

Curso de Engenharia da Computação

Material Disciplina Estrutura de dados – II

Estruturas de Busca de Dados

Busca de dados por Seleção

Parte-III

Prof. Wagner Santos C. de Jesus

wsantoscj@gmail.com

Conceito de Busca de Palavras

Conceito

No contexto computacional, esse problema é conhecido como substring searching (*Procurar de Subcadeia*) ou String Matching (*Coincidência de Cadeias*) e pode ser formulado assim:

Encontrar as ocorrências de um dado vetor t em um dado vetor p .

Objeto de estudo

Para se localizar todas as ocorrências de um vetor $p[1..n]$ em um vetor $t[1..m]$.

Dizemos que um vetor $p[]$ ocorre em um vetor $t[]$ se existe k no intervalo $n..m$ tal que $p[1..n]$ seja sufixo de $t[1..k]$.

Aplicações da Técnica

- Pesquisas na Web (padrões)
- Aplicações linguísticas
- Cadeia de DNA
- Engenharia de software (controle de versões)


Objetivo:


- Descobrir se o string P (padrão) está em T.
- Força bruta: Caminha elemento a elemento da esquerda para a direita.
- Normalmente será retornado a primeira ocorrência do padrão, mas pode-se modificar o algoritmo para que continue encontrando mais padrões na string.

Problema da busca

Dizemos que um vetor $t[1..n]$ ocorre em um vetor $p[1..m]$ se existe k no intervalo $m..n$, tal que $t[1..n]$ é sufixo de $p[1..k]$.

3	1	3	1	4	3	1	4	1	3	1	4	1	5	9	3	1	4	1	5	9	2	6	3	1	4
									3	1	4	1	5	9											

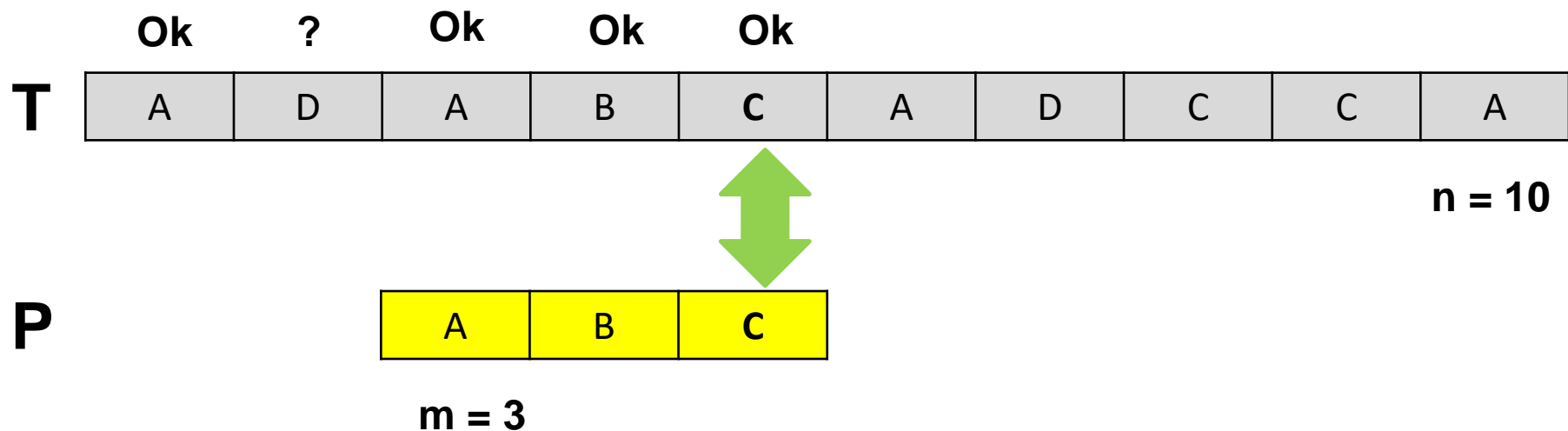




Funcionamento



Sequencia de deslocamento



Padrão encontrado
começando na posição - 2

Total de seis comparações para encontrar o padrão.

Problema

Encontrar o número de ocorrências de $p[1..m]$ em $t[1..n]$.

Suporemos que t e p são vetores de caracteres, embora o problema também faça sentido para outros tipos de dados. Diremos então que t seria o texto e p a palavra a ser encontrada (padrão).

Encontrando padrões e texto usando Força

Algoritmo ForcaBruta(T, P) Bruta

// T possui n caracteres e P possui m caracteres

Para i de 0 até (n – m) faça

 j <- 0

 enquanto (j < m .e. T[i+j] == P[j]) faça

 j <- j + 1

 se j == m então

 retorne i

Fim-para

Retorne “Não existe substring em T igual a P.”

Preencha a tabela abaixo com os valores do
algoritmo

t	p	i	m	n	j

t = “Aula na Univap, sábado” => t(n)

p = “Univap” => p(m)

Exercício (Proposto)

Implementar e testar o algoritmo de força bruto para encontrar padrões em cadeias fornecidas. Caso não encontre o padrão retornar (-1).

