Geometric Algorithms

[**Learn more about Geometric Algorithms in DSA Self Paced Course**](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=geeksforgeeks&utm_medium=articles+geometric_lp+header_link_click&utm_campaign=dsa+course+tracker)

[**Practice Problems on Geometric Algorithms**](https://practice.geeksforgeeks.org/topics/geometry/?utm_source=geeksforgeeks&utm_medium=articles+geometric_lp+header_link_click&utm_campaign=practice+tracker)

[**Recent Articles on Geometric Algorithms**](https://www.geeksforgeeks.org/category/algorithm/geometric/)

These algorithms are designed to solve Geometric Problems. They require in-depth knowledge of different mathematical subjects like combinatorics, topology, algebra, differential geometry, etc.

**For Example**: [Comparing Slopes of two lines](https://www.geeksforgeeks.org/program-find-slope-line/), [Finding Equation of a plane](https://www.geeksforgeeks.org/program-to-find-equation-of-a-plane-passing-through-3-points/) etc.

**Topics:**

* [Pattern Printing](https://www.geeksforgeeks.org/geometric-algorithms/#pattern)
* [Lines](https://www.geeksforgeeks.org/geometric-algorithms/#lines)
* [Triangle](https://www.geeksforgeeks.org/geometric-algorithms/#triangle)
* [Rectangle | Square | Circle](https://www.geeksforgeeks.org/geometric-algorithms/#rectangle)
* [Quadilateral](https://www.geeksforgeeks.org/geometric-algorithms/#quadrilateral)
* [3D Objects](https://www.geeksforgeeks.org/geometric-algorithms/#3D)
* [Polygon & Convex Hull](https://www.geeksforgeeks.org/geometric-algorithms/#polygon)
* [Problems on Geometric Algorithm](https://www.geeksforgeeks.org/geometric-algorithms/#standard)

**Pattern Printing:**

1. [Print lower triangle with alternate ‘\*’ and ‘#’](https://www.geeksforgeeks.org/print-lower-triangle-with-alternate-and/)
2. [Print the pattern 1\*2\*5\*6 –3\*4](https://www.geeksforgeeks.org/print-the-pattern-1256-34/)
3. [Python Program to print the pattern ‘G’](https://www.geeksforgeeks.org/python-program-to-print-the-pattern-g/)
4. [Pascal’s Triangle](https://www.geeksforgeeks.org/pascal-triangle/)
5. [Program to print pyramid pattern](https://www.geeksforgeeks.org/program-to-print-pyramid-pattern/)
6. [Program to print the diamond shape](https://www.geeksforgeeks.org/program-print-diamond-shape/)
7. [Hour-glass Pattern](https://www.geeksforgeeks.org/hour-glass-pattern/)
8. [Program to print V and inverted-V pattern](https://www.geeksforgeeks.org/program-for-inverted-v-pattern/)
9. [Program to print hollow pyramid, diamond pattern and their modifications](https://www.geeksforgeeks.org/program-print-hollow-pyramid-diamond-pattern/)
10. [Code to Generate the Map of India (With Explanation)](https://www.geeksforgeeks.org/code-to-generate-the-map-of-india-with-explanation/)

**Lines:**

1. [Program to find the mid-point of a line](https://www.geeksforgeeks.org/program-find-mid-point-line/)
2. [Section formula (Point that divides a line in given ratio)](https://www.geeksforgeeks.org/section-formula-point-divides-line-given-ratio/)
3. [Program to find slope of a line](https://www.geeksforgeeks.org/program-find-slope-line/)
4. [Program to find line passing through 2 Points](https://www.geeksforgeeks.org/program-find-line-passing-2-points/)
5. [Program for Point of Intersection of Two Lines](https://www.geeksforgeeks.org/program-for-point-of-intersection-of-two-lines/)
6. [Reflection of a point about a line in C++](https://www.geeksforgeeks.org/reflection-point-line-c/)
7. [Find points at a given distance on a line of given slope](https://www.geeksforgeeks.org/find-points-at-a-given-distance-on-a-line-of-given-slope/)
8. [Given n line segments, find if any two segments intersect](https://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/)
9. [Count maximum points on same line](https://www.geeksforgeeks.org/count-maximum-points-on-same-line/)
10. [Minimum lines to cover all points](https://www.geeksforgeeks.org/minimum-lines-cover-points/)

**Triangle:**

1. [Check whether triangle is valid or not if sides are given](https://www.geeksforgeeks.org/check-whether-triangle-valid-not-sides-given/)
2. [Check whether a given point lies inside a triangle or not](https://www.geeksforgeeks.org/check-whether-a-given-point-lies-inside-a-triangle-or-not/)
3. [C program to find area of a triangle](https://www.geeksforgeeks.org/c-program-find-area-triangle/)
4. [Check if right triangle possible from given area and hypotenuse](https://www.geeksforgeeks.org/check-right-angles-possible-given-area-hypotenuse/)
5. [Count Integral points inside a Triangle](https://www.geeksforgeeks.org/count-integral-points-inside-a-triangle/)
6. [Find all angles of a given triangle](https://www.geeksforgeeks.org/find-angles-given-triangle/)
7. [Program to find Circumcenter of a Triangle](https://www.geeksforgeeks.org/program-find-circumcenter-triangle-2/)
8. [Number of Triangles that can be formed given a set of lines in Euclidean Plane](https://www.geeksforgeeks.org/number-triangles-can-formed-given-set-lines-euclidean-plane/)
9. [Program to calculate area of Circumcircle of an Equilateral Triangle](https://www.geeksforgeeks.org/program-calculate-area-circumcircle-equilateral-triangle/)
10. [Program to find third side of triangle using law of cosines](https://www.geeksforgeeks.org/program-find-third-side-triangle-using-law-cosines/)

