PUC Minas

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Curso : Engenharia de *Software*
Disciplina : Algoritmos e Estruturas de Dados II
Professora : Eveline Alonso Veloso

Exercícios sobre Tabelas *hash*

1) Dada uma tabela *hash* cuja função de transformação é representada por `h(x) = x % 11`, faça a inserção dos registros com chaves: 35, 49, 11, 18, 13, 29, 32 e 90, nessa ordem. Utilize, para tratamento de possíveis colisões, listas encadeadas.

2) Substitua XXXXXXXXXXXX pelas 12 primeiras letras de seu nome, desprezando brancos e letras repetidas, para resolver esta questão. Se você não tiver 12 letras diferentes em seu nome, complete com as letras PQRSTUVWXYZ, nessa ordem.

   Considere também uma tabela de espalhamento inicialmente vazia de tamanho 13.

   Desenhe o conteúdo da tabela resultante da inserção de registros com as chaves XXXXXXXXXXXX, nessa ordem, usando:

   a) A função de transformação `h(chave) = chave % 13`. Para resolver possíveis colisões, deve ser utilizado o método de *hash* direta com área de reserva de tamanho 5.

   b) A função de transformação `h(chave) = (chave % 13 + k * (chave % 11)) % 13`, onde `k` indica o número de tentativas de se inserir a chave na tabela, começando com o valor 0. Dessa forma, para resolver possíveis colisões, deve-se empregar o método de endereçamento aberto, com *rehash*.

   c) Listas encadeadas para resolver as colisões. Use a função de transformação `h(chave) = chave % 13`.

   Considere que o número correspondente a uma letra é dado pela posição dessa letra no alfabeto.

3) Implemente a classe `TabelaHash<K, V>` correspondente à implementação de uma tabela *hash* direta com área de reserva. Devem ser codificados, pelo menos, os métodos de inserção, pesquisa e remoção de pares chave-valor da tabela *hash*.

4) Implemente o método `private void restaurar()` na classe `TabelaHash<K, V>` correspondente à implementação de uma tabela *hash* direta com *rehash*. Esse método deve ser capaz de reorganizar a tabela *hash* removendo pares chave-valor assinalados como removidos e reposicionando as entradas restantes, se for o caso.

Os exercícios de 5 a 11 devem ser resolvidos na classe `TabelaHash<K, V>`, que corresponde a uma tabela *hash* que trata possíveis colisões por meio de listas encadeadas simples, implementada durante as aulas da disciplina.

5) Implemente o método `public boolean vazia()`, que deve retornar `verdadeiro` se a tabela *hash* estiver vazia, ou seja, não estiver armazenando nenhum mapeamento chave-valor; e `falso`, no caso contrário.

6) Implemente o método `public int obterTamanho()`, que retorna o número de mapeamentos chave-valor presentes na tabela *hash*. Realize as modificações necessárias na classe para a correta execução desse método.

7) Implemente o método `public boolean contemValor(V valor)`, que deve retornar `verdadeiro` se a tabela *hash* estiver mapeando uma ou mais chaves para o valor especificado como parâmetro para esse método; e `falso`, no caso contrário. Implemente também os métodos que julgar necessários na classe `Lista<E>`.

8) Implemente o método `public Lista<Entrada<K, V>> obterTodasEntradas()`, capaz de retornar uma lista encadeada simples com todos os mapeamentos chave-valor contidos na tabela *hash*. Implemente também os métodos que julgar necessários na classe `Lista<E>`, assim como a classe `Entrada<K, V>`.

9) Implemente o método `public Lista<K> obterTodasChaves()`, responsável por retornar uma lista encadeada simples com todas as chaves presentes na tabela *hash*. Implemente também os métodos que julgar necessários na classe `Lista<E>`.

10) Implemente o método `public Lista<V> obterTodosValores()`, responsável por retornar uma lista encadeada simples com todos os valores presentes na tabela *hash*. Implemente também os métodos que julgar necessários na classe `Lista<E>`.

11) Implemente o método `public void substituir(K chave, V novoValor)`, que substitui o valor associado à `chave` passada como parâmetro para o valor especificado no parâmetro `novoValor`.