Pesquisa em Memória Primária

Algoritmos e Estruturas de Dados II

Pesquisa em Memória Primária

- Pesquisa:
  - Recuperação de informação em um grande volume de dados.
  - Informação é dividida em registros e cada registro contém uma chave.
- Objetivo:
  - Encontrar itens com chaves iguais a chave dada na pesquisa.
- Aplicações:
  - Contas em um banco
  - Reservas de uma companhia aérea

Pesquisa em Memória Primária

- Escolha do método de busca
  - Quantidade de dados envolvidos.
  - Frequência com que operações de inserção e retirada são efetuadas.

- Métodos de pesquisa:
  - Pesquisa sequencial
  - Pesquisa binária
  - Árvore de pesquisa
    - Árvores binárias de pesquisa sem balanceamento
    - Árvores binárias de pesquisa com balanceamento
  - Pesquisa digital
  - Hashing

Tabelas de Símbolos

- Estrutura de dados contendo itens com chaves que suportam duas operações
  - Inserção de um novo item
  - Retorno de um item que contém uma determinada chave.

- Tabelas são também conhecidas como **dicionários**
  - Chaves – palavras
  - Item – entradas associadas as palavras (significado, pronúncia)

Tipo Abstrato de Dados

- Considerar os algoritmos de pesquisa como tipos abstratos de dados (TADs), com um conjunto de operações associado a uma estrutura de dados,
  - Há independência de implementação para as operações.

- Operações:
  - Inicializar a estrutura de dados
  - Pesquisar um ou mais registros com uma dada chave
  - Inserir um novo registro
  - Remover um registro específico
  - Ordenar os registros

Pesquisa Sequencial

- Método de pesquisa mais simples
  - A partir do primeiro registro, pesquisa sequencialmente até encontrar a chave procurada
- Registros ficam armazenados em um vetor (arranjo).
- Inserção de um novo item
  - Adiciona no final do vetor.
- Remoção de um item com chave específica
  - Localiza o elemento, remove-o e coloca o último item do vetor em seu lugar.

Pesquisa Sequencial

```
# define MAX              10
typedef int TipoChave;

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

typedef int Indice;

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

Pesquisa Sequencial

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


/* retorna 0 se não encontrar um registro com a chave x */
Indice Pesquisa(TipoChave x, Tabela *T){
int i;

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

Pesquisa Sequencial

```
void Insere(Registro Reg, Tabela *T) {

    if (T->n == MAX)
       printf("Erro : tabela cheia\n");
    else {
       T->n++;
       T->Item[T->n] = Reg;
    }
}

void Remove(TipoChave x, Tabela *T) {
Int idx;
    idx = Pesquisa(x, T);

