AULA e06
Algoritmos e Estruturas de Dados I

Árvores Binárias

Luciano Antonio Digiampietri

Árvores Binárias

Como já visto, árvores binárias são estruturas de dados nas quais cada nó pode ter até dois filhos.

Árvores Binárias

Como já visto, árvores binárias são estruturas de dados nas quais cada nó pode ter até dois filhos.

- A motivação utilizada na última aula foi a possibilidade de se realizar busca binária.

Árvores Binárias

Como já visto, árvores binárias são estruturas de dados nas quais cada nó pode ter até dois filhos.

- A motivação utilizada na última aula foi a possibilidade de se realizar busca binária.

- Para isso é necessário que haja uma *ordem* entre as chaves dos elementos da árvore e assim são criadas as chamadas árvores binárias de pesquisa (ou árvores binárias de busca).

Árvores Binárias

Porém existem árvores binárias que não utilizam essa *regra* de organização entre as chaves dos seus elementos.

Árvores Binárias

Porém existem árvores binárias que não utilizam essa *regra* de organização entre as chaves dos seus elementos.

Este é o assunto da aula de hoje!

Árvores Binárias

Porém existem árvores binárias que não utilizam essa *regra* de organização entre as chaves dos seus elementos.

Este é o assunto da aula de hoje!

O usuário irá nos dizer em qual posição um nó irá ser inserido.

Árvores Binárias

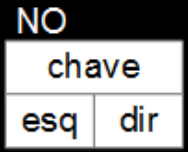
Implementaremos as seguintes funções:

- **Busca por um nó a partir de uma chave;**
- **Criação do nó raiz;**
- **Inserção de um nó.**

Modelagem

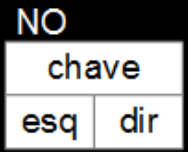
Modelagem

```
#include <stdio.h>
#include <malloc.h>
#define true 1
#define false 0



typedef int bool;
typedef int TIPOCHAVE;

typedef struct aux{
  TIPOCHAVE chave;
  struct aux *esq, *dir;
} NO;

typedef NO* PONT;
```



Modelagem

```c
#include <stdio.h>
#include <malloc.h>
#define true 1
#define false 0

typedef enum{esquerdo, direito} LADO;

typedef int bool;
typedef int TIPOCHAVE;

typedef struct aux{
  TIPOCHAVE chave;
  struct aux *esq, *dir;
} NO;

typedef NO* PONT;
```



Busca

Por não haver uma *ordem* entre as chaves não é possível realizar uma busca eficiente:

Busca

Por não haver uma *ordem* entre as chaves não é possível realizar uma busca eficiente:

- **Procuraremos a chave buscada na raiz;**

Busca

Por não haver uma *ordem* entre as chaves não é possível realizar uma busca eficiente:

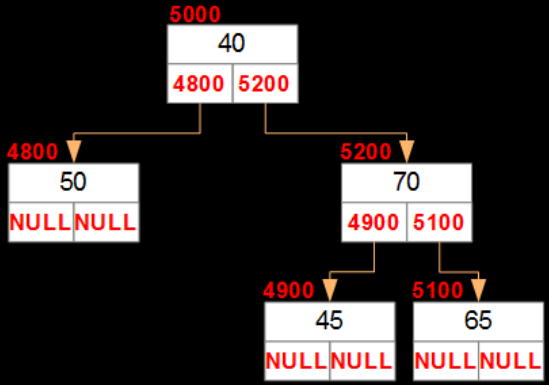
- **Procuraremos a chave buscada na raiz;**
- **Se não encontrarmos, procuraremos na sub-árvore à esquerda;**

Busca

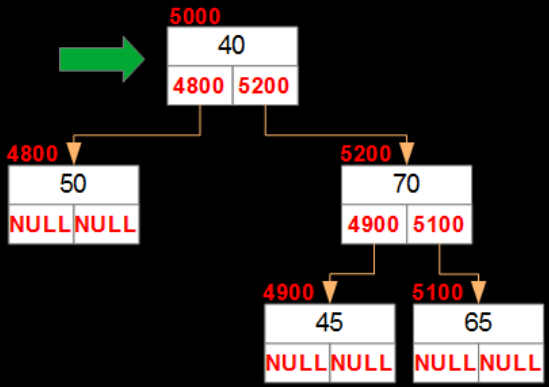
Por não haver uma *ordem* entre as chaves não é possível realizar uma busca eficiente:

- Procuraremos a chave buscada na raiz;
- Se não encontrarmos, procuraremos na sub-árvore à esquerda;
- Se não encontrarmos, procuraremos na sub-árvore à direita.

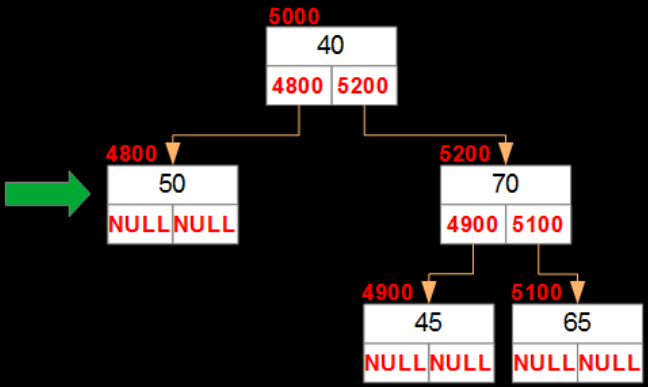
Busca


5000
40
4800
5200
4800
50
NULL
NULL
5200
70
4900
5100
4900
45
NULL
NULL
5100
65
NULL
NULL

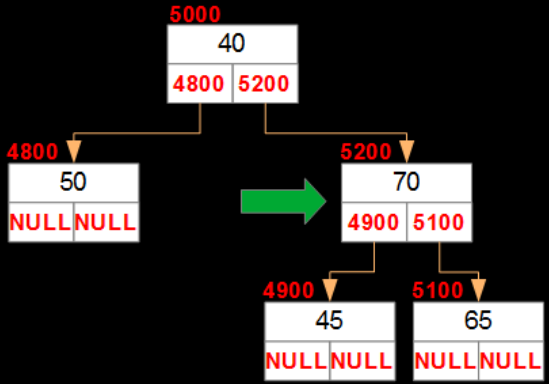
Busca


5000
40
4800
5200
4800
50
NULL
NULL
5200
70
4900
5100
4900
45
NULL
NULL
5100
65
NULL
NULL

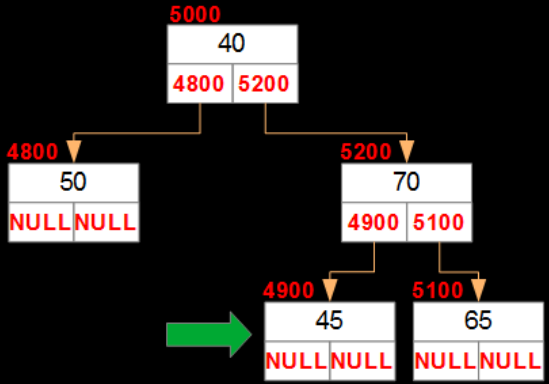
Busca


5000
40
4800
5200
4800
50
NULL
NULL
5200
70
4900
5100
4900
45
NULL
NULL
5100
65
NULL
NULL

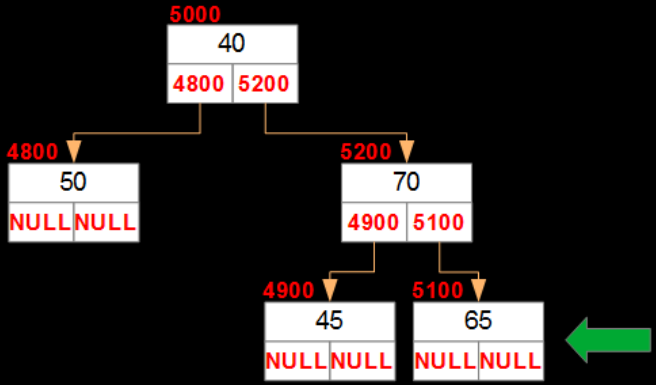
Busca


5000
40
4800
5200
4800
50
NULL
NULL
5200
70
4900
5100
4900
45
NULL
NULL
5100
65
NULL
NULL

Busca


5000
40
4800
5200
4800
50
NULL
NULL
5200
70
4900
5100
4900
45
NULL
NULL
5100
65
NULL
NULL

Busca


5000
40
4800
5200
4800
50
NULL
NULL
5200
70
4900
5100
4900
45
NULL
NULL
5100
65
NULL
NULL

Busca

```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){



}
```

Busca

```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){
  if (raiz == NULL) return NULL;




}
```

Busca

```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;



}
```

Busca

```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  PONT aux = buscarChave(ch,raiz->esq);
  if (aux) return aux;

}
```

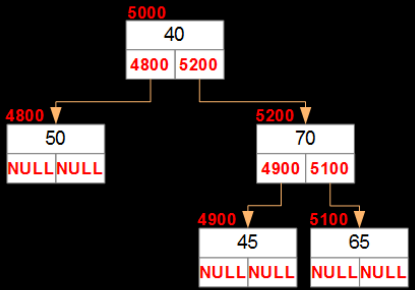
Busca

```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  PONT aux = buscarChave(ch,raiz->esq);
  if (aux) return aux;
  return buscarChave(ch,raiz->dir);
}
```

Busca

```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  PONT aux = buscarChave(ch,raiz->esq);
  if (aux) return aux;
  return buscarChave(ch,raiz->dir);
}
```

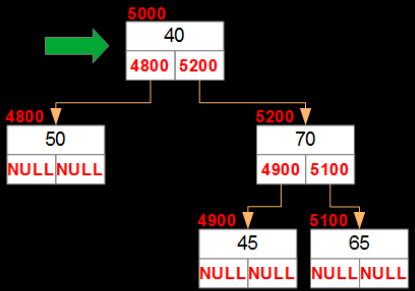
Chamadas recursivas:



Busca

```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  PONT aux = buscarChave(ch,raiz->esq);
  if (aux) return aux;
  return buscarChave(ch,raiz->dir);
}
```

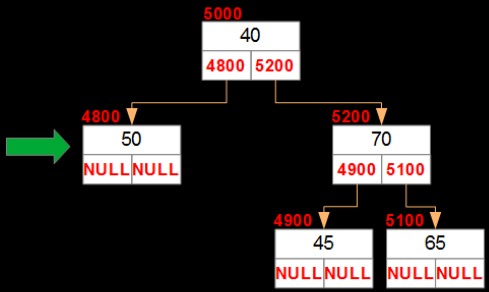
Chamadas recursivas:
1. ch=45 raiz=5000



Busca

```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  PONT aux = buscarChave(ch,raiz->esq);
  if (aux) return aux;
  return buscarChave(ch,raiz->dir);
}
```

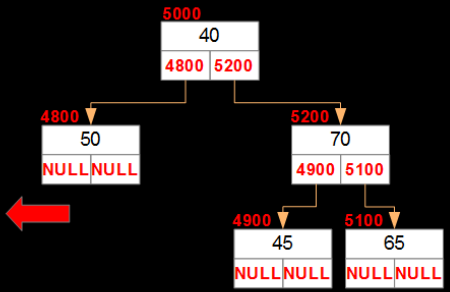
Chamadas recursivas:
1. ch=45 raiz=5000
2. ch=45 raiz=4800



Busca

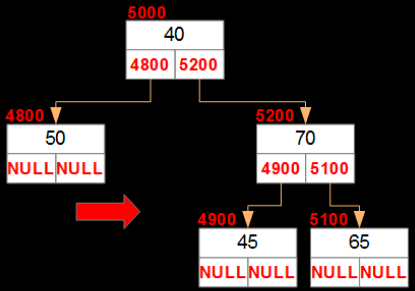
```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  PONT aux = buscarChave(ch,raiz->esq);
  if (aux) return aux;
  return buscarChave(ch,raiz->dir);
}
```

```
Chamadas recursivas:
1. ch=45 raiz=5000
2. ch=45 raiz=4800
3. ch=45 raiz=NULL
```



Busca

```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  PONT aux = buscarChave(ch,raiz->esq);
  if (aux) return aux;
  return buscarChave(ch,raiz->dir);
}
```

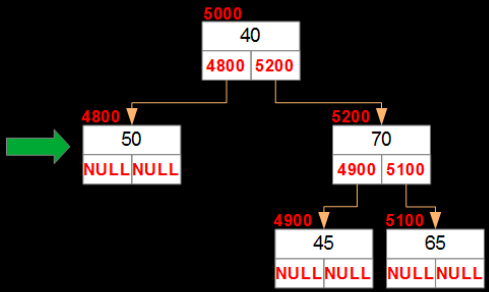


Chamadas recursivas:
1. ch=45 raiz=5000
2. ch=45 raiz=4800
4. ch=45 raiz=NULL

Busca

```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  PONT aux = buscarChave(ch,raiz->esq);
  if (aux) return aux;
  return buscarChave(ch,raiz->dir);
}
```

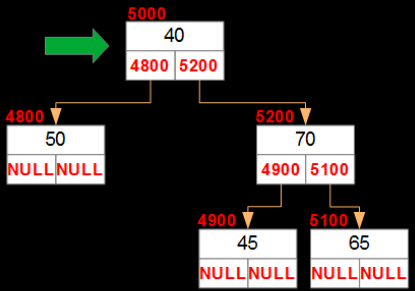
Chamadas recursivas:
1. ch=45 raiz=5000
2. ch=45 raiz=4800



Busca

```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  PONT aux = buscarChave(ch,raiz->esq);
  if (aux) return aux;
  return buscarChave(ch,raiz->dir);
}
```

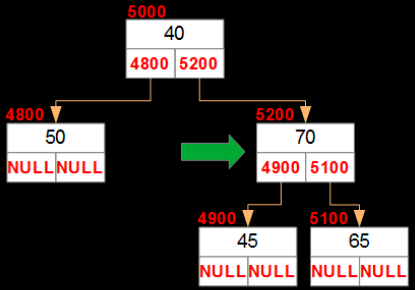
Chamadas recursivas:
1. ch=45 raiz=5000



Busca

```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  PONT aux = buscarChave(ch,raiz->esq);
  if (aux) return aux;
  return buscarChave(ch,raiz->dir);
}
```

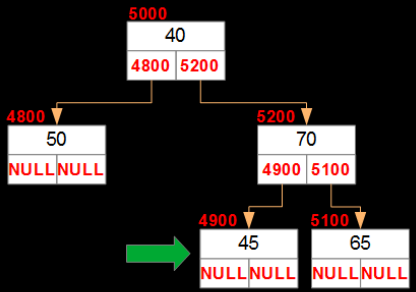
```
Chamadas recursivas:
1. ch=45 raiz=5000
5. ch=45 raiz=5200
```



Busca

```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  PONT aux = buscarChave(ch,raiz->esq);
  if (aux) return aux;
  return buscarChave(ch,raiz->dir);
}
```

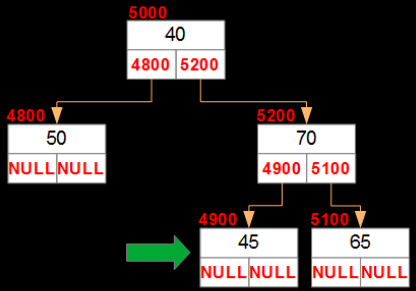
```
Chamadas recursivas:
1. ch=45 raiz=5000
5. ch=45 raiz=5200
6. ch=45 raiz=4900
```



Busca

```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  PONT aux = buscarChave(ch,raiz->esq);
  if (aux) return aux;
  return buscarChave(ch,raiz->dir);
}
```

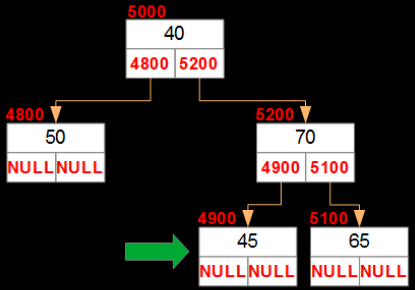
```
Chamadas recursivas:
1. ch=45 raiz=5000
5. ch=45 raiz=5200
6. ch=45 raiz=4900    -> 4900
```



Busca

```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  PONT aux = buscarChave(ch,raiz->esq);
  if (aux) return aux;
  return buscarChave(ch,raiz->dir);
}
```

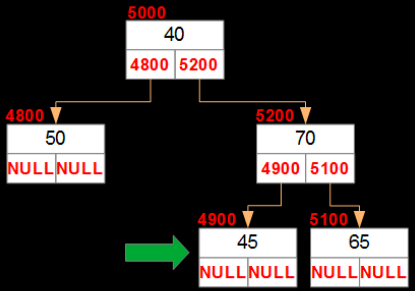
```
Chamadas recursivas:
1. ch=45 raiz=5000
5. ch=45 raiz=5200    -> 4900
```



Busca

```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  PONT aux = buscarChave(ch,raiz->esq);
  if (aux) return aux;
  return buscarChave(ch,raiz->dir);
}
```

```
Chamadas recursivas:
1. ch=45 raiz=5000   -> 4900
```



Criação do nó raiz

O nó raiz tem uma característica especial: ele não é filho de nenhum outro nó.

Criação do nó raiz

O nó raiz tem uma característica especial: ele não é filho de nenhum outro nó.

- O usuário passará o endereço de memória no qual o endereço da raiz será salvo;

Criação do nó raiz

O nó raiz tem uma característica especial: ele não é filho de nenhum outro nó.

- O usuário passará o endereço de memória no qual o endereço da raiz será salvo;
- Também irá passar o valor da chave da raiz.

Criação de um novo nó

```
PONT criarNovoNo(TIPOCHAVE ch){
   PONT novoNo = (PONT) malloc(sizeof(NO));




}
```

Criação de um novo nó

```
PONT criarNovoNo(TIPOCHAVE ch){
   PONT novoNo = (PONT) malloc(sizeof(NO));
   novoNo->esq = NULL;
   novoNo->dir = NULL;
   novoNo->chave = ch;

}
```

Criação de um novo nó

```
PONT criarNovoNo(TIPOCHAVE ch){
   PONT novoNo = (PONT) malloc(sizeof(NO));
   novoNo->esq = NULL;
   novoNo->dir = NULL;
   novoNo->chave = ch;
   return novoNo;
}
```

Criação de um novo nó

```
PONT criarNovoNo(TIPOCHAVE ch){
   PONT novoNo = (PONT) malloc(sizeof(NO));
   novoNo->esq = NULL;
   novoNo->dir = NULL;
   novoNo->chave = ch;
   return novoNo;
}

void criarRaiz(PONT* raiz, TIPOCHAVE chave){
    *raiz = criarNovoNo(chave);
}
```

Criação de um novo nó

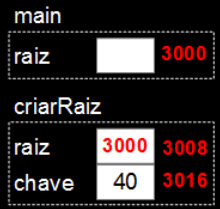
```
int main(){
   PONT raiz;
```


```
PONT criarNovoNo(TIPOCHAVE ch){
   PONT novoNo = (PONT) malloc(sizeof(NO));
   novoNo->esq = NULL;
   novoNo->dir = NULL;
   novoNo->chave = ch;
   return novoNo;
}

void criarRaiz(PONT* raiz, TIPOCHAVE chave){
    *raiz = criarNovoNo(chave);
}
```

Criação de um novo nó

```
PONT criarNovoNo(TIPOCHAVE ch){
   PONT novoNo = (PONT) malloc(sizeof(NO));
   novoNo->esq = NULL;
   novoNo->dir = NULL;
   novoNo->chave = ch;
   return novoNo;
}

void criarRaiz(PONT* raiz, TIPOCHAVE chave){
    *raiz = criarNovoNo(chave);
}
```

```
int main(){
   PONT raiz;
   criarRaiz(&raiz,40);
```



Criação de um novo nó

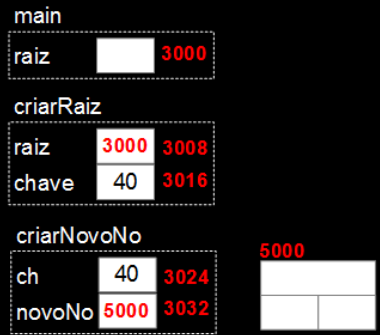
```
PONT criarNovoNo(TIPOCHAVE ch){
   PONT novoNo = (PONT) malloc(sizeof(NO));
   novoNo->esq = NULL;
   novoNo->dir = NULL;
   novoNo->chave = ch;
   return novoNo;
}

void criarRaiz(PONT* raiz, TIPOCHAVE chave){
    *raiz = criarNovoNo(chave);
}
```

```
int main(){
   PONT raiz;
   criarRaiz(&raiz,40);
```

Criação de um novo nó

```
PONT criarNovoNo(TIPOCHAVE ch){
   PONT novoNo = (PONT) malloc(sizeof(NO));
   novoNo->esq = NULL;
   novoNo->dir = NULL;
   novoNo->chave = ch;
   return novoNo;
}

void criarRaiz(PONT* raiz, TIPOCHAVE chave){
    *raiz = criarNovoNo(chave);
}
```

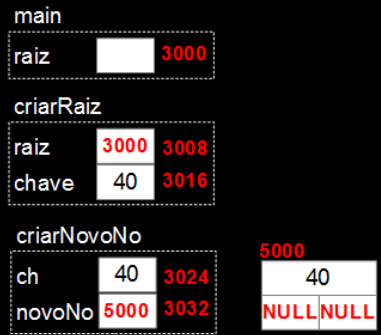
```
int main(){
   PONT raiz;
   criarRaiz(&raiz,40);
```



