완전 탐색

최소 직사각형

- 모든 명함을 담을 수 있는 지갑의 최소크기 반환 (명함은 가로 또는 세로로 돌려서 넣을 수도 있음)
- sizes : [명함가로길이, 명함세로길이] 를 각 원소로 갖는 배열

```
sizes : [[60, 50], [30, 70], [60, 30], [80, 40]] // 총 4개의 명함
result : 4000  // 50 * 80 크기의 지갑에 모두 담을 수 있음
```

- 각 원소 [width, height] 에서 width와 height 중 **더 작은 값을 뽑아 비교해서 가장 큰 값**과 **큰 값을 뽑아 비교해서 가장 큰 값** 을 곱하면 됨

```
def solution(sizes):
    answer = 0
    width = []
    height = []
    for size in sizes:
        if size[1] >= size[0]:
            width.append(size[0])
            height.append(size[1])
        else:
            width.append(size[1])
            height.append(size[0])

    maxWidth = max(width)
    maxHeight = max(height)
    answer = maxWidth * maxHeight
    return answer
```

▼ 다른 풀이

```
def solution(sizes):
    return max(max(x) for x in sizes) * max(min(x) for x in sizes)
```

모의고사

```
answers : 문제의 정답이 담긴 배열
1번 수포자 : [1,2,3,4,5...]
2번 수포자 : [2,1,2,3,2,4,2,5...]
3번 수포자 : [3,3,1,1,2,2,4,4,5,5...]
의 규칙을 반복하며 답을 찍는다

answer : 가장 많은 문제를 맞힌 사람을 담은 배열

# 예
answers = [1,2,3,4,5]
answer = [1] # 1번 수포자가 5개 모두 맞혔으므로 제일 많은 문제를 맞힘

answers = [1,3,2,4,2]
answer =[1,2,3] # 1,2,3번 수포자가 각각 2개씩 맞힘
```

- 내 코드

```
def solution(answers):
    answer = []
    # 수포자 1의 패턴
    pattern1 = [1,2,3,4,5] # len = 5
    # 수포자 2의 패턴
    pattern2 = [2,1,2,3,2,4,2,5] # len = 8
    # 수포자 3의 패턴
```

```
    pattern3 = [3,3,1,1,2,2,4,4,5,5] # len = 10
    score=[0,0,0] # 1,2,3번 수포자가 맞힌 문제의 개수를 저장

    for i in range(0, len(answers)):
        if pattern1[i%5] == answers[i]:
            score[0] += 1
        if pattern2[i%8] == answers[i]:
            score[1] += 1
        if pattern3[i%10] == answers[i]:
            score[2] += 1

    for i in range(0,3):
        if score[i] == max(score):
            answer.append(i+1)

    return answer
```

▼ enumerate() 함수

- “파이썬 답게” 인덱스와 원소를 동시에 접근하며 for문을 돌린다
- enumerate()는 **인덱스 번호**와 **원소**를 tuple 형태로 반환

```
def solution(answers):
    pattern1 = [1,2,3,4,5]
    pattern2 = [2,1,2,3,2,4,2,5]
    pattern3 = [3,3,1,1,2,2,4,4,5,5]
    score = [0, 0, 0]
    result = []

    for idx, answer in enumerate(answers): # enumerate() 사용
        if answer == pattern1[idx%len(pattern1)]:
            score[0] += 1
        if answer == pattern2[idx%len(pattern2)]:
            score[1] += 1
        if answer == pattern3[idx%len(pattern3)]:
            score[2] += 1

    for idx, s in enumerate(score): # enumerate() 사용
        if s == max(score):
            result.append(idx+1)

    return result
```

⭐ 소수 찾기 ⭐

- 주어진 string(numbers)에 포함된 숫자로 만들 수 있는 소수가 몇개인지 구하시오

```
# example 1
numbers : "17"
return : 3 # "17"로 만들 수 있는 소수는 [7,17,71]

# example 2
numbers : "011"
return : 2 # [11,101] - 11과 011은 같은 숫자로 취급
```

