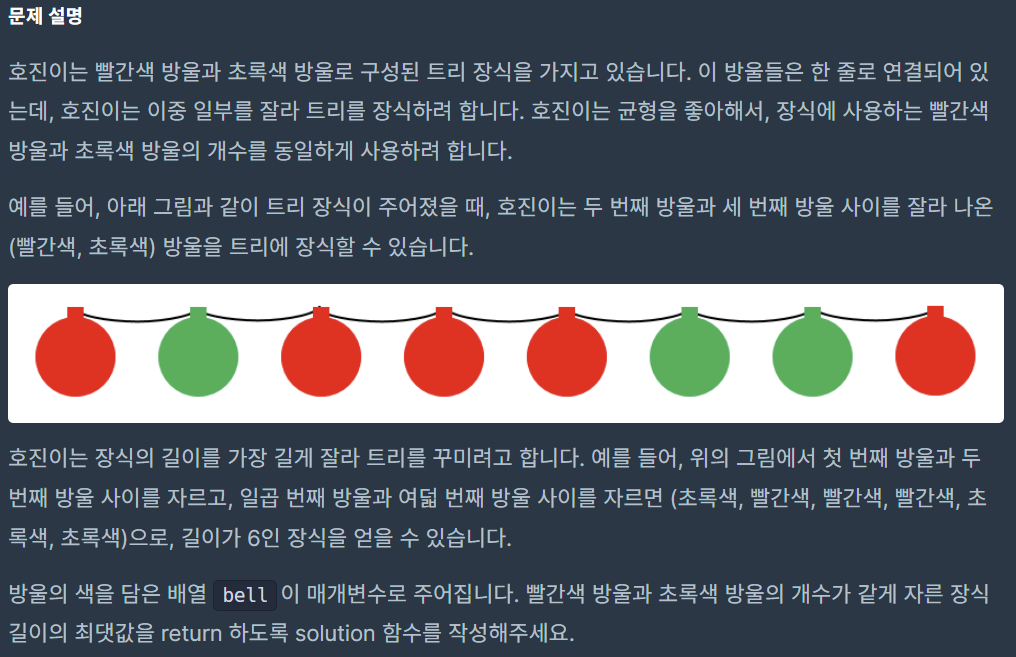
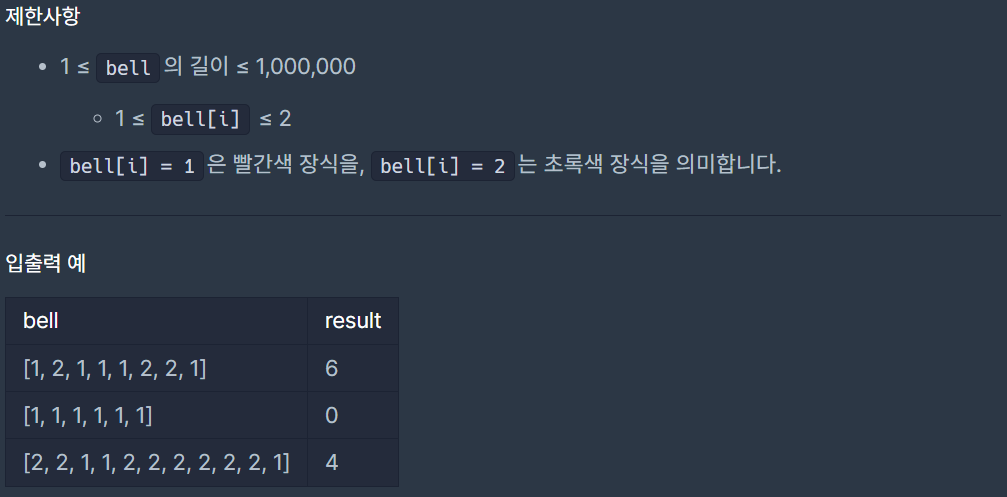
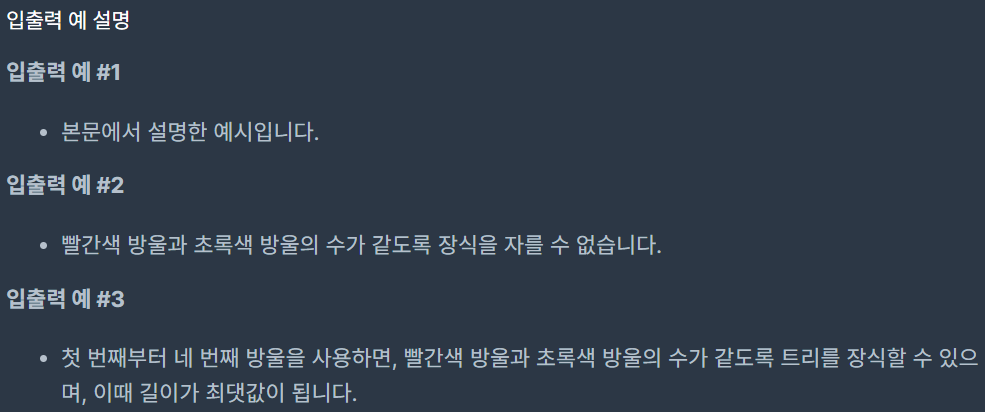
[1] 방울







### (1)지문 이해 및 풀이 계획

예시로 [1, 2, 1, 1, 1, 2, 2, 1] 이 주어졌을 때,

1. 1을 -1로, 2을 1로 치환합니다. => [-1, 1, -1, -1, -1, 1, 1, -1]
2. 누적합을 구합니다. => [0, 0 + -1, 0 + -1 + 1, 0 + -1 + 1 + -1, ...] => [0, -1, 0, -1, -2, -3, -2, -1, -2]
3. 이 누적합의 i번재 요소는 1~i번째 방울까지 봤을떄 초록방울이 얼마가 더 많았냐 의미합니다. 예를 들어 i=3일때 누적합[i]는 -1이며 이는 1~3번째 방울까지 봤을때 빨간색이 초록보다 1개 더 많음을 의미합니다.
4. 따라서 3번째 방울부터 시작해서 구간을 설정할때 색의 갯수를 맞추려면 끝에 -1을 또 찾아야 합니다. 예를 들어, i=7일때 누적합[i]가 -1이고 이는 곧 bell[3:7]의 구간의 초록/빨간 갯수가 같음을 알 수 있습니다.
5. 앞서 말했듯이 색이 같은 경우는 숫자가 같은 양끝이 두 색이 같은 경우입니다. 따라서 누적합에 있는 각 값에 대하 가장 멀리 떨어져있는 길이를 구하면 됩니다. 예를 들어 0이 경우,  
   [**0**, -1, **0**, -1, -2, -3, -2, -1, -2] 인 경우이고, i=0, 2 이므로 구간의 길이는 2-0 입니다.  
   -1의 경우,  
   [0, **-1**, 0, -1, -2, -3, -2, **-1**, -2] 인 경우이고, i=1, 7 이므로 구간의 길이는 7-1 입니다. 다른 값도 마찬자기로 계산하면(-2은 최대 4 떨어져있음...), i=1, 7 일때 값이 최대가 됨을 알 수 있습니다.

