Estruturas de Índice

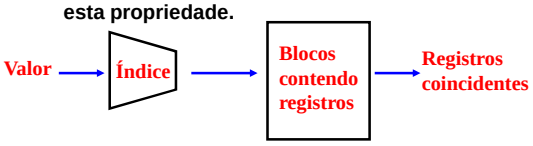
Valéria Times

TÓPICOS

- Conceitos Básicos
- Indexação e Entradas no Índice
- Índices sobre Arquivos Classificados
- Índices Secundários
- Árvores B
- Tabelas de Hash

Estruturas de Índice

- **Índice:**
  - Qualquer estrutura de dados que recebe como entrada uma propriedade de registros (em geral, o valor de um ou mais campos) e se destina a rapidamente encontrar registros com esta propriedade.



  - Permite localizar um registro examinando apenas uma pequena fração de todos os registros possíveis.

Estruturas de Índice

- **Chave de Pesquisa:**
  - Campo(s) em cujos valores o índice se baseia.
  - Atributo(s) sobre o(s) qual(is) recebemos valores e realizamos pesquisas, através de um índice, de *tuplas* com valores coincidentes.
  - Não é o mesmo que chave primária (conjunto mínimo de campos que unicamente identifica um registro na relação).
  - Um Índice em um arquivo acelera as seleções no campo de chave de pesquisa para o Índice.
    - Qualquer subconjunto do campo de uma relação pode ser a chave de pesquisa para um Índice na relação.

Estruturas de Índice

- **Indexação:**
  - Um Índice acelera certos tipos de consultas, mas não todas.
  - Muitas alternativas existem, cada uma mais indicada para uma situação e não muito para outras.
  - Índices mantêm informações auxiliares para direcionar a busca aos dados desejados.
  - Um Índice contém uma coleção de entradas e um mecanismo eficiente para localizar todas as entradas para uma dada chave de pesquisa (K).
  - Cada entrada no Índice (K*) contém informação suficiente para permitir a recuperação de um ou mais registros de dados contendo K.

Estruturas de Índice

- **Alternativas para Entradas no Índice:**
  - Uma entrada K* permite a recuperação de um ou mais registros de dados com chave = K.
  - Principais alternativas: Cada entrada consiste:
    - Próprio registro de dados com chave = K.
    - <K, ID do registro com chave = K>
    - <K, lista de IDs dos registros com chave = K>
  - Se a primeira delas for usada:
    - Não existe a necessidade de manter registros de dados separadamente.
    - A estrutura de índice é uma organização de arquivo para registros (Heap ou ordenados).

Estruturas de Índice

- Alternativas para Entradas no Índice:
  - Se a primeira delas for usada: (Cont.)
    - No máximo, um índice numa coleção de registros de dados pode usá-la.
      - Senão, dados são duplicados levando à redundância e potencial de inconsistência.
  - Se uma das duas últimas forem adotadas:
    - Porção da estrutura de indexação usada para direcionar a busca é muito menor do que com a primeira.
    - Se mais do que um índice é requerido no arquivo, então no máximo um índice pode usar 1a. Alternativa e o restante deve usar 2a./3a.

Estruturas de Índice

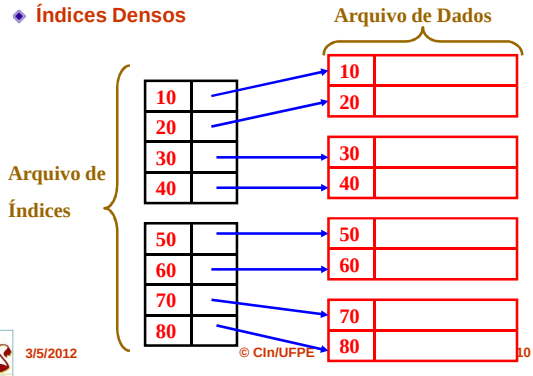
- Alternativas para Entradas no Índice:
  - Se uma das duas últimas forem adotadas (Cont.)
    - 3a. Alternativa é mais compacta do que a 2a., mas leva a tamanhos variáveis de valores indexados, mesmo se a chave de pesquisa é de tamanho fixo.
  - Há muitas estruturas de dados diferentes que servem como índices:
    - Índices simples em arquivos classificados
    - Índices secundários em arquivos não classificados
    - Árvores B
    - Tabelas de Hash

Estruturas de Índice

- Índices sobre Arquivos Classificados
  - Cada índice consiste em um par chave-ponteiro mantido em um arquivo de índices.
    - Pares são mantidos em ordem classificada de seus valores de chave.
  - Baseia-se na classificação do arquivo de dados sobre os atributos do índice.
  - Útil quando a chave de pesquisa é a chave primária da relação.
  - Definição:Consiste na classificação do arquivo de dados de acordo com alguma chave de pesquisa e na aplicação de um índice sobre este arquivo.

