1.复数乘法[哔哩哔哩2020校园招聘]

输入两个表示复数的字符串，输出它们相乘的结果的字符串

复数字符串用a+bi表示(a, b 为整数, i为虚数单位，i^2= -1)

输入描述:

```
两个表示复数的字符串
```

输出描述:

```
两个数相乘的结果的字符串
```

输入例子1:

```
1+2i
2+1
```

输出例子1:

```
0+5i
```

例子说明1:

```
(1+2i)(2+i) = (2 + i + 4i + 2i * i) = 0 + 5i
```

输入例子2:

```
1+-2i
3+4i
```

输出例子2:

```
11+-2i
```

例子说明2:

```
(1+-2i)(3+4i) = (3 + 4i - 6i - 8i * i) = 11+-2i
```

```
/*
复数乘法:  (a + bi)*(c + di) = ac + bd*i^2 + cbi + adi
其中i^2 = -1
*/
#include <string>
#include <iostream>
using namespace std;
```

```cpp
//字符串转整形
int stringtoInt(string s){
    int num = 0;
    int flag = 1;
    for (int i = 0; i < s.size(); i++){
        //如果有负号，说明是负数
        if (s[i] == '-')
            flag = -1;
        //只利用数字字符，i不使用
        else if (s[i] >= '0' && s[i] <= '9')
            num = num * 10 + (s[i] - '0');
    }
    return num * flag;
}

int main(){
    string s1, s2;
    while (cin >> s1 >> s2){
        //截取整数部分
        string s1First = s1.substr(0, s1.find('+'));
        string s2First = s2.substr(0, s2.find('+'));
        //截取复数部分
        //复数1
        string s1Second = s1.substr(s1.find('+') + 1);
        //复数2
        string s2Second = s2.substr(s2.find('+') + 1);
        //把字符串转换成整数
        int s1F = stringtoInt(s1First);
        int s2F = stringtoInt(s2First);
        int s1S = stringtoInt(s1Second);
        int s2S = stringtoInt(s2Second);
        //整数部分
        int F = s1F*s2F - s1S*s2S;
        //复数
        int S = s1F*s2S + s1S*s2F;
        cout << F << "+" << S << "i" << endl;
    }
    return 0;
}

/*java*/
public class Main
{
    public static int stringtoInt(String s){
        int num = 0;
        int flag = 1;
        for (int i = 0; i < s.length(); i++){
            //如果有负号，说明是负数
            if (s.charAt(i) == '-')
                flag = -1;
                //只利用数字字符，i不使用
            else if (s.charAt(i) >= '0' && s.charAt(i) <= '9')
                num = num * 10 + (s.charAt(i) - '0');
```

```
        }
        return num * flag;
    }

    public static void main(String[] args) {
        String s1, s2;
        Scanner scanner = new Scanner(System.in);
        s1 = scanner.nextLine();
        s2 = scanner.nextLine();
        //截取整数部分
        String s1First = s1.substring(0, s1.indexOf('+'));
        String s2First = s2.substring(0, s2.indexOf('+'));
        //截取复数部分
        //复数1
        String s1Second = s1.substring(s1.indexOf('+') + 1);
        //复数2
        String s2Second = s2.substring(s2.indexOf('+') + 1);
        //把字符串转换成整数
        int s1F = stringtoInt(s1First);
        int s2F = stringtoInt(s2First);
        int s1S = stringtoInt(s1Second);
        int s2S = stringtoInt(s2Second);
        //整数部分
        int F = s1F*s2F - s1S*s2S;
        //复数
        int S = s1F*s2S + s1S*s2F;
        System.out.println(F + "+" + S + "i");
    }
}
```

2.一年中的第几天[哔哩哔哩2020校园招聘]

输入一个"YYYY-MM-dd"格式的日期字符串，输出该天是当年的第几天（1 月 1 日是每年的第 1 天）

输入描述:

一个"YYYY-MM-dd"格式的表示日期的字符串

输出描述:

该天是当年的第几天

输入例子1:

2019-01-09

输出例子1:

9

输入例子2:

```
2004-03-01
```

输出例子2:

```
61
```

例子说明2:

```
2004年为闰年，所以是第31+29+1=61天
```

```
/*
此题可以先累加month-1月的天数，然后加上month的天数，如果包含闰年的2月，天数再补一
*/
#include <iostream>
#include <string>
#include <vector>
using namespace std;

int stringToInt(string s){
    int count = 0;
    for (int i = 0; i < s.size(); i++){
        count = count * 10 + (s[i] - '0');
    }
    return count;
}

int main(){
    string s;
    while (cin >> s){
        //用一个数组保存一年中当当前月累加的天数
        static int days[13] = { 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365
};
        vector<string> vec;
        string s1;
        //按照"-"分割年月日
        for (int i = 0; i < s.size(); i++){
            if (s[i] == '-')
            {
                vec.push_back(s1);
                //清空，保存下一部分
                s1 = "";
            }
            else{
                s1 += s[i];
            }
        }
        //保存天
        vec.push_back(s1);
        //转整形

        int year = stringToInt(vec[0]);
```

```
        int month = stringToInt(vec[1]);
        int day = stringToInt(vec[2]);
        int ret = days[month - 1] + day;
        //如果是闰年，并且包含2月，需要补一天
        if (((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) && month > 2)
            ret++;
        cout << ret << endl;
    }
    return 0;
}

/*java*/
public class Main
{
    public static int stringToInt(String s){
        int count = 0;
        for (int i = 0; i < s.length(); i++){
            count = count * 10 + (s.charAt(i) - '0');
        }
        return count;
    }

    public static void main(String[] args) {
        String s;
        Scanner scanner = new Scanner(System.in);
        s = scanner.nextLine();
        //用一个数组保存一年中当当前月累加的天数
        int[] days = { 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 };
        List<String> vec = new ArrayList<>();
        String s1 = "";
        //按照"-"分割年月日
        for (int i = 0; i < s.length(); i++){
            if (s.charAt(i) == '-')
            {
                vec.add(s1);
                //清空，保存下一部分
                s1 = "";
            }
            else{
                s1 += s.charAt(i);
            }
        }
        //保存天
        vec.add(s1);
        //转整形

        int year = stringToInt(vec.get(0));
        int month = stringToInt(vec.get(1));
        int day = stringToInt(vec.get(2));
        int ret = days[month - 1] + day;
        //如果是闰年，并且包含2月，需要补一天
        if (((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) && month > 2)
            ret++;
```

```
        System.out.println(ret);
    }
}
```

3.k个一组翻转链表[哔哩哔哩2020校园招聘]

