**Bài toán Tam giác phân đa giác**

**(Polygon Triangulation)**

# Bài toán thực tế

Bảo tàng nghệ thuật trưng bày nhiều bức tranh quý dễ bị trộm bởi tội phạm. Do đó người ta cần lắp các camera để giám sát, bảo vệ bảo tàng khỏi những tên trộm. Giả sử rằng camera có khả năng quay quanh trục thẳng đứng để ghi hình. Bài toán đặt ra là cần trang bị bao nhiêu camera và lắp đặt chúng ở những vị trí nào để mọi vị trí trong bảo tàng đều có thể được giám sát bởi ít nhất một camera và đồng thời tiết kiệm được chi phí mua camera.

# Định nghĩa bài toán

Cho một đa giác đơn (không có hố) n đỉnh. Hỏi cần chọn ra bao nhiêu điểm và vị trí của chúng (để đặt camera) để mọi điểm trong đa giác đơn có thể được nhìn thấy bởi ít nhất một điểm trong tập điểm đã chọn.

# Một số nhận xét về bài toán

Dễ thấy 2 đa giác có cùng số đỉnh nhưng đa giác nào phức tạp hơn thì cần nhiều camera hơn. Đa giác lồi chỉ cần đúng 1 camera. Ta sẽ đưa ra cận trên cho số camera cần thiết là hàm số theo n – số đỉnh của đa giác. Cận trên này sẽ đủ tốt cho bất kì đa giác đơn n đỉnh. ***Sẽ là tốt hơn nếu ta tìm được số camera tối thiểu cho bất kì đa giác nào cho trước thay vì chỉ tìm được cận trên trong trường hợp tồi nhất, nhưng bài toán tối ưu đó là NP khó !***

Vì đa giác có hình dạng phức tạp, khó đưa ra được số camera cần thiết nên ta phân rã đa giác thành các tam giác. Nếu ta đặt 1 camera vào vị trí bất kì trong tam giác thì chắc chắn tam giác đó được bảo vệ bởi camera này.

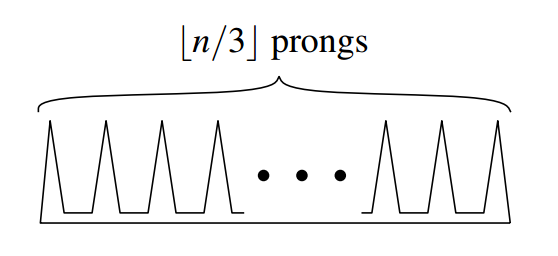
Gọi đường chéo (diagonal) là đoạn thẳng nối bất kì 2 đỉnh của đa giác và nằm bên trong đa giác. Phép tam giác phân (triangulation) là phép phân rã đa giác thành các tam giác bằng một tập đường chéo không giao nhau có lực lượng lớn nhất. Ta cần sự cực đại về lực lượng để tránh trường hợp một đỉnh của đa giác lại nằm trên cạnh của một tam giác (khi đa giác có 3 đỉnh thẳng hàng). Một đa giác có thể được phân thành tam giác theo nhiều cách khác nhau.

Mỗi đa giác đơn luôn tồn tại phép phân tam giác. Bất kì phép tam giác phân đa giác n đỉnh nào cũng gồm chính xác n-2 tam giác (n-3 đường chéo diagonal).

Với n-2 camera đặt ở n-2 tam giác thì ta chắc chắn bảo vệ được đa giác, nhưng cận trên này chưa sát. Nếu đặt camera ở các đường chéo thì mỗi camera có khả năng bảo vệ được 2 tam giác kề 2 bên, nên cận trên sẽ là khoảng n/2. Nhưng ta sẽ làm được tốt hơn nếu đặt camera ở đỉnh tam giác. Khi đó mỗi camera có thể bảo vệ được nhiều tam giác hơn.

Thuật toán sẽ tiếp cận theo cách này bằng cách tô màu mỗi đỉnh của các tam giác bởi 3 màu. Sau đó chọn ra màu có số đỉnh nhỏ nhất và ***đặt camera tại những đỉnh có màu này***. Khi đó số camera tối đa là n/3 (phần nguyên non). Đồ thị đối ngẫu của phép tam giác phân là cây. Ta chứng minh được là phép tô màu này luôn tồn tại trong đa giác có thể phân thành tam giác. Thuật toán đưa ra kết quả là cần n/3 camera. Cận này thực sự là sát vì ta chỉ ra được 1 trường hợp mà đa giác đó cần n/3 camera để bảo vệ. Vì thế trong các cách tiếp cận đưa ra cận trên cho số camera cần thiết thì kết quả này là tốt nhất.