**Rectangle | Square | Circle:**

1. [Check whether a given point lies inside a rectangle or not](https://www.geeksforgeeks.org/check-whether-given-point-lies-inside-rectangle-not/)
2. [Program for Area And Perimeter Of Rectangle](https://www.geeksforgeeks.org/program-area-perimeter-rectangle/)
3. [Find if two rectangles overlap](https://www.geeksforgeeks.org/find-two-rectangles-overlap/)
4. [How to check if given four points form a square](https://www.geeksforgeeks.org/check-given-four-points-form-square/)
5. [Number of rectangles in N\*M grid](https://www.geeksforgeeks.org/number-rectangles-nm-grid/)
6. [Program to find Circumference of a Circle](https://www.geeksforgeeks.org/program-find-circumference-circle/)
7. [Program to find area of a circle](https://www.geeksforgeeks.org/c-program-find-area-circle/)
8. [Check whether a point exists in circle sector or not](https://www.geeksforgeeks.org/check-whether-point-exists-circle-sector-not/)
9. [Check if two given circles touch or intersect each other](https://www.geeksforgeeks.org/check-two-given-circles-touch-intersect/)
10. [Program to find area of a Circular Segment](https://www.geeksforgeeks.org/program-find-area-circular-segment/)
11. [Check if a line touches or intersects a circle](https://www.geeksforgeeks.org/check-line-touches-intersects-circle/)
12. [Area of a Circumscribed Circle of a Square](https://www.geeksforgeeks.org/area-circumscribed-circle-square/)
13. [Find minimum radius such that atleast k point lie inside the circle](https://www.geeksforgeeks.org/find-minimum-radius-atleast-k-point-lie-inside-circle/)
14. [Angular Sweep (Maximum points that can be enclosed in a circle of given radius)](https://www.geeksforgeeks.org/angular-sweep-maximum-points-can-enclosed-circle-given-radius/)

**Quadrilateral:**

1. [Number of parallelograms when n horizontal parallel lines intersect m vertical parallellines](https://www.geeksforgeeks.org/number-of-parallelograms-when-n-horizontal-parallel-lines-intersect-m-vertical-parallellines/)
2. [Program for Circumference of a Parallelogram](https://www.geeksforgeeks.org/program-circumference-parallelogram/)
3. [Program to calculate area and perimeter of Trapezium](https://www.geeksforgeeks.org/program-calculate-area-perimeter-trapezium/)
4. [Program to find area of a Trapezoid](https://www.geeksforgeeks.org/program-find-area-trapezoid/)
5. [Find all possible coordinates of parallelogram](https://www.geeksforgeeks.org/find-possible-coordinates-parallelogram/)
6. [Maximum area of quadrilateral](https://www.geeksforgeeks.org/maximum-area-quadrilateral/)
7. [Check whether four points make a parallelogram](https://www.geeksforgeeks.org/check-whether-four-points-make-parallelogram/)
8. [Find the Missing Point of Parallelogram](https://www.geeksforgeeks.org/find-missing-point-parallelogram/)

**3D Objects:**

1. [Find the perimeter of a cylinder](https://www.geeksforgeeks.org/find-perimeter-cylinder/)
2. [Program for Volume and Surface area of Frustum of Cone](https://www.geeksforgeeks.org/program-for-volume-and-surface-area-of-frustum-of-cone/)
3. [Program to calculate volume of Ellipsoid](https://www.geeksforgeeks.org/program-calculate-volume-ellipsoid/)
4. [Program for volume of Pyramid](https://www.geeksforgeeks.org/program-for-volume-of-pyramid/)
5. [Calculate volume and surface area of a cone](https://www.geeksforgeeks.org/calculate-volume-surface-area-cone/)
6. [Calculate Volume and Surface area Of Sphere](https://www.geeksforgeeks.org/calculate-volume-surface-area-sphere/)
7. [Program for Volume and Surface Area of Cuboid](https://www.geeksforgeeks.org/program-for-volume-and-surface-area-of-cuboid/)
8. [Program for Volume and Surface Area of Cube](https://www.geeksforgeeks.org/program-volume-surface-area-cube/)
9. [Pythagorean Quadruple](https://www.geeksforgeeks.org/pythagorean-quadruple/)
10. [LS3/NS3 sphere generation algorithm and its implementation](https://www.geeksforgeeks.org/ls3ns3-sphere-generation-algorithm-implementation/)

**Polygon and Convex Hull:**

1. [How to check if a given point lies inside or outside a polygon?](https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/)
2. [Area of a polygon with given n ordered vertices](https://www.geeksforgeeks.org/area-of-a-polygon-with-given-n-ordered-vertices/)
3. [Tangents between two Convex Polygons](https://www.geeksforgeeks.org/tangents-two-convex-polygons/)
4. [Find number of diagonals in n sided convex polygon](https://www.geeksforgeeks.org/find-number-diagonals-n-sided-convex-polygon/)
5. [Convex Hull using Jarvis’ Algorithm or Wrapping](https://www.geeksforgeeks.org/convex-hull-using-jarvis-algorithm-or-wrapping/)
6. [Quickhull Algorithm for Convex Hull](https://www.geeksforgeeks.org/quickhull-algorithm-convex-hull/)
7. [Deleting points from Convex Hull](https://www.geeksforgeeks.org/deleting-points-convex-hull/)
8. [Minimum area of a Polygon with three points given](https://www.geeksforgeeks.org/minimum-area-polygon-three-points-given/)

