實作題

邏輯運算子 (Logic Operators)

問題描述

小蘇最近在學三種邏輯運算子 AND、OR 和 XOR。這三種運算子都是二元運算子，也就是說在運算時需要兩個運算元，例如 a AND b。對於整數 a 與 b，以下三個二元運算子的運算結果定義如下列三個表格：

a AND b

	b 為 0	b 不為 0
a 為 0	**0**	**0**
a 不為 0	**0**	**1**

a OR b

	b 為 0	b 不為 0
a 為 0	**0**	**1**
a 不為 0	**1**	**1**

a XOR b

	b 為 0	b 不為 0
a 為 0	**0**	**1**
a 不為 0	**1**	**0**

舉例來說：

(1) 0 AND 0 的結果為 0，0 OR 0 以及 0 XOR 0 的結果也為 0。

(2) 0 AND 3 的結果為 0，0 OR 3 以及 0 XOR 3 的結果則為 1。

(3) 4 AND 9 的結果為 1，4 OR 9 的結果也為 1，但 4 XOR 9 的結果為 0。

請撰寫一個程式，讀入 a、b 以及邏輯運算的結果，輸出可能的邏輯運算為何。

評分說明

輸入包含若干筆測試資料，每一筆測試資料的執行時間限制(time limit)均為 1 秒，依正確通過測資筆數給分。其中：

第 1 子題組 80 分，a 和 b 的值只會是 0 或 1。

第 2 子題組 20 分，$0 \le a, b < 10{,}000$。

輸入格式

輸入只有一行，共三個整數值，整數間以一個空白隔開。第一個整數代表 a，第二個整數代表 b，這兩數均為非負的整數。第三個整數代表邏輯運算的結果，只會是 0 或 1。

輸出格式

輸出可能得到指定結果的運算，若有多個，輸出順序為 AND、OR、XOR，每個可能的運算單獨輸出一行，每行結尾皆有換行。若不可能得到指定結果，輸出 IMPOSSIBLE。（注意輸出時所有英文字母均為大寫字母。）

範例一：輸入

```
0 0 0
```

範例一：正確輸出

```
AND
OR
XOR
```

範例二：輸入

```
1 1 1
```

範例二：正確輸出

```
AND
OR
```

範例三：輸入

```
3 0 1
```

範例三：正確輸出

```
OR
XOR
```

範例四：輸入

```
0 0 1
```

範例四：正確輸出

```
IMPOSSIBLE
```

解題思路

- 讀取輸入 a b c
  - 切割、轉型
  - 將大於0的a b值轉換為1，降低程式的複雜度
- 驗證a與b 以and、or、xor 運算結果是否為c
  - 以boolean變數and_op、or_op、xor_op 儲存
- 運算符號運算結果為result時印出該運算符號
  - 三個運算符號均無法取得該結果列印"IMPOSSIBLE"


10610P1.py - D:/APCS/APCS考古題/10610P1.py (3.8.1)
File Edit Format Run Options Window Help
inArr = input().split() #切割輸入字串
#轉型，並將大於0的a b值轉換為1
a = 1 if(int(inArr[0])>0) else 0
b = 1 if(int(inArr[1])>0) else 0
c = int(inArr[2])
#驗證a與b 以and、or、xor 運算結果是否為c
and_op = True if((a and b)==c) else False
or_op = True if((a or b)==c) else False
xor_op = True if((a != b)==c) else False
#運算符號運算結果為c時印出該運算符號
if(and_op):
print('AND')
if(or_op):
print('OR')
if(xor_op):
print('XOR')
#三個運算符號均無法取得該結果列印"IMPOSSIBLE"
if(not(and_op or or_op or xor_op)):
print('IMPOSSIBLE')
Ln: 23 Col: 0

實作題　交錯字串 (Alternating Strings)

問題描述

一個字串如果全由大寫英文字母組成，我們稱為大寫字串；如果全由小寫字母組成則稱為小寫字串。字串的長度是它所包含字母的個數，在本題中，字串均由大小寫英文字母組成。假設 k 是一個自然數，一個字串被稱為「k-交錯字串」，如果它是由長度為 k 的大寫字串與長度為 k 的小寫字串交錯串接組成。

舉例來說，「StRiNg」是一個 1-交錯字串，因為它是一個大寫一個小寫交替出現；而「heLLow」是一個 2-交錯字串，因為它是兩個小寫接兩個大寫再接兩個小寫。但不管 k 是多少，「aBBaaa」、「BaBaBB」、「aaaAAbbCCCC」都不是 k-交錯字串。

