

Estruturas de Dados

Árvores 2–3 e B

Universidade Estadual Vale do Acaraú – UVA

Paulo Regis Menezes Sousa

paulo_regis@uvanet.br

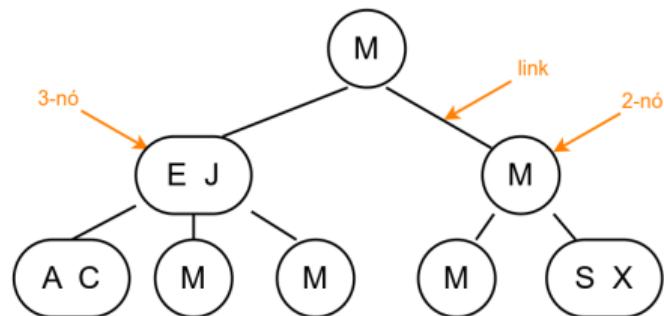
Árvores 2–3

Busca

Inserção

Árvores B

- Para manter o balanceamento de uma árvore de busca podemos permitir que a árvore guarde **mais de uma chave por nó**.
- Em uma árvore de busca **binária** temos dois links e uma chave (**2-nó**).
- Uma outra estratégia é permitir que a árvore possua também, três links e duas chaves (**3-nó**).

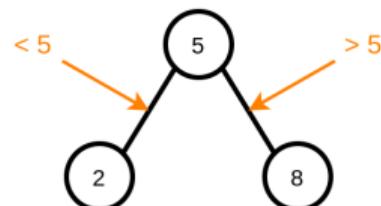


Definição

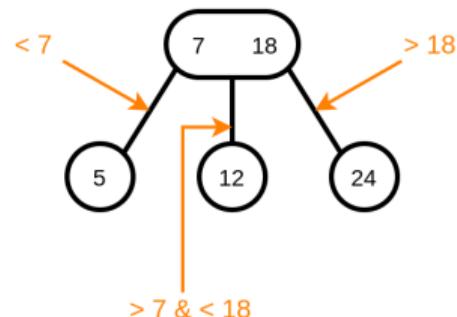
Uma Árvore de Busca 2–3 é uma árvore com nós de três tipos

- I. Nó vazio
- II. 2-nó, uma chave e dois links
- III. 3-nó, com duas chaves e três links

