Estruturas de Dados 2

Prof. Silvia Brandão

2024.1

Pesquisa em Memória Interna

- De modo geral, é o estudo de como recuperar informação a partir de uma grande massa de informação previamente armazenada.
- A informação é dividida em registros.
- Cada registro possui uma chave para ser usada na pesquisa.
- **Objetivo da pesquisa**: Encontrar uma ou mais ocorrências de registros com chaves iguais à chave de pesquisa.
  - Pesquisa com sucesso X sem sucesso.

Métodos de Pesquisa em Memória

- Algumas técnicas de busca em memória interna são:
  - Busca Sequencial
  - Busca Binária
  - Busca por Interpolação
  - Busca em Árvores
  - Hashing

- O objetivo é encontrar um dado registro com o menor custo
- Cada técnica possui vantagens e desvantagens

Simulador:
https://www.cs.usfca.edu/~galles/visualization/Search.html

Pesquisa Sequencial

- Método de pesquisa mais simples: a partir do primeiro registro, pesquise sequencialmente até encontrar a chave procurada; então pare.
- Armazenamento de um conjunto de registros por meio de um array.
- A Busca (find):
  - Pesquisa retorna o índice do registro que contém a chave x;
  - Caso não esteja presente, o valor retornado é -1.
  - A implementação não suporta mais de um registro com uma mesma chave, pois retorna o primeiro encontrado.

Busca Sequencial – algoritmo

```
i = 0
encontrado = 0 //Falso
Enquanto (i < TAMANHO && !encontrado) {
    se (vetor[i] == valor) {
        encontrado = 1 //sucesso
    } senão {
        i++
    }
}
```

```
// Tratamento do Resultado
se (encontrado)
    escreva(“O valor procurado está na posição”, i)
senão
    escreva(“O valor não foi encontrado”)
```

