<https://leetcode.com/problems/number-of-atoms/description/>

Given a string formula representing a chemical formula, return the count of each atom.

The atomic element always starts with an uppercase character, then zero or more lowercase letters, representing the name.

One or more digits representing that element's count may follow if the count is greater than 1. If the count is 1, no digits will follow.

For example, "H2O" and "H2O2" are possible, but "H1O2" is impossible.

Two formulas are concatenated together to produce another formula.

For example, "H2O2He3Mg4" is also a formula.

A formula placed in parentheses, and a count (optionally added) is also a formula.

For example, "(H2O2)" and "(H2O2)3" are formulas.

Return the count of all elements as a string in the following form: the first name (in sorted order), followed by its count (if that count is more than 1), followed by the second name (in sorted order), followed by its count (if that count is more than 1), and so on.

The test cases are generated so that all the values in the output fit in a 32-bit integer.

**Example 1:**

**Input:** formula = "H2O"

**Output:** "H2O"

**Explanation:** The count of elements are {'H': 2, 'O': 1}.

**Example 2:**

**Input:** formula = "Mg(OH)2"

**Output:** "H2MgO2"

**Explanation:** The count of elements are {'H': 2, 'Mg': 1, 'O': 2}.

**Example 3:**

**Input:** formula = "K4(ON(SO3)2)2"

**Output:** "K4N2O14S4"

**Explanation:** The count of elements are {'K': 4, 'N': 2, 'O': 14, 'S': 4}.

**Constraints:**

1 <= formula.length <= 1000

formula consists of English letters, digits, '(', and ')'.

formula is always valid.

**Attempt 1: 2023-02-25**

**Solution 1: Stack + HashMap (120 min)**

**这个解法是典型的应用L394.Decode String模版的做法，首先创建一个stack来总体处理，然后对字符串每个字符的处理基本分为三种情况处理：**

**1.遇到正括号：每次遇到一个正括号代表即将展开对一段全新的对括号包裹下的字符串的处理，既然是全新处理，就必须在stack中压入一个用于处理的全新map，注意，每遇到一次正括号就是必须是全新的一个map用于全新的处理，类似recursion递归那么个意思**

**2.遇到反括号：完成对一段括号包裹下的字符串的终结处理，注意不仅要包含对当前一段的终结处理，还要利用stack的性质，比如stack.pop()把前一段的处理结果和当前一段的终结处理结果有效链接起来，反映在当前map和stack.pop()出来的前一段的map按要求融合**

**3.非括号：当前一段括号包裹下的字符串的处理，反映在map上**

**Style 1: HashMap**

class Solution {

    public String countOfAtoms(String formula) {

        Stack<Map<String, Integer>> stack = new Stack<>();

        Map<String, Integer> map = new HashMap<>();

        int n = formula.length();

        int i = 0;

        while(i < n) {

            char c = formula.charAt(i);

            if(c == '(') {

                stack.push(map);

                map = new HashMap<>();

                i++;

            } else if(c == ')') {

                i++;

                int val = 0;

                while(i < n && Character.isDigit(formula.charAt(i))) {

                    val = val \* 10 + (formula.charAt(i) - '0');

                    i++;

                }

                val = val == 0 ? 1 : val;

                if(!stack.isEmpty()) {

                    Map<String, Integer> temp = map;

                    map = stack.pop();

                    for(String atom : temp.keySet()) {

                        map.put(atom, temp.get(atom) \* val + map.getOrDefault(atom, 0));

                    }

                }

            } else {

                int j = i + 1;

                while(j < n && Character.isLowerCase(formula.charAt(j))) {

                    j++;

                }

                String atom = formula.substring(i, j);

                int val = 0;

                while(j < n && Character.isDigit(formula.charAt(j))) {

                    val = val \* 10 + (formula.charAt(j) - '0');

                    j++;

                }

                val = val == 0 ? 1 : val;

                map.put(atom, map.getOrDefault(atom, 0) + val);

                i = j;

            }

        }

        StringBuilder sb = new StringBuilder();

        List<String> list = new ArrayList<>(map.keySet());

        Collections.sort(list);

        for(String key : list) {

            sb.append(key);

            if(map.get(key) > 1) {

                sb.append(map.get(key));

            }

        }

        return sb.toString();

    }

}

Time Complexity: O(N)

Space Complexity: O(N)

