936. Stamping The Sequence

Leetcode Link

Problem Explanation

You are given a string (`target`) and a sequence of characters (`stamp`). At first, the `target` string is completely hidden, represented by `?` marks. The aim is to reveal the `target` string by applying the `stamp` repeatedly against the `target` string. On each turn, you may place the stamp over the mask (a substring of the hidden string), and replace every mark in the sequence with the corresponding character from the `stamp`. You are not limited to stamping from left to right -- you can use a stamp anywhere on the current sequence. If stamping is possible, return the sequence of indices on the target string where the leftmost character of the stamp is applied at each turn. If `target` string cannot be formed by stamping, return an empty list.

As an example: Input: stamp = "abc", target = "ababc" At the first step, we can stamp on the first 3 characters, so our operation is "?????" -> "abc??". The index of the left-most letter being stamped at this turn is 0. At the second step, we can stamp on last 3 characters, so our operation is "abc??" -> "ababc". The index of the left-most letter being stamped at this turn is 2. So, we return [0,2] for this example. Other valid outputs would be [0,1,2] or [1,0,2].

Solution Explanation

The solution uses a greedy algorithm. The key observation is that for any overlap between two stamps, the stamp applied later must cover an unchanged part of the target string and a previously stamped part. Therefore, we should focus on applying stamps on the leftmost still unchanged parts, which means we are applying stamps in reverse order. The algorithm begins from the rightmost part of the target string, tries to match the stamp starting from each position i in the target, and if possible, replaces the matched part of the target string with `*` characters and records this position i. We repeat this process until we cannot stamp out a new substring, then reverse our answer and return it.

```
1
2 text
3                     Initial target: ??????
4 Abc at 2nd position : ???abc -> ??abcabc -> ?abcabc* -> Abcabc**
5 Abc at 0th position : Abcabc** -> **abcabc*
6 Abc at 0th position : **abcabc* -> *****abc
7 Abc at 3rd position : ******abc -> ********
```

Python Solution

```python
python
class Solution:
    def movesToStamp(self, stamp: str, target: str) -> List[int]:
        # Initialize variables
        stamp = list(stamp)
        target = list(target)
        stamp_len = len(stamp)
        target_len = len(target)
        visited = [False]*target_len
        ans = []

        def can_stamp(start):
            # Check if we can apply the stamp at this position
            i = start
            j = 0
            stamped = False
            while j<stamp_len and i<target_len:
                if target[i] == '?':
                    i += 1
                    j += 1
                elif target[i] == stamp[j]:
                    i += 1
                    j += 1
                    stamped = True
                else:
                    return False
            if j == stamp_len:
                return stamped
            return False

        # Try to apply the stamp at each position
        for _ in range(target_len):
            for i in range(target_len-stamp_len+1):
                # If this position is not visited and we can stamp at this position
                if not visited[i] and can_stamp(i):
                    # Apply the stamp
                    for j in range(i,i+stamp_len):
                        target[j] = '?'
                    # Note that we visited this position
                    visited[i] = True
                    ans.append(i)
                    break
            # If we cannot stamp anymore, just break
            else:
                break

        # Check if we stamped the whole target successfully
        if all(c == '?' for c in target):
            ans.reverse()
            return ans
        else:
            return []
```

Java Solution

For solutions in other languages like Java, Javascript, C++, C#, the overall algorithm doesn't change. I am providing the Java solution below with similar comments for better understanding.

```
1
2 java
3 class Solution {
4     public int[] movesToStamp(String stamp, String target) {
5         char[] s = stamp.toCharArray();
6         char[] t = target.toCharArray();
7         List<Integer> res = new ArrayList<>();
8         boolean[] visit = new boolean[t.length];
9         int stars = 0;
10
11        while (stars < t.length) {
12            boolean doneReplace = false;
13            for (int i = 0; i <= t.length - s.length; i++) {
14                if (!visit[i] && canReplace(t, i, s)) {
15                    stars = doReplace(t, i, s.length, stars);
16                    doneReplace = true;
17                    res.add(i);
18                    visit[i] = true;
19
20                    if (stars == t.length) {
21                        break;
22                    }
23                }
24            }
25
26            if (!doneReplace) {
27                return new int[0];
28            }
29        }
30
31        int[] ans = new int[res.size()];
32        for (int i = 0; i < res.size(); i++) {
33            ans[i] = res.get(res.size() - 1 - i);
34        }
35        return ans;
36    }
37
38    private boolean canReplace(char[] t, int p, char[] s) {
39        for (int i = 0; i < s.length; i++) {
40            if (t[i + p] != '*' && t[i + p] != s[i]) {
41                return false;
42            }
43        }
44        return true;
45    }
46
47    private int doReplace(char[] t, int p, int len, int count) {
48        for (int i = 0; i < len; i++) {
49            if (t[i + p] != '*') {
50                t[i + p] = '*';
51                count++;
52            }
53        }
54        return count;
55    }
56 }
```

Notice for each language the function or method that checks if we can apply the `stamp` at a given position and the function or method that applies the `stamp` at that position and marks it as visited, is the center of attention. Then the greedy algorithm comes into play where we apply the stamp at any valid position we can find until we can't anymore, then we check if the whole string is stamped out.# JavaScript Solution

Implementing the solution in JavaScript, we use the functionality of Array.prototype.every() method and Array.prototype.slice() method for processing the arrays representing the string inputs.

```
1
2 javascript
3 var movesToStamp = function(stamp, target) {
4     let t = target.split(''), s = stamp.split('');
5     let tlen = t.length, slen = s.length, res = [], turn;
6     let done = Array(tlen).fill(false);
7     let total = 0;
8
9     const canReplace = (t, i, s, changeToStar) => {
10        let replaced = false;
11        for(let j = 0; j < slen; j++) {
12            if(t[i+j] === '?') {
13                continue;
14            }
15            if(t[i+j] !== s[j]) {
16                return false;
17            }
18            if(!changeToStar) {
19                replaced = true;
20            } else {
21                t[i+j] = '?';
22            }
23        }
24        return replaced;
25    }
26
27    do {
28        turn = false;
29        for(let i = 0; i <= tlen - slen; i++) {
30            if(!done[i] && canReplace(t, i, s, false)) {
31                total += slen;
32                turn = done[i] = true;
33                res.push(i);
34                canReplace(t, i, s, true);
35            }
36        }
37    } while(turn && total < tlen )
38
39    if(total === tlen) {
40        return res.reverse();
41    } else {
42        return [];
43    }
44 };
```

In the JavaScript solution, we use the JavaScript function `split()` to convert the target and stamp strings into arrays of characters, `t` and `s` respectively. These arrays are then processed in the `canReplace()` function which checks if we can apply the `stamp` at a given position and also applies the `stamp` at that position. Like the other solutions, this solution employs a greedy algorithm to identify valid positions for `stamp` application until none can be found. Finally, the function checks whether the entire target string has been stamped out. If it has, the function returns the positions at which the `stamp` was applied in reverse order; if not, it returns an empty array.

Got a question? Ask the Teaching Assistant anything you don't understand.