arvoresBinariaBusca

November 25, 2021

1 Árvore Binária de Busca

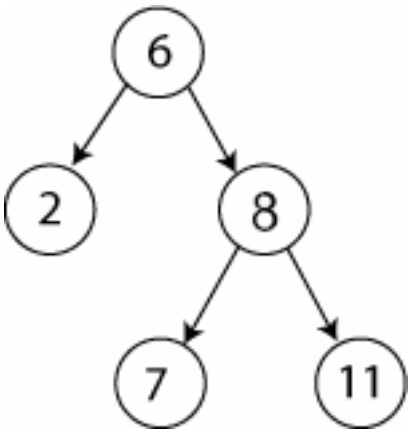
Dado um elemento x:

- Todos os elementos chaves da subárvore da esquerda de x são menores que x
- Todos os elementos chaves da subárvore da direita de x são maiores que x
- As subárvores esquerda e direita também são árvores binárias de busca

Também são conhecidas como dicionários binários.

Armazena campos chaves, dados que identificam univocamente uma informação

- RG, CPF, etc.



[1]:
```
#include <iostream>
#include <string>

using namespace std;
```

[2]:
```
class ArvoreBuscaBinaria {
    private:
        struct elemento {
            int valor;
            elemento *elementoEsquerda;
            elemento *elementoDireita;
        };
        typedef elemento *PonteiroElemento;
```

```
        PonteiroElemento raiz;
        // outras operações: métodos auxiliares
        int totalElementos(PonteiroElemento &e);
        int totalFolhas(PonteiroElemento &e);
        int altura(PonteiroElemento &e);
        void listarPreOrdem(PonteiroElemento &e);
        void inserir(PonteiroElemento &e, int x);
        bool remover(PonteiroElemento &e, int x);
        bool pesquisar(PonteiroElemento &e, int x);
        void removerMenor(PonteiroElemento &q, PonteiroElemento &r);
        int minimo(PonteiroElemento &e);
        int maximo(PonteiroElemento &e);
    public:
        ArvoreBuscaBinaria();
        bool vazia();
        bool cheia();
        bool inserir(int x);
        bool remover(int x);
        bool pesquisar(int x);
        // outras operações
        int totalElementos();
        int totalFolhas();
        int altura();
        void listarPreOrdem();
        int minimo();
        int maximo();
};
```

1.1 Construtor

[3]:
```
ArvoreBuscaBinaria::ArvoreBuscaBinaria() {
    raiz = nullptr;
}
```

[4]:
```
ArvoreBuscaBinaria minhaArvore;
```

1.2 Vazia

[5]:
```
bool ArvoreBuscaBinaria::vazia() {
    return raiz == nullptr;
}
```

[6]:
```
if (minhaArvore.vazia()) {
    cout << "Está vazia!";
}
```

```
Está vazia!
```

1.3 Cheia

[7]:
```
bool ArvoreBuscaBinaria::cheia() {
    return false;
}
```

[8]:
```
if (! minhaArvore.cheia()) {
    cout << "Não está cheia!";
}
```

```
Não está cheia!
```

1.4 Inserir

[9]:
```
// Método Público
bool ArvoreBuscaBinaria::inserir(int x) {
    inserir(raiz, x);
    return true;
}
```

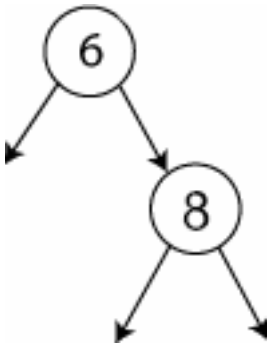
[10]:
```
// Método Privado
void ArvoreBuscaBinaria::inserir(PonteiroElemento &e, int x) {
    PonteiroElemento p;
    if (e == nullptr) {
        p = new elemento;
        p->valor = x;
        p->elementoEsquerda = nullptr;
        p->elementoDireita = nullptr;
        e = p;
        return;
    }
    if (x < e->valor) {
        inserir(e->elementoEsquerda, x);
    } else {
        inserir(e->elementoDireita, x);
    }
}
```

[11]:
```
minhaArvore.inserir(6);
```

```
minhaArvore.inserir(6);
    inserir(raiz, 6); // raiz: NULL
```

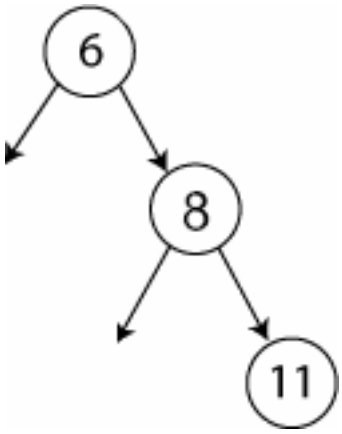
[12]:
```
minhaArvore.inserir(8);
```

```
minhaArvore.inserir(8);
    inserir(raiz, 8); // e: 6
        inserir(e->elementoDireita, 8); // e: NULL
```



