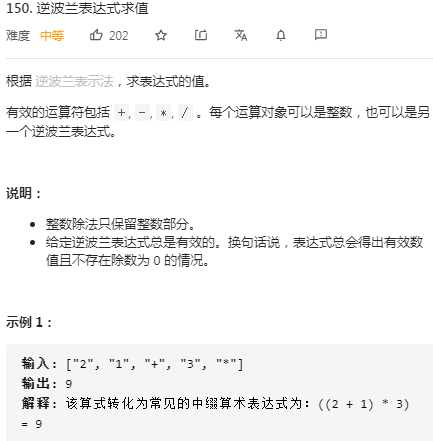
# 一：栈的概述

栈先进先出。

# 二：栈的应用

## 逆波兰表达式求值【leetcode-150】



一般的表达式称为中缀表达式，及操作符在中间，如a+b,逆波兰表达式也叫后缀表达式，如a b + .

基本思想:

遍历后缀表达式，将碰到的数字字符入栈，每次碰到操作符都将弹出2个操作数进行运算将结果压入栈中。最后将栈中元素弹出即为结果。知得注意的是当碰到操作符“-”或者“/”时，后出栈的操作数是被减数或者被除数。

代码：

#define STACK\_WIDTH\_SIZE 20

#define STACK\_HEIGHT\_SIZE 10000

typedef struct tokenStack {

    char tokenStack[STACK\_HEIGHT\_SIZE][STACK\_WIDTH\_SIZE];

    int top;

}TokenStack\_T;

void TokenStackInit(TokenStack\_T\* stack)

{

    stack->top = -1;

    memset(stack->tokenStack, 0, sizeof(char) \* STACK\_WIDTH\_SIZE \* STACK\_HEIGHT\_SIZE);

}

int TokenStackIsFull(TokenStack\_T\* stack)

{

    return stack->top == STACK\_HEIGHT\_SIZE ? 1 : 0;

}

int TokenStackIsEmpty(TokenStack\_T\* stack)

{

    return stack->top == -1 ? 1 : 0;

}

char\* TokenPop(TokenStack\_T\* stack)

{

    if (TokenStackIsEmpty(stack) != 1) {

        return stack->tokenStack[stack->top--];

    }

    return NULL;

}

void TokenPush(TokenStack\_T\* stack, char\* str)

{

    if (TokenStackIsFull(stack) != 1) {

        stack->top++;

        memcpy(&stack->tokenStack[stack->top], str, strlen(str) + 1);

    }

}

int TokenIsDigtal(char\* str)

{

    if (strcmp("+", str) == 0 || strcmp("-", str) == 0 || strcmp("\*", str) == 0 || strcmp("/", str) == 0 ||

        strcmp("(", str) == 0 || strcmp(")", str) == 0) {

            return 0;

    }

    return 1;

}

// ["2", "1", "+", "3", "\*"]

int evalRPN(char \*\* tokens, int tokensSize)

{

    int tempRet, i;

    int input1, input2;

    TokenStack\_T stack = {0};

    TokenStackInit(&stack);

    char buffer[STACK\_WIDTH\_SIZE] = {0};

    if (tokens == NULL || \*tokens == NULL || tokensSize == 0) {

        return 0;

    }

    for (i = 0; i < tokensSize; i++) {

        if (TokenIsDigtal(tokens[i]) == 1) {

            TokenPush(&stack, tokens[i]);

        } else {

            input1 = atoi(TokenPop(&stack));

            input2 = atoi(TokenPop(&stack));

            switch (tokens[i][0]) {

                case '+':

                    sprintf(buffer, "%d", input2 + input1);

                    TokenPush(&stack, buffer);

                    break;

                case '-':

                    sprintf(buffer, "%d", input2 - input1); // 这个一定是后出栈的是被减数

                    TokenPush(&stack, buffer);

                    break;

                case '\*':

                    sprintf(buffer, "%d", input2 \* input1);

                    TokenPush(&stack, buffer);

                    break;

                case '/':

                    sprintf(buffer, "%d", input2 / input1);// 这个一定是后出栈的是被除数

                    TokenPush(&stack, buffer);

                    break;

                default:

                    break;

            }

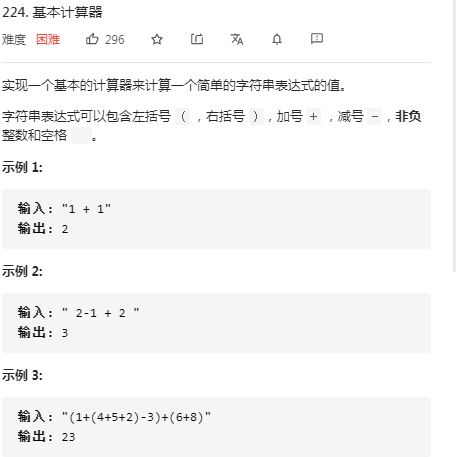
        }

    }

    return atoi(TokenPop(&stack));

}

## 基本计算器【leetcode-224，处理字符数字+递归标记解决括号】



基本思想：

第一个想法是可以将中缀表达式转化成后缀表达式，然后计算，当时比较麻烦。

首先考虑没有括号的时候，并将操作数都看成带上符号如+5, -3 等，这样计算结果的时候统统可以直接累加。比如5-4 看成+5 + -4,当每次碰到新的符号时就将当前的num入栈。

int Step(char \*s){

    int num = 0, i;

    int tmpret, sum;

    int len = strlen(s);

    char opData = '+'; // 初始化为'+'

    char buffer[STACK\_WIDTH\_SIZE] = {0};

    TokenStack\_T stack = {0};

    char c;

    TokenStackInit(&stack);

    for (i = 0; i < len; i++) {

        c = s[i];

        s[i] = ' '; // 处理完用空格标记 主要是因为碰到左括号递归，避免当递归返回，又会继续处理同个字符

        if (isdigit(c) != 0) {

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

        }

        if (c == '(') { // 碰到左括号进行递归

            num = Step(&s[i + 1]);

            continue;

        }

        // 碰到下一个数字或者结束,将数字入栈

        if ((isdigit(c) == 0 && c != ' ') || i == len - 1) {

            switch (opData) {

                case '+':

                    sprintf(buffer, "%d", num);

                    TokenStackPush(&stack, buffer);

                    break;

                case '-':

                    sprintf(buffer, "%d", -num);

                    TokenStackPush(&stack, buffer);

                    break;

                case '\*': // 碰到乘除直接计算完入栈

                    tmpret = atoi(TokenStackPop(&stack));

                    sprintf(buffer, "%d", tmpret \* num);

                    TokenStackPush(&stack, buffer);

                    break;

                case '/':

                    tmpret = atoi(TokenStackPop(&stack));

                    sprintf(buffer, "%d", tmpret / num);

                    TokenStackPush(&stack, buffer);

                    break;

            }

            opData = c;

            num = 0;

        }

// 碰到右括号结束递归

        if (c == ')') {

            break;

        }

    }

    sum = 0;

    while (TokenStackIsEmpty(&stack) != 1) {

        sum += atoi(TokenStackPop(&stack));

    }

    return sum;

}

int calculate(char \* s)

{

    return Step(s);

}