本題的目標是對於給定 k 值，在一個輸入字串找出最長一段連續子字串滿足 k-交錯字串的要求。例如 k=2 且輸入「aBBaaa」，最長的 k-交錯字串是「BBaa」，長度為 4。又如 k=1 且輸入「BaBaBB」，最長的 k-交錯字串是「BaBaB」，長度為 5。

請注意，滿足條件的子字串可能只包含一段小寫或大寫字母而無交替，如範例二。此外，也可能不存在滿足條件的子字串，如範例四。

輸入格式

輸入的第一行是k，第二行是輸入字串，字串長度至少為1，只由大小寫英文字母組成(A~Z, a~z)並且沒有空白。

輸出格式

輸出輸入字串中滿足k-交錯字串的要求的最長一段連續子字串的長度，以換行結尾。

範例一：輸入

```
1
aBBdaaa
```

範例一：正確輸出

```
2
```

範例二：輸入

```
3
DDaasAAbbCC
```

範例二：正確輸出

```
3
```

範例三：輸入

```
2
aafAXbbCDCCC
```

範例三：正確輸出

```
8
```

範例四：輸入

```
3
DDaaAAbbCC
```

範例四：正確輸出

```
0
```

提示：根據定義，要找的答案是大寫片段與小寫片段交錯串接而成。本題有多種解法的思考方式，其中一種是從左往右掃描輸入字串，我們需要紀錄的狀態包含：目前是在小寫子字串中還是大寫子字串中，以及在目前大(小)寫子字串的第幾個位置。根據下一個字母的大小寫，我們需要更新狀態並且記錄以此位置為結尾的最長交替字串長度。

另外一種思考是先掃描一遍字串，找出每一個連續大(小)寫片段的長度並將其記錄在一個陣列，然後針對這個陣列來找出答案。

評分說明：

輸入包含若干筆測試資料，每一筆測試資料的執行時間限制(time limit)均為 1 秒，依正確通過測資筆數給分。其中：

第 1 子題組 20 分，字串長度不超過 20 且 k=1。

第 2 子題組 30 分，字串長度不超過 100 且 $k \leq 2$。

第 3 子題組 50 分，字串長度不超過 100,000 且無其他限制。

解題思路

- 先造一個 list，檢查每一個字母，如果是大寫就給 0，小寫則給 1。例如字串是 ABfADFccTRggg，先造出[0,0,1,0,0,0,1,1,0,0,1,1,1]
- 算出連續大寫與連續小寫片段的長度。上例中可建出[2, 1, 3, 2, 2, 3]
- 最後一步我們在 seg[]中要找連續的 k

- 交錯字串長度K=2
- 字串：ABfADFccTRggg
- 轉換後的陣列：[2, 1, 3, 2, 2, 3]
- 最大交錯字串長度：8

```python
k = int(input())
a = input()
uorl = [] # 01 list
#將字母轉換成數字
for c in a:
    if c>='A' and c<='Z': uorl.append(0)
    elif c>='a' and c<='z': uorl.append(1)
seg = []
my_len = 1
for i in range(1,len(uorl)):
    if uorl[i] == uorl[i -1]: #前後數字一樣相加
        my_len += 1
    else:
        seg.append(my_len)
        my_len = 1
seg.append(my_len)
#print(seg)
longest = 0
m = len(seg)
le = 0
```

轉換後的陣列：[2, 1, 3, 2, 2, 3]

```python
while le<m:
    while le < m and seg[le] != k: #找到連續k字串的開始
        le+=1
    if le >= m:break;
    ri = le + 1 #找到下一格數字是否為K
    while ri < m and seg[ri] == k:
        ri += 1
    t = (ri - le) * k   #算出長度*k 總長度4
    if le > 0 and seg[le-1] > k: #左邊大於k也算在長度內
        t+=k
    if ri < m and seg[ri]  > k: #右邊大於k也算在長度內
        t += k
    if t > longest:longest = t #最大的交錯字串
    le = ri + 1 #找下一個看看
#print(longest)
```

