Universidade Federal
de Ouro Preto

ENYA LUÍSA GOMES DOS SANTOS
19.4.4201

RESUMO I - PESQUISA EXTERNA

Resumo apresentado por exigência da disciplina BCC203 - Estrutura de Dados II, da Universidade Federal de Ouro Preto.
Professor: Guilherme Tavares Assis

Enya Luísa Gomes dos Santos
19.2.4201

Introdução sobre pesquisa externa e acesso sequencial indexado

O que é?

Nem sempre uma quantidade de dados cabe dentro da memória principal, logo se tem a necessidade de armazenar esses dados em memória externa. Entretanto, além de armazenar esses dados, há também a necessidade de encontrar esses dados e trazê-los para memória principal de acordo com a necessidade. Visto isso, é fundamental algoritmos que buscam facilitar a pesquisa desses dados externos.

Quando se faz uma busca em memória principal há diversos métodos, como busca linear, busca binária, entre outros, e esse métodos tem sua eficiência avaliada pelo número de comparações realizada, já na pesquisa externa. a quantidade de transferências de dados entre memórias principal e secundária é a métrica para avaliar os métodos.

Sistema de paginação

É uma estratégia que pode promover a implementação eficiente de métodos de pesquisa externa. Essa estratégia utiliza blocos de dados, que são subdivisões do arquivo com uma quantidade de dados que cabem na memória principal, esses blocos são chamados de em páginas. A memória principal é dividida em molduras de páginas de igual tamanho das páginas da memória secundária. Desse modo, a complexidade de transferência é reduzida para o número de páginas e não para o número dos *n* dados.

Tipos de páginas:

- **Páginas ativas** - páginas que se encontram na memória principal.
- **Páginas inativas** - páginas que ainda se encontram em memória secundária e ainda não foram para a memória principal, e no momento que houver necessidade elas irão para a principal.

O mecanismo de sistema de paginação possui duas funções principais são elas:

- **Transferência de páginas** - onde páginas da memória secundária serão levadas para memória principal, e quando finalizada sua utilidade voltarão para memória secundária.
- **Mapeamento de endereços** - determinar qual página da memória secundária um programa está endereçando, e encontrar a moldura de tal página na memória principal, caso exista.

Endereçamento de um item

Os itens estão contidos dentro de uma página, deste modo é necessário uma referência por uma sequência binária para referenciar esses itens. Onde a primeira parte bits representa o número da página na qual ele se encontra e a segunda parte representa a posição do item dentro da sua página. Por exemplo, imaginando um conjunto de dados em um arquivo, se decidido que cada item será referenciado por seis bits, onde os 4 primeiros diz respeito a página e os dois últimos ao item dentro dessa página, temos uma possibilidade de haver no máximo 16 páginas (2^4) e no máximo 4 itens por página (2^2). Exemplo: 001011.

Logo é importante atentarmos a quantidade de bits, pois dependendo da quantidade de dados as possibilidades de bits para mapeamento não serão suficientes.

Esse mapeamento é dados por uma tabela de páginas - podendo ser um vetor - , onde nela será armazenado o endereçamento das páginas e itens. Esse mapeamento serve para fazer uma correspondência entre as páginas que estão no arquivo e as páginas que estão, naquele momento, em memória principal. Representação na imagem abaixo:

Enya Luísa Gomes dos Santos
19.2.4201

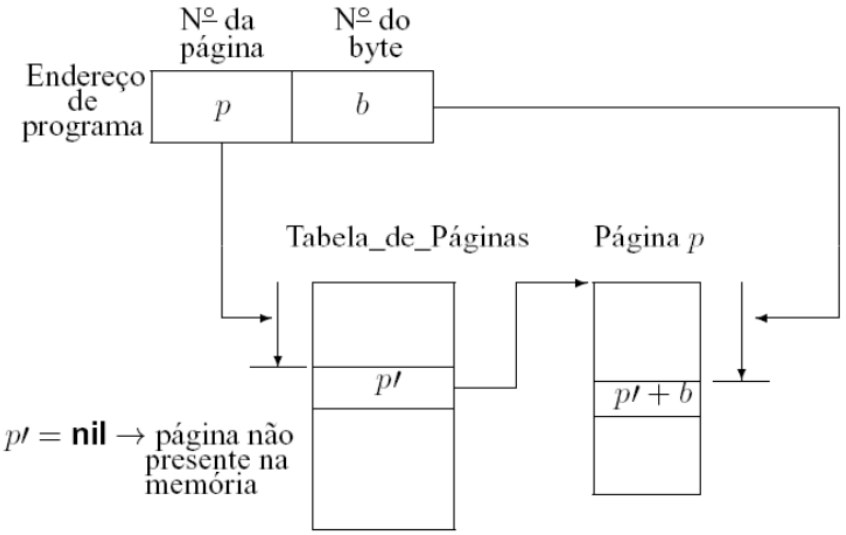

Figura 01 - Endereçamento de página

Quando acontecer de um programa precisar de uma página *p* que não esteja na memória principal (p' = null), a página p deve ser trazida da memória secundária para a memória principal e a tabela de páginas deve ser atualizada. Se não houver uma moldura de página vazia, uma página deverá ser removida da memória principal. O ideal é remover a página que não será referenciada pelo período de tempo mais longo no futuro, e para termos uma noção disso, são utilizadas as políticas de remoção.