**Standard Problems on Geometric Algorithm:**

1. [Finding the vertex, focus and directrix of a parabola](https://www.geeksforgeeks.org/finding-vertex-focus-directrix-parabola/)
2. [Find Simple Closed Path for a given set of points](https://www.geeksforgeeks.org/find-simple-closed-path-for-a-given-set-of-points/)
3. [Number of Integral Points between Two Points](https://www.geeksforgeeks.org/number-integral-points-two-points/)
4. [Optimum location of point to minimize total distance](https://www.geeksforgeeks.org/optimum-location-point-minimize-total-distance/)
5. [Find perimeter of shapes formed with 1s in binary matrix](https://www.geeksforgeeks.org/find-perimeter-shapes-formed-1s-binary-matrix/)
6. [Draw geometric shapes on images using OpenCV](https://www.geeksforgeeks.org/draw-geometric-shapes-images-using-opencv/)
7. [Find if it’s possible to rotate the page by an angle or not](https://www.geeksforgeeks.org/find-possible-rotate-page-angle-not/)
8. [Equable Shapes](https://www.geeksforgeeks.org/equable-shapes/)

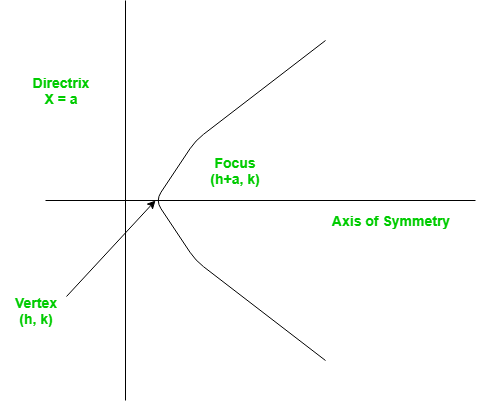
**Easy Questions:**

**Finding the vertex, focus and directrix of a parabola**

**Problem –** Find the vertex, focus and directrix of a parabola when the coefficients of its equation are given.

A set of points on a plain surface that forms a curve such that any point on that curve is equidistant from the focus is a **parabola.**

**Vertex** of a parabola is the coordinate from which it takes the sharpest turn whereas a is the straight line used to generate the curve.



The standard form of a parabola equation is

. Given the values of a, b and c; our task is to find the coordinates of vertex, focus and the equation of the directrix.

**Example –**

Input : 5 3 2  
Output : Vertex:(-0.3, 1.55)  
 Focus: (-0.3, 1.6)  
 Directrix: y=-198  
Consult the formula below for explanation.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

This problem is a simple example of implementations of formulae. Given below are the required set of formulae which will help us tackle the problem.

For a parabola in the form

Vertex:

Focus:

Directrix:

# Function to calculate Vertex,

# Focus and Directrix

**def** parabola(a, b, c):

**print**("Vertex: (" , (**-**b **/** (2 **\*** a)),

        ", ", (((4 **\*** a **\*** c) **-** (b **\*** b))

**/** (4 **\*** a)), ")", sep **=** "")

**print**("Focus: (" , (**-**b **/** (2 **\*** a)),

    ", ", (((4 **\*** a **\*** c) **-** (b **\*** b) **+** 1)

**/** (4 **\*** a)), ")", sep **=** "")

    print("Directrix: y=", c **-** ((b **\*** b)

**+** 1) **\*** 4 **\*** a, sep **=** "")

# Driver Function

a **=** 5

b **=** 3

c **=** 2

parabola(a, b, c)

# This code is contributed by Smitha.

**Output –**

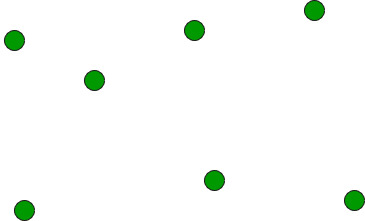
Vertex:(-0.3, 1.55)  
Focus: (-0.3, 1.6)  
Directrix: y=-198

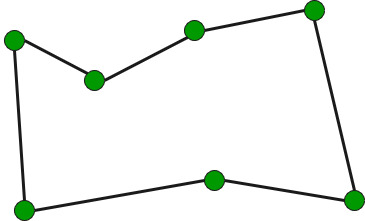
**Time Complexity:**O(1)

**Auxiliary Space:**O(1)

**Find Simple Closed Path for a given set of points**

Given a set of points, connect the dots without crossing.





