Resumo 1 - EDII

Aluno: *Nathann Zini dos Reis*

Matrícula: *19.2.4007*

Aula 3 - Introdução sobre pesquisa externa e acesso sequencial indexado

Conceito: A finalidade da pesquisa externa é fazer, de forma mais eficiente, para cada caso, a busca de dados que, devido ao tamanho, estão em memória secundária (ou não estão na memória principal).

Na pesquisa externa, a métrica de complexidades é o número de transferências de dados entre as memórias principal e secundária, diferindo da pesquisa interna que mede a complexidade pelo numero de movimentações e comparações.

Sistema de paginação

É um sistema que consiste em estratégias que pode promover a implementação eficiente de métodos de pesquisa externas.

Um bloco de memória é nomeado de página. Para diminuir o número linear de transferências da memória secundária para a principal, os arquivos em memórias secundárias é divididos em blocos de memórias (páginas) e transferir o bloco inteiro, ao invés de cada um arquivo, diminuindo, portanto, o número de transferência.

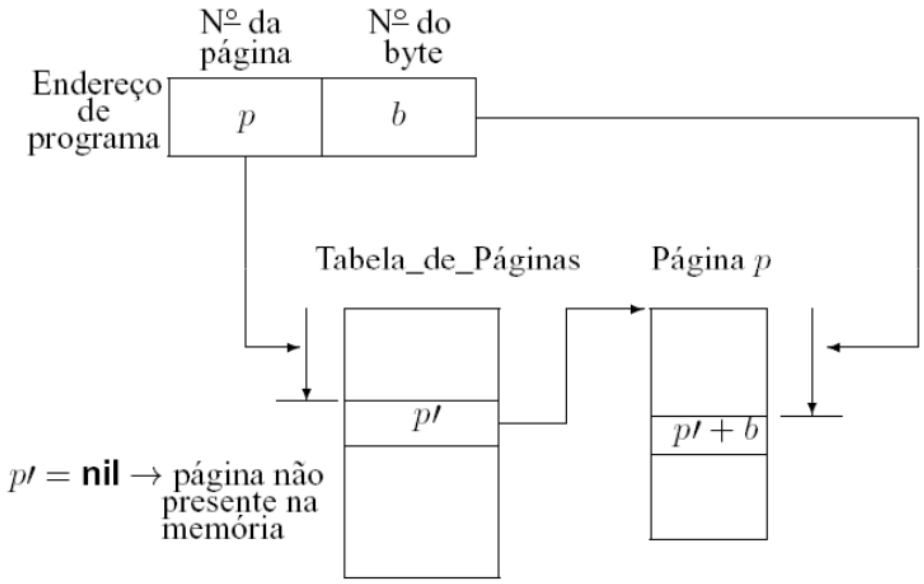
É importante fazer o mapeamento de endereços da memória para saber se o endereço/arquivo que quer procurar/achar em memória secundária já não está em memória principal. A referência para o bloco de memória/paginas é feita por bits que vão representar à pagina e depois ao item dentro da paginas. Ex:

_ _ _ _ | _ _

página | ítem

Método da tabela de páginas para verificar se a página se encontra ou não na memória principal; Nessa tabela vai conter o endereço da página e um valor que indica se ele se encontra na memória principal (#) ou se ele não se encontra na memória principal (NULL).

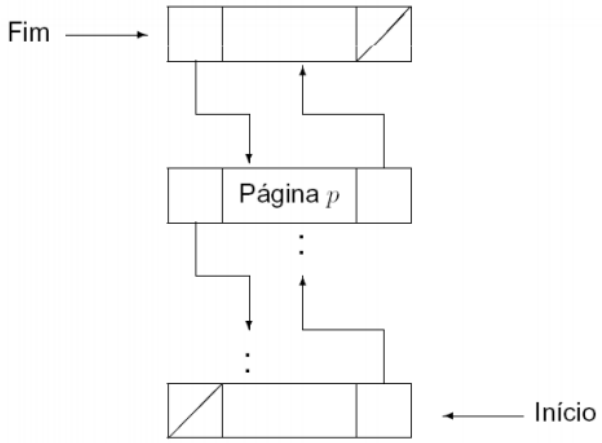
Sistema de Paginação



Caso a página não se encontre na memória principal, deverá, portanto, carregar da memória secundária a página desejada. Caso a memória principal esteja cheia, tem-se que tirar alguma página que esteja lá dentro. Para isso, é melhor e mais correto retirar a página que menos foi utilizada ou foi utilizada por último. Uma estratégia é organizar a tabela de páginas como uma fila, colocando sempre no início da fila a pagina que foi utilizada. Dessa forma, a que estiver em último da fila é a que foi menos utilizada.