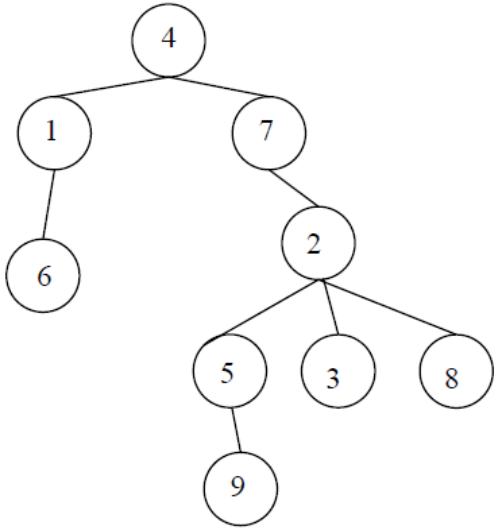
實作題

樹狀圖分析 (Tree Analyses)

問題描述

本題是關於有根樹(rooted tree)。在一棵 n 個節點的有根樹中，每個節點都是以 1~n 的不同數字來編號，描述一棵有根樹必須定義節點與節點之間的親子關係。一棵有根樹恰有一個節點沒有父節點(parent)，此節點被稱為根節點(root)，除了根節點以外的每一個節點都恰有一個父節點，而每個節點被稱為是它父節點的子節點(child)，有些節點沒有子節點，這些節點稱為葉節點(leaf)。在當有根樹只有一個節點時，這個節點既是根節點同時也是葉節點。

在圖形表示上，我們將父節點畫在子節點之上，中間畫一條邊(edge)連結。例如，圖一中表示的是一棵 9 個節點的有根樹，其中，節點 1 為節點 6 的父節點，而節點 6 為節點 1 的子節點；又 5、3 與 8 都是 2 的子節點。節點 4 沒有父節點，所以節點 4 是根節點；而 6、9、3 與 8 都是葉節點。



圖一

樹狀圖中的兩個節點 u 和 v 之間的距離 d(u,v)定義為兩節點之間邊的數量。如圖一中，$d(7, 5) = 2$，而 $d(1, 2) = 3$。對於樹狀圖中的節點 v，我們以 h(v)代表節點 v 的高度，其定義是節點 v 和節點 v 下面最遠的葉節點之間的距離，而葉節點的高度定義為 0。如圖一中，節點 6 的高度為 0，節點 2 的高度為 2，而節點 4 的高度為 4。此外，我們定義 H(T)為 T 中所有節點的高度總和，也就是說 $H(T) = \sum_{v \in T} h(v)$。給定一個樹狀圖 T，請找出 T 的根節點以及高度總和 H(T)。

輸入格式

第一行有一個正整數 n 代表樹狀圖的節點個數，節點的編號為 1 到 n。接下來有 n 行，第 i 行的第一個數字 k 代表節點 i 有 k 個子節點，第 i 行接下來的 k 個數字就是這些子節點的編號。每一行的相鄰數字間以空白隔開。

輸出格式

輸出兩行各含一個整數，第一行是根節點的編號，第二行是 H(T)。

範例一：輸入

```
7
0
2 6 7
2 1 4
0
2 3 2
0
0
```

範例一：正確輸出

```
5
4
```

範例二：輸入

```
9
1 6
3 5 3 8
0
2 1 7
1 9
0
1 2
0
0
```

範例二：正確輸出

```
4
11
```

評分說明：

輸入包含若干筆測試資料，每一筆測試資料的執行時間限制(time limit)均為 1 秒，依正確通過測資筆數給分。測資範圍如下，其中 k 是每個節點的子節點數量上限：

第 1 子題組 10 分，$1 \le n \le 4$, $k \le 3$, 除了根節點之外都是葉節點。

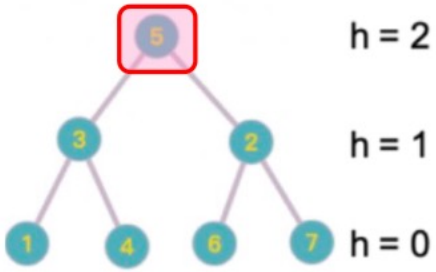
第 2 子題組 30 分，$1 \le n \le 1{,}000$, $k \le 3$。

第 3 子題組 30 分，$1 \le n \le 100{,}000$, $k \le 3$。

第 4 子題組 30 分，$1 \le n \le 100{,}000$, k 無限制。

提示：輸入的資料是給每個節點的子節點有哪些或沒有子節點，因此，可以根據定義找出根節點。關於節點高度的計算，我們根據定義可以找出以下遞迴關係式：(1)葉節點的高度為 0；(2)如果 v 不是葉節點，則 v 的高度是它所有子節點的最大高度加一。也就是說，假設 v 的子節點有 a, b 與 c，則 h(v)=max{ h(a), h(b), h(c) }+1。以遞迴方式可以計算出所有節點的高度。

```
nodes 7
1 0
2 2 6 7
3 2 1 4
4 0
5 2 3 2
6 0
7 0
```



