**문제 설명 - 캐시**

지도개발팀에서 근무하는 제이지는 지도에서 도시 이름을 검색하면 해당 도시와 관련된 맛집 게시물들을 데이터베이스에서 읽어 보여주는 서비스를 개발하고 있다.

이 프로그램의 테스팅 업무를 담당하고 있는 어피치는 서비스를 오픈하기 전 각 로직에 대한 성능 측정을 수행하였는데, 제이지가 작성한 부분 중 데이터베이스에서 게시물을 가져오는 부분의 실행시간이 너무 오래 걸린다는 것을 알게 되었다.

어피치는 제이지에게 해당 로직을 개선하라고 닦달하기 시작하였고, 제이지는 DB 캐시를 적용하여 성능 개선을 시도하고 있지만 캐시 크기를 얼마로 해야 효율적인지 몰라 난감한 상황이다.

어피치에게 시달리는 제이지를 도와, DB 캐시를 적용할 때 캐시 크기에 따른 실행시간 측정 프로그램을 작성하시오.

입력 형식

캐시 크기(cacheSize)와 도시이름 배열(cities)을 입력받는다.

cacheSize는 정수이며, 범위는 0 ≦ cacheSize ≦ 30 이다.

cities는 도시 이름으로 이뤄진 문자열 배열로, 최대 도시 수는 100,000개이다.

각 도시 이름은 공백, 숫자, 특수문자 등이 없는 영문자로 구성되며, 대소문자 구분을 하지 않는다. 도시 이름은 최대 20자로 이루어져 있다.

출력 형식

입력된 도시이름 배열을 순서대로 처리할 때, "총 실행시간"을 출력한다.

조건

캐시 교체 알고리즘은 LRU(Least Recently Used)를 사용한다.

cache hit일 경우 실행시간은 1이다.

cache miss일 경우 실행시간은 5이다.

입출력 예제

캐시크기(cacheSize) 도시이름(cities) 실행시간

3 ["Jeju", "Pangyo", "Seoul", "NewYork", "LA", "Jeju", "Pangyo", "Seoul", "NewYork", "LA"] 50

3 ["Jeju", "Pangyo", "Seoul", "Jeju", "Pangyo", "Seoul", "Jeju", "Pangyo", "Seoul"] 21

2 ["Jeju", "Pangyo", "Seoul", "NewYork", "LA", "SanFrancisco", "Seoul", "Rome", "Paris", "Jeju", "NewYork", "Rome"] 60

5 ["Jeju", "Pangyo", "Seoul", "NewYork", "LA", "SanFrancisco", "Seoul", "Rome", "Paris", "Jeju", "NewYork", "Rome"] 52

2 ["Jeju", "Pangyo", "NewYork", "newyork"] 16

0 ["Jeju", "Pangyo", "Seoul", "NewYork", "LA"] 25

해설 보러가기

solution.py

def solution(cacheSize, cities):

answer = 0

cache = []

for i in range(len(cities)):

found = False

for j in range(len(cache)):

if cities[i].upper() == cache[j].upper():

city = cache.pop(j)

cache.append(city)

answer += 1

found = True

break

if found == False:

answer += 5

if len(cache) < cacheSize:

cache.append(cities[i])

elif cacheSize != 0:

cache.pop(0)

cache.append(cities[i])

return answer

채점 결과

정확성: 50.0

효율성: 0.0

합계: 50.0 / 50

종료까지

00:39:49

**문제 설명 2**

스파이들은 매일 다른 옷을 조합하여 입어 자신을 위장합니다.

예를 들어 스파이가 가진 옷이 아래와 같고 오늘 스파이가 동그란 안경, 긴 코트, 파란색 티셔츠를 입었다면 다음날은 청바지를 추가로 입거나 동그란 안경 대신 검정 선글라스를 착용하거나 해야 합니다.

종류 이름

얼굴 동그란 안경, 검정 선글라스

상의 파란색 티셔츠

하의 청바지

겉옷 긴 코트

스파이가 가진 의상들이 담긴 2차원 배열 clothes가 주어질 때 서로 다른 옷의 조합의 수를 return 하도록 solution 함수를 작성해주세요.

제한사항

clothes의 각 행은 [의상의 이름, 의상의 종류]로 이루어져 있습니다.

스파이가 가진 의상의 수는 1개 이상 30개 이하입니다.

같은 이름을 가진 의상은 존재하지 않습니다.

clothes의 모든 원소는 문자열로 이루어져 있습니다.

모든 문자열의 길이는 1 이상 20 이하인 자연수이고 알파벳 소문자 또는 '\_' 로만 이루어져 있습니다.

스파이는 하루에 최소 한 개의 의상은 입습니다.

입출력 예

clothes return

[["yellowhat", "headgear"], ["bluesunglasses", "eyewear"], ["green\_turban", "headgear"]] 5

[["crowmask", "face"], ["bluesunglasses", "face"], ["smoky\_makeup", "face"]] 3

입출력 예 설명

예제 #1

headgear에 해당하는 의상이 yellow\_hat, green\_turban이고 eyewear에 해당하는 의상이 blue\_sunglasses이므로 아래와 같이 5개의 조합이 가능합니다.

1. yellow\_hat

2. blue\_sunglasses

3. green\_turban

4. yellow\_hat + blue\_sunglasses

5. green\_turban + blue\_sunglasses

예제 #2

face에 해당하는 의상이 crow\_mask, blue\_sunglasses, smoky\_makeup이므로 아래와 같이 3개의 조합이 가능합니다.

1. crow\_mask

2. blue\_sunglasses

3. smoky\_makeup

출처

def solution(clothes):

cloth\_type = {} # Dictionary 선언

for i in range(len(clothes)):

cloth\_type[clothes[i][1]] = 0 # 0으로 초기화

for i in range(len(clothes)):

cloth\_type[clothes[i][1]] += 1

#print(clothes[i][1], cloth\_type[clothes[i][1]])

answer = 1

for name in cloth\_type.keys():

answer \*= cloth\_type[name] + 1

answer -= 1

return answer

**문제 설명 3**

문자열 s가 주어질 때, s의 부분문자열(Substring)중 가장 긴 팰린드롬의 길이를 return 하는 solution 함수를 완성해 주세요.

예를들면, 문자열 s가 "abcdcba"이면 7을 return하고 "abacde"이면 3을 return합니다.

제한사항

문자열 s의 길이 : 2,500 이하의 자연수

문자열 s는 알파벳 소문자로만 구성

입출력 예

s answer

"abcdcba" 7

"abacde" 3

입출력 예 설명

입출력 예 #1

4번째자리 'd'를 기준으로 문자열 s 전체가 팰린드롬이 되므로 7을 return합니다.

입출력 예 #2

2번째자리 'b'를 기준으로 "aba"가 팰린드롬이 되므로 3을 return합니다.

solution.py

def solution(s):

answer = 0

str\_len = len(s)

print("{}, length: {}".format(s, str\_len))

for i in range(str\_len):

for j in range(i + 1, str\_len):

if (i + j) % 2 == 1:

forward\_stop\_index = int((i + j) / 2)

backward\_stop\_index = int((i + j) / 2) + 1

else:

forward\_stop\_index = int((i + j) / 2)

backward\_stop\_index = int((i + j) / 2)

k = i

l = j

#print("k: {}, forward\_stop\_index: {}, l:{} backward\_stop\_index: {}".

