Lab 11

Problem 1

You are given a short story and a pattern. Your task is to find all occurrences of the pattern in the story using the brute-force string matching algorithm. The story is a string of text, and you need to find the starting indices where the pattern appears in the story.

Input:

- **Story**: `"Alice walked into the forest. Alice wondered what she would find. Suddenly, Alice saw a rabbit."`
- **Pattern**: `"Alice"`

Task:

Describe **step by step** how you would apply the brute-force string matching algorithm to find all occurrences of the pattern `"Alice"` in the given story.

Answer

Step 1: Align the pattern at the beginning of the text

We start by aligning the pattern `"Alice"` with the first position of the text, which is the beginning of the string `"Alice walked into the forest. Alice wondered what she would find. Suddenly, Alice saw a rabbit."`.

Step 2: Compare each character of the pattern to the corresponding character in the text

We will now compare each character of the pattern `"Alice"` with the corresponding character in the text.

- **First alignment** (at the start of the text):
  - Compare `"A"` (pattern) with `"A"` (text) → match.
  - Compare `"l"` (pattern) with `"l"` (text) → match.
  - Compare `"i"` (pattern) with `"i"` (text) → match.

    - Compare `"c"` (pattern) with `"c"` (text) → match.
    - Compare `"e"` (pattern) with `"e"` (text) → match.
- All characters match. So, the pattern `"Alice"` matches at the **first position** of the text, which is index **0** (0-based index).

Step 3: Move the pattern one position to the right

Now that we have matched the first occurrence, we move the pattern one position to the right and repeat the comparison.

- **Second alignment** (shift the pattern one position to the right):
    - The second alignment starts at position 1 in the text: `"lice walked into the forest. Alice wondered what she would find. Suddenly, Alice saw a rabbit."`.
    - Compare `"A"` (pattern) with `"l"` (text) → mismatch.
- Since there's a mismatch, we move the pattern one more position to the right and start over.

Step 4: Continue shifting and comparing

We continue shifting the pattern one position to the right and comparing until we reach the next successful match.

- **Alignment** (shift the pattern to position 30):
    - The fourth alignment starts at position 30 in the text: `"Alice wondered what she would find. Suddenly, Alice saw a rabbit."`.
    - Compare `"A"` (pattern) with `"A"` (text) → match.
    - Compare `"l"` (pattern) with `"l"` (text) → match.
    - Compare `"i"` (pattern) with `"i"` (text) → match.
    - Compare `"c"` (pattern) with `"c"` (text) → match.
    - Compare `"e"` (pattern) with `"e"` (text) → match.

- All characters match again. So, the pattern `"Alice"` matches at position **30** (0-based index).

5. **Alignment** (shift the pattern to position 76):
    - The fifth alignment starts at position 62 in the text: `"Alice saw a rabbit."`.
    - Compare `"A"` (pattern) with `"A"` (text) → match.
    - Compare `"l"` (pattern) with `"l"` (text) → match.
    - Compare `"i"` (pattern) with `"i"` (text) → match.
    - Compare `"c"` (pattern) with `"c"` (text) → match.
    - Compare `"e"` (pattern) with `"e"` (text) → match.
6. All characters match again. So, the pattern `"Alice"` matches at position **76** (0-based index).

Conclusion

- The pattern `"Alice"` is found at positions **0**, **30**, and **76** in the text.

Problem 2

You are analyzing customer reviews to check if any of them mention a specific complaint keyword.

- The **pattern** you're searching for is: `LATE`
- The **text** from a feedback entry is:
  `"THE DELIVERY WAS VERY LATE AND DAMAGED THE PACKAGE"`

To efficiently find the pattern, you decide to apply **Horspool's Algorithm**, which is optimized for fast string searching using a **shift table** and right-to-left comparisons.

Tasks:

1. **Build the shift table** for the pattern `LATE` using Horspool's rule:
2. **Step by step**, describe how Horspool's Algorithm compares characters and performs shifts on the given text.

