thị815

Giao Cắt Hai Phân Hoạch

Seminar Tin

KSTN Toán - Tin K60:
Nguyễn Anh Tú
Phạm Anh Tuấn
Trận Trọng Cường

Viện Toán Ứng và Tin Học

Đại Học Bách Khoa Hà Nội

Ngày 2 Tháng 12 Năm 2018

Mục lục

Lời Nói Đầu

Trên thực tế, để cho dễ sử dụng, thì các bản đồ thường được chia thành nhiều lớp, mà trong đó mỗi lớp lưu một thông tin nhất định. Ví dụ như trên bản đồ của một quốc gia, người ta thường hay lưu thành nhiều bản đồ con, một bản đồ lưu thông tin đường giao thông, một bàn đồ lưu thông tin vị trị của các thành phổ, tỉnh thành, một bản đồ lưu thông tin về sông, hồ, v.v... Bên cạnh thông tin về vị trí địa lý thì một bản đồ cũng có thể nhưng thông tin khác, như là vị trị đống dân cư, vị trí có động vật nguy hiểm, v.v...

Và khi sử dụng bản đồ, chúng ta thường mong muốn sẽ tìm được cho mình đường đi đơn giản, nhanh và an toàn nhất. Điều này đòi hỏi ta phải kết hợp được nhưng thông tin từ những bản đồ con khác nhau. Chính vì vậy một bài toán được đặt ra là khi ta có nhiều bản đồ con được đặt chồng lên nhau thì làm thế nào ta có thể biết được đâu là vùng ta cần tìm kiếm.

Trong bài báo cáo này, nhóm chúng em xin được trình bày cách thức giải quyết bài toán này. Bài báo cáo sẽ trình bày chi tiết về cấu trúc dữ liệu để biểu diễn một bản đồ (hay còn gọi là một phân hoạch), tư tưởng, cách xây dựng và độ phức tạp của thuật toán giải quyết vấn đề chồng chất hai phân hoạch

1 Giao Điểm Đoạn Thẳng Trong Mặt Phẳng

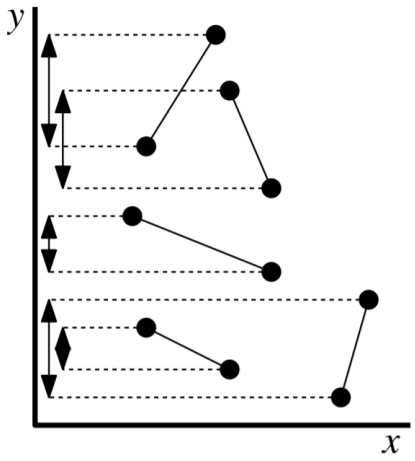
1.1 Ý Tưởng Thuật Toán

Trong mục này, chúng ta sẽ đi giải quyết bài toán tìm giao điểm của các đoạn thẳng trong mặt phẳng được phát biểu cụ thể như sau: cho một tập S gồm n đoạn thẳng trên một mặt phẳng, liệt kê tất cả các giao điểm giữa các đoạn thẳng đó.

Để giải quyết được vấn đề này không khó, vì ta có thể đơn giản là kiêm tra qua tất cả các cặp cạnh trong mặt phẳng xem chúng nó giao nhau không. Giải thuật brute-force này chạy với độ phức tạp thời gian là $O(n^2)$. Tuy nhiên trong thực tế, hầu hết các cạnh không giao hoặc giao với một số ít các cạnh còn lại, vì thế số giao điểm có trên mặt phẳng nhỏ hơn nhiều so với bình phương số cạnh. Và vì vậy, ta mong muốn có một thuật toán chạy tốt hơn trong trường hợp này. Hay nói cách khác, một thuật toán mà tốc độ thời gian chạy không chỉ phụ thuộc vào số đoạn thẳng có trong mặt phẳng mà còn phụ thuộc số giao điểm giữa chúng.

Ta có nhận xét rằng, những cạnh ở gần nhau mới có thể giao nhau. Bây giờ, ta sẽ sự dủng quán sát này để xây dựng thuật toán toàn giao điểm. Gọi $S = \{s_1, s_2, ..., s_n\}$ là tập hợp các đoạn thẳng. Ta không muốn đi kiểm tra những đoạn thẳng nằm cách xa nhau trong mặt phẳng. Để làm được điều này, ta gọi $y - interval$ của một đoạn thẳng là hình chiếu của đoạn thẳng đó lên trục Oy. Ta có thể dễ dàng chứng minh rằng, nếu $y - interval$ của 2 đoạn thẳng không trùng nhau thì chúng sẽ không giao nhau. Vì thế ta chỉ cần phải đi kiểm tra sự giao nhau giữa 2 đoạn thẳng có $y - interval$ trùng lên nhau.

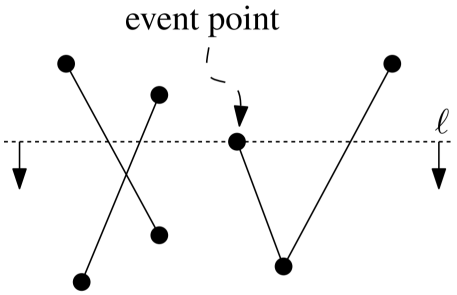
Để làm được điều này ta sử dụng một đường thẳng quét (sweep line) l đi từ trên xuống, xuất phát từ cạnh nằm trên cùng trong mặt phẳng. Khi đường thẳng quét từ trên xuống, ta chú ý đến tất các đoạn thẳng được giao với đường thẳng quét l (ta còn gọi giải thuật này là sweep line algorithm). Ta gọi trạng thái của đường sweep line l là tập hợp các đoạn thẳng giao với nó, trạng thái của đường thẳng quét sẽ được thay đổi trong quá trình l được quét từ



Hình 1: $y - interval$ các đoạn thẳng
Nguồn [1]

trên xuống. Trạng thái của đường thẳng quét chỉ cần phải cập nhật khi đường thẳng đi qua 1 số điểm đặc biệt. Ta gọi những điểm này là điểm sự kiện (event point). Trong giải thuật này, event points là các đầu mút của các đoạn thẳng.

Khi l gặp 1 event point, nó phải cập nhập trạng thái (status) và thực hiện các kiểm tra giao điểm. Cụ thể là nếu event point là điểm đầu của đoạn thẳng, thì đoạn thẳng đó phải được thêm vào trong status, và sau đó đoạn thẳng được kiểm tra xem có giao điểm với các đoạn thẳng có trong status hay không. Nếu event point là điểm cuối của đoạn thẳng thì đoạn thẳng đước được bỏ ra khỏi status.