Políticas de remoção de páginas da memória principal

- Menos Recentemente Utilizada (LRU):
  - Um dos algoritmos mais utilizados
  - Remove a página menos recentemente utilizada
- Menos Frequentemente Utilizada (LFU):
  - Remove a página menos frequentemente utilizada.
  - Desvantagem: uma página recentemente trazida da memória secundária tem um baixo número de acessos e pode ser removida.
- Ordem de Chegada (FIFO):
  - Remove a página que se encontra na memória principal há mais tempo.
  - Algoritmo mais simples e barato de manter.

- Desvantagem: ignora o fato de que a página mais antiga pode ser a mais referenciada

Em suma, através de uma memória externa com dados, esses são divididos em páginas, com o endereçamento de itens cria-se molduras de páginas na memória principal e há a política vinculada a tabela de página e sistema de remoção que facilita o controle de forma eficiente do sistema de paginação. O objetivo do sistema de paginação é diminuir a quantidade de acessos que se fazer na memória secundária para trazer dados à memória principal, e pode ser implementado em qualquer método de pesquisa externa, por exemplo, em conjunto com algum método de pesquisa externa, o sistema de paginação pode ser adicionado com a finalidade de diminuir a quantidade de acessos que se fazer na memória secundária.

Acesso Sequencial Indexado

É um método de pesquisa externa. Utiliza o princípio da pesquisa sequencial através de um índice, e esse facilita e melhora muito a pesquisa. Tal método tem o pré-requisito que exige que os dados estejam ordenados pelo campo chave.

O primeiro passo desse método é construir um índice de páginas, que consiste em um vetor com duas informações:

1. A primeira sendo o número da página
2. A segunda a chave de menor valor da página, ou seja a chave do primeiro item, já que está ordenado.

Exemplo abaixo de um índice de páginas:

8	16	21	37	44
1	2	3	4	5

Figura 02 - Exemplo de um índice de páginas

A busca/pesquisa é feita através desse índice de páginas comparando o valor do segundo item com o valor item a ser pesquisado (verificado se é maior,

menor ou igual), e assim será possível encontrar a página em que o item está pesquisado para que essa seja carregada em memória principal.

Com o índice da página encontrado, precisa-se encontrar essa na memória secundária e para pular as páginas até a página a ser carregada na memória principal, pode-se usar o fseek do início até (qtd. de páginas * qtd. de itens na página * sizeof do item). Com isso a página pode ser carregada na memória principal (com fread) em um vetor e se faz uma varredura para encontrar o item, pode ser, essa pesquisa pode ser sequencial (Linear O(n)) quando a quantidade de itens é pequena, ou pode ser uma pesquisa binária (Logarítmica O(log n)) para quantidade de itens grandes já que os dados estão ordenados.

Implementação

É necessário o tipo *item* e o tipo da estrutura da tabela. Pode-se representar o vetor de índice de páginas apenas guardando as chaves e as páginas poderão ser representadas pelo próprio índice do vetor.

Código da implementação bruta do método de 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; //página
	int chave; //chave do item
} tipoindice;

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

int pesquisa (tipoindice tab[], int tam, tipoitem* item, FILE *arq) {
	tipoitem pagina[ITENSPAGINA];
	int i, quantitens;
	long desloc;
	// procura pela página onde o item pode se encontrar
```

```
	i = 0;

	while (i < tam && tab[i].chave <= item->chave) i++;
	// caso a chave desejada seja menor que a 1a chave, o item
	// não existe no arquivo
	if (i == 0) return 0;
	else {
	// a ultima página pode não estar completa
		if (i < tam) quantitens = ITENSPAGINA;
		else {
			fseek (arq, 0, SEEK_END);
			quantitens = (ftell(arq)/sizeof(tipoitem))%ITENSPAGINA;
//assim quando a página está totalmente completa
			//o mod será 0, resolvendo esse problema no proximo if
			if(quantitens == 0) quantitens = ITENSPAGINA;
		}
		// lê a página desejada do arquivo
		desloc = (tab[i-1].posicao-1)*ITENSPAGINA*sizeof(tipoitem);
		fseek (arq, desloc, SEEK_SET);
		fread (&pagina, sizeof(tipoitem), quantitens, arq);

		// pesquisa sequencial na página lida
		for (i=0; i < quantitens; i++)
			if (pagina[i].chave == item->chave) { // poderia retornar 0
quando o pagina[i].chave > item->chave, pois desse modo nn existe.
				*item = pagina[i]; return 1;
			}
		return 0;
	}
}

int main () {
	tipoindice tabela[MAXTABELA];
	FILE *arq;
	tipoitem x;
	int pos, cont;

	// abre o arquivo de dados
	if ((arq = fopen("livros.bin","rb")) == NULL) {
		printf ("Erro na abertura do arquivo\\n");
		return 0;
	}

	// gera a tabela de índice das páginas
	cont = 0; pos = 0;
	while (fread(&x, sizeof(x), 1, arq) == 1) {
		cont++;
		if (cont%ITENSPAGINA == 1) { //será verdadeira somente quando
estiver pegando o primeiro item da página
```

```
                    tabela[pos].chave = x.chave;
                    tabela[pos].posicao = pos+1;
                    pos++;
              }
       }

       //A função acima pode ser melhorada, pois ela lê todos os arquivos mas nn
usa todos os dados, uma solução seria
       //usar um fseek, a outra seria ler 4 itens no fread e na tabela guarda o
x na posição 0. A melhor seria a opção
       //2 pois ela faz menos acessos a memoria secundaria só com o fread e não
com fread + fseek.

