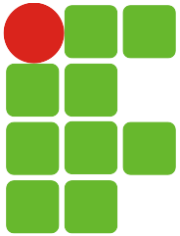
Estrutura de Dados

Profa. Marta Talitha Carvalho

Aula 6: Árvore

Árvore



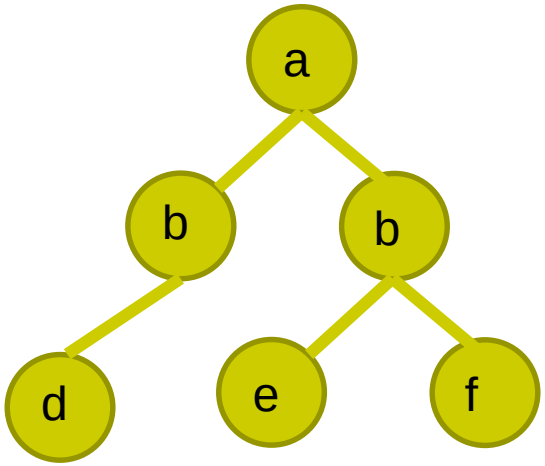
- Até agora estudamos estruturas lineares, ou seja, sequência ordenada de dados.
- Entretanto não são adequadas para representar dados hierárquicos.

Por exemplo: estrutura hierárquica do diretório do Windows.

Árvore Binária

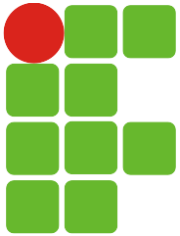
Características de um árvore?

- Representação



(a(b(d)) (c(e)(f))

Árvore



Quais são as características de uma árvore?

- **CARACTERÍSTICA 1:** Existe apenas **1 nó raiz** que contém zero ou mais árvores (nós filhos).

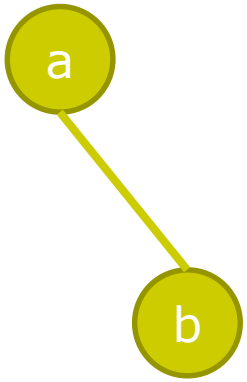
- **CARACTERÍSTICA 2:** Nós com filhos são chamados de **nós internos**.

- **CARACTERÍSTICA 3:** Nó sem filhos são chamados de **folhas** ou **nós externos**.

Árvore

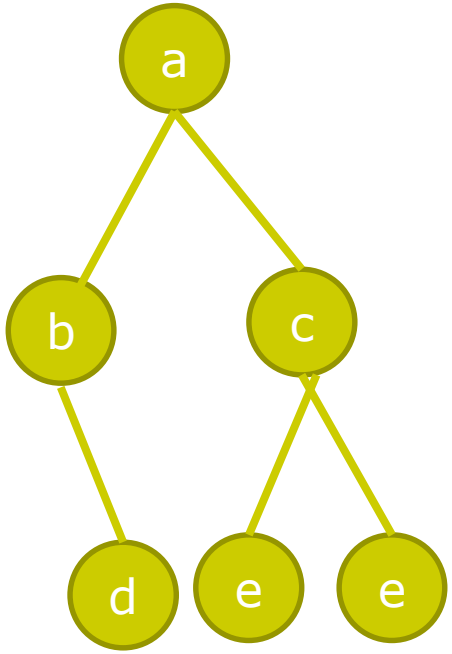
- **CARACTERÍSTICA 4: A altura da árvore**: Só existe um caminho da árvore da raiz para qualquer nó. Com isto, podemos definir a altura de uma árvore como sendo o comprimento do caminho mais longo da raiz até uma das folhas.

1



Qual é altura?

2



Árvore

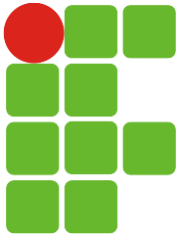
Porque saber sobre a altura?

- **A altura pode determinar a eficiência nas operações de busca, inserção e remoção de nó.**

calcular o número de nós de uma árvore **binária cheia** em função da sua altura:

$$2^{h+1} - 1$$

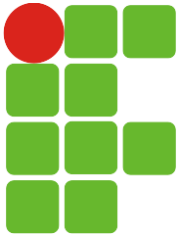
Árvore



- **CARACTERÍSTICA 5:** A forma mais natural para definir uma estrutura árvore é usando **recursividade**.

- Removendo **raiz** de uma árvore, de fato, obtemos uma coleção de árvores.

Árvore Binária



- O número de filhos permitido por nó e as informações armazenadas em cada nó diferenciam os diversos tipos de árvores.