Sistema de Paginação

- Políticas de remoção de páginas da memória principal:
  - Menos Recentemente Utilizada (LRU):
    - Remove a página menos recentemente utilizada.
    - Princípio: comportamento futuro deve seguir o passado recente.



Funcionamento:

- Toda vez que uma página é utilizada, ela é colocada no fim da fila.
- A página que está no início da fila é a página LRU.
- A nova página trazida da memória secundária deve ser colocada na moldura que contém a página LRU.

8

Acesso Sequencial Indexado

é um método em que cada item é lido sequencialmente por meio de uma chave de pesquisa (um index). Para aumentar a eficiência o arquivo deve estar ordenado pelo campo chave/index;

Antes de realizar a busca, deve ser feito o "pré processamento" que é um vetor que vai ser armazenado o número da página e a chave do primeiro item (que vai ser o menor valor, pois está ordenado)

3	14	25	41
1	2	3	4

1	3 5 7 11	2	14 17 20 21	3	25 29 32 36	4	41 44 48

Após encontrar qual a página desejada através da verificação pela chave dos itens, deverá carregar essa página para memória principal. Ao abrir o arquivo, o ponteiro estará sempre no início. Portanto deverá mover o ponteiro para a página/bloco desejado. Para tanto, deverá pular o paginas * o tamanho de cada página utilizando o fseek e ler com o fread.

exemplo de estrutura para tabela:

Acesso Sequencial Indexado

```
#include <iostream>
#include <stdio.h>
using namespace std;

#define ITENSPAGINA 4
#define MAXTABELA 100

// definição de uma entrada da tabela de índice das páginas
typedef struct {
   int posicao;
   int chave;
} tipoindice;

// definição de um item do arquivo de dados
typedef struct {
   char titulo[31];  int chave;  float preco;
} tipoitem;

                                        // continuando ...
```

Exemplo de geração de tabela de indice:

```
// gera a tabela de índice das páginas
cont = 0; pos = 0;
while (fread(&x, sizeof(x), 1, arq) == 1) {
    cont++;
    if (cont%ITENSPAGINA == 1) {
        tabela[pos].chave = x.chave;
        tabela[pos].posicao = pos+1;
        pos++;
    }
}                                                    //
```

porém não é o mais eficiente, pois passa por todos os itens desnecessariamente. Uma alternativa é ao invés de ler item por item, lê a pagina toda e coloca a chave do primeiro item lido na tabela;

Ao fazer a pesquisa, é importante verificar se o item que deseja achar não é menor que o primeiro item da primeira pagina e também verificar se a ultima pagina não está cheia.

Aula 4 - Árvore binária de pesquisa externa e árvore B

Conceito de Árvore binária de pesquisa

- é uma estrutura de dados que consiste em um "struct" que contem um dado/item e duas referências para seus "filhos", um à esquerda e um à direita. E cada um desses filhos contém seus próprios itens e referências aos filhos deles. Até o último filho apontar para nulo, ou seja, não tem mais filhos, mas folhas.

- para simular o comportamento do dado da arvore sendo guardado em memória principal, na memoria secundaria, ou arquivo, ao invés de guardar apenas o item, deve se guardar também dois outros valores representando se há ou se não há filhos para aquele nó. caso haja, coloque o valor referente a posição que o filho daquele nó está no arquivo à medida que você vá adicionando valores na árvore.

Introdução à Árvore B

Árvore B é um tipo de árvore n-ária, ou seja, ela pode ter mais que dois filhos, ao contrário da árvore binária.

propriedades de controles de quantidades de filhos por página/nó e itens

- Em uma árvore B de ordem **m**, tem-se:
  - página raiz: contém entre **1** e **2m** itens;
  - demais páginas: contém, no mínimo, **m** itens e **m+1** descendentes e, no máximo, **2m** itens e **2m+1** descendentes;
  - páginas folhas: aparecem todas no mesmo nível;
  - itens: aparecem dentro de uma página em ordem crescente, de acordo com suas chaves, da esquerda para a direita.

Por exemplo, em uma árvore B de ordem 3, o nó raiz, ou página raiz, contém de 1 á 6 itens. as demais páginas contém, nó mínimo de 3 itens e 4 descendentes e, no máximo, 6 itens e 7 descedentes;

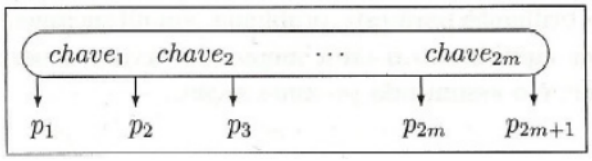
Ou seja, todas páginas da árvore B tem a mesma quantidade máxima de itens; A raiz é a única que pode ter um item só. A quantidade de filhos dependem da quantidade de itens que a página possui, sempre terá 1 filho a mais que a quantidade de itens. Exemplo: 3 itens → 4 filhos...

