1453. Maximum Number of Darts Inside of a Circular Dartboard

Problem Description

In this problem, you have a square wall with a circular dartboard. You have been asked to throw darts at the wall blindfolded and each throw results are represented as an array of points in a 2D plane. The task is to find out the maximum number of points that fall within or lie on any circular dartboard of a given radius, `r`.

Let's have a look at an example:

Input: points = [[-2,0], [2,0], [0,2], [0,-2]], r = 2 Output: 4

Here all the dart throws fall on the dartboard of radius 2 unit, centered on the origin. Thus, the maximum number of points that can fit inside this circular dartboard is 4.

Approach

The given problem can be solved using a geometric approach. We generate all the pairs of the given points and find out the 2 possible centers of the maximum circle containing those two points. Later on, we check for each center's location if it holds those points or not and keep track of the maximum count of points that can be included.

To make this process easier, we can create a `Point` structure to represent a point in 2D coordinate space and use some geometry functions that can calculate distance, create a circle given two points.

Pseudo-code:

1. Convert given array of points into a list (vector) of Point objects.
2. For each pair of points, construct two circles where each point is on the edge of the circle.
3. For each of these circles, calculate the number of points within that circle.
4. Keep track of the maximum number of points encountered.

Now let's translate this approach into solutions in different languages:

Python Solution

```
python
from typing import List
from cmath import phase, rect
import math

class Solution:
    def numPoints(self, points: List[List[int]], r: int) -> int:
        xs = [x + y*1j for x, y in points]
        n = len(xs)

        def test(c):
            return sum(abs(x-c) <= r + 10**-7 for x in xs)

        def get_centers(P, Q):
            diam = abs(P-Q)
            M = (P + Q) / 2
            h = (r**2 - (diam / 2)**2) ** .5
            delta = h / diam * (Q - P) * 1j
            return M + delta, M - delta

        res = max(test(P) for P in xs)
        for i in range(n):
            for j in range(i):
                for C in get_centers(xs[i], xs[j]):
                    res = max(res, test(C))
        return res
```

Java Solution

```
java
public class Solution {
    public int numPoints(int[][] points, int r) {
        int[][] pts = points;
        int n = pts.length;
        int res = 1;
        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                double cX = (pts[i][0] + pts[j][0]) / 2.0;
                double cY = (pts[i][1] + pts[j][1]) / 2.0;
                double x = Math.abs(cX - pts[i][0]);
                double y = Math.abs(cY - pts[i][1]);
                double d = Math.sqrt(x * x + y * y);
                if (d > r) {
                    continue;
                }
                double[] center = new double[]{cX, cY};
                int count = 0;
                for (int k = 0; k < n; ++k) {
                    double dx = center[0] - pts[k][0], dy = center[1] - pts[k][1];
                    if (Math.sqrt(dx*dx + dy*dy) <= r + 1e-6) {
                        ++count;
                    }
                }
                res = Math.max(res, count);
            }
        }
        return res;
    }
}
```

Javascript Solution

```
javascript
let numPoints = function(points, r) {
    let pointsList = points;
    let n = pointsList.length, res = 1, x = new Array(n), y = new Array(n);
    for (let i = 0; i < n; ++i) {
        x[i] = pointsList[i][0]*1.0, y[i] = pointsList[i][1]*1.0;
    }
    for (let i = 0; i < n; ++i) {
        for (let j = i+1; j < n; ++j) {
            let dis = ((x[i]-x[j])*(x[i]-x[j]) + (y[i]-y[j])*(y[i]-y[j]));
            if (dis > 4.0*r*r) continue;
            let ang1 = Math.atan2(y[j]-y[i], x[j]-x[i]);
            let ang2 = Math.acos(dis/(4.0*r));
            ang1 -= Math.PI/2.0;
            let ang = ang1-ang2;
            let cx = x[i] + r*Math.cos(ang), cy = y[i] + r*Math.sin(ang);
            let tmp = 0;
            for (let k = 0; k < n; ++k) {
                let dx = cx - x[k], dy = cy - y[k];
                if (dx*dx+dy*dy <= 1.0*r*r+1e-7) ++tmp;
            }
            res = Math.max(res, tmp);
        }
    }
    return res;
};
```

