Problem 770: Basic Calculator IV

Problem Information

Difficulty: Hard
Acceptance Rate: 49.54%
Paid Only: No
Tags: Hash Table, Math, String, Stack, Recursion

Problem Description

Given an expression such as `expression = "e + 8 - a + 5"` and an evaluation map such as `{"e": 1}` (given in terms of `evalvars = ["e"]` and `evalints = [1]`), return a list of tokens representing the simplified expression, such as `["-1*a","14"]`

* An expression alternates chunks and symbols, with a space separating each chunk and symbol. * A chunk is either an expression in parentheses, a variable, or a non-negative integer. * A variable is a string of lowercase letters (not including digits.) Note that variables can be multiple letters, and note that variables never have a leading coefficient or unary operator like `"2x"` or `"-x"`.

Expressions are evaluated in the usual order: brackets first, then multiplication, then addition and subtraction.

* For example, `expression = "1 + 2 * 3"` has an answer of `["7"]`.

The format of the output is as follows:

* For each term of free variables with a non-zero coefficient, we write the free variables within a term in sorted order lexicographically. * For example, we would never write a term like `"b*a*c"`, only `"a*b*c"`. * Terms have degrees equal to the number of free variables being multiplied, counting multiplicity. We write the largest degree terms of our answer first, breaking ties by lexicographic order ignoring the leading coefficient of the term. * For example, `"a*a*b*c"` has degree `4`. * The leading coefficient of the term is placed directly to the left with an asterisk separating it from the variables (if they exist.) A leading coefficient of 1 is still printed. * An example of a well-formatted answer is `["-2*a*a*a", "3*a*a*b", "3*b*b", "4*a", "5*c", "-6"]`. * Terms (including constant terms) with coefficient `0` are not included. * For example, an expression of `"0"` has an output of `[]`.

Note: You may assume that the given expression is always valid. All intermediate results will be in the range of `[-231, 231 - 1]`.

Example 1:

Input: expression = "e + 8 - a + 5", evalvars = ["e"], evalints = [1] **Output:** ["-1*a","14"]

Example 2:

Input: expression = "e - 8 + temperature - pressure", evalvars = ["e", "temperature"], evalints = [1, 12] **Output:** ["-1*pressure","5"]

Example 3:

Input: expression = "(e + 8) * (e - 8)", evalvars = [], evalints = [] **Output:** ["1*e*e","-64"]

Constraints:

* `1 <= expression.length <= 250` * `expression` consists of lowercase English letters, digits, `'+'`, `'-'`, `'*'`, `'('`, `')'`, `' '`. * `expression` does not contain any leading or trailing spaces. * All the tokens in `expression` are separated by a single space. * `0 <= evalvars.length <= 100` * `1 <= evalvars[i].length <= 20` * `evalvars[i]` consists of lowercase English letters. * `evalints.length == evalvars.length` * `-100 <= evalints[i] <= 100`

Code Snippets

C++:

```
class Solution {
public:
vector<string> basicCalculatorIV(string expression, vector<string>& evalvars,
vector<int>& evalints) {

}
};
```

Java:

```
class Solution {
public List<String> basicCalculatorIV(String expression, String[] evalvars,
int[] evalints) {

}
}
```

Python3:

```
class Solution:
def basicCalculatorIV(self, expression: str, evalvars: List[str], evalints:
List[int]) -> List[str]:
```