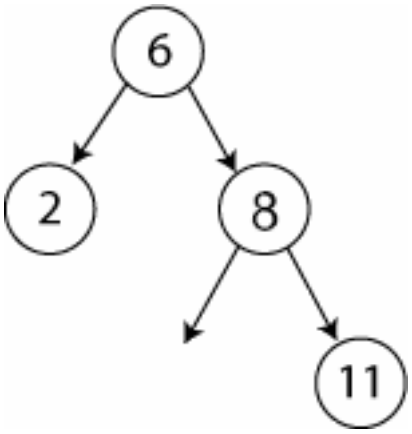
[13]:
```
minhaArvore.inserir(11);
```

```
minhaArvore.inserir(11);
    inserir(raiz, 11); //e:6
        inserir(e->elementoDireita, 11); //e:8
            inserir(e->elementoDireita, 11); //e:NULL
```



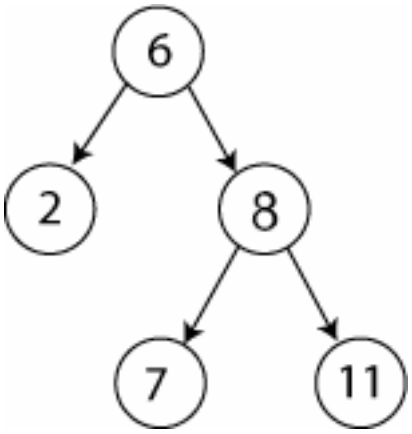
[14]:
```
minhaArvore.inserir(2);
```

```
minhaArvore.inserir(2);
    inserir(raiz, 2); // e: 6
        inserir(e->elementoEsquerda, 2); // e: NULL
```



[15]:
```
minhaArvore.inserir(7);
```

```
minhaArvore.inserir(7);
    inserir(raiz, 7); // e: 6
        inserir(e->elementoDireita, 7); // e: 8
            inserir(e->elementoEsquerda, 7); // e: NULL
```



1.5 Listar: pré ordem

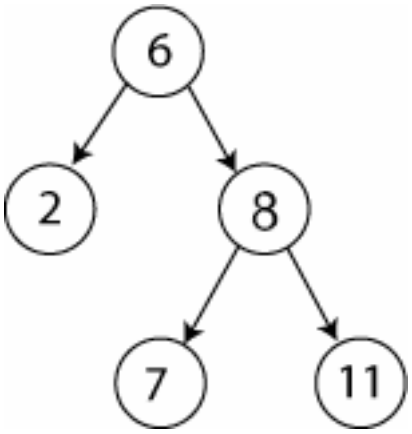
[16]:
```
// Método Público
void ArvoreBuscaBinaria::listarPreOrdem() {
    listarPreOrdem(raiz);
}
```

[17]:
```
// Método Privado
void ArvoreBuscaBinaria::listarPreOrdem(PonteiroElemento &e)
{
    if (e != NULL) {
        cout << e->valor << "\n";
        listarPreOrdem(e->elementoEsquerda);
        listarPreOrdem(e->elementoDireita);
```

```
    }
}
```

[18]:
```
minhaArvore.listarPreOrdem();
```

```
6
2
8
7
11
```



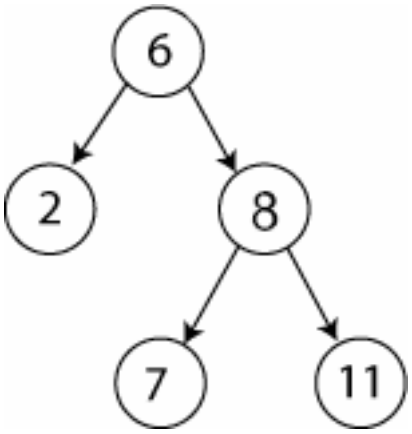
1.6 Remover

[19]:
```
// Método Público
bool ArvoreBuscaBinaria::remover(int x) {
    return remover(raiz, x);
}
```

[20]:
```
// Método Privado
bool ArvoreBuscaBinaria::remover(PonteiroElemento &e, int x) {
    PonteiroElemento p;
    if (e == nullptr) {
        return false;
    }

    if(x < e->valor) {
        return remover(e->elementoEsquerda, x);
    }

    if(x > e->valor) {
        return remover(e->elementoDireita, x);
    }

    if(x == e->valor) {
```

```
        p = e;
        if(p->elementoDireita == nullptr) {
            e = p->elementoEsquerda;
        } else {
            if(p->elementoEsquerda == nullptr) {
                e = p->elementoDireita;
            } else {
                removerMenor(p, p->elementoDireita);
            }
        }
        delete p;
        return true;
    }
    return false;
}
```