Para realização de pesquisa dentro de uma árvore B, terá que comparar se o valor está antes do primeiro item da pagina, ou depois do ultimo, ou se é intermediário entre dois itens da mesma pagina, e então seguir para o

página que o apontador em questão referencia até achar o item desejado. Caso não ache e chegue em apontador NULO, é porque a busca não foi sucedida.

A página, ou nó, é guardado na memória através de um struct composto por um vetor de Registro tamanho 2m (cada registro tem uma chave e outros componentes) e outro vetor de Apontador tamanho 2m+1 que é do tipo página capaz de guardar ponteiros que referenciam à outra página

- Forma geral de uma página de uma árvore B de ordem **m**:




```
typedef long TipoChave;

typedef struct TipoRegistro {
    TipoChave Chave;
    /* outros componentes */
} TipoRegistro;

typedef struct TipoPagina* TipoApontador;

typedef struct TipoPagina {
    short n;
    TipoRegistro r[MM];
    TipoApontador p[MM + 1];
} TipoPagina;
```

Pesquisa dentro da Árvore B

Árvore B: Pesquisa

```
void Pesquisa(TipoRegistro *x, TipoApontador Ap)
{ long i = 1;
  if (Ap == NULL)
  { printf("TipoRegistro nao esta presente na arvore\n");
    return;
  }
  while (i < Ap->n && x->Chave > Ap->r[i-1].Chave) i++;
  if (x->Chave == Ap->r[i-1].Chave)
  { *x = Ap->r[i-1];
    return;
  }
  if (x->Chave < Ap->r[i-1].Chave)
  Pesquisa(x, Ap->p[i-1]);
  else Pesquisa(x, Ap->p[i]);
}
```

Pesquisa sequencial para se encontrar o intervalo desejado

Verifica se a chave desejada foi localizada

Ativação recursiva da Pesquisa em uma das subárvores (esquerda ou direita)

26

Ao realizar uma pesquisa, você analisa quantos itens tem dentro da página e ao mesmo tempo, item por item da pagina, analisa se ele é maior que o item desejado para saber se ele está ou não naquela página. Caso não esteja, verifica qual a posição intermediária que o item desejado está dentro da página para saber qual apontador deve ser acessado para chegar no item desejado. Ao achar o item, passa ele por referência e finaliza a recursividade.

Para a impressão da árvore B, basta verificar cada item de cada página se ele possui filho à esquerda até chegar à folha, e imprime os valores. Dessa forma, imprime os valores em ordem crescente.

Caminhamento

Árvore B: Caminhamento

```
void Imprime(TipoApontador arvore){
   int i = 0;

   if (arvore == NULL) return;

   while (i <= arvore->n) {
      Imprime(arvore->p[i]);
      if (i != arvore->n)
         cout << arvore->r[i].Chave << " ";
      i++;
   }

}
```

Aula 5 - Árvore B (inserção)

características da árvore B :

- cada nó, com exceção da raiz, que pode ter 1 item, tem de m a 2m itens e m+1 a 2m+1 filhos.
- todas as folhas estão no mesmo nível, ou seja, a árvore é sempre balanceada.

Princípio da inserção:

- Para inserir um item na árvore, deve-se, similarmente à arvore binaria, percorrer a árvore analisando se o valor é maior ou menor

que o item a ser inserido até chegar na página em que desejamos inserir.

- Se a página em que deseja-se inserir um novo item não estiver vazia, pode-se apenas inserir o novo item nela;
- Caso a página esteja cheia, mas o pai dela não estiver, basta apenas dividir a página em duas criando uma nova pagina, colocando os dois valores maiores na nova pagina, os item novo inserido vai ser inserido na pagina e o valor do meio é subido para o pai desse pagina e o seu apontador a esquerda aponta para a pagina antiga, agora com 2 números (o novo item e o menor item que já existia na pagina anteriormente) e o apontador à direita aponta para a nova página criada com os outros dois maiores valores da página anteriormente.
- Quando tanto a pagina quanto os pais estiverem cheios ao fazer a inserção de um novo item, deverá dividir a raiz em duas paginas e criar uma nova raiz, caracterizando, portanto, o crescimento da árvore. (**A árvore sempre cresce para cima, a fim de manter o balanceamento)**