▼ 무지성 3중 for문으로 풀긴 풀었습니다만… (내 풀이)

```
import itertools

def solution(numbers):
    answer = 0
    items = []
    results =[]

    # numbers의 숫자들을 하나씩 분해해서 items에 집어넣기
    # items = list(numbers) 로 줄일 수 있음
    for num in numbers:
        items.append(num)

    # 1,2...len(numbers)개를 뽑아 순열permutation 생성 (배치 순서가 고려됨)
    # "17" -> [('1'),('7'),('1','7'),('7','1')]
    for i in range(1,len(numbers)+1):
        for elem in set(itertools.permutations(items,i)):
```

```
            string=''
            # 각 순열의 원소를 모두 이어붙인 후 integer로 변환해 results에 담음
            # 이 과정에서 '011' 은 11이 됨
            for j in range(0,i):
                string += elem[j]
            results.append(int(string))

    # set(results)로 중복된 값 제거 후 소수 찾기
    # ex) [11,11] -> (11)
    for num in set(results):
        if num == 2:
            answer += 1
        if num > 2:
            for i in range(2,num):
                if num%i == 0:
                    break
                if i == num-1:
                    answer +=1

    return answer
```

- 다른 사람의 풀이 (내 풀이 짧게 쓰는법)

```
from itertools import permutations
def solution(n):
    a = set()
    for i in range(len(n)):
        a |= set(map(int, map("".join, permutations(list(n), i + 1))))
    a -= set(range(0, 2)) # 0이랑 1은 소수가 아니므로 제거
    for i in range(2, int(max(a) ** 0.5) + 1): # n의 제곱근까지만 확인
        a -= set(range(i * 2, max(a) + 1, i))
    return len(a)
```

- ...를 통해 배워갈 내용
  - **map**

```
# map
사용법 : list(map(함수,리스트))
map은 지정된 함수로 리스트의 요소를 처리해준다
ex)
>> a = [1.2, 2.5, 3.7, 4.6]
>> a = list(map(int, a))
>> a
[1,2,3,4]
```

  - **join**

```
# join
사용법 : "{구분자}".join(리스트)
join은 매개변수로 들어온 리스트에 있는 요소를 합쳐서 하나의 문자열로 바꿔준다
구분자를 사용하면 각 요소 사이에 구분자를 넣어서 합침.
ex)
"_".join(['a','b','c']) -> 'a_b_c'
```

  - **에라토스테네스의 체**

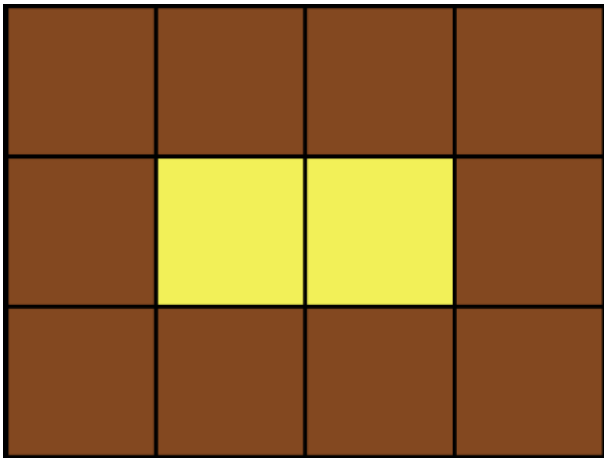
```
# 에라토스테네스의 체는 소수를 판별하는 알고리즘이다
# i = 2부터 시작해서 i의 배수에 해당하는 수를 배열에서 모두 지운다
# 2부터 시작해서 남아있는 수는 모두 소수이다

# 효율적인 방법은 n의 제곱근까지만(sqrt(n)) 확인하는 것(위의 코드도 마찬가지)
# n = a * b 이면 a,b 둘 중 하나는 무조건 sqrt(n) 이하이기 때문

### 증명)
### n = a * b이고 n = m * m (m = sqrt(n)) 인 상태에서 a와 b가 자연수이려면 다음의 세가지 경우 중 하나만 가능하다
### 1. a = m 이고 b = m
### 2. a < m 이고 b > m
### 3. a > m 이고 b < m
### 따라서  min(a, b)는 m과 같거나 m보다 작다 : min(a, b) <= m
### 즉 a와 b 둘 중 하나는 sqrt(n) 이하이다.
```

