Introducción

La heurística de buscar un patrón consiste en tomar un problema, resolver casos pequeños o iniciales, y descubrir algo que esté pasando. Esta es una idea muy general, pero puede ayudar a descubrir varias cosas. Por ejemplo, puede que:

- Tras hacer algunos casos veamos una fórmula.
- Después de jugar un poco con el problema, veamos que hay un ciclo que aparece.
- Al hacer algunas iteraciones de un problema, veamos que ciertos números siempre son crecientes.
- Etcétera.

El encontrar un patrón puede a veces dar de manera inmediata la solución que estamos buscando. Sin embargo, en la mayoría de los casos es simplemente un paso intermedio dentro de una solución más compleja.

La exploración del problema puede ser hecha a mano o de manera computacional. En las siguientes secciones veremos algunos ejemplos de esto en acción.

Fibonaccis y residuos que dejan

La sucesión de Fibonacci es una de las más sencillas de definir. Además, rapidamente encontramos patrones interesantes en ella. Es la sucesión que comienza con 0, luego 1 y a partir de ahí cada número es la suma de los dos anteriores. Así, los primeros números de la sucesión de Fibonacci son:

$$0,1,1,2,3,5,8,13,21,\ldots$$

En símbolos, es la sucesión F_n tal que $F_0=0$, $F_1=1$ y para $n\geq 0$ cumple que $F_{n+2}=F_n+F_{n+1}$.
Consideremos el siguiente problema.

Problema. Sea $\{F_n\}$ la sucesión de Fibonacci.

- Encuentra para cuántos enteros n en $\{0,1,2,\ldots,100\}$ se cumple que F_n es un múltiplo de 3.
- Encuentra para cuántos enteros n en $\{0,1,2,\ldots,1000\}$ se cumple que F_n es un múltiplo de 5.

Solución. Comencemos con el primer inciso. Lo que podríamos hacer es comenzar a escribir a mano algunos números de Fibonacci y ver si encontramos algún patrón evidente: $0,1,1,2,3,5,8,13,21,\ldots$.

Hasta aquí, parece ser que los números de Fibonacci que son múltiplos de 3 son F_0 y F_4. Esto es muy poca evidencia para hacer una conjetura decente. Hagamos más casos.

En vez de realizar esta labor a mano, podemos pedirle a la computadora que haga muchos casos más. El siguiente es un bloque de código en Python. Tras ejecutarlo, muestra los primeros 15 números de Fibonacci y su residuo al dividirse entre 3. Por el momento no es tan importante por qué el código funciona. Más adelante platicaremos de cómo se nos puede ocurrir esto.

```
a,b=0,1

for j in range(15):
    print("Para n={}, Fn es {}. Su residuo al dividir entre 3 es {},".format(j,a,a%3))
    a,b=b,a+b
```

```
Para n=0, Fn es 0. Su residuo al dividir entre 3 es 0,
Para n=1, Fn es 1. Su residuo al dividir entre 3 es 1,
Para n=2, Fn es 1. Su residuo al dividir entre 3 es 1,
Para n=3, Fn es 2. Su residuo al dividir entre 3 es 2,
Para n=4, Fn es 3. Su residuo al dividir entre 3 es 0,
```

```
Para n=5, Fn es 5. Su residuo al dividir entre 3 es 2,
Para n=6, Fn es 8. Su residuo al dividir entre 3 es 2,
Para n=7, Fn es 13. Su residuo al dividir entre 3 es 1,
Para n=8, Fn es 21. Su residuo al dividir entre 3 es 0,
Para n=9, Fn es 34. Su residuo al dividir entre 3 es 1,
Para n=10, Fn es 55. Su residuo al dividir entre 3 es 1,
Para n=11, Fn es 89. Su residuo al dividir entre 3 es 2,
Para n=12, Fn es 144. Su residuo al dividir entre 3 es 0,
Para n=13, Fn es 233. Su residuo al dividir entre 3 es 2,
Para n=14, Fn es 377. Su residuo al dividir entre 3 es 2,
```

Aquí parece ser mucho más claro cuándo un número de Fibonacci es múltiplo de 3. De acuerdo a lo anterior, los números de Fibonacci que son múltiplos de 3 son F_0, F_4, F_8 y F_{12}, de donde ahora tenemos más evidencia para conjeturar lo siguiente.

Conjetura. El Fibonacci F_n es múltiplo de 3 si y sólo si n es múltiplo de 4.

De hecho, la exploración nos permite hacer una conjetura mucho más fuerte que nos dice qué residuos van dejando los números de Fibonacci al dividirse entre 3.

Conjetura. Los residuos de la sucesión de Fibonacci al dividir entre 3 se ciclan, con un ciclo de periodo 8 que es $(0,1,1,2,0,2,2,1)$.

