제 2회 ALchemy대회 에디토리얼

서론 (및 여담)

2회차 알케미 내부 대회 에디토리얼입니다. 이번 대회 또한 백준에서 문제를 선정해온 형태로,

저번 1회차 대회보다 쉬운 난이도로 준비해왔습니다. 알고리즘 사전 지식을 최대한 배제한 문제들을 위주로 내보였기 때문에 초반부 문제들은 이번 에디토리얼을 참고해 다시 풀어보셔도 좋을 듯 하네요

A	B	C	D	E	F
2 / 20	2 / 38	1 / 36	2 / 53	1 / 53	1 / 66
1 / 154	1 / 158	1 / 163	1 / 164	1 / 177	1 / 200
1 / 10	2 / 36	1 / 25	1 / 31	1 / 51	0 / --
1 / 136	2 / 153	1 / 171	0 / --	1 / 239	0 / --
1 / 227	2 / 241	2 / 233	0 / --	0 / --	0 / --
4 / 298	0 / --	1 / 243	0 / --	0 / --	0 / --
2 / 25	3 / --	0 / --	0 / --	0 / --	0 / --

대회 스코어보드

시험 기간임에도 적지 않은 분들이 참여해주셨습니다. 예상했던 것보다 빠르게 문제를 해결하셔서 놀랐습니다👍

문제 해설

A ~ F, B4부터 S3까지 총 6문제를 준비했습니다.
대부분의 참가자가 Python3를 이용하여 제출하여, 에디토리얼 정답 코드도 Python3를 기준으로 작성되었습니다.

사실 작성자가 다른 언어를 잘 못씁니다.. ㅎㅎ;

A - 치킨댄스를 추는 곰곰이를 본 임스 2

문제 링크

난이도 : B4

분류 : 구현, 문자열, 파싱

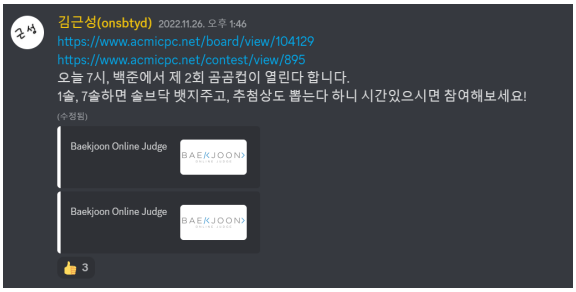
간단한 문자열 파싱 문제입니다. 각 언어의 **Split 함수**를 이용하여 '-' 를 기준으로 분리하거나,
앞 문자가 "D-"으로 동일한 점을 이용하여 **인덱스 2**부터 숫자를 받으면 되겠습니다.

지문에 나와있는 대로 **숫자가 90 이하**일 때만 카운트를 세면 정답이 됩니다!

여담으로 자유게시판에서 홍보한 곰곰컵2 대회의 A번이기도 합니다. 기출 문제인 셈이죠

A번은 참여한 모두가 풀 수 있기를 바라면서 아주 쉬운 난이도의 문제로 선정했습니다.

실제로 참가자 모두가 풀었네요👏👏👏



▼ 코드

```
#치킨댄스를 추는 곰곰이를 본 임스 2
n = int(input())
cnt = 0
for _ in range(n):
    d = input().split('-')
    day = int(d[1])

    if day <= 90:
        cnt += 1

print(cnt)
```

B - 노솔브 방지문제야!!

