Área do Conhecimento de Ciências Exatas e Engenharias
Bacharelado em Ciência da Computação

Índices Baseados em Árvores

Organização de Arquivos

Prof. Daniel Luis Notari

Setembro - 2017

Problematização

“Como utilizar adequadamente os recursos de memória RAM e memória secundária para utilizar árvores como índices para organizações de arquivos físicas baseadas em registros?”

Sinopse

- É um arquivo auxiliar associado a uma tabela cuja finalidade é acelerar o tempo de acesso às linhas de uma tabela.
- Um índice é composto por chaves que se baseiam no conteúdo de uma ou mais colunas da tabela.
- Seu uso em banco de dados é semelhante ao índice remissivo de um livro, onde após procurar o termo desejado no índice, encontra-se facilmente a página desejada.
- A técnica de indexação é o método de otimização mais importante, pois é o que consegue obter melhores resultados em menor espaço de tempo.

Sinopse

- A utilização de índices é recomendada quando uma coluna for usada como parâmetro em um dos seguintes argumentos :
  - cláusula *where* de comandos *select* e *update*;
  - Ordenação de valores utilizando a cláusula *order by*;
  - Junções entre tabelas.
- A utilização de índices tem a desvantagem do tempo adicional gasto para atualizá-lo quando um novo registro for inserido no banco de dados.
- Antes de criar um índice, deve-se compreender as características do banco de dados, consultas e colunas de dados, para poder auxiliar a projetar melhores índices.

Momento 1: Índices baseados em Árvores

Índice baseado em árvore

- São necessários muito mais acessos à memória externa para localizar um registro com um determinado valor de chave.
- O número de acessos à memória externa,
  - necessários para localizar um registro com um valor de chave específico, é a principal razão para o frequente uso de árvores.
- Normalmente utilizados árvores para consulta de valores da chave primária.

Índice baseado em árvore

- Um método de alocar
  - os nodos deve de uma árvore deve tentar minimizar o número de acessos à memória externa para recuperar o registro desejado.
- A eficiência das árvores de pesquisa
  - é medida em termos do comprimento médio do caminho de pesquisa e/ou da complexidade associada com a inserção ou com a remoção de nodos.
- Há dois tipos básicos de árvore de pesquisa:
  - esquemas de crescimento irrestrito e
  - esquemas de crescimento restritos.

Índice baseado em árvore

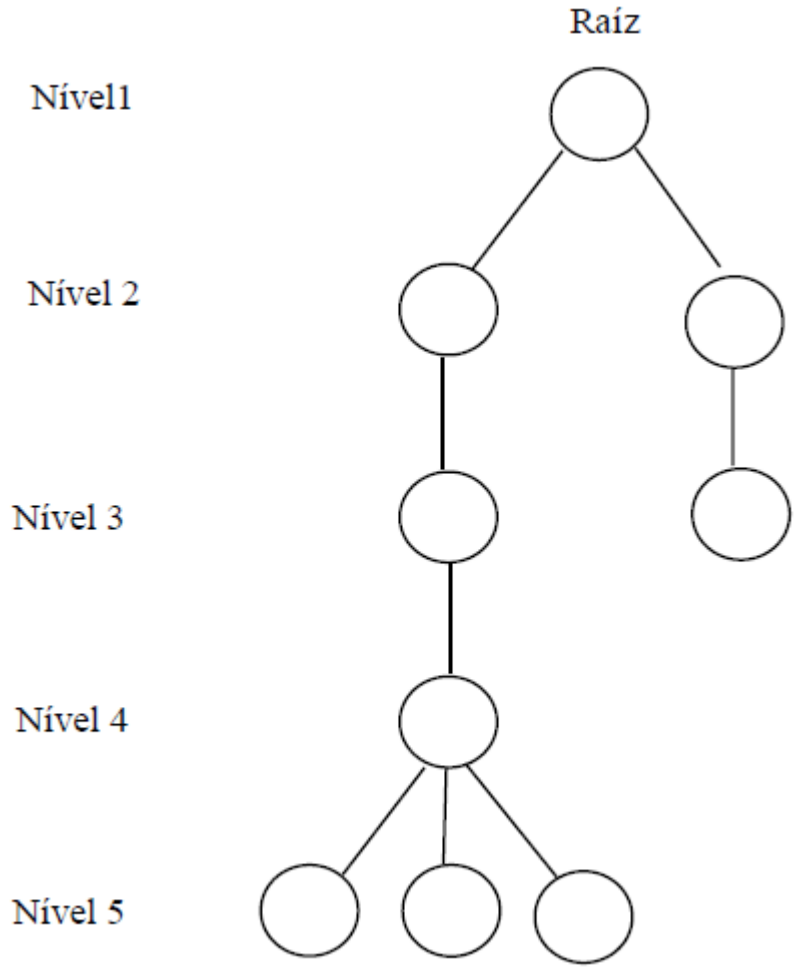
- As árvores com crescimento restrito
  - têm quase sempre melhores características de busca do que as árvores com crescimento irrestrito,
  - porque o comprimento médio do caminho de pesquisa é menor, mas a complexidade associada com a inserção e retirada de nodos é muito maior.
- O número de ramos que
  - tem que ser percorridos da raiz de uma árvore até um nodo x é chamado de comprimento do caminho do nodo x.

Índice baseado em árvore

- A raiz tem caminho de comprimento 1,
  - seus descendentes diretos têm caminhos de comprimento 2, e assim por diante.
- Um nodo no nível i tem caminho de comprimento i.
- O comprimento do caminho de uma árvore é definido como a soma dos comprimentos de todos os seus nodos.

