1. Thuật toán divide conquer để tìm bao lồi - độ phức tạp O(nlogn)

2. Thuật toán Chan’s để tìm bao lồi: độ phức tạp O(nlogh) với h là số điểm trong bao lồi.

3. Thuật toán Graham - scanning để tìm bao lồi: độ phức tạp O(nlogn

4. Thuật toán Jarrvis: độ phức tạp O(N2)

5. Thuật toán Monotone chain: độ phức tạp O(N\*logN)

Mã giả:

Chan:

Thuật toán Chan(v):

Đầu vào: Danh sách điểm v = [p1, p2, ..., pn] (mỗi điểm là một cặp tọa độ (x, y))

Đầu ra: Bao lồi của tập điểm v

Bước 1: Khởi tạo:

- hulls = [] // Danh sách chứa bao lồi của các nhóm con

- n = v.size() // Số lượng điểm trong danh sách

- h = 0 // Số điểm trong bao lồi cuối cùng (dùng để kiểm tra độ phức tạp)

Bước 2: Lặp qua các giá trị t từ 0 đến n-1 (t là số lần thử với kích thước nhóm con):

- t\_max = n-1 // Giá trị tối đa của t, là số lần thử chia nhóm

Lặp qua các giá trị m từ 1 đến 2^(2^t) - 1 (m là kích thước của mỗi nhóm con):

- m\_max = 2^(2^t) - 1 // Giá trị tối đa của m (số điểm trong mỗi nhóm)

Bước 2.1: Chia danh sách v thành các nhóm con có kích thước m:

- Chia v thành các nhóm con `chunks`, mỗi nhóm có m điểm

Lặp qua các chỉ số i từ 0 đến n với bước nhảy m (tạo ra các nhóm con có kích thước m):

- Nếu (i + m) <= n:

chunk = v[i, i + m - 1] // Tạo nhóm con từ vị trí i đến i+m-1

- Nếu (i + m) > n:

chunk = v[i, n - 1] // Lấy tất cả các điểm còn lại từ vị trí i đến hết danh sách

- hull\_con = GrahamScan(chunk) // Tính bao lồi cho nhóm con

- hulls.push\_back(hull\_con) // Thêm bao lồi của nhóm con vào danh sách `hulls`

Bước 2.2: Khởi tạo danh sách `hull` chứa các cặp điểm (pair) để xây dựng bao lồi tổng thể.

- hull = [] // Danh sách bao lồi tổng thể

- extreme\_pair = extreme\_hullpt\_pair(hulls) // Tìm cặp điểm cực trị ban đầu từ danh sách `hulls`

- hull.push\_back(extreme\_pair) // Thêm cặp điểm cực trị vào bao lồi tổng thể

Bước 2.3: Lặp qua các giá trị từ 1 đến m:

- Lặp m lần để tìm các cặp điểm tiếp theo và xây dựng bao lồi hoàn chỉnh

Lặp qua i từ 1 đến m:

- pair p = next\_hullpt\_pair(hulls, hull) // Tìm cặp điểm tiếp theo từ các bao lồi con

- Nếu p == hull[0]: // Nếu cặp điểm tiếp theo là điểm đầu tiên trong bao lồi

- output = [] // Khởi tạo danh sách output

- Lặp qua mỗi cặp điểm trong `hull`:

- output.push\_back(hulls[hull[j].first][hull[j].second]) // Thêm điểm vào output từ danh sách `hulls`

- output.pop\_back() // Loại bỏ điểm dư thừa

- Trả về output // Trả về bao lồi hoàn chỉnh

- hull.push\_back(p) // Nếu chưa hoàn thành, tiếp tục thêm cặp điểm vào bao lồi

Bước 3: Nếu không tìm được bao lồi hoàn chỉnh sau khi lặp qua tất cả các nhóm con, trả về danh sách rỗng.

- return [] // Nếu không tìm được bao lồi hợp lệ, trả về danh sách rỗng

Bước 4: Trả về bao lồi hoàn chỉnh từ danh sách `hull`.

- return output // Trả về bao lồi được xây dựng từ danh sách `hull`

Divide and conquer:

Hàm merger(a, b):

Đầu vào: a và b là hai danh sách các điểm (x, y) đại diện cho bao lồi của hai tập điểm

Đầu ra: Bao lồi của hợp hai bao lồi a và b

Khởi tạo:

- n1 = a.size() // Kích thước của bao lồi a

- n2 = b.size() // Kích thước của bao lồi b

- ia = 0, ib = 0 // Chỉ số khởi tạo của a và b

Tìm điểm cực trị của a và b:

- Lặp qua các điểm trong a để tìm điểm có x lớn nhất (ia)

- Lặp qua các điểm trong b để tìm điểm có x nhỏ nhất (ib)

Xác định điểm trên và điểm dưới của bao lồi kết hợp:

- Tìm điểm cực trị phía trên và dưới của bao lồi kết hợp từ hai bao lồi con.