       fflush (stdout);
       printf("Código do livro desejado: ");
       scanf("%d", &x.chave);
       // ativa a função de pesquisa
       if (pesquisa (tabela, pos, &x, arq))
              printf ("Livro %s (codigo %d) foi localizado", x.titulo, x.chave);
       else
              printf ("Livro de código %d nao foi localizado",x.chave);

       fclose (arq);
       return 0;
}
```

Árvore binária de pesquisa externa e árvore B

Árvore binária de pesquisa

Uma árvore binária é formada por nodos, onde esses possuem dois filhos, um filho na esquerda e um filho na direita, essa estrutura facilita a pesquisa. O exemplo abaixo representa uma árvore binária de pesquisa, onde os itens com o valor menor os pais estão à direita e com valor maior à esquerda.

Enya Luísa Gomes dos Santos
19.2.4201

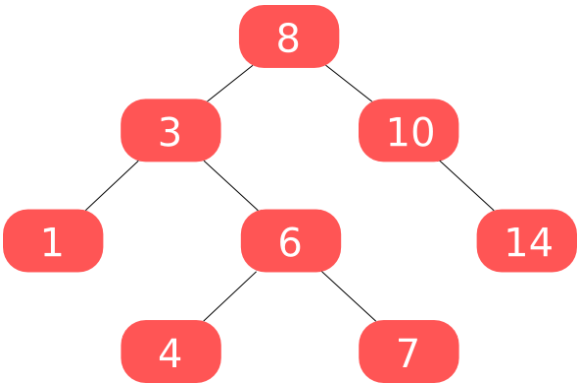

Figura 03 - Árvore binária de pesquisa

Exemplificação da transformação de um arquivo de dados externo em um arquivo de dados externo que simula uma árvore binária em memória secundária na imagem abaixo:

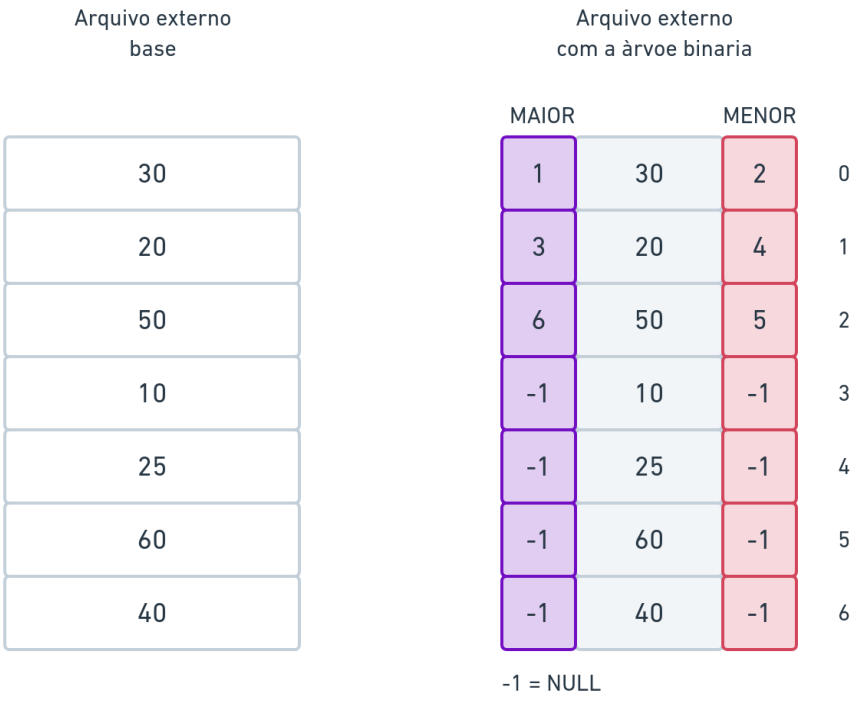

Figura 04 - Transformação de um arquivo externo inicial para uma arquivo de árvore binária externa

A complexidade de comparações dessa busca também é O(log n) assim como uma árvore binária de pesquisa em memória principal. Já a complexidade de número de transferência (leitura) também é O(log n).

Para diminuir o número de transferência entre a memória secundária e a memória principal fim de economizar, os nodos de uma árvore podem ser agrupados em páginas, como apresentado no exemplo abaixo:

Enya Luísa Gomes dos Santos
19.2.4201

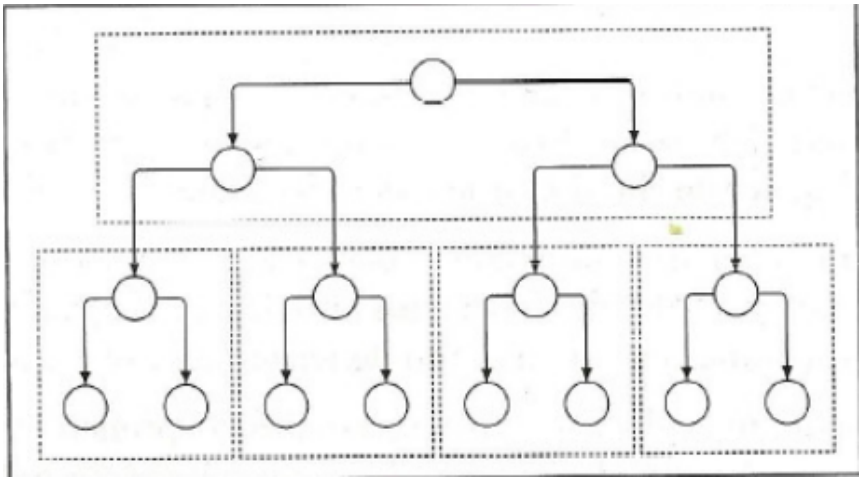

Figura 05 - Página de árvores

Porém uma organização ótima é difícil de ser obtida durante a construção da árvore (problema complexo de otimização).

