Pesquisa Externa

→ Medida de Complexidade: comparações e, principalmente, transferências entre as memórias secundárias e principal.

- secundária → sequencial
- primária → direto

→ Sistema de paginação: subdivisão de um arquivo em páginas que correspondam a um bloco que caiba em memória principal.

- mapeamento de endereços
- transferência de páginas

Paginação do arquivo → consulta tabela de páginas → * página na memória principal

*LRU: Least Recently Used → enfileiramento de páginas

*LFU. Least frequently Used

* FIFO. First In First Out

→ Acesso Sequencial Indexado: leitura sequencial até o buscado ou maior. Requer ordenação prévia e um arquivo índice de páginas.

Localizar a página que pode conter o item a partir do índice (chave do primeiro item da página) → Realizar uma pesquisa sequencial na página localizada

→ Árvore Binária de pesquisa externa: cada elemento possui dois apontadores (inteiros) para seus nós filhos. Desta forma, para acessar o maior filho de um nó pai, basta apontar o valor inteiro * sizeof(registro). Possui a mesma ordem de complexidade que uma árvore em memória principal: log n

• O sistema de paginação também vai aplica a esse método e economiza acessos à transferências entre as memórias. Todavia, a economia não é constante haja vista que páginas muitas vezes estarão incompletas.

→ Wuntz e Ugalis otimizaram o sistema de paginação ao considerar a proximidade dos nós de uma árvore. Toda. Via, a ocupação média das páginas é baixa. Solução: Árvore B.

→ Árvore B. proposta para equilibrar o crescimento da árvore e permitir inserções e retirados.

* Árvore n-ária: possui mais de dois filhos

• página raiz → entre 1 e 2m itens

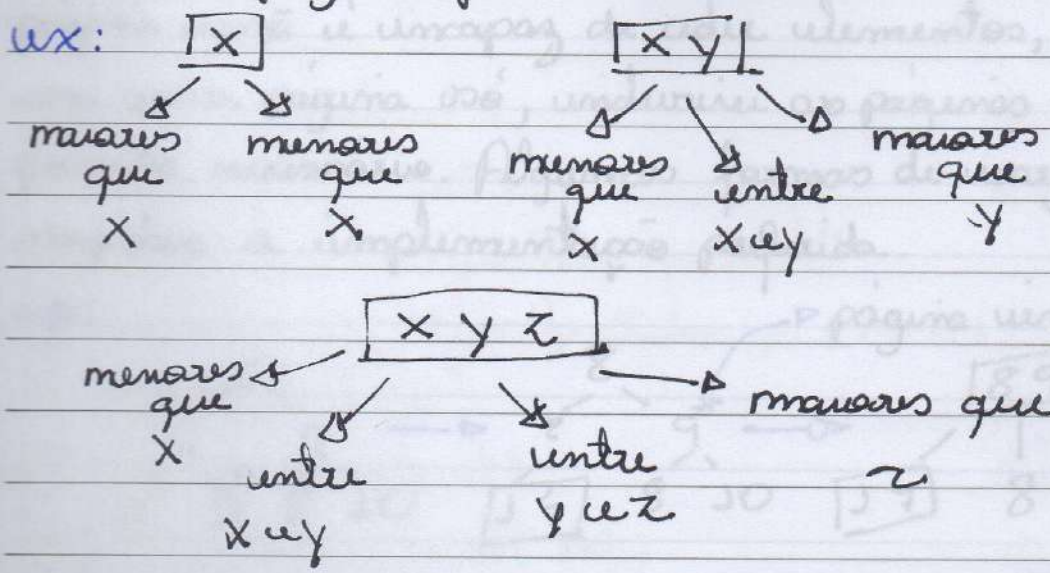
• demais páginas → mínimo m itens e m+1 descendentes
e máximo 2m itens e 2m+1 desc.

• páginas folhas → todas no mesmo nível, ou seja, impossível de desbalancear.