给你一个链表，每 k 个节点一组进行翻转，请返回翻转后的链表。

如果节点总数不是 k 的整数倍，那么请将最后剩余的节点保持原有顺序。

示例：

给定这个链表：1->2->3->4->5

当 k = 2 时，应当返回: 2->1->4->3->5

当 k = 3 时，应当返回: 3->2->1->4->5

输入描述:

```
第一行：依次输入链表中的各个元素，以"#"结束

第二行：每组数量k
```

输出描述:

```
处理后的链表中的各个元素，以"->"连接
```

输入例子1:

```
1 2 3 4 5 #
2
```

输出例子1:

```
2->1->4->3->5
```

输入例子2:

```
1 2 3 4 5 #
3
```

输出例子2:

```
3->2->1->4->5
```

```
/*
此题可以用字符串保存每个元素，用vector保存所有元素，组内进行元素翻转
*/
#include <iostream>
#include <string>
```

```cpp
#include <vector>
using namespace std;

//注意每个元素的长度可能不为1，所以每个元素用string保存
void reverse(vector<string>& vec, int start, int end) {
    while (start < end) {
        vec[start].swap(vec[end]);
        start++;
        end--;
    }
}

int main() {
    string str;
    vector<string> strVec;
    //输入链表元素
    while (cin >> str) {
        if (str != "#") {
            strVec.push_back(str);
        }
        else {
            break;
        }
    }
    int k;
    cin >> k;
    int start;
    start = 0;
    //如果不够一组，就不进行翻转
    //每k个元素进行翻转
    while (start + k - 1 <= strVec.size() - 1) {
        reverse(strVec, start, start + k - 1);
        //下一组的起始位置
        start = start + k;
    }
    for (int i = 0; i < strVec.size() - 1; i++) {
        cout << strVec[i] << "->";
    }
    cout << strVec.back() << endl;
    return 0;
}

/*java*/
public class Main
{
    public static void reverse(String[] vec, int start, int end) {
        while (start < end) {
            String tmp = vec[start];
            vec[start] = vec[end];
            vec[end] = tmp;
            start++;
            end--;
        }
```

```
    }

    public static void main(String[] args) {
        String str;
        Scanner scanner = new Scanner(System.in);
        str = scanner.nextLine();
        String[] strArr = str.split(" ");
        int k;
        k = scanner.nextInt();
        int start;
        start = 0;
        //如果不够一组，就不进行翻转
        //每k个元素进行翻转
        while (start + k - 1 < strArr.length - 1) {
            reverse(strArr, start, start + k - 1);
            //下一组的起始位置
            start = start + k;
        }
        for (int i = 0; i < strArr.length - 2; i++) {
            System.out.print(strArr[i] + "->");
        }
        System.out.println(strArr[strArr.length - 2]);
    }
}
```

4.递增子序列瓜子二手车2019秋招

判断一个无序数组中是否存在长度为3的递增子序列。（不要求连续）（满足O(n)的时间复杂度和O(1)的空间复杂度。）

输入描述:

```
第一行一个正整数 1 <= n <= 100000

第二行n个整数a1,a2,...,an, (1<=ai<=1e9)
```

输出描述:

```
如果存在，输出"true",否则输出"false"。（不含引号）。
```

输入例子1:

```
5
12 8 36 9 20
```

输出例子1:

```
true
```

```
/*
```

```
此题需要找的递增序列可以不连续，所以可以记录遍历当前位置时的最小的两个值，如果当前位置的值大于第二小的值，则可以说明存在这样的递增序列。这里需要注意，第二小的值必须在第一小的值的后面，如果更新了第一小的值，则也需要更新第二小的值。
*/
#include<iostream>
#include<vector>
#include<climits>
using namespace std;

int main()
{
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++)
        cin >> a[i];
    int first = a[0], second = INT_MAX;
    for (int i = 1; i < n; i++)
    {
        if (a[i] < first)
        {
            first = a[i];
            //如果更新了最小值，则第二小的值需要在后面找，所以此处需要更新
            second = INT_MAX;
        }
        else if (a[i] > first && a[i] < second)
            second = a[i];
        else if (a[i] > first && a[i] > second)
        {
            cout << "true" << endl;
            return 0;
        }
    }
    cout << "false" << endl;
    return 0;
}

/*java*/
public class Main
{

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int[] a = new int[n];
        for (int i = 0; i < n; i++)
            a[i] = scanner.nextInt();
        int first = a[0], second = Integer.MAX_VALUE;
        for (int i = 1; i < n; i++)
        {
            if (a[i] < first)
            {
```

```
                first = a[i];
                //如果更新了最小值，则第二小的值需要在后面找，所以此处需要更新
                second = Integer.MAX_VALUE;
            }
            else if (a[i] > first && a[i] < second)
                second = a[i];
            else if (a[i] > first && a[i] > second)
            {
                System.out.println("true");
                return;
            }
        }
        System.out.println("false");
    }
}
```

5.硬币划分瓜子二手车2019秋招

有1分，2分，5分，10分四种硬币，每种硬币数量无限，给定n分钱(n <= 100000)，有多少中组合可以组成n分钱?

