Yaren BOLAT 19232810032

ECON381 Fall 2024

Homework Assignment 3

Report: Algorithm to Generate an 8-Character Password Based on Keyboard Layout and Valid Moves

Introduction

Passwords are a crucial aspect of security in today's digital world. It’s essential that passwords are both strong (difficult to guess) and easy to type (convenient and quick). This becomes especially important when using devices that don't have attached keyboards, such as smart TVs, where you might need to use a remote control to select characters on a virtual keyboard.

In this assignment, we will develop an algorithm to generate a strong 8-character password based on the following set of rules:

1. The user will select the first character of the password, and the system will suggest the remaining 7 characters.
2. For each character selected, the system will find a list of characters that are at least 2 but no more than 3 moves away on a typical QWERTY keyboard.
3. The system will randomly select the next character from this list of valid moves.
4. The process will stop when the password reaches 8 characters.

Question 1: Which Distance Metric is Usable for Distances Between Keys?

The most suitable distance metric for calculating distances between keys on a keyboard is the **Manhattan distance**.

Why Manhattan Distance?

- The Manhattan distance measures how far two points are by summing the absolute differences in their horizontal and vertical positions. On a keyboard, keys are arranged in rows and columns, so you can think of each key's position as a grid coordinate.

For example, to get from the key 'g' to the key 'd', we would count how many horizontal and vertical moves are required. This is exactly what the Manhattan distance does: it calculates how far two points are on a grid by counting the total horizontal and vertical steps.

Formula for Manhattan Distance:

If two keys are at positions (row1, col1) and (row2, col2) on the keyboard, the Manhattan distance is calculated as:

distance = |row1 - row2| + |col1 - col2|

Question 2: Would You Need a Particular Data Structure to Represent the Keyboard Layout?

Yes, to represent the keyboard layout, we would use a **2D array** (also called a list of lists).

Why a 2D Array?

- The keyboard has a natural row and column structure, so a 2D array is a good way to represent it. Each position in the array corresponds to a key on the keyboard.

For example, the first few rows of a QWERTY keyboard could be represented as:

```
keyboard = [
  ['q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p'],
  ['a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l'],
  ['z', 'x', 'c', 'v', 'b', 'n', 'm'],
  ['1', '2', '3', '4', '5', '6', '7', '8', '9', '0']
]
```

When Should We Replace the 2D Array?

Once the distances between keys are calculated, we no longer need the 2D array. A more efficient data structure, like a **HashMap** or **Adjacency List**, can store just the valid moves (keys that are 2-3 moves away from a given key).

Question 3: What Java Data Structure Would Be the Best Suited for Mapping Each Key to a List of Valid Moves?

A **HashMap** is the best data structure for this task.

Why a HashMap?

- A **HashMap** provides efficient key-value mapping. In our case, the **key** would be a character (e.g., 'd'), and the **value** would be a **List** of characters (valid moves that are 2-3 moves away from that key).

The HashMap structure would look like:

HashMap<Character, List<Character>> validMoves = new HashMap<>();

Advantages of Using a HashMap:

- **Fast Lookups**: You can quickly retrieve the list of valid moves for any key.
- **Efficient Storage**: Each key has an associated list of valid moves, making it easy to store and retrieve the necessary data.

Question 4: Pseudocode for Creating an 8-Character Password Using the Data Structure

Now, let's outline the pseudocode for generating the 8-character password.

```
1. Initialize the keyboard layout (optional, only needed for distance calculation)

2. Create a HashMap to store valid moves for each key:

  - HashMap<Character, List<Character>> validMoves
```

3. Populate the HashMap with valid moves for each key:

   - For each key, calculate the keys that are 2-3 moves away and store them in the map.

4. Define the function to generate the password:

```
function generatePassword(startingKey):


    password = [startingKey]  // Start password with the first selected key.

    currentKey = startingKey


    // Loop to generate the next 7 characters

    for i from 1 to 7:

       // Get the list of valid moves for the current key

       validMoveList = validMoves.get(currentKey)


       // Randomly choose the next key from the valid moves

       nextKey = randomChoice(validMoveList)


       // Add the next key to the password

       password.append(nextKey)

       currentKey = nextKey


    return password
```

5. Example usage:

```
startingKey = 'd'  // User selects the first key.

password = generatePassword(startingKey)

print("Generated password: " + password)
```

Explanation of the Algorithm:

1. We first initialize the keyboard layout and create the validMoves map.
2. We populate this map with the valid moves for each key based on their Manhattan distances.
3. The function generatePassword starts with a user-chosen key and then randomly selects the next key from the valid moves, continuing until 8 characters are generated.

Question 5: Compute the List of Valid Moves for Specific Keys

Let's compute the valid moves for the keys 'a', 'f', 'h', '8', '0', and 'p'.

Here are the valid moves for each key (based on 2-3 moves away on a QWERTY layout):

1. **For key 'a':**
   - Valid moves: ['q', 's', 'd', 'z', 'x']
2. **For key 'f':**
   - Valid moves: ['d', 'g', 'r', 'v', 't']
3. **For key 'h':**
   - Valid moves: ['g', 'j', 'k', 'f', 'b']
4. **For key '8':**
   - Valid moves: ['7', '9', '5']
5. **For key '0':**
   - Valid moves: ['9']
6. **For key 'p':**
   - Valid moves: ['o', 'i', 'l']

These lists contain keys that are within 2-3 moves away from the respective starting key.

Conclusion

This report presents an approach to generating a strong yet easy-to-type 8-character password, specifically designed for use on devices like smart TVs where typing is not done via a regular keyboard. The algorithm uses the **Manhattan distance** to calculate the minimum moves between keys and generates a password by randomly selecting valid moves (keys 2-3 moves away).

- The **Manhattan distance** is the best choice for measuring distances between keys.
- A **2D array** can represent the keyboard layout, and once distances are calculated, a **HashMap** can store valid moves.
- The pseudocode outlines the steps for generating the password using this approach.

This solution strikes a balance between password security and ease of typing, making it ideal for use in restricted input environments like smart TVs.