Hình 2: Sweep line
Nguồn [1]

Tuy nhiên, nếu chỉ làm như vậy, vẫn có trường hợp ta vẫn phải kiểm tra một số bình phương số cạnh lần giao điểm giữa hai cạnh. Để khắc phục điều này ta sẽ sắp xếp các đoạn thẳng theo thứ tự trái qua phải, và ta chỉ kiểm tra giao điểm giữa 2 đoạn thẳng khí chụng nằm cạnh nhau. Khi sweep line l tiếp tục đi xuống và vào một thời điểm nào đó thì thứ tự các cạnh trong status cần phải được thay đổi. Để giải quyết được vấn đề này, biến status phải được sắp xếp theo thứ tự từ trái sang phải. Và biến status không chỉ còn phải thay đổi khi gặp các đầu mút, mà biến status cũng phải được cập nhật thứ tự khi l đi qua giao điểm (intersection point).
Bồ đề 1.1 [1] Gọi s_i và s_j là hai đoan thẳng giao nhau tại một điểm trong p và không có đoạn thẳng thứ ba cũng đi qua p. Khi đó tồn tại một điểm sự kiện nằm trên p làm cho s_i và s_j kề nhau trong biến status.

Bồ đề trên bảo đảm tính đúng đắn của thuật toán ta vừa xây dựng: Khi sweep line gặp một event point, nó sẽ dừng lại để kiểm tra giao điểm hoặc đổi thứ tự các cạnh trong biến status. Các biến status gồm các đầu mút của các đoạn và giao điểm giữa chúng. Đầu mút của các đoạn ta hoàn toàn biết trước, còn các giao điểm ta phải tính toán trong quá trình sweep line l quét mặt phẳng.

Các bước xử lý khi gặp event point được miêu tả cụ thể như sau. Khi đường thẳng quét gặp đinh trên của một đoạn thẳng, đoạn thẳng đó sẽ được kiểm tra giao điểm với 2 hàng xóm nằm cạnh nó trong biến status. Chỉ nhưng giao điểm nằm dưới đường thẳng l mới cần được chú ý, vì những điểm nằm trên sweep line đã được xử lí xong. Khi đường thẳng quét gặp event point là giao điểm của 2 đoạn thẳng thì hai đoạn thẳng này đổi vị trí cho nhau trong event point, khi thay đổi vị trí, thuật toán tiếp tục kiểm tra giao điểm của 2 đoạn thẳng với cái hàng xóm mới. Tiếp theo, khi sweep line gặp event point là đầu mút dưới của một cạnh, ta xóa cạnh đó khỏi biến status. Khi đó 2 hàng xóm của cạnh được xóa trở thành hàng xóm của nhau và cần được kiểm tra xem chúng có giao điểm với nhau hay không.

1.2 Cấu Trúc Dữ Liệu và Thuật Toán

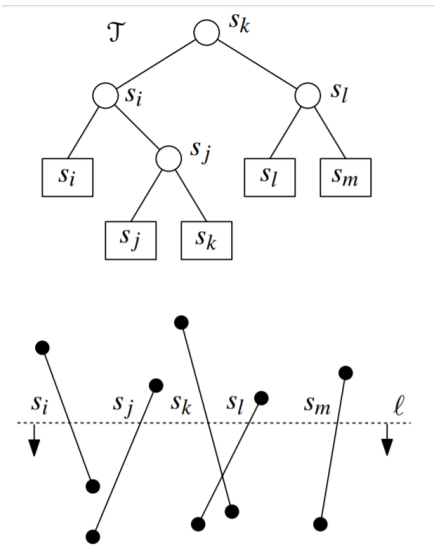
Đầu tiên ta cần xây dựng cấu trúc dữ liệu lưu các event point. Ở đây, ta sự dụng cấu trúc cây tự căn bằng để lưu các event point, và gọi cây này là Q. Ta cần thao tác xóa phần có tung độ lớn nhất trong cây event, trả về đỉnh đó và xử lí. Nếu hai event point có cùng tung độ thì ta sẽ lấy event có hoạnh độ thấp hơn. Cụ thể, ta thực hiện xây dựng cây event Q như sau: Ta định nghĩa phép so sánh $\prec$ giữa các event point. Với hai event point p và q, ta có $p \prec q$ khi và chỉ khi $p_y > q_y$ hoặc $p_y = q_y$ và $p_x < q_x$. Các event point được luu trong cây Q theo phép so sánh $\prec$. Với cấu trúc dữ liệu này, các thao tác thêm và xóa event đều mất $O(logm)$, trong đó m là số các event đang được lưu ở trong cây Q.

Thứ hai, ta cần xây dựng cấu trúc dữ liệu cho biến biến status. Cấu trúc dữ liệu cho status, kí hiêu là T, được sử dụng để tìm kiếm hàng xóm của một đoạn thẳng s, thêm vào đó ta cũng cần phải thêm và xóa các đoạn. Ở đây, ta tiếp tục sử dụng cây tự cân bằng để giải quyết vấn đề này. Cụ thể như sau, ta lưu các đoạn giao với sweep line là các lá của cây tự cân bằng T tho thứ tự từ trái sang phải. Tiếp theo, ta cần phải lưu thông tin các nút trong của cây để thuận tiện cho việc tìm kiếm. Tại mỗi đỉnh trong ta lưu cạnh nằm bên phải cùng của cây con trái đỉnh đó.

Giả sử ta cần tìm trên T cạnh nằm ngay bên trái của một điểm p nằm trên đường thẳng quét. Tại mỗi đỉnh trong của cây, chúng ta kiểm tra xem p nằm bên trái hay bên phải so với cạnh tương ứng với nút đó. Nếu cạnh đó nằm ở bên phải của điểm p, ta duyệt đến con trái của nút đó, và ngược lại đến khi ta duyệt đến một lá. Khi đó hoặc là nút ta thu được hoặc là lá nằm ngay bên trái so với nút đó tương ứng với đoạn thẳng mà ta cần tìm. Cuối cùng, ta có thể miêu tả thuật toán như sau.