输入描述:

```
输入整数n.(1<=n<=100000)
```

输出描述:

```
输出组合数，答案对1e9+7取模。
```

输入例子1:

```
13
```

输出例子1:

```
16
```

```
/*
此题可以通过动态规划解答
状态F(i,j)：用i种硬币组成j分钱的方法总数
转移方程：
    如果第i个硬币的面值大于j，则只能用i - 1种硬币进行组合
        F(i,j) = F(i - 1, j)
    如果第i个硬币面值小于j，则可以利用,也可以不利于，两种组合产生的方法互斥
        F(i, j) = F(i - 1,  j) + F(i, j - coins[i - 1])
    此处F(i, j - coins[i - 1])表示：第i个硬币可能使用了多次
初始化：
    F(0, j) = 0：没有硬币可以用
    F(i, 0) = 1：不使用任何硬币
返回值
    F(size, n)
*/
```

```cpp
#include<iostream>
#include<vector>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> coins{ 1, 2, 5, 10 };
    int num = coins.size();
    vector<vector<int>> methodNum(num + 1, vector<int>(n + 1, 0));

    for (int i = 0; i <= num; ++i)
        //初始化
        methodNum[i][0] = 1;
    for (int i = 1; i <= coins.size(); ++i) {
        for (int j = 1; j <= n; ++j) {
            if (j >= coins[i - 1]) {
                methodNum[i][j] = (methodNum[i - 1][j] + methodNum[i][j - coins[i - 1]]) %
1000000007;
            }
            else
                methodNum[i][j] = methodNum[i - 1][j];
        }
    }
    cout << methodNum[num][n]<<endl;
}

/*java*/
public class Main
{

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();

        int[] coins = { 1, 2, 5, 10 };
        int num = coins.length;
        int[][] methodNum = new int[num + 1][n + 1];

        for (int i = 0; i <= num; ++i)
            //初始化
            methodNum[i][0] = 1;
        for (int i = 1; i <= coins.length; ++i) {
            for (int j = 1; j <= n; ++j) {
                if (j >= coins[i - 1]) {
                    methodNum[i][j] = (methodNum[i - 1][j] + methodNum[i][j - coins[i -
1]]) % 1000000007;
                }
                else
                    methodNum[i][j] = methodNum[i - 1][j];
            }
        }
```

```
        System.out.println(methodNum[num][n]);
    }
}

//bfs或者dfs 通过率只有40%
#include <vector>
#include <iostream>
#include <queue>
using namespace std;
struct Node
{
    int curSum = 0;
    int prev = 0;

    Node(int cur = 0, int pre = 0)
        :curSum(cur)
        , prev(pre)
    {}
};

int bfs(vector<int>& coins, int target)
{
    queue<Node> q;

    q.push(Node());
    int num = 0;
    while (!q.empty())
    {
        Node cur = q.front();
        q.pop();
        for (int i = cur.prev; i < coins.size(); ++i)
        {
            if (coins[i] > target)
                continue;
            if (cur.curSum + coins[i] >= target)
            {
                if (cur.curSum + coins[i] == target)
                    ++num;
                continue;
            }
            q.push(Node(cur.curSum + coins[i], i));
        }
    }
    return num;
}

int main() {
    int n;
    cin >> n;
    vector<int> coins{ 1, 2, 5, 10 };
    cout << bfs(coins, n) << endl;
    return 0;
}
```

6.合并二叉树[瓜子二手车2019秋招]

已知两颗二叉树，将它们合并成一颗二叉树。合并规则是：都存在的结点，就将结点值加起来，否则空的位置就由另一个树的结点来代替。例如： 两颗二叉树是: Tree 1
1
/ \
3 2 /
5

Tree 2 2 / \ 1 3 \ \ 4 7

合并后的树为 3 / \ 4 5 / \ \ 5 4 7

输入描述:

```
第一行输入整数n，m。(分别表示树1和树2的节点数,1<=n，m<=100)
接下来n行，第i行三个整数l,r,v，(0<=l,r<=n ， 0<=v<=100) ,表示第一棵树的第i个结点的左儿子编号，右儿子编号和权值。
接下来m行，第i行三个整数l,r,v，(0<=l,r<=n ， 0<=v<=100) ,表示第二棵树的第i个结点的左儿子编号，右儿子编号和权值。
（对于左右儿子，如果编号为0表示空。保证如果儿子不为空，则儿子编号大于父亲编号。）
```