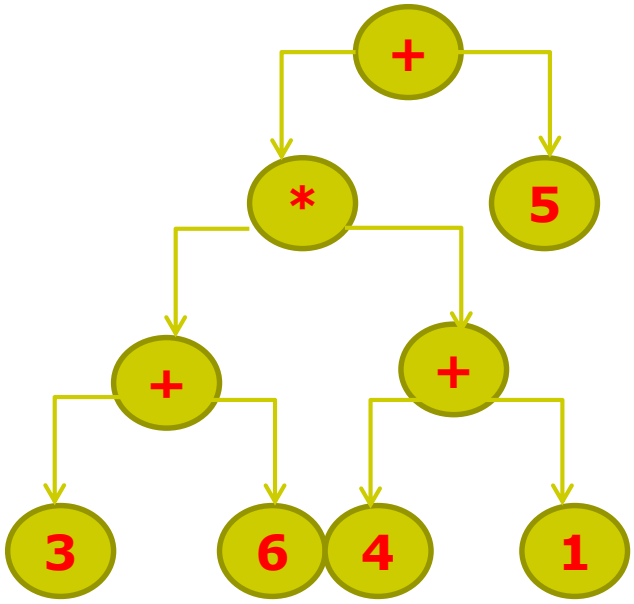
x: árvore B, **árvore binária de busca** ou árvore binária de pesquisa (alguns chamam também de arvore de pesquisa binária), árvore AVL, etc.

Árvore Binária

Podemos pensar no seguinte exemplo de utilização: REPRESENTAÇÃO DE UMA EXPRESSÃO.

(3+6) * (4+1) + 5

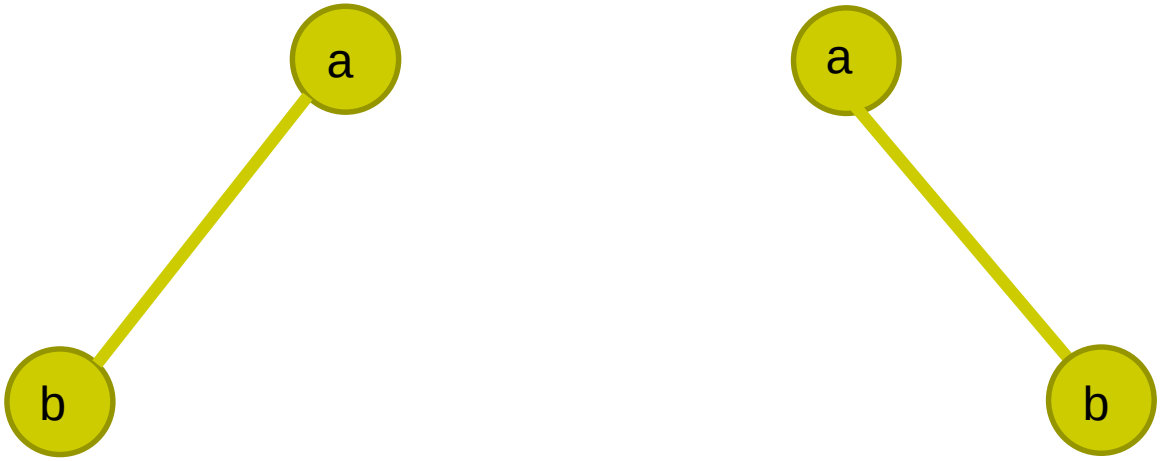
Lembre-se que usamos pilha para armazenar. Por que neste caso é melhor usar a estrutura árvore?



Árvore Binária

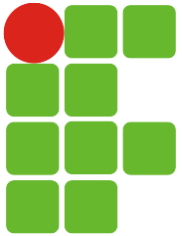
Características da árvore binária?

- **CARACTERÍSTICA 1:** Numa subárvore binária, distingui-se uma subárvore esquerda de uma direita. Exemplo:



São consideradadas diferentes, ou seja, enquanto a primeira tem a **sad** a outra tem a **sae.**

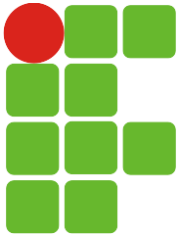
Árvore Binária



CARACTERÍSTICA 2: As árvores binárias possuem no máximo dois filhos por nó. Isso é chamados GRAU de uma árvore.

- O GRAU de uma árvore binária é 2.

Árvore Binária



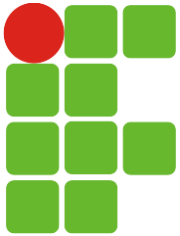
Representação em C:

- Ex: Uma árvore cujos valores armazenados são caracteres:

```
struct arv{
        char info;
        struct arv* esq;
        struct arv* dir;
}
typedef struct arv Arv;
```

- Da mesma forma que uma lista encadeada é representada para o primeiro nó, a estrutura da árvore como um todo é **representada por um ponteiro para o nó raiz**.

