FCT/Unesp – Presidente Prudente
Departamento de Matemática e Computação

Análise de Algoritmos de Busca

Prof. Danilo Medeiros Eler
danilo.eler@unesp.br

Apresentação adaptada (ver referências)

Departamento de Matemática e Computação
Faculdade de Ciências e Tecnologia
Unesp – Universidade Estadual Paulista
Presidente Prudente/SP, Brasil

Introdução

- Busca é uma tarefa muito comum em computação
- Certos métodos de organização/ordenação de dados podem tornar o processo de busca mais eficiente
- Vários métodos e estruturas de dados podem ser empregados para se fazer uma busca

Introdução

- O problema da busca (ou pesquisa)

  "*Dado um conjunto de elementos, onde cada um é identificado por uma chave, o objetivo da busca é localizar, nesse conjunto, o elemento que corresponde a uma chave específica*"

Termos Relacionados

- Tabela: termo genérico, pode ser qualquer estrutura de dados usada para armazenamento interno e organização dos dados

- Uma tabela é um conjunto de elementos, chamados registros

Termos Relacionados

- A tabela pode ser:
  - Um vetor de registros
  - Uma lista encadeada
  - Uma árvore
  - Etc.
- A tabela pode ficar:
  - Totalmente na memória (busca interna)
  - Totalmente no armazenamento auxiliar (busca externa)
  - Dividida entre ambos

Termos Relacionados

- Algoritmo de busca
  - Formalmente, é o algoritmo que aceita um argumento **a** e tenta encontrar o registro cuja chave seja **a**

Tipos de Busca

- Alguns dos tipos de busca são
  - Busca Sequencial
  - Busca Binária
  - Busca em Árvores
  - Hashing
- O objetivo é encontrar um dado registro com o **menor custo**
  - Cada técnica possui vantagens e desvantagens

Busca Sequencial

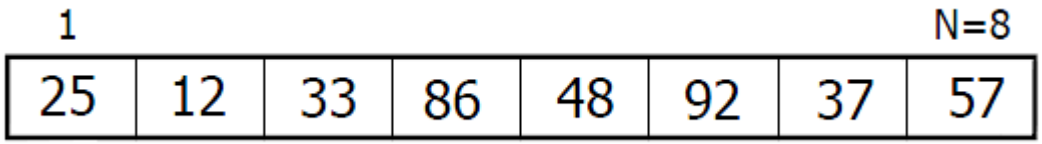
Busca Sequencial

- A busca seqüencial é a forma mais simples de busca
  - É aplicável a uma tabela organizada como um vetor ou como uma lista encadeada

Busca Sequencial

- Busca mais simples que há
  - Percorre-se registro por registro em busca da chave

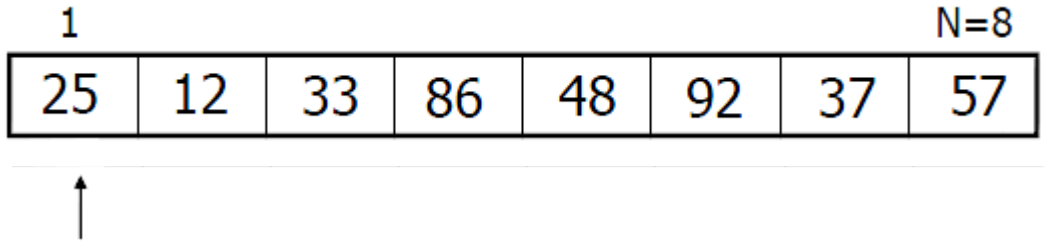
Procure por 48



Busca Sequencial

- Busca mais simples que há
  - Percorre-se registro por registro em busca da chave

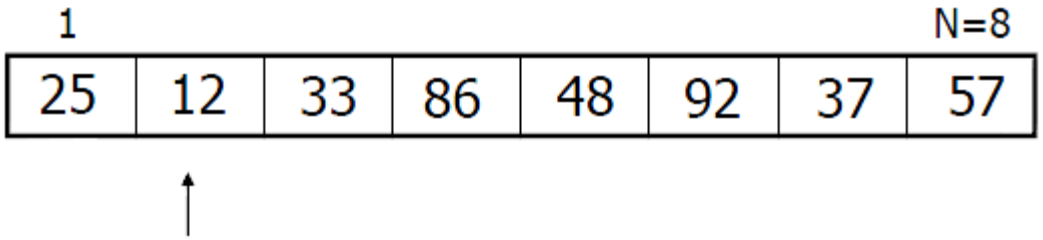
Procure por 48



Busca Sequencial

- Busca mais simples que há
  - Percorre-se registro por registro em busca da chave

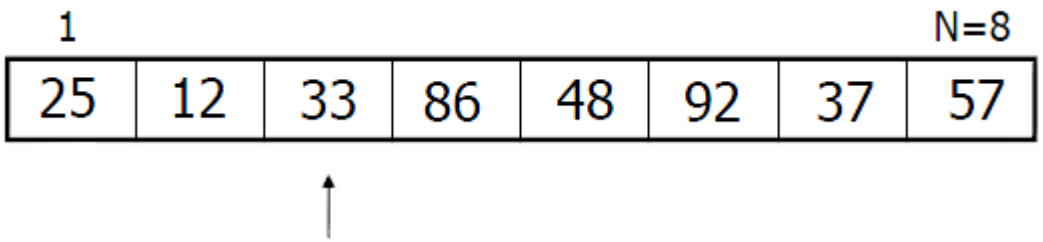
Procure por 48



Busca Sequencial

- Busca mais simples que há
  - Percorre-se registro por registro em busca da chave

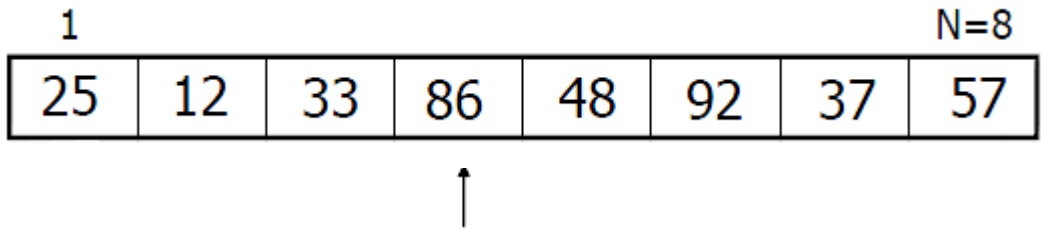
Procure por 48



Busca Sequencial

- Busca mais simples que há
  - Percorre-se registro por registro em busca da chave

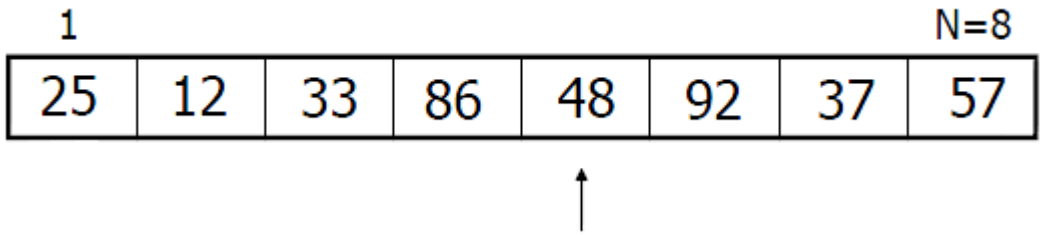
Procure por 48



Busca Sequencial

- Busca mais simples que há
  - Percorre-se registro por registro em busca da chave

