9661 돌 게임 7

문제 링크: https://www.acmicpc.net/problem/9661

초반 몇번의 경우를 시도 해본다면 5개의 답이 반복된다는 것을 알 수 있다. 여기서 힘든 것은 이 5개의 반복이 왜 4^n *(n은 임의 정수)*에 항상 성립하는지를 증명하는 것이다. 수학적으로 완벽하게 증명은 하지 못 하였지만 내가 이해한 수준까지만을 설명해보겠다.

먼저 처음 20가지의 경우를 써보겠다 (S = SK, C = CY) SCSSC SCSSC SCSSC SCSSC

여기서 인지해야 할 점은, 인풋이 1부터 3까지는 SY는 무조건 1을 뽑아야하고 4부터 15까지는 1 혹은 4를 뽑을 수 있다. 만약 5라는 값이 인풋으로 주어 졌다면, S는 1혹은 4를 뽑을 수가 있다. 이때 1을 뽑는다면 4번째의 결과가 S이기에 답은 C가 된다. (C가 되는 이유는 4번째의 답이 S일 때 전제 조건이 'S가 시작한다'이기 때문이다. 만약 S가 1을 뽑아서 4라는 수가 남고 그게 C부터 시작이라면 답이 C이다.) 같은 원리로 S가 4를 뽑더라도 1의 답이 S이기에 5의 답은 C가 된다. 즉 반드시 C가 이긴다는 것이다. 여기서 주의 깊게 봐야할 점은 어떤 경우라도 C가 이기기 위해선 첫번째 S의 선택 후 남게 되는 값이 어떤 접근으로라도 C가 이기면 된다는 사실이다. 예를 들어 6이라는 인풋이 주어지면 S의 선택은 1과 4가 있다. 여기서 1을 선택하면 5라는 값이 남고 5라는 값의 답은 C(S 기준에서)이고 4를 선택하면 2라는 값이 남고 답은 C(S 기준에서)이다. 여기서 기준은 C로 바뀌기 때문에 어떤 경우라도 S가 이긴다. 다음으로 C 가나오는 7번째의 경우는 S가 1을 뽑으면 6으로 어떤 경우에도 C가 이기고 4를 뽑더라도 2가 남아 어떤 경우에도 C가 이기게 된다. 그렇다면 여기서 n번째 값의 답을 구하기 위해서는 n-1, n-4, ... n-4^n 까지의 상태의 값을 구해서 모두 S일 경우만 C로 출력을 해주면 된다.(시작이 S이기 때문에 S가 실질적으로 주도권을 가지고 경우가 2경우라면 반드시 S가 이긴다고 생각하면 된다.) 그런데 여기서 우리는 n-1의 값이 이미 존재하는 값들의 배열 중의 한 값임을 알기에 n-1로 새로 만드는 배열은 기존의 SCSSC SCSSC SCSSC SCSSC ... 와 같다. 다만 다른점은 시작 인덱스가 다르다는 것이다. 이는 n-4, n-16, ...에도 해당되는 사실이다. 즉 n-1의 배열은 기존 배열의 시작 인덱스 1 기준으로 2와 6에서 새로운 SCSSC가 시작되고 n-4는 기존 배열 시작 인덱스 1 기준으로 5, 10에서 시작 된다. 4의 어떤 수를 제곱하더라도 그 마지막 값이 6 혹은 4이기 때문에, 실질적으로는 2 혹은 6에서 시작하는 배열 하나와 1 혹은 5 기준으로 시작하는 SCSSC의 반복이라고 생각하면 된다. (이는 스터디 때 보충 설명을 하겠다). 그렇다면 해당 값은 14까지만 확인하여 n번째의 값을 결정하는 n-1, n-4의 배열을 통해서 값이 반복된다는 것을 증명함으로 써 그 어떤 큰 수라도 증명된다는 것을 알 수 있다.

