AEDII -Algoritmos e Estruturas de Dados II

Aula 07 – Árvores Binária

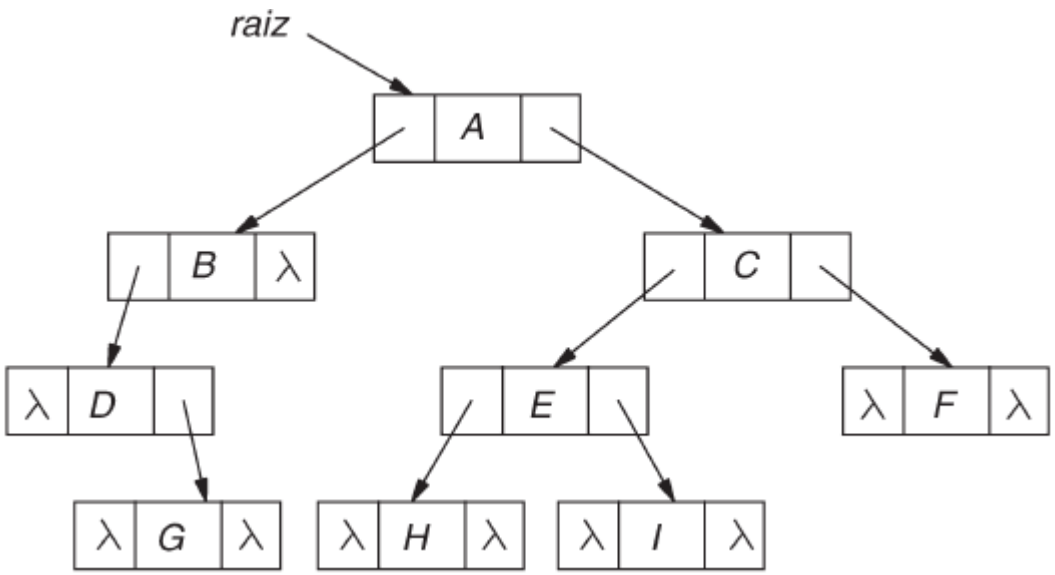
Prof. Aléssio Miranda Júnior
alessio@timoteo.cefetmg.br
2Q-2019

15/08/2019

Uma árvore binária

Representação de uma árvore binária

λ =NULL

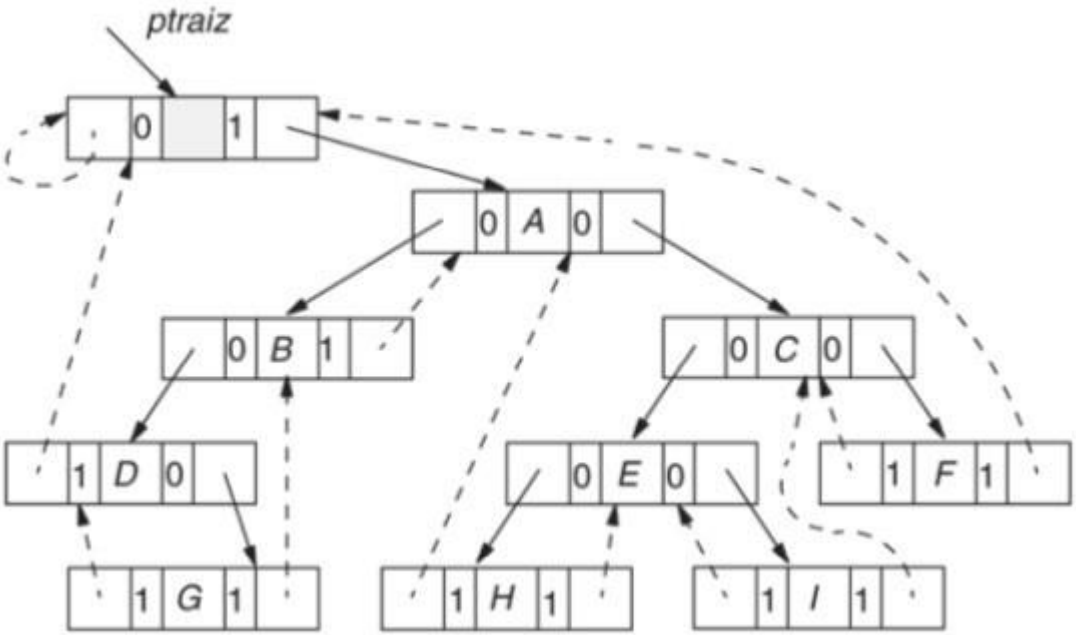


- **Para uma árvore binária de n vértices:**
- São requeridas 2n+1 unidades de memória para sua representação

- **n+1** unidades de memória são iguais a NULL.

Por que não aproveitar esse espaço de memoria?

Árvore binária com costura

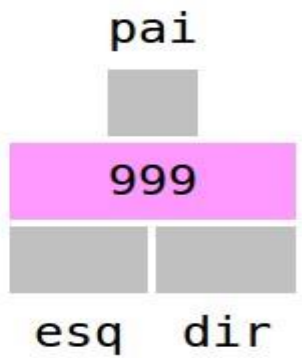


Sem uso de uma pilha é fácil percorrer a árvore.

