ALGORITMOS Y COMPLEJIDAD

ACTIVIDAD 12

Tomás Perotti - Mateo Medus - Gonzalo Giampietri

Los números de Catalan

Los números de Catalan forman una secuencia de números naturales que aparece en varios problemas de conteo que habitualmente son recursivos. El enésimo número de Catalán se obtiene aplicando la fórmula recursiva o su equivalente coeficiente binomial:

Fórmula recursiva:

$$C_n = \begin{cases} \text{si } n = 0 & \Rightarrow 1 \\ \text{si } n > 0 & \Rightarrow \dfrac{2(2n-1)}{n+1} C_{n-1} \end{cases}$$

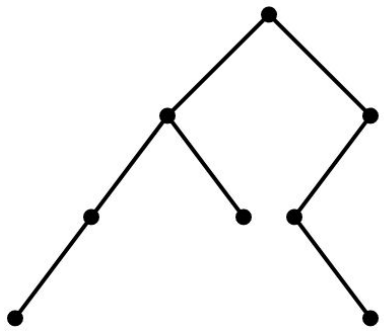
Coeficiente binomial:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!} \qquad \text{con } n \geq 0.$$

Por ejemplo, para los primeros 6 números los números de Catalan asociados son:
1 para n = 0 1 para n = 1
2 para n = 2 5 para n = 3
14 para n = 4 42 para n = 5

Dado que un árbol binario de búsqueda es un árbol binario, la demostración que se presenta para árboles binarios es análoga.

Un árbol binario es un árbol donde cada nodo tiene a lo sumo dos hijos (un hijo izquierdo y un hijo derecho).
Ejemplo de árbol binario:



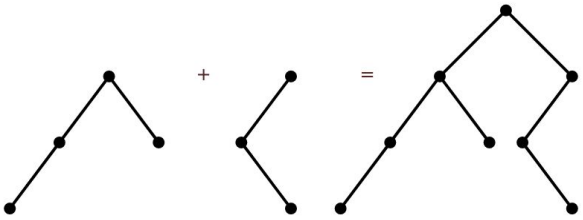
Para mostrar que C_n resuelve el número total de posibles árboles binarios para n nodos comenzamos asumiendo, por conveniencia, que permites árboles binarios vacíos tal que $C_0 = 1$. Luego, es fácil ver que:

$C_1 = 1$ 

$C_2 = 2$ 

$C_3 = 5$ 

Notar que cualquier árbol binario en al menos un nodo puede verse como dos árboles binarios (posiblemente vacíos) unidos en un nuevo árbol introduciendo un nuevo nodo raíz y haciendo que los hijos de esta raíz sean las dos raíces de los árboles originales. Ejemplo:

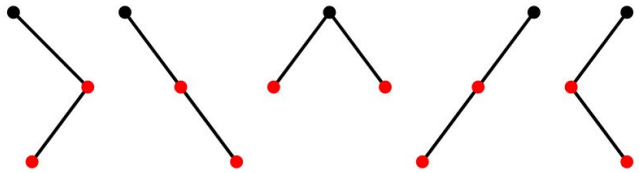


Luego, para obtener todos los posibles árboles binarios con n nodos, comenzamos con un nodo raíz, y luego para sus dos hijos insertamos otros dos árboles binarios con k y l nodos respectivamente, donde k + l = n-1. Dicho esto, se puede afirmar que:

$$C_n = \sum_{i=0}^{n-1} C_i C_{n-i-1}.$$

Por ejemplo, sabiendo que $C_0 = C_1 = 1$ y $C_2 = 2$,
$C_3 = C_0 * C_2 + C_1 * C_1 + C_2 * C_0 = 1*2 + 1*1 + 2*1 = 5$

Construcción: una vez conocidos los posibles árboles con 0,1 y 2 nodos, podemos combinarlos en todos las posibles maneras para obtener los diferentes árboles con 3 nodos.



Luego, para obtener todos los posibles árboles para n = 4 nodos, calculamos:
$C_4 = C_0 * C_3 + C_1 * C_2 + C_2 * C_1 + C_3 * C_0 = 1*5 + 1*2 + 2*1 + 5*1 = 14$.
El total de los posibles árboles con 4 nodos es: 14.

Algoritmo recursivo

```
Catalan(n)
   C_n = 0                                                    (1)

   if n = 0 ó n = 1 then                                      (2)
      C_n = 1                                                 (3)
   else
      for i = 0 to n-1
          C_n = C_n + (Catalan(i) * Catalan(n-i-1))           (4)

   end if.
   return C_n                                                 (5)
```

En mejor de los casos, n=1 ó n=2, para éstos casos el orden del tiempo de ejecución del algoritmo es constante. Cuando, n>2, el orden del tiempo de ejecución está sujeto a la cantidad de iteraciones del ciclo (4), la cual es de n veces.
Dicho esto, se analiza el orden del tiempo de ejecución del algoritmo en el peor de sus casos:

$T_{Catalan}(n) = T_1 + T_2 + T_4 = \sum^{n-1}_{i=o} (T_{catalan}(i) * T_{Catalan}(n-i-1)) + T_5$

$T_{Catalan}(n) = C + \sum^{n-1}_{i=o} (T_{catalan}(i) * T_{Catalan}(n-i-1))$, Donde C = constante.

Se puede observar que el tiempo de ejecución de $T_{Catalan}(n)$ está sujeto a $\sum^{n-1}_{i=o} (T_{catalan}(i) * T_{Catalan}(n-i-1))$).

$\sum^{n-1}_{i=o} (T_{catalan}(i) * T_{Catalan}(n-i-1)) =$

$\sum^{n-2}_{i=o} T_{catalan}(i) * T_{catalan}(n-i-1) + T_{catalan}(n-1) * T_{catalan}(n-n+1-1) =$

$1/n \binom{2i-2}{i-1} + T_{catalan}(n-1) = 1/n \binom{2(n-1)}{n-1} + T_{catalan}(n-1)$

Luego, El tiempo de ejecución de $T_{catalan}(n)$ está sujeto a $1/n \binom{2(n-1)}{n-1}$

$T_{catalan}(n) = 1/n \binom{2(n-1)}{n-1} >= (4^{(n-1)} / (n-1) -1) * 1/n =$

$4^n / 4(n^2-2) >= 4^n / 4n^2 \in \Omega(4^n/n^2)$

$T_{catalan}(n) \in \Omega(4^n/n^2)$

Algoritmo en programación dinámica (según fórmula recursiva)

Si bien el algoritmo anterior es correcto, toma un tiempo enorme para ser computado, si observamos bien como se calcula el N de catalán, nos podremos dar cuenta que existen múltiples subproblemas idénticos que se resuelven más de una vez, es decir, existe una superposición de problemas. Es por esto que, es deseable aplicar programación dinámica para disminuir el tiempo que requiere calcular nuevamente cada subproblema. Para lograr esto, guardaremos en una estructura especial, el valor de cada número de catalán que calculemos con anterioridad.

Inicializo Cat[0...N] con -1 de 0...N

Catalan(n)

```
if Cat[n] != -1 then
                return Cat[n]          (1)
Cn = 0
if  n == 0 then                        (2)
                 return Cn = 1
for i = 0 to N-1                       (3)
        Cn = Cn + C(i) * C(n-i-1)
Cat[n] = Cn
return Cn
```

$T_{Catalan}(n) = T_1 + T_2 + T_3 = \sum^{n-1}_{i=o} (T_{catalan}(i) * T_{Catalan}(n-i-1))$
$T_{Catalan}(n) = C + \sum^{n-1}_{i=o} (T_{catalan}(i) * T_{Catalan}(n-i-1))$, Donde C = constante.
$T_{Catalan}(n) = C + \sum^{n-1}_{i=o} (O(n))$
$T_{Catalan}(n) = O(n^2)$

Ahora, el **tiempo de ejecución** para calcular Catalan(n) es de orden $O(n^2)$, ya que, debemos computar todos los números de Catalan entre 0 y n-1 y, cada uno de estos, será computado solo una vez en tiempo lineal, por lo tanto, el tiempo de ejecución es de $O(n^2)$. Luego, el **espacio de ejecución** que toma el algoritmo es de O(n) lo cual proviene de almacenar los N números de catalán en la estructura auxiliar.

OPTIMALIDAD

Para probar la optimalidad de nuestra solución utilizaremos la hipótesis inductiva.
Para esto partimos de decir que para cualquier número n ingresado, se obtendrá el número de Catalan optimal C_n.

Para el caso base (n=0), el número de catalán será 1 (constante).

Para el caso general, asumimos para nuestra HI que C_{n-1} es optimal.
Luego, se observa que:

$$C_n = \sum_{i=1}^{n-2} (C_i * C_{n-i-1}) + C_0 * C_{n-1} + C_{n-1} * C_0$$
$$= \sum_{i=1}^{n-2} (C_i * C_{n-i-1}) + 1 * C_{n-1} + C_{n-1} * 1$$
$$= \sum_{i=1}^{n-2} (C_i * C_{n-i-1}) + 2 * C_{n-1}$$

Ahora, supongamos que C_n no es optimal, y sea C'_n el número n de Catalan optimal.

Si w es el resultado de a C'_n restarle $2C'_{n-1}$, podemos ver a w como el número de Catalán para n-1. Podemos repetir el cálculo de arriba y obtener que $C'_n = w + 2C'_{n-1}$.

Entonces, $C_n = w + 2C_{n-1} > C'_n - 2C_{n-1} = w$

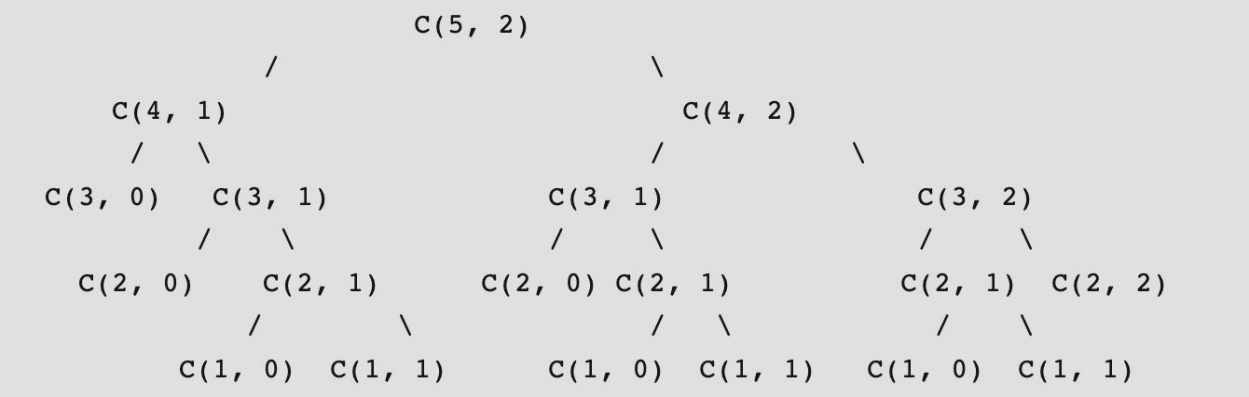
Por HI, C_{n-1} es optimal para n-1, entonces existe una contradicción que surgió de suponer que C_n no era optimal.
Por lo tanto, C_n es optimal para todo n>=0.

Algoritmo en programación dinámica (según coeficiente binomial)

El coeficiente binomial C(n,k) puede ser calculado de la siguiente forma recursivamente:

$$C(n,k) = \begin{cases} 1 & \text{si } C(n,0) \text{ ó } C(n,n) \\ C(n-1, k-1) + C(n-1, k) & \text{en otro caso} \end{cases}$$

Al realizar una traza de C(5,2), podemos notar rápidamente que existe superposición de problemas, con lo cual, es correcto aplicar programación dinámica a este problema.



Luego, si utilizamos la siguiente fórmula:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!} \qquad \text{con } n \geq 0.$$

En conjunto, con el algoritmo de coeficiente binomial, podemos calcular Catalan(n) de la siguiente manera utilizando programación dinámica:

Nota: se utilizará una estructura especial llamada **triángulo de Pascal**, para guardar los valores ya calculados.

```
Catalan(n)
        return      1/(n+1)   . CoBinomial(2n, n)

CoBinomial(n, k)
        array C[n+1][k+1]
        for i = 0 to N
                for j = 0 to min(i,k)
                        if (j == 0 or j == i)
                                C[i][j] = 1
                        else
                                C[i,j] = C[i-1, j-1] + C[i-1, j]          (1)
                endfor
        endfor
        return C[n,k]
```

Este algoritmo, toma un **tiempo de ejecución:**

$T_{Binomial}(n, k) = T_1 = \sum^{n}_{i=1} \sum^{min(i,k)}_{j=1}(C[i-1, j-1] + C[i-1, j])$
$T_{Binomial}(n, k) = \sum^{n}_{i=1} \sum^{min(i,k)}_{j=1}(O(1) + O(1))$
$T_{Binomial}(n, k) = \sum^{n}_{i=1} (O(k))$
$T_{Binomial}(n, k) = O(nk)$

Luego, su **espacio** se podría decir intuitivamente que también O(nk) debido a su estructura especial. Sin embargo, no es necesario guardar la tabla entera, sino que es suficiente con guardar un vector de tamaño k, representando la línea actual, y actualizar el vector de izquierda a derecha. Finalmente, el **espacio** es de **O(k)**.

OPTIMALIDAD

Por cada subproblema que encontramos, es decir, por cada C[i,j] debemos establecer cuál es la solución optimal para dicho subproblema, en este caso la solución optimal consiste en encontrar la solución a C[i-1, j-1] y C[i-1,j], en los casos no triviales, y luego sumarlos. Para demostrarlo desarrollaremos una traza siguiendo el algoritmo anteriormente mencionado:

```
Catalan(2)
        return (1/(2+1)) * CoBinomial(4, 2)
```

i	j	C[i][j]
0	0	C[0][0] = 1 (j == 0)
1	0	C[1][0] = 1 (j == 0)
	1	C[1][1] = 1 (j == i)
2	0	C[2][0] = 1
2	1	C[2][1] = C[1][0] (1) + C[1][1] (1) = **2**
	2	C[2][2] = 1 (j == i)
3	0	C[3][0] = 1
	1	C[3][1] = C[2][0] (1) + C[2][1] (2) = **3**
	2	C[3][2] = C[2][1] (2) + C[2][2] (1) = **3**
4	0	C[4][0] = 1
	1	C[4][1] = C[3][0] (1) + C[3][1] (3) = **4**
	2	C[4][2] = C[3][1] (3) + C[3][2] (3) = **6**

Luego, en cada iteración de la traza, calculamos C[i,j] exactamente una vez, almacenando los valores en la estructura, llevando a cabo una estrategia bottom-up característica en programación dinámica. Por lo tanto, por cada vez que debemos resolver un subproblema, o es trivial, como C[i][i] = 1 o puede calcularse utilizando los valores previamente calculados en la estructura, los cuales fueron calculados de forma optimal, y reemplazarlos en la siguiente fórmula para encontrar el resultado deseado: **C[i,j] = C[i-1, j-1] + C[i-1, j]**, simplemente basta con buscar ambos valores en la estructura especial y sumarlos. De esta manera, cada subproblema está conformado a su vez por soluciones optimales de sus respectivos subproblemas y encontramos una

subestructura optimal, el cual es uno de los principios que deben cumplirse en programación dinámica. La prueba formal de esto, es análoga a la anterior. Finalmente, como se indica en la traza C[4][2] es 6 y **Catalan(2)** = ⅓ * 6 = 2