Reflections over a geometric problem

April 12, 2023

1 Reflections' over a geometric problem - Script de Python

En el presente *notebook* presentamos el código propuesto en la exposición para resolver nuestro problema sobre lasers y reflexiones. Empezaremos con una habitación cuyo ancho es n y alto m. Ahora, se ubica dentro de ella un punto $A = (a_0, a_1)$ y se ubica otro punto $B = (b_0, b_1)$ en ubicaciones estrictamente distintas. Desde A disparamos un laser cuyo alcance tiene un rango máximo de d unidades. Nuestro objetivo será determinar de cuantas maneras podemos disparar el laser de forma que impacté en B, teniendo en cuenta la distancia máxima d y que el láser no pase por A en el camino.

1.1 Funciones preliminares

1.1.1 Máximo común divisor

El máximo común divisor de dos números enteros a y b es el entero positivo d tal que $d|a$ y $d|b$, y si $c|a$ y $c|b$ entonces $c \leq d$ para todo $c \in \mathbb{Z}^{\geq 0}$, notado por $g.c.d(a, b)$. Para cálcular el máximo común divisor usamos el algoritmo de Euclides basado en los siguientes dos hechos:

- Para todo $a \in \mathbb{Z}$, $g.c.d(a, 0) = |a|$
- Si $a = b \cdot q + r$ con $q, r \in \mathbb{Z}$ y $0 \leq r < b$ entonces $g.c.d(a, b) = g.c.d(b, r)$

La función que se encuentra abajo está definida por recursión para cálcular de forma eficiente el máximo común divisor con las técnicas ya dichas,

[ ]:
```
def gdc(a, b):
    if b == 0:
        return a
    else:
        return gdc(b, a % b)
```

1.1.2 Distancia entre dos puntos

En matemáticas, una función métrica sobre un conjunto X, es una función $d : X \times X \to \mathbb{R}^{0\geq}$ tal que:

- $d(x, y) = 0$ si y solo si $x = y$
- $d(x, y) = d(y, x)$
- $d(x, y) \leq d(x, z) + d(y, z)$

En este caso para determinar la distancia sobre dos puntos en $\mathbb{R}^2$ usaremos la **métrica euclidiana** definida por:

$$d((x_1, x_2), (y_1, y_2)) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

[ ]:
```
def distance(X, Y):
    return ((X[0] - Y[0])**2 + (X[1] - Y[1])**2)**0.5
```

1.2 Programa principal

La siguiente celda muestra en Python una implementación sobre las ideas propuestas en la exposición. Aunque con una cantidad grande de código, note que mucho de él es parte de lo explicado.

