Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Departamento de Computação

Aplicação da Árvore B+

Disciplina: Organização e Recuperação de Informação
Professor: Jander Moreira
Turma: C

Grupo 4:
Amaranto de Marco Lopes 231282
Miguel Antonio Moretti 231150
Renan Polo Montebelo 230928
Thiago Daniel Flauzino 230758

01/10/2003

1.1 – Introdução

Este trabalho consiste na apresentação da implementação de uma Árvore B+. São cobertas as principais dificuldades encontradas na implementação da árvore, assim como as soluções adotadas para contornar tais problemas. A linguagem adotada na implementação da árvore foi a linguagem C++.

2.1 – Visão Geral

A Árvore B+ aqui apresentada é composta de duas partes principais: a estrutura de *árvore* propriamente dita e a estrutura *nó*. A estrutura de árvore engloba um número finito de nós, mas não é exclusivamente composta por estes.

Foram criadas, portanto, duas classes distintas: a classe *arvore.h* e a classe *no.h*. A grande vantagem de separar tais estruturas é que cada uma possui seu próprio construtor e seu próprio destrutor. No entanto, a classe no.h foi implementada com todos os seus membros na interface *public*, pois dessa maneira a classe arvore.h pode manipular diretamente os membros de no.h, evitando o *overhead* de chamadas à funções membro de no.h, o que permite um ganho de desempenho global.

2.2 – A estrutura arvore.h

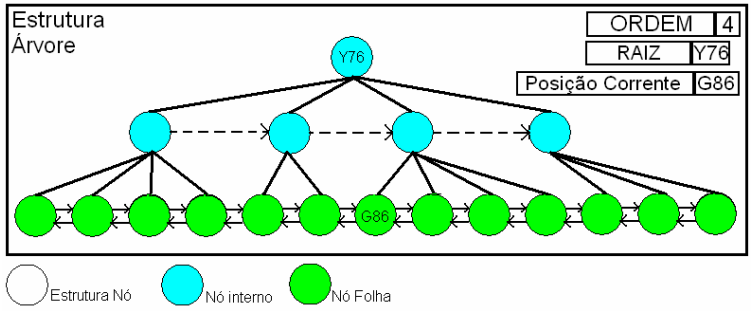
A classe arvore.h foi implementada como sendo um *template* para dois tipos de dados distintos: CHAVE e DADO. No programa fornecido para teste da árvore foram utilizados os tipos INT e FLOAT, respectivamente. Pode-se usar quaisquer outros tipos de dados, inclusive aqueles criados pelo programador, desde que satisfaçam as seguintes condições:

- CHAVE: deve ter os operadores de comparação sobrecarregados (>, >=, <, <=, ==, !=) assim como o operador de extração (<<);
- DADO: deve fornecer um construtor por cópia;

A classe arvore.h possui os seguintes membros:

- Um ponteiro para a raiz da árvore;
- Um ponteiro para a posição corrente após uma pesquisa;
- Um número inteiro positivo que armazena a ordem da árvore;
- Funções membro e funções utilitárias, que serão detalhadas mais à frente.

O construtor da árvore é um construtor *default* para ordem 30. Este também garante que a menor ordem da árvore é ordem 3. O destrutor da classe aproveita-se de uma função utilitária recursiva que apaga todos os nós, a começar pelos nós folhas até chegar à raiz. Um exemplo simplificado de árvore é o seguinte:



Segue uma breve descrição das principais funções:

- **void insere(CHAVE &, DADO &)**

O papel da função insere da árvore é apenas determinar em qual nó deverá ser feita a inserção, pois cabe ao nó a inserção propriamente dita, ou seja, a inserção da classe arvore.h nada mais é do que um apontador percorrendo a árvore entre os nós internos corretos até chegar ao nó folha correto. Casos especiais: a raiz é um nó folha ou não existe raiz (árvore vazia).

- **bool pesquisa(CHAVE &, DADO &)**

A função pesquisa percorre os nós internos, chegando até um nó folha candidato a possuir a chave procurada. Então a própria árvore encarrega-se de varrer o nó folha à procura da chave. Isso é possível pois a classe arvore.h tem acesso aos membros da classe nó. Caso especial: árvore vazia.

