# Homework

Nguyen Pham Phuong Nam - 23520978
Pham Huynh Long Vu - 23521813

21 November 2024

## Problem 1: The Garden

### Problem Analysis

- **Convex Hull:** To find the shortest rope that encircles all the trees, it is necessary to determine the convex hull of the set of points on the 2D plane.
  - The convex hull is the smallest convex polygon enclosing all the points.
  - Algorithms like Graham Scan or Jarvis March can be used to compute the convex hull.
- **Perimeter Calculation:** After determining the convex hull, compute the total perimeter of the convex polygon by summing the distances between consecutive points on the hull.
- **Point Order:** The order of points on the convex hull is returned in a counterclockwise direction from the algorithm.

### Algorithm

- **Technique:** Use the Graham Scan algorithm, which is efficient with a time complexity of $O(n \log n)$.
- **Steps:**
  1. Sort all points by their $x$-coordinates and $y$-coordinates (if $x$-coordinates are equal).
  2. Construct the lower and upper hulls by iterating through the points and removing those that make the polygon non-convex.
  3. Merge the lower and upper hulls to get the convex hull.
  4. Calculate the total perimeter by summing the Euclidean distances between consecutive points on the hull.

### Pseudocode

```
FUNCTION convex_hull(points):
    SORT points by x-coordinate, and by y-coordinate if x is equal
    INITIALIZE lower = []  # Lower hull
    FOR point IN points:
        WHILE lower has at least 2 points AND the current point makes the hull non-convex:
            REMOVE the last point from lower
        ADD the current point to lower
    INITIALIZE upper = []  # Upper hull
    FOR point IN points in reverse order:
        WHILE upper has at least 2 points AND the current point makes the hull non-convex:
            REMOVE the last point from upper
        ADD the current point to upper
    REMOVE duplicate points at the junction of lower and upper
    RETURN lower + upper

FUNCTION distance(p1, p2):
    RETURN sqrt((p1.x - p2.x)^2 + (p1.y - p2.y)^2)

FUNCTION shortest_rope(points):
    hull = convex_hull(points)
    perimeter = 0
    FOR i IN 0 TO len(hull) - 1:
        perimeter += distance(hull[i], hull[(i + 1) % len(hull)])
    RETURN perimeter, hull

INPUT n
INPUT points
perimeter, hull = shortest_rope(points)
PRINT "Shortest rope length:", perimeter
PRINT "Points on the rope (counterclockwise):", hull
```

## Problem 2: Intersecting Gardens

### Problem Analysis

- **Intersection of Convex Polygons:**
  - To compute the area of the intersection of two convex polygons, use the Sutherland-Hodgman or Weiler-Atherton algorithm.
  - The algorithm works by clipping one polygon using the edges of the other polygon.
- **Area Calculation:**

– Once the intersection polygon is obtained, compute its area using the formula:

$$Area = \frac{1}{2} \left| \sum_{i=1}^{n} (x_i y_{i+1} - y_i x_{i+1}) \right|$$

where $(x_{n+1}, y_{n+1}) = (x_1, y_1)$ to close the polygon.

## Algorithm

- **Technique:**
  1. Use the Sutherland-Hodgman algorithm to clip one polygon against the edges of the other.
  2. Calculate the area of the resulting intersection polygon using the formula above.

## Pseudocode

```
FUNCTION polygon_area(polygon):
    area = 0
    n = number of points in polygon
    FOR i IN 0 TO n - 1:
        x1, y1 = polygon[i]
        x2, y2 = polygon[(i + 1) % n]
        area += x1 * y2 - y1 * x2
    RETURN ABS(area) / 2

FUNCTION inside(point, edge_start, edge_end):
    # Determine if a point is inside the half-plane defined by an edge
    RETURN (edge_end.x - edge_start.x) * (point.y - edge_start.y) >
           (edge_end.y - edge_start.y) * (point.x - edge_start.x)

FUNCTION intersection(edge_start1, edge_end1, edge_start2, edge_end2):
    # Compute the intersection point of two line segments using parameterization
    # Let p1 and p2 be endpoints of the first segment
    # Let p3 and p4 be endpoints of the second segment
    x1, y1 = edge_start1
    x2, y2 = edge_end1
    x3, y3 = edge_start2
    x4, y4 = edge_end2
    denom = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4)
    IF denom == 0:
        RETURN NULL # No intersection, lines are parallel or collinear
    t = ((x1 - x3) * (y3 - y4) - (y1 - y3) * (x3 - x4)) / denom
    u = ((x1 - x3) * (y1 - y2) - (y1 - y3) * (x1 - x2)) / denom
    IF 0 <= t <= 1 AND 0 <= u <= 1:
        intersection_point = (x1 + t * (x2 - x1), y1 + t * (y2 - y1))
        RETURN intersection_point
    ELSE:
        RETURN NULL

FUNCTION sutherland_hodgman(subject_polygon, clip_polygon):
    output_list = subject_polygon
    FOR edge IN clip_polygon:
        input_list = output_list
        output_list = []
        s = input_list[-1] # Start with the last point in the input list
        FOR e IN input_list:
            IF inside(e, edge.start, edge.end):
                IF NOT inside(s, edge.start, edge.end):
                    output_list.ADD(intersection(edge.start, edge.end, s, e))
                output_list.ADD(e)
            ELSE IF inside(s, edge.start, edge.end):
                output_list.ADD(intersection(edge.start, edge.end, s, e))
            s = e
    RETURN output_list

INPUT polygon_A, polygon_B
intersection_polygon = sutherland_hodgman(polygon_A, polygon_B)
intersection_polygon = sutherland_hodgman(intersection_polygon, polygon_B)

area = polygon_area(intersection_polygon)
OUTPUT area
```