<https://leetcode.com/problems/basic-calculator/>

Given a string s representing a valid expression, implement a basic calculator to evaluate it, and return *the result of the evaluation*.

**Note:** You are **not** allowed to use any built-in function which evaluates strings as mathematical expressions, such as eval().

**Example 1:**

Input: s = "1 + 1"

Output: 2

**Example 2:**

Input: s = " 2-1 + 2 "

Output: 3

**Example 3:**

Input: s = "(1+(4+5+2)-3)+(6+8)"

Output: 23

**Constraints:**

* 1 <= s.length <= 3 \* 105
* s consists of digits, '+', '-', '(', ')', and ' '.
* s represents a valid expression.
* '+' is **not** used as a unary operation (i.e., "+1" and "+(2 + 3)" is invalid).
* '-' could be used as a unary operation (i.e., "-1" and "-(2 + 3)" is valid).
* There will be no two consecutive operators in the input.
* Every number and running calculation will fit in a signed 32-bit integer.

**Attempt 1: 2022-11-26**

**Solution 1:  Stack**

**Style 1: Stack store both number and sign for current brace [当前所属括号的符号] (120 min)**

class Solution {

public int calculate(String s) {

int result = 0;

int sign = 1;

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

s = s.replaceAll(" ", "");

int i = 0;

while(i < s.length()) {

char c = s.charAt(i);

if(Character.isDigit(c)) {

int num = c - '0';

while(i + 1 < s.length() && Character.isDigit(s.charAt(i + 1))) {

num = num \* 10 + s.charAt(i + 1) - '0';

i++;

}

result += num \* sign;

} else if(c == '+') {

sign = 1;

} else if(c == '-') {

sign = -1;

} else if(c == '(') {

stack.push(result);

stack.push(sign);

result = 0;

sign = 1;

} else if(c == ')') {

result = result \* stack.pop() + stack.pop();

}

i++;

}

return result;

}

}

**Refer to**

<https://leetcode.com/problems/basic-calculator/discuss/62362/JAVA-Easy-Version-To-Understand>!!!!!

**Style 2: Stack only store sign for current brace [当前所属括号的符号] (120 min)**

class Solution {

public int calculate(String s) {

int result = 0;

int sign = 1;

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

stack.push(sign);

s = s.replaceAll(" ", "");

int i = 0;

while(i < s.length()) {

char c = s.charAt(i);

if(Character.isDigit(c)) {

int num = c - '0';

while(i + 1 < s.length() && Character.isDigit(s.charAt(i + 1))) {

num = num \* 10 + s.charAt(i + 1) - '0';

i++;

}

result += num \* sign \* stack.peek();

} else if(c == '+') {

sign = 1;

} else if(c == '-') {

sign = -1;

} else if(c == '(') {

//stack.push(result);

stack.push(sign \* stack.peek());

//result = 0;

sign = 1;

} else if(c == ')') {

//result = result \* stack.pop() + stack.pop();

stack.pop();

}

i++;

}

return result;

}

}

**Refer to**

<https://segmentfault.com/a/1190000003796804>

**栈法**

**复杂度**

时间 O(N) 空间 O(N)

**思路**

很多人将该题转换为后缀表达式后（逆波兰表达式）求解，其实不用那么复杂。题目条件说明只有加减法和括号，由于加减法是相同顺序的，我们大可以直接把所有数顺序计算。难点在于多了括号后如何处理正负号。我们想象一下如果没有括号这题该怎们做：因为只有加减号，我们可以用一个变量sign来记录上一次的符号是加还是减，这样把每次读到的数字乘以这个sign就可以加到总的结果中了。有了括号后，整个括号内的东西可一看成一个东西，这些括号内的东西都会受到括号所在区域内的正负号影响（比如括号前面是个负号，然后括号所属的括号前面也是个负号，那该括号的符号就是正号）。但是每多一个括号，都要记录下这个括号所属的正负号，而每当一个括号结束，我们还要知道出来以后所在的括号所属的正负号。根据这个性质，我们可以使用一个栈，来记录这些括号所属的正负号。这样我们每遇到一个数，都可以根据当前符号，和所属括号的符号，计算其真实值。

**注意**

先用String.replace()去掉所有的空格

**代码**

public class Solution {

public int calculate(String s) {

// 去掉所有空格

s = s.replace(" ", "");

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

// 先压入一个1进栈，可以理解为有个大括号在最外面

stk.push(1);

int i = 0, res = 0, sign = 1;

while(i < s.length()){

char c = s.charAt(i);

// 遇到正号，将当前的符号变为正号

if(c=='+'){

sign = 1;

i++;

// 遇到负号，将当前的符号变为负号

} else if(c=='-'){

sign = -1;

i++;

// 遇到左括号，计算当前所属的符号，压入栈中

// 计算方法是当前符号乘以当前所属括号的符号

} else if(c=='('){

stk.push(sign \* stk.peek());

sign = 1;

i++;

// 遇到右括号，当前括号结束，[当前所属括号的符号]出栈

} else if(c==')'){

stk.pop();

i++;

// 遇到数字，计算其正负号并加入总结果中

} else {

int num = 0;

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

num = num \* 10 + s.charAt(i) - '0';

i++;

}

res += num \* sign \* stk.peek();

}

}

return res;

}

}