Índice baseado em árvore

- O comprimento interno da árvore é:
- PI = 1 x 1 + 2 x 2 + 2 x 3 + 1 x 4 + 3 x 5 = 30
- e o comprimento médio de caminho interno e PI = 30/9 = 3,33.



Índice baseado em árvore

- A terminologia comprimento médio de caminho refere-se ao
  - número médio de comparações ou tempo médio de pesquisa,
  - é usada em diversas referências para expressar as características de pesquisa das árvores.

Momento 2: Árvore de Pesquisa Binária

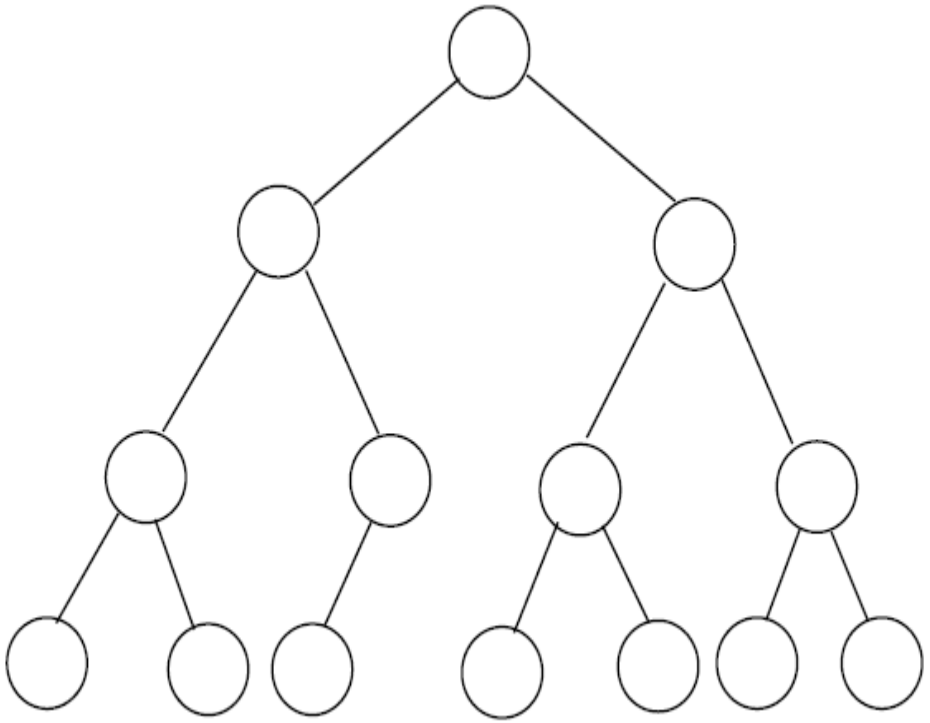
Árvores de Pesquisa Binária

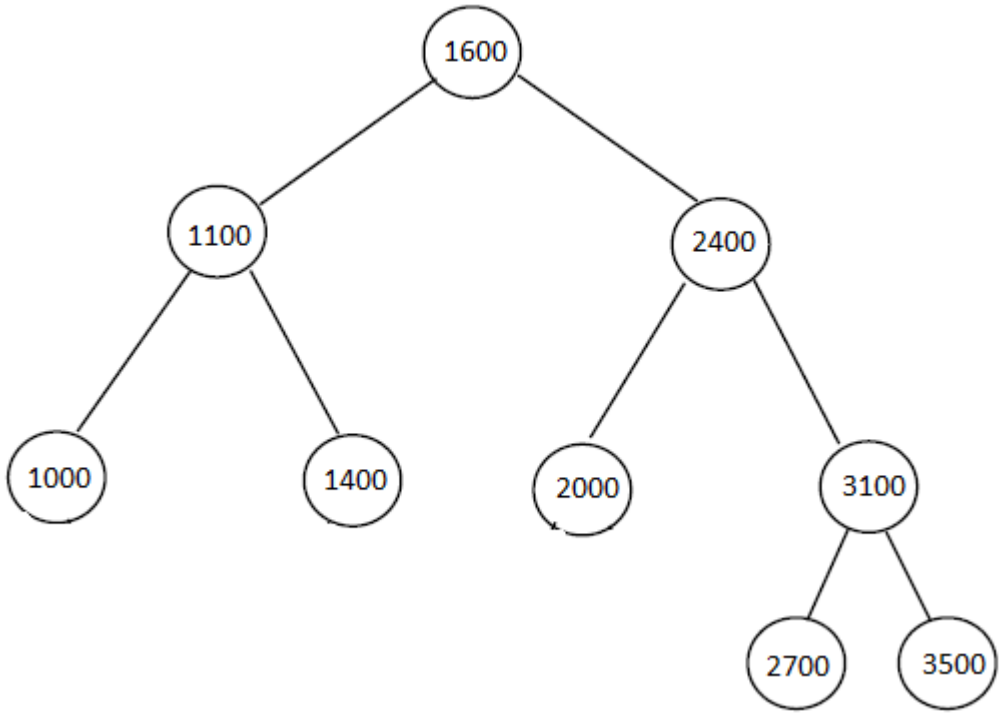
- Uma árvore de pesquisa binária
  - é uma árvore binária organizada de modo que para cada nodo x,
    - todos os valores de chaves dos nodos da subárvore à esquerda do nodo x sejam menores que o valor de chave do nodo x e,
    - os da subárvore à direita do nodo x, sejam maiores que o valor de chave do nodo x.
- Tipos:
  - Árvores de Pesquisa Binária Aleatória e Perfeitamente Balanceada
  - Árvores de Pesquisa Binária Ótimas
  - Árvores AVL

Árvores de Pesquisa Binária Aleatória é Perfeitamente Balanceada

- Uma árvore de pesquisa binária
  - de crescimento irrestrito construída com valores de chave em ordem aleatória é chamada árvore de pesquisa binária aleatória.
- A melhor solução
  - para a forma de uma árvore de pesquisa aleatória é aquela que seja perfeitamente balanceada.
- É perfeitamente balanceada
  - se para cada nodo o número de nodos (altura) nas subárvores à direita e à esquerda diferir de no máximo em um.