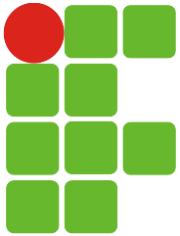
Árvore Binária



Operação com árvore binária:

```
Arv* inicializa(){
    return NULL;
}
```

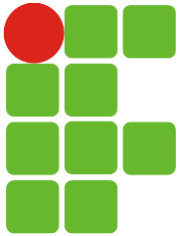
Árvore Binária



Operação com árvore:

```
Arv* cria(char c, Arv* sae, Arv* sad){
      Arv* p = (Arv*) malloc(sizeof(Arv));
      p->info =c;
      p->esq = sae;
      p->dir = sad;
      return p;
}
```

Árvore Binária



- As duas funções: inicializa e cria representam os dois casos da definição recursiva de árvores binária:

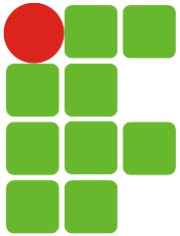
Uma árvore binária (Arv* a) é

vazia (a=inicializa()) => Condição Parada

ou é composto por raiz e duas

subárvores(a=cria(c,sae,sad))=> Chamada recursiva

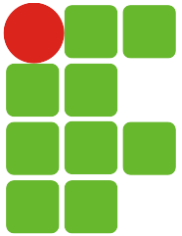
Árvore Binária



- Assim com posse dessas informações podemos criar árvores mais complexas:

```
Arv* a1 = cria('d', inicializa(), inicializa());
Arv* a2 = cria('b', inicializa(),a1);
Arv* a3 = cria('e', inicializa(), inicializa());
Arv* a4 = cria('f', inicializa(), inicializa());
Arv* a5 = cria('c', a3, a4);
Arv* a6 = cria('a', a2, a5);
```

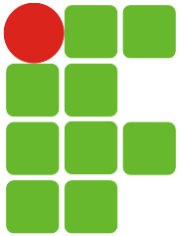
Árvore Binária



- Alternativamente, a árvore poderia ser criada com uma única atribuição, seguindo a sua estrutura, "recursivamente".

```
Arv* a = cria('a',  cria('b' , inicializa(),
                                  cria('d',
                                          inicializa(),
                                          inicializa())
                                  )
                                  cria('c',
                                          cria('e',
                                                  inicializa(),
                                                  inicializa()),
                                          cria('f',
                                                  inicializa(),
                                                  inicializa()))

                );
```

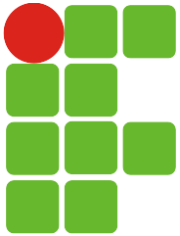
Árvore Binária



- Outras operações:

```
int vazia(Arv* a){
        return a == NULL;
}


void imprime(Arv* a){
        if (!vazia(a)){
                printf("%c", a->info);
                imprime(a->esq);
                imprime(a->dir);
        }
}
```

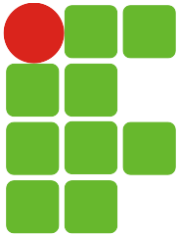
Árvore Binária



- Retorna a árvore atualizada:

```
Arv* libera(Arv* a) {
        if (!vazia(a)){
                libera(a->esq);
                libera(a->dir);
                free(a);
        }
        return NULL;
}
```

Árvore Binária

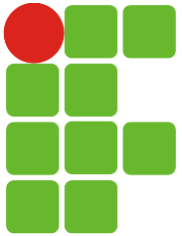


- Buscando a ocorrência de um caracter na árvore:

```
int busca(Arv* a,char c){
        if(vazia(a))
                return 0;
        else
                return a->info==c || busca(a->esq,c) || busca(a->dir,c);
```

Usando o operador || (“ou”) faz com que a busca seja interrompida assim que o elemento é encontrado. Isto acontece porque se a== a->info for verdadeiro, as duas outras expressões não chegam a ser avaliadas.

Árvore Binária



- Ordem de percurso em árvores binárias:

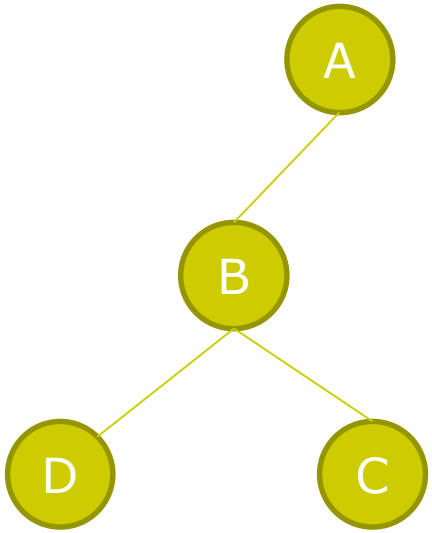
```
imprime(a->esq);
imprime(a->dir);
printf("%c",a->info);
```

```
libera(a->esq);
libera(a->dir);
free(a);
```

Muitas operações em árvores binárias envolvem o percurso em subárvores, executando alguma ação de tratamento em cada nó, de forma que é comum percorrer uma árvore em uma das seguintes ordens:

- Pré-ordem= **trata raiz**, percorre sae, percorre sad.
- Ordem simétrica = percorre sae, **trata raiz**, percorre sad.
- Pós-ordem = percorre sae, percorre sad, **trata raiz**.

