Árvores B

Evandro Eduardo Seron Ruiz
evandro@usp.br

Universidade de São Paulo

Conteúdo desta apresentação

1. Introdução
2. Definições formais
3. Operações básicas
4. Exercícios

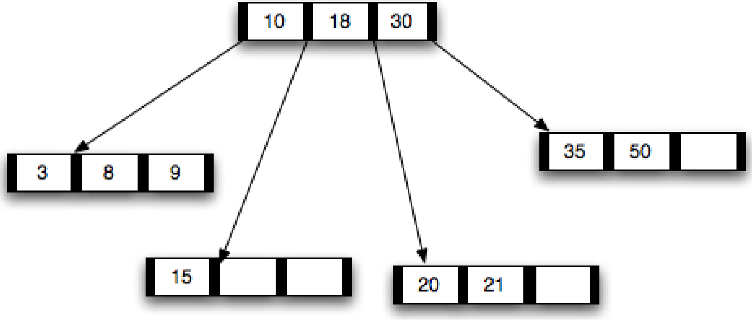
O que são as árvores B

- Formalizadas por Bayer e McCreight em 1972
- São árvores de pesquisa balanceadas
- Projetadas para trabalhar com conjuntos de dados $>$ RAM
- Projetadas para trabalhar com memória de acesso direto (=discos)
- Muitos SGBD usam árvores B para armazenar informações
- Praticamente todos s.o. usam árvores B

Árvores B

- *B-trees*: nós podem ter muitos filhos, até milhares
- Tipicamente entre 50 e 2000 filhos por nó
- Existe *customização*, ou seja, tamanho dos nós $\equiv$ a páginas do disco
- Árvores com n nós tem $h \sim \log(n)$
- Complexidade de busca: $O(\log n)$

Uma árvore B



Fonte: http://scienceblogs.com/goodmath/2008/07/btrees_balanced_search_trees_f.php

Referencial Teórico

Cormen, T.H. (2002)

Algoritmos – teoria e prática
tradução da segunda edição norte-americana
Editora Campus, 2002

Por que usar árvores B

- Armazenar grandes conjuntos de dados
- Todos os dados precisam ser armazenados em disco (menos o nó raíz)
- Cada página de memória $\equiv$ um nó

Por que usar árvores B

- Armazenar grandes conjuntos de dados
- Todos os dados precisam ser armazenados em disco (menos o nó raíz)
- Cada página de memória $\equiv$ um nó
- Dois fatores são limitantes principais no acesso a disco
  - Número de acessos aos disco
  - Tempo de processamento da informação

Número de acessos a disco

- Quantas páginas precisam ser lidas/gravadas no disco?
- Tempo de acesso depende de onde está a informação (qual trilha)

Número de acessos a disco

- Quantas páginas precisam ser lidas/gravadas no disco?
- Tempo de acesso depende de onde está a informação (qual trilha)
- Se trilha atual é longe da próxima trilha a ser lida
- Considerar local na trilha, se disco precisa dar mais uma volta

Número de acessos a disco

- Quantas páginas precisam ser lidas/gravadas no disco?
- Tempo de acesso depende de onde está a informação (qual trilha)
- Se trilha atual é longe da próxima trilha a ser lida
- Considerar local na trilha, se disco precisa dar mais uma volta
- Disco de 7.200 RPM $\Longrightarrow$ 1 volta = 8,33 milissegundos
- $\sim$ cinco ordens de magnitude $>$ acesso a RAM
- Equivalente ao tempo para acessar 100.000 dados na RAM