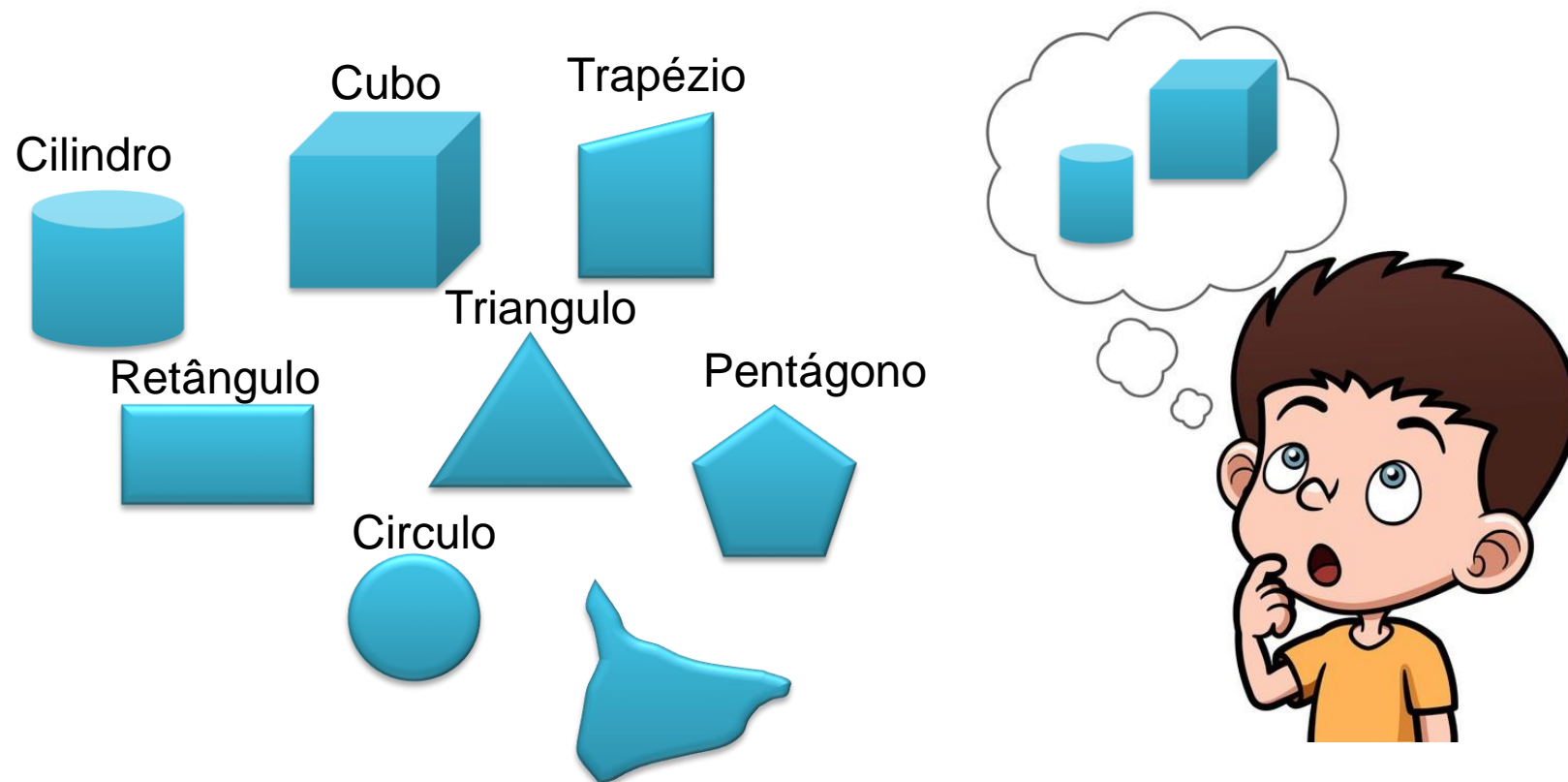
```
inteiro ForcaBruta(T, P) {  
    // Implementação  
}
```

Métodos de Seleção

Conceito de Seleção

Selecionar consiste em pesquisar informação com características específicas ou relevantes.

Quais das figuras geométrica conhecidas abaixo, apresentam características de tridimensionalidade ?



Aplicabilidade

Implementar algoritmos, que realizam, a varredura em uma dada sequencia, determinando as características acerca dos seus elementos.

Apresentam-se nesta situação os seguintes algoritmos

- Maior Valor;
- Menor Valor;
- Primeiro valor que serve;
- Melhor valor que serve;
- Pior valor que serve;
- K-ésimo menor valor.

Condição para se implementar algoritmos de seleção

Vamos considerar uma estrutura de dados com elementos inteiros positivos e de sequencias não vazias. Pelos quais os algoritmos podem ser simplificados, uma vez que não tem de lidar com está situação anómala.

Características das Funções de Seleção

Com exceção do algoritmo do K-ésimo menor valor, os demais possuem dois ou mais parâmetros incluídos a sequência de valores e, se necessário o valor que serve de referencia à pesquisa.

Elementos Principais de Seleção de uma Sequência

Conceito

Para se testar uma sequencia:

Elementos	Descrição
pSeq	Elementos da sequencia
pInicio	Posição inicial da sequencia
pFim	Posição final da sequencia
pVal	Valor base a ser procurado, em uma sequencia.

Implementação dos Métodos de Seleção

Funcionamento da Seleção

Um método de seleção tem como comportamento, realizar a varredura entre o índice inicial e o índice final fornecidos, se pretendermos analisar toda a sequencia, basta invocar a função a partir do primeiro elemento da sequencia, que é o índice zero, até ao ultimo elemento da sequencia.