O algoritmo com alocação sequencial armazena os nodos em posições consecutivas na página à medida que vão surgindo, não considerando o formato físico da árvore, esse método tem a vantagem de que utiliza todo o espaço disponível na página, mas também possui a desvantagem de que os nodos de uma página estão relacionado pela ordem de entrada dos itens e não pela localidade na árvore, como por exemplo, o node pode estar na posição 3 do arquivo e seu filho à esquerda na posição 46, esse fato piora o tempo de pesquisa de forma bem considerável.

A fim de resolver esse problema, Muntz e Uzgalis (1970) propuseram um método de alocação de nodos em páginas que considerasse a proximidade dos nodos dentro da árvore. Esse método segue algumas regras:

- O novo nodo é sempre colocado na mesma página do nodo pai.
- Se a página do nodo pai estiver cheia, o novo nodo é colocado no início de uma nova página criada.

Sua vantagem é que o número de acessos na pesquisa é próximo do ótimo, e sua desvantagem se diz respeito a ocupação média das páginas, em que essa é baixa, e as páginas podem conter número diferentes de itens. A partir desse problema surgiu a solução chamada árvore B.

Enya Luísa Gomes dos Santos
19.2.4201

Árvore B

Foi criada em 1972, e é muito utilizada até hoje. A árvore B mantém o crescimento da árvore equilibrado e permite inserções e retiradas. Nesse tipo de árvore a complexidade é logarítmica, o diferencial é que, em nó (página) ocorre uma varredura, essa sendo no máximo O(n).

É uma árvore n-ária, ou seja, os nodos podem possuir mais de dois filhos, por esse motivo os nodos são chamados de páginas.

Árvore B de ordem m:

- 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.

Exemplificação, em uma árvore B de ordem m = 2 com três níveis, todas as páginas contêm de 2 a 4 itens, exceto a raiz que pode conter de 1 a 4 itens.

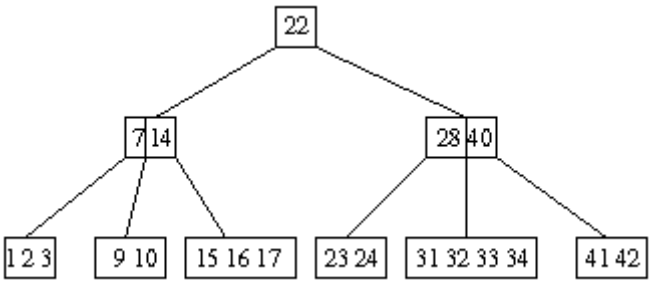

Figura 06 - Árvore B

Dentro de uma determinada página se guarda:

1. A quantidade de itens dentro da página
2. Os apontadores para os filhos
3. Os itens da página

Enya Luísa Gomes dos Santos
19.2.4201

Figura 07 - Estrutura de um tipo página

Código de uma árvore B (Inicializar, Pesquisar, Caminhar):

