ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

Khoa Khoa học máy tính

Môn học Phân Tích Và Thiết Kế Thuật Toán

Bài tập Hình học

Cao Lê Công Thành
MSSV: 23521437
Đặng Quang Vinh
MSSV: 23521786

Mục lục

1 Bài tập 1

1.1 Phân tích

Bài toán yêu cầu chúng ta tính toán độ dài ngắn nhất của sợi dây để bao quanh tất cả các cây trên mặt phẳng tọa độ, tương đương với việc tìm **bao lồi** (convex hull) của một tập hợp các điểm. Sau khi tìm được bao lồi, ta sẽ tính **chu vi** của bao lồi đó, đây chính là chiều dài ngắn nhất của sợi dây.

1.2 Các bước chính

- **Tìm bao lồi**: Để tìm được bao lồi, chúng ta sẽ sử dụng thuật toán **Monotone Chain**. Thuật toán này tìm các điểm thuộc bao lồi thông qua việc xây dựng hai phần của bao lồi: **nửa dưới** và **nửa trên**.
- **Tính chu vi của bao lồi**: Sau khi có được bao lồi, ta tính tổng độ dài các đoạn thẳng nối các đỉnh liên tiếp của bao lồi. Đây chính là chu vi của đa giác bao lồi.

1.3 Các kỹ thuật sử dụng

- **Tích có hướng (Cross Product)**: Để kiểm tra xem ba điểm có tạo thành một góc lồi hay không, chúng ta sử dụng tích có hướng giữa ba điểm. Nếu tích có hướng dương, ba điểm này tạo thành góc lồi; nếu tích có hướng âm, chúng tạo thành góc lõm. Nếu tích có hướng bằng 0, ba điểm nằm trên một đường thẳng.

  Công thức tính tích có hướng của ba điểm $O(x_1, y_1)$, $A(x_2, y_2)$, và $B(x_3, y_3)$ là:

  $$\text{cross}(O, A, B) = (x_2 - x_1) \times (y_3 - y_1) - (y_2 - y_1) \times (x_3 - x_1)$$

  Nếu giá trị này dương, ba điểm tạo góc lồi. Nếu giá trị âm, chúng tạo góc lõm.

- **Thuật toán tìm đường bao lồi (Monotone Chain)**: Thuật toán này chia quá trình tìm bao lồi thành hai phần: nửa dưới và nửa trên. Đầu tiên, ta sắp xếp các điểm, sau đó xây dựng nửa dưới và nửa trên của bao lồi, loại bỏ các điểm không thuộc bao lồi.

1.4 Mã giả

Dưới đây là mã giả cho thuật toán Monotone Chain để tìm bao lồi và tính chu vi của nó.

```
function convex_hull(A):
    // Sắp xếp các điểm theo tọa độ (x, y)
    Sắp xếp A tăng dần theo (x, y)

    // Xây dựng nửa dưới của bao lồi
    Hull = []
    for mỗi điểm Ai  A:
        while (Hull có ít nhất 2 điểm và không tạo góc lồi với điểm Ai):
            Loại bỏ điểm cuối trong Hull
        Thêm Ai vào Hull

    // Xây dựng nửa trên của bao lồi
    for mỗi điểm Ai  A (duyệt ngược):
        while (Hull có ít nhất 2 điểm và không tạo góc lồi với điểm Ai):
            Loại bỏ điểm cuối trong Hull
        Thêm Ai vào Hull

    // Loại bỏ các điểm trùng nhau ở đầu và cuối Hull
    return Hull

function calculate_perimeter(Hull):
    ChuVi = 0
    for mỗi cạnh (Ai, Ai+1)  Hull:
        ChuVi += độ dài cạnh (Ai, Ai+1)
    return ChuVi
```

1.5 Giải thích mã giả

- **Convex Hull (bao lồi)**:
  - **Sắp xếp điểm**: Đầu tiên, ta sắp xếp tất cả các điểm theo tọa độ (x, y). Đây là bước quan trọng để thuận tiện trong việc xây dựng bao lồi.
  - **Xây dựng nửa dưới**: Duyệt qua từng điểm và kiểm tra xem điểm đó có tạo góc lồi với các điểm đã có trong **Hull** hay không. Nếu không, loại bỏ điểm cuối trong **Hull** cho đến khi điểm mới tạo thành một góc lồi.
  - **Xây dựng nửa trên**: Quá trình này tương tự nhưng duyệt các điểm theo thứ tự ngược lại.

