UNIVERSIDADE CATÓLICA DE PERNAMBUCO

# Algoritmos e Estrutura de Dados II

# Métodos de Pesquisa em Memória Primária

Prof Márcio Bueno

ed2tarde@marciobueno.com / ed2noite@marciobueno.com

# Pesquisa

- Por pesquisa (procura ou busca) entende-se o ato de recuperar uma informação em um conjunto de dados
- Como a atividade de ordenação, a atividade de busca é de grande importância devido a sua utilização freqüente nos mais diversos tipos de software.
- Exemplos:
  - Procurar o nº de telefone de uma pessoa em uma agenda de telefones.
  - Procurar por uma palavra num texto.
  - Dado um número entre 0 e 1000, adivinhar o número que se escolheu.

# Pesquisa

- Numa pesquisa cada unidade de informação é armazenada em uma estrutura do tipo registro contendo um campo chave (inteiro, string, etc.) além de outros campos.

```
tipo
    item = registro
    chave : inteiro
    {outros campos}
fim
```

- O conjunto dos registros de informação normalmente é armazenado como:
  - Listas lineares (vetores ou listas encadeadas)
  - Árvores binárias
- Este conjunto usualmente é chamado de tabela ("vida curta") ou de arquivo ("vida longa").

# Pesquisa

- O objetivo da pesquisa é encontrar uma ou mais ocorrências de registros com chaves iguais à chave de pesquisa
  - Esta operação pode resultar em sucesso ou insucesso.

# Algoritmo de Pesquisa

- A pesquisa é *uma tarefa muito utilizada*
  - As rotinas que a executam devem ser eficientes (executar no menor tempo possível)
  - O tempo de pesquisa depende do algoritmo de pesquisa utilizado
    - A escolha desse algoritmo depende diretamente:
      - Da quantidade de dados envolvidos
      - Da freqüência das operações de **inserção** e de **exclusão** de registros
  - Quando a operação de pesquisa é muito mais freqüente do que a de inserção, deve-se minimizar o tempo de pesquisa, através da ordenação dos registros.

# Algoritmos de Pesquisa em Memória Primária

- Pesquisa Seqüencial (ou Linear)
- Pesquisa Seqüencial com Sentinela
- Pesquisa Binária
- Pesquisa por Interpolação
- Pesquisa Direta (*Hashing*)

# Algoritmos de Pesquisa em Memória Secundária

- Árvore Binária
- Árvore B
- Árvore B+
- Árvore B*
- Árvore Patrícia

UNIVERSIDADE CATÓLICA DE PERNAMBUCO

# Pesquisa Seqüencial ou Linear

# Pesquisa Seqüencial

- Método bastante simples.
- Utilizado quando os dados não estão ordenados pela chave de pesquisa.
- Princípio:
  - Inicia a pesquisa pelo primeiro registro, avança seqüencialmente (registro por registro) e termina:
    - Com sucesso: chave pesquisada é encontrada, ou;
    - Sem sucesso: todos os registros são pesquisados, mas a chave não é encontrada.

# Pesquisa Seqüencial - Implementação

```
int pesqSeq(int chave, int v[], int n) {
    int i;
    for (i = 0; i < n; i++) {
        if (v[i] == chave) {
            return (i);
        }
    }
    return (-1); // Índice inválido
}
```

# Pesquisa Seqüencial

- Análise
  - Para uma pesquisa com sucesso, temos:
    - 1 iteração no melhor caso;
    - N iterações no pior caso;
    - (N + 1) / 2 iterações no caso médio.
  - Para uma pesquisa sem sucesso, temos:
    - N + 1 iterações.
  - O número total de comparações são:
    - Melhor caso: 2
    - Pior caso: ( n + 1 ) + n = 2n + 1
    - Médio caso: (2n + 1 + 2 ) / 2 = ( 2n + 3 )/ 2

# Pesquisa Seqüencial

- Análise
  - De forma geral o algoritmo é O(n) em complexidade
  - Este algoritmo é a melhor solução para o problema da pesquisa em tabelas desordenadas com poucos registros.

UNIVERSIDADE CATÓLICA DE PERNAMBUCO

# Pesquisa Seqüencial com Sentinela

# Pesquisa Seqüencial com Sentinela

- Embora muito simples, o algoritmo de pesquisa seqüencial pode ser acelerado, atribuindo-se a chave de pesquisa ao registro contido na posição N+1.
- Com isso, este registro fictício passa funcionar como sentinela: mesmo na pior das hipóteses, a chave será encontrada na posição N+1.
- Se durante a pesquisa seqüencial o elemento procurado for encontrado em uma posição anterior a N+1, isto significa que o elemento está na tabela.
- No entanto, se o elemento procurado só for encontrado na posição N+1, isto significa que ele não está na tabela.

