Pesquisa em Memória Primária

Livro “Projeto de Algoritmos” – Nívio Ziviani

Capítulo 5 – Seções 5.1 e 5.2

http://www2.dcc.ufmg.br/livros/algoritmos/

Pesquisa em Memória Primária

- Introdução - Conceitos Básicos
- Pesquisa Seqüencial
- Pesquisa Binária
- Árvores de Pesquisa
  - Árvores Binárias de Pesquisa sem Balanceamento
  - Árvores Binárias de Pesquisa com Balanceamento
- Pesquisa Digital
  - Trie , Patricia
- Transformação de Chave (Hashing)
  - Listas Encadeadas, Endereçamento Aberto, Hashing Perfeito

Introdução - Conceitos Básicos

- 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 Pesquisa sem sucesso.**

Introdução - Conceitos Básicos

- **Tabelas & Arquivos**
  - Conjunto de registros
  - Tabela: associada a entidades de vida curta, criadas na memória interna durante a execução de um programa.
  - Arquivo: geralmente associado a entidades de vida mais longa, armazenadas em memória externa.
  - Distinção não é rígida:
    - tabela: arquivo de índices
    - arquivo: tabela de valores de funções.

Escolha do Método de Pesquisa mais Adequado a uma Determinada Aplicação

- Depende principalmente:
  1. Quantidade dos dados envolvidos.
  2. Arquivo 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

Algoritmos de Pesquisa
Tipos Abstratos de Dados

- É importante considerar os algoritmos de pesquisa como tipos abstratos de dados, com um conjunto de operações associado a uma estrutura de dados, de tal forma que haja uma independência de implementação para as operações.

- Operações mais comuns:
  1. Inicializar a estrutura de dados.
  2. Pesquisar um ou mais registros com determinada chave.
  3. Inserir um novo registro.
  4. Retirar um registro específico.
  5. Ordenar um arquivo para obter todos os registros em ordem de acordo com a chave.
  6. Ajuntar dois arquivos para formar um arquivo maior.

Dicionário

- Nome comumente utilizado para descrever uma estrutura de dados para pesquisa.
- Dicionário é um tipo abstrato de dados com as operações:
  1. Inicializa
  2. Pesquisa
  3. Insere
  4. Retira

- Analogia com um dicionário da língua portuguesa:
  - Chaves $\Leftarrow\Rightarrow$ palavras
  - Registros $\Leftarrow\Rightarrow$ entradas associadas com
    - 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:

Pesquisa Seqüencial

```
# define MAX              10
typedef long TipoChave;

typedef struct {
   TipoChave Chave;
   /* outros componentes */
} Registro;

typedef int Indice;

typedef struct {
   Registro Item[MAX + 1];
   Indice n;
} Tabela;
```

Pesquisa Seqüencial

- Implementação para as operações Inicializa, Pesquisa :

```
void Inicializa(Tabela *T)
{
    T->n = 0;
}

Indice Pesquisa(TipoChave x, Tabela *T)
{ int i;

  T->Item[0].Chave = x;
  i = T->n + 1;
  do {
       i--;
    }  while (T->Item[i].Chave != x);
  return i;
}
```

Pesquisa Seqüencial

- Implementação para a operacao Insere:

```
void Insere(Registro Reg, Tabela *T)
{
     if (T->n == MAX)
        printf("Erro : tabela cheia\n");
     else
     {
        T->n++;
        T->Item[T->n] = Reg;
     }
}
```

Pesquisa Seqüencial

- Pesquisa retorna o índice do registro que contém a chave x;
- Caso não esteja presente, o valor retornado é zero.
- A implementação não suporta mais de um registro com uma mesma chave.
- Para aplicações com esta característica é necessário incluir um argumento a mais na função Pesquisa para conter o índice a partir do qual se quer pesquisar.

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 função 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.

Pesquisa Seqüencial

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 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**

  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 o processo até que a chave seja encontrada, ou fique apenas um registro cuja chave é diferente da procurada, significando uma pesquisa sem sucesso.

Exemplo de Pesquisa Binária para a 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*

Pesquisa Binária

```
Indice Binaria(TipoChave x, Tabela *T) {

  Indice i, Esq, Dir;

  if (T->n == 0) return 0;
  else {
     Esq = 1;
     Dir = T->n;
     do {
        i = (Esq + Dir) / 2;
        if (x > T->Item[i].Chave)
           Esq = i + 1;
        else
           Dir = i - 1;
     }
     while ( (x != T->Item[i].Chave) &&
             (Esq <= Dir) );

     if (x == T->Item[i].Chave)
         return i;
     else
         return 0;
  }
}
```

Pesquisa 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.
  - Conseqüentemente, a pesquisa binária não deve ser usada em aplicações muito dinâmicas.