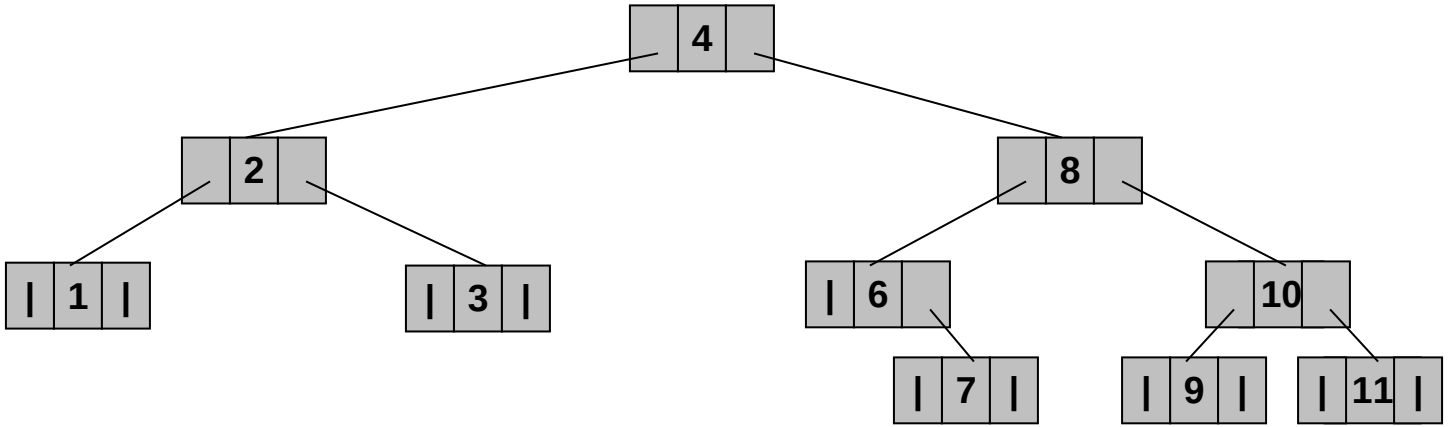
- A inserção começa com uma busca, procurando pelo valor, mas se não for encontrado, procuram-se as subárvores da esquerda ou direita, como na busca. Eventualmente, alcança-se a folha, inserindo-se então o valor nesta posição. Ou seja, a raiz é examinada e introduz-se um nó novo na subárvore da esquerda se o valor novo for menor do que a raiz, ou na subárvore da direita se o valor novo for maior do que a raiz

- A fim de introduzir um nó novo na árvore, seu valor é primeiro comparado com o valor da raiz. Se seu valor for menor que a raiz, é comparado então com o valor do filho da esquerda da raiz. Se seu valor for maior, está comparado com o filho da direita da raiz. Este processo continua até que o nó novo esteja comparado com um nó da folha, e então adiciona-se o filho da direita ou esquerda, dependendo de seu valor

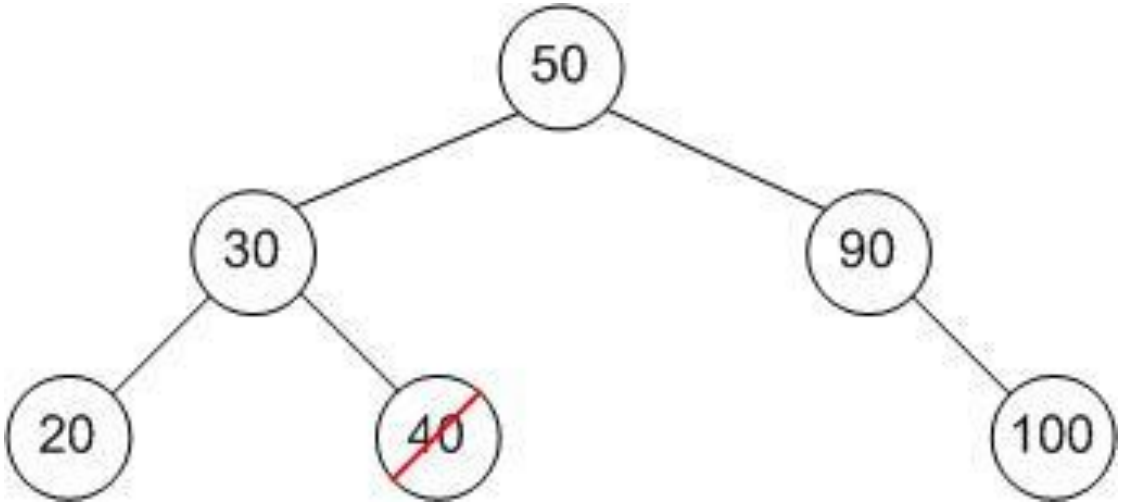
```
void insereR (t_no *&raiz, t_aluno al)
{
    if (raiz==NULL)
    {
            raiz=(t_no*)malloc(sizeof(t_no));
            raiz->info=al;
            raiz->esq=NULL;
            raiz->dir=NULL;
    }
    else
    {
            if (al.ra < raiz->info.ra)
            {
                    insereR(raiz->esq,al);
            }
            else
            {
                    insereR(raiz->dir,al);
            }
    }
}
```