**Style 2: TreeMap**

class Solution {

    public String countOfAtoms(String formula) {

        Stack<TreeMap<String, Integer>> stack = new Stack<>();

        TreeMap<String, Integer> map = new TreeMap<>();

        int i = 0;

        int n = formula.length();

        while(i < n) {

            char c = formula.charAt(i);

            if(c == '(') {

                stack.push(map);

                map = new TreeMap<>();

                i++;

            } else if(c == ')') {

                i++;

                int val = 0;

                while(i < n && Character.isDigit(formula.charAt(i))) {

                    val = val \* 10 + formula.charAt(i) - '0';

                    i++;

                }

                val = val == 0 ? 1 : val;

                if(!stack.isEmpty()) {

                    TreeMap<String, Integer> temp = map;

                    map = stack.pop();

                    for(String atom : temp.keySet()) {

                        map.put(atom, temp.get(atom) \* val + map.getOrDefault(atom, 0));

                    }

                }

            } else {

                int j = i + 1;

                while(j < n && Character.isLowerCase(formula.charAt(j))) {

                    j++;

                }

                String atom = formula.substring(i, j);

                int val = 0;

                while(j < n && Character.isDigit(formula.charAt(j))) {

                    val = val \* 10 + formula.charAt(j) - '0';

                    j++;

                }

                val = val == 0 ? 1 : val;

                map.put(atom, map.getOrDefault(atom, 0) + val);

                i = j;

            }

        }

        StringBuilder sb = new StringBuilder();

        for(String atom : map.keySet()) {

            sb.append(atom);

            if(map.get(atom) > 1) {

                sb.append(map.get(atom));

            }

        }

        return sb.toString();

    }

}

Time Complexity: O(N)

Space Complexity: O(N)

**Refer to**

<https://leetcode.com/problems/number-of-atoms/solutions/109345/java-solution-using-stack-and-map/>

class Solution {

public String countOfAtoms(String formula) {

Stack<Map<String,Integer>> stack= new Stack<>();

Map<String,Integer> map= new HashMap<>();

int i=0,n=formula.length();

while(i<n){

char c=formula.charAt(i);i++;

if(c=='('){

stack.push(map);

map=new HashMap<>();

}

else if(c==')'){

int val=0;

while(i<n && Character.isDigit(formula.charAt(i)))

val=val\*10+formula.charAt(i++)-'0';

if(val==0) val=1;

if(!stack.isEmpty()){

Map<String,Integer> temp= map;

map=stack.pop();

for(String key: temp.keySet())

map.put(key,map.getOrDefault(key,0)+temp.get(key)\*val);

}

}

else{

int start=i-1;

while(i<n && Character.isLowerCase(formula.charAt(i))){

i++;

}

String s= formula.substring(start,i);

int val=0;

while(i<n && Character.isDigit(formula.charAt(i))) val=val\*10+ formula.charAt(i++)-'0';

if(val==0) val=1;

map.put(s,map.getOrDefault(s,0)+val);

}

}

StringBuilder sb= new StringBuilder();

List<String> list= new ArrayList<>(map.keySet());

Collections.sort(list);

for(String key: list){

sb.append(key);

if(map.get(key)>1) sb.append(map.get(key));

}

return sb.toString();

}

}

Using TreeMap to avoid sorting at last, same complexity AFAIK. Just for your reference:

<https://leetcode.com/problems/number-of-atoms/solutions/109345/java-solution-using-stack-and-map/comments/264534>

class Solution {

public String countOfAtoms(String formula) {

TreeMap<String, Integer> map = new TreeMap<>();

Stack<TreeMap> stack = new Stack<>();

int i = 0, len = formula.length();

while (i < len) {

if (formula.charAt(i) == '(') {

stack.push(map);

map = new TreeMap<>();

i++;

} else if (formula.charAt(i) == ')') {

int val = 0;

i++;

while (i < formula.length() && Character.isDigit(formula.charAt(i))) {

val = val \* 10 + formula.charAt(i++) - '0';

}

val = val == 0 ? 1 : val;

if (!stack.isEmpty()) {

TreeMap<String, Integer> temp = map;

map = stack.pop();

for (String atom : temp.keySet()) {

map.put(atom, temp.get(atom) \* val + map.getOrDefault(atom, 0));