[21]:
```
void ArvoreBuscaBinaria::removerMenor(PonteiroElemento &q, PonteiroElemento &r)
{
    if(r->elementoEsquerda != nullptr) {
        removerMenor(q, r->elementoEsquerda);
    } else {
        q->valor = r->valor;
        q = r;
        r = r->elementoDireita;
    }
}
```

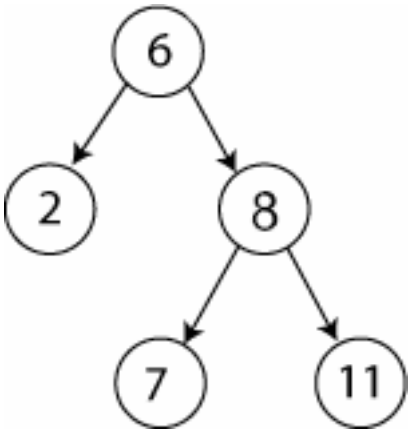


[22]:
```
minhaArvore.remover(13)
```

[22]: false

```
minhaArvore.remover(13);
    remover(raiz, 13); // e: 6
        return remover(e->elementoDireita, 13); // e: 8
```

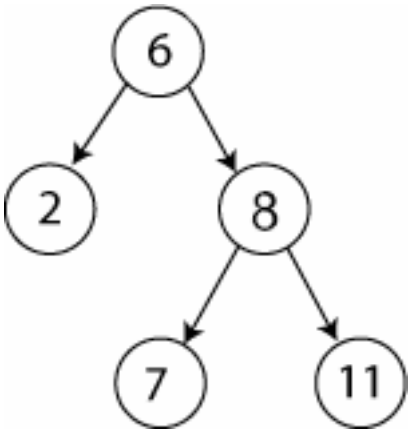
```
            return remover(e->elementoDireita, 13); // e: 11
                return remover(e->elementoDireita, 13); // e: NULL
                // false
            // false
        // false
    // false
// false
```



[23]:
```
minhaArvore.remover(1)
```

[23]: false

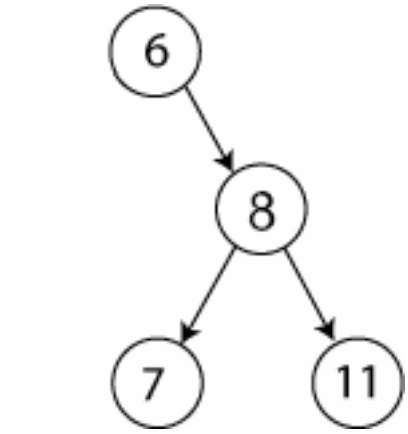
```
minhaArvore.remover(1);
    return remover(raiz, 1); // e: 6
        return remover(e->elementoEsquerda, 1); // e: 2
            return remover(e->elementoEsquerda, 1); // e: NULL
            // false
        // false
    // false
// false
```



[24]:
```
minhaArvore.remover(2)
```

[24]: true

```
minhaArvore.remover(2);
    return remover(raiz, 2); // e: 6
        return remover(e->elementoEsquerda, 2); // e: 2
        // true
    // true
// true
```



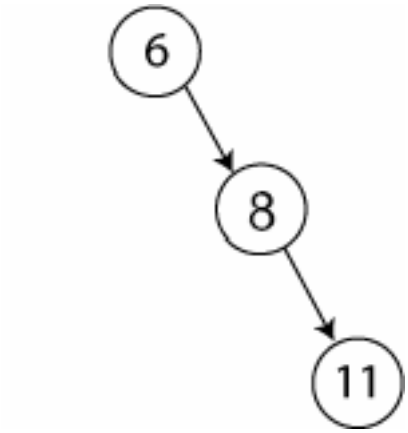
[25]:
```
minhaArvore.listarPreOrdem();
```

```
6
8
7
11
```

[26]:
```
minhaArvore.remover(7)
```

[26]: true

```
minhaArvore.remover(7);
    return remover(raiz, 7); // e: 6
        return remover(e->elementoDireita, 7); // e: 8
            return remover(e->elementoEsquerda, 7); // e: 7
            // true
        // true
    // true
// true
```



