Interval Tree

Trong một số bài toán, việc giải quyết thông thường là rất khó khăn, không phải về mặt giải thuật mà là về độ phức tạp của thuật toán. Độ phức tạp càng cao thì thời gian chương trình chạy càng lớn. Để giải quyết vấn đề này, người ta đã nghĩ ra những cấu trúc dữ liệu đặc biệt, một trong số đó là Interval Tree.

Xét bài toán quen thuộc sau:

Bài toán:

Cho N hình chữ nhật có tọa độ nguyên nằm trên mặt phẳng tọa độ và có các cạnh song song với các trục. Mỗi hình chữ nhật được cho bởi tọa độ đỉnh trái trên và đỉnh phải dưới.

Yêu cầu: Hãy tính diện tích mà tất cả các hình chữ nhật này phủ lên.

Input: COVER.INP

- Dòng đầu là số N – số hình chữ nhật ($1 \le N \le 10000$)
- N dòng tiếp theo, dòng thứ i gồm 4 số X_i, Y_i, A_i, B_i thể hiện tọa độ đỉnh trái trên và đỉnh phải dưới của hình chữ nhật thứ i ($-20000 \le X_i, Y_i, A_i, B_i \le 20000$)

Output: COVER.OUT

- Gồm một dòng duy nhất là đáp số của bài toán

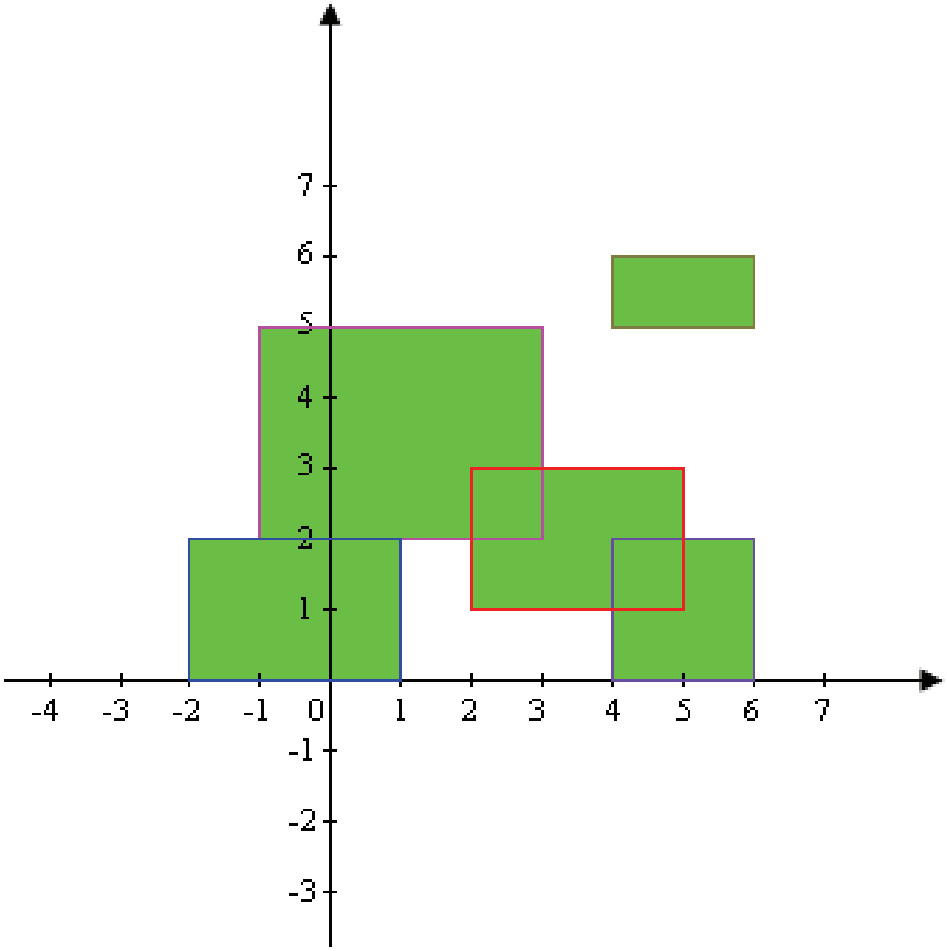
Thời gian chạy: 0.5s

Ví dụ:

COVER.INP	COVER.OUT
5 -2 2 1 0 4 2 6 0 -1 5 3 2 4 6 6 5 2 3 5 1	28

Minh họa:

Interval Tree



Hướng dẫn:

Để tiện cho việc phân tích bài toán, chúng ta sẽ gọi hoành độ của các đỉnh hình chữ nhật là H_1, H_2, …, H_{2*N} được sắp xếp theo chiều tăng; tung độ của các đỉnh hình chữ nhật là T_1, T_2, …, T_{2*N} (giả sử các tung độ được liệt kê theo chiều tăng - ở đây chú ý rằng: chỉ có hoành độ được sắp xếp tăng, còn tung độ là do chúng ta quy ước). Cách làm thông thường là sắp xếp H theo chiều tăng như đã giả sử ở trên, sau đó tính diện tích ở từng khe H_1H_2, H_2H_3, …, $H_{2*N-1}H_{2*N}$. Như vậy, độ phức tạp của thuật toán của chúng ta là $O(N^2)$, thời gian là $T(N(\log_2 N + N))$, khó có thể đáp ứng thời gian chạy là 1s. Để cho tiện, tất cả dữ liệu của chúng ta đều coi như sắp xếp bằng Quick Sort.

Trong hoàn cảnh đó, rất may mắn, chúng có một cấu trúc dữ liệu đặc biệt khác có thể thực hiện được điều này với độ phức tạp $O(N\log_2 N)$ và thời gian chạy là $T(2N\log_2 N)$. Đó là dùng Interval Tree, bản chất của Interval hết sức đơn giản, chúng ta sẽ xét rõ hơn qua phân tích chi tiết bài toán này.

Thuật toán của chúng ta về bản chất cũng giống với thuật toán vừa đề cập:

- Sắp xếp các hoành độ tăng dần.
- Tính diện tích che phủ từng khe H_1H_2, H_2H_3, …, $H_{2*N-1}H_{2*N}$ của hoành độ, lấy tổng, ta được đáp số của bài toán.

Trước hết, ta gán cho mỗi hoành độ đã được sắp xếp một nhãn, nhãn đó sẽ gồm Low, High lưu lại tung độ của hình chữ nhật chứa hoành độ đó và nhãn Open. Open sẽ chứa giá trị *true* nếu đó là đỉnh trên trái của hình chữ nhật, *false* nếu đó là đỉnh dưới phải của hình chữ nhật.

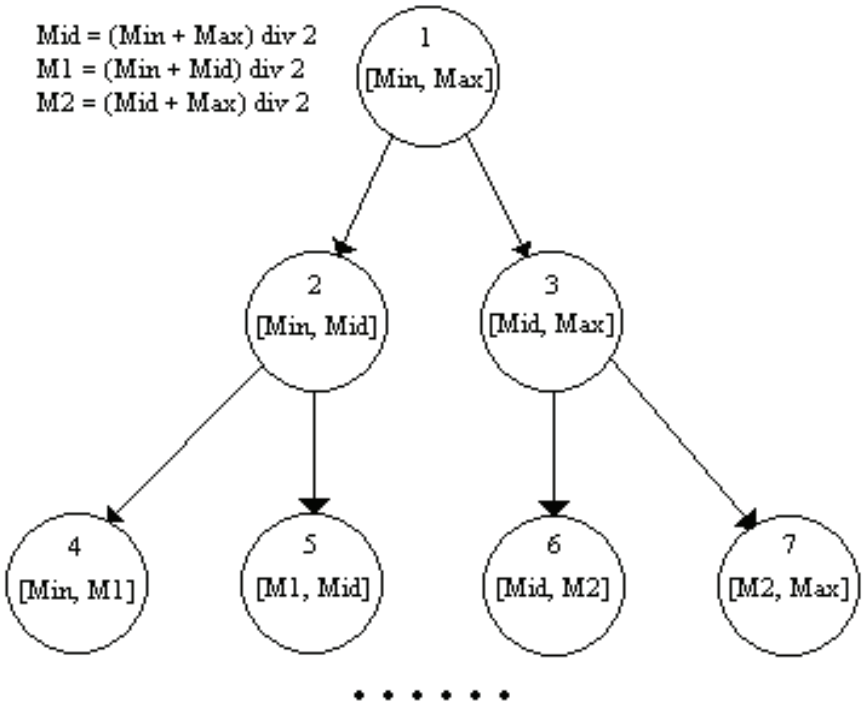
Với đề bài, ta sẽ có các hoành độ được gán nhãn (sau khi đã sắp xếp) như sau:

Interval Tree

```
H[1] = -2; Label[1].Low = 0; Label[1].High = 2; Label[1].Open = true;
H[2] = -1; Label[2].Low = 2; Label[2].High = 5; Label[2].Open = true;
...
```

Tiếp theo ta sẽ xây dựng một cây nhị cây nhị phân đầy đủ, mỗi đỉnh sẽ tượng trung cho số hình chữ nhật che phủ trên đoạn [A, B] thuộc tung độ, mỗi đỉnh gồm có 2 phần Number và Cover với ý nghĩa:

1) Number: số lượng hình chữ nhật che phủ toàn bộ [A, B]
2) Cover: tổng số đoạn tung độ bị che phủ trên đoạn [A, B]



Đỉnh 1 của cây có đoạn [A, B] tương ứng là [Min, Max] (với Min, Max tương ứng là *tung độ nhỏ nhất và lớn nhất* trong số các hình chữ nhật); đỉnh 2 có đoạn tương ứng [A, B] là [1, Max div 2]; đỉnh 3 là [Max div 2, Max],… tạo thành một cây nhị phân đầy đủ. Đơn giản hơn, với mỗi đỉnh chứa thông tin về đoạn [A, B], hai nút con tương ứng của nó sẽ lưu trữ thông tin về đoạn ***[A, (A+B) div 2]*** và ***[(A+B) div 2, B]***. Sở dĩ đỉnh con thứ hai không xét từ [(A+B) div 2 + 1, B] bởi lẽ chúng ta đang xét theo *tung độ* chứ không phải là xét theo *độ dài đoạn thẳng*, nếu xét như vậy ta sẽ không bị sót đoạn [(A+B) div 2, (A+B) div 2 + 1].

Quá trình xây dựng cây nhị phân sẽ được đồng nhất với quá trình tính diện tích, có nghĩa là với mỗi khe H_1H_2, H_2H_3, …, $H_{2*N-1}H_{2*N}$ chúng ta sẽ xây dựng cây nhị phân tương ứng với hình chữ nhật đang xét. Lưu ý rằng "hình chữ nhật đang xét" ở đây nói về hình chữ nhật liên quan đến các hoành độ H sau khi đã sắp xếp, bởi như đã nói ở trên, H được gán nhãn để đại diện cho một đỉnh của hình chữ nhật.

Giả sử rằng chúng ta đang xét đến khe thứ i và i-1, đỉnh k của cây mang đoạn [A, B], có thể xảy ra 2 trường hợp sau:

1. Đỉnh H[i] đang xét là đỉnh trái trên của hình chữ nhật:

   Có 3 khả năng xảy ra:

   - Nếu hình chữ nhật che phủ đoạn [A, B], tức là **(Low ≤ A) and (B ≤ High)**:

Khi đó số hình chữ nhật bị che phủ phải tăng lên 1 và tổng số che phủ chính bằng High – Low. Ta không cần phải xét đến các nút con của nó.

