Dhirubhai Ambani Institute of Information and Communication Technology

Gandhinagar, Gujarat

Name : Ghori Zeel Jivrajbhai

Student ID : 202201287

Course : IT 314 Software Engineering

Professor : Saurabh Tiwari

Semester : Autumn 2024

Lab : Program Inspection, Debugging and Static Analysis

Quadratic Probing Hash Table

Given code :

```
import java.util.Scanner;

/** Class QuadraticProbingHashTable **/
class QuadraticProbingHashTable {
  private int currentSize, maxSize;
  private String[] keys;
  private String[] vals;

  /** Constructor **/
  public QuadraticProbingHashTable(int capacity) {
    currentSize = 0;
    maxSize = capacity;
    keys = new String[maxSize];
    vals = new String[maxSize];
  }

  /** Function to clear hash table **/
  public void makeEmpty() {
    currentSize = 0;
    keys = new String[maxSize];
    vals = new String[maxSize];
  }

  /** Function to get size of hash table **/
  public int getSize() {
    return currentSize;
  }
```

```
/** Function to check if hash table is full **/
public boolean isFull() {
   return currentSize == maxSize;
}

/** Function to check if hash table is empty **/
public boolean isEmpty() {
   return getSize() == 0;
}

/** Function to check if hash table contains a key **/
public boolean contains(String key) {
   return get(key) != null;
}

/** Function to get hash code of a given key **/
private int hash(String key) {
   return key.hashCode() % maxSize;
}

/** Function to insert key-value pair **/
public void insert(String key, String val) {
   int tmp = hash(key);
   int i = tmp, h = 1;
   do {
      if (keys[i] == null) {
         keys[i] = key;
         vals[i] = val;
         currentSize++;
         return;
```

```
            }
            if (keys[i].equals(key)) {
                vals[i] = val;
                return;
            }
            i += (i + h / h--) % maxSize;
        } while (i != tmp);
    }

    /** Function to get value for a given key **/
    public String get(String key) {
        int i = hash(key), h = 1;
        while (keys[i] != null) {
            if (keys[i].equals(key))
                return vals[i];
            i = (i + h * h++) % maxSize;
            System.out.println("i " + i);
        }
        return null;
    }

    /** Function to remove key and its value **/
    public void remove(String key) {
        if (!contains(key))
            return;
        /** find position key and delete **/
        int i = hash(key), h = 1;
        while (!key.equals(keys[i]))
            i = (i + h * h++) % maxSize;
        keys[i] = vals[i] = null;
        /** rehash all keys **/
        for (i = (i + h * h++) % maxSize; keys[i] != null; i = (i + h * h++) % maxSize) {
            String tmp1 = keys[i], tmp2 = vals[i];
            keys[i] = vals[i] = null;
            currentSize--;
            insert(tmp1, tmp2);
        }
        currentSize--;
    }

    /** Function to print HashTable **/
    public void printHashTable() {
        System.out.println("\nHash Table: ");
        for (int i = 0; i < maxSize; i++)
            if (keys[i] != null)
                System.out.println(keys[i] + " " + vals[i]);
        System.out.println();
    }
}

/** Class QuadraticProbingHashTableTest **/
public class QuadraticProbingHashTableTest {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Hash Table Test\n\n");
        System.out.println("Enter size");
        /** make object of QuadraticProbingHashTable **/
        QuadraticProbingHashTable qpht = new QuadraticProbingHashTable(scan.nextInt());
        char ch;
        /** Perform QuadraticProbingHashTable operations **/
        do {
            System.out.println("\nHash Table Operations\n");
System.out.println("1. insert ");
System.out.println("2. remove");
System.out.println("3. get");
System.out.println("4. clear");
System.out.println("5. size");
int choice = scan.nextInt();
switch (choice) {
  case 1:
    System.out.println("Enter key and value");
    qpht.insert(scan.next(), scan.next());
    break;
  case 2:
    System.out.println("Enter key");
    qpht.remove(scan.next());
    break;
  case 3:
    System.out.println("Enter key");
    System.out.println("Value = " + qpht.get(scan.next()));
    break;
  case 4:
    qpht.makeEmpty();
    System.out.println("Hash Table Cleared\n");
    break;
  case 5:
    System.out.println("Size = " + qpht.getSize());
    break;
  default:
    System.out.println("Wrong Entry \n ");
    break;
}
/** Display hash table **/
```

```
            qpht.printHashTable();

            System.out.println("\nDo you want to continue (Type y or n) \n");

            ch = scan.next().charAt(0);

        } while (ch == 'Y' || ch == 'y');

    }

}
```