Nós, filhos e pais

```
1  public class No {
2
3      private Integer valor;
4      private No esquerda;
5      private No direita;
6      ...
```

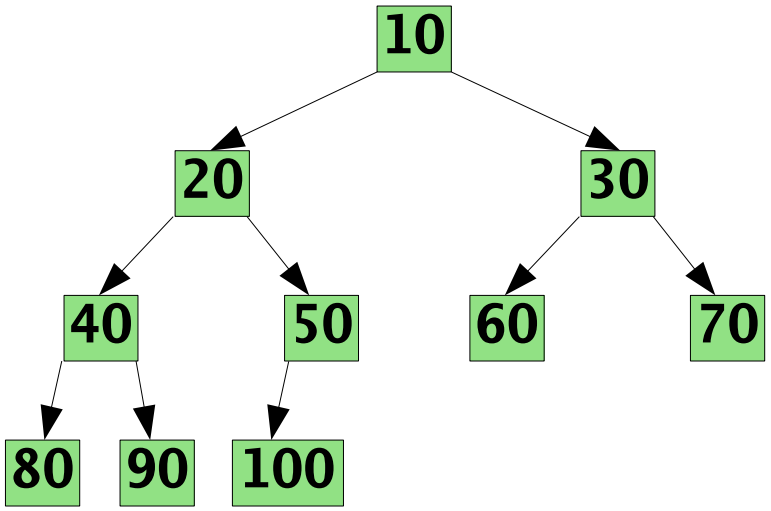

```
1  public class No {
2
3      private Integer valor
4      private No pai;
5      private No esquerda;
6      private No direita;
7      ...
```



Teste01.java

teste01.java

```
1  10 20 30 40 5  06 70 80 90 100
                  0
```

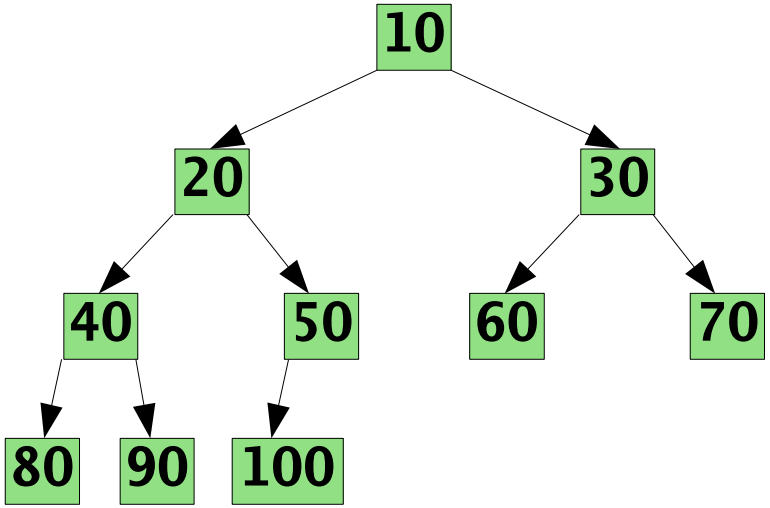



```
1  public void imprimirFolhas(No no){
2      if(no != null){
3          imprimirFolhas(no.getEsquerda());
4          if((no.getEsquerda() == null)
5                  && (no.getDireita()== null)){
6              System.err.println(" " + no.getValor());
7          }
8          imprimirFolhas(no.getDireita());
9      }
10 }
```

```
Varredura e-r-d:
80      40      90      20      100     50      10      60      30      70
Folhas da arvore:
80      90      100     60      70
Altura da arvore:
3
```

teste01.java (atividade)

1	10 20 30 40 5 0	06	70 80 90 100



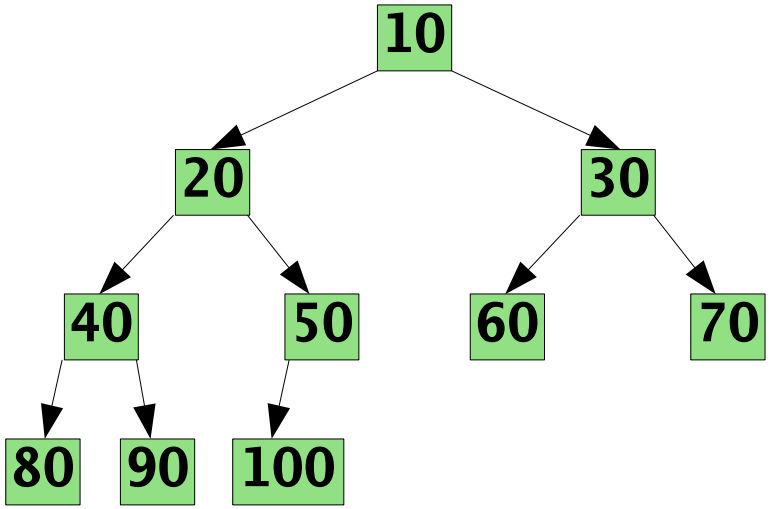
Crie uma função que permita preencher corretamente o ponteiro para o pai de cada nó.

```
1  public void preenchePai(No no){
2
3
4          //...
5
6  }
```

teste01.java (solução)

1	10 20 30 40 5 0	06	70 80 90 100

Crie uma função que permita preencher corretamente o ponteiro para o pai de cada nó.




```java
public void preenchePaiDadoFilho(No pai, No filho){
    if(filho != null){
        filho.setPai(pai);
        preenchePaiDadoFilho(filho, filho.getEsquerda());
        preenchePaiDadoFilho(filho, filho.getDireita());
    }
}

public void preenchePai(No no){
    preenchePaiDadoFilho(no, no);
}
```

```
preenchePai(raiz){
No ultimo = UltimoErd(raiz);
System.out.println("Pai do ultimo do E-R-D: " + ultimo.getPai().getValor());
```

```
Pai do ultimo no e-r-d:
30
```

teste01.java (outras respostas)

```
// Aluno01
 public void preenchePai_v1(No r) {
        if (r != null) {
            if (r.getEsquerda() != null){
                r.getEsquerda().setPai(r);

preenchePai_v1(r.getEsquerda());
            }
            if(r.getDireita() != null){
                r.getDireita().setPai(r);

preenchePai_v1(r.getDireita());
            }
        }
        if(r.getPai() == null){
            r.setPai(r);
        }
    }
```

