Decode String

17 September 2025 10:00 PM

Given an encoded string **s**, decode it by expanding the pattern k[substring], where the substring inside brackets is written k times. Return the final decoded string.

Examples:

Input: *s = "3[b2[ca]]"*

Output*: bcacabcacabcaca*

Explanation:

Inner substring "2[ca]" breakdown into "caca".

Now , new string becomes "3[bcaca]"

Similarly "3[bcaca]" becomes "bcacabcacabcaca" which is final result.

Input*: s = "3[ab]"*

Output*: ababab*

Explanation*: The substring "ab" is repeated 3 times, giving "ababab".*

Table of Content

- [Approach 1] Using Two Stacks - O(n) Time and O(n) Space
- [Approach 2] Using Single Stack - O(n) Time and O(n) Space
- [Alternate Approach] Without Using Stack - O(n) Time and O(n) Space

[Approach 1] Using Two Stacks - O(n) Time and O(n) Space

Use two stacks (one for numbers, one for characters). When ']' is found, pop till '[' to form a substring, repeat it using th e top number, and push back — final stack gives the decoded string.

Steps of implementation:

1. Initialize two stacks → one for numbers and one for characters.
2. Traverse the string character by character.
   => If it's a number → push it to the number stack.
   => If it's an alphabet (a–z) or '[' → push it to the character stack.
   => If it's a closing bracket ']':
3. Whenever any close bracket (']') is encountered, pop the characters from the character stack until open bracket ('[') is foun d in the stack. Also, pop the top element from the integer stack, say n. Now make a string repeating the popped character n number of time. Now, push all character of the string in the stack.
4. After traversing whole string, integer stack will be empty and last string which will be formed will be the given result.
   Illustration:

```python
def decodedString(s):
    numStack = []
    charStack = []
    temp = ""
    res = ""

    i = 0
    while i < len(s):
        cnt = 0

        # If Digit, convert it into number and
        # push it into integer stack.
        if s[i].isdigit():
            while i < len(s) and s[i].isdigit():
                cnt = cnt * 10 + int(s[i])
                i += 1
            i -= 1
            numStack.append(cnt)

        # If closing bracket ']' is encountered
        elif s[i] == ']':
            temp = ""

            cnt = numStack.pop()

            # Pop element till opening bracket '[' is not found
            # in the character stack.
            while charStack[-1] != '[':
                temp = charStack.pop() + temp
            charStack.pop()

            # Repeating the popped string 'temp' count number of times.
            res = temp * cnt

            # Push it in the character stack.
            for c in res:
                charStack.append(c)

            res = ""
        else:
            charStack.append(s[i])
        i += 1

    # Pop all the elements, make a string and return.
    while charStack:
        res = charStack.pop() + res

    return res


if __name__ == "__main__":
    s = "3[b2[ca]]"
    print(decodedString(s))
```

Output

```
bcacabcacabcaca
```

[Approach 2] Using Single Stack - O(n) Time and O(n) Space

In this approach, we use a single stack to store both characters and digits. Instead of maintaining a separate integer stack for storing repetition counts, we store the digits directly in the main stack. The key observation is that the number always appears befo re the opening bracket '['. This allows us to retrieve it later without needing an extra stack.

Steps of implementation:

- Initialize an empty stack.
- Push characters onto the stack until ']' is encountered.
- When ']' is found:
  => Pop characters to form the substring until '[' is found, then remove '['.
  => Extract the preceding number from the stack and convert it to an integer.
  => Repeat the substring and push the expanded result back onto the stack.
- After traversal, pop all characters from the stack, reverse them, and return the final decoded string.

```
def decodeString(s: str) -> str:
    st = []

    for i in range(len(s)):

        # Push characters into the stack until ']' is encountered
        if s[i] != ']':
            st.append(s[i])

        # Decode when ']' is found
        else:
            temp = []

            # Pop characters until '[' is found
            while st and st[-1] != '[':
                temp.append(st.pop())
            temp.reverse()
            st.pop()

            num = []

            # Extract the number (repetition count) from the stack
            while st and st[-1].isdigit():
                num.insert(0, st.pop())

            # Convert extracted number to integer
            number = int("".join(num))
            repeat = "".join(temp) * number

            # Push the expanded string back onto the stack
            st.extend(repeat)

    # Pop all characters from stack to form the final result
    return "".join(st)

if __name__ == "__main__":
    str_val = "3[b2[ca]]"
    print(decodeString(str_val))
```

Output

```
bcacabcacabcaca
```

[Alternate Approach] Without Using Stack - O(n) Time and O(n) Space

The approach is like traversing the encoded string character by character while maintaining a result string. Whenever a closi ng bracket ']' is found, we extract the substring enclosed within the matching opening bracket '[' and retrieve the number that indicates how many times the substring should be repeated. This repeated substring is then added back to the current result. By continu ing this process until the end of the string, we obtain the fully decoded output.

```python
def decodedString(s):
    res = ""

    for i in range(len(s)):

        if s[i] != ']':
            res += s[i]

        else:
            temp = ""
            while res and res[-1] != '[':
                temp = res[-1] + temp
                res = res[:-1]

            # Remove the opening bracket from the result string.
            res = res[:-1]

            # Extract the preceding number and convert it to an integer.
            num = ""
            while res and res[-1].isdigit():
                num = res[-1] + num
                res = res[:-1]
            p = int(num)

            # Append the substring to the result
            #string, repeat it to the required number of times.
            res += temp * p

    return res


if __name__ == "__main__":
    s = "3[b2[ca]]"
    print(decodedString(s))
```

Output

```
bcacabcacabcaca
```

Decode the string | DSA Problem

From <https://www.geeksforgeeks.org/dsa/decode-string-recursively-encoded-count-followed-substring/>