Podemos usar esta conjetura para sospechar cuál es la respuesta al problema: hay 26 enteros n en el conjunto $\{0,1,2,\ldots,100\}$ tales que F_n es múltiplo de 3, a saber cada uno de los múltiplos de 4 en ese conjunto.

Por supuesto, para estar totalmente seguros de esta respuesta tendríamos que verificar que en efecto el patrón se cumple. Más adelante, cuando repasemos el principio de inducción, retomaremos este problema y daremos una demostración formal de que los residuos se ciclan así.

La segunda parte del problema queda para que tú realices una conjetura. $\square$

Suma por renglones en el triángulo de Pascal

Otro objeto matemático en el cual aparecen varios patrones interesantes es el triángulo de Pascal. A continuación se muestran sus primeros renglones.

					1						
				1		1					
			1		2		1				
		1		3		3		1			
	1		4		6		4		1		
1		5		10		10		5		1	

El triángulo de Pascal está construido como sigue. Al inicio, se coloca un uno hasta arriba. Luego, se van agregando renglones, cada vez con un número más que el renglón anterior, con las siguientes reglas:

- Si la posición está en los extremos, el número que se agrega es 1.
- Si la posición es intermedia, entonces es la suma de los dos números que tenga arriba.

Así, por ejemplo, nota que el 10 en el sexto renglón del triángulo es la suma de los dos números que tiene arriba, es decir $10=4+6$. A partir de estas nociones básicas, podemos hacernos una pregunta con respecto a la suma de los elementos de cada renglón.

Pregunta. ¿Cuál es el primer renglón del triángulo de Pascal en el cual la suma de todos sus elementos es mayor o igual a un millón?

Solución. El problema nos invita a explorar cómo es la suma de los elementos en cada uno de los renglones del triángulo de Pascal. Coloquemos dicha suma hasta la derecha de cada renglón. Por ejemplo, como en el cuarto renglón la suma de los elementos es \(1+3+3+1=8\), en la siguiente figura hemos puesto a \(8\) hasta la derecha, en negritas.

					1						**1**
				1		1					**2**
			1		2		1				**4**
		1		3		3		1			**8**
	1		4		6		4		1		**16**
1		5		10		10		5		1	**32**

Aquí hay un patrón evidente (¿cuál?). Si quisiéremos obtener más evidencia de que este patrón sigue, podríamos pedirle a Python que hiciera más renglones del triángulo de Pascal, por ejemplo, que haga los primeros \(10\) renglones. Esto se logra mediante el siguiente código. Una vez más, puedes estudiar el código pero a estas alturas no es totalmente necesario que entiendas a profundidad qué está haciendo.

```
renglon=[1]
print("El renglón {} tiene suma {}".format(renglon,sum(renglon)))

for j in range(1,10):
    new_renglon=[]
    for k in range(j+1):
        if k==0 or k==j:
            new_renglon.append(1)
        else:
            new_renglon.append(renglon[k-1]+renglon[k])
    renglon=new_renglon
    print("El renglón {} tiene suma {}".format(renglon,sum(renglon)))
```

```
El renglón [1] tiene suma 1
El renglón [1, 1] tiene suma 2
El renglón [1, 2, 1] tiene suma 4
El renglón [1, 3, 3, 1] tiene suma 8
El renglón [1, 4, 6, 4, 1] tiene suma 16
El renglón [1, 5, 10, 10, 5, 1] tiene suma 32
El renglón [1, 6, 15, 20, 15, 6, 1] tiene suma 64
El renglón [1, 7, 21, 35, 35, 21, 7, 1] tiene suma 128
El renglón [1, 8, 28, 56, 70, 56, 28, 8, 1] tiene suma 256
El renglón [1, 9, 36, 84, 126, 126, 84, 36, 9, 1] tiene suma 512
```

Con esto podemos realizar una conjetura. Para que nuestra conjetura quede un poco más simple, pensemos en que el renglón de hasta arriba es el renglón \(0\), el siguiente es el renglón \(1\) y así sucesivamente.

Conjetura. La suma de los números en el renglón \(n\) del triángulo de Pascal es \(2^n\).

Más adelante, cuando hablemos de doble conteo, tendremos más herramientas para demostrar esto. Sin embargo, de momento tomémoslo como un hecho y usémoslo para responder el problema. Si queremos que los números de un renglón excedan un millón, entonces necesitamos que \(2^n\geq 1000000\). Aprovechemos que podemos usar Python aquí para saber cuándo sucede esto.

```
n=0
while 2**n<1000000:
    n+=1
print(n)
```

```
20
```

Así, la primera vez que 2^n excede un millón (y por lo tanto la respuesta a nuestro problema) es con $n=20$. $\square$

Hasta el final de la solución pudimos haber simplemente copiado el código anterior para que Python hiciera la suma de nuevo. Esto no sería ningún problema, y lo haría muy rápido. Sin embargo, de manera intuitiva es fácil convencerse de que gracias a la conjetura se puede plantear el nuevo código y con él se hacen «muchas menos operaciones». Más adelante, cuando hablemos de complejidad computacional, formalizaremos esto.