Answer

Build the Shift Table for Pattern `LATE`

Pattern: `LATE`
Length of the pattern: 4

We compute shift values for all characters **except** the last character (`E`), using the formula:

```
shift = pattern length - position - 1
```

Character	Position	Shift
L	0	4 - 0 - 1 = 3
A	1	4 - 1 - 1 = 2
T	2	4 - 2 - 1 = 1

For all other characters not in the pattern, the default shift value is 4 (the length of the pattern).

Final Shift Table:

```
L: 3
A: 2
T: 1
All others: 4
```

Step-by-Step Execution of Horspool's Algorithm

Text: `"THE DELIVERY WAS VERY LATE AND DAMAGED THE PACKAGE"`
Pattern: `"LATE"`
Pattern length: 4

We align the pattern with the text and compare characters from **right to left**. The algorithm shifts the pattern based on the character in the text that aligns with the last character of the pattern (`E`).

Step 1
Text index 0–3: `THE`
Pattern: `LATE`

- Compare `E` (pattern) with `' '` (text[3]): mismatch
- `' '` is not in the pattern → shift = 4

- New alignment starts at index 4

Step 2
Text index 4–7: `DELI`
Pattern: `LATE`

- Compare `E` with `I` (text[7]): mismatch
- `I` is not in the pattern → shift = 4
- New alignment starts at index 8

Step 3
Text index 8–11: `VERY`
Pattern: `LATE`

- Compare `E` with `Y` (text[11]): mismatch
- `Y` is not in the pattern → shift = 4
- New alignment starts at index 12

Step 4
Text index 12–15: ` WAS`
Pattern: `LATE`

- Compare `E` with `S` (space at text[15]): mismatch
- `S` is not in the pattern → shift = 4
- New alignment starts at index 16

Step 5
Text index 16–19: ` VER`
Pattern: `LATE`

- Compare `E` with `R` (text[19]): mismatch
- `R` is not in the pattern → shift = 4
- New alignment starts at index 20

Step 6
Text index 16–19: `Y LA`
Pattern: `LATE`

- Compare `E` with `A` (text[23]): mismatch
- `A` is in the pattern → shift = 2
- New alignment starts at index 22

Step 6
Text index 22–25: `LATE`
Pattern: `LATE`

- Compare `E` with `E` (text[25]): match
- Compare `T` with `T` (text[24]): match
- Compare `A` with `A` (text[23]): match
- Compare `L` with `L` (text[22]): match
- All characters match → pattern found at index 22

Conclusion: The pattern `LATE` is found at position 22 in the text using Horspool's Algorithm.

Problem 3

You are implementing an auto-complete system that uses a **Trie** data structure. The system must support inserting words, checking if a word exists, and verifying whether any word starts with a given prefix.

Given the following list of words:

```
["cat", "cap", "can", "camp", "camera"]
```

Tasks:

1. Draw the **Trie structure** that results from inserting all the given words.
2. Use the Trie to determine whether the prefix `"cam"` exists in the structure.
3. Check whether the word `"capo"` exists in the Trie.

<u>**Answer**</u>

Trie Structure

```
  (root)
    |
    c
    |
    a
 / / \ \
t p  n  m
| |  |  | \
\0\0 \0 p  e
```

```
|    \
\0    r
      |
      a
      |
      \0
```

Check if the Prefix "cam" Exists

Perform a prefix traversal from the root, following the characters `'c' → 'a' → 'm'`. Since this is a prefix check, we only need to confirm that each character node exists; we do not need to find a `\0` marker.

Steps:

1. Start at the root.
2. Go to child `'c'` → exists.
3. From `'c'`, go to child `'a'` → exists.
4. From `'a'`, go to child `'m'` → exists.

Conclusion: All characters of the prefix `"cam"` are found.
Result: The prefix `"cam"` **exists** in the Trie.

Check if the Word `"capo"` Exists

Perform a word-level traversal: `'c' → 'a' → 'p' → 'o'`, and ensure that the last character leads to a `\0` marker to denote a full word. If any character is missing or no termination is found, the word does not exist.