Trường hợp cần n/3 camera cần thiết như hình sau

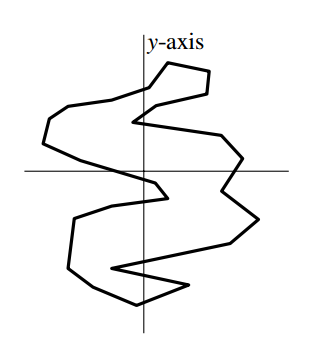


***Như vậy ta cần phân rã 1 đa giác đơn P thành các tam giác. Sau đó tô màu các đỉnh của các tam giác bởi 3 màu và chọn ra màu có số đỉnh ít nhất. Cuối cùng đặt camera tại các đỉnh có màu được chọn.***

Bài toán phân rã 1 đa giác đơn P n đỉnh thành các tam giác là bài toán khó, ta có thuật toán hiệu quả để áp dụng cho 1 số loại đa giác. Ví dụ với đa giác lồi (convex polygon) thì ta nối 1 đỉnh bất kì với mọi đỉnh còn lại (ngoại trừ 2 đỉnh hàng xóm) trong thời gian O(n) là ta đã phân tam giác xong. ***Vì ta khó có thể phân tam giác trực tiếp trên đa giác đơn P nên***

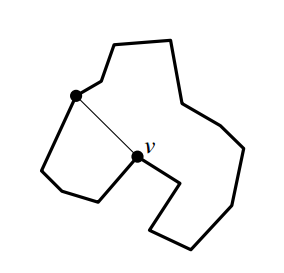
* **Cần phân rã P thành các đa giác monotone (đơn điệu)**
* **Sau đó phân mỗi đa giác monotone thành các tam giác.**

Định nghĩa đa giác monotone. Một đường cong C được gọi là monotone (đơn điệu) theo  nếu bất kì đường thẳng nào vuông góc với  cũng đều hoặc là giao với C tại 1 điểm, hoặc 1 đoạn thẳng, hoặc không giao. Một đa giác đơn P gọi là monotone theo  nếu biên của P có thể chia làm 2 đường cong mà mỗi đường cong là monotone theo . Ví dụ một đa giác monotone theo trục Oy (gọi là y-monotone). Ta nhận thấy nếu đi từ đỉnh trên cùng, dọc theo 1 đường cong (trái hoặc phải) monotone thì ta sẽ hoặc đi xuống, sang ngang, chứ không đi ngược lên.



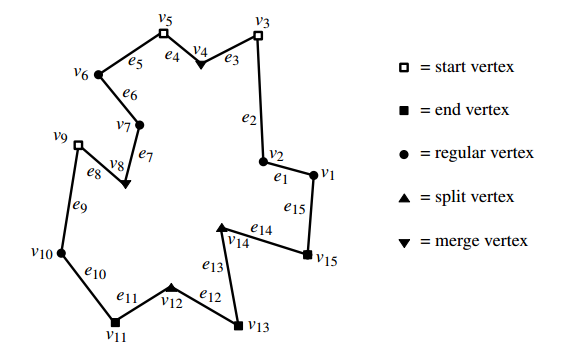
Xét bài toán phân đa giác đơn P thành các đa giác monotone. Ta nhận thấy sự vi phạm tính chất monotone (một cách cục bộ) trong đa giác P (nếu có) là bắt nguồn từ đỉnh turn. Đỉnh turn là đỉnh khi ta đi theo hướng từ 1 cạnh kề, đi qua đỉnh này sang cạnh tiếp theo thì hướng đi bị đổi hướng ngược lại (từ xuôi xuống chuyển thành ngược lên hoặc ngược lại). Chú ý là đỉnh turn v có đặc điểm là 2 đỉnh kề của đỉnh v sẽ hoặc là cùng nằm phía trên v hoặc là cùng nằm phía dưới v. Ta dùng định nghĩa nằm trên, hoặc nằm dưới một cách mở rộng. Đỉnh p nằm trên đỉnh q khi .

Để phân rã đa giác đơn P thành các đa giác monotone thì ta cần loại bỏ các đỉnh turn bằng cách thêm đường chéo (diagonal) vào đỉnh turn. Ví dụ xét đỉnh turn v có 2 đỉnh kề nằm dưới đỉnh v. Ta phải thêm đường chéo từ v đi lên để có thể khử được đỉnh turn trong 2 đa giác con thu được.