Giải Thuật FindInterSections(S)
Đầu vào. Tập hợp S các đoạn thẳng trên mặt phẳng
Đầu ra. Tập hợp các giao điểm giữa các cạnh trong S

1. Khởi tạo cây event Q. Sau đó, lần lượt thêm các nút ứng với các đầu mút



Hình 3: Ví dụ cây tự cân bằng status
Nguồn [1]

của các đoạn thẳng vào cây. Các đỉnh phía trên của các đoạn lưu thêm thông tin nó là đầu mút của cạnh nào.

2. Khởi tạo cây status T
3. **while** Q khác rỗng
4. **do** Xác định event point p ơ trong Q và xóa nó khỏi cây
5. HandleEventPoint(p)

Trong phần trên, chúng ta đã miêu ta cách thuật toán xử lý khi gặp các event point. Trong trường hợp suy biến - các đoạn thẳng đồng quy - được xứ lý trong hàm HandleEventPoint(p) như sau.

HandleEventPoint(p)

1. Gọi $U(p)$ là tập hợp các đoạn thẳng có đầu mút trên là p (Đối với đoạn thẳng nằm song song với đường thẳng quét, đầu mút bên trái được coi là đầu mút bên trên)
2. Tìm tất các đoạn thẳng trong cây status T chưa p (Các đoạn thẳng này liền kề nhau trên đường thẳng quét). Gọi $L(p)$ là tập hợp các đoạn thẳng có đầu mút dưới p và $C(p)$ chứa các đoạn thẳng có điểm trong là p

3. **if** $U(p) \cup L(p) \cup C(p)$ chứa nhiều hơn 1 cạnh
4. **then** Lưu p là giao điểm
5. Xóa các cạnh $L(p) \cup C(p)$ khỏi T
6. Thêm các đoạn thẳng $U(p) \cup C(p)$ vào T. Thứ tự của các cạnh trên cây lúc này tuân theo thứ tự từ trái qua phải khi đường thẳng quét nằm ngay dưới p
7. **if** $U(p) \cup C(p) = \emptyset$
8. **then** FindNewEvent(s_l, s_r, p) trong đó s_l và s_r là cạnh nằm bên trái và bên phải của p
9. **else** Gọi s' là đỉnh nằm tận cùng bên trái của tập $U(p) \cup C(p)$ trong cây T
10. FindNewEvent(s_l, s', p) trong đó s_l là cạnh nằm bên trái s'
11. Gọi s'' là cạnh nằm tận cùng bên phải $U(p) \cup C(p)$ trong cây T
12. FindNewEVent(s'', s_r, p) trong đó s_r là cạnh nằm bên phải s''

Thủ tục FindNewEvent(s_l, s_r, p) được thực hiện như sau.
FindNewEvent(s_l, s_r, p)

1. **if** s_l và $_r$ cắt nhau tại một điểm nằm dưới đường thẳng quét hoặc thược đường quét và ở bên phải event point p, và giao điểm chưa xuất hiện trong cây event Q
2. **then** Thêm giao điểm đó vào cây event Q

1.3 Tính Đúng Đắn và Độ Phức Tạp Thuật Toán

Tính đúng đắn và độ phức tạp thuật toán được chứng minh trong [1] thông qua các bổ đề và định lý sau.
Bổ đề 1.2 Thuật toán FindIntersections liệt kê ra được tất cả các giáo điểm giữa các đoạn thẳng.

Bổ đề 1.3 Thuật toán FindIntersection chạy với độ phức tạp thời gian là $O(nlogn + Ilogn$ trong đó n là số cạnh trong mặt phẳng, và I là sao giao

điểm giữa các cạnh trong mặt phẳng.

Định lý 1 Cho S là tập hợp gồm n đoạn thẳng trong mặt phẳng. Tất cả các giao điểm, cùng với các cạnh đi qua chúng đều có thể tìm kiếm trong thời gian $O(nlogn + Ilogn)$ và chiếm $O(n)$ bộ nhớ, với I là số giao điểm có trong mặt phẳng.

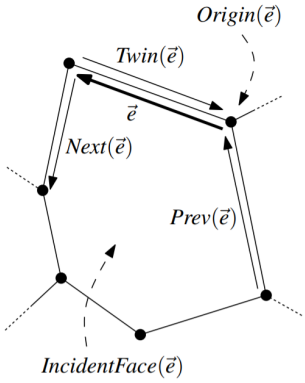
2 Cấu Trúc Nửa Cạnh

Chúng ta đã giải quyết được trường hợp đơn giản nhất của bài toán giao hai phân hoạch: giao của các đoạn thẳng trong mặt phẳng. Thực tế, các phân hoạch có cấu trúc phức tạp hơn: chúng là phân hoạch của một mặt phẳng thành các vùng được dán nhãn. Ví dụ: Một bản đồ chuyên về rừng ở Canada được phân cụm thành các vùng có nhãn "thông", "rụng lá", "bạch dương", "hỗn hợp".
Trước khi đưa ra giải thuật về việc tính toán giao của hai phân hoạch, ta phải xây dựng một biểu diễn phù hợp cho các phân hoạch. Lưu một phân hoạch như một tập hợp các đoạn thẳng không phải là một ý tưởng hay. Việc liệt kê biên của một vùng sẽ khá phức tạp. Nó sẽ tốt hơn nếu ta kết hợp các thông tin cấu trúc như: các đoạn thẳng nào là biên của vùng, vùng nào liền kề, v.v...

Trong bản báo cáo này, ta chỉ xét các phân hoạch phẳng được tạo ra bởi các đồ thị phẳng. Phân hoạch như vậy được gọi là liên thông nếu đồ thị tạo ra nó liên thông. Tương ứng mỗi nút của đồ thị là một đỉnh, mỗi cung được gọi là một cạnh. Ta chỉ xét các cạnh là đoạn thẳng, mặc dù về nguyên tắc các cạnh không cần phải thẳng. Một mặt của phân hoạch là một tập con lớn nhất (theo nghĩa bao hàm) của đồ thị và không bao gồm các đỉnh. Thế nên mặt là một miền đa giác mở mà biên của nó được giới hạn bởi các cạnh và các đỉnh của phân hoạch. Độ phức tạp của phân hoạch là tổng số đỉnh, số cạnh và số mặt của phân hoạch đó. Nếu một đỉnh là đầu mút của cạnh thì ta nói đỉnh và cạnh đó kề nhau. Tương tự, mặt và cạnh thuộc biên của nó kề nhau, mặt và đỉnh thuộc biên của nó kề nhau.

