Estruturas Avançadas de Dados II

Trabalho Grau A

Profa. Patrícia A. Jaques Maillard Semestre: 2017/2

Implemente uma tabela Hash (Hastable) utilizando **endereçamento aberto** e **encadeamento separado**. Você deve criar uma classe para cada uma das implementações, que se chamarão ***HashtableOpenAddressing*** e **HashtableSeparateChaining**. Ambas as classes devem implementar a interface **Hashtable** abaixo. Observe que você vai guardar na hashtable objetos Item, que representam pares chave e valor. Segue abaixo o código da interface e da classe Item.

```
public interface Hashtable<K,V> {

    /**
     * Remove o objeto Item de chave key na lista encadeada. Retorna o Item removido.
     * @param key - chave do objeto Item que sera removido da hashtable
     * @return o item que foi deletado ou null caso nao exista um Item com a key
     * recebida como parametro
     */
         public Item<K,V> delete (K key);

    /**
     * Insere um objeto Item na hashtable.
     * @param item - Objeto Item, instanciado previamente (contem uma key (K) e
     * um value (V))
     * @return a posicao do objeto na hashtable
     */
         public int insert (Item<K,V> item);

    /**
     * Busca o objeto Item de chave key na hashtable.
     * @param key - chave do objeto Item que sera pesquisada na hashtable
     * @return um objeto Item que tem a key igual a key recebida como parametro
     * ou null caso nao exista tal objeto na hashtable
     */
         public Item<K,V> search (K key);

    /**
     * Exibe na tela o conteúdo da hashtable no formato (posicao) valor1, valor2, valor3,
     */
         public void print ();


}
```

```
/**
 * Classe Item que representa um item contendo uma chave (key) e um valor associado a esta chave (value).
 * A chave sempre deverá ser um inteiro. Não altere essa classe, apenas utilize ela em sua implementação.
 * @author Felipe de Morais e Patricia Jaques
 */
public class Item<V> {

    /**
     * Variaveis privadas do objeto Item, que armazenam a chave (key) e um valor associado a esta chave
(value)
     */
    private int key;
    private V value;

    /**
     * Metodo construtor da classe Item
     * @param key - chave do tipo K
     * @param value - valor do tipo V
     */
    public Item(int key, V value) {
        this.key = key;
```

```
        this.value = value;
    }

    /**
     * Metodo de acesso da chave
     * @return key do tipo K
     */
    public int getKey() {
        return key;
    }

    /**
     * Metodo de acesso do valor
     * @return value do tipo V
     */
    public V getValue() {
        return value;
    }
}
```

As descrições a seguir, explicam como as classes ***HashtableOpenAddressing*** e **HashtableSeparateChaining** devem ser criadas.

Parte I: Implementando uma Hashtable com Endereçamento Aberto

Você deverá criar a classe ***HashtableOpenAddressing*** que irá implementar a interface *Hashtable*:

```
public class HashtableOpenAddressing<V> implements Hashtable<V> {
```

Você também deverá implementar o método construtor com a seguinte assinatura:

```
        public HashtableOpenAddressing(int m, int q, int probing)
```

Onde:

- m é o tamanho da tabela Hash, que é o tamanho do nosso Array de inteiros.
- q é o valor utilizado pela função de hashing duplo.
- probing é a estratégia de colisão que deverá ser utilizada. Você deverá implementar as 3 estratégias vistas em aula: Teste Linear (*Linear Probing*), Teste Quadrático (*Quadratic Probing*) e Hashing Duplo (*Double Probing*). Para isso adicione as 3 constantes a seguir na sua classe:

  ```
  public static final int LINEAR_PROBING = 0;
  public static final int QUADRATIC_PROBING = 1;
  public static final int DOUBLE_PROBING = 2;
  ```

  Estas constantes deverão ser utilizadas como probing. Ou seja, ao criar um objeto ***HashtableOpenAddressing***, você deverá informar qual teste de colisão (probing) deverá ser utilizado.

Implemente também os métodos *insert, delete*, *search e print.* Você deve seguir o pseudo-código fornecido em aula (slides) para realizar a implementação de cada um dos métodos.