Dễ thấy trong 2 đa giác thu được, đỉnh v không còn là đỉnh turn. Tương tự với trường hợp đỉnh turn có 2 đỉnh kề nằm phía trên. Khi đó ta thêm đường chéo đi xuống dưới.

Tiếp theo, sẽ phân tích cụ thể các loại đỉnh turn. *Có 5 loại đỉnh*, trong đó *có 4 loại* (ngoại trừ regular) là *đỉnh turn* (*không tồn tại loại đỉnh khác*).



Tên mỗi loại đỉnh đều bắt nguồn từ ý nghĩa khi ta quét đường sweep line từ trên xuống.

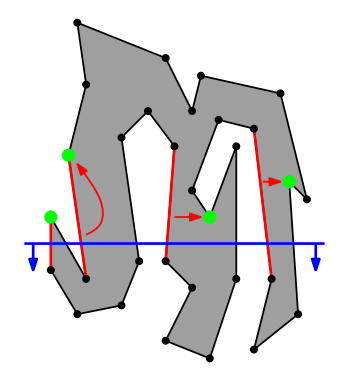
* Start vertex: Đỉnh v là đỉnh start nếu 2 đỉnh hàng xóm kề v nằm dưới v và góc trong tại v < 180 độ. Khi sweep line quét qua thì đây là đỉnh gặp đầu tiên (trong lân cận đủ nhỏ).
* Split vertex: Đỉnh v là đỉnh split nếu 2 đỉnh hàng xóm kề v nằm dưới v và góc trong tại v > 180 độ. Khi sweep line quét qua đỉnh này thì từ 1 giao điểm tách thành 2.
* End vertex: Đỉnh v là đỉnh end nếu 2 đỉnh hàng xóm kề v nằm trên v và góc trong tại v < 180 độ. Khi sweep line quét qua thì đây là đỉnh gặp cuối cùng (trong lân cận đủ nhỏ).
* Merge vertex: Đỉnh v là đỉnh split nếu 2 đỉnh hàng xóm kề v nằm trên v và góc trong tại v > 180 độ. Khi sweep line quét qua đỉnh này thì 2 giao điểm sẽ hợp thành 1.
* Regular vertex: Đỉnh có 1 đỉnh hàng xóm nằm trên và 1 đỉnh hàng xóm nằm dưới.

***Nếu một đa giác đơn P không có đỉnh split và merge thì đa giác đó là y-monotone***. Như vậy trong quá trình quét sweep line, ta sẽ thêm đường chéo vào các đỉnh split và merge để những đa giác con thu được là y-monotone (không còn 2 loại đỉnh này).

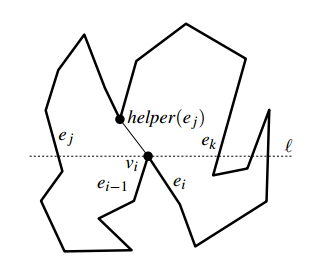
Event point là các đỉnh của đa giác. Trong quá trình quét sweep line thì không có event nào khác xuất hiện.

Xét cách thêm đường chéo vào đỉnh split. Ta phải kết nối đỉnh split v với 1 đỉnh nằm phía trên. Nhưng đó là đỉnh nào? Đỉnh cần tìm phải có xu hướng nằm gần đỉnh v vì khi đó, ta ***hi vọng đường chéo sẽ thỏa mãn được 2 điều kiện*** là: không có đường chéo nào cắt nhau (ngoài đỉnh) và không có đường chéo nào cắt cạnh của đa giác (ngoài đỉnh). Cụ thể hơn, gọi ej là cạnh gần nhất bên trái đỉnh v mà hiện đang giao với sweep line. Ta sẽ thêm đường chéo **nối v với helper(ej)**. ***helper(ej) là đỉnh thấp nhất nhưng nằm trên sweep line mà đoạn thẳng nằm ngang kẻ từ đỉnh đó nối với cạnh ej nằm trọn trong đa giác P. Chú ý helper(ej) có thể là upper (lower) endpoint của ej***.

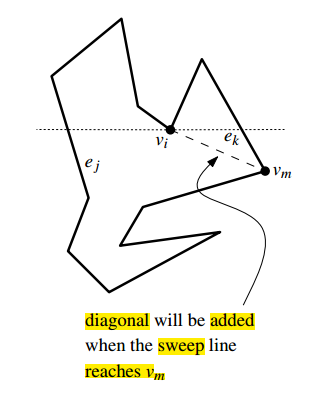
Ví dụ minh họa helper của các cạnh.