Árvore Binária



Pré-ordem: A,B,D,C

Pós-ordem:D,C,B,A

Simétrica: D,B,C,A

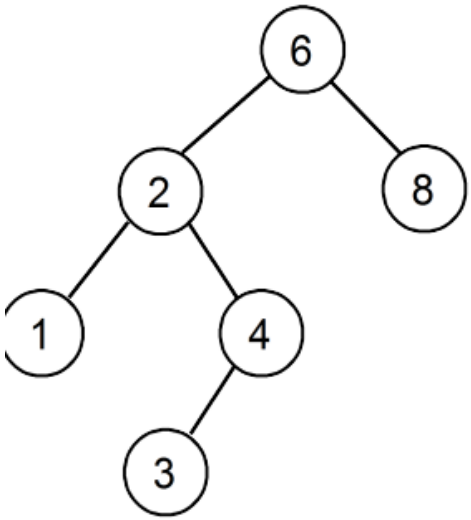
Árvore Binária de Busca

- Quando uma árvore é binária de busca:

- o valor associado à raiz é sempre maior que o valor associado a qualquer nó da sub-árvore à esquerda (sae) e

- o valor associado à raiz é sempre menor ou igual (para permitir repetições) que o valor associado a qualquer nó da sub-árvore à direita (sad)

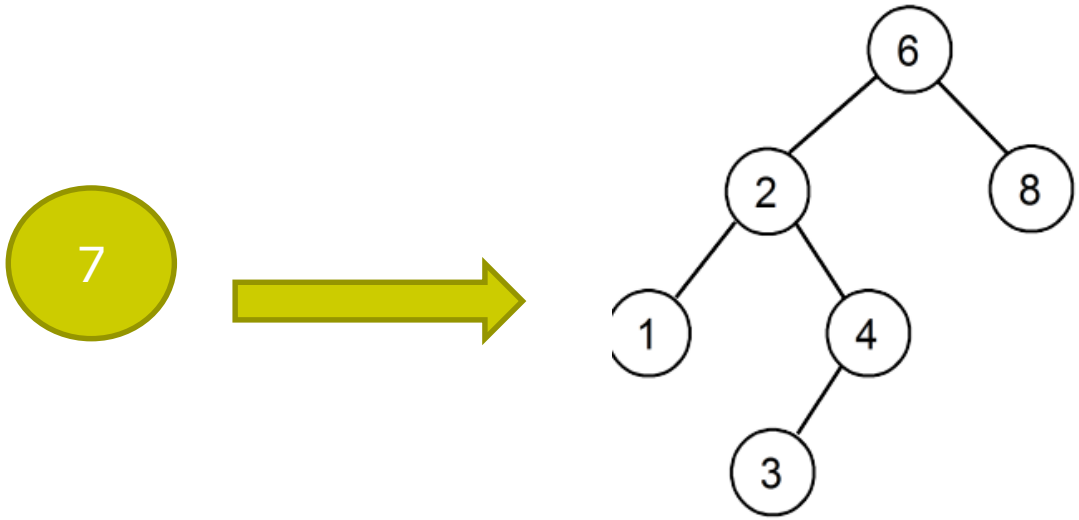
- quando a árvore é percorrida em ordem simétrica (sae - raiz - sad), os valores são encontrados em ordem não decrescente.



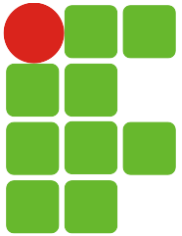
Árvore Binária de Busca

- Quero inserir ou buscar um novo nó?

- compare o valor dado com o valor associado à raiz
- se for igual, o valor foi encontrado
- e for menor, a busca continua na sae
- se for maior, a busca continua na sad



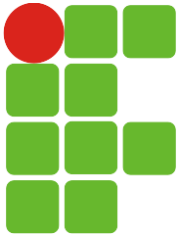
Árvore Binária de Busca



- Arvore.h

```
/*Tipo Árvore*/
typedef struct arv Arv;

/*Procedimentos Utilizados*/
Arv* inicializa(void);
void imprime(Arv* a);
int vazia(Arv* a);
Arv* insere(Arv* a, int v) ;
Arv* libera(Arv* a) ;
Arv* busca (Arv* a, int v);
Arv* excluir(Arv* a, int v);
```

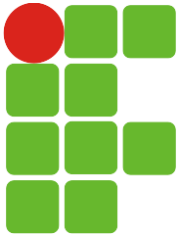
Árvore Binária de Busca



- Arvore.c

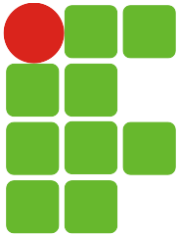
```
/*Estrutura Árvore*/
struct arv{
        int info;
        struct arv* esq;
        struct arv* dir;
};


/*Inicializa uma arvore*/
Arv* inicializa(void){
     return NULL;
}
```

