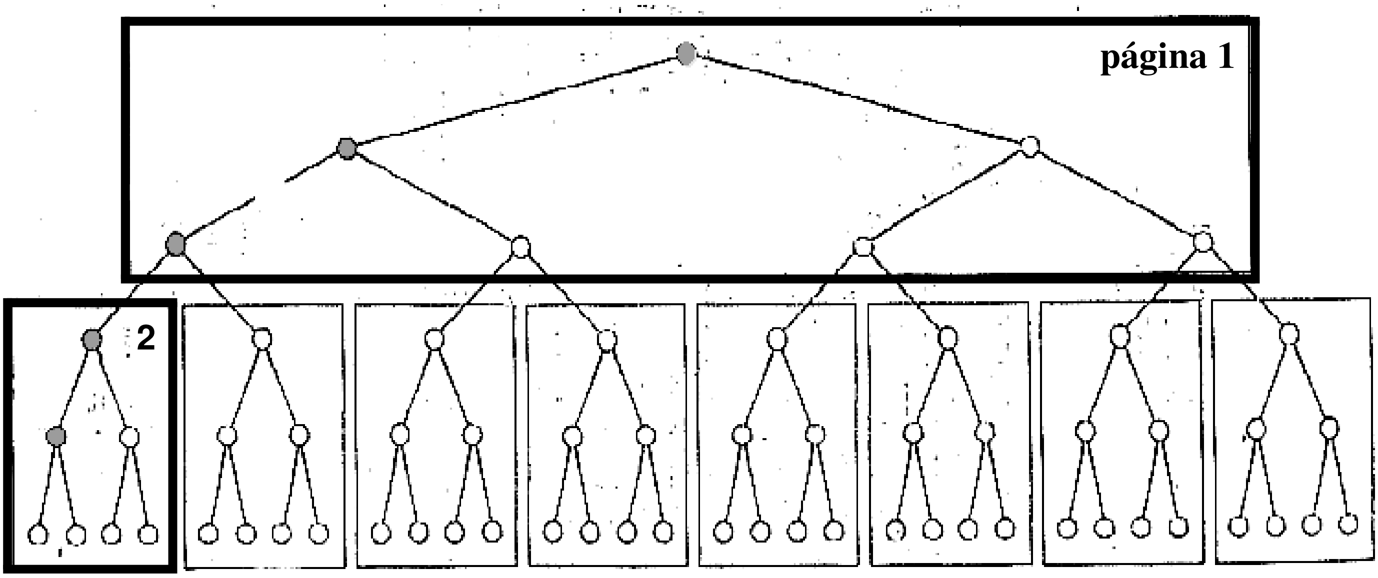
Resumo Geral

Árvores são indicadas para aplicações onde uma pesquisa rápida é muito importante, poucas alterações ocorrem em um período de tempo e a atualização da árvore (rebalanceamento) pode ser feita em batch m períodos em que a estrutura não é utilizada.

10.3.2. Árvore Binária Paginada

- Subárvores são concatenadas em um único registro.
- Adequado para árvores com nodos pequenos:
  - Árvores onde nodo possui um ponteiro p/arquivo de registros
  - Árvores com registros de dados realmente pequenos.
- A nível de arquivo de índices, um registro do arquivo possui vários nodos da árvore.



Vantagem

- Muito menos acessos a a disco são realizados.

Desvantagens

- Todos os algoritmos sofrem alterações.
- Rebalanceamento fica muito mais complexo.

10.3.5. Árvore K-D

- Árvore binária para indexação multichave. Também chamada **Árvore de Pesquisa Binária Multidimensional**.
- "Redescoberta" pela Inteligência Artificial como mecanismo de indexação de Cases de Casos em Raciocínio Baseado em Casos (CBR - Case-Based Reasoning).

Estrutura

- Uma árvore *k-d* é uma árvore onde *k* é o número de chaves para cada registro. Assim, chamamos uma árvore com três chaves de *árvore 3-d* (tridimensional), etc.
- Cada registro em uma árvore *k-d* possui, além dos dados e ponteiros, um conjunto ordenado de *k* valores de chaves ($v_0,...,v_{k-1}$).
- Associado a cada nodo *P* está um discriminador *DISC(P)*, que é utilizado para especificar qual chave da k-tupla $v_0,...,v_{k-1}$ será utilizada neste nodo para tomar uma decisão de ramificação. Geralmente o discriminador é função da profundidade do nodo (resto da divisão da profundidade pelo número de chaves).
- O filho à esquerda é chamado *loson(P)* e o à direita *hison(P)*.
- Caminhamento e Inserção (nas folhas) são realizados seguindo-se a seguinte regra de decisão (dados *Q* = chaves do registro procurado ou registro a ser incluído e P = nodo):

```
SE Kj(Q) < Kj(P) ENTÃO o registro Q está em loson(P)
SE Kj(Q) < Kj(P) ENTÃO o registro Q está em loson(P)
```

