Rust: El Rompecabezas de las Sombras

Introducción

La modelización de fenómenos del mundo real es una de las tareas fundamentales de la programación. Desde la simulación de sistemas físicos complejos y el renderizado de gráficos en videojuegos hasta el análisis de datos geoespaciales, la capacidad de representar objetos, entidades y sus interacciones a través de estructuras de datos es una habilidad esencial. En estos dominios, la geometría computacional juega un papel crucial, proporcionando las herramientas para describir y manipular formas, espacios y las relaciones entre ellos.

Descripción del Problema

Este ejercicio se centra en un escenario geométrico simple pero interesante: calcular la longitud total de las sombras proyectadas por un conjunto de entidades en un mundo bidimensional.

Imaginemos un mundo plano, con una carretera infinita que se extiende a lo largo del eje X. Sobre esta carretera, se encuentran *N* "flatlanders" (seres planos), cada uno con una posición *X* y una altura *H* determinadas.

Una fuente de luz, ubicada infinitamente lejos al oeste, ilumina a estos seres con un ángulo *theta* respecto al suelo. Cada flatlander proyecta una sombra hacia el este. La longitud de la base de esta sombra (*L*) se puede calcular mediante la siguiente relación trigonométrica:

L = H / tan(theta)

Por lo tanto, un flatlander en la posición *X* con altura *H* proyectará una sombra que cubre el intervalo de la carretera desde *[X, X + L]*.

Ejemplo 1: Una Sola Sombra

Supongamos que tenemos un único flatlander en la posición *X=10* con una altura *H=20*, y la luz incide con un ángulo *theta=45º*.

La longitud de la sombra es:

L = 20 / tan(45º) = 20 / 1 = 20

El intervalo de la carretera cubierto por la sombra es *[10, 30]*.

Ejemplo 2: Sombras Superpuestas

El desafío principal surge cuando múltiples flatlanders proyectan sombras que pueden superponerse.

Consideremos dos flatlanders y un ángulo de *45º*:

- **Flatlander 1:** Posición 0, Altura 10.
  - Su longitud de sombra es *L_1 = 10 / tan(45º) = 10*.
  - Cubre el intervalo *[0, 10]*.
- **Flatlander 2:** Posición 5, Altura 10.
  - Su longitud de sombra es *L_2 = 10 / tan(45º) = 10*.
  - Cubre el intervalo *[5, 15]*.

Las sombras se superponen. El área total cubierta es la unión de los dos intervalos, resulta en el intervalo único *[0, 15]*, por lo tanto, la longitud total cubierta es **15**.

Ejercicio

El ejercicio propuesto es implementar en Rust un programa que resuelva el problema del rompecabezas de las sombras, calculando la longitud total cubierta para el input dado. Siguiendo con los siguientes formatos y restricciones:

1. Formato de Input

El input se recibirá por la entrada estándar (stdin).

La primera línea contendrá dos números: un entero *theta (10 <= theta <= 80)* que representa el ángulo en grados, y un entero *N (1 <= N <= 10^5)* que representa el número de flatlanders.

Las siguientes *N* líneas contendrán cada una dos números: un entero *X (0 <= X <= 3 . 10^5)* y un entero *H (1 <= H <= 1000)*, que representan la posición y la altura de cada flatlander, respectivamente.

Ejemplo de Input:

```
45 2
0 10
5 10
```

2. Formato de Output

El programa deberá imprimir **únicamente** el resultado por salida estándar (stdout) en una única línea con un número de punto flotante. La respuesta se considerará correcta si el error absoluto o relativo no excede *10^-4*

Ejemplo de Output:

```
15.0000000000000
```

> **IMPORTANTE: Se ejecutará una suite de tests automatizados para probar el correcto funcionamiento del programa, por eso es importante respetar el formato indicado.**

Ejemplos Adicionales

Input:

```
30 3
50 150
0 100
100 200
```

Output:

```
446.4101615137755
```

Input:

```
45 3
50 150
0 100
100 200
```

Output:

```
300.00000000000006
```

Recursos Útiles

- Structs
- Módulo `std::f64` para funciones trigonométricas y constantes (recordar que las funciones trigonométricas en Rust operan con radianes).

Restricciones

- Escribir el programa sin utilizar `.unwrap()` o `.expect()`. Todo caso de error deberá manejarse idiomáticamente con las estructuras y funciones brindadas por el lenguaje.
- No se permite que el programa lance un panic!().
- No se permite utilizar la función exit(). Se deberá salir del programa finalizando el scope de la función `main`.
- No se permite utilizar el módulo mem para la manipulación de memoria.
- Para realizar un uso adecuado de memoria y respetar las reglas de ownership se deberá evitar el uso de .clone() y .copy() en las estructuras principales de datos.
- Todo el programa puede ser resuelto con lo aprendido en clase hasta la presentación de este ejercicio. No se espera (ni se acepta) que se utilicen estructuras relacionadas a concurrencia o redes para resolución de este ejercicio.

Requerimientos no funcionales

Los siguientes son los requerimientos no funcionales para la resolución del proyecto:

- El proyecto deberá ser desarrollado en la última versión estable de Rust, usando las herramientas de la biblioteca estándar.
- Se deben implementar tests unitarios y de integración de las funcionalidades que se consideren más importantes.
- No se permite utilizar crates externos.
- El código fuente debe compilarse en la versión estable del compilador y no se permite utilizar bloques `unsafe`.
- El código deberá funcionar en ambiente Unix / Linux.
- El programa deberá ejecutarse en la línea de comandos, leyendo de `stdin` y escribiendo a `stdout`.
- La compilación no debe arrojar `warnings` del compilador, ni del linter `clippy`.
- Las funciones y los tipos de datos (`struct`) deben estar documentados siguiendo el estándar de `cargo doc`.
- El código debe formatearse utilizando `cargo fmt`.
- Las funciones no deben tener una extensión mayor a 30 líneas. Si se requiriera una extensión mayor, se deberá particionarla en varias funciones.

- Cada tipo de dato implementado debe ser colocado en una unidad de compilación (archivo fuente) independiente.

ADVERTENCIA! El proyecto deberá realizarse de manera individual. Cualquier tipo de copia significa la expulsión automática de la materia. No está permitido el uso de código generado por ninguna IA, ni copiar código de soluciones existentes en internet.

IMPORTANTE: no cumplir con todas las restricciones y requisitos no funcionales implica la reentrega automática del ejercicio. Se debe prestar minuciosa atención a cada uno de los detalles.

Fechas de entrega

Primer entrega: 01/09

Luego de la primer entrega se harán las correcciones correspondientes y se podrá volver a entregar el ejercicio en dos oportunidades más.

La forma de entrega se comunicará por el canal de avisos.

No aprobar el trabajo imposibilitará la continuidad en la materia.