Một cấu trúc nửa cạnh lữu trữ các thông tin về mặt, cạnh và đỉnh của

phân hoạch. Bên cạnh thông tin hình học, ta có thể lưu trữ thêm các thông tin bổ sung. Để quét biên của một mặt phẳng theo thứ tự ngược chiều kim đồng hồ, ta chỉ cần lưu lại con trỏ mỗi cạnh tới cạnh tiếp theo. Đôi khi, ta cũng cần thiết phải đi theo đường biên theo hướng ngược lại, nên ta lưu lại con trỏ tới cạnh liền trước. Một cạnh thuộc hai mặt nên ta cũng cần hai con trỏ để lưu trữ nó. Nó rất thuận tiện để xét hai mặt khác nhau của một cạnh như là hai nửa cạnh. Điều đó có nghĩa từ nửa cạnh ta có thể biết được 1 mặt chứa nửa cạnh đó. Hai nửa cạnh của một cạnh ta gọi chúng là hai nửa cạnh đối nhau hay hai nửa cạnh song sinh. Để xác định hướng một nửa cạnh: theo ngược chiều kim đồng hồ, ta đi xung quanh biên của mặt của nửa cạnh sao cho tay trái ta hướng vào miền trong của mặt. Sau khi xác định hướng của một nửa cạnh, ta dễ dàng xác định điểm đầu và điểm cuối của nửa cạnh. Nếu nửa cạnh $\vec{e}$ có điểm đầu là v và điểm cuối là w thì nửa cạnh đối của $\vec{e}$ sẽ có điểm đầu là w và điểm cuối là v. Để biết biên của một mặt, ta cần lưu lại con trỏ trong thuộc tính mặt một nửa cạnh bất kỳ thuộc mặt đó. Xuất phát từ nửa cạnh đó ta có thể di chuyển tới nửa cạnh tiếp theo và quét hết biên của mặt chứa nửa cạnh đó.

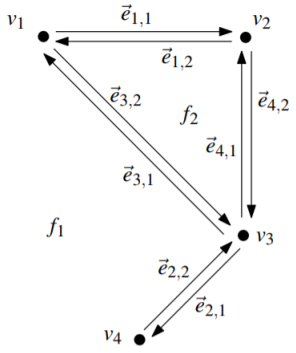


Hình 4: Cấu trúc nửa cạnh
Nguồn [1]

Tóm lại, Cấu trúc nửa cạnh bao gồm 3 thuộc tính: các đỉnh, các mặt, các nửa cạnh. Trong đó:

- Các đỉnh v lưu lại tọa độ của v trong một trường gọi là $Coordinates$ và con trỏ $IncidentEdge()$ của một nửa cạnh có v làm gốc.

- Các mặt f lưu con trỏ $OuterComponent$ cho một nửa cạnh thuộc biên của f. Đối với mặt ngoài (mặt không giới hạn), con trỏ này là **nil**. Các mặt cũng lưu danh sách $InnerComponent$ - chứa các con trỏ chỉ tới một nửa cạnh tương ứng với mỗi hố trong mặt đó.
- Các nửa cạnh $\vec{e}$ lưu con trỏ $Origin$ - gốc của nửa cạnh, con trỏ $Twin$ - nửa cạnh đối của $\vec{e}$ và con trỏ $IncidentFace$ - mặt kề với nửa cạnh $\vec{e}$. Chúng ta không cần lưu đỉnh cuối của $\vec{e}$ vì nó là $Origin(Twin(\vec{e}))$. Gốc của nửa cạnh được chọn sao cho $IncidentFace(\vec{e})$ nằm ở bên trái của $\vec{e}$ khi ta đi từ đỉnh gốc đến đỉnh cuối của $\vec{e}$. Bản ghi nửa cạnh cũng lưu con trỏ $Next(\vec{e})$ và $Prev(\vec{e})$ tương ứng với cạnh tiếp theo và cạnh trước thuộc biên của $IncidentFace(\vec{e})$. Như vậy, $Next(\vec{e})$ là nửa cạnh thuộc biên của $IncidentFace(\vec{e})$ mà có đỉnh cuối của $\vec{e}$ là gốc và $Prev(\vec{e})$ là nửa cạnh thuộc biên của $IncidentFace(\vec{e})$ mà có $Origin(\vec{e})$ là đỉnh cuối.



Vertex	Coordinates	IncidentEdge
v_1	(0,4)	$\vec{e}_{1,1}$
v_2	(2,4)	$\vec{e}_{4,2}$
v_3	(2,2)	$\vec{e}_{2,1}$
v_4	(1,1)	$\vec{e}_{2,2}$

Face	OuterComponent	InnerComponents
f_1	**nil**	$\vec{e}_{1,1}$
f_2	$\vec{e}_{4,1}$	**nil**

Half-edge	Origin	Twin	IncidentFace	Next	Prev
$\vec{e}_{1,1}$	v_1	$\vec{e}_{1,2}$	f_1	$\vec{e}_{4,2}$	$\vec{e}_{3,1}$
$\vec{e}_{1,2}$	v_2	$\vec{e}_{1,1}$	f_2	$\vec{e}_{3,2}$	$\vec{e}_{4,1}$
$\vec{e}_{2,1}$	v_3	$\vec{e}_{2,2}$	f_1	$\vec{e}_{2,2}$	$\vec{e}_{4,2}$
$\vec{e}_{2,2}$	v_4	$\vec{e}_{2,1}$	f_1	$\vec{e}_{3,1}$	$\vec{e}_{2,1}$
$\vec{e}_{3,1}$	v_3	$\vec{e}_{3,2}$	f_1	$\vec{e}_{1,1}$	$\vec{e}_{2,2}$
$\vec{e}_{3,2}$	v_1	$\vec{e}_{3,1}$	f_2	$\vec{e}_{4,1}$	$\vec{e}_{1,2}$
$\vec{e}_{4,1}$	v_3	$\vec{e}_{4,2}$	f_2	$\vec{e}_{1,2}$	$\vec{e}_{3,2}$
$\vec{e}_{4,2}$	v_2	$\vec{e}_{4,1}$	f_1	$\vec{e}_{2,1}$	$\vec{e}_{1,1}$