Processo de Inserção:

```
void Insere(TipoRegistro Reg, TipoApontador *Ap)
{ short Cresceu;
  TipoRegistro RegRetorno;
  TipoPagina *ApRetorno, *ApTemp;
  Ins(Reg, *Ap, &Cresceu, &RegRetorno, &ApRetorno);
  if (Cresceu)  /* Arvore cresce na altura pela raiz */
  { ApTemp = (TipoPagina *)malloc(sizeof(TipoPagina));
    ApTemp->n = 1;
    ApTemp->r[0] = RegRetorno;
    ApTemp->p[1] = ApRetorno;
    ApTemp->p[0] = *Ap;   *Ap = ApTemp;
  }
}
```

- A variável "Cresceu" é responsável para verificar se foi encontrada a página folha (True) para a inserção ou não (False). E também é usada para saber se foi necessário criar uma nova raiz, ou seja, se aumentou o nivel da arvore;
- RegRetorno é o registro que foi/vai ser adicionado na pagina e o apRetorno é o filho à direita do registro retornado e o ApTemp é a nova pagina criada

```
void Ins(TipoRegistro Reg, TipoApontador Ap, short *Cresceu,
         TipoRegistro *RegRetorno,  TipoApontador *ApRetorno)
{ long i = 1; long j;
  TipoApontador ApTemp;
  if (Ap == NULL)
  { *Cresceu = TRUE; (*RegRetorno) = Reg; (*ApRetorno) = NULL;
    return;
  }
  while (i < Ap->n && Reg.Chave > Ap->r[i-1].Chave)  i++;
  if (Reg.Chave == Ap->r[i-1].Chave)
  { printf(" Erro: Registro ja esta presente\n"); *Cresceu = FALSE;
    return;
  }
```

- Primeiramente verifica se a árvore é vazia, se for vazia, indica que cresceu para no método anterior se crie uma nova raiz ou se caso já tenha sido chamada recursivamente, criar uma nova pagina;
- a variável **i** vai representar onde, na página, que irá ser inserido o novo valor;
- verifica se o item já existe;

```
if (Reg.Chave < Ap->r[i-1].Chave) i--;
Ins(Reg, Ap->p[i], Cresceu, RegRetorno, ApRetorno);
if (!*Cresceu) return;
if (Ap->n < MM)    /* Pagina tem espaco */
  { InsereNaPagina(Ap, *RegRetorno, *ApRetorno);
    *Cresceu = FALSE;
    return;
  }
/* Overflow: Pagina tem que ser dividida */
ApTemp = (TipoApontador)malloc(sizeof(TipoPagina));
ApTemp->n = 0;  ApTemp->p[0] = NULL;
```

- Agora verifica se o item vai ser adicionado antes ou depois do próximo valor da pagina e chama recursivamente a própria função para adicionar na página até que haja espaço ou tenha encontrado a folha para então dividi-la em duas folhas/paginas;
- Verifica se a folha foi encontrada, e adiciona na pagina folha caso haja espaço; Se não houver espaço, cria-se uma nova pagina temporária;

```
  if (i < M + 1)
  { InsereNaPagina(ApTemp, Ap->r[MM-1], Ap->p[MM]);
    Ap->n--;
    InsereNaPagina(Ap, *RegRetorno, *ApRetorno);
  }
  else InsereNaPagina(ApTemp, *RegRetorno, *ApRetorno);
  for (j = M + 2; j <= MM; j++)
    InsereNaPagina(ApTemp, Ap->r[j-1], Ap->p[j]);
  Ap->n = M;  ApTemp->p[0] = Ap->p[M+1];
  *RegRetorno = Ap->r[M];  *ApRetorno = ApTemp;
}
```

- Então analisa se o item que deseja inserir é menor ou maior que a primeira metade da pagina folha, casa seja, o ultimo item dessa pagina é inserido na temporaria, e o novo item é inserido na página.
- é passado para a página temporária os demais itens da segunda metade da pagina anterior;
- após isso, a pagina é setada a quantidade de elementos para a quantidade minima M, retorna o item do meio que terá que ir para a pagina pai; e retorna o ponteiro para a nova pagina temporária que foi criada com a segunda metade dos itens da pagina;
- caso a pagina pai esteja incompleta, apenas é adicionada o item que era do meio da pagina filha e o apontador direito apontando pra nova pagina que foi criada; Caso o pai esteja cheio o processo é repetido até encontrar uma pagina que não esteja cheia ou então criar uma nova raiz;

```
void InsereNaPagina(TipoApontador Ap,
                    TipoRegistro Reg, TipoApontador ApDir)
{ short NaoAchouPosicao;
  int k;
  k = Ap->n;  NaoAchouPosicao = (k > 0);
  while (NaoAchouPosicao)
    { if (Reg.Chave >= Ap->r[k-1].Chave)
      { NaoAchouPosicao = FALSE;
        break;
      }
      Ap->r[k] = Ap->r[k-1];
      Ap->p[k+1] = Ap->p[k];
      k--;
      if (k < 1) NaoAchouPosicao = FALSE;
    }
  Ap->r[k] = Reg;
  Ap->p[k+1] = ApDir;
  Ap->n++;
}
```