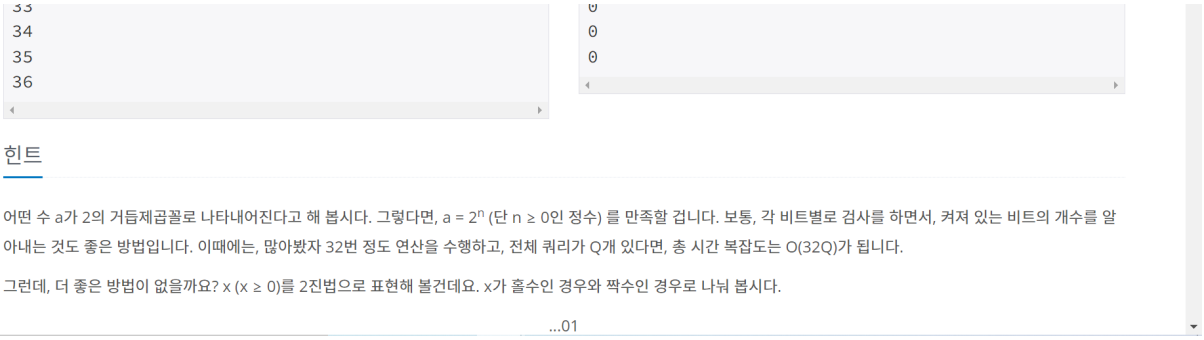
문제 링크

난이도 : B3

분류 : 수학, 구현

이 문제의 이름이 **노솔브 방지문제야!!** 인 이유는..

예제 입력.. 그 밑..



놀랍게도 힌트가 있습니다!

따라서 (a&(-a)) == a라면 2의 거듭제곱이고, 그렇지 않으면 2의 거듭제곱이 아닙니다.

힌트의 끝에서 이런 결론을 문제에서 제시해주는데요.
문제 해결 방식은 크게 **세 가지**가 있습니다.

1. **(a&(-a)) == a**
   힌트에서 나온 대로, 해당 수식을 이용하여 거듭제곱임을 판별하는 것이죠. 간단합니다.
2. **2로 나눠보기**
   2의 거듭제곱 꼴의 숫자라면 2^k의 형태일 것이고, 그렇지 않다면 $2^k * p$ 의 형태일 것입니다.
   따라서 2로 나누어 줄 수 있는 만큼 나눈다면, 2의 거듭제곱 꼴일 때만 1이 나오게 됩니다!

3. **이진수 형태로 나타내기**
   2의 거듭제곱인 수들은 이진수로 나타낼 경우, 모두 $100\ldots00$ 의 형태를 가지게 됩니다.
   제일 큰 비트만 1이고, 나머지는 0이므로, 이진수로 나타냈을 때 1이 한번만 나오는 것과도 같죠

다만, 해당 문제의 숫자 입력 개수가 **최대 100만**이기 때문에 숫자를 입력 받는 과정에서 시간이 엄청 오래 걸립니다. 이를 해결하기 위해 각 언어에선 입출력을 빠르게 해주는 방법이 있는데,

```
import sys
input = sys.stdin.readline
```

파이썬의 경우 해당 코드를 앞에 적어주고 진행하면 됩니다!

이에 대한 자세한 설명은.. 빠른 A+B 문제로 대체하겠습니다. 참고하세요. 다른 언어도 있어요

빠른 입출력에 대한 사전 지식이 없는 참가자가 있을 것이라 생각해, A번보다 높은 난이도로 책정하여 B번에 위치하게 되었습니다.

▼ 코드

```
#노솔브 방지문제야!!
import sys
input = sys.stdin.readline

q = int(input())
for query in range(q):
    n = int(input())
    if (n&(-n)) == n:
        print(1)
    else:
        print(0)
```

C - 뚊

문제 링크

난이도 : B1

분류 : 구현, 문자열

문제 자체는 매우 직관적입니다.

첫 번째 문자열의 각 문자를 두 배로 늘려서 두 번째 문자열과 같은지 비교하는 문제입니다. 구현 방식은 다양하나, 반복문을 통해 **첫 번째 문자열의 각 문자에 대해 두 번째 문자열에 해당하는 위치**를 확인하면 되겠습니다.

다만, 해당 문제는 기본적으로 입력이 n줄씩 들어오기 때문에 이를 저장하기 위해 배열, 특히 **2차원 배열**에 대한 이해가 필요합니다! 따라서 C번에 위치하게 되었습니다.

TMI : 뚦을 영문에서 그대로 입력하면 Eyfa이 나옵니다.