카펫

- 카펫은 다음과 같이 노란색 타일을 갈색 타일이 둘러싼 형태임



```
# 매개변수
brown : 갈색 격자의 수
yellow : 노란색 격자의 수
return : 카펫의 가로, 세로 크기가 순서대로 담긴 배열

# 노란색 격자는 1개 이상이다
# 카펫의 가로 길이 >= 카펫의 세로 길이
```

- 에라토스테네스 체에서 증명한 것(n= a*b일때 a와 b 중 하나는 반드시 sqrt(n) 이하이다)을 바탕으로 가능한 노란색 타일의 배치를 구함

```
# yellow = 24 라면 가능한 노란색 타일의 가로 * 세로 경우의 수는 :
[24, 1]
[12, 2]
[8, 3]
[6, 4]

# 각 경우에서 갈색 타일의 개수를 구해서 주어진 brown 값과 비교
```

```
def solution(brown, yellow):
    answer = []
    for i in range(1, int(yellow**0.5)+1):
        if yellow%i == 0:
            y_height = i
            y_width = yellow/i
        else:
            continue

        b_height = y_height+2
        b_width = y_width+2
        if (b_height-1)*2 + (b_width-1)*2 == brown:
            answer.extend([b_width, b_height])
    return answer
```

피로도

```
# 각 던전을 탐험하기 위해선 현재 피로도가 최소 필요 피로도보다 같거나 커야한다
# 각 던전을 지날 때 소모 피로도만큼 깎인다 (다음 던전을 탐험하려면 이전 던전에서 깎인 피로드 >= 다음 던전의 최소필요피로도 이어야 한다
dungeons : 각 던전의 [최소필요피로도, 소모 피로도] 가 담긴 2차원 배열
answer : 탐험할 수 있는 최대 던전 수

# 던전은 어떠한 순서대로도 탐험할 수 있다
```

- 내 풀이

```
import itertools

def solution(k, dungeons):
    answer = -1

    # 던전 탐험 순서의 모든 경우의 수
    # 각 case는 dungeons의 인덱스들의 permutation이다
    for case in list(itertools.permutations(range(0,len(dungeons)))):
        crntpower = k
        exploration = 0
        for i in range(0, len(dungeons)):
            if dungeons[case[i]][0] > crntpower:
                break
            crntpower -= dungeons[case[i]][1]
            exploration += 1

        # answer은 exploration의 최대값
        answer = max(answer, exploration)
    return answer
```

전력망을 둘로 나누기

```
wires : 전선 연결 정보
        ex) wires = [[1,2],[2,3],[3,4]] 면 1 -- 2 -- 3 -- 4 로 연결된 연결망
전선들 중 하나를 끊어서 만들어지는 두개의 연결망의 node 개수 차이가 가장 작아야 함
```