Criação da classe Seleção

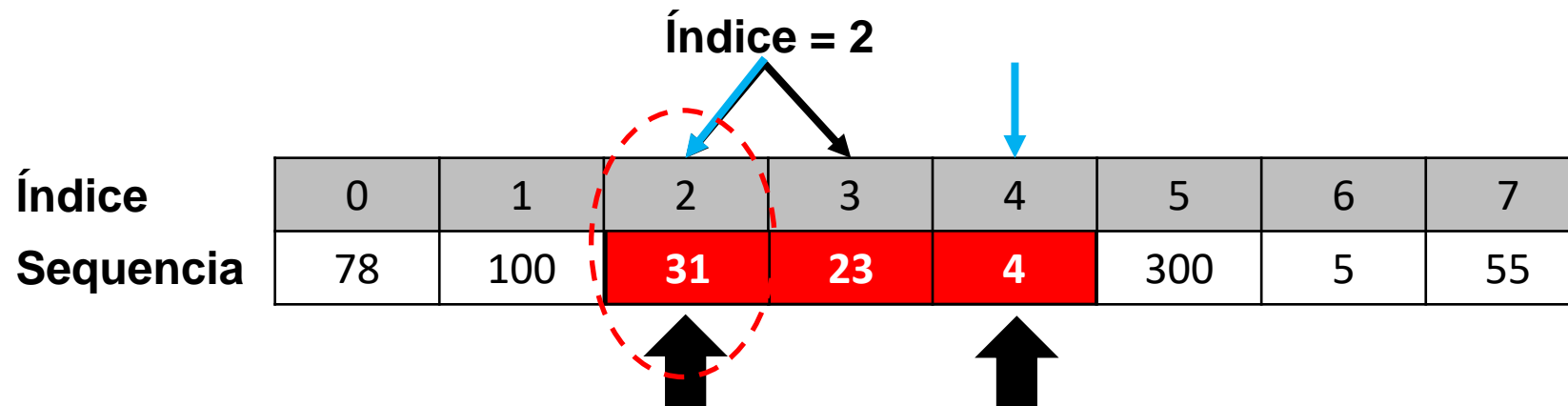
Selecao
<code>+int maxSelecao() +int minSelecao() +int primeiroSelecao() +int melhorSelecao() +int piorSelecao() +void kSelecao()</code>

O método de pesquisa por indexação, consiste em determinar a busca usando o deslocamentos em determinados índices da sequencia.

Seleção do Maior Valor em uma Sequencia

Conceito Maior valor

Determinar e devolver o índice correspondente a posição do elemento da sequencia que armazena o maior valor.



Índice	0	1	2	3	4	5	6	7
Sequencia	78	100	31	23	4	300	5	55

Método que encontra o maior Valor.

```
publico estatico inteiro maxSelecao(inteiro pSeq[], inteiro pInicio, inteiro pFim) {  
    inteiro max = pInicio;  
    para(int indice=pInicio+1; indice <= pFim; indice++)  
        se (pSeq[indice] > pSeq[max]) então  
            max = indice;  
    fim-se  
    fim-para  
    retorna max;  
}
```

Conceito Menor valor

Determinar e devolver o índice correspondente a posição do elemento da sequencia que armazena o menor valor.

			Índice = 3	Índice = 4				
Índice	0	1	2	3	4	5	6	7
Sequencia	78	100	31	23	4	300	5	55

The diagram shows an array with 8 elements. The indices are 0 to 7. The values are 78, 100, 31, 23, 4, 300, 5, and 55. The element at index 4 (value 4) is circled in red. A red dashed circle highlights the element at index 4. A blue arrow points from 'Índice = 3' to the element at index 3 (23). A black arrow points from 'Índice = 4' to the element at index 4 (4). A black arrow points up to the element at index 2 (31).

Método que encontra o menor valor

```
publico estatico inteiro minSelecao(inteiro pSeq[], inteiro pInicio, inteiro pFim) {  
    inteiro min = pInicio;  
    para(inteiro indice=pInicio+1; indice <= pFim; indice++){  
        se(pSeq[indice] < pSeq[min]) então  
            min = indice;  
    fim-para  
    fim-enquanto  
    retorna min;  
}
```

Seleção do Primeiro Valor mais Próximo do Procurado

Conceito do valor que serve

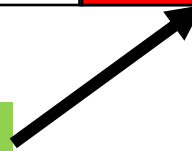
Existe alguns problema cuja solução otimizada requer que se procure um determinado valor em uma sequencia, mas se este valor não existir, podemos em alternativa utilizar um valor próximo do valor pretendido (first fit).

Conceito do primeiro valor mais próximo (valor que serve)

```
publico estatico inteiro primeiroSelecao(inteiro pSeq[], inteiro plnicio, inteiro pFim, int pVal){
    para (inteiro indice = plnicio; indice <= pFim; indice++)
        se (pSeq[indice] <= pVal) então
            return indice;
        fim-se
    fim-para
    return -1;
}
```

0	1	2	3	4	5	6	7
78	100	31	23	4	300	5	55

pVal = 35



Seleção do Melhor Valor que serve

Conceito valor mais próximo do valor pretendido

A presente estratégia de otimização consiste em procurar o valor mais próximo do valor pretendido, ou seja, o melhor valor (best fit).

