2023년 7월 26일 알고리즘 문제 풀이

백준 6549

문제 링크

시도1

나의 생각

왼쪽부터 반복문을 통해 조건에 따라 스택에 사각형의 넓이를 넣고 수정하는 방법을 사용하였다. 스택에는 사각형의 높이와 너비가 원소인 배열을 집어넣었다.

예를 들어, 이번 사각형의 높이가 4라면 [4,1]을 넣어주는 방법이다. 맨 처음을 위해 스택이 비어있을 땐 조건과 무관하게 스택에 삽입하였다. 이후는 최근 스택에 들어간 사각형의 높이(이전 사각형)와 현재 반복문에서 주목하는 사각형(현재 사각형)의 높이를 비교하였다. 만약 현재 사각형의 높이가 더 크거나 같다면, 스택에 들어 있는 사각형은 너비를 1 늘렸다. 이후는 현재 사각형과 더 이전에 들어갔던 사각형들의 높이를 비교하였다. 그래서 너비를 1 늘려주거나 만약 현재 사각형의 높이가 작으면 너비를 늘리지 않고 스택에 들어 있는 사각형의 높이와 너비를 곱하여 넓이를 구한 후 최댓값과 비교하여 저장하는 방법을 사용하였다. 로직을 정리해보자면 다음과 같다. 문제에서는 각 케이스별로 출력해야 하지만, 편의를 위해 하나의 케이스에서 발생하는 로직만 표시하였다.

1. 반복문을 통해 사각형 하나씩 살펴본다.(각 테스트케이스가 아니라 케이스 내에서 사각형의 배열을 순회하는 반복문임)
2. 만약 스택이 비어있다면, 현재 사각형의 높이와, 너비인 1을 원소로 하는 배열을 스택에 삽입한다.
3. 스택이 비어있지 않다면, 최근 스택에 들어간 사각형의 높이와 현재 사각형의 높이를 비교한다.
4. 현재 사각형의 높이가 높거나 같다면, 이전 스택에 들어있는 **모든** 사각형의 너비를 1씩 더해준다.
5. 현재 사각형의 높이가 낮다면, 스택에 포함된 사각형 중 높이가 더 높은 사각형들은 너비와 곱해서 넓이를 구해준다.
6. 초기값을 0으로 설정한 최댓값과 비교하여 가장 큰 값으로 업데이트하 저장한다.
7. 1에서 시작한 반복문이 끝난 후, 스택에 남아있는 사각형이 있다면 5,6 의 방식을 통해 최댓값을 업데이트한다.
8. 출력한다.

4에서는 모든 사각형이라고 단정지은 반면, 5에서는 일부 사각형만 로직에 포함되는 이유가 있다. 스택에 최근에 들어온 사각형일 수록 이전보다 높이가 높다. 애초에 조건상 더 낮은 사각형은 스택에 들어올 수 없기 때문이다. 다시 말해, 가장 최근에 들어온 사각형의 높이가 스택에 있는 사각형 중 높이가 가장 큰 사각형이다. 따라서 4에서는 모든 사각형이 현재 사각형보다 높이가 낮을 수밖에 없다.

반면, 이 성질로 인해 5에서는 모든 사각형이라고 단정할 수 없다. 가장 최근의 사각형은 현재 사각형보다 높이가 높지만 스택에 깊숙히 들어갈 수록 현재 사각형보다 높이가 낮은 사각형이 나타날 수 있기 때문이다. 그래서 비교가 필요하고 더 낮은 사각형이 나타난다면 그 이후에 만날 사각형들은 모두 높이가 현재 사각형보다 낮기 때문에 굳이 비교가 무의미하다.

결과

시간초과

코드

```python
import sys
while True:
    case = list(map(int,sys.stdin.readline().split()))
    n = case[0]
    if n == 0:
        break
    squares = case[1:]
    stk = []
    max_val = 0
    for i in range(n):
        h = squares[i]
        stk.append([h,1])
        if not stk:
            continue
        for j in stk:
            if j[0] == 0:
                continue
            if h >= j[0]:
                j[1] += 1
            else:
                val = j[0]*j[1]
                max_val = max(max_val,val)
                j[0] = 0
    for k in stk:
        tmp = k[0]*k[1]
        max_val = max(max_val,tmp)
    print(max_val)
```

n은 100000이하의 자연수이다. 따라서 최악의 경우에는 n^2만큼 시간복잡도가 발생하고, 이로 인해 시간초과가 발생한 것 같다.

