IMD0029 – ESTRUTURAS DE DADOS BÁSICAS I

PROF. EIJI ADACHI M. BARBOSA

Lista de Exercícios – Tabela de Dispersão

Obs.: Além das questões nesta lista, vejam também as questões das provas passadas.

Questão 01: Você foi designado a implementar uma tabela de dispersão de uma forma bem específica. Em caso de colisões, a sua implementação deverá dar prioridade aos itens mais novos. Isto é, se durante a inserção de um novo par (key, value) ocorrer uma colisão num determinado índice X, o novo par ocupará a posição X e o par que estava anteriormente na posição X deverá ser deslocado para uma nova posição. Nesta questão, é necessário implementar apenas a função de inserir seguindo a seguinte assinatura:

```
bool HashTable::Put( Key, Value );
```

Nesta implementação, quando ocorrer uma colisão, busque por uma nova posição usando a sondagem linear vista em sala de aula.

Questão 02: Considere que você tem à sua disposição uma tabela de dispersão que já oferece as seguintes funções implementadas corretamente:

```
HashTable::HashTable( size ); // Construtor
bool HashTable::Put( key, size ); // Inserir
Value HashTable::Remove( key ); // Remover
Value HashTable::Get( key ); // Buscar
```

Use esta implementação da tabela de dispersão para resolver o seguinte problema: dada uma sequência de palavras, conte quantas vezes cada palavra ocorre nesta sequência. Considere que nesta sequência de palavras pode ocorrer repetição de palavras, como num texto normal. Assuma que esta sequência de palavras é um array que pode ser manipulado como mostrado abaixo:

```
palavras[0...N-1]
PARA i DE 0 ATÉ N-1 FAÇA:
      palavra = palavras[i]
      // CONTAR PALAVRA
FIM_PARA
```

Dica: Use as operações da tabela de dispersão considerando que a própria palavra é a chave da tabela de dispersão e o item é o número de vezes que esta palavra ocorreu até um determinado momento.

Obs.: Considere que quando a função Put recebe uma chave que já existe na tabela, o valor novo sobrescreve o antigo. Ou seja, é como se o valor fosse atualizado.

Questão 03: Considere novamente que você tem a sua disposição uma tabela de dispersão que já oferece as funções mostradas na questão anterior. Usando estas funções, resolva o seguinte problema: dada uma sequência de palavras armazenadas num array, registre as posições em que cada palavra ocorre neste array. Considere que nesta sequência pode ocorrer repetição de palavras, como num texto normal. Assuma que esta sequência de palavras também é um array que pode ser manipulado como mostrado na questão anterior.
Dica: Use as operações da tabela de dispersão considerando que a própria palavra é a chave da tabela de dispersão e o item é uma lista de inteiros que representam as posições em que cada palavra ocorreu até um determinado momento.

Questão 04: Suponha que alguém implementou uma tabela de dispersão com redimensionamento dinâmico. Nesta implementação, todas as vezes que o fator de carga fica maior do que 0.5, a tabela é expandida. Similarmente, todas as vezes que o fator de carga fica menor do que 0.25, a tabela é reduzida. Após uma bateria de testes, verificou-se que esta implementação não é muito eficiente. Explique e justifique por que esta implementação não é eficiente. Sugira uma melhoria.
Dica: considere um caso em que a tabela está com uma quantidade de itens exatamente menor do que o limite do fator de carga (ex: tabela com tamanho 17 e com 8 elementos), isto é, já na próxima inserção será realizada uma expansão da tabela. Neste estado da tabela, qual seria o comportamento da tabela caso fossem realizadas operações: inserir, remover, inserir, remover, inserir, remover…?

Questão 05: Um anagrama é um jogo de palavras em que tentamos criar outras palavras por meio do rearranjo das letras de uma determinada palavra base. As novas palavras criadas devem empregar todas as letras originais da palavra base exatamente uma vez. Um exemplo bastante conhecido de anagrama é o nome da personagem Iracema, do escritor José de Alencar, que é um anagrama da palavra America. Nesta questão, você deverá implementar um programa que recebe como entrada uma palavra S e retorna uma lista de palavras {p1, p2, p3, ...pn} em que cada palavra pi é um anagrama de S.

Para construir tal solução, siga a seguinte abordagem. Inicialmente, construa um dicionário de anagramas. Para implementar tal dicionário, precisamos primeiro "aprender" palavras de um determinado idioma. Isto pode ser feito passando para o seu programa arquivos contendo textos comuns, como notícias e/ou livros. O seu programa deverá processar tais arquivos e construir uma tabela de dispersão com chave do tipo String e valor do tipo List<String>. Para cada palavra encontrada no arquivo texto (ex.: America) o seu programa deverá primeiro passar todos os caracteres desta palavra para caixa alta (ex.: AMERICA) e, em seguida, ordenar os caracteres de tal palavra (ex.:AACEIMR). Perceba que palavras diferentes e que são anagramas entre si possuem a mesma cadeia de caracteres ordenados (ex.: AMERICA e IRACEMA tem cadeia de caracteres ordenados AACEIMR). Esta string contendo os caracteres da palavra base será usada como a chave da tabela de dispersão e a palavra base é inserida na lista associada a esta chave (ex.: <AACEIMR, {AMERICA, IRACEMA}).

Uma vez que terminamos de processar todas as palavras dos arquivos “de aprendizado” teremos à disposição uma tabela de dispersão T que nos ajudará a resolver o problema proposto. Ao recebermos uma palavra de entrada S, basta passarmos S para caixa alta, ordenar os caracteres de S e fazer uma busca em T. A lista retornada por T será uma lista de anagramas de S.