IMD0029 - Estrutura de Dados Básicas 1 – 2017.1 – Prova 03
Prof. Eiji Adachi M. Barbosa

Nome:_____

Matrícula: _____

ANTES DE COMEÇAR A PROVA, leia atentamente as seguintes instruções:

- Esta é uma prova escrita de caráter individual e sem consultas a pessoas ou material (impresso ou eletrônico).
- A prova vale 10,0 pontos e o valor de cada questão é informado no seu enunciado.
- **Preze por respostas legíveis, bem organizadas e simples.**
- As respostas devem ser fornecidas preferencialmente em caneta. Respostas fornecidas a lápis serão aceitas, mas eventuais questionamentos sobre a correção não serão aceitos.
- Celulares e outros dispositivos eletrônicos devem permanecer desligados durante toda a prova.
- Desvios éticos ou de honestidade levarão à anulação da prova do candidato (nota igual a zero).

Questão 1: A tabela hash é uma estrutura de dados bastante adequada para aplicações que requerem o gerenciamento eficiente de uma coleção de itens sem a necessidade de estabelecer uma ordem entre estes itens. Embora no pior caso a busca numa tabela hash possa demorar tanto quanto procurar um elemento numa lista encadeada – o tempo O(n) no pior caso – na prática, a tabela hash funciona muito bem. Sob hipóteses razoáveis, o tempo esperado para as operações de inserir, remover e buscar numa tabela hash é O(1). Nesta questão, considere uma Tabela de Dispersão definida seguindo o método de endereçamento, conforme as estruturas a seguir:

```
HashTable
{
     HashEntry data[];
     int quantity;
     int size;
}
```

```
HashEntry
{
      K key;   // chave de tipo K
      V value; // valor de tipo V
}
```

Neste contexto, faça o que se pede em cada item abaixo:

a) (2,0 pontos) Implemente um método de busca para a tabela de dispersão. Considere que se retorna nulo quando a busca tem insucesso. O seu método deverá ter a seguinte assinatura:

```
// Recebe uma chave (key) e retorna o seu valor V correspondente
V HashTable::get(K key)
```

b) (2,0 pontos) Implemente um método de inserção para uma tabela de dispersão. O seu método deverá ter a seguinte assinatura:

```
// Insere uma chave (key) e um valor V na tabela
// Retorna true, caso a inserção ocorra com sucesso, false caso contrário.
bool HashTable::put(K key, V value)
```

Considere para os itens a) e b) desta questão que a tabela resolve colisões pelo método do *Endereçamento Aberto com* ***Sondagem Quadrática***.

Obs.: Para os itens a) e b) desta questão, você pode assumir que:
- *Um método implementando a função hash já está implementado e que você pode reusá-lo.*
- *Métodos de inserir, remover e buscar na tabela não estão disponíveis para reuso.*
- *Não é necessário preocupar-se em checar se a tabela está cheia ou vazia, tampouco é necessário preocupar-se em checar a necessidade de realizar o redimensionamento da tabela. Ou seja, basta mostrar como buscar e como inserir um elemento na tabela.*

Questão 2: (1,0 ponto) Em tabelas de dispersão implementadas seguindo o endereçamento aberto, todos os elementos estão armazenados no array interno da tabela hash. Ou seja, cada posição do array interno da tabela contém um elemento (um par chave-valor), ou Nulo. Um possível problema com esta representação ocorre durante a remoção de um elemento da tabela. Explique que possível problema é este e como fizemos em sala de aula para contorná-lo.

Questão 3: (3,0 pontos) Em muitas aplicações em que empregamos tabelas hash, não sabemos com antecedência quantos objetos serão armazenados. Sem este conhecimento, podemos alocar espaço e somente mais tarde descobrir que ele é insuficiente. De modo semelhante, podemos descobrir somente mais tarde que o espaço alocado é muito superior ao necessário. Para estes casos, é possível implementar tabelas de dispersão capazes de expandir e contrair dinamicamente o seu tamanho. Em sala de aula, chamamos estas tabelas de "tabelas elásticas", ou ainda de "tabelas com redimensionamento dinâmico". Nesta questão, implemente um método de redimensionamento dinâmico, o qual poderá ser usado tanto para expandir quanto para contrair a tabela hash. Considere que o seu método de redimensionamento dinâmico será aplicado sobre a estrutura de tabela hash definida na Questão 1. Considere ainda que as colisões serão resolvidas por meio do método do *Endereçamento Aberto com* ***Sondagem Linear***. Por fim, o seu método deverá ter a seguinte assinatura:

```
void HashTable::resize(int newSize)
```

Obs.: Para esta questão, você pode assumir que um método implementando a função hash já está implementado e que você pode reusá-lo. Ainda para esta questão, assuma que métodos de inserir, remover e buscar na tabela não estão disponíveis para reuso.

Questão 4: (2,0 pontos) Dicionários são tipos abstratos de dados que modelam coleções de elementos, em que cada elemento corresponde a um par chave (Key) - valor (Value). Tipicamente, dicionários são implementados usando tabelas de dispersão, ou árvores balanceadas. Suponha que você tem à sua disposição um dicionário já implementado corretamente e que provê as seguintes operações:

```
Dictionary:: Dictionary ( ); // Construtor
bool Dictionary::Put( K key, V value ); // Inserir
V Dictionary::Remove( K key ); // Remover
V Dictionary::Get( K key ); // Buscar
```

Nesta questão, reuse este dicionário para resolver o seguinte problema: dado um array de strings de entrada, contendo algumas strings repetidas, monte um dicionário que registre quantas vezes cada palavra ocorre no array de entrada. Sua função deverá ter a seguinte assinatura:

```
Dictionary countWords(string[] input, int inputSize)
```

Obs.: Nesta questão, você deverá apenas reusar o Dicionário; você não deverá tentar de alguma forma modificar a sua implementação. Assuma que cada string no array de entrada possui apenas caracteres alfanuméricos e que todas as letras das strings estão em caixa alta (i.e., estão maiúsculas).