시도2

나의 생각

시간 복잡도를 개선하기 위해 더이상 반복문 수행이 무의미한 경우에는 `break`를 통해 반복문을 종결시키려고 하였다. 그럼에도 불구하고 예제 출력은 올바르게 됐지만 오답처리 되었다. 기존에 스택이 비어있는지 확인하는 조건이 선행되었는데 스택이 있을 때의 조건문을 먼저 설정하였더니 정답이 되었다.

생각해보니 위의 코드에서 바로 스택에 `append`를 통해 삽입해주었으니 스택이 비어있을 리가 없고 스택이 비었다는 조건문이 제대로 작동될 리가 없었다

애초에 로직 자체를 조건문이 무의미하도록 작성한 탓이였다. 기존에 풀면서 이유는 모르겠지만 그냥 조건문의 순서를 바꾸었더니 정답처리 되었다. 코딩테스트 때도 못풀겠으면 우선 시도해보고 이유를 찾아야 하나..

결과

정답

코드

```python
import sys

while True:
    arr = list(map(int,sys.stdin.readline().split()))
    n = arr[0]
    if n == 0:
        break
    stk = []
    ans = 0
    for i in range(n+1):
        if i == 0:
            continue
        h = arr[i]
        if stk and stk[-1][0] > h:
            while stk:
                tmp = stk.pop()
                if stk:
                    dif = stk[-1][1]+1
                else:
                    dif = 1
                cnt = tmp[0]*(i-dif)
                ans = max(ans,cnt)
                if not stk or stk[-1][0] <= h:
                    break
        if not stk or stk[-1][0] <= h:
            stk.append([h,i])
    while stk:
        tmp = stk.pop()
        if stk:
            dif = stk[-1][1]+1
        else:
            dif = 1
        cnt = tmp[0]*(n+1-dif)
        ans = max(ans,cnt)
    print(ans)
```

백준 10000

문제 링크

시도1

나의 생각

각 원의 중심과 반지름이 주어지니 원과 y=0 선이 닿는 교점인 왼쪽 점과 오른쪽 점을 배열로 만들었다. 가장 왼쪽 점과 오른쪽 점을 양 끝으로 두고 원의 갯수가 n이고 점의 갯수는 2n이 된다는 것을 이용해 list를 만들면 어떨지 생각해보았다. 근데 생각해보니 양 쪽 끝이 고정된 값이 아니고, 어떤 점들이 같은 원에 있는 점인지 생각하는게 복잡했다. 결국 실패

결과

오답

코드

```
import sys

n = int(sys.stdin.readline())
arr = []
for i in range(n):
    x,r = map(int,sys.stdin.readline().split())
    arr.append([x+r,x-r])
arr.sort(key = lambda x : x[1])
graph = [0 * (2*n+1)]
cnt = 0
graph[1] = arr[0][1]
```

시도2

나의 생각

왼쪽 점과 오른쪽 점을 원소로 하는 배열은 하나의 원을 의미한다. 이 배열들을 왼쪽 점을 기준으로 오름차순으로 정렬하였다. 왼쪽 점읮 좌표가 같으면 오른쪽 점을 내림차순으로 정렬하였다. 즉 원이 왼쪽에 있을 수록, 반지름이 클 수록 앞이다. 반복문을 통해 각 원을 스택에 삽입하고 다음 원들이 스택에 들어있는 원의 내부를 나누어주는지 확인하였다. 기본적으로 원이 생기면 '내부'라는 새로운 영역이 생기므로 영역의 갯수에 1을 더해주었고, 시작할때는 외부 영역도 있으니 초기값을 0이 아닌 1로 시작하였다. 또 내부가 나뉘어지면 내부 영역이 2개가 되므로 또 1을 더해주었다. 스택에 원을 넣고 다음 원을 볼 때는 왼쪽 점이 스택에 있는 원과 왼쪽 점의 좌표가 같은지, 같다면 오른쪽 점은 어디인지를 표시해서 어디까지는 끊기지 않고 원이 있는지 표시하였다. 만약 스택에 있는 원의 오른쪽 점까지 끊기지 않고 이어진다면 영역 값에 +1을 해주고 스택에서 제거하였다. 끊겼다면 스택에서 제거하고 다음 원을 검사하였다.

이 로직의 문제는 전체에서 내부가 나누어지는 원이 하나일때만 고려할 수있다는 것이였다.