### (2)문제 풀이(정답 코드)

from itertools import accumulate

def solution(bell):

coors\_start = {}

coors\_end = {}

for i, x in enumerate(accumulate([0] + [-1 if b == 1 else 1 for b in bell])):

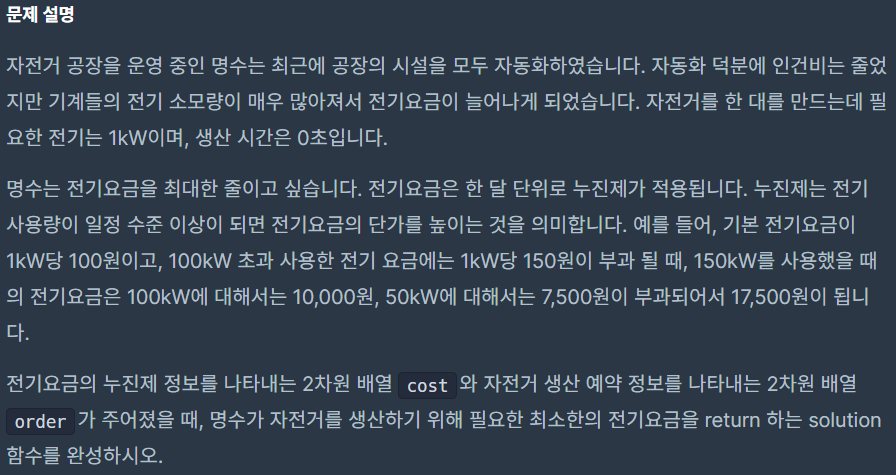
if x not in coors\_start:

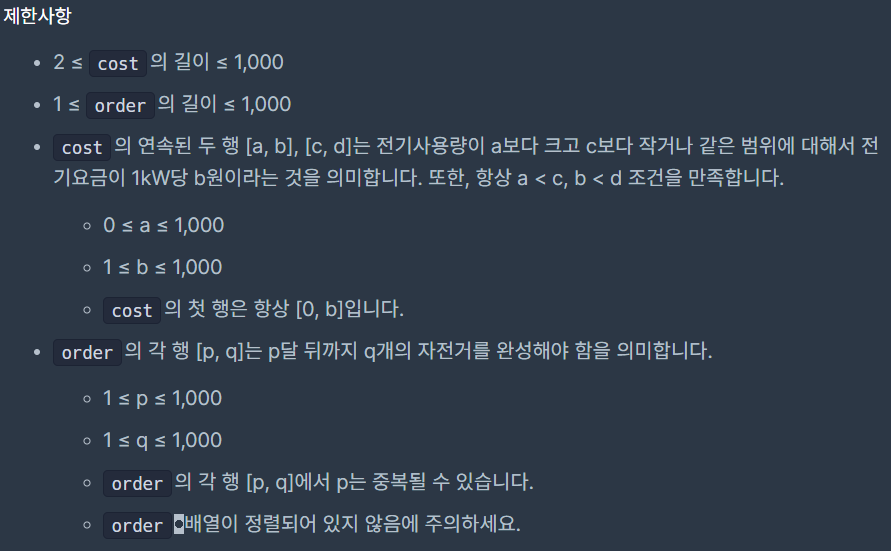
coors\_start[x] = i

coors\_end[x] = i

return max(coors\_end[x] - coors\_start[x] for x in coors\_end)

[2] 자전거 공장







### (1)지문 이해 및 풀이 계획

1. 가장 효율적인 방법은 모든 생산을 달에 골고루 배치하는 것입니다. 하지만 주문이 앞쪽에 몰리면 골고루 할 수가 없습니다. 이때는 몰린 주문량을 우선 앞쪽에 골고루 배치하고, 나머지 수량을 뒤에 골고루 배치하면 됩니다.
2. Part 1은 order의 month를 절대값에서 구간값으로 변경하는 과정입니다.  
   예를 들어 order = [[2, 300], [5, 10], [6, 100]] 일때 2개월(0~2)동안 300개를 만들고, 다음 3개월(2~5)동안 10개를 만들고 다음 1개월(5~6)동안 100개를 만들면 됩니다. 즉 다음과 같이 만들어야 합니다 => \_order = [[2, 300], [3, 10], [1, 100]]
3. Part2에서는 이제 이를 순차적으로 보면서 뒷쪽의 주문량이 더 많은 경우 앞쪽이랑 합쳐주는 과정입니다. 예를 들어 [2, 300] 요청이 있고 다음 [3, 10] 있는데, [2, 300] 은 월별 300 / 2 = 150개의 주문을 소화해야하고 [3, 10] 은 월별 10 / 3 = 3.33개의 주문을 소화해야합니다. 주문이 앞쪽에 몰린 경우이므로 이는 합칠 수 없습니다. (300 / 2 > 10 / 3 : 풀이에서는 월에 0이 나올수 있으므로 역수로 비교함). 자 그다음 [3, 10] 이후 [1, 100] 를 마찬가지로 비교하면 10 / 3 < 100 / 1 이므로 뒷쪽에 주문이 몰린 경우므로 앞쪽으로 좀 주문량을 넘겨줄 수 있습니다. 즉 합칠 수 있습니다. [3, 10] + [1, 100] => [3+1, 10+110] => [3, 120], 새로 합쳐진 주문은 전과 비슷하게 다시 [2, 300] 비교하여 합쳐주거나 멈추면 됩니다. 최종적으로는 stack = [[2, 300], [4, 110]] 이 됩니다.
4. Part3에서는 이제 구간별 가격을 구해주면 되는데, 문제에서 제공되는 갯수를 넘길때마다 그 넘긴 갯수만큼 차액을 계속 더해주면 되는 비교적 간단한 과정입니다.

### (2)문제 풀이(정답 코드)

def solution(cost, order):

# part 1

order.sort()

\_order = [order[0]]

for i, (m, n) in enumerate(sorted(order)[1:]):

\_order.append([m - order[i][0], n])

# part 2

stack = []

for m, n in \_order:

while stack:

\_m, \_n = stack[-1]

if \_m / \_n < m / n:

break

stack.pop()

m, n = m + \_m, n + \_n

stack.append([m, n])

# part 3

answer = 0

for m, n in stack:

p\_prev = 0

for t, p in cost:

if m \* t >= n:

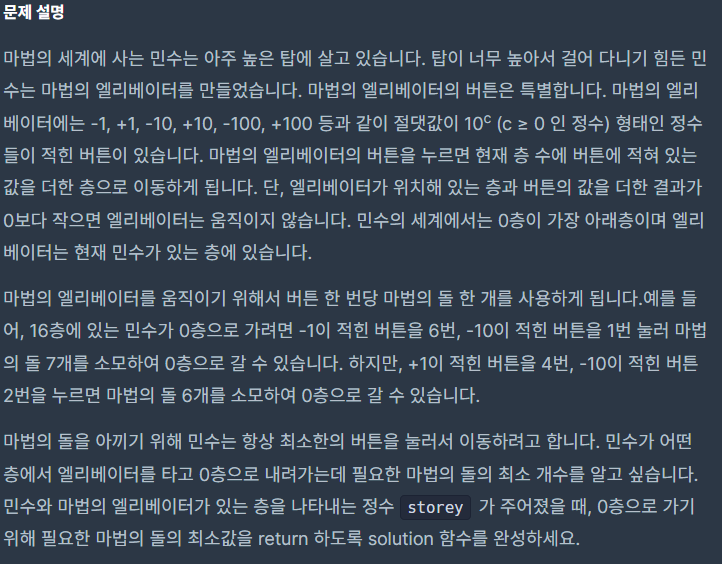
break

answer += (n - m \* t) \* (p - p\_prev) # 넘긴 갯수 \* 차액

p\_prev = p

return answer

[3] 마법의 엘리베이터





### (1)지문 이해 및 풀이 계획

뒷자리가 0인 수는 0을 빼도 누르는 버튼수가 같습니다. (뒤가 0이면 +-1 버튼을 누를 일이 없으므로 /10버튼을 누르면 결국 같음.) 따라서 자리수 단위로 감소하는 재귀로 접근하면 쉽게 해결할 수 있습니다. 예를 들어 solution(2554)는 solution(255) + 4 와 solution(256) + 6 둘 중 더 적은쪽일것입니다. solution(255) + 4은 -1을 4번 하고 2550을 만든 경우고, solution(256) + 6은 +1을 6번하고 2560을 만든 경우입니다.

