



합

더 맵게

- Leo는 모든 음식의 스코빌 지수를 K 이상으로 만들고 싶다
- 모든 음식의 스코빌 지수가 K 이상이 될 때까지 음식을 섞는다
- 모든 음식의 스코빌 지수를 K 이상으로 만들 수 없다면 return -1

scoville : 모든 음식들의 스코빌 지수 리스트

섞은 음식의 스코빌 지수 = 가장 맵지 않은 음식의 스코빌 지수 + (두번째로 맵지 않은 음식의 스코빌 지수)*2

- 자료구조 배우게 해놓고 정작 min heap 하나하나 구현해놓으면 효율성 테스트 통과 못함
 - 그래도 heap 만드는데 성공해서 기분은 좋다...^^ㅎ

▼ 내 풀이(min heap 구현)

```
import math

def checkifdone(scoville, K):
    if scoville[0] < K:
        return False
    return True

def swap(scoville, idx1, idx2):
    temp = scoville[idx1]
    scoville[idx1] = scoville[idx2]
    scoville[idx2] = temp

def makeheap(scoville):
    length = len(scoville)
    for i in range(math.floor(length/2)-1, -1, -1):
        min_heapify(scoville, i)

def min_heapify(scoville, crntidx):
    leftnode = 2 * crntidx + 1
    rightnode = 2 * crntidx + 2
    length = len(scoville)

    if leftnode > length-1 : # crnt has no child
        return

    if rightnode > length - 1: # crnt node has no right child
        if scoville[leftnode] < scoville[crntidx]:
            swap(scoville, leftnode, crntidx)
    else : # crnt node has both child
        if scoville[leftnode] < scoville[rightnode] and scoville[leftnode] < scoville[crntidx]:
            # left node is smaller than crnt node
            swap(scoville, leftnode, crntidx)
            min_heapify(scoville, leftnode)
        elif scoville[rightnode] < scoville[leftnode] and scoville[rightnode] < scoville[crntidx]:
            # right node is smaller than crnt node
            swap(scoville, rightnode, crntidx)
            min_heapify(scoville, rightnode)
```

```

def extract_min(scoville):
    extracted = scoville[0]
    swap(scoville, 0, len(scoville)-1)
    scoville.pop()
    makeheap(scoville)
    return extracted

def mixelement(scoville, elem1, elem2):
    newelem = elem1 + elem2 * 2
    scoville.append(newelem)
    makeheap(scoville)

def solution(scoville, K):
    answer = 0
    makeheap(scoville)

    while checkifdone(scoville, K) == False:
        makeheap(scoville)
        element1 = extract_min(scoville)
        element2 = extract_min(scoville)
        mixelement(scoville, element1, element2)

        answer += 1

    if len(scoville) == 1 and scoville[0] < K:
        # len(scoville) = 1 이어도 마지막 남은 음식이 K를 넘는다면 성공했으므로
        # 마지막 남은 음식의 스코빌 지수가 K보다 작을 때만 -1을 리턴한다
        return -1

    return answer

```

- ★ **heapq 라이브러리를 사용한 풀이(존내간단)**

- heapq : binary tree 기반의 min heap 자료구조
- 웬진 모르겠는데 이걸로 하니까 정확성 테스트 12번도 통과함
- 원소들이 항상 정렬된 상태로 추가되고 삭제됨

▼ heappush : min heap을 유지하면서 원소를 추가

```
heappush(heap, 4)
```

▼ heappop : 가장 작은 원소(root)를 삭제하고 min heap을 유지

```

heappop(heap)

# 최소값 삭제하고 얻기
root = heappop(heap)
# 최소값 삭제하지 않고 얻기
root = heap[0]

```

```

from heapq import heapify, heappush, heappop

def checkifdone(scoville, K):
    if scoville[0] < K:
        return False
    return True

def solution(scoville, K):
    answer = 0

```

```

heapify(scoville)

while checkifdone(scoville, K) == False:
    element1 = heappop(scoville)
    element2 = heappop(scoville)

    newelem = element1 + element2 * 2
    heappush(scoville, newelem)
    answer += 1


    if len(scoville) == 1 and scoville[0] < K:
        # len(scoville) = 1 이어도 마지막 남은 음식이 K를 넘는다면 return answer
        return -1

return answer

```

파이썬의 heapq 모듈로 힙 자료구조 사용하기

데이터를 정렬된 상태로 저장하기 위해서 사용하는 파이썬의 heapq(힙큐) 내장 모듈에 대해서 알아보겠습니다. heapq 모듈은 이진 트리(binary tree) 기반의 최소 힙(min heap) 자료구조를 제공합니다. 자바에 익숙하신 분이라면 PriorityQueue 클래스를 생각하시면 이해가 쉬우

 <https://www.daleseo.com/python-heapq/>

< DaleSeo / >