<https://leetcode.com/problems/decode-string/>

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].

**Attempt 1: 2023-10-22**

**Solution 1: Two Stacks (120 min)**

**Style 1: We must create new StringBuilder object 'sb' for each level of brackets, don't use 'sb.setLength(0)'**

class Solution {

    public String decodeString(String s) {

        Stack<Integer> intStack = new Stack<>();

        Stack<StringBuilder> resultStack = new Stack<>();

        int num = 0;

// StringBuilder 'sb' is the running storage responsible for

// storing any string inside a bracket, it will expand previous

// stored string to new value when traveling from most inside

// bracket level to outside bracket level

// e.g for input s = "3[a2[c]x]"

// sb changing from "c" -> "acc" -> "accx" -> "accxaccxaccx"

        StringBuilder sb = new StringBuilder();

        for(int i = 0; i < s.length(); i++) {

            char c = s.charAt(i);

            if(Character.isDigit(c)) {

                num = 10 \* num + (c - '0');

            } else if(c == '[') {

                intStack.push(num);

                resultStack.push(sb);

                num = 0;

                //sb.setLength(0);

                // -> cannot use this to avoid overwriting existing

                // 'preResult' string stored in StringBuilder object

                // 'sb', we have to create new object as a refresh

                // StringBuilder to receive chars inside current

                // brackets and build a temporary string as

                // 'stringInCurrentBrackets'

                sb = new StringBuilder();

            } else if(c == ']') {

                int repeat = intStack.pop();

                //StringBuilder tmp = resultStack.pop();

                StringBuilder tmp = sb;

                sb = resultStack.pop();

                for(int j = 0; j < repeat; j++) {

                    sb.append(tmp);

                }

            } else {

                sb.append(c);

            }

        }

        return sb.toString();

    }

}

**Style 2: A more readable way**

class Solution {

    public String decodeString(String s) {

        Stack<Integer> intStack = new Stack<>();

        Stack<StringBuilder> resultStack = new Stack<>();

        int num = 0;

// StringBuilder 'sb' is the running storage responsible for

// storing any string inside a bracket, it will expand previous

// stored string to new value when traveling from most inside

// bracket level to outside bracket level

// e.g for input s = "3[a2[c]x]"

// sb changing from "c" -> "acc" -> "accx" -> "accxaccxaccx"

        StringBuilder sb = new StringBuilder();

        for(int i = 0; i < s.length(); i++) {

            char c = s.charAt(i);

            if(Character.isDigit(c)) {

                num = 10 \* num + (c - '0');

            } else if(c == '[') {

                intStack.push(num);

                resultStack.push(sb);

                num = 0;

                //sb.setLength(0);

                // -> cannot use this to avoid overwriting existing

                // 'preResult' string stored in StringBuilder object

                // 'sb', we have to create new object as a refresh

                // StringBuilder to receive chars inside current

                // brackets and build a temporary string as

                // 'stringInCurrentBrackets'

                sb = new StringBuilder();

            } else if(c == ']') {

                // A more readable way

                int repeat = intStack.pop();

                StringBuilder stringInCurrentBrackets = sb;

                StringBuilder preResult = resultStack.pop();

                for(int j = 0; j < repeat; j++) {

                    preResult.append(stringInCurrentBrackets);

                }

                sb = preResult;

            } else {

                sb.append(c);

            }

        }

        return sb.toString();

    }

}

**Refer to**

<https://leetcode.com/problems/decode-string/solutions/87534/simple-java-solution-using-stack/comments/92413>

        Stack<Integer> intStack = new Stack<>();

        Stack<StringBuilder> strStack = new Stack<>();

        StringBuilder cur = new StringBuilder();

        int k = 0;

        for (char ch : s.toCharArray()) {

            if (Character.isDigit(ch)) {

                k = k \* 10 + ch - '0';

            } else if ( ch == '[') {

                intStack.push(k);

                strStack.push(cur);

                cur = new StringBuilder();

                k = 0;

            } else if (ch == ']') {

                StringBuilder tmp = cur;

                cur = strStack.pop();

                for (k = intStack.pop(); k > 0; --k) cur.append(tmp);

            } else cur.append(ch);

        }

        return cur.toString();

