# 题目

给定圆的半径和圆心的位置，实现函数randPoint，在圆中产生均匀随机点。

实现Solution类:

Solution(double radius, double x\_center, double y\_center) 用圆的半径 radius 和圆心的位置 (x\_center, y\_center) 初始化对象

randPoint()返回圆内的一个随机点。圆周上的一点被认为在圆内。答案作为数组返回 [x, y] 。

示例 1：

输入:

["Solution","randPoint","randPoint","randPoint"]

[[1.0, 0.0, 0.0], [], [], []]

输出: [null, [-0.02493, -0.38077], [0.82314, 0.38945], [0.36572, 0.17248]]

解释:

Solution solution = new Solution(1.0, 0.0, 0.0);

solution.randPoint ();//返回[-0.02493，-0.38077]

solution.randPoint ();//返回[0.82314,0.38945]

solution.randPoint ();//返回[0.36572,0.17248]

提示：

0 < radius <= 108

-107 <= x\_center, y\_center <= 107

randPoint最多被调用3 \* 104次

# 分析

## 方法一：随机函数/拒绝采样

思路：

我们使用 srand(time(NULL)) 来设置随机数种子，以便在每次运行时产生不同的随机序列。然后使用 rand() 函数来生成随机数。需要注意的是，rand()函数的随机性可能没有 mt19937 那么好，但对于这个问题，它应该是足够的。

代码：

class Solution {

private:

double radius;

double x\_center;

double y\_center;

public:

Solution(double radius, double x\_center, double y\_center) {

this->radius = radius;

this->x\_center = x\_center;

this->y\_center = y\_center;

srand(time(NULL)); // 设置随机种子

}

vector<double> randPoint() {

double r = radius \* sqrt((double)rand() / RAND\_MAX); // 极坐标半径

double theta = 2 \* M\_PI \* ((double)rand() / RAND\_MAX); // 极坐标角度

double x = x\_center + r \* cos(theta); // 转换为直角坐标系

double y = y\_center + r \* sin(theta);

return {x, y};

}

};

或：

uniform\_real\_distribution 的工作原理基于均匀分布的数学模型。均匀分布是指在一个指定区间内，每一个数值被选中的概率是相等的。在C++中，uniform\_real\_distribution 通过一个随机数生成器（如 mt19937）和指定的区间范围（包括下界和上界），来生成满足均匀分布的随机浮点数。

具体来说，uniform\_real\_distribution 会首先获取随机数生成器产生的一个随机数，然后将其映射到指定的区间范围内。这个映射过程通常是线性的，即通过一个线性变换将随机数生成器产生的随机数转换为指定区间的浮点数。

class Solution {

private:

double radius;

double x\_center;

double y\_center;

mt19937 rng; // 随机数生成器

public:

Solution(double radius, double x\_center, double y\_center) {

this->radius = radius;

this->x\_center = x\_center;

this->y\_center = y\_center;

random\_device rd;

rng.seed(rd()); // 使用随机设备生成种子

}

vector<double> randPoint() {

uniform\_real\_distribution<double> uni(0, 1); // 均匀分布 [0, 1)

double r = radius \* sqrt(uni(rng)); // 极坐标半径

double theta = 2 \* M\_PI \* uni(rng); // 极坐标角度

double x = x\_center + r \* cos(theta); // 转换为直角坐标系

double y = y\_center + r \* sin(theta);

return {x, y};

}

};