- Remoção: para se criar este procedimento deve-se fazer uma análise. Pois, se o elemento a ser removido tiver apenas um descendente, a remoção será simples. Mas se o elemento a ser removido tiver dois descendentes, ele deverá ser substituído por aquele que estiver mais a direita em sua subárvore esquerda (maior dos menores); ou por aquele que estiver mais a esquerda em sua subárvore direita (menor dos maiores).

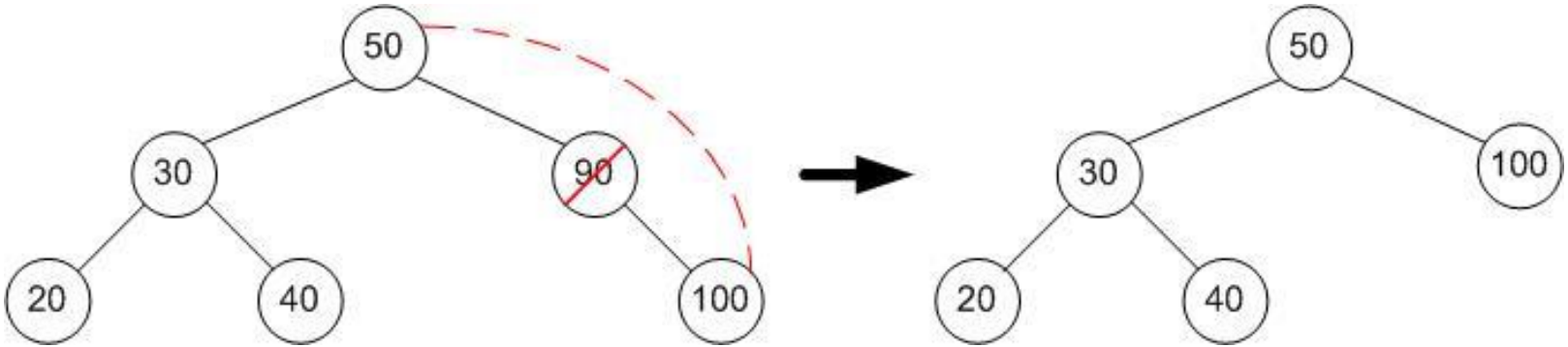
- Exemplo: na árvore da transparência 25, se removêssemos o nó com chave 5, poderíamos substituí-lo pelo nó com chave 4 (como mostra a figura abaixo), ou pelo nó com chave 6.



- Exclusão na folha
  - A exclusão na folha é a mais simples, basta removê-lo da árvore

