INDEXAÇÃO E HASHING

Leila Lisiane Rossi – IFC-Videira

Ordenação Arquivo

- **Arquivo Sequencial** – Se segue a ordem de alguma chave
- **Sem Ordenação** – Heap – em qualquer posição
- **Hash** – Função para definir a posição

INDEXAÇÃO - Conceito

- No contexto de estrutura de dados é uma referência associada a uma **chave** usada para fins de otimização, permitindo a localização mais rápida de um registro quando efetuada uma consulta.
- No contexto de Banco de Dados um índice é uma **estrutura auxiliar associado a uma tabela**.
- Um índice para um arquivo em um sistema de banco de dados funciona quase da mesma forma que o índice de um livro.

INDEXAÇÃO - Objetivo

- **Acelerar o tempo de acesso** às linhas de uma tabela, criando ponteiros para os dados armazenados em colunas específicas. O Banco de Dados usa o índice de maneira semelhante ao **índice remissivo** de um livro, verifica um determinado assunto no índice e depois localiza a sua posição em uma determinada página.

Importance of Indexing

Fonte: (IAM Academy, 2021)

PRINCIPAIS TIPOS DE ÍNDICES

- **Índices Ordenados** – baseados em uma ordem classificatória dos valores
- **Índices de *hash*** – baseados em uma distribuição uniforme de valores por um intervalo de *buckets*. O *bucket* ao qual um valor é atribuído é determinado por uma função – **função de *hash***

FATORES AVALIAÇÃO DA TÉCNICA

- **Tipos de Acesso** – Os tipos de acesso podem incluir a localização de registros com um **valor de atributo especificado** e a localização de registros cujos valores de atributos se encontrem em um **intervalo especificado**.
- **Tempo de Acesso –** Tempo gasto para encontrar determinado item de dados, ou conjunto de itens, usando a técnica em questão

FATORES AVALIAÇÃO DA TÉCNICA

- **Tempo de Inserção –** O tempo gasto para inserir um novo item de dados
- **Tempo de exclusão –** O tempo gasto para excluir um item de dados
- **Espaço Adicional –** O espaço adicional ocupado por uma estrutura de índice

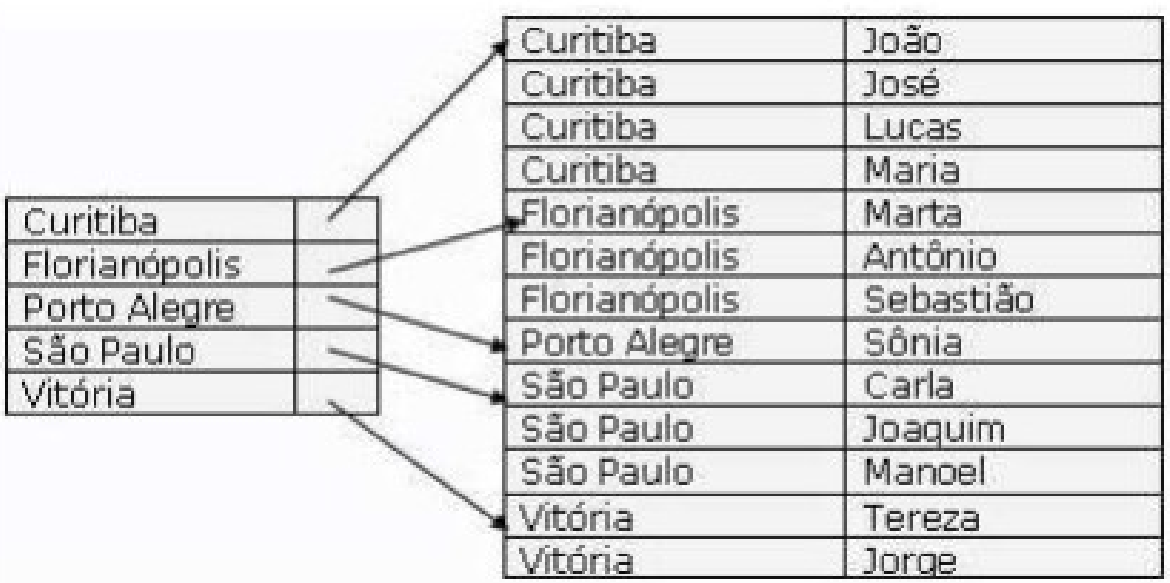
Chave de Busca

- Um atributo ou conjunto de atributos utilizados para pesquisar registros em um arquivo
- É diferente de chave primária, chave candidata e superchave

Índices Ordenados

- Para obter um acesso aleatório rápido aos registros em um arquivo, podemos usar uma estrutura de índice
  - Cada estrutura de índice está associada a uma determinada **chave de busca**
  - Um arquivo pode ter vários índices, em diferentes chaves de busca
  - Se o arquivo contendo os registros for ordenado sequencialmente, um **índice de agrupamento** é um índice cuja chave de busca também define a ordem sequencial do arquivo – **índices primários**

Índice Primário

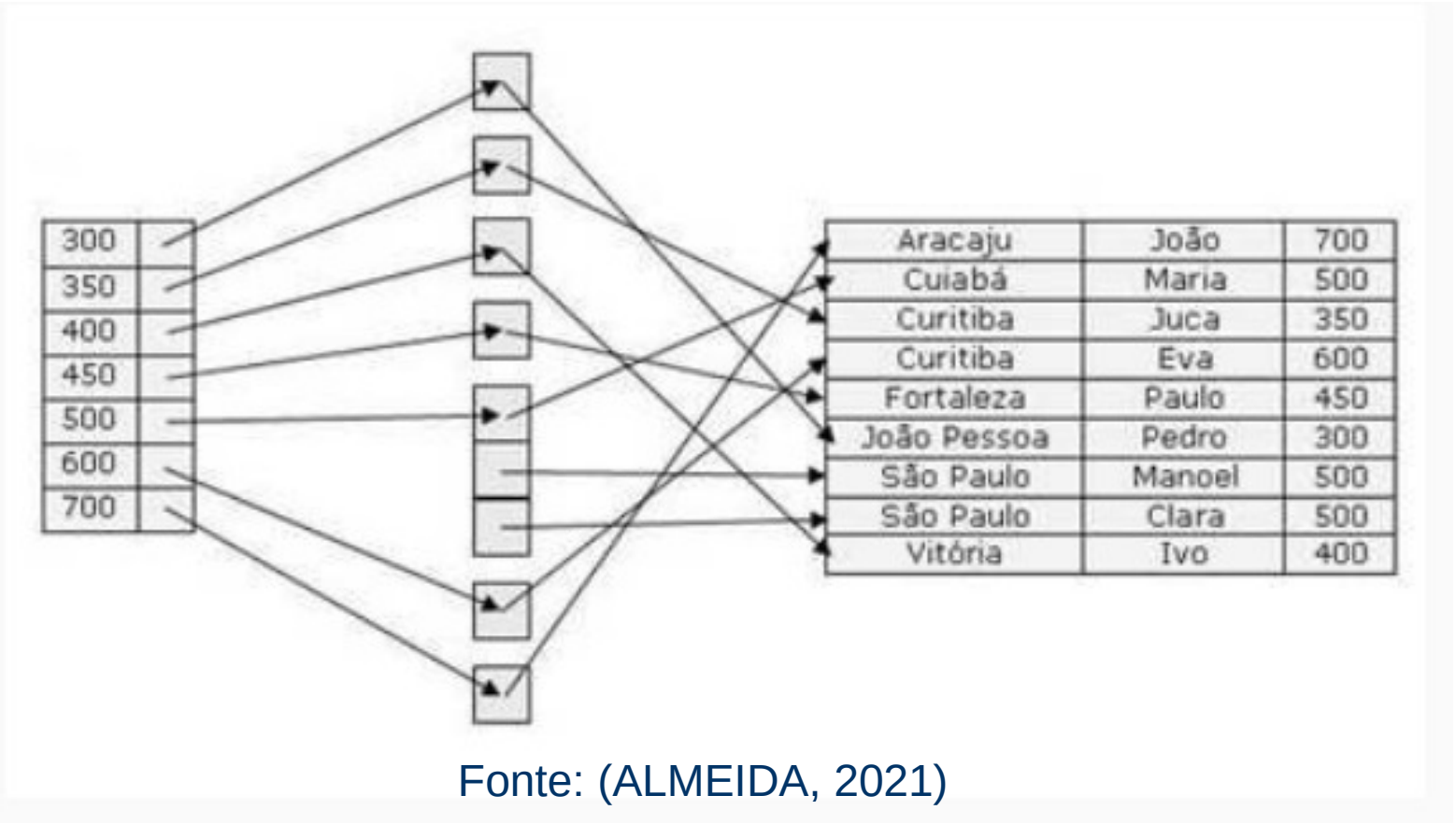


Fonte: (ALMEIDA, 2021)

Índices Ordenados

- Os índices cuja chave de busca especifica uma ordem diferente da ordem sequencial do arquivo são chamados **índices não de agrupamento**, ou índices secundários.

Índice Secundário



Fonte: (ALMEIDA, 2021)

Índices Secundários

- Os índices secundários precisam ser **densos**, com uma entrada de índice para cada valor de chave de busca, e um ponteiro para cada registro no arquivo. Um índice agrupado pode ser esparso, armazenando apenas alguns dos valores da chave de busca, pois sempre é possível encontrar registros com valores intermediários da chave de busca por um acesso sequencial a uma parte do arquivo.