- **void imprime()**

A árvore percorrerá todos os nós POR NÍVEL, a começar pela raiz e descendendo até os nós folhas, sempre imprimindo na tela as chaves dos nós do nível corrente da esquerda para a direita. Para percorrer nós internos por nível, estes foram encadeados aproveitando uma posição ociosa do vetor de ponteiros dos nós internos. Essa encadeação é usada só e somente para esse fim, não influenciando em nada a inserção ou remoção da árvore.

- **bool remove(const CHAVE &, INFO &);**

A função remove percorre a árvore, fazendo uma pesquisa para verificar se a chave a ser excluída está na árvore, e a partir daí faz um processo recursivo onde exclui a chave do nó e faz o tratamento de subutilização, para cada ciclo de recursividade.

2.3 – A estrutura no.h

A estrutura no.h foi criada principalmente para desfrutar de um construtor e destrutor próprios, que garantem alocação e liberação de memória corretamente. Além disso, toda a interface da classe é *public*, para garantir que a classe árvore acesse os elementos dos nós diretamente, aumentando o desempenho.

São membros da classe no.h:

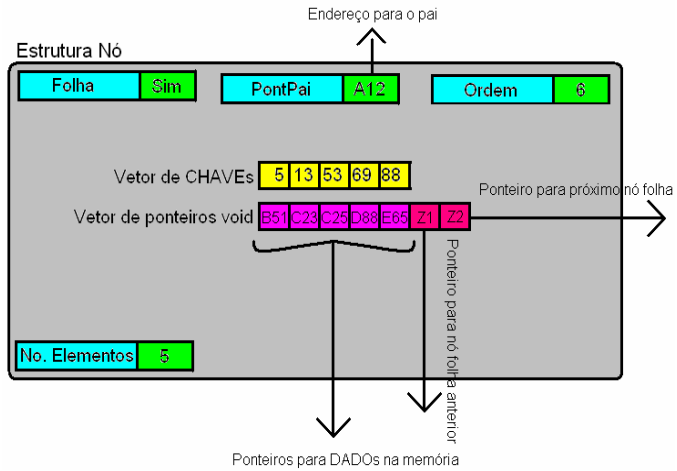
- **bool folha**, que indica se é um nó folha ou nó interno;
- **um vetor do tipo CHAVE**, alocado dinamicamente;

- **um vetor de ponteiros void**, alocado dinamicamente, que aponta ora para outros nós, ora para um DADO na memória;
- **int nro_elementos**, que indica quantas chaves o nó possui atualmente;
- **int ordem**, que armazena a ordem da árvore à qual pertence;
- **um ponteiro para o nó pai;**

O construtor da classe aloca o vetor de CHAVEs com o tamanho ”ordem – 1” e o vetor de ponteiros void com o tamanho “ordem + 1”, isso porque as folhas são duplamente encadeadas. O vetor de ponteiros é inicializado com os valores 0 (NULL) pois assim é mais fácil detectar erros em tempo de execução.

O destrutor libera a memória de ambos vetores (CHAVE e de ponteiros void). Além disso, se for um nó folha que está sendo destruído é liberada a memória alocada para o DADO propriamente dito, que é alocado fora das estruturas árvore e nó.

A lista duplamente encadeada é armazenada nas duas últimas posições do array de ponteiros. Segue um esquema de uma estrutura nó:



As principais funções da classe no.h são:

- **void insere(CHAVE &, DADO &)**

Uma vez que a árvore indicou o nó no qual se dará a inserção, este se torna responsável pela própria inserção. É verificado se há espaço para a inserção ou se será necessário criar um novo nó, no mesmo nível, e sofrer as devidas redistribuições de ponteiros e chaves. Se houver criação de um novo nó, é preciso enviar ao pai duas informações: a chave que “subiu” e o endereço de memória do novo nó. Ao dividir o nó, a inserção definitiva é feita com uma recursão apenas, pois o processo é direto quando não há estouro de nó (caso base). O processo de enviar ao pai as informações da divisão **não** é recursivo.

