Algoritmos e Estrutura de Dados

Árvores B

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Faculdade de Engenharia

Agenda

Árvore B

Estrutura Nodo

Algoritmo 1: Nodo

```
início
    registro {
        Vetor Inteiro: key[1..2t-1]; // Vetor de chaves.
        Vetor Ponteiro Nodo: c[1..2t]; // Vetor de ponteiros.
        Inteiro: n; // Quantidade de chaves armazenadas.
        Booleano: folha; // Indica se o nodo é folha.
    } Nodo;
```

Operações Básicas

A seguir, as operações básicas a serem realizadas sobre Árvores B:

- ArvoreB-Criar
- ArvoreB-Busca
- ArvoreB-Inserir
- ArvoreB-Remover

CriaÁrvore

Pseudocódigo

Algoritmo 2: ArvoreB-Criar

Entrada: Ponteiro T para a árvore.

```
início
    novo ← ALOCA_NODO()
    novo.folha ← TRUE
    novo.n ← 0
    DISK_WRITE(novo)
    T.raiz ← novo
```

Adaptado de [Cormen et al. 2012].

- ALOCA_NODO() aloca uma página no disco de modo a ser usada como um novo nodo.
- Requer $O(1)$ operações no disco e tempo de processador $O(1)$.

Busca

Pseudocódigo

Algoritmo 3: ArvoreB-Busca

Entrada: Ponteiro para a raiz r, chave k a ser buscada na árvore.
Saída: O nodo que contém k ou NULL caso não encontre.

início
    i← 1
    enquanto $(i \leq r.n)$ e $(k > r.key[i])$ **faça**
        $i \leftarrow i + 1$
    se $(i \leq r.n)$ e $(k = r.key[i])$ **então**
        retorna (r, i)
    se $r.folha$ **então**
        retorna NULL
    senão
        DISK_READ($r.c_i$)
        retorna ArvoreB-Busca($r.c[i]$, k)

Adaptado de [Cormen et al. 2012].

Divisão

Pseudocódigo

Algoritmo 4: ArvoreB-Divide-Filho

Entrada: Um nodo interno não-cheio x, um índice i, um nodo cheio y filho de x, tal que $y = x.c_i$.

```
início
    z ← ALOCA_NODO()
    z.folha ← y.folha
    z.n ← t-1
    para (j ← 1 até t-1) faça
        z.key[j] ← y.key[j + t]
    se (NOT(y.folha)) então
        para (j ← 1 até t) faça
            z.c[j] ← y.c[j + t]
    y.n ← t-1
```

Adaptado de [Cormen et al. 2012].

Divisão

Pseudocódigo

Algoritmo 5: ArvoreB-Divide-Filho (continuação)

início

    para $(j \leftarrow x.n + 1$ *descendo até* $i + 1)$ **faça**

        $x.c[j + 1] \leftarrow x.c[j]$

    $x.c[i + 1] \leftarrow z$

    para $(j \leftarrow x.n$ *descendo até* $i)$ **faça**

        $x.key[j + 1] \leftarrow x.key[j]$

    $x.key[i] \leftarrow y.key[t]$

    $x.n \leftarrow x.n + 1$

    DISK_WRITE(y)

    DISK_WRITE(z)

    DISK_WRITE(x)

Adaptado de [Cormen et al. 2012].

Inserção

Pseudocódigo

Algoritmo 6: ArvoreB-Inserir

Entrada: Ponteiro T para a árvore, chave k a ser inserida.

```
início
    r ← T.raiz
    se (r.n = 2t − 1) então
        s ← ALOCA_NODO()
        T.raiz ← s
        s.folha ← FALSE
        s.n ← 0
        s.c[i] ← r
        ArvoreB-Divide-Filho(s,1,r)
        ArvoreB-Inserir-NaoCheio(s,k)
    senão
        ArvoreB-Inserir-NaoCheio(r,k)
```

Adaptado de [Cormen et al. 2012].

Inserção

Pseudocódigo

Algoritmo 7: ArvoreB-Inserir-NaoCheio

```
Entrada: Nodo raiz x, chave k a ser inserida.
início
    i ← x.n
    se (x.folha) então
        enquanto (i ≥ 1) e (k < x.key[i]) faça
            x.key[i + 1] ← x.key[i]
            i ← i - 1
        x.key[i + 1] ← k
        x.n ← x.n + 1
        DISK_WRITE(x)
    senão
        enquanto (i ≥ 1) e (k < x.key[i]) faça
            i ← i - 1
        i ← i + 1
        DISK_READ(x.c[i])
        se (x.c[i].n = 2t − 1) então
            B-Tree-Split-Child(x,i,x.c[i])
            se (k > x.key[i]) então
                i ← i + 1
        ArvoreB-Inserir-NaoCheio(x.c[i], k)
```

Adaptado de [Cormen et al. 2012].

Remoção

Considere k a chave a ser removida e x o nodo que contém k. Existem três casos:

1. Se a chave k está em x, um nodo folha, e x contém pelo menos t chaves.
   - A remoção é trivial.
2. Se a chave k está em x, um nodo interno. Existem três casos a considerar:
   a. Verifique se o filho à esquerda y possui pelo menos t chaves. Nesse caso, a chave predecessora k' de k é movida para x, e em seguida remove-se k.
   b. Simetricamente, verifique se o filho à direita z possui pelo menos t chaves. Nesse caso, a chave sucessora k' de k é movida para x, e em seguida remove-se k.
   c. Caso contrário, se ambos y e z tem apenas $t - 1$ chaves, realize a união de k.
      - Todos os elementos em z e em y passam a pertencer a um único nodo, em conjunto com a chave k.
      - Com isso, k e o ponteiro para z são removidos de x. y agora contém $2t - 1$ chaves, e subsequentemente k é removida.

Remoção

3. Se a chave k não foi encontrada no nodo x.
   - É necessário determinar a subárvore $x.c[i]$ apropriada que deve conter k.

   a. Ao determinar o filho que contém k, verifica-se se este possui apenas $t - 1$ chaves.
      - ★ Verifique se um dos irmãos de $x.c[i]$ possui pelo menos t chaves. Se possível, movimente uma chave do irmão à esquerda ou à direita para x.
      - ★ Se $x.c[i]$ tiver filhos, verifique também se um deles possui pelo menos t chaves. Nesse caso, movimente uma chave de um filho.

   b. Caso contrário, $x.c[i]$, todos os seus filhos e irmãos contém apenas $t - 1$ chaves. Dessa forma, resta apenas fazer a intercalação de $x.c[i]$ com um de seus irmãos, o que envolve mover uma chave de x para baixo.

Referências Bibliográficas

CORMEN, T. H. et al. *Algoritmos: Teoria e Prática*. 3. ed. São Paulo: Campus, 2012. ISBN 978-0-262-03384-8.

Material Complementar

Animações

- https://www.cs.usfca.edu/ galles/visualization/BTree.html
  - Obs.: Escolha "Max Degree $= 4^n$ para árvore de ordem $t = 2$.
- http://cs.armstrong.edu/liang/animation/web/24Tree.html
  - Árvore 2-3-4