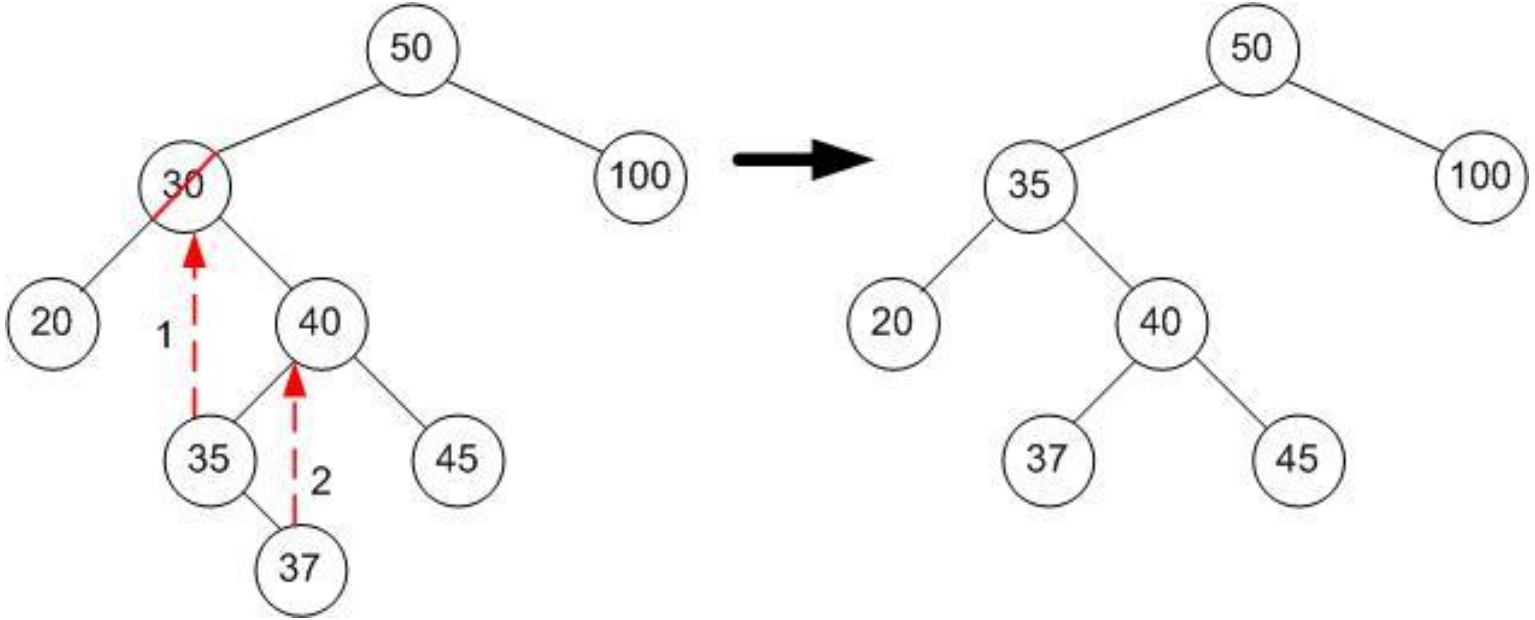


- Exclusão de um nó com um filho
  - Excluindo-o, o filho sobe para a posição do pai



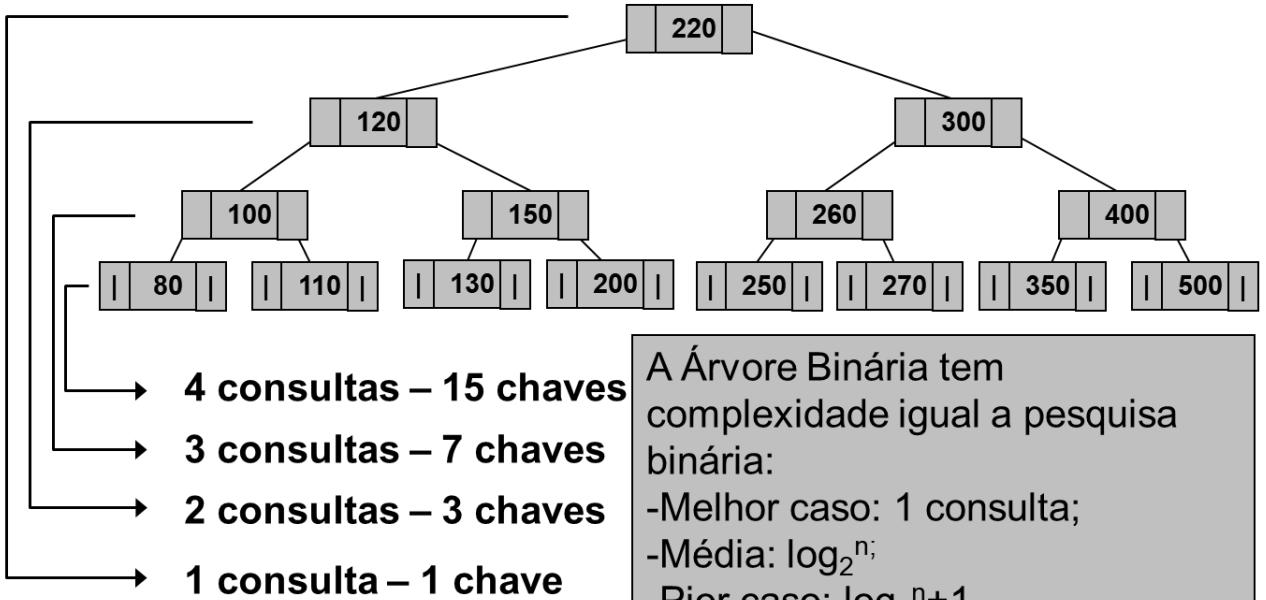
- Exclusão de um nó com dois filhos
  - Neste caso, pode-se operar de duas maneiras diferentes. Pode-se substituir o valor do nó a ser retirado pelo valor sucessor (o nó mais à esquerda da subárvore direita) ou pelo valor antecessor (o nó mais à direita da subárvore esquerda), removendo-se aí o nó sucessor (ou antecessor)