}

}

} else {

int j = i + 1;

while (j < formula.length() && Character.isLowerCase(formula.charAt(j))) {

j++;

}

String atom = formula.substring(i, j);

int val = 0;

while (j < formula.length() && Character.isDigit(formula.charAt(j))) {

val = val \* 10 + (formula.charAt(j++) - '0');

}

val = val == 0 ? 1 : val;

map.put(atom, map.getOrDefault(atom, 0) + val);

i = j;

}

}

StringBuilder sb = new StringBuilder();

for (String atom : map.keySet()) {

sb.append(atom);

sb.append(map.get(atom) == 1 ? "" : map.get(atom));

}

return sb.toString();

}

}

**Solution 2: Stack + Recursion (120 min)**

**这里的递归模式和L394.Decode String中的写法完全一致，就是说对于'(', ')' 和普通字符的分类处理非常类似**

class Solution {

    int index = 0;

    public String countOfAtoms(String formula) {

        TreeMap<String, Integer> result = helper(formula);

        StringBuilder sb = new StringBuilder();

        for(String atom : result.keySet()) {

            sb.append(atom);

            if(result.get(atom) > 1) {

                sb.append(result.get(atom));

            }

        }

        return sb.toString();

    }

    private TreeMap<String, Integer> helper(String formula) {

        TreeMap<String, Integer> map = new TreeMap<>();

        int n = formula.length();

        while(index < n) {

            char c = formula.charAt(index);

            if(c == '(') {

                index++;

                TreeMap<String, Integer> temp = helper(formula);

                int val = 0;

                while(index < n && Character.isDigit(formula.charAt(index))) {

                    val = val \* 10 + formula.charAt(index) - '0';

                    index++;

                }

                val = val == 0 ? 1 : val;

                for(String atom : temp.keySet()) {

                    map.put(atom, temp.get(atom) \* val + map.getOrDefault(atom, 0));

                }

            } else if(c == ')') {

                index++;

                return map;

            } else {

                int j = index + 1;

                while(j < n && Character.isLowerCase(formula.charAt(j))) {

                    j++;

                }

                String atom = formula.substring(index, j);

                int val = 0;

                while(j < n && Character.isDigit(formula.charAt(j))) {

                    val = val \* 10 + formula.charAt(j) - '0';

                    j++;

                }

                val = val == 0 ? 1 : val;

                map.put(atom, map.getOrDefault(atom, 0) + val);

                index = j;

            }

        }

        return map;

    }

}

Time Complexity: O(N)

Space Complexity: O(N)

**Refer to**

<https://leetcode.com/problems/number-of-atoms/solutions/2028413/java-treemap-easy-recursion/>

class Solution {

private int index;

public String countOfAtoms(String formula) {

if (formula.length() <= 1) {

return formula;

}

index = 0;

Map<String, Integer> countMap = helper(formula.toCharArray());

StringBuilder answer = new StringBuilder();

for (Map.Entry<String, Integer> entry: countMap.entrySet()) {

answer.append(entry.getKey());

if(entry.getValue() > 1) {

answer.append(entry.getValue());

}

}

return answer.toString();

}

private Map<String, Integer> helper(char[] formula) {

Map<String, Integer> result = new TreeMap<>();

while(index < formula.length) {

if(formula[index] == '(') {

index++;

Map<String, Integer> subResult = helper(formula);

int count = getCount(formula);

for(Map.Entry<String, Integer> entry: subResult.entrySet()) {

result.put(entry.getKey(),

result.getOrDefault(entry.getKey(), 0) + entry.getValue() \* count);

}

} else if (formula[index] == ')') {

index++;

return result;

} else {

String name = getName(formula);

result.put(name, result.getOrDefault(name, 0) + getCount(formula));

}

}

return result;

}

private String getName(char[] formula) {

StringBuilder name = new StringBuilder().append(formula[index++]);

while (index < formula.length && Character.isLowerCase(formula[index])) {

name.append(formula[index++]);

}

return name.toString();

}

private int getCount(char[] formula) {

StringBuilder number = new StringBuilder();

while (index < formula.length && Character.isDigit(formula[index])) {

number.append(formula[index++]);

}

return number.length() == 0 ? 1 : Integer.valueOf(number.toString());

}

}

**Refer to**

[L394.Decode String](note://10BBA6EDBB7C4E2A98219E94D95AECAF)