

Universidade Federal de Goiás
Aluno: Luiz Guilherme Dias Catulio
Curso: Bacharelado em Ciência da Computação
Disciplina: Algoritmos e Estruturas de Dados 2

Teste de Mesa para árvores binárias

A árvore implementada possui como filho esquerdo de cada nó valores estritamente menores que o do nó atual, e no lado direito valores maiores ou iguais ao do nó atual

CPF fictício a ser inserido	526.659.351-94
-----------------------------	----------------

Funções

```
17
18 void btree_insert(btree *arvore, node *endereco){ //filhos a esquerda se
19     node *temp1 = NULL; //futuro pai
20     node *temp2 = arvore->root;
21     while(temp2 != NULL){ //procura um lugar adequado para o filho, se
22         //contrario, procuramos pela direita ate encontrarmos um node vazio
23         temp1 = temp2;
24         if(endereco->informacoes.chave < temp2->informacoes.chave){
25             temp2 = temp2->l_son;
26         }else{
27             temp2 = temp2->r_son;
28         }
29     }
30     endereco->father = temp1;
31     if(temp1 == NULL){
32         arvore->root = endereco;
33     }else if(endereco->informacoes.chave < temp1->informacoes.chave){
34         temp1->l_son = endereco;
35     } else{
36         temp1->r_son = endereco;
37     }
38 }
```

```

150 node *tree_search(node *endereco, int chave){
151     if(endereco != NULL){
152         node *temporario = endereco;
153         while(temporario != NULL && temporario->informacoes.chave != chave){
154             if(chave < temporario->informacoes.chave){
155                 temporario = temporario->l_son;
156             }else{
157                 temporario = temporario->r_son;
158             }
159         }
160         return temporario;
161     } else{
162         return NULL;
163     }
164 }

```

```

126 void btree_delete_node(btree *arvore, node *endereco){ //tudo depende se endereco
    tem filhos
127     /*
128     substituímos o endereco pelo seu sucessor presente em sua subarvore direita
129     */
130     if(endereco != NULL){
131         if(endereco->l_son == NULL){ //aqui e em baixo nos simplesmente elevamos
            os filhos para ficarem no lugar do pai
132             transplant(arvore, endereco, endereco->r_son);
133         }else if(endereco->r_son == NULL){ //nao tem filhos
134             transplant(arvore, endereco, endereco->l_son);
135         } else {
136             node *temporario = tree_minimun(endereco->r_son);
137             if(temporario->father != endereco){
138                 transplant(arvore, temporario, temporario->r_son);
139                 temporario->r_son = endereco->r_son;
140                 temporario->r_son->father = temporario;
141             }
142             transplant(arvore, endereco, temporario);
143             temporario->l_son = endereco->l_son;
144             temporario->l_son->father = temporario;
145         }
146         free(endereco);
147     }
148 }
149


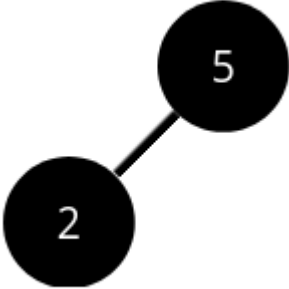
```

```

39 void transplant(btree *arvore, node *end1, node *end2){ //trocamos os pais de end1
    e end2 um pelo outro
40     if(end1->father == NULL){ //estamos trocando a raiz pr outro node
41         arvore->root = end2;
42     } else if(end1 == (end1->father->l_son)){ //ou seja, end1 é menor que seu pai
43         end1->father->l_son = end2;
44     } else{
45         end1->father->r_son = end2;
46     }
47
48     if(end2 != NULL){ //troca simples e direta do pai de end2 pelo end1
49         end2->father = end1->father;
50     }
51
52     /*
53     lembrando que aqui nao fazemos nenhuma comparacao, apenas fazemos a troca
54     */
55 }
56

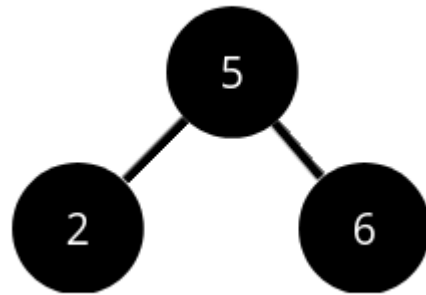
```