- Nếu hình chữ nhật nằm hoàn toàn ngoài đoạn [A, B], tức là **(High ≤ A) or (B ≤ Low)**: (vì chúng ta đang xét đến tung độ nên phải thêm "=" bởi nếu có "=" thì số phần bị che phủ cũng không tăng lên):

  Khi đó, chắc chắn hình chữ nhật cũng không che phủ lên các nút con của nó, ta không phải làm gì cả.

- Nếu hình chữ nhật giao với đoạn [A, B], tức là trường hợp còn lại:

  Ta xét đến các nút con của nút k. Nếu như số hình chữ nhật bao phủ lên k là 0 chứng tỏ tổng số che phủ của k bằng tổng số che phủ của các nút con của nó.

```
Procedure Open_True(A, B, k, Low, High);
Begin
   if A + 1 >= B then {Xét đến nút lá – trường hợp đặc biệt}
      begin
         if (Low <= A) and (B <= High) then Cover[k] := 1
         else Cover[k] := 0;
         exit;
      end;
    if (Low <= A) and (B <= High) then {Che phủ [A, B]}
      begin
         Number[k] := Number[k] + 1;
         Cover[k] := B – A;
         exit;
      end;
    if (High <= A) or (B <= Low) then exit; {Nằm ngoài}
    {Giao nhau}
    Open_True(A, (A+B) div 2, 2*k, Low, High); {Xét nút con trái}
    Open_True((A+B) div 2, A, 2*k+1, Low, High); {Xét nút con phải}
    if Number[k] = 0 then Cover[k] := Cover[2*k] + Cover[2*k+1];
End;
```

2. Đỉnh H[i] đang xét là đỉnh phải dưới của hình chữ nhật:

   Có 3 khả năng xảy ra:

   - Nếu hình chữ nhật che phủ đoạn [A, B], tức là **(Low ≤ A) and (B ≤ High)**:

     Khi đó số hình chữ nhật bị che phủ phải giảm lên 1. Nếu như không còn hình chữ nhật nào che phủ nó thì tổng số che phủ bằng tổng số che phủ của các nút con của nó. Ta không cần phải xét đến các nút con.

   - Nếu hình chữ nhật nằm hoàn toàn ngoài đoạn [A, B], tức là **(High ≤ A) or (B ≤ Low)**: (vì chúng ta đang xét đến tung độ nên phải thêm "=" bởi nếu có "=" thì số phần bị che phủ cũng không tăng lên):

     Khi đó, chắc chắn hình chữ nhật cũng không che phủ lên các nút con của k, ta không phải làm gì cả.

   - Nếu hình chữ nhật giao với đoạn [A, B], tức là trường hợp còn lại:

Ta xét đến các nút con của nút k. Nếu như số hình chữ nhật bao phủ lên k là 0 chứng tỏ tổng số che phủ của k bằng tổng số che phủ của các nút con của nó.

```
Procedure Open_False(A, B, k, Low, High);
Begin
   if A + 1 >= B then {Xét đến nút lá – trường hợp đặc biệt}
      begin
         if (Low <= A) and (B <= High) then
             begin
                 Number[k] := Number[k] - 1;
                 if Cover[k] = 0 then Cover[k] := Cover[2*k] + Cover[2*k+1];
             end;
         exit;
      end;
   if (Low <= A) and (B <= High) then {Che phủ [A, B]}
      begin
         Number[k] := Number[k] - 1;
         if Cover[k] = 0 then Cover[k] := Cover[2*k] + Cover[2*k+1];
         exit;
      end;
   if (High <= A) or (B <= Low) then exit; {Nằm ngoài}
   {Giao nhau}
   Open_False(A, (A+B) div 2, 2*k, Low, High); {Xét nút con trái}
   Open_False((A+B) div 2, A, 2*k+1, Low, High); {Xét nút con phải}
   if Number[k] = 0 then Cover[k] := Cover[2*k] + Cover[2*k+1];
End;
```

Để tiện cho việc lưu trữ, Interval Tree của chúng ta sẽ được lưu trữ bằng mảng, đỉnh i có 2 con là 2*i và 2*i+1.