O inserir na pagina funciona adicionando um novo item na pagina, caso a pagina estiver vazia, e atribuindo null para o apontador a direita desse item e incrementando a quantidade de itens nessa pagina; Caso a pagina contenha algum valor, é verificado em qual posição da pagina o item vai ser adicionado antes de o fazer;

Aula 6 - Árvore B: Remoção

Características da remoção:

- Apenas podem ser removidos itens que estejam localizados nas folhas;

- Importante lembrar que deve-se sempre ser mantido a propriedade de que as páginas devem ter pelo menos M itens cada (não podem ter apenas 1 item);
- Caso ao remover um item da página folha ela fique com menos que o mínimo permitido de itens, deve-se pegar "emprestado" com uma pagina irmã, caso ela possa emprestar (ou seja, tenha pelo menos M+1 itens); desse modo, o valor mais a direita, ou o mais a esquerda dependendo, da pagina irmã, sobe para a pagina pai e o valor mais a esquerda ou o mais a direita anteriormente dessa pagina pai, desce para a página de onde foi removido o item;
- Se nenhuma página irmã puder emprestar um item, a página irmã é fundida com a página cujo item foi removido juntamente com registro que intermedia essas duas páginas;
- A análise é feita novamente para a pagina pai para verificar se ela satisfaz a condição de ter pelo menos M itens; Caso ela não tenha pelo menos M itens, ela pega "emprestado" com a irmã se puder, caso não possa, essa página é fundida com a irmã e o registro intermediária da pagina pai e a verificação é feita novamente com a página pai até que a condição seja satisfeita ou então a raiz fique vazia e seja eliminada, tornando o primeiro filho a nova raiz;
- Caso o item que deseja-se remover não seja um registro da folha, deve-se ser trocado com o registro mais a esquerda do filho a direita ou com o registro mais a direita do filho a esquerda até que ele chegue na folha e seja removido;
- A casos de que uma pagina irmã possa emprestar um registro enquanto a outra irmã teria que fazer a fusão. Localmente, o empréstimo é melhor e mais rápido, porém, a longo prazo, a fusão é melhor no sentido de que diminuirá o número de páginas e eventualmente diminuirá o tamanho/nível da árvore;

```
void Retira(TipoChave Ch, TipoApontador *Ap)
{ short Diminuiu;
  TipoApontador Aux;
  Ret(Ch, Ap, &Diminuiu);
  if (Diminuiu && (*Ap)->n == 0)  /* Arvore diminui na altura */
  { Aux = *Ap;    *Ap = Aux->p[0];
    free(Aux);
  }
}
```

"Diminuiu" indica se uma quantidade menor do que **m** itens passa a ocupar a página.

Método para iniciar a remoção do item; No fim, verifica se a raiz da árvore está vazia. Caso esteja, o primeiro filho é feito como nova raiz e logo a árvore diminui de tamanho;

```
void Ret(TipoChave Ch, TipoApontador *Ap, short *Diminuiu)
{ long j, Ind = 1;
  TipoApontador Pag;
  if (*Ap == NULL)
  { printf("Erro: registro nao esta na arvore\n"); *Diminuiu = FALSE;
    return;
  }
```