Procure por 48



Busca Sequencial

- Algoritmo de busca seqüencial em um vetor A, com N posições (0 até N-1), sendo x a chave procurada

```
for (i=0; i<n; i++)
   if (A[i]==x)
      return(i);      /*chave encontrada*/
return(-1);           /*chave não encontrada*/
```

Busca Sequencial

- Uma maneira de tornar o algoritmo mais eficiente é usar um sentinela
  - Sentinela: consiste em adicionar um elemento de valor x no final da tabela
  - O sentinela garante que o elemento procurado será encontrado, o que elimina uma expressão condicional, melhorando a performance do algoritmo

```
A[N]=x;
for(i=0; x!=A[i]; i++)
    ;
if (i<n) return(i); /*chave encontrada*/
else return(-1);    /*sentinela encontrado*/
```

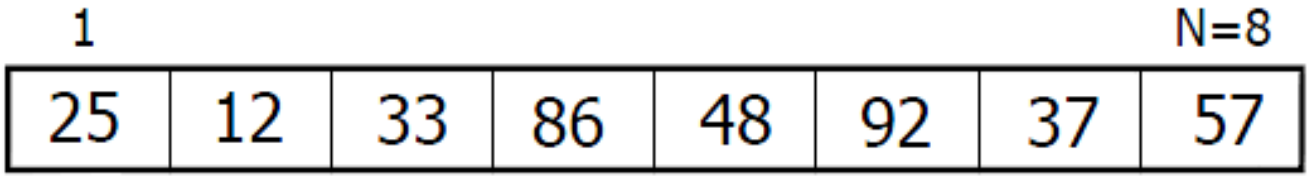
Outra Representação

- Limitações do vetor
  - Tamanho fixo

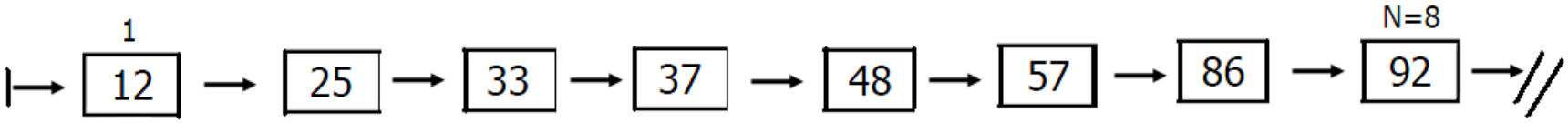
- Alternativa
  - Lista encadeada
    - O que muda na busca seqüencial?

Representação

- Vetor

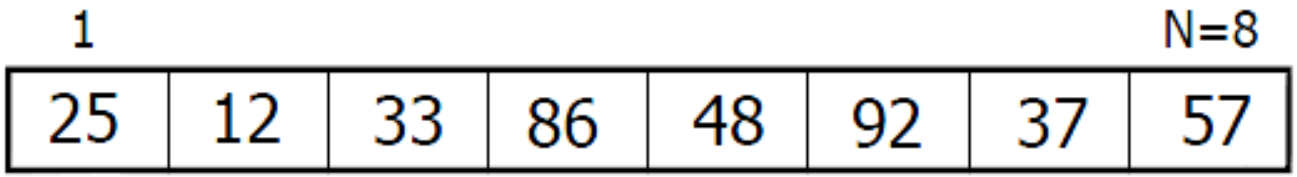


- Lista Encadeada



Busca Sequencial

- Complexidade
  - Se o registro for o primeiro: 1 comparação
  - Se o registro procurado for o último: N comparações
  - Se a busca for mal sucedida: N comparações
    - Logo, a busca sequência, no pior caso é O(n)



Busca Sequencial

- Complexidade
  - Se for igualmente provável que a chave de busca apareça em qualquer posição da tabela?

1							N=8
25	12	33	86	48	92	37	57

Busca Sequencial

- Complexidade
  - Se for igualmente provável que a chave de busca apareça em qualquer posição da tabela?

1							N=8
25	12	33	86	48	92	37	57

$$\frac{1}{n}+\frac{2}{n}+\frac{3}{n}+\ldots+\frac{n-1}{n}+\frac{n}{n}$$

Busca Sequencial

- Complexidade de Caso Médio
  - Se for igualmente provável que a chave de busca apareça em qualquer posição da tabela?

1							N=8
25	12	33	86	48	92	37	57

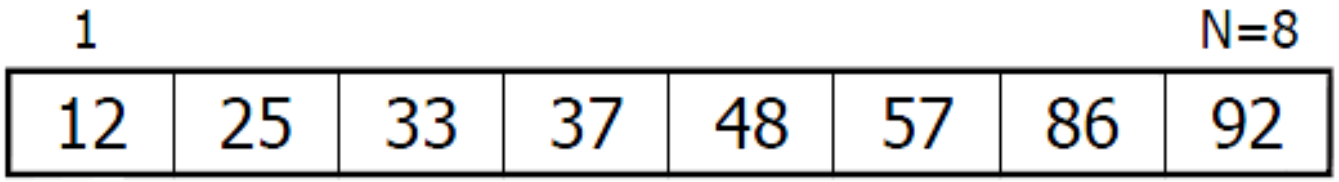
$$\frac{1}{n}+\frac{2}{n}+\frac{3}{n}+\ldots+\frac{n-1}{n}+\frac{n}{n}$$

$$\frac{1}{n}(1+2+3+\ldots+n-1+n)$$

$$\frac{1}{n}\left(\frac{n(n+1)}{2}\right)=\frac{n(n+1)}{2n}=\boxed{\frac{(n+1)}{2}}$$

Busca Sequencial

- Se os dados estiverem ordenados, há diferença na complexidade de uma busca sequencial?



Busca Binária

Busca Binária

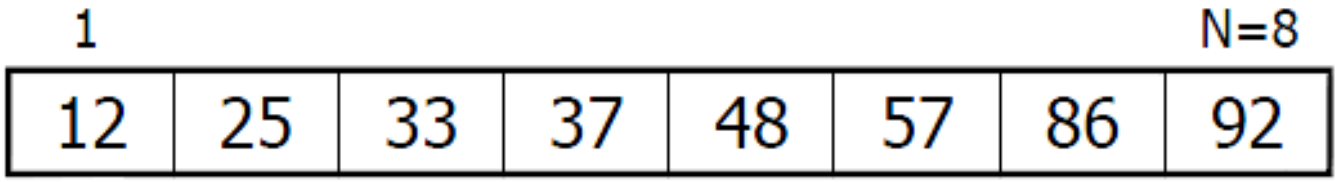
- Se os dados estiverem ordenados em um arranjo, pode-se tirar vantagens dessa ordenação
  - Busca binária

A[i]<=A[i+1], se ordem crescente

A[i]>=A[i+1], se ordem decrescente

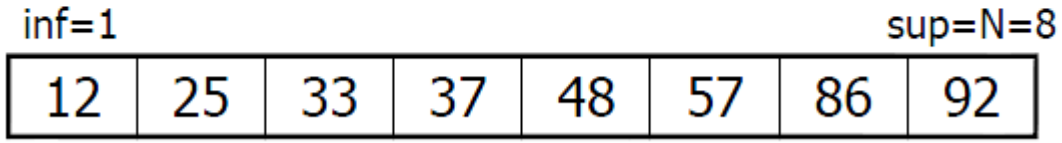
Busca Binária

- O elemento buscado é comparado ao elemento do meio do arranjo
  - Se igual, busca bem-sucedida
  - Se menor, busca-se na metade inferior do arranjo
  - Se maior, busca-se na metade superior do arranjo