Árvore Binária de Busca



```
int vazia(Arv* a){
    if (a == NULL) return 1;
    else return 0;
}
```

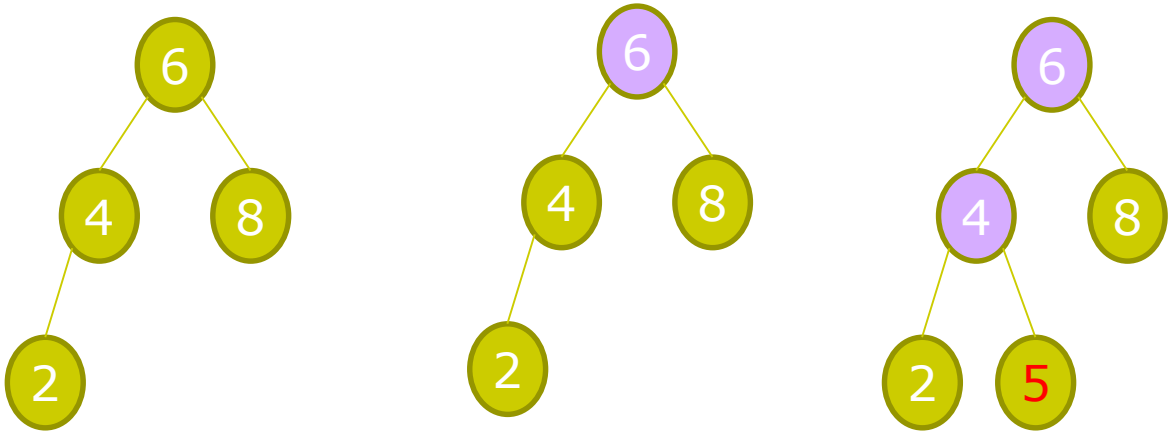
Árvore Binária de Busca



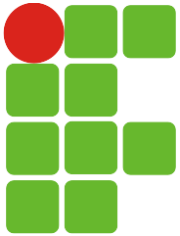
```
Arv* insere(Arv* a, int v) {
    int t;
    if (vazia(a) == 1) {
       Arv* no = (Arv*) malloc(sizeof (Arv));
       no->info= v;
       no->esq = no->dir= NULL;
       return no;
    }
    t = a->info;
    if (v < t)
       a->esq= insere (a->esq, v);
    else
       a->dir = insere (a->dir, v);
    return a;
}
```

Árvore Binária de Busca

Exemplo: Incluir número 5

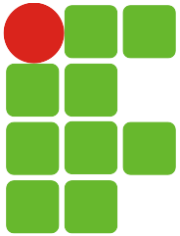


Árvore Binária de Busca



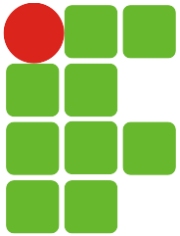
```
Arv* busca (Arv* a, int v)
{
        if (vazia(a) == 1)
                return NULL;
        else if (a->info > v)
                return busca (a->esq, v);
        else if (a->info < v)
                return busca (a->dir, v);
        else return a;
}
```

Árvore Binária de Busca



```
Arv* excluir(Arv* a, int v)
{

        if (vazia(a) == 1)
                return NULL;
        else if (a->info > v)
                a->esq = (Arv*) excluir(a->esq,v);
        else if (a->info < v)
                a->dir = (Arv*) excluir(a->dir,v);
        else{/*achou o nó a ser excluido*/

                /*nó sem filhos*/
                if (a->esq == NULL && a->dir==NULL){
                        free(a);
                        a = NULL;
                }
```

Árvore Binária de Busca

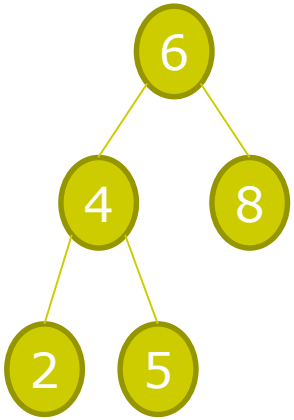


```
/*nó só tem filho à direita*/
else if (a->esq == NULL){
        Arv* t = a;
        a = a->dir;
        free(t);
}

/*só tem filho à esquerda*/
else if(a->dir == NULL){
        Arv* t = a;
        a = a->esq;
        free(t);
}
```