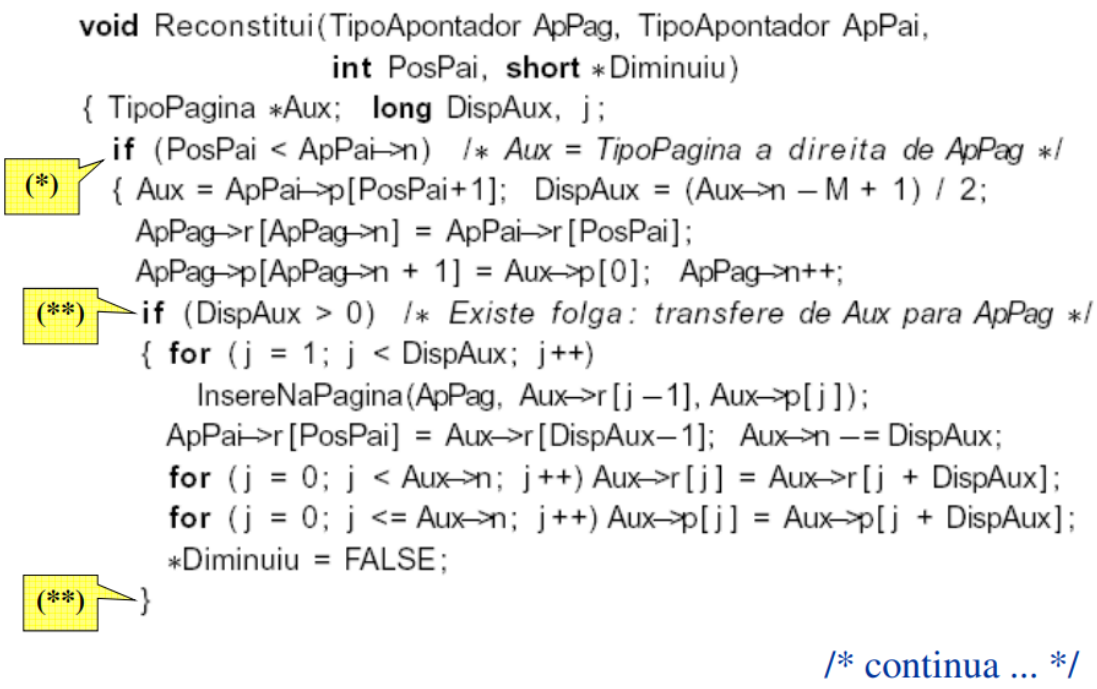
Primeiro verifica se a árvore é nula;

```
Pag = *Ap;
while (Ind < Pag->n && Ch > Pag->r[Ind-1].Chave) Ind++;
if (Ch == Pag->r[Ind-1].Chave)
{ if (Pag->p[Ind-1] == NULL)    /* TipoPagina folha */
  { Pag->n--;
    *Diminuiu = (Pag->n < M);
    for (j = Ind; j <= Pag->n; j++)
      { Pag->r[j-1] = Pag->r[j];  Pag->p[j] = Pag->p[j+1]; }
    return;
  }
  /* TipoPagina nao e folha: trocar com antecessor */
  Antecessor(*Ap, Ind, Pag->p[Ind-1], Diminuiu);
  if (*Diminuiu)
  Reconstitui(Pag->p[Ind-1], *Ap, Ind - 1, Diminuiu);
  return;
}                                          /* continua ... */
```

47

```
  if (Ch > Pag->r[Ind-1].Chave) Ind++;
  Ret(Ch, &Pag->p[Ind-1], Diminuiu);
  if (*Diminuiu) Reconstitui(Pag->p[Ind-1], *Ap, Ind - 1, Diminuiu);
}
```

Primeiro, similarmente à inserção, a árvore é percorrida até que seja encontrada o valor que deseja ser removido; Faz a verificação se a página é folha ou não, caso seja folha, o item é apenas removido e verifica se a pagina tem a quantidade mínima ou não de itens. Caso tenha, nada é feito e a função apenas é finalizada, casa não tenha a quantidade mínima, é chamado o método de reconstituir através do verificador "diminui";



- o valor posPai recebe a posição do ponteiro da pagina filho (Para saber qual a posição que o filho está na página pai);
- é verificado se há filho à direita/ pagina irmã para pegar emprestado ou fundir com ela, usando a posição da pagina filho e verificando a quantidade de elementos que o pai tem; se a posição for menor que a quantidade de itens é porque há paginas irmã a direita;
- é atribuído à aux o apontador da pagina irmã a direita e o DispAux é a quantidade de itens que podem ser emprestados da pagina irmã;
- é verificado então se há itens a ser emprestado, caso tenha é feito a transferência do elemento mais a esquerda da pagina irmã para a pagina pai;
- Caso não tenha itens para emprestar é feito a fusão;

```
else /* Fusao: intercala Aux em ApPag e libera Aux */
  { for (j = 1; j <= M; j++) InsereNaPagina(ApPag, Aux->r[j-1], Aux->p[j]);
    free(Aux);
    for (j = PosPai + 1; j < ApPai->n; j++)
      { ApPai->r[j-1] = ApPai->r[j]; ApPai->p[j] = ApPai->p[j+1]; }
    ApPai->n--;
    if (ApPai->n >= M) *Diminuiu = FALSE;
  }
}
```

(**)

(**)

(*)