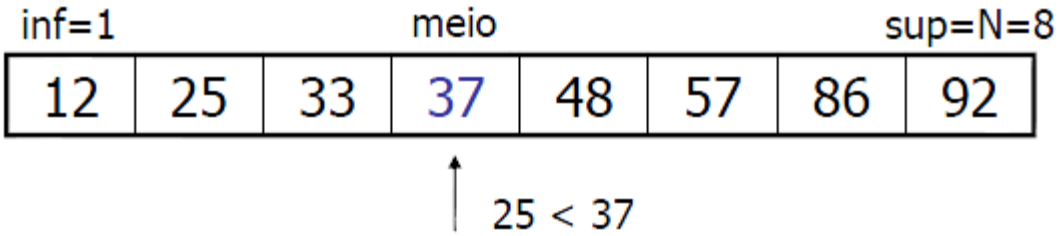
Busca Binária

- Busca-se por 25



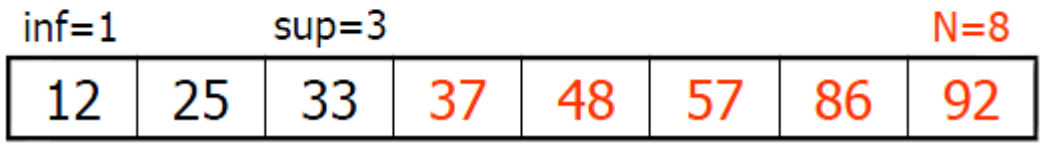
Busca Binária

- Busca-se por 25



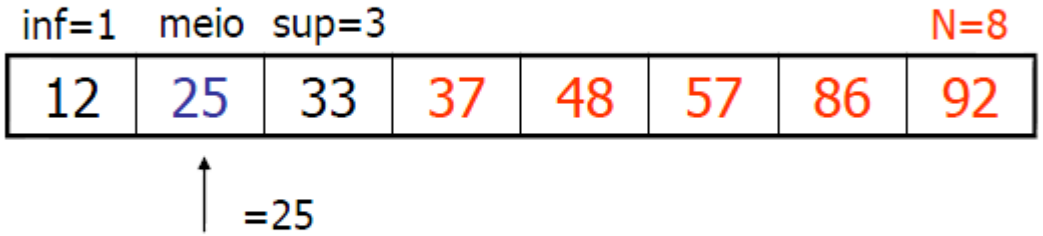
Busca Binária

- Busca-se por 25



Busca Binária

- Busca-se por 25



Em cada passo, o tamanho do arranjo em que se busca é dividido por 2

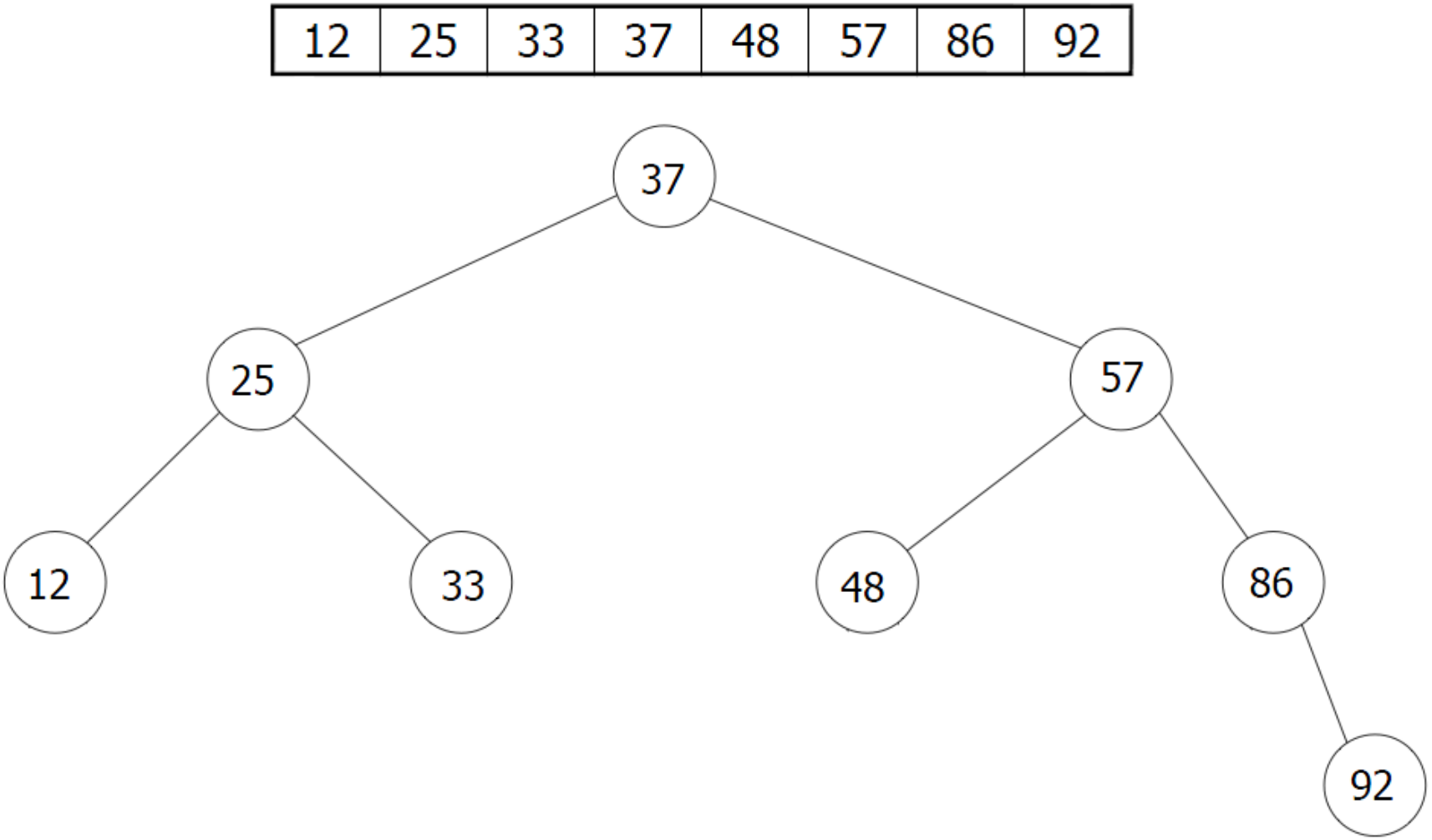
Busca Binária

```
BUSCA-BINÁRIA (V[], início, fim, e)
   i = meio(início, fim);
   se (v[i] == e)
       retorna i;
   fimse
   se (inicio == fim)
        retorna -1;
   senão se (V[i] < e)
                  BUSCA-BINÁRIA(V, i+1, fim, e);
             senão
                  BUSCA-BINÁRIA(V, inicio, i-1, e);
             fimse
    fimse
fim
```