### (2)문제 풀이(정답 코드)

def solution(storey):

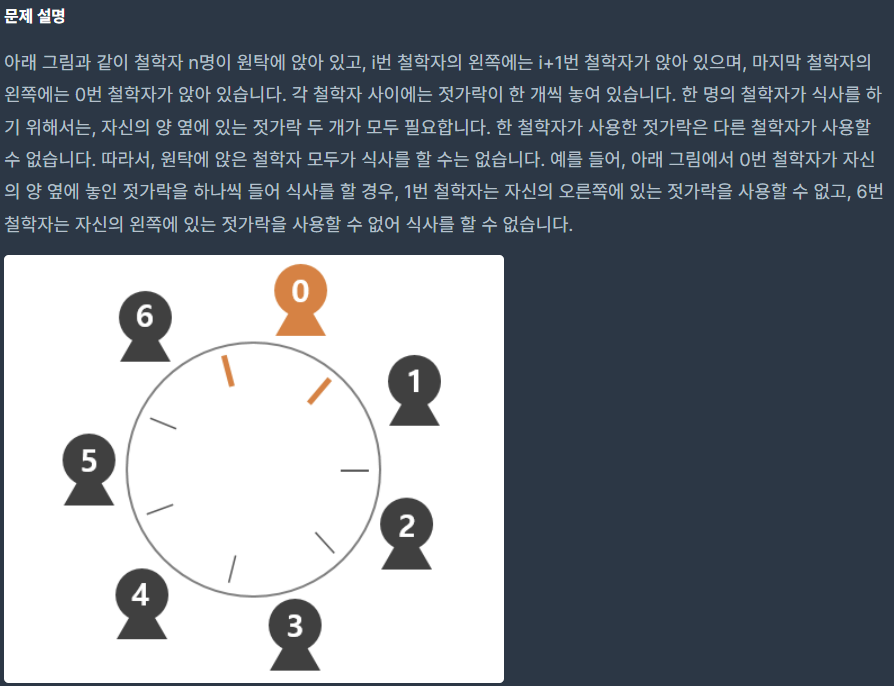
if storey <= 1:

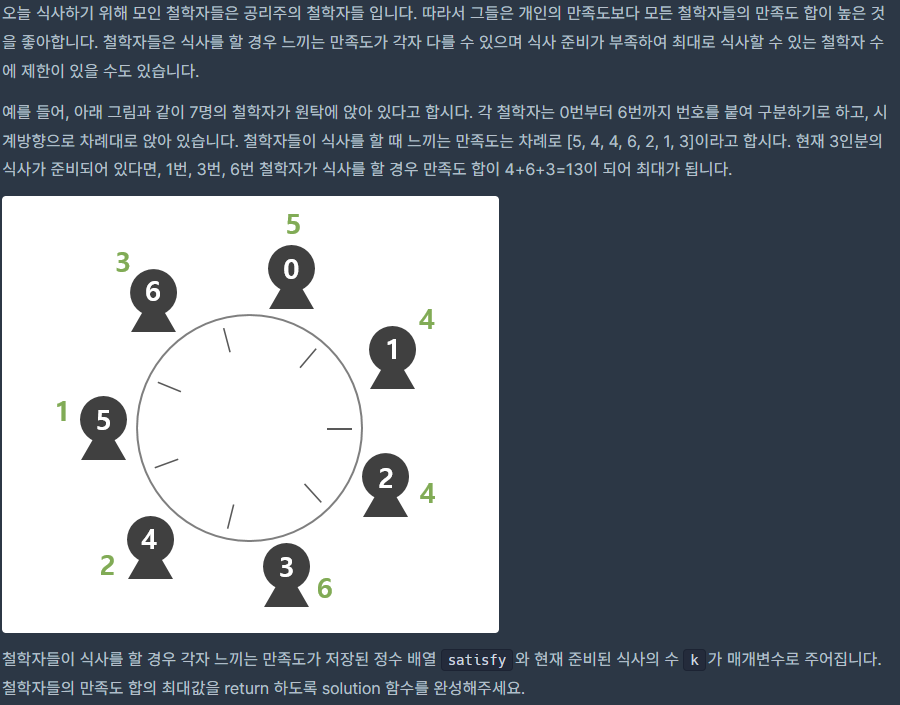
return storey

q, r = divmod(storey, 10)

return min(solution(q) + r, solution(q+1) + (10-r) )

[4] 식사하는 철학자들







### (1)지문 이해 및 풀이 계획

* 예시: satisfy: [6, 1, 9, 10, 8], k = 2

1. 만족도가 가장 높은 사람이 식사를 하는 경우, 그렇지 않은 경우 두가지 경우로 나누어 생각해보자.
   1. 만족도가 가장 높은 사람이 식사를 하는 경우: 양 옆 사람이 못먹는다 정도만 반영하면 됨.
   2. 만족도가 가장 높은 사람이 식사를 하지 않는 경우, 이 경우 항상 양 옆사람이 식사를 해야만 한다. 그렇지 않으면 만족도가 가장 높은 사람이 식사를 최대값을 위해 포기할 이유가 없다. 예를 들어 위의 경우 만족도가 가장 높은 0번째 철학자는 만족도가 10으로 최대이다. 10 옆 8, 9 가 식사를 안하는 경우, 최대값을 위해서 10이 식사를 안할 이유가 없다. 8은 식사를 안하고 9만 식사를 하는 경우, 9대신 10이 먹는 경우가 +1 더 이득이다. 9는 식사를 안하고 8만 식사를 하는 경우도 마찬가지. 따라서 10이 식사를 안할만한 경우의 수는 9, 8 이 모두 식사를 하는 경우이다.
2. [6, 1, 9, 10, 8], k = 2 의 문제를 => 10 + [6, 1, (9 + 8 - 10)], k = 1 의 문제로 바꿔 생각할 수 있다. 여기서 6, 1 을 선택하는 것은 그 전 문제에서의 6, 1 을 선택하는 것과 같고, (9 + 8 - 10) 을 선택하는 것은 10대신 그 양옆을 선택하는 것과 동치이다. 이런식으로 문제를 조금씩 줄여나가면서
   1. [6, 1, 9, 10, 8], k = 2
   2. 10 + [6, 1, (9 + 8 - 10) = 7], k = 1
   3. 10 + 7, k= 0
   4. 17을 정답으로 구할 수 있다.
3. 테스트 케이스 [5, 4, 4, 6, 2, 1, 3], k = 3 의 경우 다음과 같다.
   1. [5, 4, 4, 6, 2, 1, 3], k = 3
   2. 6 + [5, 4, (4+2-6)=0, 1, 3], k = 2
   3. 6 + 5 + [(3+4-5)=2, 0, 1], k = 1
   4. 6 + 5 + 2, k = 0
   5. 정답은 13
4. 테스트 케이스 [10, 1, 1, 10, 1, 1, 10, 1], k=4 의 경우 다음과 같다.
   1. [10, 1, 1, 10, 1, 1, 10, 1], k=4
   2. 10 + [(1+1-10)=-8, 1, 10, 1, 1, 10], k=3
   3. 10 + 10 + [-8, (1+1-10)=-8, 1, 10], k=2
   4. 10 + 10 + 10 + [-8, -8, ((1-8-10)=-17], k=1
   5. 음수만 남았으므로 더 이상 선택하지 않음: 정답 30

### (2)문제 풀이(정답 코드)

import heapq

def solution(satisfy, k):

n = len(satisfy)

l = list(range(-1, n-1)) # 왼쪽 index

r = list(range(1, n+1)) # 오른쪽 index

l[0] = n - 1

r[n - 1] = 0

visit = [0 for \_ in range(n)] # 식사를 할 수 있는지

hq = [(-s, i) for i, s in enumerate(satisfy)]

heapq.heapify(hq)

ans = 0

while hq and k:

s, idx = heapq.heappop(hq)

if visit[idx]:

continue

ans += -s

satisfy[idx] = satisfy[l[idx]] + satisfy[r[idx]] - satisfy[idx]

if satisfy[idx] > 0:

heapq.heappush(hq, (-satisfy[idx], idx))

visit[l[idx]] = visit[r[idx]] = True

l[idx] = l[l[idx]]

r[idx] = r[r[idx]]