```
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;

void Inicializa (TipoApontador Arvore)
{
	Arvore = NULL;
}

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]);
}

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++;
        }
}
```

Árvore B - Inserção

Como se cria uma árvore B?

A árvore B cresce para cima a fim de garantir seu balanceamento.

Quando se quer inserir um novo elemento e há espaço para ele na página, o processo é bem parecido com a árvore binária, tendo o detalhe de inserir o item na página de forma ordenada, já quando o objetivo é inserir um novo item um uma folha que já está com o número máximo de itens, a funcionamento é o seguinte, ocorre a divisão dos itens no meio, a parte com o itens maiores passaram a ser uma nova folha, os dos itens menores permanece na folha e o item do meio passa a pertencer a sua página pai. Garantindo assim a propriedade da árvore.

Exemplo de inserção de itens em uma árvore B:

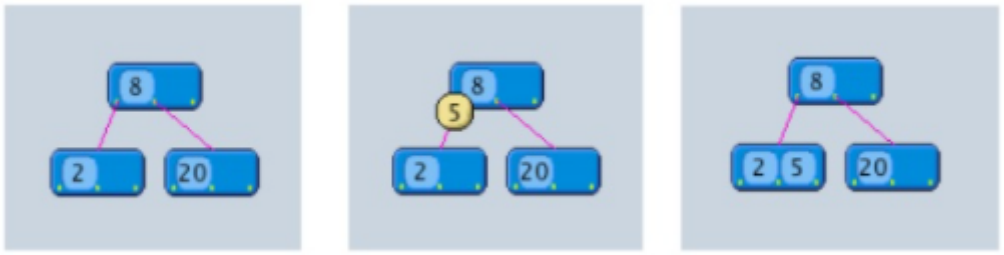

Figura 08 - Inserindo o 5 na árvore B. Fonte:
https://pt.slideshare.net/ThiagoChaves/apresentao-sobre-rvores-b

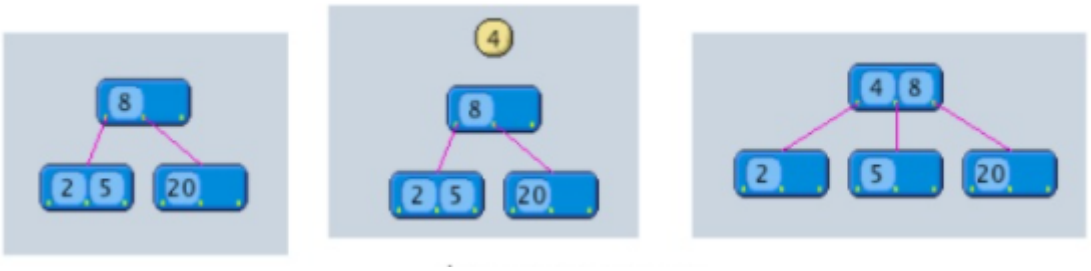

Figura 09 - Inserindo o 4 na árvore B. Fonte:
https://pt.slideshare.net/ThiagoChaves/apresentao-sobre-rvores-b

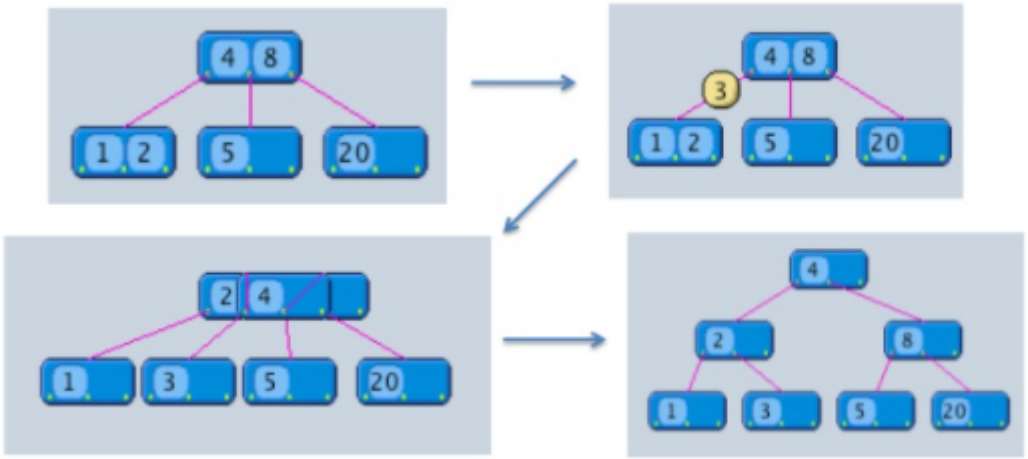

Figura 10 -Inserindo o 3 na árvore B. Fonte:
https://pt.slideshare.net/ThiagoChaves/apresentao-sobre-rvores-b

Em suma, em uma árvore B, temos duas situações:

1. **A página adequada para o item ser inserido não está cheia**, logo basta, simplesmente, inserir o item no array dos itens.
2. **A página adequada para o item ser inserido já está cheia**, nessa

situação cria-se uma nova página com esse item, divide a página que está cheia, o elemento do meio vai para a página pai e os elementos que sobraram na divisão vão para a nova página, e essa se torna filho do item do meio que agora se encontra na página pai.

Árvore B - Remoção

A primeira coisa a ser verificada em uma remoção, é se o item a ser removido está em uma página folha, pois, só é possível remover itens que estejam presentes dentro de uma página folha.

Uma pesquisa é feita para se encontrar o item a ser retirado, caso o item a ser retirado não esteja em uma página folha, antes de fazer a retirada o item é substituída por sua chave antecessora (que está na árvore mais à direita da subárvore à esquerda) ou sucessora (que está na árvore mais à esquerda da subárvore à direita), caso esteja em uma página folha ele será retirado de forma mais simples.

