2021-09-10

<코딩테스트>

백준-2920-음계 15분 주어진 시간

내가 짠 코드

n = list(map(int, input().split()))

a = [1, 2, 3, 4, 5, 6, 7, 8]

b = [8, 7, 6, 5, 4, 3, 2, 1]

if n == a:

print('ascending')

elif (n == b):

print('descending')

else:

print('mixed')

나동빈씨 코드와 그 코드를 보면서 느낀 점.

A = list(map(int, input().split(‘ ‘)))

###

split(‘ ‘)을 붙임으로써 3 5. …9 이런식으로 입력을 주면

[‘3’, ‘5’, … ‘9’] 이런식으로 값이 나오게 된다. 문자열 형태로 리스트에 담긴다.

이것을 map을 이용해서 각각의 원소를 int형으로 바꿔주게 된다.

그럼

[3, 5, …, 9] 이런 식으로 int 형으로 들어가 있는 배열이 만들어지게 되고

그것을 a라는 변수에 담겠다. 라는 의미

차례대로 index가 3은 0 5는 1 … 이런 식으로 간다.

인덱스 다음 값이 내가 정한 인덱스 값보다 크다면 오름차순을 의미하는 것이고 게속 작다면 내림차순

두 가지가 섞여 있으면 mixed를 의미하는 것이다.

###

Ascending = True

Descending = True

For i in range(1, 8):

if a[I] > a[I-1]:

descending = False

elif a[I] < a[i-1]:

ascending = False

If ascending:

print(‘ascending’)

Elif descending:

print(‘descending’)

else:

print(‘mixed’)

## PyPy3로 제출하는 경우 일반적으로 시간은 더 단축되지만 메모리는 좀 더 커진다.

* 굉장히 구조적으로 조건에 따라 결과값을 나누는게 확실하게 보인다.
* 다만 내가 짠 코드도 돌아가긴 하는데 역시 다른 사람의 코드를 봐야하는구나를 느끼며 더 찾아보기 시작했다.

백준 2798

#문제 풀이 핵심 아이디어

1. 카드 중 3개씩 뽑는 모든 경우의 수는 C(n,3)이며, n은 최대 100
2. 따라서 단순히 3중 반복문으로 모든 경우의 수를 확인하여 문제를 해결할 수 있다.

전체 경우의 수는 n(n-1)(n-2)를 3!로 나눈 것인데 전체 경우의 수를 전부 다 고려한다고 해도 n이 최대 100밖에 안되기 때문에

대략 이것을 러프하게 100만이라고 설정을 하더라도 컴퓨터가 충분히 계산할 수 있을 정도의 적은 양이기 때문에 이 문제를 완전탐색을 이용해서 풀어도 문제가 된다.

파이썬 대략 1초에 2000만정도의 연산을 수행한다.

따라서 그냥 완전탐색으로 해도 무리가 없기 때문에 완전탐색을 진행한다.

N, m = list(map(int, input().split(‘ ‘)))

Data = list(map(int, input().split(‘ ‘)))

Result = 0

Length = len(data)

# 데이터의 length를 구한 다음에 3중 반복문을 이용해서 모든 경우의 수를 다 구한다.

# 5개의 카드 중 3개를 뽑는다고 하면 1개를 고정 시키고 또 한개를 고정시키고 한개를 바꿔가는 식으로 하나씩 전부다 해보는 식이 밑에 식이다.

# 내 생각 : 나동빈씨가 적은 것 중 range를 구성하는 방법이 그냥 하는 것보다 좀 더 직관적으로 이해하고 보기 좋다.

백준 1874

n = int(input())

# final\_number = list(map(int, input().split()))

# # 최종적으로 만들고 싶은 수열이라 final\_number라고 명시

# abc\_number = final\_number.sort() # 입력받은 final\_number값을 오름차순 정렬한 수열

# for i in range(n):

# if final\_number == abc\_number:

# break

# else:

# list를 오름차순으로

##나동빈씨 풀이

n = int(input())

count = 1

stack = []

result = []

for i in range(1, n+1): # 데이터 개수 만큼 반복

data = int(input())

while count <= data: # 입력하는 데이터 도달할 때까지 삽입 ( 원소를 넣을 때 )

