**REPORT LAB 7**

**HashDoubleApp**

**1. Class and Constructor**

* **HashDoubleApp Class**: Manages the hash table and related operations.
* **Constructor**: Initializes the hash table with a specified size, sets the number of elements and total probe length to zero.

**2. Hash Functions**

* **hash1 Method**: Primary hash function, calculates **key % tableSize**.
* **hash2 Method**: Secondary hash function for step size, calculates **1 + (key % (tableSize - 1))**.

**3. Insert Method**

* **insert Method**: Inserts a key into the hash table.
  + Calculates the initial hash positions using **hash1** and **hash2**.
  + If the initial position is occupied, it probes the next position using the step size from **hash2**.
  + Continues probing until an empty slot is found.
  + Records the number of steps (probe length) taken for each insertion.

**4. Find Method**

* **find Method**: Searches for a key in the hash table.
  + Similar probing mechanism as **insert**.
  + Reports the number of steps taken to find the key or confirms if the key is not found.

**5. Display and Statistics**

* **displayTable Method**: Prints the current state of the hash table.
* **getAverageProbeLength Method**: Calculates and returns the average probe length for all insertions.

**6. Simulation Methods**

* **simulateLoadFactors Method**: Tests different load factors by varying table sizes and number of keys inserted.
* **demonstratePrimeVsNonPrime Method**: Compares performance between prime and non-prime table sizes.

**Initial Insertions and Table State**

1. **Inserting Keys**: Inserts a predefined set of keys into the hash table using double hashing.
   * **Probe Output**: For each insertion, the program prints the key, step count, initial hash position, step size, and final insertion index.
   * **Table Output**: Displays the final state of the hash table after all insertions.

**Finding Keys**

1. **Finding Existing Key (23)**: Probes until the key is found or determines it's not present.
   * **Expected Result**: Should print the steps taken and the index where the key is found.
2. **Finding Non-Existing Key (50)**: Probes until it confirms the key is not in the table.
   * **Expected Result**: Should print the steps taken and indicate the key is not found.

**Average Probe Length**

1. **Calculation**: Computes the average probe length for all insertions.
   * **Expected Result**: Prints the average probe length, indicating the efficiency of the hash table.

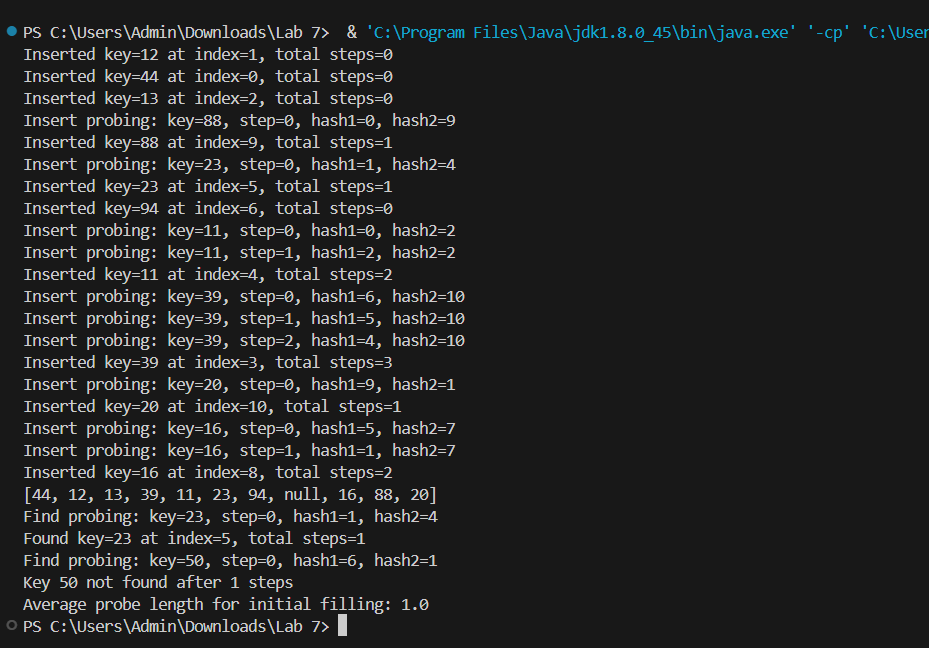
**Load Factor Simulation**

1. **Simulated Load Factors**: Tests how different table sizes and load factors affect the average probe length.
   * **Expected Result**: Prints the hash table state and average probe length for each configuration, demonstrating how higher load factors generally increase probe lengths.

**Prime vs Non-Prime Table Size**

1. **Prime Table Size (11)**: Inserts keys into a table of prime size.
   * **Expected Result**: Shows efficient probe lengths and distribution of keys.
2. **Non-Prime Table Size (10)**: Inserts keys into a table of non-prime size.
   * **Expected Result**: Typically shows less efficient probe lengths and potentially more clustering.

int[] keys = {12, 44, 13, 88, 23, 94, 11, 39, 20, 16};

The output demonstrates the insert and find operations, probe lengths for each operation, and how load factors and table sizes (prime vs non-prime) affect performance.  


**HashChainApp**

1. **Class and Constructor**: The **HashChainApp** class manages the hash table using separate chaining. The constructor initializes the hash table with linked lists.
2. **Hash Function**: The **hash** method calculates the hash index for a given key.
3. **Insert Method**: Inserts a key into the hash table and appends it to the corresponding linked list (chain). It also updates the total probe length, considering the current chain size.
4. **Find Method**: Searches for a key in the hash table by traversing the linked list at the calculated hash index. It reports the steps taken to find the key.
5. **Display Table Method**: Prints the current state of the hash table, showing all keys in each chain.
6. **Get Average Probe Length Method**: Calculates and returns the average probe length for all insertions.
7. **Simulate Load Factors Method**: Tests different load factors by creating hash tables of various sizes and inserting a different number of elements to see how the average probe length is affected.
8. **Main Method**: Demonstrates the insertion, finding, and average probe length calculation with sample data.

**Initial Insertions and Table State**

1. **Inserting Keys**: Inserts a predefined set of keys into the hash table using separate chaining.
   * **Output**: For each insertion, the program prints the key, the index where it is inserted, and the current chain size at that index.
   * **Table Output**: Displays the final state of the hash table after all insertions.

**Finding Keys**

1. **Finding Existing Key (23)**: Searches for the key and reports the index and position in the chain.
   * **Expected Result**: Should print the index and chain position where the key is found.
2. **Finding Non-Existing Key (50)**: Searches for the key and confirms if it is not present.
   * **Expected Result**: Should print the steps taken and indicate that the key is not found.

**Average Probe Length**

1. **Calculation**: Computes the average probe length for all insertions.
   * **Expected Result**: Prints the average probe length, indicating the efficiency of the hash table.

**Load Factor Simulation**

1. **Simulated Load Factors**: Tests how different table sizes and load factors affect the average probe length.
   * **Output Result**: Prints the hash table state and average probe length for each configuration, demonstrating how higher load factors generally increase probe lengths.

int[] keys = {12, 44, 13, 88, 23, 94, 11, 39, 20, 16};

int tableSize = 11;