É importante destacar que as propriedades da árvore B têm que ser mantidas durante a remoção, enquanto essas propriedades não sejam atendidas o processo de remoção não finaliza.

Simplificando a estratégia

O primeiro passo é verificar se o item a ser removido está uma uma página folha, se for uma página folha ocorra a remoção do item e o vetor é rearranjado. Se a página onde ocorreu a remoção tiver perdido suas propriedades, olha-se a página irmã vizinha, se a página irmã vizinha puder emprestar um item, o item é emprestado, se não ocorre a fusão das duas páginas. Caso o item não esteja em uma página folha, o item é substituído pela chave da árvore antecessora ou pela sucessora.

Exemplificação de casos:

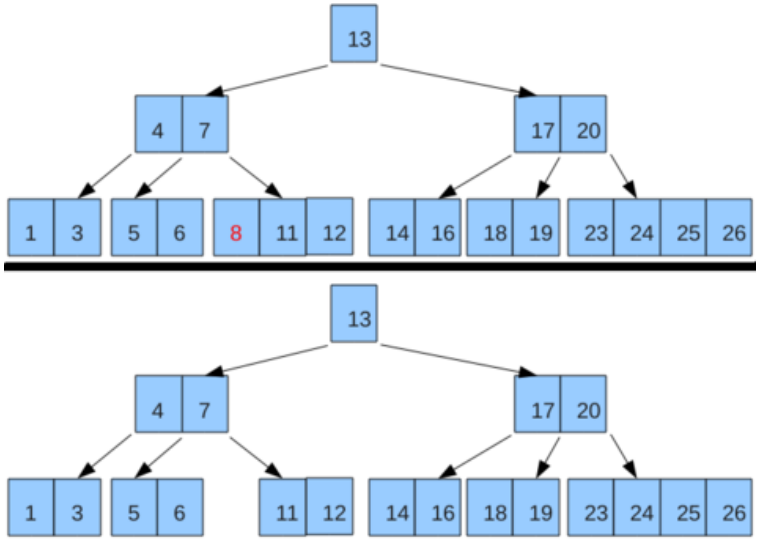

Figura 11 - Remoção da chave 8 e posterior reorganização da estrutura. Fonte: https://www.wikiwand.com/pt/%C3%81rvore_B#/Remo%C3%A7%C3%A3o

Neste caso, a remoção da chave 8 não causa o underflow na página folha em que ela está, portanto ela é simplesmente apagada e as outras chaves são reorganizadas mantendo sua ordenação.

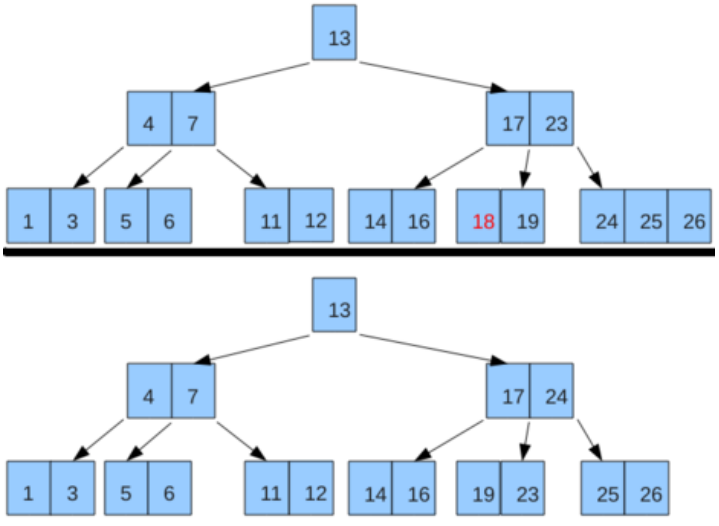

Figura 12 - Remoção da chave 18 e posterior redistribuição das chaves. Fonte: https://www.wikiwand.com/pt/%C3%81rvore_B#/Remo%C3%A7%C3%A3o

No caso desta figura é apresentado a técnica de redistribuição de chaves. Na remoção da chave 18, a página que contém essa chave possui uma página irmã à direita com um número superior ao mínimo de chaves (página com chaves 24, 25 e

26) e, portanto, estas podem ser redistribuídas entre elas de maneira que no final nenhuma delas tenha um número inferior ao mínimo permitido.

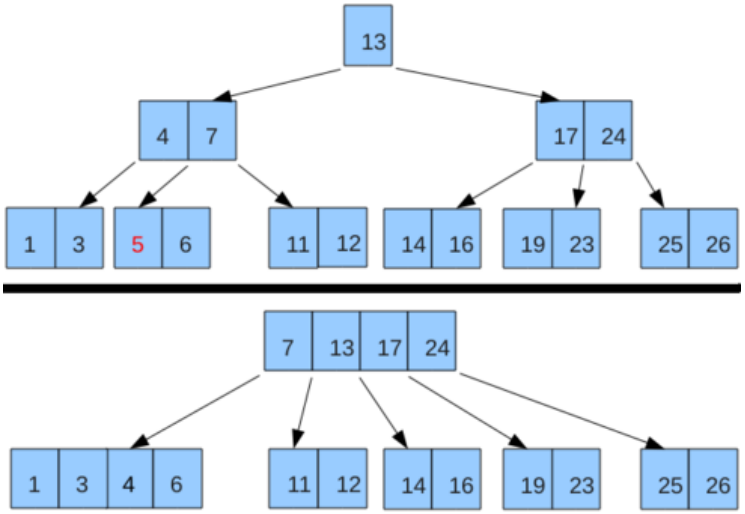

Figura 13 - Remoção da chave 5 e posterior concatenação com página irmã à esquerda. Fonte: https://www.wikiwand.com/pt/%C3%81rvore_B#/Remo%C3%A7%C3%A3o

Nesta figura foi removida a chave 5, como não foi possível utilizar a técnica de redistribuição, pois as páginas irmãs possuem o número mínimo de chaves, então foi necessário concatenar o conteúdo da página que continha a chave 5 com sua página irmã à esquerda e a chave separadora pai. Ao final do processo a página pai fica com uma única chave (underflow) e é necessário diminuir a altura da árvore de maneira que o conteúdo da página pai e sua irmã, juntamente com a raiz, sejam concatenados para formar uma página única.

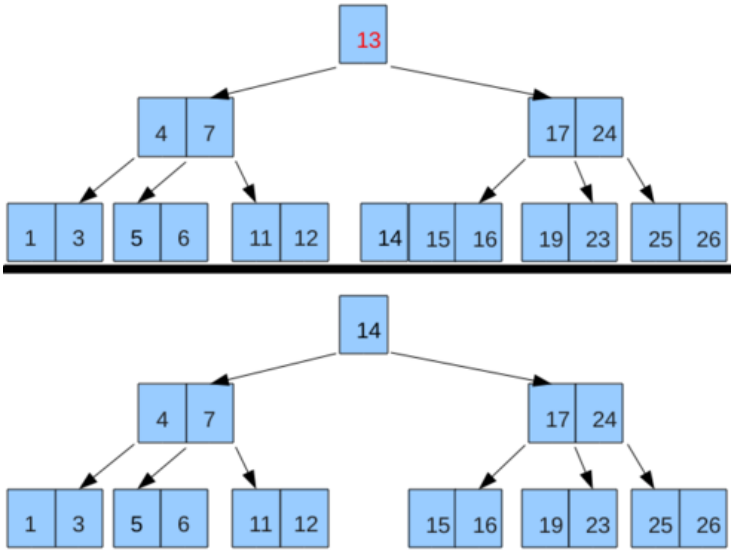