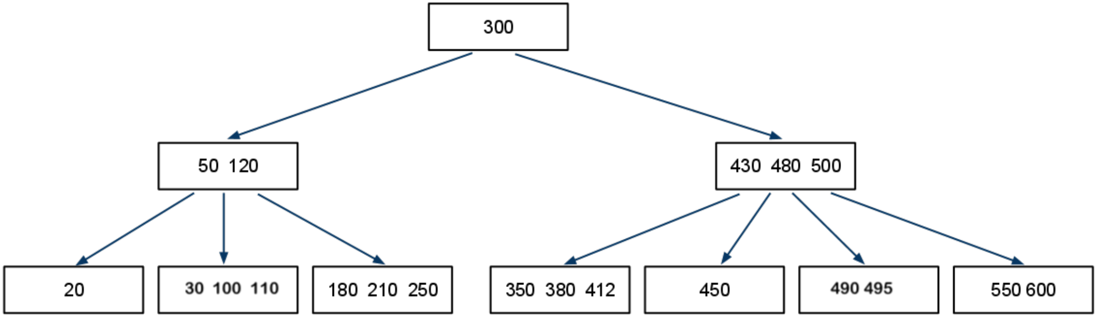
Tempo de processamento da informação lida

- Árvores B manipulam informações que não cabem totalmente na RAM
- Páginas selecionadas são copiadas para a RAM
- Tamanho da memória principal não limita tamanho das árvores B
- Páginas devem ser processadas
- Tempo de processamento depende da complexidade das tarefas

As chaves

- Dado x um nó da árvore B de grau t
- x contém $(t-1)$ chaves
- x pode conter t apontadores não nulos, logo
- x pode conter t filhos
- Ver próxima imagem

Uma árvore B



$(t = 4)$

Definições práticas

Uma árvore B de ordem t é uma árvore t-ária de busca com as seguintes propriedades:

- A raíz é uma folha ou tem, ao menos, dois nós filhos
- Cada nó, a exceção da raíz e das folhas, tem entre $t/2$ e t filhos

Definições práticas

Uma árvore B de ordem t é uma árvore t-ária de busca com as seguintes propriedades:

- A raíz é uma folha ou tem, ao menos, dois nós filhos
- Cada nó, a exceção da raíz e das folhas, tem entre $t/2$ e t filhos
- O caminho da raíz para cada folha é sempre do mesmo comprimento
  Ou seja, é balanceada

Definições práticas

Uma árvore B de ordem t é uma árvore t-ária de busca com as seguintes propriedades:

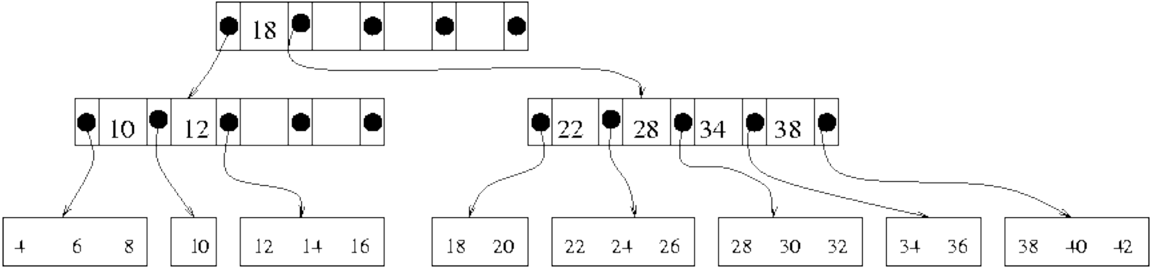
- A raíz é uma folha ou tem, ao menos, dois nós filhos
- Cada nó, a exceção da raíz e das folhas, tem entre $t/2$ e t filhos
- O caminho da raíz para cada folha é sempre do mesmo comprimento
  Ou seja, é balanceada
- Raíz, nó interno e folha são tipicamente $\equiv$ páginas de disco

Definições práticas

Uma árvore B de ordem t é uma árvore t-ária de busca com as seguintes propriedades:

- A raíz é uma folha ou tem, ao menos, dois nós filhos
- Cada nó, a exceção da raíz e das folhas, tem entre $t/2$ e t filhos
- O caminho da raíz para cada folha é sempre do mesmo comprimento
  Ou seja, é balanceada
- Raíz, nó interno e folha são tipicamente $\equiv$ páginas de disco
- Cada nó interno tem até $(t-1)$ chaves e t apontadores para os nós filhos
- Os registros (dados) são tipicamente armazenados nas folhas, mas isso não é uma regra