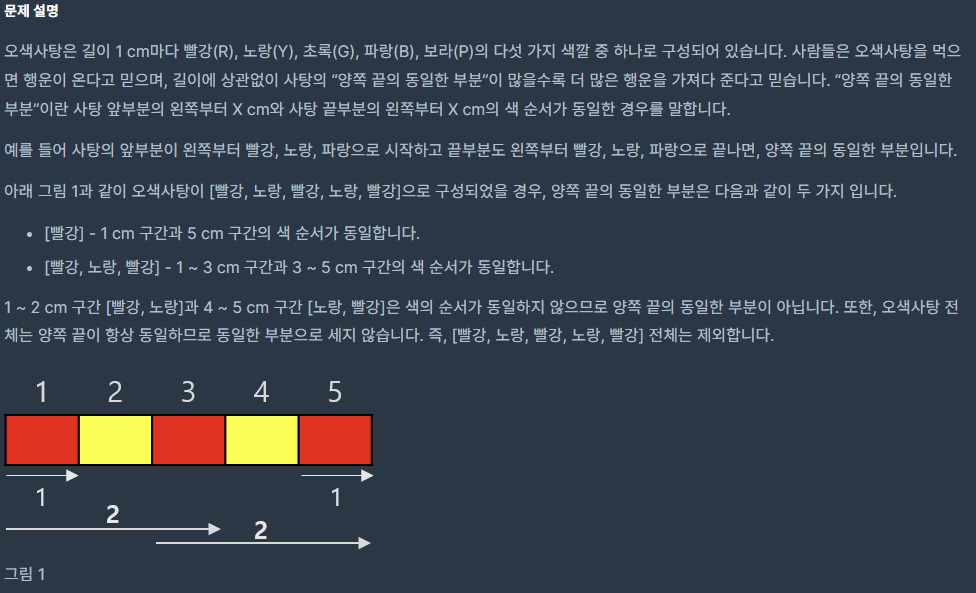
l[r[idx]] = idx

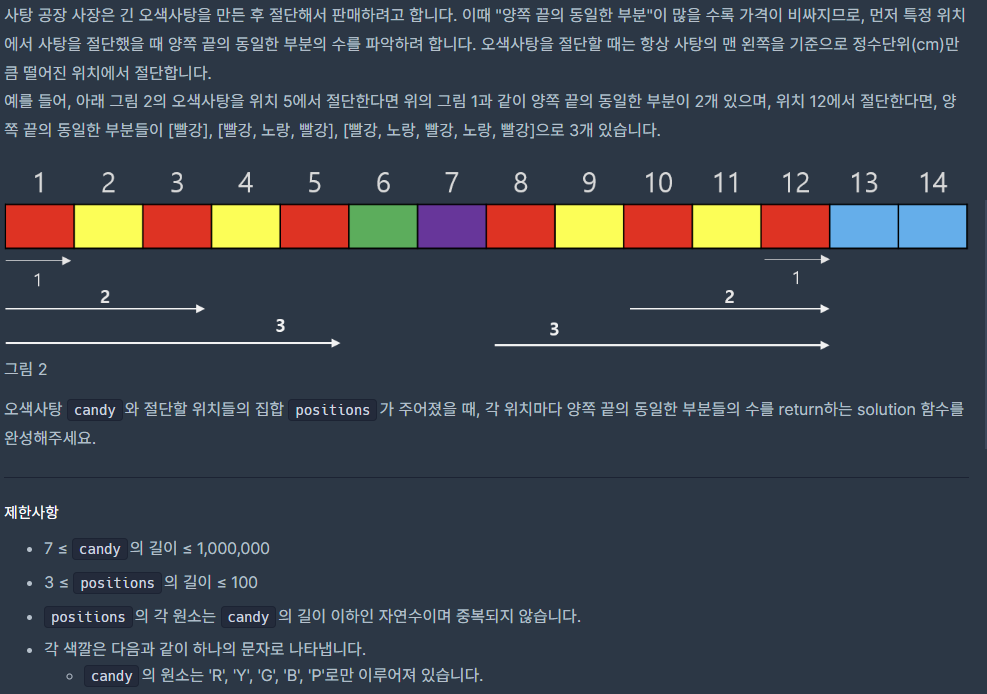
r[l[idx]] = idx

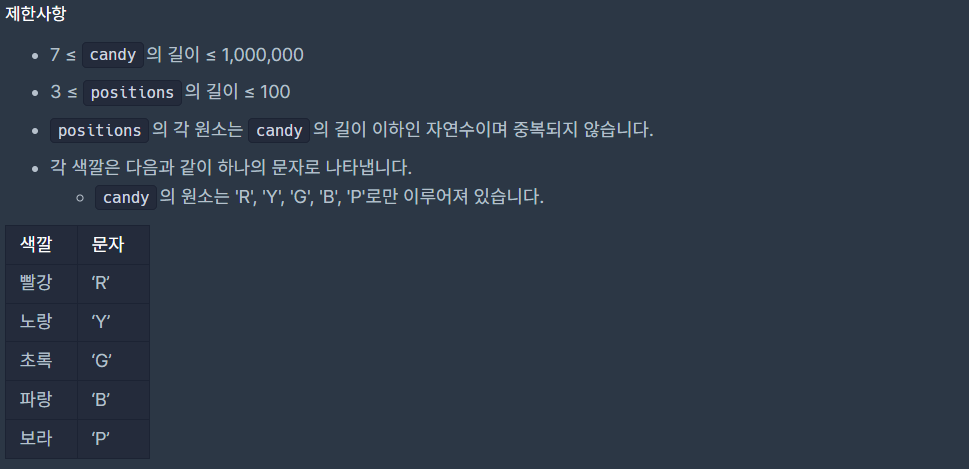
k -= 1

return ans

[5] 행운의 오색사탕









### (1)지문 이해 및 풀이 계획

cache에 얼마나 첫번째부터 일치하는지에 대한 정보를 저장하는 것을 목표로 합니다. 계속 일치하면 쭉 이어나가고 아닌경우, 일치하는 구간이 나올때까지 cache를 타고 계속 뒤로 갑니다. cache가 완성되면 각 position에 대해 cache를 얼마만큼 탈 수 있느냐가 곧 정답이 됩니다.

### (2)문제 풀이(정답 코드)

def solution(candy, positions):

# part1

n\_match, cache = 0, [0]\*len(candy)

for i in range(1, len(candy)):

while n\_match and candy[n\_match] != candy[i]:

n\_match = cache[n\_match]

if candy[n\_match] == candy[i]:

n\_match += 1

cache[i] = n\_match

# part2

answer = []

for pos in positions:

ans = 0

while cache[pos-1]:

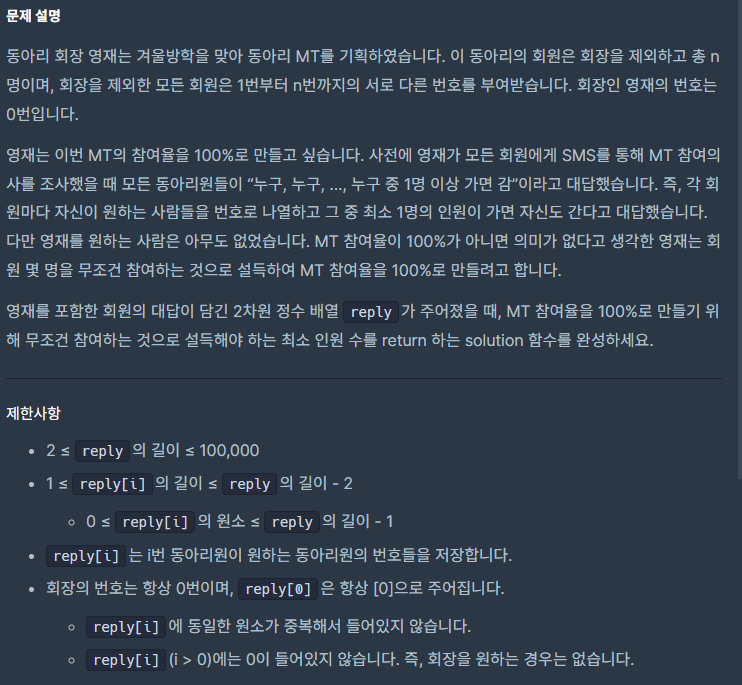
pos = cache[pos-1]