Índices Secundários

- Se um índice secundário armazenar apenas algumas das chaves de busca, os registro com valores intermediários de chave de busca podem estar em qualquer lugar no arquivo e, em geral não podemos encontrá-los sem pesquisar o arquivo inteiro.
- Um índice secundário sobre uma chave candidata se parece com um índice de agrupamento denso, exceto que os registros apontados por valores sucessivos no índice não são armazenados sequencialmente. Porém, em geral, os índices secundários podem ter uma estrutura diferente dos índices de agrupamento.

Índices Secundários

- Se a chave de busca de um índice agrupado não for uma chave candidata, é suficiente que o índice aponte para o primeiro registro com um valor específico para a chave de busca, pois os outros registros podem ser apanhados por uma varredura sequencial do arquivo.
- Ao contrário, se a chave de busca de um índice secundário não for uma chave candidata, não será suficiente apontar apenas para o primeiro registro com cada valor de chave de busca.

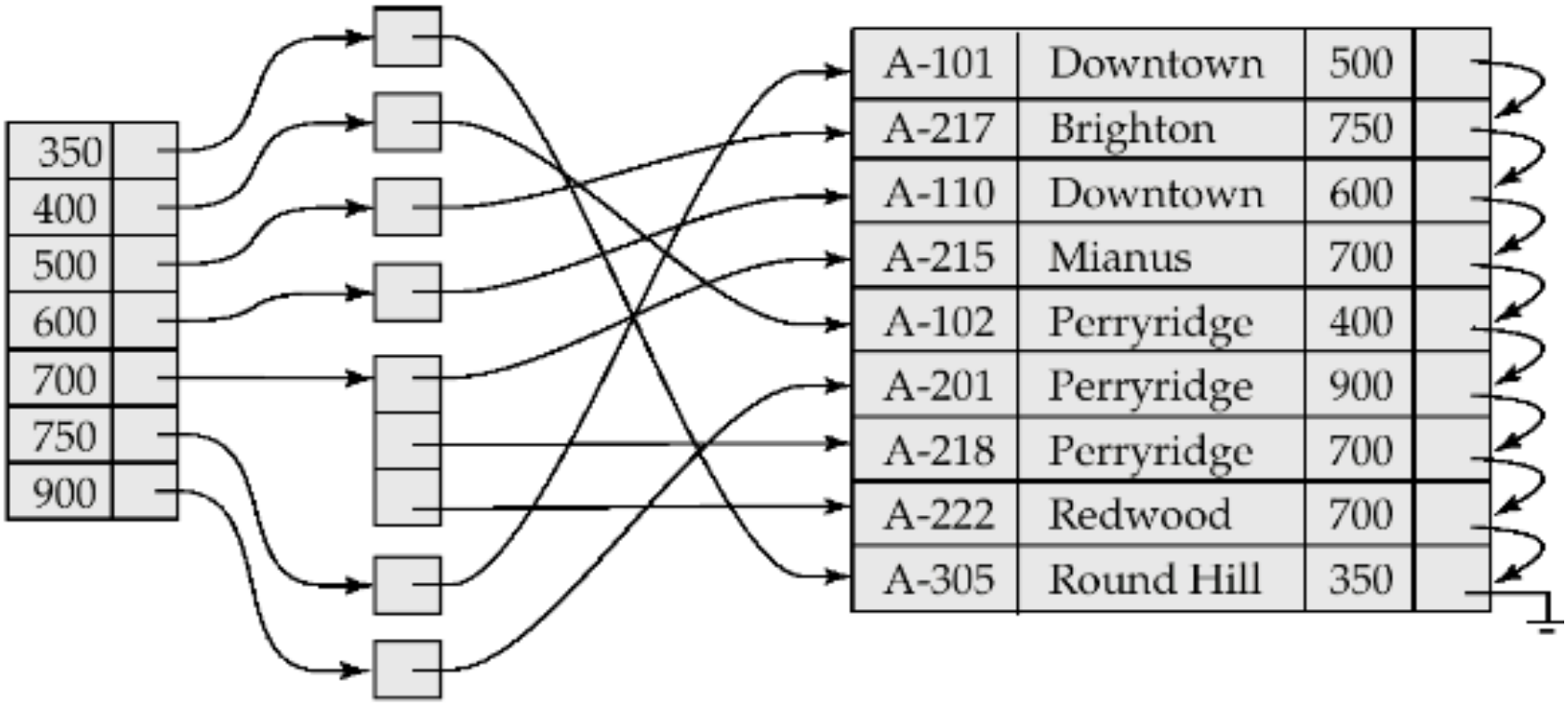
Índices Secundários

- Os registros restantes com o mesmo valor de chave de busca poderiam estar em qualquer lugar no arquivo, pois os registros são ordenados pela chave de busca do índice agrupado, e não pela chave de busca do índice secundário. **Portanto, um índice secundário precisa ter ponteiros para todos os registros.**

Índices Secundários

- Podemos usar um nível extra de indireção para implementar índices secundários sobre chaves de busca que não são chaves candidatas. Os ponteiros nesse índice secundário não apontam diretamente para o arquivo. Em vez disso, cada um aponta para um *bucket* que contém ponteiros para o arquivo.

Índices Secundários



Índices Secundários

- Uma varredura sequencial na ordem do índice agrupado é eficiente porque os registros no arquivo são armazenados fisicamente na mesma ordem do índice. Porém, não podemos (exceto em casos especiais raros) armazenar um arquivo fisicamente ordenado pela chave de busca do índice agrupado e pela chave de busca de um índice secundário. Como a ordem da chave secundária e a ordem da chave física são diferentes, se tentarmos varrer o arquivo sequencialmente na ordem da chave secundária, a leitura de cada registro provavelmente exigirá a leitura de um novo bloco do disco, o que é muito lento.

Índices Secundários

- O procedimento descrito anteriormente para exclusão e inserção também pode ser aplicado a índices secundários; as ações tomadas são aquelas descritas para índices densos armazenando um ponteiro para cada registro no arquivo. Se um arquivo tiver vários índices, sempre que o arquivo for modificado, cada índice terá de ser atualizado.

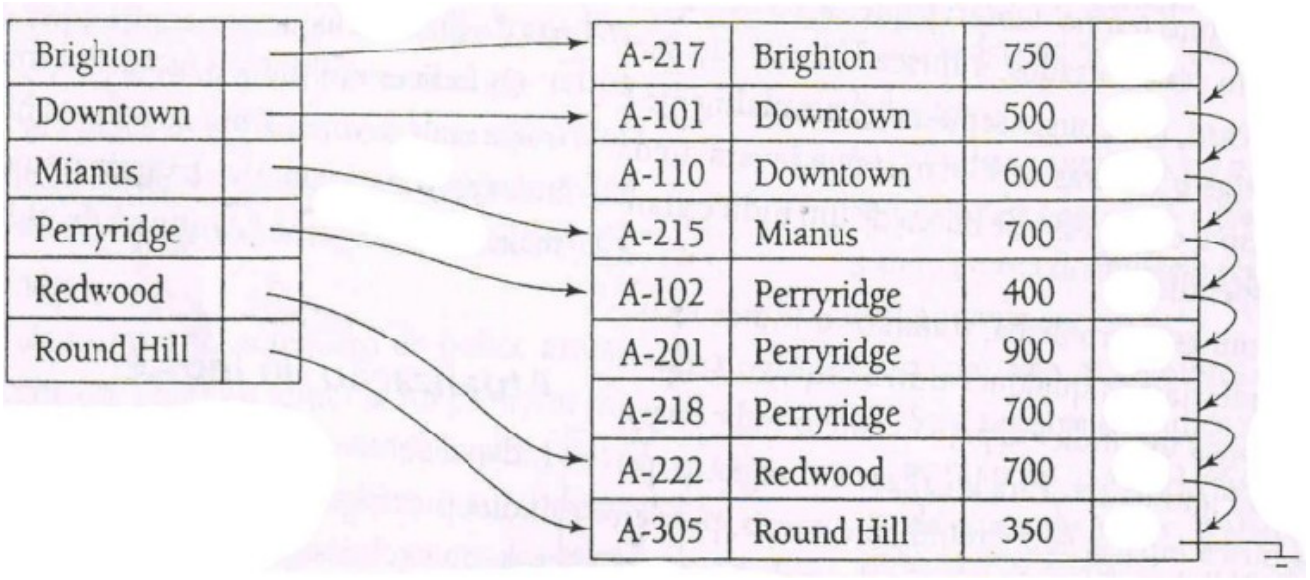
Índices Secundários

- Os índices secundários melhoram o desempenho das consultas que usam chaves diferentes da chave de busca do índice agrupado. Porém, eles impõe uma sobrecarga significativa na modificação do banco de dados.
- O projetista de um banco de dados decide quais índices secundários são desejados com base em uma estimativa da frequência relativa das consultas e modificações.

