Print Queue

Last Edited	@May 11, 2020 9:03 PM
Tags	python 백준

백준 1966번 '프린터 큐'

문제

여러분도 알다시피 여러분의 프린터 기기는 여러분이 인쇄하고자 하는 문서를 인쇄 명령을 받은 '순서대로', 즉 먼저 요청된 것을 먼저 인쇄한다. 여러 개의 문서가 쌓인다면 Queue 자료구조에 쌓여서 FIFO - First In First Out - 에 따라 인쇄가 되게 된다. 하지만 상근이는 새로운 프린터기 내부 소프트웨어를 개발하였는데, 이 프린터기는 다음과 같은 조건에 따라 인쇄를 하게 된다.

1. 현재 Queue의 가장 앞에 있는 문서의 '중요도'를 확인한다.
2. 나머지 문서들 중 현재 문서보다 중요도가 높은 문서가 하나라도 있다면, 이 문서를 인쇄하지 않고 Queue의 가장 뒤에 재배치 한다. 그렇지 않다면 바로 인쇄를 한다.

예를 들어 Queue에 4개의 문서(A B C D)가 있고, 중요도가 2 1 4 3 라면 C를 인쇄하고, 다음으로 D를 인쇄하고 A, B를 인쇄하게 된다.

여러분이 할 일은, 현재 Queue에 있는 문서의 수와 중요도가 주어졌을 때, 어떤 한 문서가 몇 번째로 인쇄되는지 알아내는 것이다. 예를 들어 위의 예에서 C문서는 1번째로, A 문서는 3번째로 인쇄되게 된다.

입력

첫 줄에 test case의 수가 주어진다. 각 test case에 대해서 문서의 수 N(100이하)와 몇 번째로 인쇄되었는지 궁금한 문서가 현재 Queue의 어떤 위치에 있는지를 알려주는 M(0이상 N미만)이 주어진다. 다음줄에 N개 문서의 중요도가 주어지는데, 중요도는 1 이상 9 이하이다. 중요도가 같은 문서가 여러 개 있을 수도 있다. 위의 예는 N=4, M=0(A 문서가 궁금하다면), 중요도는 2 1 4 3이 된다.

출력

각 test case에 대해 문서가 몇 번째로 인쇄되는지 출력한다.