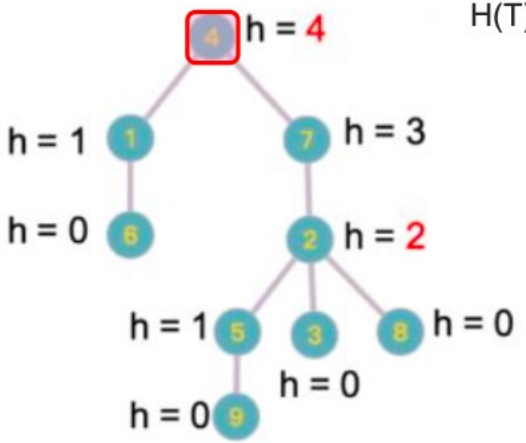
h = 2

h = 1

h = 0

H(T) = h(1)+h(2)+h(3)+h(4)+h(5)+h(6)+h(7)
= 0 + 1 + 1 + 0 + 2 + 0 + 0 = 4

```
nodes 9
1 1 6
2 3 5 3 8
3 0
4 2 1 7
5 1 9
6 0
7 1 2
8 0
9 0
```

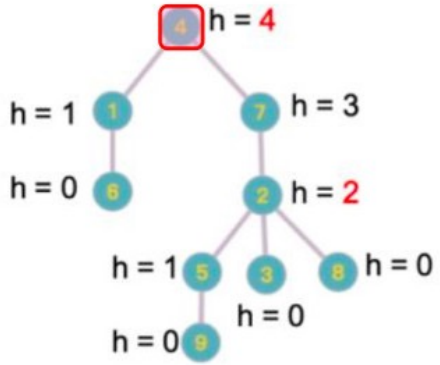


H(T) = h(1)+h(2)+h(3)+h(4)+h(5)+h(6)+h(7)+h(8)+h(9)
= 1 + 2 + 0 + 4 + 1 + 0 + 3 + 0 + 0 = 11

```python
def h(data,no):                          #計算節點的高度
    height=0                             #記錄目前最大的高度,預設值為0
    if len(data[no]):
        for i in range(len(data[no])):
            temp=h(data,data[no][i])+1
            height = max(temp,height)
        return height;                   #傳回子節點最大的高度
    else:
        return 0                         #葉節點為遞迴函數的結束出口

n=int(input())                           #n節點總數
data={}
for i in range(n):                       #輸入每列資料
    temp = input().split()
    subnode=int(temp[0])
    row=[]
    if subnode>0:                        #有子節點才輸入
        for j in range(1,subnode+1):
            row.append(int(temp[j]))
    data[i+1]=row
#print(data)
```

```
nodes 9
1 1 6
2 3 5 3 8
3 0
4 2 1 7
5 1 9
6 0
7 1 2
8 0
9 0
```




```python

root, total, highest = 0, 0, 0
for i in range(1,n+1):                   #依序計算每一節點的最大高度
    high=h(data,i)                       #節點的最大高度
    if high>highest:                     #如果該節點高度大於目前最大高度
        highest=high                     #將 hi 設為目前最大高度
        root=i                           #將該節點編號設為根節點編號
    total+=high                          #累計所有節點的最大高度
                root 一定是高度最高的節點
print("%d" %root)                        #根節點編號
print("%d" %total)                       #所有節點最大高度總和

```

實作題

物品堆疊(Stacking)

問題描述

某個自動化系統中有一個存取物品的子系統，該系統是將 N 個物品堆在一個垂直的貨架上，每個物品各佔一層。系統運作的方式如下：每次只會取用一個物品，取用時必須先將在其上方的物品貨架升高，取用後必須將該物品放回，然後將剛才升起的貨架降回原始位置，之後才會進行下一個物品的取用。

