**Kick Start 2020 Round C**

1. Countdown

첫번째 문제 답게 사실상 굉장히 쉬웠다. 문제를 이해하는 것도 쉬울 뿐만 아니라 구현을 함에 있어서 그냥 2중 for문을 사용하면 되는 것이었고, 따라서 코드가 길기는 했지만 쉬웠다. ‘k-countdown’이라는 용어를 정의하는데, 이는 k부터 1까지의 수를 순서대로 나열이 되는 경우가 주어진 문자열에 있으면 k—countdown의 개수를 출력하는 것이었고, 시간 초과 제한이 없다 보니 간단하게 풀 수 있었다.

2. Stable Wall

두번째 문제부터 확 어려워진 느낌이었다. 원래는 두번째 문제 정도는 20-30분 컷을 해야 하는 거지만 문제 이해가 너무 어려웠다. 이렇게 어떤 알고리즘을 주제로 푸는 문제인지 모르는 상태에서 문제를 푸는 것이 훨씬 어려운 것 같은 느낌이다.

이 문제는 greedy나 topological sort를 이용하면 되는 문제인데, 일단 문제 이해부터 하는게 맞다.

문제 자체는 어떤 특정한 규칙을 갖고 벽을 쌓는데 stable하지 않는 벽이 되는 경우에는 -1을, 아닌 경우에는 그 규칙에 맞는 문자열을 출력하라는 것이었다. 그러니까 밑에 쌓은 벽돌보다 위에 쌓은 벽돌을 각각 a,b라 하면 무조건 a는 b보다 우선 순위가 높아야만 하는 것이다. 그렇기 때문에 그래프를 이용해서 원래는 board에 노드 연결이 되면 해당 board칸에 1을 넣는 식으로 진행 했었고 그 이후에 topological sort를 이용하면 되는 것이라고 생각했다. 그러나 그렇게 하니 답으로 출력이 안되는 값들이 있었는데, 기본적인 tc는 맞는데 이것은 맞지 않는 것으로 보아 모든 경우를 따져주지 않고 하나의 경우/규칙만 각 문자당 고려하였기 때문이라고 생각한다.

따라서 고쳐서 풀었으며, 이번에는 함수의 정의없이 모든 경우를 구하였다. 이렇게 구한 뒤에 우선순위가 높은 것 순서대로 출력을 하였고, boolean 개념을 이용하여 최종 답을 도출했다.

추가로 topological sort알고리즘을 푸는 방법은

일단 알고리즘은 위상 정렬의 과정에서 그래프에 남아있는 정점중에 진입 차수가 0인 정점이 없다면 위상정렬 알고리즘은 중단 되고 이러한 그래프로 표현된 문제는 실행이 불가능하게 된다.

1. 들어오는 간선 수가 0인 정점 선택

- 여러 개가 존재할 경우 어떤 것을 선택해도 무방

- 초기에 간선 수가 0인 모든 정점을 큐에 삽입

2. 선택된 정점과 부속된 간선 삭제

- 선택된 정점을 큐에서 삭제

- 선택된 정점에 부속된 모드 간선에 대해 간선의 수를 감소

3. 모든 정점이 삭제/선택되면 알고리즘 종료

t = int(input())  
**for** case **in** range(t):  
 r,c = map(int, input().split())  
 past = **''** dic = dict()  
 **for** j **in** range(r):  
 file = input()  
 **for** k **in** file:  
 **if** k **not in** dic:  
 dic[k] = set()  
 **if** len(past) != 0:  
 **for** l **in** range(c):  
 currentalph = past[l]  
 pastalph = file[l]  
 **if** currentalph != pastalph:  
 dic[currentalph].add(pastalph)  
 *#wall을 세울 때 pastalpha보다 currentalpha가 먼저 와야 한다.  
 #즉 dic의 key가 value보다 우선 순위가 높은 값이다.* past = file  
 order = []  
 bad = **False  
 for** i **in** range(len(dic)):  
 a = 0  
 **for** k **in** dic:  
 **if** len(dic[k]) == 0:  
 *#현재로서 우선순위 1순위이면* order.append(k)  
 **for** j **in** dic:  
 **if** k **in** dic[j]:  
 dic[j].remove(k)  
 *#len(dic[k]) == 0이기 때문에 분명 우선순위가 현재로서 제일 높기 때문에 queue에서 popleft하는 것과 같은 원리* dic[k].add(**'beasdera'**)  
 *#아무거나 넣어주면 됨* a = 1  
 **break  
 if** a == 0: *#아직 dic전체를 돌지 않았는데 우선순위 높은 것이 존재하지 않는다면, 즉 cycle을 돌 상황이라면 break* bad = **True  
 break  
 if** bad: *#steady wall이 아닐 경우, 즉 bad == True 일 떄* print(**'Case #'**+str(case+1)+**': '**+str(-1))  
 **else**: *#stead wall일 경우* answer = **''  
 for** i **in** order:  
 answer += str(i)  
 print(**'Case #'**+str(case+1)+**': '**+answer)