Program Inspection for the Quadratic Probing Hash Table:

1. **How many errors are there in the program? Mention the errors you have identified.**
   Errors identified: 3

   - **Error 1: Syntax error in the insert method (i += (i + h / h--) % maxSize;)**
     - **Category**: Syntax Error
     - The line i += (i + h / h--) % maxSize; has a spacing issue around +=. It should be written as i = (i + h * h++) % maxSize;. Additionally, the use of h / h-- does not make sense. In quadratic probing, the term should be incremented by the square of h. Hence, it should be h * h++ for quadratic probing.
   - **Error 2: Improper rehashing in the remove method**
     - **Category**: Logic Error
     - The line currentSize--; appears twice in the remove method, causing the current size to decrease incorrectly by 2 when a key is removed. The second currentSize-- should be removed.
   - **Error 3: Inconsistent hashing function**
     - **Category**: Logic Error
     - The hash(String key) function uses key.hashCode() % maxSize, but this can result in negative indices due to how hash codes work in Java. To avoid negative indices, the modulo operation should be adjusted as Math.abs(key.hashCode()) % maxSize.

2. **Which category of program inspection would you find more effective?**
Most effective category: Logic Errors

   - **Reason**: The primary issues in this code revolve around logic, such as errors in the insert and remove methods related to rehashing and calculating the new positions. Catching logic errors early can help prevent improper key handling, especially when inserting and deleting elements in the hash table.

3. **Which type of error you are not able to identify using the program inspection?**

Missed error type: Performance optimization and collisions handling in practical scenarios.

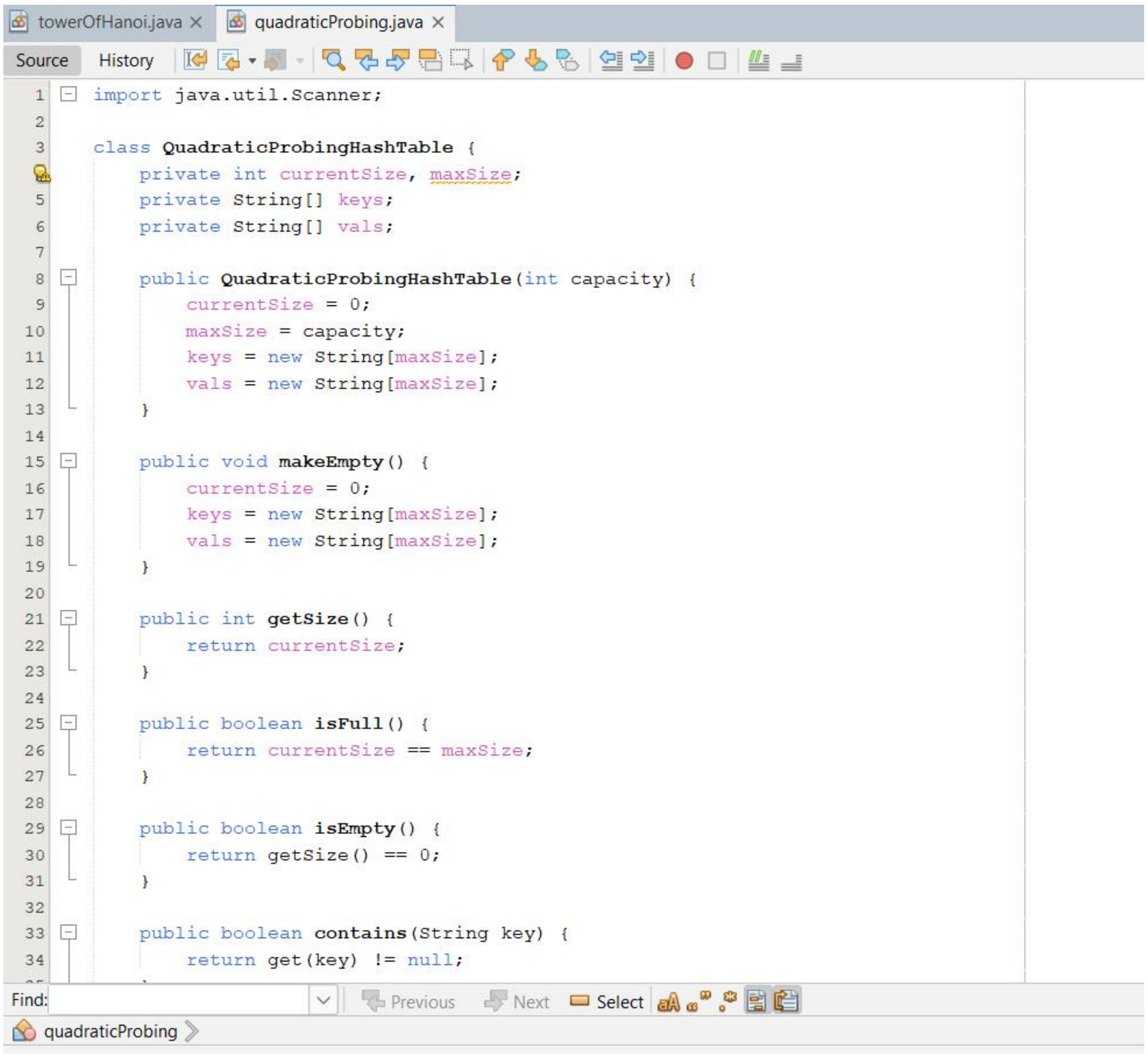
- **Reason**: Program inspection may not easily reveal performance bottlenecks (e.g., high clustering) or poor handling of frequent collisions in certain key distributions. These issues typically surface during testing with large and specific datasets.