Ví dụ minh họa thêm đường chéo vào đỉnh split vi.



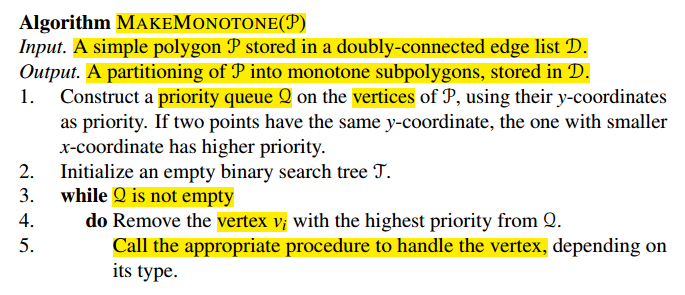
Xét cách thêm đường chéo vào đỉnh merge. Thao tác này có vẻ khó hơn so với khi thêm đường chéo vào split vertex bởi vì ta cần nối đỉnh merge với một đỉnh thấp hơn – đỉnh mà ta chưa gặp. Cách tìm đỉnh để thêm đường chéo như sau. Gọi ej là cạnh bên trái gần nhất của đỉnh merge vi. Lúc sweep line quét qua vi thì helper(ej) là vi. Khi sweep line di chuyển xuống, gặp đỉnh đầu tiên là vm thì helper(ej) là vm. Đây chính là đỉnh nối với vi tạo thành đường chéo cần tìm. Nếu ta không gặp đỉnh nào thì khi sweep line đạt đến đỉnh lower của ej ta sẽ thiết lập helper(ej) thành đỉnh lower của ej. Như vậy, khi helper của một cạnh được thay đổi thì nếu helper cũ là đỉnh merge thì **ta thêm đường chéo bằng cách nối helper cũ với helper mới**.



Trong cấu trúc cây nhị phân tìm kiếm cân bằng T lưu các cạnh hiện đang giao với sweep line thì ta chỉ lưu các cạnh mà phần bên phải của nó nằm trong đa giác bởi vì khi thêm đường chéo vào đỉnh split, merge ta chỉ xét cạnh bên trái gần nhất (chứ không xét cạnh bên phải gần nhất).

# Thuật toán phân đa giác thành các đa giác monotone

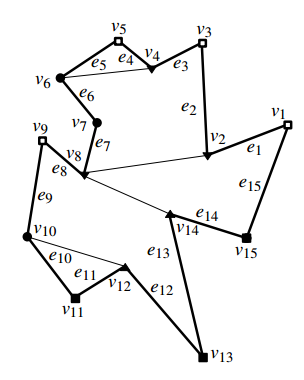
Sơ đồ chung của thuật toán phân đa giác thành các đa giác monotone như sau.



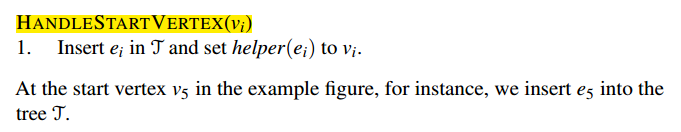
Có 2 thao tác chính cần thực hiện khi xử lý mỗi loại đỉnh là

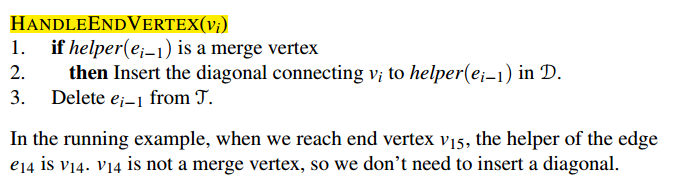
* Kiểm tra có cần phải thêm đường chéo không
* Cập nhật cấu trúc BBST T (cập nhật danh sách các cạnh hiện đang giao với sweep line hoặc cập nhật helper).

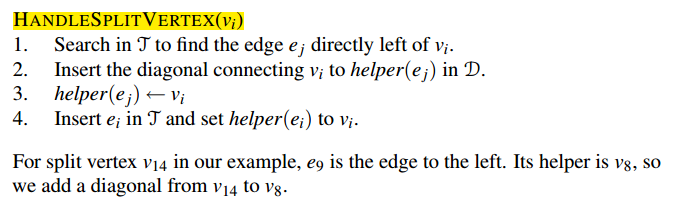
Hình sau minh họa ví dụ.