# format(k, forward\_stop\_index, l, backward\_stop\_index))

matched = True

while k <= forward\_stop\_index and l >= backward\_stop\_index:

if s[k] == s[l]:

#print("s[{0}] = {1} s[{2}] = {3}".format(k, s[k], l, s[l]))

k = k + 1

l = l - 1

continue

else:

matched = False

break

if matched:

new\_answer = j - i + 1

#print("new\_answer = {}".format(new\_answer))

if new\_answer > answer:

answer = new\_answer

return answer

실행 결과

채점을 시작합니다.

정확성 테스트

테스트 1 〉 통과 (0.06ms, 10.2MB)

테스트 2 〉 통과 (0.06ms, 10.2MB)

테스트 3 〉 통과 (3.08ms, 10.3MB)

테스트 4 〉 통과 (2.83ms, 10.2MB)

테스트 5 〉 통과 (3.16ms, 10.3MB)

테스트 6 〉 통과 (2.81ms, 10.3MB)

테스트 7 〉 통과 (2.86ms, 10.3MB)

테스트 8 〉 통과 (2.53ms, 10.2MB)

테스트 9 〉 통과 (5.21ms, 10.3MB)

테스트 10 〉 통과 (3.22ms, 10.3MB)

테스트 11 〉 통과 (6.85ms, 10.4MB)

테스트 12 〉 통과 (13.26ms, 10.3MB)

테스트 13 〉 통과 (2.19ms, 10.2MB)

테스트 14 〉 통과 (8.37ms, 10.3MB)

테스트 15 〉 통과 (7.80ms, 10.3MB)

테스트 16 〉 통과 (8.67ms, 10.4MB)

테스트 17 〉 실패 (0.01ms, 10.3MB)

테스트 18 〉 실패 (0.01ms, 10.2MB)

테스트 19 〉 통과 (2.48ms, 10.3MB)

테스트 20 〉 통과 (4.86ms, 10.4MB)

테스트 21 〉 통과 (5.92ms, 10.4MB)

효율성 테스트

테스트 1 〉 통과 (1826.54ms, 10.2MB)

테스트 2 〉 실패 (시간 초과)

채점 결과

정확성: 31.4

효율성: 7.7

합계: 39.0 / 50

종료까지

**문제 설명 4**

하드디스크는 한 번에 하나의 작업만 수행할 수 있습니다. 디스크 컨트롤러를 구현하는 방법은 여러 가지가 있습니다. 가장 일반적인 방법은 요청이 들어온 순서대로 처리하는 것입니다.

예를들어

- 0ms 시점에 3ms가 소요되는 A작업 요청

- 1ms 시점에 9ms가 소요되는 B작업 요청

- 2ms 시점에 6ms가 소요되는 C작업 요청

와 같은 요청이 들어왔습니다. 이를 그림으로 표현하면 아래와 같습니다.

한 번에 하나의 요청만을 수행할 수 있기 때문에 각각의 작업을 요청받은 순서대로 처리하면 다음과 같이 처리 됩니다.

- A: 3ms 시점에 작업 완료 (요청에서 종료까지 : 3ms)

- B: 1ms부터 대기하다가, 3ms 시점에 작업을 시작해서 12ms 시점에 작업 완료(요청에서 종료까지 : 11ms)

- C: 2ms부터 대기하다가, 12ms 시점에 작업을 시작해서 18ms 시점에 작업 완료(요청에서 종료까지 : 16ms)

이 때 각 작업의 요청부터 종료까지 걸린 시간의 평균은 10ms(= (3 + 11 + 16) / 3)가 됩니다.

하지만 A → C → B 순서대로 처리하면

- A: 3ms 시점에 작업 완료(요청에서 종료까지 : 3ms)

- C: 2ms부터 대기하다가, 3ms 시점에 작업을 시작해서 9ms 시점에 작업 완료(요청에서 종료까지 : 7ms)

- B: 1ms부터 대기하다가, 9ms 시점에 작업을 시작해서 18ms 시점에 작업 완료(요청에서 종료까지 : 17ms)

이렇게 A → C → B의 순서로 처리하면 각 작업의 요청부터 종료까지 걸린 시간의 평균은 9ms(= (3 + 7 + 17) / 3)가 됩니다.

각 작업에 대해 [작업이 요청되는 시점, 작업의 소요시간]을 담은 2차원 배열 jobs가 매개변수로 주어질 때, 작업의 요청부터 종료까지 걸린 시간의 평균을 가장 줄이는 방법으로 처리하면 평균이 얼마가 되는지 return 하도록 solution 함수를 작성해주세요. (단, 소수점 이하의 수는 버립니다)

제한 사항

jobs의 길이는 1 이상 500 이하입니다.

jobs의 각 행은 하나의 작업에 대한 [작업이 요청되는 시점, 작업의 소요시간] 입니다.

각 작업에 대해 작업이 요청되는 시간은 0 이상 1,000 이하입니다.

각 작업에 대해 작업의 소요시간은 1 이상 1,000 이하입니다.

하드디스크가 작업을 수행하고 있지 않을 때에는 먼저 요청이 들어온 작업부터 처리합니다.

입출력 예

jobs return

[[0, 3], [1, 9], [2, 6]] 9

입출력 예 설명

문제에 주어진 예와 같습니다.

0ms 시점에 3ms 걸리는 작업 요청이 들어옵니다.

1ms 시점에 9ms 걸리는 작업 요청이 들어옵니다.

2ms 시점에 6ms 걸리는 작업 요청이 들어옵니다.

import operator

def find\_next\_job(jobs, prev\_end\_time):

min\_index = -1

min\_take\_time = 1e9

i = 0

while i < len(jobs):

start\_time, take\_time = jobs[i]

if start\_time < prev\_end\_time and take\_time < min\_take\_time:

min\_take\_time = take\_time

min\_index = i

elif start\_time > prev\_end\_time:

break

i += 1

return min\_index

def solution(jobs):

jobs.sort(key=operator.itemgetter(0, 1))

answer = 0

jobs\_len = len(jobs)

print(jobs)

current\_time = 0

while len(jobs) > 0:

next\_job\_index = find\_next\_job(jobs, current\_time)

if next\_job\_index == -1:

start\_time, take\_time = jobs.pop(0)

current\_time = start\_time + take\_time

answer += take\_time

else:

start\_time, take\_time = jobs.pop(next\_job\_index)

if start\_time <= current\_time:

current\_time = current\_time + take\_time

answer += current\_time - start\_time

else:

current\_time = start\_time + take\_time

answer += take\_time

answer = int(answer/jobs\_len)

#print(answer)

return answer

**문제 설명 5**

N개의 아파트가 일렬로 쭉 늘어서 있습니다. 이 중에서 일부 아파트 옥상에는 4g 기지국이 설치되어 있습니다. 기술이 발전해 5g 수요가 높아져 4g 기지국을 5g 기지국으로 바꾸려 합니다. 그런데 5g 기지국은 4g 기지국보다 전달 범위가 좁아, 4g 기지국을 5g 기지국으로 바꾸면 어떤 아파트에는 전파가 도달하지 않습니다.

