Árvores 2-3 e B-Árvores

Última aula de Estrutura de dados

Mac2301 - primeiro semestre de 2002

1 Árvores mais sofisticadas

Para grandes aplicações com banco de dados, as seguintes operações são muito utilizadas:

1. Um grande conjunto de registros é atualizado frequentemente;
2. A busca é feita por uma, ou uma combinação de chaves;
3. Busca por faixa de chaves tambem é utilizada (por exemplo: encontrar todos os registros com idade entre 10 e 20 anos).

Nestes BDs, é necessária uma forma bem eficiente de representar os dados. Um forma seria o uso de árvores de busca binária, ou árvores AVL. Entretanto estas estruturas podem ficar desbalanceadas, e mesmo em condições favoráveis a diferença de altura entre folhas pode ser considerável. Estas diferenças são sentidas principalmente quando as árvores são muito grandes e não podem ser armazenadas inteiramente na mémoria, e devem ser guardadas no disco.

Para garantir que a estrutura esteja sempre perfeitamente balanceada (todas as folhas estão no mesmo nível, ou profundidade), novas estruturas de dados serão necessárias.

1.1 Árvore 2-3

A árvore 2-3 não é binária, mas obedece às seguintes propriedades:

- Cada nó contém uma, ou duas chaves;
- Cada nó interno tem dois (se tem uma chave), ou três (se tem duas chaves) filhos;
- Todas as folhas estão no mesmo nível.

Além disto, a árvore 2-3 tem as mesmas propriedades que uma árvore binária de busca. Para cada nó, o valor de todos os decendentes na árvore a esquerda são menores que a primeira chave, os valores na árvore do centro são maiores ou iguais a primeira chave. Se existe uma terceira sub-árvore, as chaves da árvore central são menores do que o valor da segunda chave, e as chaves armazenadas na árvore da direita são maiores ou iguais a segunda chave. Veja um exemplo abaixo, na figura 1:

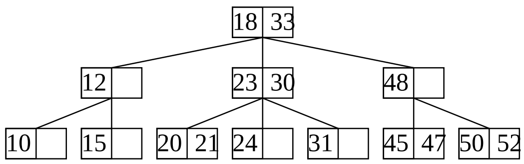

Figura 1: Exemplo de árvore 2-3.

O código para a estrutura segue abaixo:

```
typedef struct _no23 {
   int  lkey,             // chave esquerda
        rkey,             // chave direita
        nkeys;            // número de chaves
   struct _no23 *left,    // ponteiro ao filho esquerdo
                *center,  // ponteiro ao filho central
                *right;   // ponteiro ao filho direito
} no23;
```

a seguinte função retorna o nó com a chave (se este estiver na árvore):

```
no23 *find(no23 *raiz, int key) {
    if (raiz==NULL)
        return NULL;      // não encontrou
    if (key == raiz->lkey)
        return raiz;      // é a chave esquerda
    if ((raiz->nkeys == 2) && (key == raiz->rkey))
        return raiz;      // é a chave direita
    if (key < raiz->lkey)
        return find(raiz->left, key);
    else if (raiz->nkeys == 1)
        return find(raiz->center, key);
    else if (key < raiz->rkey)
        return find(raiz->center, key);
    else
        return find(raiz->right, key);
}
```

A inserção em uma árvore 2-3 é parecida com a inserção em uma árvore binária de busca, o novo registro é armazenado em um nó folha. O mecanismo é o seguinte: primeiro deve se encontrar o nó folha onde deveria estar a chave a ser enserida. Se este nó contém apenas um valor, basta adicionar o novo valor a este nó. Veja um exemplo disto na figura seguinte.

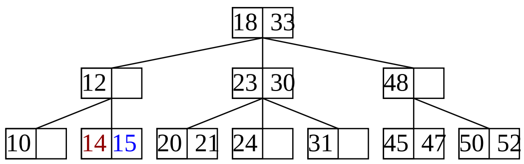

Figura 2: Neste exemplo, na inserção da chave 14, primeiro se descobre a sua posição e depois a inserção é efetuada na posição correta.

Se a chave deve ser inserida em um nó N que já contém duas chaves, mais espaço deve ser criado. Considere os dois valores já existentes, e a nova chave. Primeiro o nó N é quebrado em dois nós, N_1 (antigo N) e N_2. A menor chave fica em N_1, e a maior fica em N_2. Em seguida o valor intermediário é passado para o nó pai de N acompanhado de um ponteiro para N_2. Este mecanismo é denominado promoção. A chave promovida é então inserida no nó pai. Caso este tenha apenas duas chaves, esta chave e um ponteiro para N_1 são adicionados. Caso contrário, se o pai já continha três chaves, o processo de quebra e promoção é repetido. Veja na figura 3 como fica a árvore inicial após a inserção da chave 55.

A próxima figura mostra a árvore da figura 1 após a inserção da chave 19, que vai provocar a quebra, e promoção da raiz.