Busca Binária

- Vantagens
  - Eficiência da busca
  - Simplicidade da implementação
- Desvantagens
  - Nem todo arranjo está ordenado
  - Exige o uso de um arranjo para armazenar os dados
    - Faz uso do fato de que os índices do vetor são inteiros consecutivos
  - Inserção e remoção de elementos são ineficientes
    - Realocação de elementos

Busca Binária



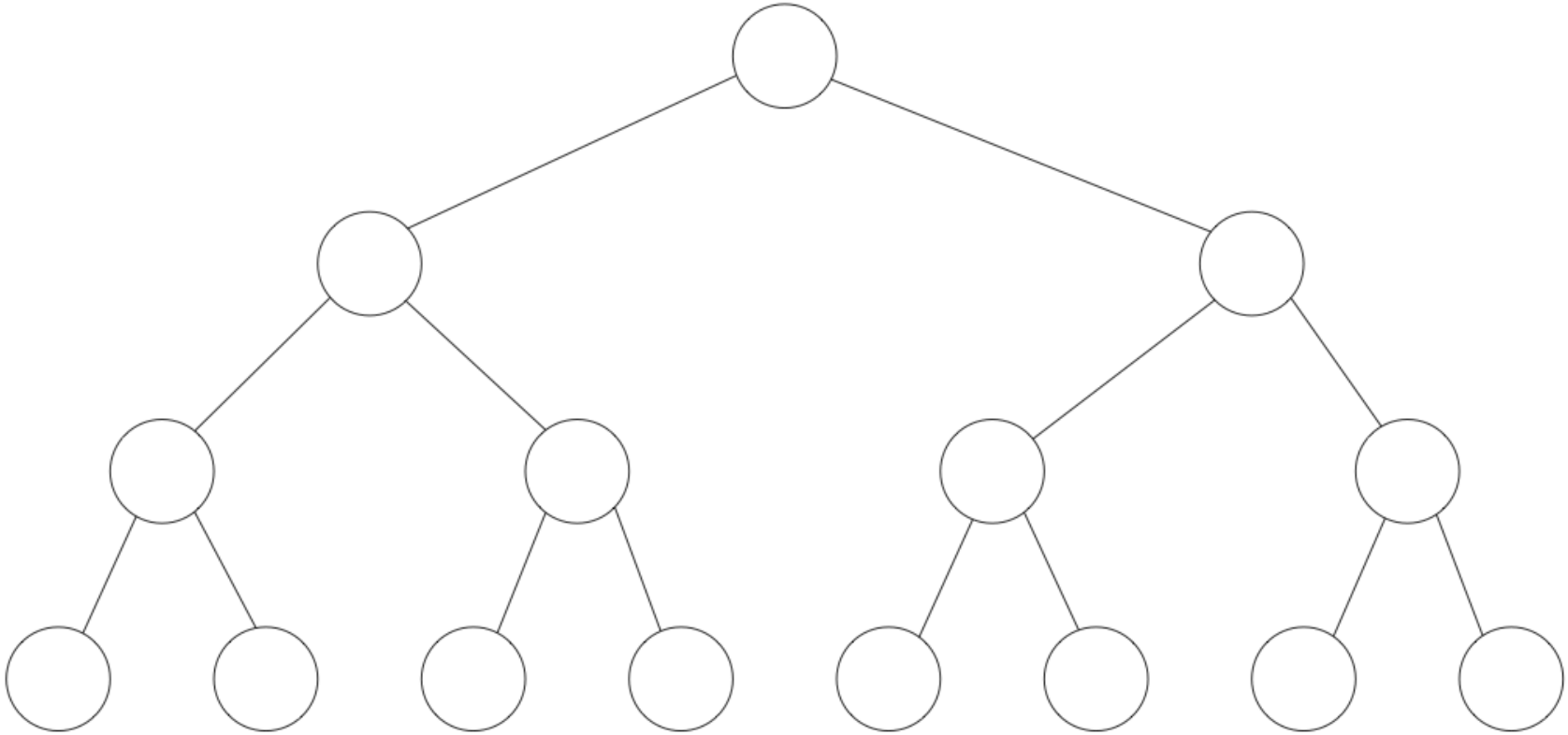
Busca Binária

- Melhor Caso
  - Ocorre quando o elemento buscado é o que está na posição central do vetor
  - A busca ocorre em tempo constante, sendo necessária uma única comparação
- Pior Caso
  - Ocorre quando a busca é executada até a última divisão do vetor (inicio == fim). Esse é o caso em que o elemento buscado está nessa última ou quando ele não está no vetor
  - O custo computacional é O(log n)
    - Altura de uma árvore binária balanceada

Busca em Árvores

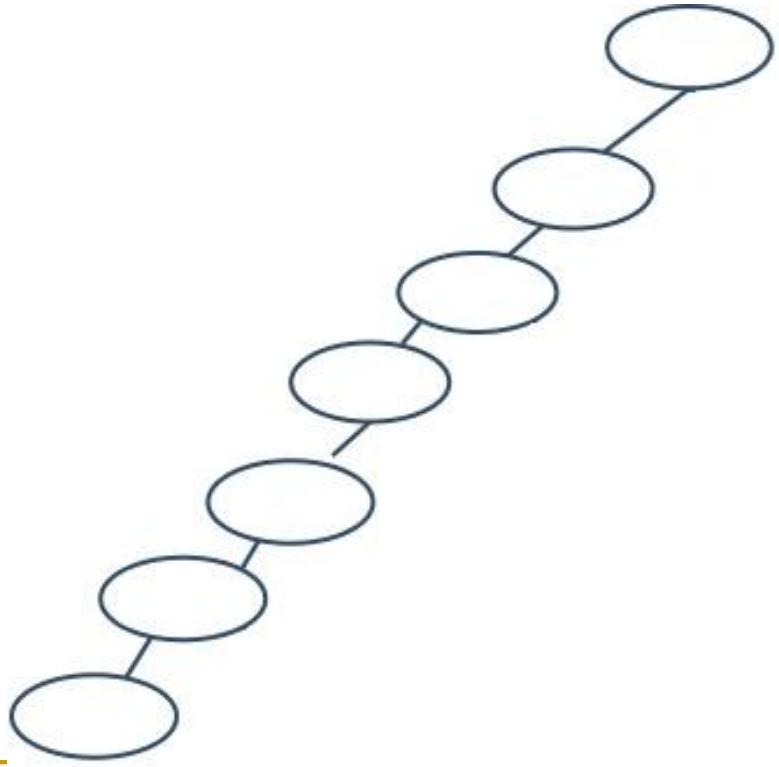
Busca em Árvores

- Na busca em árvores, o melhor caso ocorre quando a árvore está balanceada, tendo o custo da busca binária (log n)



Busca em Árvores

- O pior caso ocorre quando os elementos são inseridos em ordem e a árvore não é balanceada. Nesse caso, a busca ocorre em O(n)