**Example:**

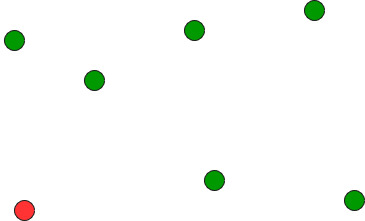
Input: points[] = {(0, 3), (1, 1), (2, 2), (4, 4),  
 (0, 0), (1, 2), (3, 1}, {3, 3}};

Output: Connecting points in following order would  
 not cause any crossing  
 {(0, 0), (3, 1), (1, 1), (2, 2), (3, 3),  
 (4, 4), (1, 2), (0, 3)}

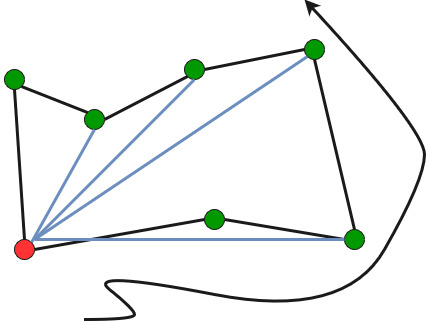
**We strongly recommend you to minimize your browser and try this yourself first.**

The idea is to use sorting.

* Find the bottom-most point by comparing y coordinate of all points. If there are two points with same y value, then the point with smaller x coordinate value is considered. Put the bottom-most point at first position.



* Consider the remaining n-1 points and sort them by polar angle in counterclockwise order around points[0]. If polar angle of two points is same, then put the nearest point first.
* Traversing the sorted array (sorted in increasing order of angle) yields simple closed path.



**How to compute angles?**

One solution is to use trigonometric functions.

Observation: We don’t care about the actual values of the angles. We just want to sort by angle.

Idea: Use the [orientation](https://www.geeksforgeeks.org/orientation-3-ordered-points/) to compare angles without actually computing them!

**Number of Integral Points between Two Points**

Given two points **p** (x1, y1) and **q**(x2, y2), calculate the number of integral points lying on the line joining them.

**Example:** If points are (0, 2) and (4, 0), then the number of integral points lying on it is only one and that is (2, 1).

Similarly, if points are (1, 9) and (8, 16), the integral points lying on it are 6 and they are (2, 10), (3, 11), (4, 12), (5, 13), (6, 14) and (7, 15).

[We strongly recommend that you click here and practice it, before moving on to the solution.](https://practice.geeksforgeeks.org/problems/count-integral-points5445/1)

**Simple Approach**

Start from any of the given points, reach the other end point by using loops. For every point inside the loop, check if it lies on the line that joins given two points. If yes, then increment the count by 1. Time Complexity for this approach will be O(min(x2-x1, y2-y1)).

**Optimal Approach**

1. If the edge formed by joining **p** and **q** is parallel   
 to the X-axis, then the number of integral points   
 between the vertices is :   
 abs(p.y - q.y)-1

2. Similarly if edge is parallel to the Y-axis, then   
 the number of integral points in between is :  
 abs(p.x - q.x)-1

3. Else, we can find the integral points between the  
 vertices using below formula:  
 GCD(abs(p.x - q.x), abs(p.y - q.y)) - 1

**How does the GCD formula work?**

The idea is to find the equation of the line in simplest form, i.e., in equation ax + by +c, coefficients a, b and c become co-prime. We can do this by calculating the GCD (greatest common divisor) of a, b and c and convert a, b and c in the simplest form.

Then, the answer will be (difference of y coordinates) divided by (a) – 1. This is because after calculating ax + by + c = 0, for different y values, x will be number of y values which are exactly divisible by a.

Below is the implementation of above idea.

# Python3 code to find the number of

# integral points lying on the line

# joining the two given points

# Class to represent an Integral point

# on XY plane.

**class** Point:

**def** \_\_init\_\_(self, a, b):

        self.x **=** a

        self.y **=** b

# Utility function to find GCD

# of two numbers GCD of a and b

**def** gcd(a, b):

**if** b **==** 0:

**return** a

**return** gcd(b, a **%** b)

# Finds the no. of Integral points

# between two given points.

**def** getCount(p, q):

    # If line joining p and q is parallel

    # to x axis, then count is difference

    # of y values

**if** p.x **==** q.x:

**return** abs(p.y **-** q.y) **-** 1

    # If line joining p and q is parallel

    # to y axis, then count is difference

    # of x values

**if** p.y **==** q.y:

**return** abs(p.x **-** q.x) **-** 1

**return** gcd(abs(p.x **-** q.x),

               abs(p.y **-** q.y)) **-** 1

# Driver Code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    p **=** Point(1, 9)

    q **=** Point(8, 16)

    print("The number of integral points",

          "between ({}, {}) and ({}, {}) is {}" .

           format(p.x, p.y, q.x, q.y, getCount(p, q)))