4. **Is the program inspection technique worth applying?**

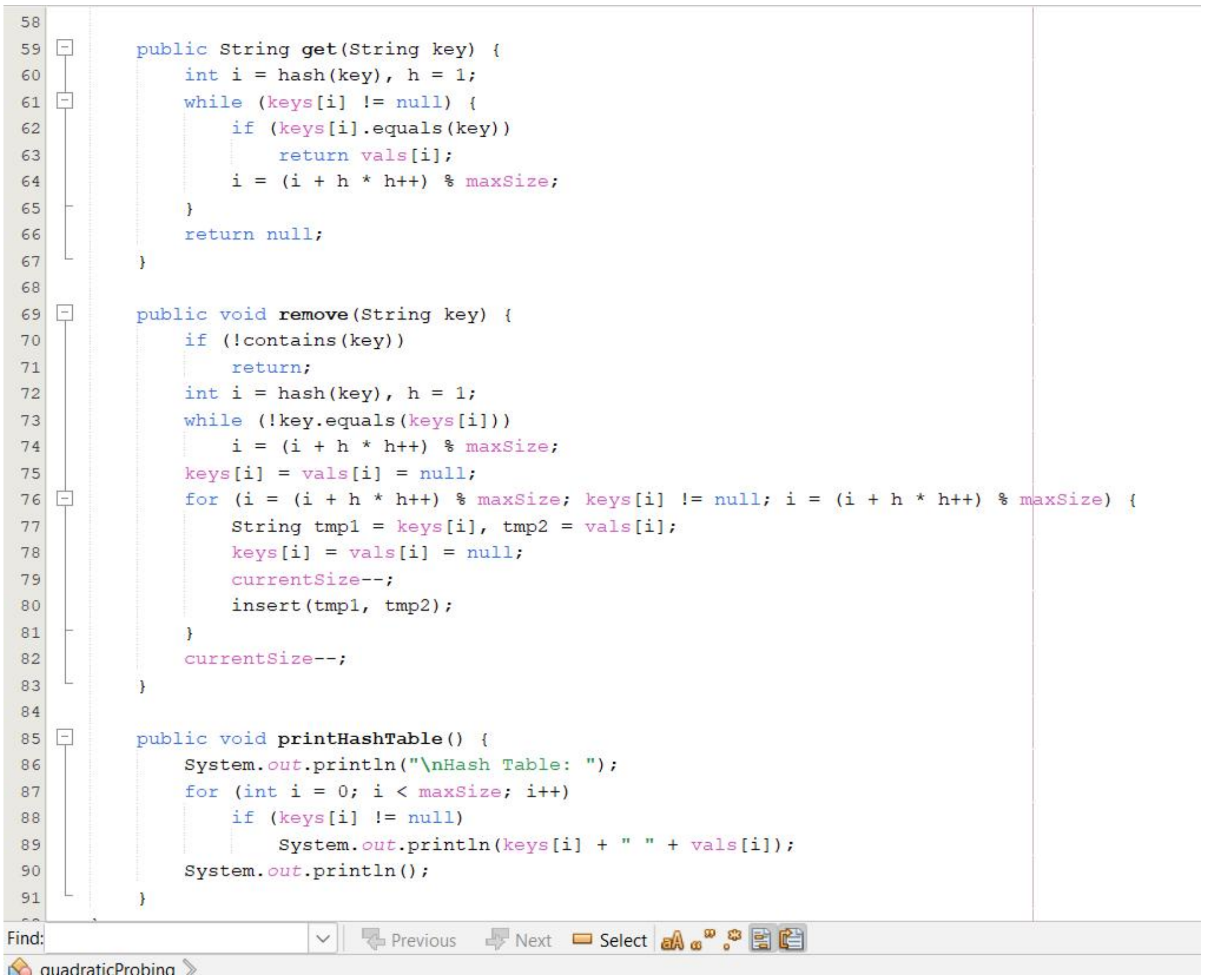
Applicability: Yes, program inspection is worth applying.