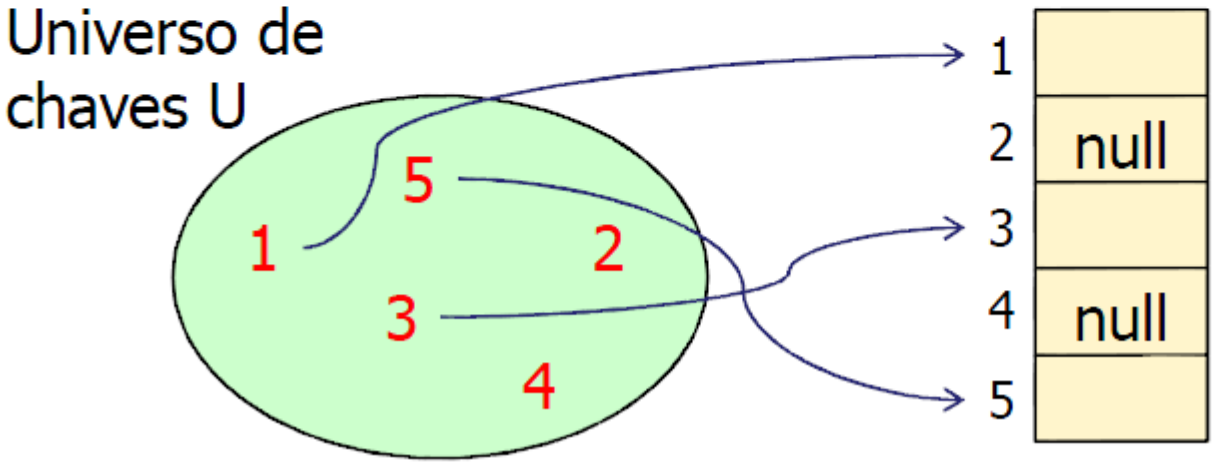
Hashing

Hashing

- Técnica que busca realizar as operações de inserção, remoção e busca em tempo constante
- Essa característica é muito importante quando se trabalha com **armazenamento secundário em disco**
  - Acesso a um determinado endereço é bastante lento

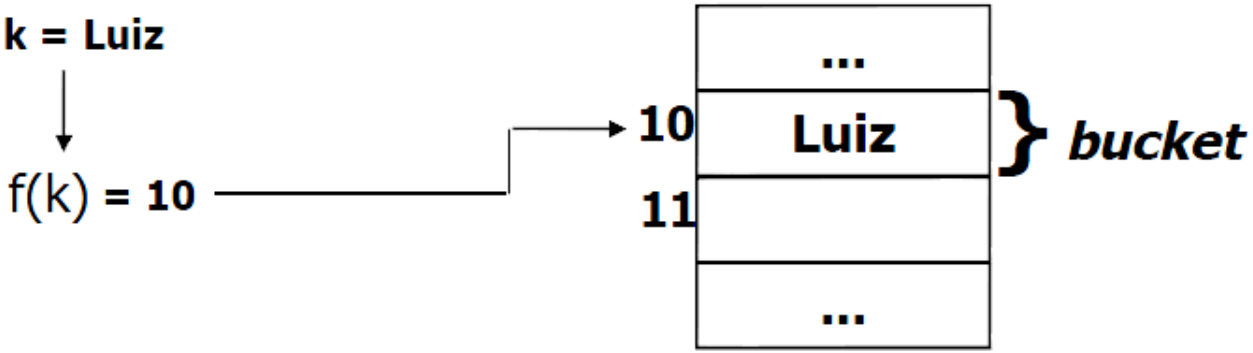
Hashing

- Acesso em tempo constante
  - Tradicionalmente, endereçamento direto em um arranjo
    - Cada chave k é mapeada na posição p do arranjo
      - Função de mapeamento f(k)=p



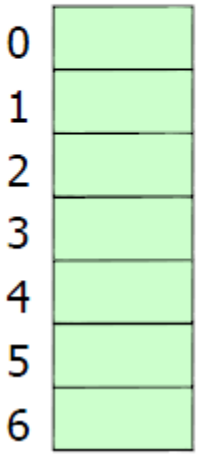
Hashing

- Conceitos relacionados
  - A função **f** é chamada de função *hash*
  - **f**(k) retorna o valor *hash* de k
    - Usado como endereço para armazenar a informação cuja chave é k
  - k pertence ao *bucket* **f**(k)



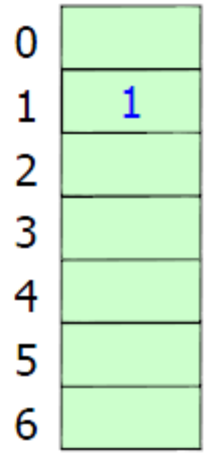
Hashing

- Exemplo
  - Seja B um arranjo de 7 elementos
    - Inserção dos números 1, 5, 10, 20, 25, 24



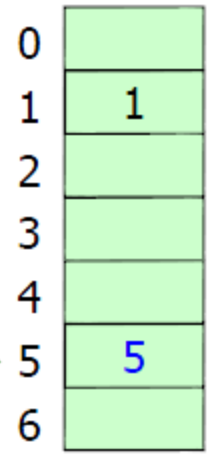
Hashing

- Exemplo
  - Seja B um arranjo de 7 elementos
    - Inserção dos números 1, 5, 10, 20, 25, 24
      - 1 % 7 = 1



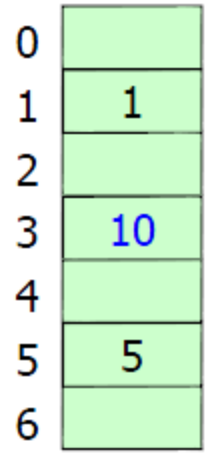
Hashing

- Exemplo
  - Seja B um arranjo de 7 elementos
    - Inserção dos números 1, 5, 10, 20, 25, 24
      - 5 % 7 = 5



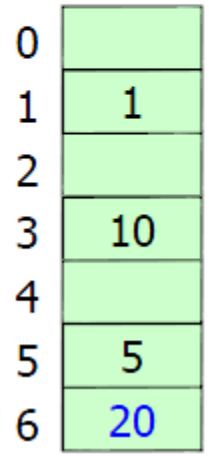
Hashing

- Exemplo
  - Seja B um arranjo de 7 elementos
    - Inserção dos números 1, 5, 10, 20, 25, 24
      - 10 % 7 = 3



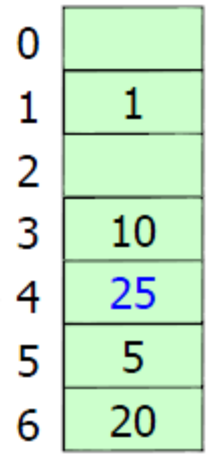
Hashing

- Exemplo
  - Seja B um arranjo de 7 elementos
    - Inserção dos números 1, 5, 10, 20, 25, 24
      - 20 % 7 = 6



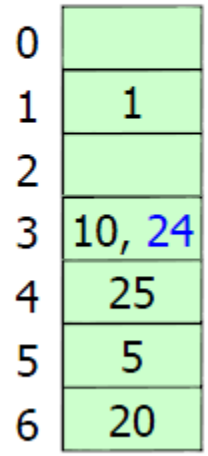
Hashing

- Exemplo
  - Seja B um arranjo de 7 elementos
    - Inserção dos números 1, 5, 10, 20, 25, 24
      - 25 % 7 = 4