- 각 wire 연결 정보에 대해서 좌측 연결망의 노드 개수(l_count), 우측 연결망의 노드 개수(r_count)를 구한다

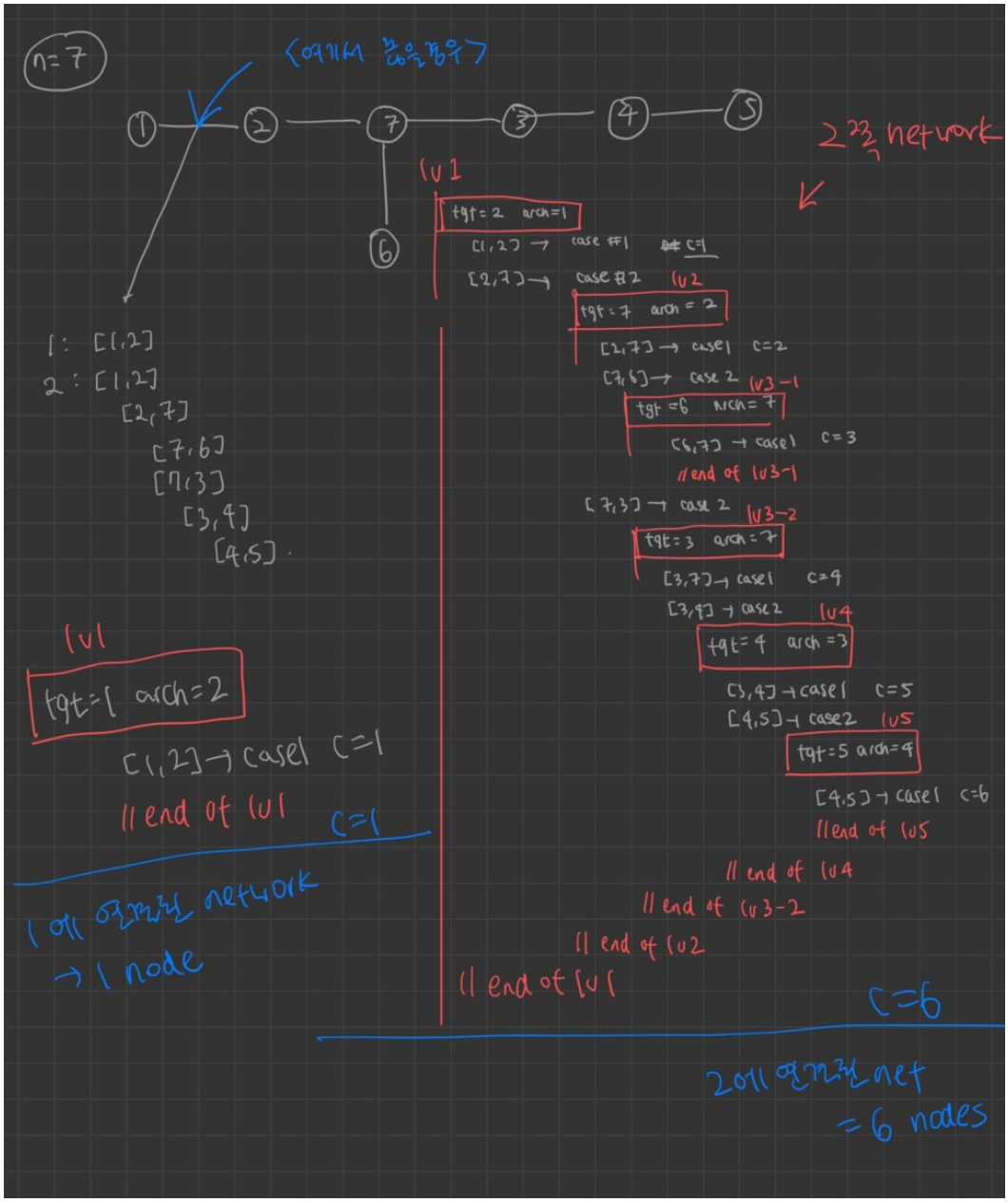
```
def countNodes(wires, tgt_node, arch_node, count):
    for wire in wires:
        if tgt_node in wire:
            connectednode = wire[:]
            connectednode.remove(tgt_node)
            if connectednode[0] == arch_node: # case 1
                count += 1
                continue
            else: # case 2
                # 이어져 있는 wire들을 따라가며 count를 증가시킨다
                count = countNodes(wires, connectednode[0], tgt_node, count)


    return count

def solution(n, wires):
    answer = len(wires)
    for l_node, r_node in wires:
        l_count = 0
        r_count = 0
        l_count = countNodes(wires, l_node, r_node,l_count) # 좌측 연결망의 node 개수
        r_count = countNodes(wires,r_node,l_node,r_count) # 우측 연결망의 node 개수

        if abs(l_count-r_count) < answer:
            answer = abs(l_count-r_count)
    return answer
```