Figura 14 - Remoção da chave 13 e promoção da menor chave da subárvore à direita de 13. Fonte: https://www.wikiwand.com/pt/%C3%81rvore_B#/Remo%C3%A7%C3%A3o

A remoção da chave 13 nesse caso foi realizada com a substituição do 13 pelo menor número da subárvore à direita de 13 que era o 14. Essa troca não causou o underflow da página em que estava o 14 e, portanto, não gerou grandes alterações na árvore.

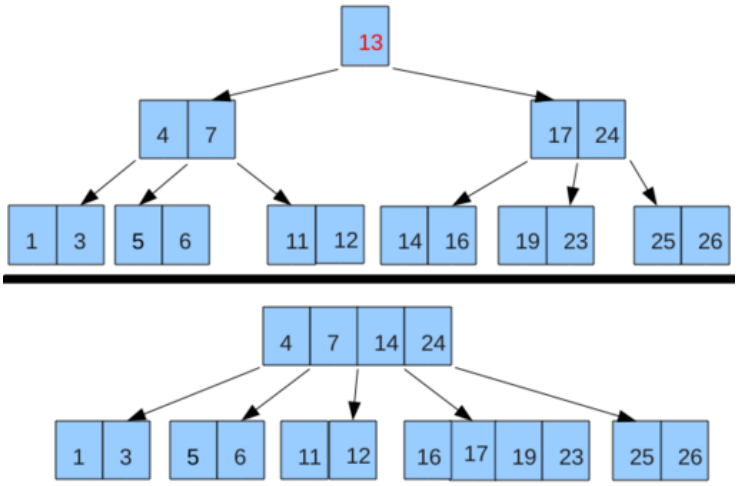

Figura 15 - A Chave 14 é promovida para a raiz, o que causa underflow em sua página. Fonte: https://www.wikiwand.com/pt/%C3%81rvore_B#/Remo%C3%A7%C3%A3o

Caso semelhante ao anterior, mas esse ocorre o underflow da página que contém a menor chave da subárvore à direita de 13. Com isso, como não é possível a redistribuição, concatena-se o conteúdo desta página com sua irmã à direita o que gera também underflow da página pai. O underflow da página pai também é resolvido com a concatenação com sua irmã e a raiz, resultando na diminuição da altura da árvore.

Árvore B*

É uma forma de se implementar a árvore B, ou seja, uma extensão da árvore B. A árvore B* segue as características de uma árvore B.

Características:

- Todos os itens estão armazenados no último nível, ou seja, o nível das páginas folha.
- Os níveis acima do último nível constituem um índice cuja organização é a de uma árvore B.

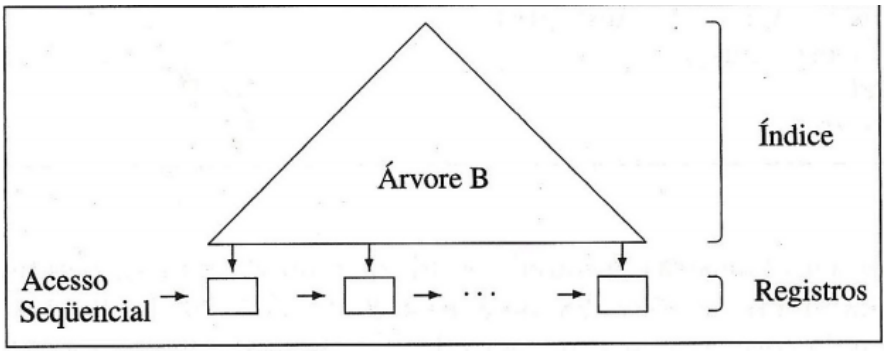

Figura 16 - Estrutura de uma árvore B*

Uma característica **adicional** é que, como os registros estão todos em páginas folhas, é possível fazer o encadeamento entre essas páginas como uma lista.

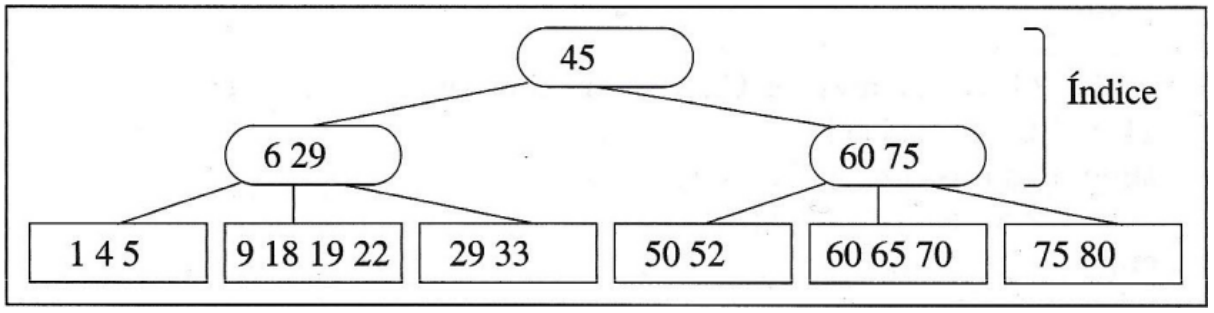

Figura 17 - Árvore B*

Os itens dispostos nos níveis de índice não necessariamente representam registros que existem no arquivo, esses registros estão todos dentro das páginas folhas. Os índices representam apenas uma ferramenta para buscar os registros úteis, ou seja, existentes.

Dentro de uma página que não seja folha (página interna), se guarda um vetor de chaves, apontadores para o filho e um *n* quantidades de chaves, já dentro das páginas folhas (página externa), se guarda um vetor de registros e *n* quantidades de itens.

Obs.: Em uma página externa, seu filho pode estar apontando tanto para um ponteiro do tipo da página interna, quanto para um ponteiro do tipo da página externa. Para isso a estrutura de uma árvore B* trabalha com uma estrutura seletiva, como pode ser visto no código a seguir.