Melhor valor mais próximo

```
publico estatico inteiro melhorSelecao(inteiro pSeq[], inteiro pInicio, inteiro pFim, inteiro pVal){
    inteiro melhor = primeiroSelecao(pSeq,pInicio,pFim,pVal);
    se (melhor == -1) então
        return -1;
    fim-se
    para (inteiro indice=melhor+1; indice <= pFim; indice++){
        se (pSeq[indice] > pSeq[melhor] && pSeq[indice] <= pVal) então
            melhor = indice;
        fim-se
    fim-para
    return melhor;
}
```

0	1	2	3	4	5	6	7
78	100	31	23	4	300	5	55

pVal = 30

Seleção do pior valor que serve

Conceito do pior valor a ser encontrado

Essa estratégia de otimização consiste em procurar o valor menos próximo do valor pretendido, ou seja, o pior valor (worst fit).

Pior valor mais próximo

```
publico estatico inteiro piorSelecao(inteiro pSeq[], inteiro pInicio, inteiro pFim, inteiro pVal){
    inteiro pior = primeiroSelecao(pSeq,pInicio,pFim,pVal);
    se (pior == -1) então
        return -1;
    fim-se
    para (inteiro indice=pior+1; indice <= pFim; indice++)
        se(pSeq[indice] < pSeq[pior]) então
            pior = indice;
    fim-se
    fim-para
    return pior;
}
```

0	1	2	3	4	5	6	7
78	100	31	23	4	300	5	55

pVal = 30



Observação

Caso haja a pesquisa iniciada no elemento de índice 5 da sequência, com busca no valor 30, não encontraremos qualquer valor que não exceda trinta, assim os algoritmos primeiro que serve, o melhor que serve e o pior que serve devolvem o índice -1.

0	1	2	3	4	5	6	7
78	100	31	23	4	300	100	330

pVal = 30



Seleção do K-ésimo menor valor

Conceito K-ésimo elemento

Consiste em pegar o K-ésimo elemento da sequência como pivô. Depois, percorrer a sequência da esquerda para a direita, para encontrar valores maiores do que o pivô, e da direita para a esquerda para encontrar valores menores que ele. Encontrado o respectivo valor, em seguida será trocado.

0	1	2	3	4	5	6	7
31	23	4	100	90	78	300	330

Implementação K-ésimo elemento

```
publico estatico kSelecao(inteiro pSeq[],inteiro pInicio,inteiro pFim, inteiro pK){  
    inteiro i = pInicio;  
    inteiro j = pFim;  
    inteiro pivot = pSeq[--pK];  
    se(pInicio < pFim) então  
        faça{  
            enquanto(pSeq[i] < pivot) i++;  
            enquanto(pSeq[j] > pivot) j--;  
            se(i < j) então  
                troca(pSeq, i,j);  
                i++;  
                j--;  
            fim-se  
        }enquanto(i<j);  
    fim-se  
}
```

Troca posição dos elementos

```
publico estatico troca(inteiro[] pSeq, inteiro pl, inteiro pJ){
    inteiro temp = pSeq[pl];
    pSeq[pl] = pSeq[pJ];
    pSeq[pJ] = temp;
}
```

0	1	2	3	4	5	6	7
31	23	4	100	90	78	300	330

Métodos de Busca de Dados

Métodos de Busca

A maneira mais simples de pesquisar uma sequencia é a pesquisa sequencial, também chamada pesquisa linear, que consiste em analisar todos os elementos da sequencia, metodicamente até que encontre o elemento da posição.

Método Busca

- Busca Sequencial
- Busca Sequencial ordenada
- Busca Binária
- Busca Ternária
- Busca por Interpolação

Busca Sequencial

```
public static int pesquisaSequencia(int[] pSeq, int pInicio, int pFim, int pVal){  
    for(int indice=pInicio; indice <= pFim; indice++){  
        if(pSeq[indice] == pVal)  
            return indice;  
    }  
    return -1;  
}
```

0	1	2	3	4	5	6	7
4	23	31	78	90	100	300	330

=

0	1	2	3	4	5	6	7
4	23	31	78	90	100	300	330



pVal

78

Indice = 3

Busca Sequencia Ordenada


0	1	2	3	4	5	6	7
4	23	31	78	90	100	300	330

Busca Sequencia, Ordenada

```
public static int buscaSeqOrdenada(int[] pSeq, int plnicio, int pFim, int pVal){
    int indice;
    for(indice=plnicio;indice<=pFim;indice++){
        if(pSeq[indice] >= pVal) break;
    }
    if(indice <= pFim && pSeq[indice] == pVal)
        return indice;
    else
        return -1;
}
```

