"... Abrir um livro do Knuth tem um efeito aterrorizante muito útil nos computadores" (Bill Gates).

Inclusão em Árvores B

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR

Inserir

Toda nova chave é inserida em uma folha.

Se a folha estiver cheia (com *2t-1* chaves).

Necessária uma função para dividir um nodo *y* em dois com *t-1* chaves em cada.

A divisão é feita ao redor da chave mediana *y.chaves[t]*.

Inserir

Toda nova chave é inserida em uma folha.

Se a folha estiver cheia (com *2t-1* chaves).

- Necessária uma função para dividir um nodo *y* em dois com *t-1* chaves em cada.
- A divisão é feita ao redor da chave mediana *y.chaves[t]*.
- A chave mediana é movida para o pai de *y*.

Inserir

Toda nova chave é inserida em uma folha.

Se a folha estiver cheia (com *2t-1* chaves).

Necessária uma função para dividir um nodo *y* em dois com *t-1* chaves em cada.

A divisão é feita ao redor da chave mediana *y.chaves[t]*.

A chave mediana é movida para o pai de *y*.

Mas o pai pode estar cheio ...

Podemos precisar repetir até a raiz.

Inserir

Toda nova chave é inserida em uma folha.

Se a folha estiver cheia (com *2t-1* chaves).

Necessária uma função para dividir um nodo *y* em dois com *t-1* chaves em cada.

A divisão é feita ao redor da chave mediana *y.chaves[t]*.

A chave mediana é movida para o pai de *y*.

Mas o pai pode estar cheio ...

Podemos precisar repetir até a raiz.

Para evitar subir até a raiz, é comum dividir todo nodo cheio conforme descemos na árvore para buscar a posição onde a chave será inserida.

DividirFilho

função **dividirFilho**(x,i)
entrada: nodo interno x não cheio e um índice i tal que x.filhos[i] é um filho cheio de x. Ambos x e x.filhos[i] devem estar na memória principal.
saída: o nodo apontado por x.filhos[i] é dividido em dois e x é ajustado.

```
y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```

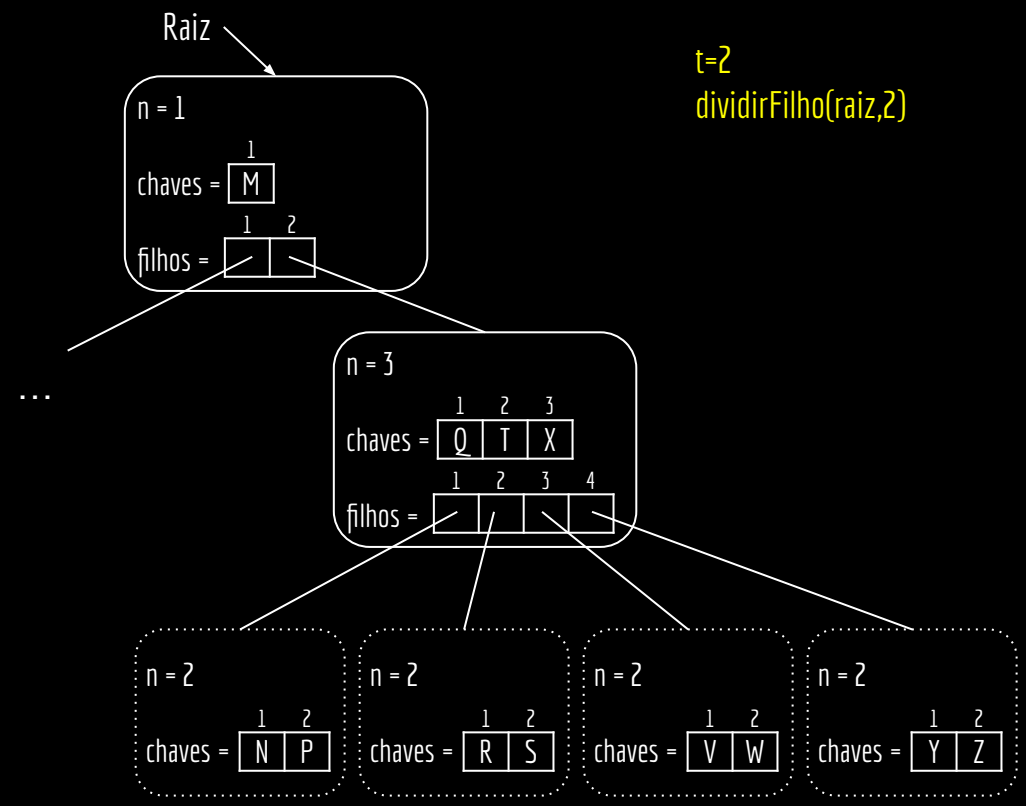
DividirFilho

```
função dividirFilho(x,i)
entrada: nodo interno x não cheio e um índice i tal que x.filhos[i] é um filho cheio de
x. Ambos x e x.filhos[i] devem estar na memória principal.
saída: o nodo apontado por x.filhos[i] é dividido em dois e x é ajustado.

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```

Em 2 loops pelo bem da clareza. Para uma implementação mais eficiente, tente fazer tudo em um único loop.

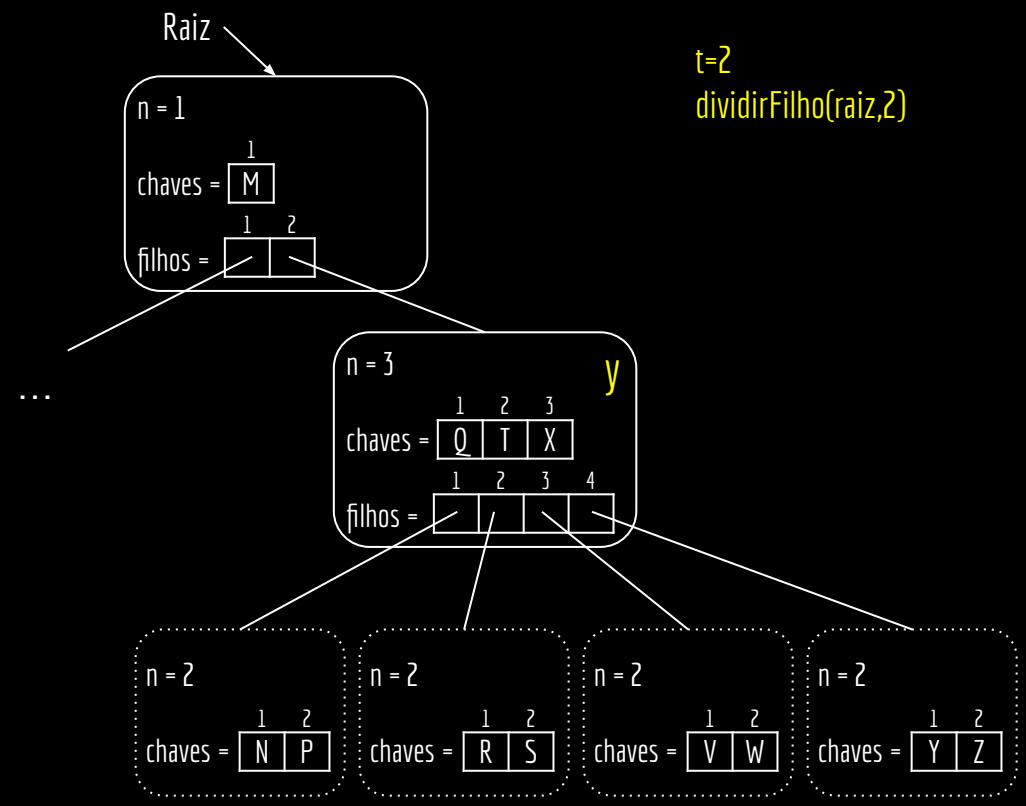
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



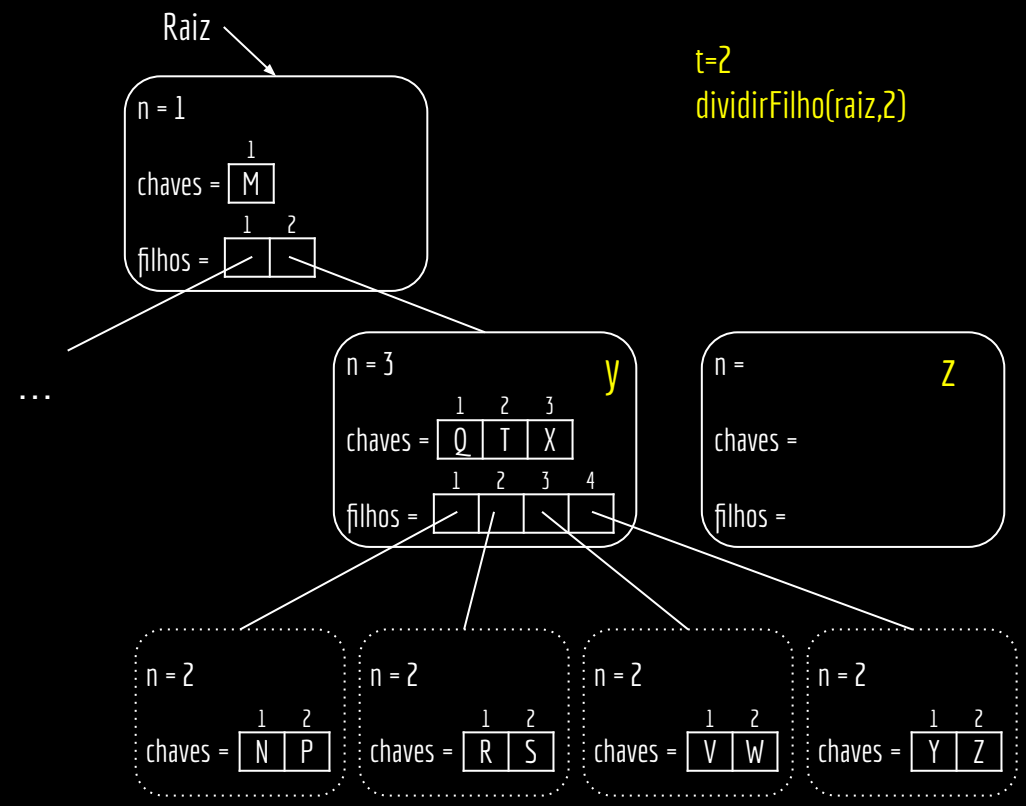
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



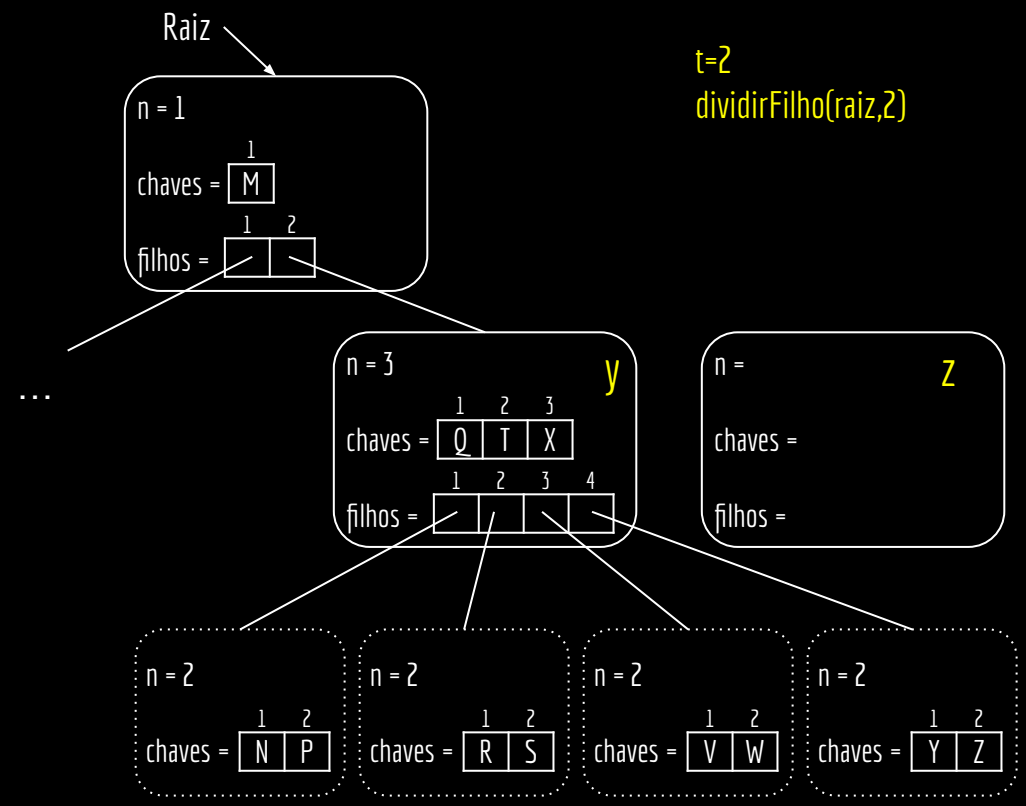
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



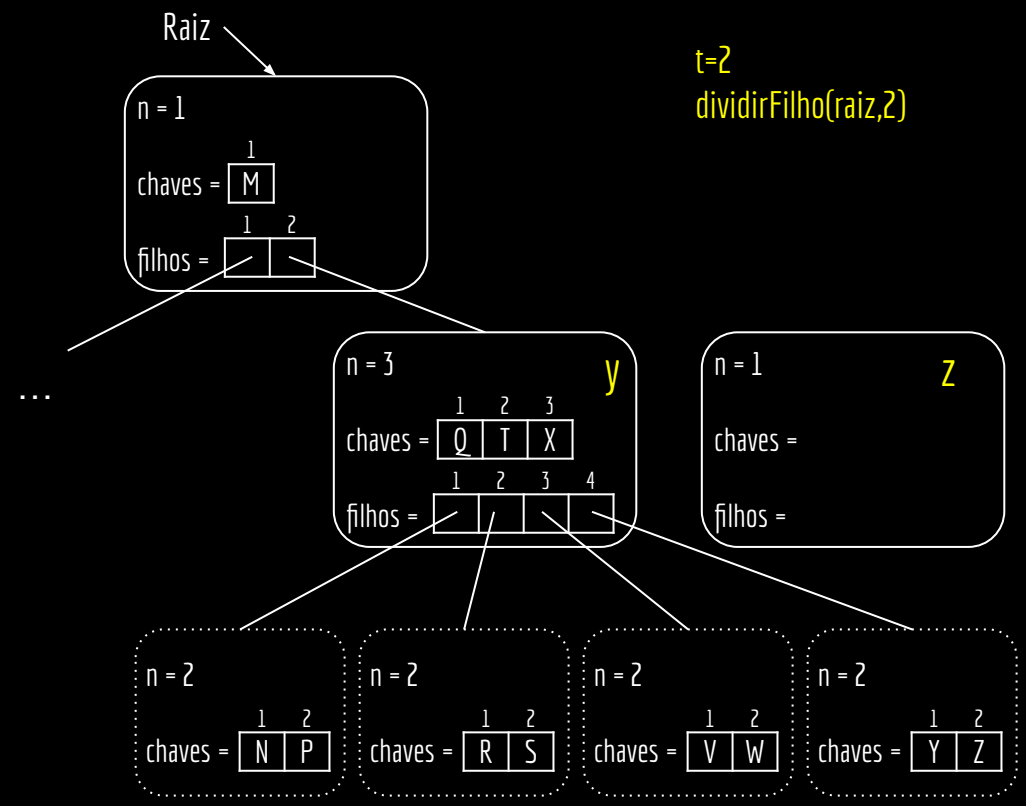
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



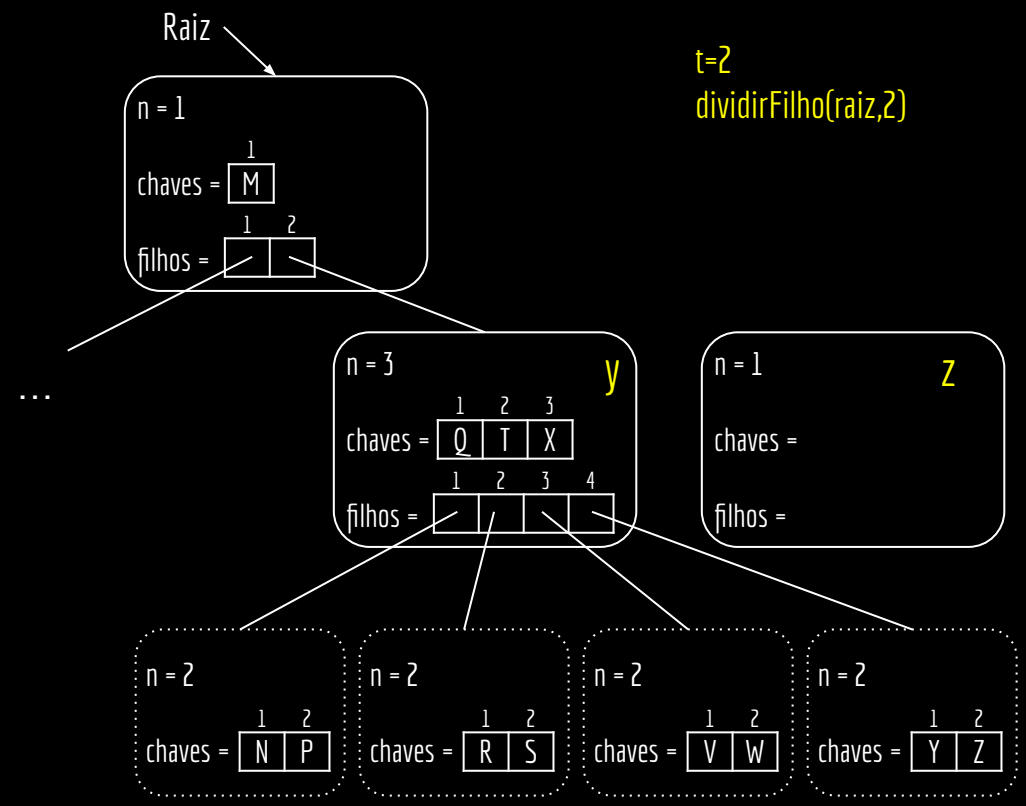
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



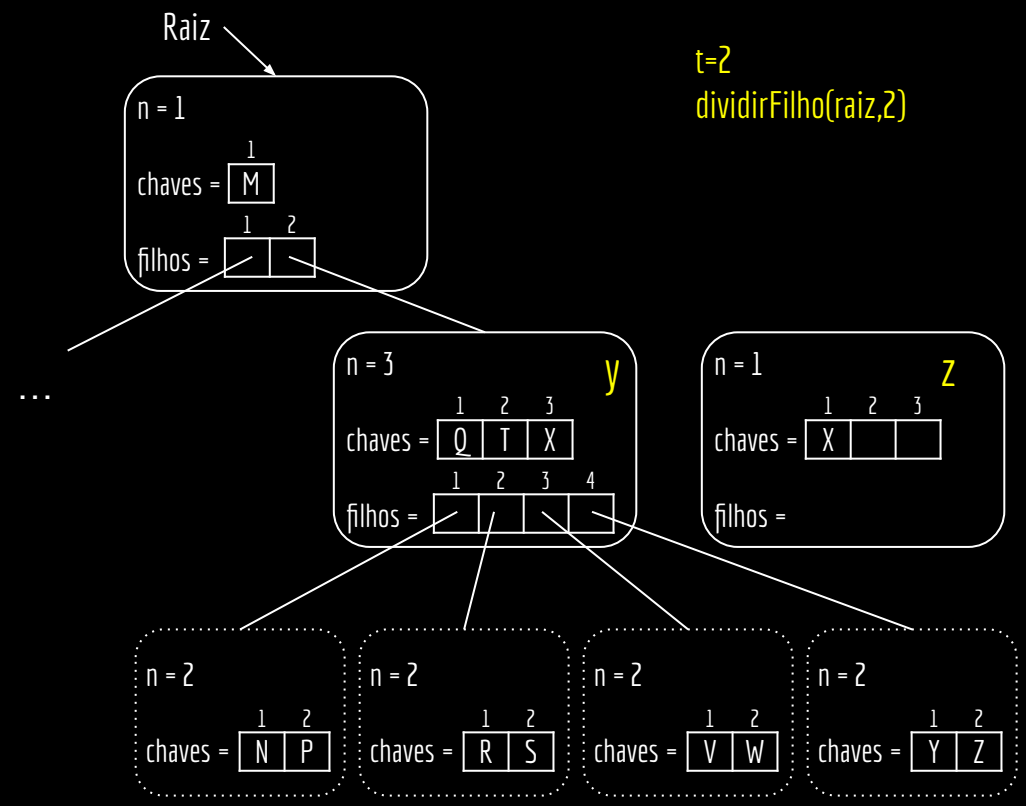
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
      z.chave[j] = y.chave[j+t]
se y não é folha
      para j=1 até t
            z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
      x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
      x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



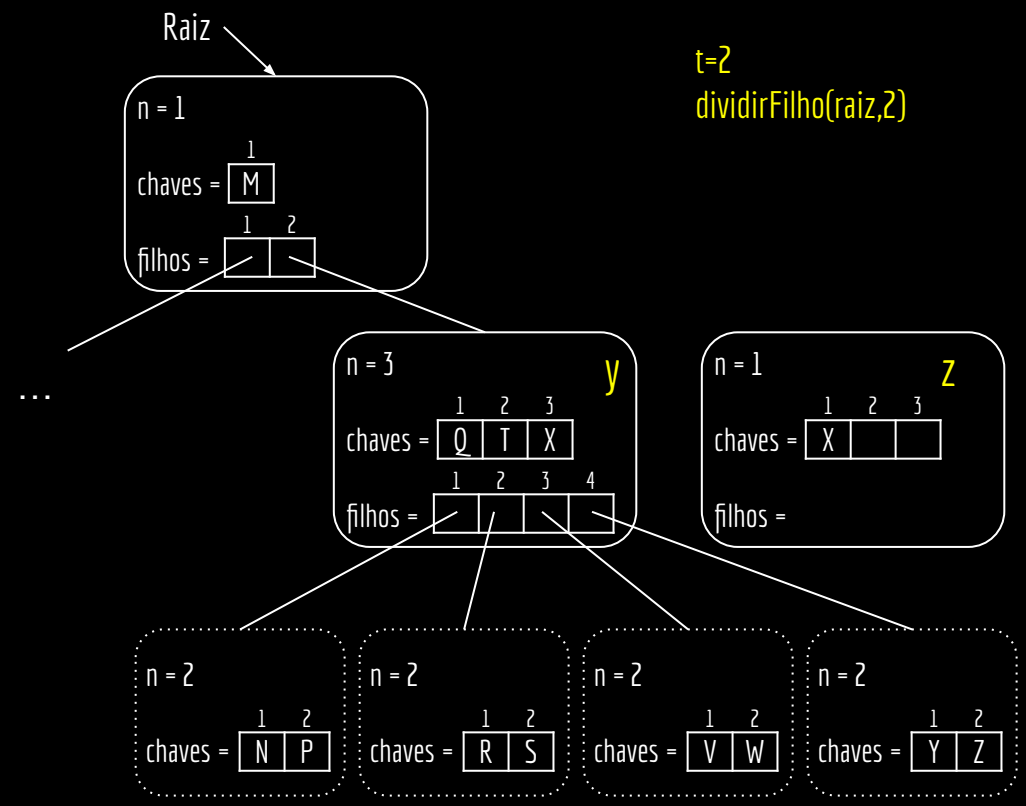
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
      z.chave[j] = y.chave[j+t]
se y não é folha
      para j=1 até t
            z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
      x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
      x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



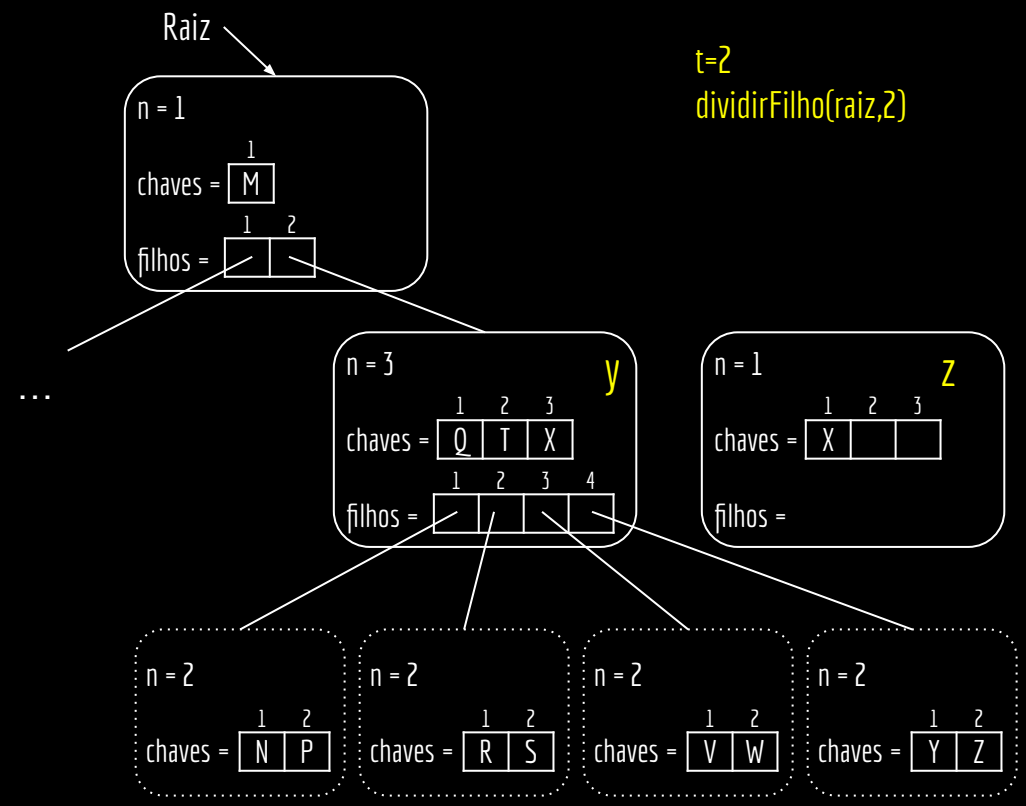
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



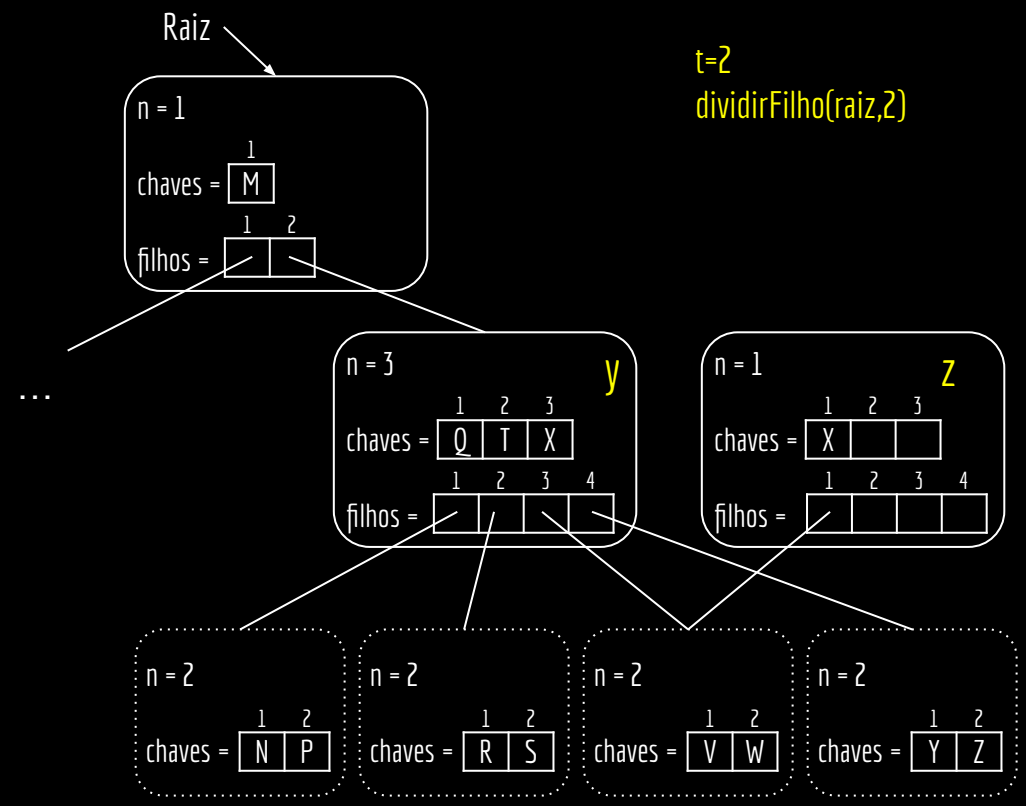
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



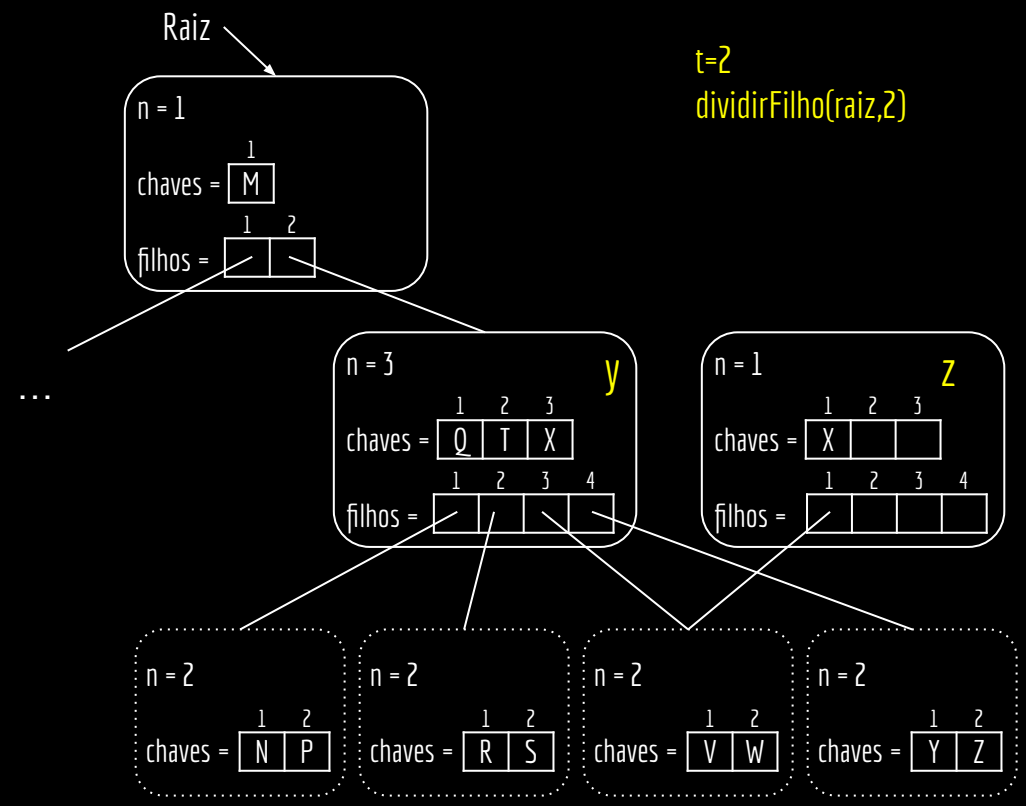
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



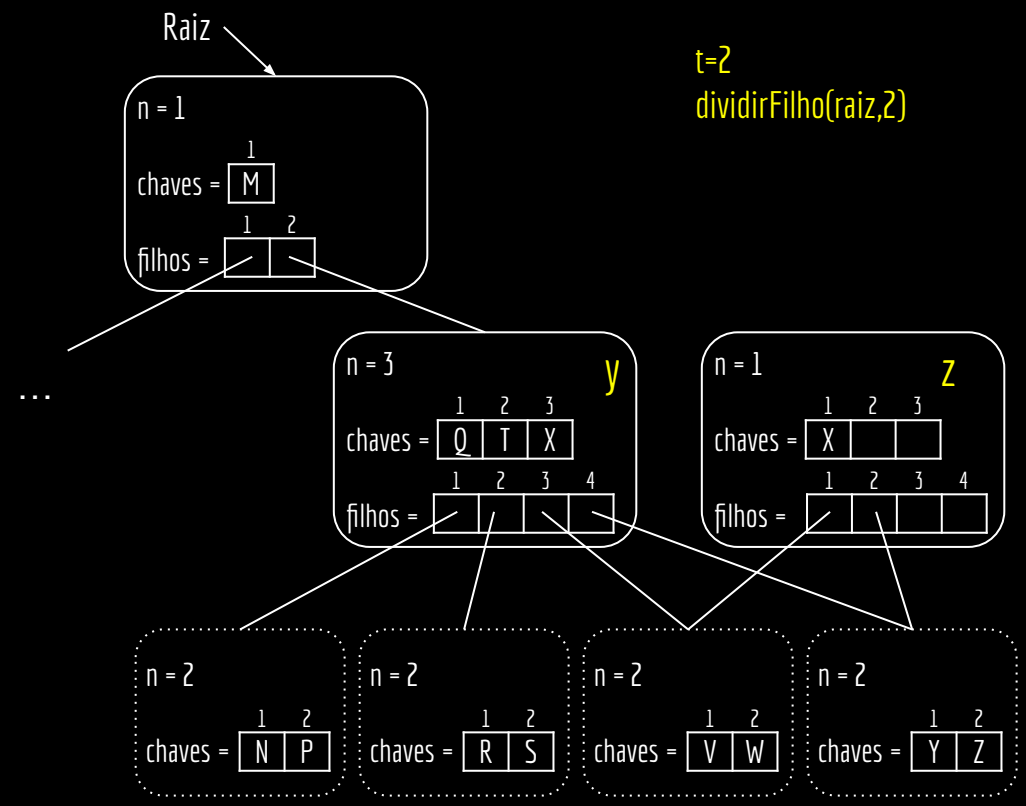
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



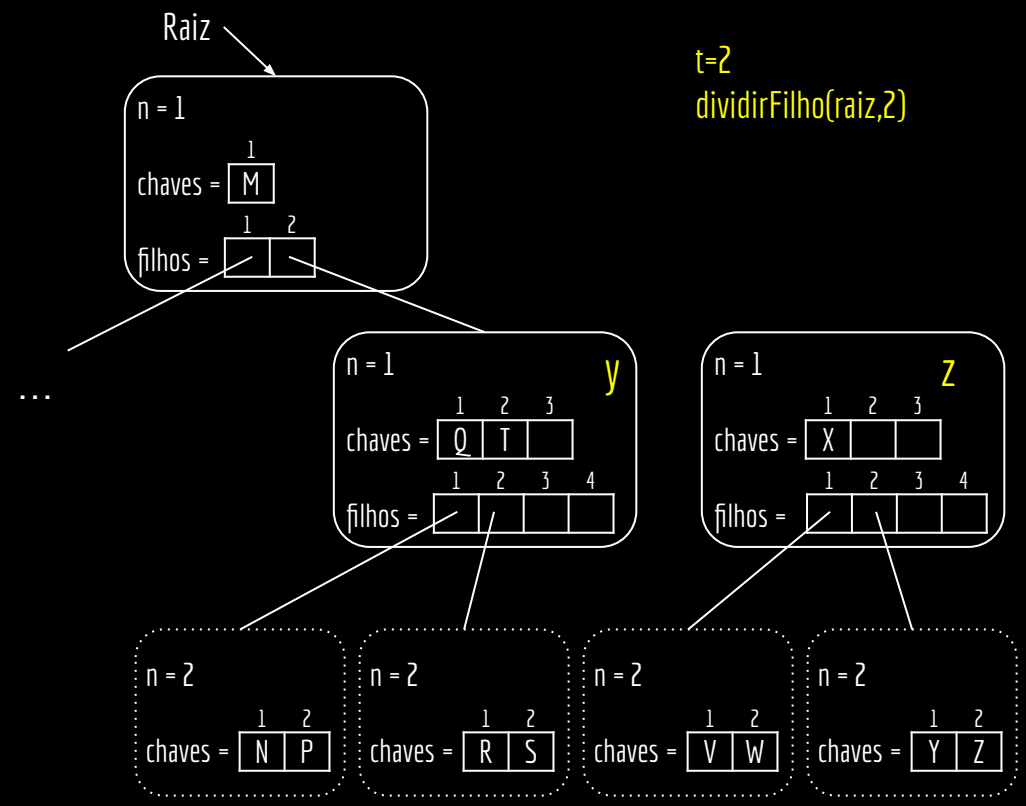
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



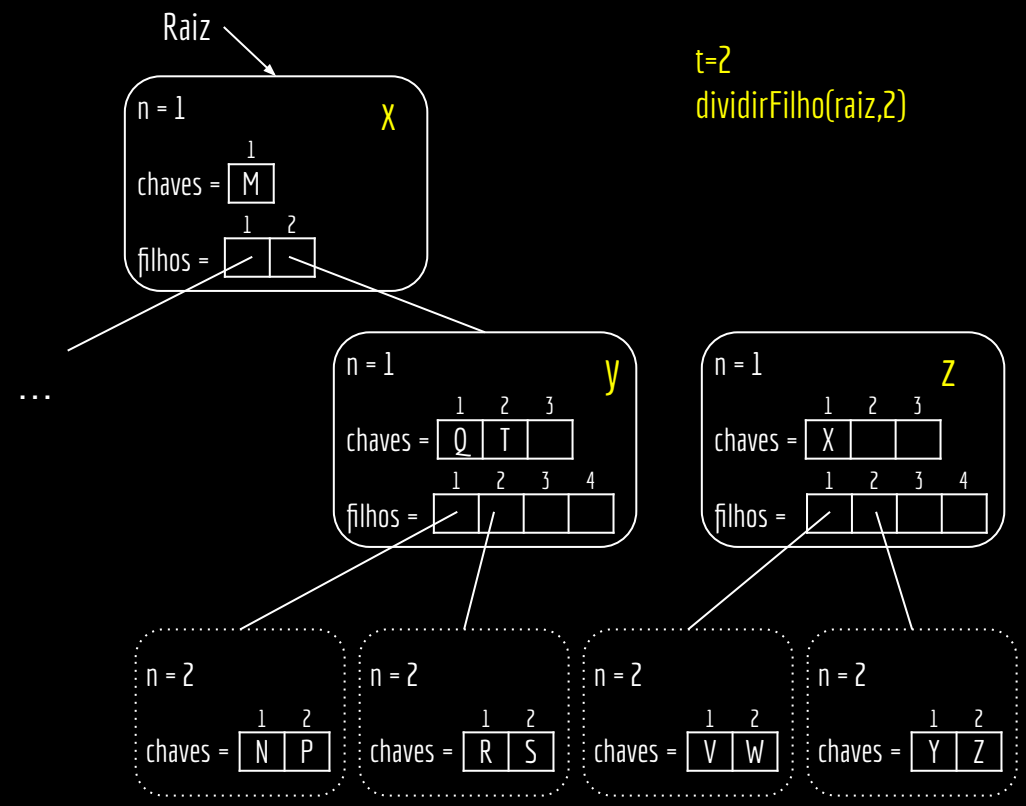
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
      z.chave[j] = y.chave[j+t]
se y não é folha
      para j=1 até t
            z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
      x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
      x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



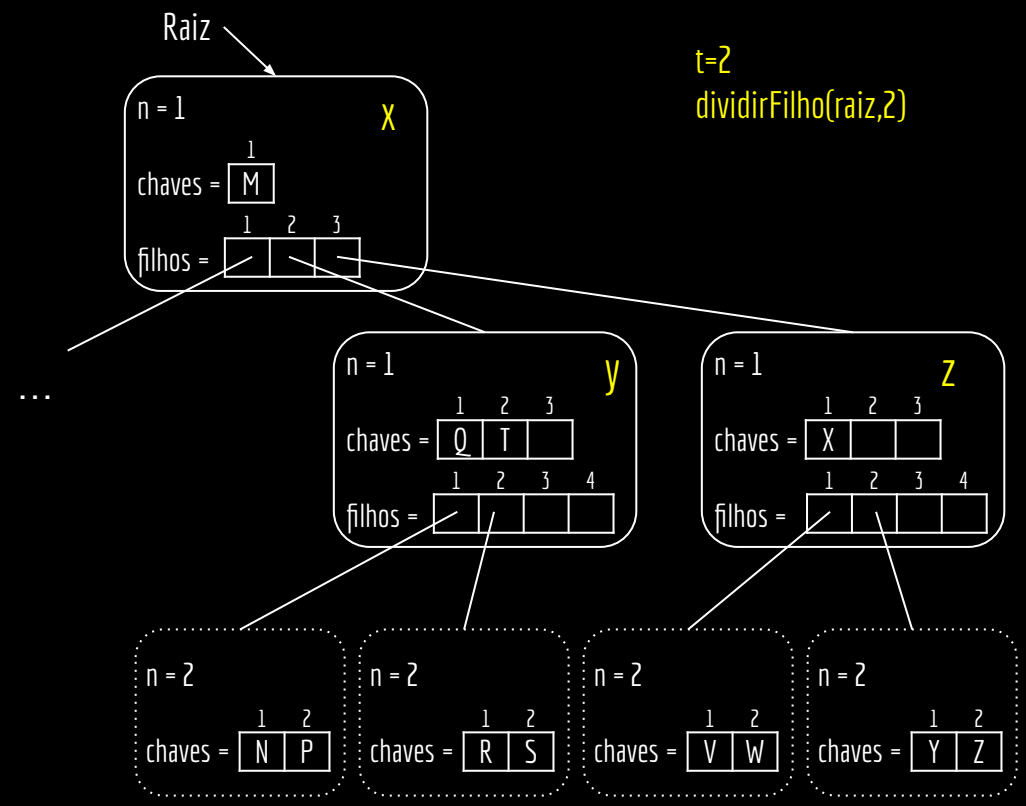
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



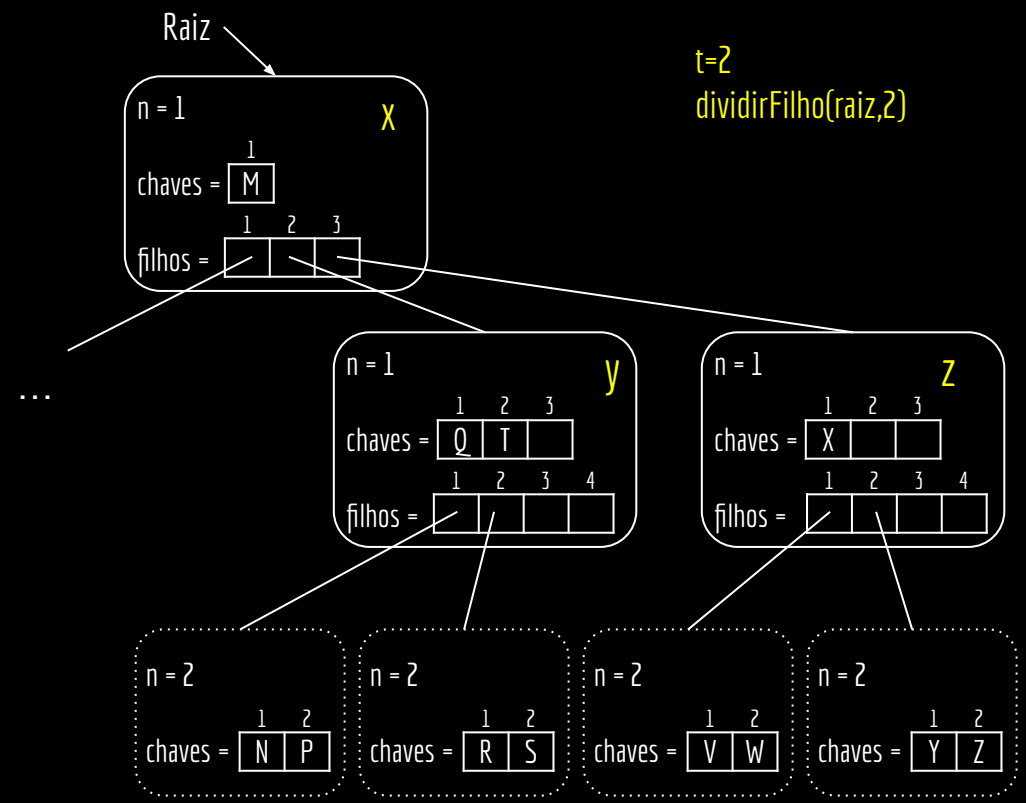
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



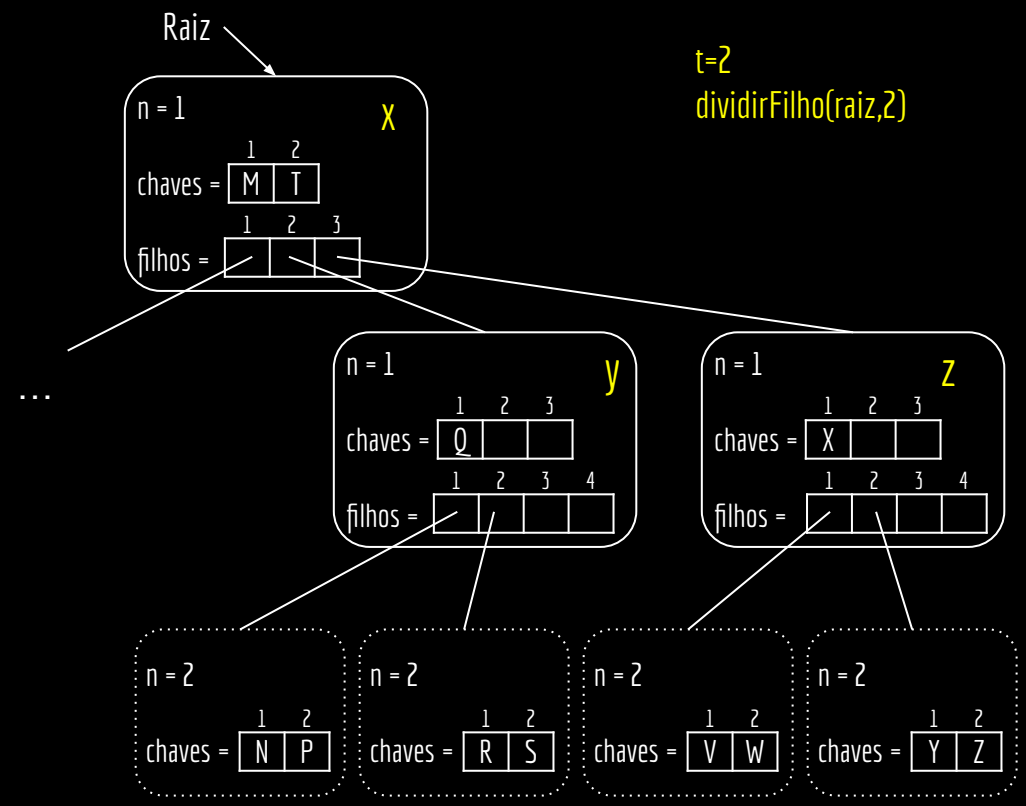
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
      z.chave[j] = y.chave[j+t]
se y não é folha
      para j=1 até t
            z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
      x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
      x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



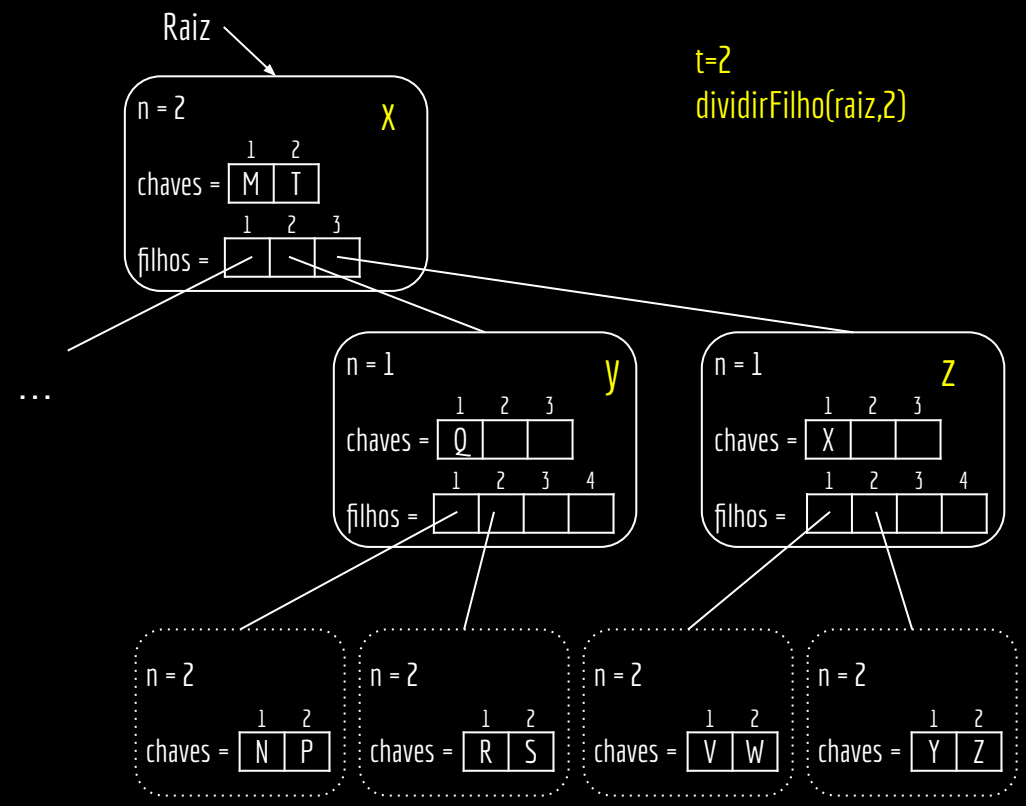
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



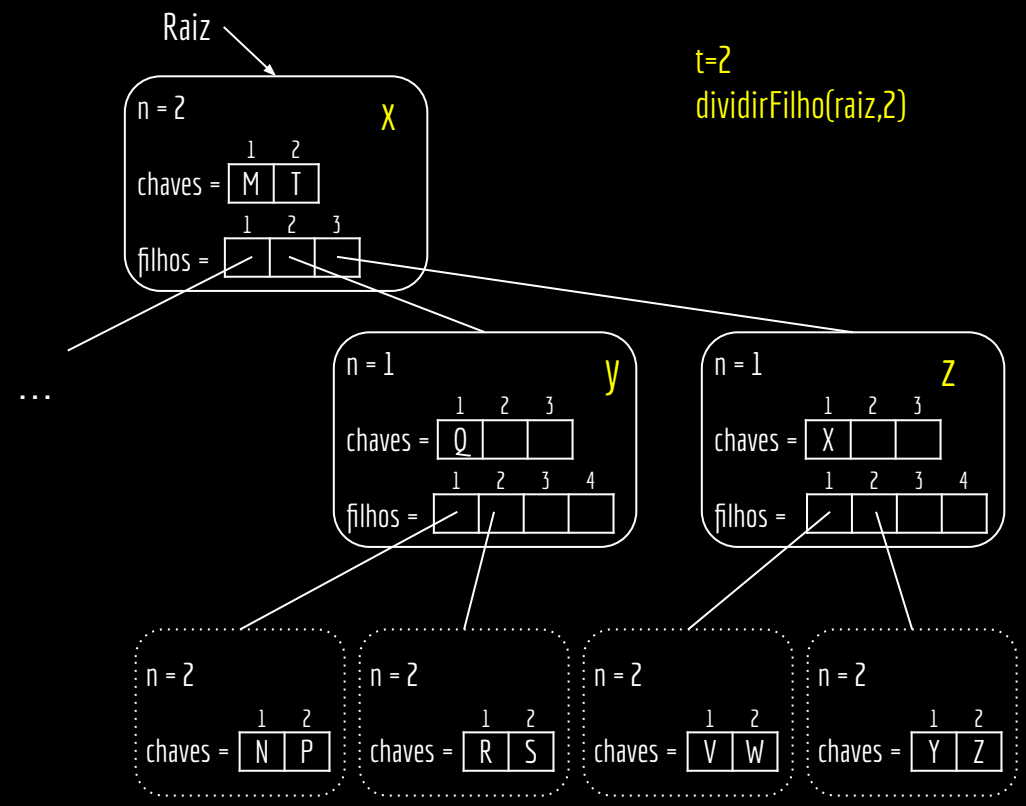
Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



Teste de Mesa

```
função dividirFilho(x,i)

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```



Custo

```
função dividirFilho(x,i)
entrada: nodo interno x não cheio e um índice i tal que x.filhos[i] é um filho cheio de
x. Ambos x e x.filhos[i] devem estar na memória principal.
saída: o nodo apontado por x.filhos[i] é dividido em dois e x é ajustado.

y = x.filhos[i]
z = alocarNodo()
z.ehFolha = y.ehFolha
z.n = t-1
para j=1 até t-1
     z.chave[j] = y.chave[j+t]
se y não é folha
     para j=1 até t
          z.filhos[j] = y.filhos[j+t]
y.n = t-1
para j = x.n+1 até i+1 passo -1
     x.filhos[j+1] = x.filhos[j]
x.filhos[i+1] = z
para j = x.n até i passo -1
     x.chaves[j+1] = x.chaves[j]
x.chave[i] = y.chave[t]
x.n = x.n+1
armazenar(y)
armazenar(z)
armazenar(x)
```

Devido aos loops, o algoritmo tem um custo de tempo $\Theta(t)$.

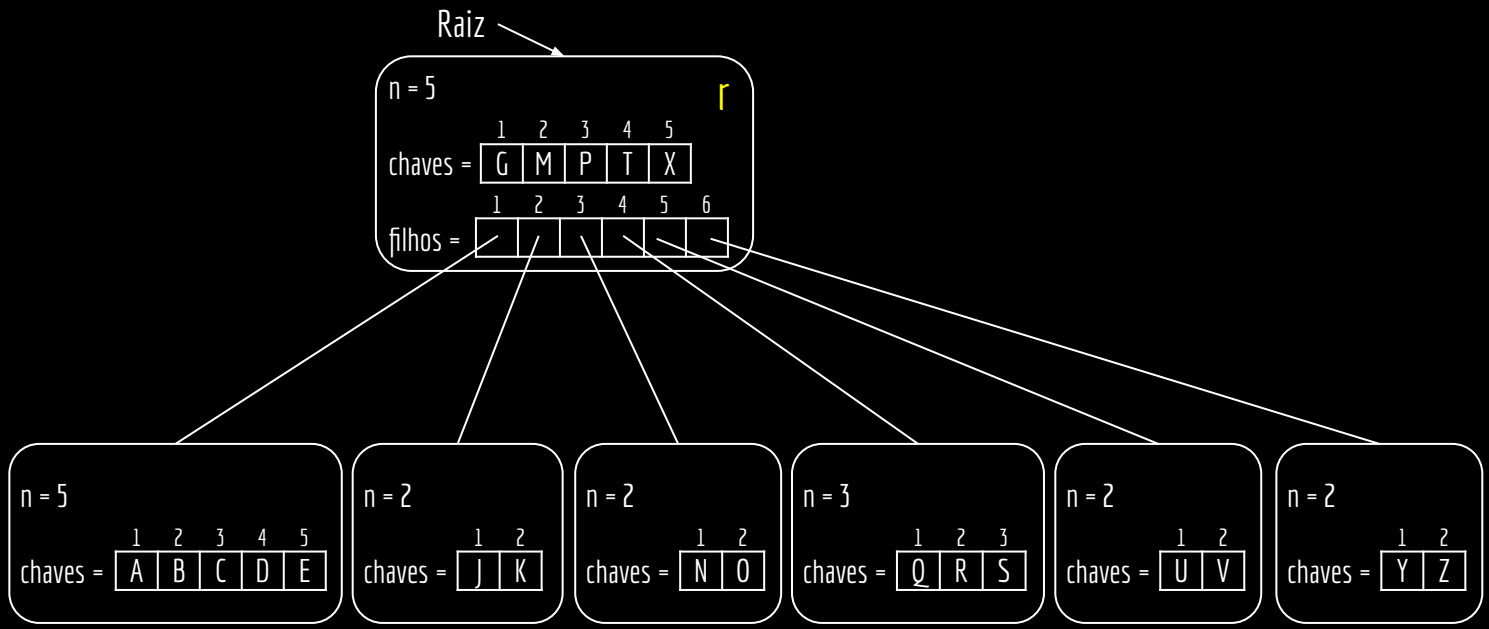
Custo $O(1)$ de operações com a memória secundária.

Inserir

```
função inserirArvoreB(T,k)
entrada: árvore B T e chave k.
saída: a chave k é inserida em T.

r = T.raiz
se r.n == 2t-1
     s = dividirRaiz(T)
     inserirNaoCheio(s,k)
senão
     inserirNaoCheio(r,k)
```

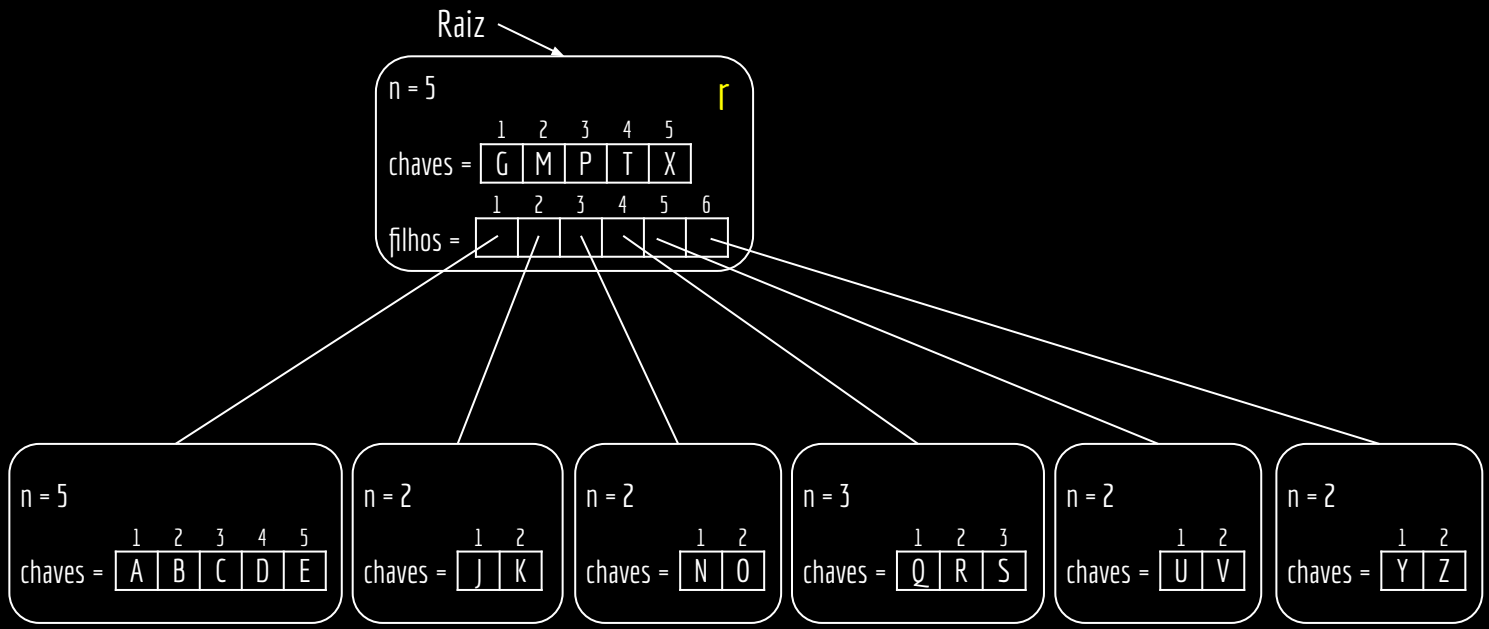
```
função inserirArvoreB(T,k)

r = T.raiz
se r.n == 2t-1
     s = dividirRaiz(T)
     inserirNaoCheio(s,k)
senão
     inserirNaoCheio(r,k)
```

t=3
inserirArvoreB(T,L)



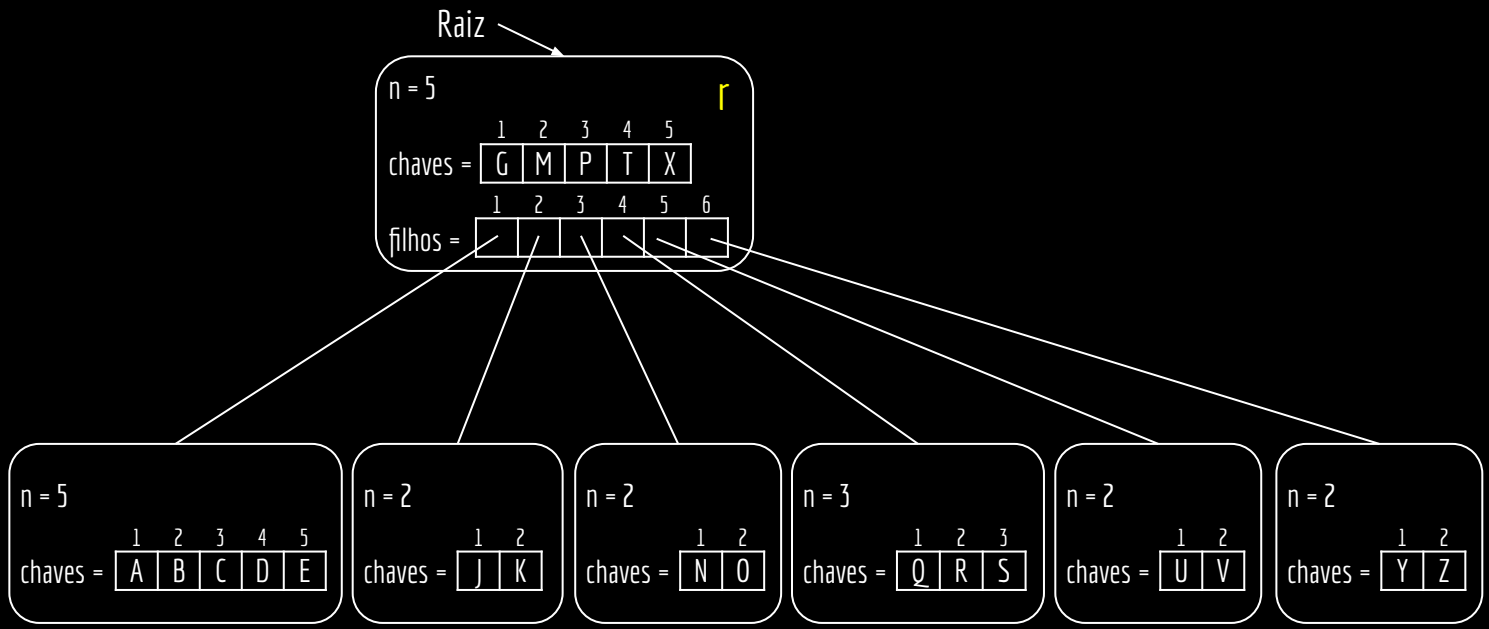
```
função inserirArvoreB(T,k)

r = T.raiz
se r.n == 2t-1
     s = dividirRaiz(T)
     inserirNaoCheio(s,k)
senão
     inserirNaoCheio(r,k)
```

t=3
inserirArvoreB(T,L)



```
função inserirArvoreB(T,k)

r = T.raiz
se r.n == 2t-1
     s = dividirRaiz(T)
     inserirNaoCheio(s,k)
senão
     inserirNaoCheio(r,k)
```

t=3
inserirArvoreB(T,L)



DividirRaiz

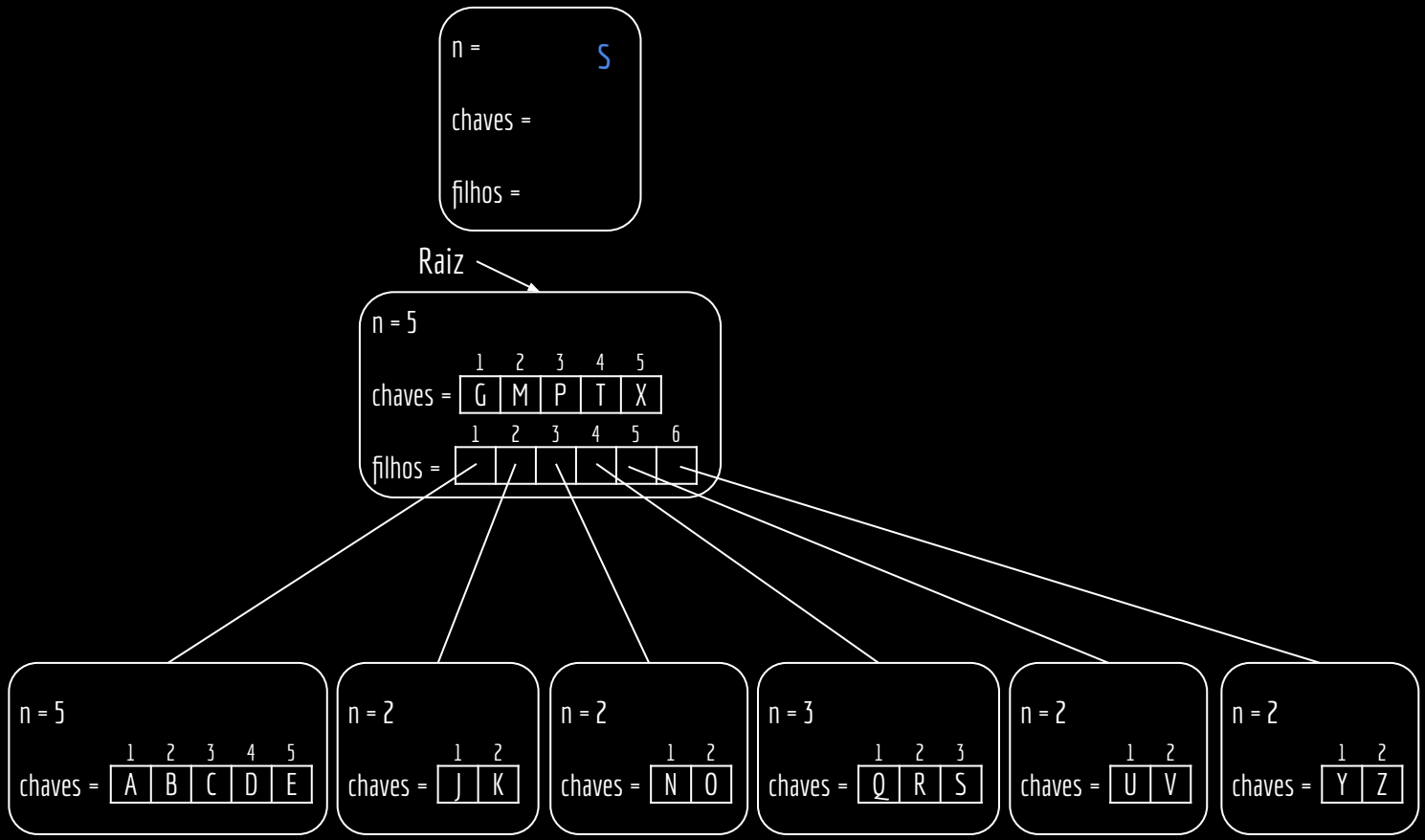
função **dividirRaiz**(T)
entrada: árvore B T.
saída: é criada uma nova raiz. A raiz anterior é dividida em duas, onde cada parte se torna filha da nova raiz.

```
s = alocarNodo()
s.ehFolha = falso
s.n = 0
s.filhos[1] = T.raiz
T.raiz = s
dividirFilho(s,1)
retorne s
```

função **dividirRaiz**(T)

```
s = alocarNodo()
s.ehFolha = falso
s.n = 0
s.filhos[1] = T.raiz
T.raiz = s
dividirFilho(s,1)
retorne s
```

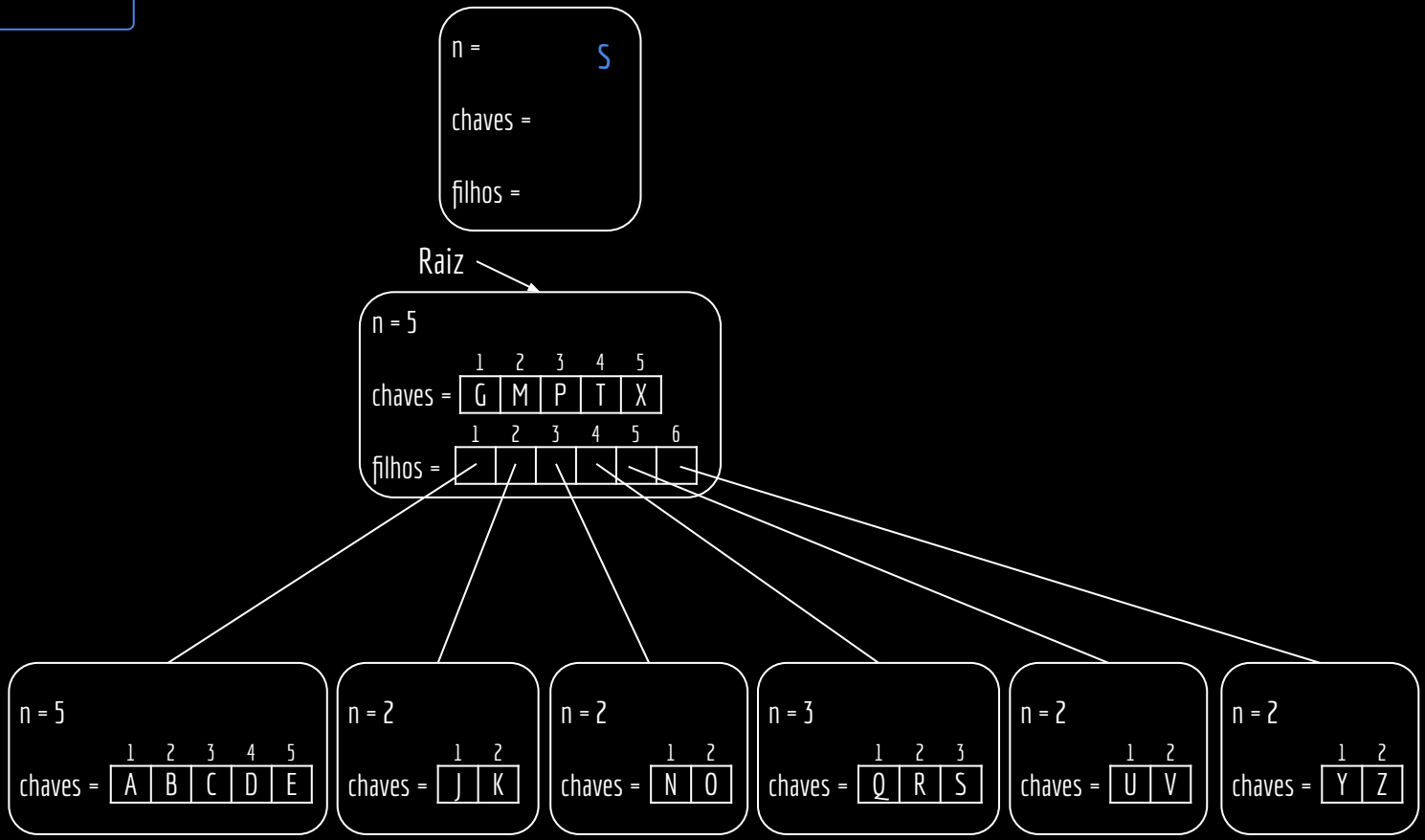
t=3
inserirArvoreB(T,L)



função **dividirRaiz**(T)

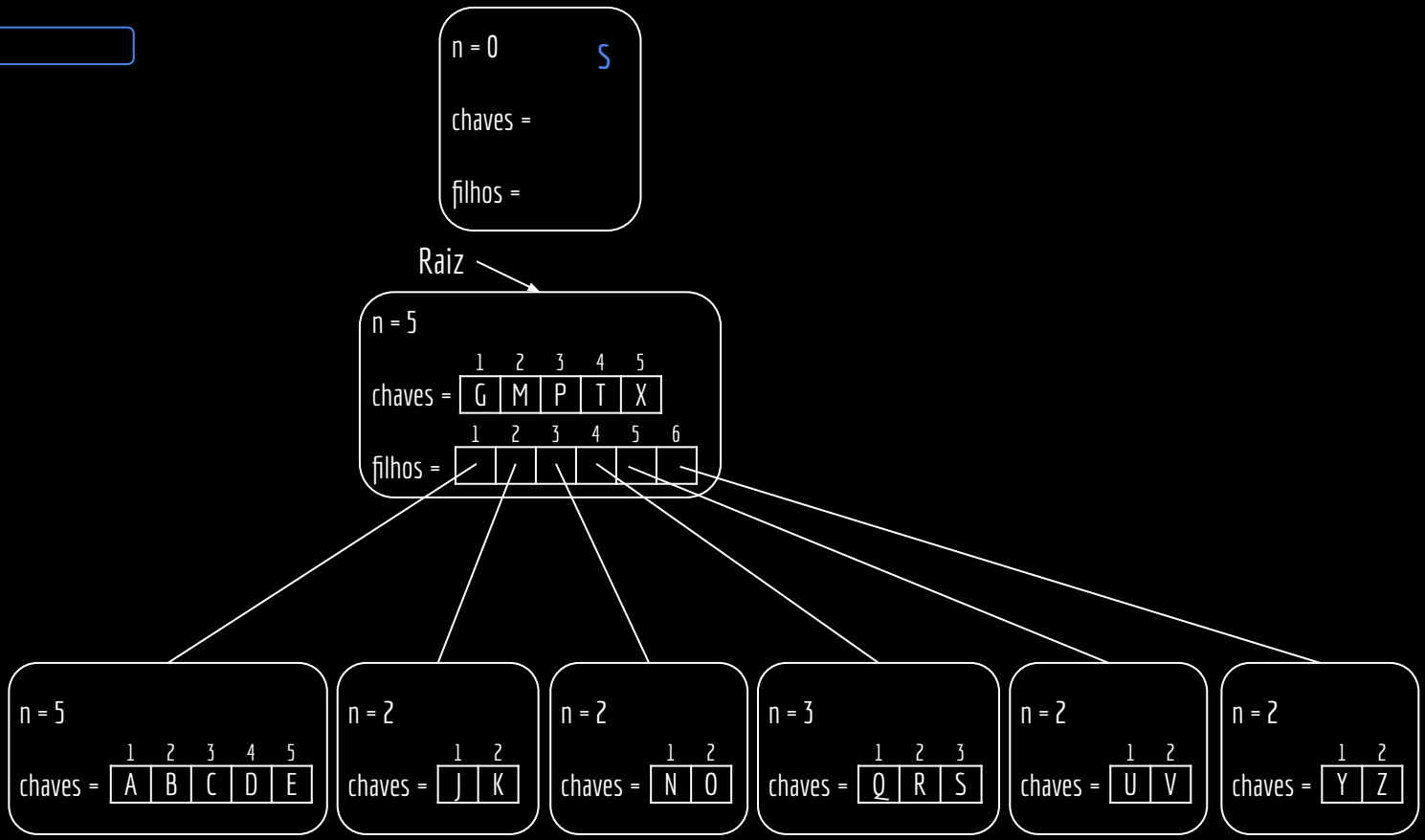
```
s = alocarNodo()
s.ehFolha = falso
s.n = 0
s.filhos[1] = T.raiz
T.raiz = s
dividirFilho(s,1)
retorne s
```

t=3
inserirArvoreB(T,L)



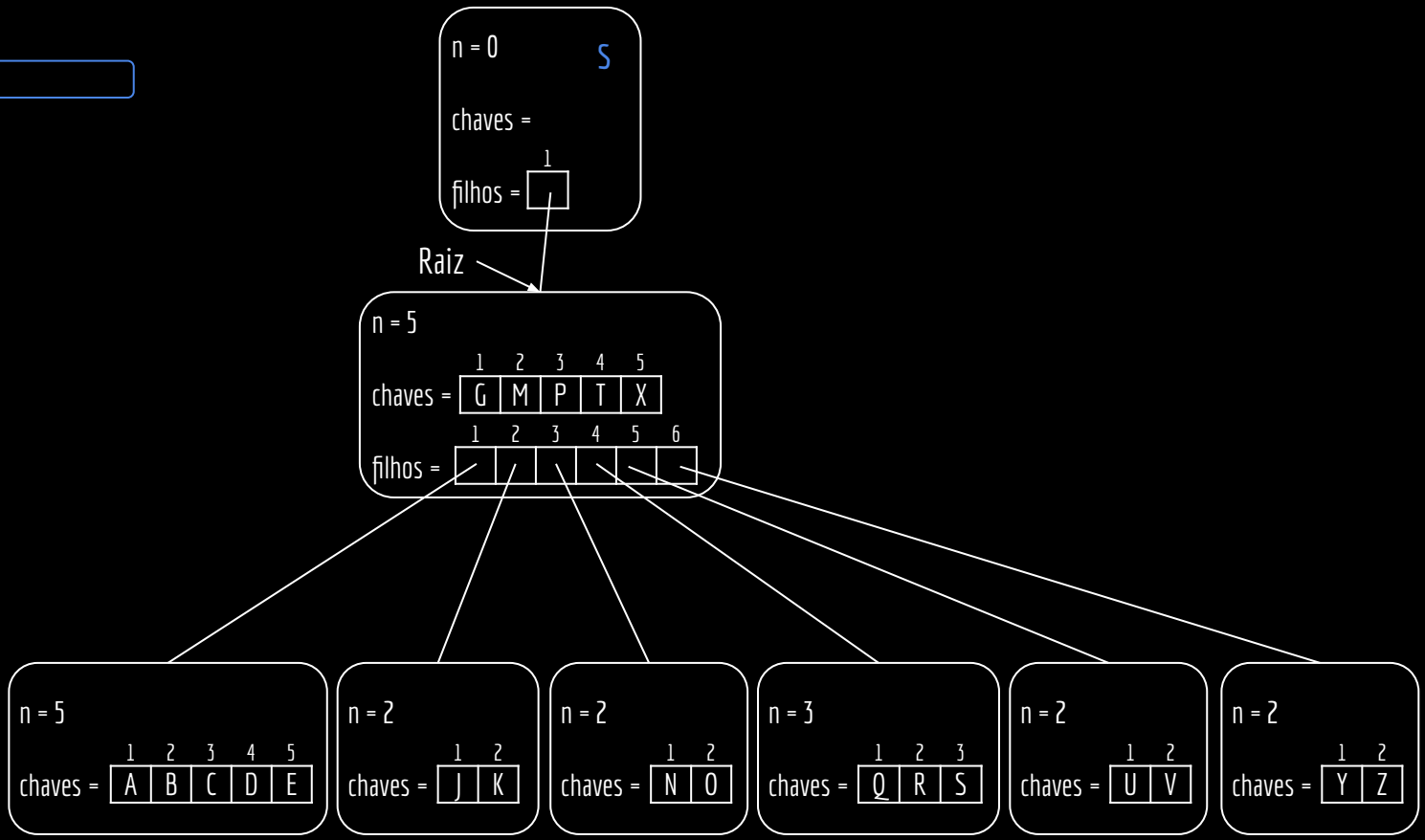
```
função dividirRaiz(T)

s = alocarNodo()
s.ehFolha = falso
s.n = 0
s.filhos[1] = T.raiz
T.raiz = s
dividirFilho(s,1)
retorne s
```

t=3
inserirArvoreB(T,L)



```
função dividirRaiz(T)

s = alocarNodo()
s.ehFolha = falso
s.n = 0
s.filhos[1] = T.raiz
T.raiz = s
dividirFilho(s,1)
retorne s
```

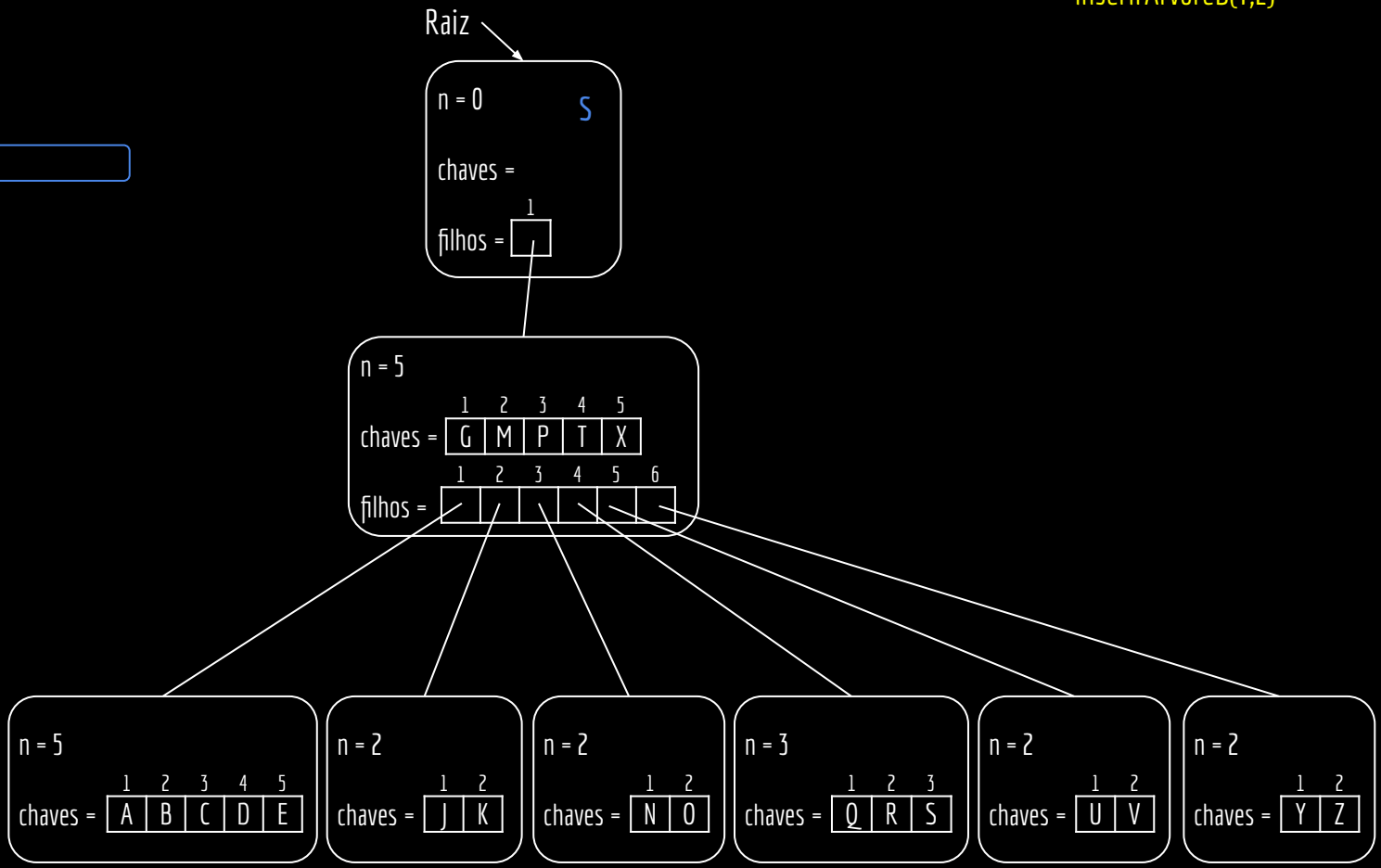
t=3
inserirArvoreB(T,L)



função **dividirRaiz**(T)

```
s = alocarNodo()
s.ehFolha = falso
s.n = 0
s.filhos[1] = T.raiz
T.raiz = s
dividirFilho(s,1)
retorne s
```

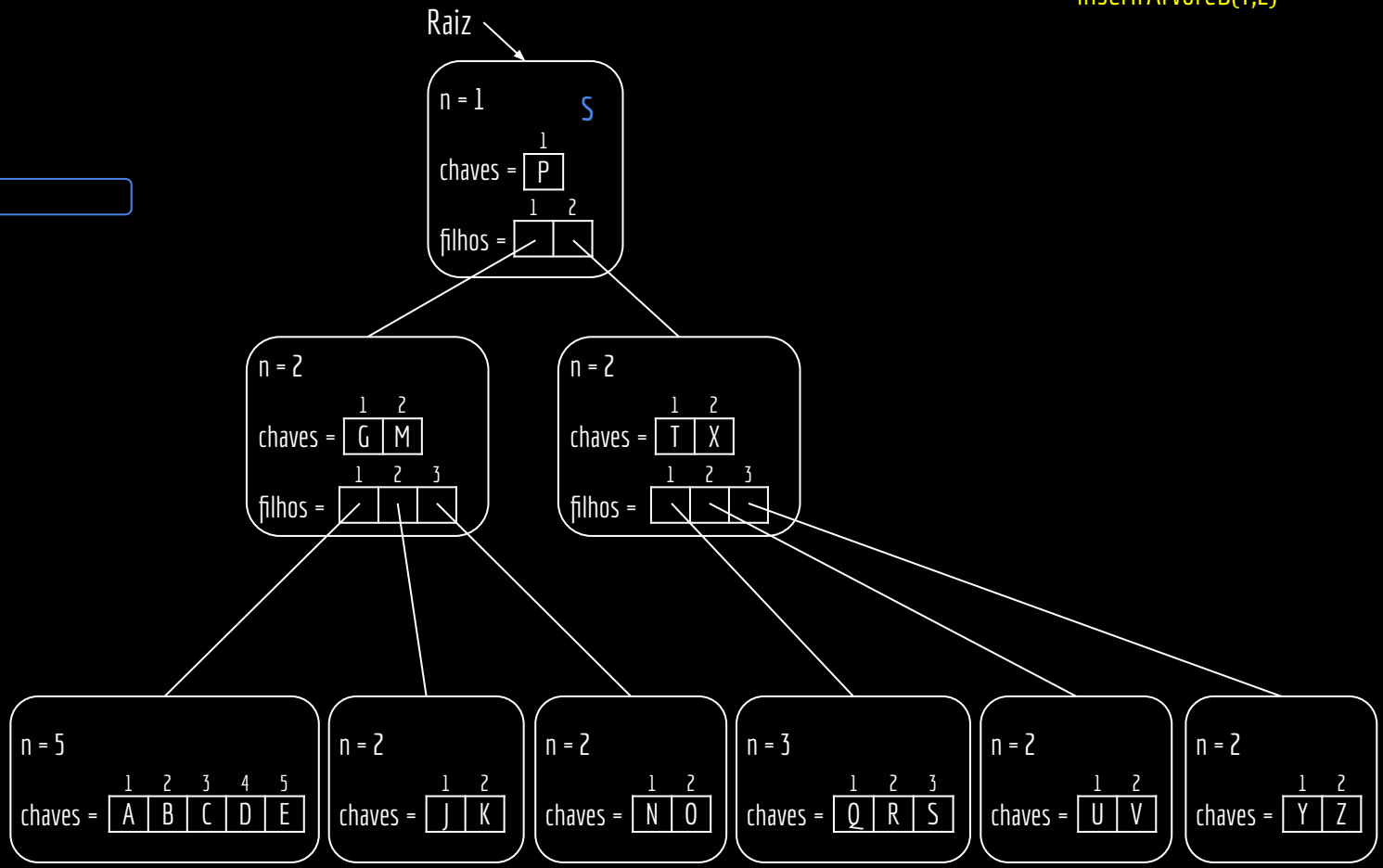
t=3
inserirArvoreB(T,L)



função **dividirRaiz**(T)

```
s = alocarNodo()
s.ehFolha = falso
s.n = 0
s.filhos[1] = T.raiz
T.raiz = s
dividirFilho(s,1)
retorne s
```

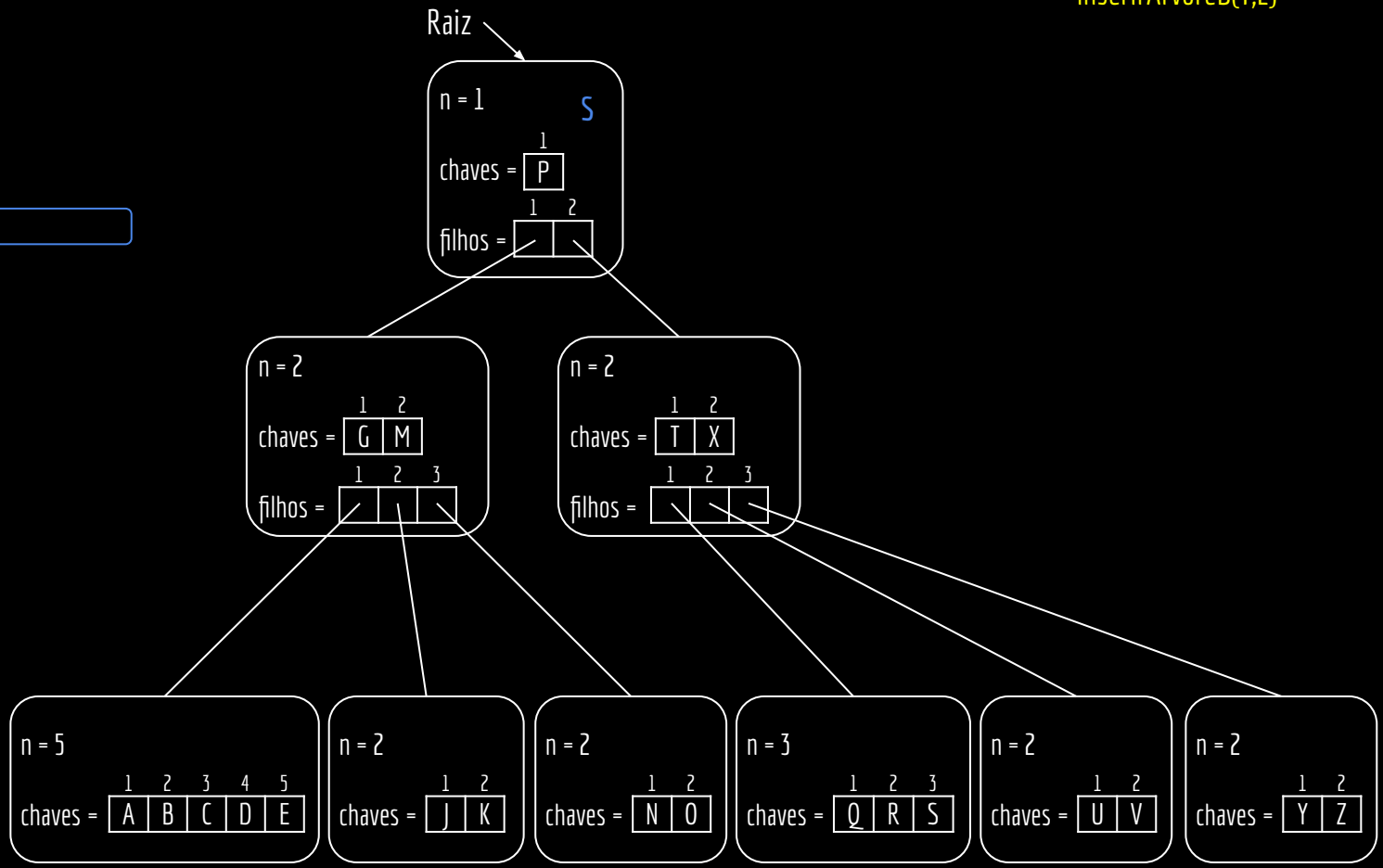
t=3
inserirArvoreB(T,L)



função **dividirRaiz**(T)

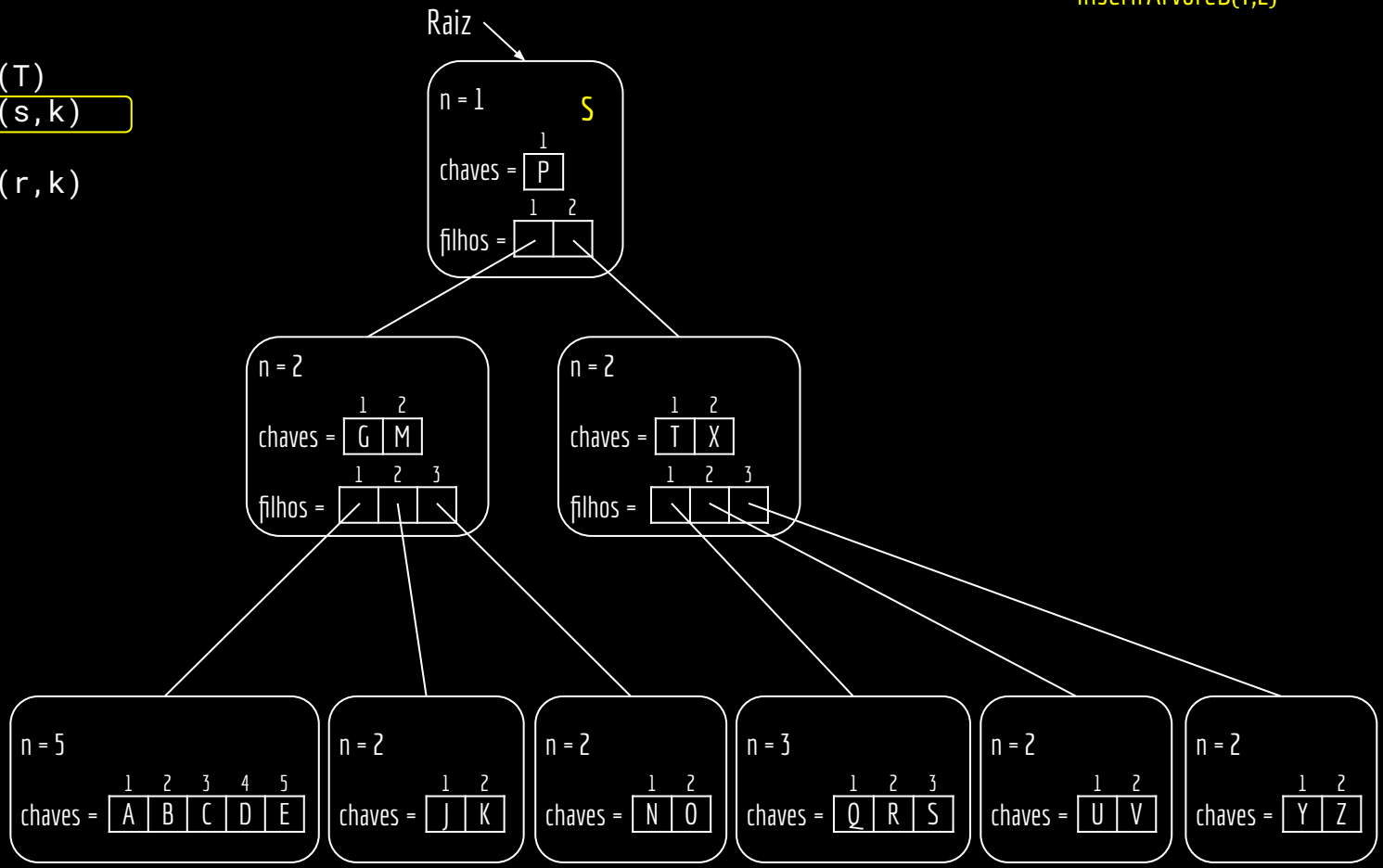
```
s = alocarNodo()
s.ehFolha = falso
s.n = 0
s.filhos[1] = T.raiz
T.raiz = s
dividirFilho(s,1)
retorne s
```

t=3
inserirArvoreB(T,L)



```
função inserirArvoreB(T,k)

r = T.raiz
se r.n == 2t-1
     s = dividirRaiz(T)
     inserirNaoCheio(s,k)
senão
     inserirNaoCheio(r,k)
```

t=3
inserirArvoreB(T,L)



InserirNaoCheio

```
função inserirNaoCheio(x,k)
entrada: nodo x não cheio, e a chave k a ser inserida.
saída: a chave k é inserida na árvore B.

i = x.n
se x é folha
     enquanto i ≥ 1 e k < x.chaves[i]
          x.chave[i+1] = x.chave[i]//abrindo espaço
          i = i - 1
     x.chave[i+1] = k
     x.n = x.n + 1
     armazenar(x)
senão
     enquanto i ≥ 1 e k < x.chaves[i]
          i = i - 1
     i = i + 1 //encontrado filho onde a chave pertence
     carregar(x.filhos[i])
     se x.filhos[i].n == 2t - 1 //filho está cheio
          dividirFilho(x,i)
          se k > x.chaves[i] //depois da divisão, para que nodo a chave vai?
               i = i + 1
     inserirNaoCheio(x.filhos[i],k) //recursão
```

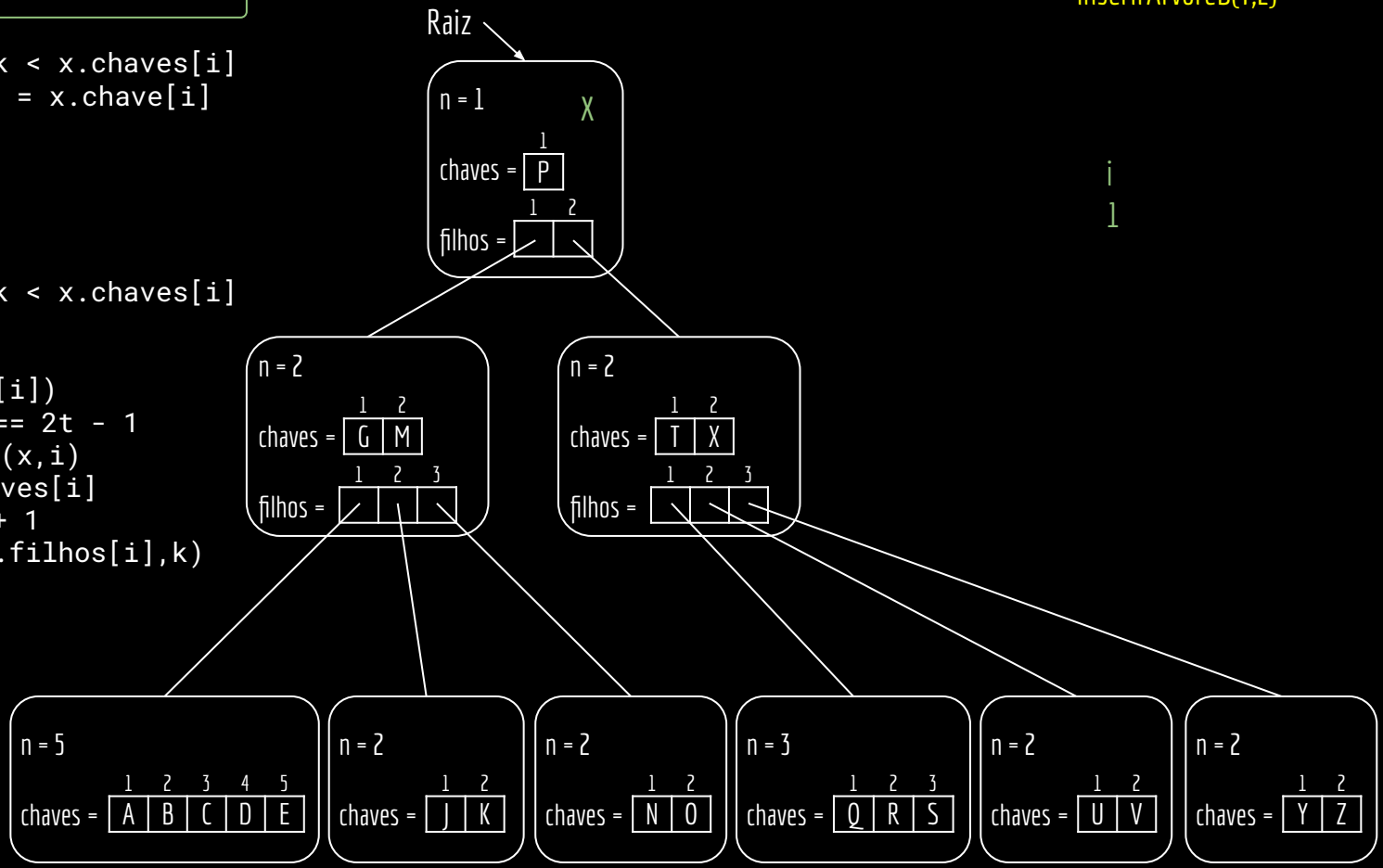
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

1



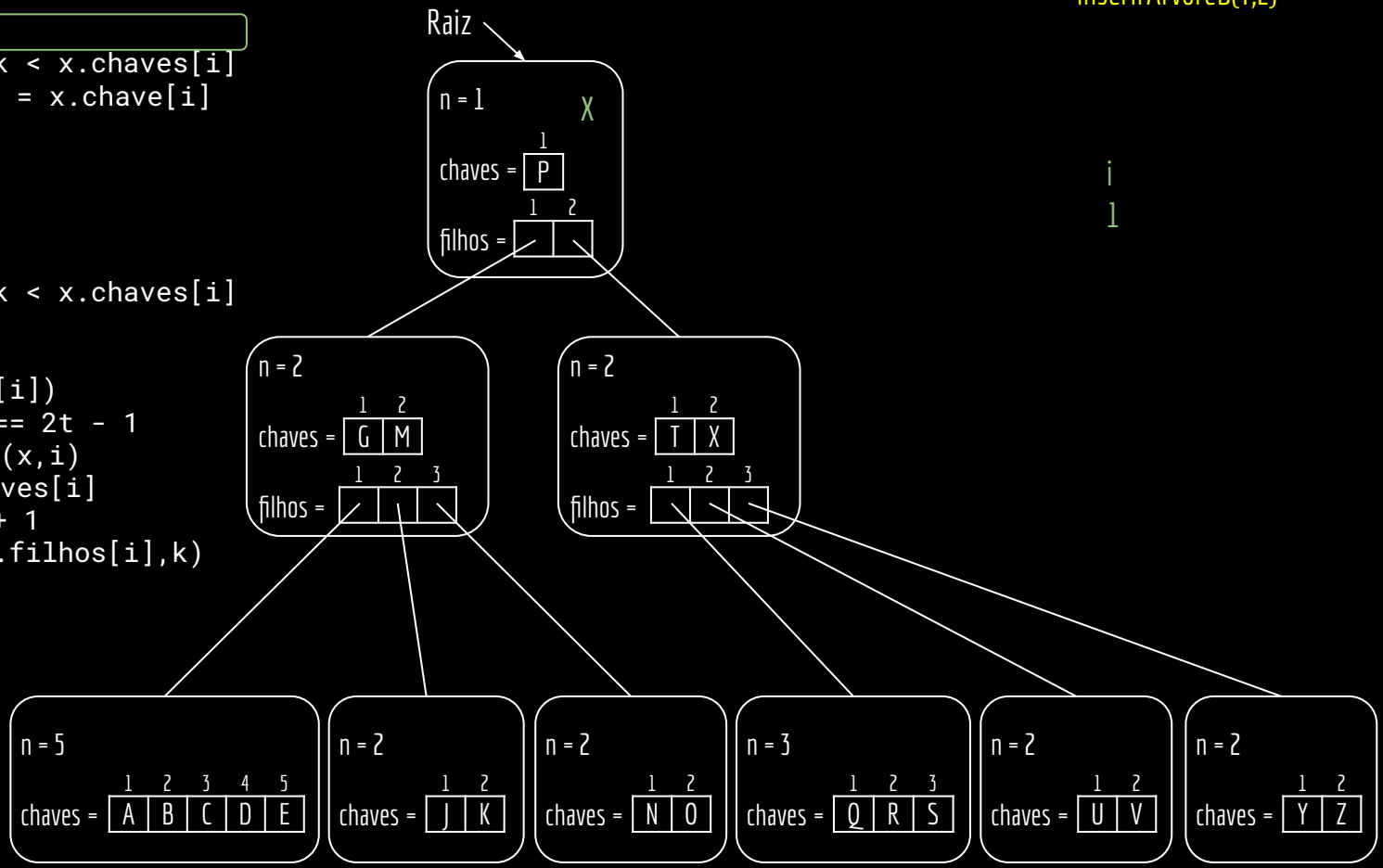
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

1



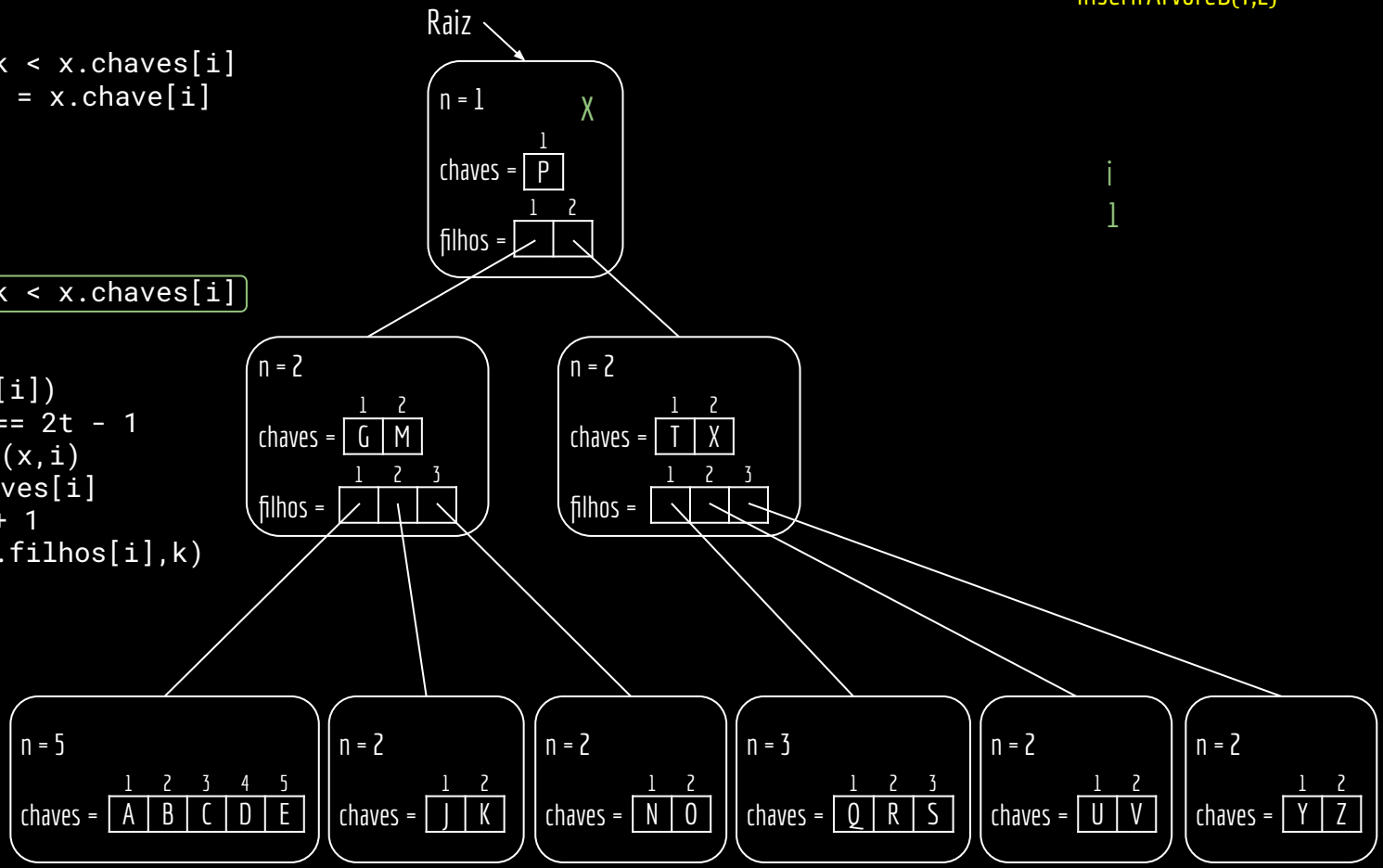
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

1



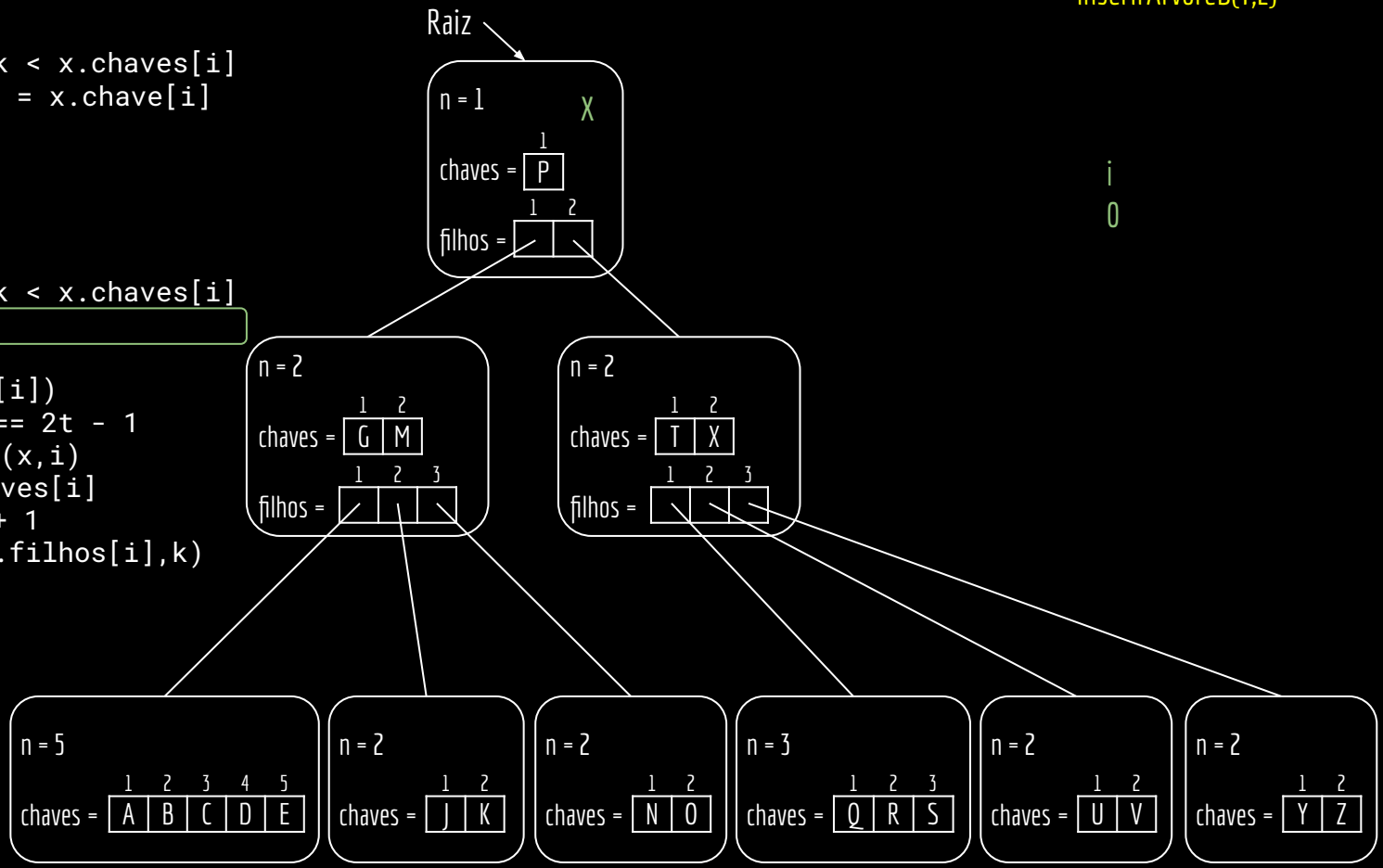
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

0



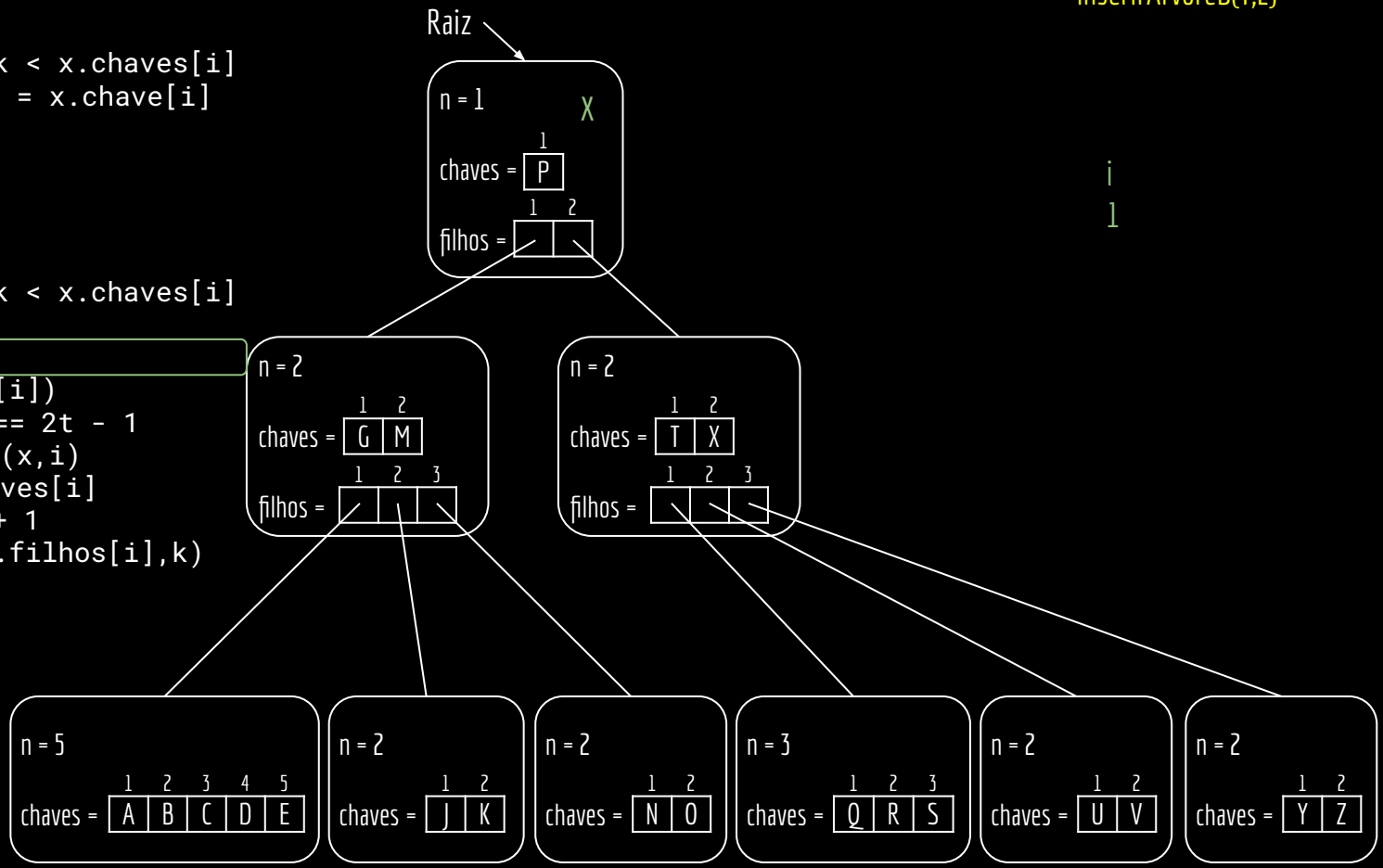
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

1



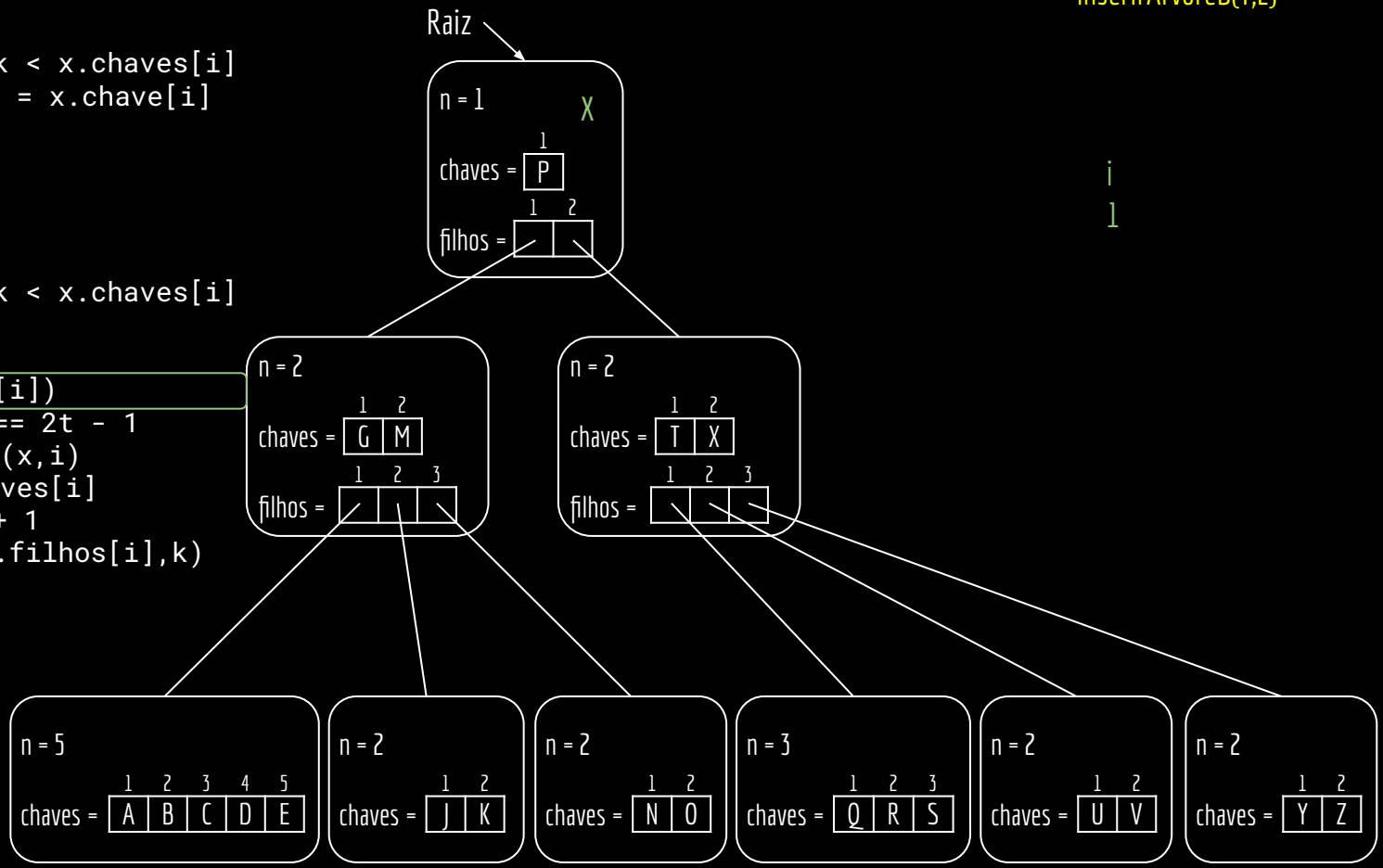
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

1



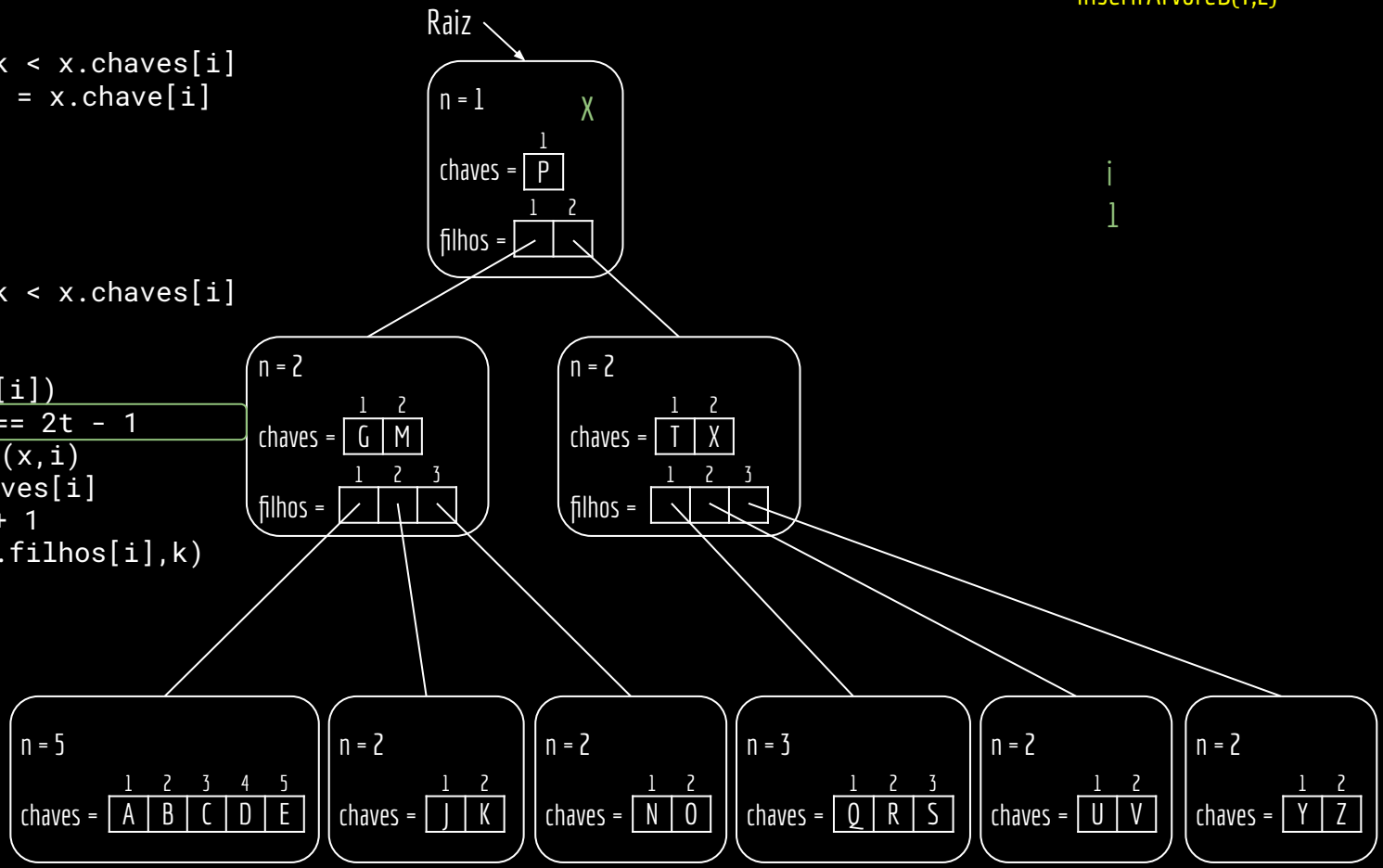
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

1



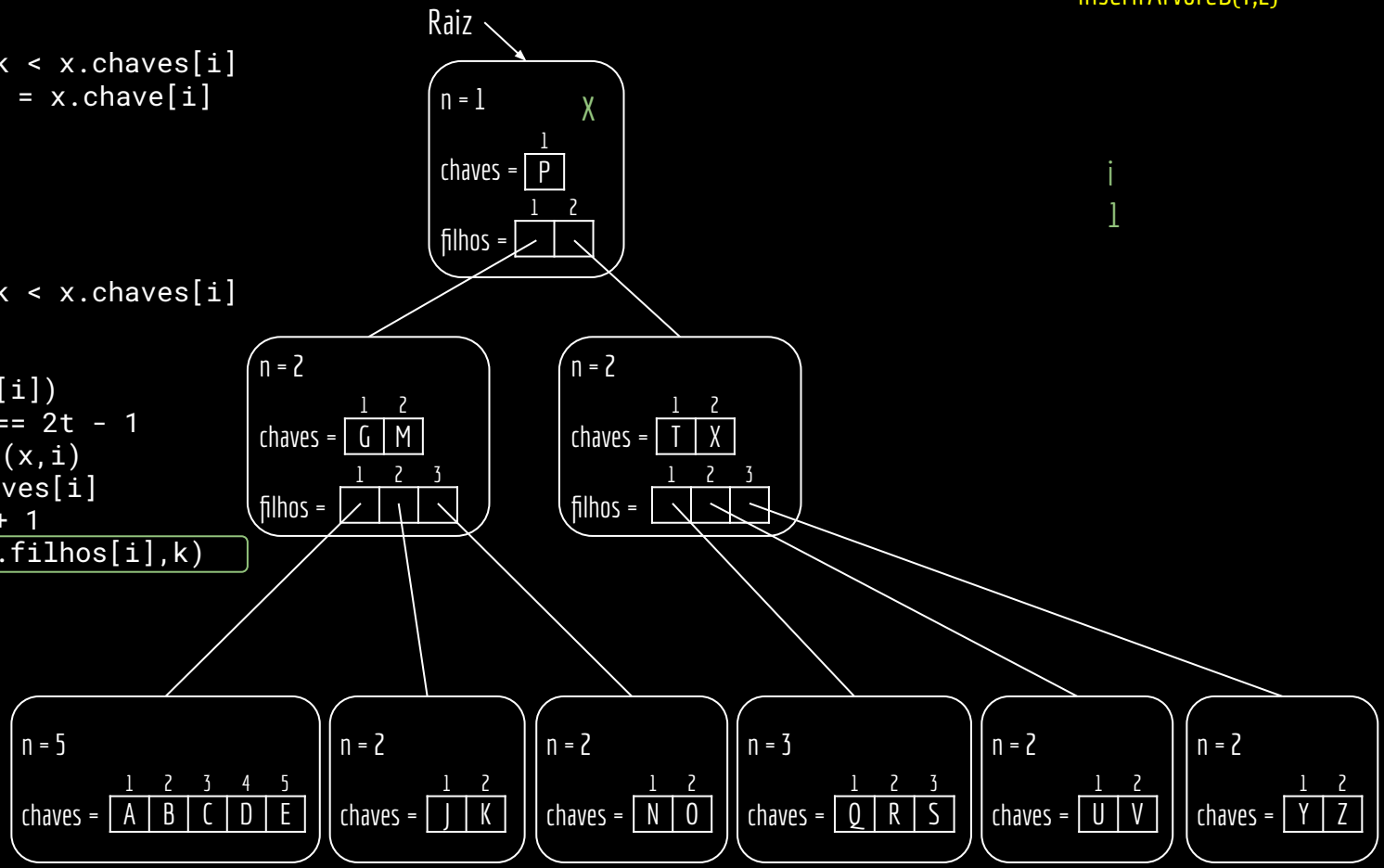
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

1



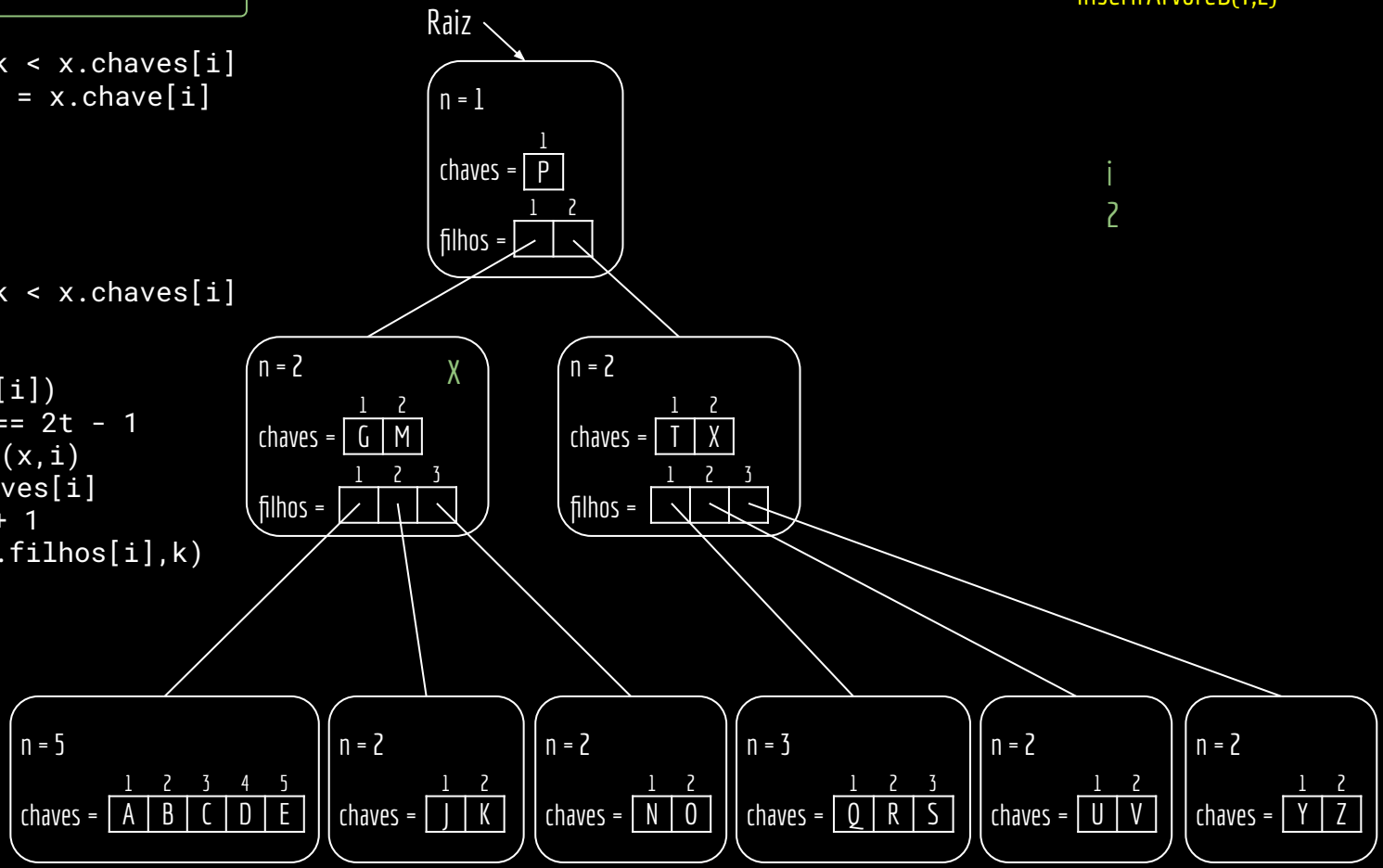
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

2



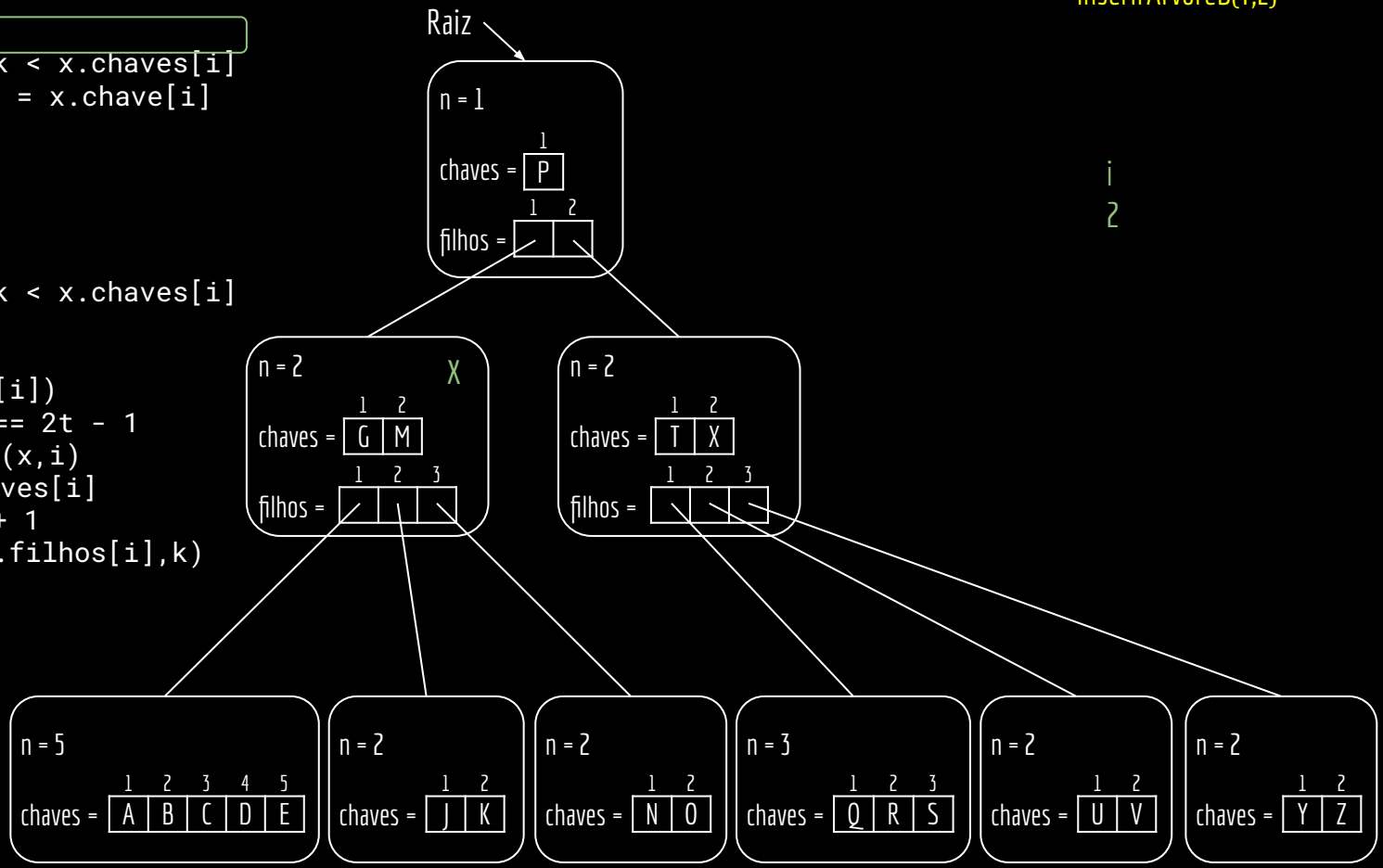
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

2



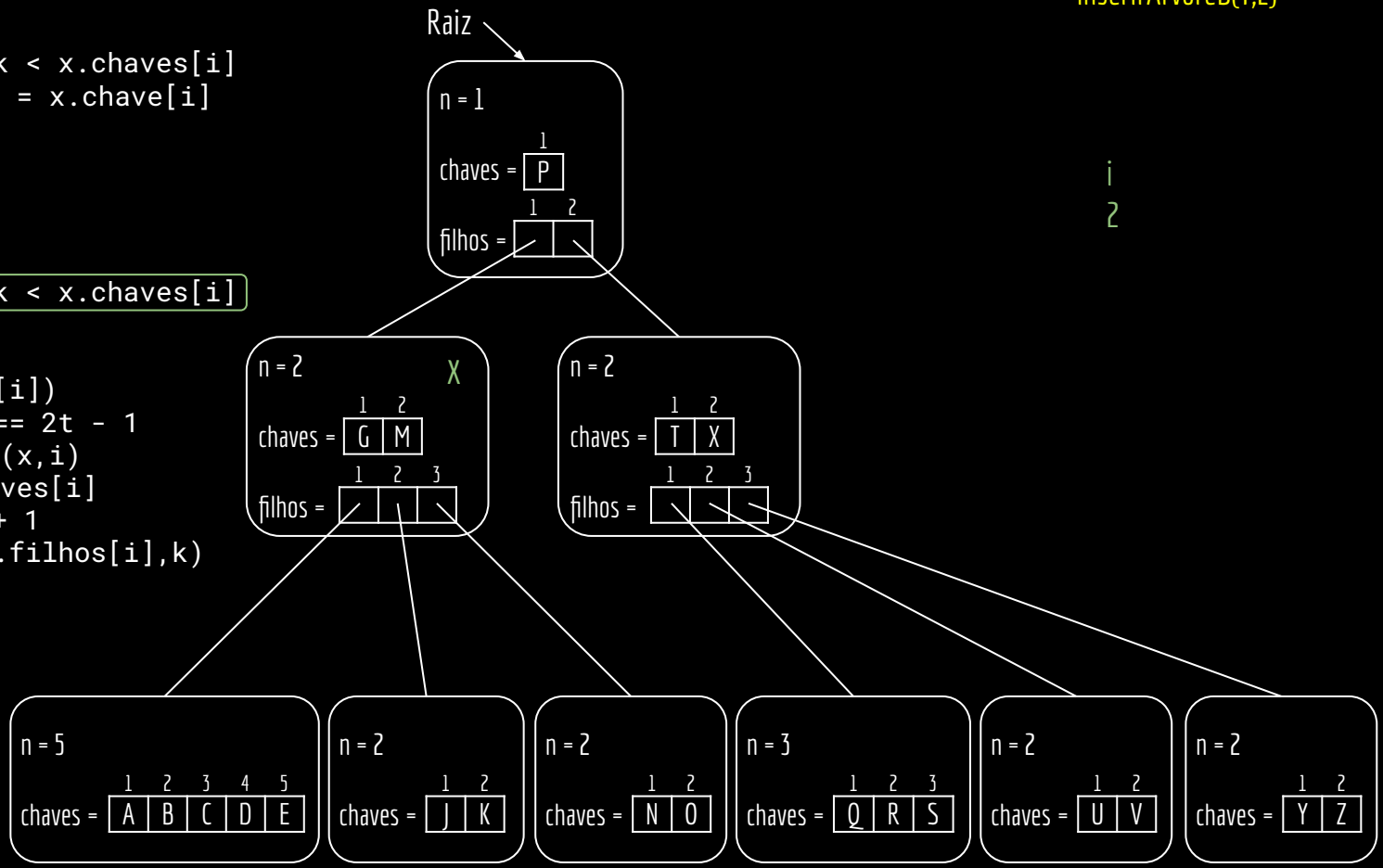
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

2



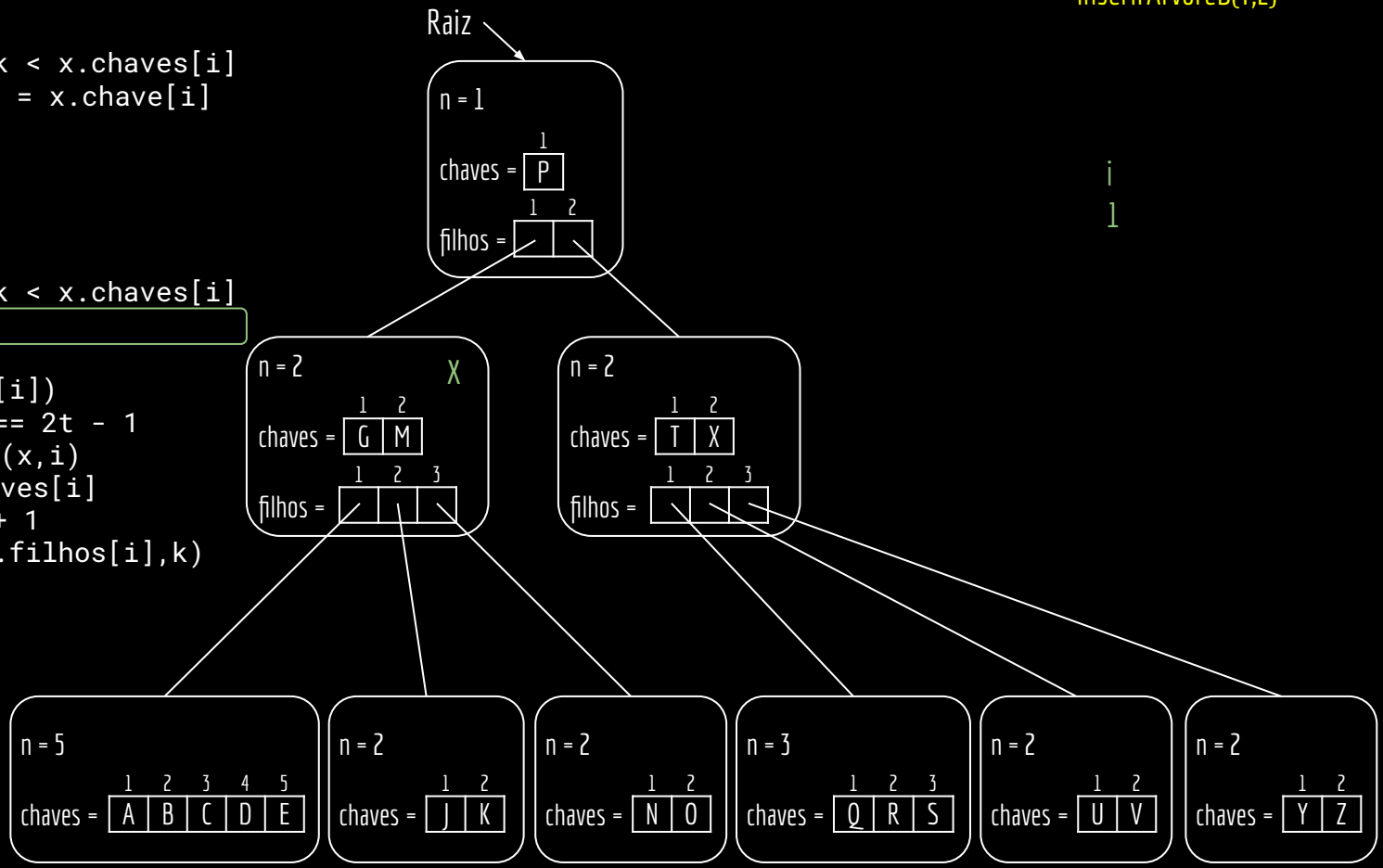
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

1



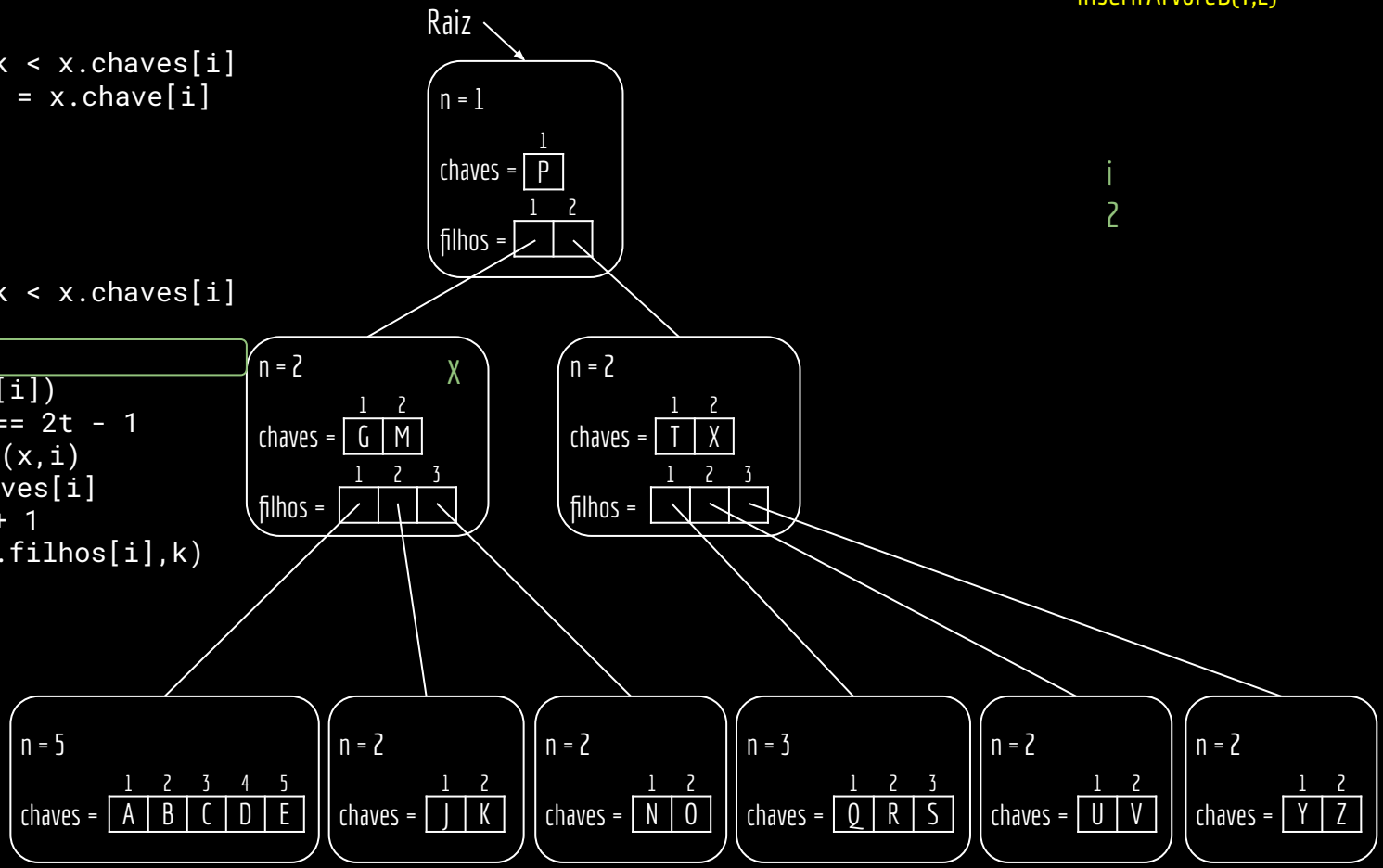
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

2



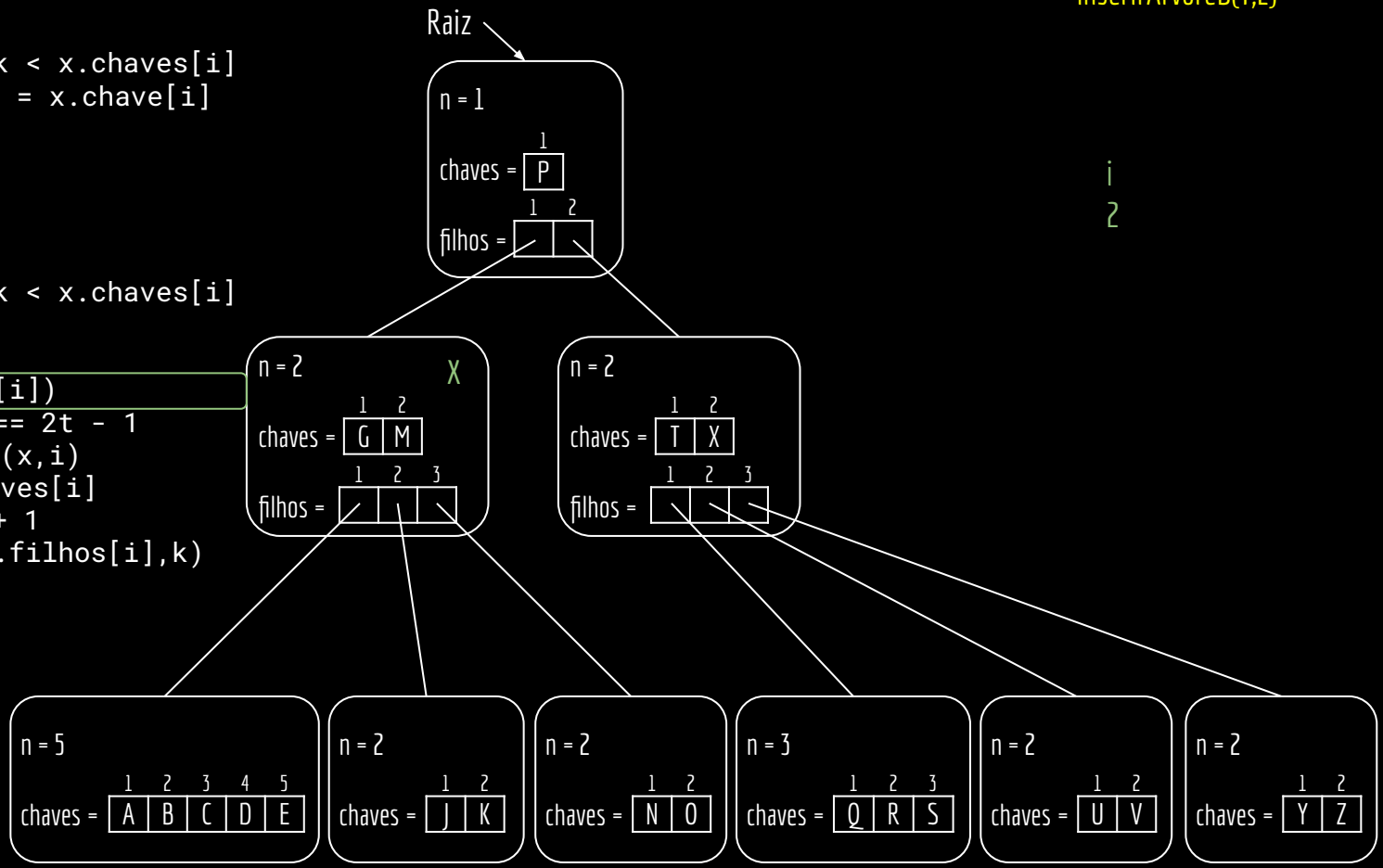
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

2



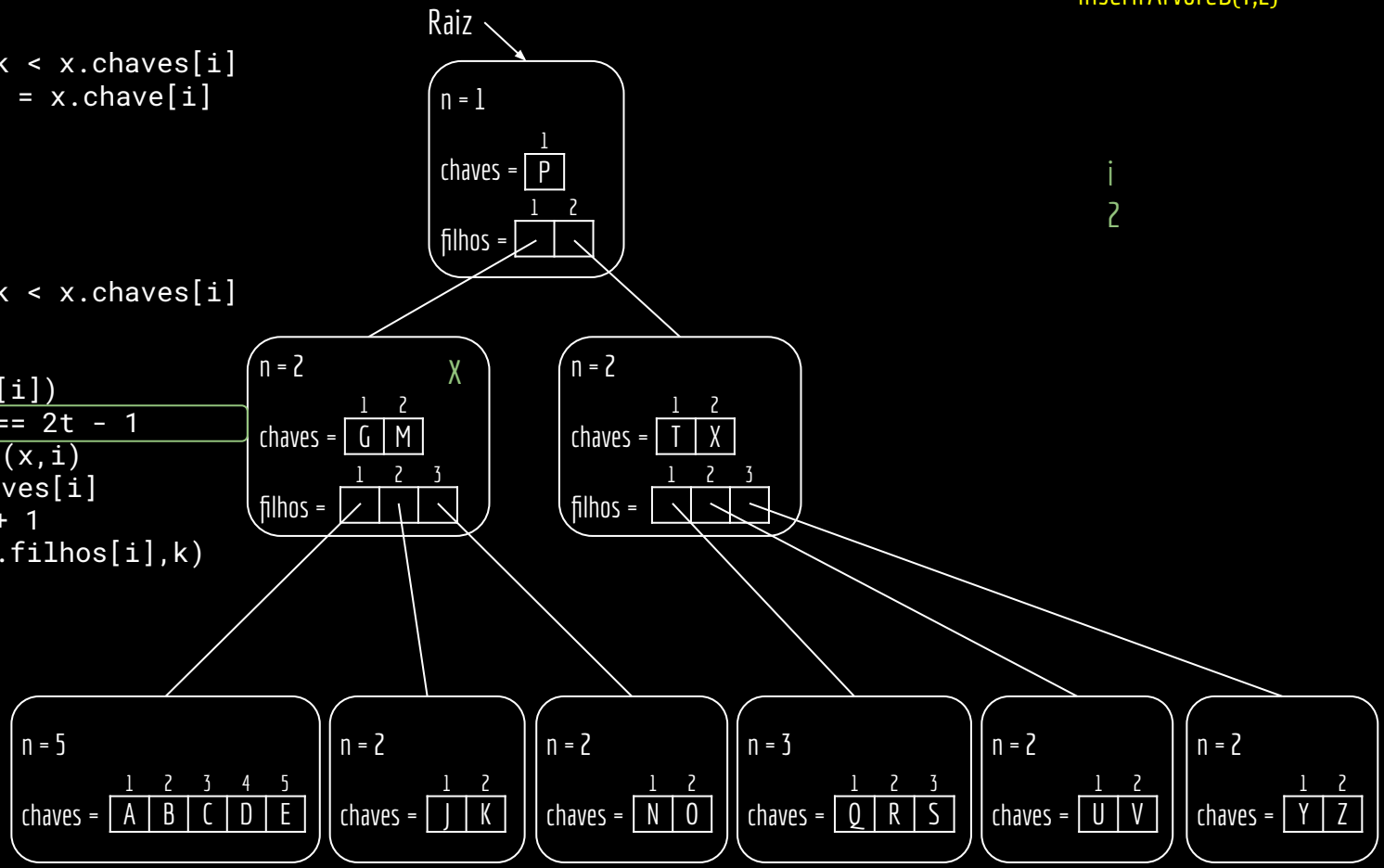
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

2



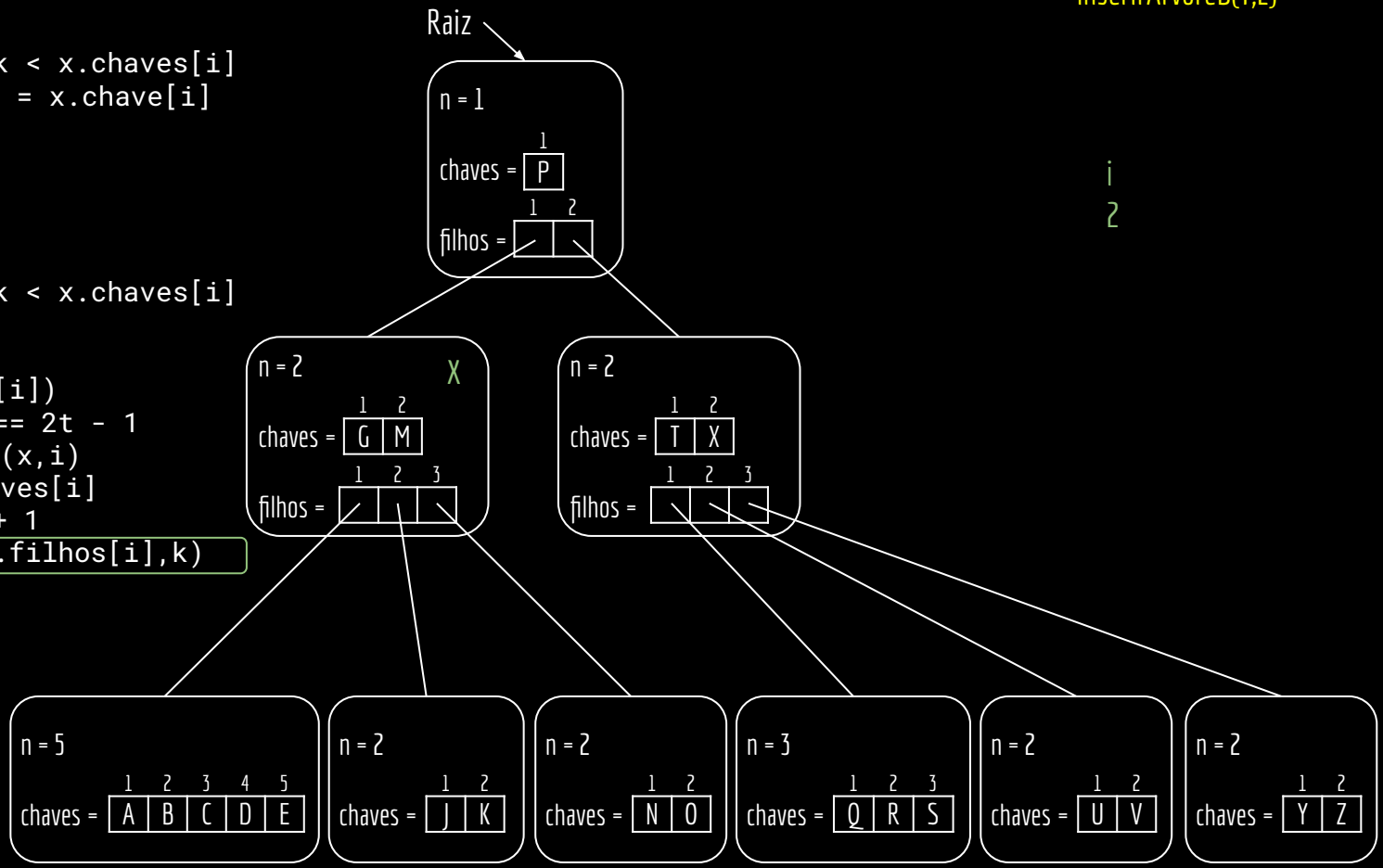
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

2



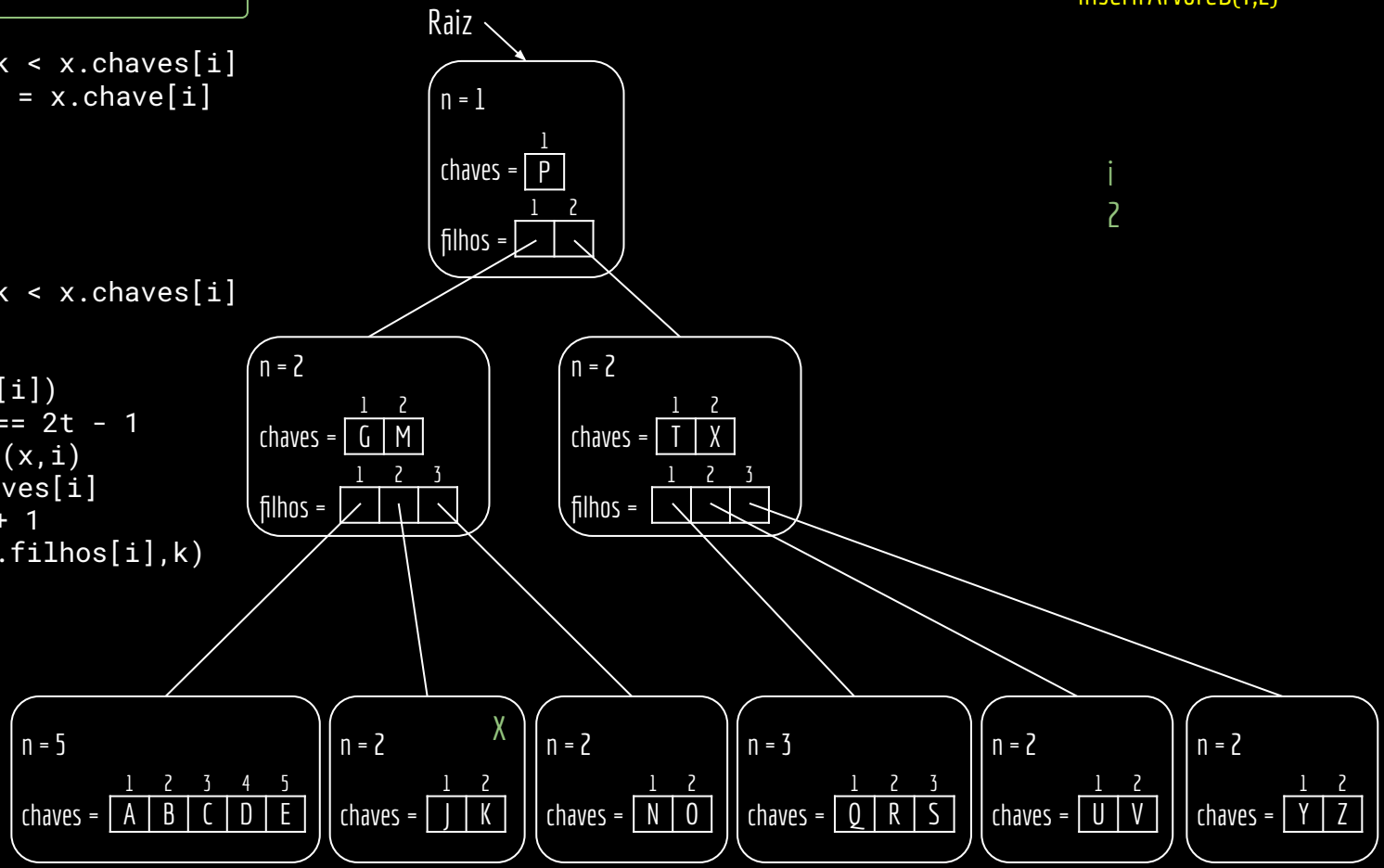
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

2



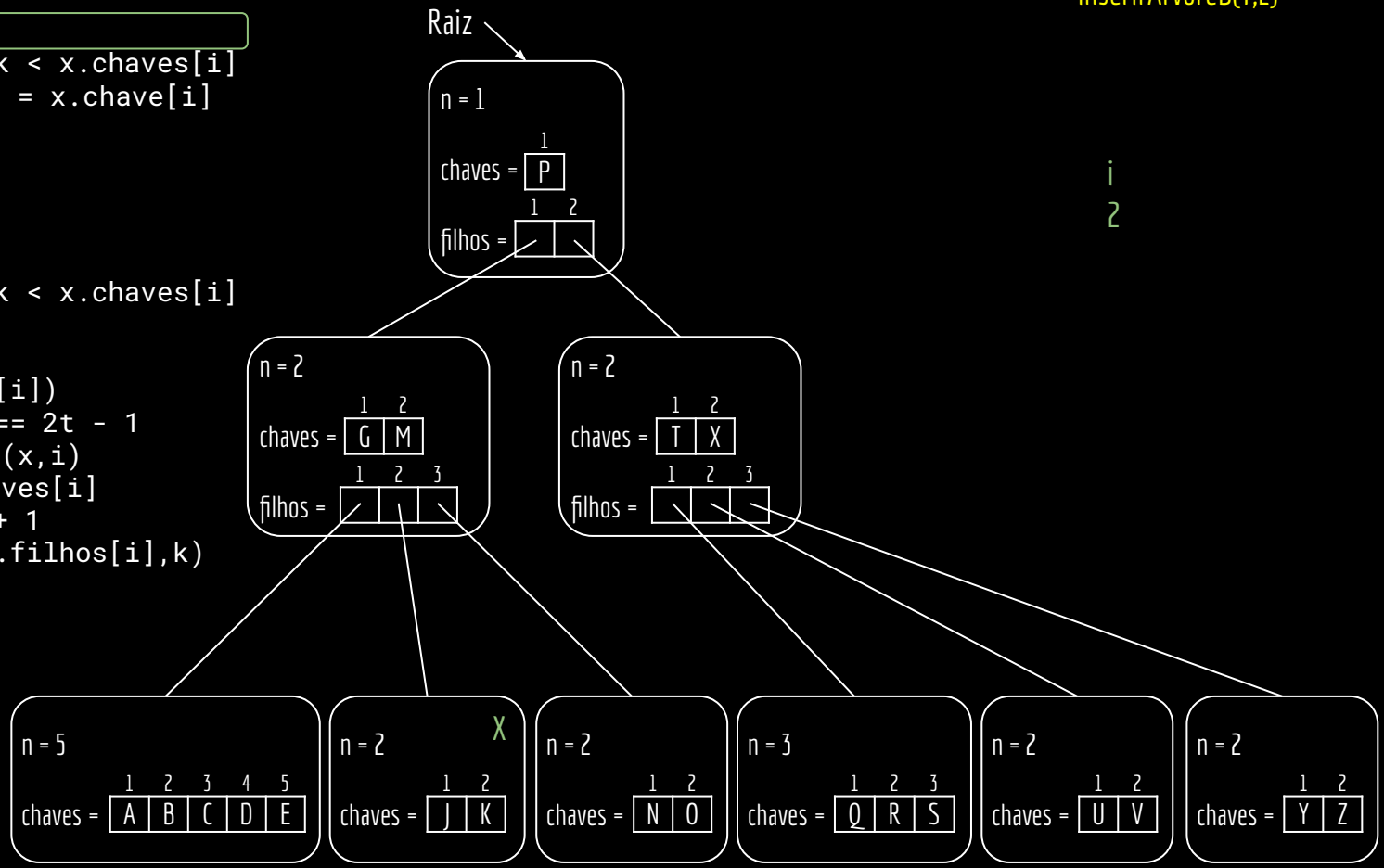
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

2



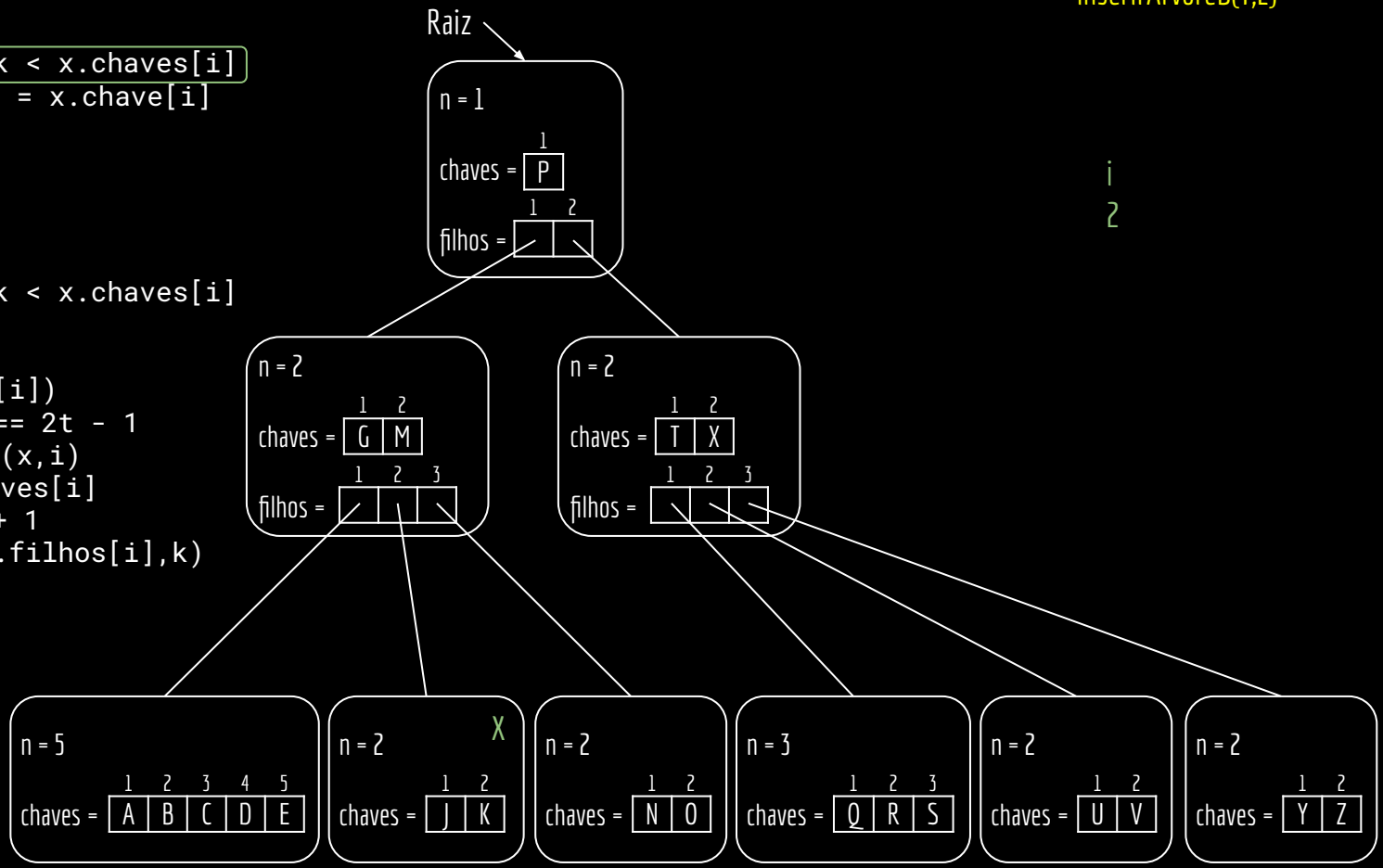
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

2



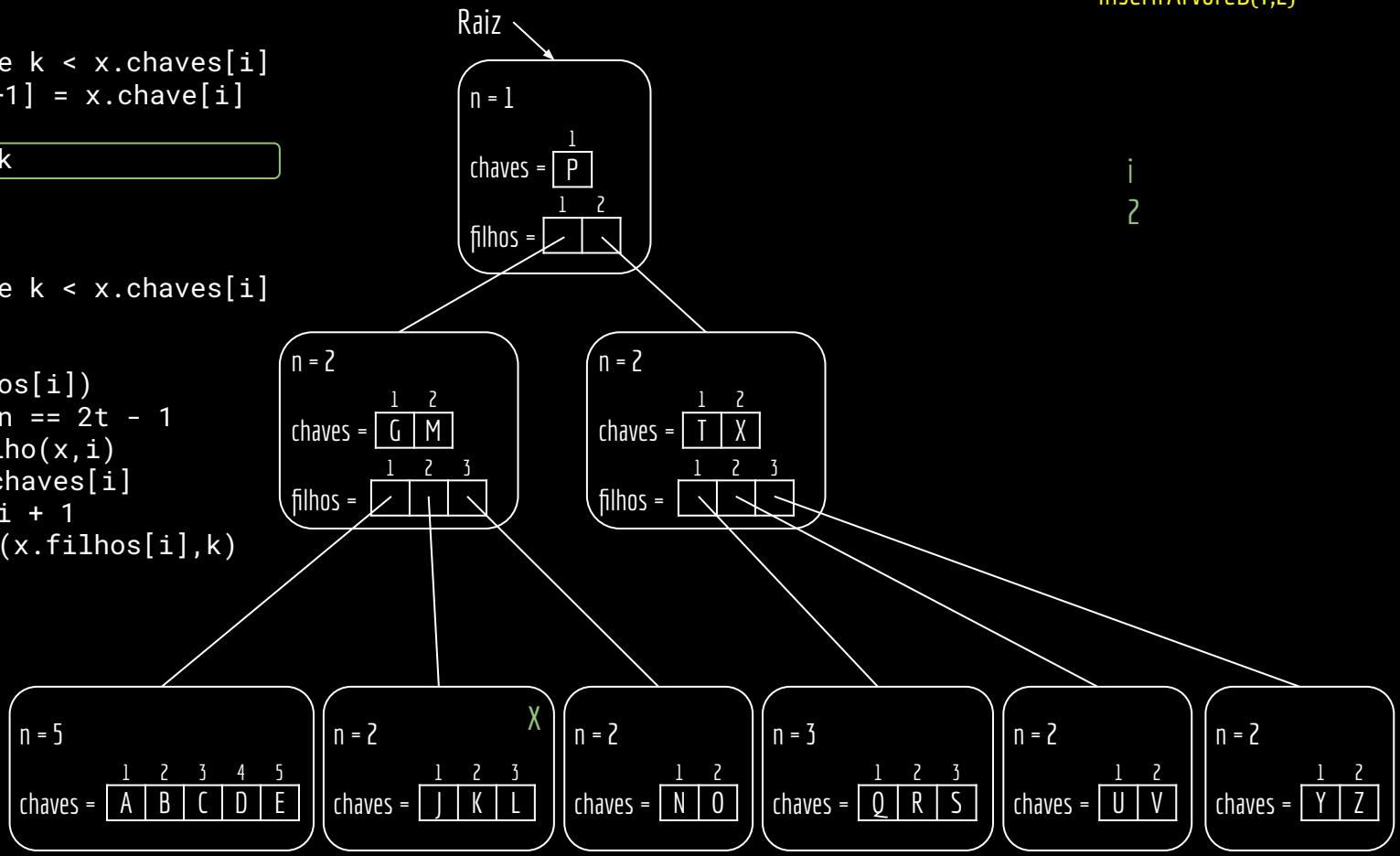
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

2



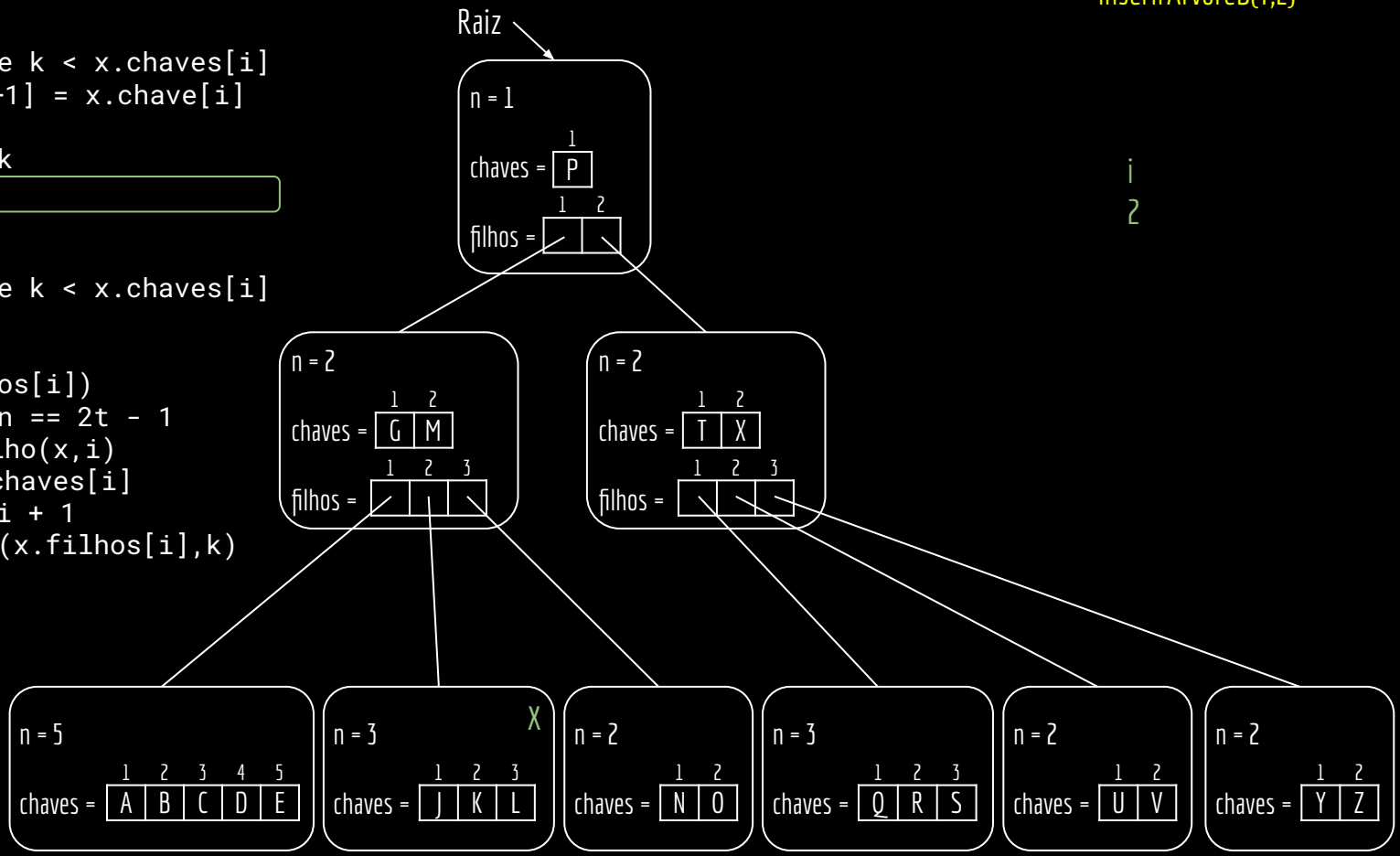
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

i

2



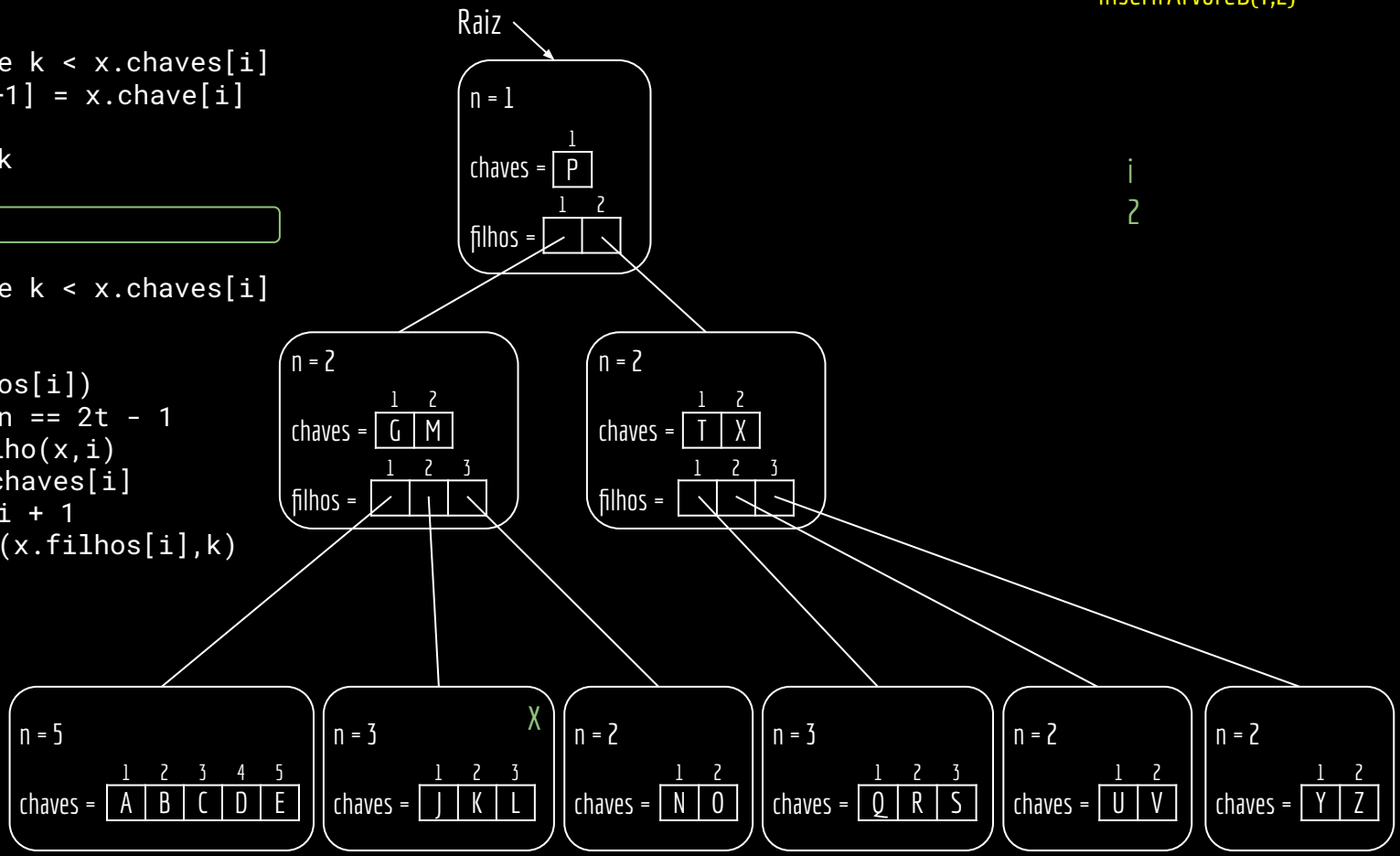
```
função inserirNaoCheio(x,k)
i = x.n
se x é folha
      enquanto i ≥ 1 e k < x.chaves[i]
            x.chave[i+1] = x.chave[i]
            i = i - 1
      x.chave[i+1] = k
      x.n = x.n + 1
      armazenar(x)
senão
      enquanto i ≥ 1 e k < x.chaves[i]
            i = i - 1
      i = i + 1
      carregar(x.filhos[i])
      se x.filhos[i].n == 2t - 1
            dividirFilho(x,i)
            se k > x.chaves[i]
                  i = i + 1
      inserirNaoCheio(x.filhos[i],k)
```

t=3

inserirArvoreB(T,L)

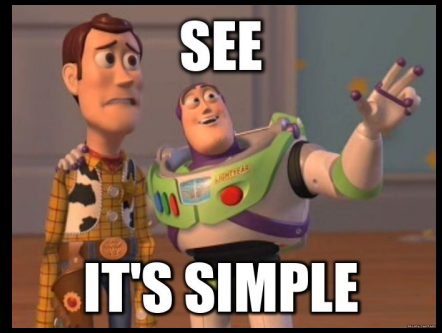
i

2



```
função inserirArvoreB(T,k)

r = T.raiz
se r.n == 2t-1
     s = dividirRaiz(T)
     inserirNaoCheio(s,k)
senão
     inserirNaoCheio(r,k)
```

t=3
inserirArvoreB(T,L)



Faça você mesmo

Considerando o estado da árvore do slide anterior, faça o teste de mesa para `inserirArvoreB(T,F)`.

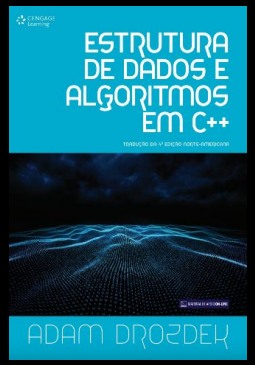
Uso do espaço

Por definição, uma Árvore B tem pelo menos de 50% de suas chaves ocupadas.

Logo, cerca de 50% do espaço para chaves pode ser desperdiçado.

Simulações com inserções e exclusões aleatórias mostram que, em média, a árvore se mantém 69% cheia.

Veja uma discussão em Drozdek (2016).

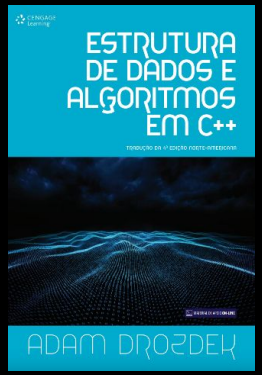


Exercícios

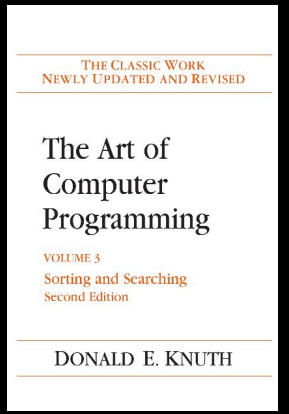
1. Implemente os algoritmos em C. Assuma que todos os nodos estão na memória principal.

Referências

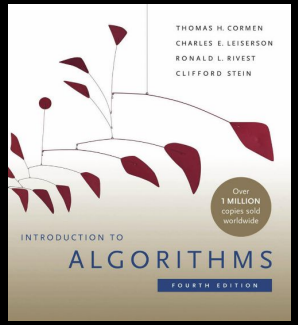
Estrutura de Dados e Algoritmos em C++. A. Drozdek. 4a ed. 2016.



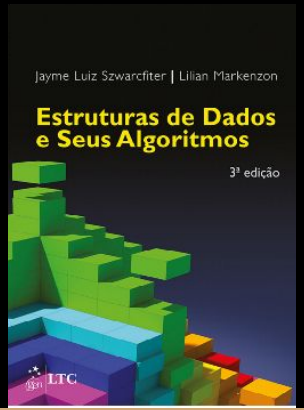
Knuth, D. The Art of Computer Programming: Volume 3: Sorting and Searching. 1998.



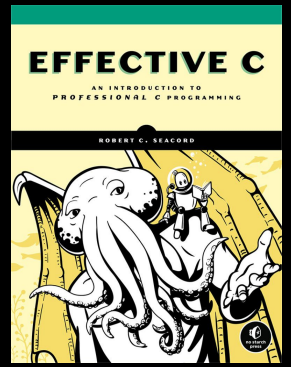
T. Cormen, C. Leiserson, R. Rivest, C. Stein. Algoritmos: Teoria e Prática. 4a ed. 2022.



Szwarcfiter, Markenzon, L. Estruturas de dados e seus algoritmos.2010.



Seacord, R. C. Effective C: An ntroduction to Professional C Programming. 2020.



Licença