Solution to Problem 5.3.21

Problem Modify the Rabin-Karp algorithm to search for a given pattern where the middle character is treated as a "wildcard" (any character can match it).

—

Approach to Modify Rabin-Karp

To handle the wildcard in the middle of the pattern, we:

1. Exclude the wildcard character when computing the hash of the pattern and substrings of the text.
2. Use a rolling hash to compute hashes efficiently while ignoring the wildcard character.
3. Perform a verification step to compare the characters of the pattern and the text, skipping the wildcard position.

—

Modified Rabin-Karp Algorithm (Python Implementation)

```python
def rabin_karp_with_wildcard(text, pattern):
    n = len(text)
    m = len(pattern)
    mid = m // 2  # Middle character index (wildcard
        position)
    prime = 101  # A prime number for modular arithmetic
    base = 256   # Number of characters in the input
        alphabet

    # Compute the hash of the pattern (ignoring the wildcard
        )
    def compute_hash(s, ignore_index):
        h = 0
        for i in range(len(s)):
            if i != ignore_index:  # Ignore the wildcard
                position
                h = (h * base + ord(s[i])) % prime
        return h

    # Rolling hash function
    def roll_hash(prev_hash, old_char, new_char,
        ignore_index, window_start):
        if window_start != ignore_index:
            prev_hash = (prev_hash - ord(old_char) * (base
                ** (m - 1)) % prime + prime) % prime
        prev_hash = (prev_hash * base + ord(new_char)) %
            prime
        return prev_hash

    # Precompute the hash for the pattern and the first window in the text
    pattern_hash = compute_hash(pattern, mid)
    text_hash = compute_hash(text[:m], mid)

    matches = []

    # Sliding window through the text
    for i in range(n - m + 1):
        if pattern_hash == text_hash:  # Hashes match, verify character-by-character
            match = True
            for j in range(m):
                if j != mid and text[i + j] != pattern[j]:  # Skip wildcard position
                    match = False
                    break
            if match:
                matches.append(i)  # Store the starting index of the match

        # Update the hash for the next window
        if i < n - m:
            text_hash = roll_hash(
                text_hash, text[i], text[i + m], mid, i
            )

    return matches


# Example Usage
text = "ABCABCDAB?CDE"
pattern = "AB?CD"
matches = rabin_karp_with_wildcard(text, pattern)
print("Matches found at indices:", matches)
```

—

Explanation

1. **Hash Calculation**:

   - The hash of the pattern is calculated while ignoring the contribution of the wildcard character.
   - Similarly, the substrings of the text are hashed while ignoring the middle character.

2. **Hash Comparison**:

- If the hash of a substring matches the hash of the pattern, perform a verification step.
- Compare each character of the substring with the pattern, skipping the wildcard position.

3. **Rolling Hash**:
   - The rolling hash is updated efficiently by subtracting the contribution of the first character, adding the contribution of the new character, and ignoring the wildcard.

—

Example Input and Output

Input:

```
Text:    ABCABCDAB?CDE
Pattern: AB?CD
```

Output:

```
Matches found at indices: [2, 7]
```

Explanation:

- The pattern matches at indices 2 and 7 because the wildcard can match any character.
- At index 2: `ABCABCD` matches `AB?CD`.
- At index 7: `DAB?CD` matches `AB?CD`.

—

Complexity Analysis

- **Time Complexity**:
  - Hash computation: $O(m)$, where m is the length of the pattern.
  - Rolling hash and verification: $O(n)$, where n is the length of the text.
  - Total: $O(n + m)$.
- **Space Complexity**:
  - $O(1)$: The space usage is constant.

—

Summary

To modify the Rabin-Karp algorithm for a pattern with a wildcard in the middle:

- Ignore the wildcard character when computing and comparing hashes.
- Use a rolling hash for efficient computation.
- Verify matches by comparing characters, skipping the wildcard position.

This approach retains the efficiency of Rabin-Karp while accommodating the wildcard.