1. 2839

- 예외의 경우 생각

- 우선 적으로 최선의 값으로 고려

- 반복문, 조건문 활용시 어떤게 더 적합할지 생각

2. 11399

- 적극적인 변수 생성

- 적극적인 함수 활용(해당 문제에선 sort(), split() 함수를 사용함)

3. 11047

- 리스트의 크기 유념(처음 정하기 애매하니까 append()적극 활용)

- 각 변수의 의미, 용도 코딩 하면서 잘 파악하고 있기

4. 1931

- numpy 못씀ㅋㅋ – 백준

- pop(index),

- 변수 초기화가 for문 밖에서 인지 안에서 인지 확실히 인지

- 문제 조건 자체에서 어떻게 고려하면 문제를 풀 수 있을 지를 떠오르자

- 원래는 2중 포문을 사용했는데 시간이 너무 길어짐

- 애초에 끝나는 시간을 먼저 고려하면 1차원 포문으로 끝낼 수 있다

- sort(key= lambda x:(x[0],x[1]))로 이차원 리스트를 정렬할 수 있음

5.1026

\*point에 관해

-두개의 리스트를 할당 연산자로 묶으면 두 리스트가 향하는 주소가 같은 거임

-그 중하나의 값이 변동 되었을 때 어떤 변수로 접근하든 그 안의 값은 다 바뀐 거

\*큰 값 추출 등:

-제일 큰 값을 가져오고 제외 시킬 때 pop()을 사용해서 그 다음 거를 찾으면

Index가 바뀌므로 원 list의 index가 중요한 경우는 그 형태를 그대로 두는게 중요

-조건을 추가해서 index는 손쉽게 가져올 수 있게 한다

-모든 것에 대해 순서를 바꾸면 결국 돌아온다. 해당 자리에 해당 값을 넣는 다는 생각으로 해야 함

6.1541

- split 구분자 여러 개 할 때는 re.split(‘[./,]’,a) 이런식으로

- 굳이 모든 마이너스와 플러스의 개수를 다 셀 필요는 없다. 마이너스는 하나만 있으면 되고 – \*(문제에서 잘 파악 -그리디 알고리즘의 메인 사고)

- 플러스가 앞에 나와 있을 경우 그 인덱스 보다 count를 해서 마이너스 앞 플러스가 몇 개 있는 지를 파악하면 훨씬 간편하다

- find(),index()로 문자열에 찾고자하는 값의 index를 반환할 수 있는데,

이 함수들 두 번째 인자는 인덱스의 start 포인트를 찾는 거다

7. 5585

- 너무 잘했음 코드 간결하게 쓰기

8.2217

-한가지 조건 상정후 하나씩 늘려가기

9.10162

-문제 조건 하나씩 적어나가기

10.10610

-문자열 인덱싱 잘 활용하면 좋음 – 리스트랑 비슷한 특성

-처음입력 받으면 기본이 문자열인 걸 기억 – 굳이 split안 해도 하나하나 가져올 수 잇음:

리스트에 문자열 인덱스 0부터 끝까지 집어넣음으로써

-문자열에다 연이어 붙이는 건 + 라는 좋은 연산자가 있음을 기억

11.1789

-두번 틀림 – 내 건 맞게 잘 나왔는데

- for문대신 while문으로 작성했더니 맞음 -> 아마 for문에서 범위가 너무 커지면 오류 나는 듯

12.1946

-2중 포문 최소화!

-리스트 min(),max() --- np.array.max를 대체할 수 있다!

-원래는 min(), 2중포문등을 사용했는데 다 시간초과->temp사용했는데도 시간 초과

-input() 시간초과 초래하기가 쉬운듯..sys.stdin.readline()쓰란다

-출력은 상관 없었음

-min(), 2중포문 둘다 시간 복잡도가 커졌음 🡪 temp를 사용할 것

13. 13305

-oil=list(map(int,sys.stdin.readline().split()))