Triángulo de Pascal con saltos de tres en tres

En los problemas anteriores el patrón que debemos encontrar sale muy rápido. Esto no necesariamente será el caso cuando tengamos problemas más complicados. Veamos un problema en el que la exploración y los patrones que debemos encontrar son mucho más elaborados.

Problema. ¿Cuánto suman los elementos del renglón 100 del triángulo de Pascal si los sumamos comenzando con el primero y saltando de tres en tres?

El problema sugiere ver en cada renglón cuánto suman los elementos si saltamos de tres en tres. Es decir, para los primeros renglones del triángulo de Pascal la siguiente figura muestra en negritas los números que tendríamos que sumar en cada renglón.

					1						
				1		1					
			1		2		1				
		1		3		3		**1**			
	1		4		6		**4**		1		
1		5		10		**10**		5		1	

Específicamente, queremos ver qué sucede en el renglón 100.

Solución. Si comenzamos a explorar el problema de manera directa, es muy probable que no encontremos un patrón de manera inmediata para la suma de los elementos que nos interesan. Llamando A_n a la suma de los elementos del renglón n saltando de 3 en 3 número que nos interesa, hasta el momento tenemos la siguiente tabla:

Valor de n	**Valor de A_n**
0	1
1	1
2	1
3	2
4	5
5	11

Aquí no hay nada obvio sucediendo. ¿Qué sucede si le pedimos a Python que obtenga más valores?

```
renglon=[1]
```

```
print("El renglón {} tiene suma {}".format(renglon,sum(renglon)))

for j in range(1,15):
    new_renglon=[]
    for k in range(j+1):
        if k==0 or k==j:
            new_renglon.append(1)
        else:
            new_renglon.append(renglon[k-1]+renglon[k])
    renglon=new_renglon
    mult_3=renglon[0::3]
    print("El renglón {} saltando de 3 en 3 tiene suma {}".format(renglon,sum(mult_3)))
```

```
El renglón [1] tiene suma 1
El renglón [1, 1] saltando de 3 en 3 tiene suma 1
El renglón [1, 2, 1] saltando de 3 en 3 tiene suma 1
El renglón [1, 3, 3, 1] saltando de 3 en 3 tiene suma 2
El renglón [1, 4, 6, 4, 1] saltando de 3 en 3 tiene suma 5
El renglón [1, 5, 10, 10, 5, 1] saltando de 3 en 3 tiene suma 11
El renglón [1, 6, 15, 20, 15, 6, 1] saltando de 3 en 3 tiene suma 22
El renglón [1, 7, 21, 35, 35, 21, 7, 1] saltando de 3 en 3 tiene suma 43
El renglón [1, 8, 28, 56, 70, 56, 28, 8, 1] saltando de 3 en 3 tiene suma 85
El renglón [1, 9, 36, 84, 126, 126, 84, 36, 9, 1] saltando de 3 en 3 tiene suma 170
El renglón [1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1] saltando de 3 en 3 tiene suma 341
El renglón [1, 11, 55, 165, 330, 462, 462, 330, 165, 55, 11, 1] saltando de 3 en 3 tiene suma 683
El renglón [1, 12, 66, 220, 495, 792, 924, 792, 495, 220, 66, 12, 1] saltando de 3 en 3 tiene suma 1366
El renglón [1, 13, 78, 286, 715, 1287, 1716, 1716, 1287, 715, 286, 78, 13, 1] saltando de 3 en 3 tiene suma 2731
El renglón [1, 14, 91, 364, 1001, 2002, 3003, 3432, 3003, 2002, 1001, 364, 91, 14, 1] saltando de 3 en 3 tiene suma 5461
```

Tampoco hay nada muy obvio sucediendo. En ocasiones algunos problemas son así. No basta con explorar lo que nos están pidiendo, sino que además debemos explorar otros elementos del problema que debemos introducir por nuestra cuenta. Para este problema la clave es ver qué sucede no sólo con las sumas que nos interesan, sino también con aquellas cuando empezamos desfasados en \(1\) o en \(2\) elementos.

Así, tomemos \(B_n\) como la suma de los elementos del renglón \(n\) saltando de \(3\) en \(3\), pero comenzando en el primer elemento. La siguiente figura muestra qué números estamos sumando.