Índices em Arquivos Ordenados

- Índices Densos



Índices em Arquivos Ordenados

- Índices Densos (Cont.)
  - Cada chave de pesquisa no arquivo de índices está associada a um ponteiro para um registro do arquivo de dados com o valor da chave.
  - Tem este nome porque cada chave do arquivo de dados está representada no índice.
  - Existe uma entrada no arquivo de índices (par chave-ponteiro) para cada registro do arquivo de dados.
  - Blocos de índices do índice denso mantêm as chaves classificadas na mesma ordem do arquivo de dados.

Índices em Arquivos Ordenados

- Índices Densos (Cont.)
  - Espera-se usar um número menor de blocos para o índice que para o próprio arquivo já que chaves e ponteiros possivelmente ocupam menos espaço que registros completos.
  - É especialmente vantajoso quando ele (mas não o arquivo de dados) cabe na memória principal.
  - Dada sua chave de pesquisa, usando um índice, pode-se encontrar qualquer registro com apenas uma operação de E/S de disco por pesquisa.

Índices em Arquivos Ordenados

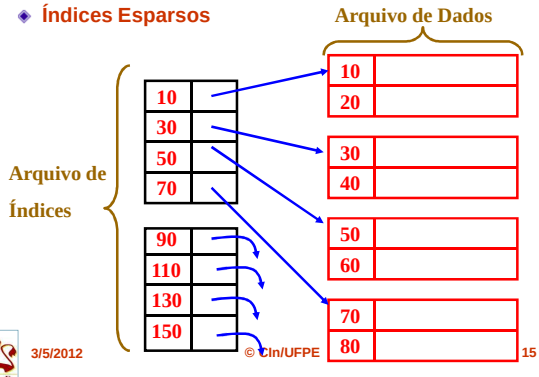
- Índices Densos (Cont.)
  - Admite consultas que buscam registros com um determinado valor de chave de pesquisa.
  - Dado o valor de uma chave K:
    - pesquisa-se os blocos de índices para encontrar K.
    - Segue-se o ponteiro associado até o registro com chave K.
  - Existem vários fatores que tornam a pesquisa baseada em índices mais eficiente do que parece.

Índices em Arquivos Ordenados

- Índices Densos (Cont.)
  - Razões da eficiência do uso de índices:
    - Número de blocos de índices em geral é pequeno quando comparado com o número de blocos de dados.
    - Como as chaves são classificadas, pode-se usar a pesquisa binária para encontrar uma dada chave nos blocos de índices.
      - Se tem N blocos, então examina-se $\log_2 N$
    - Índice pode ser pequeno o bastante para ser mantido permanentemente em buffers da memória principal.

Índices em Arquivos Ordenados

- Índices Esparsos



Índices em Arquivos Ordenados

- Índices Esparsos
  - Apenas alguns dos registros de dados estão representados no arquivo de índices.
    - Geralmente é mantido um único índice por bloco do arquivo de dados.
  - Tem um par chave-ponteiro para cada bloco do arquivo de dados.
  - A chave associada a um ponteiro para um bloco é a primeira chave encontrada neste bloco.
  - Usa menos espaço ao custo de um tempo maior para encontrar um registro, dada a sua chave.

Índices em Arquivos Ordenados

- Índices Esparsos (Cont.)
  - Dado o valor de uma chave K:
    - Pesquisa-se os blocos de índices para encontrar a maior chave que seja ≤ a K.
      - Como o arquivo de índices está ordenado por chave, pode-se usar a pesquisa binária para localizar esta entrada.
    - Segue-se o ponteiro associado a um bloco de dados.
    - Pesquisa-se o bloco em busca do registro com a chave K.
    - Bloco deve ter informações de formato suficientes para que seus registros possam ser identificados.

Índices em Arquivos Ordenados

- Índices Densos x Índices Esparsos
  - Densos:
    - Permitem responder a consultas do tipo:
      - *Existe um registro com o valor de chave igual a K?*
      - Isto pode ser feito sem ter de recuperar o bloco que contém o registro.
      - O fato de K existir no índice denso é suficiente para garantir a existência do registro com a chave K.

Por outro lado...

Índices em Arquivos Ordenados

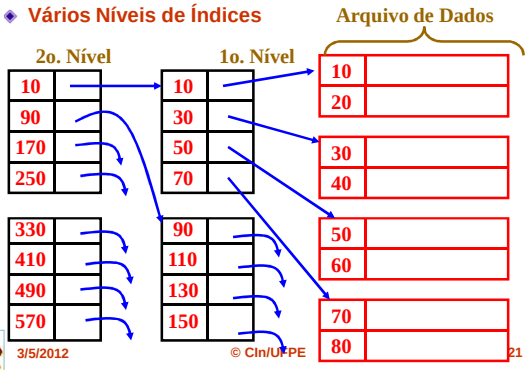
- Índices Densos x Índices Esparsos (Cont.)
  - Esparsos
    - A mesma consulta, exigiria uma operação de E/S de disco para recuperar o bloco no qual a chave K poderia ser encontrada.