# count가 1부터 n까지 증가를 하면서 데이터를 스택에 넣는 것이다.

stack.append(count)

count += 1 # 어떤 수에 도달할 때까지 하나씩 1을 더해가면서 그 원소를 stack에 더해가는 것이다.

result.append('+')

# result는 연산이 끝난 다음에 그 결과가 출력되기 위한 내용들이 담기게 된다. ('+' or '-'로)

# 삽입이 될때마다 이 result에는 +가 담긴다.

# => 특정한 수에 도달할 때까지 stack에 넣기만 한다.

if stack[-1] == data: #스택의 최상위 원소가 데이터와 같을 때 출력 ( 원소를 뽑을 때 )

stack.pop

result.append('-')

# 삽입이 될 때마다 이 result에는 -가 된다.

# 내림차순 형태가 아니라면 우리는 스택에서 원하는 해당 데이터를 뽑을 수 없다.

else: # 불가능한 경우

print('NO')

exit(0)

# => 스택에 최상위 원소가 데이터와 같은지 검사를 한다. 같을 때만 뽑고 아니면 프로그램 종료.

# => exit(0) : 프로그램 종료

# 종료 코드는 0과 1

# exit(0) 오류 또는 문제없이 깨끗한 출구를 뜻한다.

# exit(1) 오류 또는 문제가 있음을 의미하므로 프로그램이 종료 된다.

# 0이 아닌 종료 코드는 비정상 종료로 처리되며 때때로 오류 코드는 문제점을 나타낸다. 오류코드 0은 성공적인 종료를 의미한다.

# break는 루프를 탈출하는 것이다.

# for foreach, while 등 여러 번 반복되는 루프의 경우에 해당 된다.

# 루프 이후에 등장하는 코드를 계속 실행한다.

# if문은 반복문이 아니라 조건 충족시 1번만 실행되는 코드이므로 해당없다.

# return은 함수 탈출 또는 인클루드 탈출

# 함수 안에서 쓰면 함수 실행을 종료하고, 그 함수를 호출했던 지점으로 돌아가서 계속 실행한다.

# 함수 밖에서 쓰면 현재 파일을 인클루드 했던 파일로 돌아가서 계속 실행한다.

# 함수 밖인데 인클루드한 것도 없으면 그냥 종료된다. 즉 exit과 동일한 효과가 된다.

# exit, die = 무조건 종료

# 말 그대로 프로그램이 죽어버린다.

print('\n' .join(result)) # 가능한 경우 줄바꿈을 기준으로 출력을 한다.

# ==> [+, +, +, +, -, -]가 있다면 이것을

# +

# +

# +

# +

# -

# - 이렇게 출력할 수 있도록 해 주는 것이 print('\n' .join(result)) 이다.

### 이 코드를 보고 궁금한 것들

# .join 과 .apppend, .pop, exit의 의미 및 사용 형태

############# 나동빈씨는 위의 코드를 스택 수열을 풀어보는 간단한 코드라고 했다. 난 너무 복잡하게 생각한 것 같다.

###### 어떤 식으로 생각하면 저런 흐름을 가져갈 수 있는지 고민해보자. 내가 어떤 부분에서 너무 복잡하게 생각했는지 (왜냐면 생각보다 간단한 문제를 나는 너무 복잡하게 생각하는 것 같다.)

#### 추가로 내가 생각한 핵심포인트와 나동빈씨가 핵심 포인트의 차이는 무엇이며 어떠한 방법으로 접근하면 이것을 해결할 수 있는지 생각해보자.

##================================================================================================##

# 스택과 큐에 대한 개념 확실하게 구분하기

* 내가 너무 복잡하게 생각하는 경향이 있다.
* 단순하게 생각하고 그것을 구현하는 방식을 익히면 좀 더 편할 것 같은 느낌.