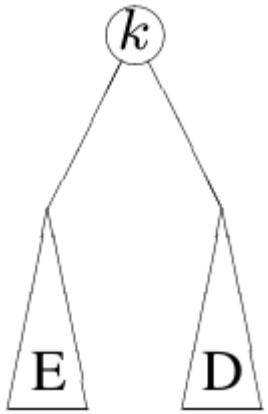
```
// Augusto Abreu
 Void
 preenchePai(no*r)
 {
   r>pai = r;
 preenche(r);
}


void preenche(no *r) {
    if (r>esq != NULL)
    {  (r>esq)>pai = r;
    preenche(r>esq);
 }
    if (r>dir != NULL)
    {  (r>dir)>pai = r;
    preenche(r>dir);
 }
}
```

Árvore binária de busca

Árvore de busca binária

Árvores binárias de busca

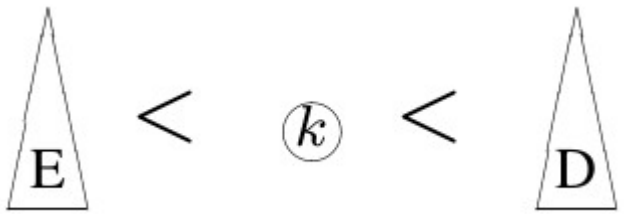
Para qualquer nó que contenha um registro:



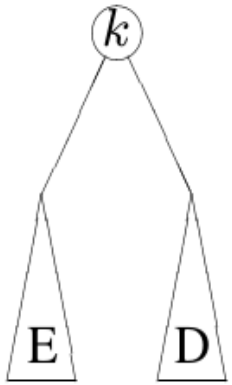
Chaves únicas!

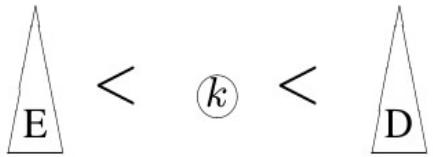
Temos a relação invariante:

1. Todos os registros com chaves menores estão na sub-árvore à esquerda.
2. Todos os registros com chaves maiores estão na sub-árvore à direita.

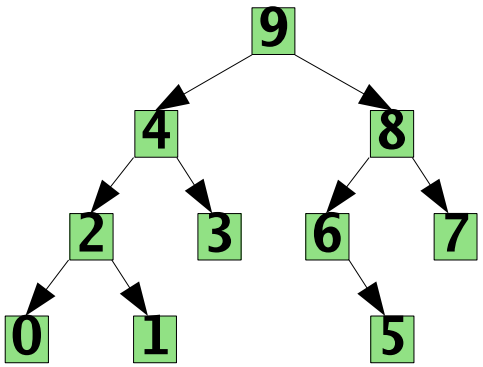


Árvores binárias de busca

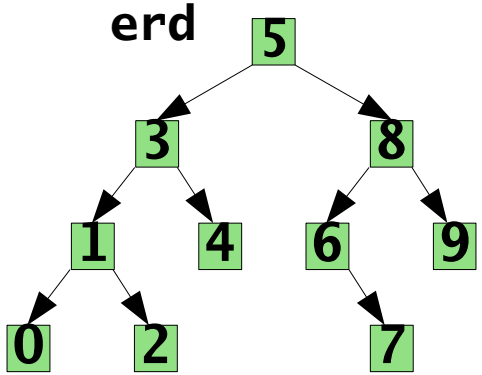




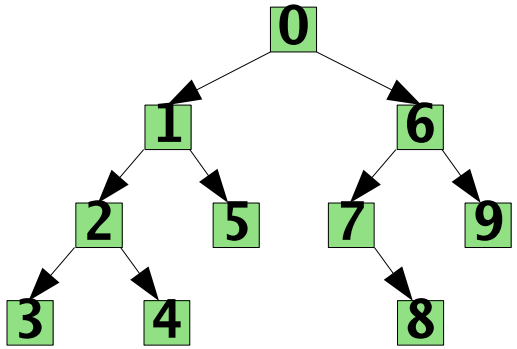
edr



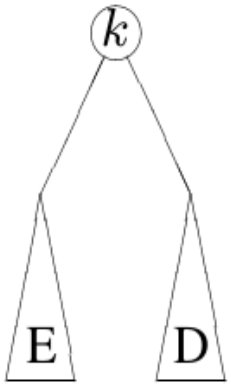
erd

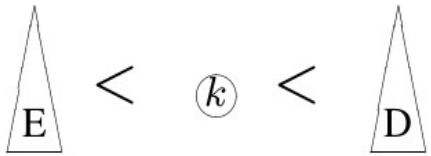


red



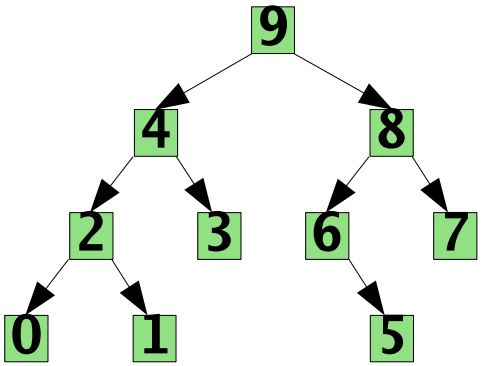
Árvores binárias de busca



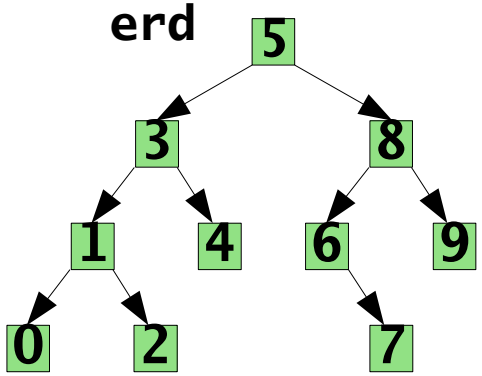


Em uma ABB a ordem e-r-d das chaves é crescente!