결과

오답

코드

```python
import sys

n = int(sys.stdin.readline())
arr = []
for _ in range(n):
    x,r = map(int,sys.stdin.readline().split())
    arr.append([x-r,x+r]) # 왼쪽점과 오른쪽 점을 원소로 하는 배열 삽입
arr.sort(key = lambda x: (x[0], -x[1])) # 왼쪽 점이 가장 작은 순서로, 같은 원들은 오른쪽 점이 가장 큰 순서로 정렬
cnt = 1 # 원이 그려지면 무조건 가장 밖은 생기므로 1부터 시작
right = arr[0][1] # 가장 첫번째 원의 오른쪽 끝점 설정
stk = [] # 현재 가장 밖으로 설정한 원을 저장해둔다.

full = False # 내부가 절반으로 나뉘어졌는지 표시하는 flag
for dot in arr:
    if not stk: # 처음 원이므로 가장 큼. 시작점, 끝점을 설정
        stk.append(dot)
        tmp = stk[0][0] # 시작점
        right = stk[0][1] # 끝점
        max_val = stk[0][0] # 현재 원 내부의 왼쪽부터 안 끊기고 최대로 이어진 지점. 맨처음엔 왼쪽에서 시작
        cnt += 1 # 원이 그려졌으니 나뉘어짐
        continue
    if dot[0] == tmp: # 시작점이 같은 원. 내접한다는 뜻
        cnt += 1
        if dot[1] != right: # 시작점이 같은 원 중 가장 큰 원의 끝점이 아직 닿지 않아서 절반으로 나누지 못했음.
            max_val = dot[1] # 시작점부터 여기까지는 끊기지 않고 이어졌다. 아직 큰 원의 내부가 반으로 나눠질 가능성 존재
    else: # 시작점이 기존과 달라졌다.
        if dot[0] >= right: # 기존에 가장 큰 원 밖으로 새롭게 시작되었음. 재설정
            stk.pop()
            stk.append(dot)
            tmp = stk[0][0]
            right = stk[0][1]
            max_val = stk[0][0]
            cnt += 1
            continue
        else: # 아직 기존 원 내부에 있음.
            if dot[0] > max_val: # 아직 내부지만 기록해둔 끝점과 시작점이 달라짐. 내부가 절반으로 나눠질 가능성 사라짐.
                cnt += 1
            elif dot[0] == max_val: # 끝점에서 다시 시작점이 시작되었음. 여전히 가능성은 존재.
                max_val = dot[1]
                cnt += 1
                if dot[1] == right: # 끝 점이 가장 큰 원의 끝점에 닿았음. 내부는 반으로 갈라졌음. 나눠진 공간을 +1 해야함.
                    cnt += 1
                    full = True
                else: # 아직 닿진 않았으나 더 확인해봐야함
                    max_val = dot[1]
print(cnt)
```

시도3

나의 생각

위의 문제를 수정해서 다시 코드를 작성하였다. 예제는 기대 출력값이 올바르게 나왔지만 문제는 오답처리 되었다. 이 로직의 반례를 찾는데 7시간이 걸렸다..

결과

오답

코드

```
import sys

n = int(sys.stdin.readline())
arr = []
for _ in range(n):
    x, r = map(int, sys.stdin.readline().split())
    arr.append([x-r, x+r])
arr.sort(key=lambda x: (x[0], -x[1]))
stk = []
cnt = 1

for dot in arr: # 원 하나씩 본다.
    cnt += 1 # 원이 있으면 원 내부 라는 영역이 무조건 생성되므로 +1
    if not stk: # 맨 처음 차례라면 일단 넣는다.
        stk.append(dot)
        continue
    left = stk[-1][0] # 현재 가장 가까운 밖의 원의 왼쪽 끝과 오른쪽 끝 설정
    right = stk[-1][1]
    if dot[0] == left: # 현재 가장 가까운 밖의 원과 왼쪽 끝이 동일할 때
        stk.append(dot) # 이 원을 새로운 가장 가까운 밖의 원으로 설정
    elif dot[0] > left: # 가장 가까운 밖의 원과 왼쪽 끝이 다를 때
        stk.pop() # 기존의 가까운 밖의 원은 내부를 나눌 수 없게 됐으니 제거
        if dot[0] < right: # 하지만 여전히 가장 가까운 밖의 원의 내부에 있을 때
            stk.append(dot) # 현재의 원을 새로운 가장 가까운 밖의 원으로 설정
        elif dot[0] == right: # 이번 원이 가장 가까운 밖의 원의 끝에서부터 시작되는 원이라면?
            if not stk: # 현재 어떤 원의 내부가 아니라면?
                stk.append(dot) # 가장 가까운 원으로 설정
            elif dot[1] != stk[-1][1]: # 원의 내부지만 오른쪽 끝이 닿지 않았다면?
                stk.append(dot) # 그 원을 가장 가까운 밖의 원으로 설정
            elif dot[1] == stk[-1][1]: # 가장 가까운 밖의 원의 오른쪽 끝에 닿았다.
                cnt += 1 # 위아래로 영역이 나눠졌으므로 1 추가
                stk.pop() # 만족한 원은 제거
                stk.append(dot) # 검사를 위한 원 설정
        else: # 기존 밖의 원의 외부이면서 접하지 않는다면?
            stk.append(dot) # 새로운 목표 설정

print(cnt)
```

반례

출처 : https://velog.io/@cherrym/baekjoon-10000-areas-made-of-circle

같은 정글 6기 옆반 분이신 것같은데 감사합니다..