▼ 코드

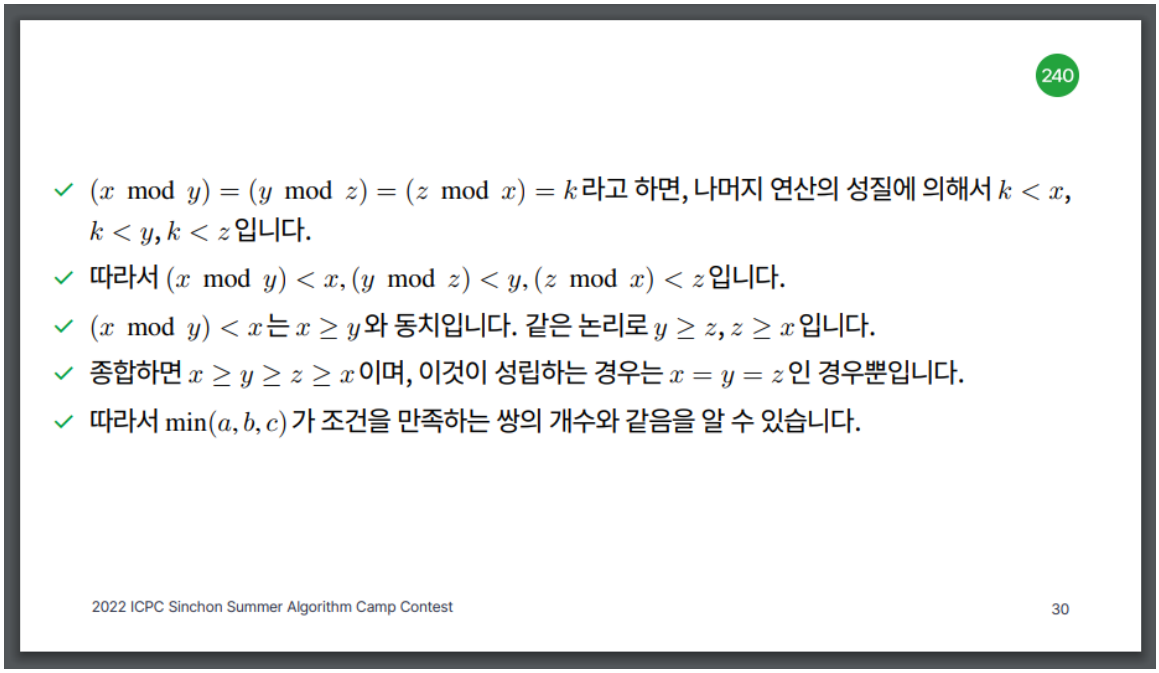
```
#뚦
n,m = map(int,input().split())
fir = [input() for _ in range(n)]
sec = [input() for _ in range(n)]

res = True
for i in range(n):
    for j in range(m):
        tmp1 = fir[i][j] == sec[i][j*2]
        tmp2 = fir[i][j] == sec[i][j*2+1]

        res &= tmp1 & tmp2

print("Eyfa" if res else "Not Eyfa")
```

D - 단순한 문제 (Large)

문제 링크

난이도 : S3

분류 : 수학, 정수론



출처 : 2022 ICPC Sinchon Summer Algorithm Camp Contest Editorial

숫자의 범위가 크지 않다면, 브루트포스 방식으로 일일이 찾아보는 방식이 가능합니다. 그러나 문제에서 제시되는 테스트 케이스가 60만개, 각 숫자의 범위가 10만까지로,

이는 주어진 조건을 잘 관찰하여, **나머지 연산의 성질**을 이용해 답을 찾아내야 하는 문제입니다. 더 좋은 설명은 이 문제가 출제된 본 대회의 에디토리얼(사진)을 참고하세요.
나머지 연산에 익숙하지 않다면 어려운 문제였을 것입니다.

