#Assignment: asg7

#Name: Satish Dodda

#ULID: sdodda1

#Secret Directory: behappy

**Methods Descriptions:**

**@point1 is a tuple with two points which contains (x,y) coordinates**

**@point2 is a tuple with two points which contains (x,y) coordinates**

**distance(point1, point2)**

* here it will compute the Euclidean distance between two points.
* point1 is the coordinates of the first point (x, y)
* point2 is the coordinates of the second point (x, y)
* Using the formula sqrt((x\_2 - x\_1)^2 + (y\_2 - y\_1)^2) ​ to calculate the distance between two points
* it will return the float value it will represent the distance between point1 and point2.

**@polygon\_vertices it will contain the list of points**

**polygon\_perimeter(polygon\_vertices)**

calculates the perimeter of a polygon by adding the distances between consecutive vertices. and it will done by using a for loop, and it will return the total perimeter as a float number

**@origin is a tuple with two points which contains (x,y) coordinates**

**@point\_a is a tuple with two points which contains(x,y) coordinates**

**@point\_b is a tuple with two points which contains(x,y) coordinates**

**cross\_product(origin, point\_a,point\_b):**

cross\_product method will find the cross product of vectors formed by three points, and it will return a float value indicating the orientation, positive indicates counterclockwise, negative indicates clockwise, and zero indicates collinear points.

**@polygon\_vertices is a list of tuples and each tuple contains (x,y) coordinates represent the polygon vertices**

**is\_convex\_polygon(polygon\_vertices):**

this method will check whether it is a convex polygon or not , it finds cross\_product on each set of three consecutive points to check for consistent orientation if the direction of the turn changes, the polygon is non-convex.

**Example images : these images are generated by me by using python program**

**A graph of a triangle

Description automatically generated**

**Above image is an example of a convex polygon here no edge is over lapping with other edge so it is a convex polygon**

**A graph of a graph of a graph

Description automatically generated with medium confidence**

**This Image is not a convex polygon here v7 and v12 is crossing with each other so it is not at a convex polygon**

**@polygon\_vertices is a list of points representing the polygon vertices**

**@index\_i,index\_j,index\_k indices of the three points forming the triangle**

**compute\_triangle\_perimeter(polygon\_vertices , index\_i, index\_j,index\_k)**

this method will compute the perimeter of the triangle formed by three points within the polygon and will calculate the perimeter by finding the distance between these three points, and it will return a floating point perimeter

**@polygon\_vertices a list of points representing the polygon vertices**

**@track is a 2d list ,this is used to store the dividing points for the triangulation**

**compute\_chord\_sum(polygon\_vertices, track):**

this method uses the add chords method as a helper method, and it uses the tractable and tracks all unique chords from the minimum triangulation it will also return the sum of their lengths

@polygon\_vertices is a list of points representing the polygon vertices

**compute\_minimum\_triangulation(polygon\_vertices)**

this method uses dynamic programming to calculate minimum triangulation costs by dividing the polygon into smaller segments and finding the lowest-cost triangulation for each segment. It also finds the chords sum for the convex polygon and it will return a pair containing total triangulation cost and list of chords

**@polygon\_number is basically a test case number**

**@trianuglation\_weight the optimal triangulation weight**

**@chords is a list of chords in triangulation**

**displayOutput(polygon\_number , triangulation\_weight,chords):**

displays the triangulation results and min\_cost for forming the triangulation

@file\_path it is a file number

**read\_polygons(file\_path)**

reads each polygon from the file, separates polygons by "--", and for each convex polygon, calculates and displays the triangulation results.

**Algorithm:**

1. **File Reading and Polygon Setup:**

reads the file and separates polygons when ever it encounters ‘—' in line

else stores each polygon as a list of vertices in the format (x, y)

1. **Checking is it a convex polygon or not :**

for each polygon, is\_convex\_polygon checks if it is a convex polygon or not and determines convexity by analyzing the direction of turns between triplets of consecutive points using cross-product

1. **Dynamic Programming for Minimum Triangulation:**

for adjacent vertices, no triangulation is neededit is our base case here

it will use compute\_minimum\_triangulation to calculate the minimum cost of triangulation by dividing the polygon into smaller sections and fill a dynamic programming table dp with triangulation costs and a backtracking table track to record dividing points

it will calculates the cost of forming a triangle with each dividing point k and updates dp with the minimum cost

finally it adds the polygons perimeter to the total triangulation cost and uses compute\_chord\_sum to get the sum of chords

1. **Chords and Display of Results:**

It will use the compute\_chord\_sum method to reconstruct chords from the backtracking table track.

It will use displayOutput to show the triangulation weight, number of chords, and details of each chord for convex polygons

1. **Handling Non-Convex Polygons:**

If polygons identified as non-convex polygon by is\_convex\_polygon, outputs a message indicating they are not convex and skips the triangulation calculation

1. **Final Output:**

It will process all polygons one by one and display the triangulation results for each convex polygon

**Data Structures Used:**

vertices: is a list to store the vertices points of each polygon

dp : is a 2d list it is used to store the minimum triangulation cost between pairs of vertices

track: is a 2d list for backtracking that records dividing points in triangulation

chords: is a list that stores the unique chords as pairs of vertices representing internal diagonals in the triangulation

distince\_chords: is a set that is used to prevent the duplicates of chords

**Time Complexity:**

reading all lines in the file and storing points is O(n)

checking convexity takes O(m) for each polygon with m vertices

filling the dp table is O(m^3)because it iterates over vertex triplets to calculate minimum triangulation costs

calculating chords based on the track table is O(m^2)

max among these time complexity is fill filling the dp table , here Total time complexity will O(m^3)

**Space Complexity:**

Here max maximum space is required for both track and dp tables that is O(m^2) ,so total space complexity will O(m^2)

**Problems Faced:**

**While implementing the algorithm, I face some issues with the code logic , so I used print statements to debug the code and developed an python gui program to generate and display the polygon , It helped me to debug and code and I did changes in the code accordingly and I have spent some time to learn math like vector and sclars , finding the cross product**