• itens → ordem crescente.

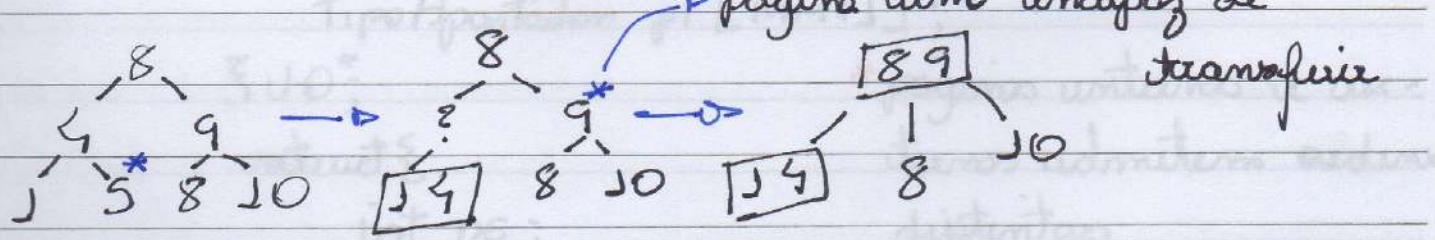
• todas as páginas possuem de 2 a 4 itens.

ex:



• forma geral de ordem m. chaves [2m]
apontadores [2m+1]

* Inserção: crescimento vertical para cima garante o balanceamento ao fazer com que todos nós folhas permaneçam no mesmo nível. Para inserir um elemento em uma página vazia, basta proceder como uma árvore binária e inseri-lo normalmente. No caso de uma página cheia, seguem os passos:
1. Criar uma nova página e alocar metade dos elementos da página cheia.
2. Transferir o elemento mediano para a página superior
obs. caso esse processo acometa uma página raiz cheia, uma nova raiz e um novo nó são criados, fazendo com que a árvore cresça para cima.
* Remoção: só é possível remover elementos de páginas folhas. Para remover itens que não estejam em páginas folhas, buscamos seu antecessor ou sucessor imediato (presente em alguma página folha) e o substituímos por ele.
obs. caso a remoção de um item viole a propriedade da árvore, é necessário reorganizar páginas irmãs e páginas superiores de modo a restaurar a propriedade da árvore. Quando a página irmã é incapaz de ceder elementos, fundem-se ambos em uma página só, undoing as páginas irmãs superiores quando necessário. Algumas formas de reorganização estão sujeitas à implementação preferida.
ex:



• Com o objetivo de tornar a árvore B mais eficiente, foram criadas propriedades adicionais. Uma implementação nesse contexto é, por exemplo, a árvore B^*.

→ Árvore B^*: 1. Todos os níveis, com exceção do último, constituem um índice organizado conforme uma árvore B.
2. Os registros são armazenados apenas no último nível, ou seja, nas páginas folhas.

obs: as chaves nos índices não necessariamente correspondem a itens úteis, servindo apenas para guiar o caminhamento na árvore.

obs: é possível adicionar apontadores sequenciais nos registros para fins, por exemplo, de impressão.

→ Estrutura:

```
typedef enum {Interna, Externa} TipoIntExt;
```

↳ enumeração: valores integrais que têm nomes como variáveis mas são apenas objetos de leitura.

```
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 estrutura toma forma de U0 ou U1 conforme o valor de pt:

* páginas internas e externas admitem ordens distintas.

tilibra

→ Pesquisa: semelhante a uma árvore B, mas só termina no encontro de uma página folha.

→ Inserção: essencialmente igual à árvore B, exceto quando uma página folha é dividida em duas. Nesse processo, uma cópia da chave mediana é promovida à página pai e o próprio item fica retido na página folha à direita.

→ Remoção: essencialmente igual à árvore B, mas em casos onde a propriedade da árvore é violada em alguma página, o registro é transferido diretamente entre as páginas fundidas sem passar previamente pelo pai.