Índices densos

- **Índice denso** – Um registro de índice aparece para cada valor de chave de busca no arquivo
- Em um índice de agrupamento denso, o registro de índice contém **o valor da chave de busca** e um **ponteiro** para o primeiro registro de dados com esse valor da chave de busca

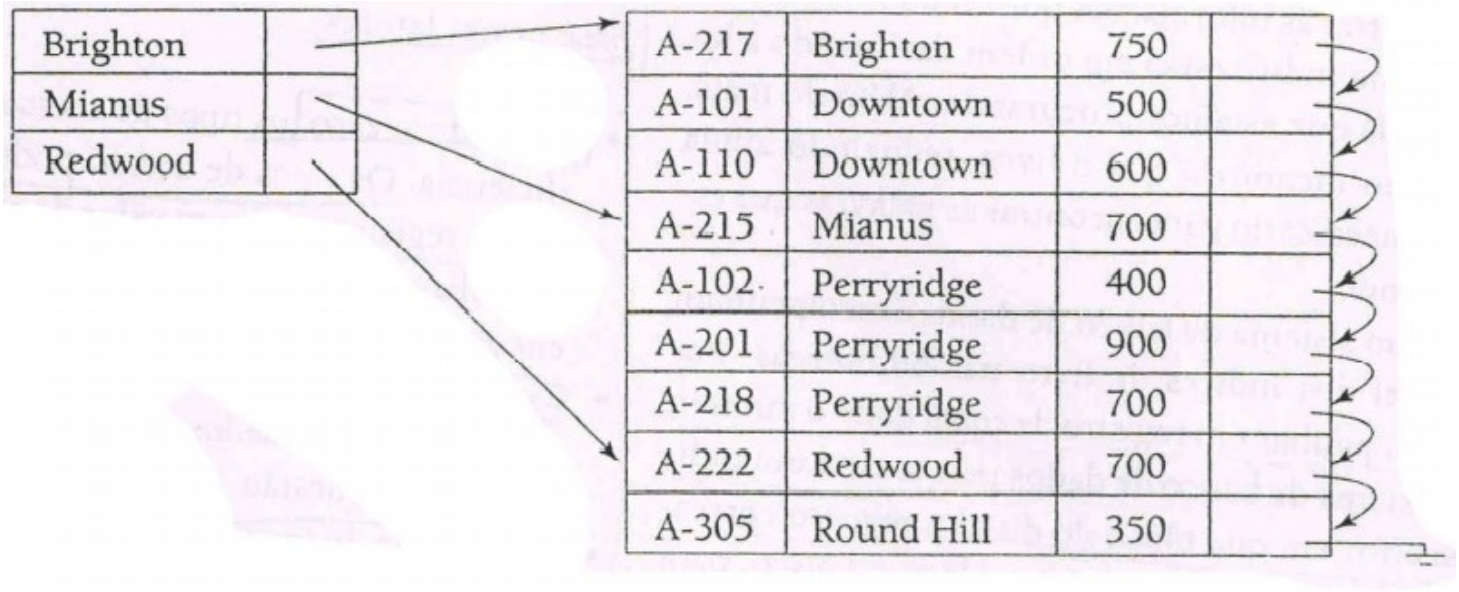
Índices densos



Índices esparsos

- **Índice esparso** – Um registro de índice aparece para somente alguns dos valores da chave de busca
- Cada registro de índice contém um valor de chave de busca e um ponteiro para o primeiro registro de dados com esse valor da chave de busca. Para localizar um registro, encontramos a entrada de índice com o maior valor de chave de busca que é menor ou igual ao valor de chave de busca pelo qual estamos procurando

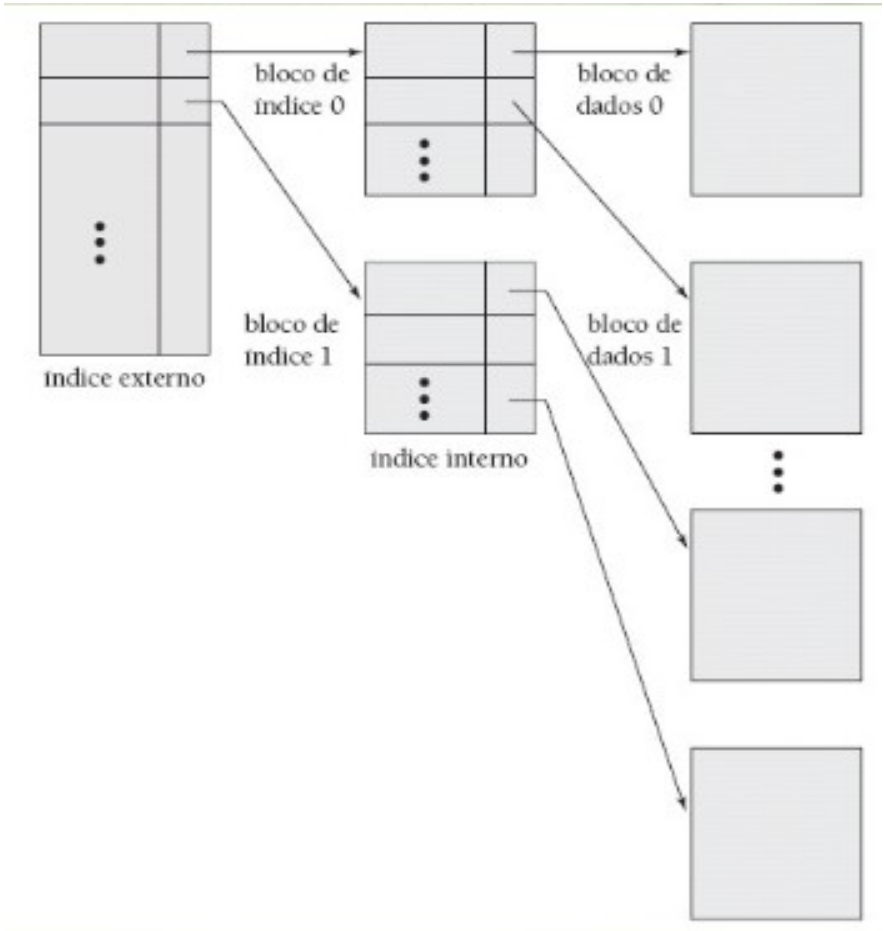
Índices esparsos



Índices multiníveis

- **Índices multiníveis** – Os índices com dois ou mais níveis são chamados de índices multiníveis
  - A busca de registros com um índice multinível exige muito menos operações de **Entrada/Saída** do que a busca binária de registros. Cada nível de índice poderia corresponder a uma unidade de armazenamento físico. Assim, podemos ter índices nos níveis de trilha, cilindro e disco.
  - Os índices multiníveis estão bastante relacionados às estruturas de árvores, como as árvores binárias usadas para a indexação na memória.
  - **Exemplo: Árvore B+**

Índices multiníveis



Atualização do índice

- Algoritmo para atualização de índices de único nível
  - **Inserção**
    - Primeiro o sistema realiza uma pesquisa usando o valor de chave de busca que aparece no registro a ser inserido. As ações que o sistema realiza em seguida dependem se o índice é denso ou esparso:
      - **Índices densos:**
        - 1. Se o valor da chave de busca não aparece no índice, o sistema insere um registro de índice com o valor de chave de busca no índice, na posição apropriada.

Atualização do índice

- Algoritmo para atualização de índices de único nível
  - **Inserção**
    - 2. Caso contrário, as seguintes ações são tomadas:
    - a) Se o registro de índice armazena ponteiros para todos os registros com o mesmo valor da chave de busca, o sistema acrescenta um ponteiro para o novo registro no registro de índice.
    - b) Caso contrário, o registro de índice armazena um ponteiro somente no primeiro registro com o valor da chave de busca. O sistema, então, coloca o registro sendo inserido após os outros com os mesmos valores de chave de busca.

Atualização do índice

- **Índices esparsos:** Consideramos que o índice armazena uma entrada para cada bloco. Se o sistema cria um novo bloco, ele insere no índice o primeiro valor de chave de busca (na ordem de chave de busca) que aparece no novo bloco. Por outro lado, se o novo registro tiver o menor valor de chave de busca em seu bloco, o sistema atualiza a entrada de índice apontando para o bloco; se não, o sistema não faz qualquer mudança no índice.

Atualização do índice

- Algoritmo para atualização de índices de único nível
  - **Exclusão** : Para excluir um registro, o sistema primeiro pesquisa o registro a ser excluído. A ações que o sistema toma em seguida dependem se o índice é denso ou esparso.
    - **Índices densos**
      - 1. Se o registro excluído foi o único registro com seu valor específico de chave de busca, então o sistema retira do índice o registro de índice correspondente.
      - 2. Caso contrário, as seguintes ações são tomadas:
      - a) Se o registro de índice armazena ponteiros para todos os registros com o mesmo valor de chave de índice, o sistema exclui do registro de índice o ponteiro para o registro excluído.
      - b) Caso contrário, o registro de índice armazena um ponteiro somente para o primeiro registro com o valor da chave de busca. Nesse caso, se o registro excluído foi o primeiro registro com o valor da chave de busca, o sistema atualiza o registro de índice para que aponte para o próximo registro.

Atualização do índice

- **Índices esparsos**
  - 1. Se o registro não tiver um registro de índice com o valor de chave de busca do registro excluído, nada precisa ser feito com o índice.
  - 2. Caso contrário, o sistema toma as seguintes ações:
  - a) Se o registro excluído foi o único registro com sua chave de busca, o sistema substitui o registro de índice correspondente por um registro de índice para o próximo valor da chave de busca (na ordem de chave de busca). Se o próximo valor de chave de busca já tiver uma entrada de índice, a entrada é excluída, em vez de ser substituída.
  - b) Caso contrário, se o registro de índice para o valor da chave de busca apontar para o registro sendo excluído, o sistema atualiza o registro de índice para que aponte para o próximo registro com o mesmo valor da chave de busca.

Arquivos de índice de árvore B+

- A principal desvantagem da organização de arquivo sequencial indexada é que o **desempenho diminui** enquanto o arquivo cresce, tanto para pesquisas de índice quanto para varreduras sequenciais pelos dados. Embora essa degradação possa ser remediada pela reorganização do arquivo, as reorganizações frequentes não são desejáveis.

Arquivos de índice de árvore B+

- A estrutura de índice de árvore B+ é a mais utilizada das várias estruturas de índice que mantêm sua **eficiência** apesar da inserção e exclusão de dados. Um índice de árvore B+ tem a forma de uma árvore balanceada, em que cada caminho da raiz da árvore até uma folha da árvore é do mesmo tamanho.
- Cada nó não-folha na árvore tem entre *n*/2 e n filhos, onde *n* é fixo para uma árvore em particular.