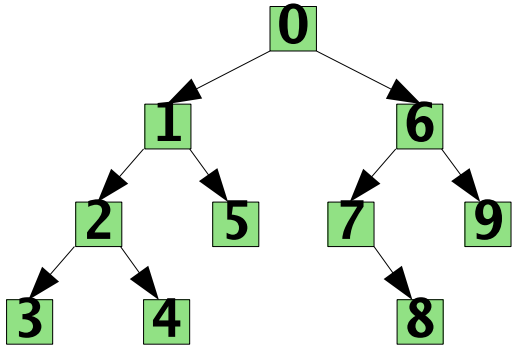
edr



erd



red

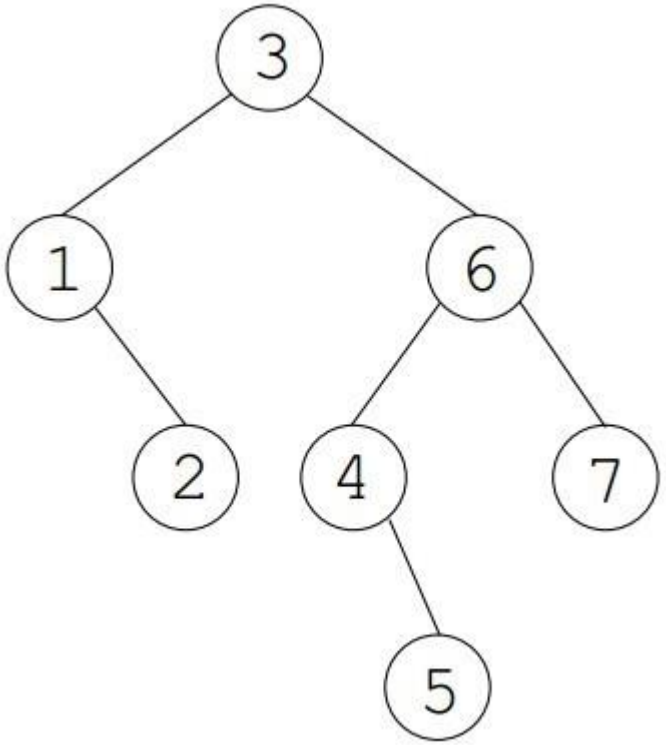


Árvores binárias de busca

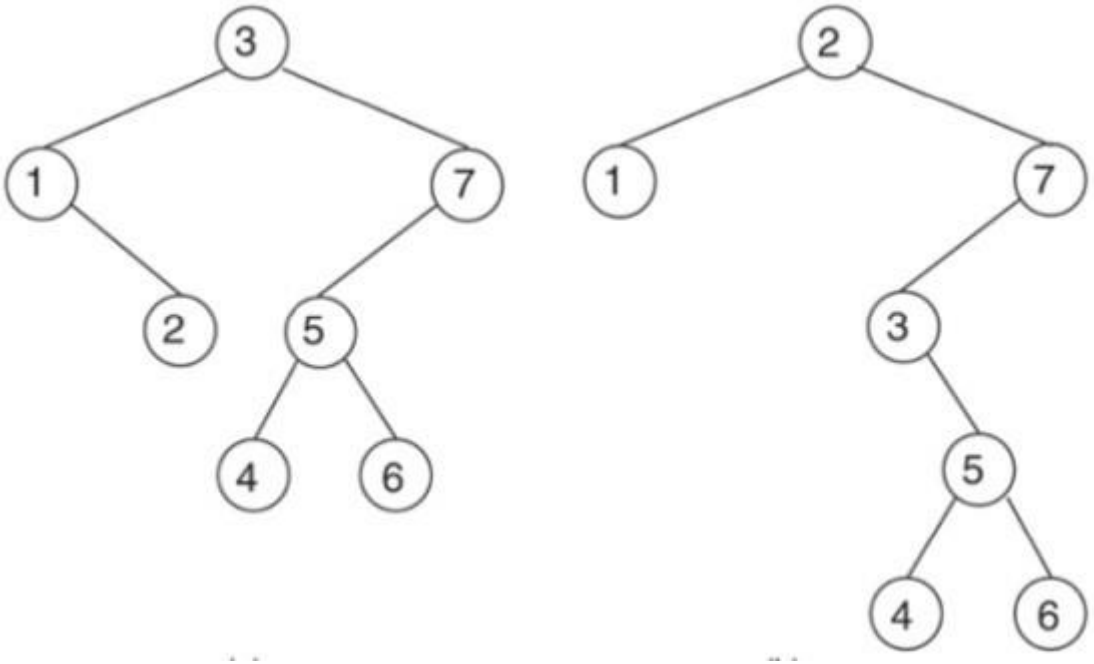
- As ABB permitem minimizar o tempo de acesso no pior caso.

- Para cada chave, separe as restantes em **maiores** ou
- **menores**.

- A estrutura hierárquica com divisão binária: **uma árvore binária**.