Construção da árvore

Número inserido: 5		
Operação	Linha executada	
Inserção na raiz	30 e 31	
Número inserido: 2		
Operação	Linha executada	
2 < 5	24	
Inserção à esquerda	32 e 33	

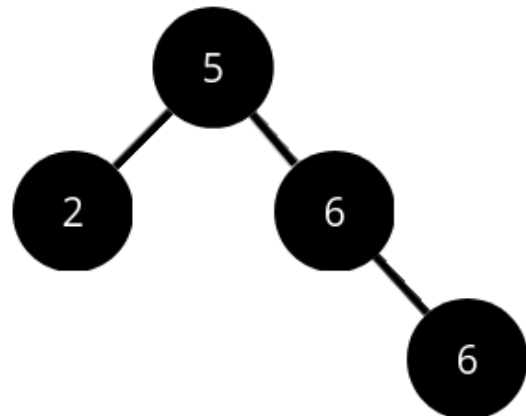
Número inserido: 6

Operação	Linha executada
$6 > 5$	25 e 26
Inserção à direita	34 e 35



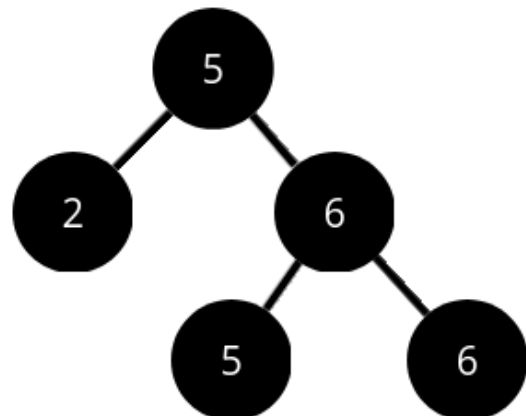
Número inserido: 6

Operação	Linha executada
$6 > 5$	25 e 26
$6 \geq 6$	25 e 26
Inserção à direita	34 e 35



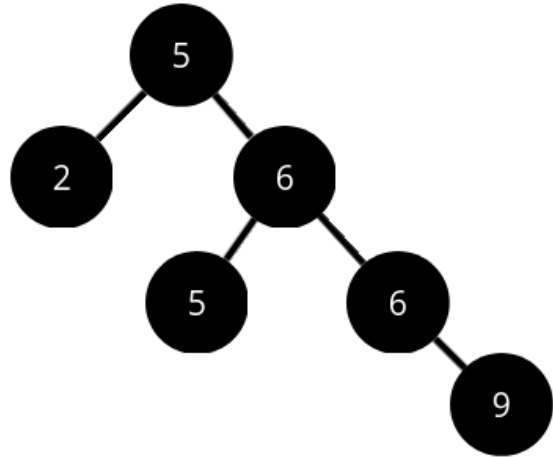
Número inserido: 5

Operação	Linha executada
$5 \geq 6$	25 e 26
$5 < 6$	23 e 24
Inserção à esquerda	32 e 33



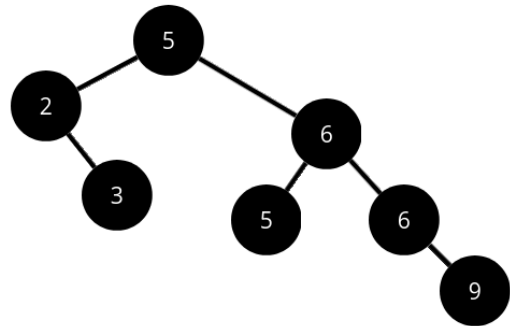
Número inserido: 9

Operação	Linha executada
$9 > 5$	25 e 26
$9 > 6$	25 e 26
$9 > 6$	25 e 26
Inserção à direita	34 e 35