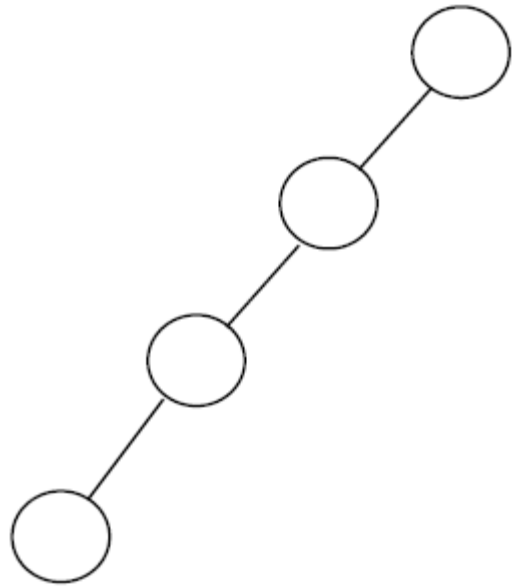
Árvores de Pesquisa Binária Aleatória e Perfeitamente Balanceada



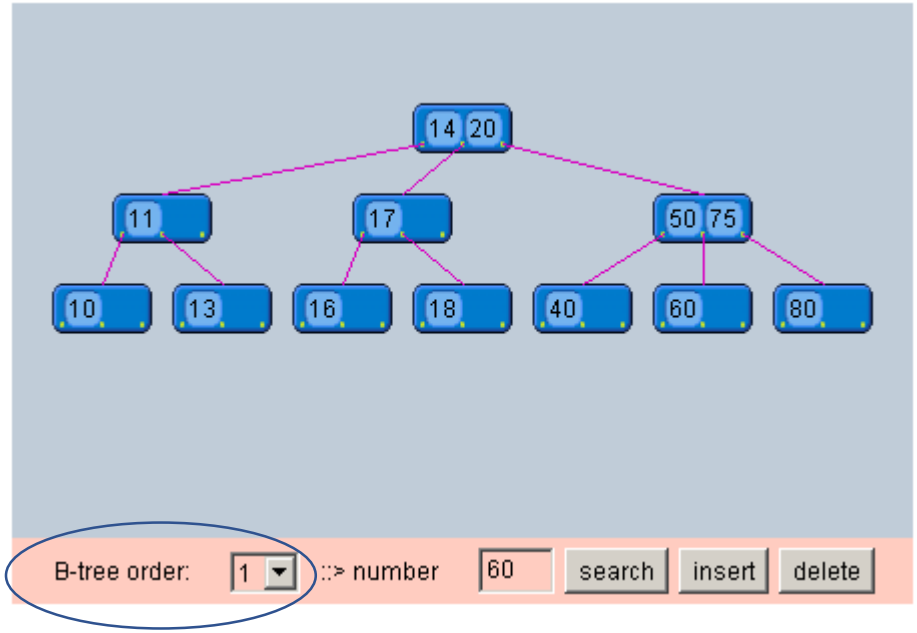


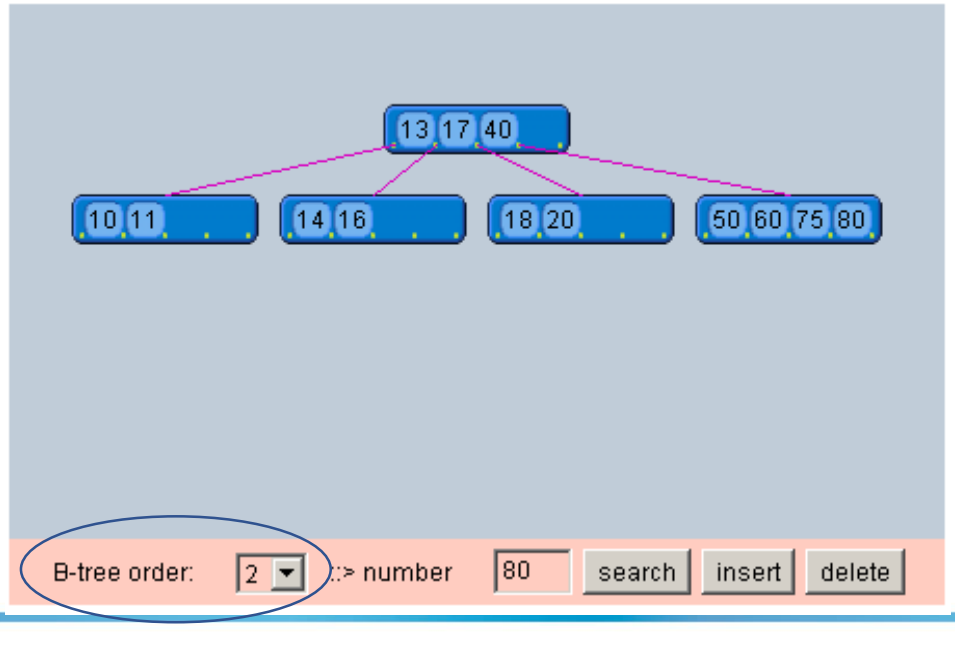
Árvores de Pesquisa Binária Aleatória e Perfeitamente Balanceada

- A pior forma de uma árvore de pesquisa aleatória é a árvore degenerada,
  - isto é, uma árvore construída quando todos os valores de chave chegam em ordem crescente ou decrescente.