# Pesquisa Seqüencial com Sentinela

- A idéia básica da pesquisa seqüencial com sentinela é usar o elemento procurado como indicação que a tabela acabou

Elimina a necessidade
de a cada passo no laço
testar se já chegou ao final
da tabela

# Pesquisa Seqüencial com Sentinela - Implementação

```
int pesqSeqSent(int chave, int v[], int n) {
    int i = 0;
    vet[n] = chave;
    while (vet[i] != chave) {
        i++;
    }
    if( i < n ) return i;
    return (-1); // Índice inválido
}
```

UNIVERSIDADE CATÓLICA DE PERNAMBUCO

# Pesquisa Binária

# Pesquisa Binária

- A pesquisa em uma tabela pode ser mais eficiente se os registros forem mantidos em ordem.
- **Princípio**:
  - Similar ao utilizado quando se procura o nome de um assinante em um catálogo telefônico impresso.
  - Compara-se a chave procurada com a chave do registro central do conjunto.
  - Esta comparação indica: a chave foi encontrada, ou em qual das metades a pesquisa deve prosseguir, segundo este mesmo princípio.

# Pesquisa Binária

- Algoritmo:
  - Para saber se uma chave está presente na tabela, compare a chave com o registro que está no 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 - Implementação

```
int pesqBin(int chave, int v[], int n) {
    int inicio = 0;
    int meio;
    int fim = n - 1;
    while (inicio <= fim) {
      meio = (inicio + fim) / 2;
      if (chave < v[meio]) {
          fim = meio - 1;
      } else if (chave > v[meio]) {
          inicio = meio + 1;
      } else {
          return meio;
      }
    }
    return -1; // Índice Impossível
}
```

# Pesquisa Binária

- Análise:
  - A cada iteração o número de elementos a serem pesquisados é reduzido à metade:
    - N, N/2, N/4, N/8, ..., **N**/$2^k$
  - Queremos que $N/2^k \leq 1$, logo $k \geq \log_2 N$
  - A chave pesquisada deve ser comparada com o último elemento restante, portanto o número máximo de comparações é $1 + \log_2 N$.
  - Conclusão: Sua ordem de complexidade é $O(\log_2 N)$.

# Pesquisa Binária

- Análise:
  - Exemplos:
    - Para tabela de 16 elementos $\Rightarrow$ 4 iterações
    - Para 1024 elementos $\Rightarrow$ 10 iterações
    - Para 1000000 elementos $\Rightarrow$ 20 iterações
  - $\log_2 n$ cresce muito devagar com o aumento de $n$
    - Desempenho muito superior em relação ao algoritmo da pesquisa seqüencial

# Pesquisa Binária - Implementação

- Exercício:
  - Implemente uma versão recursiva da função pesquisa binária.
  - Protótipo da função: *int pesqBinRec(int chave, int v[], int ini, int fim);*

# Pesquisa Binária - Implementação Recursiva

```
int pesqBinRec(int chave, int v[], int ini, int
fim) {
     int meio = (ini + fim) / 2;
     if ( ini > fim )
          return -1;
     if (chave == v[meio])
          return meio;
     else if (chave < v[meio])
      return pesqBinRec(chave, v, ini, meio - 1);
     else
      return pesqBinRec(chave, v, meio + 1, fim);
}
```

UNIVERSIDADE CATÓLICA DE PERNAMBUCO

# Pesquisa por Interpolação

# Pesquisa por Interpolação

- Se as chaves estiverem uniformemente distribuídas dentro da lista, a **pesquisa por interpolação** pode ser ainda mais eficiente do que a binária.

- O algoritmo é o mesmo da *pesquisa binária*, adotando-se a fórmula abaixo para o cálculo da variável "*meio*" (que neste caso não será obrigatoriamente o meio da tabela):
  - meio= ini + ((fim-ini)*(chave-v[ini])) / (v[fim]-v[ini]);

# Pesquisa Binária - Implementação

```
int pesqInter(int chave, int v[], int n) {
    int ini = 0;
    int meio;
    int fim = n - 1;
    while (ini <= fim) {

      meio = ini + ((fim-ini)*(chave-v[ini])) / (v[fim]-v[ini]);

      printf("\n O indice do meio foi: %i", meio);
      if (chave < v[meio]) {
          fim = meio - 1;
      } else if (chave > v[meio]) {
          ini = meio + 1;
      } else {
          return meio;
      }
    }
    return -1; // Índice Impossível
}
```

# Pesquisa por Interpolação

- Análise
  - Se as chaves estiverem uniformemente esse método exigirá $\log_2(\log_2 n)$ comparações
  - Entretanto, se as chaves não estiverem uniformemente distribuídas, o método degrada sua eficiência e torna-se ruim
    - No pior caso se compara com a busca seqüencial.
  - Em situações práticas as chaves tendem a se aglomerar em torno de determinados valores e não são uniformemente distribuídas
    - Por exemplo, agenda telefônica.

# Exercício

1) Apresente um exemplo prático em que a busca por interpolação terá um melhor desempenho que a busca binária

# Exercícios

1) Modifique a função *insertionSort* apresentada no curso para que a inserção dos elementos não-ordenados no vetor ordenado seja feita usando pesquisa binária. Chame esta nova função de *insertionSortPB*. Protótipo da função: *void insertionSortPB(int v[], int n);*

2) Mostre, através de um exemplo, um caso em que *insertioSortPB* funciona melhor que *insertionSort*.