예를 들어 11개의 아파트가 쭉 늘어서 있고, [4, 11] 번째 아파트 옥상에는 4g 기지국이 설치되어 있습니다. 만약 이 4g 기지국이 전파 도달 거리가 1인 5g 기지국으로 바뀔 경우 모든 아파트에 전파를 전달할 수 없습니다. (전파의 도달 거리가 W일 땐, 기지국이 설치된 아파트를 기준으로 전파를 양쪽으로 W만큼 전달할 수 있습니다.)

초기에, 1, 2, 6, 7, 8, 9번째 아파트에는 전파가 전달되지 않습니다.

기지국설치1\_pvskxt.png

1, 7, 9번째 아파트 옥상에 기지국을 설치할 경우, 모든 아파트에 전파를 전달할 수 있습니다.

기지국설치2\_kml0pb.png

3개의 아파트보다 더 많은 아파트 옥상에 기지국을 설치할 경우에도 모든 아파트에 전파를 전달할 수 있습니다.

기지국설치3\_xhv7r3.png

이때, 우리는 기지국을 최소로 설치하면서 모든 아파트에 전파를 전달하려고 합니다. 위의 예시에선 최소 3개의 아파트 옥상에 기지국을 설치해야 모든 아파트에 전파를 전달할 수 있습니다.

아파트의 개수 N, 현재 기지국이 설치된 아파트의 번호가 담긴 1차원 배열 stations, 전파의 도달 거리 W가 매개변수로 주어질 때, 모든 아파트에 전파를 전달하기 위해 증설해야 할 기지국 개수의 최솟값을 리턴하는 solution 함수를 완성해주세요

제한사항

N: 200,000,000 이하의 자연수

stations의 크기: 10,000 이하의 자연수

stations는 오름차순으로 정렬되어 있고, 배열에 담긴 수는 N보다 같거나 작은 자연수입니다.

W: 10,000 이하의 자연수

입출력 예

N stations W answer

11 [4, 11] 1 3

16 [9] 2 3

입출력 예 설명

입출력 예 #1

문제의 예시와 같습니다

입출력 예 #2

초기에, 1~6, 12~16번째 아파트에는 전파가 전달되지 않습니다.

기지국설치4\_nqfrmm.png

3, 6, 14번째 아파트 옥상에 기지국을 설치할 경우 모든 아파트에 전파를 전달할 수 있습니다.

기지국설치5\_zh4ebk.png

solution.py

https://programmers.co.kr/skill\_checks/301001#

def solution(n, stations, w):

answer = 0

coverage\_unit = 2 \* w + 1

j = 1

for i in range(len(stations)):

start\_index = stations[i] - w

end\_index = stations[i] + w

if j < start\_index:

need\_unit = start\_index - j

if need\_unit % coverage\_unit == 0:

answer += int(need\_unit / coverage\_unit)

else:

answer += int(need\_unit / coverage\_unit) + 1

j = end\_index + 1

if i == (len(stations) - 1):

if j < n:

need\_unit = n - j + 1

if need\_unit % coverage\_unit == 0:

answer += int(need\_unit / coverage\_unit)

else:

answer += int(need\_unit / coverage\_unit) + 1

return answer

실행 결과

채점을 시작합니다.

정확성 테스트

테스트 1 〉 실패 (0.00ms, 10.3MB)

테스트 2 〉 통과 (0.01ms, 10.2MB)

테스트 3 〉 통과 (0.01ms, 10.2MB)

테스트 4 〉 통과 (0.01ms, 10.3MB)

테스트 5 〉 통과 (0.01ms, 10.3MB)

테스트 6 〉 통과 (0.01ms, 10.3MB)

테스트 7 〉 통과 (0.01ms, 10.3MB)

테스트 8 〉 통과 (0.01ms, 10.3MB)

테스트 9 〉 통과 (0.01ms, 10.2MB)

테스트 10 〉 통과 (0.00ms, 10.3MB)

테스트 11 〉 통과 (0.01ms, 10.3MB)

테스트 12 〉 통과 (0.01ms, 10.3MB)

테스트 13 〉 통과 (0.01ms, 10.3MB)

테스트 14 〉 통과 (0.00ms, 10.3MB)

테스트 15 〉 실패 (0.01ms, 10.3MB)

테스트 16 〉 통과 (0.01ms, 10.3MB)

테스트 17 〉 통과 (0.01ms, 10.2MB)

테스트 18 〉 통과 (0.01ms, 10.2MB)

테스트 19 〉 통과 (0.01ms, 10.3MB)

테스트 20 〉 통과 (0.01ms, 10.3MB)

테스트 21 〉 통과 (0.02ms, 10.3MB)

효율성 테스트

테스트 1 〉 통과 (2.99ms, 10.5MB)

테스트 2 〉 통과 (3.84ms, 10.6MB)

테스트 3 〉 통과 (3.36ms, 10.5MB)

테스트 4 〉 통과 (3.58ms, 10.6MB)

채점 결과

정확성: 31.9

효율성: 14.8

합계: 46.6 / 50

종료까지

자물쇠와 열쇠

**문제 설명 6**

고고학자인 "튜브"는 고대 유적지에서 보물과 유적이 가득할 것으로 추정되는 비밀의 문을 발견하였습니다. 그런데 문을 열려고 살펴보니 특이한 형태의 자물쇠로 잠겨 있었고 문 앞에는 특이한 형태의 열쇠와 함께 자물쇠를 푸는 방법에 대해 다음과 같이 설명해 주는 종이가 발견되었습니다.

잠겨있는 자물쇠는 격자 한 칸의 크기가 1 x 1인 N x N 크기의 정사각 격자 형태이고 특이한 모양의 열쇠는 M x M 크기인 정사각 격자 형태로 되어 있습니다.

자물쇠에는 홈이 파여 있고 열쇠 또한 홈과 돌기 부분이 있습니다. 열쇠는 회전과 이동이 가능하며 열쇠의 돌기 부분을 자물쇠의 홈 부분에 딱 맞게 채우면 자물쇠가 열리게 되는 구조입니다. 자물쇠 영역을 벗어난 부분에 있는 열쇠의 홈과 돌기는 자물쇠를 여는 데 영향을 주지 않지만, 자물쇠 영역 내에서는 열쇠의 돌기 부분과 자물쇠의 홈 부분이 정확히 일치해야 하며 열쇠의 돌기와 자물쇠의 돌기가 만나서는 안됩니다. 또한 자물쇠의 모든 홈을 채워 비어있는 곳이 없어야 자물쇠를 열 수 있습니다.

열쇠를 나타내는 2차원 배열 key와 자물쇠를 나타내는 2차원 배열 lock이 매개변수로 주어질 때, 열쇠로 자물쇠를 열수 있으면 true를, 열 수 없으면 false를 return 하도록 solution 함수를 완성해주세요.

제한사항

key는 M x M(3 ≤ M ≤ 20, M은 자연수)크기 2차원 배열입니다.

lock은 N x N(3 ≤ N ≤ 20, N은 자연수)크기 2차원 배열입니다.

