Proyecto Intermedio: Algoritmos Computacionales

Juárez O. Gissel gisselj@ciencias.unam.mx
Universidad Nacional Autónoma de México, Facultad de Ciencias

28 de abril de 2023

1. Aproximación de π

Instrucciones:
Una forma para estimar el valor de $\pi(3.141592\ldots)$ *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).*

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

$$\pi \approx 4\frac{N_{\text{rojo}}}{N_{\text{total}}}.$$

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 1, asegúrate de incluir el conteo del número de puntos rojos, número de puntos totales, y la respectiva estimación de* π *(6 puntos).*
2. *En promedio, ¿cuántos puntos necesitas generar para obtener una precisión de* $\pm$ *0.01 ? (2 puntos)*
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) (2 puntos).*

Resolución:

1.1. Resolución

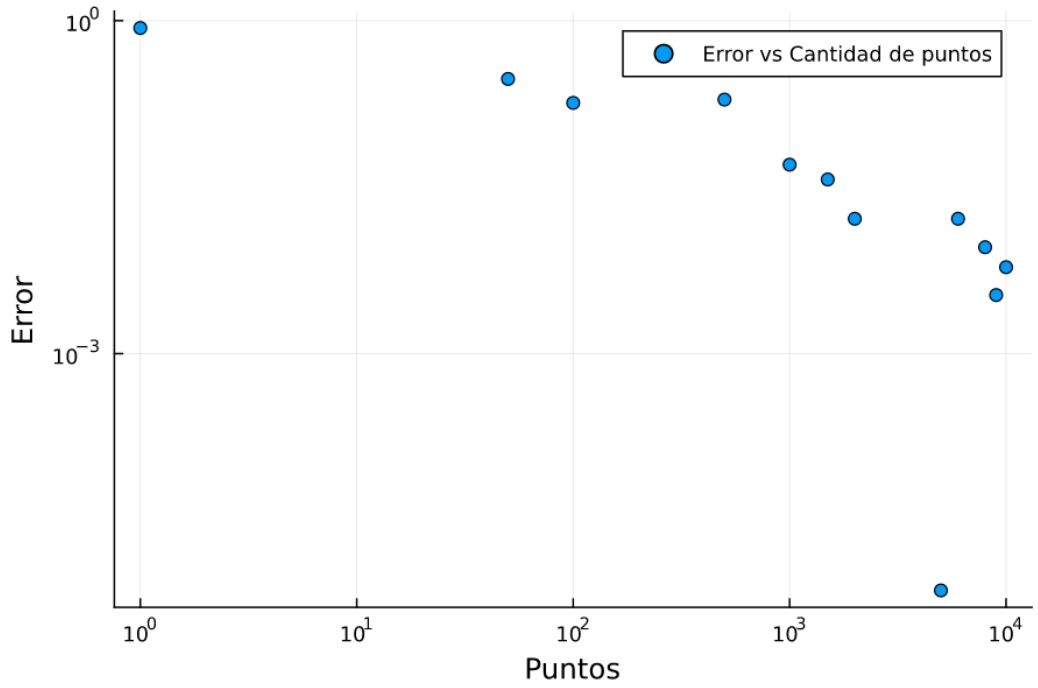
```
[function approx_pi(n)
dentro = 0 #iremos contando los puntos dentro del círculo
    a = 0.5*cos.(0:0.01:2*pi) #definimos el círculo, para poderlo graficar (sólo de referencia, no se
    b = 0.5*sin.(0:0.01:2*pi)
    plot(a, b, color=:red, aspect_ratio=:equal, xlims=(-0.5,0.5), ylims=(-0.5,0.5), legend=false) #lo
for i in 1:n #recorre el rango hasta n
    x, y = rand() .- 0.5, rand() .- 0.5 #se generan coordenadas random, que van del 0 a 0.5 negativo y
    if x^2 + y^2 <= 0.25 #usamos la norma euclidiana para definir si estpán o no en el círculo, despej
        dentro += 1 #aumentamos el contador a 1
        scatter!([x], [y], color = :red, marker = :dot) #se agrega un punto dentro del círculo, color
    else
        scatter!([x], [y], color = :blue, marker = :dot) #se agrega un punto fuera del círculo, color
    end
end
    display(plot()) #ésto fue mi intento para que mostrara algo graficado pero no lo logré UnU
f = dentro / n #Hacemos la aproximación a pi
return f * 4
end]
```

1.2. Resolución

Para obtener esa precisión necesitamos de aproximadamente, más de 10,000 puntos, cuando evaluamos ésta n obtuvimos una aproximación de pi = 3.1516. Queda claro que entre más puntos utilicemos, mejor será la aproximación, sin embargo, se tiene que considerar también el tiempo que se tarda en hacer el cálculo, pues a partir de 10,000 ya se tarda mucho.

1.3. Resolución

Podemos observar en la gráfica, que el error va disminuyendo conforme la cantidad de puntos va aumentando, el error, por supuesto, tiende a 0 cuando n tiende a infinito.