# This code is contributed by Rituraj Jain

**Output:**

The number of integral points between (1, 9) and (8, 16) is 6

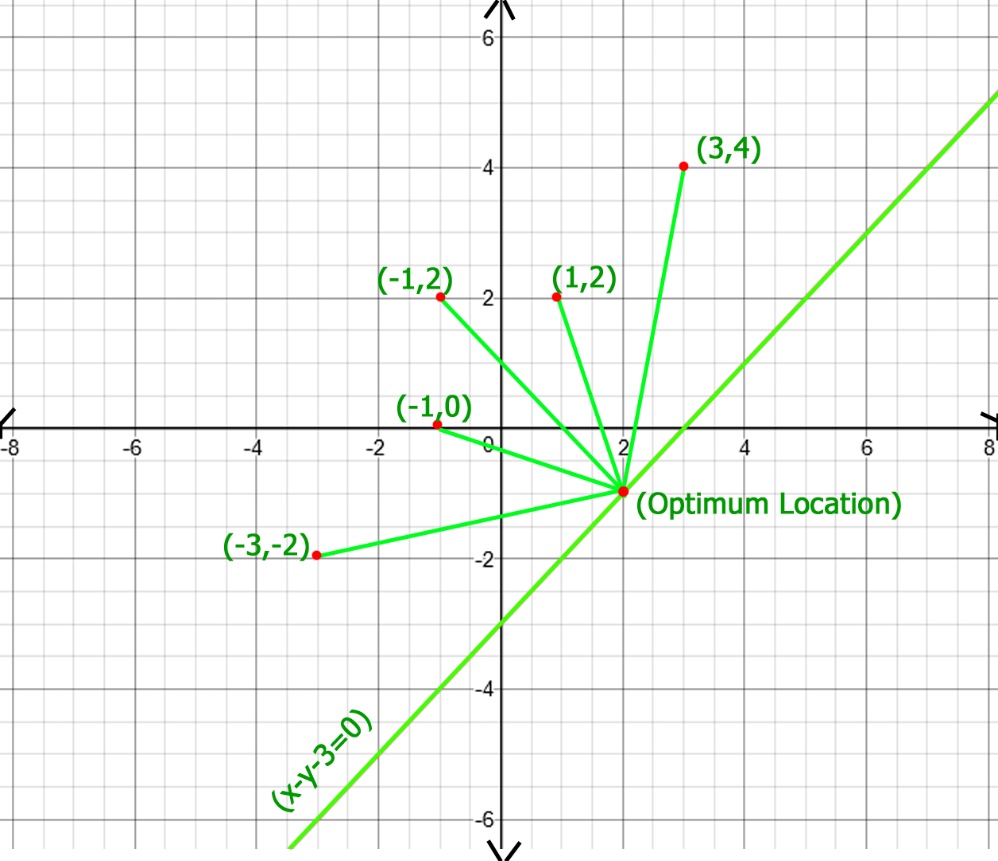
**Time Complexity:** O(log(min(a,b))), as we are using recursion to find the GCD.

**Auxiliary Space:** O(log(min(a,b))), for recursive stack space.

**Optimum location of point to minimize total distance**

Given a set of points as and a line as ax+by+c = 0. We need to find a point on given line for which sum of distances from given set of points is minimum.

**Example:**



In above figure optimum location of point of x - y - 3 = 0 line   
is (2, -1), whose total distance with other points is 20.77,   
which is minimum obtainable total distance.

Recommended Problem

Optimum location of point to minimize total distance

[Solve Problem](https://practice.geeksforgeeks.org/problems/optimum-location-of-point-to-minimize-total-distance/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 657

If we take one point on given line at infinite distance then total distance cost will be infinite, now when we move this point on line towards given points the total distance cost starts decreasing and after some time, it again starts increasing which reached to infinite on the other infinite end of line so distance cost curve looks like a U-curve and we have to find the bottom value of this U-curve.

As U-curve is not monotonically increasing or decreasing we can’t use binary search for finding bottom most point, here we will use ternary search for finding bottom most point, ternary search skips one third of search space at each iteration, you can read more about ternary search [here](https://en.wikipedia.org/wiki/Ternary_search).

So solution proceeds as follows, we start with low and high initialized as some smallest and largest values respectively, then we start iteration, in each iteration we calculate two mids, mid1 and mid2, which represent 1/3rd and 2/3rd position in search space, we calculate total distance of all points with mid1 and mid2 and update low or high by comparing these distance cost, this iteration continues until low and high become approximately equal.

# A Python3 program to find optimum location

# and total cost

**import** math

**class** Optimum\_distance:

    # Class defining a point

**class** Point:

**def** \_\_init\_\_(self, x, y):

            self.x **=** x

            self.y **=** y

    # Class defining a line of ax + by + c = 0 form

**class** Line:

**def** \_\_init\_\_(self, a, b, c):

            self.a **=** a

            self.b **=** b

            self.c **=** c

    # Method to get distance of point

    # (x, y) from point p

**def** dist(self, x, y, p):

**return** math.sqrt((x **-** p.x) **\*\*** 2 **+**

                         (y **-** p.y) **\*\*** 2)

    # Utility method to compute total distance

    # all points when choose point on given

    # line has x-coordinate value as X

**def** compute(self, p, n, l, x):

        res **=** 0

        y **= -**1 **\*** (l.a**\***x **+** l.c) **/** l.b

        # Calculating Y of chosen point

        # by line equation

**for** i **in** range(n):

            res **+=** self.dist(x, y, p[i])

**return** res

    # Utility method to find minimum total distance

**def** find\_Optimum\_cost\_untill(self, p, n, l):

        low **= -**1e6

        high **=** 1e6

        eps **=** 1e**-**6 **+** 1

        # Loop until difference between low

        # and high become less than EPS

**while**((high **-** low) > eps):

              # mid1 and mid2 are representative x

            # co-ordiantes of search space

            mid1 **=** low **+** (high **-** low) **/** 3

            mid2 **=** high **-** (high **-** low) **/** 3

            dist1 **=** self.compute(p, n, l, mid1)

            dist2 **=** self.compute(p, n, l, mid2)