Arquivos de índice de árvore B+

- Veremos que a estrutura de árvore B+ impõe sobrecarga de desempenho na inserção e na exclusão, além de aumentar o espaço adicional. A sobrecarga é aceitável até mesmo para arquivos modificados com frequência, pois o custo da reorganização do arquivo é evitado. Além do mais, como os nós podem estar vazios até a metade (se tiverem o número mínimo de filhos), existe algum espaço desperdiçado. Esse espaço adicional também é aceitável dados os benefícios de desempenho da estrutura de árvore B+.

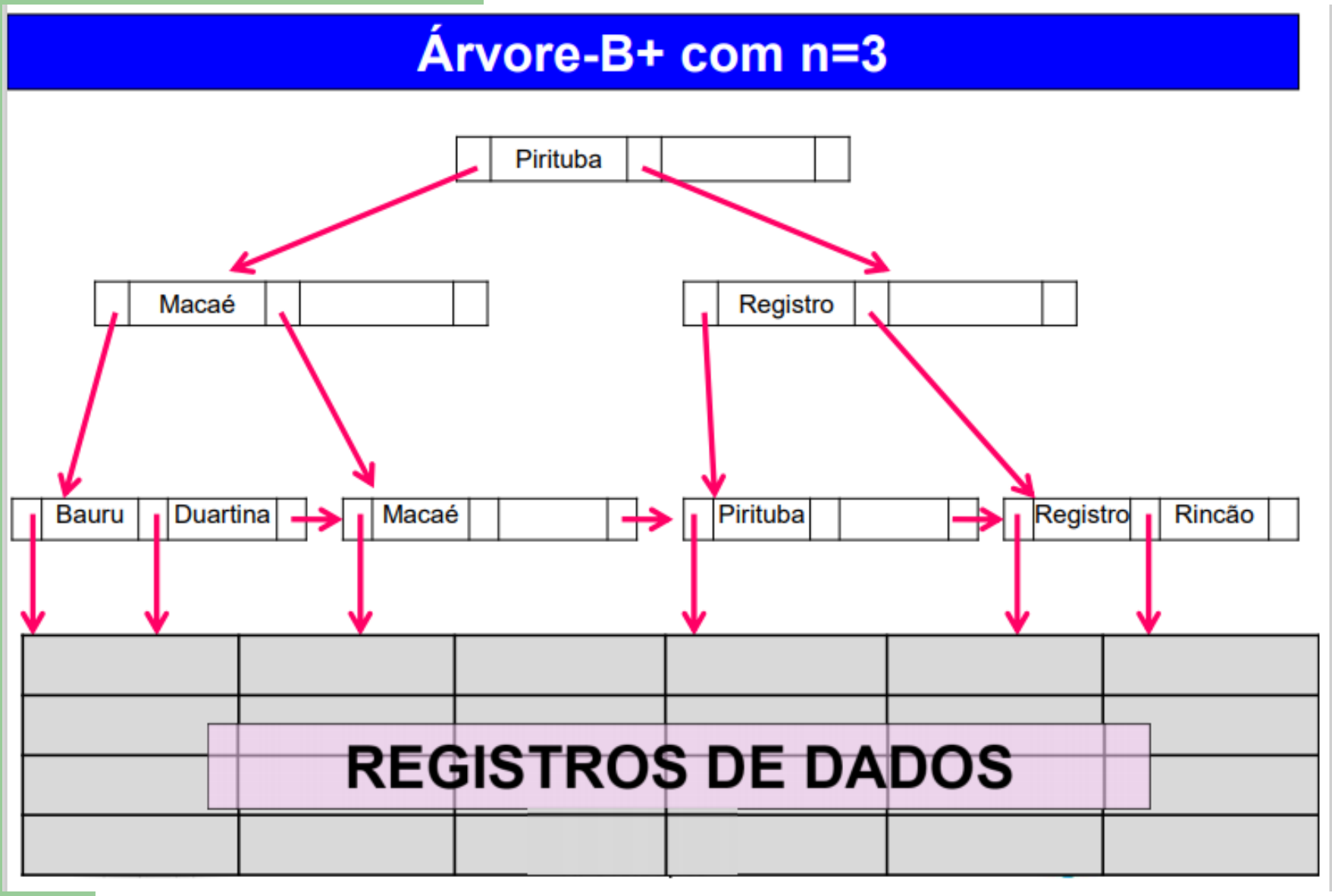
Estrutura de uma Árvore B+

- Um índice de árvore B+ é um índice de **multinível**, mas tem uma estrutura que difere daquela do arquivo sequencial indexado multinível. A Figura a seguir mostra um nó típico de uma árvore B+. Ele contém até *n*-1 valores de chave de busca *K*1, *K*2,...,*K*n-1 e *n* ponteiros *P*1,*P*2,...,*Pn*. Os valores de chave de busca dentro de um nó são mantidos em ordem classificada; assim, se *i* < *j*, então *K*i < *K*j.

Árvore B - Árvore B+

- **Árvore B**
  - Árvore de busca balanceada
  - Árvore de busca de ordem *n* -> todos os nós (exceto a raiz) tem no mínimo [(*n*-1/)2] chaves e no máximo *n*-1 chaves
  - Todos os nós internos com *k* chaves possuem *k*+1 filhos
  - Chaves não se repetem nos nós

- **Árvore B+**
  - Semelhante à Árvore B
  - Nós internos só possuem chaves – dados (registros) estão somente nos nós-folha
  - Chaves podem se repetir

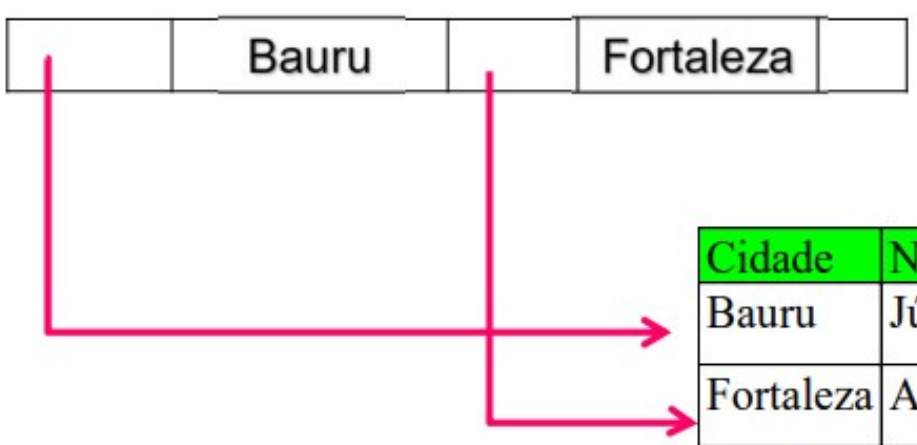


Fonte: (NUNES, 2021)

Arvore B+

- **Árvore B+**
  - mantém eficiência, mesmo com inserção e remoção de dados
  - forma de uma árvore balanceada
  - cada nó não-folha possui entre (n/2 e n filhos)
  - impõe uma sobrecarga de desempenho na inclusão e remoção aceitável porque evita o custo de reorganização
  - desperdício de espaço aceitável (existe se houver nós com mínimo de filhos) compensado pelo desempenho)

Arvore B+



Nó folha

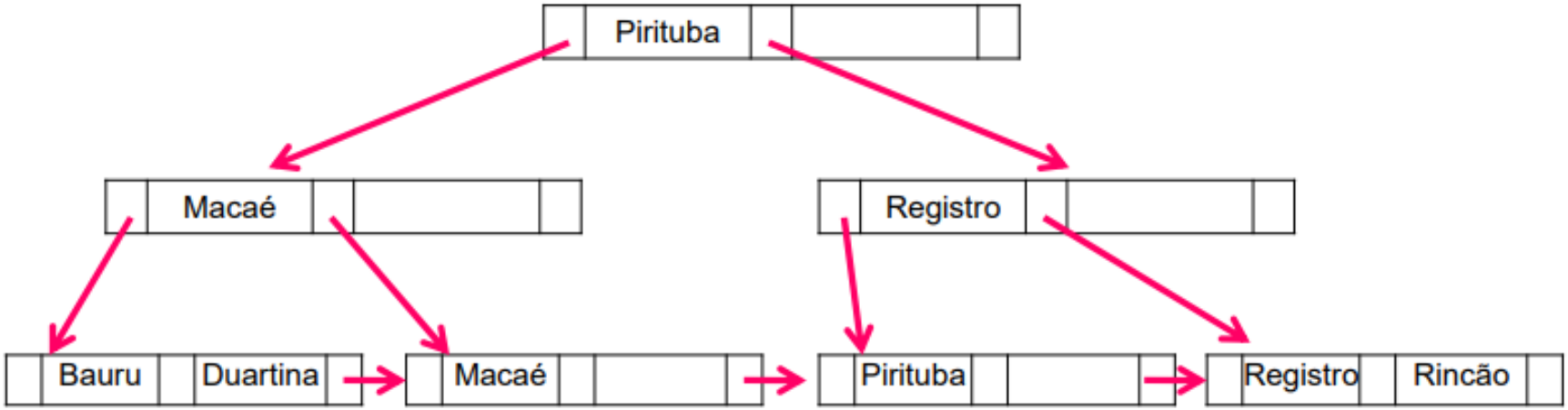
Cidade	Nome	Endereço
Bauru	Júlia	Rua dos Anjos, 123
Fortaleza	Ana	Av. Antonio, 865
Fortaleza	Bruno	Av. Antonio, 865

Fonte: (NUNES, 2021)