M은 항상 N 이하입니다.

key와 lock의 원소는 0 또는 1로 이루어져 있습니다.

0은 홈 부분, 1은 돌기 부분을 나타냅니다.

입출력 예

key lock result

[[0, 0, 0], [1, 0, 0], [0, 1, 1]] [[1, 1, 1], [1, 1, 0], [1, 0, 1]] true

입출력 예에 대한 설명

자물쇠.jpg

key를 시계 방향으로 90도 회전하고, 오른쪽으로 한 칸, 아래로 한 칸 이동하면 lock의 홈 부분을 정확히 모두 채울 수 있습니다.

혼자 풀기가 막막하다면, 풀이 강의를 들어보세요 (클릭)

solution.py

import copy

def rotation(key):

m = len(key)

rotated\_key = [[0] \* m for \_ in range(m)]

#tmp\_key = copy.deepcopy(key)

for i in range(m):

for j in range(m):

rotated\_key[j][m - 1 - i] = key[i][j]

return rotated\_key

def check\_key(key, lock):

m = len(key)

n = len(lock)

# shift\_i from 0 to n + m -2

# shift\_j from 0 to n + m -2

for shift\_i in range(n + m - 1):

for shift\_j in range(n + m - 1):

matched = True

x = m - 1 - shift\_i

y = m - 1 - shift\_j

for i in range(n):

for j in range(n):

if (x + i >= 0 and x + i <= m - 1 and

y + j >= 0 and y + j <= m - 1):

key\_value = key[x + i][y + j]

else:

key\_value = 0

key\_result = key\_value + lock[i][j]

if key\_result != 1:

matched = False

break

if not matched:

break

if matched:

return True

return False

def solution(key, lock):

# Check key without rotation

answer = check\_key(key, lock)

if answer:

return answer

# Check key with 90, 180, 270 degree

for \_ in range(3):

key = rotation(key)

answer = check\_key(key, lock)

if answer:

return answer

return answer실행 결과

채점을 시작합니다.

정확성 테스트

테스트 1 〉 통과 (0.18ms, 10.3MB)

테스트 2 〉 통과 (0.03ms, 10.2MB)

테스트 3 〉 통과 (2.34ms, 10.3MB)

테스트 4 〉 통과 (0.03ms, 10.3MB)

테스트 5 〉 통과 (1.19ms, 10.3MB)

테스트 6 〉 통과 (1.86ms, 10.3MB)

테스트 7 〉 통과 (7.65ms, 10.3MB)

테스트 8 〉 통과 (16.05ms, 10.3MB)

테스트 9 〉 통과 (1.67ms, 10.3MB)

테스트 10 〉 통과 (5.00ms, 10.3MB)

테스트 11 〉 통과 (29.35ms, 10.4MB)

테스트 12 〉 통과 (0.02ms, 10.2MB)

테스트 13 〉 통과 (0.91ms, 10.3MB)

테스트 14 〉 통과 (0.27ms, 10.3MB)

테스트 15 〉 통과 (1.40ms, 10.3MB)

테스트 16 〉 통과 (6.29ms, 10.3MB)

테스트 17 〉 통과 (0.13ms, 10.3MB)

테스트 18 〉 통과 (14.06ms, 10.3MB)

테스트 19 〉 통과 (0.17ms, 10.3MB)

테스트 20 〉 통과 (25.28ms, 10.1MB)

테스트 21 〉 통과 (0.50ms, 10.2MB)

테스트 22 〉 통과 (0.95ms, 10.3MB)

테스트 23 〉 통과 (0.42ms, 10.3MB)

테스트 24 〉 통과 (0.78ms, 10.3MB)

테스트 25 〉 통과 (53.21ms, 10.3MB)

테스트 26 〉 통과 (3.56ms, 10.4MB)

테스트 27 〉 통과 (20.42ms, 10.4MB)

테스트 28 〉 통과 (1.02ms, 10.4MB)

테스트 29 〉 통과 (0.84ms, 10.1MB)

테스트 30 〉 통과 (2.68ms, 10.3MB)

테스트 31 〉 통과 (23.38ms, 10.3MB)

테스트 32 〉 통과 (4.21ms, 10.3MB)

테스트 33 〉 통과 (1.48ms, 10.3MB)

테스트 34 〉 통과 (0.04ms, 10.3MB)

테스트 35 〉 통과 (0.62ms, 10.3MB)

테스트 36 〉 통과 (0.56ms, 10.3MB)

테스트 37 〉 통과 (0.34ms, 10.3MB)

테스트 38 〉 통과 (0.17ms, 10.3MB)

채점 결과

정확성: 100.0

합계: 100.0 / 100.0

**문제 설명 7 – 가장 큰 정사각형 찾기**

1와 0로 채워진 표(board)가 있습니다. 표 1칸은 1 x 1 의 정사각형으로 이루어져 있습니다. 표에서 1로 이루어진 가장 큰 정사각형을 찾아 넓이를 return 하는 solution 함수를 완성해 주세요. (단, 정사각형이란 축에 평행한 정사각형을 말합니다.)

예를 들어

1 2 3 4

0 1 1 1

1 1 1 1

1 1 1 1

0 0 1 0

가 있다면 가장 큰 정사각형은

1 2 3 4

0 1 1 1

1 1 1 1

1 1 1 1

0 0 1 0

가 되며 넓이는 9가 되므로 9를 반환해 주면 됩니다.

제한사항

표(board)는 2차원 배열로 주어집니다.

표(board)의 행(row)의 크기 : 1,000 이하의 자연수

표(board)의 열(column)의 크기 : 1,000 이하의 자연수

표(board)의 값은 1또는 0으로만 이루어져 있습니다.

입출력 예

board answer

[[0,1,1,1],[1,1,1,1],[1,1,1,1],[0,0,1,0]] 9

[[0,0,1,1],[1,1,1,1]] 4

입출력 예 설명

입출력 예 #1

위의 예시와 같습니다.

입출력 예 #2

| 0 | 0 | 1 | 1 |

| 1 | 1 | 1 | 1 |

로 가장 큰 정사각형의 넓이는 4가 되므로 4를 return합니다.

def solution(board):

n = len(board)

m = len(board[0])

all\_zero = True

for i in range(n):

for j in range(m):

if board[i][j]:

all\_zero = False

break

if not all\_zero:

break

if all\_zero:

return 0

answer = 1

for i in range(1, n):

for j in range(1, m):

a = board[i-1][j-1]

b = board[i-1][j]

c = board[i][j-1]

d = board[i][j]

min\_value = min(b, a)

min\_value = min(c, min\_value)

if d > 0:

board[i][j] = min\_value + 1

answer = max(answer, board[i][j])

#print(i, j, board[i][j])

return answer \* answer

실행 결과

채점을 시작합니다.

