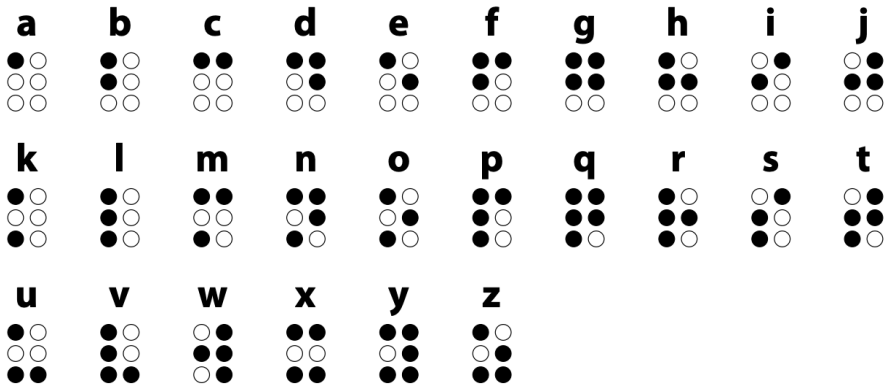
**Thinkerbell Labs Private Limited**

CIN: U72900KA2016PTC094046
4th Floor, JK Towers, 46th cross, Marenahalli Road,
8th Block, Jayanagar, Bengaluru - 560070
+91 9579077793 | contactus@thinkerbelllabs.com

# Braille Autocorrect and Suggestion System Task

## Context on Braille

Braille is a tactile writing system used by visually impaired individuals. It consists of raised dots arranged in a 2x3 grid, where each unique pattern represents a letter, number, or symbol. Since typing in Braille requires precision, errors such as missing, extra, or mistyped dots can occur frequently. An effective auto-correct system is essential to improve user experience and minimize frustration.




**You can learn QWERTY braille typing using this link: CLICK HERE**

## QWERTY Braille Typing Format

For users who do not have a Braille keyboard, Braille can be typed using a standard QWERTY keyboard. The most common method is **six-key entry**, where specific keys correspond to the six Braille dots:

- **D, W, Q, K, O,  P** represent Braille dots **1, 2, 3, 4, 5, and 6**, respectively.
- Users press multiple keys simultaneously to form a single Braille character.
- For example, pressing **D and K** together would produce the Braille letter **C**.
- Space, Enter, and Backspace function as normal for word separation and correction.

Errors in Braille input via QWERTY keyboards often stem from mispressing keys or pressing additional keys accidentally. Your task is to create an autocorrect system tailored to these issues.

## Problem Statement

Create an auto-correct and suggestion system for Braille input. Given a list of predefined Braille words, if a user enters an incorrect sequence, the system should:

1. **Suggest the closest possible word** based on the entered Braille pattern.
2. **Handle typos and missing/extra inputs** efficiently.
3. **Optimize for speed** to support large dictionaries and real-time correction.

## Requirements

- Implement a program that takes a Braille input sequence (typed via QWERTY Braille format) and suggests the most likely intended word.
- Use an efficient algorithm (such as Levenshtein distance or Trie-based approaches) to compare input sequences with the dictionary.
- Ensure that the system works well with a large dictionary and provides real-time suggestions.
- Clearly document your approach, optimizations, and trade-offs.

## Bonus

- Support multiple languages or Braille contractions.
- Implement a learning mechanism to improve suggestions based on previous inputs.

## Evaluation Criteria

- **Accuracy**: How well the system suggests the correct word.
- **Efficiency**: How quickly it processes inputs.
- **Code Quality**: Readability, structure, and maintainability.
- **Innovation**: Additional features beyond basic requirements.

## Submission

Provide a brief write-up explaining your approach, and sample test cases demonstrating the system's effectiveness.
You can complete this at your own pace (the quicker the better) and revert to engineering@thinkerbelllabs.com for any questions, doubts, clarifications, etc. anytime.