그러나, 결론이 이상하지 않나요? `x = y = z 인 경우에만 성립을 한다.`

그렇습니다! 사실 이 문제의 다른 해법은 **디버깅**에 있습니다.

작은 범위의 수에 대해서 먼저 조건을 만족하는 수들을 출력해봤다면, 규칙성이 한 눈에 보였을 것입니다.

대략 이런 식의 코드를 구상할 수 있습니다.

입력인 5 4 3 에 대해 조건이 맞는 x, y, z가 서로 같을 때밖에 보이지 않습니다.
다른 여러 숫자를 입력해도 동일하죠

여기서 **모든 숫자가 같을 때만 조건을 만족하나?** 같은 생각을 했다면 그것이 그대로 정답이 됩니다!

따라서 입력으로 들어온 **a, b, c 중 가장 작은 값을 출력**해주면 문제가 해결됩니다.

```python
a,b,c = map(int,input('입력 : ').split())
for x in range(1,a+1):
    for y in range(1,b+1):
        for z in range(1,c+1):
            if x%y == y%z == z%x:
                print(x,y,z)
```

```
문제  출력  디버그 콘솔  터미널
PS C:\Users\jjkmk\Desktop\vs> & C:/Python/python.ex
ers/jjkmk/Desktop/vs/beakjoon/낙서장/temp.py
입력 : 5 4 3
1 1 1
2 2 2
3 3 3
PS C:\Users\jjkmk\Desktop\vs>
```

모든 경우의 수를 탐색하는 코드

코드를 작성하면서 중간마다 의도한 대로 작동하는지 테스트해보는 것을 보통 **디버깅**이라고 표현합니다. 코드를 실행하고 결과를 볼 수 있는 것은 프로그래밍의 가장 큰 특징이라고 생각합니다.

입력부터 출력까지 본인이 어떤 의도로 코드를 작성했는지 생각하고, **각 코드가 올바르게 동작했는지 디버깅**을 통해 확인하는 습관을 들이길 바라면서 해당 문제를 선정했습니다.

▼ 코드

```
#단순한 문제(Large)
import sys
input = sys.stdin.readline

t = int(input())
for case in range(t):
    a,b,c = map(int,input().split())
    res = min(a,b,c)
    print(res)
```

E - 쇠막대기

문제 링크

난이도 : S3

분류 : 스택

해당 문제는 괄호 문자열에 대한 이해를 기본으로 합니다. 이 문제를 먼저 해결하고 오시는 것을 추천합니다.

문제에서 가장 작은 괄호는 레이저를 나타내고, 나머지 괄호들은 쇠막대기를 나타냅니다.
이 문제에서 잘린 쇠막대기의 개수를 세는 방법은 두 가지로 볼 수 있습니다.