Índices em Arquivos Ordenados

- Vários Níveis de Índice
  - Um único índice pode gerar vários blocos.
  - É necessário algum mecanismo para localizar blocos de índices.
    - Armazená-los em posições reservadas da memória (cilindros designados do disco).
  - Mesmo que seja possível:
    - Localizar os blocos de índice
    - Usar pesquisa binária
    - Muitas operações de E/S de disco

Índices em Arquivos Ordenados

- Vários Níveis de Índices



Índices em Arquivos Ordenados

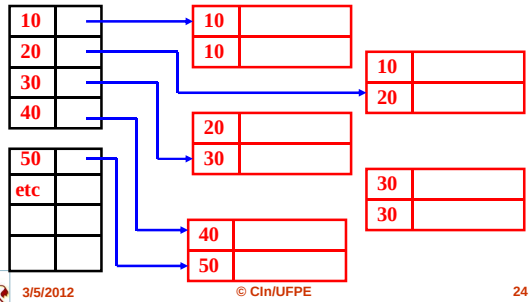
- Vários Níveis de Índice (Cont.)
  - Inserindo um índice no índice, pode tornar o uso do primeiro nível de índice mais eficiente.
    - Porém, é importante não construir muitos níveis de índices.
  - O índice do primeiro nível pode ser denso ou esparso.
    - Porém, a partir do segundo nível em diante, os índices devem ser esparsos.
    - Isto porque um índice denso do segundo nível introduz uma estrutura adicional sem oferecer qualquer vantagem.

Índices em Arquivos Ordenados

- Índices com Chaves de Pesquisa Duplicadas
  - Índices são também usados para atributos que não são de chaves da relação.
  - Uma solução simples mas não eficiente seria permitir chaves de pesquisa duplicadas no arquivo de índices.
  - Abordagens:
    - Índice Denso
    - Índice Esparso: indicando a menor chave de pesquisa em cada bloco
    - Índice Esparso: indicando a menor chave de pesquisa em cada bloco que é nova

Índices em Arquivos Ordenados

- Índices com Chaves de Pesquisa Duplicadas(Cont.)
  - Índice Denso:

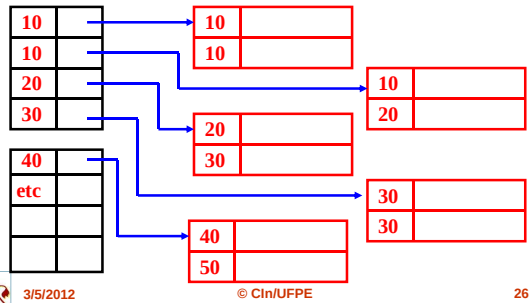


Índices em Arquivos Ordenados

- Índices com Chaves de Pesquisa Duplicadas(Cont.)
  - **Índice Denso**: Consiste em:
    - Manter uma entrada no índice denso para cada chave de pesquisa K.
    - Esta chave está associada a um ponteiro para o primeiro dos registros com K.
    - Desloca-se no arquivo de dados até encontrar todos os registros adicionais com K.
    - Este deslocamento ocorre na ordem classificada do arquivo de dados.

Índices em Arquivos Ordenados

- Índices com Chaves de Pesquisa Duplicadas(Cont.)
  - Índice Esparso para Menor Chave:

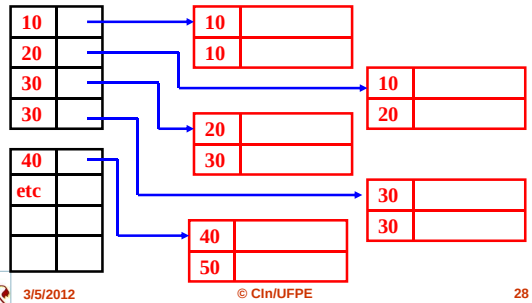


Índices em Arquivos Ordenados

- Índices com Chaves de Pesquisa Duplicadas(Cont.)
  - **Índice Esparso para Menor Chave**: Para acessar registros com chave de pesquisa = K:
    - Localiza-se a última entrada (E_1) do índice cujo valor da chave ≤ K:
    - Desloca-se em direção ao início do índice até alcançar:
      - primeira entrada ou
      - uma entrada (E_2) com chave < K
    - Todos os blocos de dados tendo registro com chave = K podem ser acessados pelas [E_2,E_1].

Índices em Arquivos Ordenados

- Índices com Chaves de Pesquisa Duplicadas(Cont.)
  - Índice Esparso para Menor Chave Nova:



Índices em Arquivos Ordenados

- Índices com Chaves de Pesquisa Duplicadas(Cont.)
  - **Índice Esparso para Menor Chave Nova**:
    - Cada entrada de índice para um bloco de dados contém a menor chave de pesquisa que é *nova*.
      - Nova: seu valor não existe em blocos anteriores.
    - Se não há nenhuma chave de pesquisa nova em um bloco, então sua entrada de índice contém a única chave de pesquisa encontrada neste bloco.