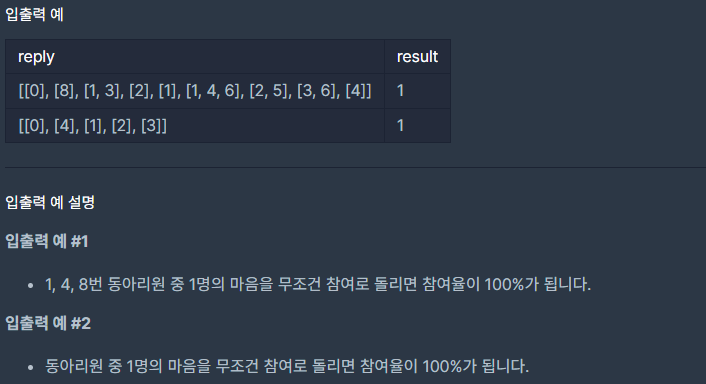
ans += 1

answer.append(ans)

return answer

[6] 가면 감





### (1)지문 이해 및 풀이 계획

1. 가면가는 사람이 없는 사람 (빈 리스트)인 사람이 있다면 그 사람은 무조건 참가해야합니다. 그 사람이 가면 따라가는 사람들을 잘 전파합니다.
2. (빈 리스트)인 사람이 없다면 항상 cycle인 그래프가 있습니다. 이를 찾아서 합쳐줍니다.
3. 1-2를 모든 사람이 갈때까지 반복합니다.

### (2)문제 풀이(정답 코드)

def solution(reply):

# 그래프 생성

remaining = set(range(1, len(reply)))

forward = [set(l) for l in reply]

reverse = [set() for \_ in range(len(reply))]

for i, l in enumerate(reply):

for j in l:

reverse[j].add(i)

leafs = [i for i, l in enumerate(reply) if not l]

def remove\_cycle():

nonlocal remaining

n = next(iter(remaining))

nexts = {}

while n not in nexts:

nexts[n] = next(iter(forward[n]))

n = nexts[n]

cycles = [n]

s\_f, s\_r = forward[n], reverse[n]

while 1:

\_n = nexts[cycles[-1]]

if \_n == n:

break

s\_f.update(forward[\_n])

s\_r.update(reverse[\_n])

cycles.append(\_n)

cycles = set(cycles)

for i in s\_r:

forward[i] -= cycles

forward[i].add(n)

for i in s\_f:

reverse[i] -= cycles

reverse[i].add(n)

remaining -= set(cycles)

remaining.add(n)

s\_f -= set(cycles)

s\_r -= set(cycles)

if s\_f:

return []

else:

return [n]

def remove\_leafs(leafs):

nonlocal remaining

to\_remove = set(leafs)

while leafs:

leaf = leafs.pop()

for p in reverse[leaf]:

if p not in to\_remove:

leafs.append(p)

to\_remove.add(p)

remaining -= to\_remove

return leafs

ans = len(leafs)

length = len(remaining)

while remaining:

if leafs:

leafs = remove\_leafs(leafs)

else:

leafs = remove\_cycle()

ans += len(leafs)

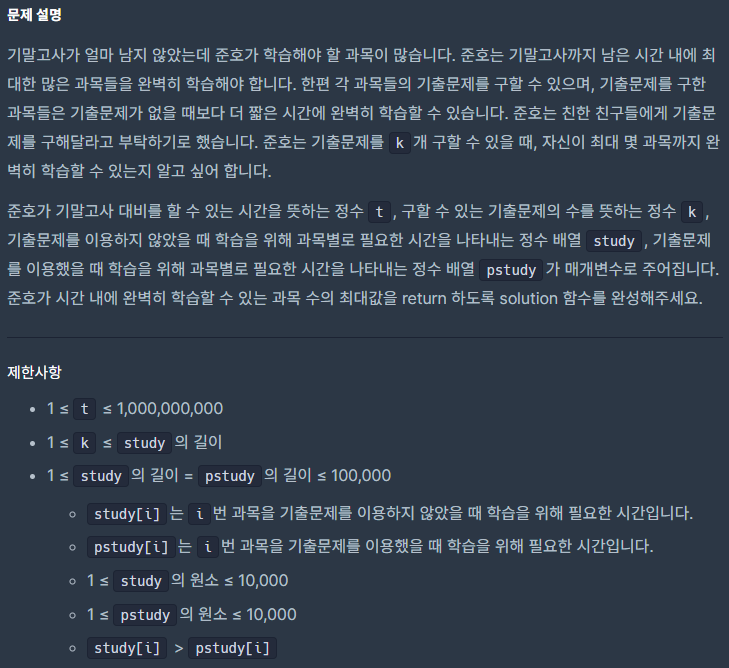
if len(remaining) >= length:

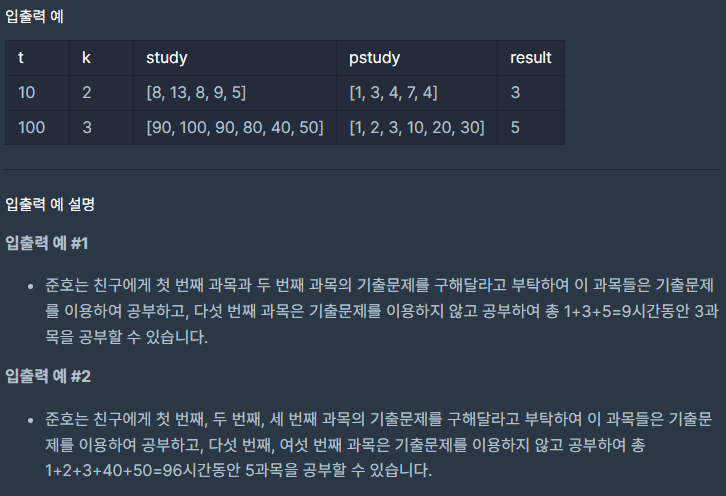
raise ValueError

length = len(remaining)

return ans

[7] 기말고사





### (1)지문 이해 및 풀이 계획

1. 일단은 study, pstudy에서 가장 적은 cost인것을 t가 넘어갈 때까지 계속 더해주면 된다. 이것이 문제가 되는 경우는 두가지인데  
   a. 하나는 pstudy에서 뽑은것이 k개가 넘어갈때이고  
   b. 두번째는 study에서 뽑은 것과 pstudy에서 뽑은것이 겹칠때이다.
2. a의 경우 diff라는 heapq를 사용하여 해결할 수 있다. diff에는 k개의 0으로 초기화되어있으며 (처음 k개는 아무 페널티없이 pstudy에서 뽑을 수 있으므로), 만약에 pstudy에서 어떠한 과목을 뽑았다면 diff에 study와의 차이값을 넣어준다. 이렇게 pstudy에서 뽑은 과목이 k개가 된 경우, 그 다음에 또 pstudy에서 어떤 과목을 뽑게 된다면 diff가 가장 작은 과목을 pstudy -> study로 바꾸면서까지 뽑을 가치가 있는 것인지 판단을 하게 된다. (diff[0] + paper\_hq[pidx][0] < study\_hq[sidx][0]) 그러할 가치가 없는 경우 pstudy 대신 study에서 뽑으면 된다.
3. b의 경우 visited를 두어 겹치지 않도록 체크를 할 수 있다.