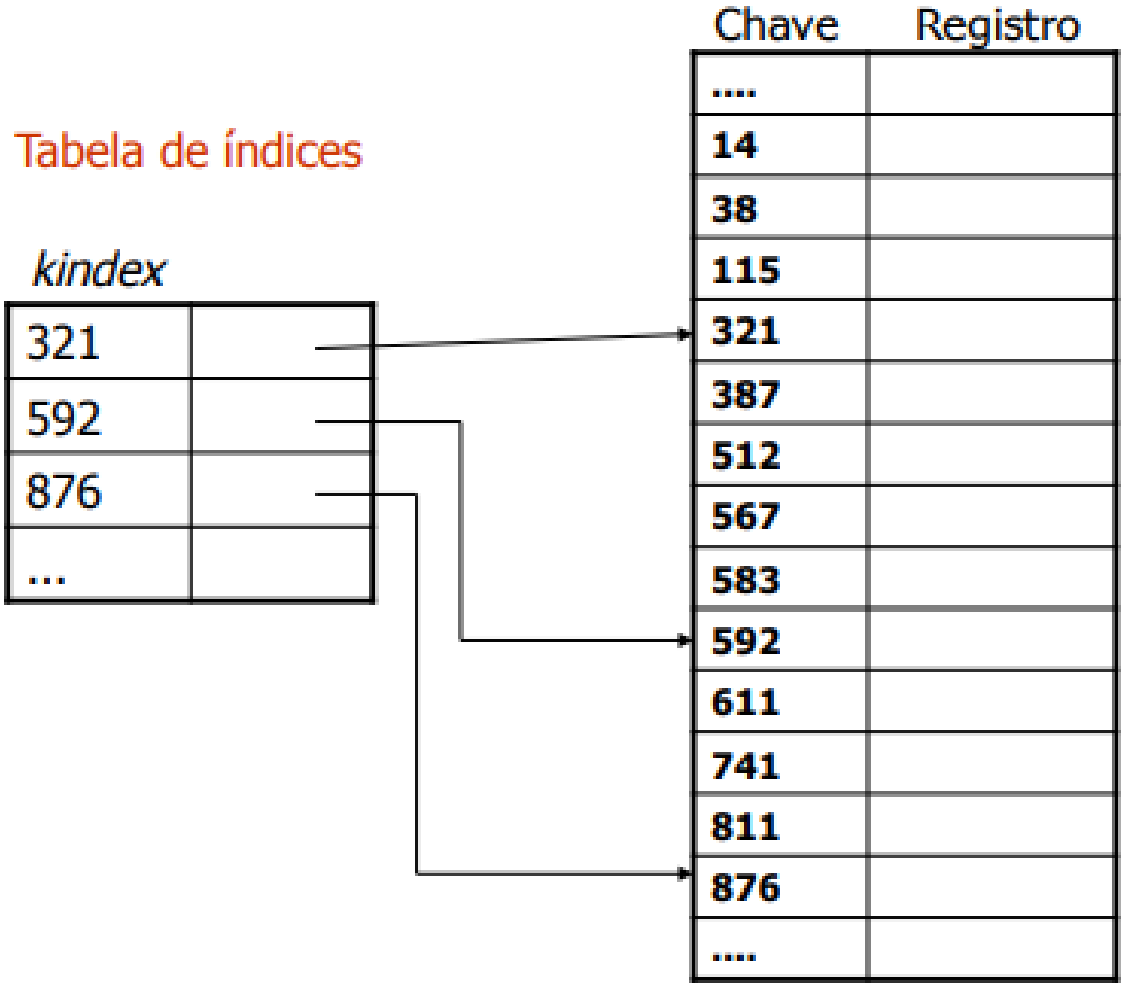
Busca Sequencial

- Análise:
  - Pesquisa **com sucesso**:
    - melhor caso : C(n) = 1
    - pior caso : C(n) = n
    - caso médio: C(n) = (n + 1) / 2
  - Pesquisa **sem sucesso**:
    - C (n) = n + 1.
  - O algoritmo de pesquisa sequencial é a melhor escolha para o problema de pesquisa com n < 25.

Busca Sequencial Indexada

- Existe uma tabela auxiliar, chamada **tabela de índices**, além do próprio arquivo ordenado.
- Cada elemento na tabela de índices contém uma chave (kindex) e um indicador do registro no arquivo que corresponde a kindex
  - Faz-se a busca a partir do ponto indicado na tabela, sendo que a busca não precisa ser feita desde o começo
- Pode ser implementada como um vetor ou como uma lista encadeada
  - O indicador da posição na tabela pode ser um ponteiro ou uma variável inteira