Arvore B+ Consultas

- Todos os registros com valor de chave de procura = k
- **Algoritmo:**
  - Examinar nó-raiz procurar menor valor que seja maior ou igual a k. Chamamos o valor encontrado de Ki
  - Seguir o ponteiro Pi até próximo nó. Se K < K1 ,seguimos P1 até outro nó
  - Se temos m ponteiros nos nós e K Km-1 seguir Pm até o outro nó Repetir até alcançar o nó folha ou o bucket

Arvore B+ Consultas



Fonte: (NUNES, 2021)

Arvore B+ Atualizações

- divisão de nó muito grande em inserção
- combinação de nós pequenos (< que n/2 ponteiros) na remoção
- garantia de balanceamento

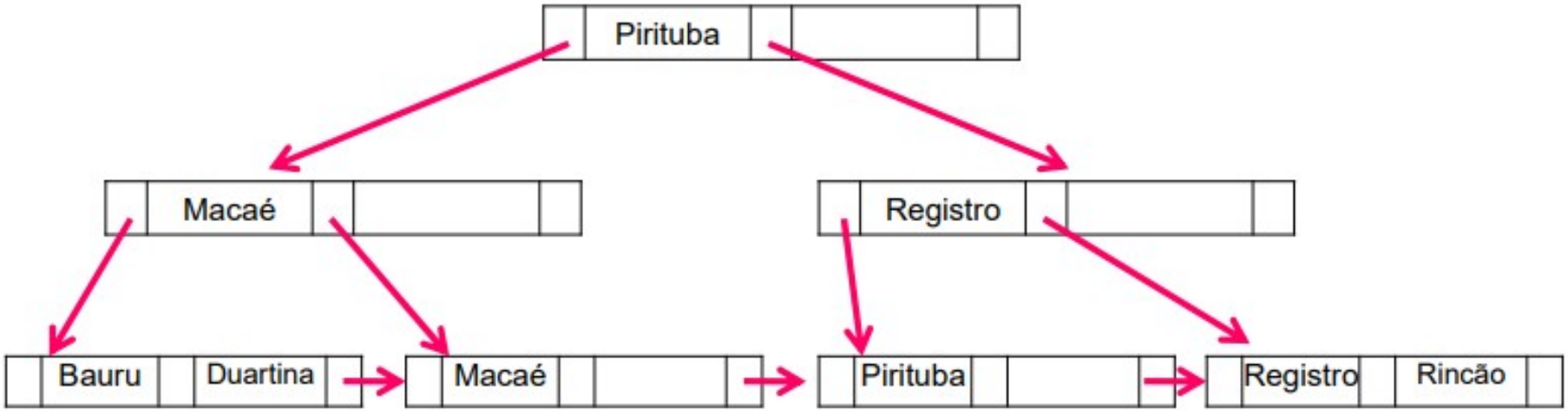
Arvore B+ Inserção sem divisão de nós

- encontrar nó folha, conforme definido na busca
- se valor procurado aparece no nó folha: adicionar novo registro ao arquivo e ponteiro no *bucket* (se necessário)
- se valor procurado não aparece no nó folha:
  - inserir valor no nó folha, mantendo a ordem das chaves de procura
  - adicionar novo registro ao arquivo e ponteiro no *bucket* (se necessário)

Arvore B+ Inserção com divisão de nós

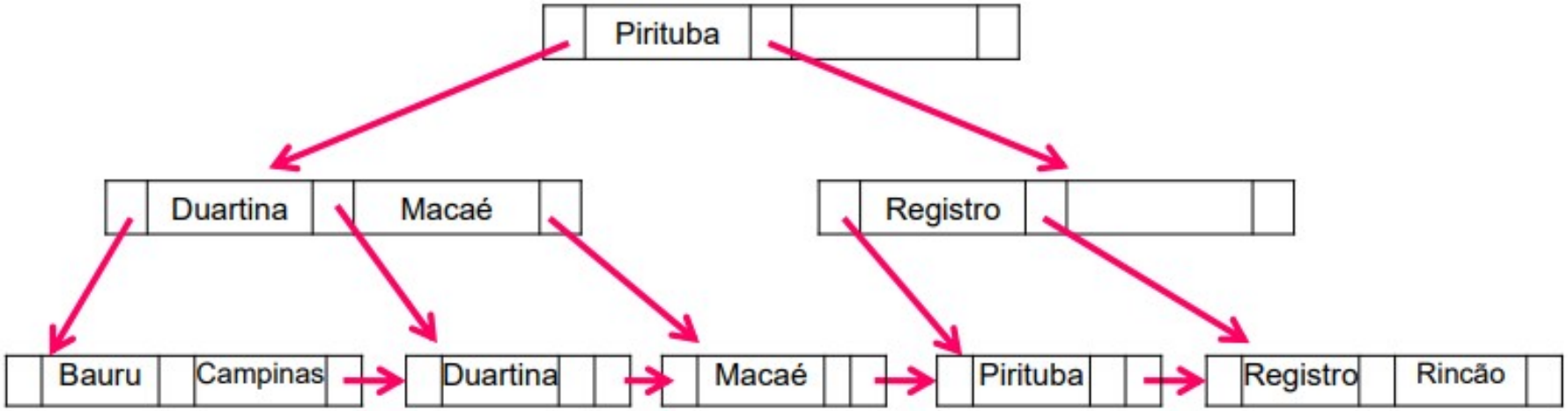
- encontrar nó folha, conforme definido na busca
- se não há espaço para inserir o valor de chave procura
  - dividir o nó em dois (considerando n-1 valores existentes + valor inserido)
    - primeiro nó (já existente): n/2 primeiros
    - segundo nó (novo): valores restantes
- Inserir o novo nó folha na estrutura da árvore
  - se não houver espaço dividir o pai
  - Pior caso: todos os nós divididos, até a própria raiz (árvore se torna mais profunda se raiz for dividida)

✓ Exemplo: inserir Campinas na árvore dada:



Fonte: (NUNES, 2021)

- ✓ Exemplo: inserir Campinas na árvore dada:



Fonte: (NUNES, 2021)

Inserção Árvore B+

https://www.youtube.com/watch?v=VATkI7GtU8Y

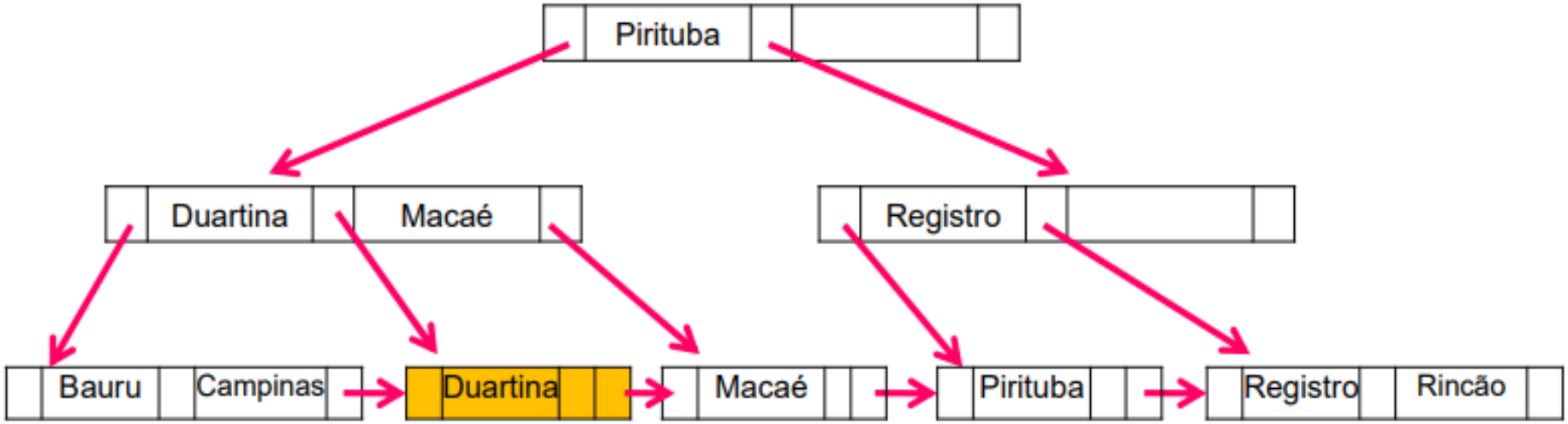
Arvore B+ Remoção sem junção de nós

- encontrar nó folha, conforme definido na busca
- Excluir registro do arquivo
- Remover valor da chave de procura do nó-folha se não houver *bucket* associado àquele valor ou se o *bucket* tornar-se vazio como resultado da remoção