– **Kết hợp hai nửa**: Sau khi xây dựng nửa dưới và nửa trên, kết hợp chúng lại để tạo thành bao lồi hoàn chỉnh.

- **Tính chu vi**: Sau khi có được bao lồi, ta tính chu vi bằng cách tính độ dài các cạnh nối các điểm trong bao lồi. Độ dài của một cạnh giữa hai điểm $A(x_1, y_1)$ và $B(x_2, y_2)$ là:

$$\text{Distance}(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Chu vi là tổng độ dài của tất cả các cạnh này.

1.6 Phần Code:

```python
import math

def cross(o, a, b):
    return (a[0] - o[0]) * (b[1] - o[1]) - (a[1] - o[1]) * (b[0] - o[0])

def convex_hull(points):
    points = sorted(points)
    lower = []
    for p in points:
        while len(lower) >= 2 and cross(lower[-2], lower[-1], p) <= 0:
            lower.pop()
        lower.append(p)

    upper = []
    for p in reversed(points):
        while len(upper) >= 2 and cross(upper[-2], upper[-1], p) <= 0:
            upper.pop()
        upper.append(p)

    return lower[:-1] + upper[:-1]

def calculate_perimeter(hull):
    perimeter = 0
    for i in range(len(hull)):
        j = (i + 1) % len(hull)
        perimeter += math.dist(hull[i], hull[j])
    return perimeter
```

```
28
29 if __name__ == '__main__':
30     points = [(0, 0), (2, 2), (2, 0), (3, 1), (1, 3),
           (1, 0)]
31     hull = convex_hull(points)
32     perimeter = calculate_perimeter(hull)
33     print("Bao loi:", hull)
34     print("Chu vi bao loi:", perimeter)
```

2 Bài tập 2

2.1 Phân tích thuật toán

Để tính diện tích giao nhau giữa hai đa giác lồi, ta thực hiện các bước như sau:

1. **Xác định vùng giao nhau:**
   - Sử dụng thuật toán **Sutherland-Hodgman** hoặc **Weiler-Atherton** để cắt một đa giác bởi một đa giác khác. Các thuật toán này xác định các đoạn thẳng hoặc các điểm giao cắt, từ đó tìm được vùng giao nhau.
   - Do các đa giác đều lồi, vùng giao nhau cũng sẽ là một đa giác lồi.
2. **Tính diện tích của đa giác giao nhau:**
   - Khi đã biết các đỉnh của vùng giao nhau (theo thứ tự ngược chiều kim đồng hồ), diện tích được tính bằng công thức:

$$\text{Diện tích} = \frac{1}{2}\left|\sum_{i=1}^{k}(x_i y_{i+1} - y_i x_{i+1})\right|$$

     trong đó, $(x_{k+1}, y_{k+1}) = (x_1, y_1)$ để khép kín đa giác.
3. **Phân tích độ phức tạp:**
   - Nếu m và n là số đỉnh của hai đa giác, việc tìm giao điểm giữa hai đa giác có độ phức tạp $O(mn)$.
   - Sau khi tìm được vùng giao nhau, việc tính diện tích mất $O(k)$, với k là số đỉnh của đa giác giao.

2.2 Kĩ thuật áp dụng

- **Thuật toán hình học tính giao điểm:**
  - Dùng các phép tính vector và xác định giao điểm của các cạnh.
  - Sử dụng thuật toán cắt đa giác (**Sutherland-Hodgman**) để loại bỏ các điểm nằm ngoài một đa giác.
- **Tính diện tích đa giác:**
  - Dựa vào tích có hướng giữa các vector cạnh của đa giác.
- **Kĩ thuật tối ưu:**
  - Chỉ cần kiểm tra cạnh của một đa giác với cạnh của đa giác còn lại.
  - Lưu ý các điều kiện đặc biệt khi hai đa giác không giao nhau hoặc một đa giác nằm hoàn toàn trong đa giác kia.