    /* se encontrou o item, troca pelo último, reduz o n */
    if (idx) T->Item[idx] = T->Item[T->n--];
}
```

Pesquisa Sequencial

- 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

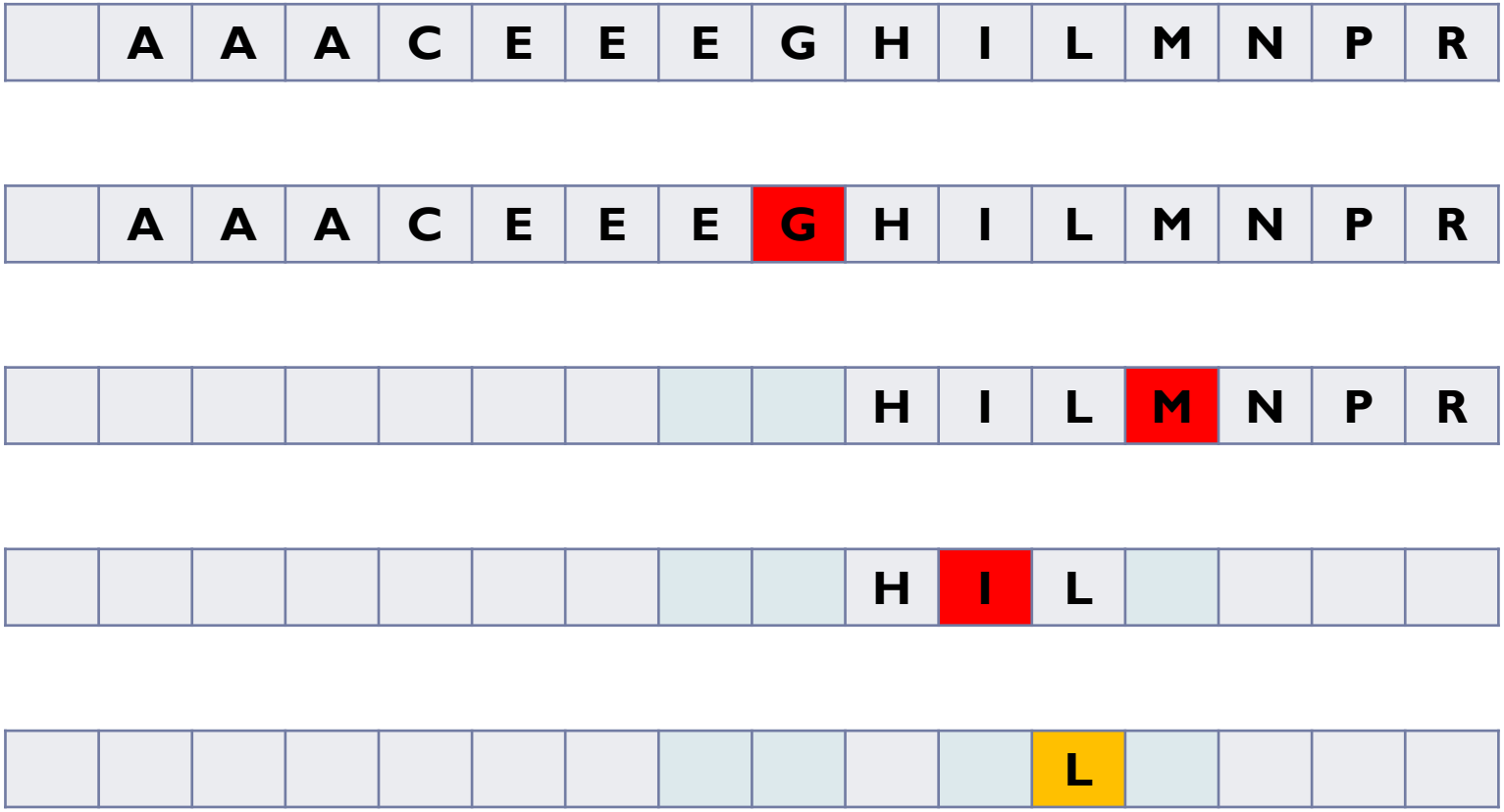
Pesquisa Binária

- Redução do tempo de busca aplicando o paradigma dividir para conquistar.
  1. Divide o vetor em duas partes
  2. Verifica em qual das partes o item com a chave se localiza
  3. Concentra-se apenas naquela parte

- Restrição: chaves precisam estar ordenadas
  - Manter chaves ordenadas na inserção pode levar a comportamento quadrático.
  - Se chaves estiverem disponíveis no início, um método de ordenação rápido pode ser usado.
  - Trocas de posições podem reduzir a eficiência.

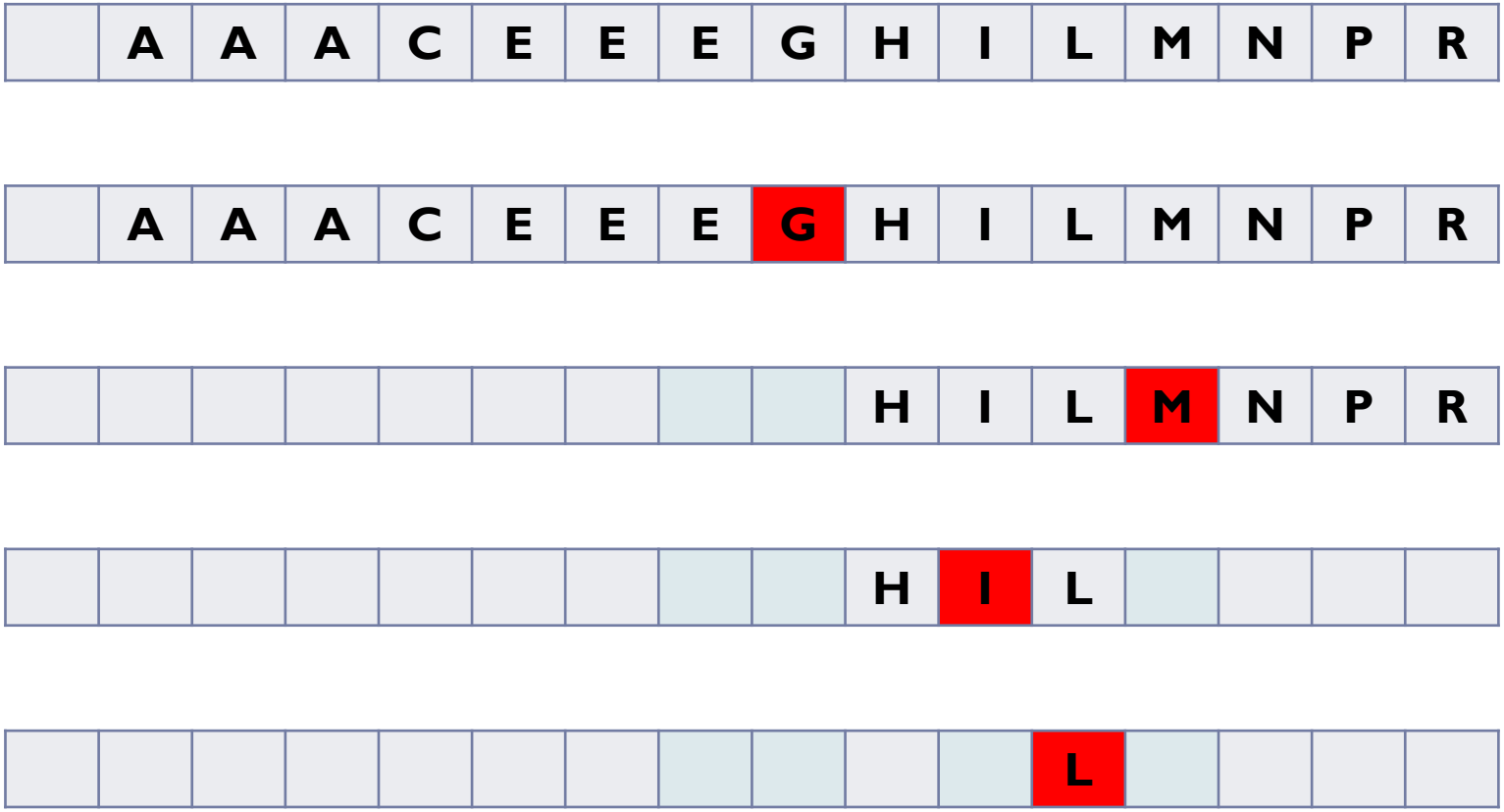
Pesquisa Binária

- Exemplo: pesquisa pela chave L



Pesquisa Binária

- Exemplo: pesquisa pela chave J



Pesquisa Binária

```
Indice Binaria(TipoChave x, Tabela *T) {
Indice i, Esq, Dir;

if (T->n == 0) return 0; /* vetor vazio */

Esq = 1;
Dir = T->n;
do {
   i = (Esq + Dir) / 2;
   if (x > T->Item[i].Chave)
      Esq = i + 1; /* procura na partição direita */
   else
      Dir = i - 1; /* procura na partição esquerda */
}
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
  - Alto custo para manter a tabela ordenada: a cada inserção na posição p da tabela implica no deslocamento dos registros a partir da posição p para as posições seguintes.
  - Portanto, a pesquisa binária não deve ser usada em aplicações muito dinâmicas.