Árvores binárias de busca

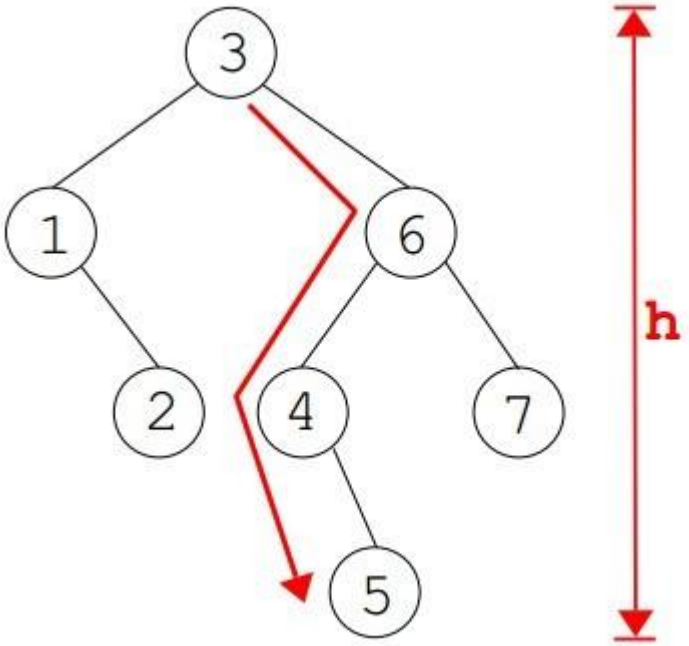


Ambas árvores binárias contém as mesmas chaves, mas sua estrutura é diferente

Complexidade de busca em umaABB

- Busca em ABB = **caminho da raiz até a chave desejada**

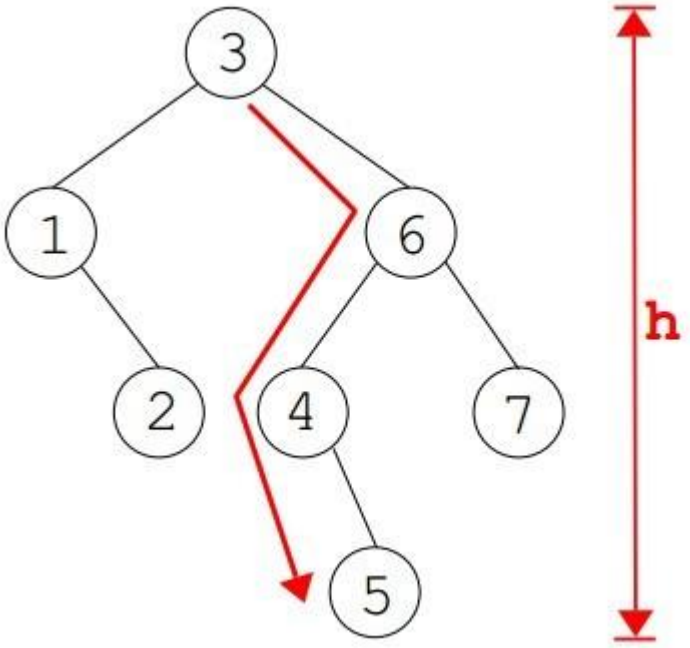
(ou até a folha, caso a chave não exista)



Complexidade de busca em umaABB

- Busca em ABB = **caminho da raiz até a chave desejada**

(ou até a folha, caso a chave não exista



Pior caso:

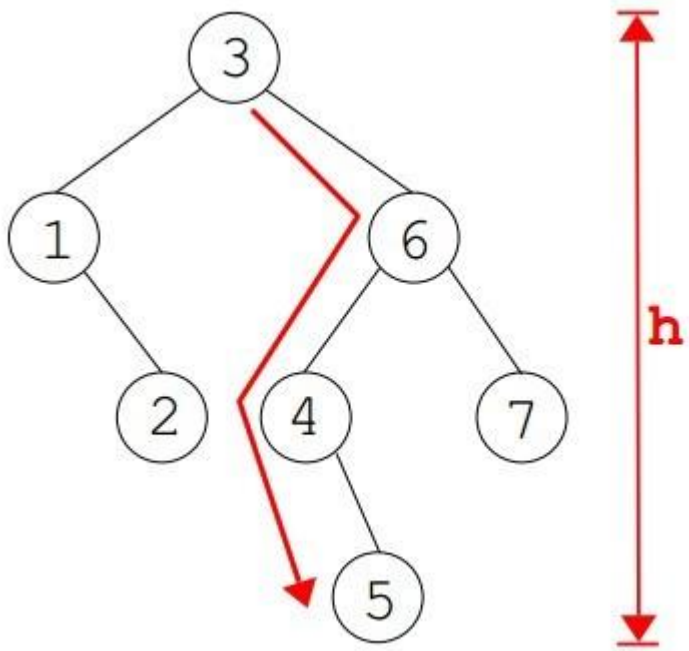
Maior caminho até a folha = altura da árvore

Complexidade: $O(h)$

Complexidade de busca em umaABB

- Busca em ABB = **caminho da raiz até a chave desejada**

(ou até a folha, caso a chave não exista)



Pior caso:

Maior caminho até a folha = altura da árvore

Complexidade: $O(h)$

Uma árvore binária balanceada é aquela com altura $O(\lg n)$

Complexidade de busca em umaABB

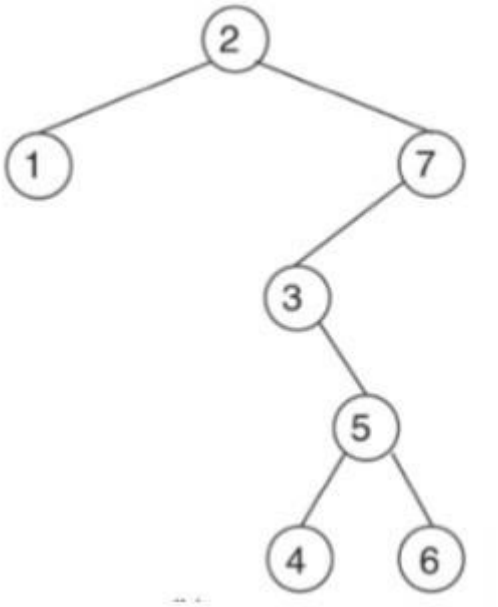
n	lg(n)
2	1
32	5
512	9
8192	13
131072	17
2097152	21
33554432	25
536870912	29
8589934592	33
137438953472	37
2199023255552	41
35184372088832	45
562949953421312	49
9007199254740990	53
144115188075856000	57
2305843009213690000	61
36893488147419100000	65
5.9029581035871E+020	69
9.4447329657393E+021	73
1.5111572745183E+023	77
2.4178516392293E+024	81
3.8685626227668E+025	85
6.1897001964269E+026	89
9.9035203142831E+027	93
1.5845632502853E+029	97
2.5353012004565E+030	101
4.0564819207303E+031	105
6.4903710731685E+032	109
1.0384593717070E+034	113
8.3076749736557E+034	116