정확성 테스트

테스트 1 〉 통과 (0.00ms, 10.2MB)

테스트 2 〉 통과 (0.05ms, 10.2MB)

테스트 3 〉 통과 (0.06ms, 10.3MB)

테스트 4 〉 통과 (0.06ms, 10.2MB)

테스트 5 〉 통과 (0.06ms, 10.3MB)

테스트 6 〉 통과 (0.02ms, 10.3MB)

테스트 7 〉 통과 (0.02ms, 10.4MB)

테스트 8 〉 통과 (0.01ms, 10.3MB)

테스트 9 〉 통과 (0.02ms, 10.3MB)

테스트 10 〉 통과 (0.05ms, 10.3MB)

테스트 11 〉 통과 (0.02ms, 10.2MB)

테스트 12 〉 통과 (0.03ms, 10.2MB)

테스트 13 〉 통과 (0.02ms, 10.3MB)

테스트 14 〉 통과 (0.05ms, 10.3MB)

테스트 15 〉 통과 (0.06ms, 10.3MB)

테스트 16 〉 통과 (0.06ms, 10.3MB)

테스트 17 〉 통과 (0.06ms, 10.3MB)

테스트 18 〉 통과 (1.49ms, 10.2MB)

테스트 19 〉 통과 (1.78ms, 10.2MB)

효율성 테스트

테스트 1 〉 통과 (632.36ms, 31.2MB)

테스트 2 〉 통과 (703.83ms, 30.6MB)

테스트 3 〉 통과 (703.23ms, 30.9MB)

채점 결과

정확성: 59.6

효율성: 40.4

합계: 100.0 / 100.0

**문제 설명 8 – 순간이동**

OO 연구소는 한 번에 K 칸을 앞으로 점프하거나, (현재까지 온 거리) x 2 에 해당하는 위치로 순간이동을 할 수 있는 특수한 기능을 가진 아이언 슈트를 개발하여 판매하고 있습니다. 이 아이언 슈트는 건전지로 작동되는데, 순간이동을 하면 건전지 사용량이 줄지 않지만, 앞으로 K 칸을 점프하면 K 만큼의 건전지 사용량이 듭니다. 그러므로 아이언 슈트를 착용하고 이동할 때는 순간 이동을 하는 것이 더 효율적입니다. 아이언 슈트 구매자는 아이언 슈트를 착용하고 거리가 N 만큼 떨어져 있는 장소로 가려고 합니다. 단, 건전지 사용량을 줄이기 위해 점프로 이동하는 것은 최소로 하려고 합니다. 아이언 슈트 구매자가 이동하려는 거리 N이 주어졌을 때, 사용해야 하는 건전지 사용량의 최솟값을 return하는 solution 함수를 만들어 주세요.

예를 들어 거리가 5만큼 떨어져 있는 장소로 가려고 합니다.

아이언 슈트를 입고 거리가 5만큼 떨어져 있는 장소로 갈 수 있는 경우의 수는 여러 가지입니다.

처음 위치 0 에서 5 칸을 앞으로 점프하면 바로 도착하지만, 건전지 사용량이 5 만큼 듭니다.

처음 위치 0 에서 2 칸을 앞으로 점프한 다음 순간이동 하면 (현재까지 온 거리 : 2) x 2에 해당하는 위치로 이동할 수 있으므로 위치 4로 이동합니다. 이때 1 칸을 앞으로 점프하면 도착하므로 건전지 사용량이 3 만큼 듭니다.

처음 위치 0 에서 1 칸을 앞으로 점프한 다음 순간이동 하면 (현재까지 온 거리 : 1) x 2에 해당하는 위치로 이동할 수 있으므로 위치 2로 이동됩니다. 이때 다시 순간이동 하면 (현재까지 온 거리 : 2) x 2 만큼 이동할 수 있으므로 위치 4로 이동합니다. 이때 1 칸을 앞으로 점프하면 도착하므로 건전지 사용량이 2 만큼 듭니다.

위의 3가지 경우 거리가 5만큼 떨어져 있는 장소로 가기 위해서 3번째 경우가 건전지 사용량이 가장 적으므로 답은 2가 됩니다.

제한 사항

숫자 N: 1 이상 10억 이하의 자연수

숫자 K: 1 이상의 자연수

입출력 예

N result

5 2

6 2

5000 5

입출력 예 설명

입출력 예 #1

위의 예시와 같습니다.

입출력 예 #2

처음 위치 0 에서 1 칸을 앞으로 점프한 다음 순간이동 하면 (현재까지 온 거리 : 1) x 2에 해당하는 위치로 이동할 수 있으므로 위치 2로 이동합니다. 이때 1 칸을 앞으로 점프하면 위치3으로 이동합니다. 이때 다시 순간이동 하면 (현재까지 온 거리 : 3) x 2 이동할 수 있으므로 위치 6에 도착합니다. 이 경우가 건전지 사용량이 가장 적으므로 2를 반환해주면 됩니다.

입출력 예 #3

위와 같은 방식으로 합니다.

solution.py

https://programmers.co.kr/skill\_checks/301641#

def check\_cost(m):

if m == 1:

return 1

if m % 2 == 0:

return check\_cost(int(m/2))

else:

return check\_cost(m - 1) + 1

def solution(n):

ans = check\_cost(n)

return ans;

실행 결과

채점을 시작합니다.

정확성 테스트

테스트 1 〉 통과 (0.00ms, 10.2MB)

테스트 2 〉 통과 (0.00ms, 10.2MB)

테스트 3 〉 통과 (0.00ms, 10.2MB)

테스트 4 〉 통과 (0.01ms, 10.2MB)

테스트 5 〉 통과 (0.01ms, 10.2MB)

테스트 6 〉 통과 (0.01ms, 10.2MB)

테스트 7 〉 통과 (0.01ms, 10.2MB)

테스트 8 〉 통과 (0.01ms, 10.2MB)

테스트 9 〉 통과 (0.01ms, 10.3MB)

테스트 10 〉 통과 (0.01ms, 10.2MB)

테스트 11 〉 통과 (0.01ms, 10.2MB)

테스트 12 〉 통과 (0.01ms, 10.2MB)

테스트 13 〉 통과 (0.01ms, 10.2MB)

테스트 14 〉 통과 (0.01ms, 10.2MB)

테스트 15 〉 통과 (0.01ms, 10.2MB)

테스트 16 〉 통과 (0.01ms, 10.3MB)

테스트 17 〉 통과 (0.01ms, 10.3MB)

테스트 18 〉 통과 (0.01ms, 10.3MB)

효율성 테스트

테스트 1 〉 통과 (0.03ms, 10.2MB)

테스트 2 〉 통과 (0.03ms, 10.1MB)

테스트 3 〉 통과 (0.02ms, 10.2MB)

테스트 4 〉 통과 (0.02ms, 10.1MB)

테스트 5 〉 통과 (0.02ms, 10.2MB)

테스트 6 〉 통과 (0.02ms, 10.1MB)

테스트 7 〉 통과 (0.02ms, 10.3MB)

테스트 8 〉 통과 (0.01ms, 10.2MB)

테스트 9 〉 통과 (0.01ms, 10.1MB)

테스트 10 〉 통과 (0.01ms, 10.2MB)

채점 결과

정확성: 30.0

효율성: 20.0

합계: 50.0 / 50

종료까지

**문제 설명 8 - 124나라**

124 나라가 있습니다. 124 나라에서는 10진법이 아닌 다음과 같은 자신들만의 규칙으로 수를 표현합니다.

124 나라에는 자연수만 존재합니다.

124 나라에는 모든 수를 표현할 때 1, 2, 4만 사용합니다.