Índices em Arquivos Ordenados

- **Índice Esparso para Menor Chave Nova (Cont.)**:

Para acessar registros com chave de pesquisa = K:

- Examina-se o índice para a primeira entrada cuja chave é:
  - Igual a K ou
  - Menor que K, mas a próxima chave é > K.
- Segue-se o ponteiro dessa entrada e se pelo menos um registro com chave = K for encontrado no bloco, então:
  - segue-se adiante, varrendo blocos adicionais até encontrar todos os registros com chave = K.

Índices em Arquivos Ordenados

- Gerenciamento de Índices durante Modificações:
  - Alteração de um Índice pode criar os mesmos tipos de problemas no arquivo de Índices que inserções e exclusões criam no arquivo de dados.
  - Mesmas técnicas usadas para reorganizar arquivo de dados podem ser usadas para Índices.
    - Blocos de Estouro:
      - São criados se espaço extra é preciso.
      - São eliminados se um número suficiente de registros são excluídos.
      - Não têm entradas em um Índice Esparso.
        - São considerados extensões de seu bloco primário.

Índices em Arquivos Ordenados

- Novos Blocos:
  - Se novos blocos de dados são inseridos, então é preciso criar suas entradas no Índice Esparso.
  - Se novos blocos de índices são criados então devem ser localizados de alguma forma (ex. outro nível de índice).
- Blocos Adjacentes:
  - Se não há espaço para inserir registro(s) no bloco, então alguns deles podem ser transferidos para blocos adjacentes.
  - Se blocos adjacentes estiverem quase vazios, então eles podem ser combinados.

Índices em Arquivos Ordenados

- Observações:
  - Apenas blocos vazios podem ser criados ou destruídos.
  - Criar ou destruir bloco(s) de estouro vazio não tem efeito sobre nenhum tipo de índice:
    - Denso: se refere apenas a registros.
    - Esparso:apenas possui entradas para blocos primários.
  - Criar ou destruir bloco(s) de dados não tem efeito sobre índices densos:
    - Esparso:é afetado, pois deve-se inserir/eliminar uma entrada de índice para o bloco criado/destruído.

Índices em Arquivos Ordenados

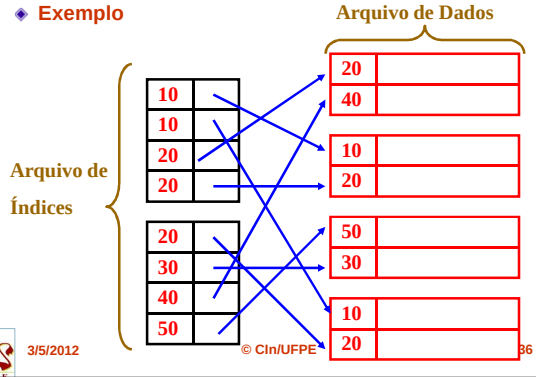
- Observações:
  - Inserir ou Eliminar registros resulta na mesma ação sobre um Índice Denso.
    - Denso: pares chave-ponteiro para os registros são inseridos ou eliminados.
    - Esparso: não há efeito.
      - Exceção: quando o registro é o primeiro de seu bloco. Neste caso, o valor de chave correspondente no índice esparso deve ser atualizado.
  - Deslizar um registro (dentro de um bloco ou entre blocos) resulta na mesma ação do caso anterior para ambos tipos de índice.

Índices Secundários

- Um arquivo de dados pode ter vários índices para facilitar uma variedade de consultas.
- Um índice sobre uma chave de pesquisa K pode ser criado até mesmo se o arquivo de dados não estiver classificado por K.
- Serve para o propósito de qualquer índice.
  - Estrutura de dados que facilita a localização de registros, dado um valor para um ou mais campos.
- Consiste em pares chave-ponteiro:
  - chave de pesquisa que não precisa ser exclusiva.
  - pares no arquivo de índices são classificados pelo valor da chave, a fim de otimizar a busca.

Índices Secundários

- Exemplo



Índices Secundários

- Dados não são classificados pela chave de pesquisa (mas chaves no arquivo de índices são).
- Ponteiros em um bloco de índices podem ir para muitos blocos de dados diferentes, em vez de irem para um ou mais blocos sucessivos.
- Pode resultar em um número maior de operações de E/S do que haveria para recuperar os mesmos registros, se fosse usado um índice primário.
- Ordem dos registros nos blocos de dados não pode ser controlada.
  - Estão presumivelmente ordenados de acordo com algum(ns) outro(s) atributo(s).

Índices Secundários

- É um índice denso, normalmente com duplicatas.
- Se um segundo nível de índice for inserido na estrutura de dados, então ele deve ser esparso.
- Primários *versus* Secundários
  - Primários: determinam a localização de registros indexados.
  - Secundários:
    - Não determinam a colocação de registros no arquivo de dados.
    - Informam sobre as localizações atuais de registros.
      - Localizações podem ter sido decididas por um índice primário sobre algum outro campo.