Árvore Binária de Busca

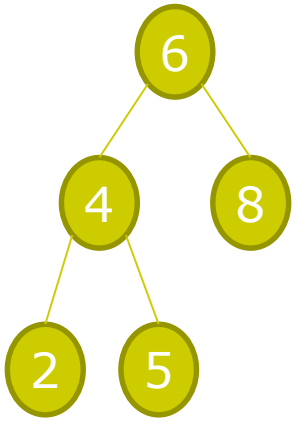
```
            /*Nó tem os dois filhos*/
            else{
                    Arv* f = a->esq;
                    while(f->dir != NULL){
                            f = f->dir;
                    }
                    a->info = f->info;
                    f->info = v;
                    a->esq = excluir(a->esq,v);


            }
    }

    return a;
}
```

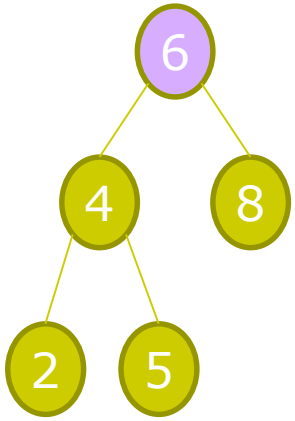
Exemplo: Excluir número 6

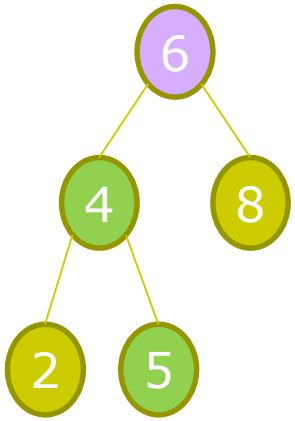


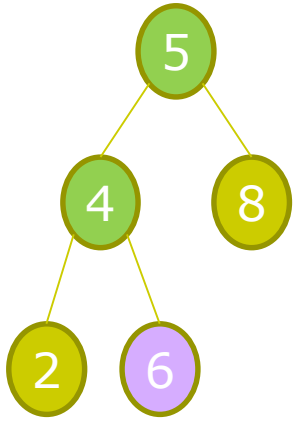
Árvore Binária de Busca

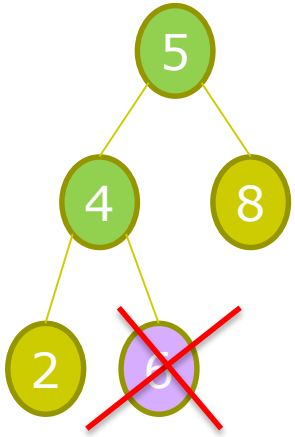
Exemplo: Excluir número 6



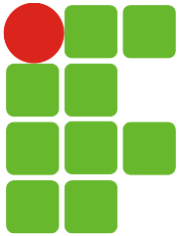






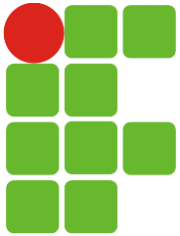


Árvore Binária de Busca



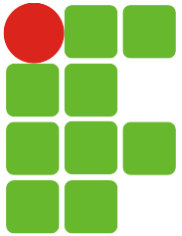
```
int main (){
        /*Inicia uma arvore A*/
        Arv* a;

        a=inicializa();
        a = insere(a,1);
        a = insere(a,9);
        a = insere(a,5);
        a = insere(a,3);
        a = insere(a,17);
        a = insere(a,3);
        imprime(a);
```

Árvore Binária de Busca



```
        if (busca(a,19) ==NULL){
                printf("\n NAO Encontrado!");
        }else{
                printf("\n Encontrado!" );
        }
        a = excluir(a,17);
        imprime(a);
        a= libera(a);


        return 0;
}
```

