394. Decode String

Medium | Topics | Companies

Given an encoded string, return its decoded string.

The encoding rule is: `k[encoded_string]`, where the `encoded_string` inside the square brackets is being repeated exactly `k` times. Note that `k` is guaranteed to be a positive integer.

You may assume that the input string is always valid; there are no extra white spaces, square brackets are well-formed, etc. Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, `k`. For example, there will not be input like `3a` or `2[4]`.

The test cases are generated so that the length of the output will never exceed 10^5.

Example 1:

```
Input: s = "3[a]2[bc]"
Output: "aaabcbc"
```

Example 2:

```
Input: s = "3[a2[c]]"
Output: "accaccacc"
```

Example 3:

```
Input: s = "2[abc]3[cd]ef"
Output: "abcabccdcdcdef"
```

Constraints:

- `1 <= s.length <= 30`
- `s` consists of lowercase English letters, digits, and square brackets `'[]'`.
- `s` is guaranteed to be **a valid** input.
- All the integers in `s` are in the range `[1, 300]`.

Accepted 713.1K | Submissions 1.2M | Acceptance Rate 58.5%

Approach 1: Using stack

→ Take a stack of strings.

→ num = 0

```
if ( Si is digit)
    num = num * 10 + Si - '0'
```

```
else if ( Si is [ )
{
        Push num as integer to stack
        make num = 0
        Push [
}
else if ( Si is ] )
{
        string temp
        while ( stack top is not [ )
        {
                temp = stack top + temp
                pop
        }

        Pop top  i.e. [
        now pop k from stack and take
    it as integer.
        string temp2

        while ( k -- )
            temp2 = temp2 + temp

        push temp2
}

else
{
    Push Si as string
}
```

After scanning the entire input string, construct the answer from stack.

```
class Solution {
public:
    string decodeString(string s) {
        int i=0;
        int num=0;
        stack<string> st;

        while(i<s.length()){ O(n)
            if(isdigit(s[i])) num=num*10 + (s[i] - '0');
            else if(s[i] == '['){
                st.push(to_string(num));
                num=0;
                st.push("[");
            }
            else if(s[i] == ']'){
                string temp="";
                while(st.top() != "["){ O(x)
                    temp=st.top()+temp;
                    st.pop();
                }
                st.pop();
                int k=stoi(st.top());
                st.pop();
                string temp2="";
                while(k--) temp2=temp2+temp; O(k)

                st.push(temp2);
            }
            else{
                string temp;
                temp.push_back(s[i]);
                st.push(temp);
```

```
            }

            i++;
        }

        string ans="";

        while(!st.empty()){  O(n)
            ans = st.top() + ans;
            st.pop();
        }

        return ans;
    }
};
```

$\Sigma O(x) = O(n)$

$\Sigma O(k) = O(n)$ because string concatenation takes linear time.

$T(n): O(3n) = O(n)$

$S(n): O(n)$