Phần khó nhất là việc xây dựng cây đã xong, chương trình xử lý của chúng ta sẽ bao gồm sắp xếp theo H và xét các khe H_1H_2, H_2H_3, …, $H_{2*N-1}H_{2*N}$:

```
Procedure Solve;
begin
   Sort; {Sắp xếp dãy}
   Open_True(Min, Max, 1, Label[1].Low, Label[1].High);
   Area := 0;
   for i := 2 to 2*n do
      begin
         Area := Area + Cover[1] * (H[i] – H[i-1]); {Tính diện tích}
         if Label[i].Open then
            Open_True(Min, Max, 1, Label[i].Low, Label[i].High)
         else
            Open_False(Min, Max, 1, Label[i].Low, Label[i].High);
      end;
end;
```

Chú ý:

- Một số bạn sẽ cho rằng nếu như tại một hoành độ có nhiều đỉnh đóng hoặc nhiều đỉnh mở thì sao? Câu trả lời là không sao cả, bởi lẽ lúc đó H[i] – H[i-1] sẽ bằng 0 và diện tích sẽ không được tính vào Area. Thêm nữa, các đỉnh có chung hoành độ sẽ được mở hoặc đóng cho đến hết rồi mới được thực sự tính vào diện tích.
- Cần phân biệt Number là ***số hình chữ nhật phủ lên đoạn [A, B]*** trong một đỉnh của cây chứ không phải ***số hình chữ nhật được đoạn [A, B] phủ***. Sở dĩ chúng ta cần đến Number bởi nếu có một hình chữ nhật phủ kín đoạn [A, B]
- Nhiều bạn mới đọc sẽ thắc mắc: Chúng ta đang xét đến các khe, làm sao có thể xét được hết các hình chữ nhật có chung hoành độ được? Câu trả lời nằm ở chú ý thứ nhất: tất cả các hình chữ nhật có chung hoành độ, bất kể là đỉnh trên trái hay dưới phải đều được xét hết trước khi diện tích được cộng chính thức vào tổng.

Loại cây mà chúng ta vừa sử dụng được gọi là Interval Tree. Vậy, Interval Tree là gì? Có thể nói vắn tắt lại rằng Interval Tree thực chất là một loại cây nhị phân đầy đủ với mỗi đỉnh chứa các thông tin về một đoạn [A, B] xác định. 2 con của nó chứa thông tin về đoạn [A, (A+B) div 2] và [(A+B) div 2 + 1, B]. Tùy theo mục đích sử dụng Interval Tree mà chúng ta có các loại Interval Tree khác nhau.

Mấu chốt của bài toán là chúng ta nhận ra được tầm quan trọng của việc xác định đoạn mở và đoạn đóng trên một trục nhất định (ở bài toán trên chúng ta chọn trục tung). Theo cách giải thông thường, chúng ta phải lặp để xác định sự chồng chéo nhau của các đoạn nguyên, độ phức tạp của thuật giải thông thường là rất cao. Nhưng giờ đây với Interval Tree, độ phức tạp chỉ là $O(\log_2 N)$, với trường hợp xấu nhất.

Hướng tối ưu cho bài toán: Nếu xét riêng bài toán tính diện tích này, chúng ta sẽ những hướng tối ưu sau (các bạn tự nâng cấp chương trình)

- Thay vì tìm Max, Min của tung độ, ta cho một mảng lưu lại giá trị của các tung độ có thể trong đề bài, sắp xếp lại chúng. Sau này, lời gọi các thủ tục Open_True và Open_False sẽ có các tham số chiếu đến *phần tử của mảng* này (vẫn bằng cách chia nhị phân) chứ không tham chiếu trực tiếp đến *giá trị tung độ thực sự* nữa.
- Có thể gộp Open_True và Open_False lại thành một thủ tục duy nhất, giảm thiểu chi phí khi cài đặt chương trình.