def solution(sizes):  
 s = len(sizes) # 샘플 명함개수  
 for i in range(s):  
 if sizes[i][0] > sizes[i][1]: # 둘중 더 큰 값이 우측에 오도록 바꿔준다  
 sizes[i][0],sizes[i][1]=sizes[i][1],sizes[i][0]  
 sizeS = max(sizes])[0] # 둘중 작은 값들의 집합중 가장 큰값  
 sizeB = max(sizes, key=lambda x:x[1])[1] # 둘중 큰 값들의 집합중 가장 큰값  
 answer = sizeB\*sizeS # 지갑의 크기를 답으로 넘겨준다  
 return answer

리뷰: 처음에는 단순히 세로와 가로를 반복하며 최대값을 푸는 문제로 접근했지만 가로와 세로를 바꾸는 요구사항을 보고 다시 생각하게 되었습니다. 우선 가로와 세로를 따로 구할것 없이 가로 세로중 더 큰값을 우측으로 저장하고 가로 세로중 작은 값을 왼쪽으로 저장한뒤 좌측에서 최대값과 우측에서 최대값을 비교하면 어떨까 하는 생각을 하였는데 다행히 생각하던대로 답을 도출해낼 수 있었습니다. 단순히 max 함수를 이용하면 자동으로 0번째 행을 기준으로 최대값을 주게 되는데 람다를 이용하여 따로 리스트를 추출하지 않고 구할수 있는 것을 배웠습니다.

def ma(edges,nodeMap):  
 global cnt # 엣지를 돌때 동안 카운트가 바뀌기에 카운트를 전역변수로  
 x,y = edges.pop(), edges.pop() # 엣지에서 꺼낸 두 값을 x 와 y에 저장  
 if a[x]==0 or a[y]==0: return # 둘중 하나가 가중치가 0이라면 바로 탈출  
 # 리프 노드를 우선순위로 놓는다  
 prior = x if len(nodeMap[x]) ==1 else y  
 other = x if prior!=x else y # 우선순위 노드가 아닌것을 other로 설정  
 while True:  
 # 우선순위 노드를 0으로 만들어 준다  
 if a[prior]<0 : # 만약 가중치가 음수라면 +1을 해준다  
 a[prior] += 1 # 우선순위 노드는 +1  
 a[other] -= 1 # other 는 -1  
 else: # 만약 가중치가 음수가 아니라면  
 a[prior] -= 1  
 a[other] += 1  
 cnt +=1  
 if a[prior] == 0: break # 만약 가중치가 0이라면 탈출  
 return cnt  
  
a = [-5,0,2,1,2]  
edges = [[0,1],[3,4],[2,3],[0,3]]  
cnt = 0  
def solution(a, edges):  
 nodeMap = [[] for \_ in range(len(a))] # 각 노드의 연결된 노드를 저장할 맵 생성  
 for i in range(len(edges)):  
 nodeMap[edges[i][0]].append(edges[i][1]) # 연결된 노드를 저장  
 nodeMap[edges[i][1]].append(edges[i][0])  
 # 각 노드를 방문하여 연결된 노드들을 저장  
 for i in range(len(edges)): # 엣지 방문하여 카운트 시작  
 ma(edges[i],nodeMap)  
 if cnt==0: answer=-1 # 만약 cnt가 0이라면 답은 -1로  
 else: answer = cnt # 아니라면 답은 카운트  
 return answer

리뷰: 처음에는 bfs 나 dfs 로 접근하려고 했지만 단순히 노드를 방문해서 한번에 연산을 끝내는 것이 아닌 엣지들을 기준으로 탐색이 시작되는 점을 생각하여 문제에 접근하였습니다. 우선 노드맵을 통하여 각 행에 행에 해당하는 수와 연결된 노드를 저장하였는데 다시 생각해보니 단순히 각 노드의 수만 세었으면 됬을것같다는 생각을 하였습니다. 하지만 Counter를 써도 다중 접근해야되기 때문에 지금의 방법이 더 나아 보일것같다는 생각을 하였습니다. 우선 핵심은 우선순위를 정하는 것이였습니다. 리프노드의 경우 탐색이 끝나게 되면 더이상 수정할 수 없기 때문에 리프노드를 우선순위로 놓고 리프노드가 음수일 경우와 양수일 경우를 구분하여 음수일땐 더해주고 양수일때는 빼주었습니다. 그리고 만약 엣지에서 연결된 두 노드중 하나의 가중치가 0이라면 연산을 할 필요가 없다고 생각되어 바로 탈출하도록 하였습니다. Cnt를 전역변수로 설정하여 어쩔수 없이 바깥에 선언하게 되었는데 함수안에서 cnt를 설정하는 법을 더 고민해봐야 할것같습니다. 그리고 아마 dfs 나 bfs 로도 풀수 있을것같은데 좀더 고민해봐야 할것같습니다.