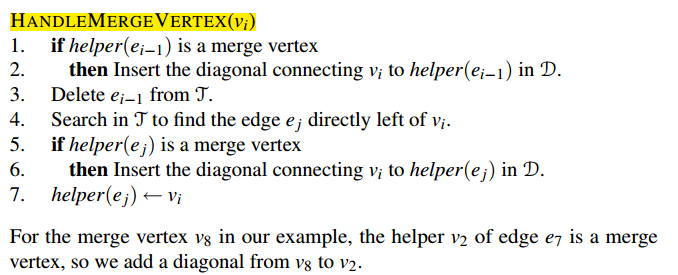


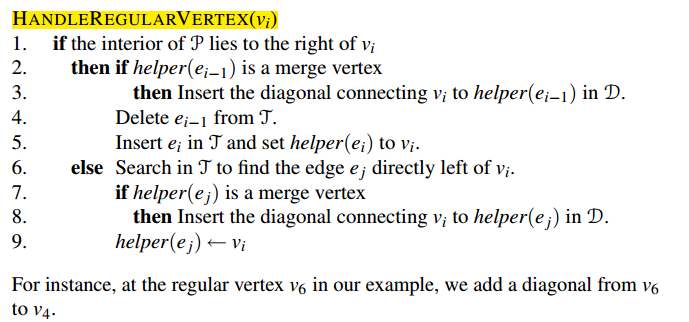
Tiếp theo sẽ trình bày các bước để xử lý 5 loại đỉnh và minh họa qua ví dụ trên.











**Thuật toán sử dụng 3 CTDL:**

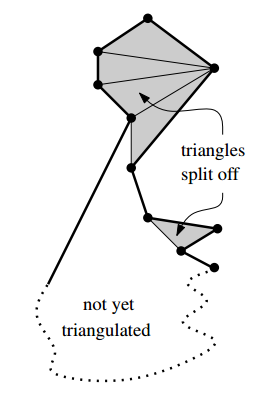
* Quá khứ: DCEL D biểu diễn đa giác P cùng với các đường chéo mà ta đã xác định bên trên sweep line.
* Hiện tại: BBST T lưu các cạnh hiện đang giao với sweep line mà có phần bên phải cạnh là nằm trong đa giác P.
* Tương lai: Priority queue Q lưu các event point là các đỉnh của đa giác P

Mỗi bước xử lý 1 event thì thuật toán thực hiện 1 thao tác trên Q, tối đa 1 truy vấn, 1 xóa, 1 chèn trên T, chèn tối đa 2 đường chéo vào D. Do đó thời gian xử lý 1 sự kiện là O(logn). **Vậy độ phức tạp của thuật toán phân đa giác P thành đa giác monotone là O(nlogn)**. **Bộ nhớ sử dụng là O(n)**.

# Thuật toán tam giác phân đa giác monotone

Đầu vào là một đa giác strictly y-monotone. Thuật toán sử dụng stack S để lưu những đỉnh đã thăm nhưng vẫn còn cần phải thêm đường chéo trong tương lai. Thuật toán sẽ xử lý từng đỉnh của đa giác, cố gắng thêm đường chéo từ đỉnh xử lý đến các đỉnh nằm trong stack S nhiều nhất có thể. Các tam giác được hình thành khi thêm đường chéo sẽ bị loại khỏi đa giác trong bước sau.

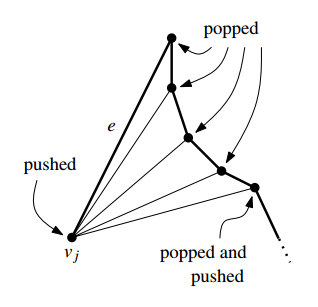
Sau mỗi lần xử lý 1 đỉnh thì thuật toán luôn duy trì hình dạng của phần đa giác P chưa được phân tam giác và nằm phía trên đỉnh vừa thăm luôn có dạng gồm 2 biên, trong đó 1 biên là một phần của 1 cạnh trong P, biên kia là chuỗi các đỉnh **reflex** (các đỉnh có góc trong ≥ 180 độ). Stack S sẽ lưu các đỉnh reflex này.