Árvore de Pesquisa Binária





Árvores de Pesquisa Binária Ótimas

- Quando não é possível determinar a frequência com que cada valor de chave será acessado,
  - pode-se supor que cada nodo da árvore possui chances iguais de ser acessado.
- Se for possível conhecer essa probabilidade de acesso,
  - pode-se construir uma árvore de pesquisa ótima,
  - onde os nodos que possuem maior probabilidade de acesso possuirão caminhos de acesso menores (mais curtos)
  - e os nodos com menor probabilidade poderão possuir caminhos maiores (mais longos).

Árvores de Pesquisa Binária Ótimas

- Pode-se perceber
  - que as árvores de pesquisa para uma determinada distribuição,
  - normalmente, podem não ser árvores perfeitamente balanceadas.
  - Mas o caminho médio de acesso será pequeno.
- A árvore de pesquisa binária ótima somente é útil quando se tem informação constante,
  - ou seja, não serão realizadas inclusões e exclusões dinamicamente.
- Deve-se conhecer
  - A frequência de acesso aos nodos,
  - com essas frequências não são conhecidas com muita precisão,
  - Pode não ser vantajoso gastar muito esforço na construção de uma árvore de pesquisa ótima utilizando pesos imprecisos.

Momento 3: Árvore AVL

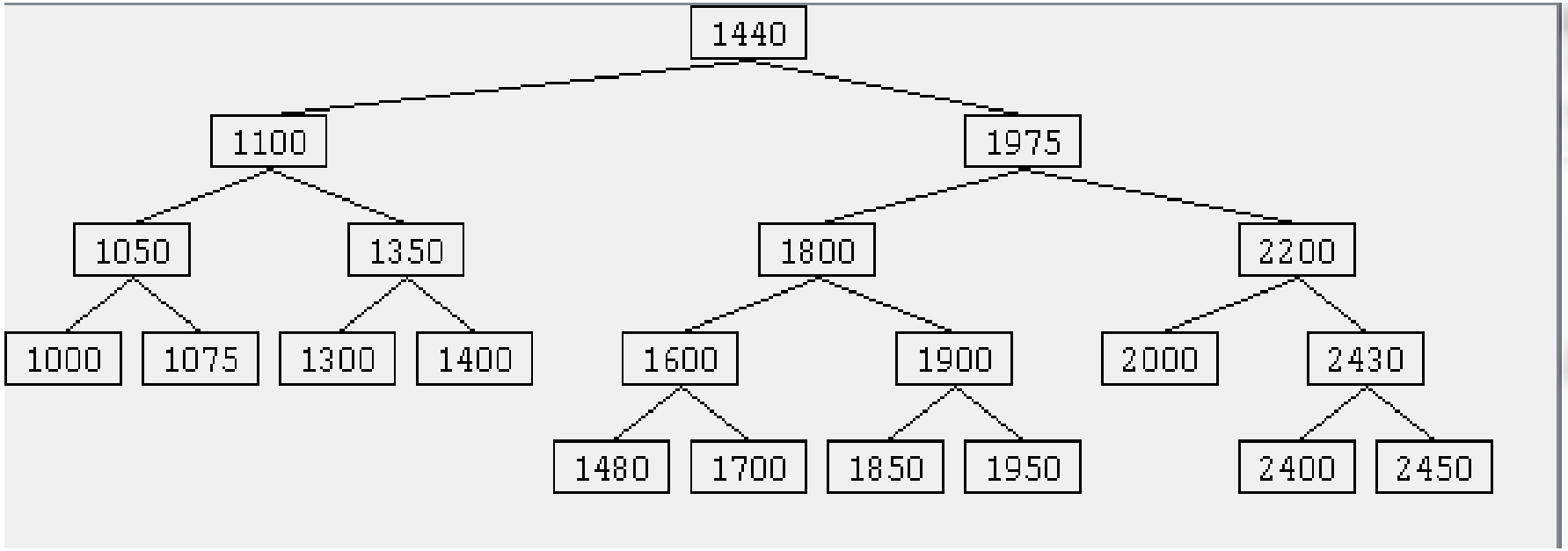
Árvores AVL

- Um requisito para uma árvore dinâmica
  - é que as inserções e retiradas sejam executadas facilmente,
  - mantendo a árvore dentro da classe desejada.
- Apesar de a árvore AVL possibilitar inserções e exclusões,
  - só deve-se utilizar este tipo de árvore quando o número de consultas for muito maior que o número de inclusões/retiradas,
  - pois a performance deste pode ser baixa.

Árvores AVL

- São uma solução para situações que requeiram árvores de pesquisa binária dinâmicas.
- É uma árvore de pesquisa binária de crescimento restrito,
  - que relaxa os critérios de balanceamento da árvore perfeitamente balanceada,
  - possibilitando assim a reestruturação após inserções e retiradas menos complexas.
- Com esse relaxamento tem-se um pequeno acréscimo ao tempo médio de pesquisa.