输出描述:

```
输出合并后树按层遍历的各结点权值，相邻权值之间以一个空格相间。
```

输入例子1:

```
4 5
2 3 1
4 0 3
0 0 2
0 0 5
2 3 2
0 4 1
0 5 3
0 0 4
0 0 7
```

输出例子1:

```
3 4 5 5 4 7
```

```
/*
此题可以首先通过节点的位置关系建立树，其次合并两棵树，再使用dfs算法层序遍历节点
*/
#include<iostream>
#include<algorithm>
#include<vector>
#include<queue>
using namespace std;
```

```cpp
struct node {
    node* left;
    node* right;
    int value;
    node()
        :left(nullptr)
        ,right(nullptr)
        ,value(0){}
};
//此处孩子的编号从1开始计数
void creatTree(vector<node>& treeVec, int num)
{
    //按照顺序存储树的各节点，并建立相互之间的联系
    for (int i = 1; i <= num; i++)
    {
        int l, r, v;
        cin >> l >> r >> v;
        if (l != 0)
            treeVec[i].left = &treeVec[l];
        if (r != 0)
            treeVec[i].right = &treeVec[r];
        treeVec[i].value = v;
    }
}

//合并树的节点
node* mergeTree(node* root1, node* root2)
{
    if (root1 != nullptr && root2 != nullptr)
    {
        root1->left = mergeTree(root1->left, root2->left);
        root1->right = mergeTree(root1->right, root2->right);
        root1->value += root2->value;
        return root1;
    }
    return root1 == nullptr ? root2 : root1;
}
//层序遍历，使用dfs算法
void leverOrder(node* root)
{
    queue<node*> q;
    q.push(root);
    while (!q.empty())
    {
        node* cur = q.front();
        q.pop();
        cout << cur->value << " ";
        if (cur->left)
            q.push(cur->left);
        if (cur->right)
            q.push(cur->right);
    }
    cout << endl;
```

```
}

int main()
{
    int n, m;
    cin >> n >> m;
    vector<node> treeVec1(n + 1);
    vector<node> treeVec2(m + 1);
    creatTree(treeVec1, n);
    creatTree(treeVec2, m);
    node* root1 = &treeVec1[1];
    node* root2 = &treeVec2[1];
    node* root = mergeTree(root1, root2);
    leverOrder(root);
    return 0;
}

/*java*/
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;
import java.util.Scanner;

class node
{
    int val = 0;
    node left = null;
    node right = null;
}

public class Main
{
    public static void createTree(Scanner scanner, node[] array, int n)
    {
        //Scanner scanner = new Scanner(System.in);
        for(int i = 1; i <= n; ++i)
        {
            int left = scanner.nextInt();
            int right = scanner.nextInt();
            int val = scanner.nextInt();
            if(left != 0)
                array[i].left = array[left];
            if(right != 0)
                array[i].right = array[right];
            array[i].val = val;
        }
    }

    public static node merge(node root1, node root2)
    {
        if(root1 != null && root2 != null)
        {
            root1.left = merge(root1.left, root2.left);
```

```
            root1.right = merge(root1.right, root2.right);
            root1.val = root1.val + root2.val;
            return root1;
        }
        return root1 == null ? root2 : root1;
    }

    public static void BFS(node root)
    {
        Queue<node> q = new LinkedList<>();
        if(root != null)
        {
            q.offer(root);
        }
        while(!q.isEmpty())
        {
            node cur = q.poll();
            System.out.print(cur.val + " ");
            if(cur.left != null)
                q.offer(cur.left);
            if(cur.right != null)
                q.offer(cur.right);
        }
        System.out.println();
    }

    public static void main(String[] args) {
        int n, m;
        Scanner scanner = new Scanner(System.in);
        n = scanner.nextInt();
        m = scanner.nextInt();

        node[] array1 = new node[n + 1];
        node[] array2 = new node[m + 1];

        for(int i = 0; i <= n; ++i)
            array1[i] = new node();
        for(int i = 0; i <= m; ++i)
            array2[i] = new node();

        createTree(scanner,array1, n);
        createTree(scanner,array2, m);
        node root1 = null, root2 = null;
        if(array1.length > 1)
            root1 = array1[1];
        if(array2.length > 1)
            root2 = array2[1];
        node root = merge(root1, root2);
        BFS(root);
    }
}
```