### (2)문제 풀이(정답 코드)

from heapq import heappush, heappop, heapify

def solution(t, k, study, pstudy):

visited = [0]\*len(study)

diff = [0]\*k

study\_hq = [(s, i) for i, s in enumerate(study)]

paper\_hq = [(p, i) for i, p in enumerate(pstudy)]

study\_hq.sort()

paper\_hq.sort()

sidx, pidx = 0, 0

while sidx < len(study):

if diff[0] + paper\_hq[pidx][0] < study\_hq[sidx][0]:

c, i = paper\_hq[pidx]

c += diff[0]

heappop(diff)

heappush(diff, study[i] - pstudy[i])

else:

c, i = study\_hq[sidx]

if c > t:

break

t -= c

visited[i] = 1

while sidx < len(study) and visited[study\_hq[sidx][1]]:

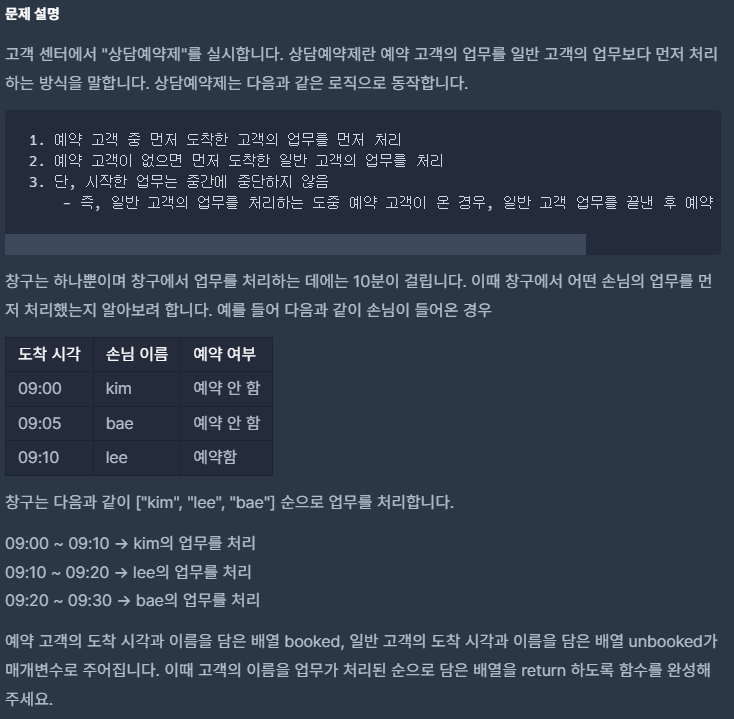
sidx += 1

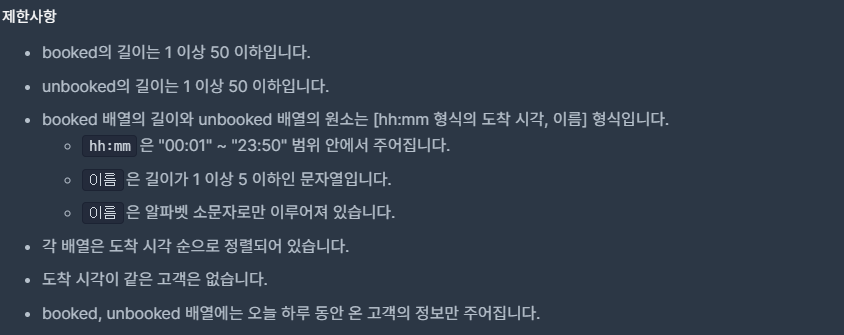
while pidx < len(study) and visited[paper\_hq[pidx][1]]:

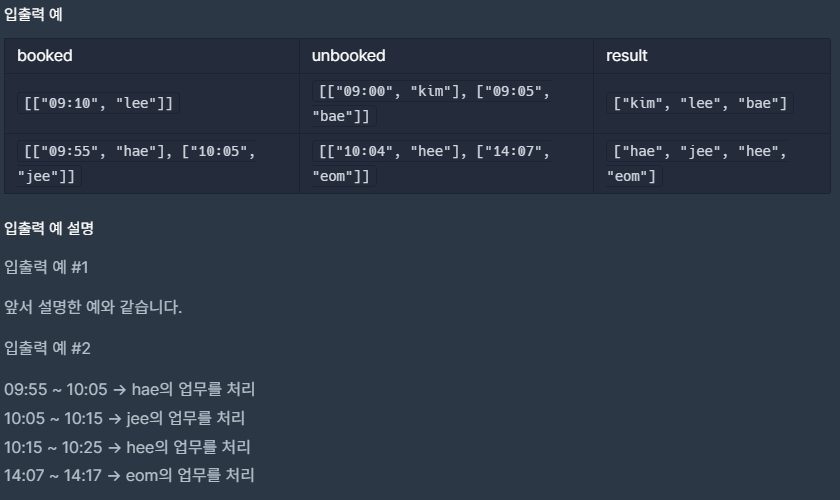
pidx += 1

return sum(visited)

[8] 중간고사 – 상담예약제







### (1)지문 이해 및 풀이 계획

Merge Sort와 비슷한 느낌으로 구현해주시면 됩니다. string의 시간을 적절히 숫자로 바꿔주는 작업이 필요합니다. 필수로 필요하진 않지만 예외처리를 줄이기 위해 booked/unbooked 끝에 dummy data를 추가하였습니다.

### (2)문제 풀이(정답 코드)

def parse\_time(t):

h, m = map(int, t.split(':'))

return 60\*h + m

def solution(booked, unbooked):

booked = [(parse\_time(t), name) for t, name in booked] + [(1000000, None)]

unbooked = [(parse\_time(t), name) for t, name in unbooked] + [(1000000, None)]

booked.sort()

unbooked.sort()

b, u, t, answer = 0, 0, 0, []

while b < len(booked) and u < len(unbooked):

t1, t2 = booked[b][0], unbooked[u][0]

if t1 <= t:

answer.append(booked[b][1])

b += 1

t += 10

elif t2 <= t:

answer.append(unbooked[u][1])

u += 1

t += 10

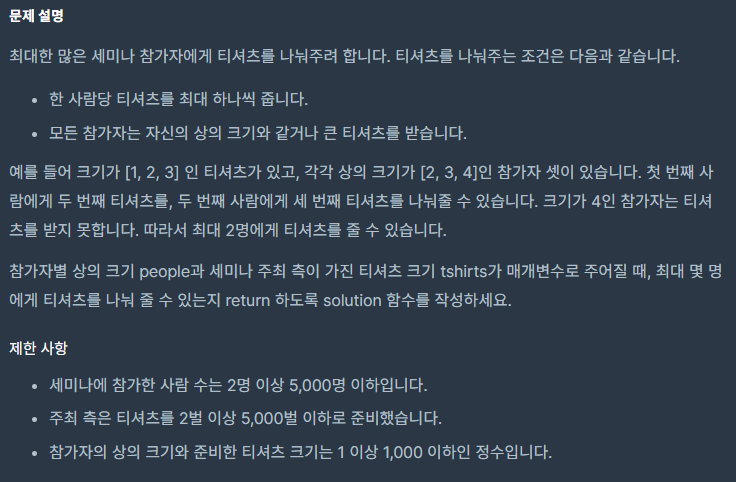
else:

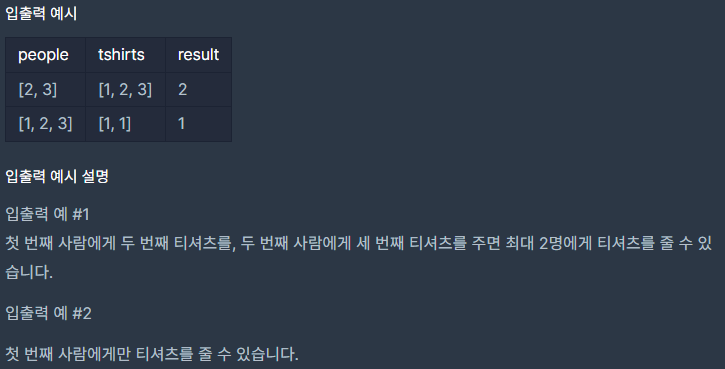
t = min(t1, t2)

answer.pop()

return answer

[9] 중간고사 – 티셔츠





### (1)지문 이해 및 풀이 계획

작은 사람부터 작은 옷부터 입히면 가장 효율적입니다. 즉, 정렬 후 tshirts을 하나씩 살펴보면서 people보다 크면 입히고, 아니면 넘기면 됩니다. 구현은 이전 문제와 같이 투포인터를 활용하면 됩니다.