[27]:
```
minhaArvore.listarPreOrdem();
```

```
6
8
11
```

[28]:
```
minhaArvore.remover(8)
```

[28]: true

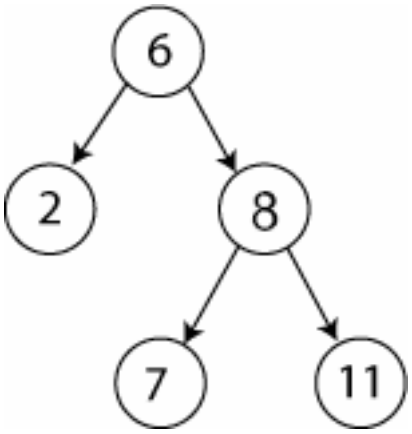
```
minhaArvore.remover(8);
    return remover(raiz, 8); // e: 6
        return remover(e->elementoDireita, 8); // e: 8
        // true
    // true
// true
```

[29]:
```
minhaArvore.listarPreOrdem();
```

```
6
11
```

[30]:
```
//restaurando arvore
minhaArvore.remover(6);
minhaArvore.remover(8);
minhaArvore.remover(11);

minhaArvore.inserir(6);
minhaArvore.inserir(2);
minhaArvore.inserir(8);
minhaArvore.inserir(7);
minhaArvore.inserir(11);

minhaArvore.listarPreOrdem();
```

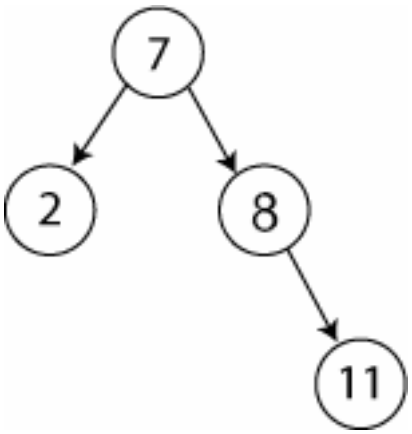
```
6
2
8
7
11
```



[31]:
```
minhaArvore.remover(6)
```

[31]: true

```
minhaArvore.remover(6);
    return remover(raiz, 6); // 6
        removerMenor(p, p->elementoDireita); // q: 6 r: 8
            removerMenor(q, r->elementoEsquerda); // q: 6 r: 7
            // q: 7 , r->elementoEsquerda: NUll
        // p: 7 , p->elementoDireita: 8
```



[32]:
```
minhaArvore.listarPreOrdem();
```

```
7
2
```

```
8
11
```

[33]:
```
//restaurando arvore
minhaArvore.remover(7);
minhaArvore.remover(2);
minhaArvore.remover(8);
minhaArvore.remover(11);

minhaArvore.inserir(6);
minhaArvore.inserir(2);
minhaArvore.inserir(8);
minhaArvore.inserir(7);
minhaArvore.inserir(11);

minhaArvore.listarPreOrdem();
```

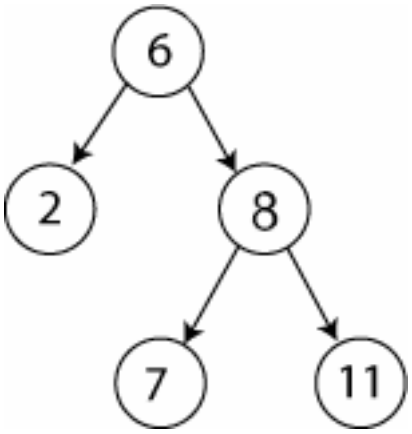
```
6
2
8
7
11
```

1.7 Pesquisar

[34]:
```
// Método Público
bool ArvoreBuscaBinaria::pesquisar(int x)
{
    return pesquisar(raiz, x);
}
```

[35]:
```
// Método Privado
bool ArvoreBuscaBinaria::pesquisar(PonteiroElemento &e, int x){
    if (e == nullptr)
        return false;

    if (x < e->valor)
        return pesquisar(e->elementoEsquerda, x);

    if (x > e->valor)
        return pesquisar(e->elementoDireita, x);

    if (x == e->valor)
        return true;

    return false; // nunca executará, só para evitar o warning
}
```

[36]:
```
// 2 -> 6 -> 7 -> 8 -> 11
```



[37]:
```
for (int i=0; i<15 ; i++)
    if (minhaArvore.pesquisar(i))
        cout << i << ": achou!" << endl;
    else
        cout << i << ": não achou!" << endl;
```

```
0: não achou!
1: não achou!
2: achou!
3: não achou!
4: não achou!
5: não achou!
6: achou!
7: achou!
8: achou!
9: não achou!
10: não achou!
11: achou!
12: não achou!
13: não achou!
14: não achou!
```