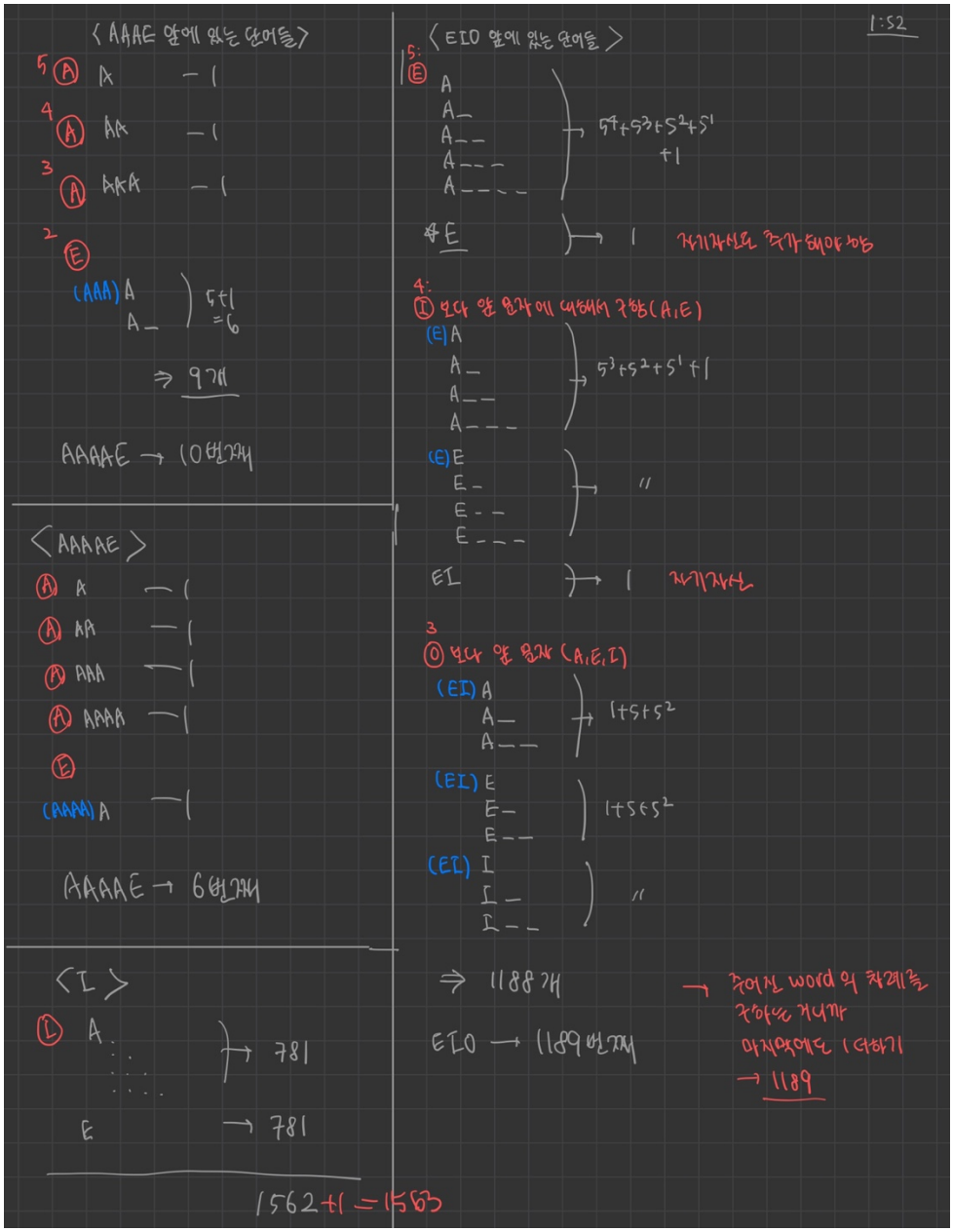
- wires = [[1,2],[2,7],[3,7],[3,4],[4,5],[6,7]] 에서 [1,2]를 끊었을 경우 정리(l_count = 1, r_count = 6)



- 🔔 **union find로 풀어보기**

모음사전

- 사전에 알파벳 A,E,I,O,U만을 사용하여 만들 수 있는 길이 5 이하의 모든 단어가 수록되어 있다. 이 사전에서 주어진 word가 몇번째 오는 지 구하시오
- A → AA → AAA → ... / AE는 AEA보다 우선한다




```
def solution(word):
    answer = 0
    # dic : 각 문자보다 우선한 문자의 개수를 저장 ex) "I" 보다 우선 순위인 문자는 "A","E" 2개
    dic ={"A":0, "E":1,"I":2,"O":3,"U":4}
    for i in range(0, len(word)):
        answer +=  dic[word[i]] * ((5**(5-i))-1)/4 + 1
        # 등비수열의 합 공식
        # +1 은 자기자신(word[i])이 붙는 경우 : answer는 word 이전의 단어 개수가 아니라 word의 위치를 구하는 것이므로 마지막 letter의 경우에도 1을 추가한다
    return answer
```