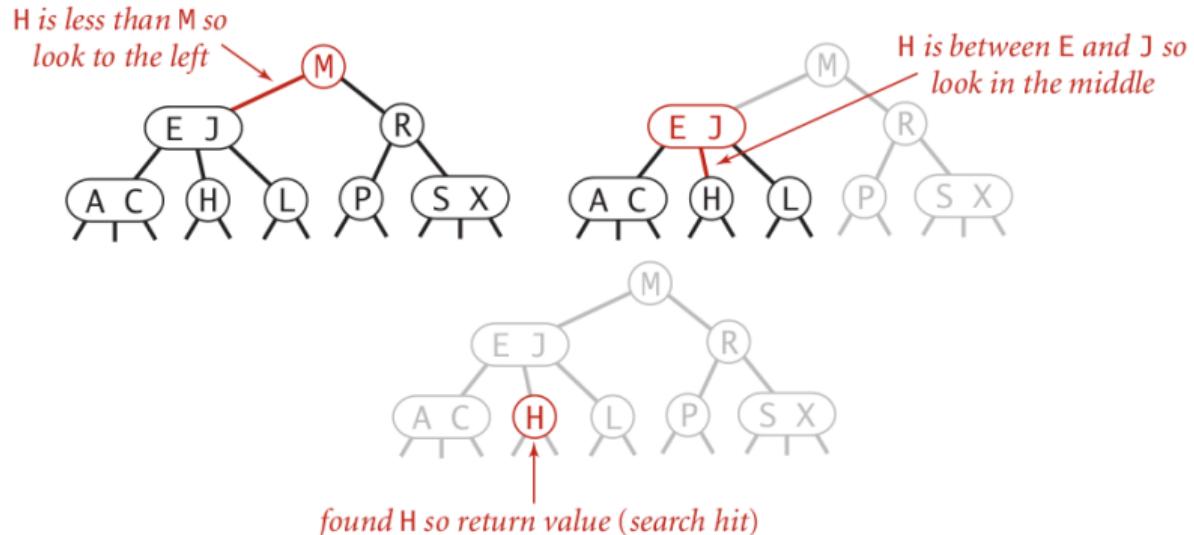
2-nó



3-nó

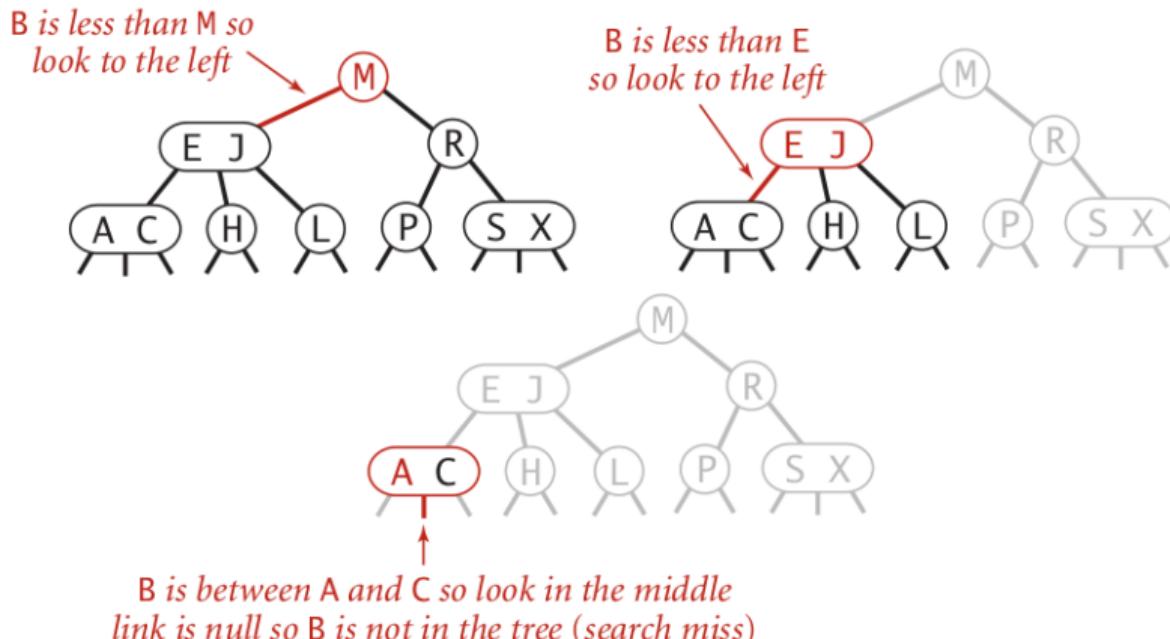


- A busca em árvores 2–3 generalizam o conceito de busca em BTSSs (*Binary Search Trees*).



Search Hit (*busca com sucesso*).

- A busca em árvores 2–3 generalizam o conceito de busca em BTSSs (*Binary Search Trees*).

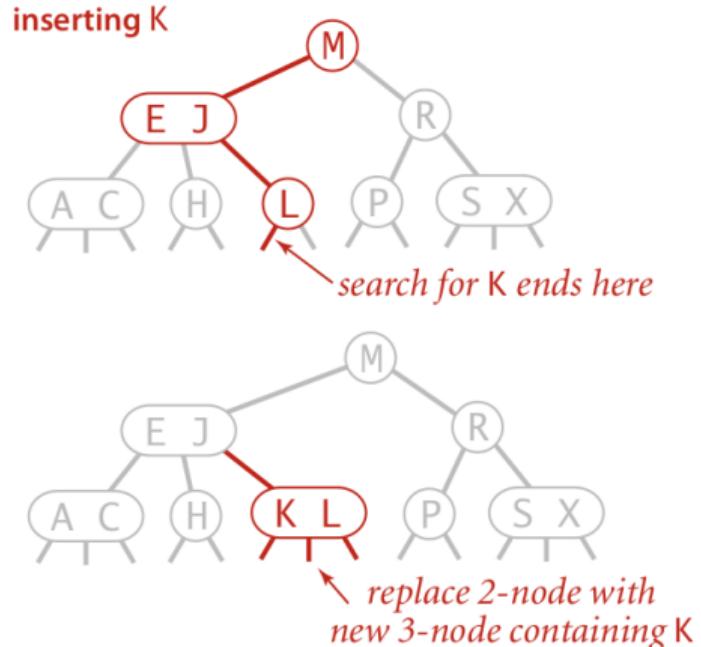


Search Hit (*busca sem sucesso*).

