Mason Legere
230128712
mlegere@unbc.ca

Assignment 5
CPSC 482

2021-04-04

Given a convex polygon with integer coordinates specified in a clock-wise order and a line provided in the format $ax + by = c$ this algorithm determines the minimum distance between the line and a vertex of the polygon. The idea behind this algorithm is based starting with an arbitrary vertex in the polygon and then performing binary search to find the minimum at each point. The distance between an arbitrary point $x = (x_0, y_0) \in \mathbb{R}^2$ is given by

$$\frac{|ax_0 + by_0 - c|}{\sqrt{a^2 + b^2}}$$

and is used throughout this algorithm. Within this algorithm, the application of the procedure is referred to as `dist(·,·)`. Further we also make use of a sub-protocol `brute_force_min_distance` that iterates through the points of polygon one by one and returns the minimum distance. We only use this at the end of the algorithm for a finite number of points.

Algorithm 1:

Result: The minimum distance between P and L
$P \leftarrow$ Convex polygon with integer coordinates ;
$L \leftarrow$ Line in $\mathbb{R}^2$;
$n \leftarrow$ Number of vertices of P ;
$i \leftarrow 0$;
skip_size $\leftarrow$ `ceil`$(n/2)$;
while *step_size* > 1 **do**
    right_dist $\leftarrow$ `dist`$(P$`[(`$i + 1$`) mod` n`],` $L)$;
    left_dist $\leftarrow$ `dist`$(P$`[(`$i - 1$`) mod` n`],` $L)$;
    if *left_dist* $<$ *right_dist* **then**
        $i \leftarrow$ ($i-$skip_size) mod n;
    else
        $i \leftarrow$ ($i-$skip_size) mod n;
    end
    skip_size = `int`(skip_size)$/2$;
end
return `brute_force_min_distance`(P[$i - 3$:$i + 2$],L)

As the assignment does not ask directly for correctness we will "prove" it in an informal manner. We note that because the polygon is convex if we were to index the vertices by their x-coordinate and map these indexes to their respective distance to the line we obtain a function which, following a rotation (with respect shifting of the indices to positive/negative) that such a function is either strictly increasing/decreasing. Consequently we can simply traverse down the side. As we do not know the shift needed for this to occur in practice we can instead use binary search allowing for "wrap around" of the polygon. As our main condition is that the *step_size* is larger than 1 and at each stage we divide by 2, this is clearly a $\mathcal{O}(log(n))$ algorithm, where n has the same meaning as that in the algorithm.