Exemplo AVL


1440
1100
1975
1050
1350
1800
2200
1000
1075
1300
1400
1600
1900
2000
2430
1480
1700
1850
1950
2400
2450

Momento 4: Árvore de Pesquisa de Caminho Múltiplo

Árvores de Pesquisa de Caminho Múltiplo

- Oferecem uma alternativa viável e prática às árvores de pesquisa binária.
- Isso é especialmente verdade
  - na construção e manutenção de grandes árvores de busca,
  - nas quais as inserções e retiradas são necessárias,
  - mas em que a memória interna do computador não é suficiente.

Árvores de Pesquisa de Caminho Múltiplo

- Uma organização
  - que requeira poucos acessos à memória externa para localizar um registro é muito desejável
  - e a árvore de pesquisa de caminho múltiplo é uma solução prática para esse problema.
- Tipos:
  - Estruturas Trie
  - Árvores B

Estruturas Trie

- Uma estrutura *trie* é uma árvore m-ária completa,
  - na qual cada nodo consiste em *m* componentes um nodo pode ser considerado como um valor de *m-lugares*.
- Os componentes de uma *trie* são dígitos ou letras.
- No lugar de acessar
  - os registros com base na comparação entre o valor da chave de acesso e a chave de cada registro,
  - usa-se a representação de um valor de uma chave como sequencia de dígitos ou caracteres alfabéticos.

Estruturas Trie

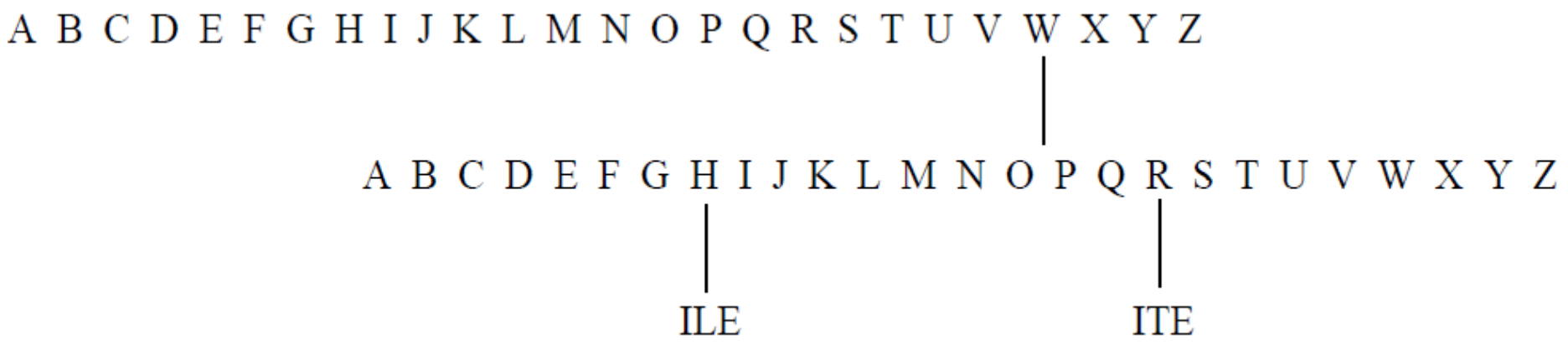
- Cada nodo do nível I
  - representa um conjunto de valores de chave que começa com uma sequencia de I caracteres.
- Um nodo (do nível I)
  - especifica uma ramificação de m caminhos dependendo do i-ésimo caractere do valor de chave,
  - isto é, no nível I o i-ésimo caractere é usado para determinar o nodo a ramificar-se para o nível (I+1).

Estruturas Trie

- Para construir uma estrutura *trie* para um determinado conjunto de *n* valores de chave,
  - os valores são, inicialmente, colocados em ordem classificada.
  - A *trie* é construída, normalmente, à base de um nível de cada vez.
- A estrutura *trie* possibilita
  - que se manipule índices não numéricos mesmo que estes tenham tamanho variável.
- Mesmo que o tamanho das palavras seja grande,
  - não é necessário ter-se um nível para cada *caractere* da maior palavra, apesar de esta ser a técnica mais simples, pode envolver muito gasto de memória.
- O que é necessário
  - é possuir um número suficiente de níveis para identificar cada palavra.

Estruturas Trie

- Exemplo:
  - suponha que em um dicionário tenhamos as seguintes palavras: WHILE, WRITE.
  - Neste caso são necessários apenas dois níveis e já podemos identificar cada uma das palavras.



Estruturas Trie

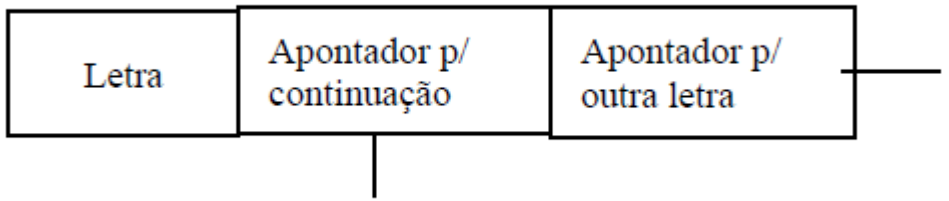
- As chaves numéricas também são beneficiadas pelas estruturas *trie*.
- O número máximo de níveis de uma *trie* para um número será o número máximo de dígitos que esse número poderá possuir.
- O tamanho do caminho de acesso a cada chave será constante e igual a altura da árvore.
- O tempo médio de pesquisa
  - em que *n* é o número de valores de chave e *m* é o número de componentes de cada nodo.