- Se dois valores de chave são iguais, a decisão é tomada com base nos demais valores de chave na seguinte ordem (super-chave):

$$S_j(P) = K_j(P), K_{j+1}(P), ..., K_{k-1}(P), K_0(P), ..., K_{j-1}(P)$$

10.4. Técnicas de Gerência de Arquivos Utilizando Listas

- Técnicas de indexação através de listas provêem uma solução excelente para a recuperação de chaves secundárias.
- Exemplos típicos de chave secundária: **`CARGO, DEPARTAMENTO, IDADE`**, etc.
- **Lista**: estrutura de acesso inerentemente seqüencial.
  Não é adequada: para conjuntos de dados com somente um ou muito poucos registros por valor de chave.
  Situação ideal: Poucos valores possíveis para uma chave e muitos registros possuindo este valor de chave.
- Suportam os três tipos de solicitações de recuperação.
- Se a aplicação possui uma chave tipicamente primária (Ex.: **`nome, cpf, id`**), a melhor técnica é utilizar uma técnica de indexação primária (árvore-B p.ex.) para a organização geral do arquivo e utilizar uma das duas técnicas de listas encadeadas abaixo para a indexação somente de chaves secundárias.

10.4.1. Atualização de Arquivos com Índices em Lista

Independentemente do tipo de indexação por lista, dois aspectos são gerais:

1. Um registro pode ser membro de duas ou mais listas.
2. Marcar um registro como excluído é aqui **muito** mais prático do que atualizar diversas listas.

10.4.3. Lista Invertida

A organização de um arquivo **Lista Invertida** consiste de um diretório contendo uma ou mais listas-índice e um ou mais arquivos endereços de de registros de dados, além do arquivo de registrso de dados em si.

O diretório é constituído por dois grupos de arquivos:

1. Listas-índice para valores de chaves;
2. Arquivo com apontadores para registros organizado por valores de chave.

Uma entrada do índice de uma chave possui os seguintes dados:

1. O valor da chave;
2. Um indicador para a lista de registros contendo o valor de chave;
3. O número de registros da lista.

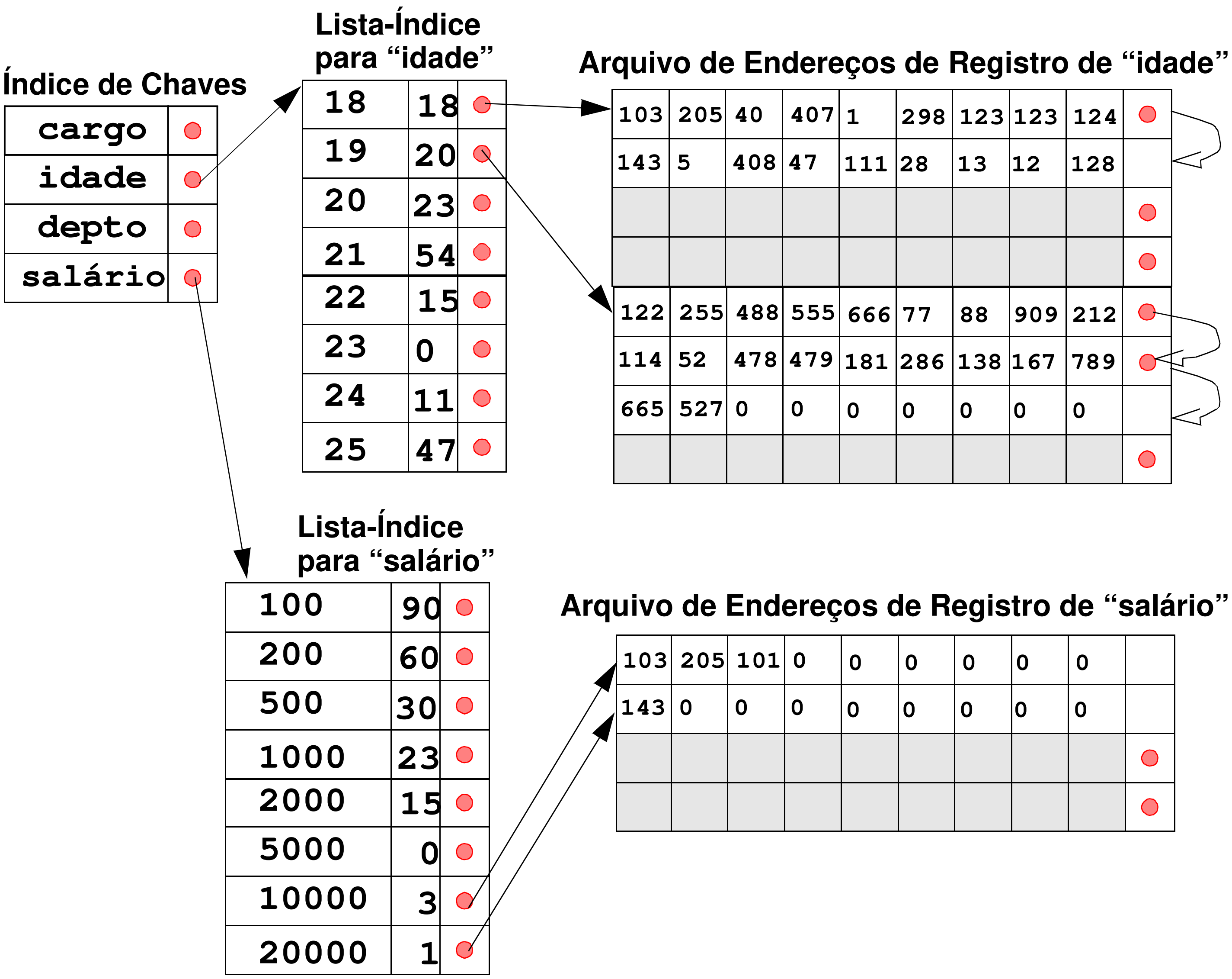
Uma entrada da lista de registros possui os seguintes dados:

1. Um conjunto de m campos contendo endereços de registros de dados;
2. Um apontador para a posição do próximo registro da lista de registros.

Isto está representado no exemplo adiante:

Exemplo (as listas-índice para **`cargo`** e **`depto`** estão omitidas, da mesma forma só foram representados endereços de registro para **`idade`** = 18 e 20 e **`salário`** = 10.000 e 20.000):

Exemplo parcial de um Diretório p/Lista Invertida



Índice de Chaves

cargo	●
idade	●
depto	●
salário	●

Lista-Índice para "idade"

18	18	●
19	20	●
20	23	●
21	54	●
22	15	●
23	0	●
24	11	●
25	47	●

Arquivo de Endereços de Registro de "idade"

103	205	40	407	1	298	123	123	124	●
143	5	408	47	111	28	13	12	128	
									●
									●
122	255	488	555	666	77	88	909	212	●
114	52	478	479	181	286	138	167	789	●
665	527	0	0	0	0	0	0	0	
									●

Lista-Índice para "salário"

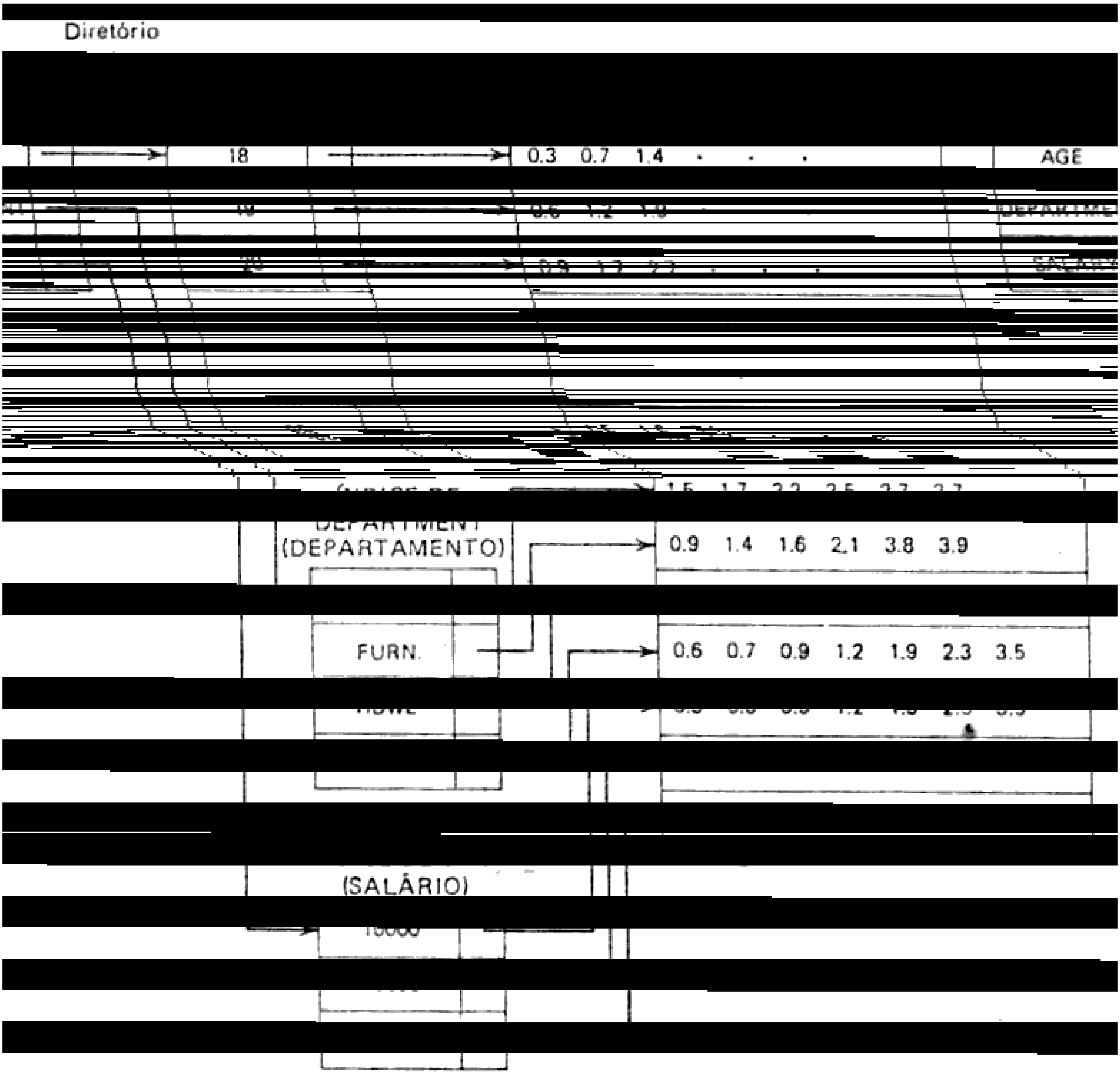
100	90	●
200	60	●
500	30	●
1000	23	●
2000	15	●
5000	0	●
10000	3	●
20000	1	●

Arquivo de Endereços de Registro de "salário"

103	205	101	0	0	0	0	0	0	
143	0	0	0	0	0	0	0	0	
									●
									●

- No exemplo acima endereços de registro dos arquivos presentes no gráfico estão representados por setas, endereços de registros do arquivo de registros de dados por valores de endereço relativo.
- Evidentemente podem existir vários arquivos de endereços de registro e várias listas-índice. Alternativamente podemos colocar os registros de endereços todos em um único arquivo.

Exemplo (Claybrook 77):



[redacted]

- Neste exemplo somente um arquivo para listas de endereços de registro é utilizado.
- Os endereços do exemplo (`1.5, 0.7, 2.7`) também se referem a uma forma de endereçamento relativo mais antiga, onde um arquivo é dividido em “células”.

Técnica de Pesquisa