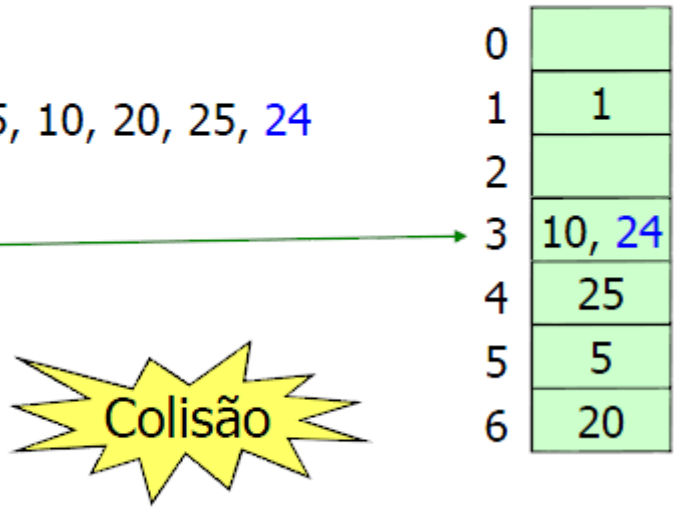
Hashing

- Exemplo
  - Seja B um arranjo de 7 elementos
    - Inserção dos números 1, 5, 10, 20, 25, 24
      - 24 % 7 = 3



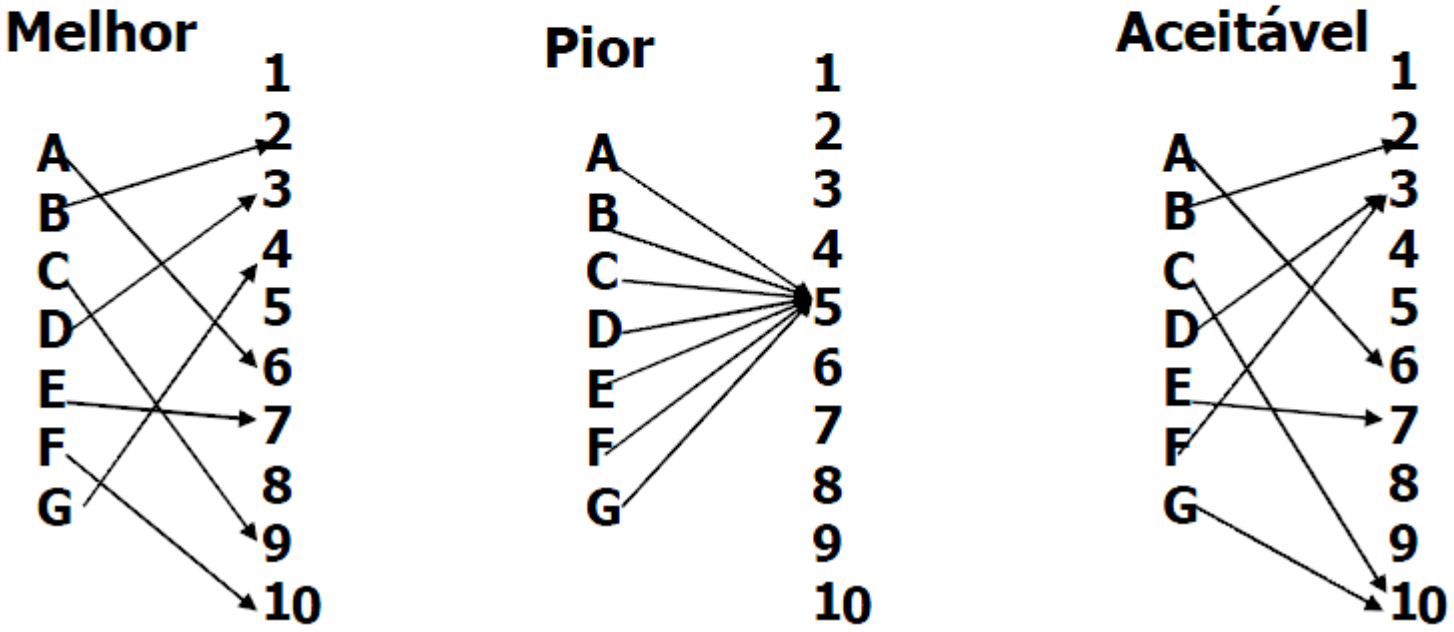
Hashing

- Exemplo
  - Seja B um arranjo de 7 elementos
    - Inserção dos números 1, 5, 10, 20, 25, 24
      - 24 % 7 = 3



Hashing

- Colisão: ocorre quando a função *hash* produz o mesmo endereço para chaves diferentes
  - As chaves com mesmo endereço são ditas "sinônimos"



Hashing

- Segredos para um bom *hashing*
  - Escolher uma boa função hash (em função dos dados)
    - Distribui uniformemente os dados, na medida do possível
      - Hash uniforme
    - Evita colisões
    - É fácil/rápida de computar
  - Estabelecer uma boa estratégia para tratamento de colisões

Hashing

- Vantagens
  - Acesso direto e, portanto, rápido
    - Via indexação do arranjo
- Desvantagens
  - Uso ineficiente do espaço de armazenamento

Hashing

- Há muitos tipos de funções hash e muitas maneiras de tratar as colisões
- Qual a complexidade de tempo de busca de uma função hash?

Hashing

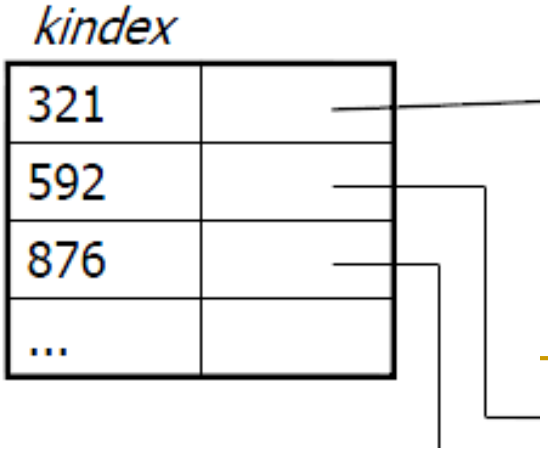
- Há muitos tipos de funções hash e muitas maneiras de tratar as colisões
- Qual a complexidade de tempo de busca de uma função hash?
  - Melhor Caso $\Omega(1)$
  - Pior Caso $O(n)$