```
typedef long TipoChave;

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

```
//tipo enumerado formado por dois valores, Interna ou Externa
typedef enum {Interna, Externa} TipoIntExt;

typedef struct TipoPagina* TipoApontador;

//Exemplo de acesso à um registro seletivo:
//  Ap->UU.U0.ni
//  Ap->UU.U1.ne
```

A pesquisa em uma árvore b* ocorre da seguinte forma, a árvore é percorrida - assim como em uma árvore b - através das chaves da páginas internas, até encontrar uma página externa onde, se existir o elemento, seu registro estará lá, e a busca dentro da página externa é feita em seu vetor. Código da pesquisa em uma árvore b*:

```
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

Figura 18 - Algoritmo de pesque em uma árvore B*

Árvore B* - Inserção

Semelhante à inserção da árvore B, com a diferença de que o tratamento para páginas podem ser diferentes com o registro seletivo, como por exemplo, tratar

uma página interna (não folha) e uma página externa (folha).

Em uma árvore B*, o que muda em relação à árvore B é que, é quando na página adequada para inserir o item já está cheia, já quando há espaço o comportamento é o mesmo de uma árvore B. E essa mudança é que, o **elemento do meio, além de subir para uma página pai, ele deve, também, continuar na nova página criada.**

Árvore B* - Remoção

Também é semelhante à remoção da árvore B, com a diferença de que o tratamento para páginas podem ser diferentes com o registro seletivo, como por exemplo, tratar uma página interna (não folha) e uma página externa (folha).

A diferença ocorre quando a propriedade da árvore b* é violada, caso isso não ocorra, a remoção ocorre de forma simples dentro da página folha. A diferença é que o elemento da página pai não irá “descer” para a página externa no momento do **empréstimo**. O item emprestado também vai virar um índice (o primeiro item da página fundida), ocupando o lugar de seu item pai.

Já na **fusão**, quando a irmã não pode emprestar um item, o item pai não irá descer como em uma árvore b, o que ocorre é que, a fusão ocorre da mesma forma, mas o elemento do pai não desce e sim é retirado da página interna.

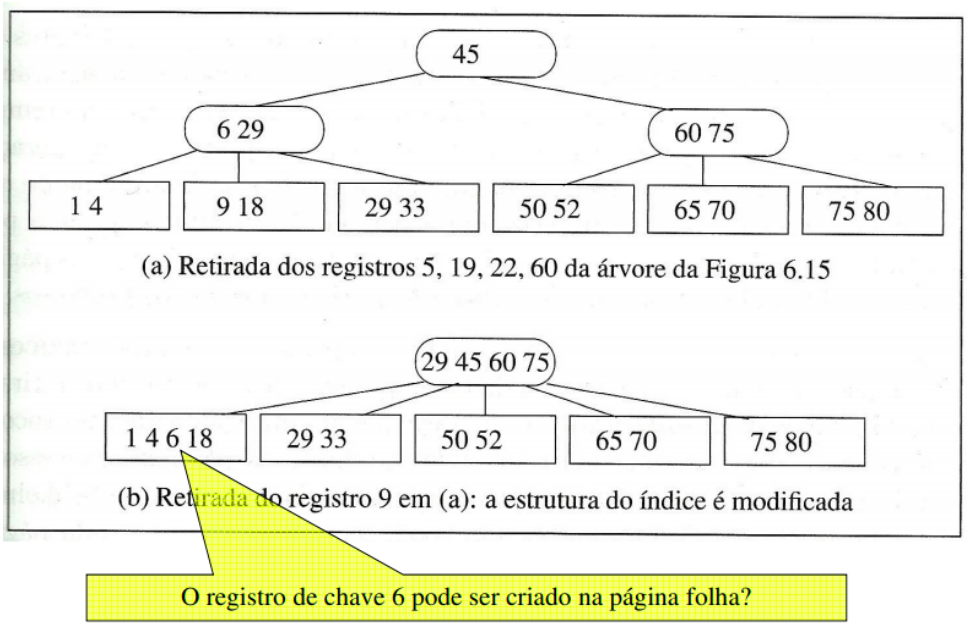

Figura 19 - Inserção de uma árvore b* de forma incorreta