Representação de uma árvore B



Fonte: `http://lcm.csa.iisc.ernet.in/dsa/node122.html`
($t = 5$)

Definição de árvore B

Uma árvore B, de ordem $n = t$, enraizada num nó T tem as seguintes propriedades:

- Todo nó x tem os seguintes campos ou características:
  - Cada nó x armazena t apontadores e $(t-1)$ chaves
  - As $(t-1)$ chaves $(k_1, k_2, \ldots, k_{t-1})$ são armazenadas em ordem não decrescente[1] $(k_1 \leq k_2 \leq \ldots \leq k_{t-1})$
  - Cada apontador p_i, t.q. $1 \leq i \leq t$, aponta para o filho i
  - Valor booleano **folha** que indica se x é folha ou nó interno

[1] Evitaremos chaves repetidas, portanto, podemos dizer *crescente*.

. . . ainda definições

- Se x é um nó interno, x contém t apontadores para os filhos $f_1, f_2, \ldots, f_t$
- Podem existir apontadores para `NULL`

. . . ainda definições

- Se x é um nó interno, x contém t apontadores para os filhos $f_1, f_2, \ldots, f_t$
- Podem existir apontadores para `NULL`
- As chaves k_i separam os intervalos de chaves armazenadas, ou seja, todas as chaves na subárvore apontadas por p_1 são menores que k_1; e
- Para $2 \leq i \leq (t-1)$, os apontadores $k_{i-1} \leq p_i < k_i$, ou seja
- Todas as chaves na subárvore apontada por p_i, $2 \leq i \leq (t-1)$, são maiores ou iguais a k_{t-1} e menores que k_i

. . . e mais

- Toda folha tem a mesma profundidade que é h, altura da árvore
- Dado t, $t \geq 2$, como o grau mínimo de uma árvore B
  - Todo nó diferente da raíz deve ter pelo menos $t - 1$ chaves. Deste modo, todo nó interno, diferente da raíz, tem pelo menos t filhos
  - Todo nó pode conter no máximo $2t - 1$ chaves, ou seja, um nó interno pode ter no máximo $2t$ filhos
  - Exe: $t = 2$ significa $(2 * 2 - 1) = 3$ chaves, ou seja, filhos.
  - Dizemos que um nó é **completo** se ele contém exatamente $(2t - 1)$ chaves.

A árvore B mais simples ocorre quando $t = 2$ $(2 = \frac{4}{2})$, ou seja, todo nó interno tem 2, 3 ou 4 filhos. Chamamos estas árvores de **árvore 2-3-4**

Altura de uma árvore B. Teorema

- Sabemos que o número de acessos a uma árvore binária de busca é proporcional a altura h da árvore

Altura de uma árvore B. Teorema

- Sabemos que o número de acessos a uma árvore binária de busca é proporcional a altura h da árvore
- Uma árvore B é uma árvore de busca

Altura de uma árvore B. Teorema

- Sabemos que o número de acessos a uma árvore binária de busca é proporcional a altura h da árvore
- Uma árvore B é uma árvore de busca

Uma árvore B com n chaves, altura h e grau mínimo $t \geq 2$ satisfaz a relação:

$$h \leq \log_t \frac{n+1}{2}$$

Prova

Imagine uma árvore B, altura h, com o número mínimo de nós, ou seja, $(t-1)$ chaves

- O nó raíz contém uma chave (ao menos)
- Árvore assemelha-se a uma árvore binária de busca

Prova

Imagine uma árvore B, altura h, com o número mínimo de nós, ou seja, $(t-1)$ chaves

- O nó raíz contém uma chave (ao menos)
- Árvore assemelha-se a uma árvore binária de busca
- Todos os nós internos contém $(t-1)$ chaves
- Existe 1 nó raíz (nível 0),
- Existem 2 $(2h^0)$ nós com nível 1,

Prova

Imagine uma árvore B, altura h, com o número mínimo de nós, ou seja, $(t-1)$ chaves