            # If mid2 point gives more total

            # distance, skip third part

**if** (dist1 < dist2):

                high **=** mid2

            # If mid1 point gives more total

            # distance, skip first part

**else**:

                low **=** mid1

        # Compute optimum distance cost by

        # sending average of low and high as X

**return** self.compute(p, n, l, (low **+** high) **/** 2)

    # Method to find optimum cost

**def** find\_Optimum\_cost(self, p, l):

        n **=** len(p)

        p\_arr **=** [None] **\*** n

        # Converting 2D array input to point array

**for** i **in** range(n):

            p\_obj **=** self.Point(p[i][0], p[i][1])

            p\_arr[i] **=**  p\_obj

**return** self.find\_Optimum\_cost\_untill(p\_arr, n, l)

 # Driver Code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    obj **=** Optimum\_distance()

    l **=** obj.Line(1, **-**1, **-**3)

    p **=** [ [ **-**3, **-**2 ], [ **-**1, 0 ],

          [ **-**1, 2 ], [ 1, 2 ],

          [ 3, 4 ] ]

**print**(obj.find\_Optimum\_cost(p, l))

# This code is contributed by Sulu\_mufi

**Output**

20.7652

**Time Complexity:** O(n2)

**Auxiliary Space:** O(n)

**Find perimeter of shapes formed with 1s in binary matrix**

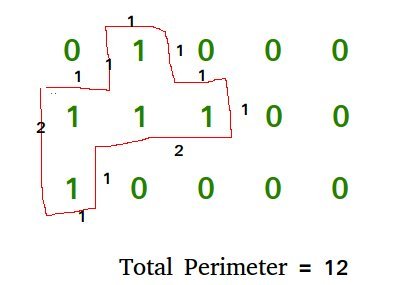
Given a matrix of **N** rows and **M** columns, consist of 0’s and 1’s. The task is to find the perimeter of subfigure consisting only 1’s in the matrix. Perimeter of single 1 is 4 as it can be covered from all 4 side. Perimeter of double 11 is 6.

| 1 | | 1 1 |

**Examples:**

**Input :** mat[][] =   
 {  
 1, 0,  
 1, 1,  
 }  
**Output :** 8  
Cell (1,0) and (1,1) making a L shape whose perimeter is 8.

**Input :**  mat[][] =   
 {   
 0, 1, 0, 0, 0,  
 1, 1, 1, 0, 0,  
 1, 0, 0, 0, 0  
 }  
**Output :** 12



Recommended Problem

Find perimeter of shapes

The idea is to traverse the matrix, find all ones and find their contribution in perimeter. The maximum contribution of a 1 is four if it is surrounded by all 0s. The contribution reduces by one with 1 around it.

Algorithm for solving this problem:

1. Traverse the whole matrix and find the cell having value equal to 1.
2. Calculate the number of closed side for that cell and add, 4 – number of closed side to the total perimeter.

Below is the implementation of this approach:

# Python3 program to find perimeter of area

# covered by 1 in 2D matrix consists of 0's and 1's.

R **=** 3

C **=** 5

# Find the number of covered side for mat[i][j].

**def** numofneighbour(mat, i, j):

    count **=** 0;

    # UP

**if** (i > 0 **and** mat[i **-** 1][j]):

        count**+=** 1;

    # LEFT

**if** (j > 0 **and** mat[i][j **-** 1]):

        count**+=** 1;

    # DOWN

**if** (i < R**-**1 **and** mat[i **+** 1][j]):

        count**+=** 1

    # RIGHT

**if** (j < C**-**1 **and** mat[i][j **+** 1]):

        count**+=** 1;

**return** count;

# Returns sum of perimeter of shapes formed with 1s

**def** findperimeter(mat):

    perimeter **=** 0;

    # Traversing the matrix and finding ones to

    # calculate their contribution.

**for** i **in** range(0, R):

**for** j **in** range(0, C):

**if** (mat[i][j]):

                perimeter **+=** (4 **-** numofneighbour(mat, i, j));

**return** perimeter;

# Driver Code

mat **=** [ [0, 1, 0, 0, 0],

        [1, 1, 1, 0, 0],

        [1, 0, 0, 0, 0] ]

**print**(findperimeter(mat), end**=**"\n");

# This code is contributed by Akanksha Rai

**Output**

12

**Time Complexity:** O(R x C).

**Auxiliary Space:**O(1), since no extra space has been taken.

**Draw geometric shapes on images using OpenCV**

[OpenCV](https://www.geeksforgeeks.org/set-opencv-anaconda-environment/)provides many drawing functions to draw geometric shapes and write text on images. Let’s see some of the drawing functions and draw geometric shapes on images using OpenCV.

Some of the drawing functions are :

***cv2.line() :****Used to draw line on an image.*

***cv2.rectangle() :****Used to draw rectangle on an image.*

***cv2.circle() :****Used to draw circle on an image.*

***cv2.putText() :****Used to write text on image.*

To demonstrate the uses of the above-mentioned functions we need an image of size 400 X 400 filled with a solid color (black in this case). Inorder to do this, We can utilize ***numpy.zeroes*** function to create the required image.

* Python3

# Python3 program to draw solid-colored

# image using numpy.zeroes() function

**import** numpy as np

**import** cv2

# Creating a black image with 3 channels

# RGB and unsigned int datatype

img **=** np.zeros((400, 400, 3), dtype **=** "uint8")

cv2.imshow('dark', img)

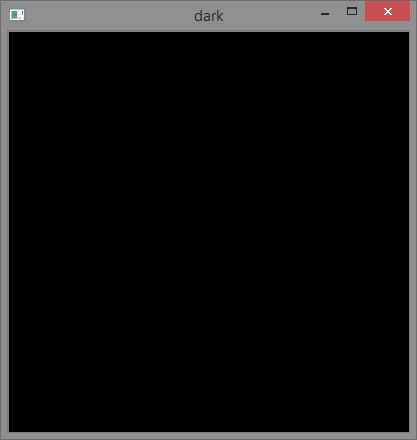
# Allows us to see image

# until closed forcefully

cv2.waitKey(0)

cv2.destroyAllWindows()

**Output :**



Now, let’s draw some geometric shapes on this solid black image.