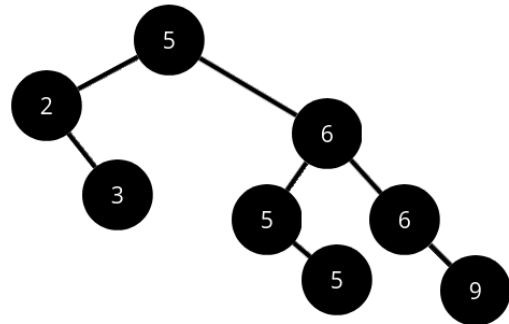
Número inserido: 3

Operação	Linha executada
$3 < 5$	23 e 24
$3 > 2$	25 e 26
Inserção à direita	32 e 33



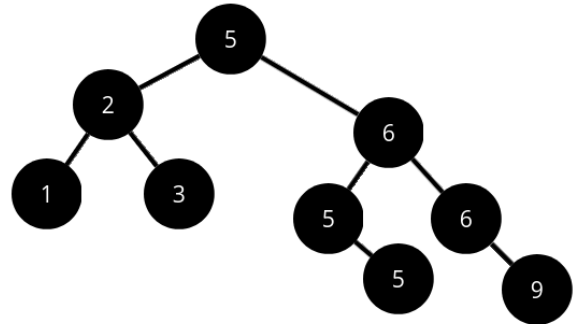
Número inserido: 5

Comparação	Linha executada
$5 \geq 5$	25 e 26
$5 < 6$	23 e 24
$5 \geq 5$	25
Inserção à direita	34 e 35



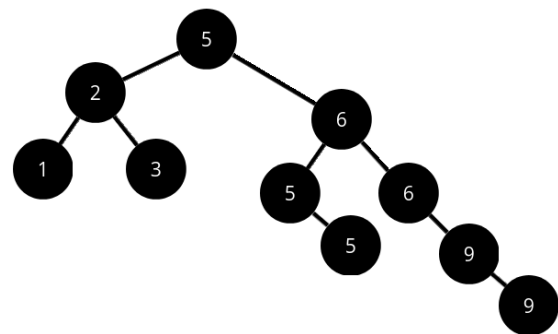
Número inserido: 1

Comparação	Linha executada
$1 < 5$	23 e 24
$1 < 2$	23 e 24
Inserção à esquerda	32 e 33



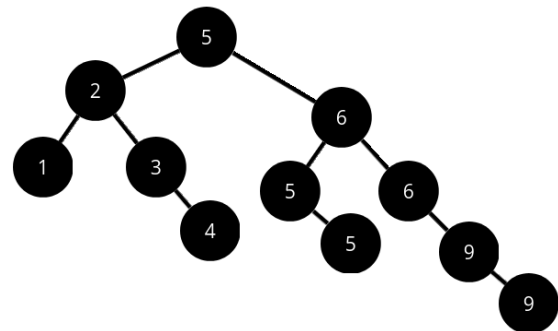
Número inserido: 9

Comparação	Linha executada
$9 > 5$	25 e 26
$9 > 6$	25 e 26
$9 > 6$	25 e 26
$9 \geq 9$	25 e 26
Inserção à direita	34 e 35



Número inserido: 4

Operação	Linha executada
$4 < 5$	23 e 24
$4 > 2$	25 e 26
$4 > 3$	25 e 26
Inserção à direita	34 e 35

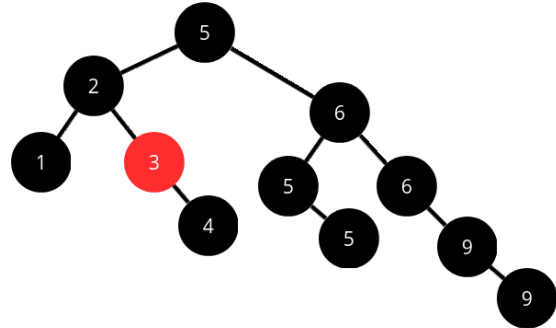


Exclusão do 7º número

Valor do 7º número: 3

Busca no valor 3 na árvore:

Operação	Linha executada
$3 < 5$	154 e 155
$3 > 2$	156 e 157
$3 = 3$	160
Retorno do endereço	160



Exclusão do valor 3

Operação	Linha executada
endereço != NULL	130
Nó não possui filho esquerdo, chamada da função transplant	131 e 132
Nó de valor 4 toma o lugar de seu pai de valor 3	44 e 45