>=

0	1	2	3	4	5	6	7
4	23	31	78	90	100	300	330



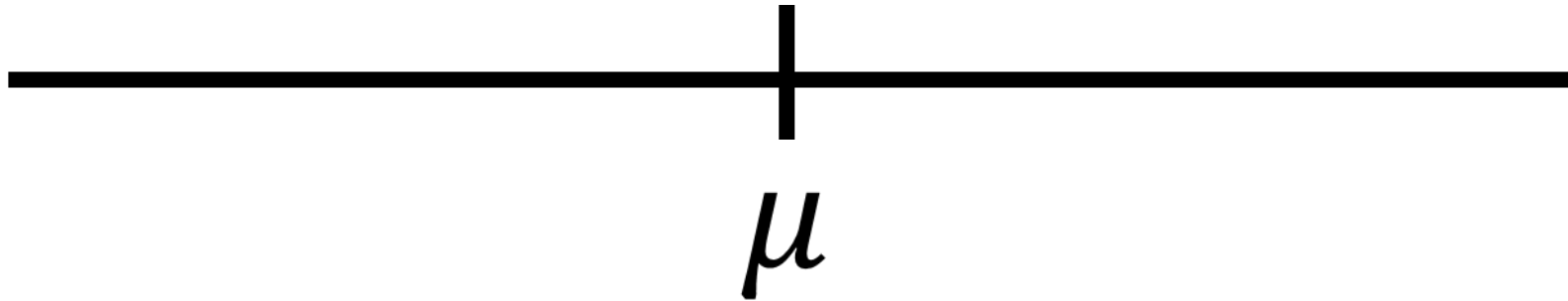
pVal
78

indice = 3 <= pFim = 7

Pesquisa Binária (Bisseccção)

Conceito Pesquisa Binária

Consiste em verificar em uma sequencia ordenada, utilizando uma partição sucessiva da mesma em duas metades, com objetivo de diminuir o número de elementos a analisar.



Importante (Pesquisa Binária)

Para que a pesquisa binária funcione, os elementos da sequência devem estar ordenados.

Algoritmo Pesquisa Binária

```
publico statico inteiro pesquisaBinaria(inteiro[] pSeq, inteiro pInicio, inteiro pFim, int pVal){
    min = pInicio;
    max = pFim;
    med;
    enquanto (min <= max) faça
        med = (min + max) / 2;
        se(pSeq[med] == pVal)
            return med;
        se(pSeq[med] < pVal)
            min = med + 1;
        senão
            max = med - 1;
    fim-enquanto
    return -1;
}
```

0	1	2	3	4	5	6	7
4	23	31	78	90	100	300	330

Operador Binário

Esta formula de cálculo pode dar problema para sequencias muito grandes. Nesse caso, se o valor pesquisado se encontra na metade final da sequencia, a soma dos índices inicial e final pode exceder o máximo inteiro positivo $(2^{32}-1) = 4.294.967.295$ e ocasionar em overflow. A forma correta e eficiente de calcular a posição média é dada pela seguinte expressão:

$$\text{med} = (\text{min} + \text{max}) \ggg 1;$$

Algoritmo Pesquisa Binária

```
publico statico inteiro pesquisaBinaria(inteiro[] pSeq, inteiro plnicio, inteiro pFim, int pVal){
    min = plnicio;
    max = pFim;
    med;
    enquanto (min <= max) faça
        med = (min + max) >>> 1;
        se(pSeq[med] == pVal)
            return med;
        se(pSeq[med] < pVal)
            min = med + 1;
        senão
            max = med - 1;
    fim-enquanto
    return -1;
}
```

128	64	32	16	8	4	2	1
0	0	0	0	0	1	1	1

Pesquisa Ternária

Conceito (Tricotômica)

Em cada passo da pesquisa, o número de elementos da sequencia que tem de ser analisados é reduzido a um terço, pelo que este método de pesquisa é potencialmente mais eficiente do que a pesquisa binária.



Primeiro Passo

Encontrar o intervalo da sequencia ordenada:

$$nelem = \max - \min + 1$$

0	1	2	3	4	5	6	7
4	23	31	78	90	100	300	330

$$nelem = 7 - 0 + 1 = 8$$

Encontrar os pivôs

Se $nelem \bmod 3 = 0$

$$fpivot = min + \left(\frac{nelem}{3}\right) - 1$$

$$spivot = min + \left(\frac{2nelem}{3}\right) - 1$$

senão

$$fpivot = min + \left(\frac{nelem}{3}\right)$$

$$spivot = min + \left(\frac{2nelem}{3}\right)$$

```
publico statico inteiro pesquisaTernaria(inteiro[]pSeq, inteiro plnicio, inteiro pFim, inteiro pVal) {  
    inteiro min = plnicio;  
    inteiro max = pFim;  
    inteiro nelem;  
    inteiro fpivot;  
    inteiro spivot;  
    enquanto (min < max) faça  
        nelem = max - min + 1;  
        se(nelem % 3 == 0) então  
            fpivot = min + (nelem/3) - 1;  
            spivot = min + (2*nelem/3)-1;  
        senão {  
            fpivot = min + (nelem/3);  
            spivot = min + (2*nelem/3);  
        fim-se  
        se (pSeq[fpivot] > pVal) então  
            max = fpivot - 1;  
        senão  
            se(pSeq[spivot] > pVal) então  
                min = fpivot;  
                max = spivot - 1;  
            senão  
                min = spivot;  
        fim-se  
    fim-enquanto  
    se (pSeq[min] == pVal) então  
        retorno min;  
    senão  
        retorno -1;  
}
```