Estruturas Trie

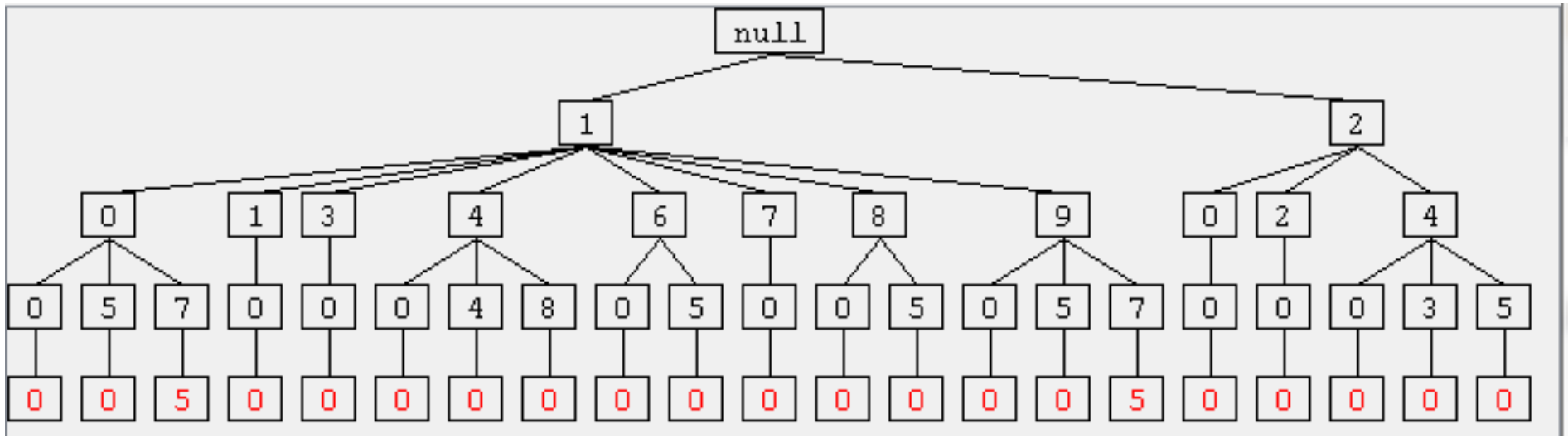
- Uma estrutura *trie* pode ser implementada de diversas maneiras.
- Uma seria utilizar uma lista entre cada um dos elementos possíveis para cada nodo da árvore,
  - o que possibilitaria não utilizar espaço por aqueles elementos não usados.
- Outra forma seria utilizar um vetor para cada um desses elementos,
  - o que gastaria mais memória, entretanto torna mais fácil o acesso a cada caractere, de cada nível.

Estruturas Trie

- A utilização de estruturas dinâmicas torna mais eficaz a utilização da memória.
- Neste contexto poderíamos definir cada nodo da seguinte maneira:



Árvore Trie


null
1
2
0
1
3
4
6
7
8
9
0
2
4
0
5
7
0
0
0
4
8
0
5
0
0
5
0
5
7
0
0
0
3
5
0
0
5
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0

Momento 5: Árvore B

Árvores B

- É uma árvore de pesquisa de caminho múltiplo e de crescimento restrito.
- Uma árvore B de ordem m é uma árvore que satisfaz as seguintes propriedades:
  1. Todo nodo tem o número de descendentes menor ou igual a m.
  2. Todo nodo, exceto a raiz e os nodos terminais, tem o número de descendentes maior ou igual a m/2.

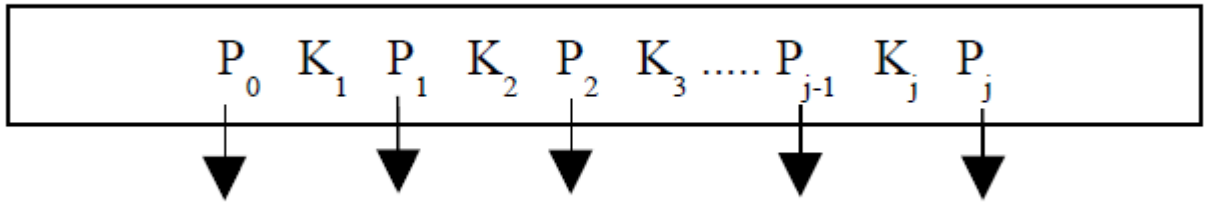
Árvores B

3. A raiz tem pelo menos dois descendentes, a menos que seja nodo terminal.
4. Todos os nodos terminais aparecem no mesmo nível e carregam nenhuma informação.
5. Um nodo interno com *k* descendentes contém *k-1* valores de chave.

- Todos os valores de chave
  - em cada nodo aparecem da esquerda para a direita
  - e o número de descendentes de cada nodo é exatamente maior em um do que o número de valores de chave de cada nodo.