**Kick Start 2020 Round B**

1. Bike Tour(2중 for문)

항상 첫번째 문제는 정말 초보도 풀 수 있을 만큼 쉬운 것 같다. 이 문제는 그냥 1에서 마지막 이전 번째까지 순서대로 탐색을 하면서 양쪽 보다 높이가 높은 경우에만 count += 1를 하면 되는 것이었다.

2. Bus Routes(이분 탐색/완전 탐색)

처음에는 이 문제가 배수 문제일 것이라는 생각을 하지는 못하고 그냥 탐색 문제일 것이라고 생각을 해서 dfs를 이용해서 문제를 풀었다. bfs문제를 푼 방법과 동일한 방법으로 dfs문제도 풀었는데, 기본적으로 주어진 testcase는 둘 다 통과를 무난하게 했지만 실제 testcase는 각각 시간 초과, 메모리 초과가 발생했다. 어떤 방법으로 탐색을 하던 그 문제가 발생할 것 같아 완전 탐색이 아닌 이분 탐색의 방법을 사용하기로 결정했다.

아래 코드가 처음에 내가 작성한 코드이다. 알고리즘은 맞으나 시간 복잡도가 문제이다.

t = int(input())  
**def** dfs(board, v, now, start):  
 **global** ans  
 a,b = len(board), len(board[0])  
 **if** v == a-1:  
 **if** board[v][-1] == 1:  
 ans = max(ans, start+1)  
 **return  
 elif** v < a-1:  
 **for** i **in** range(now, b):  
 **if** board[v+1][i] == 1:  
 dfs(board, v+1, i,start)  
  
**for** case **in** range(t):  
 n,d = map(int, input().split())  
 file = list(map(int, input().split()))  
 board = [[0]\*d **for** \_ **in** range(n)]  
 **for** k **in** range(n):  
 **for** j **in** range(d):  
 **if** (j+1)%file[k] == 0:  
 board[k][j] = 1  
 ans = 0  
 **for** i **in** range(d):  
 **if** board[0][i] == 1:  
 dfs(board,0,i,i)  
 print(**'Case #%s: %s'** %(str(case+1), str(ans)))

이 문제는 버스가 오는 시간 주기를 알 때 마지막 버스를 탈 수 있는 가장 늦은 시간을 구하는 문제이다.

따라서 아래 코드가 최종적으로 나오는 정답이다. 이분 탐색을 이용하여 해결하는 방법을 사용하였는데, 우선 이분 탐색으로 정답 가능 범위를 0~d일 사이에서 좁혀 나갔다. 정답 가능 범위의 가운데를 시작일로 잡고 여행을 시작할 때에 d일 이내에 여행을 마칠 수 있는지 확인하는 방법이다. 확인은 버스 노선을(버스 종류를) 처음부터 끝까지 탐색 하면서 현재 날짜에서 탐색중인 버스 노선을 탈 수 있는 최단 일을 구하면서 날짜를 갱신한다. 현재 날짜를 day, 버스 노선의 배차 간격을 a라고 할 때 현재 날짜가 a로 나누어 떨어지면 날짜를 그대로 유지하지만 나누어 떨어지지 않는다면 날짜가 배차 간격 a로 나누어 떨어질 수 있게끔 갱신해 준다. 즉, day += a-(day%a)를 해 주는 것이다. 이렇게 한 이후에 day가 d보다 작거나 같다면 정답이 가능한 경우이기 때문에 left를 m으로 갱신하고 반대는 정답이 불가하므로 right를 m-1으로 갱신한다. 또한, m은 무조건 (left+right+1)//2으로 하고 left < right일 때 만 while 문을 이용해 반복한다.

이렇게 해서 while 문을 빠져 나오게 되면 가장 늦게 출발 가능한 경우가 left이다.

**if** day % a > 0:  
 day += a-(day%a)  
 **if** day <= d:  
 left, right = m, right  
 **else**:  
 left, right = left, m-1  
 print(**'Case #%s: %s'** %(str(case+1), str(left)))

t = int(input())  
**for** case **in** range(t):  
 n,d = map(int, input().split())  
 file = list(map(int, input().split()))  
 left, right = 0,d  
 m = 0  
 **while** left < right:  
 m = (left+right+1)//2  
 day = m  
 **for** i **in** range(n):