Índices Secundários

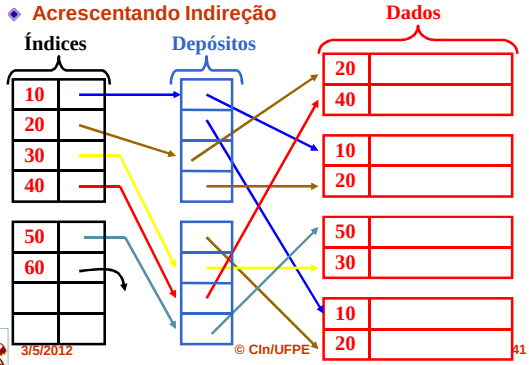
- Primários *versus* Secundários (Cont.)
  - Conseqüência:
    - Não faz sentido falar em um índice esparso secundário.
      - Índice secundário não pode ser usado para prever a localização de registros cujas chaves não estejam explícitas no arquivo de índices.
    - Desse modo, índices secundários são sempre densos.

Índices Secundários

- Indireção em Índices Secundários
  - Se um valor de chave de pesquisa aparece N vezes no arquivo de dados, então o valor é escrito N vezes no arquivo de índices.
    - Pode haver desperdício de espaço.
    - Seria melhor manter apenas um único valor da chave para todos os ponteiros para registros de dados com este valor.
  - Pode-se evitar a repetição de valores usando um nível de indireção (depósito) entre o arquivo de índices secundário e o arquivo de dados.

Índices Secundários

- Acrescentando Indireção



Índices Secundários

- Indireção em Índices Secundários (Cont.)
  - Existe um par chave-ponteiro para cada chave de pesquisa K:
    - ponteiro: aponta para uma posição no arquivo de depósito, o qual contém o depósito correspondente a K.
    - Seguindo esta posição, até a próxima posição apontada pelo índice, encontram-se ponteiros para todos os registros com chave = K.
  - Este esquema poupa espaço, desde que os valores da chave de pesquisa sejam maiores que os ponteiros.
  - Entretanto....

Índices Secundários

- Mesmo quando chaves e ponteiros têm um tamanho comparável, há uma vantagem importante no uso da indireção:
  - pode-se usar os ponteiros nos depósitos para responder a consultas sem recuperar a maioria dos registros de dados.
  - quando há várias condições para uma consulta e cada condição tem um índice secundário, pode-se:
    - obter apenas os registros cujos ponteiros de depósitos satisfazem a todas condições (interseção de conjunto de ponteiros).
    - poupar custo de E/S para recuperar registros que satisfazem a apenas algumas condições.

Índices Secundários

- Exemplo:
  - Considere a relação:

ALUNO

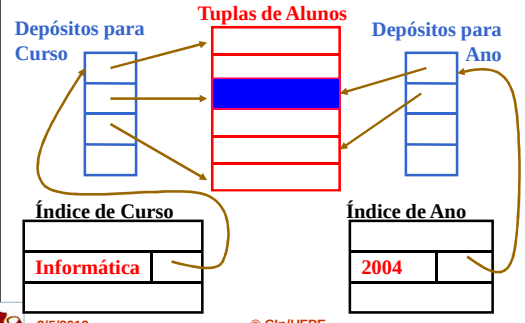
Matrícula	Nome	Ano	Curso

  - Suponha que:
    - existem índices secundários com depósitos indiretos sobre Curso e Ano.
    - consulta abaixo foi submetida:

```
SELECT  Matricula, Nome
FROM  Aluno
WHERE Curso  = 'Informatica' AND Ano = 2004
```


Índices Secundários

- Exemplo (Cont.)



Índices Secundários

- Aplicações:
  - Existem estruturas de dados que necessitam de índices secundários.
    - Arquivo de Clusters: duas ou mais relações são armazenadas com seus registros misturados.
    - Exemplo:

FILME

Nome	Ano	Duração	NomeEstúdio

ESTÚDIO

Nome	Endereço	Diretor

Índices Secundários

```
SELECT  Nome, Ano
FROM  Filme
WHERE NomeEstúdio  = 'Disney'
```

- Se a consulta acima é freqüente, então pode-se ordenar as tuplas pelo nome do estúdio (Índice Primário com Duplicatas).
- Porém, tal classificação não otimiza operações de junção.
- Se junções são freqüentes, então pode-se torná-las eficientes, usando estrutura de arquivos em clusters.

Índices Secundários

- estrutura de arquivos em clusters:

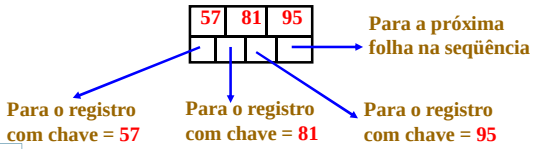
- Aumenta as chances de localizar registros para um dado estúdio e filme no mesmo bloco.
- Porém, precisa-se ainda localizar o filme ou estúdio específico de forma eficiente.
- Desse modo, precisa-se de dois índices secundários:
  - sobre nome de Filme
  - sobre nome de Estúdio

Árvores B

- Organiza blocos de dados em uma árvore.
- Família de estruturas de dados comumente usada em sistemas comerciais.
  - Variante usada com maior freqüência: Árvore B+
- Mantém automaticamente tantos níveis de índices quantos sejam apropriados para o tamanho do arquivo sendo indexado.
- Gerencia o espaço dos blocos usados
  - cada bloco fica entre meio cheio e completamente cheio durante todo o tempo.
  - nenhum bloco de estouro é necessário.
- Árvore é equilibrada: todos os caminhos desde a raiz até uma folha têm o mesmo comprimento.