예를 들어서 124 나라에서 사용하는 숫자는 다음과 같이 변환됩니다.

10진법 124 나라 10진법 124 나라

1 1 6 14

2 2 7 21

3 4 8 22

4 11 9 24

5 12 10 41

자연수 n이 매개변수로 주어질 때, n을 124 나라에서 사용하는 숫자로 바꾼 값을 return 하도록 solution 함수를 완성해 주세요.

제한사항

n은 500,000,000이하의 자연수 입니다.

입출력 예

n result

1 1

2 2

3 4

4 11

def change124(n):

if n<=3:

return '124'[n-1]

else:

q, r = divmod(n-1, 3)

return change124(q) + '124'[r]

print(change124(10))

def get\_digit\_str(units, unit\_index, sequence, number):

#print(units, unit\_index, sequence)

if unit\_index == 0:

return str(number[sequence])

index = int(sequence / units[unit\_index - 1])

sequence = sequence % units[unit\_index - 1]

#print(units, unit\_index, index, sequence)

result = str(number[index]) + get\_digit\_str(units, unit\_index - 1, sequence, number)

return result

def solution(n):

number = [1, 2, 4]

units = [3]

max\_value = 3

i = 1

while True:

if n <= max\_value:

break

max\_value += units[i - 1] \* 3

units.append(units[i - 1] \* 3)

i += 1

prev\_max\_value = max\_value - units[i - 1]

sequence = n - prev\_max\_value - 1

start\_unit\_index = i - 1

#print(max\_value, units, i - 1, prev\_max\_value, sequence)

answer = get\_digit\_str(units, start\_unit\_index, sequence, number)

#print("answer: {}".format(answer))

return answer

**문제 설명 – 프린터**

일반적인 프린터는 인쇄 요청이 들어온 순서대로 인쇄합니다. 그렇기 때문에 중요한 문서가 나중에 인쇄될 수 있습니다. 이런 문제를 보완하기 위해 중요도가 높은 문서를 먼저 인쇄하는 프린터를 개발했습니다. 이 새롭게 개발한 프린터는 아래와 같은 방식으로 인쇄 작업을 수행합니다.

1. 인쇄 대기목록의 가장 앞에 있는 문서(J)를 대기목록에서 꺼냅니다.

2. 나머지 인쇄 대기목록에서 J보다 중요도가 높은 문서가 한 개라도 존재하면 J를 대기목록의 가장 마지막에 넣습니다.

3. 그렇지 않으면 J를 인쇄합니다.

예를 들어, 4개의 문서(A, B, C, D)가 순서대로 인쇄 대기목록에 있고 중요도가 2 1 3 2 라면 C D A B 순으로 인쇄하게 됩니다.

내가 인쇄를 요청한 문서가 몇 번째로 인쇄되는지 알고 싶습니다. 위의 예에서 C는 1번째로, A는 3번째로 인쇄됩니다.

현재 대기목록에 있는 문서의 중요도가 순서대로 담긴 배열 priorities와 내가 인쇄를 요청한 문서가 현재 대기목록의 어떤 위치에 있는지를 알려주는 location이 매개변수로 주어질 때, 내가 인쇄를 요청한 문서가 몇 번째로 인쇄되는지 return 하도록 solution 함수를 작성해주세요.

제한사항

현재 대기목록에는 1개 이상 100개 이하의 문서가 있습니다.

인쇄 작업의 중요도는 1~9로 표현하며 숫자가 클수록 중요하다는 뜻입니다.

location은 0 이상 (현재 대기목록에 있는 작업 수 - 1) 이하의 값을 가지며 대기목록의 가장 앞에 있으면 0, 두 번째에 있으면 1로 표현합니다.

입출력 예

priorities location return

[2, 1, 3, 2] 2 1

[1, 1, 9, 1, 1, 1] 0 5

입출력 예 설명

예제 #1

문제에 나온 예와 같습니다.

예제 #2

6개의 문서(A, B, C, D, E, F)가 인쇄 대기목록에 있고 중요도가 1 1 9 1 1 1 이므로 C D E F A B 순으로 인쇄합니다.

출처

solution.py

from collections import deque

def solution(priorities, location):

q = deque()

# q = [ (index, priority) for index, priority in enumerate(priorities)]

# q.pop(0)으로 queue처럼 사용 가능

for index, priority in enumerate(priorities):

q.append((index, priority))

priorities.sort(reverse=True)

sequence = 1

while q:

index, priority = q.popleft()

#print(index, priority)

if priority == priorities[0]:

if location == index:

return sequence

priorities.pop(0)

sequence += 1

else:

q.append((index, priority))

return sequence

실행 결과

채점을 시작합니다.

정확성 테스트

테스트 1 〉 통과 (0.04ms, 10.3MB)

테스트 2 〉 통과 (0.12ms, 10.3MB)

테스트 3 〉 통과 (0.04ms, 10.3MB)

테스트 4 〉 통과 (0.02ms, 10.2MB)

테스트 5 〉 통과 (0.01ms, 10.4MB)

테스트 6 〉 통과 (0.03ms, 10.2MB)

테스트 7 〉 통과 (0.04ms, 10.2MB)

테스트 8 〉 통과 (0.08ms, 10.4MB)

테스트 9 〉 통과 (0.01ms, 10.3MB)

테스트 10 〉 통과 (0.05ms, 10.2MB)

테스트 11 〉 통과 (0.08ms, 10.2MB)

테스트 12 〉 통과 (0.02ms, 10.2MB)

테스트 13 〉 통과 (0.10ms, 10.2MB)

테스트 14 〉 통과 (0.01ms, 10.3MB)

테스트 15 〉 통과 (0.01ms, 10.2MB)

테스트 16 〉 통과 (0.02ms, 10.2MB)

테스트 17 〉 통과 (0.13ms, 10.2MB)

테스트 18 〉 통과 (0.02ms, 10.3MB)

테스트 19 〉 통과 (0.13ms, 10.2MB)

테스트 20 〉 통과 (0.03ms, 10.2MB)

채점 결과

정확성: 100.0

합계: 100.0 / 100.0

**문제 설명 – 소수 찾기**

한자리 숫자가 적힌 종이 조각이 흩어져 있습니다. 흩어진 종이 조각을 붙여 소수를 몇 개 만들 수 있는지 알아내려 합니다.

각 종이 조각에 적힌 숫자가 적힌 문자열 numbers가 주어졌을 때, 종이 조각으로 만들 수 있는 소수가 몇 개인지 return 하도록 solution 함수를 완성해주세요.

제한사항

numbers는 길이 1 이상 7 이하인 문자열입니다.

numbers는 0~9까지 숫자만으로 이루어져 있습니다.

"013"은 0, 1, 3 숫자가 적힌 종이 조각이 흩어져 있다는 의미입니다.

입출력 예

numbers return

"17" 3

"011" 2

입출력 예 설명

예제 #1

[1, 7]으로는 소수 [7, 17, 71]를 만들 수 있습니다.

예제 #2

[0, 1, 1]으로는 소수 [11, 101]를 만들 수 있습니다.

11과 011은 같은 숫자로 취급합니다.