- **void recebe_insere(CHAVE &, pontno)**

Quando há espaço suficiente no nó interno para a inserção da CHAVE e do ponteiro do novo nó, este é feito diretamente (caso base). Quando há “estouro”, o nó interno é responsável por criar o seu nó “irmão” e redistribuir as chaves e ponteiros. O nó então precisa enviar ao seu pai as informações do

novo nó interno criado: a chave que "sobe" e o endereço de memória do novo nó. Este processo se repete até que a raiz se divida ou até então quando não houver estouro em algum nível.

3.1 – Inserção

A inserção é realizada de um modo "hierárquico", sendo cada nó responsável pelas operações no seu nível e a árvore funcionando como um "disparador" do processo.

Quando a estrutura árvore recebe uma chamada para uma inserção de uma CHAVE e de um DADO, a árvore percorre as chaves dos nós internos descendendo à procura de um nó folha que possa conter a CHAVE e DADO que estão sendo inseridos. Uma vez que chegou na folha que sofrerá a inserção, a árvore passa para tal nó a CHAVE e o DADO, e a partir desse momento a árvore não tem mais nada a ver com o processo de inserção, sendo de inteira responsabilidade do nó folha a inserção.

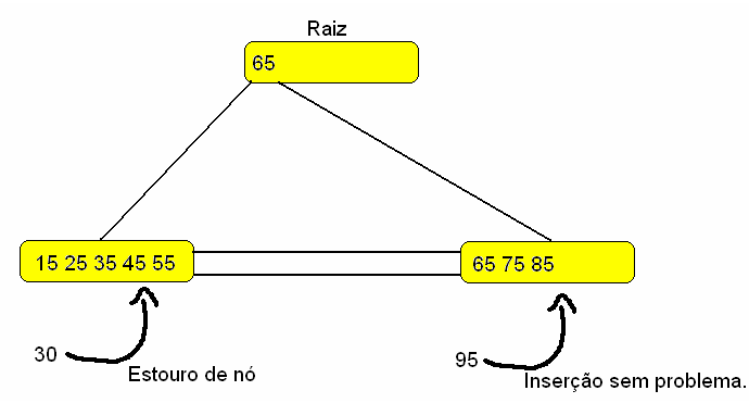
Nesse ponto existem duas possibilidades: tem espaço para a inserção no nó ou há um "estouro" de nó. No primeiro caso, é trivial inserir a CHAVE e o DADO. Já no segundo, é de responsabilidade do próprio nó criar o novo nó "irmão", redistribuir chaves e ponteiros e encadear os nós.

Se não há estouro, o processo de inserção acaba aqui. Por outro lado, se há estouro é preciso que o nó "avise" para o nó pai que houve uma divisão de nó. Então o nó passa para o pai duas informações: a chave que vai "subir" e o endereço do novo nó recém criado. A partir desse momento o nó não influi mais em nada na inserção, sendo toda a responsabilidade transferida para o nó pai, onde o processo se repete até que em algum nível não haja estouro de nó ou então a raiz se divida. Note que o processo de avisar o pai não é recursivo, mas sim do tipo "efeito dominó".

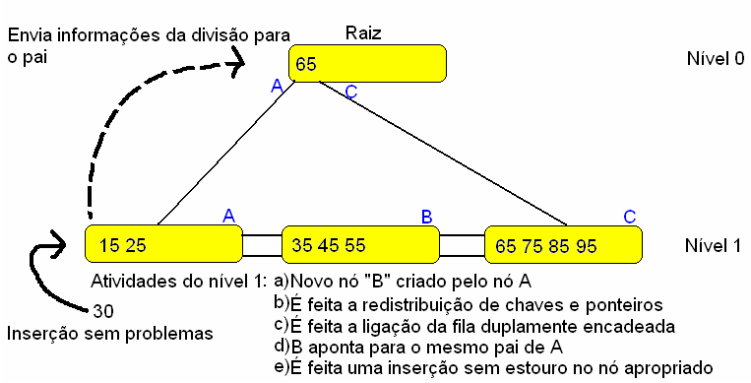
A única recursão do processo ocorre após a divisão de um nó, quando todas as chaves e ponteiros já foram redistribuídos: uma vez que os nós foram divididos, é chamada a inserção no nó apropriado, caindo no caso base pois há espaço no nó. Sempre é realizada somente uma recursão.