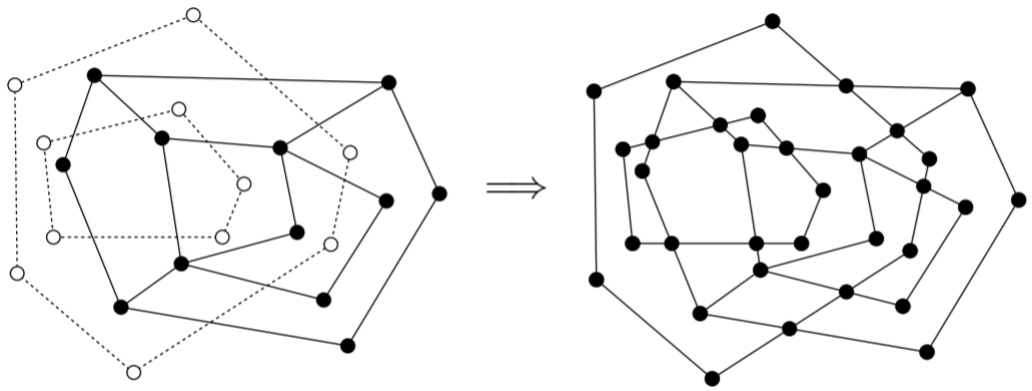
Hình 5: Ví dụ về cấu trúc nửa cạnh
Nguồn [1]

3 Giao Phân Hoạch

3.1 Phát Biểu Bài Toán

Bây giờ, ta đã có cấu trúc dữ liệu nửa cạnh là một phương pháp tốt để biểu diện một phân hoạch. Bài toán đặt ra ở đây là nếu có hai phân hoạch được đặt chồng lên nhau thì khi đó các mặt mới được tạo ra và việc cập nhật lại thông tin ở trên dữ liệu nửa cạnh sẽ như thế nào.

Cụ thể, ta kí hiệu giao của phần hoạch S_1 và phân hoạch S_2 là phân hoạch $O(S_1, S_2)$ và f được gọi là một mặt trong phân hoạch mới nếu tồn tại f_1 thuộc S_1 và $f_2$thuộc S_2 sao cho f là tập con liên thông cực đại (theo nghĩa bao hàm) của $f_1 \cap f_2$. Mô tả như hình vẽ.



Hình 6: Giao hai phân hoạch
Nguồn [1]

Bài toán đặt ra ở đây là chúng ta cần phải xây dựng lại cấu trúc nửa cạnh cho phân hoạch mới được tạo ra từ hai phân hoạch.

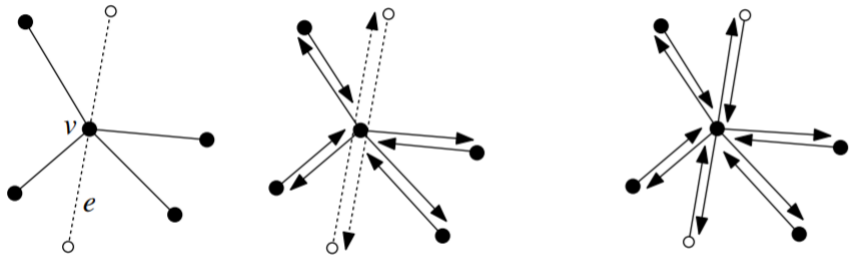
3.2 Xây Dựng Thuật Toán

Chúng ta thấy rằng các cạnh của phân hoạch S_1 giao cắt với các cạnh của phân hoạch S_2 tạo thêm các đỉnh, canh và mặt mới. Tuy nhiên ta không cần phải cập nhật thông tin của tất cả các cạnh mà chỉ cần quan tâm đến giao điểm giữa cạnh của phân hoạch này với cạnh của phân hoạch kia. Và các thông tin về mặt của phân hoạch mới hoàn toàn có thể xây dựng thông qua

dữ liệu về các nửa cạnh mới.

Như vậy việc đầu tiên ta cần làm là tìm giao điểm của các cạnh giữa hai phân hoạch, cập nhập thêm các đỉnh và nửa cạnh mới. Để làm được điều này, ta tận dụng giải thuật tìm giao điểm giữa các cạnh mà ta đã biết. Đầu tiên, ta ghép danh sách nửa cạnh D_1 của phân hoạch S_1 và danh sách nửa cạnh D_2 của phân hoạch S_2 thành danh sách nửa cạnh mới D. Sau đó chạy giải thuật tìm giao điểm trên danh sách nửa cạnh D. Ở đây, có khác so với thuật toán giao cắt các đoạn thẳng miêu tả ở phần trước đó là trong quá trình quét mặt phẳng, ta sẽ đồng thời xây dựng cấu trúc nửa cảnh.
Trước tiên, ta miêu tả giải thuật quét mặt phẳng để xây dựng nên các nửa cạnh, còn thông tin về mặt ta sẽ xử lý sau. Ta chạy giải thuật quét mặt phẳng tìm giao điểm như đã miêu tả ở phần trên cho tập cạnh D. Ngoài việc, xử lý hai cấu trúc cây event và cây status Q và T, bây giờ ta lưu thêm danh sách của nửa cạnh D. Khi khởi tạo, danh sách này gồm bản sao danh sách nửa cạnh của hai phân hoạch S_1 và S_2. Trong quá trình quét mặt phẳng, ta sẽ dần dần cập nhập danh sách nửa cạnh trở thành danh sách nửa cạnh đúng.

Khi đường thẳng quét gặp một event point, đầu tiên, ta sẽ cập nhập cây event Q và cây status T. Nếu điểm sự kiện đó chỉ kề với những cạnh thuộc cùng một phân hoạch, thì khi đó, đỉnh này là đỉnh có thể sự dụng lại được và ta không cần phải làm gì thêm. Nếu đỉnh kề với các cạnh thuộc cả hai phân hoạch, khi đó ta phải cập nhập thông tin cho D. Để làm được điều này, ta đi xét 3 trường hợp.



Hình 7: Xác định con trỏ next, prev
Nguồn [1]

Trường hợp thứ nhất là trong quá trình quét ta thấy cạnh e của S_1 đi qua đỉnh v của S_2. Cạnh e khi đó cần phải được thay thế bằng hai cạnh e' và e''. Trong dánh sách nửa cạnh D, hai cạnh mới thêm này tương ứng với 4 nửa cạnh. Đầu tiên, ta tạo mới 2 nửa cạnh đều có gốc là v, hai nửa cạnh ứng với cạnh e được giữ nguyên gốc là hai đầu mút của cạnh e. sau đó, ta gán 2 nửa cạnh này là twin với 2 nửa cạnh mới được tạo. Khi đó e' và e'' được biểu diện 2 nửa cạnh, một mới được tạo, một được lấy từ nửa cạnh của e. Bây giờ, ta chỉ cần tìm các con trỏ next và prev cho 4 nửa cạnh này. Ta xử lý các trường hợp dính tới 2 đầu mút của cạnh e được miêu tả như hình vẽ.

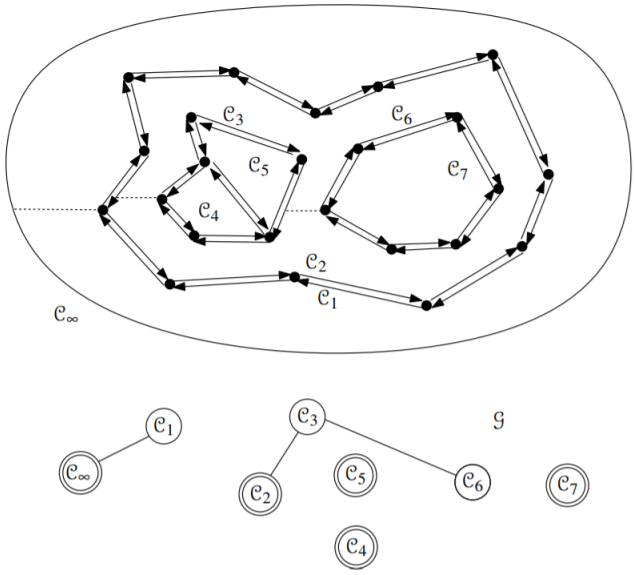
Đối với các trường hợp liên quan tới đỉnh v, ta cần phải gán các con trỏ next và prev cho 4 nửa cạnh biểu diễn cho e' và e'' từ phân hoạch S_2. Để làm được điều này ta xác định thứ tự của e' và e'' nằm ở đâu khi liệt kê theo chiều kim đồng hồ các cạnh kề với đỉnh v. Trường hợp các cạnh của hai phân hoạch cắt không tại đầu mút thì có thể được xứ lý bằng cách, khi tìm được một giao điểm, ta cho giao điểm đó vào trong phân hoạch S_2 và xử lý như trường hợp cạnh của S_1 đi qua đỉnh của S_2, trường hợp đỉnh của hai phân hoạch trùng nhau, tương đương với việc giải quyết việc cập nhật lại con trỏ Next(), Prev() của các nửa cạnh có đầu mút là đỉnh chung đó, thủ tục này là một phần của thủ tục xứ lý trương hợp đầu tiên. Ta có nhận xét rằng tất cả giao điểm thuật toán quét mặt phẳng tìm được đều là giao điểm giữa 2 phần hoạch. Vi vậy, danh sách các đỉnh và danh sách các nửa cạnh có thể được tìm ra trong độ phức tạp thời gian là $O(nlogn + klogn)$, trong đó n là tổng số đỉnh của 2 phân hoạch, và k là độ phức tạp của phần hoạch.

Tiếp theo, khi ta đã có được dữ liệu đúng cho các nửa cạnh, ta cần phải xác định các mặt của phân hoạch mới. Tức là với mỗi mặt trong phân hoạch $O(S_1, S_2)$, ta phải chỉ ra đâu là nửa cạnh của đường biên ngoài OuterComponent(f) và danh sách các nửa cạnh ứng vớ mỗi đường biên của các hố InnerComponent(f). Thêm vào đó, mỗi cạnh đều có thêm con trỏ $IncidentFace$ chỉ tới mặt mà nó kề với. Cuối cùng, mỗi mặt cần được gán nhán xem nó thuộc mặt nào của 2 phân hoạch cũ.

Ta biết rằng, xuất phát từ một nửa cạnh, ta có thể tìm được đường biên

của mặt kề với cạnh đó. Nhưng để có thể xác định đường biên đó là đường biên trong hay đường biên ngoài, ta cần phải xác định đỉnh v nằm tận cùng bên trái của đường biên đó. Ta biết rằng một mặt bất kì đều nằm bên trái theo hướng đi của nửa cạnh. Vì vậy ta chỉ cần tính số đó góc của hai nửa cạnh có đầu mút là v, nếu số đo lớn hơn 180 ta biết đấy là đường biên hố, còn số đo nhỏ hơn 180 ta biết đấy là đường biên ngoài.

Để có thể xác định các đường biên bao mặt nào trong phân hoạch, ta xây dựng một đồ thị G. Đồ thị G được xây dựng như sau, mỗi đường biên ứng với một nút trong đồ thị, và có thêm 1 nút cho đường biên ngoài của mặt không giới hạn. Hai đỉnh trong đồ thị G được nối với nhau bằng một cạnh khi và chi khi một đỉnh ứng với đường biên trong và đỉnh còn lại ứng với đường biên chứa cạnh gần nhất bên trái với đỉnh ở tận cùng bên trái của đường biên trong. Minh họa như hình vẽ.



Hình 8: Đồ thị biểu diễn đường biên
Nguồn [1]

Bổ Đề 3.1[1] Mỗi một thành phân liên thông của đồ thị G ứng với duy nhất một tập các đường biên với mỗi mặt của phân hoạch. Dựa vào bổ đề trên, ta thấy rằng nếu ta có thể xây dựng được đồ thị G thì ta có thông tin về các mặt của phân hoạch bằng việc trong quá trình quét mặt phẳng ta để

ý đến cạnh gần nhất bên trái của mỗi đỉnh và quét hết các cạnh để thu được thông tin của đường biên.

Việc cuối cùng chúng ta còn phải xử lý đó là ta cần phải gán nhẫn cho các mặt mới được tạo ra. Cụ thể là ta cần tim xem mặt mới đã thuộc vào những mặt nào của hai phân hoạch ban đầu. Xét một đỉnh v bất kì thuộc mặt f, nếu đỉnh v là giao của 2 cạnh thuộc 2 phân hoạch khác nhau thì khi đó ta có thể kết luận được mặt nào thuộc S_1 và mặt nào thuộc S_2 chứa f bằng việc dựa vào con trỏ $IncidentFace$ của 2 cạnh cắt nhau tạo thành đỉnh v. Trong trường hợp v không là giao điểm mà là đỉnh thuộc S_1, khi đó chúng ta chỉ biết mặt thuộc S_1 chứa f. Để tìm xem mặt nào thuộc S_2 chứa f, ta cần phải tìm ra mặt nào của S_2 chứa đỉnh v. Ta xử lý điều này bằng cách duy trỳ thêm 2 cây T_1 và T_2 có cấu trúc như cây status T. Chỉ khác rằng cây T_1 sẽ chỉ quét trên phân hoạch S_1 còn cây T_2 sẽ chỉ quét trên phân hoạch S_2. Do ở đây, S_1 và S_2 đã là hai phân hoạch nên ta biết ra sẽ khồng còn giao điểm nằm trong hai phân hoạch, do đó khi sử dụng đường thẳng quét, ta chỉ quan tâm đến thứ tự của các cạnh trên đường thẳng quét. Sử dụng 2 cây mới này, trong quá trình quét mặt phẳng, khi gặp một đỉnh v mà ta mới chỉ biết nó thuộc mặt nào của S_1 khi đó ta sử dụng cây T_2 để biết cạnh nào thuộc S_2 nằm ngay bên trái và ngay bên phải v. Như vậy ta có thể biết được mặt nào thuộc S_2 chứa v. Ở đây ta xây dựng thêm đúng 2 cây, và mỗi bước truy vấn có độ phức tạp thời gian là $O(logn)$ nên độ phức tạp thời gian cho việc tính toán vẫn là $O(nlogn)$.

Cuối cùng, ta có thuật toán tìm giao cắt của hai phân hoạch được miêu tả như sau.

Giải Thuật MapOverlay(S_1,S_2)
Input: Hai phân hoạch S_1 và S_2 được lưu dưới dạng danh sách nửa cạnh.
Output: Hợp của S_1 và S_2 lưu trong danh sách nửa cạnh D.

1. Sao chép danh sách nửa cạnh của S_1 và S_2 tới danh sách nửa cạnh mới D.

2. Tìm giao của các cạnh của S_1 và S_2 với thuật toán đường quét. Tại mỗi điểm sự kiện, ta cần thực hiện thêm:

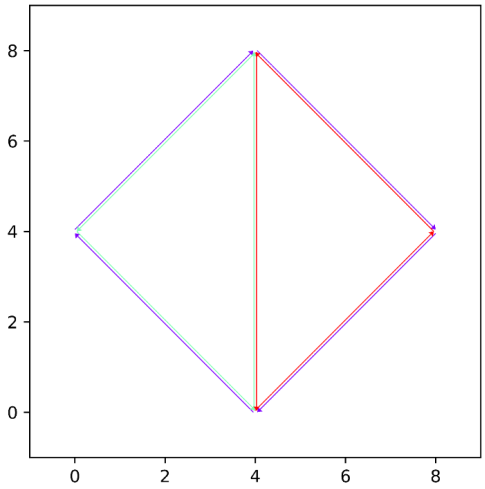
- Cập nhật D nếu điểm sự kiện liên quan đến các cạnh thuộc cả S_1 và S_2.
- Lưu lại nửa cạnh gần nhất phía bên trái của mỗi điểm sự kiện

3. Xác định các chu trình bao trong $O(S_1, S_2)$ bằng cách duyệt qua D.
4. Xây dựng đồ thị G, mỗi đỉnh của đồ thị ứng với một chu trình bao, các cạnh nối các đỉnh ứng với chu trình bao quanh lỗ và đỉnh ứng với chu trình gần nhất về phía bên trái của đỉnh trái cùng của lỗ đó, sau đó xác định các thành phần liên thông trong G.
5. Với mỗi thành phần liên thông trong G ta thực hiện: Với C là chu trình bao ngoài ở trong thành phần liên thông trên, và f là mặt giới hạn bởi chu trình đó. Đặt OuterComponent(f) bằng một nửa cạnh nào đó của C, và xây dựng danh sách InnerComponents(f) chứa con trỏ tới các nửa cạnh thuộc các lỗ trong mặt đó. Trỏ con trỏ IncidentFace của tất cả các nửa cạnh thuộc chu trình C tới f.
6. Gán tên các mặt của $O(S_1, S_2)$ theo các mặt của S_1 và S_2.

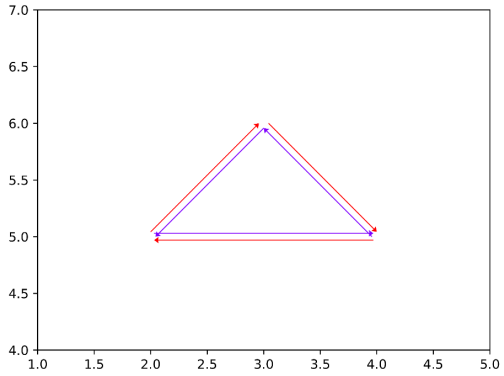
Định Lý 2[1]. Giả sử phân hoạch S_1 có n_1 cạnh, phân hoạch S_2 có n_2 cạnh, và đặt $n = n_1 + n_2$. Khi đó thuật toán xây dựng giao của hai phân hoạch miêu tả như trên chạy với độ phức tạp thời gian là $O(nlog(n)+klog(n)$ trong đó k là số cạnh của phân hoạch $O(S_1, S_2)$
Chứng minh. Gộp hai danh sách nửa cạnh chạy trong thời gian $O(n)$. Thuật toán quét mặt phẳng kèm với việc xây dừng đồ thị biểu diễn mặt và gán tên cho các mặt, như đã trình bày ở trên chạy với độ phực tạp thời gian $O(nlogn + klogn$. Vậy độ phức tạp cho thuật toán giao cắt 2 phân hoạch là $O(nlogn + klogn)$.

4 Kết Quả Thực Nghiệm

Trong phần này, bài báo cáo sẽ trình bày về kết quả thu được từ chương trình.