Busca Indexada

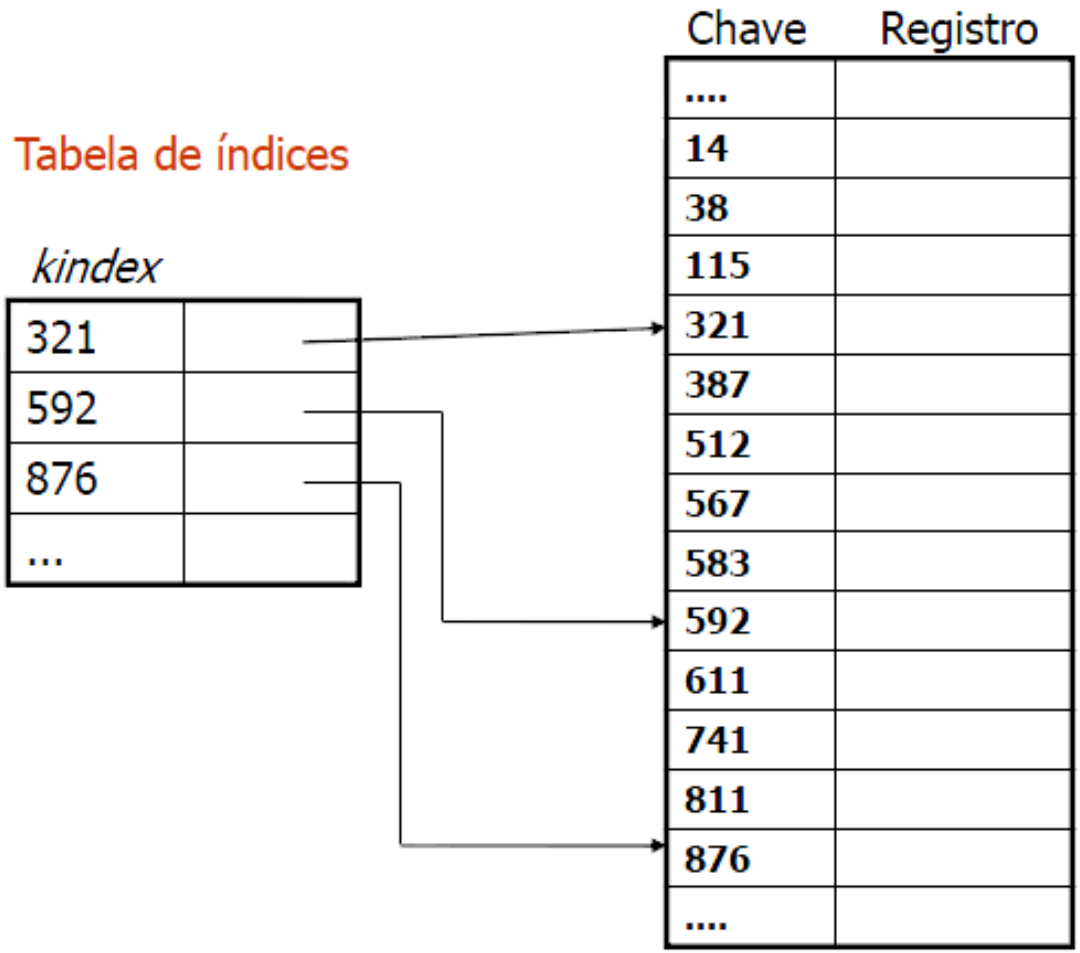
Busca Indexada

- 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

Tabela de índices



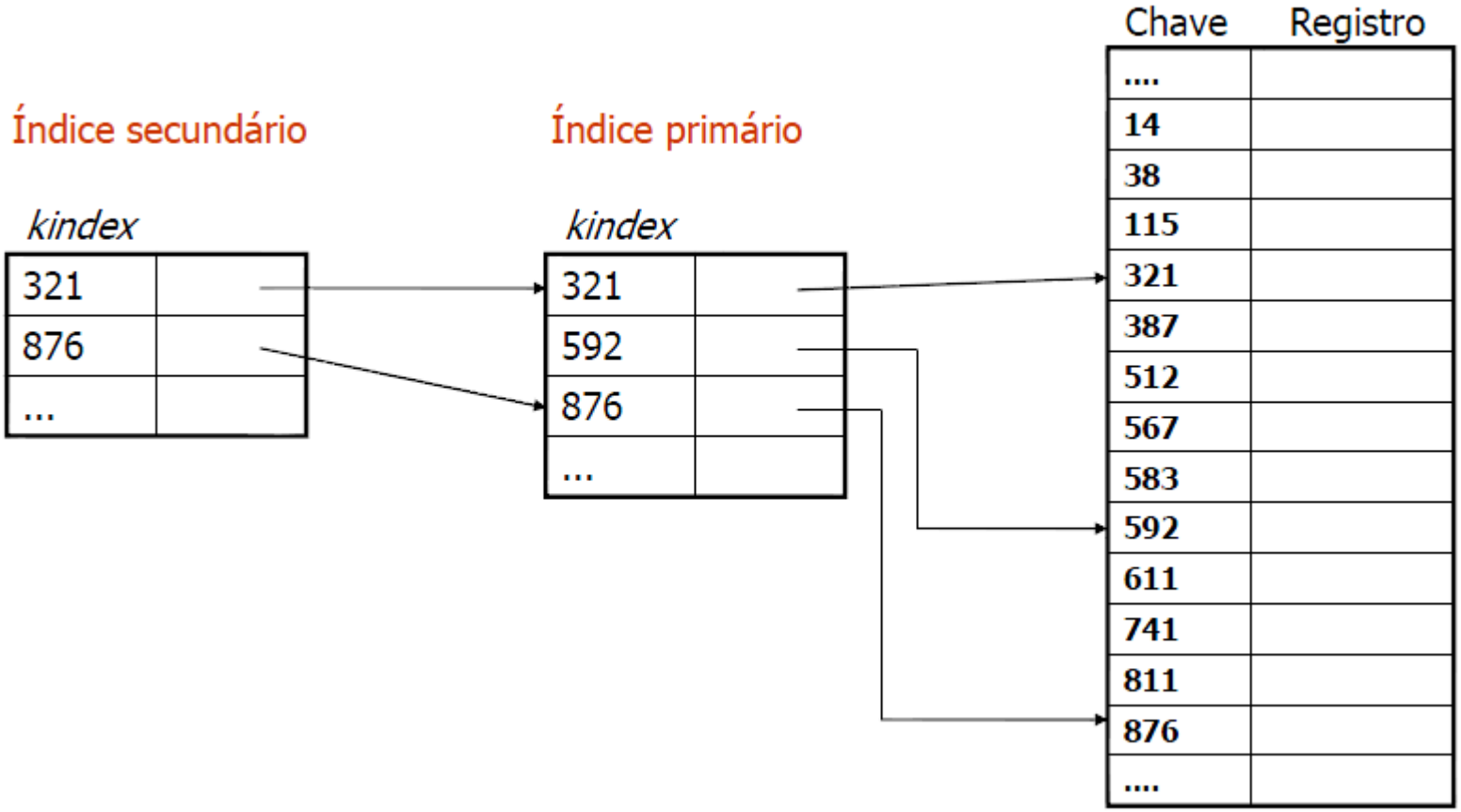
Busca Indexada



Busca 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 Indexada



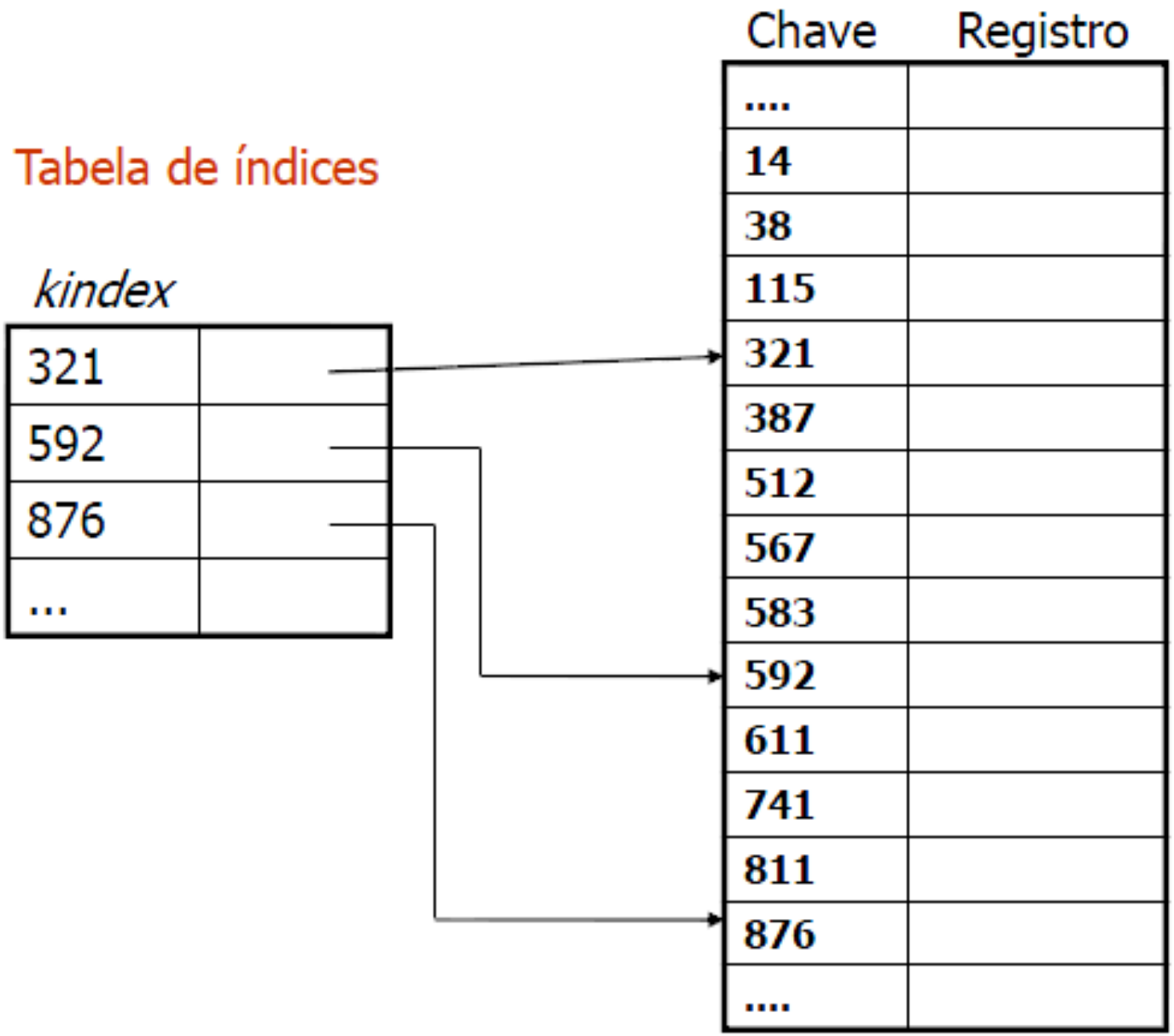
Busca Indexada

- Vantagem
  - Os itens na tabela poderão ser examinados seqüencialmente sem que todos os registros precisem ser acessados
    - O tempo de busca diminui consideravelmente
- Desvantagens
  - Exige espaço adicional para armazenar a(s) tabela(s) de índices

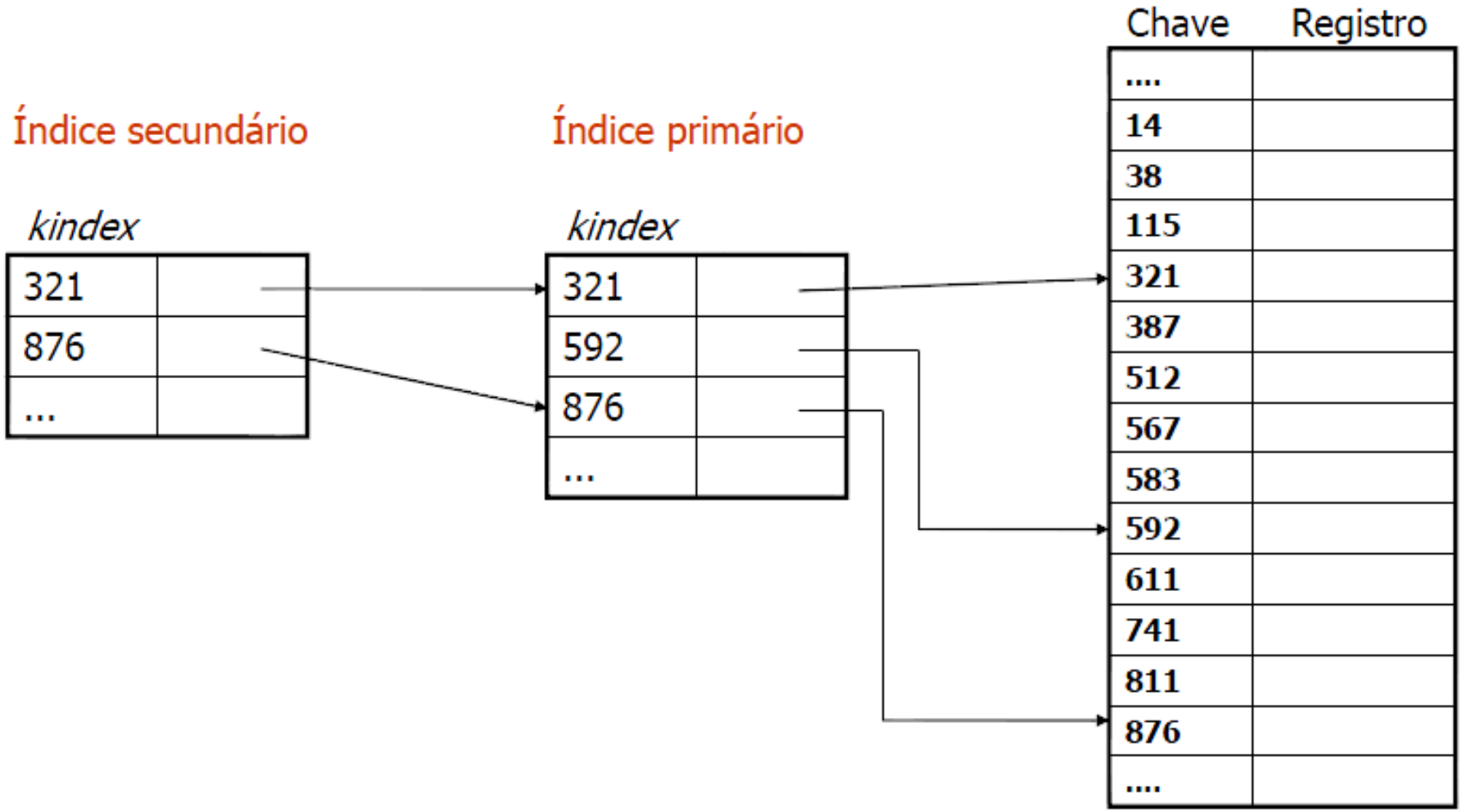
Busca Indexada

- Em tabelas com índices podemos utilizar os dois tipos de busca apresentados
- Elas são primeiramente aplicadas no índice e em seguida na tabela

Tabela de índices



Busca Indexada



Adaptado de

Métodos de Busca

Parte 1

SCC-201 Introdução à Ciência da Computação II

Rosane Minghim

2009/2011

Baseado no material dos Professores Rudinei Goularte e Thiago Pardo

Referências Bibliográficas

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; (2002). Algoritmos –Teoria e Prática. Tradução da 2ª edição americana. Rio de Janeiro. Editora Campus
- TAMASSIA, ROBERTO; GOODRICH, MICHAEL T. (2004). Projeto de Algoritmos -Fundamentos, Análise e Exemplos da Internet
- ZIVIANI, N. (2007). Projeto e Algoritmos com implementações em Java e C++. São Paulo. Editora Thomson