Árvore Balanceada

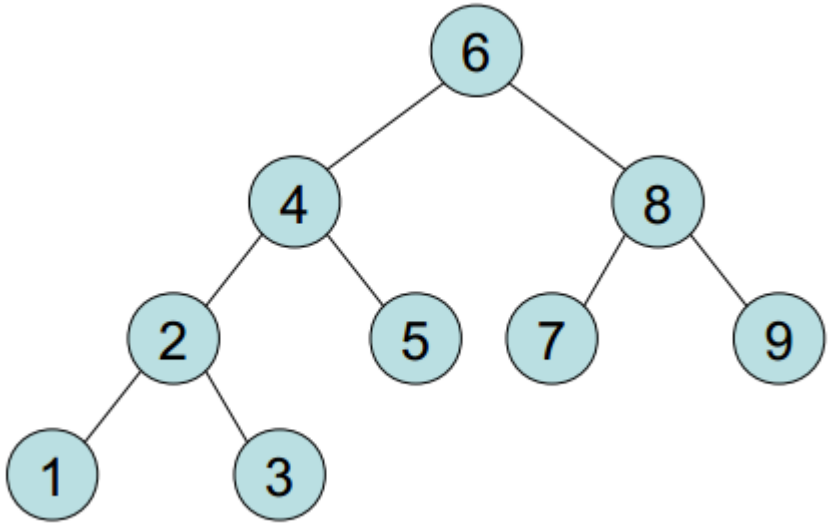


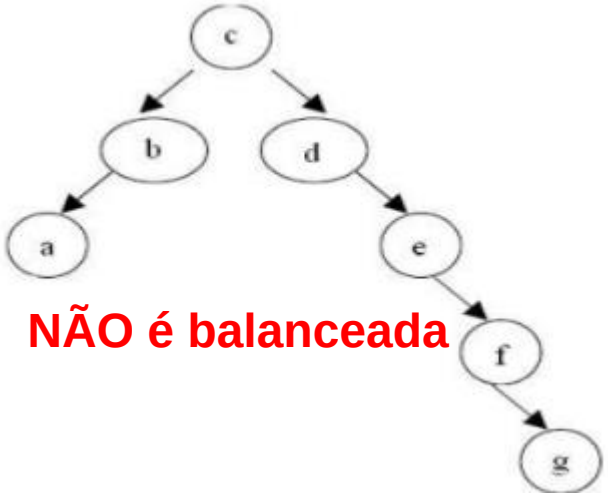
- Características: balanceada ou degeneradas.
- Uma árvore é considerada balanceada, se, e somente se, para cada um de seus nós, a altura das sub-árvores à direita e à esquerda forem iguais , ou difiram em apenas uma unidade. (árvore AVL)

- O balanceamento de um NÓ é definido como a altura de sua subárvore esquerda menos a altura de sua subárvore direita.

Árvore Balanceada

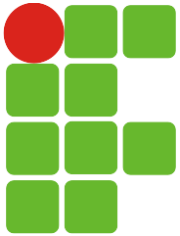
- Características: balanceada ou degeneradas.





NÃO é balanceada

Fator de Balanceamento



- Para cada nó, defini-se um fator de balanceamento (FatBal), que deve ser -1, 0 ou 1. Ele é o responsável por avisar que a árvore está desbalanceada.

- FatBal = altura (sub-árvore direita) – altura (sub-árvore esquerda)

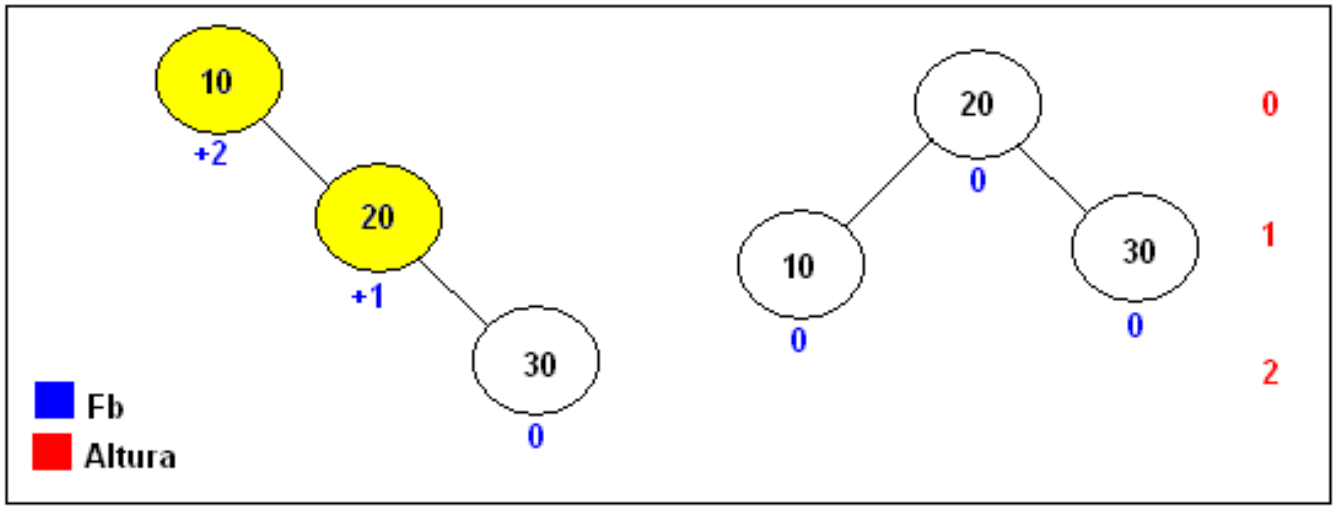
- FatBal = -1, quando a sub-árvore da esquerda é um nível mais alto que a direita.

- FatBal = 0, quando as duas sub-árvores tem a mesma altura.

