

Árvore 2-3

Luigi Wagner
Rafael Alessandro
Rafael Falcão

Ciências da Computação – Disciplina: Estruturas de Dados – 1
Instituto de Informática
Universidade Federal de Goiás

21 de novembro de 2017

Sumário

- 1 Introdução
- 2 Pesquisa
- 3 Inserção
- 4 Remoção
- 5 Códigos
- 6 Aplicações
- 7 Questionário
- 8 Referências Bibliográficas

Introdução

Uma Árvore 2-3 é uma árvore onde cada nó com filho (nó interno) tem também 2 filhos (2-node) e 1 elemento de dados (chave) ou 3 filhos (3-nodes) e 2 elementos de dados (chaves). Os nós externos a árvore (nós-folha) não tem filhos e possuem um ou dois elementos de dados (chaves).

Introdução

Propriedades

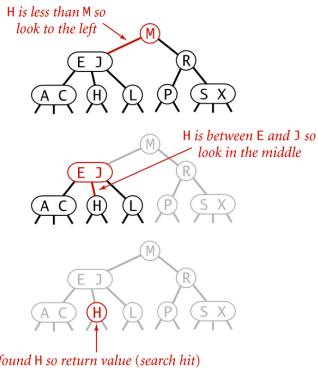
As principais propriedades de uma Árvore 2-3 são:

- Cada nó interno tem dois filhos (2-node) se tem uma chave, ou três filhos (3-node) se tem duas chaves;
- Cada nó não-folha tem 2 ou 3 filhos. Se tem 2 filhos tem 1 item de dados e se tem 3 filhos tem 2 itens de dados;
- Todos os dados são ordenados;
- Todas as folhas estão no mesmo nível- assim como na Árvore B;
- Cada nó folha tem 1 ou 2 campos.

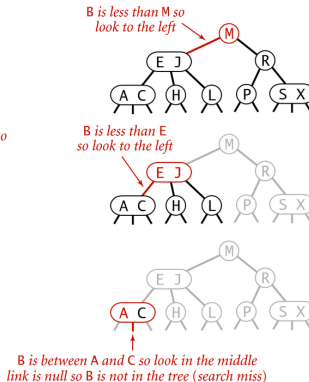
Como, por exemplo:

Introdução

successful search for H



unsuccessful search for B

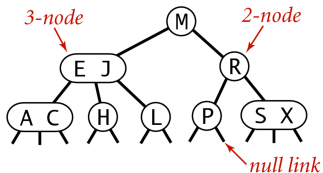


Search hit (left) and search miss (right) in a 2-3 tree

Figura: <https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/figuressw/Chapter3/TTsearch.png>

Introdução

Anatomia



Anatomy of a 2-3 search tree

Figura: <https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/figuressw/Chapter3/TTanatomy.png>

Sumário

- 1 Introdução
- 2 Pesquisa
- 3 Inserção
- 4 Remoção
- 5 Códigos
- 6 Aplicações
- 7 Questionário
- 8 Referências Bibliográficas

Pesquisa em uma Árvore 2-3

Vamos começar pelo mais básico procedimento a ser realizado ao lidar com uma [Árvore](#).

Pesquisa em uma Árvore 2-3

Exemplo de uma Árvore 2-3

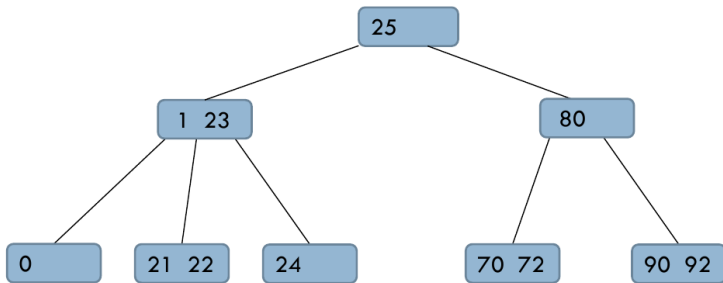


Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Pesquisa em uma Árvore 2-3

Pesquisando pela chave 72 na Árvore:

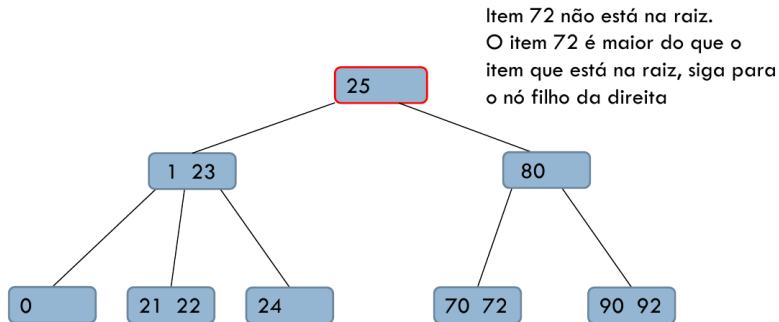


Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Pesquisa em uma Árvore 2-3

Pesquisando pela chave 72 na Árvore:

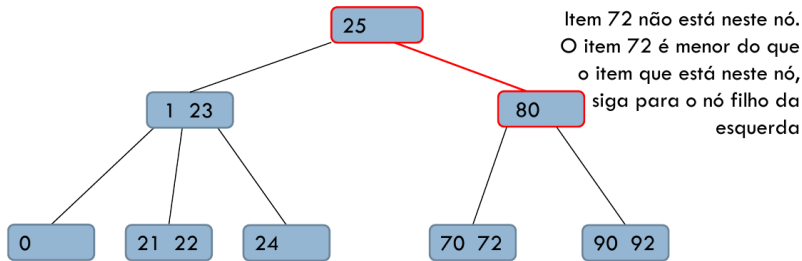


Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Pesquisa em uma Árvore 2-3

Pesquisando pela chave 72 na Árvore:

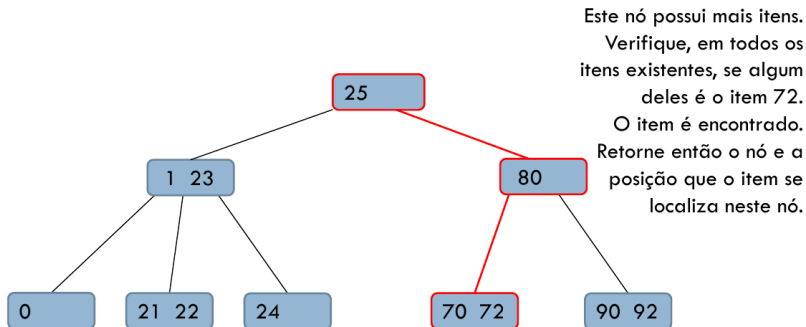


Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Sumário

- 1 Introdução
- 2 Pesquisa
- 3 Inserção**
- 4 Remoção
- 5 Códigos
- 6 Aplicações
- 7 Questionário
- 8 Referências Bibliográficas

Inserção em uma Árvore 2-3

Continuando os procedimentos, vamos à *Inserção* em uma *Árvore 2-3*.

Inserção em uma Árvore 2-3

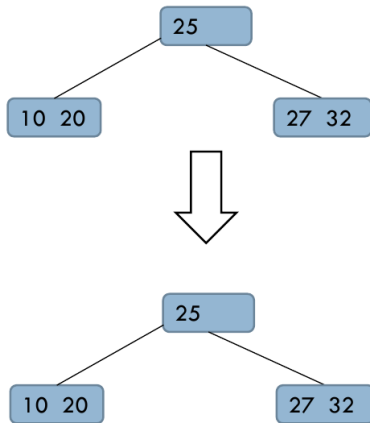
Inserção

No âmbito das **Árvores 2-3**, daremos uma olhada na inserção.

- Assemelha-se a inserção em uma **Árvore 2-3** à inserção em uma Árvore binária de busca.
- Estrutura básica (**Struct**);
- Alocação da **Árvore 2-3**.

Inserção em uma Árvore 2-3

Inserindo a chave 50 na Árvore:



Página folha está cheia.
Solução: Dividir a
página e subir o
elemento do meio para
a página pai

Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Inserção em uma Árvore 2-3

Inserindo a chave 50 na Árvore:

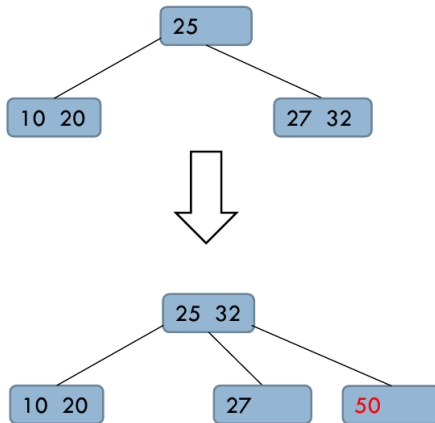


Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Inserção em uma Árvore 2-3

Inserindo a chave 5 na Árvore:

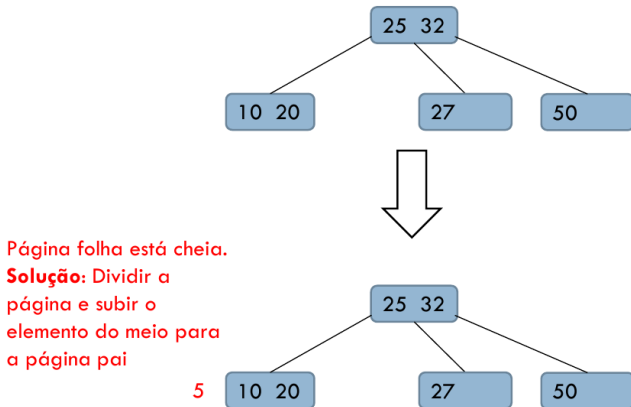


Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Inserção em uma Árvore 2-3

Inserindo a chave 5 na Árvore:

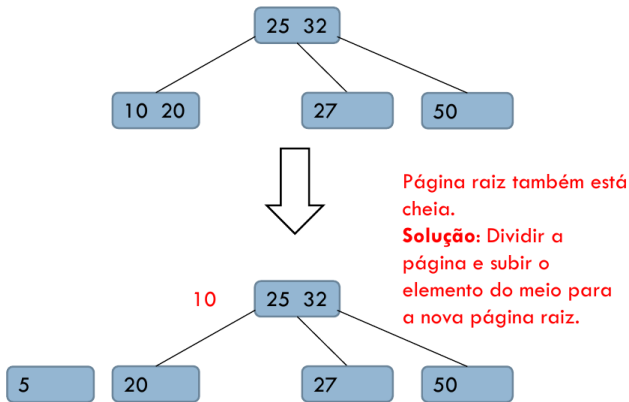


Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Inserção em uma Árvore 2-3

Inserindo a chave 5 na Árvore:

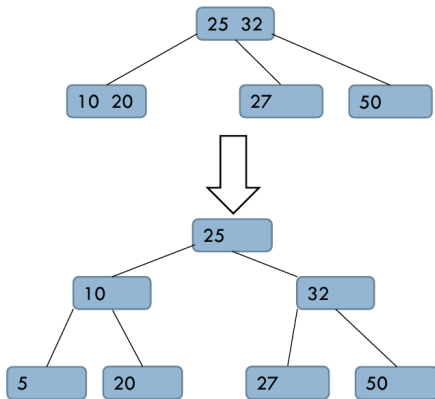


Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Sumário

- 1 Introdução
- 2 Pesquisa
- 3 Inserção
- 4 Remoção**
- 5 Códigos
- 6 Aplicações
- 7 Questionário
- 8 Referências Bibliográficas

Remoção em uma Árvore 2-3

Hora de tratarmos do mais complexo dos procedimentos relacionados à uma *Árvore 2-3*, a Remoção.

Remoção em uma Árvore 2-3

Começaremos pelo mais simples:
Exemplo 1:

□ Remover item 73.

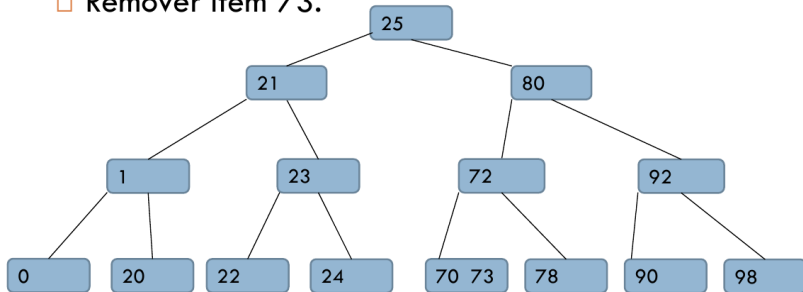


Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Remoção em uma Árvore 2-3

□ Remover item 73.

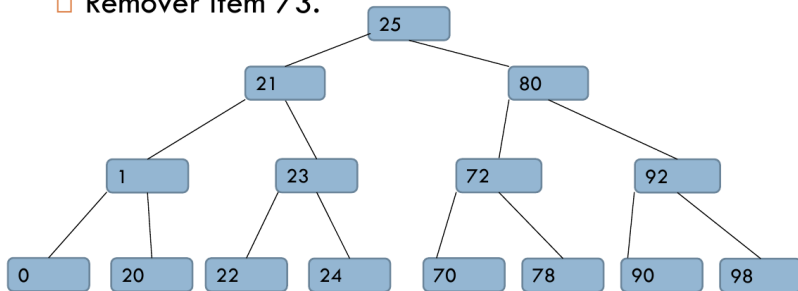


Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Remoção em uma Árvore 2-3

Exemplo 2:

□ Remover item 20.

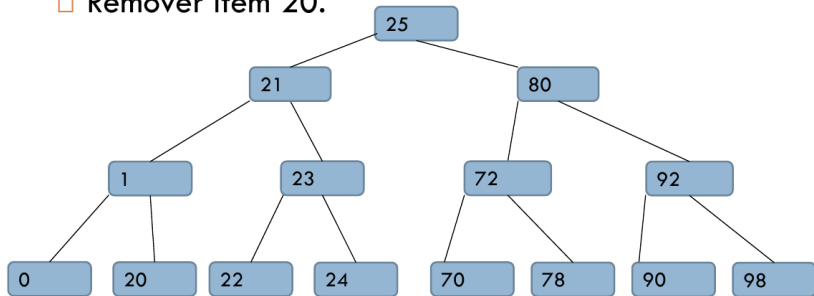


Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Remoção em uma Árvore 2-3

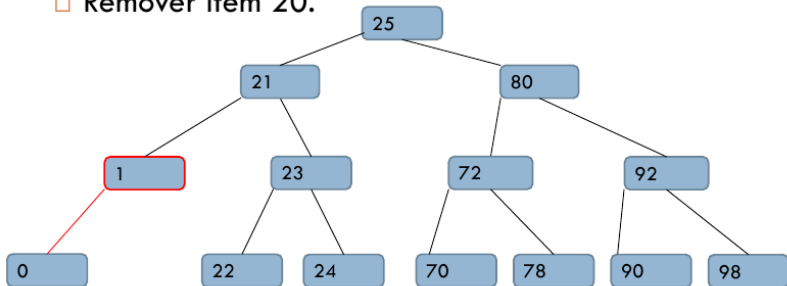
Problema!

Houston, we have a problem.

Repare que não podemos remover diretamente o 20, temos que reorganizar a Árvore para manter suas propriedades válidas. No caso, a propriedade referida é a de um nó ter 2 ou três filhos, não é permitido apenas um filho.

Remoção em uma Árvore 2-3

□ Remover item 20.



Erro!

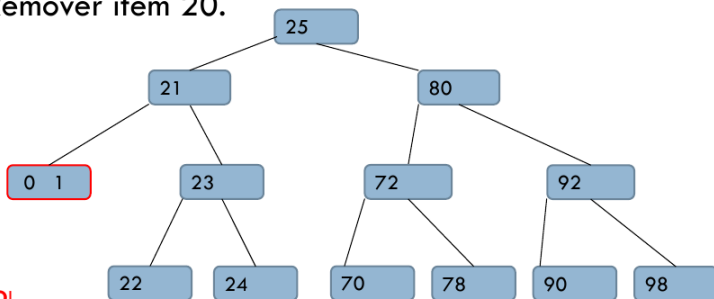
Nó não pode ter apenas um filho.

Solução: junte o item do único filho com o item do nó pai

Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Remoção em uma Árvore 2-3

□ Remover item 20.



ERRO!

Assim como na árvore B, os nós folhas devem estar todos no mesmo nível.

Solução: agrupar os nós folhas, para que todas as folhas fiquem no mesmo nível.

Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Remoção em uma Árvore 2-3

□ Remover item 20.

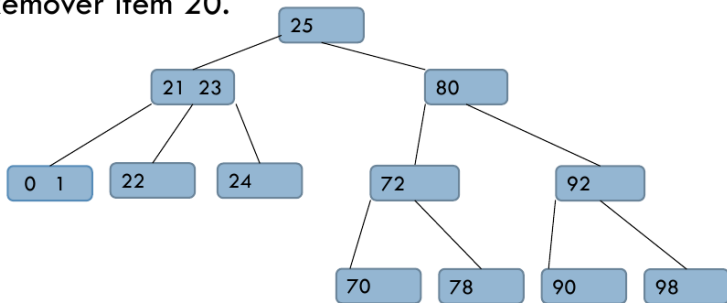
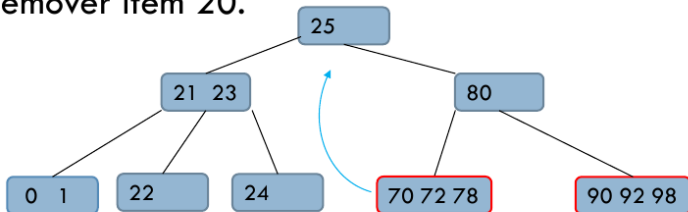


Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Remoção em uma Árvore 2-3

□ Remover item 20.



ERRO!

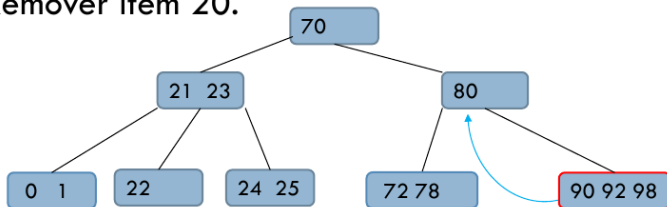
Os nós podem ter no máximo 2 itens de dados

Solução: levar o menor elemento da subárvore da direita para a raiz, e descer o item da raiz para a subárvore da esquerda

Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Remoção em uma Árvore 2-3

□ Remover item 20.



ERRO!

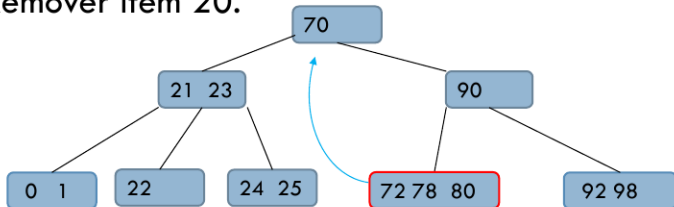
Os nós podem ter no máximo 2 itens de dados.

Solução: levar o menor elemento da subárvore da direita para a raiz, e descer o item da raiz para a subárvore da esquerda.

Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Remoção em uma Árvore 2-3

□ Remover item 20.



ERRO!

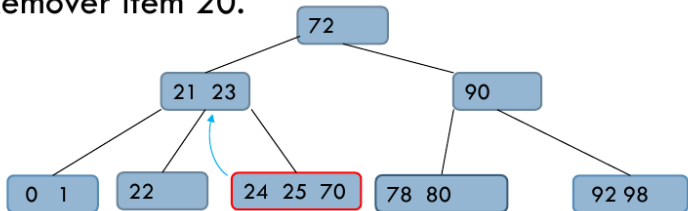
Os nós podem ter no máximo 2 itens de dados.

Solução: levar o menor elemento da subárvore da direita para a raiz, e descer o item da raiz para a subárvore da esquerda.

Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Remoção em uma Árvore 2-3

□ Remover item 20.



ERRO!

Os nós podem ter no máximo 2 itens de dados.

Solução: levar o menor elemento da subárvore da direita para a raiz, e descer o item da raiz para a subárvore da esquerda.

Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Remoção em uma Árvore 2-3

□ Remover item 20.

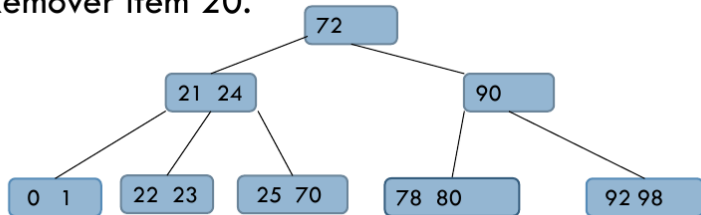
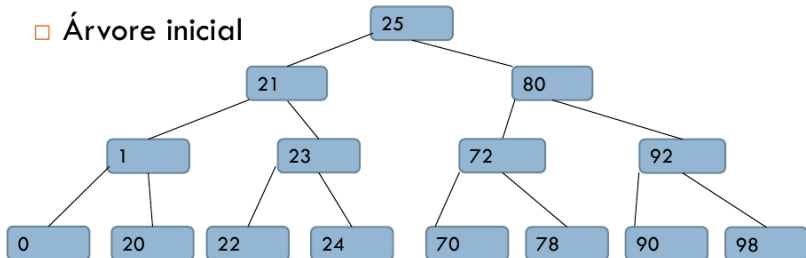


Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Remoção em uma Árvore 2-3

□ Árvore inicial



□ Árvore após remoção do item 20

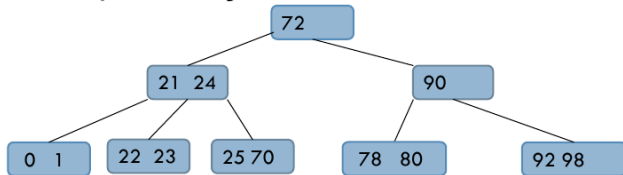


Figura: Imagem retirada de [2] (vide *Referências Bibliográficas*)

Remoção em uma Árvore 2-3

Entenderam o porquê do *complexo* agora, ou seja, tenham plena certeza de suas inserções ou sofrerão com sua remoção!

Sumário

- 1 Introdução
- 2 Pesquisa
- 3 Inserção
- 4 Remoção
- 5 Códigos**
- 6 Aplicações
- 7 Questionário
- 8 Referências Bibliográficas

Programas

Agora que já vimos como funciona e os princípios básicos das [Árvores 2-3](#), daremos uma olhada nas estruturas e algoritmos para a manipulação e armazenamento das supracitadas [Árvores 2-3](#).

- 1 Estrutura básica ([Struct](#));
- 2 Alocação da [Árvore 2-3](#).

Código

Estrutura de uma Árvore 2-3

```
1  typedef struct _no23 {  
2      int lkey, // chave esquerda  
3      rkey, // chave direita  
4      nkeys; // numero de chaves  
5      struct _no23 *left, // ponteiro ao filho esquerdo  
6      *center, // ponteiro ao filho central  
7      *right; // ponteiro ao filho direito  
8  } no23;
```

Código

Alocação de uma Árvore 2-3

```
1      no23 *23tree_alloc() {  
2          no23 *t;  
3          t = malloc(sizeof(no23));  
4          t->nkeys = 0;  
5          t->lkey = t->rkey = 0;  
6          t->left = t->center = t->right = NULL;  
7          return t;  
8      }
```


Código

Busca em uma Árvore 2-3

```
1      no23 *find(no23 *raiz, int key) {  
2          if(raiz==NULL)  
3              return NULL; // nao encontrou  
4          if(key == raiz->lkey)  
5              return raiz; // retorna chave esquerda  
6          if((raiz->nkeys == 2) && (key == raiz->rkey))  
7              return raiz; // retorna a chave direita  
8          if(key < raiz->lkey)  
9              return find(raiz->left, key);  
10         else if(raiz->nkeys == 1)  
11             return find(raiz->center, key);  
12         else if (key < raiz->rkey)  
13             return find(raiz->center, key);  
14         else  
15             return find(raiz->right, key);  
16     }
```

Código

Inserção em uma Árvore 2-3

```

1      no23 *insere( no23 **no, int val, int *rval){
2          no23 *paux, *paux2;
3          int vaux, promov;
4          if (*no == NULL) { // arvore vazia
5              *no = (no23 *) malloc (sizeof(no23));
6              *no = criaNo(val, 0, 0, NULL, NULL, NULL); //cria no folha com valor
7              return NULL; // nada a fazer depois
8          }
9          if (isLeaf(*no)){ // chegou a folha
10             if ((*no)->nkeys == 1){ // caso facil
11                 adicionaChave(*no, val, NULL);
12                 return NULL;
13             }else{
14                 paux = quebraNo(*no, val, &vaux, NULL);
15                 *rval = vaux;
16                 return paux;
17             }
18         }

```

Código

```

1
2     else{// continua a procura
3         if (val < (*no)->lkey)
4             paux = insere( &((*no)->left), val, &vaux);
5         else if (((*no)->nkeys == 1) || (val < (*no)->rkey))
6             paux = insere( &((*no)->center), val, &vaux);
7         else
8             paux = insere( &((*no)->right), val, &vaux);
9             if(paux == NULL) // nao promoveu
10                 return NULL;
11             else if ((*no)->nkeys == 1){
12                 adicionaChave(*no, vaux, paux);
13                 return NULL;
14             }else{
15                 paux2 = quebraNo(*no, vaux, &promov, paux);
16                 *rval = promov;
17                 return paux2;
18             }
19     }
20 }
```

Quebra Nó

```

1  no23 *quebraNo(no23 *no, int val, int *rval, no23 *subarvore){
2      no23 *paux;
3      if (val > no->rkey) { // val esta mais a direita
4          *rval = no->rkey; // promove a antiga maior
5          paux = no->right;
6          no->right = NULL; // elimina o terceiro filho
7          no->nkeys = 1; // atualiza o numero de chaves
8          return criaNo(val, 0, 1, paux, subarvore, NULL);
9      } else if (val >= no->lkey) { // val esta no meio
10         *rval = val; // continua sendo promovido
11         paux = no->right;
12         no->right = NULL;
13         no->nkeys = 1;
14         return criaNo(no->rkey, 0, 1, subarvore, paux, NULL);
15     } else { // val esta a mais a esquerda
16         *rval = no->lkey; // primeiro cria o noh a direita
17         paux = criaNo(no->rkey, 0, 1, no->center, no->right, NULL);
18         no->lkey = val; // em seguida arruma o noh a esquerda
19         no->nkeys = 1;
20         no->right = NULL;
21         no->center = subarvore;
22         return paux;
23     }
24 }

```

Sumário

- 1 Introdução
- 2 Pesquisa
- 3 Inserção
- 4 Remoção
- 5 Códigos
- 6 Aplicações**
- 7 Questionário
- 8 Referências Bibliográficas

Aplicações

Objetivo

- Árvores 2-3 foram inventadas para garantir um bom desempenho no **pior** caso.
- Se você estiver satisfeito com um bom desempenho médio, basta usar um Árvore Binária de Busca

Altura

Considere uma árvore 2-3 com N nós.

- Se temos apenas nós simples, a altura da árvore será $\lg N$
- Se temos apenas nós duplos, a altura da árvore será $\log_3 n$ que é igual a $[0.63 \lg N]$

Conclusão: A altura nunca passa de $\lg N$.

Aplicações

Proposição

Em uma árvore 2-3 com N nós, **busca** e **inserção** nunca visitam mais que $\lg N$ nós - mesmo no pior caso.

- Cada visita faz no máximo 2 comparações de chaves.

Aplicações

Observação

Numa árvore 2-3, alguns nós envolvem uma comparação entre chaves enquanto outros envolvem três comparações.

- Assim, o número de comparações de chaves é no máximo o dobro do número de nós visitados

Time complexity in big O notation

Algorithm	Average	Worst Case
Space	$O(n)$	$O(n)$
Search	$O(\log n)$	$O(\log n)$
Insert	$O(\log n)$	$O(\log n)$
Delete	$O(\log n)$	$O(\log n)$

Figura: Imagem retirada de [1] (vide *Referências Bibliográficas*)

Sumário

- 1 Introdução
- 2 Pesquisa
- 3 Inserção
- 4 Remoção
- 5 Códigos
- 6 Aplicações
- 7 Questionário**
- 8 Referências Bibliográficas

Questionário

Questão 01

Qual a altura de uma árvore 2-3 que tem 1 bilhão de chaves?

Questionário

Questão 01 – Resposta Esperada

Uma Árvore com 1 bilhão de chaves possui exatamente altura

Questionário

Questão 02

Quantas chaves, no máximo, pode conter uma árvore 2-3 de altura 2? Qual o número mínimo de chaves em uma árvore 2-3 de altura 2?

Questionário

Questão 02 – Resposta Esperada

- Máximo: Oito (8) chaves: 2 da raiz, 2 em cada um dos três filhos possíveis.
- Mínimo: Três (3) chaves, pois armazena 2 chaves por nó.

Questionário

Questão 03

Qual o número máximo de comparações de chaves em uma árvore 2-3?

Questionário

Questão 03 – Resposta Esperada

É o dobro do número de nós visitados.

Sumário

- 1 Introdução
- 2 Pesquisa
- 3 Inserção
- 4 Remoção
- 5 Códigos
- 6 Aplicações
- 7 Questionário
- 8 Referências Bibliográficas**

Referências Bibliográficas

- ❶ https://pt.wikipedia.org/wiki/%C3%81rvore_2-3
- ❷ <https://www.passeidireto.com/disciplina/algoritmos-e-estrutura-de-dados-iii?type=6&materialid=1012744>
- ❸ <https://www.ime.usp.br/~gold/cursos/2002/mac2301/aulas/b-arvore/b-arvore.html>
- ❹ COOPER, K. and TORCZON, L. – 2011 : *Chapter 2 – Scanners*;
- ❺ GRUNE, *et al.* – 2012 : *Chapter 2 – Program Text to Tokens – Lexical Analysis.*

Referências Bibliográficas

Se desejar, pode encerrar com uma imagem, uma citação, etc.



Figura: by Mark Kostabi