					1						
				1		**1**					
			1		**2**		1				
		1		**3**		3		1			
	1		**4**		6		4		**1**		
1		**5**		10		10		**5**		1	

Y tomemos \(C_n\) como la suma de los elementos del renglón \(n\) saltando de \(3\) en \(3\), pero comenzando en el segundo elemento. La siguiente figura muestra qué números estamos sumando.

					1						
				1		1					
			1		2		**1**				
		1		3		**3**		1			
	1		4		**6**		4		1		
1		5		**10**		10		5		**1**	

Ahora sí, exploremos conjuntamente a los valores de \(A_n\), \(B_n\) y \(C_n\). En la siguiente tabla puedes ver los valores para \(n\),

A_n, B_n y C_n.

Valor de n	Valor de A_n	Valor de B_n	Valor de C_n
0	1	0	0
1	1	1	0
2	1	2	1
3	2	3	3
4	5	5	6
5	11	10	11

Aunque no hayamos hecho cuentas computacionales, ¡el patrón comienza a revelarse! Parece ser que sucede todo lo siguiente:

- Para cada n, dos de los valores A_n, B_n y C_n son iguales y el tercero es distinto sólo en una unidad.
- Alternadamente, el distinto es mayor y menor.
- Ciclicamente el disinto es A_n, luego C_n, luego B_n.
- La suma de A_n con B_n es B_{n+1}.
- La suma de B_n con C_n es C_{n+1}.
- La suma de C_n con A_n es A_{n+1}.

Una vez más, dejaremos pendiente la demostración de estas conjeturas (todas ellas ciertas) y usaremos su validez de momento sin demostración. Lo que sugieren es que para cuando $n=100$, se tiene que el número distinto a los otros dos es C_{100} y que es una unidad más grande que A_{100} y B_{100}.

Pero además por el problema anterior sabemos algo crucial de A_{100}, B_{100} y C_{100}: su suma es 2^{100}. De esta manera, si $A_{100}=x$, entonces $B_{100}=x$ y $C_{100}=x+1$ y por lo tanto

$$3x+1=2^{100}.$$

De aquí obtenemos el valor $A_{100}=x=\frac{2^{100}-1}{3}$ que buscábamos. $\square$

Tarea moral

Los siguientes problemas te ayudarán a practicar lo visto en esta entrada. Para resolverlos, necesitarás usar herramientas matemáticas, computacionales o ambas.

1. Si se dibuja una recta en el plano, entonces queda dividido en dos regiones. Si se dibujan dos rectas (no paralelas), queda dividido en cuatro. Si de dibujan tres rectas (sin paralelas ni tripes intersecciones), queda dividido en siete. ¿Qué sucede con cuatro rectas? ¿Cuántas regiones nuevas se hacen? ¿Y si son en total 10 rectas? ¿Si son 10000?
2. Considera la sucesión a_n definida como sigue. El valor de a_0 es 0, el de a_1 es 1, el de a_2 es 2 y para $n\geq 0$ se cumple que $a_{n+3}=a_{n+2}-a_{n+1}+a_n$. ¿Sucederá en algún momento que a_n exceda 100? ¿Qué sucede si cambiamos los números iniciales a $a_0=3$, $a_1=5$ y $a_2=9$?
3. Hay otra cosa interesante que sucede con el triángulo de Pascal. En vez de sumar por renglones, podemos sumar por *diagonales*. Una diagonal del triángulo de Pascal consiste en comenzar con el primer número de un renglón, luego el segundo del renglón de arriba, luego el tercero del renglón de arriba y así sucesivamente hasta que ya no podamos sumar más.

   Un ejemplo de diagonal es el siguiente:

					1						

				1		**1**					
			1		2		1				
		1		3		3		1			
	1		4		6		4		1		
1		5		10		10		5		1	

Si sumamos en esta diagonal, da \(2\).

Aquí hay otro ejemplo de una diagonal:

					1						
				1		1					
			1		2		1				
		1		3		**3**		1			
	1		**4**		6		4		1		
1		5		10		10		5		1	

Si sumamos en esta diagonal, da \(8\).

¿Cuánto suman las entradas de la diagonal que comienza en el renglón \(n\)?

4. En el último problema de la entrada pudimos simplemente haberle pedido a Python de manera computacional que encontrara el valor buscado. Haz esto y verifica que coincide con la fórmula que encontramos. ¿Qué ventaja tiene entonces haber encontrado una fórmula?
5. ¿Será cierto que para cualquier entero positivo \(k\) la sucesión de Fibonacci tiene una infinidad de múltiplos de \(k\)? Modificando el código de este capítulo, realiza varios experimentos computacionales para conjeturar si es cierto o no.

Anterior
Fundamentos de combinatoria

Siguiente
Principio de las casillas

Por Leonardo Ignacio Martínez Sandoval