출처

solution.py

from itertools import permutations

def is\_prime\_number(value):

if value == 1:

return False

if value == 2:

return True

for num in range(2, value):

if value % num == 0:

return False

return True

​

def solution(numbers):

answer = 0

permute\_list = []

for i in range(1, len(numbers) + 1):

permute = permutations(numbers, i)

permute\_list.append(set(permute))

#print(permute\_list)

num\_str\_list = []

for permute in permute\_list:

for element in permute:

num\_str = ''

skip\_zero = True

for i in range(len(element)):

if element[i] == '0' and skip\_zero:

continue

else:

skip\_zero = False

num\_str += element[i]

if num\_str != '':

num\_str\_list.append(num\_str)

print(set(num\_str\_list))

for num\_str in set(num\_str\_list):

if is\_prime\_number(int(num\_str)):

answer += 1

return answer

실행 결과

같은 코드로 채점한 결과가 있습니다.

정확성 테스트

테스트 1 〉 통과 (0.35ms, 10.4MB)

테스트 2 〉 통과 (160.04ms, 10.8MB)

테스트 3 〉 통과 (0.05ms, 10.4MB)

테스트 4 〉 통과 (11.63ms, 10.5MB)

테스트 5 〉 통과 (1.56ms, 10.4MB)

테스트 6 〉 통과 (0.04ms, 10.4MB)

테스트 7 〉 통과 (0.11ms, 10.4MB)

테스트 8 〉 통과 (2.40ms, 10.4MB)

테스트 9 〉 통과 (0.08ms, 10.4MB)

테스트 10 〉 통과 (2603.78ms, 10.8MB)

테스트 11 〉 통과 (123.19ms, 10.5MB)

테스트 12 〉 통과 (0.74ms, 10.5MB)

채점 결과

정확성: 100.0

합계: 100.0 / 100.0

**순열 만들기**

primeSet = set()

def isPrime(number):

if number in (0, 1):

return False

for i in range(2, number):

if number % i == 0:

return False

return True

def permute(str1, str2):

if str1 != "":

if isPrime(int(str1)):

primeSet.add(int(str1))

for i in range(len(str2)):

permute(str1 + str2[i], str2[:i] + str2[i + 1:])

def solution(numbers):

permute("", numbers)

answer = len(primeSet)

return answer

**문제 설명 - 가장 먼 노드**

n개의 노드가 있는 그래프가 있습니다. 각 노드는 1부터 n까지 번호가 적혀있습니다. 1번 노드에서 가장 멀리 떨어진 노드의 갯수를 구하려고 합니다. 가장 멀리 떨어진 노드란 최단경로로 이동했을 때 간선의 개수가 가장 많은 노드들을 의미합니다.

노드의 개수 n, 간선에 대한 정보가 담긴 2차원 배열 vertex가 매개변수로 주어질 때, 1번 노드로부터 가장 멀리 떨어진 노드가 몇 개인지를 return 하도록 solution 함수를 작성해주세요.

제한사항

노드의 개수 n은 2 이상 20,000 이하입니다.

간선은 양방향이며 총 1개 이상 50,000개 이하의 간선이 있습니다.

vertex 배열 각 행 [a, b]는 a번 노드와 b번 노드 사이에 간선이 있다는 의미입니다.

입출력 예

n vertex return

6 [[3, 6], [4, 3], [3, 2], [1, 3], [1, 2], [2, 4], [5, 2]] 3

입출력 예 설명

예제의 그래프를 표현하면 아래 그림과 같고, 1번 노드에서 가장 멀리 떨어진 노드는 4,5,6번 노드입니다.

from collections import deque

def solution(n, edge):

INF = 1e9

dist = [INF] \* (n + 1)

nodes = [[] for \_ in range(n + 1)]

for start, end in edge:

nodes[start].append(end)

nodes[end].append(start)

q = deque()

q.append((0, 1))

while q:

distance, n = q.popleft()

if dist[n] > distance:

dist[n] = distance

for next\_node in nodes[n]:

if distance + 1 < dist[next\_node]:

q.append((distance + 1, next\_node))

max\_value = max(dist[1:])

answer = dist[1:].count(max\_value)

return answer

실행 결과

채점을 시작합니다.

정확성 테스트

테스트 1 〉 통과 (0.02ms, 10.3MB)

테스트 2 〉 통과 (0.02ms, 10.2MB)

테스트 3 〉 통과 (0.06ms, 10.2MB)

테스트 4 〉 통과 (0.28ms, 10.3MB)

테스트 5 〉 통과 (1.83ms, 10.5MB)

테스트 6 〉 통과 (3.02ms, 11.1MB)

테스트 7 〉 통과 (49.41ms, 18.2MB)

테스트 8 〉 통과 (53.32ms, 20.5MB)

테스트 9 〉 통과 (70.25ms, 22.1MB)

채점 결과

정확성: 100.0

합계: 100.0 / 100.0

**문제 설명 – H-Index**

H-Index는 과학자의 생산성과 영향력을 나타내는 지표입니다. 어느 과학자의 H-Index를 나타내는 값인 h를 구하려고 합니다. 위키백과1에 따르면, H-Index는 다음과 같이 구합니다.

어떤 과학자가 발표한 논문 n편 중, h번 이상 인용된 논문이 h편 이상이고 나머지 논문이 h번 이하 인용되었다면 h의 최댓값이 이 과학자의 H-Index입니다.

어떤 과학자가 발표한 논문의 인용 횟수를 담은 배열 citations가 매개변수로 주어질 때, 이 과학자의 H-Index를 return 하도록 solution 함수를 작성해주세요.

제한사항

과학자가 발표한 논문의 수는 1편 이상 1,000편 이하입니다.

논문별 인용 횟수는 0회 이상 10,000회 이하입니다.

입출력 예

citations return

[3, 0, 6, 1, 5] 3

입출력 예 설명

이 과학자가 발표한 논문의 수는 5편이고, 그중 3편의 논문은 3회 이상 인용되었습니다. 그리고 나머지 2편의 논문은 3회 이하 인용되었기 때문에 이 과학자의 H-Index는 3입니다.

※ 공지 - 2019년 2월 28일 테스트 케이스가 추가되었습니다.

solution.py

def solution(citations):

citations.sort(reverse=True)

papers\_num = len(citations)

#print(papers\_num)

for i in range(papers\_num):

cite = citations[i]

# i + 1 means the number of paper

if i + 1 > cite:

return i

elif i + 1 == cite:

return i + 1

return papers\_num

실행 결과

채점을 시작합니다.

정확성 테스트

테스트 1 〉 통과 (0.09ms, 10.1MB)

테스트 2 〉 통과 (0.25ms, 10.1MB)

테스트 3 〉 통과 (0.14ms, 10.2MB)

테스트 4 〉 통과 (0.18ms, 10.2MB)

테스트 5 〉 통과 (0.17ms, 10.2MB)

테스트 6 〉 통과 (0.25ms, 10.2MB)

테스트 7 〉 통과 (0.09ms, 10.2MB)

테스트 8 〉 통과 (0.02ms, 10.1MB)

테스트 9 〉 통과 (0.05ms, 10.2MB)

테스트 10 〉 통과 (0.12ms, 10.2MB)

테스트 11 〉 통과 (0.27ms, 10.1MB)

테스트 12 〉 통과 (0.03ms, 10.1MB)

테스트 13 〉 통과 (0.28ms, 10.2MB)

테스트 14 〉 통과 (0.19ms, 10.3MB)

테스트 15 〉 통과 (0.26ms, 10.2MB)

테스트 16 〉 통과 (0.01ms, 10.2MB)

채점 결과

정확성: 50.0

효율성: 0.0

합계: 50.0 / 50

**문제 설명 – 주식가격**

초 단위로 기록된 주식가격이 담긴 배열 prices가 매개변수로 주어질 때, 가격이 떨어지지 않은 기간은 몇 초인지를 return 하도록 solution 함수를 완성하세요.