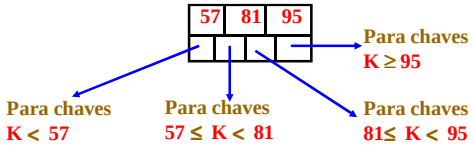
Árvores B

- Pode ter qualquer número de camadas, mas no geral:
  - a raiz
  - uma camada intermediária
  - as folhas
- Uma folha típica de uma árvore B+:



Árvores B

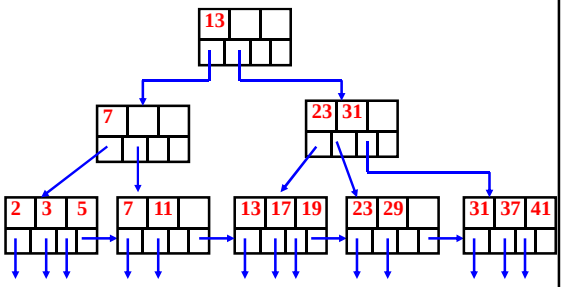
- Um nó interior típico de uma árvore B+:



- Cada árvore B tem um parâmetro N:
  - Determina o layout de todos os seus blocos.
  - Cada bloco tem espaço para N valores da chave de pesquisa e (N + 1) ponteiros.
  - Cada bloco tem N pares chave-ponteiro e um ponteiro extra.

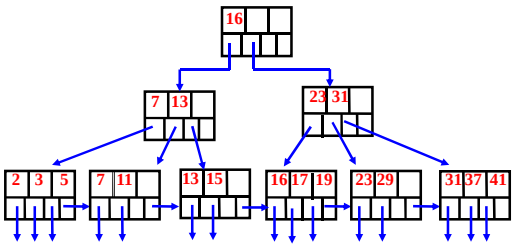
Árvores B

- Uma árvore B+:



Árvores B

- Uma árvore B+:



Árvores B

- Existem várias regras que limitam o que pode aparecer nos blocos de uma árvore B:
  - Na raiz:
    - No geral, existem pelo menos dois ponteiros.
    - Todos os ponteiros indicam blocos da árvore no nível abaixo.
  - Em uma folha:
    - Último ponteiro aponta para o próximo bloco de folhas à direita (com chaves mais altas).
    - Pelo menos (N + 1) / 2 dos ponteiros são usados e apontam para registros de dados
    - I-ésimo ponteiro (se usado) aponta para um registro com i-ésima chave.

Árvores B

- Em uma folha (Cont.):
  - Apenas um ponteiro pode ser usado para acessar outro bloco no mesmo nível.
- Em um nó interior:
  - Em cada bloco, N+1 ponteiros podem ser usados para acessar blocos no próximo nível mais baixo.
    - Pelo menos (N + 1) / 2 deles são de fato usados (exceto para raiz).
- É permito que a raiz tenha apenas uma chave e dois ponteiros para nós filhos.
- Blocos com N chaves e (N+1) ponteiros são organizados em uma estrutura de árvore, com as folhas apontando para registros de dados.

Árvores B

- Pesquisa em árvores B:
  - Busca começa na raiz e termina em uma folha.
  - Procedimento de busca para localizar um registro com chave = K:
    - BASE: Estando em um nó folha:
      - Examina-se as chaves desta folha.
      - Se i-ésima chave = K, segue-se o i-ésimo ponteiro para acessar o registro desejado.
    - INDUÇÃO: Estando em um nó interior:
      - Se N ponteiros são usados, então existirão (N-1) chaves (K_1, K_2, ..., $K_{(N-1)}$).

Árvores B

- Pesquisa em árvores B:
  - INDUÇÃO: (Cont.)
    - Se $K < K_1$, o primeiro nó filho deve ser examinado, se $K_1 \leq K < K_2$, o segundo filho deve ser examinado e assim por diante.
    - Aplica-se recursivamente o procedimento de busca a este nó filho.

Árvores B

- Consultas de Intervalos em árvores B:
  - Solicitações de registros cujo valor da chave de pesquisa reside em um dado intervalo.
  - Tais consultas são facilitadas por índices de árvores B.
  - Procedimento:
    - Para localizar todas as chaves do intervalo [a,b] nas folhas de uma árvore B, faz-se inicialmente uma busca para localizar chave = a
    - Segue-se a busca por folhas cujas chaves ≥ a.
    - Cada uma das chaves encontradas tem um ponteiro para um registro cuja chave pertence ao intervalo desejado.

