```python
# Authored By: Jayant Sogikar
import tkinter as tk
import math
import random


root = tk.Tk()


# root.geometry("300x300-1000-1000")
# root.state('zoomed')


root.title("Lines In Circle")

width = 500
height = 500
win = tk.Canvas(root,width=width,height=height)
win.pack()

# Experiment with flag at line 66, set it to any value between 0 and 2, can be random too, i like setting it to 0

class CustomLine:
    def __init__(self, fromPoint, toPoint,theta, angle, dist = float('inf')):
        self.fromPoint = fromPoint
        self.toPoint = toPoint
        self.theta = theta
        self.angle = angle
        self.dist = dist


    # Return true if line segments AB and CD intersect
    def intersect(self,line1,line2):
        A,B,C,D = line1[0],line1[1],line2[0],line2[1]
        def ccw(A,B,C):
            return (C[1]-A[1]) * (B[0]-A[0]) > (B[1]-A[1]) * (C[0]-A[0])
        return ccw(A,C,D) != ccw(B,C,D) and ccw(A,B,C) != ccw(A,B,D)


    def slope(self,P1, P2):
        # dy/dx
        # (y2 - y1) / (x2 - x1)
        if P2[0] - P1[0] == 0: return 0
        return(P2[1] - P1[1]) / (P2[0] - P1[0])

    def y_intercept(self,P1, slope):
        # y = mx + b
        # b = y - mx
        # b = P1[1] - slope * P1[0]
        return P1[1] - slope * P1[0]
    # Return the point of intersection
    def line_intersect(self,m1, b1, m2, b2):
        if m1 == m2:
            print ("These lines are parallel!!!")
            return None
        # y = mx + b
        # Set both lines equal to find the intersection point in the x direction
        # m1 * x + b1 = m2 * x + b2
        # m1 * x - m2 * x = b2 - b1
        # x * (m1 - m2) = b2 - b1
        # x = (b2 - b1) / (m1 - m2)
        x = (b2 - b1) / (m1 - m2)
        # Now solve for y -- use either line, because they are equal here
        # y = mx + b
        y = m1 * x + b1
        return x,y


    def update_ToPoint(self, radius):
        a = self.toPoint

        flag = 1
        flag = random.randint(0,1)


        x = a[0]
        y = a[1]

        if flag == 0: x += random.randint(radius - 50 , radius )
        elif flag == 1: y += random.randint(radius - 50 , radius )
        # else:
        #     x += new_dir
        #     y += new_dir




        self.toPoint = [x,y]




    def getDetails(self):
        return (self.fromPoint,self.toPoint)

def draw():
    # Set the radius
    radius = min(width,height) // 2 - 50
    # Create a circle
    win.create_oval((width//2) - radius,(height//2) - radius, (width//2) + radius,(height//2) + radius)

    # Specify lines to be drawn
    nLines = random.randint(300,600)
    angle = (2 * math.pi) / float(nLines)
    points = [x for x in range(nLines)]
    random.shuffle(points)
    lineAngles = points
    allLines = []
    for theta in lineAngles:
        # Point on the circle at an angle
        pos1 = [(math.cos(theta * angle) * (radius)) + width // 2, (math.sin(theta* angle) * (radius )) + height // 2]
        # Point at the center of the circle
        d = [(width // 2) ,(height // 2) ]
        # Create an object with the above vals
        line = CustomLine(pos1,d,theta, angle)
        # Radomly change the point
        line.update_ToPoint(radius)
        # Append the object to a list
        allLines.append(line)
    drewLines = []

    def checkIfOutCircle(x,center_x,y,center_y,radius):
        return ((x - center_x)**2) + ((y - center_y)**2) > radius**2
    # We now have to check if the line to be drawn intersects any other pre-drawn lines or not
    # If it does, then we reset to the toPoint in it to the point of intersection
    for i in range(len(allLines)):

        line1 = [[allLines[i].fromPoint[0],allLines[i].fromPoint[1]],[allLines[i].toPoint[0],allLines[i].toPoint[1]]]
        for j in range(len(drewLines)):
            line2 = [[drewLines[j].fromPoint[0],drewLines[j].fromPoint[1]],[drewLines[j].toPoint[0],drewLines[j].toPoint[1]]]
            if allLines[i].intersect(line1,line2):
```

```
                slope1 = allLines[i].slope(line1[0],line1[1])
                slope2 = allLines[j].slope(line2[0],line2[1])
                yA = allLines[i].y_intercept(line1[0],slope1)
                yB = allLines[i].y_intercept(line2[0],slope2)
                a = allLines[i].line_intersect(slope1,yA,slope2,yB)
                if a:
                    x , y = a[0], a[1]
                    dist = ((line1[0][0] - x)**2)  + ((line1[0][1] - y)**2)
                    if dist < allLines[i].dist:
                        allLines[i].toPoint = [x,y]
                        allLines[i].dist = dist
        if checkIfOutCircle(allLines[i].fromPoint[0], width // 2, allLines[i].fromPoint[1], height // 2, radius) or checkIfOutCircle(allLines[i].toPoint[0], width // 2, allLines
            drewLines.append(allLines[i])
            s,d = allLines[i].getDetails()
            win.create_line(s[0],s[1],d[0],d[1])
            win.update()

draw()

# Can save using this

# win.pack()
# win.update()
# win.postscript(file="file_name.eps", colormode='color')
# img = Image.open("file_name.eps")
# img.save('filename.png')

root.mainloop()
```

```
// Authored By: Jayant Sogikar
#include<omp.h>
#include<stdio.h>
#include<stdlib.h>

int main(){

   // Init a parallel section
 #pragma omp parallel
 {
  // Finding thread number
  int t = omp_get_thread_num();

  // output
  printf("Hello world from thread number %d \n",t);
 }

 return 0;
}

// gcc hw.c -fopenmp
// ./a.out
```

```
# Authored By: Jayant Sogikar
from collections import defaultdict


def solution(map):

    fromA = defaultdict(lambda : float('inf'))
    fromB = defaultdict(lambda : float('inf'))

    r, c = len(map) , len(map[0])
    fromA[(0,0)] = 1

    q = [(0,0,1)]

    while q:
        x, y, score = q.pop(0)



        for (i,j) in [(0,1), (1,0),(-1,0), (0, -1)]:
            if 0 <= x + i < r and 0 <= y + j < c and fromA[(x + i,y + j)] == float('inf'):

                fromA[(x + i,y + j)] = score + 1

                if map[x + i][y + j] == 0 : q.append((x + i, y + j, score + 1))


    q = [(r - 1, c - 1, 1)]
    fromB[(r-1,c-1)] = 1

    while q:
        x, y, score = q.pop(0)


        for (i,j) in [(0,1), (1,0),(-1,0), (0, -1)]:
            if 0 <= x + i < r and 0 <= y + j < c and fromB[(x + i,y + j)] == float('inf'):

                fromB[(x + i,y + j)] = score + 1

                if map[x + i][y + j] == 0 : q.append((x + i, y + j, score + 1))


    minVal = float('inf')



    for i in range(r):
        for j in range(c):
            # print(fromA[(i,j)], fromB[(i,j)], fromA[(i,j)] + fromB[(i,j)])
            minVal = min(minVal, fromA[(i,j)] + fromB[(i,j)])


    return minVal - 1




print(solution([[0, 1, 1, 0], [0, 0, 0, 1], [1, 1, 0, 0], [1, 1, 1, 0]]))
```