ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN - ĐIỆN TỬ

BÁO CÁO

BÀI TẬP LỚN
MÔN HỌC PHƯƠNG PHÁP TÍNH
(MT1009)

ĐỀ TÀI 14
Finding shortest paths for autonomous robots in 2D and 2,5D

GVHD: .....

Lớp: AN01

Nhóm số: 02

Danh sách thành viên:

STT	Họ và tên	MSSV
1	Lưu Gia Huy	2411194
2	Lê Đức Minh	2414110
3		
4		

TP. HỒ CHÍ MINH, 2025

Lời mở đầu

I like to acknowledge ...

Lời cảm ơn

I like to acknowledge ...

Tóm tắt

I like to acknowledge ...

Mục lục

1 Giới thiệu **1**
1.1 Giới thiệu đề tài . . . 1
1.2 Các hướng giải quyết liên quan . . . 1

2 Cơ sở lý thuyết **2**
2.1 Các định nghĩa và bổ đề . . . 2
2.2 Thuật toán đề xuất . . . 3

3 Triển khai thuật toán **6**
3.1 Tam giác hóa . . . 6
3.2 Dual tree . . . 6
3.3 Sleeve . . . 7
3.4 Lee & Preparata . . . 7
3.5 Kết quả . . . 7
3.5.1 Độ phức tạp của các thuật toán . . . 7
3.5.2 Thời gian chạy thực tế . . . 7

4 Tổng kết **8**
4.1 Nhận xét . . . 8
4.2 Hướng phát triển . . . 8

5 Tài liệu tham khảo **8**

6 Phụ lục **9**

Danh sách hình vẽ

1 Đa giác đã tam giác hóa và dual tree của nó. 2
2 Minh họa về sự lồi vào trong của $D\left(s, v_i^{(j)}\right)$. 3
3 Hai trường hợp của bước tổng quát. 4
4 Chứng minh đường đi ngắn nhất phải đi qua v. 5

Danh sách bảng

1 Độ phức tạp của các thuật toán 7

1 Giới thiệu

1.1 Giới thiệu đề tài

Trong lập trình Robot tự động, việc tìm đường đi ngắn nhất giữa hai điểm giúp tiết kiệm tài nguyên, năng lượng, và thời gian. Điều này không chỉ nâng cao hiệu suất làm việc của Robot mà còn tối ưu hóa lợi nhuận trong nhiều ứng dụng thực tế.

Trong bài viết này, nhóm chúng em sẽ nghiên cứu và triển khai thuật toán của D. T. Lee và F. P. Preparata nhằm tìm đường đi ngắn nhất trong một đa giác đơn (*simple polygon*). Thuật toán này giúp cải thiện đáng kể thời gian xử lý so với các phương pháp truyền thống, có nhiều ứng dụng trong lĩnh vực robot tự động và điều hướng thông minh.

1.2 Các hướng giải quyết liên quan

Bài toán tìm đường đi ngắn nhất giữa hai điểm phân biệt (s: điểm bắt đầu, t: điểm kết thúc) trong mặt phẳng 2D là một trong những bài toán quan trọng của lý thuyết đồ thị và hình học tính toán. Có nhiều phương pháp tiếp cận bài toán này, trong đó phương pháp cổ điển sử dụng thuật toán Dijkstra trên đồ thị visbility graph, với độ phức tạp $O(n^2)$.

Tuy nhiên, thuật toán của Lee và Preparata cho phép tìm đường đi ngắn nhất trong một đa giác đơn với thời gian $O(n)$ nếu bỏ qua bước tiền xử lý tam giác hóa. Như vậy, tổng độ phức tạp của thuật toán phụ thuộc vào quá trình tam giác hóa đa giác. Hiện nay, đã có những nghiên cứu chỉ ra rằng có thể tam giác hóa một đa giác đơn trong thời gian $O(n)$ [1]. Do đó, thuật toán Lee và Preparata kết hợp với phương pháp tam giác hóa nhanh có thể là một giải pháp hiệu quả để giải quyết bài toán này.