Árvores B

- Consultas de Intervalos em árvores B: (Cont.)
  - Procedimento:
    - Se em uma dada folha, não for encontrada uma chave > b, segue-se o ponteiro na folha atual para a próxima folha, continuando com este processo de verificação de chaves e ponteiros associados até que:
      - uma chave > b seja localizada (condição de parada) ou
      - final da folha seja alcançado e neste caso, segue-se para a folha seguinte e repete-se o procedimento.

Árvores B

- Inserção em árvores B:
  - Trata-se de um processo recursivo.
  - Tenta-se encontrar um lugar para a nova chave na folha apropriada.
  - Se houver espaço, é feita a inserção da chave na folha encontrada.
  - Se não houver espaço na própria folha, desmembra-se a folha em duas e divide-se as chaves entre os dois novos nós.
    - Cada novo nó estará ocupado até a metade ou um pouco acima da metade.

Árvores B

- **Inserção em árvores B: (Cont.)**
  - Divisão de nós em um nível repercute no nível acima como se um novo par chave-ponteiro precisasse ser inserido no nível mais alto.
    - Aplica-se recursivamente esta estratégia.
  - Se não houver espaço ao fazer uma inserção na raiz, então:
    - Divide-se a raiz em dois nós.
    - Cria-se uma nova raiz no próximo nível mais alto.
      - Nova raiz terá os dois nós resultantes da divisão como seus filhos.

Árvores B

- **Eliminação em árvores B:** Para remover um registro com uma dada chave K, deve-se:
  - Localizar o registro e seu par chave-ponteiro em uma folha da árvore B.
  - Eliminar o próprio registro do arquivo de dados e o par chave-ponteiro da árvore B.
  - Se o nó onde ocorreu a remoção ainda tiver pelo menos, o número mínimo de chaves e ponteiros, então o processo é encerrado.
  - Porém, se a restrição sobre o número mínimo de chaves por bloco for violada, deve-se realizar :
    - Balanceamento de nós adjacentes
    - Junção de nós adjacentes

Árvores B

- **Eliminação em árvores B: (Cont.)**

Para um nó N cujo conteúdo está abaixo do mínimo:

- Balanceamento de nós adjacentes
  - Se um dos irmãos adjacentes do nó N tiver um número de chaves e ponteiros maior que o número mínimo:
    - par chave-ponteiro pode ser movido para N, mantendo-se a mesma ordem das chaves.
    - possivelmente, chaves no pai de N devam ser ajustadas para refletir a nova situação
      - menor chave do nó adjacente movida para N

Árvores B

Caso difícil ocorre quando nenhum nó adjacente pode ser usado para fornecer uma chave extra a N.

- Junção de nós adjacentes
  - Tem-se dois nós adjacentes:
    - um com uma quantidade de chaves abaixo do mínimo permitido (N).
    - com o número mínimo de chaves (M).
    - juntos não terão mais chaves e ponteiros do que é permitido haver em um único nó.
  - Mescla-se os dois nós, eliminando um deles.
  - Ajusta-se as chaves no pai e elimina-se um par chave-ponteiro.
  - Checa-se se o nó pai ainda está cheio o bastante (aplicação recursiva do algoritmo).

Árvores B

- **Eficiência de árvores B:**
  - São estruturas de índices de vários níveis com recursos de crescimento eficientes.
  - Permitem consulta, inserção, e exclusão de registros com poucas operações de E/S por operação de arquivo.
  - Se o número de chaves por bloco (N) é grande (≥10), divisões/mesclagens de blocos são raras.
    - No geral, tais operações afetam apenas duas folhas e o seu nó pai
  - Porém.....

Árvores B

- **Eficiência de árvores B:**
  - Toda busca por um registro com uma dada chave, exige que se vá da raiz até uma dada folha.

$$\text{Número de Operações E/S} = \begin{cases} \text{Número de níveis da árvore B} \\ + \\ \text{Número de blocos do arquivo de dados manipulados} \end{cases}$$

  - Para os tamanhos típicos de chaves, ponteiros e blocos, três níveis de árvore B são suficientes para um BD de médio porte.
  - Pode-se usar menos de três operações de E/S por consulta através da árvore B.

Árvores B

- Para reduzir o número de operações de E/S:
  - Manter o bloco raiz de uma árvore B permanentemente bufferizado na memória principal.
    - Toda consulta realizada através de uma árvore B de 3 níveis exigirá apenas 2 leituras de disco.
  - Em algumas situações, pode fazer sentido manter também os nós do 2o. nível da árvore B bufferizados na memória.
    - Reduz a consulta da árvore B a uma única operação, além das operações que sejam necessárias para manipulação dos blocos do próprio arquivo de dados.

Árvores B

- Aplicações: Ferramenta poderosa para construção de índices. Exemplos:
  - Chave de pesquisa da árvore B é a chave primária do arquivo de dados.
  - Índice é denso.
  - Arquivo de dados pode ou não estar classificado pela chave primária.
  - Arquivo de dados está ordenado pela sua chave primária.
  - Árvore B é um índice esparso com par chave-ponteiro em uma folha para cada bloco do arquivo de dados.