Arvore B+ Remoção com junção de nós

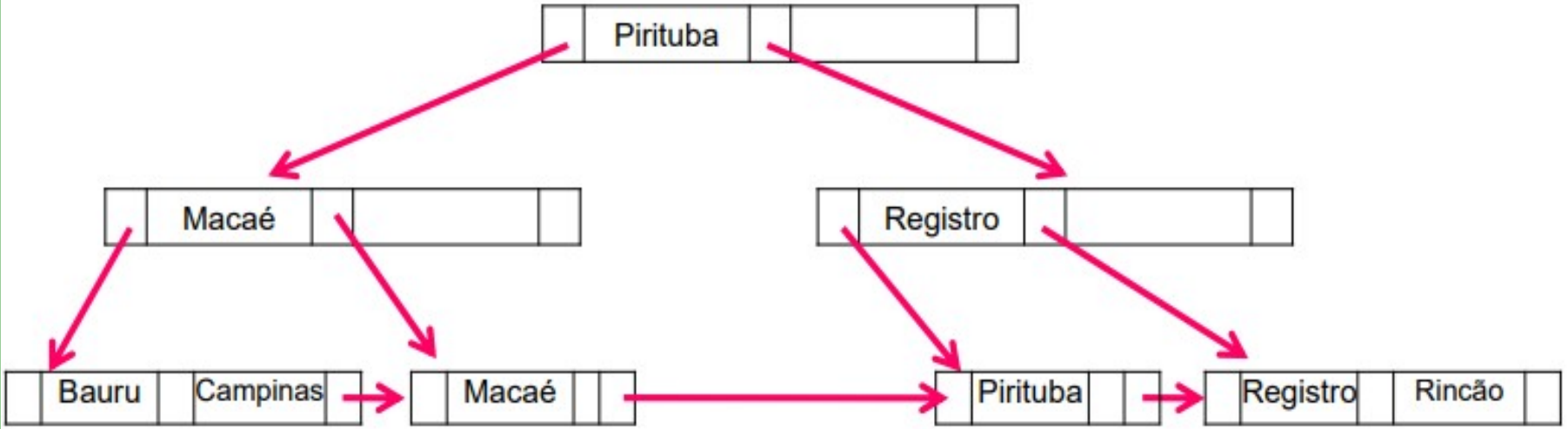
- encontrar nó folha, conforme definido na busca
- se folha fica vazia ou muito pequena, é necessário eliminar nó (lembrar regra da quantidade de nós)
  - remover do nó pai o ponteiro para o nó a ser eliminado
  - se o pai também ficar muito pequeno ou vazio
    - olhar o nó irmão (nó não folha que contém a chave de procura)
      - se nó irmão tiver espaço - fundir os nós
      - se nó irmão não tiver espaço - redistribuir os nós

- ✓ Exemplo: remover Duartina da árvore dada:



Fonte: (NUNES, 2021)

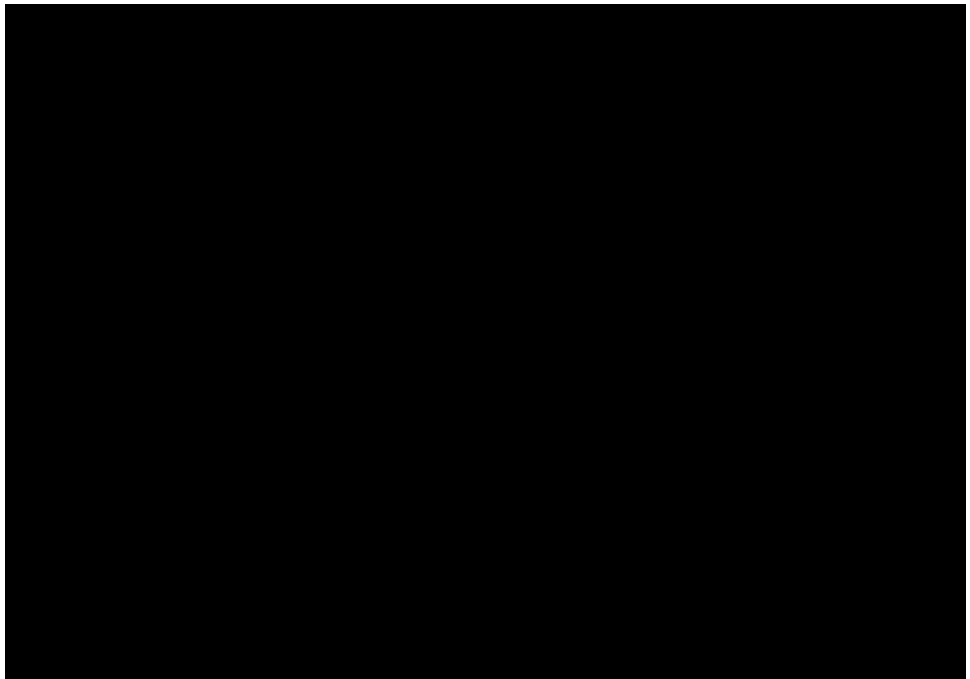
- ✓ Exemplo: remover Duartina da árvore dada:



Fonte: (NUNES, 2021)

Remoção Árvore B+

https://www.youtube.com/watch?v=_kYOntAPyCE



Organização de Arquivos de Árvores B+

- Em uma organização de árvore B+, os nós de folha da árvore armazenam **registros**, em vez de armazenar ponteiros para registros.
- Como os registros normalmente são maiores que os ponteiros, o número máximo de registros que podem ser armazenados em um nó de folha é menor que o número de ponteiros em um nó não-folha. Porém, os nós de folha ainda precisam estar cheios pelo menos até a metade.

Organização de Arquivos de Árvores B+

- A inserção e a exclusão de registros de uma organização de arquivo de árvore B+ são tratadas da mesma maneira que a inserção e a exclusão de entradas em um índice de árvore B+.

Indexando Strings

- A criação de índices de árvore B+ sobre atributos com valor de *string* ocasiona dois problemas.
  - *Strings* podem ter tamanho variável
  - *Strings* podem ser longas, levando a um baixo *fanout* e uma altura de árvore consequentemente maior

  Com as chaves de busca de tamanho variável, diferentes nós podem ter diferentes *fanouts*, mesmo que estejam cheios. Um nó precisa ser dividido se estiver cheio, ou seja, não existe espaço para acrescentar uma nova entrada, independente de quantas entradas de busca ele tenha.

Indexando Strings

- De modo semelhante, os nós podem ser mesclados ou as entradas re-distribuídas, dependendo da fração de espaço utilizada nos nós, em vez de nos basearmos no número máximo de entradas que o nó pode aceitar.
- O *fanout* dos nós pode ser aumentado usando uma técnica chamada **compactação de prefixo**. Com a compactação de prefixo, não armazenamos o valor inteiro da chave de busca nos nós internos. Só armazenamos um prefixo de cada valor de chave de busca que seja suficiente para distinguir entre os valores de chave nas subárvores que ela separa.

Indexando Strings

- Por exemplo, se tivéssemos sobre nomes, o valor de chave em um nó interno poderia ser um prefixo de um nome; pode ser suficiente armazenar “Silb” em um nó interno, em vez do nome “Silberschatz” completo, se os valores mais próximos nas duas subárvores que ela separa forem, digamos, “Silas” e “Silver”, respectivamente.

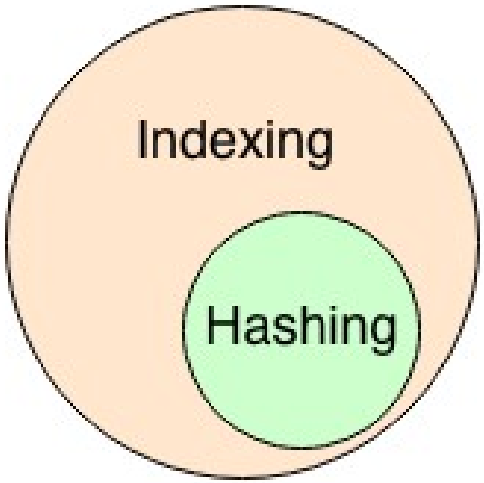
Arquivos de Índice de Árvore B

- Índices de árvore B são semelhantes aos índices de árvore B+. A principal distinção entre as duas técnicas é que uma árvore B elimina o armazenamento redundante de valores de chave de busca.
- Como as chaves de busca não são repetidas na árvore B, podemos armazenar o índice em menos nós de árvore do que no índice de árvore B+ correspondente.
- Porém, como as chaves de busca que aparecem em nós não-folha não aparecem em outro lugar na árvore B, somos forçados a incluir um campo ponteiro adicional para cada chave de busca em um nó não-folha.

Arquivos de Índice de Árvore B

- Esses ponteiros adicionais apontam para os registros de arquivo ou *buckets* para a chave de busca associada. Os nós de folha são iguais aos das árvores B+. Nos nós não-folha, os ponteiros *Pi* são os ponteiros de árvore que usamos também para árvores B+, enquanto os ponteiros *Bi* são ponteiros de *bucket* ou registro de arquivo.

HASHING



HASHING

- Indexação é um nome geral para um processo de particionamento destinado a acelerar as pesquisas de dados.
- A indexação pode particionar o conjunto de dados com base em um valor de um campo ou uma combinação de campos.
- Ele também pode particionar o conjunto de dados com base em um valor de uma função, chamada função hash, calculada a partir dos dados em um campo ou uma combinação de campos. Nesse caso específico, a indexação é chamada de hashing de dados.

HASHING

- As tabelas de *hash* são uma forma extremamente simples, fácil de se implementar e intuitiva de se organizar grandes quantidades de dados. Associa chaves de pesquisa a valores. Seu objetivo é, a partir de uma **chave simples**, fazer uma busca rápida e obter o valor desejado
- Possui como ideia central a divisão de um universo de dados a ser organizado em subconjuntos mais gerenciáveis.

HASHING

- **Conceitos Fundamentais das Tabelas de *Hash***
  - **Tabela de *Hash*:** Estrutura que permite o acesso aos subconjuntos
  - **Função de *Hashing*:** Função que realiza um mapeamento entre valores de chaves e entradas na tabela

FUNÇÃO DE *HASH*

- A função de *hash* ideal distribui as chaves armazenadas uniformemente por todos os *buckets*, de modo que cada um tenha o mesmo número de registros.
  - A distribuição é uniforme – a função de *hash* atribui a cada *bucket* o mesmo número de valores de chave de busca do conjunto de todos os valores de chave de busca possíveis.
  - A distribuição é aleatória – cada *bucket* terá quase o mesmo número de valores atribuídos a ele, independente da distribuição real dos valores de chave de busca.

