Pesquisa em Memória Primária

Prof. Tiago Massoni

Engenharia da Computação

Poli - UPE

Busca (pesquisa)

- 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**
  - Encontrar uma ou mais ocorrências de registros com chaves iguais à chave de pesquisa.
- **Pesquisa com sucesso** X **Pesquisa sem sucesso**

Conceitos

- Conjunto de registros ou arquivos: tabelas
- **Tabela:** associada a entidades de vida curta, criadas na memória interna durante a execução de um programa
- **Arquivo:** associado a entidades de vida mais longa, armazenadas em memória externa
  - **Distinção não é rígida**

Tipos de pesquisa

- Depende principalmente
  - Quantidade dos dados envolvidos
  - Pode estar sujeito a inserções e retiradas freqüentes
- Se conteúdo do arquivo é estável é importante minimizar o tempo de pesquisa, sem preocupação com o tempo necessário para estruturar o arquivo

TAD Dicionário

- Nome comumente utilizado para descrever uma estrutura de dados para pesquisa
- **Operações**
  1. Inicializa
  2. Pesquisa
  3. Insere
  4. Retira
- Analogia com um dicionário da língua portuguesa:
  - Chaves = palavras
  - Registros = entradas associadas com cada palavra
    - pronúncia
    - definição
    - sinônimos
    - outras informações

Pesquisa seqüencial

- Método de pesquisa mais simples
- A partir do primeiro registro, pesquise seqüencialmente até encontrar a chave procurada; então pare
- Armazenamento de um conjunto de registros por meio do tipo estruturado arranjo
  - Se ordenado, então pesquisa sem sucesso é mais eficiente

Pesquisa seqüencial

```
public class Tabela {
  private Comparable registros[]; private int n;
  public Tabela (int maxN){
    this.registros= new Comparable[maxN+1];
    this.n= 0;
  }

  public int pesquisa (Comparable reg){
    this.registros[0]= reg; //sentinela
    int i = this.n;
    while (this.registros[i].compareTo(reg)!=0)
      i--;
    return i ;
  }

  public void insere (Comparable reg){
   if(this.n == (this.registros.length-1))
     /*ERRO! Tabela cheia!!*/
   this.registros[++this.n]= reg;
  }
}
```

Pesquisa seqüencial

- Cada registro contém um campo chave que identifica o registro
  - Além da chave, podem existir outros componentes em um registro, que não têm influência nos algoritmos
- O método pesquisa retorna o índice do registro que contém a chave no parâmetro; caso não esteja presente, o valor retornado é zero
- Essa implementação não suporta mais de um registro com a mesma chave
  - Formas alternativas: retornar lista de índices, por exemplo

Pesquisa seqüencial

- Utilização de um registro sentinela na posição zero do array
  - Garante que a pesquisa sempre termina (se o índice retornado por pesquisa for zero, a pesquisa foi sem sucesso)
  - Não é necessário testar se i > 0, devido a isto
- O anel interno da pesquisa é extremamente simples
  - o índice i é decrementado e a chave de pesquisa é comparada com a chave que está no registro
- Isto faz com que esta técnica seja conhecida como pesquisa seqüencial rápida

Custo da pesquisa sequencial

- 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 seqüencial é a **melhor escolha** para o problema de pesquisa em tabelas com até **25 registros**

Pesquisa binária

- Pesquisa em tabela pode ser mais eficiente se registros forem mantidos em ordem
- Para saber se uma chave está presente na tabela:
  - Compare a chave com o registro que está na posição do meio da tabela
  - **Se** a chave é menor **então** o registro procurado está na primeira metade da tabela
  - **Se** a chave é maior **então** o registro procurado está na segunda metade da tabela
  - Repita o processo até que a chave seja encontrada, ou fique apenas um registro cuja chave é diferente da procurada, significando uma pesquisa sem sucesso

Pesquisa binária

```
public int binaria(Comparable chave){
  if(this.n==0) return 0;
  int esq= 1 ,dir= this.n, i;
  do{
    i=(esq + dir)/2;
    if(chave.compareTo(this.registros[i])>0)
       esq= i + 1;
    else dir= i - 1;
  }while((chave.compareTo(this.registros[i])!=0)
        && (esq<=dir));
  if(chave.compareTo(this.registros[i])==0)
    return i;
  else return 0;
}
```

Exemplo: procura pela chave G

	1	2	3	4	5	6	7	8
Chaves iniciais:	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*
	A	*B*	*C*	***D***	*E*	*F*	*G*	*H*
					E	***F***	*G*	*H*
							G	*H*

Análise da pesquisa binária

- A cada iteração do algoritmo, o tamanho da tabela é dividido ao meio
- 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
  - A pesquisa binária não deve ser usada em aplicações muito dinâmicas