Árvores B

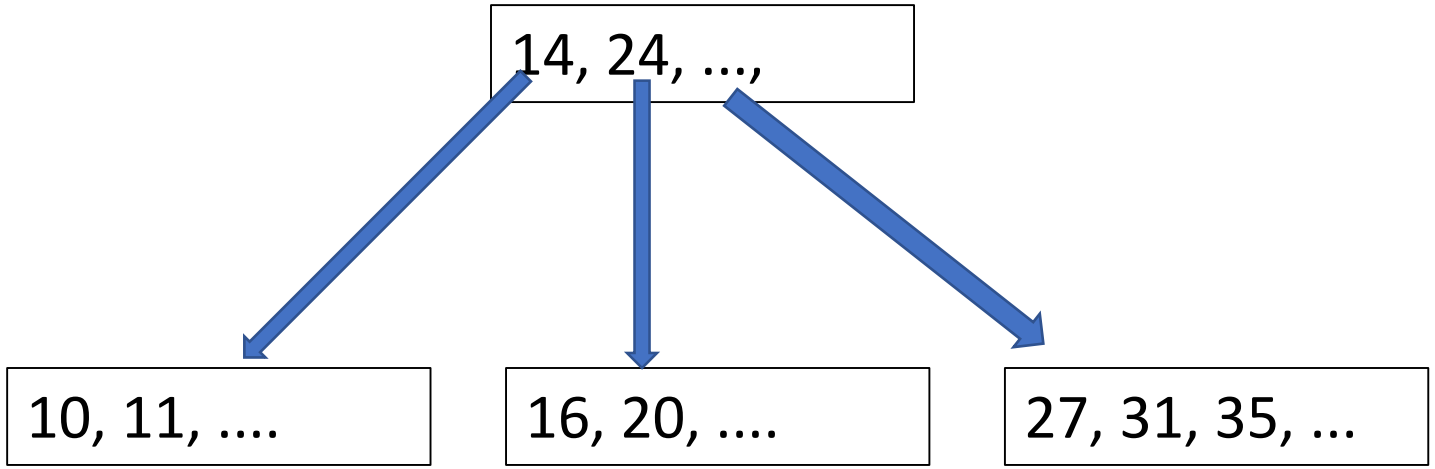
- Um nodo que contém j valores de chave e j+1 indicadores, pode ser representado da seguinte maneira:



- onde, K1 < K2 < K3 < ... < Kj-1, < Kj, 1 <= j < m
- e cada p_i indica uma subárvore contendo

valores de chave entre os valores Ki e Ki+1.

Árvores B



Árvore B de ordem 2

Árvores B

- Fazer uma pesquisa em uma árvore B por um certo valor de chave é como se segue:
  - Um nodo, começando com o nodo raiz, é trazido para a memória interna e pesquisado,
  - Possivelmente usando uma pesquisa binária para grandes valores de *j* para o dado argumento de valor de chave dentre os valores de *K*.
  - Se a busca tiver êxito, então o valor de chave é localizado, mas se a busca não tiver êxito porque o valor encontra-se entre dois valores de K então o valor poderá estar na subárvore que é indicada por P.

Árvores B

- A eficácia de pesquisa da árvore B
  - é determinada pela forma da árvore
  - e a forma de árvore é determinada pela ordem *m*.
- Quando *m* é muito pequeno,
  - uma árvore é alta e estreita e quando *m* é muito grande a árvore é grande e espessa.
- É preferível utilizar-se um valor grande para *m*,
  - entretanto sem chegar ao extremo de ter-se apenas um nível na árvore,
  - pois neste caso a consulta dentro do nodo poderia tornar-se ineficiente.

Árvores B

- As árvores de pesquisa de caminho múltiplo (como a árvore B)
  - possuem uma relação importante entre o fator de ramificação dos nodos da árvore (s) e a altura da árvore.
- O fator de ramificação é o número de nodos
  - que podem emanar dos nodos e é geralmente,
  - aproximadamente, o mesmo número de chaves que podem ser colocadas em um nodo (podendo exceder em um o número de chaves).