**Draw a line :**

*cv2.line(imageObjectName, (‘start\_coordinates’), (‘end\_coordinates’), (‘color\_in\_bgr’), ‘line\_thickness’)*

* Python3

# Python3 program to draw line

# shape on solid image

**import** numpy as np

**import** cv2

# Creating a black image with 3 channels

# RGB and unsigned int datatype

img **=** np.zeros((400, 400, 3), dtype **=** "uint8")

# Creating line

cv2.line(img, (20, 160), (100, 160), (0, 0, 255), 10)

cv2.imshow('dark', img)

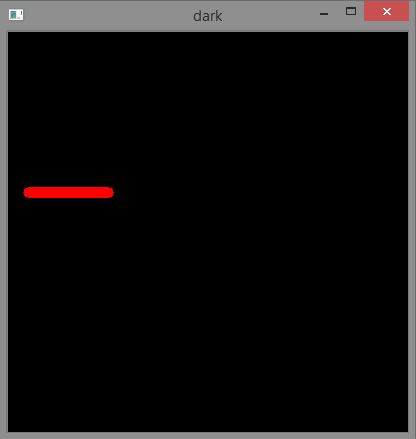
# Allows us to see image

# until closed forcefully

cv2.waitKey(0)

cv2.destroyAllWindows()

**Output :**



**Draw a rectangle :**

*cv2.rectangle(imageObjectName, (‘top\_left\_vertex\_coordinates’), (‘lower\_right\_vertex\_coordinates’), (‘stroke\_color\_in\_bgr’), ‘stroke\_thickness’)*

* Python3

# Python3 program to draw rectangle

# shape on solid image

**import** numpy as np

**import** cv2

# Creating a black image with 3

# channels RGB and unsigned int datatype

img **=** np.zeros((400, 400, 3), dtype **=** "uint8")

# Creating rectangle

cv2.rectangle(img, (30, 30), (300, 200), (0, 255, 0), 5)

cv2.imshow('dark', img)

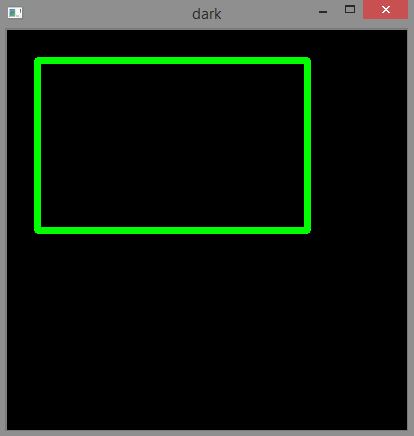
# Allows us to see image

# until closed forcefully

cv2.waitKey(0)

cv2.destroyAllWindows()

**Output :**



**Draw a Circle :**

*cv2.circle(imageObjectName, (‘center\_coordinates’), (‘circle\_radius’), (‘color\_in\_bgr’), ‘stroke\_thickness’)*

* Python3

# Python3 program to draw circle

# shape on solid image

**import** numpy as np

**import** cv2

# Creating a black image with 3

# channels RGB and unsigned int datatype

img **=** np.zeros((400, 400, 3), dtype **=** "uint8")

# Creating circle

cv2.circle(img, (200, 200), 80, (255, 0, 0), 3)

cv2.imshow('dark', img)

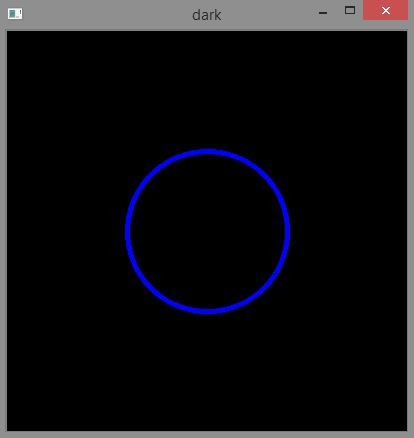
# Allows us to see image

# until closed forcefully

cv2.waitKey(0)

cv2.destroyAllWindows()

**Output :**



**Writing text :**

*cv2.putText(imageObjectName, ‘TextContent’, (‘text\_starting\_point\_coordinates’), ‘fontToBeUsed’, ‘font\_size’, (‘text\_color’, ‘text\_thickness’, ‘line\_type’)*

* Python

# Python3 program to write

# text on solid image

**import** numpy as np

**import** cv2

# Creating a black image with 3

# channels RGB and unsigned int datatype

img **=** np.zeros((400, 400, 3), dtype **=** "uint8")

# writing text

font **=** cv2.FONT\_HERSHEY\_SIMPLEX

cv2.putText(img, 'GeeksForGeeks', (50, 50),

            font, 0.8, (0, 255, 0), 2, cv2.LINE\_AA)

cv2.imshow('dark', img)

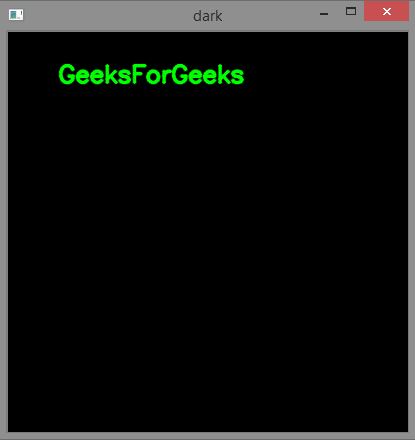
# Allows us to see image

# until closed forcefully

cv2.waitKey(0)

cv2.destroyAllWindows()

**Output :**



**Applications of drawing shapes on images :**

* Drawing geometrical shapes can help us highlight the particular portions of an image.
* Geometrical shapes like line can help us point or identify particular regions in image.
* Writing text on certain regions of images can add description to that region.