每一次升高某些物品所需要消耗的能量是以這些物品的總重來計算，在此我們忽略貨架的重量以及其他可能的消耗。現在有 N 個物品，第 i 個物品的重量是 w(i)而需要取用的次數為 f(i)，我們需要決定如何擺放這些物品的順序來讓消耗的能量越小越好。舉例來說，有兩個物品 w(1)=1、w(2)=2、f(1)=3、f(2)=4，也就是說物品 1 的重量是 1 需取用 3 次，物品 2 的重量是 2 需取用 4 次。我們有兩個可能的擺放順序(由上而下)：

- (1,2)，也就是物品 1 放在上方，2 在下方。那麼，取用 1 的時候不需要能量，而每次取用 2 的能量消耗是 w(1)=1，因為 2 需取用 f(2)=4 次，所以消耗能量數為 w(1)*f(2)=4。
- (2,1)，也就是物品 2 放在 1 的上方。那麼，取用 2 的時候不需要能量，而每次取用 1 的能量消耗是 w(2)=2，因為 1 需取用 f(1)=3 次，所以消耗能量數=w(2)*f(1)=6。

在所有可能的兩種擺放順序中，最少的能量是 4，所以答案是 4。再舉一例，若有三物品而 w(1)=3、w(2)=4、w(3)=5、f(1)=1、f(2)=2、f(3)=3。假設由上而下以(3,2,1)的順序，此時能量計算方式如下：取用物品 3 不需要能量，取用物品 2 消耗 w(3)*f(2)=10，取用物品 1 消耗(w(3)+w(2))*f(1)=9，總計能量為 19。如果以(1,2,3)的順序，則消耗能量為 3*2+(3+4)*3=27。事實上，我們一共有 3!=6 種可能的擺放順序，其中順序(3,2,1)可以得到最小消耗能量 19。

輸入格式

輸入的第一行是物品件數 N，第二行有 N 個正整數，依序是各物品的重量 w(1)、w(2)、…、w(N)，重量皆不超過 1000 且以一個空白間隔。第三行有 N 個正整數，依序是各物品的取用次數 f(1)、f(2)、…、f(N)，次數皆為 1000 以內的正整數，以一個空白間隔。

輸出格式

輸出最小能量消耗值，以換行結尾。所求答案不會超過 63 個位元所能表示的正整數。

範例一(第 1、3 子題)：輸入

```
2
20 10
1 1
```

範例一：正確輸出

```
10
```

範例二(第 2、4 子題)：輸入

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

範例二：正確輸出

```
19
```

評分說明：輸入包含若干筆測試資料，每一筆測試資料的執行時間限制(time limit)均為 1 秒，依正確通過測資筆數給分。其中：

第 1 子題組 10 分，N = 2，且取用次數 f(1)=f(2)=1。

第 2 子題組 20 分，N = 3。

第 3 子題組 45 分，N ≤ 1,000，且每一個物品 i 的取用次數 f(i)=1。

第 4 子題組 25 分，N ≤ 100,000。

No.	W	f	cost
1	20	1	10
2	10	1	20

找成本最小的組合

	f(No)		能量消耗
1(10)	1	0 * 1	0
2(20)	1	10 * 1	10
		總能量	10

No.	W	f	cost
1	3	1	9
2	4	2	16
3	5	3	21

	f(No)		能量消耗
3(5)	3	0 * 3	0
2(4)	2	5 * 2	10
1(3)	1	(5+4)*1	9
		總能量	19

```python
items = []                                                  #物品串列
min_energy = 0;                                             #最小消耗能量
total = 0;                                                  #物品重量總和

N = int(input())                                            #物品個數
weights = input().split()                                   #各物品重量
frequent = input().split()                                  #物品取用次數

for i in range(0, len(weights)):                            #逐一加入物件資料
    temp=[]                                                 #暫存各物件的物品重量及取用次數
    temp.append(int(weights[i]))                            #加入物品重量
    temp.append(int(frequent[i]))                           #加入物品取用次數
    items.append(temp)                                      #附加到物件串列
#print(items)
for i in range(N-1):                                        #將最小消耗能量由小到大排序
    for j in range(N-1-i):
        if items[j][0]*items[j+1][1] > items[j+1][0]*items[j][1]:
            items[j], items[j+1] = items[j+1], items[j]     #互相交換
#print(items)
for i in range(N-1):                                        #逐層處理
    total += items[i][0];                                     #前面物品重量總和
    min_energy += total * items[i+1][1]                     #最小能量消耗值總和
print("%d" %min_energy)                                     #輸出最小能量消耗值,以換行結尾
```

找到每個階段最小的成本

算出每階段的成本
公式 = 之前的重量合 * 目前的次數