Steps:

1. Start at the root.
2. Go to child `'c'` → exists.
3. From `'c'`, go to child `'a'` → exists.
4. From `'a'`, go to child `'p'` → exists.
5. From `'p'`, attempt to go to child `'o'` → **not found**.

Conclusion: The character `'o'` is missing in the path.
Result: The word `"capo"` **does not exist** in the Trie.

Problem 4

In a genetics research lab, scientists are analyzing DNA sequences to detect specific **harmful gene patterns** associated with increased risk of hereditary disorders. You are given a DNA sequence and a known **harmful gene pattern**, and your task is to find all locations where this pattern occurs in the sequence using an **efficient string matching algorithm**.

Each occurrence of the harmful pattern corresponds to a potential mutation site. If the pattern appears **more than twice** in the sequence, the DNA is considered at risk for **genetic instability**, which may contribute to cancer or other genetic disorders.

Input:

- **DNA sequence**:
  `"ATCGTACGATCGGATCATCG"`

- **Harmful gene pattern**:
  `"ATCG"`

Task:

1. Use an **efficient string matching algorithm** to find all starting indices (0-based) where the pattern occurs in the DNA sequence.

2. Describe **step by step** how your algorithm searches for the pattern in the DNA string. Based on the number of matches found, determine whether the sequence should be flagged for **genetic instability**.

<u>Answer</u>

We are given:

- DNA sequence: `"ATCGTACGATCGGATCATCG"`
- Harmful gene pattern: `"ATCG"`

We will use an efficient string matching algorithm, specifically **Horspool's Algorithm**, to find all occurrences of the pattern in the DNA sequence.

Step 1: Construct the Shift Table

Horspool's algorithm uses a shift table to determine how far the pattern can be moved when a mismatch occurs. The shift value for each character in the alphabet (in this case, A, T, C, G) is calculated based on its rightmost position in the pattern, excluding the last character.

Pattern: `"ATCG"` (length = 4)
We exclude the last character `'G'` when building the table.

We compute the shift values as follows:

- `'A'` is at index 0 → shift = 3 (4 - 1 - 0)
- `'T'` is at index 1 → shift = 2 (4 - 1 - 1)
- `'C'` is at index 2 → shift = 1 (4 - 1 - 2)
- `'G'` does not appear in the first three characters → shift = 4 (pattern length)

Final shift table:

```
A: 3
T: 2
C: 1
others (including G): 4
```

Step 2: Apply Horspool's Algorithm

We align the pattern with the text and compare characters from right to left. If the characters all match, we record the position. If a mismatch occurs, we shift the pattern according to the shift table using the character aligned with the last character of the pattern.

We compare substrings of length 4 from index 0 to 16.

Detailed steps:

1. Index 0 → Text window: `"ATCG"`
    - G == G → match
    - C == C → match
    - T == T → match
    - A == A → match
      Pattern found at index 0
      Shift by shift[G] = 4 → Next index: 4

2. Index 4 → Text window: `"TACG"`
    - G == G → match
    - C == C → match
    - A != T → mismatch
      Shift by shift[G] = 4 → Next index: 8
3. Index 8 → Text window: `"ATCG"`
    - G == G → match
    - C == C → match
    - T == T → match
    - A == A → match
      Pattern found at index 8
      Shift by shift[G] = 4 → Next index: 12
4. Index 12 → Text window: `"GATC"`
    - C != G → mismatch
      Shift by shift[C] = 1 → Next index: 13
5. Index 13 → Text window: `"ATCA"`
    - A != G → mismatch
      Shift by shift[A] = 3 → Next index: 16
6. Index 16 → Text window: `"ATCG"`
    - G == G → match
    - C == C → match
    - T == T → match
    - A == A → match
      Pattern found at index 16
      Shift by shift[G] = 4 → End of string

Step 3: Final Result

The pattern `"ATCG"` is found at the following indices in the DNA sequence:
[0, 8, 16]

Step 4: Interpretation

Since the harmful gene pattern appears more than two times, the DNA sequence is flagged as potentially unstable.