**Find if it’s possible to rotate the page by an angle or not.**

You are given three points a, b, c on a page. Find if it’s possible to rotate the page around the point by an angle, such that the new position of ‘a’ is same as the old position of ‘b’, and the new position of ‘b’ is same as the old position of ‘c’. If such angle exists print “Yes”, else “No”.

**Examples:**

Input : a1 = 0, a2 = 1, b1 = 1, b2 = 1,  
 c1 = 1, c2 = 0  
Output : Yes  
Explanation : Rotate the page by 90 degree.

Input : a1 = 1, a2 = 1, b1 = 0, b2 = 0,  
 c1 = 1000, c2 = 1000  
Output : No

Recommended Problem

Rotate Page

[Mathematical](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Mathematical&sortBy=submissions)

[Algorithms](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Algorithms&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/rotate-page0923/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 510

Rotation of page by some angle is only possible if the distance between points ‘a’ and ‘b’ is equal to distance between points ‘b’ and ‘c’. But if the points are on same line, there is no rotation at point ‘b’. The problem has no solution when ‘a’, ‘b’, ‘c’ are in the same line or**dis(a, b) != dis(b, c)**

# Python3 program to fill an

# array with frequencies.

# Function to find if it's possible

# to rotate page or not

**def** possibleOrNot(a1, a2, b1, b2, c1, c2):

    # Calculating distance b/w points

    dis1 **=** (pow(b1 **-** a1, 2) **+**

            pow(b2 **-** a2, 2))

    dis2 **=** (pow(c1 **-** b1, 2) **+**

            pow(c2 **-** b2, 2))

    # If distance is not equal

**if**(dis1 !**=** dis2):

        print("No")

    # If the points are in same line

**else if** (b1 **==** ((a1 **+** c1) **//** 2.0) **and**

          b2 **==** ((a2 **+** c2) **//** 2.0)):

        print("No")

**else**:

        print("Yes")

# Driver Code

# Points a, b, and c

a1, b1, c1 **=** 1, 2, 3

a2 **=** b2 **=** c2 **=** 0

possibleOrNot(a1, a2, b1, b2, c1, c2)

# This code is contributed by Anant Agarwal.

**Output:**

No

**Time Complexity:** O(logn)

**Auxiliary Space:** O(1)

**Equable Shapes**

A shape is equable if its area is equal to its perimeter. Given ordered coordinates of polygon find whether the shape is equable or not.

**Examples :**

Input : X[] = {0, 5, 0}  
 Y[] = {0, 0, 12}  
Output : Equable Shape

Input : X[] = {0, 4, 4, 0}  
 Y[] = {0, 0, 4, 4}  
Output : Equable Shape

Input: X[] = {0, 6, 6, 0}  
 Y[] = {0, 0, 4, 4}  
Output: Not Equable Shape

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We can find area of polygon using shoelace formula which is described in [Area of a polygon with given n ordered vertices](https://www.geeksforgeeks.org/area-of-a-polygon-with-given-n-ordered-vertices/). We can also find its perimeter simply by adding distances between adjacent points.

# Python 3 program to find equable shape

# To calculate area of polygon

**import** math

**def** polygonArea(X, Y, n):

    area **=** 0.0

    # Calculate value of area

    # using shoelace  formula

    j **=** n **-** 1

**for** i **in** range(n):

        area **+=** (X[j] **+** X[i]) **\*** (Y[j] **-** Y[i])

        # j is previous vertex to i

        j **=** i

**return** abs(area **/** 2.0)

# To calculate perimeter of polygon

**def** polygonPerimeter(X, Y, n):

    perimeter **=** 0.0

    # Calculate value of perimeter

    j **=** n **-** 1

**for** i **in** range(n):

        perimeter **+=** math.sqrt((X[j] **-** X[i]) **\*** (X[j] **-** X[i]) **+**

                          (Y[j] **-** Y[i]) **\*** (Y[j] **-** Y[i]))

        # j is previous vertex to i

        j **=** i

**return** perimeter

# To find equable shape

**def** equableShape(X, Y, n):

    # Find area and perimeter of polygon if

    # they are equal then it is equable shape

**if** (polygonPerimeter(X, Y, n) **==** polygonArea(X, Y, n)):

**print**("Equable Shape")

**else**:

**print**("Not Equable Shape")

#  Driver program to test above function

X **=** [ 0, 5, 0 ]

Y **=** [ 0, 0, 12 ]

n **=** len(X)

equableShape(X, Y, n)

# This code is contributed by Azkia Anam.

**Output :**

Equable Shape

**Time Complexity:** O(NlogN)

**Auxiliary Space:** O(N)

**Medium Questions:**

**Hard Questions:**