**Refer to**

<https://grandyang.com/leetcode/394/>

**我们也可以用迭代的方法写出来，当然需要用 stack 来辅助运算，我们用两个 stack，一个用来保存个数，一个用来保存字符串，我们遍历输入字符串，如果遇到数字，我们更新计数变量 cnt；如果遇到左括号，我们把当前 cnt 压入数字栈中，把当前t压入字符串栈中；如果遇到右括号时，我们取出数字栈中顶元素，存入变量k，然后给字符串栈的顶元素循环加上k个t字符串，然后取出顶元素存入字符串t中；如果遇到字母，我们直接加入字符串t中即可，参见代码如下：**

    class Solution {

        public:

        string decodeString(string s) {

            string t = "";

            stack<int> s\_num;

            stack<string> s\_str;

            int cnt = 0;

            for (int i = 0; i < s.size(); ++i) {

                if (s[i] >= '0' && s[i] <= '9') {

                    cnt = 10 \* cnt + s[i] - '0';

                } else if (s[i] == '[') {

                    s\_num.push(cnt);

                    s\_str.push(t);

                    cnt = 0; t.clear();

                } else if (s[i] == ']') {

                    int k = s\_num.top(); s\_num.pop();

                    for (int j = 0; j < k; ++j) s\_str.top() += t;

                    t = s\_str.top(); s\_str.pop();

                } else {

                    t += s[i];

                }

            }

            return s\_str.empty() ? t : s\_str.top();

        }

    };

**Solution 2: Recursion (120 min)**

**Style 1: For loop**

class Solution {

    // The int i is a globle parameter to indicate the current

    // index of char in the String s.

    int i = 0;

    public String decodeString(String s) {

        return helper(s);

    }

    private String helper(String s) {

        StringBuilder sb = new StringBuilder();

        int num = 0;

        for(; i < s.length(); i++) {

            char c = s.charAt(i);

            if(Character.isDigit(c)) {

                num = num \* 10 + c - '0';

            } else if(c == '[') {

                // Skip '['

                i++;

                String next = helper(s);

                for(int j = 0; j < num; j++) {

                    sb.append(next);

                }

                // reset the value for the next input in same level

                num = 0;

            } else if(c == ']') {

                // Skip ']' but no need 'i++'

                // Why no need to 'i++' here ?

                // Test out: "3[a]2[bc]"

                // Initially start with level 1 recursion, i = 0, char c = '3'

                // Then for loop i++ on level 1 recursion -> i = 1,

                // char c = '[', i++ -> i = 2, skip '['

                // Then go into level 2 recursion, i keep = 2 (even hit for loop

                // logic, but not hit i++ on for loop yet), char c = 'a'

                // Then for loop i++ on level 2 recursion -> i = 3,

                // char c = ']', break, Important !! -> no i++ here

                // Finish level 2 recursion and move back to level 1, i keep as 3

                // Then for loop i++ on level 1 recursion -> i = 4,

                // char c = '2', then continue similar logic on level 1 recursion

                // ... etc.

                // Note: we don't need i++ to skip ']', it auto skipped when move

                // back to parent level

                //i++;

                break;

            } else {

                sb.append(c);

            }

        }

        return sb.toString();

    }

}

**Style 2: While loop**

class Solution {

    // The int i is a globle parameter to indicate the current

    // index of char in the String s.

    int i = 0;

    public String decodeString(String s) {

        return helper(s);

    }

    private String helper(String s) {

        StringBuilder sb = new StringBuilder();

        int num = 0;

        while(i < s.length()) {

            char c = s.charAt(i);

            if(Character.isDigit(c)) {

                num = num \* 10 + c - '0';

            } else if(c == '[') {

                // Skip '['

                i++;

                String next = helper(s);

                for(int j = 0; j < num; j++) {

                    sb.append(next);

                }

                // reset the value for the next input in same level

                num = 0;

            } else if(c == ']') {

                // Skip ']' but no need 'i++'

                // Why no need to 'i++' here ?