이거 편함: 공백 들어간 한줄의 정수를 띄어쓰기 고려 각각 한칸씩 리스트에 넣어줌

-문제의 본질 파악 필요

-문제에서 나오는 실생활이랑 코딩 상에서만 고려야해야할 부분 구분지을 수 있어야 됨

**\*14.1339**

-strip()으로 맨끝 공백문자 제거 가능

-str타입 변수 = “”.join(list 타입 변수) – 리스트 문자열로 변환 가능

-dictionary등 리스트 제외 다른 타입들도 적극 활용할 수 있으면 좋을 듯

-보기 복잡하면 함수를 만들어서 역할 배분

-변수 적극 활용하는 동시에 이게 꼭 필요한지, 혹시 역할이 겹치진 않는지 – 머리가 헷갈릴 수 있음

**\*15. 1715**

큐로 푸는 문제

* List로 하면 한계: 시간 초과 / 하나하나 계산 해줘야하는 불편함
* 우선순위 큐라는 걸 써보면 좋을 듯
* 각 문제마다 맞는 데이터 타입이 있다는 걸 생각
* Del로 리스트 삭제가능 del a[0] 이런식으로

16.4796

- map()의 용도 – int여러개 담겨있는 2차원 리스트 만드려면 필요함

17.1439 – ceil() 올림 –MATH 안에 있음

18.1744

-plus랑 minus경우 나눠서 생각

-예외 적인 경우 1, 0, -1을 빠르게 잘 파악

- 결국 그리디는 제일 쉬운 방법을 보고, 그게 안 통하는 예외 경우를 잡아주면 됨

- 그 두 경우를 잘 구분해주고 서로 영향이 가지않게끔

19.16953.

- 역발상—문제를 뒤틀어서 보자 –

문제에선 a->b였지만 b->a로 만드는 걸 생각하면 편하다

- 어떤 조건이 우선 순위인지, 그 조건을 달리 말하면 어떤 의미인지

2만 곱할 수 있다 – 2의 배수여야만한다

맨 뒤의 1을 뺄 수 있다 – 2의 배수가 아닌경우는 1을 빼면 10의 배수인지 확인 이게 우선순위

왜냐면 1을 뺄 수 있으면 2의 배수가 아니기 때문

-exit()사용해도 문제 없음!

20. 1202

우선순위 큐:

데이터를 정렬된 상태로 저장하는 우선순위 큐:

데이터 추가는 어떤 순서로 해도 상관 없지만, 제거 될 땐 가장 작은 값을 제거함 – heapq 모듈

Que=PriorityQueue(maxsize)

힙 모듈은 리스트와 별개의 자료구조가 아니다

Heap =[]

Heapq.heappush(heap, 원소) – O(logN)의 시작 복잡도

Heappop()-> 힙에서 제일 작은 원소 삭제후 리턴 – o(logN)

Heapify() -> 기존 리스트를 힙으로 변환 – o(N)

[응용]최대 힙 – 튜플로 원소를 추가. (우선순위, 값) -> (-value, value)

* Heapq.heappop(heap)[1]

[응용]k번째 최소값/최대값

* K번 heappop하면됨

[응용]힙 정렬

* 힙을 만들고, heappop을 이용하면서 다른 리스트에 append()

21.2864

-문자열 대체함수 replace는 대체된 결과를 반환하는 함수

22.1080 - 다시보기

-정 복잡해서 어디서부터 손댈지 모르면 구현부터 해버리자

- map()은 하나씩 빼서 리스트로 만들어줌

- strip()은 마지막 개행문자 없애줌

23.2437

- 문제를 쉽게 풀 수 있는 방법은 존재한다

-다시보기

24.1449

- 문제 이해부터 확실히

25. 1543

-replace()에 세번째 인자는 한번에 몇 개를 치환할지 결정함

-지우고 나서 그 다음 영향을 받는지 안 받는지도 생각

26.1783

- 모든 경우를 깔끔히 정리하는 연습 필요

-다시보기

27.11000

-우선순위 큐가 자동으로 작은 값을 빼준다는 생각

-다시 보기

28.2847

-역발상

난이도가 낮은거부터 계산하면 빼고 난다음 뒤로 갈수록 그 전에 것도 다시 고려할 필요가 있음

* 그렇기에 난이도 높은 거부터 계산하면 바로 위에 거(난이도 한 단계 낮은 거)가 기준점이 된다. 그에 반해 난이도가 낮은거는 기준점이 안 될 수도 있음

29.1700

문제 이해 필요:

멀티탭으로 상정한 N만큼 미리 입력 받은 값은 멀티탭이 실질적으로 아님

0-k까지의 전자기기를 하나씩 비어 있는 멀티탭에 꼽는 다고 생각하면 정리가 됨,

그리고 개수가 아닌 가장 나중에 사용하는 거를 뽑으면 됨

리스트.count(셀 값)

리스트.index(찾을 위치)

* 다시 풀어보기