Khi xử lý đỉnh vj ta phân biệt 2 trường hợp sau

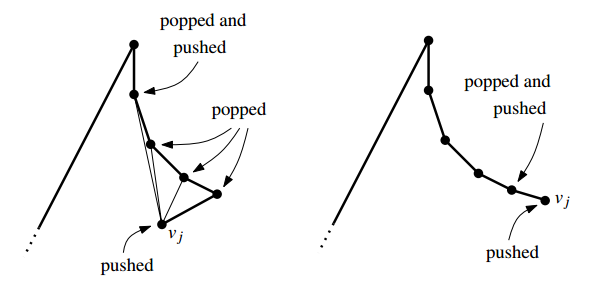
* Vj nằm trên biên khác với chuỗi reflex

Thuật toán thực hiện thêm đường chéo từ vj tới mọi đỉnh nằm trong stack (ngoại trừ đỉnh nằm ở đáy stack). Pop mọi đỉnh trong stack, push đỉnh mà nằm trên đỉnh stack (trước khi xử lý vj) và push vj. Lúc này stack chứa 2 đỉnh.

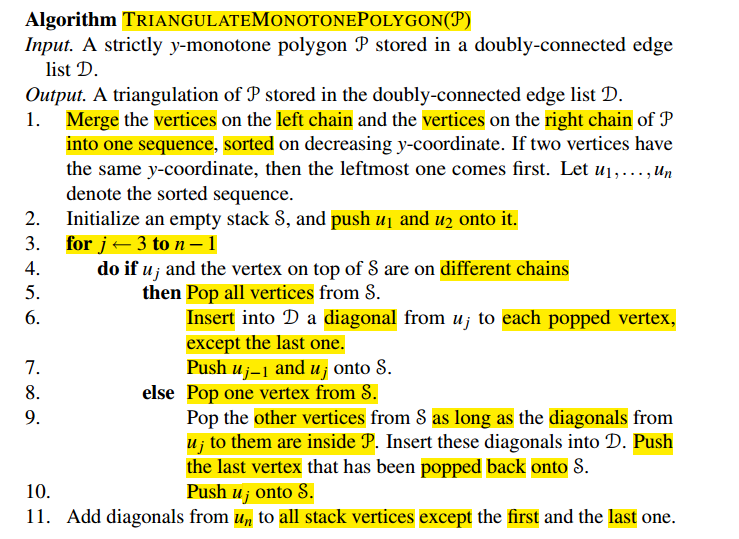


* Vj nằm cùng biên với chuỗi reflex

Thuật toán sẽ thêm đường chéo từ vj tới các điểm nằm trên của stack nhiều nhất có thể miễn là đường chéo thêm vào là hợp lệ. Pop các đỉnh được thêm đường chéo khỏi stack (ngoại trừ đỉnh cuối cùng). Push vj vào stack.



Sơ đồ toàn bộ thuật toán phân đa giác strictly y-monotone thành các tam giác.



Phân tích độ phức tạp của thuật toán tam giác phân.

* Bước 1 thực hiện trong O(n) (trộn 2 danh sách đã sắp xếp giống trộn trong Merge sort)
* Bước 2: O(1)
* Bước 3: Thực hiện n-3 vòng lặp, mỗi lần thực hiện 1 số lần push và pop trên stack và thêm đường chéo trong D. Cụ thể, mỗi vòng lặp push tối đa 2 đỉnh. Do đó nếu tính cả bước 2 thì số lần push tối đa là 2n-4. Vì thế số lần pop tối đa là 2n-4 (không vượt quá số lần push). Do đó độ phức tạp của vòng for này là O(n).
* Bước 11: O(n)

**Vậy độ phức tạp của thuật toán tam giác phân một đa giác strictly y-monotone là O(n). Bộ nhớ sử dụng là O(n)**.

Đa giác monotone có thể dùng phép ‘quay nhẹ’ để thành strictly monotone.

# Kết luận

Độ phức tạp của thuật toán tam giác phân một đa giác đơn là:

* Phân rã đa giác đơn thành các đa giác monotone trong O(nlogn). Bộ nhớ sử dụng là O(n).
* Tam giác phân đa giác monotone trong O(n). Bộ nhớ sử dụng là O(n).

Vậy bài toán tam giác phân một đa giác đơn thực hiện trong O(nlogn) và bộ nhớ sử dụng là O(n).

# Tài liệu tham khảo

1. [*Computational Geometry - Algorithms and Applications*](http://www.cs.uu.nl/geobook/) by de Berg, Cheong, van Kreveld, and Overmars, third edition, 2008
2. <http://www.cs.uu.nl/docs/vakken/ga/slides3.pdf>
3. <https://www.cs.umd.edu/class/fall2014/cmsc754/lectures.shtml>