Árvores B

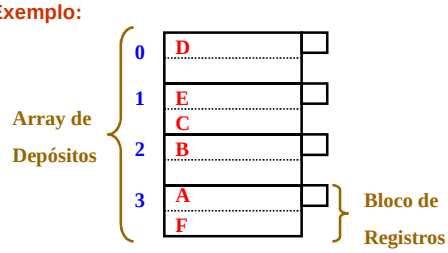
- Aplicações: (Cont.)
  - Arquivo de dados está classificado por um atributo não chave.
  - Este atributo é a chave de pesquisa para a árvore B.
  - Para cada chave de pesquisa K que aparece no arquivo de dados existe um par chave-ponteiro em uma folha.
  - Ponteiro conduz ao primeiro dos registros que tem K como seu valor de classificação.

Tabelas de Hash

- Há uma função h (chamada função de hash) que:
  - Toma uma chave de pesquisa (chamada chave de hash) como argumento.
  - Calcula um inteiro no intervalo de 0 a (B-1).
    - B é o total número de depósitos.
- Há um array de depósitos (indexado de 0 a (B-1)) que consiste em um array de cadeias de blocos.
- Se um registro tem a chave de pesquisa K, então ele é vinculado ao depósito de número h(K).
- Registros são inseridos no bloco pertencente ao seu depósito o qual é identificado pela função hash h(K).
- Se um bloco do depósito estourar, uma cadeia de blocos de estouro pode ser adicionada ao depósito.

Tabelas de Hash

- Exemplo:



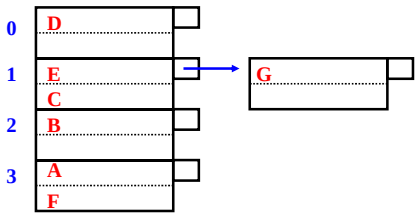
- Cada bloco possui informações adicionais no seu cabeçalho.
  - São usadas para encadear blocos de estouro.

Tabelas de Hash

- Inserção em uma tabela de hash:
  - Para inserir um novo registro com chave de pesquisa K, calcula-se h(K).
  - Se o depósito com número h(K) tiver espaço:
    - Faz-se a inserção do registro no primeiro bloco deste depósito ou em um de seus blocos de estouro.
  - Se nenhum dos blocos da cadeia referente ao depósito h(K) tiver espaço:
    - Adiciona-se um novo bloco de estouro à cadeia
    - Armazena-se o novo registro no bloco criado.

Tabelas de Hash

- **Exemplo de Inserção:** Adição de um registro com chave = G e que h(G) = 1.



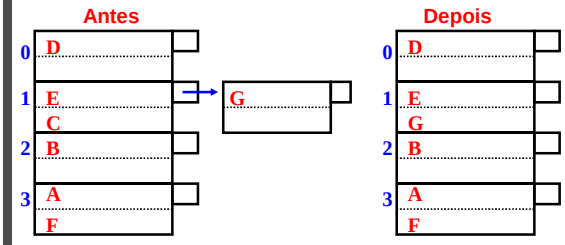
Inclusão de um bloco adicional em um depósito de uma tabela de hash

Tabelas de Hash

- **Exclusão em uma tabela de hash:**
  - Para eliminar um registro com chave de pesquisa K, calcula-se h(K) para identificação do depósito.
  - Acessa-se o depósito e realiza-se uma busca por registros com este valor de chave.
  - Todos os registros encontrados são eliminados.
  - Pode haver a necessidade de consolidação de blocos de uma cadeia em um único bloco menor.
    - Existe o risco de que ocorra a inserção e exclusão de registros de um mesmo depósito alternadamente.

Tabelas de Hash

- **Exemplo de Exclusão:** Eliminação de um registro com chave = C onde h(C) = 1.



Se o registro com K = A fosse excluído, o registro com K = F seria movido para o início do bloco.

Tabelas de Hash

- **Eficiência de índices de tabelas de hash:**
  - Deve-se tentar minimizar o número de blocos por depósito para reduzir número de operações de E/S
  - Em geral, este número é significativamente melhor do que índices esparsos ou densos diretos ou ainda índices de árvores B.
  - Porém, tabelas de hash não admitem consultas de intervalos.
  - Podem ser:
    - Estáticas: o numero de depósitos (B) é fixo.
    - Dinâmicas: B pode variar, sendo igual ao número de registros dividido pelo número de registros que podem caber em um bloco.

Tabelas de Hash

- **Escolha da função de hash:**
  - Mapeia valores da chave de pesquisa em depósitos, particionando os registros em muitos grupos pequenos.
  - Deve ser fácil de calcular.
  - No geral, quando as chaves são inteiros:
    - Calcula-se o resto de K / B
    - Com freqüência B é número primo
  - Quando as chaves são strings de caracteres:
    - Trata-se cada caracter como um inteiro.
    - Soma-se os valores inteiros.
    - Calcula-se o resto da Soma / B