Uma árvore binária balanceada é aquela com altura $O(\lg n)$

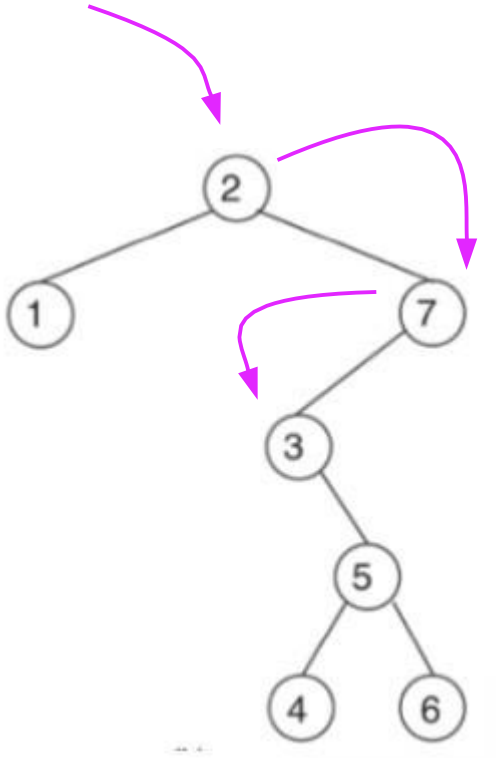
Uma árvore binária é balanceada (ou equilibrada) se, em cada um de seus nós, as subárvores esquerda e direita tiverem aproximadamente a mesma altura.

Busca de uma chaveem uma ABB

Chave =3



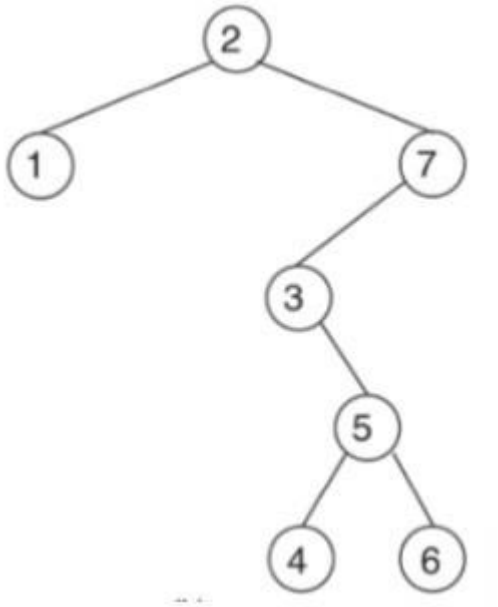
Busca de uma chaveem uma ABB



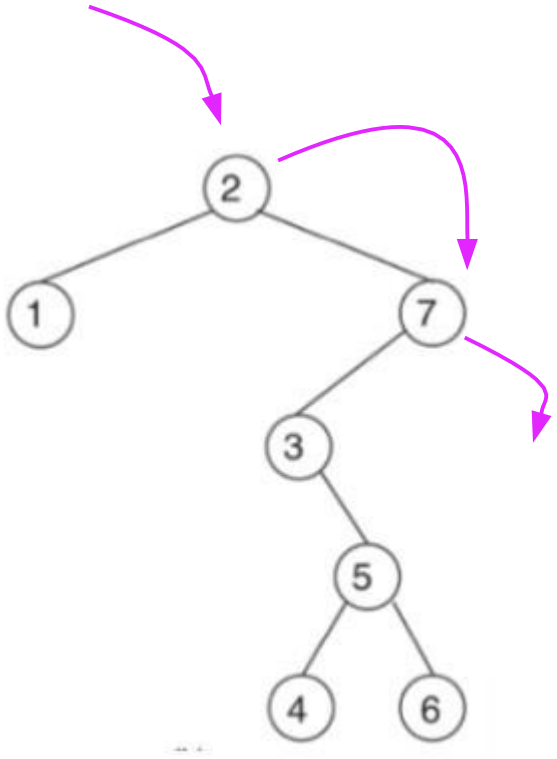
Chave =3

Busca de uma chaveem uma ABB

Chave =30

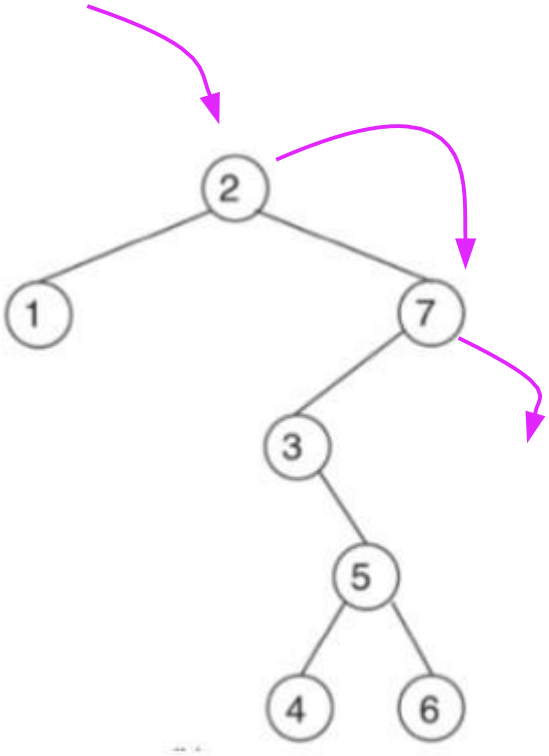


Busca de uma chaveem uma ABB



Chave =30

Busca de uma chaveem uma ABB



Chave =30

```
1  public No busca(No r, Integer chave){
2      if(r != null || Objects.equals(r.getValor(), chave)){
3          return r;
4      }
5      if(r.getValor() > chave){
6          return busca(r.getEsquerda(), chave);
7      }
8      else{
9          return busca(r.getDireita(), chave);
10     }
11 }
```