- Pesquisas de chaves em arquivos **Lista Invertida** são realizadas também através da geração de tabelas.
- Ao contrário da multilista, aqui as tabelas podem ser bem mais simples e a pesquisa para a construção destas pode ser bem mais efetiva, sem realmente ler os registros de dados.
- Pesquisas de chave simples geram uma tabela de saída com todos os registros com o valor de chave para aquela chave, depois de ter sido gerada uma tabela de endereços de registros com o valor de chave.
- Pesquisas booleanas são as mais complexas. Nestas pesquisas, várias tabelas de endereços são geradas, iniciando-se sempre pelo valor de chave com o menor número de ocorrências.
  Em pesquisas conjuntivas, endereços não encontrados p/ outras chaves são retirados.
  Em pesquisas disjuntivas, são adicionados.
  Com a tabela de endereços resultante é criada a tabela de registros de dados definitiva. Assim somente os registros realmente necessários vão ser lidos do arquivo de registros de dados.

Vantagens

- Bom suporte para todos os tipos de pesquisas, desde que o campo pesquisado seja indexado.
- Uma das melhores técnicas para chaves secundárias com altas taxas de redundância de valor de chave.
- O fato de podermos realizar todas as pesquisas de chaves que possuam índices diretamente nas tabelas de endereços provê uma vantagem imensa sobre a multilista.

Desvantagens

- Em pesquisas booleanas onde pelo menos dois dos elementos procurados não estão indexados, temos de criar tabelas contendo cópias de todos os registros que satisfizeram a parte indexada da consulta.
- Se mais de um elemento não está indexado e a pesquisa é conjuntiva, isto pode gerar tabelas grandes e se tornar caro.
- Atualização é complexa, pois todos os arquivos de índices devem ser atualizados.