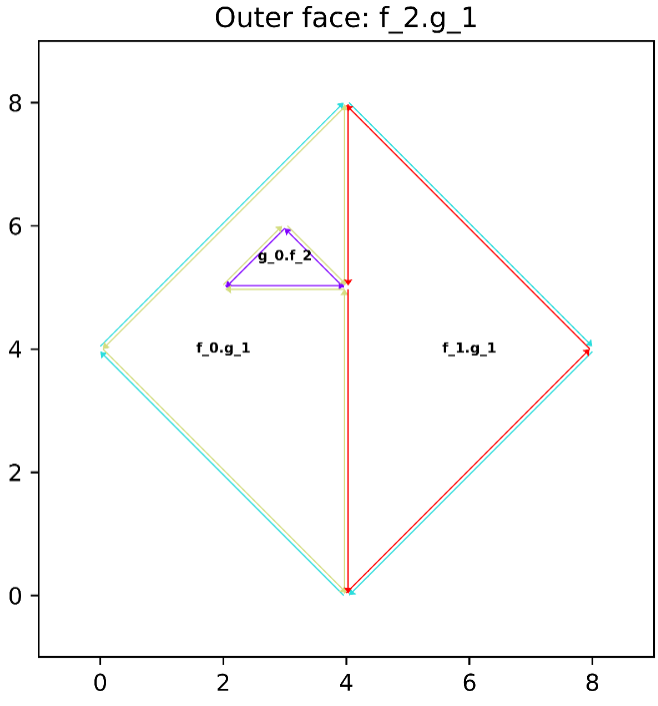
(a) Phân hoạch S_1



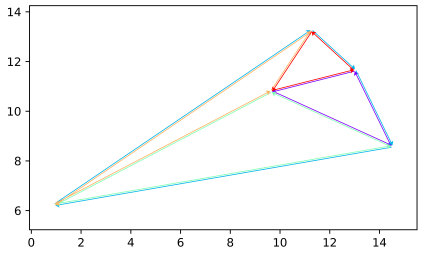
(b) Phân hoạch S_2

Hình 9: Input

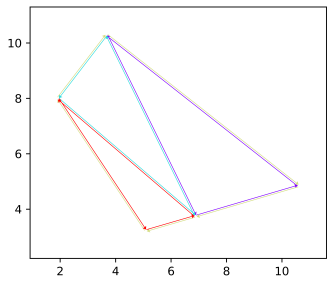
Thuật toán trả về cấu trúc nửa cạnh DCEL cho giao hai phân hoạch, và chỉ ra những mặt mới được tạo ra thuộc vào mặt nào của phân hoạch ban đầu



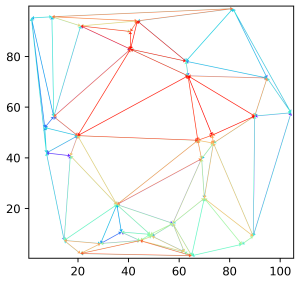
Hình 10: Output



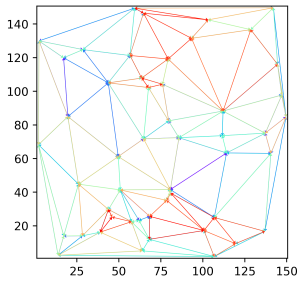
(a) S_1



(b) S_2



(c) S_1



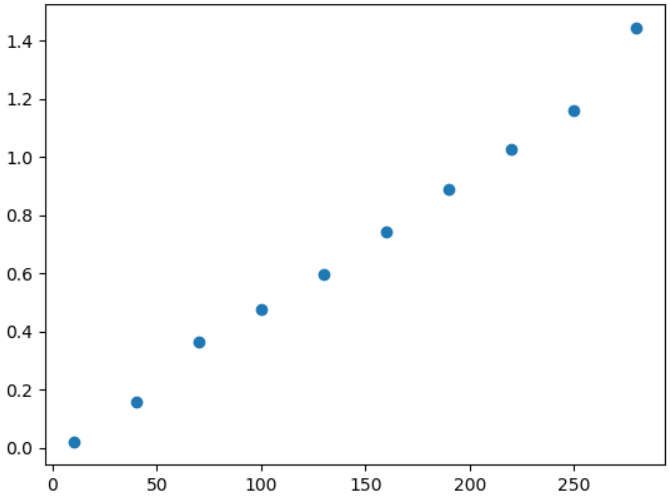
(d) S_2

Hình 11: Test cases

Để kiểm thử cho kết quả lý thuyết về độ phức tạp thời gian của thuật toán, một bộ test cases gồm 10 cấu trúc nửa cạnh. Kích cỡ của các phân hoạch tăng dần từ 10 đỉnh đến 280 đỉnh.

Kết quả về thời gian thu được.

Lý thuyết chứng minh rằng độ phức tạp của thuật toán là $O(nlogn+klogn)$ trong đó n là tổng số đỉnh của 2 phân hoạch ban đầu, và k là tổng số đỉnh, cạnh, và mặt của phân hoạch mới. Ta thấy rằng k có thể là một số bình phương của số đỉnh và trong trường hợp xấu nhất thì thuật toán chạy trong thời gian $O(n^2logn)$. Tuy nhiên, thực nghiệm cho thấy khi sinh ngẫu nhiên các phân hoạch thì số giao điểm, số mặt mới được tạo ra không quá lơn so với so với số đỉnh. Vì vậy, đồ thị giữa thời gian chạy của thuật toán và kích cỡ của phân hoạch là một đồ thị gần giống với $nlogn$. Do đó, thuật toán này trong phần lớn các trường hợp chạy tốt hơn thuật toán brute-force mà ta đã nhắc đến ban đầu.



Hình 12: Đồ thị thời gian chạy - kích thước của phân hoạch

5 Tổng Kết

Tài Liệu Tham Khảo

1. *Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmanrs*, Computational Geometry: Algorithms and Applications, *Third Edition*, Springer.

Phân Công Công Việc

Nguyễn Anh Tú

- Tìm hiểu chung về bài toán
- Thực hiện viết báo cáo về bài toán tìm giao cắt giữa các đường thẳng
- Thực hiện viết báo cáo về bài toán tìm giao giữa hai phân hoạch
- Giải quyết vấn đề đặt tên cho các mặt phẳng
- Xây dựng test cases, và thực viết báo cáo kết quả thực nghiệm

Phạm Anh Tuấn

- Tìm hiểu chung về bài toán
- Thực hiện viết chương trình giao cắt các đường thẳng
- Xây dựng cấu trúc DCEL riêng cho bài toán giao phân hoạch (kế thừa cấu trúc DCEL)
- Thực hiện viết chương trình tìm giao giữa hai phân hoạch
- Giải quyết vấn đề đặt tên cho các mặt phẳng

Trần Trọng Cường

- Tìm hiểu chung về bài toán
- Thực hiện viết báo cáo phần cấu trúc nửa cạnh DCEL
- Thực hiện xây dựng cấu trúc nửa cạnh DCEL
- Hộ trợ viết chương trình giao cắt các đường thẳng