Pesquisa

Interpolação

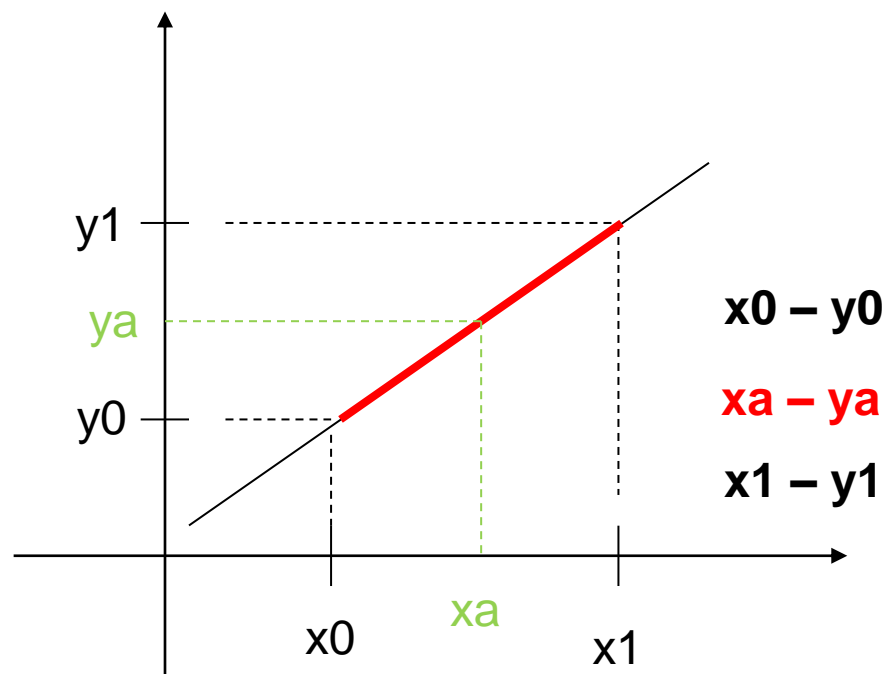
Conceito de Pesquisa por Interpolação

Método de pesquisa é semelhante ao da pesquisa binária, mas em vez de calcular a próxima posição de pesquisa usando os índices dos elementos extremos da sequência, usa-se os seus valores e o valor procurado.

Demonstração Matemática

Interpolação Linear

Interpolação Linear



$$\frac{y_a - y_0}{y_1 - y_0} = \frac{x_a - x_0}{x_1 - x_0}$$

$$y_a - y_0 = \frac{x_a - x_0}{x_1 - x_0} \cdot (y_1 - y_0)$$

$$y_a = y_0 + \frac{x_a - x_0}{x_1 - x_0} \cdot (y_1 - y_0)$$

Observação

Para execução da busca por interpolação os valores devem estar ordenados, usando a ordenação crescente.

Equação de Interpolação Linear

$$medio = mínimo + \left[\frac{valor\ Procurado - Seq[mínimo]}{Seq[máximo] - Seq[mínimo]} x (máximo - mínimo) \right]$$

Mínimo = Menor Índice da sequencia;
Máximo = Maior índice da sequencia;
Valor Procurado = Valor a ser encontrado na sequencia
Seq[mínimo] = Menor valor da sequencia
Seq[máximo] = Maior valor da sequencia
(máximo – mínimo) = Intervalo da sequencia a ser testada.

Algoritmo de Pesquisa Interpolação

```
publico statico inteiro interpolacaop(inteiro[] pSeq,  
inteiro plnicio, inteiro pFim, inteiro pVal){
```

```
    inteiro min = plnicio
```

```
    inteiro max = pFim
```

```
    inteiro med
```

```
    inteiro dif
```

```
    se(pVal < pSeq[min] || pVal > pSeq[max])
```

```
        retorna -1
```

```
    fim-se
```

Continuação, Interpolação

```
enquanto min <= max faça
    dif = pSeq[max] - pSeq[min]
    se(dif == 0)
        med = min
    senão
        med = min + (pVal - pSeq[min]) * (int) (max - min) / dif
    fim-se
    se(pSeq[med] == pVal)
        retorna med
    fim-se
    se(pSeq[med] > pVal)
        max = med - 1
    senão
        min = med + 1
    fim-se
fim-enquanto
retorna -1;
}
```

Pesquisa por Dispersão

Conceito de Dispersão

Função de Dispersão (FD)

Uma função que transforma um valor, seja ele numérico ou alfanumérico, num valor numérico, que está enquadrado num determinado intervalo de valores, designa-se por função de dispersão (hash function).

Conceito Tabela Hash

Uma tabela hash, vem ser uma técnica de estrutura de dados que permite armazenar valores associados as chaves.

Funcionamento função Hash()

0		→ k_2
1		
2		→ k_3
3		→ k_1
4		

inserir(k_1, v)

- inserir(k_1) = 3

inserir(k_2, v)

- inserir(k_2) = 0

inserir(k_3, v)