- O nó raíz contém uma chave (ao menos)
- Árvore assemelha-se a uma árvore binária de busca
- Todos os nós internos contém $(t-1)$ chaves
- Existe 1 nó raíz (nível 0),
- Existem 2 $(2h^0)$ nós com nível 1,
- $2t$ nós no nível 2,
- $2t^2$ no nível 3... até a altura h
- Em h a árvore B terá $2t^{(h-1)}$ nós

Altura de uma árvore B

portanto...

$$n \geq 1 + (t-1)\sum_{i=1}^{h} 2t^{(i-1)}$$

$$n \geq 1 + 2(t-1)\frac{t^h - 1}{t-1}$$

$$n \geq 2t^h - 1$$

isolando t^h...

continuação da prova

relembrando...

$$n \geq 2t^h - 1$$

podemos obter

$$t^h \leq \frac{n+1}{2}$$

aplicando-se o operador $\log_t$ em ambos os lados da desigualdade

$$h \leq \log_t \frac{n+1}{2}$$

ou seja...

O número de acessos a disco é proporcional a altura h, e no pior caso...

- Se $n \geq 1$, para qualquer árvore B de n nós,
- altura h, e
- grau mínimo $t \geq 2$ é

$$h \leq \log_t \frac{n+1}{2}$$

ou seja, o número de acessos é realmente pequeno comparativamente a capacidade da árvore.

Lembro $\log_{10}(100.000) = 5$ e $\log_{10}(100.000.000) = 8$

Quais são elas?

- Busca; e
- Criação da árvore

Convenções

- O nó raíz de uma árvore binária está sempre na memória principal
- A leitura do disco para o elemento raíz nunca é exigida

Convenções

- O nó raíz de uma árvore binária está sempre na memória principal
- A leitura do disco para o elemento raíz nunca é exigida
- A escrita do nó raíz é exigida sempre que o nó raíz for modificado
- Todos os nós passados como parâmetros já foram submetidos a uma operação de leitura do disco sobre seus dados

Pesquisa numa árvore B

- A pesquisa numa árvore B é semelhante a pesquisa em árvores binárias de busca
- Decisão não é binária
- Decisão ramificada em várias vias, de acordo com o número de filhos
- Vejamos agora um pseudo-código clássico que não emula uma árvore B, mas mantém as operações de disco

Algoritmo em Cormen *et. al.*

Raíz x (apontador), chave p a ser pesquisada

```
btree_search (x, p)
  i=1
  while i <= n (numero chaves) e p > chave ki
    do i = i+1
  if i <= n e p=ki
    return (x, i)
  if folha(x)
    return NULL
  else
    disk-read (ci) // filho
    return btree_search (ci, p)
```

Algoritmo em Ziviani

```
btree_search (x, p)
  i=1
  while i <= n (numero chaves) e p > chave ki
    do i = i+1
  if p=ki
    x = ki
  else
  if p < ki
    btree_search(k(i-1),p)
  else
    btree_search(k, p)
```

Criação de uma árvore B

- Para construir uma árvore B precisamos criar um nó raíz vazio, e depois
- Inserir novas chaves
- Usamos, nos dois caso, um procedimento de alocação de nós, ou seja, de páginas no disco
- Procedimento de alocação de nós no disco não exige leitura de dados desta mídia

Algoritmo em Cormen *et. al.*

```
btree_create (bt)
  x = aloca_no()
  x.folha = true
  x.n = 0 //numero de chaves
  disk-write (x)
  bt.raiz = x
```

Exercícios

- Por que não permitimos um grau mínimo $t = 1$ neste TAD?
- Supor que um milhão de páginas sejam armazenadas numa árvore B completa, com $t = 50$. Quantos acessos a disco seriam necessários para recuperar um registro?
- Codifique, em pseudo-código, uma estrutura de dados que represente um nó deste TAD
- Codifique, em pseudo-código, as operações de criação e busca de elementos neste TAD

Nada substitui uma árvore