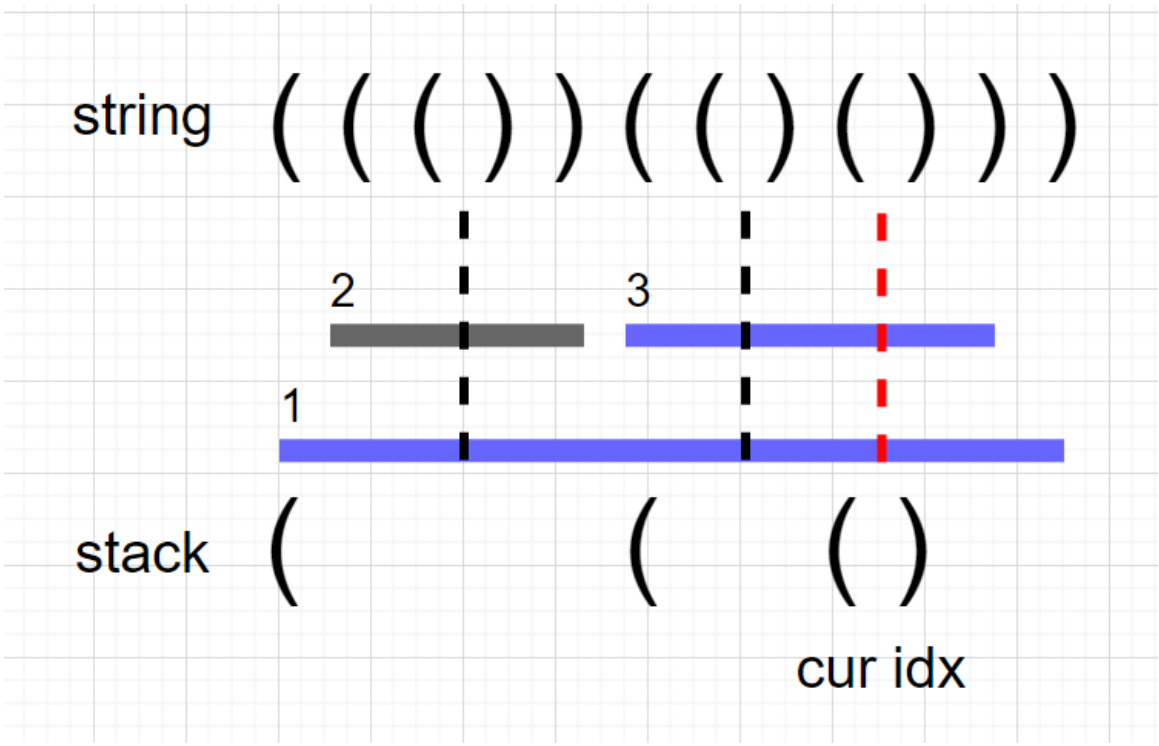
첫 번째는 각 쇠막대기가 몇 개의 레이저를 만나 잘렸는지 개수를 세는 것이고,
두 번째는 **각 레이저가 몇 개의 쇠막대기를 잘랐는지 개수를 세는 것**입니다.

결과적으로 큰 차이는 없으나, 저희는 두 번째 방법에 초점을 맞춰서 문제를 해결하겠습니다.

괄호 문자열은 일반적으로 문자열을 읽어가면서 `(`이 나올 때 stack에 저장하고, `)`이 나올 때마다 stack에서 pop해주어 올바른 괄호쌍을 찾습니다.

이때 `)`이 나올 때 본 문자열의 바로 전 인덱스가 `(`이라면 해당 괄호는 레이저를 의미하게 됩니다.

또한 이 레이저가 자르는 쇠막대기의 개수는 레이저를 제외하고 현재까지 닫히지 않은 괄호의 개수와 같습니다. 즉 `len(stack) - 1`입니다.



현재 인덱스에서 빨간색 레이저와, 파란색 1, 3번 쇠막대기가 만난다.

이렇게 괄호 문자열을 읽어 **각 레이저가 쇠막대기를 자른 횟수**를 얻을 수 있고, 이 **총합**을 C라고 하겠습니다.

이제 다시 쇠막대기를 기준으로 봤을 때, 1개의 쇠막대기가 i번 잘린다면 $i+1$개의 조각이 되는 것을 알 수 있습니다.

총 쇠막대기의 개수가 s개일 때, 각 쇠막대기에 1번부터 s번까지 번호를 붙이겠습니다.
그리고 각각의 쇠막대기가 잘린 횟수를 $a_1, a_2, ...a_s$이라 하면,

전체 합은 쇠막대기가 잘린 횟수의 총합이니 아까 계산한 C입니다.

즉 $\sum_{i=1}^{s} a_i = C$이라고 할 수 있는데, 각각의 쇠막대기는 잘린 횟수보다 1개씩 많은 조각을 가지게 됩니다.

다시 말해 잘린 쇠막대기 조각의 총합은 $\sum_{i=1}^{s}(a_i+1)$, 즉 $C+s$가 문제의 정답이 됩니다!

따라서 괄호가 닫힐 때, 해당 괄호가 레이저라면 `len(stack) - 1`으로 잘린 횟수를 세고, 해당 괄호가 쇠막대기라면 `+1`로 쇠막대기의 개수를 세주면 문제를 해결할 수 있습니다.