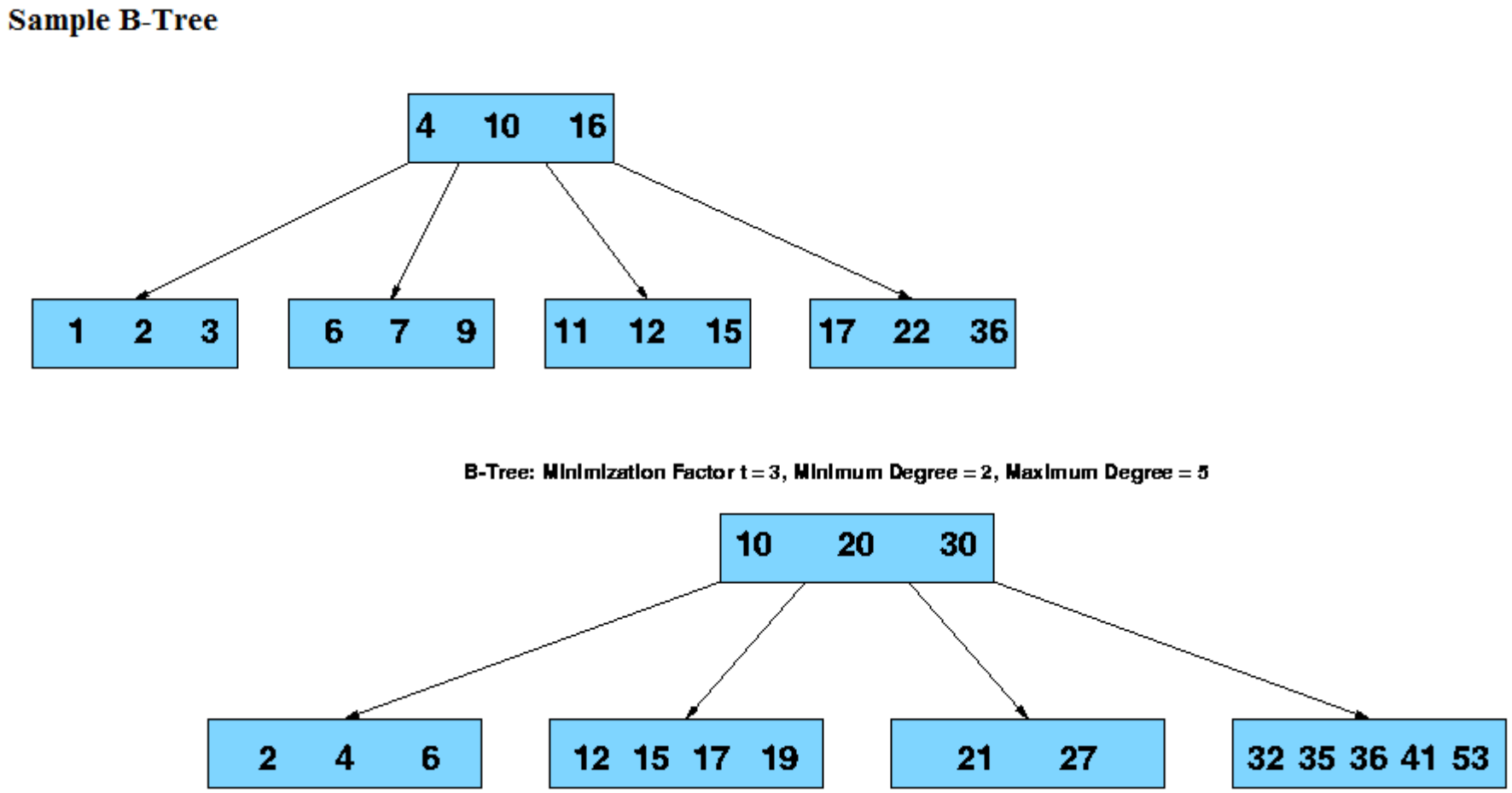
Árvores B

- À medida que o valor de s aumenta,
  - a altura da árvore diminui;
  - contudo nodos maiores demoram mais tempo para serem carregados para a memória.
- A medida que *s* diminui,
  - a altura de uma árvore aumenta;
  - assim sendo, mas nodos devem ser carregados para a memória e mais nodos devem ser acessados para localizar-se uma determinada chave.

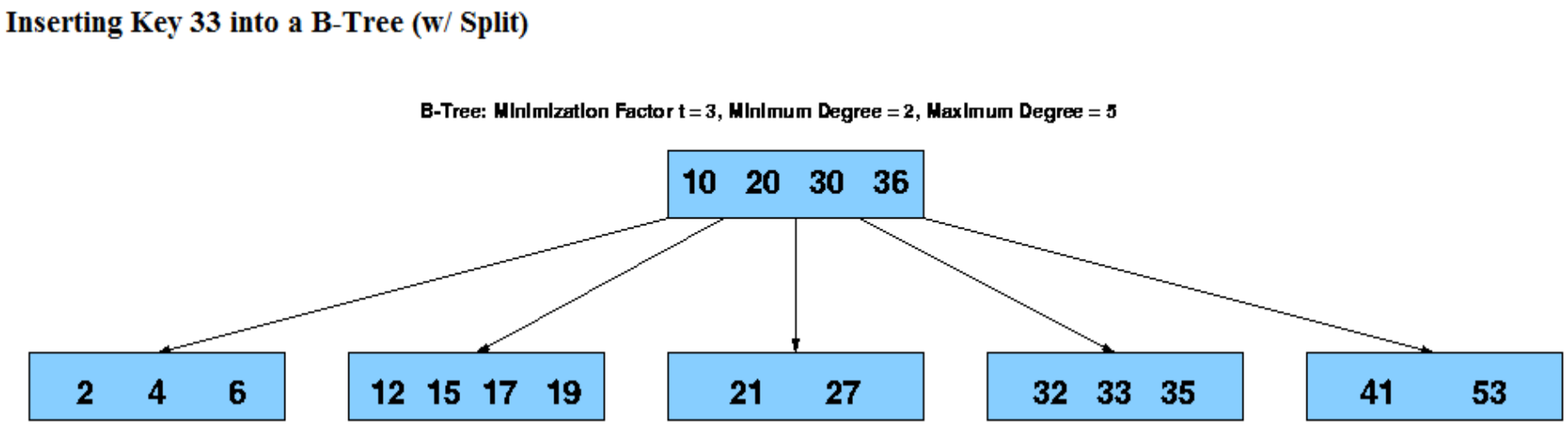
Árvores B

- A utilização de árvores B com arquivos tem como objetivo facilitar o acesso aos registros, minimizando os acessos à memória externa,
  - por esta razão, como em outras estruturas utilizadas para organizar as chaves em memória interna, o ideal é que toda a estrutura de chaves seja mantida na memória.
- Para associar-se as chaves nos seus registros é
  - suficiente ter-se em cada nodo as chaves representadas como duplas <chave,endereço>,
  - onde o endereço é a posição relativa do registro ao qual a chave pertence dentro do arquivo de dados.

Árvores B - Exemplo



Árvores B - Exemplo



Momento 6: Exercícios

Problematização

Qual é a resposta ?

““Como utilizar adequadamente os recursos de memória RAM e memória secundária para utilizar árvores como índices para organizações de arquivos físicas baseadas em registros?”

Exercícios

1) Porque utilizar estruturas de árvores como índices?

2) A estrutura de índice deve ser manipulada em arquivo ou em memória?

3) Qual a relação entre o tamanho do arquivo e o tamanho do índice e, por consequência, o tamanho de memória a ser utilizado?

4) Qual o tipo de índice que deve ser utilizado com árvores?

Exercícios

5) Agora, vamos pesquisar como são implementados alguns índices em um SGBDR. Para tanto, descrevam a estrutura de dados utilizada para cada tipo:

- Cluster Index Scan
- Not-cluster Index
- Exclusive Index
- Index Seek
- Index Scan
- Hash Match
- Merge Join
- Nested Loop Join