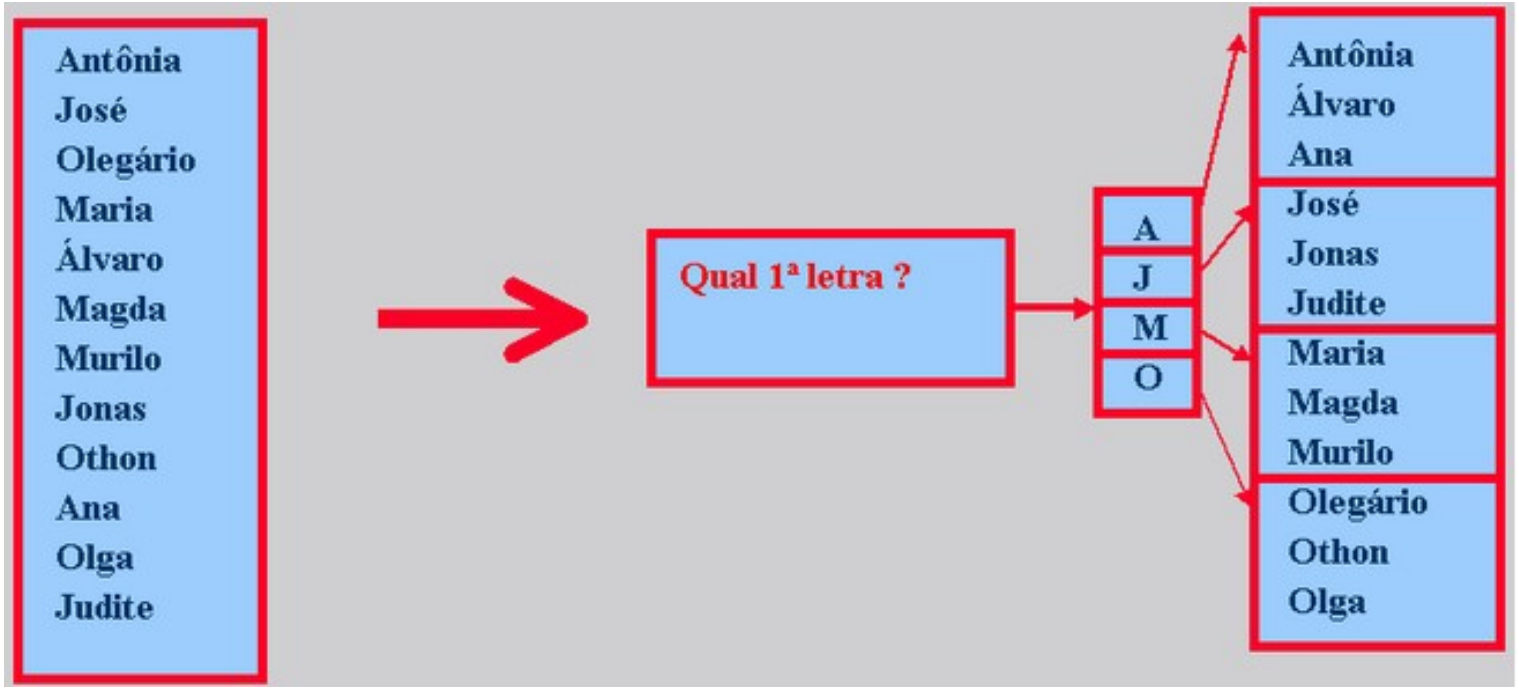
HASHING

- Quando se pretende armazenar um grande conjunto de dados, classificáveis segundo um critério, é necessário:
  - **Criar um critério simples para dividir este universo em subconjuntos com base em alguma qualidade do domínio das chaves.**

HASHING

- Saber em qual procurar e colocar uma chave
- Gerenciar estes subconjuntos bem menores por algum método simples

HASHING


Antônia
José
Olegário
Maria
Álvaro
Magda
Murilo
Jonas
Othon
Ana
Olga
Judite
Qual 1ª letra ?
A
J
M
O
Antônia
Álvaro
Ana
José
Jonas
Judite
Maria
Magda
Murilo
Olegário
Othon
Olga

ÍNDICE *HASH*

- Usa a função *hash* para encontrar a posição do índice e do índice vai para o arquivo.

HASHING ESTÁTICO

- As organizações de arquivos baseadas na técnica de *hashing* permitem evitar o acesso a uma estrutura de índice.
- Em uma **organização de índice de *hash*,** organizamos as chaves de busca, com seus ponteiros associados, para uma estrutura de arquivo de *hash.*

HASHING DINÂMICO - Multinível

- Várias técnicas de *hashing dinâmico* permitem que a função de *hash* seja modificada dinamicamente para acomodar o crescimento ou encolhimento do banco de dados.
  - *Hashing extensível* – Lida com as mudanças no tamanho do banco de dados, dividindo e unindo os *buckets* enquanto o banco de dados cresce e encurta. Como resultado, a eficiência do espaço é mantida.

HASHING

- **Problemas com o uso de Tabelas de Hash**
  - Determinar uma função de *hashing* que minimize o número de colisões
  - Obter os mecanismos eficientes para tratar as colisões

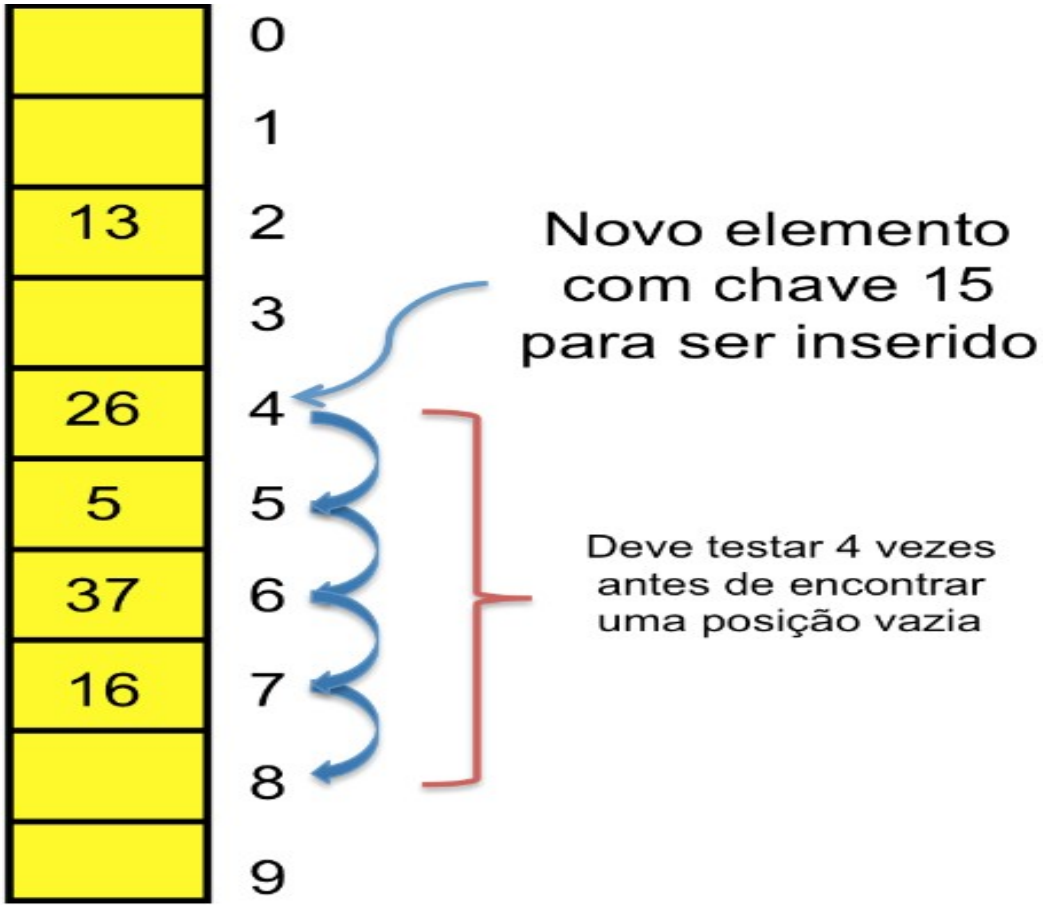
HASHING

- **Colisão**
  - Mesmo índice para duas chaves diferentes
    - **Mecanismos de Tratamento**
      - Endereçamento Aberto
      - Endereçamento Fechado

HASHING

- **Endereçamento Aberto**
  - A informação é armazenada na própria tabela de dispersão
    - Teste Linear
    - *Hashing* Duplo

TESTE LINEAR



TESTE LINEAR

- **Vantagem:** Simplicidade
- **Desvantagem:** Se ocorrerem muitas colisões, pode ocorrer um *clustering* de chaves em uma certa área. Isso pode fazer com que sejam necessários muitos acessos para recuperar um certo registro.