제한사항

prices의 각 가격은 1 이상 10,000 이하인 자연수입니다.

prices의 길이는 2 이상 100,000 이하입니다.

입출력 예

prices return

[1, 2, 3, 2, 3] [4, 3, 1, 1, 0]

입출력 예 설명

1초 시점의 ₩1은 끝까지 가격이 떨어지지 않았습니다.

2초 시점의 ₩2은 끝까지 가격이 떨어지지 않았습니다.

3초 시점의 ₩3은 1초뒤에 가격이 떨어집니다. 따라서 1초간 가격이 떨어지지 않은 것으로 봅니다.

4초 시점의 ₩2은 1초간 가격이 떨어지지 않았습니다.

5초 시점의 ₩3은 0초간 가격이 떨어지지 않았습니다.

※ 공지 - 2019년 2월 28일 지문이 리뉴얼되었습니다.

solution.py

1

def solution(prices):

2

answer = []

3

return answer

실행 결과

실행 결과가 여기에 표시됩니다.

문제 설명 - 주식가격 (기준 시점의 가격보다 떨어질 때까지 걸린 시간을 구하는 문제)

초 단위로 기록된 주식가격이 담긴 배열 prices가 매개변수로 주어질 때, 가격이 떨어지지 않은 기간은 몇 초인지를 return 하도록 solution 함수를 완성하세요.

제한사항

prices의 각 가격은 1 이상 10,000 이하인 자연수입니다.

prices의 길이는 2 이상 100,000 이하입니다.

입출력 예

prices return

[1, 2, 3, 2, 3] [4, 3, 1, 1, 0]

입출력 예 설명

1초 시점의 ₩1은 끝까지 가격이 떨어지지 않았습니다.

2초 시점의 ₩2은 끝까지 가격이 떨어지지 않았습니다.

3초 시점의 ₩3은 1초뒤에 가격이 떨어집니다. 따라서 1초간 가격이 떨어지지 않은 것으로 봅니다.

4초 시점의 ₩2은 1초간 가격이 떨어지지 않았습니다.

5초 시점의 ₩3은 0초간 가격이 떨어지지 않았습니다.

※ 공지 - 2019년 2월 28일 지문이 리뉴얼되었습니다.

solution.py

def solution(prices):

answer = []

price\_len = len(prices)

for i in range(price\_len):

j = i + 1

while j < price\_len:

if prices[j] < prices[i]:

j += 1

break

else:

j += 1

answer.append(j - i - 1)

#print(answer)

return answer

실행 결과

같은 코드로 채점한 결과가 있습니다.

정확성 테스트

테스트 1 〉 통과 (0.00ms, 10.2MB)

테스트 2 〉 통과 (0.03ms, 10.2MB)

테스트 3 〉 통과 (0.45ms, 10.3MB)

테스트 4 〉 통과 (0.51ms, 10.3MB)

테스트 5 〉 통과 (0.65ms, 10.3MB)

테스트 6 〉 통과 (0.02ms, 10.1MB)

테스트 7 〉 통과 (0.29ms, 10.3MB)

테스트 8 〉 통과 (0.34ms, 10.3MB)

테스트 9 〉 통과 (0.02ms, 10.2MB)

테스트 10 〉 통과 (0.66ms, 10.3MB)

효율성 테스트

테스트 1 〉 통과 (109.56ms, 18.8MB)

테스트 2 〉 통과 (84.47ms, 17.5MB)

테스트 3 〉 통과 (135.74ms, 19.5MB)

테스트 4 〉 통과 (85.27ms, 18.2MB)

테스트 5 〉 통과 (60.13ms, 17MB)

채점 결과

정확성: 66.7

효율성: 33.3

합계: 100.0 / 100.0

문제 설명 - 카펫

Leo는 카펫을 사러 갔다가 아래 그림과 같이 중앙에는 노란색으로 칠해져 있고 테두리 1줄은 갈색으로 칠해져 있는 격자 모양 카펫을 봤습니다.

Leo는 집으로 돌아와서 아까 본 카펫의 노란색과 갈색으로 색칠된 격자의 개수는 기억했지만, 전체 카펫의 크기는 기억하지 못했습니다.

Leo가 본 카펫에서 갈색 격자의 수 brown, 노란색 격자의 수 yellow가 매개변수로 주어질 때 카펫의 가로, 세로 크기를 순서대로 배열에 담아 return 하도록 solution 함수를 작성해주세요.

제한사항

갈색 격자의 수 brown은 8 이상 5,000 이하인 자연수입니다.

노란색 격자의 수 yellow는 1 이상 2,000,000 이하인 자연수입니다.

카펫의 가로 길이는 세로 길이와 같거나, 세로 길이보다 깁니다.

입출력 예

brown yellow return

10 2 [4, 3]

8 1 [3, 3]

24 24 [8, 6]

출처

※ 공지 - 2020년 2월 3일 테스트케이스가 추가되었습니다.

※ 공지 - 2020년 5월 11일 웹접근성을 고려하여 빨간색을 노란색으로 수정하였습니다.

solution.py

def solution(brown, yellow):

answer = [1, 1]

yellow\_x = 1

while True:

if yellow % yellow\_x == 0:

yellow\_y = int(yellow / yellow\_x)

brown\_cnt = (yellow\_x + 2) \* 2 + yellow\_y \* 2

if brown\_cnt == brown:

x = yellow\_x + 2

y = yellow\_y + 2

answer = [y, x]

break

yellow\_x += 1

return answer

실행 결과

같은 코드로 채점한 결과가 있습니다.

정확성 테스트

테스트 1 〉 통과 (0.01ms, 10.2MB)

테스트 2 〉 통과 (0.00ms, 10.3MB)

테스트 3 〉 통과 (0.11ms, 10.1MB)

테스트 4 〉 통과 (0.00ms, 10.3MB)

테스트 5 〉 통과 (0.00ms, 10.3MB)

테스트 6 〉 통과 (0.03ms, 10.2MB)

테스트 7 〉 통과 (0.06ms, 10.2MB)

테스트 8 〉 통과 (0.07ms, 10.2MB)

테스트 9 〉 통과 (0.07ms, 10.2MB)

테스트 10 〉 통과 (0.09ms, 10.2MB)

테스트 11 〉 통과 (0.00ms, 10.2MB)

테스트 12 〉 통과 (0.00ms, 9.97MB)

테스트 13 〉 통과 (0.01ms, 10.3MB)

채점 결과

정확성: 100.0

합계: 100.0 / 100.0