Nhóm 3

Subtask 1 – Thuật toán trâu ($n \leq 1000$)

1. Kỹ thuật sử dụng

- Kỹ thuật: Decrease and Conquer
- Phân loại: Dạng duyệt từng phần tử và giảm dần bài toán.

Giải thích:
Ở mỗi bước, ta xét thêm một học sinh mới, và quyết định xem học sinh đó có thể gia nhập một đội đã tồn tại hay phải tạo đội mới. Bài toán được giảm dần bằng cách lần lượt thêm từng học sinh vào kết quả đã có.

2. Nhận xét để chọn kỹ thuật

Đối với subtask nhỏ, ta có thể duyệt toàn bộ các học sinh trước đó cho mỗi học sinh hiện tại mà không lo vượt thời gian.
Bài toán yêu cầu kiểm tra điều kiện "bao phủ", tức là xem một học sinh có thể nằm trong phạm vi đội của học sinh khác hay không. Do đó, cách kiểm tra tuần tự là hợp lý.

3. Mô tả thuật toán

1. Với mỗi học sinh i, ta tính:
   - `L_i = x_i - r_i`
   - `R_i = x_i + r_i`

     (đây là đoạn mà học sinh i có thể bao phủ).
2. Sắp xếp tất cả học sinh theo `L_i` tăng dần, nếu bằng nhau thì theo `R_i` giảm dần.
3. Duyệt qua từng học sinh:
   - Tìm giá trị `max_R` lớn nhất trong các học sinh trước đó.
   - Nếu `R_i > max_R`, học sinh i không nằm trong phạm vi bao phủ nào → cần tạo đội mới.
4. Biến đếm `count` là số đội tối thiểu.

4. Cài đặt thuật toán

```python
import sys
input = sys.stdin.readline

n = int(input())
students = []

for _ in range(n):
    x, r = map(int, input().split())
    L, R = x - r, x + r
    students.append([L, R])
```

```python
students.sort(key=lambda p: (p[0], -p[1]))

count = 0
for i in range(n):
    max_R = -10**18
    for j in range(i):
        max_R = max(max_R, students[j][1])
    if students[i][1] > max_R:
        count += 1

print(count)
```

Subtask 2 – Thuật toán tối ưu ($n \le 5 \times 10^5$)

1. Kỹ thuật sử dụng

Thuật toán được xây dựng dựa trên kỹ thuật Transform and Conquer, kết hợp với tư tưởng tham lam (Greedy).

Thay vì kiểm tra từng cặp học sinh xem có thể vào chung đội hay không (rất tốn thời gian), ta biến đổi bài toán sang một dạng dễ xử lý hơn – đó là bài toán về đoạn bao phủ trên trục số.

Cụ thể, với mỗi học sinh `i`, ta định nghĩa:

$$L_i = x_i - r_i, R_i = x_i + r_i$$

Đoạn `[L_i, R_i]` thể hiện phạm vi mà học sinh `i` có thể "bao phủ".
Nếu học sinh `j` có đoạn nằm hoàn toàn trong đoạn của `i`, thì `j` có thể gia nhập đội của `i`.

Như vậy, mục tiêu của chúng ta trở thành:
Tìm số đoạn bao phủ tối thiểu sao cho tất cả các học sinh đều nằm trong ít nhất một đoạn.

2. Nhận xét để chọn kỹ thuật

Sau khi chuyển bài toán sang dạng đoạn [L, R], ta nhận ra rằng nếu sắp xếp các đoạn theo L tăng dần (và nếu L bằng nhau thì theo R giảm dần), thì ta có thể duyệt toàn bộ danh sách chỉ một lần.
Mỗi khi gặp một đoạn có R lớn hơn tất cả các đoạn trước (tức là mở rộng vùng bao phủ), ta cần tạo thêm một đội mới.

Chiến lược này là tham lam, vì ở mỗi bước ta luôn chọn mở rộng phạm vi bao phủ xa nhất có thể để giảm số đội phải tạo ra.

3. Mô tả thuật toán

1. Sắp xếp danh sách các học sinh theo:
   - L tăng dần
   - Nếu hai học sinh có cùng L, thì sắp theo R giảm dần
2. Khởi tạo:

3. Duyệt lần lượt từng đoạn:

- Nếu `R > max_R`, nghĩa là đoạn này mở rộng phạm vi bao phủ hơn các đoạn trước → cần **tạo đội mới**.
  Khi đó:

```
count += 1
max_R = R
```

1. Sau khi duyệt hết, biến `count` chính là số đội tối thiểu cần tạo.

4. Cài đặt thuật toán

PYTHON

```python
import sys
input = sys.stdin.readline

def solve():
 n = int(input())
 students = []

 for _ in range(n):
     x, r = map(int, input().split())
     L, R = x - r, x + r
     students.append((L, R))

 # Sắp xếp theo L tăng, R giảm
```

```
    students.sort(key=lambda p: (p[0], -p[1]))

    count = 0
    max_R = -10**18

    for L, R in students:
        if R > max_R:
            count += 1
            max_R = R

    print(count)

if __name__ == "__main__":
    solve()
```