2.3 Mã giả

Bước 1: Cắt đa giác để tìm giao nhau (Sutherland-Hodgman)

```
Input: Đa giác A với m đỉnh, đa giác B với n đỉnh
Output: Đa giác giao (nếu có)

function SutherlandHodgman(A, B):
    result = A  // Khởi tạo vùng giao ban đầu là toàn bộ đa giác A

    for mỗi cạnh e của đa giác B:
        temp = []  // Đa giác giao tạm thời
        for mỗi cạnh f của đa giác result:
            if f nằm trong B:
                temp.append(f)
            if e và f giao nhau tại điểm p:
                temp.append(p)
        result = temp

    return result
```

Bước 2: Tính diện tích đa giác

```
Input: Đa giác với k đỉnh {P1(x1, y1), P2(x2, y2), ..., Pk(xk, yk)}
Output: Diện tích của đa giác

function Area(Polygon):
    area = 0
    for i = 1 to k:
        x1, y1 = Polygon[i]
        x2, y2 = Polygon[(i % k) + 1]  // Điểm kế tiếp (khép kín)
        area += (x1 * y2 - y1 * x2)
    return abs(area) / 2
```

Tổng hợp

```
Input: Đa giác A và B
Output: Diện tích vùng giao

function IntersectionArea(A, B):
    PolygonIntersection = SutherlandHodgman(A, B)
    if PolygonIntersection is not empty:
        return Area(PolygonIntersection)
    else:
        return 0
```

2.4 Áp dụng thực tế

Khi giải bài toán này, cần lưu ý kiểm tra các trường hợp đặc biệt:

- Hai đa giác không giao nhau → Diện tích là 0.
- Một đa giác nằm hoàn toàn trong đa giác kia → Diện tích bằng diện tích đa giác nhỏ hơn.

2.5 Code

```python
def is_point_inside_polygon(point, polygon):
    """Kiem tra diem co nam trong da giac khong."""
    x, y = point
    n = len(polygon)
    inside = False

```

```
    for i in range(n):
        x1, y1 = polygon[i]
        x2, y2 = polygon[(i + 1) % n]

        if ((y1 > y) != (y2 > y)) and (x < (x2 - x1) *
           (y - y1) / (y2 - y1) + x1):
            inside = not inside

    return inside


def compute_intersection(p1, p2, q1, q2):
    """Tinh giao diem giua hai doan thang (neu co)."""
    def cross(v1, v2):
        return v1[0] * v2[1] - v1[1] * v2[0]

    d = (p2[0] - p1[0], p2[1] - p1[1])
    e = (q2[0] - q1[0], q2[1] - q1[1])
    dp = (q1[0] - p1[0], q1[1] - p1[1])

    det = cross(d, e)
    if abs(det) < 1e-10:  # Cac doan thang song song
       hoac dong nhat
        return None

    t = cross(dp, e) / det
    if 0 <= t <= 1:
        return (p1[0] + t * d[0], p1[1] + t * d[1])
    return None


def sutherland_hodgman_clip(subject_polygon,
   clip_polygon):
    """Cat mot da giac theo thuat toan
       Sutherland-Hodgman."""
    result = subject_polygon[:]

    for i in range(len(clip_polygon)):
        new_result = []
        c1 = clip_polygon[i]
        c2 = clip_polygon[(i + 1) % len(clip_polygon)]

        for j in range(len(result)):
```

```
            s = result[j]
            e = result[(j + 1) % len(result)]

            if is_point_inside_polygon(e, [c1, c2]):
                if not is_point_inside_polygon(s, [c1,
                    c2]):
                    new_result.append(compute_intersection(s,
                        e, c1, c2))
                new_result.append(e)
            elif is_point_inside_polygon(s, [c1, c2]):
                new_result.append(compute_intersection(s,
                    e, c1, c2))

        result = new_result

    return result


def polygon_area(polygon):
    """Tinh dien tich da giac."""
    n = len(polygon)
    area = 0

    for i in range(n):
        x1, y1 = polygon[i]
        x2, y2 = polygon[(i + 1) % n]
        area += x1 * y2 - y1 * x2

    return abs(area) / 2


def intersection_area(polygon1, polygon2):
    """Tinh dien tich giao nhau giua hai da giac."""
    intersection = sutherland_hodgman_clip(polygon1,
        polygon2)
    if not intersection or len(intersection) < 3:
        return 0
    return polygon_area(intersection)


# Vi du minh hoa:
polygon1 = [(0, 0), (4, 0), (4, 4), (0, 4)]  # Da giac
    loi dau tien
```

```
84 polygon2 = [(2, 2), (6, 2), (6, 6), (2, 6)]  # Da giac
       loi thu hai
85
86 area = intersection_area(polygon1, polygon2)
87 print(f"Dien tich giao nhau: {area}")
```