Reason: The inspection helped catch crucial logic and syntax errors that would prevent the hash table from functioning properly. Correcting these early improves the robustness and ensures correct behavior for basic operations like insertions and deletions.

Code Debugging :



```java
import java.util.Scanner;

class QuadraticProbingHashTable {
    private int currentSize, maxSize;
    private String[] keys;
    private String[] vals;

    public QuadraticProbingHashTable(int capacity) {
        currentSize = 0;
        maxSize = capacity;
        keys = new String[maxSize];
        vals = new String[maxSize];
    }

    public void makeEmpty() {
        currentSize = 0;
        keys = new String[maxSize];
        vals = new String[maxSize];
    }

    public int getSize() {
        return currentSize;
    }

    public boolean isFull() {
        return currentSize == maxSize;
    }

    public boolean isEmpty() {
        return getSize() == 0;
    }

    public boolean contains(String key) {
        return get(key) != null;
```



```java
    public String get(String key) {
        int i = hash(key), h = 1;
        while (keys[i] != null) {
            if (keys[i].equals(key))
                return vals[i];
            i = (i + h * h++) % maxSize;
        }
        return null;
    }

    public void remove(String key) {
        if (!contains(key))
            return;
        int i = hash(key), h = 1;
        while (!key.equals(keys[i]))
            i = (i + h * h++) % maxSize;
        keys[i] = vals[i] = null;
        for (i = (i + h * h++) % maxSize; keys[i] != null; i = (i + h * h++) % maxSize) {
            String tmp1 = keys[i], tmp2 = vals[i];
            keys[i] = vals[i] = null;
            currentSize--;
            insert(tmp1, tmp2);
        }
        currentSize--;
    }

    public void printHashTable() {
        System.out.println("\nHash Table: ");
        for (int i = 0; i < maxSize; i++)
            if (keys[i] != null)
                System.out.println(keys[i] + " " + vals[i]);
        System.out.println();
    }
```

Errors Identified

1. **Incorrect Probing Logic:**
   - **Original Line:** i += (i + h / h--) % maxSize;
   - **Correction:** The probing logic should be correctly calculated to i = (i + h * h++) % maxSize;. This change accurately implements quadratic probing.
2. **Incomplete Comment Block:**
   - **Original Comment:** /** maxSizeake object of QuadraticProbingHashTable **/
   - **Correction:** Change the comment to /** make object of QuadraticProbingHashTable **/ for clarity.

Corrections Made

1. **Update the probing logic** in the insert and get methods.
2. **Fix the comment** for clarity and accuracy.