Após implementar a classe HashtableOpenAddressing, crie uma classe de teste chamada OpenAdressingMain. Nesta classe, crie 3 objetos HashtableOpenAddressing, ambos com m = 11 e q = 7. Cada um deles deve implementar uma estratégia diferente de colisão.

```
hash1 = new HashtableOpenAddressing(11, 7, Hashtable.LINEAR_PROBING);
hash2 = new HashtableOpenAddressing(11, 7, Hashtable.QUADRATIC_PROBING);
hash3 = new HashtableOpenAddressing(11, 7, Hashtable.DOUBLE_PROBING);
```

Chaves = {7, 17, 36, 100, 106, 205}

Adicione estas chaves em cada um dos hashs e faça testes de pesquisa, deleção e novas inserções. Exiba o conteúdo dessas hashtable com o método print.

Parte II: Implementando uma hashtable com encadeamento separado

Implemente uma tabela Hash (Hastable) utilizando encadeamento separado (Separate Chaining). Lembrando que no encadeamento separado, temos um array, que representa a hashtable, onde cada posição desse array armazena uma lista encadeada.

Você deverá criar a classe ***HashtableSeparateChaining*** que irá implementar a interface *Hashtable*:

```
public class HashtableSeparateChaining<V> implements Hashtable<V>{
```

Você também deverá implementar o método construtor com a seguinte assinatura:

```
    public HashtableSeparateChaining(int m) {
```

Onde, m é o tamanho da tabela hash.

Se desejar, você pode usar a classe LinkedList da API do Java como implementação de lista encadeada.

Implemente também os métodos *insert, delete*, *search e print.* Você deve seguir o pseudo-código fornecido em aula (slides) para realizar a implementação de cada um dos métodos.

Crie uma classe de teste (chamada SeparateChainingMain) com um método main, onde você instancie a classe ***HashtableSeparateChaining***. Adicione as chaves {7, 17, 36, 100, 106, 205} nesse hash e faça testes de pesquisa, deleção e novas inserções. Exiba o conteúdo dessas hashtable com o método print.

Parte III: Resolvendo o problema de achar chaves duplicadas

Você também deverá implementar o problema de eliminar chaves duplicadas. Crie uma nova classe com um método que recebe um array de inteiros como parâmetro. Retorne uma nova array sem as chaves duplicadas na array original. Implemente uma solução com complexidade O(n), usando uma das implementações de hashtable que você fez.

Implemente um método main em uma classe de teste chamada Main. Chame o método criado passando como parâmetro a array a seguir:

{1, 1, 2, 3, 4, 5, 1, 6, 7, 6, 1, 5, 8, 9, 2, 10, 7, 6}

Exiba o conteúdo do array retornado.

Enviar a solução pelo Moodle (http://www.moodle.unisinos.br).

Data final de entrega: (**06/10/2017**).

Data de apresentação: (**13/10/2017**).

O código deve estar dentro de um pacote Java chamado "*NomeSobrenomeTrabGA*". Compactar apenas os arquivos *.java.

ATENÇÃO:

1) Uma vez entregue o trabalho, ao aluno está optando pelo cálculo abaixo da nota do Grau A, independentemente da nota do trabalho (mesmo que tire zero no trabalho): **Nota do Grau = 0,2*Teste + 0,5*Prova + 0,3*Trabalho**

2) **Trabalhos plagiados** (cópia de colegas, da Internet, de colegas de semestres anteriores, etc) tem sua nota automaticamente zerada, mesmo que apenas parte do trabalho tenha sido plagiada. Além disso, no caso de trabalhos iguais na turma, as notas de todos os alunos envolvidos serão zeradas, não importa quem tenha copiado. Além disso, todos os casos de plágio são avisados ao coordenador do curso dos alunos envolvidos.

3) Todos os **trabalhos deverão ser apresentados em aula**. Trabalhos não apresentados não receberão notas.

4) Trabalhos entregues atrasados serão descontados 0,5 pontos por dia.

Não se esqueça de colocar comentários sobre sua lógica. Também não esqueça de adicionar seu nome.