- inserir(k_3) = 2

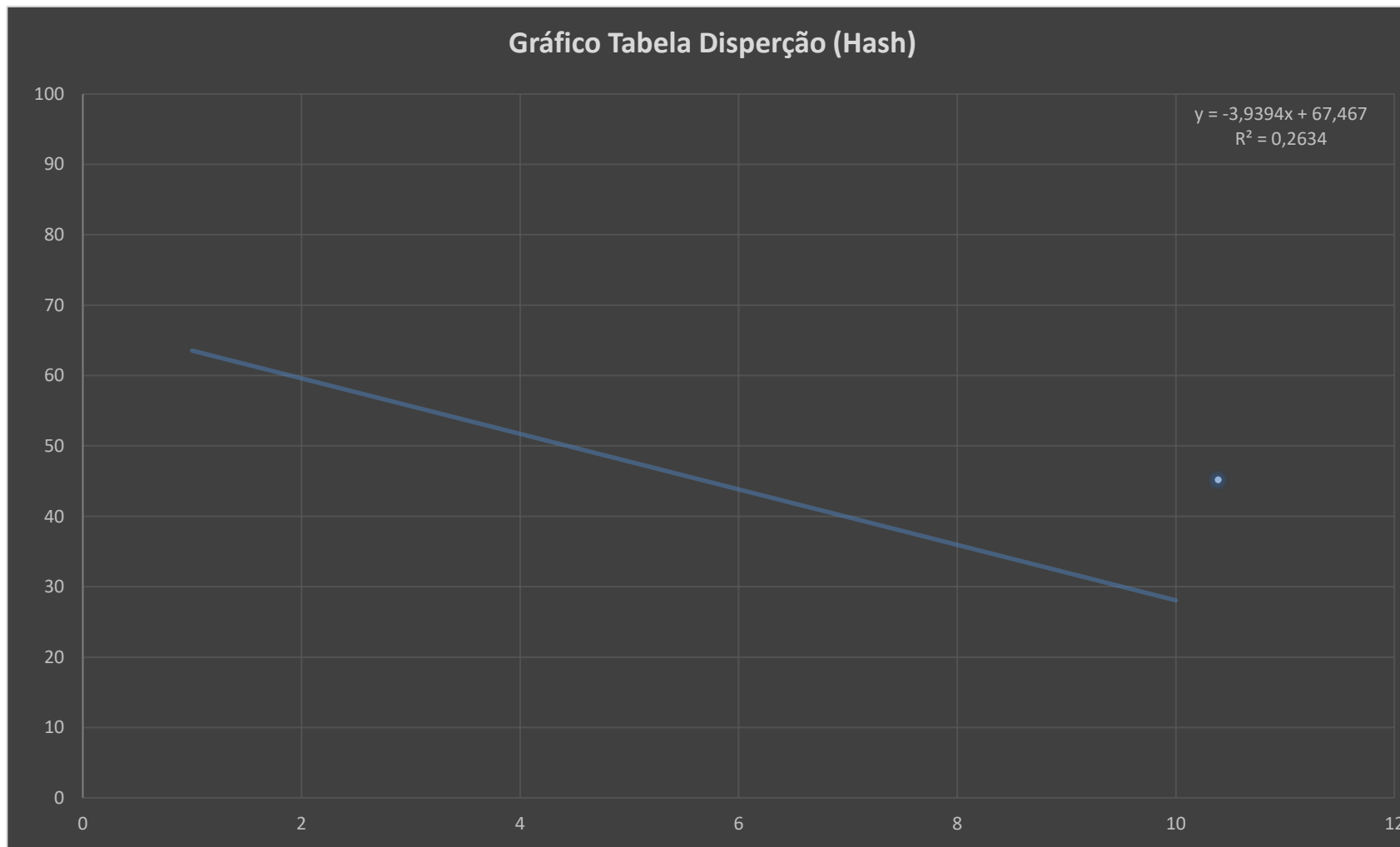
inserir(k_4, v)

- inserir(k_4) = 0

Colisão

Para realizar a busca, basta usar o mesmo processo, verificando a igualdade do elemento da posição.

Dispersão de valores (Hash)



Antes (Hash) = [30,45,66,70,90,50,32,35,18,22]

Depois (Hash) = [30,70,90,50,32,45,66,35,18,22]

Característica de uma Função de Dispersão

- Ser fácil e rápida de calcular (Eficiência).
- Quando um valor no intervalo definido pelo limites da sequencia (Modular).
- Assegurar uma boa dispersão.
- Produzir poucas colisões.

Fator de Carregamento

Define-se como fator de carregamento (load fator) de uma tabela de dispersão α como sendo a razão, entre o número de elementos armazenados num dado instante (E) é a dimensão da sequencia (N), ou seja.

$$\alpha = \frac{E}{N}$$

Sendo o fator de carregamento ideal $\alpha \leq 0.5$.