Função que recebe uma chave e a raiz da árvore e devolve o nó cujo conteúdo for igual a chave. Se o nó não existir

Teste02.java

Teste02.java (implemente)

```java
public class ArvoreBB {
    //...
    public No inserirNoNaArvore(int valor){
        return inserirNoNaArvore(raiz, valor);
    }

    public No inserirNoNaArvore(No no, int valor){
    //...

public class Teste02 {
    public static void main(String[] args) {
        int i;
        int valor;
        int n = 20;
        Scanner entrada = new Scanner(System.in);

        ArvoreBB raiz = null;

        for(i = 0; i< n; i++){
            valor = entrada.nextInt();
            raiz.inserirNoNaArvore(valor);
        }
    }
}
```

```
$ javac Teste02.java
$ ./java Teste02
  1 2 3 4 5 6 7 8 9 0
```

Teste02.java (solução)

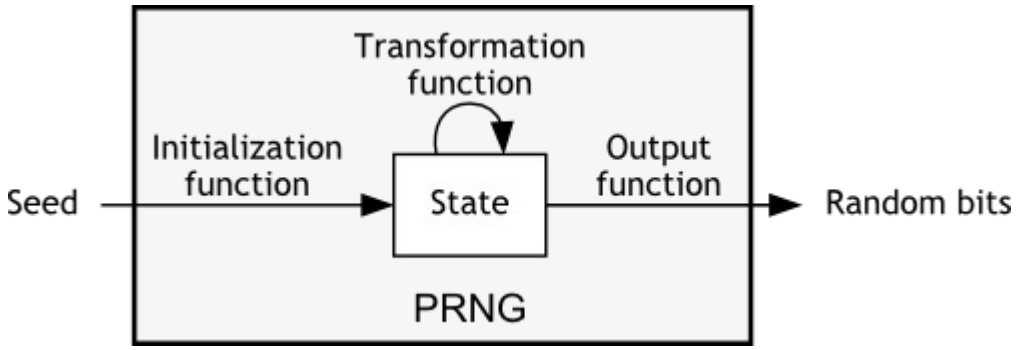
```java
public No inserirNoNaArvore(No no, int valor){
    No filho = null;
    No pai = null;

    if(busca(no, valor) == null){
       //Criacao do Novo No
       No novoNo = new No(valor);
       novoNo.setEsquerda(null);
       novoNo.setDireita(null);

       if(no == null){
           return novoNo;
       }
       else{
           filho = no;
           while(filho != null){
               pai = filho;
               if(filho.getValor() > novoNo.getValor()){
                   filho = filho.getEsquerda();
               }else{
                   filho = filho.getDireita();
               }
           }

           if(pai != null){
                if(pai.getValor() > novoNo.getValor()){
                    pai.setEsquerda(novoNo);
                }else{
                    pai.setDireita(novoNo);
                }
           }
       }
       else{
           System.out.println("Chave já presente ja Arvore: " + valor);
       }

       return no;
   }
```

Teste02.java (solução)

```
$ javac Teste02.java
$ ./java Teste02 < vetor.dat

Altura da arvore:37
```

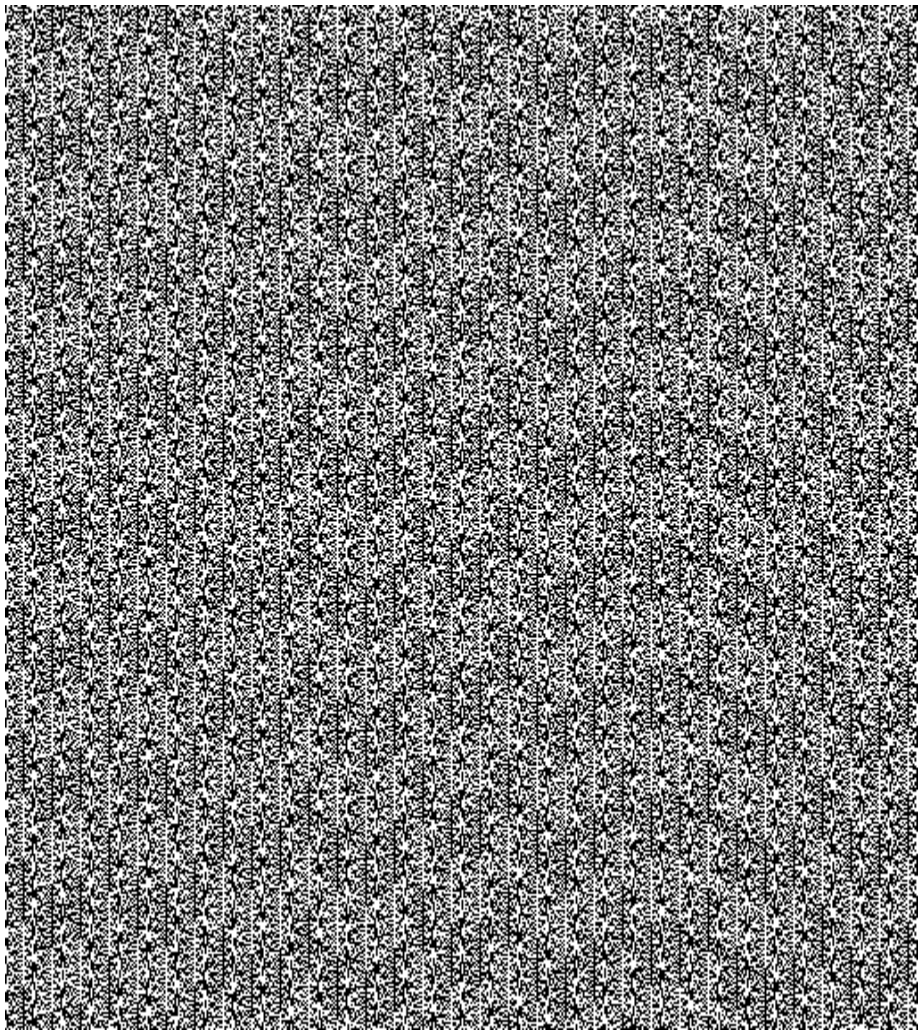
vetor.dat (contem númerosaleatórios)