                // Test out: "3[a]2[bc]"

                // Initially start with level 1 recursion, i = 0, char c = '3'

                // Then for loop i++ on level 1 recursion -> i = 1,

                // char c = '[', i++ -> i = 2, skip '['

                // Then go into level 2 recursion, i keep = 2 (even hit for loop

                // logic, but not hit i++ on for loop yet), char c = 'a'

                // Then for loop i++ on level 2 recursion -> i = 3,

                // char c = ']', break, Important !! -> no i++ here

                // Finish level 2 recursion and move back to level 1, i keep as 3

                // Then for loop i++ on level 1 recursion -> i = 4,

                // char c = '2', then continue similar logic on level 1 recursion

                // ... etc.

                // Note: we don't need i++ to skip ']', it auto skipped when move

                // back to parent level

                //i++;

                break;

            } else {

                sb.append(c);

            }

            i++;

        }

        return sb.toString();

    }

}

**Refer to**

<https://grandyang.com/leetcode/394/>

这道题让我们把一个按一定规则编码后的字符串解码成其原来的模样，编码的方法很简单，就是把重复的字符串放在一个括号里，把重复的次数放在括号的前面，注意括号里面有可能会嵌套括号，这题可以用递归和迭代两种方法来解，我们首先来看递归的解法，把一个括号中的所有内容看做一个整体，一次递归函数返回一对括号中解码后的字符串。给定的编码字符串实际上只有四种字符，数字，字母，左括号，和右括号。那么我们开始用一个变量i从0开始遍历到字符串的末尾，由于左括号都是跟在数字后面，所以首先遇到的字符只能是数字或者字母，如果是字母，直接存入结果中，如果是数字，循环读入所有的数字，并正确转换，那么下一位非数字的字符一定是左括号，指针右移跳过左括号，对之后的内容调用递归函数求解，注意我们循环的停止条件是遍历到末尾和遇到右括号，由于递归调用的函数返回了子括号里解码后的字符串，而我们之前把次数也已经求出来了，那么循环添加到结果中即可，参见代码如下：

    class Solution {

        public:

        string decodeString(string s) {

            int i = 0;

            return decode(s, i);

        }

        string decode(string s, int& i) {

            string res = "";

            int n = s.size();

            while (i < n && s[i] != ']') {

                if (s[i] < '0' || s[i] > '9') {

                    res += s[i++];

                } else {

                    int cnt = 0;

                    while (s[i] >= '0' && s[i] <= '9') {

                        cnt = cnt \* 10 + s[i++] - '0';

                    }

                    ++i;

                    string t = decode(s, i);

                    ++i;

                    while (cnt-- > 0) {

                        res += t;

                    }

                }

            }

            return res;

        }

    };

<https://leetcode.com/problems/decode-string/solutions/738090/c-recursion-and-short/>

Four cases are considered. 1st when we encounter a [. 2nd when we encounter a digit. 3rd encountering a ]. and last simple words.

**Please note that index i is passed by reference not by value.**

class Solution {

public:

    string helper(int &i,string s){

        int num=0;

        string word="";

        for(;i<s.length();i++)

        {

            if(s[i]>='0' and s[i]<='9')

                num=num\*10+s[i]-'0';

            else if(s[i]=='[')

            {

                string ans=helper(++i,s);

                for(;num>0;num--)word+=ans;

            }

            else if(s[i]==']')

                return word;

            else

                word+=s[i];

        }

        return word;

    }

    string decodeString(string s) {

        int i=0;

        return helper(i,s);

    }

};

<https://leetcode.com/problems/decode-string/solutions/1634874/java-dfs-0ms-easy-understanding/>

The idea is: for the string in [ ] pair, we can recurse it as the source string. The codes go into the lower level at '[', and back to parent at ']'.

**The int pos is a globe parameter to indicate the current index of char in the String s.**

private int pos = 0;

    public String decodeString(String s) {

    int n = s.length(), repeat = 0;

    StringBuilder buf = new StringBuilder();

    while (pos < n) {

    char c = s.charAt(pos);

    if (c >= 'a' && c <= 'z') {

    buf.append(c);

    } else if (c >= '0' && c <= '9') {

    repeat = repeat \* 10 + (c - '0');

    } else if (c == '[') {

    pos++;  // skip the char '['

    String str = decodeString(s);  // pos is diff for each call

    for (int i = 0; i < repeat; i++)

  buf.append(str);

    repeat = 0;  // reset the value for the next input

    } else if (c == ']')

    break;

    pos++;

    }

    return buf.toString();

    }