unipac.br
Barbacena

Bacharelado em Ciência da Computação

Estruturas de Dados

Material de Apoio

Parte XVI – *Pesquisa em Memória Primária*

Prof. Nairon Neri Silva
naironsilva@unipac.br

2º sem / 2021

Pesquisa em Memória Primária

- A pesquisa em memória primária visa recuperar informação a partir de uma grande massa de informação previamente armazenada.
- O objetivo da pesquisa é encontrar uma ou mais ocorrências de registros com chaves iguais à ***chave de pesquisa.***
- O retorno da pesquisa pode ser
  - Com sucesso
  - Sem sucesso

Pesquisa em Memória Primária

- A escolha do método de pesquisa mais adequado a uma determinada aplicação depende:
  - Quantidade dos dados envolvidos
  - Arquivo estar sujeito a inserções e retiradas frequentes, ou ser praticamente estável (se conteúdo do arquivo é estável é importante minimizar o tempo de pesquisa, sem preocupação com o tempo necessário para estruturar o arquivo)

Pesquisa Sequencial

- Método mais simples.
- Funcionamento: a partir do primeiro registro, pesquise sequencialmente até encontrar a chave procurada, então pare.

Pesquisa Sequencial

- A pesquisa sequencial é ineficiente porque, no pior caso, compara ***x*** com cada um dos elementos do vetor.
- Essa implementação não suporta mais de um registro com uma mesma chave (é necessário incluir mais um argumento na função).

Pesquisa Sequencial - Algoritmo

```
/*Esta função recebe um número x e um vetor[0..n-1], e
   retorna k, de modo que v[k] == x, ou -1 caso não
   encontre o elemento x */

int buscaSequencial(int v[], int n, int x){
    int k;
    k = n - 1;
    while((k>=0) && (v[k]!=x)){
        k--;
    }
    return k;
}
```

Exercício prático – Busca Sequencial

Modifique o algoritmo de busca sequencial para que ele se transforme em um algoritmo de atualização, ou seja, ao encontrar um elemento igual, mude o seu valor para um valor passado para a função. A função terá os seguintes parâmetros:

- o vetor com os números
- o tamanho do vetor
- o número buscado
- o valor que será inserido no lugar do número buscado

No método main(), faça o teste da função criada com um vetor de números aleatórios.

Pesquisa binária

- Pesquisa em um vetor pode ser mais eficiente ⇒ ***se os registros forem mantidos em ordem.***
- Para saber se uma chave está presente:
  - 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

Pesquisa binária - Exemplo

- Exemplo: buscar pelo elemento ***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 - Algoritmo

```
int pesquisaBinaria(int v[], int n, int x){
    int e, m, d;
    e = 0;
    d = n-1;
    while(e <= d){
        m = (e + d)/2;
        if(v[m] == x){
            return m;
        }else if(v[m] < x){
            e = m + 1;
        }else if(v[m] > x){
            d = m - 1;
        }
    }
    return -1;
}
```

Pesquisa binária

- Análise
  - A cada iteração do algoritmo, o tamanho do vetor é dividido ao meio.
  - Ressalva: o custo para manter o vetor ordenado é alto: a cada inserção na posição ***p*** do vetor 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.

Pesquisa binária – versão recursiva

```
int pesquisaBinariaR(int v[], int e, int d, int x) {
    int m;
    if (e > d)
        return -1;
    else{
        m = (e + d) / 2;
        if (v[m] == x)
            return m;
        else if (x < v[m])
            return pesquisaBinariaR(v, e, m-1, x);
        else
            return pesquisaBinariaR(v, m+1, d, x);
    }
}
```