HASHING DUPLO

- Uma **função *hash*** auxiliar é usada para calcular o incremento

Hashing Duplo

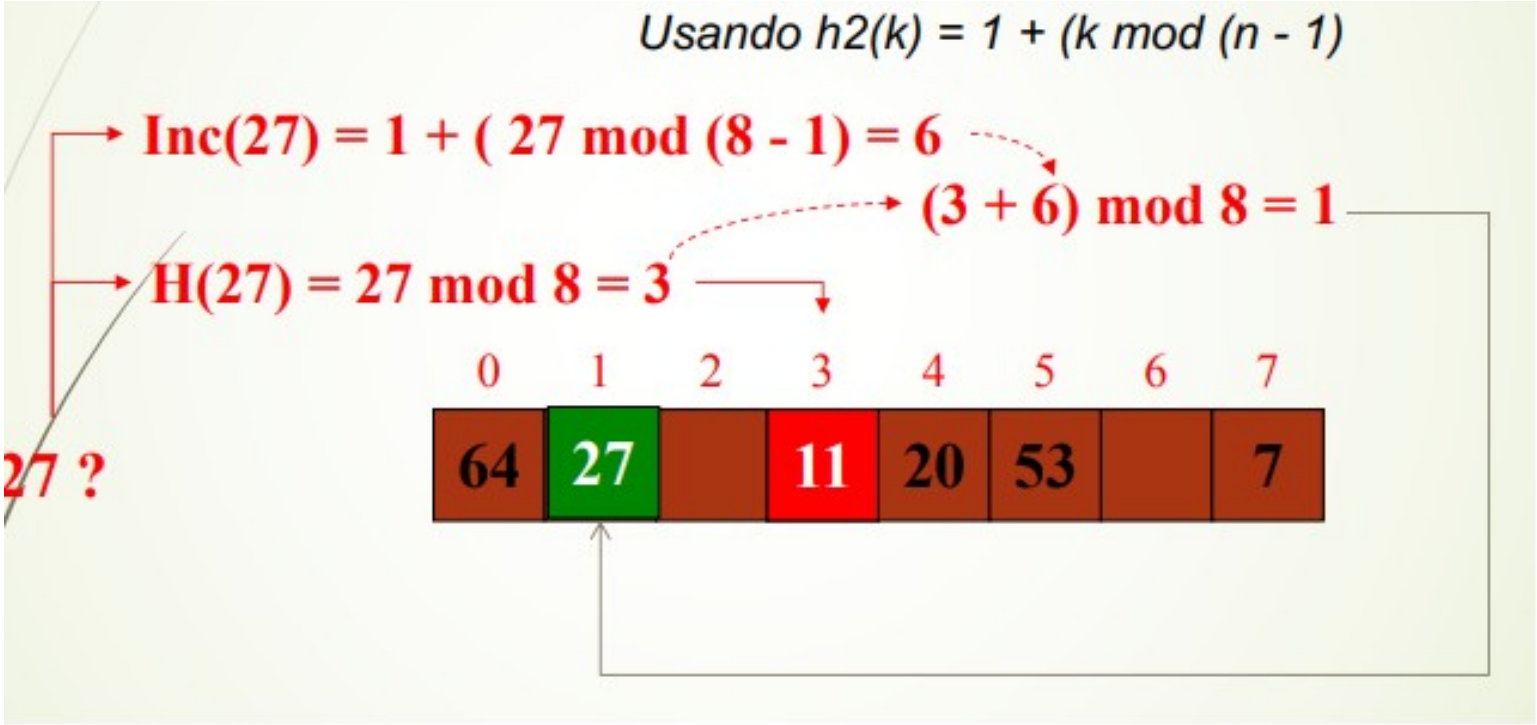
- Para o primeiro cálculo:

  h(k) = k mod N
- Caso não haja colisão, inicialmente calculamos h2(k), que pode ser definida como:

  H2(k) = 1 + (k mod (N-1))
- Em seguida calculamos a função re-hashing como sendo:

  rh(i,k) = (i + h2(k)) mod N

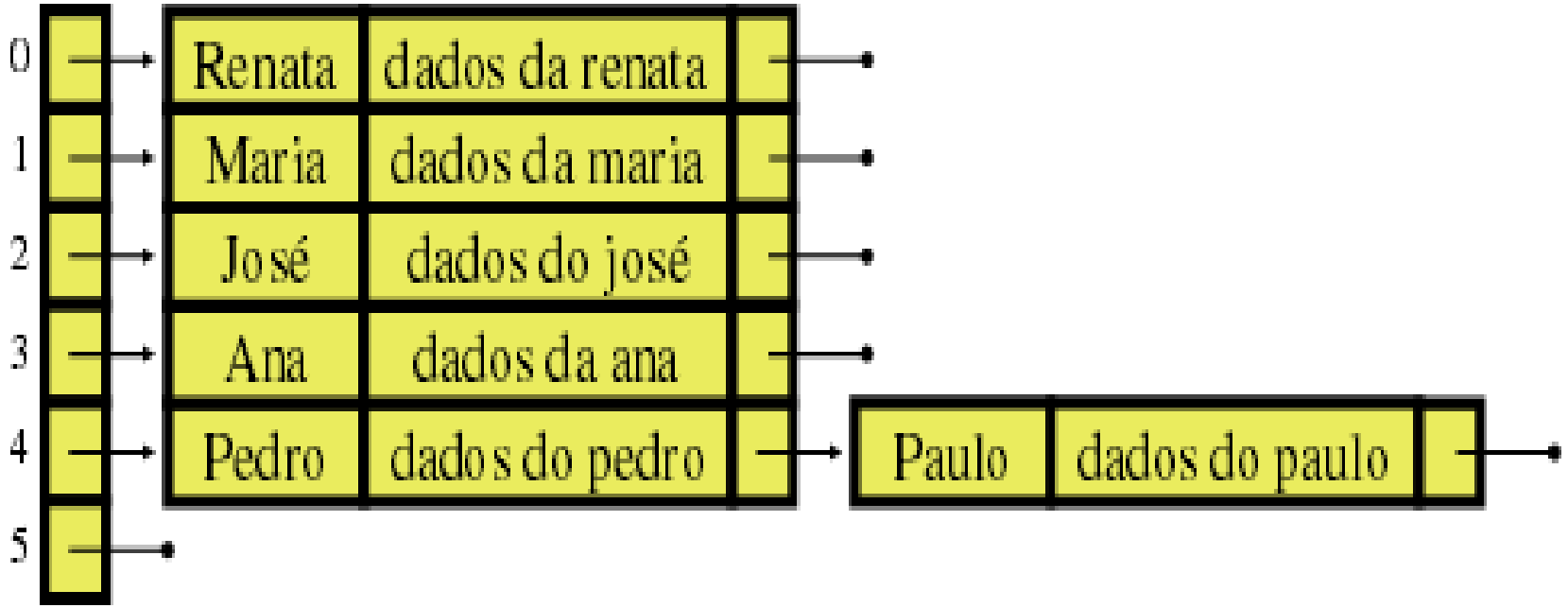
Hashing Duplo



Hashing

- **Endereçamento Fechado**
  - **A posição de inserção não muda**
    - **Encadeamento**
  - A informação é armazenada em estruturas encadeadas fora da tabela de dispersão

Lista Encadeada



VANTAGENS HASHING

- Simplicidade
- Escalabilidade
- Eficiência para grande número de elementos
- Aplicação imediata a arquivos

DESVANTAGENS HASHING

- Dependência da escolha da função de *hashing*
- Ineficiência para os últimos elementos das listas ligadas

Armazenamento e Indexação - PostgreSQL

- Os bancos de dados gerenciados por um servidor *PostgreSQL* são particionados pelo administrador em *clusters* de banco de dados, em que todos os dados e metadados associados a um *cluster* são armazenados no mesmo diretório do sistema de aequivos. Ao contrário dos sistemas comerciais, o *PostgreSQL* não admite *"tablespaces"* que dariam ao administrador de banco de dados o controle preciso do local de aramazenamento de cada objeto físico individual.

Armazenamento e Indexação - PostgreSQL

- Além disso, o *PostgreSQL* admite apenas **“sistemas de arquivos preparados”**, evitando o uso de partições de disco brutas.
- A simplicidade no projeto do sistema de armazenamento do *PostGreSQL* potencialmente leva a algumas limitações de desempenho. A falta de suporte para *tablespace* limita as possibilidades de usar de forma eficiente os recursos de armazenamento disponíveis, em particular, vários discos operando em paralelo.

Armazenamento e Indexação - PostgreSQL

- Além do mais, o tamanho de bloco de todos os objetos do banco de dados é fixo, que pode oferecer desempenho inferior ao lidar com armazenamento que trata de blocos de dados maiores. O uso de sistemas de arquivos preparados resulta no *buffer* duplo, em que um bloco é primeiro lido do disco para o cache do sistema de arquivos antes de ser copiado para o *pool* de *buffer* do *PostGreSQL*.

Armazenamento e Indexação - PostgreSQL

- https://www.youtube.com/watch?v=fsG1XaZEa78

TRABALHO I

- Armazenamento e Indexação – Banco Relacional
  - Vídeo

REFERÊNCIAS

- ALMEIDA, Eduardo – UFPR – Disponível em: http://www.inf.ufpr.br/eduardo/ensino/ci218/ - Acesso em: 10 de Fevereiro de 2021
- ELMASRI,NAVATHE – Sistemas de Banco de Dados – São Paulo - Pearson, 2011
- IAM Academy – Disponível em: https://blog.identityclasses.com/indexing-in-sailpoint-iiq/ - Acesso em: 25 de Fevereiro de 2021
- KORTH, SILBERSCHATZ - Sistemas de Banco de Dados – São Paulo, Campus, 2006
- NETO, Acácio Feliciano e outros – Engenharia da Informação – São Paulo – McGraw-Hill Ltda, 1988
- NUNES, Fátima L.S. – Indexação e Hashing – Parte 2 – Slides – Disciplina de Laboratório de Base de Dados – Sistemas de Informação – Universidade de São Paulo - USP – Disponível em: https://webcache.googleusercontent.com/search?q=cache:tUIo5ULX4M0J:https://edisciplinas.usp.br/mod/resource/view.php%3Fid%3D2311216+&cd=3&hl=pt-BR&ct=clnk&gl=br Acesso em: 10 de Fevereiro de 2021
- WANGENHEIM, Aldo Von; Estrutura de Dados – Disponível em: https://www.inf.ufsc.br/~aldo.vw/estruturas/Hashing/ - Acesso em: 10 de Fevereiro de 2021
- WIKIPEDIA – Hash - Disponível em: www.wikipedia.org – Acesso em: 10 de Fevereiro de 2021