- é adicionado na pagina filho todos os elementos da pagina irmã e a pagina irmã é liberada e a pagina pai é remanejada movendo para esquerda todos os elementos seguintes; e verifica se a pagina pai tem a quantidade mínima de elementos. Caso tenha, diminui recebe FALSE e nada é feito, caso não tenha, diminui recebe TRUE e o mesmo processo é feito para a pagina pai;

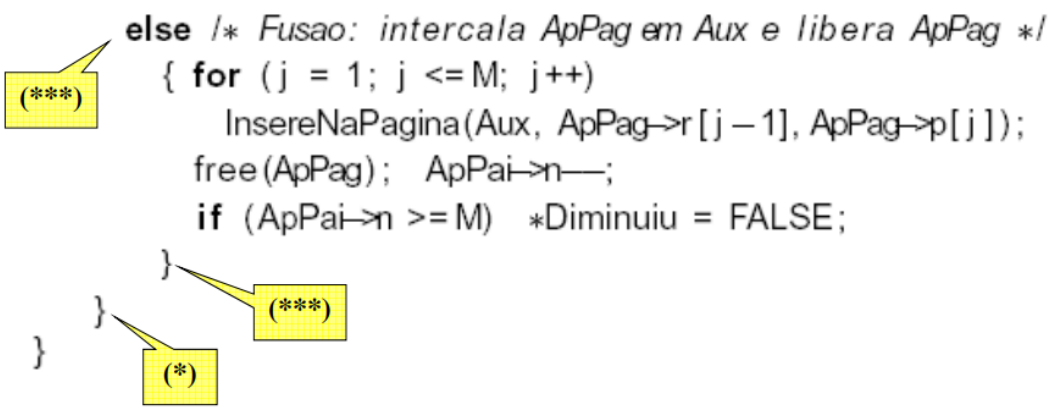
```
else /* Aux = TipoPagina a esquerda de ApPag */
  { Aux = ApPai->p[PosPai-1]; DispAux = (Aux->n - M + 1) / 2;
    for (j = ApPag->n; j >= 1; j--) ApPag->r[j] = ApPag->r[j-1];
    ApPag->r[0] = ApPai->r[PosPai-1];
    for (j = ApPag->n; j >= 0; j--) ApPag->p[j+1] = ApPag->p[j];
    ApPag->n++;
    if (DispAux > 0) /* Existe folga: transf. de Aux para ApPag */
    { for (j = 1; j < DispAux; j++)
        InsereNaPagina(ApPag, Aux->r[Aux->n - j],
                       Aux->p[Aux->n - j + 1]);
      ApPag->p[0] = Aux->p[Aux->n - DispAux + 1];
      ApPai->r[PosPai-1] = Aux->r[Aux->n - DispAux];
      Aux->n -= DispAux;  *Diminuiu = FALSE;
    }
```

(*)

(***)

(***)

/* continua ... */



similarmente ao processo feito com a página irmã à direita, buscando seu elemento mais a esquerda ou então fazendo a fusão, o mesmo é feito para pagina irmã à esquerda, buscando seu elemento mais a direita ou então fazendo a fusão;

Aula 7 - Árvore B*

Árvore B* é uma das várias maneiras de implementar uma árvore B;

é importante lembrar que, apesar do método de implementação da árvore B, todas as propriedades bases da árvore B devem ser mantidas;

Características adicionais:

- Todos os itens de uma árvore B* **devem** estar armazenados em páginas folhas;
- é divida na segunda estrutura:
  - uma estrutura que possuem TODOS os apontadores da árvore até chegar na página folha; Todos os itens dessa estrutura não são itens uteis, ou seja, não estão de fato indexado no arquivo, eles servem de base de comparação para fazer o caminhamento dentro da árvore até chegar nos itens uteis (as paginas folhas), ou seja, são apenas chaves de comparação;