Busca Sequencial Indexada

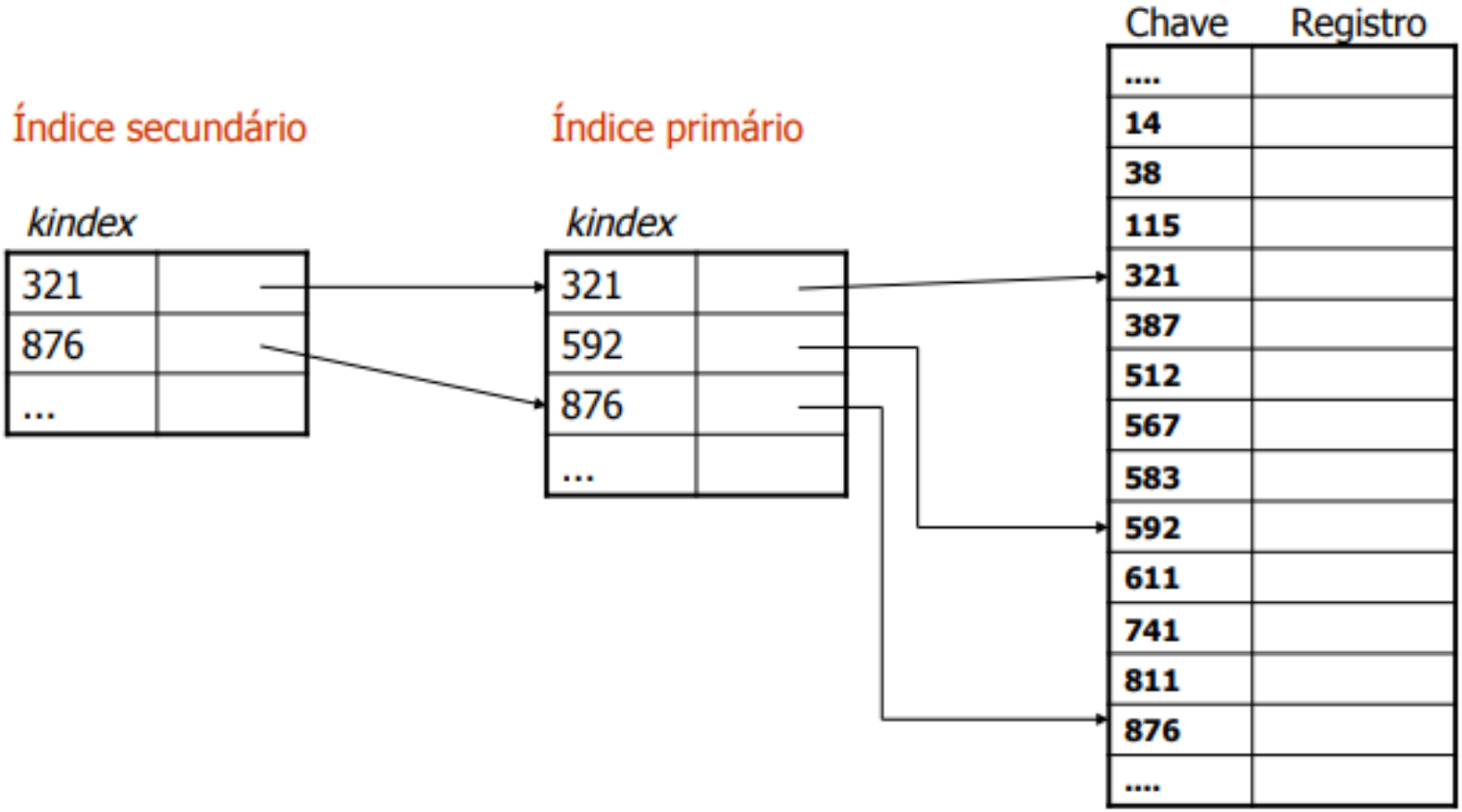


E, se a tabela for muito grande?

Busca Sequencial Indexada

Se a tabela for muito grande, pode-se ainda usar a tabela de índices secundária.

- O índice secundário é um índice para o índice primário.



Busca Sequencial Indexada

- **Vantagem**
  - Os itens na tabela poderão ser examinados sequencialmente
  - sem que todos os registros precisem ser acessados
  - O tempo de busca diminui consideravelmente

- **Desvantagens**
  - A tabela tem que estar ordenada
  - Exige espaço adicional para armazenar a(s) tabela(s) de índices

Busca Binária

- Pesquisa em tabela pode ser mais eficiente se registros forem mantidos em ordem
- Para saber se uma chave está presente na tabela
  1. Compare a chave com o registro que está na posição do meio da tabela.
  2. Se a chave é menor então o registro procurado está na primeira metade da tabela.
  3. Se a chave é maior então o registro procurado está na segunda metade da tabela.
  4. Repita até que a chave seja encontrada ou que se constate que a chave não existe na tabela.

Busca Binária – algoritmo

```
int direita, esquerda, meio
encontrado = 0 //Falso
esquerda = 0
direita = TAMANHO - 1
//Busca
Enquanto (esquerda<=direita && !encontrado){
    meio=(direita+esquerda)/2
    se (vetor[meio] == valor)
       encontrado = 1 //Verdadeiro
    senão
        se (valor < vetor[meio])
            direita = meio - 1
        senão
            esquerda = meio + 1
}
```

```
// Tratamento do Resultado
se (encontrado)
    escreva("O valor procurado
está na posição", i)
senão
    escreva("O valor não foi
encontrado")
```

Busca Binária

- Análise
  - A cada iteração do algoritmo, o tamanho da tabela é dividido ao meio.
  - **Logo**: o número de vezes que o tamanho da tabela é dividido ao meio é cerca de **log n**.
  - **Ressalva**: o custo para manter a tabela ordenada é alto: a cada inserção na posição p da tabela implica no deslocamento dos registros a partir da posição p para as posições seguintes.
  - Consequentemente, a pesquisa binária não deve ser usada em aplicações muito dinâmicas.
  - A busca binária pode ser usada com a organização de tabela sequencial indexada.

Exercício

- Escrever uma sub-rotina de busca binária por um elemento em um arranjo ordenado
  - Versão recursiva
  - Versão não recursiva

Busca por Interpolação

- Se as chaves estiverem uniformemente distribuídas, esse método pode ser ainda mais eficiente do que a busca binária
- Com chaves uniformemente distribuídas, pode-se esperar que x esteja aproximadamente na posição

meio = inf + (sup – inf) * ((x – A[inf]) / (A[sup] – A[inf]))

sendo que **inf** e **sup** são redefinidos iterativamente como na busca binária.

- **Complexidade**: *O(log(log(n)))* se as chaves estiverem uniformemente distribuídas. Raramente precisará de mais comparações.
  - Se as chaves não estiverem uniformemente distribuídas, a busca por interpolação pode ser tão ruim quanto uma busca sequencial
- **Desvantagem**: Em situações práticas, as chaves tendem a se aglomerar em torno de determinados valores e não são uniformemente distribuídas. Por exemplo: há uma quantidade maior de nomes começando com “S” do que com “Q”

Referência

- Ziviani, Nivio. **Projeto de Algoritmos com implementações em JAVA e C++.**

- Search. https://www.cs.usfca.edu/~galles/visualization/Search.html