CS 421 - Project 2 Analysis

Cy Dixon, Kelcee Gabbard

November 15, 2024

1) The Solution

This project implements a phone number word representation tool by mapping words from a dictionary file into hash tables. The hash tables are used to efficiently store and retrieve words based on their numeric phone number equivalents.

The solution ensures that phone numbers can be broken into their area code, exchange, and number components to find matching word representations.

2) List of Data Structures

- **HashTable:** Used to store words based on their numeric equivalents. Each hash table uses separate chaining to resolve collisions.
- **ListNode:** Represents nodes in the linked lists for separate chaining in the hash table.
- **Keypad Mapping:** A dictionary mapping phone keypad digits to letters (e.g., '2' maps to 'ABC').

3) Complexity Analysis

Loading Words into Hash Tables

- **Word-to-Number Conversion:** $O(L)$, where L is the word length.
- **Insertion into Hash Table:**
  - Best Case: $O(1)$ (no collisions).
  - Worst Case: $O(n)$, where n is the number of collisions in a hash slot.
- **Overall Complexity:** $O(W \cdot (L + n))$, where W is the total number of words.

Searching Phone Numbers

- **10-Digit Search:** Best case $O(1)$, worst case $O(n)$.
- **7-Digit Search:** Best case $O(1)$, worst case $O(n)$.
- **3-Digit and 4-Digit Searches:** Best case $O(1)$ per table, worst case $O(n)$.
- **Phone Number Parsing:** $O(\log_{10} N)$, where N is the phone number length.
- **Overall Complexity:** $O(\log_{10} N + 4 \cdot n)$, assuming up to 4 hash tables are checked.

4) Code

Paste your code below:

```
import math

# Phone keypad mapping for letters to numbers
keypad_mapping = {
    ’2’: ’ABC’, ’3’: ’DEF’, ’4’: ’GHI’,
    ’5’: ’JKL’, ’6’: ’MNO’, ’7’: ’PQRS’,
    ’8’: ’TUV’, ’9’: ’WXYZ’
}

class ListNode:
    """Node for separate chaining in hash table."""
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.next = None

class HashTable:
    def __init__(self, size):
        self.size = size
        self.table = [None] * size  # Array of linked lists for separate chaining

    def hash_function(self, key):
        """Hash function using multiplication method."""
        A = 0.6180  # Approximation of the golden ratio
        fractional_part = (key * A) % 1
        return math.floor(self.size * fractional_part)

    def insert(self, key, value):
        """Insert a key-value pair into the hash table."""
        index = self.hash_function(key)
        if self.table[index] is None:
            self.table[index] = ListNode(key, value)
        else:
            current = self.table[index]
            while current.next is not None:
                current = current.next
            current.next = ListNode(key, value)

    def search(self, key):
        """Search for a key in the hash table and return the associated value."""
        index = self.hash_function(key)
        current = self.table[index]
        results = []
        while current is not None:
            if current.key == key:
                results.append(current.value)
            current = current.next
        return results  # Return list of matchesa

    def print_table(self):
        """Print the contents of the hash table."""
        for i in range(self.size):
            print(f"Index {i}:", end=" ")
            current = self.table[i]
            if current is None:
                print("Empty")
            else:
                # Traverse linked list at this index and print all nodes
                while current is not None:
                    print(f"({current.key}, {current.value})", end=" -> ")
                    current = current.next
                print("None")  # End of linked list

# Define separate hash tables for different number lengths
table_size = 5003
hash_tables = {
    10: HashTable(table_size),  # 10-digit numbers
    7: HashTable(table_size),   # 7-digit numbers
    3: HashTable(table_size),   # 3-digit exchanges
    4: HashTable(table_size)    # 4-digit numbers
}

def word_to_number(word):
    """Convert a word to a numeric phone number based on keypad mapping."""
    number = ""
    for char in word.upper():
        for key, letters in keypad_mapping.items():
            if char in letters:
                number += key
                break
    return int(number)

def insert_word(word):
    """Insert a word into the appropriate hash tables based on its length."""
    number = word_to_number(word)
    length = len(str(number))
    if length in hash_tables:
        hash_tables[length].insert(number, word)

def load_words_from_file(filename="all_words.txt"):
    """Load words from a file and insert them into the hash tables."""
    try:
        with open(filename, 'r') as file:
            for line in file:
                word = line.strip()
                if 2 <= len(word) <= 10:  # Only words with 2 to 10 letters
                    insert_word(word)
        #print("All words loaded successfully.")
    except FileNotFoundError:
        print(f"File '{filename}' not found.")

def search_phone_number(phone_number):
    """Search for a word-based representation of a phone number."""

    """ NEED TO REMOVE THE 1 IN THE BEGINING IF IT IS THERE """

    phone_number = int(phone_number)

    if phone_number >= 10**10:
        phone_number = int(str(phone_number)[1:])

    #seperate the numbers into parts
    area_code, last_seven = divmod(phone_number, 10000000)
    #print('areacode: ',area_code,'last seven: ', last_seven)
    exchange, number = divmod(last_seven, 10000)
    #print('exchange: ',exchange,'number: ',number)

    # Check 10-digit representation
    if phone_number >= 10**9:  # Ensure it's a 10-digit number
        results = hash_tables[10].search(phone_number)
        if results:
            return [f"1-{word}" for word in results]

    # Check 7-digit representation
    results = hash_tables[7].search(last_seven)
    if results:
        return [f"1-{area_code}-{word}" for word in results]

    # Check 3-digit exchange and 4-digit number separately
    exchange_results = hash_tables[3].search(exchange)
    number_results = hash_tables[4].search(number)
    if exchange_results and number_results:
        return [f"1-{exchange}-{ex}-{num}" for ex in exchange_results for num in
            number_results]

    # Only 3-digit exchange
    if exchange_results:
        return [f"1-{exchange}-{word}-{number}" for word in exchange_results]

    # Only 4-digit number
    if number_results:
        return [f"1-{area_code}-{exchange}-{word}" for word in number_results]
```

```python

    # Default: return number in standard format
    return f"1-{area_code}-{exchange}-{number}"


def main():
    # Load words from all_words.txt
    # Inserts them into the hash table
    load_words_from_file()

    # Get phone number from user
    user_input = input("Enter Phone Number: ")

    print(search_phone_number(user_input))
if __name__ == "__main__":
    main()
```