2 Cơ sở lý thuyết

2.1 Các định nghĩa và bổ đề

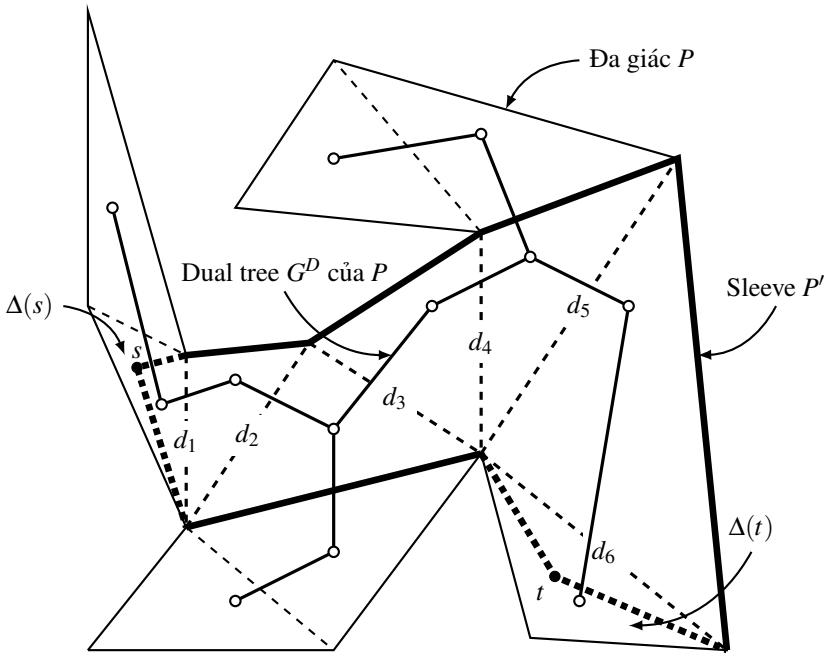
Phần này giới thiệu một số định nghĩa sẽ được sử dụng trong bài:

Định nghĩa 1. *Một chuỗi đa giác*[1] $P = \overline{q_1 q_2 \cdots q_k}$ *là chuỗi của các điểm* $q_i (i = 1, 2, \cdots, k)$ *với mỗi cặp* q_i *và* q_{i+1} *là một đoạn thẳng và không có hai đoạn thẳng không liên tiếp nào cắt nhau.*

Định nghĩa 2. *Một đa giác đơn*[2] *n đỉnh* $P = (q_1, q_2, \cdots, q_n)$ *là một chuỗi đa giác* $\overline{q_1 q_2 \cdots q_{n+1}}$ *với* $q_{n+1} = q_1$. *Đường chéo của P là đoạn thẳng* $\overline{q_i q_j}, j \neq i+1$ *và không cắt bất kỳ cạnh nào của P. Khi đó P được coi như đã tam giác hóa nếu phần bên trong nó được chia thành* $n-2$ *tam giác bởi* $n-3$ *đường chéo.*

Các đa giác đơn đã tam giác hóa có một tính chất thú vị như sau: Chúng ta có thể xem như đa giác đó là một đồ thị phẳng G trong cùng mặt phẳng. Mỗi tam giác là một miền trong[3] của G. Từ đó xây dựng đồ thị đối ngẫu G^D của G, mỗi mặt của G là một điểm của G^D và mỗi cạnh của G^D được nối từ hai điểm mà mặt của chúng có cạnh chung trong G. Khi bỏ đi các điểm thuộc miền ngoài, G^D trở thành một cây (tree) với các đỉnh có bậc tối đa là 3:

Định nghĩa 3. *Dual tree*[4] *của một đa giác đơn đã tam giác hóa P là một đồ thị* $T = (V, E)$ *sao cho mỗi nút V tương ứng với một tam giác của P và mỗi cạnh của E được nối bởi hai điểm khác nhau thuộc V nếu tam giác tương ứng của chúng có cạnh chung là một đường chéo của P. Đường chéo của P và cạnh tương ứng của E trong T được gọi là dual.*



Hình 1: *Đa giác đã tam giác hóa và dual tree của nó.*

Cần chú ý rằng có vô số cách để tam giác hóa một đa giác đơn và một trong những thuật toán sử dụng trong bài này sử dụng thuật toán $O(n \log n)$ [2].