예제 입력 1 복사

```
3
1 0
5
4 2
1 2 3 4
6 0
1 1 9 1 1 1
```

예제 출력 1 복사

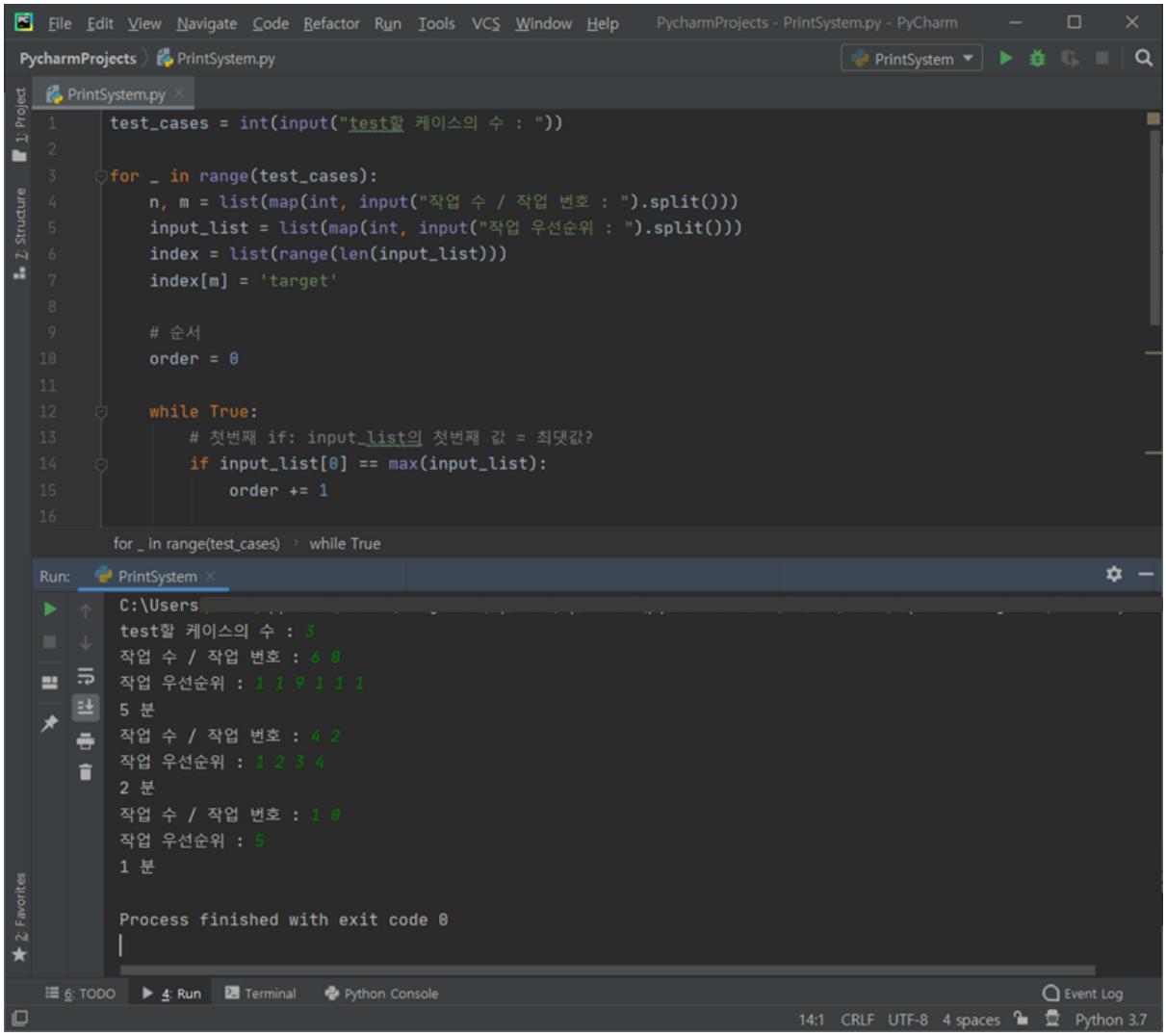
```
1
2
5
```

Python Code :

```
test_cases = int(input("test할 케이스의 수 : "))

for _ in range(test_cases):
    n, m = list(map(int, input("작업 수 / 작업 번호 : ").split()))
    input_list = list(map(int, input("작업 우선순위 : ").split()))
    index = list(range(len(input_list)))
    index[m] = 'target'

    # 순서
    order = 0

    while True:
        # 첫번째 if: input_list의 첫번째 값 = 최댓값?
        if input_list[0] == max(input_list):
            order += 1

            # 두번째 if: index의 첫 번째 값 = "target"?
            if index[0] == 'target':
```

```
                print(order,"분")
                break
            else:
                input_list.pop(0)
                index.pop(0)

        else:
            input_list.append(input_list.pop(0))
            index.append(index.pop(0))
```

Result :



Etc.

1. append vs. extend

append()는 object를 맨 뒤에 추가합니다.

```
x = [1, 2, 3]
x.append([4, 5])
print (x)
```

```
>> [1, 2, 3, [4, 5]]
```

extend()는 iterable 객체(리스트, 튜플, 딕셔너리 등)의 엘리멘트를 list에 appending 시킵니다.

```
x = [1, 2, 3]
x.extend([4, 5])
print (x)
```

```
>> [1, 2, 3, 4, 5]
```

2. remove vs. pop vs. del

remove는 지우려는 값을 입력하는 방식입니다. 만약 그 값이 리스트 내에 2개 이상 있다면 순서상 가장 앞에 있는 값을 지웁니다.

```
a = [1,2,1,3,4,5,1]
a.remove(1)

print(a)
```

```
>> [2, 1, 3, 4, 5, 1]
```

pop과 **del**은 지우고자 하는 리스트의 인덱스를 받아서 지우는 방식입니다. pop은 지워진 인덱스의 값을 반환하지만 del은 반환하지 않습니다. 이 차이 때문에 del이 pop보다 수행속도가 더 빠릅니다.

```
a = [1,2,1,3,4,5,1] #pop
removed = a.pop(1)
print(a)


b = [1,2,1,3,4,5,1] #del
del b[1]
print(b)
```

```
>> [1, 1, 3, 4, 5, 1]
[1, 1, 3, 4, 5, 1]
```

*del은 pop과 다르게 리스트의 범위를 지정해 삭제할 수 있습니다.

```
a = [1,2,1,3,4,5,1]
del a[:2]
print(a)
```

```
>> [1, 3, 4, 5, 1]
```