7.求表达式 f(n)结果末尾0的个数小红书2019年校园招聘

输入一个自然数n，求表达式 f(n) = 1! * 2! * 3!.....n! 的结果末尾有几个连续的0?

输入描述:

自然数n

输出描述:

f(n)末尾连续的0的个数

输入例子1:

11

输出例子1:

9

```
/*
此题的要求是求解末尾0的个数，如果要产生0，则需要产生因子10，所以相当于需要计算出10的数量。
但是10不是最小的因子，他还可以分解成2 * 5，所以需要找出2和5的个数，最后取两者中最小的，即为0的个数。
*/
#include <iostream>
#include <algorithm>
using namespace std;
//计算i中包含因子n的个数
int calNumber(int num, int n){
    int cnt = 0;
    while (num >= n && num % n == 0){
        ++cnt;
        num /= n;
    }
    return cnt;
}
//计算所有2和5的个数，然后取最小
int main(){
    int n;
    cin >> n;
    int num2 = 0, num5 = 0;
    for (int i = 1; i <= n; i++){
        //整个式子中包含的数字i共有：n + 1 - i个
        num2 += calNumber(i, 2) * (n + 1 - i);
        num5 += calNumber(i, 5) * (n + 1 - i);
    }
    cout << min(num2, num5) << endl;
    return 0;
}

/*java*/
public class Main
{
```

```
    public static int calNumber(int num, int n){
        int cnt = 0;
        while (num >= n && num % n == 0){
            ++cnt;
            num /= n;
        }
        return cnt;
    }
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();

        int num2 = 0, num5 = 0;
        for (int i = 1; i <= n; i++){
            //整个式子中包含的数字i共有：n + 1 - i个
            num2 += calNumber(i, 2) * (n + 1 - i);
            num5 += calNumber(i, 5) * (n + 1 - i);
        }
        System.out.println(Math.min(num2, num5));
    }
}
```

8.字符串压缩算法小红书2019年校园招聘

输入一串字符，请编写一个字符串压缩程序，将字符串中连续出现的重复字母进行压缩，并输出压缩后的字符串。例如： aac 压缩为 1ac xxxxyyyyyyzbbb 压缩为 3x5yz2b

输入描述:

```
任意长度字符串
```

输出描述:

```
压缩后的字符串
```

输入例子1:

```
xxxxyyyyyyzbbb
```

输出例子1:

```
3x5yz2b
```

```
#include <string>
#include <iostream>
using namespace std;

int main()
{
    string str;
```

```
    getline(cin, str);
    //遍历字符串
    for(int i = 0; i < str.length(); i++)
    {
        //用来记录重复字符数量
        int cnt = 0;
        //判断是不是字符串中的重复字符
        while(str[i] == str[i+1])
        {
            i++;
            cnt++;
        }
        //先输出压缩的字符个数
        if(cnt != 0)
        {
            cout << cnt;
        }
        //再输出被压缩的字符
        cout << str[i];
    }
    return 0;
}

/*java*/
import java.util.*;
public class Main
{
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        char[] str = scanner.nextLine().toCharArray();
        //遍历字符串
        for(int i = 0; i < str.length; i++)
        {
            //用来记录重复字符数量
            int cnt = 0;

            //判断是不是字符串中的重复字符，保证不要越界
            while(i <= str.length - 2 && str[i] == str[i+1])
            {
                i++;
                cnt++;
            }
            //先输出压缩的字符个数
            if(cnt != 0)
            {
                System.out.print(cnt);
            }
            //再输出被压缩的字符
            System.out.print(str[i]);
        }
        System.out.println();
    }
}
```