### (2)문제 풀이(정답 코드)

def solution(people, tshirts):

people.sort()

tshirts.sort()

p, t, ans = 0, 0, 0

while p < len(people) and t < len(tshirts):

if tshirts[t] >= people[p]:

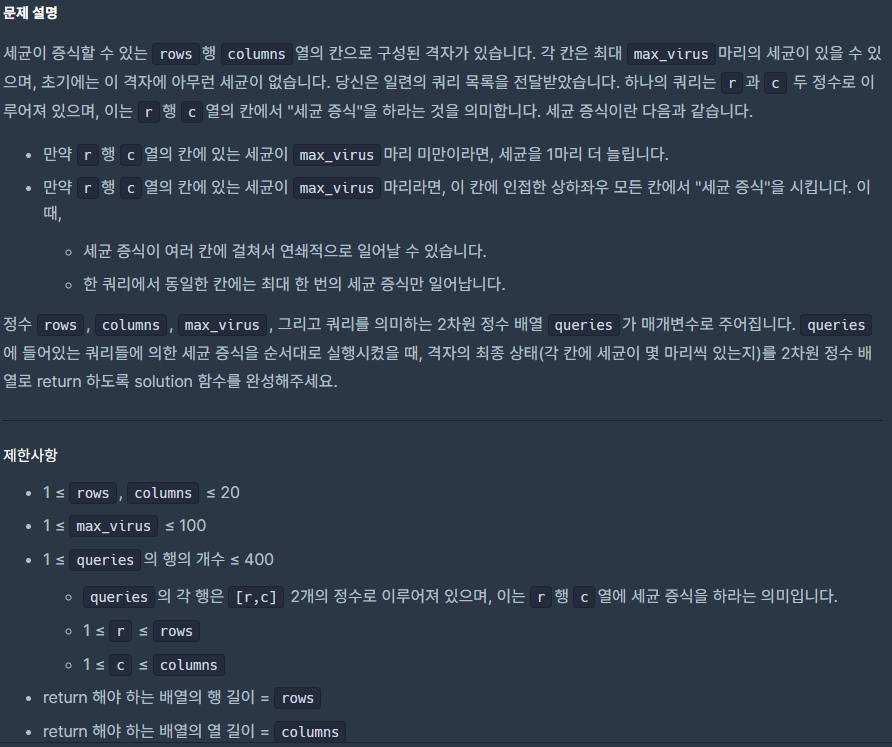
ans += 1

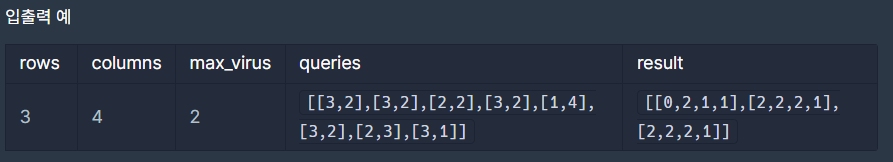
p += 1

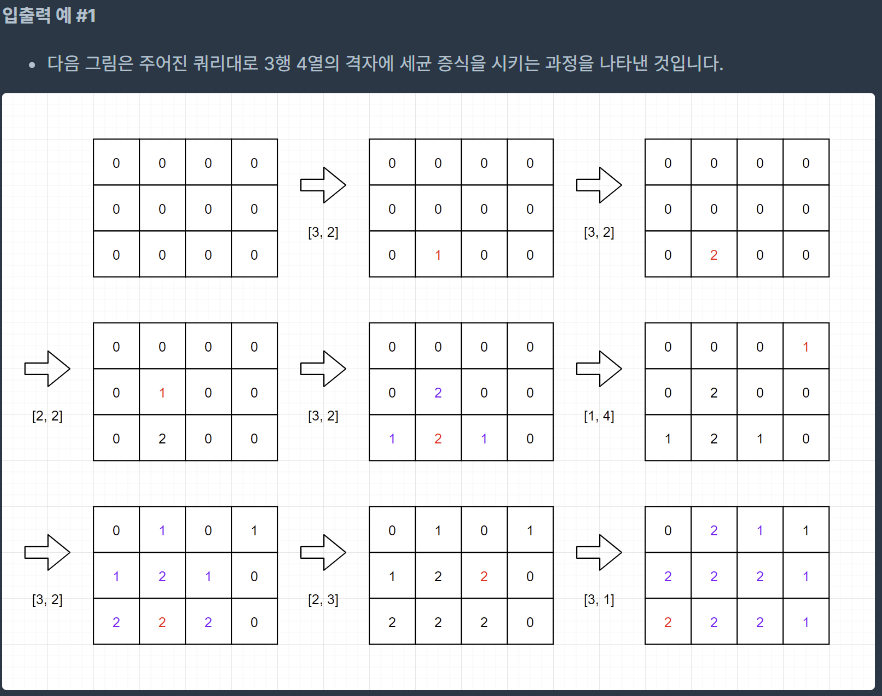
t += 1

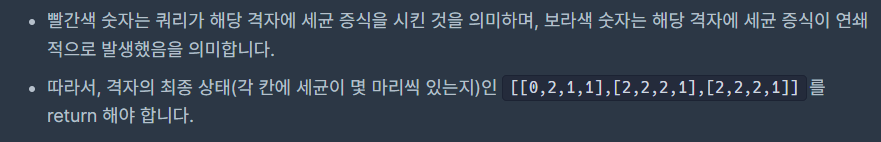
return ans

[10] 기말고사 – 판데믹









### (1)지문 이해 및 풀이 계획

DFS 기본 문제입니다. 문제에서 요구한 바와 같이 max\_virus이하인 경우는 숫자를 1 증가시키고 max\_virus인 경우는 일반 DFS와 같이 neighbor를 stack에 추가하면 됩니다.

### (2)문제 풀이(정답 코드)

def solution(rows, columns, max\_virus, queries):

grid = [[0]\*columns for \_ in range(rows)]

def action(i, j):

stack = [(i, j)]

visited = set([(i, j)])

while stack:

i, j = stack.pop()

if grid[i][j] < max\_virus:

grid[i][j] += 1

else:

for \_i, \_j in [(i, j+1), (i, j-1), (i+1, j), (i-1, j)]:

if 0 <= \_i < rows and 0 <= \_j < columns and (\_i, \_j) not in visited:

visited.add((\_i, \_j))

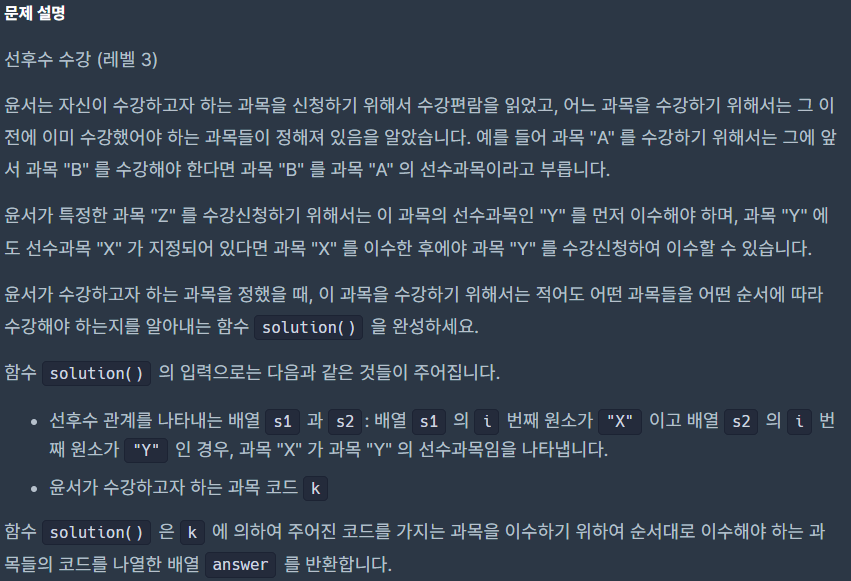
stack.append((\_i, \_j))

