Number of Atoms

Given a chemical `formula` (given as a string), return the count of each atom.

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

1 or more digits representing the count of that element 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 concatenated together 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.

Given a formula, output 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.

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}.
```

Note:

- All atom names consist of lowercase letters, except for the first character which is uppercase.
- The length of `formula` will be in the range `[1, 1000]`.
- `formula` will only consist of letters, digits, and round parentheses, and is a valid formula as defined in the problem.

Solution 1

```
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+ form
ula.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();
    }
}
```

written by kay_deep original link here

Solution 2

```
/**
 * formula : unit[]
 * unit : atom (digits)
 *    | '(' formula ')' digits
 * atom : upper lower[]
 * upper : '[A-Z]'+
 * lower : '[a-z]'+
 * digits : '[0-9]'+
 */
```

```
class Solution {
public:
    string countOfAtoms(string formula) {
        string output;
        const int n = formula.size();
        int i = 0;
        map<string, int> counts = parseFormula(formula, i);
        for (pair<string, int> p : counts) {
            output += p.first;
            if (p.second > 1) output += to_string(p.second);
        }
        return output;
    }

private:
    map<string, int> parseFormula(string& s, int& i) {
        map<string, int> counts;
        const int n = s.size();
        while (i < n && s[i] != ')') {
            map<string, int> cnts = parseUnit(s, i);
            merge(counts, cnts, 1);
        }
        return counts;
    }

    map<string, int> parseUnit(string& s, int& i) {
        map<string, int> counts;
        const int n = s.size();
        if (s[i] == '(') {
            map<string, int> cnts = parseFormula(s, ++i); // ++i for '('
            int digits = parseDigits(s, ++i); // ++i for ')'
            merge(counts, cnts, digits);
        }
        else {
            int i0 = i++; // first letter
            while (i < n && islower(s[i])) { i++; }
            string atom = s.substr(i0, i - i0);
            int digits = parseDigits(s, i);
            counts[atom] += digits;
        }
        return counts;
    }

    int parseDigits(string& s, int& i) {
        int i1 = i;
        while (i < s.size() && isdigit(s[i])) { i++; }
        int digits = i == i1 ? 1 : stoi(s.substr(i1, i - i1));
        return digits;
    }

    void merge(map<string, int>& counts, map<string, int>& cnts, int times) {
        for (pair<string, int> p : cnts) counts[p.first] += p.second * times;
    }
};
```

written by alexander original link here

Solution 3

The first regex `([A-Z]{1}[a-z]?|\(|\)|\d+)` splits up the input string into a few kinds of tokens for parsing: (1) An atom (2) A number (3) Open bracket (4) Closing bracket. These are the only tokens we need to do our parsing.

An input of `Mg(OH)2` will be tokenized into: `['Mg', '(', 'O', 'H', ')', '2']`. An input of `K4(ON(SO3)2)2` will be tokenized into: `['K', '4', '(', 'O', 'N', '(', 'S', 'O', '3', ')', '2', ')', '2']`.

As we iterate through the tokens, there are three cases that we need to handle:

1. Open bracket - We push a new dictionary onto a stack to keep track of the atoms and its count in this current group
2. Close bracket - The next token might be a number/count. Check whether if it is a count. If it is, multiply all the atoms at the top of the stack by the count and combine it with a dictionary below it in the stack.
3. Normal atom - The next token might be a number/count. Check whether if it is a count. If it is, add that atom and its count to the top of the stack.

Cases 2 and 3 are very similar, so we can combine them.

At the end, sort the atoms alphabetically and format them nicely to be returned.

- *Yangshun*

```
class Solution(object):
    def countOfAtoms(self, formula):
        import re
        from collections import defaultdict
        tokens = list(filter(lambda c: c, re.split('([A-Z]{1}[a-z]?|\(|\)|\d+)', for
mula)))
        stack, i = [defaultdict(int)], 0
        while i < len(tokens):
            token = tokens[i]
            if token == '(':
                stack.append(defaultdict(int))
            else:
                count = 1
                # Check if next token is a number.
                if i + 1 < len(tokens) and re.search('^\d+$', tokens[i + 1]):
                    count, i = int(tokens[i + 1]), i + 1
                atoms = stack.pop() if token == ')' else { token: 1 }
                # Combine counts of atoms.
                for atom in atoms:
                    stack[-1][atom] += atoms[atom] * count
            i += 1
        return ''.join([atom + (str(count) if count > 1 else '') for atom, count in
sorted(stack[-1].items())])
```

written by yangshun original link here