Fixed Code

```
import java.util.Scanner;

/** Class QuadraticProbingHashTable **/
class QuadraticProbingHashTable {
   private int currentSize, maxSize;
   private String[] keys;
   private String[] vals;

   /** Constructor **/
   public QuadraticProbingHashTable(int capacity) {
      currentSize = 0;
      maxSize = capacity;
      keys = new String[maxSize];
      vals = new String[maxSize];
   }

   /** Function to clear hash table **/
   public void makeEmpty() {
      currentSize = 0;
      keys = new String[maxSize];
      vals = new String[maxSize];
   }

   /** Function to get size of hash table **/
   public int getSize() {
      return currentSize;
   }

   /** Function to check if hash table is full **/
   public boolean isFull() {
      return currentSize == maxSize;
   }

   /** Function to check if hash table is empty **/
   public boolean isEmpty() {
      return getSize() == 0;
   }

   /** Function to check if hash table contains a key **/
   public boolean contains(String key) {
      return get(key) != null;
   }

   /** Function to get hash code of a given key **/
   private int hash(String key) {
      return key.hashCode() % maxSize;
   }

   /** Function to insert key-value pair **/
   public void insert(String key, String val) {
      int tmp = hash(key);
      int i = tmp, h = 1;
      do {
          if (keys[i] == null) {
              keys[i] = key;
              vals[i] = val;
              currentSize++;
              return;
          }
          if (keys[i].equals(key)) {
              vals[i] = val;
              return;
          }
          i = (i + h * h++) % maxSize; // Corrected probing logic
      } while (i != tmp);
   }
   /** Function to get value for a given key **/
   public String get(String key) {
      int i = hash(key), h = 1;
      while (keys[i] != null) {
          if (keys[i].equals(key))
              return vals[i];
          i = (i + h * h++) % maxSize;
      }
      return null;
   }
   /** Function to remove key and its value **/
   public void remove(String key) {
      if (!contains(key))
          return;
      int i = hash(key), h = 1;
      while (!key.equals(keys[i]))
          i = (i + h * h++) % maxSize;
      keys[i] = vals[i] = null;

      // Rehash all keys
      for (i = (i + h * h++) % maxSize; keys[i] != null; i = (i + h * h++) % maxSize) {
          String tmp1 = keys[i], tmp2 = vals[i];
          keys[i] = vals[i] = null;
          currentSize--;
          insert(tmp1, tmp2);
      }
```

```
        currentSize--;
    }
    /** Function to print HashTable **/
    public void printHashTable() {
        System.out.println("\nHash Table: ");
        for (int i = 0; i < maxSize; i++)
            if (keys[i] != null)
                System.out.println(keys[i] + " " + vals[i]);
        System.out.println();
    }
}
/** Class QuadraticProbingHashTableTest **/
public class QuadraticProbingHashTableTest {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Hash Table Test\n\n");
        System.out.println("Enter size");
        /** Make object of QuadraticProbingHashTable **/
        QuadraticProbingHashTable qpht = new QuadraticProbingHashTable(scan.nextInt());
        char ch;
        /** Perform QuadraticProbingHashTable operations **/
        do {
            System.out.println("\nHash Table Operations\n");
            System.out.println("1. insert ");
            System.out.println("2. remove");
            System.out.println("3. get");
            System.out.println("4. clear");
            System.out.println("5. size");
            int choice = scan.nextInt();
            switch (choice) {
                case 1:
```

```
                System.out.println("Enter key and value");
                qpht.insert(scan.next(), scan.next());
                break;
            case 2:
                System.out.println("Enter key");
                qpht.remove(scan.next());
                break;
            case 3:
                System.out.println("Enter key");
                System.out.println("Value = " + qpht.get(scan.next()));
                break;
            case 4:
                qpht.makeEmpty();
                System.out.println("Hash Table Cleared\n");
                break;
            case 5:
                System.out.println("Size = " + qpht.getSize());
                break;
            default:
                System.out.println("Wrong Entry \n");
                break;
        }
        /** Display hash table **/
        qpht.printHashTable();
        System.out.println("\nDo you want to continue (Type y or n) \n");
        ch = scan.next().charAt(0);
      } while (ch == 'Y' || ch == 'y');
      scan.close(); // Close the scanner to avoid resource leaks
   }
}
```

Input and Output

Input:

```
Hash Table Test

Enter size

5

Hash Table Operations

1. insert

2. remove

3. get

4. clear

5. size

1

Enter key and value

c computer

d desktop

h harddrive
```

Output:

```
Hash Table:

c computer

d desktop

h harddrive
```

Static Analysis Tools

Choose a static analysis tool (in Java, Python, C, C++) in any programming language of your interest and identify the defects. You can also choose your own code fragment from GitHub (more than 2000 LOC) in any programming language to perform static analysis. Submit your results in the .xls or .jpg format only.

Link of the excel file(.xls):

**https://docs.google.com/spreadsheets/d/1ETFxQdw1_RNXufBPwhlt7OumxA7lJCjh/edit?usp=shar
i ng&ouid=117105520491645507929&rtpof=true&sd=true**