10
-1 5
-4 2
-1 1
0 2
3 1
-1 8
15 8
11 4
11 20
27 4

이 값을 input으로 넣으면 output은 **13**이 나와야한다.

디버깅을 해보니 알 수 있었다.

왼쪽 점의 좌표가 7인 원을 처음으로 만날때, 기존 스택에 들어있는 원의 갯수는 4개이다. 각 왼쪽 점과 오른쪽 점의 좌표는 다음과 같다.

[-9, 31], [-6, 4], [-2,2], [2,4]

내 로직에 의하면 [-6,4] 원은 이미 내부가 반으로 나눠졌음이 명확해 졌으니 영역값인 `cnt`를 1 더해주고 스택에서 `pop` 되었어야 한다. 하지만 가장 최근에 스택에 들어간 원이 아니여서 고려되지 않았다.

따라서 기댓값인 13이 나오지 않고 11이 나왔다.

시도4

나의 생각

이전 시도의 문제점은 원의 내부를 검사하던 중 새로 나타난 원을 스택에 넣으면서 검사하는 대상이 변경된 것에 있다고 판단하였다. 따라서 각 원마다 하나씩 검사를 끝마치도록 코드를 수정하였다. 어차피 반복문을 통해 모든 원이 검사가 될테니 중간에 변경할 필요가 없었다.

결과

정답

코드

```python
import sys

n = int(sys.stdin.readline())
circles =[]
stk=[]
ans = n+1 # 원이 생기면 원 내부 라는 새로운 영역이 생김. 가장 처음에는 밖도 생기니 +1

```

```python
for i in range(n):
    x,r = map(int, sys.stdin.readline().split())
    circles.append([x-r, x+r]) # 원의 왼쪽 끝, 오른쪽 끝 저장
circles.sort(key = lambda x: (x[0], -x[1])) # 왼쪽 끝을 기준으로 오름차순 정렬, 같을땐 오른쪽 끝을 내림차순 정렬

start = circles[0][0] # 가장 왼쪽에 있는 원의 왼쪽
end = circles[0][1] # 가장 왼쪽에 있는 원의 오른쪽

for i in range(1,n): # 가장 왼쪽 원 제외하고 반복문
    x1 = circles[i][0] # 주목하는 원의 왼쪽
    x2 = circles[i][1] # 주목하는 원의 오른쪽
    if start == x1: # 이전 원과 왼쪽이 동일하다면?
        stk.append([x2,end]) # x1부터 x2까지는 현재 주목하는 원이 차지했으니 x2부터 end까지만 원이 차면 내부를 반으로 나눈다.
        end = x2 # 새롭게 끝을 이전원의 오른쪽으로 설정
    else: # 이전 원과 왼쪽 끝이 동일하지않다.
        start = x1 # 새롭게 시작점을 현재 주목하는 원의 왼쪽끝으로 설정
        while len(stk)>0: # 스택에 원소가 있다는건 아직 내부가 반으로 나뉘어지는지 검사해야할 원이 남았다는 뜻.
            if start == stk[-1][0]: # 방금 스택에 들어간 원소의 0번 인덱스는 어디까지 원이 이어져있었는지를 말한다. 거기서부터 새로운 원이 시작된다면?
                end = x2 #새로운 원의 오른쪽을 끝점으로 설정
                if stk[-1][1] == x2: #1번 인덱스는 검사하는 원의 오른쪽이다. 새로운원이 거기에 닿았다면?
                    ans += 1 # 내부가 반으로 나뉘어졌으니 영역을 +1 한다.
                    stk.pop() # 검사가 끝났으니 stk 제거
                    break
                else: # 아직 닿지 못했다면?
                    stk[-1][0] = end # 원이 끊어지지 않고 이어져있었던 곳을 현재 원의 오른쪽으로 수정한다.
                    break
            elif start > stk[-1][0]: # 더이상 원이 이어지지 않았다. 현재 검사하는 원은 내부가 반으로 나뉘지 않음.
                end = stk.pop() # 검사할 필요 없으니 제거
            else:
                break
        end = x2 # 끝을 현재 원의 오른쪽으로 재설정.

print(ans)
```