A única exceção à idéia da hierarquia é quando um nó interno se divide, pois cabe no novo nó interno criado modificar o "ponteiro para o pai" de seus filhos.

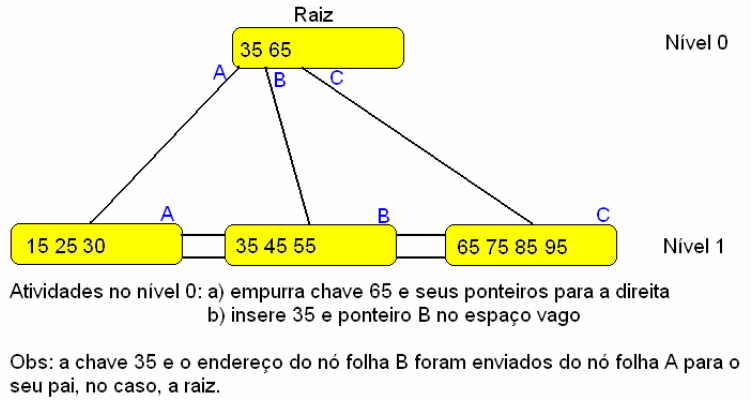
Passo 1:



Passo 2:



Passo 3:



Obs: a chave 35 e o endereço do nó folha B foram enviados do nó folha A para o seu pai, no caso, a raiz.

4.1 – Remoção

O processo de remoção foi implementado de forma recursiva, sendo dividido em duas funções: uma não recursiva que apenas faz a pesquisa da chave na árvore, faz atribuições a variáveis de controle e chama a função remove recursiva se a chave a ser excluída for encontrada. A função recursiva é implementada da seguinte forma:

- A partir de pesquisas dentro de cada nó, iniciando da raiz, descende-se a árvore até a folha desejada (já que todos os dados estão nas folhas), obedecendo à organização da árvore B+. Para cada nível da árvore, o ponteiro é passado como referência para ser aproveitado no processo de recursividade.

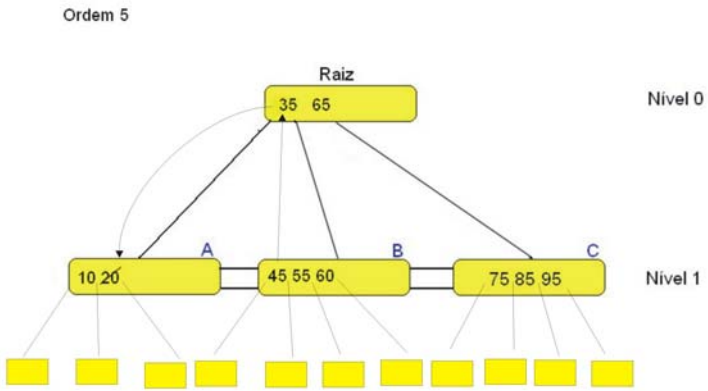
- Ao encontrar o nó onde a chave está localizada, é feita uma nova pesquisa dentro do nó para localizar a chave que será removida. As posições posteriores das chaves do nó são deslocadas uma a uma a fim de ocupar o espaço da chave removida. A variável que controla o nº de chaves é decrementada.

- A partir daí o processo de recursividade começa a retroceder, e a cada nível de ascensão verifica-se se há subutilização do nó (quando o número de chaves de cada nó for menor que a ordem da árvore menos 1 dividido por 2). Se não houver subutilização do nó, o processo de recursividade prossegue até o fim sem fazer alterações. Se houver subutilização:

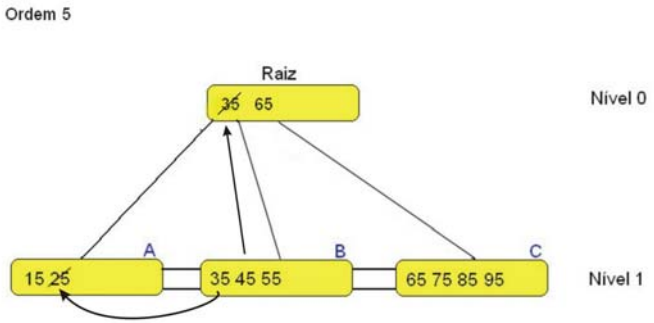
    1. Tenta fazer empréstimo do nó (irmão) esquerdo: Faz-se uma verificação se existe nó (irmão) esquerdo e se esse nó pode emprestar uma chave. Se puder, faz-se o tratamento como descrito e ilustrado no passo 2.