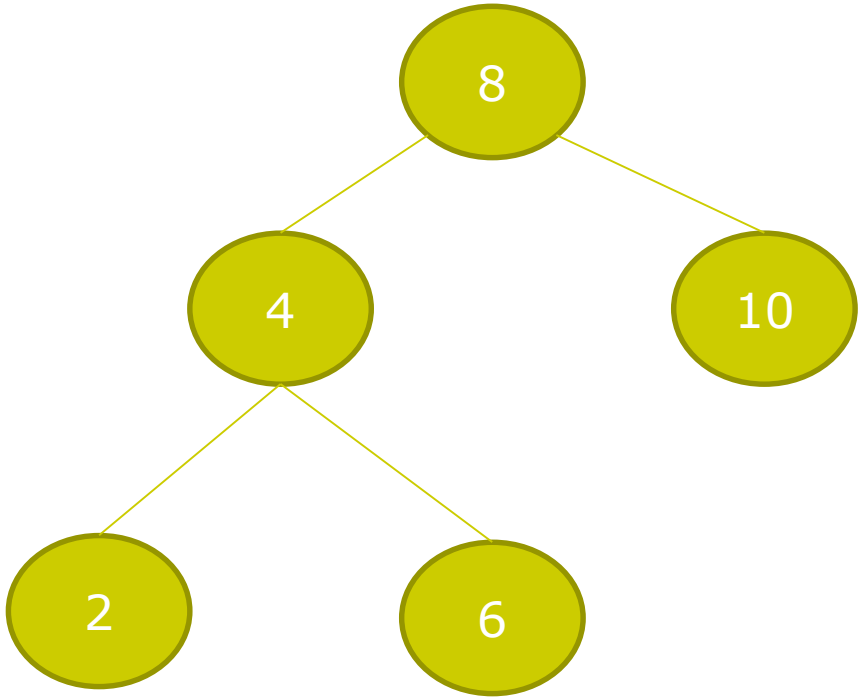
- FatBal = 1, quando a sub-árvore da direita é um nível mais alto que a esquerda.
  - Toda folha tem FB = 0

Quando é usado o FatBal?

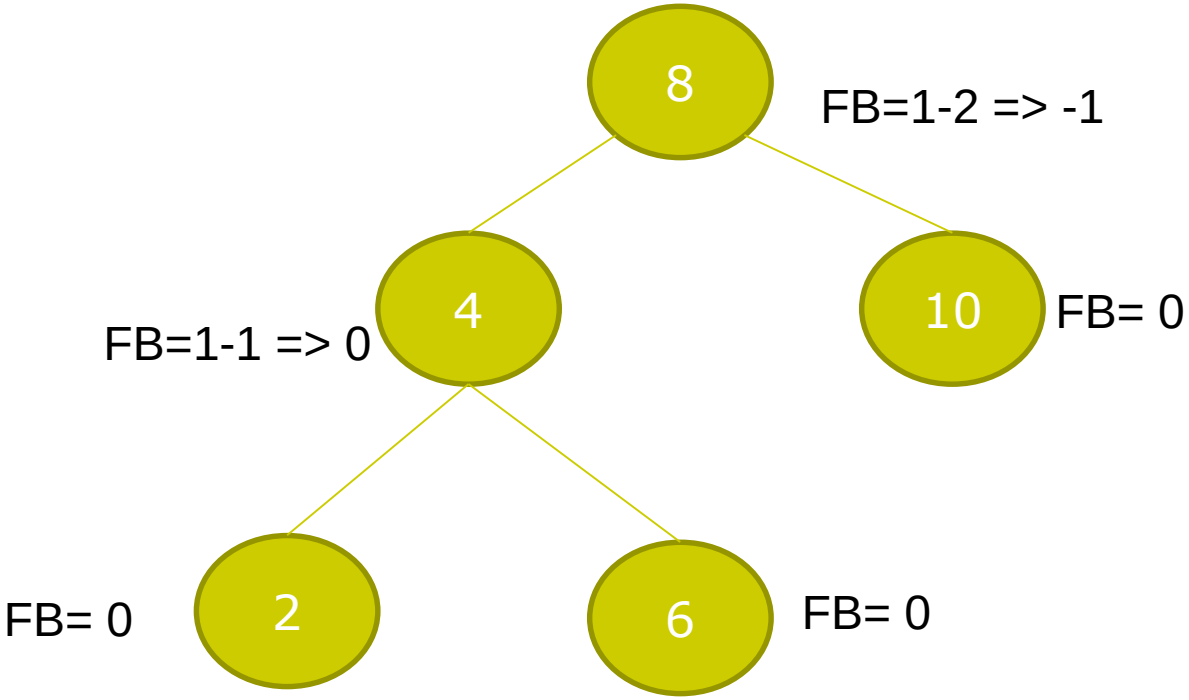
- Se a inserção/remoção afetar as propriedades de balanceamento, devemos restaurar o balanço da árvore. Esta restauração é efetuada através de Rotações na árvore.



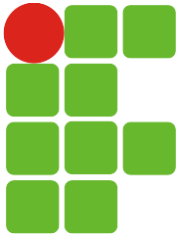
Resposta



Exercício balanceamento



Árvore Binária de Busca



Exercício 1 – crie e teste as funções em C para inserir e remover de uma árvore binária de busca. USE TAD.