[ ]:
```
from math import ceil

n, m = tuple(map(int, input().split()))
Ax, Ay = tuple(map(int, input().split()))
Bx, By = tuple(map(int, input().split()))
d = int(input())

if distance((Ax, Ay), (Bx, By)) > d:
    print(0)
else:
    TimesX = ceil(max((d - (n - Ax)) / n, (d - (n - (n - Ax))) / n))
    TimesY = ceil(max((d - (m - Ay)) / m, (d - (m - (m - Ay))) / m))


    vectorsA = set()
    vectorsB = {(Bx, By)}
    inclinations = {((Ax - Bx) // abs(gdc(Ax-Bx, Ay-By)), (Ay - By) //␣
 ↪abs(gdc(Ax-Bx, Ay-By)))}

    changePx, changeQx, changeRx, changeSx = 0, 0, 0, 0

    for i in range(-1, TimesX):
        if i % 2 == 0 and i >= 0:
            changePx -= 2 * Ax
            changeQx -= 2 * Bx
            changeRx += 2 * (n - Ax)
            changeSx += 2 * (n - Bx)
        elif i >= 0:
            changePx -= 2 * (n - Ax)
            changeQx -= 2 * (n - Bx)
            changeRx += 2 * Ax
            changeSx += 2 * Bx

        Px, Qx = Ax + changePx, Bx + changeQx
        Rx, Sx = Ax + changeRx, Bx + changeSx
```

```python
        changeAyUp, changeByUp, changeAyDown, changeByDown = 0, 0, 0, 0

        for j in range(-1, TimesY):
            if i == j == -1:
                continue

            if j % 2 == 0 and j >= 0:
                changeAyDown -= 2 * Ay
                changeByDown -= 2 * By
                changeAyUp += 2 * (m - Ay)
                changeByUp += 2 * (m - By)
            elif j >= 0:
                changeAyDown -= 2 * (m - Ay)
                changeByDown -= 2 * (m - By)
                changeAyUp += 2 * Ay
                changeByUp += 2 * By

            PyUp, QyUp = Ay + changeAyUp, By + changeByUp
            RyUp, SyUp = Ay + changeAyUp, By + changeByUp
            PyDown, QyDown = Ay + changeAyDown, By + changeByDown
            RyDown, SyDown = Ay + changeAyDown, By + changeByDown

            APNormUp, APNormDown = abs(gdc(Ax - Px, Ay - PyUp)), abs(gdc(Ax - Px, Ay - PyDown))
            AQNormUp, AQNormDown = abs(gdc(Ax - Qx, Ay - QyUp)), abs(gdc(Ax - Qx, Ay - QyDown))
            ARNormUp, ARNormDown = abs(gdc(Ax - Rx, Ay - RyUp)), abs(gdc(Ax - Rx, Ay - RyDown))
            ASNormUp, ASNormDown = abs(gdc(Ax - Sx, Ay - SyUp)), abs(gdc(Ax - Sx, Ay - SyDown))

            AQUp, AQDown = ((Ax - Qx) // AQNormUp, (Ay - QyUp) // AQNormUp), ((Ax - Qx) // AQNormDown, (Ay - QyDown) // AQNormDown)
            ASUp, ASDown = ((Ax - Sx) // ASNormUp, (Ay - SyUp) // ASNormUp), ((Ax - Sx) // ASNormDown, (Ay - SyDown) // ASNormDown)

            vectorsA.update({
                ((Ax - Px) // APNormUp, (Ay - PyUp) // APNormUp),
                ((Ax - Px) // APNormDown, (Ay - PyDown) // APNormDown),
                ((Ax - Rx) // ARNormUp, (Ay - RyUp) // ARNormUp),
                ((Ax - Rx) // ARNormDown, (Ay - RyDown) // ARNormUp)
            })

            if distance((Ax, Ay), (Qx, QyUp)) <= d and AQUp not in inclinations:
```

```
                if AQUp not in vectorsA or (AQUp == ((Ax - Px) // APNormUp, (Ay - PyUp) // APNormUp) and distance((Ax, Ay), (Qx, QyUp)) <= distance((Ax, Ay), (Px, PyUp))):
                    vectorsB.add((Qx, QyUp))
                    inclinations.add(AQUp)



            if AQDown not in vectorsA:
                if((Ax - Qx)**2 + (Ay - QyDown)**2)**0.5 <= d and AQDown not in inclinations:
                    vectorsB.add((Qx, QyDown))
                    inclinations.add(AQDown)

            elif AQDown == ((Ax - Px) // APNormDown, (Ay - PyDown) // APNormDown):
                if((Ax - Qx)**2 + (Ay - QyDown)**2)**0.5 <= d and AQDown not in inclinations and ((Ax - Qx)**2 + (Ay - QyDown)**2)**0.5 <= ((Ax - Px)**2 + (Ay - PyDown)**2)**0.5:
                    vectorsB.add((Qx, QyDown))
                    inclinations.add(AQDown)


            if ASUp not in vectorsA:
                if((Ax - Sx)**2 + (Ay - SyUp)**2)**0.5 <= d and ASUp not in inclinations:
                    vectorsB.add((Sx, SyUp))
                    inclinations.add(ASUp)

            elif ASUp == ((Ax - Rx) // ARNormUp, (Ay - RyUp) // ARNormUp):
                if((Ax - Sx)**2 + (Ay - SyUp)**2)**0.5 <= d and ASUp not in inclinations and ((Ax - Sx)**2 + (Ay - SyUp)**2)**0.5 <= ((Ax - Rx)**2 + (Ay - RyUp)**2)**0.5:
                    vectorsB.add((Sx, SyUp))
                    inclinations.add(ASUp)


            if ASDown not in vectorsA:
                if((Ax - Sx)**2 + (Ay - SyDown)**2)**0.5 <= d and ASDown not in inclinations:
                    vectorsB.add((Sx, SyDown))
                    inclinations.add(ASDown)

            elif ASDown == ((Ax - Rx) // ARNormDown, (Ay - RyDown) // ARNormDown):
```

```
                if((Ax - Sx)**2 + (Ay - SyDown)**2)**0.5 <= d and ASDown not in␣
↪inclinations and ((Ax - Sx)**2 + (Ay - SyDown)**2)**0.5 <= ((Ax - Rx)**2 +␣
↪(Ay - RyDown)**2)**0.5:
                    vectorsB.add((Sx, SyDown))
                    inclinations.add(ASUp)


    print(len(vectorsB))
```

```
3 2
1 1
2 1
4
7
```