Gebze Technical University

CSE 222 – Homework 6
High-Performance Spell Checker with Custom HashMap and Set

Due Date: 11th May 2025

In this assignment, you will implement your own high-performance versions of HashMap<K, V> and HashSet<E> using open addressing, and then use them to build a real-world application: a spell checker capable of offering suggestions based on edit distance. No Java collection libraries are allowed.

1 Introduction

1.1. Forbidden Libraries

-java.util.HashMap, java.util.HashSet, java.util.Map, java.util.Set java.util.List, java.util.ArrayList, or any java.util.*

-Any third-party data structure library

-Use only your own object-oriented programming-based data structures and classes. No Java collections are allowed. You must define custom classes (e.g., Entry<K, V>, etc.) to support the structure of your map and set.

1.2. Dictionary

You will use the dictionary.txt provided within assignment.

2 Assignment Details

2.1. GTUHashMap<K, V>

First, you will implement a fully functional hash map from scratch using open addressing with linear or quadratic probing. Your implementation must support these:

-put(K key, V value)
-get(K key)
-remove(K key)
-containsKey(K key)
-size()

2.1.1. Requirements for 2.1:

-Must use open addressing (not chaining)
-Must handle collisions using probing
-Must handle deletion via tombstones (special marker indicating a removed slot)

-**Bonus**: Rehash to a larger capacity (next prime capacity is recommended) Next prime capacity means the new table size should be the next prime number greater than twice the current capacity. This helps reduce clustering.

2.2 GTUHashSet<E>

You will build a set that internally uses your GTUHashMap<E, Object> to store elements.

When you add an element to GTUHashSet, you call map.put(key, value) inside. This reuses the efficient hashing already built for the map.

- add(E element)
- remove(E element)
-contains(E element)

-size()

2.3 Spell Checker

Using your custom GTUHashSet<String>, implement a spell checker that:

-Loads a dictionary file (50,000+ words)
-Accepts user input word(s)
-If a word is misspelled:
    -Suggests valid alternatives within edit distance <= 2 (insertion, deletion, substitution)

-Must work with your custom data structures (not Java libraries)

2.3.1 Performance Requirements:

-You must not scan the entire dictionary for suggestions.

-Instead, for a given input word, generate all edit distance ≤ 2 variants using character-level operations (insert, delete, replace), and test each one using GTUHashSet.contains().

-This limits suggestion generation to ~1,000–10,000 operations, depending on word length.

-Must respond to a word query and generate suggestions within 100ms

-Must use efficient traversal and pruning to avoid exponential edit distance checks

2.4 Java Skeletons and Testing

```
// GTUHashMap.java (only skeleton)
public class GTUHashMap<K, V> {
    private Entry<K, V>[] table;
    private int size;
```

```
    public GTUHashMap() { /* constructor */ }
    public void put(K key, V value) { }  //use linear probing here. Quadratic for the bonus.
    public V get(K key) { return null; }
    public void remove(K key) { }  // mark as deleted (tombstone)
    public boolean containsKey(K key) { return false; }
    public int size() { return 0; }
}
```

```
public class GTUHashSet<E> {
  private static final Object WORD = new Object();
  private GTUHashMap<E, Object> map;

  public GTUHashSet() { map = new GTUHashMap<>(); }
  public void add(E element) { map.put(element, WORD); }
  public void remove(E element) { map.remove(element); }
  public boolean contains(E element) { return map.containsKey(element); }
  public int size() { return map.size(); }
}
```

```
// Entry.java
public class Entry<K, V> {
  public K key;
  public V value;
  public boolean isDeleted;

  public Entry(K key, V value) {
    this.key = key;
    this.value = value;
    this.isDeleted = false;
  }
}
```

3 Bonus Tasks

Bonus 5 points for each

-Implement quadratic probing instead of linear.

-Track number of collisions and memory usage.

-Implement your own simple JUnit-style testing framework for both classes.

-Rehash to a larger capacity

4 Testing

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;


public class SpellChecker {
   public static void main(String[] args) throws IOException {
      GTUHashSet<String> dictionary = new GTUHashSet<>();

      BufferedReader reader = new BufferedReader(new
FileReader("dictionary.txt"));
      String word;
      while ((word = reader.readLine()) != null) {
         dictionary.add(word.trim());
      }
      reader.close();

      Scanner scanner = new Scanner(System.in);

      while (true) {
         System.out.print("Enter a word: ");
         String input = scanner.nextLine().trim();
```

```
        long startTime = System.nanoTime();

        if (dictionary.contains(input)) {
            System.out.println("Correct.");
        } else {
            System.out.println("Incorrect.");
            System.out.print("Suggestions: ");
            List<String> suggestions = new ArrayList<>(); //this is just an example
you cannot use List library. It must be your own implementation!
            for (String variant : generateEditDistance1(input)) {
                if (dictionary.contains(variant)) {
                    suggestions.add(variant);
                }
            }
            System.out.println(suggestions);
        }

        long endTime = System.nanoTime();
        System.out.printf("Lookup and suggestion took %.2f ms\n", (endTime -
startTime) / 1e6);
    }
  }
}
```

5 Submissions

-GTUHashMap.java
-GTUHashSet.java
-SpellChecker.java
-Entry.java
-dictionary.txt (can be submitted as-is if provided)
-You can use additional java files like EditDistanceHelper.java
-Project report: **Any report not in PDF format will not be accepted!**

6 Grading

-GTUHashMap correctness and open addressing - 30 points
-GTUHashSet built on GTUHashMap - 20 points
-SpellChecker correctness (edit distance logic) – 20 points
-SpellChecker efficiency (meets <100ms rule) – 10 points
-Code clarity and structure– 10 points
-Edge case handling and robustness – 10 points

Bonus: Advanced probing, metrics, tests (5 points for each)

6.1 Penalties:

- -100 No OOP
- -40 for no Report
- -10 points for excessive code duplication or poor abstraction
- -100 Usage of AI
- -200 Cheating.

7 Report Requirements

-Your final report must be a PDF file and **at least 10 pages** in length.

Report Formatting:
-Main headings: 18 or 20 pt and bold
-Subheadings: 16 or 18 pt and bold
-Body text: 12 pt
-Line Spacing: 1.5 lines
-Justification: Fully justified text

-The report **must include**:

-INTRODUCTION (Explain your project)
-METHODOLOGY (Explain the methods that you used)
-RESULTS AND DISCUSSION (Put your results as images and write your comments. Did you get the correct results? Explain if there are unexpected behaviors in your project)
-CONCLUSIONS (Your overall comments about the project)
-REFERENCES (Your references)

Project report writing guideline:

https://www.sjsu.edu/me/docs/projectthesis_presentation-MSME-Report%20Writing%20Guidelines-20221117.pdf

7.1 Penalties:

- -5 points for each missing page under the 10-page minimum.
- -30 points if the report is not submitted as a PDF. (No Word, no txt or no other extension!)
- -10 points for each required section missing (e.g., no test results, no explanation, no analysis)
- -10 for not enough explanation every section.
- -40 for no report.

Be thorough. A well-documented and formatted report significantly improves your final grade and ensures clarity in your implementation.

Good Luck **-Burak Dikmen**