Phương pháp của Lee và Preparata dựa trên quan sát sau: Gọi $\Delta(s)$ và $\Delta(t)$ lần lượt là hai tam giác trong P chứa s, t tương ứng. Trong T, tồn tại duy nhất đường đi π qua các đỉnh của T là các đối ngẫu của $\Delta(s)$ và $\Delta(t)$. Các cạnh trong π là đối ngẫu của các đường chéo trong P nên chuỗi các cạnh này tương ứng là chuỗi các đường chéo $d_1, d_2, \cdots, d_p$ theo thứ tự từ s tới t. Vì d_i bất kỳ chia P thành hai phần, một chứa s và một chứa t nên đường đi ngắn nhất từ s tới t trong P sẽ phải đi ngang qua mỗi đường chéo $d_1, d_2, \cdots, d_p$ một lần duy nhất. Ta có bổ đề sau (Chứng minh [3]):

Bổ đề 1. *Gọi S là tập hợp các đầu mút của các rào chắn thẳng hàng. Các đỉnh của đường đi ngắn nhất giữa s và t thuộc* $S \cup \{s, t\}$.

[1] Từ nguyên: polygonal chain
[2] Từ nguyên: simple polygon
[3] Từ nguyên: interior face
[4] Tạm dịch: cây đối ngẫu

Trong trường hợp này, S là tập hợp các đỉnh của P. Bổ đề 1 nói ta chỉ cần dựng đường ngắn nhất từ s tới hai đỉnh lần lượt của $d_1, d_2, ..., d_p$ và cuối cùng tới t. Hợp của các đường này gọi là cây đường đi ngắn nhất[5] với gốc là s. Bởi vì mỗi đường đi từ s đều đi qua một đường chéo một lần duy nhất, ta sẽ đi xây dựng thuật toán tham lam[6]: xét các tam giác lần lượt trên π, với mỗi tam giác, ta mở rộng cây đường đi ngắn nhất thêm một cạnh và một đỉnh.

2.2 Thuật toán đề xuất

Phần này nói về thuật toán tìm được đi ngắn nhất giữa s và t bên trong P thông qua thuật toán mở rộng cây đường đi ngắn nhất từng cạnh một. Bắt đầu với định nghĩa sau:

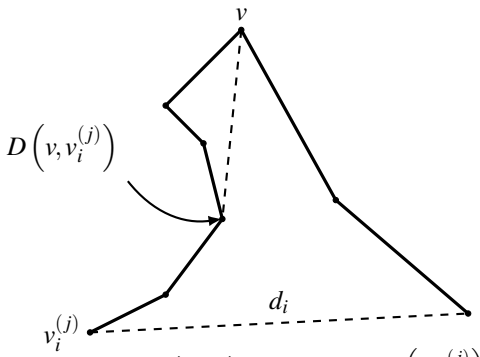
Định nghĩa 4. *Một đa giác đã tam giác hóa được gọi là sleeve[7] nếu cây đối ngẫu của nó là một chuỗi.*

Với định nghĩa này, ta xây dựng đa giác P' đối ngẫu với π sao cho s,t là các đỉnh của đa giác P'. Trong phần tiếp theo, ta sẽ giả sử rằng đa giác P là một sleeve với n đỉnh bao gồm s và t.

Gọi $v_i^{(1)}$ và $v_i^{(2)}$ lần lượt là hai đầu mút của đường chéo $d_i, 1 \le i \le n-3$, và gọi $D\left(s, v_i^{(j)}\right)$ là đường đi ngắn nhất từ s tới $v_i^{(j)}, j = 1,2$ bên trong đa giác P. Từ **??** $D\left(s, v_i^{(j)}\right)$ là chuỗi bao gồm các đỉnh của P. Gọi $D_i = D\left(s, v_i^{(1)}\right) \cup D\left(s, v_i^{(2)}\right)$, trong D_i tồn tại duy nhất một đỉnh v là đỉnh chung của cả $D\left(s, v_i^{(1)}\right)$ và $D\left(s, v_i^{(2)}\right)$, và v cách s xa nhất (theo các chuỗi); hay là hai chuỗi $D\left(s, v_i^{(1)}\right)$ và $D\left(s, v_i^{(2)}\right)$ tách ra tại v.

Giả sử ban đầu không có chuỗi D nào rỗng; ta thấy rằng: $D\left(v, v_i^{(j)}\right)$ $(j = 1,2)$ là một chuỗi đa giác lồi vào trong[8]; nghĩa là phần lồi của nó hướng vào bên trong P.