1	1999865247
2	1642492079
3	1578754864
4	526388116
5	1202816126
6	1081054700
7	1412170679
8	349667423
9	908334402
10	182300537
11	369035264
12	632798115
13	634971847
14	1564533593
15	575255984
16	2043958983
17	149010144
18	311799603
19	512011104
20	321680977
21	1266852553
22	471588485
23	202818486
24	764059859
25	1963523066
26	643592132
27	1491412607
28	589261762
29	1516859391
30	1581697502
31	1600792033
32	1369240991
33	1076705934
34	1032063249
35	1895629107

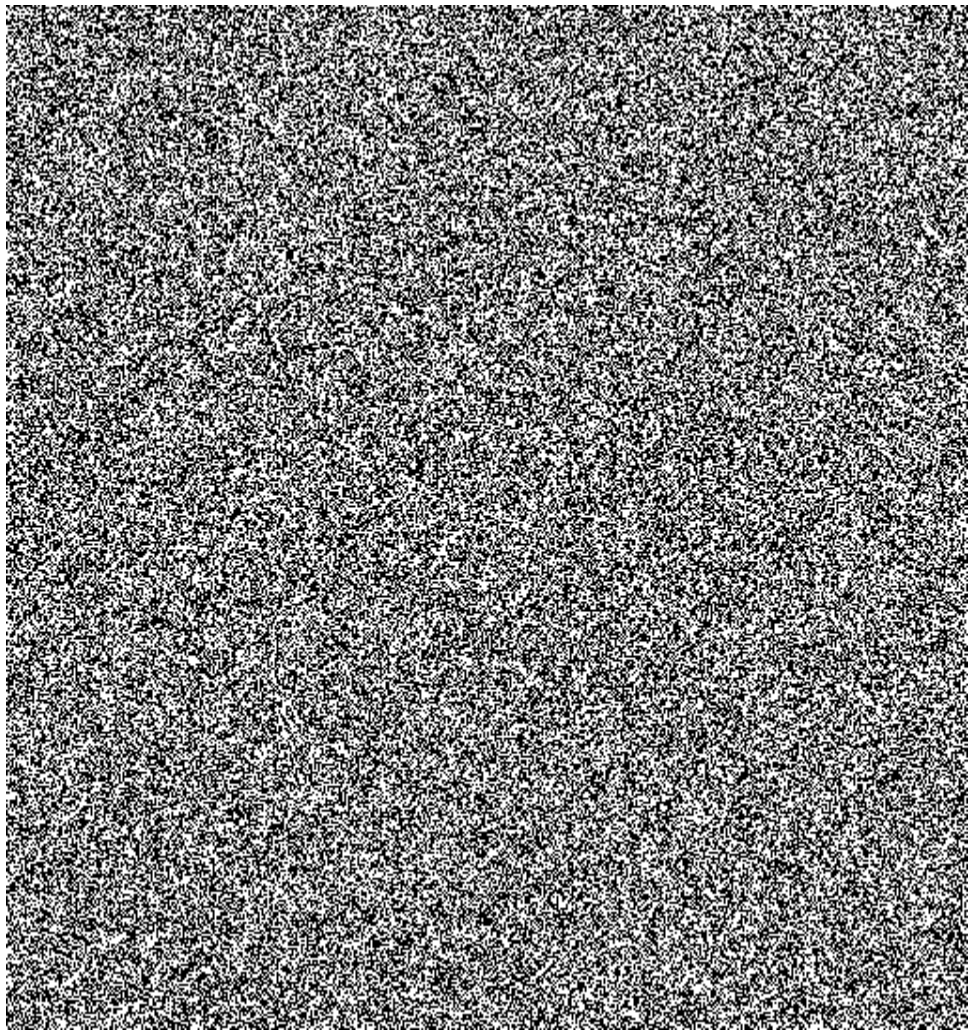


Vetor.dat (contem númerosaleatórios)

Pseudo-random

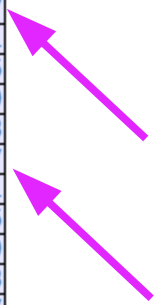


True-random



Teste02.java (solução)

n	lg(n)
2	1
32	5
512	9
8192	13
131072	17
2097152	21
33554432	25
536870912	29
8589934592	33
137438953472	37
2199023255552	41
35184372088832	45
562949953421312	49
9007199254740990	53
144115188075856000	57
2305843009213690000	61
36893488147419100000	65
5.9029581035871E+020	69
9.4447329657393E+021	73
1.5111572745183E+023	77
2.4178516392293E+024	81
3.8685626227668E+025	85
6.1897001964269E+026	89
9.9035203142831E+027	93
1.5845632502853E+029	97
2.5353012004565E+030	101
4.0564819207303E+031	105
6.4903710731685E+032	109
1.0384593717070E+034	113
8.3076749736557E+034	116



```
$ javac Teste02.java
$ ./java Teste02.exe vetor.dat

Altura da arvore:37
```