Name: Dat Quoc Ngo

NET-ID: dqn170000

Homework 2

Every triangulation has *f* faces that *N* interior (bounded) faces and 1 outer (unbounded) face. Since the triangulation has only face triangles, *f = t + 1.*

By observation, every triangle has 3 edges. Hence, *3t = 2e -h*. 3t is the total count of triangle edges that includes every edge twice except the outer face.

Given *v* and *h*,

* 2 = v - e + f = v - e + t + 1
  + e = v + t – 1 (1)
* Substitute (1) into Euler’s formula,
  + 3t = 2(v + t - 1) – h
  + t = 2v - h – 2 (2)
* Substitute (2) back to (1)
  + e = v + 2v - h - 2 – 1
  + e = 3v - h – 3

Every quadrangulation has *f* faces that *N* interior (bounded) faces and 1 outer (unbounded) face. Since the quadrangulation has only face quads, *f = q + 1.*

By observation, every quad has 4 edges. Hence, *4q = 2e -h*. 3t is the total count of quad edges that includes every edge twice except the outer face.

Given *v* and *h*,

* 2 = v - e + f = v - e + q + 1
  + e = v + q - 1 (1)
* Substitute (1) into Euler’s formula,
  + 4q = 2(v + q - 1) – h
  + 2q = 2v - h – 2
  + q = v – h/2 -1 (2)
* Substitute (2) back to (1)
  + e = v + v – h/2 - 1 - 1
  + e = 2v – h/2 – 2

From b, we argue that every quadrangulation has *4q + h = 2e*. This implies that *h* must be even otherwise the formula is invalid mathematicallly. Hence, the quadrangulation wont exist if *h* is odd.

Given two sets P={p1, p2, …, pn} and Q={q1, q2, …., qn},

To find two “crossing tangents” l1: y = a1x – b1 and l2: y = a2x – b2, it is necessary to find 4 tangent points of P and Q. We can apply Linear Programming to optimize slope of lines l1 and l2.

L2 is determined when two tangent points on P and Q are optimized to be min and max points on P and Q. Hence, the direction of line l2 is optimally upward. Hence, we simply maximize the slope of l2:

* m2 = (yQ-yP) / (xQ – xP)
* Subject to :
  + a2xP >= b2
  + yQ >= yP => xQ >= xP

Similarly, l2 is determined when two tangent points on P and Q are optimized to be max and min points on P and Q. Hence, the direction of line l1 is optimally downward. Hence, we simply minimize the slope of l1:

* m1 = (yQ – yP) / (xQ – xP)
* Subject to: a2xQ >= b2 and xP >= xQ

Each line takes O(n) time that n is number of points in either P or Q. Total time is O(n+m)

We can apply Linear Programming to optimize 3 variables a, b, and c in the parabola formula y = a + bx – cx^2 and subject to constraints:

* y >= all yP(s) for xP in P
* and y <= all yQ(s) for xQ in Q

Again, the top constraint (set Q) and the bottom constraint (set P) takes O(n) time each. Hence, if there exists an algorithm to optimize the parabola formula in O(n+m) that n and m are number of points in P and Q.

Since the parabola arch is between bottom points of Q and top points of P, the polytope constraints by P and Q is unbounded. Hence, there exists infinite optimal solutions to find the cannon shoot.

Since we compute the square, each side is length-equal. The largest length side is lying between n halfplanes. Hence, to compute the square, we simply find the maximum side length. Since it is a square, we look at only top side defined by two points (x1, y1) and (x2, y2). Note that x1 = y1 and x2 = y2 due to the square property.

We can apply Linear Programming to compute the square by maximizing the distance between the two above points, c1x2 – c2x1 subject to constraints:

* x1 – y1 = 0 and y1 <= ax1 – b => (1-a)x1 <= -b
* x2 – y2 = 0 and y2 <= ax2 – b => (1-a)x2 <= -b

We apply Linear Programming on two points. Hence, the total time complexity is O(2n) = O(n).

Assume that endpoints share no common x and y coordinates.

We can treat the subdivision process as the trapezoidal map. Hence in the corresponding trapezoidal map, each segment is a vertical line shot from a point to the top edge of the bounding rectangle; each wall is a horizontal line (bounded by two segments or by two vertical edges of the bounding rectangle). The left endpoint in the trapezoidal map is the bottom point in the given subdivision map.

Hence,

* Each point generates 4 vertices in the map, two for two intersection points of each wall intersecting with segments or vertical edges of the rectangle, two for two endpoints of each segment from a point. And 4 vertices of the bounding rectangle. Hence, 4 + 4n in total.
* Each point generates 2n + 1 boxes, since the bottom endpoint of each segment generate two boxs from right-above and left-above rectangles. There is one more rectangle, named the bottommost one. Unlike each segment in the trapezoidal map, every segment in this subdivision has their top endpoint lying on the top edge of the bounding rectangle, and (hence) yielding no more box.
* There are 4 + 8n + 4 = 8n + 8 edges. We can easily observe that no rectangle shares common edges with other rectangles.

Hence, the subdivision has O(4 + 4n + 8n + 8 + 2n + 1) = O(14n + 13) = O(n) size.

From 3a, the subdivision size is O(n) that ith point insertion results into new O(i) boxes.

Also, we convert the above subdivision problem into a trapezoidal map problem that each point insertion is equivalent to a segment insertion. Hence, each ith point/segment has an equal probability 1/I of being the last one to have been added. Following the trapezoidal map, we let δ(∆, s) = 1 if segment ∆ box depends on s segment, and 0 otherwise, then the expected value is

Diagram

Description automatically generated

Then, we swap the order of two summations and count the number of segments upon which each box depends on. A picture containing arrow

Description automatically generated

Note that each box is denoted Ti.

Since each box is bounded by at most four sides. The left and right sides are each determined by a segment. If either of these was the last to be added, then this box would exist. The top and bottom are each determined by two points or walls of two points. If either of these was the last to be added, then this box would exist.

Briefly, 

Hence, E[ki] <= 1/I \* sigma(each box in subdivision) (4) = 4/i \* (num. of boxes) = 4/i \* (2i + 1) = O(1)