Função Hash

$$f(v, k) = h$$

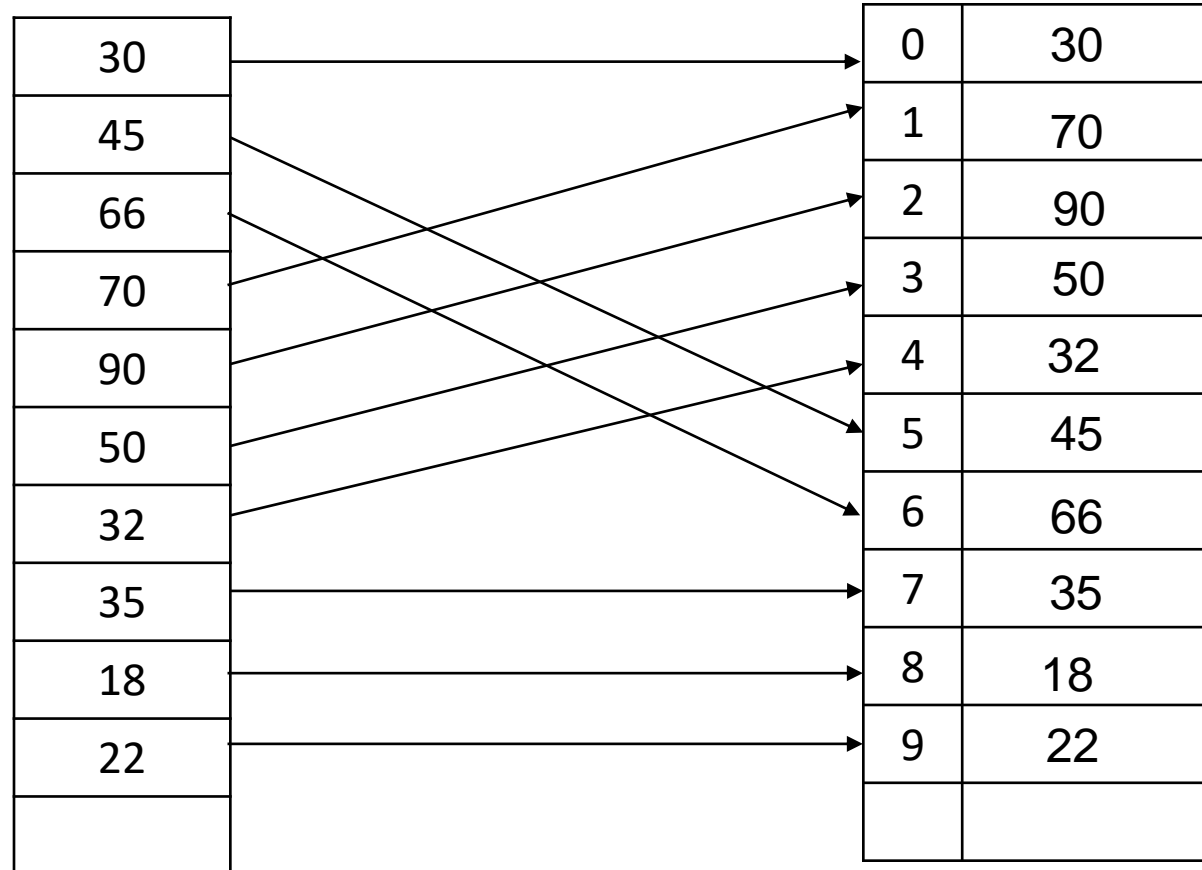
Onde: v elemento a ser armazenado e k a dimensão da tabela de dispersão.

Calculo:

$$h = v \bmod K$$

Funcionamento da Dispersão

{30,45,66,70,90,50,32,35,18,22}



Tratamento de colisão

Sondagem Linear

Conceito de Sondagem Linear

A Sondagem linear consiste em analisar as posições seguintes à posição determinada pela função de dispersão, até encontrar uma posição livre para colocar o valor pretendido.

HASH[0]	11000	← Inserir 22000 (1ª. Tentativa)
HASH[1]	10990	← Inserir 22000 (2ª. Tentativa)
HASH[2]	10892	← Inserir 22000 (3ª. Tentativa)
HASH[3]	993	← Inserir 22000 (4ª. Tentativa)
HASH[4]	22000	← Inserir 22000 (5ª. Tentativa)
HASH[10]	22010	

Observação sobre Sondagem Linear

Este método de sondagem gera as posições e verifica se estão ocupadas ou não, a partir da posição determinada pela função de dispersão, até encontrar uma posição livre ou determinar que a tabela está cheia, não sendo possível colocar o novo valor.

Implementação da Tabela de Dispersão

Implementação do Hash

```
publico statico inteiro hash(inteiro pVal, inteiro ptabdim) {  
    retorno pVal mod ptabdim;  
}
```

Calcula o hash mediante a uma chave(tamanho da tabela) e valor a ser armazenado.

Inserção na Tabela de Hash

```
publico statico inserirTabela(inteiro[] pSeq, inteiro pVal)
{
    inteiro ph = hash(pVal, pSeq.length);
    se(pSeq[ph] == 0) então
        pSeq[ph] = pVal;
    senao
        inteiro pos = (ph+1) % pSeq.length;
        enquanto (pSeq[pos] != 0 && pos != ph)
            pos = ++pos % pSeq.length;
        fim-enquanto
        se(pos != ph) {
            pSeq[pos] = pVal;
        }
    fim-se
}
fim-se
```

```
publico statico inteiro procuraTabela(inteiro[] pSeq, inteiro pVal){  
    inteiro ph = hash(pVal, pSeq.length);  
    se(pSeq[ph] == pVal) então  
        retorna ph;  
    fim-se  
senão  
    se(pSeq[ph]==0)  
        retorna -1;  
    senão  
        inteiro pos = (ph+1)%pSeq.length;  
        enquanto (pSeq[pos] != pVal && pSeq[pos] != 0 && pos != ph)  
            pos = ++pos % pSeq.length;  
        fim-enquanto  
        se (pSeq[pos] == pVal) então  
            retorna pos;  
        senão  
            retorna -1;  
        fim-se  
    fim-se  
}
```

Procura na tabela de Hash


```
publico statico Caracteres mostra(inteiro[] pSeq) {
```

```
    se (pSeq.length == 0) então
```

```
        return "[]";
```

```
    fim-se
```

```
    String str = "[";
```

```
    para (int i=0;i<=pSeq.length-1;i++)
```

```
        se(i < (pSeq.length-1)) então
```

```
            str+= pSeq[i]+",";
```

```
        senão
```

```
            str+= pSeq[i];
```

```
        fim-se
```

```
    fim-para
```

```
    return str+"]";
```

```
}
```

Mostrar dados da tabela