Inserção

Em um 2-nó

- A inserção em uma Árvore 2–3 pode se dar em um 2-nó ou em um 3-nó.
- A inserção se dá após uma busca sem sucesso e então se anexa um novo nó abaixo do último nó visitado.
- Se o último nó é um 2-nó: nós apenas substituímos o nó por um 3-nó contendo a nova chave.

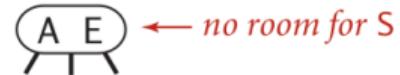


Inserção em um 2-nó.

Em um 3-nó

Inserção em um 3-nó raiz

- Nós inserimos a nova chave em um 4-nó temporário e o convertemos em um 2-nó.

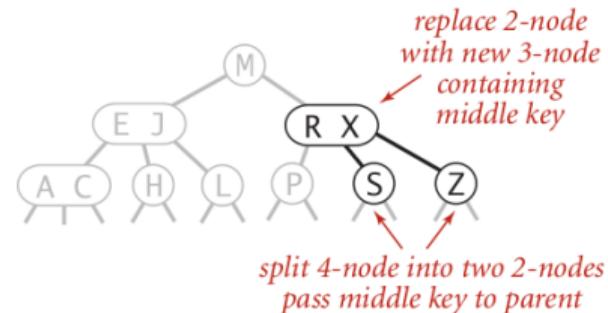
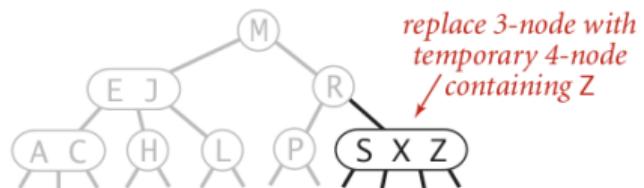
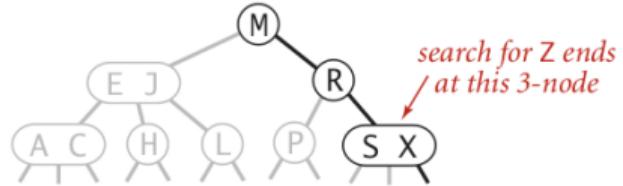
inserting S

Em um 3-nó

Inserção em um 3-nó com um 2-nó pai

- Nós inserimos a nova chave em um *4-nó temporário*, mas ao invés de criar um novo nó, movemos a chave do meio para o nó pai.