- Exclusão de um nó com dois filhos



```
void remove (t_no *&raiz, int ra)
{
	if (raiz==NULL)
	{
		printf ("Nao achou o ra.\n");
		getch();
	}
	else if (raiz->info.ra==ra) //Achou o elemento a ser excluido
	{
	 if (raiz->esq==NULL && raiz->dir==NULL) //Nao tem filhos
	 {
		free(raiz);
		raiz=NULL;
	 }
	 else if (raiz->dir==NULL) //Tem filho somente a esquerda
	 {
		t_no *aux=raiz;
		raiz=raiz->esq;
		free(aux);
	 }
	 else if (raiz->esq==NULL) //Tem filho somente a direita
	 {
		t_no *aux=raiz;
		raiz=raiz->dir;
		free(aux);
	 }
	 else //Tem os dois filhos
	 {
		t_no *aux=raiz;
		aux=aux->dir;
		while (aux->esq!=NULL)
		{
							aux=aux->esq;
		}
		raiz->info=aux->info;
		remove (raiz->dir,aux->info.ra);
	 }
	}
	else if (ra < raiz->info.ra)
	{
		remove (raiz->esq,ra);
	}
	else
	{
		remove (raiz->dir,ra);
	}
}
```

- Ordem de complexidade da árvore binária:



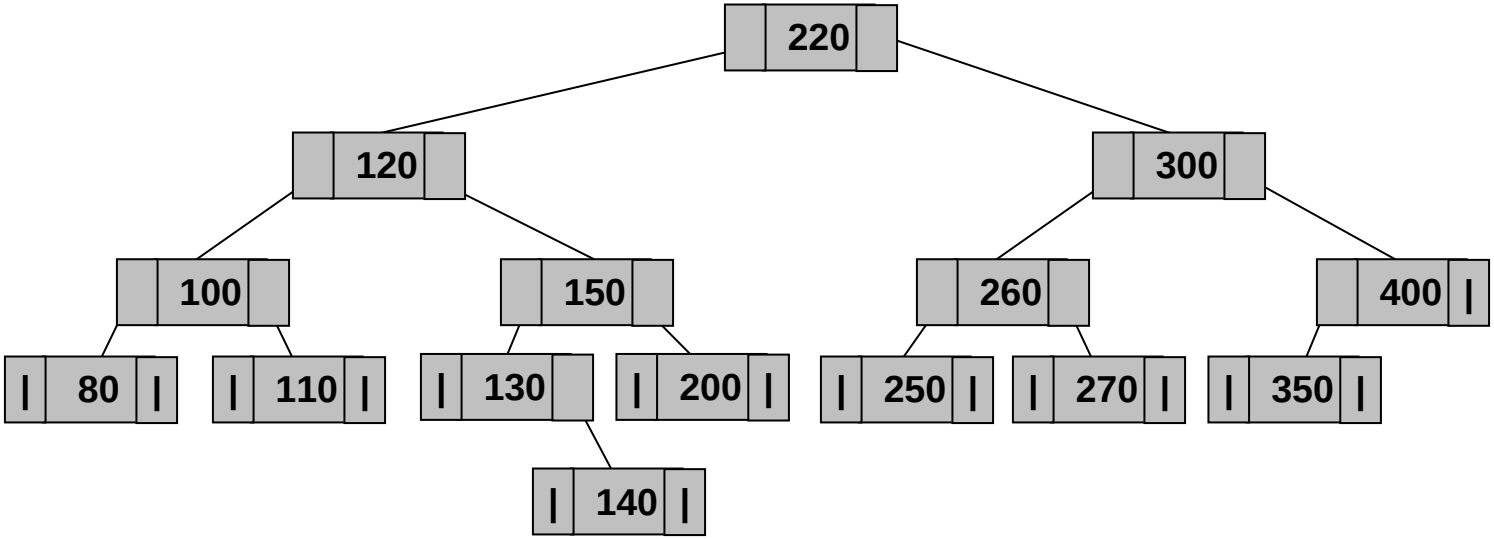
A Árvore Binária tem complexidade igual a pesquisa binária:
-Melhor caso: 1 consulta;
-Média: $\log_2^{n;}$
-Pior caso: $\log_2^{n}+1$.
Onde n é o número de elementos armazenados na árvore.

