Algoritmos Computacionales
Proyecto intermedio

Diego Alberto Barceló Nieves
Mauricio Sandoval Cuenca
Facultad de Ciencias
Universidad Nacional Autónoma de México

Elige 2 de los 3 ejercicios que se muestran a continuación y entrega las soluciones en un notebook con código de Julia a más tardar el 5 de mayo de 2022.

Ejercicio 1

El triángulo de Sierpinski es un fractal convergente a un conjunto fijo con la figura de un triángulo equilátero subdividido recursivamente en triángulos equiláteros más pequeños. Éste es uno de los ejemplos más básicos de conjuntos *autosemejantes*[1] (ver Figura 1). Existen distintos algoritmos que nos permiten obtener el triángulo de Sierpinski. A continuación, describimos uno de los más sorprendentes:

1. Sin dibujarlos, considera tres puntos en un plano que formen los vértices de un triángulo equilátero.
2. Elige un punto arbitrario dentro de la superficie del triángulo equilátero y considéralo tu *posición actual*.
3. Elige de forma aleatoria uno de los tres vértices del triángulo equilátero.
4. Obtén el punto medio entre tu posición actual y el vértice que elegiste en el paso anterior, considéralo tu nueva posición actual.
5. Dibuja el punto de tu posición actual.
6. Repite desde el paso 3.

El ejercicio consiste en implementar el algoritmo anterior y obtener una aproximación del triángulo de Sierpinski. Después, debes jugar con el número de vértices iniciales y la distancia de la posición actual al vértice elegido para obtener un fractal distinto. Como seguramente podrás notar, no todas las combinaciones devuelven un fractal.

Figura 1: Triángulo de Sierpinski

Pista En Julia, la función `rand()` nos devueelve un número entero aleatorio entre `0` y `1`. ¿Cómo podemos usarla para obtener un número aleatorio dentro de un rango arbitrario?

[1] Se dice que un objeto es autosemejante cuando una o varias partes de un todo repiten exactamente su similitud con ese todo

Ejercicio 2

En el rodaje de una película de acción, se busca encontrar cuál es el piso más alto de un edificio de 100 niveles desde el cual puede saltar un doble de acción sin provocar un accidente. Para esto, el equipo cuenta con dos maniquíes para ensayos de colisiones que pueden dejar caer desde cualquier piso, y se saben las siguientes cosas:

1. Si el maniquí se rompe al caer del piso n, entonces es no seguro que el doble salte desde ese nivel, ni desde ninguno de los superiores.
2. Si el maniquí no se rompe al caer del piso n, entonces es seguro que el doble salte desde ese nivel, y desde cualquiera de los inferiores.
3. Si el maniquí no se rompe después de ser lanzado, entonces puede ser usando de nuevo para otra prueba.
4. Un maniquí roto no puede volver a usarse[2].
5. No se tiene asegurado que el maniquí no se dañe al caer desde el primer piso, ni se sabe con total certeza que se dañará si cae desde el último piso.

El equipo no puede permitir que el doble salte de un piso sin tener total certeza de que es seguro. La forma más inmediata de encontrar el máximo piso seguro desde el que puede saltar es probando de uno en uno, sin embargo, en el peor de los casos, les tomaría 100 intentos.

Una forma más ágil de resolver el problema es aprovechando el hecho de que tienen dos maniquíes de prueba haciendo una *búsqueda binaria*. Se empieza desde el piso 50, si el maniquí no se rompe se vuelve a probar pero ahora desde el piso 75; sin embargo, si el maniquí se rompe, entones se tendrá que probar uno por uno desde el primer piso y, en el peor de los casos, tomaría 49 intentos.

Se puede probar que, para el caso de dos maniquíes y 100 pisos, la mejor solución se obtiene empezando desde el piso 14, obteniendo que, en el peor de los casos, tomaría un máximo 14 intentos encontrar el piso indicado. Para este ejercicio, escribe un algoritmo que devuelva el mínimo número de intentos necesarios (en el peor de los casos) teniendo n maniquíes y un edificio de k pisos.

[2] Asume que, si no se rompió, entonces la caída no le causó ningún daño.

Ejercicio 3

Una forma para estimar el valor de π (3.141592...) es usando el método de Monte Carlo. Primero, tomamos un cuadrado de 1×1 y un círculo inscrito de radio $\frac{1}{2}$, generamos una cantidad arbitraria de puntos uniformemente distribuidos sobre la superficie del cuadrado y coloreamos de rojo aquellos que se encuentren sobre la superficie del círculo, y de azul, aquellos que estén fuera (ver Figura 2).

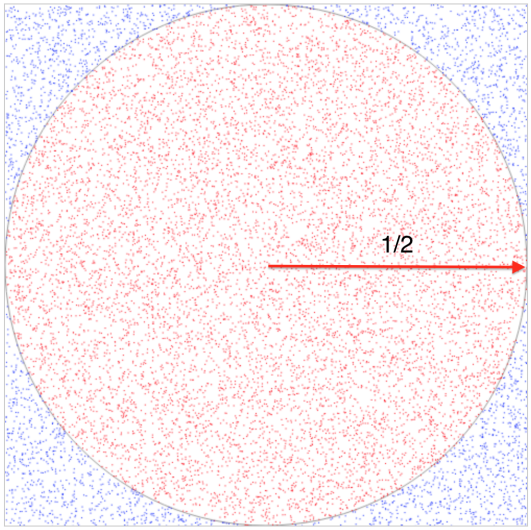

Figura 2: Aproximación de π por el método de Monte Carlo

Ahora, tenemos que el área del círculo esta dada por $\pi r^2 = \frac{\pi}{4}$, mientras que el área del cuadrado es igual a 1 por lo que, si dividimos el área del círculo entre el área del cuadrado, obtenemos $\frac{\pi}{4}$. Además, si N_{rojo} es el número de puntos rojos y N_{total} es el número total de puntos, entonces $\frac{N_{rojo}}{N_{total}}$ es una aproximación del cociente de las áreas para N_{total} lo suficientemente grande; en otras palabras,

$$\frac{\pi}{4} \approx \frac{N_{rojo}}{N_{total}},$$

de donde se sigue que

$$\pi \approx 4\,\frac{N_{rojo}}{N_{total}}. \tag{1}$$

La ecuación (1) nos da la estimación de π por el método Monte Carlo.

1. Escribe un algoritmo que estime el valor de π y que te permita visualizar algo similar al gráfico de la Figura 2, asegúrate de incluir el conteo del número de puntos rojos, número de puntos totales, y la respectiva estimación de π.

2. En promedio[3], ¿cuántos puntos necesitas generar para obtener una precisión de ± 0.01?

3. Realiza una gráfica del error de la estimación en función del número de puntos comparando contra el valor predeterminado de π de Julia (que se obtiene llamando a la constante `pi`).

[3] Al tratarse de un método aleatorio, los resultados variarán de una ejecución a otra, por eso es importante tomar el promedio u otro estadístico.