In [ ]:

```
result = ['SK','CY','SK','SK','CY']

a = int(input())
a-=1
a = a % 5

print(result[a])
```

9251 LCS

문제 링크: https://www.acmicpc.net/problem/9251

LCS라고 컴퓨터 분야에서 유명한 알고리즘 문제 중 하나이다. sorting들 처럼 이미 몇년에 걸쳐서 만들어준 알고리즘 중 하나로 익혀두길 바란다. LCS 문제 자체에 대한 3가지 경우를 안다면 간단하게 풀 수 있다.

Notation은 다음과 같다

> X = 문자열 (1부터 시작하는 i로 x_i를 사용하여 X의 문자와 위치를 표현한다)
> m = 문자열 X의 길이
>
> Y = 문자열 (1부터 시작하는 i로 y_i를 사용하여 Y잉 문자와 위치를 표현한다)
> n = 문자열 Y의 길이
>
> Z = Common subsequence (소문자와 i를 사용하여 각 element를 표현한다)
> o = 문자열 Z의 길이

경우는 다음과 같다

> **경우 1**
> x_m = y_n 이면 z_o = x_m = y_n 이다. 이 때, Z_{o-1}는 X_{m-1}과 Y_{n-1}의 LCS이다.
>
> **경우 2**
> x_m != y_n 이면 z_o != x_m이다. 이 때, Z_o는 X_{m-1}과 Y_n의 LCS이다.
>
> **경우 3**
> x_m != y_n 이면 z_o != y_m이다. 이 때, Z_o는 Y_{n-1}과 X_m의 LCS이다.

여기서 경우 2와 3은 항상 함께 발생한다

이 기본 경우들을 마음에 품고 이걸 프로그램화 하면 간단하다. Dynamic programming을 이용하여 m개의 줄과 n개의 열을 가지는 배열을 생성하고 모두 0으로 초기화한다. 만약 x_m과 y_n이 같다면, 경우 1에 따라 그보다 X_{m-1}과 Y_{n-1}의 LCS값에 +1을 m행 n열에 저장하면된다. 만약 다르다면, 경우 2, 3에 따라 배열[m][n-1]과 배열[m-1][n]의 값 중 더 큰 값을 고르면 된다. (항상 최대가 되게 하기 위하여)

사실 이 수학적인 개념을 프로그래밍에 바로 도입한다면 recursive하게 bottom_up을 사용하는 전형적인 재귀와 DP를 섞어야 하지만 이 문제의 경우 저 기본 개념을 이용하면 Top bottom 방식을 사용해도 문제가 없음을 자명할 수 있기에 Top bottom을 사용하겠다. (구현이 더 쉽고 더 직관적이라는 장점이 있다.)

In [ ]:

```
str1 = input()
str2 = input()

check = [[0 for _ in range(len(str2)+1)] for _ in range(len(str1)+1)]

i, j = 1, 1

for first in str1:
    for second in str2:
        if first == second:
            check[i][j] = check[i-1][j-1]+1
        else:
            check[i][j] = max(check[i-1][j],check[i][j-1])
        j += 1
    i += 1
    j=1

print(check[-1][-1])
```

11009 Drinks (Unsolved)

문제 링크: https://www.acmicpc.net/problem/9251

처음 시작하는 사람이 red를 뽑는 전체의 경우를 구하는 문제이다.

풀이 법은 간단하게 생각하였는데 15 이상의 숫자가 되는 순간 답이 이상하게 나온다. 정확한 이유는 모르겠다.

풀이법은 다음과 같다. red의 갯수가 N, white의 갯수가 M이라 할 때, 처음에는

$$\frac{N}{M+N}$$

을 구한다. 그 뒤

$$\frac{M}{M+N}+\frac{M}{M+N}\frac{M-1}{M+N-1}\frac{N}{M+N-3}$$

과 같이 white 공을 연속으로 두번 뽑고 그 뒤에 red를 뽑을 경우, white 공을 4번 뽑고 red를 뽑을 경우로 구하면 된다. 이 기본이 확장되어 2번, 4번 6번,... 과 같이 짝수번 반복되어서 white 공이 추출되고 그에 해당하여 M 값은 1씩 줄어들면서 모두 곱해주고 마지막에 red공을 뽑아야 함으로 남은 공 중에 N을 뽑는 경우의 수를 계산하면 풀 수 있다. 15 이상의 숫자에서는 제대로 되지 않는다. 이유를 더 고민해보자.