Chứng minh. Xét miền R_i giới hạn bởi $D\left(v, v_i^{(1)}\right), D\left(v, v_i^{(2)}\right)$ và d_i gọi là *phễu*[9] nằm trong P. Gọi $d_s, d_{s+1}, \cdots, d_{i-1}$ là các đường chéo cắt $D\left(v, v_i^{(1)}\right), D\left(v, v_i^{(2)}\right)$. Dễ thấy tam giác $\left(v, v_s^{(1)}, v_s^{(2)}\right) = R_S$ nằm trong P, theo phương pháp quy nạp thì $R_{i-1} \subset P$. Nếu $D\left(v, v_i^{(1)}\right)$ không lồi vào trong, theo bất đẳng thức tam giác, tồn tại một đường đi ngắn hơn từ v tới $v_i^{(j)}$ (xem 2). Tính chất lồi này chứng minh rằng $D\left(v, v_i^{(1)}\right)$ và $D\left(v, v_i^{(2)}\right)$ tách ra tại nhiều nhất một đỉnh v, nếu chúng tách ra tại một đỉnh u_1 khác, nghĩa là chúng phải tái hợp lại tại một đỉnh u_2 nào đó; và hai chuỗi phân biệt u_1, u_2 mới này phải cùng lồi vào trong. □



Hình 2: *Minh họa về sự lồi vào trong của* $D\left(s, v_i^{(j)}\right)$.

Tổng quát đối với D_i, các chuỗi tách ra tại đỉnh v nào đó được gọi là *đỉnh phễu*[10] của hai chuỗi lồi vào trong. Chú ý rằng một trong hai chuỗi có thể rỗng nhưng không thể cả hai vì $v_i^{(1)} \neq v_i^{(2)}$. Nếu $D\left(v, v_i^{(1)}\right)$ rỗng thì thì $D\left(v, v_i^{(2)}\right) = d_i$.

[5] Từ nguyên: shortest-path tree
[6] Từ nguyên: greedy algorithm
[7] Tạm dịch: tay áo
[8] Từ nguyên: inward-convex
[9] Từ nguyên: funnel
[10] Từ nguyên: cusp

Thuật toán xây dựng $D_1, D_2, \cdots, D_p$ và cuối cùng $D(s,t)$. Chi tiết:

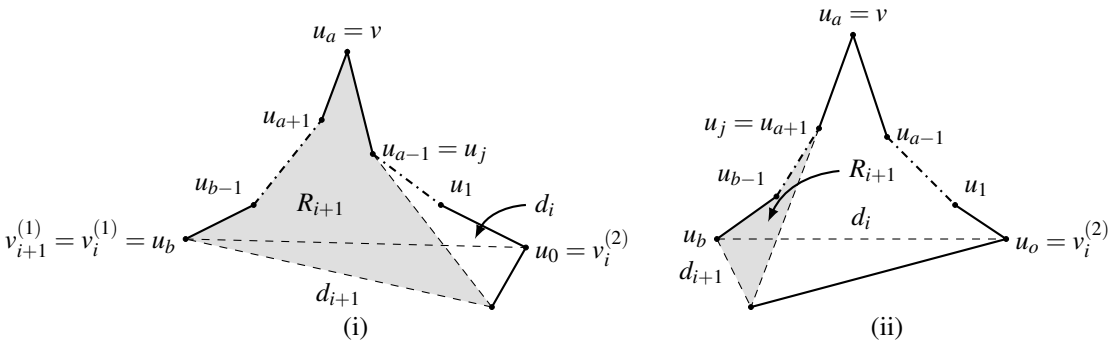
Algorithm 1 Xây dựng chuỗi D_i

Ban đầu: nối s với $v_1^{(1)}$ và $v_1^{(2)}$ để tạo D_1.
Tổng quát: Để tạo ra D_{i+1} từ D_i:
Gọi v là đỉnh phễu của D_i, tại đỉnh hai chuỗi $\overline{u_a u_{a+1} \cdots u_b}$ và $\overline{u_a u_{a-1} \cdots u_0}$ tách ra, $v = u_a, v_i^{(1)} = u_b, v_i^{(2)} = u_0$. Không mất tính tổng quát: $v_i^{(1)} = v_{i+1}^{(1)}$. Bắt đầu từ u_0, gọi j là số nguyên nhỏ nhất mà $\overline{v_{i+1}^{(2)} u_j}$ trở thành *đường hỗ trợ*[11] của đường biên của R_i. Xét hai trường hợp:

(i) $j \leq a$. Xóa tất cả cạnh $\overline{u_l u_{l+1}}, 0 \leq l \leq j-1$ và thêm cạnh $\overline{u_j v_{i+1}^{(2)}}$.