    2. Se não foi possível fazer o empréstimo da esquerda, tenta-se fazer empréstimo do nó (irmão) direito: Faz-se uma verificação se existe nó(irmão) direito e se esse nó pode emprestar uma chave. Se puder, há dois casos possíveis: Se for

nó interno, faz-se a chamada rotação de chaves, como ilustra a figura abaixo:

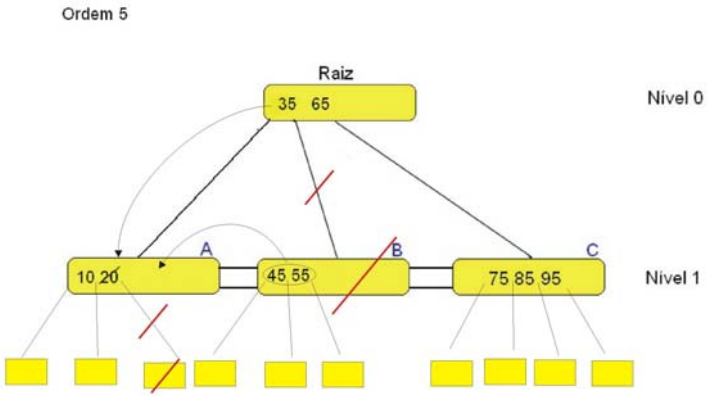


Se for nó folha, basta emprestar a primeira chave do irmão direito, alterar a chave do pai que separa os irmãos e atualizar o nº de chaves dos nós envolvidos, como na figura abaixo:

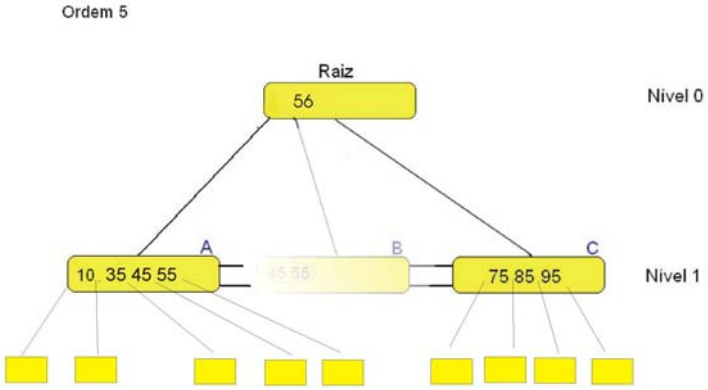


3. Se não foi possível fazer o empréstimo, então será feita uma junção com o irmão da esquerda, se ele existir. O processo é análogo a junção com a direita que está descrita e ilustrada abaixo.

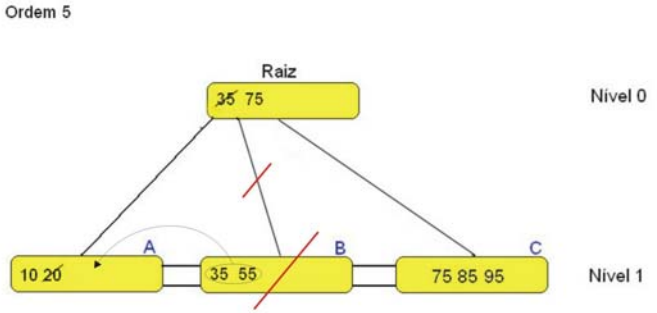
4. Em último caso, faz-se uma fusão com o irmão da direita. Há dois casos possíveis: Se for nó interno, a chave do nó pai que separava os filhos desce para o nó que será mantido e copiam-se as chaves do nó que será apagado para o nó em questão, como sugere a figura abaixo:



Como resultado, obtém-se a seguinte árvore:



Se for nó folha, juntam-se os nós irmãos, exclue-se a chave do nó pai que os separava e ajustam-se os ponteiros, como na figura:



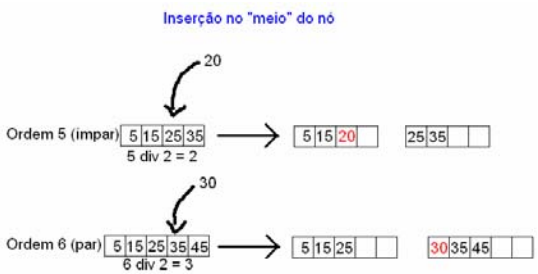
5.1 – Problemas encontrados e nossas soluções

1) Destrutor do nó precisa não só apagar os nós, mas também apagar os DADOs da memória.
Solução: Tratamento diferenciado para nó folha e nó interno. No caso de folha, é preciso varrer o vetor de ponteiros apagando os DADOS da memória.

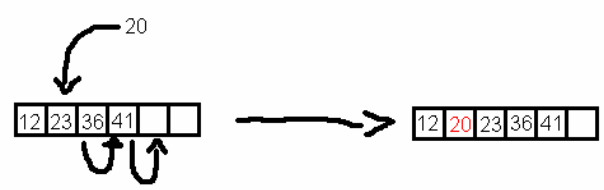
2) É preciso tratar inserções repetidas.
Solução: Antes da inserção, o próprio nó varre todo o vetor de chaves à procura da chave que está sendo inserida. Caso encontre a chave, significa que tal chave já foi inserida previamente, e então nenhuma ação precisa ser tomada, e o controle volta ao escopo do programa principal.

3) Criação de um novo nó folha exige manipulação da lista duplamente encadeada entre os nós folhas.
Solução: O único problema acontece quando ocorre a primeira divisão da raiz, quando esta ainda é folha: não existe vizinho "da direita" para ligar-se ao novo nó, então é preciso tratar esse caso especial.

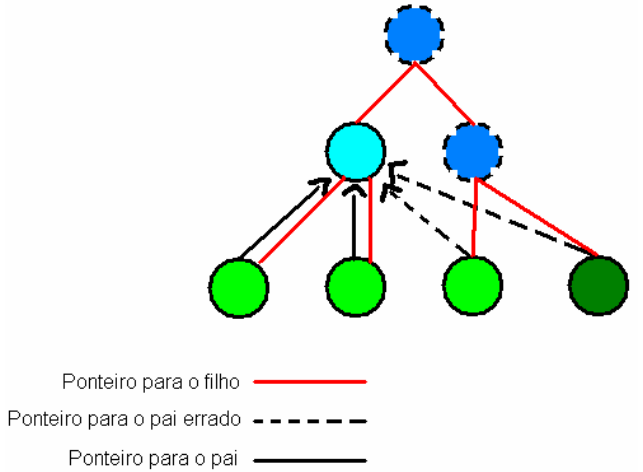
4) Nó precisa de tratamento diferenciado quando a ordem é ímpar ou par em uma inserção na posição ordem div 2.
Solução: Foi criado um fator de correção para esses casos. Supondo um nó "cheio" de ordem 4: em uma inserção na posição 4 div 2, a chave que está sendo inserida é colocada no nó à direita. Supondo a mesma situação com ordem 5, uma inserção na posição 5 div 2 deve ser feita no nó à direita. Veja o exemplo:



5) É preciso inserir ordenadamente os elementos no nó.
Solução: É encontrado o ponto de inserção da chave que deve ser inserida, e então todos os elementos à direita deste ponto são empurrados uma posição para a direita, a começar pelo elemento mais à direita e evoluindo do último elemento até o ponto de inserção.



6) Quando um nó interno se divide, alguns filhos passam a ter o ponteiro para o pai desatualizado.
Solução: é preciso varrer o novo nó atualizando o ponteiro para o pai de todos os seus filhos.



7) No destrutor da árvore é preciso garantir que todos os nós serão apagados.
Solução: foi usada uma função recursiva que apaga a árvore de "baixo para cima" e da "esquerda para a direita", até chegar à raiz.

8) Quando raiz se divide, ponteiro raiz da árvore fica desatualizado.
Solução: Após toda inserção, a árvore é responsável por verificar se a raiz foi alterada.