Criação de um novo nó

```
PONT criarNovoNo(TIPOCHAVE ch){
   PONT novoNo = (PONT) malloc(sizeof(NO));
   novoNo->esq = NULL;
   novoNo->dir = NULL;
   novoNo->chave = ch;
   return novoNo;
}

void criarRaiz(PONT* raiz, TIPOCHAVE chave){
    *raiz = criarNovoNo(chave);
}
```

```
int main(){
   PONT raiz;
   criarRaiz(&raiz,40);
```



Criação de um novo nó

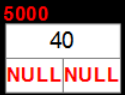
```
PONT criarNovoNo(TIPOCHAVE ch){
   PONT novoNo = (PONT) malloc(sizeof(NO));
   novoNo->esq = NULL;
   novoNo->dir = NULL;
   novoNo->chave = ch;
   return novoNo;
}

void criarRaiz(PONT* raiz, TIPOCHAVE chave){
    *raiz = criarNovoNo(chave);
}
```

```
int main(){
   PONT raiz;
   criarRaiz(&raiz,40);
```

Criação de um novo nó

```
PONT criarNovoNo(TIPOCHAVE ch){
   PONT novoNo = (PONT) malloc(sizeof(NO));
   novoNo->esq = NULL;
   novoNo->dir = NULL;
   novoNo->chave = ch;
   return novoNo;
}

void criarRaiz(PONT* raiz, TIPOCHAVE chave){
    *raiz = criarNovoNo(chave);
}
```

```
int main(){
   PONT raiz;
   criarRaiz(&raiz,40);
```



Inserção de um nó filho

O usuário irá dizer onde o nó será inserido:

Inserção de um nó filho

O usuário irá dizer onde o nó será inserido:

- O usuário passará o endereço do nó raiz;

Inserção de um nó filho

O usuário irá dizer onde o nó será inserido:

- O usuário passará o endereço do nó raiz;
- A chave do novo nó;

Inserção de um nó filho

O usuário irá dizer onde o nó será inserido:

- O usuário passará o endereço do nó raiz;
- A chave do novo nó;
- A chave do pai do novo nó;

Inserção de um nó filho

O usuário irá dizer onde o nó será inserido:

- O usuário passará o endereço do nó raiz;
- A chave do novo nó;
- A chave do pai do novo nó;
- O lado em que a inserção será feita.

Inserção de um nó filho

```
bool inserirFilho(PONT raiz, TIPOCHAVE novaChave, TIPOCHAVE chavePai, LADO lado){









}
```

Inserção de um nó filho

```
bool inserirFilho(PONT raiz, TIPOCHAVE novaChave, TIPOCHAVE chavePai, LADO lado){
  PONT pai = buscarChave(chavePai,raiz);









}
```

Inserção de um nó filho

```
bool inserirFilho(PONT raiz, TIPOCHAVE novaChave, TIPOCHAVE chavePai, LADO lado){
  PONT pai = buscarChave(chavePai,raiz);
  if (!pai) return false;









}
```

Inserção de um nó filho

```
bool inserirFilho(PONT raiz, TIPOCHAVE novaChave, TIPOCHAVE chavePai, LADO lado){
  PONT pai = buscarChave(chavePai,raiz);
  if (!pai) return false;
  PONT novo = criarNovoNo(novaChave);









  return true;
}
```

Inserção de um nó filho

```
bool inserirFilho(PONT raiz, TIPOCHAVE novaChave, TIPOCHAVE chavePai, LADO lado){
  PONT pai = buscarChave(chavePai,raiz);
  if (!pai) return false;
  PONT novo = criarNovoNo(novaChave);
  if (lado == esquerdo){

    pai->esq = novo;
  }



  return true;
}
```

Inserção de um nó filho

```
bool inserirFilho(PONT raiz, TIPOCHAVE novaChave, TIPOCHAVE chavePai, LADO lado){
  PONT pai = buscarChave(chavePai,raiz);
  if (!pai) return false;
  PONT novo = criarNovoNo(novaChave);
  if (lado == esquerdo){

    pai->esq = novo;
  }else{

    pai->dir = novo;
  }
  return true;
}
```

Inserção de um nó filho

```
bool inserirFilho(PONT raiz, TIPOCHAVE novaChave, TIPOCHAVE chavePai, LADO lado){
  PONT pai = buscarChave(chavePai,raiz);
  if (!pai) return false;
  PONT novo = criarNovoNo(novaChave);
  if (lado == esquerdo){
    novo->esq = pai->esq;
    pai->esq = novo;
  }else{
    novo->esq = pai->dir;
    pai->dir = novo;
  }
  return true;
}
```

AULA e06
Algoritmos e Estruturas de Dados I

Árvores Binárias

Luciano Antonio Digiampietri