inserting Z

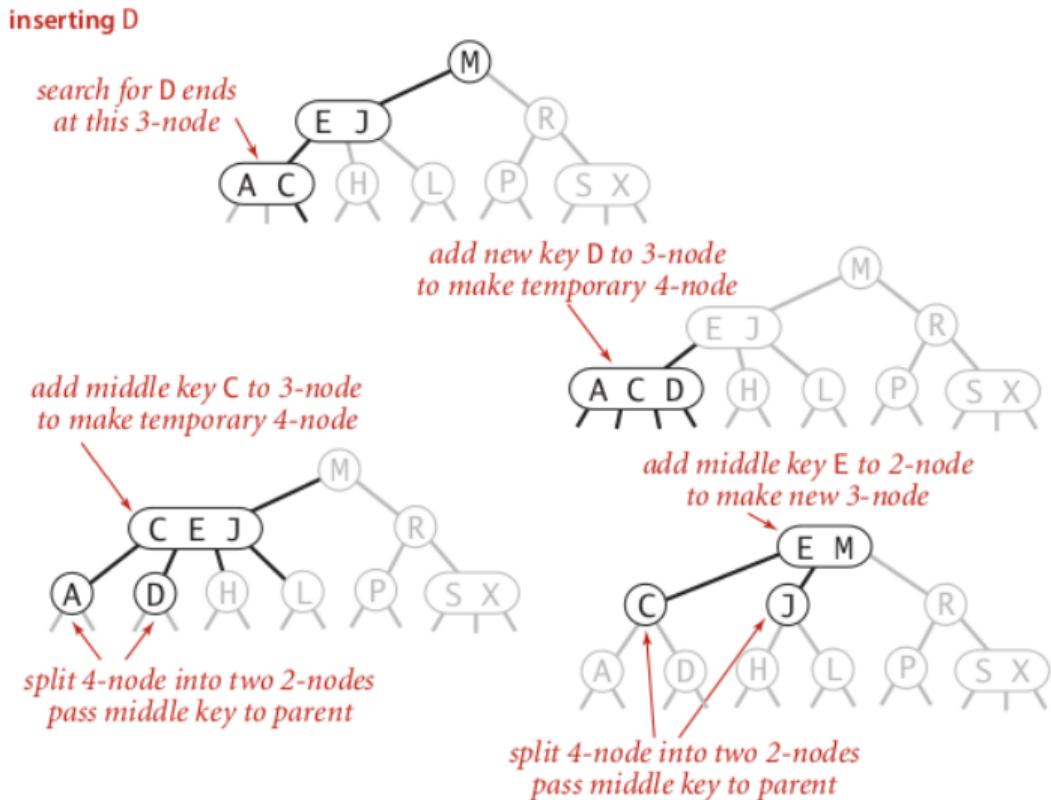


Inserção

Em um 3-nó

Inserção em um 3-nó com um 3-nó pai

- Nós inserimos a nova chave em um *4-nó temporário*, mas ao invés de criar um novo nó, movemos a chave do meio para o nó pai. Então, continuamos subindo na árvore, dividindo 4-nós e inserindo suas chaves centrais em seus pais até encontrar um 2-nó que não precise ser dividido, ou até encontrar um 3-nó que seja a raiz.



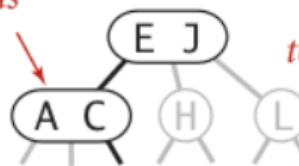
Em um 3-nó

Divisão da raiz

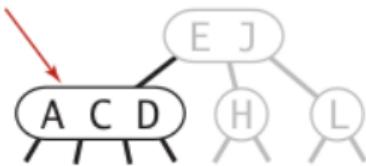
- Nós dividimos um 4-nó na raiz em três 2-nós, aumentando a altura da árvore em 1.

inserting D

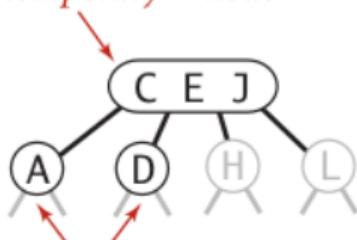
*search for D ends
at this 3-node*



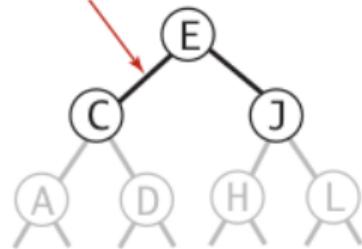
*add new key D to 3-node
to make temporary 4-node*