(ii) $j > a$. Xóa tất cả cạnh $\overline{u_l u_{l+1}}, 0 \leq l \leq j-1$ và thêm cạnh $\overline{u_j v_{i+1}^{(2)}}$; u_j trở thành đỉnh phễu mới của R_{i+1}.

Bước cuối: Khi xây dựng xong D_{n-3}, ta xem như có thêm một đường chéo chứa t (đường chéo d_{n-2}) và áp dụng lại bước Tổng quát ở trên.
Kết quả: $D(s,t)$



Hình 3: *Hai trường hợp của bước tổng quát.*

Tính đúng đắn của thuật toán dựa trên nhận xét sau: Với mọi u trong tam giác R_{i+1} xác định bởi hai đường chéo d_i, d_{i+1}, đường đi ngắn nhất từ s đến u đi qua v là đỉnh phễu của D_i. Hiển nhiên đúng với $i = 1, v = s$. Ta đi chứng minh bằng quy nạp như sau:

Chứng minh. giả sử đúng tới $1, \cdots, i-1$; xét hai chuỗi $D\left(v, v_i^{(1)}\right), D\left(v, v_i^{(2)}\right)$. Nếu một trong hai chuỗi là rỗng, chuỗi còn lại sẽ bao gồm d_i với v trở thành đỉnh của d_i. Trong trường hợp này, đường đi ngắn nhất từ s tới u phải là đoạn ngắn nhất của $D(s,v)$ và đoạn thẳng $\overline{vu}$, và hoàn tất chứng minh. Nếu cả hai chuỗi đều không rỗng, xét cạnh kề với đỉnh v trên bất kỳ chuỗi con nào. Vì P là sleeve, ít nhất một trong số chúng là đường chéo của P (kể cả khi nó không phải đường chéo có được từ thuật toán tam giác hóa P). Gọi $\overline{vv'}$ là đường chéo và $\overline{vv''}$ không phải đường chéo. Bởi vì sự lồi vào trong của hai chuỗi, các tia tạo bởi $\overline{vv'}$ và $\overline{vv''}$ cắt R_{i+1} thành ba phần. Điểm u sẽ thuộc một trong ba phần này; cả ba trường hợp đều tương tự nhau. Giả sử đường ngắn nhất từ s tới u là chuỗi $l(s,u)$ không đi qua v mà thay vào đó cắt $\overline{vv'}, \overline{vv''}$ tại $p \neq v, p_1 \neq v$. Theo định nghĩa đường đi ngắn nhất:

$$length(l(s,p)) + length(l(p,p_1)) < length(D(s,v)) + length(\overline{vp_1})$$

theo bất đẳng thức tam giác:

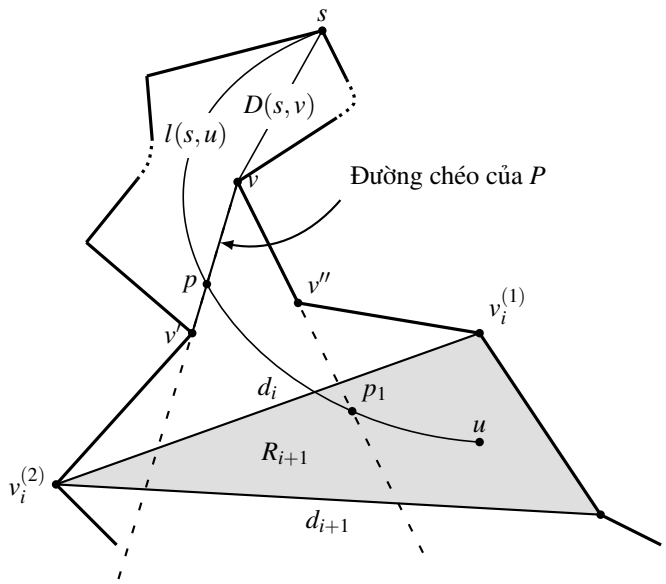
$$length(D(s,v)) + length(\overline{vp}) - length(l(s,p)) > 0$$

Từ đó:

$$length(D(s,v)) + length\left(\overline{vv'}\right) > length(l(s,p)) + length\left(\overline{pv'}\right)$$

Trường hợp này mâu thuẫn với trường hợp đầu là đường ngắn nhất từ s tới v' là qua v. □

[11] Từ nguyên: supporting line. l là đường hỗ trợ của một đường cong C nếu nó có một điểm chung với C và C nằm toàn bộ ở một phía của l



Hình 4: *Chứng minh đường đi ngắn nhất phải đi qua v.*

Độ phức tạp của thuật toán: trường hợp (i) của bước Tổng quát cần thời gian không đổi (constant time); trường hợp (ii) có thể phải kiểm tra số lượng lớn đỉnh tuy nhiên mỗi điểm chỉ cần xem xét một lần trong quá trình kiểm tra và P có $n-2$ đỉnh bên cạnh s,t nên toàn bộ thuật toán chỉ cần $O(n)$. Tuy nhiên đó là khi giả sử rằng P là sleeve; còn đối với bất kỳ một đa giác đơn P gồm n đỉnh, để chuyển hóa thành sleeve, đầu tiên chúng ta phải tam giác hóa P: $O(n\log n)$; xây dựng dual tree T cần $O(n)$. Vậy nên toàn bộ quá trình có độ phức tạp $O(n\log n)$ với quá trình tam giác hóa chiếm nhiều thời gian nhất.

3 Triển khai thuật toán

Toàn bộ code của thuật toán có thể tìm được ở link sau. https://github.com/tincuri/BTL_pptinh

3.1 Tam giác hóa

Ở bước này nhóm em sử dụng thuật toán của ông Seidel[**?**] lấy source code từ trang web sau [**?**,] Ông ấy hình thang hóa đa giác và từ đó tách đa giác đó ra thành các monotone polygon. Từ đây ông dùng một thuật toán tham lam để tam giác hóa các đa giác con này. Độ phức tạp của thuật toán hình thang hóa là $O(n\log^* n)$, với log* n là hàm log lặp[12] của n với tốc độ biến thiên rất chậm $\log_b^*(n) << \log_b(n)$ và trong thực tế thì gần tương đương với hàm hằng. Phần tam giác hóa của các monotone polygon con có độ phức tạp $O(n)$. Do đó độ phức tạp của thuật toán tam giác hóa là $O(n\log^* n)$.

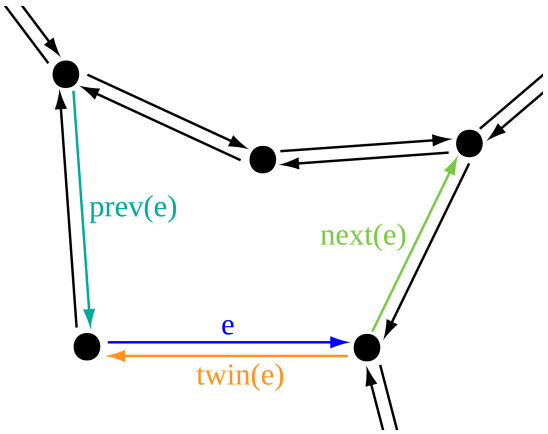
3.2 Dual tree

Ở phần này nhóm em sử dụng cấu trúc **Doubly connected edge list (dcel)** để biểu diễn hình dạng của đa giác đã được tam giác hóa.

Dcel là một kiểu dữ liệu dùng để biểu diễn một đồ thị phẳng $G(V,E)$ với $V = \{v_1, v_2, ..v_n\}$ là là tập hợp các điểm của đồ thị và $E = \{e_1, e_2, ..e_n\}$ là tập hợp các cạnh kết nối giữa các điểm. dcel gồm 3 kiểu dữ liệu: half_edge, vertex, face. *half_edge* như tên gọi của nó, là một cấu trúc dữ liệu gần giống như một nửa của một cạnh. Mỗi *half_edge* chứa các miền dữ liệu sau.

- Twin, là một cạnh đối ngược với nó.
- Origin, là điểm bắt đầu của *half_edge*.
- Next, là một *half_edge* có điểm bắt đầu là điểm kết thúc của *half_edge* này.
- Prev, là một *half_edge* có điểm kết thúc là điểm bắt đầu của *half_edge* này.
- IncidentFace, là một *face* mà có *half_edge* này làm bờ.

face và *vertex* có thêm một miền dữ liệu chứa một *half_edge* mà các dữ liệu đó kề.
Các miền dữ liệu của *half_edge* được diễn tả như hình sau.



Dcel có thể được dựng dễ dàng như sau[4]:

1. Với từng điểm v_i tạo một *vertex* tương ứng.
2. Với từng cạnh e_i tạo hai *half_edge* tương ứng, đặt Origin cho 2 *half_edge* đó.
3. Với từng *vertex*, Sắp xếp các cạnh có nó là Origin theo chiều kim đồng hồ.
4. Với hai *half_edge* e_1, e_2 kề nhau (trong thứ tự đã sắp xếp của một *vertex* ở bước 3), gán $\text{Next}(\text{Twin}(e_1)) = e_2$ và $\text{Prev}(e_2) = \text{Twin}(e_1)$.
5. Lần lượt gán một *face* cho các chuỗi *half_edge*

[12] Từ nguyên: iterative logarithm

Trong bước cuối, ta có thể lợi dụng việc gán mặt để tạo cho từ mặt một đỉnh trong đồ thị, và liên kết các đỉnh lại với nhau nếu hai mặt có chung một cạnh, mỗi liên kết có một miền dữ liệu là cạnh liên kết giữa hai mặt. Do đó ta có thể dựng được Dual tree của đa giác khi đang dựng dcel.

Như ta đã biết, để tam giác hóa một đa giác n cạnh thì ta cần thêm $n-3$ đường chéo, mà mỗi đường chéo có 2 đỉnh. Do đó, trong trường hợp trung bình, mỗi đỉnh sẽ liên kết với khoảng 4 cạnh (thêm hai cạnh của đa giác ban đầu). Vì vậy độ phức tạp trung bình của thuật toán tạo DCEL trong trường hợp tam giác hóa là $O(n)$ nếu như bảo đảm các góc có số lượng cạnh liên kết tới nó gần ngang nhau. Trong trường hợp tệ nhất, tất cả các cạnh đều liên kết tới một góc, độ phức tạp của thuật toán khi này là $O(n\log n)$ do bước sắp xếp có độ phức tạp là $O(n\log n)$.

3.3 Sleeve

Bước đầu tiên, ta cần tìm vị trí của 2 điểm cần xét, ta có thể check[13] xem điểm có nằm trong một tam giác bằng định hướng của nó với 3 đỉnh tam giác. Ta chỉ cần lặp qua các *face* đến khi nào tìm được *face* mà điểm đó nằm trong. Thuật toán có độ phức tạp là $O(n)$.

Với Dual Tree đã được tạo như trên, Nhóm em sử dụng Depth-first search (DFS) để tìm đường liên kết giữa 2 đỉnh tương ứng với 2 tam giác chứa 2 điểm. Đây chính là *sleeve*. Từ đây ta rút ra được các đường chéo cần xét từ các liên kết. Độ phức tạp của DFS là $O(V+E)$ với $V=n$ là số đỉnh còn $E=n-3$ là số cạnh. Do đó trong trường hợp này thuật toán tạo *sleeve* có độ phức tạp là $O(n)$.

3.4 Lee & Preparata

3.5 Kết quả

3.5.1 Độ phức tạp của các thuật toán

Thuật toán	Độ phức tạp trung bình	Độ phức tạp tệ nhất
Tam giác hóa	$O(n\log^* n)$	$O(n\log^* n)$
Tìm Dual Tree	$O(n)$	$O(n\log(n))$
Tìm Sleeve	$O(n)$	$O(n)$
Lee & Preparata	$O(n)$	$O(n)$

Bảng 1: *Độ phức tạp của các thuật toán*

3.5.2 Thời gian chạy thực tế

[13]

4 Tổng kết

4.1 Nhận xét

4.2 Hướng phát triển

Thay cho cấu trúc dcel, Ta cũng có thể dựng các cấu trúc khác như winged edge hay quad-edge cũng dùng để dựng một đồ thị phẳng và có thể có thời gian dựng ngắn hơn.

5 Tài liệu tham khảo

Tài liệu

[1] Bernard Chazelle. Triangulating a Simple Polygon in Linear Time. *Discrete Comput. Geom.*, 6(5):485-524, 1991.

[2] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan, Triangulating a simple polygon. *Information Processing Lett*. 7 (1978) 175-179.

[3] O. Chein and L. Steinberg, Routing past unions of disjoint rectilinear barriers. *Networks* **13** (1983) 389-398.

[4] https://cs.stackexchange.com/questions/2450/how-do-i-construct-a-doubly-connected-edge-list-given-a-set-of-line-segments

6 Phụ lục