스택과 괄호 문자열에 대한 사전 지식과, 이에 대한 활용이 필요하다고 생각되어 E번에 위치하게 되었습니다.

▼ 코드

```
#쇠막대기
st = input()
stack = []
C = 0
s = 0
for i in range(len(st)):
    if st[i] == ')':
        if st[i-1] == '(':
            C += len(stack)-1
        else:
            s += 1

        stack.pop()

    else:
        stack.append(st[i])

res = C+s
print(res)
```

F - 양치기 꿍

문제 링크

난이도 : S1

분류 : 그래프 이론, 그래프 탐색, bfs, dfs

해당 문제는 그래프 탐색에 대한 이해를 기본으로 합니다.

> bfs, dfs에 대해..
>
> 기초 bfs(너비 우선 탐색) 문제는 이 문제를,
> 기초 dfs(깊이 우선 탐색) 문제는 이 문제를 참고하세요.
>
> 인터넷에서 검색해보면서 푸는 걸 추천합니다! 처음 접하신다면 어려운 개념이에요
>
> 그리고 해당 문제는 그래프를 평면 좌표의 형태로 주기 때문에, 위에 링크된 문제들에서 조금 변형이 필요합니다. 이에 대해서는 이 문제를 추천합니다.
>
> ~~다만.. 사실 저 문제를 풀 수 있으면 이 문제도 금방 풀려요 ㅎㅎ..;~~

문제 설명으로 돌아와서, 각 울타리로 나뉘어진 영역은 **서로 독립적**입니다. 따라서 이중 반복문을 통해 전체 지도를 돌면서, `#`을 기준으로 **여러 번의 그래프 탐색**을 진행하면 되겠습니다.

하나의 그래프 탐색에서 `v, k, .` 은 똑같이 이동할 수 있는 지역으로 생각하되, v와 k에서 각각 개수를 세줍니다.
이때 한 구역의 그래프 탐색이 끝나면, 양과 늑대의 수를 비교합니다. 양의 수가 늑대의 수보다 많다면 `count_k, 0` 을 리턴하고, 그렇지 않다면 `0, count_v` 을 리턴합니다.

각 그래프 탐색이 끝날 때마다, 같은 방식으로 살아남은 양과 늑대의 개수를 세주면 문제가 해결됩니다!

문제에 어떤 응용보다는, 그래프 탐색에 대한 사전 지식이 난이도가 높다 판단되어 마지막 문제로 위치하게 되었습니다.

▼ 코드

```
#양치기 꿍
from collections import deque

def bfs(loc):
    sx,sy = loc
    visit[sx][sy] = True

    ck,cv = 0,0
    queue = deque([loc])
    while queue:
        x,y = queue.popleft()
        if d[x][y] == 'k': ck += 1
        elif d[x][y] == 'v': cv += 1

        for i in range(4):
            nx = x + dx[i]
            ny = y + dy[i]

            if 0<=nx<r and 0<=ny<c and not visit[nx][ny] and d[nx][ny] != '#':
                visit[nx][ny] = True
                queue.append((nx,ny))

    if ck > cv:
        return ck,0
    else:
        return 0,cv

dx = (-1,1,0,0)
dy = (0,0,-1,1)

r,c = map(int,input().split())
```

```
d = [input() for _ in range(r)]
visit = [[False]*c for _ in range(r)]

k,v = 0,0
for i in range(r):
    for j in range(c):
        if not visit[i][j] and d[i][j] != '#':
            tmp = bfs((i,j))
            k += tmp[0]
            v += tmp[1]

print(k,v)
```

끝으로

이번 에디토리얼 문제 해설은 고민규님께서 시간 들여 작성해주셨습니다. 시험 기간임에도 대회에 참여하신 모든 분들 고생하셨고, 이후에 더 나은 대회로 찾아뵙겠습니다. 다음 대회에서는 방학 중에 스터디와 같은 여러 활동을 진행하고 다양한 문제를 선보이고자 합니다.