*add middle key C to 3-node
to make temporary 4-node*



*split 4-node into three 2-nodes
increasing tree height by 1*

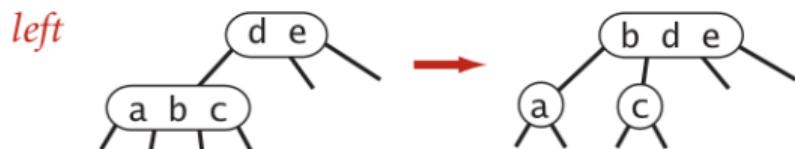


*split 4-node into two 2-nodes
pass middle key to parent*

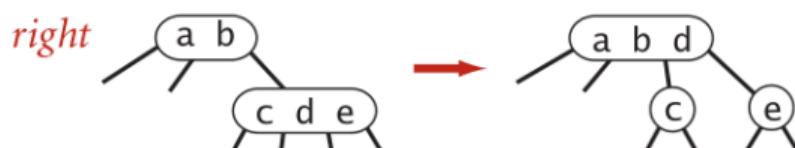
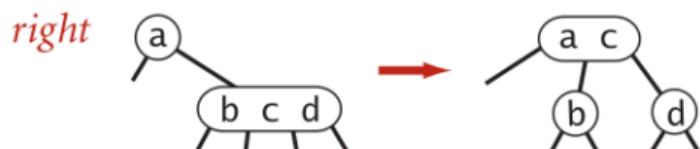
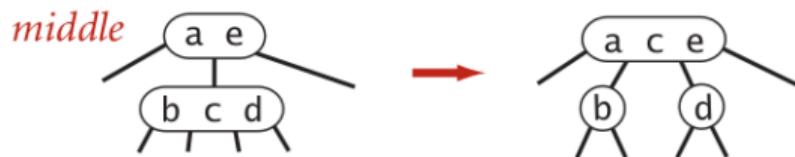
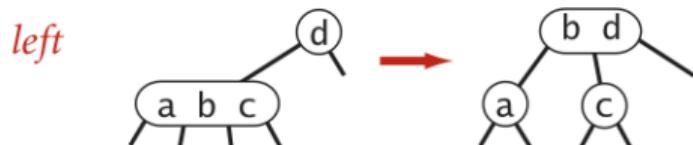
root