In [ ]:

```
# 최소 공배수를 구하는 것 이를 통하여 분모, 분자의 최소 공배수를 구하고 나눠 줄 수 있다.
def gcd(m,n):
    while True:
        if n > m :
            m, n = n, m
        r = m%n
        if (r ==0):
             return n
        else:
            m = n
            n = r

for _ in range(int(input())):
    n, m = map(int, input().split())

    real_m = m//2
    real_m = real_m*2
    mul_value = []

    result = 0

    numerator = 0
    denomi = 1

    for i in reversed(range(m+1)):
        denomi *= (n + i)
        mul_value.append(denomi)

    for i in range(m+1):
        if i %2 == 1:
            continue
        temp =0
        temp_num = 1
        while temp < i:
            temp_num *= (m-temp)
            temp += 1
        temp_num *= n
        numerator += temp_num * (denomi/mul_value[i])

    common = gcd(denomi, numerator)

    print("{}/{}".format(int(numerator/common), int(denomi/common)))
```

2812 큰수만들기

문제 링크: https://www.acmicpc.net/problem/2812
STACK 특집 때 풀던 5주차 2812번 탑 문제를 응용하여 풀었다. (*탑 문제 링크: https://www.acmicpc.net/problem/2493*)

주어진 숫자들을 앞에서 부터 순차적으로 접근하면서 그 앞에 값 중 자신의 값보다 작은 값은 stack에서 pop을 해주고 자신보다 큰 값을 만나거나, stack이 비었을 경우 자기자신을 추가하는 방식이다. 만약 123456.. 과 같은 숫자가 있고 가장 큰 한 숫자를 선택한다고 해보자. 이 경우 stack에는 1-> 2-> 3-> 4-> 5 -> 6이 들어오고 뒤의 숫자에 따라 스택은 언제든 변화가 가능하다. 이때 생각해줘야 하는 조건이 특정 경우의 경우 가장 큰 수부터 순차적으로 작아지는 스택을 만들 수 없다는 것인데 그 이유는 만약 주어진 숫자들이 42345이고 K가 3이라서 2개의 숫자를 남겨야 할 경우, 위의 설명한 방법을 사용하면 5, 하나가 남게 된다. 이를 방지하기 위해서 스택의 변화가 이뤄질 때 마다, 현재 스택의 길이와, 아직 접근하지 않은 숫자들의 갯수를 더 해서 구하고자 하는 경우의 길이와 같은 경우 중단 시켜 주면 된다. 4가 저장되어 있는 스택에서 5가 저장되려고 할 때 스택의 길이가 1이고 남은 숫자열의 길이가 1이기 때문에 stack을 제거하지 않고 stack을 이루는 값들과 남은 값으로 결과를 만들면 된다. 마지막으로 생각해줘야 하는 경우는 stack을 만들었는데 그 크기가 목표로하는 길이보다 길 경우이다. 이럴 때는 간단하게 stack에서 목표하는 길이만큼의 값만으로 문자열을 만들면 된다.

In [ ]:

```
N, K =map(int,input().split())
nums = input()

target = N-K
stack = []
len_stack = len(stack)
len_remain = len(nums)
remain = []
done = False

for i, num in enumerate(nums):
    len_remain -= 1
    if len_stack != 0:
        while stack and stack[-1] < num:
            stack.pop()
            len_stack -= 1
            if len_remain+ len_stack+1 == target:
                stack.append(num)
                len_stack +=1
                break
        else:
            len_stack+=1
            stack.append(num)
    else:
        stack.append(num)
        len_stack += 1

    # in case of sum of the number of elements in stack and remaining words equals to the target
    if len_remain + len_stack == target:
        if len_remain != 0:
            result = ''.join(stack)+''.join(nums[i+1:])
            done = True
        break

if not done:
    result = ''.join(stack[:target])

print(result)
```