- Balanceamento:
  - Busca uma distribuição equilibrada dos nós;
  - Busca otimizar a consulta;
  - Busca minimizar o número médio de comparações necessário para a localização de uma chave.

- Balanceamento por altura:
  - Busca-se minimizar a altura da árvore;
- Árvore Completamente Balanceada:
  - Uma árvore é completamente balanceada quando a distância média dos nós até a raiz for mínima;
  - Uma árvore binária é dita completamente balanceada se, para cada nó, o número de nós de suas subárvores diferem de no máximo, 1;
  - Árvore completamente balanceada é a árvore com menor altura para o seu número de nós.

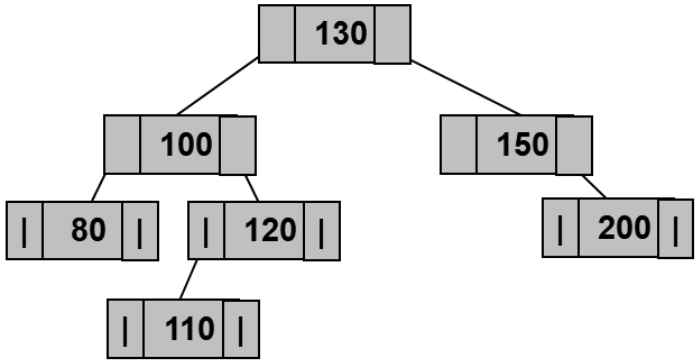
- Árvores não completamente balanceadas:
  - Uma árvore balanceada é uma árvore onde a diferença de altura de qualquer subárvore é no máximo 1;
  - O grande esforço exigido para a manutenção de uma árvore completamente balanceada pode não ser compensado pelo ganho de eficiência no processo de busca;
  - Árvore não completamente balanceadas beneficiam o processo de busca, exigindo manutenção do balanceamento pouco onerosa.

- Exemplo de árvore não completamente balanceada:

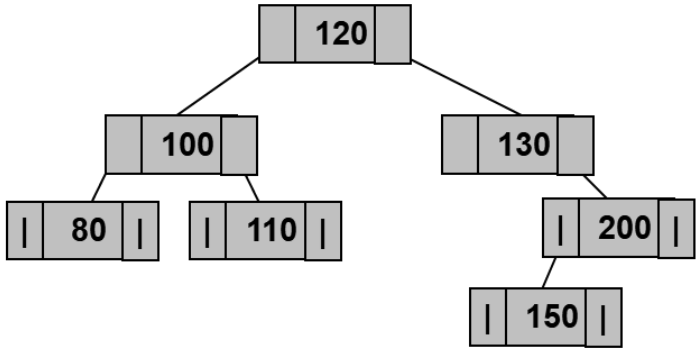


- Neste contexto, destacam-se as árvores **AVL**, concebidas em 1962, por Adel’son-Vel’skii e Landis, caracterizadas pela seguinte propriedade: *para todo nó de uma árvore AVL, a diferença entre as alturas de suas subárvores não excede a uma unidade*.

- As constantes inserções e remoções de nós de uma árvore podem provocar o desbalanceamento da mesma. Para corrigir este problema em uma árvore AVL, é necessária a aplicação de uma das quatro rotações que serão vistas a seguir.