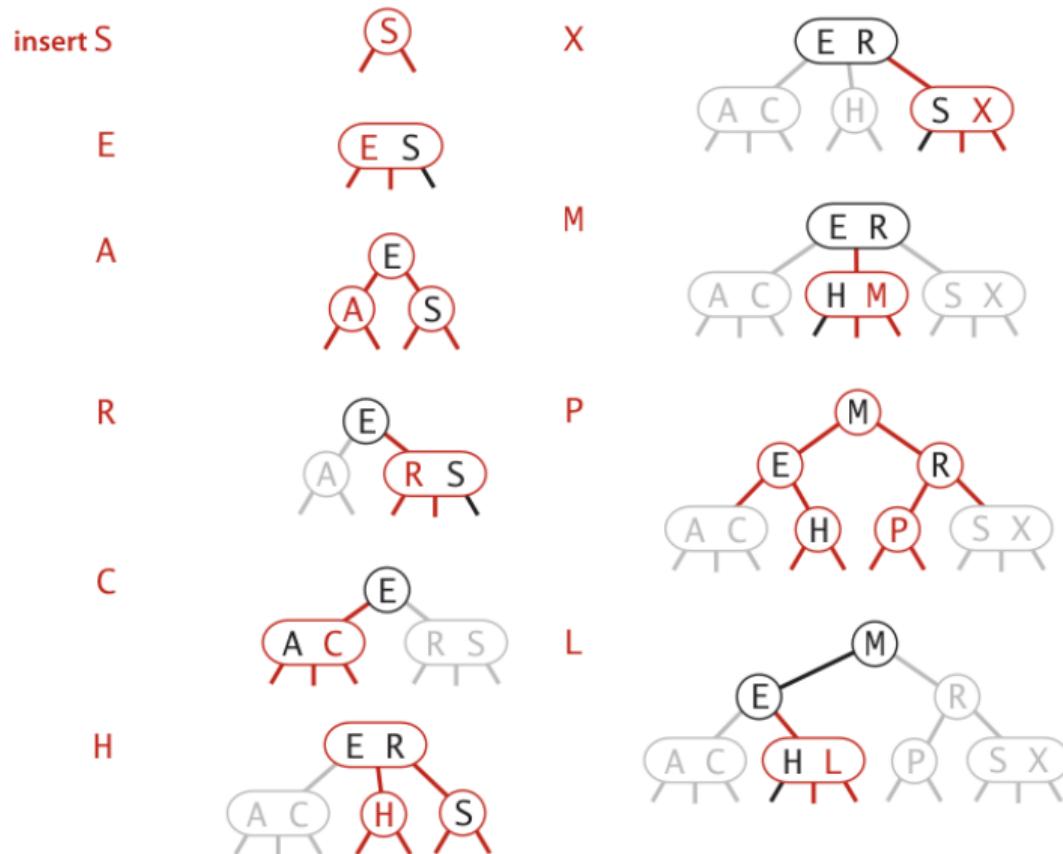


parent is a 3-node



parent is a 2-node





```
1 typedef struct Tree23 Tree23;
2
3 struct Tree23 {
4     void *valueLeft;
5     void *valueMiddle;
6     void *valueRight;
7
8     Tree23 *left;
9     Tree23 *middleLeft;
10    Tree23 *middleRight;
11    Tree23 *right;
12
13    Tree23 *parent;
14    int (*compar)(void *, void *);
15};
```

```
17 Tree23 *Tree23_alloc(int (*compar)(void *, void *));
18 void Tree23_free(Tree23 *t);
19 Tree23 *Tree23_search(Tree23 *t, void *item);
20 void Tree23_insert(Tree23 *t, void *value);
21 void *Tree23_remove(Tree23 *t, void *value);
22
23 /** Funcoes auxiliares */
24 int Tree23_nodeType(Tree23 *t);
25 Tree23 *Tree23_split4(Tree23 *t);
26 Tree23 *Tree23_split4Root(Tree23 *t);
27 Tree23 *Tree23_split4Parent2(Tree23 *t);
28 Tree23 *Tree23_split4Parent3(Tree23 *t);
```

- Uma árvore B é uma árvore auto-balanceada que armazena dados ordenados em seus nós.
- Foram inventadas por Rudolf Bayer e Edward Meyers McCreight em 1972.
- Assim como uma árvore binária permite pesquisas, inserções e remoções em tempo logarítmico.
- Cada nó pode ter m filhos e são usualmente chamados de páginas.

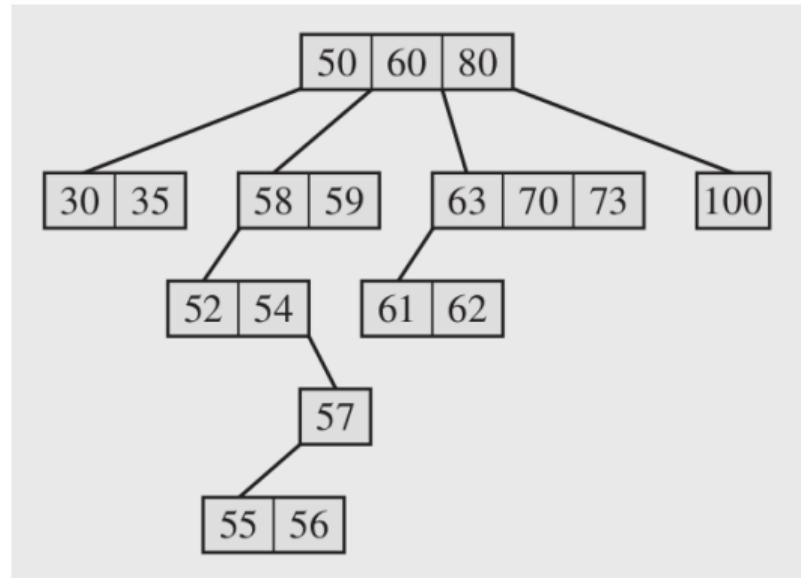


Figura 1: Exemplo de árvore B de ordem 4.



- As árvores B foram pensadas para o trabalho com dispositivos de **memória secundária**, assim, quanto **menos acessos** ao disco **melhor** será o desempenho do sistema nas operações de busca.
- Quando a informação é solicitada de um disco, a cabeça de leitura e gravação deve ser posicionada acima da parte do disco onde a informação reside.
- O disco é então girado para que todo o bloco a ser transferido para a memória passe por baixo a cabeça.