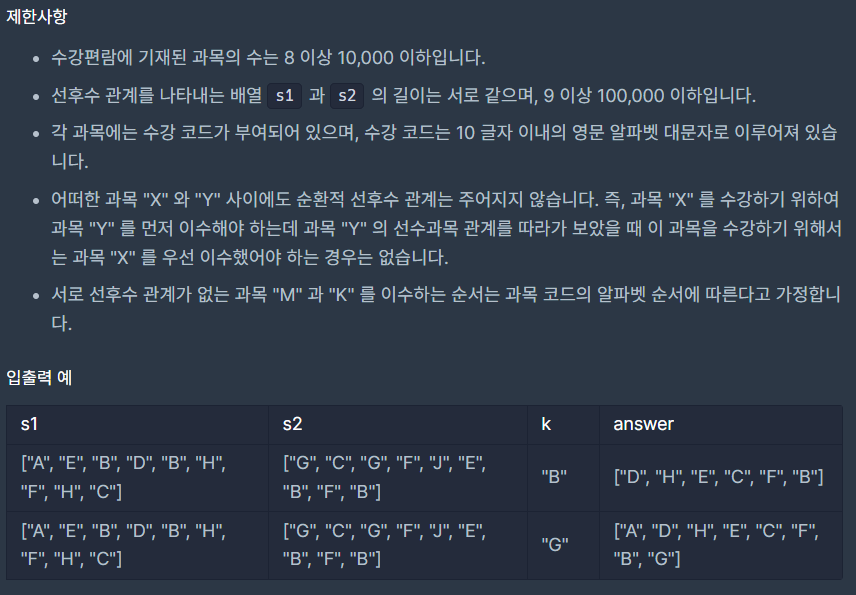
for i, j in queries:

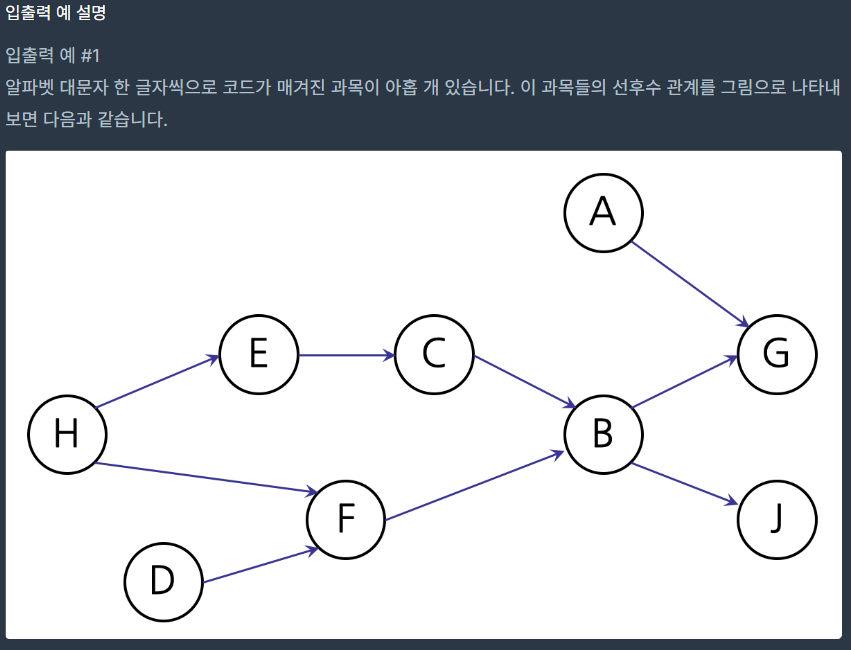
action(i-1, j-1)

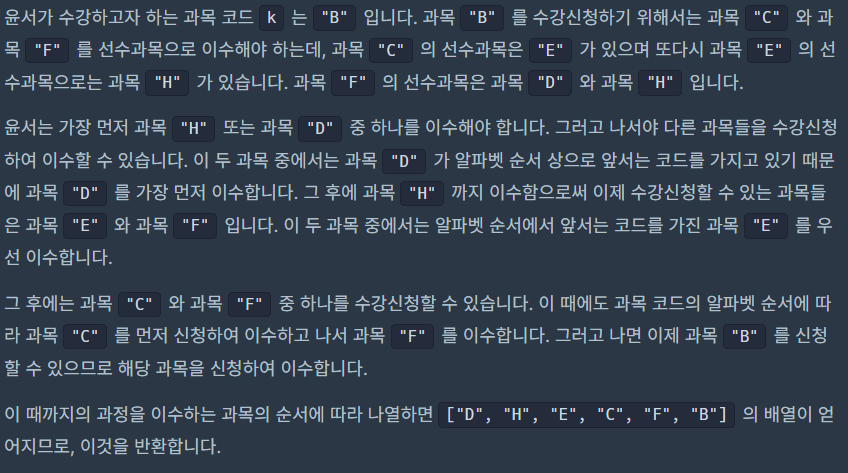
return grid

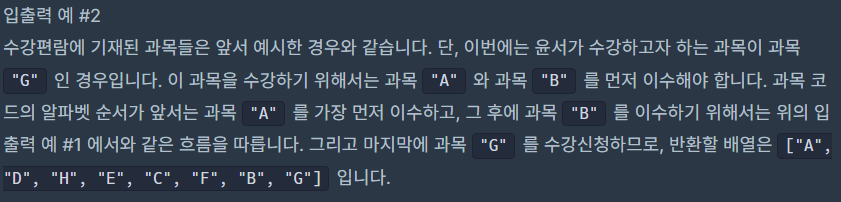
[11] 기말고사 – 선후수 수강











### (1)지문 이해 및 풀이 계획

기본적인 위상정렬 문제입니다. 위상정렬이 기본적인 알고리즘은 아니지만 말이죠. 기본적인 흐름은 다음과 같습니다.

1. 선수과목이 없는/모두 수강된 과목(leaf)을 모두 구한다.
2. 가장 작은 Leaf를 수강한다. 새로 모두 수강된 과목이 있으면 leaf에 추가한다.
3. 1-2를 계속 반복하면 된다.

위상정렬에 대한 자세한 내용은 다음을 참조하시면 됩니다.  
[https://velog.io/@kimdukbae/위상-정렬-Topological-Sorting](https://velog.io/@kimdukbae/%EC%9C%84%EC%83%81-%EC%A0%95%EB%A0%AC-Topological-Sorting)

### (2)문제 풀이(정답 코드)

from heapq import heappush, heappop

from collections import defaultdict

def solution(s1, s2, k):

# 전체 그래프 생성

graph = defaultdict(list)

for X, Y in zip(s1, s2):

graph[Y].append(X)

answer, leafs = [], []

# K에 관련된 그래프 생성

stack = [k]

visited = set([k])

graph\_k = defaultdict(list)

indegrees = defaultdict(int)

while stack:

node = stack.pop()

if graph[node]:

for prev in graph[node]:

indegrees[node] += 1

graph\_k[prev].append(node)

if prev not in visited:

stack.append(prev)

visited.add(prev)

else:

heappush(leafs, node)

# 위상정렬

while leafs:

node = heappop(leafs)

answer.append(node)

for next\_node in graph\_k[node]:

indegrees[next\_node] -= 1

if not indegrees[next\_node]:

heappush(leafs, next\_node)

return answer