    - Outra diferença é que as chaves dos filhos a esquerda são sempre menores que a chave pai e a da direita não mais é maior, mas maior ou igual a chave pai;
    - opcionalmente, pode-se colocar um apontador em cada página folha apontando para a próxima página folha a fim de fazer acesso indexado sequencial;
        - é mais eficiente para caso que queiras imprimir, por exemplo, todos os itens de uma árvore, já que só é imprimido valores das paginas folhas;
- A pesquisa é feita de modo que sempre para nas paginas folhas, onde estão os itens, diferentemente da árvore B que para ao achar o item em qualquer nível;
- A árvore é armazenada de maneira diferente:
    - 1º parte: paginas internas, referente a estruturas que guardam as chaves e os apontadores. São guardados a quantidade n de chaves na pagina, vetor de chaves e vetor de apontaores;
    - 2º parte: paginas externas, referentes as paginas folhas;. Será guardado a quantidade n de itens e o vetor de registros;
- para tratar o fato de o apontador poder apontar para diferentes tipo de estruturas, usa-se o typedef enum que vai enumerar as possibilidades de um tipo, no caso, tipos de paginas;

```
typedef long TipoChave;

typedef struct TipoRegistro {
    TipoChave Chave;
    /* outros componentes */
} TipoRegistro;

typedef enum {Interna, Externa} TipoIntExt;

typedef struct TipoPagina* TipoApontador;
```

```
typedef struct TipoPagina {
    TipoIntExt Pt;
    union {
        struct {
            int ni;
            TipoChave ri[MM];
            TipoApontador pi[MM + 1];
        } U0;
        struct {
            int ne;
            TipoRegistro re[MM2];
        } U1;
    } UU;
} TipoPagina;
```

- a variável Pt do tipoIntExt recebe os valores predefenidos no typedef enum, ou seja, recebem ou interna ou externa;
- o union limita o uso da estrutura dependendo do tipo do apontador que será controlado pela variável UU; Quando usar-se o U0, será utilizado a primeira estrutura das paginas internas. Analogamente, ao usar-se o U1, será utilizado a segunda estrutura das paginas externas;
- importante perceber que, como as estruturas das paginas internas e externas são diferentes, a ordem de cada pagina não necessariamente deve ser a mesma; Como no exemplo a cima, o tamanho do vetor de registros nas páginas externas pode ser maior que a quantidade máxima de chaves nas paginas internas; exemplo, arvore de ordem m=2 o máximo de itens por pagina internas é de 2m = 4; nas páginas externas podem ser mm2 = 8, por exemplo;

Pesquisa:

```
void Pesquisa(TipoRegistro *x, TipoApontador *Ap)
{ int i;
  TipoApontador Pag;
  Pag = *Ap;
  if ((*Ap)->Pt == Interna)
  { i = 1;
    while (i < Pag->UU.U0.ni && x->Chave > Pag->UU.U0.ri[i - 1]) i++;
    if (x->Chave < Pag->UU.U0.ri[i - 1])
    Pesquisa(x, &Pag->UU.U0.pi[i - 1]);
    else Pesquisa(x, &Pag->UU.U0.pi[i]);
    return;
  }
  i = 1;
  while (i < Pag->UU.U1.ne && x->Chave > Pag->UU.U1.re[i - 1].Chave)
    i++;
  if (x->Chave == Pag->UU.U1.re[i - 1].Chave)
  *x = Pag->UU.U1.re[i - 1];
  else printf("TipoRegistro nao esta presente na arvore\n");
}
```

Pesquisa sequencial na página interna

Ativação recursiva em uma das subárvores: a Pesquisa só pára ao encontrar uma página folha.

Pesquisa sequencial na página folha

Verifica se a chave desejada foi localizada

57

Primeiramente verifica se a pagina em questão é uma pagina interna, caso seja, será feito o encaminhamento similar à arvore B dentro das paginas internas utilizando como parâmetro de comparação as chaves das paginas internas, para saber se caminha pra esquerda ou para a direita, até chegar numa página folha;

Quando encontrado a página folha que corresponde ao intervalo das chaves que esta(ria) o item desejado, será feito uma pesquisa sequencial dentro dessa página externa folha para verificar se a chave/item desejado está dentro da página folha;

Inserção:

- Para a inserção, é essencialmente do mesmo modo que a árvore B. A diferença está quando a página folha é dividida, o elemento do meio "subiria" para a página pai e sairia das paginas folhas criadas. Na árvore B*, deve ser subido APENAS a chave do item do meio, e ele não é eliminado da página folha, mas é mantido lá como elemento, comumente é mantido na pagina folha a direita criada;

Remoção:

- Para a remoção, é similar a remoção de uma árvore B, se mantiver a propriedade da página, o registro é apenas removido da página;
    - Caso deseja-se fazer o empréstimo de uma página irmã, ao invés de mover da pagina irmã pro pai e o elemento do pai pra pagina que precisa do empréstimo, o item é movido direto da página irmã para a pagina que deseja o empréstimo e a chave da pagina pai é atualizada para o menor valor dessa pagina que precisou do empréstimo (ou seja, com o valor da chave do registro que foi "emprestado");
    - No caso da fusão, os itens das duas páginas são juntados em uma única página, a outra página é excluída e a chave da página pai que

antes interligava as duas páginas é apenas excluído da página pai, e o mesmo é feito recursivamente para a pagina pai caso necessário