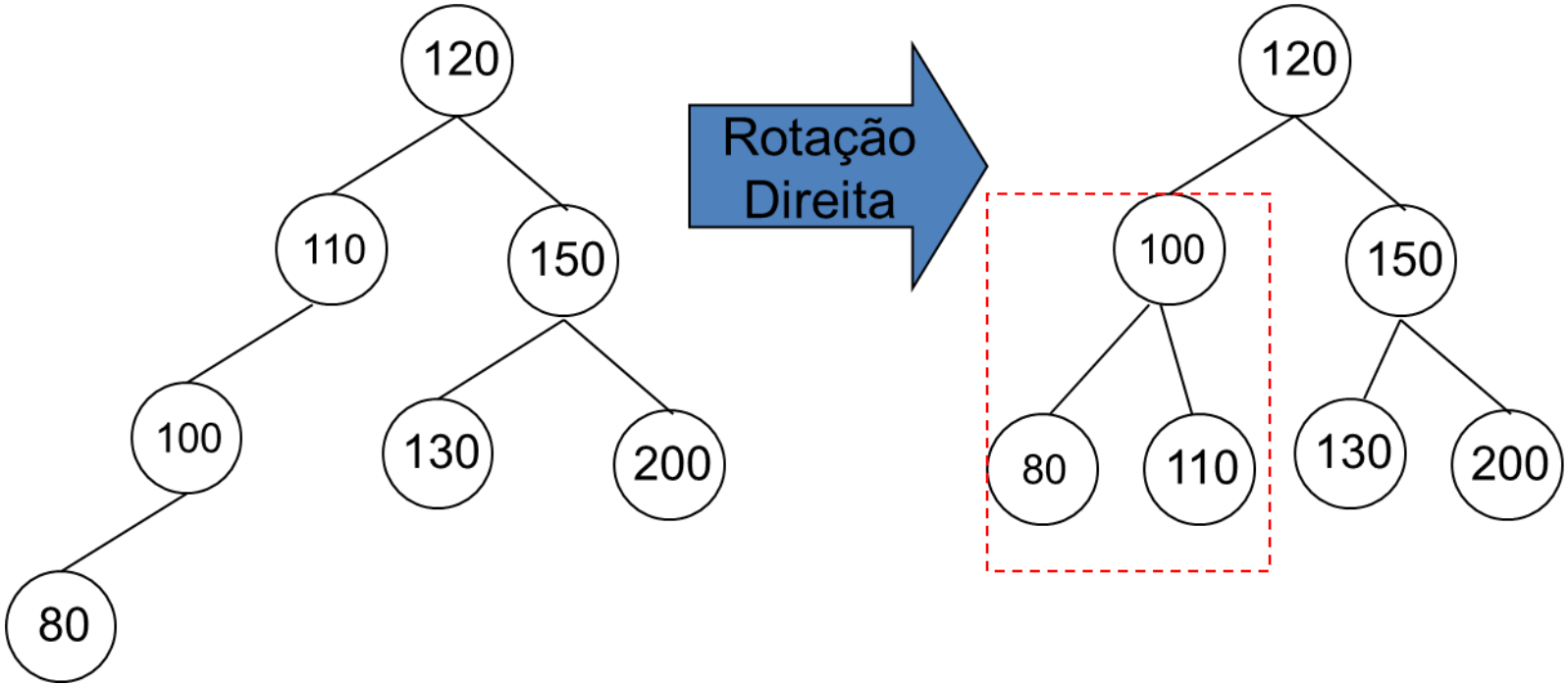


Árvore AVL

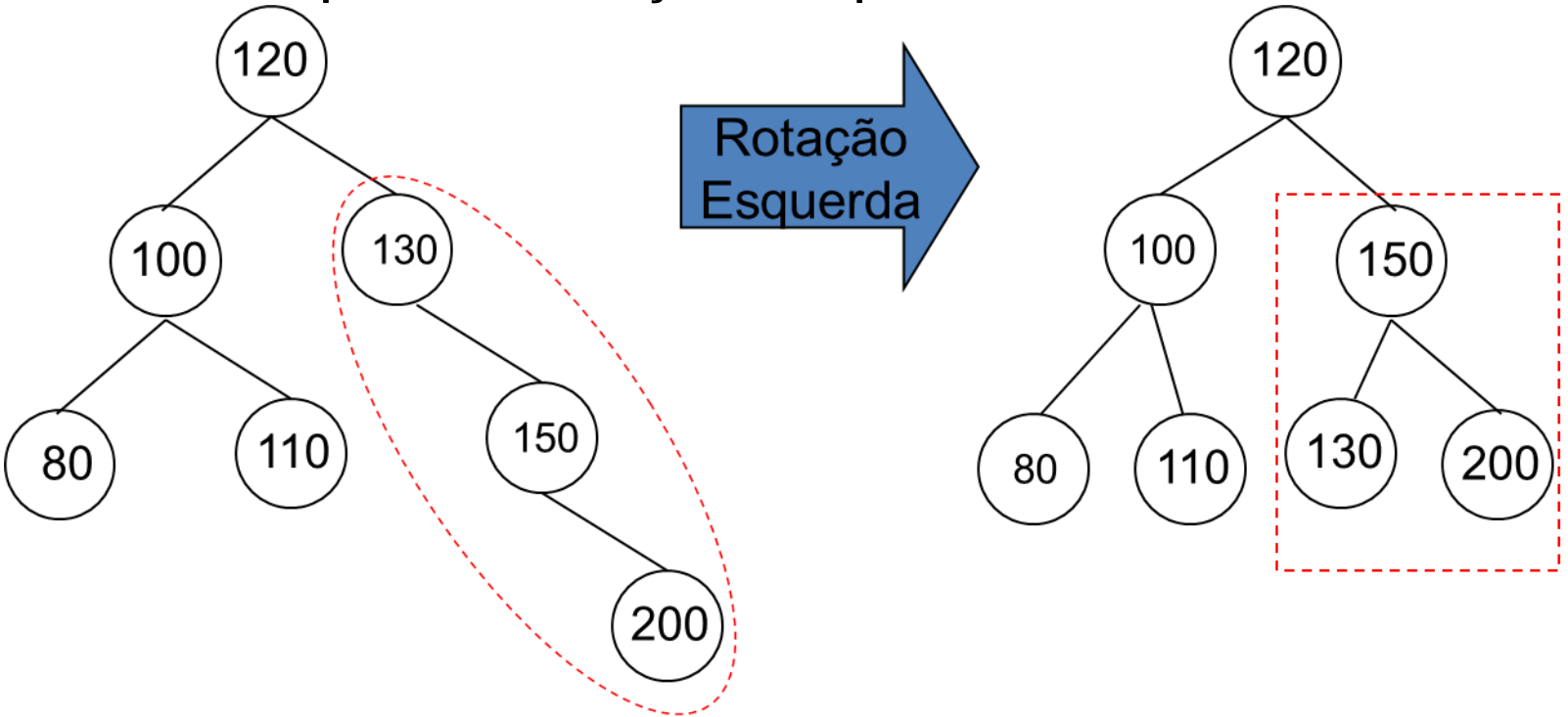