Segue abaixo o pseudo-código da inserção em uma árvore 2-3. Neste trecho de são apresentadas apenas as funções principais. Além delas são utilizados os seguintes protótipos:

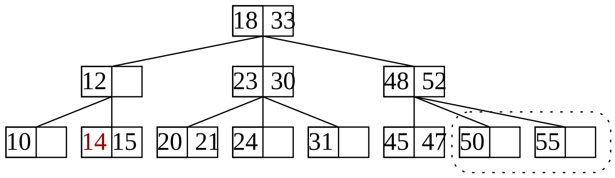

Figura 3: Para inserir a chave 55, o nó que contém 50 e 52 é quebrado, e o valor do meio é inserido no seu pai.

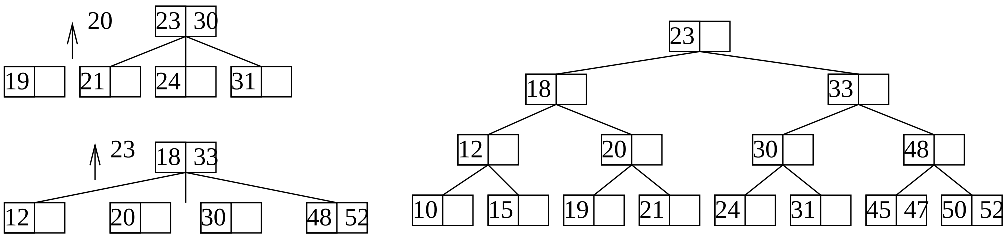

Figura 4: Situação da árvore da figura 1 após a inserção da chave 19, e consequente quebra da raiz.

```
// cria um nó com ch1, ch2 e nchaves, assim como os três ponteiros
no23 *criaNo(int ch1, int ch2, int nchaves, no23 *pl, no23 *pc, no23 *pr);
// verifica se o nó em questão é folha, volta 1 se sim, e 0 caso contrario
int isLeaf(no23 *no);
// coloca uma nova chave ch e arvore p, em um nó com apenas uma chave
void adicionaChave(no23 *no, int ch, no23 *p);
// Quebra um nó em dois, sendo val e subarvore, os novos dados
no23 *quebraNo(no23 *no, int val, int *rval, no23 *subarvore);
```

Função principal insere:

```
// insere val em no (se necessario retorna o novo no e um valor
// rval)
no23 *insere( no23 **no, int val, int *rval){
    no23 *paux, *paux2;
    int   vaux, promov;

    if (*no == NULL) {    // arvore vazia
       *no = (no23 *) malloc (sizeof(no23));
       *no = criaNo(val, 0, 0, NULL, NULL, NULL);
             // cria no folha com um valor
       return NULL;       // nada a fazer depois
    }
    if (isLeaf(*no)){     // chegou a folha
       if ((*no)->nkeys == 1){ // caso fácil
          adicionaChave(*no, val, NULL);
          return NULL;
       } else {
          paux = quebraNo(*no, val, &vaux, NULL);
          *rval = vaux;
          return paux;
       }
```

```
    } else {             // continua a procura
       if (val < (*no)->lkey)
          paux = insere( &((*no)->left), val, &vaux);
       else if (((*no)->nkeys == 1) || (val < (*no)->rkey))
          paux = insere( &((*no)->center), val, &vaux);
       else
          paux = insere( &((*no)->right), val, &vaux);
       if (paux == NULL) // nao promoveu
          return NULL;
       else
          if ((*no)->nkeys == 1){
             adicionaChave(*no, vaux, paux);
             return NULL;
          } else {
             paux2 = quebraNo(*no, vaux, &promov, paux);
             *rval = promov;
             return paux2;
          }
     }
}
```

A única função não óbvia é a quebraNo, dada abaixo:

```
no23 *quebraNo(no23 *no, int val, int *rval, no23 *subarvore){
    no23 *paux;

    if (val > no->rkey) {  // val esta mais a direita
       *rval = no->rkey;   // promove a antiga maior
       paux = no->right;
       no->right = NULL;   // elimina o terceiro filho
       no->nkeys = 1;      // atualiza o número de chaves
       return criaNo(val, 0, 1, paux, subarvore, NULL);
    } else if (val >= no->lkey){ // val esta no meio
       *rval = val;        // continua sendo promovido
       paux = no->right;
       no->right = NULL;
       no->nkeys = 1;
       return criaNo(no->rkey, 0, 1, subarvore, paux, NULL);
    } else {             // val esta a mais a esquerda
       *rval = no->lkey;
       // primeiro cria o nó a direita
       paux = criaNo(no->rkey, 0, 1, no->center, no->right, NULL);
       no->lkey = val;   // em seguida arruma o nó a esquerda
       no->nkeys = 1;
       no->right = NULL;
       no->center = subarvore;
       return paux;
    }
}
```

A remoção é relativamente complexa, excluindo-se os casos simples, de remoção em um nó folha com duas chaves, outras remoções podem levar a casos onde dois nós devem se tornar um único nó.

2 B-árvore

Uma forma ainda mais sofisticada de armazenamento é uma generalização da árvore 2-3. Uma B-árvore de ordem m tem as seguintes propriedades:

- A raiz, ou é uma folha, ou tem pelo menos dois filhos;
- Cada nó, exceto a raiz e as folhas tem entre $\lceil m/2 \rceil$ e m filhos;
- Todas as folhas estão no mesmo nível.

Para mais informações sugiro os livros da bibliografia, ou o texto disponível em: `http://www.ime.usp.br/~coelho/mac324/conteudo/B-arvore.html`