● Exemplo

- O tempo necessário para transferir 5 KB (kilobytes) de um disco que requer 40 ms (milissegundos) para localizar uma trilha, fazendo 3000 revoluções por minuto e com uma taxa de transferência de dados de 1000 KB por segundo, é

$$\text{tempo_de_acesso} = 40 \text{ ms} + 5 \text{ ms} = 45 \text{ ms}$$

- A transferência de informações de e para o disco é da ordem de milissegundos.
- A CPU processa dados na ordem dos microssegundos, 1000 vezes mais rápido, ou na ordem de nanossegundos, 1 milhão de vezes mais rápido, ou ainda mais rápido.

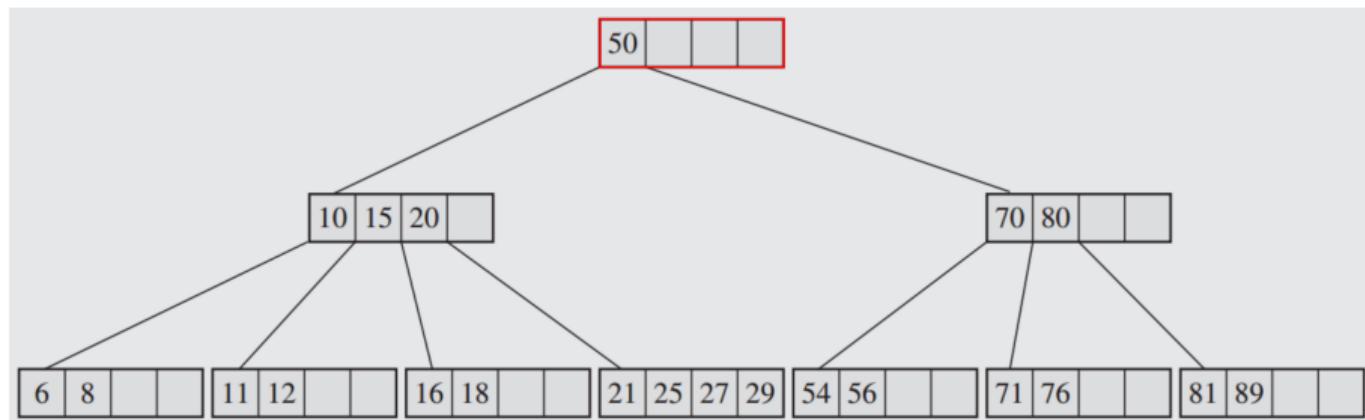
Contextualização

- Uma propriedade importante das árvores B é que o tamanho de cada nó (página) pode ser tão grande quanto o tamanho de um bloco.
- O tamanho do bloco varia para cada sistema. Pode ter 512 bytes, 4KB ou mais.
- O tamanho do bloco é o tamanho de cada nó de uma árvore B.

- De acordo com a definição de Knuth de **ordem** e **página folha** de Bayer e McCreight, uma árvore B de ordem d (número máximo de páginas filhas para uma página pai) deve satisfazer as seguintes propriedades:
 1. Cada página contém no máximo d páginas filhas
 2. Cada página, exceto a raiz e as folhas, tem pelo menos $\lceil d/2 \rceil$ páginas filhas
 3. A página raiz tem ao menos duas páginas filhas (ao menos que ela seja uma folha)
 4. Toda página folha possui a mesma profundidade, na qual é equivalente à altura da árvore
 5. Uma página não folha com k páginas filha contém $k - 1$ chaves
 6. Uma página folha contém pelo menos $\lceil d/2 \rceil - 1$ chaves e no máximo $d - 1$ chaves