Tạo bao lồi tổng hợp:

- Tạo danh sách `ret` chứa các điểm của bao lồi kết hợp.

- Thêm các điểm từ bao lồi a và b vào `ret` theo thứ tự hợp lý (từ điểm trên đến điểm dưới).

Trả về bao lồi `ret`.

Hàm bruteHull(a):

Đầu vào: a là danh sách các điểm (x, y)

Đầu ra: Bao lồi của tập điểm a

Khởi tạo:

- s = set() // Dùng để lưu các điểm thuộc bao lồi

- mid = {0, 0} // Trung tâm của các điểm (có thể dùng cho việc tính toán hướng)

Tìm tất cả các cặp điểm trên cùng một đường thẳng:

- Duyệt qua tất cả các cặp điểm trong danh sách `a` để kiểm tra xem liệu chúng có tạo thành bao lồi hay không:

- Tính phương trình đường thẳng qua hai điểm.

- Kiểm tra xem tất cả các điểm còn lại có nằm cùng một phía với đường thẳng này hay không.

- Nếu tất cả các điểm thuộc một phía, thêm hai điểm vào set `s`.

Tạo bao lồi từ các điểm đã tìm:

- Chuyển các điểm từ set `s` vào danh sách `ret`.

- Tính trung tâm các điểm trong bao lồi.

- Đặt các điểm theo thứ tự góc với trung tâm.

- Trả về bao lồi đã được sắp xếp.

Trả về bao lồi `ret`.

Hàm divide(a):

Đầu vào: a là danh sách các điểm (x, y)

Đầu ra: Bao lồi của tập điểm a

Nếu số lượng điểm trong a <= 5:

- Gọi hàm bruteHull(a) để tính bao lồi.

Khởi tạo:

- left = [] // Danh sách các điểm bên trái

- right = [] // Danh sách các điểm bên phải

- Chia danh sách `a` thành hai phần (left và right):

- left chứa các điểm từ đầu đến giữa.

- right chứa các điểm từ giữa đến cuối.

Tính bao lồi cho từng phần:

- left\_hull = divide(left) // Tính bao lồi cho phần bên trái

- right\_hull = divide(right) // Tính bao lồi cho phần bên phải

Gộp bao lồi của hai phần:

- return merger(left\_hull, right\_hull) // Gộp bao lồi của phần trái và phải.

Hàm orientation(p, q, r):

Đầu vào: p, q, r là ba điểm trong mặt phẳng 2D

Đầu ra: Giá trị biểu thị hướng của ba điểm

Tính giá trị định thức của ma trận gồm ba điểm p, q, r:

- val = (q.x - p.x) \* (r.y - p.y) - (q.y - p.y) \* (r.x - p.x)

Nếu val > 0: Hướng ngược chiều kim đồng hồ (CCW)

- return 1

Nếu val < 0: Hướng theo chiều kim đồng hồ (CW)

- return -1

Nếu val == 0: Các điểm p, q, r nằm trên một đường thẳng

- return 0

Graham scanning:

Hàm grahamScan(points):

Đầu vào: points là danh sách các điểm trong không gian 2D

Đầu ra: Bao lồi của các điểm trong points

Bước 1: Tìm điểm thấp nhất

- v\_lowest = points[0] // Khởi tạo điểm thấp nhất bằng điểm đầu tiên

- Duyệt qua các điểm từ điểm thứ 2 đến hết để tìm điểm thấp nhất:

- Nếu y của điểm i lớn hơn y của v\_lowest, cập nhật v\_lowest = điểm i

- Nếu y của điểm i bằng y của v\_lowest và x của điểm i nhỏ hơn x của v\_lowest, cập nhật v\_lowest = điểm i

Bước 2: Sắp xếp các điểm còn lại theo góc

- Sắp xếp các điểm còn lại (ngoại trừ v\_lowest) theo góc tính từ v\_lowest:

- Đối với mỗi cặp điểm a và b, tính góc giữa v\_lowest và a, v\_lowest và b

- Sắp xếp các điểm theo góc từ nhỏ đến lớn với v\_lowest

Bước 3: Tạo bao lồi

- Tạo một ngăn xếp (stack) để lưu trữ các điểm tạo thành bao lồi

- Thêm 2 điểm đầu tiên vào ngăn xếp:

- stack.push\_back(points[0])