C++ Solution

```
cpp
class Solution {
 public:
  int numPoints(vector<vector<int>>& points, int r) {
    const int n = points.size();
    int ans = 1;
    for (int i = 0; i < n; ++i)
      for (int j = i + 1; j < n; ++j) {
        const auto [x1, y1] = points[i];
        const auto [x2, y2] = points[j];
        const double d = hypot(x1 - x2, y1 - y2);
        for (const double delta = 0; delta <= M_PI * 2 + 1e-7; delta += M_PI * 1 / 180.0) {
          const double x = (x1 + x2) / 2.0 + cos(delta) * sqrt(r * r - d * d / 4);
          const double y = (y1 + y2) / 2.0 + sin(delta) * sqrt(r * r - d * d / 4);
          int cnt = 0;
          for (const auto& [xi, yi] : points)
            cnt += hypot(x - xi, y - yi) < r + 1e-7;
          ans = max(ans, cnt);
        }
      }
    return ans;
  }
};
```

C# Solution

```
csharp
public class Solution {
    public int NumPoints(int[][] points, int r) {
        int n = points.Length;
        int[] x = new int[n], y = new int[n];
        for (int i = 0; i < n; ++i) {
            x[i] = points[i][0]; y[i] = points[i][1];
        }
        int res = 1;

        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
            {
                double epsilon = 1e-7;
                double px = (x[i] + x[j]) / 2.0, py = (y[i] + y[j]) / 2.0;
                double dx = x[i] - x[j], dy = y[i] - y[j];
                double d = Math.Sqrt(dx * dx + dy * dy);
                double angle = Math.Atan2(-dx, dy);
                double da = Math.Acos(d / (2.0 * r));
                int sc = 0, sc2 = 0;
                for (int k = 0; k < n; ++k)
                {
                    double qx = x[k] - px, qy = y[k] - py;
                    double h2 = qx * qx + qy * qy;
                    if (h2 <= (double)r * r + epsilon * 2.0) sc2++;
                    if (Math.Abs(qy * dx - qx * dy) <= r * d + epsilon &&
                        (qx * dx + qy * dy) >= -epsilon &&
                        (qx * dx + qy * dy) <= d * d + epsilon) sc++;
                }
                res = Math.Max(res, Math.Max(sc, sc2));
            }

        return res;
    }
}
```

Ruby Solution

```
ruby
class Solution
  def num_points(points, r)
    res, ep, n = 1, 1e-7, points.size
    points.each_with_index do |e, i|
      points[i] = [ e[0]*1.0, e[1]*1.0 ]
    end
    0.upto(n-1) do |i|
      (i+1).upto(n-1) do |j|
        x1, y1, x2, y2 = points[i][0], points[i][1], points[j][0], points[j][1]
        dis = Math.sqrt((x2-x1)**2 + (y2-y1)**2)
        next if dis > 2.0*r
        a1 = Math.atan2(y2-y1, x2-x1)
        a2 = Math.acos(dis/(2*r))
        [ a1-a2, a1+a2 ].each do |a|
          x0, y0, tmp = x1 + r*Math.cos(a), y1 + r*Math.sin(a), 0
          0.upto(n-1) do |k|
            dx, dy = points[k][0]-x0, points[k][1]-y0
            tmp += 1 if dx*dx + dy*dy < r*r + ep
          end
          res = [res, tmp].max
        end
      end
    end
    res
  end
end
```

PHP Solution

```
php
function numPoints(array $points, int $r): int {
    $n = count($points);
    $eps = 1e-7;
    for ($i = 0; $i < $n; $i++) {
        for ($j = $i + 1; $j < $n; $j++) {
            $x1 = $points[$i][0]; $y1 = $points[$i][1];
            $x2 = $points[$j][0]; $y2 = $points[$j][1];
            $dx = $x2 - $x1; $dy = $y2 - $y1;
            $d = sqrt($dx*$dx + $dy*$dy);
            if ($d > 2*$r) continue;
            $delta = sqrt($r*$r - ($d*$d/4);
            $xx = ($x1 + $x2) / 2; $yy = ($y1 + $y2) / 2;
            foreach (array(-1, 1], [1, -1]) as $p) {
                $x = $xx + $delta*$dy*$p[0]/$d; $y = $yy + $delta*$dx*$p[1]/$d;
                $res = 0;
                for ($k = 0; $k < $n; $k++) {
                    $ddx = $x - $points[$k][0]; $ddy = $y - $points[$k][1];
                    if ($ddx*$ddx + $ddy*$ddy < $r*$r + $eps) {
                        $res++;
                    }
                }
                $out = max($out, $res);
            }
        }
    }
    return $out;
}
```