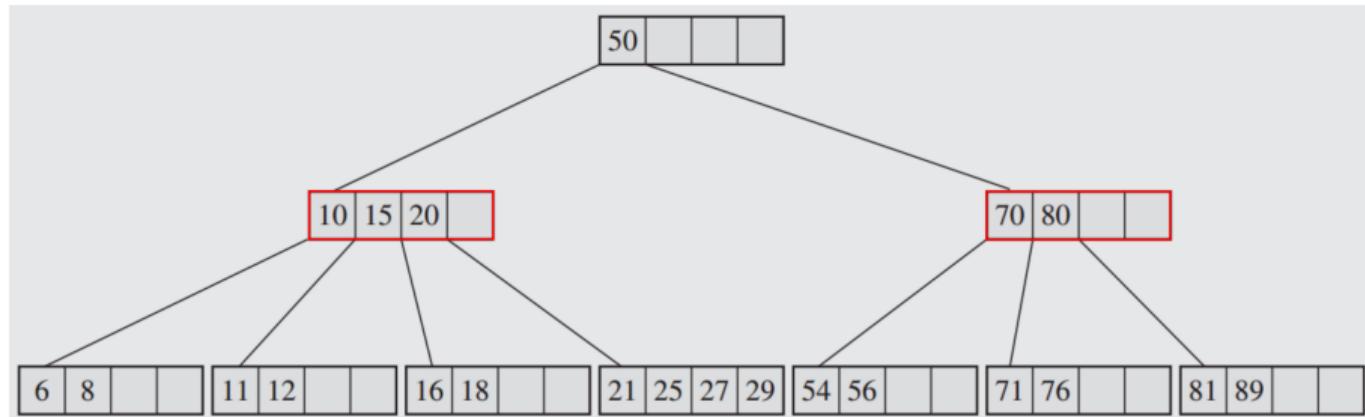
● Página raiz

- A página raiz das árvores B possuem o limite superior de $d - 1$ chaves armazenadas, mas não apresentam um número mínimo de chaves, podendo ter um número inferior a $\lceil d/2 \rceil - 1$ de chaves.



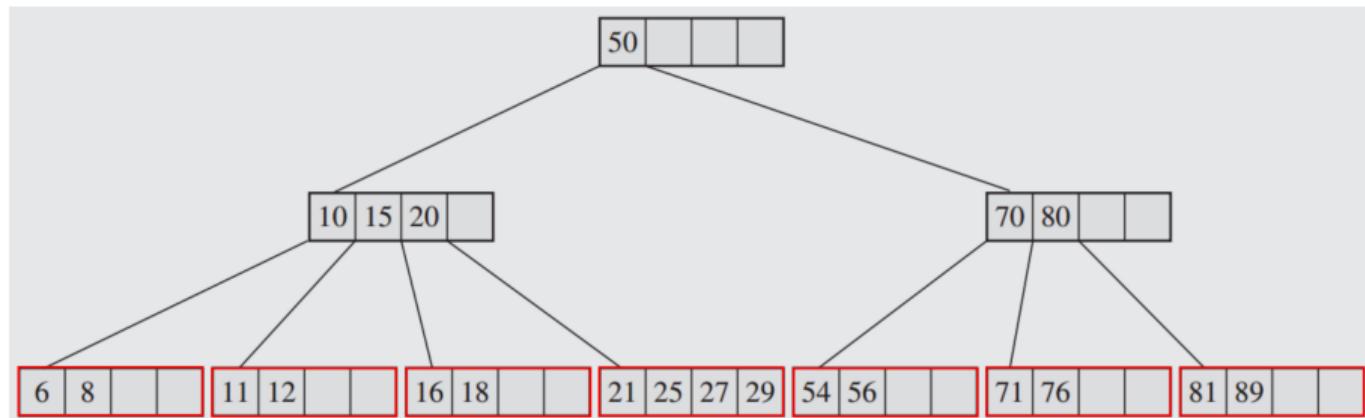
● Páginas internas

- As páginas internas são as páginas em que não são folhas e nem raiz, estas devem conter o número mínimo ($\lceil d/2 \rceil - 1$) e máximo ($d - 1$) de chaves.



● Páginas folha

- Possuem as mesmas restrições de máximo e mínimo de chaves das páginas internas, mas estes não possuem apontadores para páginas filhas.



Código 1: BTree.h

```
1 #define D 5
2
3 typedef struct BTree BTree;
4
5 struct BTree {
6     int n;
7     void *values[D-1];
8     BTree *child[D];
9     BTree *parent;
10    int (*compar)(void*, void*);
11 };
12
13 BTree *BTree_alloc(int (*compar)(void*, void*));
14 void BTree_free(BTree *b);
15 void BTree_insert(BTree *b, void *value);
16 void *BTree_search(BTree *b, void *value);
17 void *BTree_remove(BTree *b, void *value);
```