Árvore não AVL

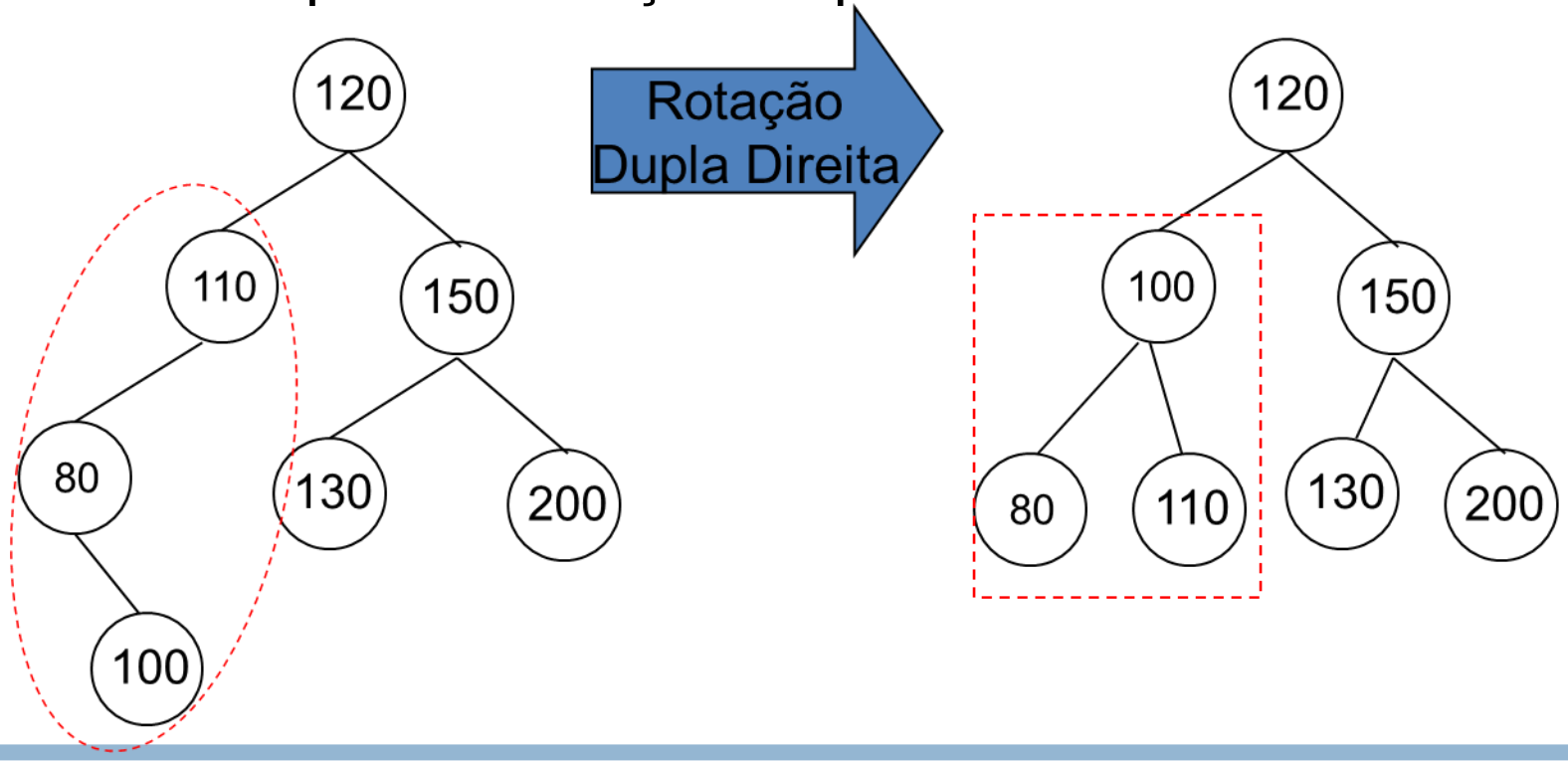
- Exemplo de Rotação Direita:



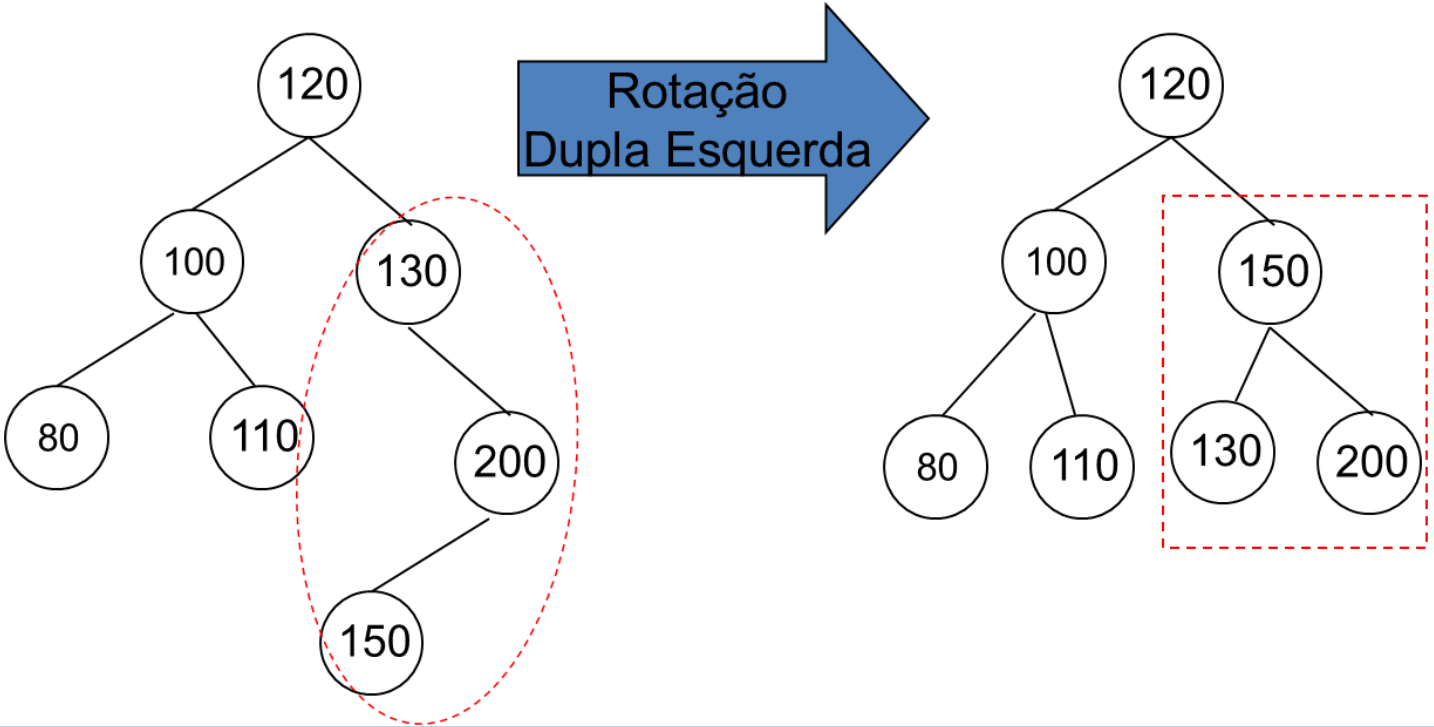
- Exemplo de Rotação Esquerda:



- Exemplo de Rotação Dupla Direita:



- Exemplo de Rotação Dupla Esquerda:



THANKS!

Any questions?

fabianoutiyama@gmail.com