- stack.push\_back(points[1])

- Duyệt qua các điểm còn lại (từ điểm thứ 3 trở đi):

- Lấy 2 điểm cuối trong ngăn xếp: p1 = stack[stacklen - 2], p2 = stack[stacklen - 1]

- Kiểm tra xem p1, p2 và điểm hiện tại có tạo thành một quay trái (left turn) không:

- Nếu là quay trái, thêm điểm hiện tại vào ngăn xếp: stack.push\_back(points[index])

- Nếu không phải quay trái, loại bỏ điểm p2 khỏi ngăn xếp (pop\_back)

Bước 4: Trả về ngăn xếp (bao lồi)

- Trả về ngăn xếp chứa các điểm tạo thành bao lồi.

Jarvis:

Hàm orientation(p, q, r):

Tính giá trị val = (q.y - p.y) \* (r.x - q.x) - (q.x - p.x) \* (r.y - q.y)

Nếu val = 0:

Trả về 0 (Ba điểm p, q, r nằm trên một đường thẳng)

Nếu val > 0:

Trả về 1 (Ba điểm tạo thành quay trái)

Nếu val < 0:

Trả về 2 (Ba điểm tạo thành quay phải)

Hàm convexHull(points, n):

Đầu vào: points là mảng các điểm trong không gian 2D, n là số lượng điểm

Đầu ra: Bao lồi của các điểm trong points

Bước 1: Nếu số lượng điểm ít hơn 3, trả về danh sách rỗng (không thể tạo bao lồi)

Nếu n < 3:

Trả về {}

Bước 2: Tìm điểm có hoành độ nhỏ nhất

- l = 0 // Khởi tạo điểm có hoành độ nhỏ nhất là điểm đầu tiên

- Duyệt qua các điểm từ i = 1 đến n-1:

- Nếu x của điểm i < x của điểm l, cập nhật l = i

Bước 3: Khởi tạo bao lồi (hull) và chọn điểm bắt đầu p = l

- p = l

- Tạo một mảng hull rỗng để lưu các điểm của bao lồi

Bước 4: Duyệt qua các điểm để tạo bao lồi

Lặp lại cho đến khi quay lại điểm ban đầu:

- Thêm điểm p vào bao lồi: hull.push\_back(points[p])

- q = (p + 1) % n // Chọn điểm tiếp theo là điểm kế bên p

- Duyệt qua các điểm còn lại từ i = 0 đến n-1:

- Nếu orientation(points[p], points[i], points[q]) == 2:

- Cập nhật q = i (chọn điểm mới q tạo thành quay trái với p và q)

- p = q (di chuyển đến điểm tiếp theo trong bao lồi)

Bước 5: Trả về bao lồi

Trả về hull

Monotone chain:

Hàm cross(p, q, r):

Tính giá trị val = (q.x - p.x) \* (r.y - p.y) - (q.y - p.y) \* (r.x - p.x)

Nếu val = 0:

Trả về 0 (Ba điểm p, q, r nằm trên một đường thẳng)

Nếu val > 0:

Trả về 1 (Ba điểm tạo thành quay trái)

Nếu val < 0:

Trả về -1 (Ba điểm tạo thành quay phải)

Hàm convex\_hull(P):

Đầu vào: P là danh sách các điểm trong không gian 2D

Đầu ra: Bao lồi của các điểm trong P

Bước 1: Nếu số lượng điểm trong P <= 3, trả về P

Nếu n <= 3:

Trả về P

Bước 2: Sắp xếp các điểm trong P theo hoành độ x, nếu bằng nhau thì theo tung độ y

Sắp xếp P theo (x, y)

Bước 3: Xây dựng bao lồi dưới

Khởi tạo k = 0

Duyệt qua các điểm P từ i = 0 đến n-1:

- Trong khi k >= 2 và cross(res[k-2], res[k-1], P[i]) <= 0:

- Giảm k (loại bỏ điểm cuối cùng trong bao lồi)

- Đặt res[k] = P[i]

- Tăng k

Bước 4: Xây dựng bao lồi trên

Khởi tạo i = n-1, t = k + 1

Duyệt qua các điểm P từ i = n-1 đến 1:

- Trong khi k >= t và cross(res[k-2], res[k-1], P[i-1]) <= 0:

- Giảm k (loại bỏ điểm cuối cùng trong bao lồi)

- Đặt res[k] = P[i-1]

- Tăng k

Bước 5: Điều chỉnh kết quả

- Đặt res có kích thước = k-1 để loại bỏ điểm dư thừa (điểm cuối trùng với điểm đầu)

Bước 6: Trả về bao lồi

Trả về res