2. Torres de Hanoi

Instrucciones:
El juego de las Torres de Hanoi consiste en tres estacas (izquierda, central y derecha) y n discos redondos de diferentes radios (perforados de forma que puedan encajar en las estacas). Inicialmente la estaca de la izquierda tiene todos los discos en orden creciente de tamaño de abajo hacia arriba.
El objetivo del juego es mover todos los discos a la estaca de la derecha, usando la estaca central. En cada movimiento se desplaza el disco del extremo superior de una estaca a la otra, con la restricción de que no está permitido que un disco de radio mayor quede encima de uno de radio menor.
Tu tarea es encontrar la secuencia de pasos que minimice el número de movimientos necesarios para cumplir el objetivo del juego. Es decir:
Entrada: Un valor entero positivo n (el número de discos).
Salida: Imprime el entero k : el número mínimo de movimientos.
Posteriormente imprime k líneas que describan la secuencia de movimientos. Cada línea debe tener dos enteros, a y b, que indican el movimiento: mover el disco de la estaca a a la estaca b, con $a, b \in 1, 2, 3$. *Se considera que 1 es el disco de la izquierda, 2 el central y 3 el de la derecha.*

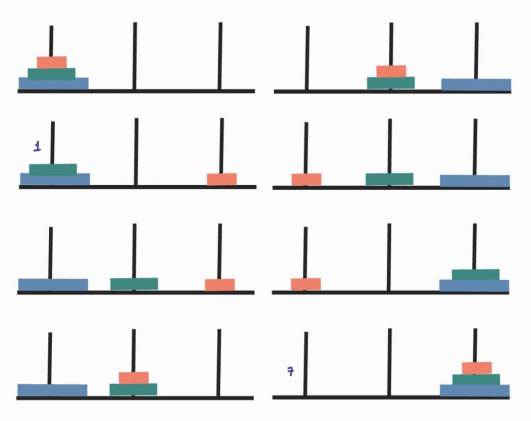
1. *Diseña un algoritmo que resuelva este problema y justifica (no es necesario escribir una demostración) por qué devuelve el número mínimo de movimientos. Represéntalo en pseudocódigo o diagrama de flujo (4 puntos).*

2. *Implementa el algoritmo en Julia. Tu programa será aceptado si devuelve las respuestas correctas para cada n, con* $1 \leq n \leq 16$. *(6 puntos)*

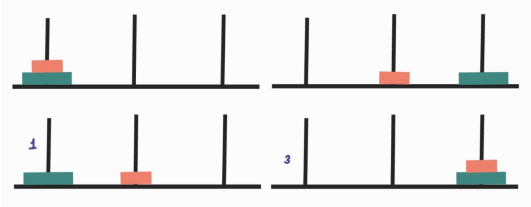
Ejemplo 2 Entrada: 2 Salida: 3 12 13 23

2.1. Resolución

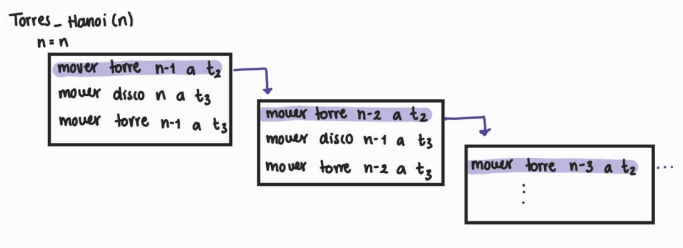
Procedemos por recursión, veamos la manera en la que lo emplearemos con un ejemplo, cuando tenemos 3 discos. Los pasos para resolver el juego de las torres de Hanoi para tres discos son los siguientes:



Pero veamos cuáles son los pasos para resolver el problema cuando tenemos únicamente dos discos:



El caso 3 entonces se resume en mover una torre con 2 discos a la torre intermedia, mover el disco más grande a la torre destino y mover la torre de 2 discos, como si deshiciéramos el arreglo, para pasarlo a latorre destino. Ésto es, entonces, usar el algoritmo para mover una torre de n-1 discos a la torre intermedia (en vez de a la torre destino), mover el disco mayor, y continuar con el algoritmo de torre n-1 a torre destino. Así usaremos la recursión, confiando en que sabemos mover una torre con n-1 discos, y nuestro caso base es con un disco, que se reduce a un movimiento de torre inicial a torre destino.
El pseudocódigo:



Sabemos que el algoritmo devuelve la cantidad mínima de pasos por propiedades de la recursión, pues se basa en el caso mínimo, cuando se tiene un disco. Aún, podemos verificar ésto pues conocemos que la cantidad mínima de movimientos necesarios para mover n discos en $2^n - 1$, al probar nuestro algoritmo verificaremos que se cumple ésto.

2.2. Resolución

Implementación en Julia:

```
[function Torres_Hanoi(n)
        #queremos obtener una lista de pares que indiquen, en la primera entrada, el número de disco, y en
        pasos = [] #arreglo vacío

        #Definimos la función de movimiento
        function movimiento(tamaño, origen, destino, intermedio, pasos) #tamaño: tamaño de la torre, origen
                if tamaño == 1 #definimos el caso base
                        push!(pasos, (origen, destino)) #agregamos al arreglo pasos el par (origen, destin
                else
                        movimiento(tamaño-1, 1, 2, 3, pasos) #Para torres de +1 disco, pasamos la torre de
```

```
                        push!(pasos, (origen, destino)) #agregar movimiento del disco más grande a la torr
                        movimiento(tamaño-1, 2, 3, 1, pasos ) #pasamos la torre de tamaño n-1 a la torre f
                end
        end

        movimiento(n,1,3,2, pasos) #en la función de torres_Hanoi, la "enlazamos" con la función movimient
end
```

El notebook de Pluto se encuentra disponible en GitHub: