GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: NÚMEROS CATALANES

1. Introducción

En combinatoria, los **números de Catalan** forman una secuencia de números naturales que aparece en varios problemas de conteo que habitualmente son recursivos.

La secuencia de números de Catalan fue descrita en el siglo XVIII por Leonhard Euler. La secuencia lleva el nombre de Eugène Charles Catalan, quien descubrió la conexión con las expresiones entre paréntesis durante su exploración del rompecabezas de las torres de Hanói.

En 1988, salió a la luz que la secuencia numérica de Catalan había sido utilizada en China por el matemático mongol Minggatu hacia 1730. Fue cuando comenzó a escribir su libro Ge Yuan Mi Lu Jie Fa *El método rápido para obtener la relación precisa de división de un círculo*, que fue completado por su alumno Chen Jixin en 1774, pero publicado sesenta años después.

De como calcular esta secuencia y su utilización en determinados problemas de programación competitiva tratará la siguiente guía.

2. Conocimientos previos

2.1. Eugène Charles Catalan

Eugène Charles Catalan (30 de mayo de 1814-14 de febrero de 1894) fue un matemático francés y belga que trabajó en la teoría de números.

Trabajó en fracciones continuas, geometría descriptiva, teoría de números y combinatoria. Dio su nombre a una superficie única (superficie periódica mínima en el espacio R^3) que descubrió en 1855. Anteriormente había enunciado la famosa conjetura de Catalan, que fue publicada en 1844 y probada finalmente en 2002 por el matemático rumano Preda Mihăilescu. Introdujo los números de Catalan para resolver un problema combinatorio.

2.2. Coeficientes binomiales

En matemáticas, los coeficientes binomiales, números combinatorios o combinaciones son números estudiados en combinatoria que corresponden al número de formas en que se puede extraer subconjuntos a partir de un conjunto dado. Sin embargo, dependiendo del enfoque que tenga la exposición, se pueden usar otras definiciones equivalentes.

2.3. Programación Dinámica

La programación dinámica es un método para reducir el tiempo de ejecución de un algoritmo mediante la utilización de subproblemas superpuestos y subestructuras óptimas.

La teoría de programación dinámica se basa en una estructura de optimización, la cual consiste en descomponer el problema en subproblemas (más manejables). Los cálculos se realizan entonces recursivamente donde la solución óptima de un subproblema se utiliza como dato de entrada al siguiente problema. Por lo cual, se entiende que el problema es solucionado en su totalidad, una vez se haya solucionado el último subproblema. Dentro de esta teoría, Bellman desarrolla el

Principio de Optimalidad, el cual es fundamental para la resolución adecuada de los cálculos recursivos. Lo cual quiere decir que las etapas futuras desarrollan una política óptima independiente de las decisiones de las etapas predecesoras. Es por ello, que se define a la programación dinámica como una técnica matemática que ayuda a resolver decisiones secuenciales interrelacionadas, combinándolas para obtener de la solución óptima.

3. Desarrollo

Hay dos fórmulas para los números catalanes: **recursiva** y **analítica**. Dado que creemos que todos los problemas donde se utilizan son equivalentes (tienen la misma solución), para la prueba de las fórmulas siguientes elegiremos la tarea que sea más fácil de realizar.

3.1. Fórmula recursiva

Los números catalanes se pueden calcular mediante la fórmula:

$$C_0 = C_1 = 1$$

$$C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}, n \geq 2$$

La suma pasa por las formas de dividir la expresión en dos partes de modo que ambas partes sean expresiones válidas y la primera parte sea lo más corta posible pero no vacía. Para cualquier i, la primera parte contiene i + 1 pares de paréntesis y el número de expresiones es el producto de los siguientes valores:

- C_i: el número de formas de construir una expresión usando los paréntesis de la primera parte, sin contar los paréntesis más externos.
- C_{n-i-1}: el número de formas de construir una expresión usando los paréntesis de la segunda parte.

El caso base es $C_0 = 1$, porque podemos construir una expresión entre paréntesis vacía usando cero pares de paréntesis.

La fórmula de recurrencia se puede deducir fácilmente del problema de la secuencia correcta de corchetes.

El paréntesis de apertura más a la izquierda l corresponde a cierto corchete de cierre r, que divide la secuencia en 2 partes que a su vez deberían ser una secuencia correcta de corchetes. Así, la fórmula también se divide en 2 partes. Si denotamos $k = r - l - 1$, entonces para r fijo, habrá exactamente $C_i C_{n-1-i}$ tales secuencias de corchetes. Sumando esto sobre todos los i admisibles, obtenemos la relación de recurrencia en C_n.

También puedes pensarlo de esta manera. Por definición, C_n denota el número de secuencias de corchetes correctas. Ahora, la secuencia se puede dividir en 2 partes de longitud i y $n-i$, cada una de las cuales debe ser una secuencia de corchetes correcta. Ejemplo:

$()(())$ se puede dividir en $()$ y $(())$, pero no se puede dividir en $)($ y $())$. Nuevamente sumando todos los i admisibles, obtenemos la relación de recurrencia en C_n.

3.2. Fórmula analítica

Los números catalanes también se pueden calcular mediante coeficientes binomiales:

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

aquí $\binom{n}{k}$ denota el coeficiente binomial habitual, es decir, el número de formas de seleccionar k objetos de un conjunto de n objetos).

La fórmula se puede explicar como sigue:

Hay un total de $\binom{2n}{n}$ formas de construir una expresión entre paréntesis (no necesariamente válida) que contenga n paréntesis izquierdos y n paréntesis derechos. Calculemos el número de expresiones que no son válidas.

Si una expresión entre paréntesis no es válida, debe contener un prefijo donde el número de paréntesis derechos exceda el número de paréntesis izquierdos. La idea es invertir cada paréntesis que pertenece a dicho prefijo. Por ejemplo, la expresión $())()($ contiene un prefijo $())$ y, después de invertir el prefijo, la expresión se convierte en $)((()($.

La expresión resultante consta de $n+1$ paréntesis izquierdo y $n-1$ paréntesis derecho. El número de tales expresiones es $\binom{2n}{n+1}$, que es igual el número de expresiones entre paréntesis no válidas. Por tanto, el número de expresiones entre paréntesis válidas se puede calcular utilizando la fórmula:

$$\binom{2n}{n} - \binom{2n}{n+1} = \binom{2n}{n} - \frac{1}{n+1}\binom{2n}{n} = \frac{1}{n+1}\binom{2n}{n}$$

Otra forma de enteder la expresión anterior es a partir del problema de los caminos monótonos en una cuadrícula. El número total de caminos monótonos en el tamaño de la matriz de $n \times n$ está dado por $\binom{2n}{n}$.

Ahora contamos el número de caminos monótonos que cruzan la diagonal principal. Considere los caminos que cruzan la diagonal principal y encuentre el primer borde que está por encima de la diagonal. Refleja el camino alrededor de la diagonal hasta el final, siguiendo este borde. El resultado es siempre un camino monótono en la cuadrícula $(n-1)\times(n+1)$. Por otro lado, cualquier camino monótono en la red $(n-1) \times (n+1)$ debe cruzar la diagonal. Por lo tanto, enumeramos todos los caminos monótonos que cruzan la diagonal principal en la red $n \times n$.

El número de caminos monótonos en la red $(n-1)\times(n+1)$ son $\binom{2n}{n-1}$. Llamemos a esos caminos *malos*. Como resultado, para obtener el número de caminos monótonos que no cruzan la diagonal principal, restamos los caminos *malos* anteriores, obteniendo la fórmula:

$$C_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{1}{n+1}\binom{2n}{n}, n \geq 0$$

Si se nos pide que calculemos los valores de C(n) para varios valores de n, puede ser mejor calcular los valores usando DP (de abajo hacia arriba). Si conocemos C(n), podemos calcular C(n+1) manipulando la fórmula como se muestra a continuación.

$$C_n = \frac{(2n)!}{n!\times n!\times(n+1)}$$

$$C_{n+1} = \frac{(2\times(n+1))!}{(n+1)!\times(n+1)!\times((n+1)+1)}$$

$$\frac{(2n+2)\times(2n+1)\times(2n)!}{(n+1)\times n!\times(n+1)\times n!\times(n+2)}$$

$$\frac{(2\times(n+1))\times(2n+1)\times[(2n)!]}{(n+2)\times(n+1)\times[n!\times n!\times(n+1)]}$$

$$\frac{4n+2}{n+2}\times C_n$$

3.3. Hallar el inesimo Catalan

Para el cálculo del los números catalanes veremos varios enfoques a partir de las fórmulas analizadas previamente.

3.3.1. Función recursiva

Siga los pasos a continuación para implementar la fórmula recursiva anterior:

- Condición base para el enfoque recursivo, cuando $n \leq 1$, devuelve 1.
- Iterar desde $i = 0$ hasta $i < n$
  - Haga una llamada recursiva a $cataln(i)$ y $cataln(n-i-1)$ y siga sumando el producto de ambos en res.
- Devuelve res

3.3.2. Programación Dinámica

Podemos observar que la implementación recursiva anterior realiza mucho trabajo repetido. Dado que hay subproblemas superpuestos, podemos utilizar la programación dinámica para ello.

A continuación se muestra la implementación de la idea anterior:

- Cree una matriz $catalan[]$ para almacenar el i-ésimo número catalán.

- Inicializar, $cataln[0]$ y $cataln[1] = 1$.
- Recorre $i = 2$ hasta el número catalán dado n.
  - Recorra $j = 0$ hasta $j < i$ y siga agregando el valor de $catalan[j] * catalan[i - j - 1]$ en $cataln[i]$.
- Finalmente, regresa $catalan[n]$

3.3.3. Coeficiente Binomiales

A continuación se detallan los pasos para calcular nC_r.

- Cree una variable para almacenar la respuesta y cambie r a $n - r$ si r es mayor que $n - r$ porque sabemos que $C(n, r) = C(n, n - r)$ si $r > n - r$
- Ejecute un bucle de 0 a $r - 1$
  - En cada iteración, actualice ans como $(ans * (n - i))/(i + 1)$, donde i es el contador de bucle.
- Entonces la respuesta será igual a $((n/1) \times ((n - 1)/2) \times \cdots \times ((n - r + 1)/r)$, que es igual a nC_r.

A continuación se detallan los pasos para calcular números catalanes usando la fórmula: $2n\frac{C_n}{(n+1)}$

- Calcule $2nC_n$ usando pasos similares a los que usamos para calcular nC_r
- Retorne el valor de $2n\frac{C_n}{(n+1)}$

Encontrar valores de números catalanes para $N > 80$ no es posible ni siquiera usando **long** , por lo que usamos **BigInteger**. Para poder usar **long** sería posible si la respuesta estuviese modulada a un valor m.

4. Implementación

4.1. C++

4.1.1. Función recursiva

```
unsigned long long catalan(unsigned long long n){
   if (n <= 1) return 1;

   unsigned long long res = 0;
   for (unsigned long long i = 0; i < n; i++)
      res += catalan(i) * catalan(n - i - 1);
   return res;
}
```

4.1.2. Programación Dinámica

```
unsigned long long catalanDP(unsigned long long n){
   unsigned long long catalan[n + 1];
   catalan[0] = catalan[1] = 1;

   for (unsigned long long i = 2; i <= n; i++) {
      catalan[i] = 0;
      for (int j = 0; j < i; j++)
         catalan[i] += catalan[j] * catalan[i-j-1];
   }
   return catalan[n];
}

unsigned long long catalanDPV2(unsigned long long n){
   unsigned long long catalan[n + 3];
   catalan[0] = 1;
   for (unsigned long long i = 0; i <= n; i++)
      catalan[i+1] = catalan[i]* (4*i+2) / (i+2);
   return catalan[n];
}
```

4.1.3. Coeficiente Binomiales

```
unsigned long long binomialCoeff(unsigned long long n, unsigned long long k) {
   unsigned long long res = 1;

   if (k>n-k) k =n-k;

   for (unsigned long long i = 0; i < k; ++i) {
      res *= (n - i); res /= (i + 1);
   }
   return res;
}

unsigned long long catalan(unsigned long long n){
   unsigned long long c = binomialCoeff(2*n,n);
   return c/(n+1);
}
```

4.1.4. Coeficiente Binomiales *BigInteger*

```
// asumimos que el bigint ya esta implementado
bigint findCatalan(int n){
   bigint b = 1;
   // calculando n!
   for (int i = 1; i <= n; i++) b = b * i;
```

```
    // calculando n! * n!
    b = b * b;

    bigint d = 1;

    // calculando (2n)!
    for (int i = 1; i <= 2 * n; i++) d = d * i;

    // calculando (2n)! / (n! * n!)
    bigint ans = d / b;

    // calculando (2n)! / ((n! * n!) * (n+1))
    ans = ans / (n + 1);
    return ans;
}
```

4.2. Java

4.2.1. Función recursiva

```
public static long catalan(long n){
    int res = 0;
    if (n <= 1L) return 1L;

    for (long i = 0; i < n; i++)
        res += catalan(i) * catalan(n - i - 1);
    return res;
}
```

4.2.2. Programación Dinámica

```
public static long catalanDP(long n){
    long catalan[] = new long[n + 2];

    catalan[0] = 1; catalan[1] = 1;

    for (long i = 2; i <= n; i++) {
        catalan[i] = 0;
        for (long j = 0; j < i; j++) {
            catalan[i] += catalan[j] * catalan[i - j - 1];
        }
    }
    return catalan[n];
}

public static long catalanDPV2(long n){
```

```
   long catalan[] = new long[n + 3];
   catalan[0] = 1;
   for (long i = 0; i <= n; i++)
      catalan[i+1] = catalan[i]* (4*i+2) / (i+2);
   return catalan[n];
}
```

4.2.3. Coeficiente Binomiales

```
public static long binomialCoeff(long n, long k){
   long res = 1;

   if (k > n - k) { k = n - k; }

   for (long i = 0; i < k; ++i) {
      res *= (n - i);
      res /= (i + 1);
   }
   return res;
}

public static long catalan(int n){
   long c = binomialCoeff(2 * n, n);
   return c / (n + 1);
}
```

4.2.4. Coeficiente Binomiales *BigInteger*

```
public static BigInteger findCatalan(int n){
   // usando BigInteger para calcular factoriales largos
   BigInteger b = new BigInteger("1");

   // calculando n!
   for (int i = 1; i <= n; i++)
      b = b.multiply(BigInteger.valueOf(i));

   // calculando n! * n!
   b = b.multiply(b);

   BigInteger d = new BigInteger("1");

   // calculando (2n)!
   for (int i = 1; i <= 2 * n; i++)
      d = d.multiply(BigInteger.valueOf(i));

   // calculando (2n)! / (n! * n!)
   BigInteger ans = d.divide(b);
```

```
    // calculando (2n)! / ((n! * n!) * (n+1))
    ans = ans.divide(BigInteger.valueOf(n + 1));
    return ans;
}
```

5. Aplicaciones

Existen múltiples problemas de combinatoria en la programación competitiva cuya solución la dan los números de Catalan. El libro *Enumerative Combinatorics: Volume 2*, de Richard P. Stanley contiene un conjunto de ejercicios que describen 66 interpretaciones distintas de los números de Catalan. Aquí se muestran algunos ejemplos:

1. C_n es el número de palabras de Dyck de longitud $2n$. Una palabra de Dyck es una cadena de caracteres que consiste en n X's y n Y's de forma que no haya ningún segmento inicial que tenga más Y's que X's. Por ejemplo, lo siguiente son las palabras de Dyck de longitud 6:

$$XXXYYY \quad XYXXYY \quad XYXYXY \quad XXYYXY \quad XXYXYY$$

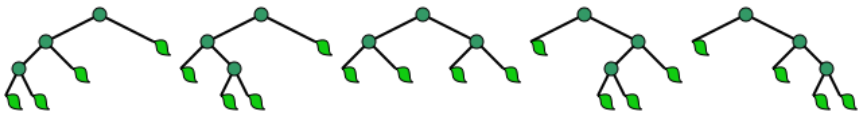
2. Reinterpretando el símbolo X como un paréntesis abierto y la Y como un paréntesis cerrado, C_n cuenta el número de expresiones que contienen n pares de paréntesis correctamente colocados:

$$((())) \quad ()(()) \quad ()()() \quad (())() \quad (()())$$

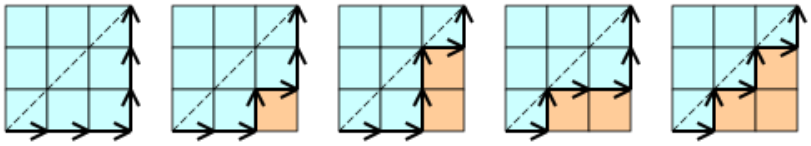
3. C_n es el número de formas distintas de agrupar $n+1$ factores mediante paréntesis (o el número de formas de asociar n aplicaciones de un operador binario). Para $n = 3$ por ejemplo, tenemos las siguientes cinco formas distintas de agrupar los cuatro factores:

$$((ab)c)d \quad (a(bc))d \quad (ab)(cd) \quad a((bc)d) \quad a(b(cd))$$

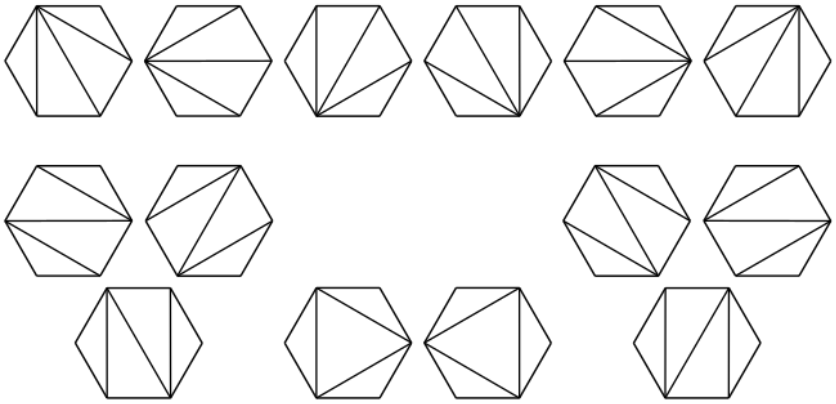
4. Las aplicaciones sucesivas de un operador binario pueden representarse con un árbol binario. En este caso, C_n es el número de árboles binarios de $n+1$ hojas, en los que cada nodo tiene cero o dos hijos:



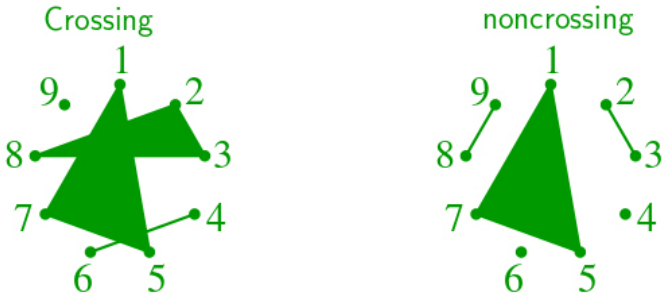
5. C_n es el número de **caminos monótonos** que se pueden trazar a través de las líneas de una malla de $n \times n$ celdas cuadradas, de forma que nunca se cruce la diagonal. Un camino monótono es aquel que empieza en la esquina inferior izquierda y termina en la esquina superior derecha, y consiste únicamente en tramos que apuntan hacia arriba o hacia la derecha. El recuento de estos caminos es equivalente a contar palabras de Dyck: X significa *moverse a la derecha* e Y significa *moverse hacia arriba*. Los siguientes diagramas muestran el caso n = 3:



6. C_n es el número de formas distintas de dividir un polígono convexo de $n+2$ lados en triángulos conectando vértices con diagonales sin que ninguna se corte. La siguiente figura ilustra el caso de las $c_4 = 14$ posibles triangulaciones para un polígono de 6 lados:



7. Número de particiones que no se cruzan del conjunto $\{1, \ldots, 2n\}$ en el que cada bloque es de tamaño 2. Una partición no se cruza si y solo si en su diagrama plano, los bloques están disjuntos (es decir, no se cruzan). Por ejemplo, debajo de dos hay particiones cruzadas y no cruzadas de $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. La partición $\{\{1, 5, 7\}, \{2, 3, 8\}, \{4, 6\}, \{9\}\}$ se cruza y la partición $\{\{1, 5, 7\}, \{2, 3\}, \{4\}, \{6\}, \{8, 9\}\}$ no se cruza.

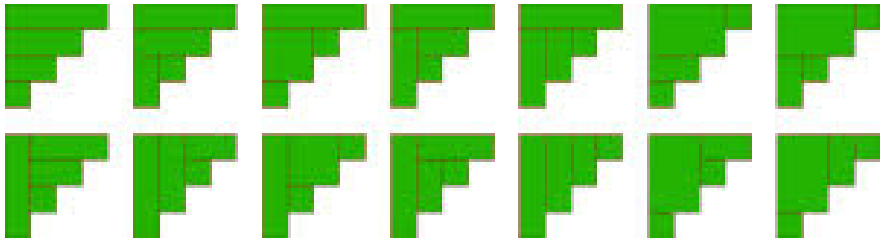


8. Número de permutaciones de longitud n que se pueden ordenar por pila (es decir, se puede demostrar que el reordenamiento se ordena por pila si y solo si no existe tal índice $i < j < k$, tal que $a_k < a_i < a_j$.

9. El número de formas de conectar los puntos $2n$ en un círculo para formar n cuerdas disjun-

tas.

10. El número de árboles binarios completos no isomorfos con n nodos internos (es decir, nodos que tienen al menos un hijo).
11. El número de formas de cubrir la escalera $1 \dots n$ usando n rectángulos (la escalera consta de n columnas, donde i^{th} columna tiene una altura i). La siguiente figura ilustra el caso n = 4:



12. Número de formas de formar una cordillera con n tramos ascendentes y n tramos descendentes que se mantienen por encima de la línea original. La interpretación de la cadena montañosa es que las montañas nunca bajarán del horizonte.
13. Puede haber una cantidad de árboles binarios sin etiquetar diferentes con n nodos.
14. Número de permutaciones de $\{1, \dots, n\}$ que evitan el patrón 123 (o cualquiera de los otros patrones de longitud 3); es decir, el número de permutaciones sin una subsecuencia creciente de tres términos. Para $n = 3$, estas permutaciones son $132, 213, 231, 312$ y 321. Para $n = 4$, son $1432, 2143, 2413, 2431, 3142, 3214, 3241, 3412, 3421, 4132, 4213, 4231, 4312$ y 4321

6. Complejidad

En esta guía vimos tres vias de implementar como calcular el enésimo número de catalán analicemos las complejidades de cada una. La primera variante es la recursiva dicha implementación tiene una complejidad temporal exponencial producto que valor del enésimo número catalán es exponencial calcularlo, lo que hace que la complejidad del tiempo sea exponencial, en cuenta a la complejidad espacial es O(N). La segunda variante utilizando programación dinámica reduce la complejidad temporal O(N^2) para la primera implementación mientras en la segunda implementación su complejidad es O(N) mientras la complejidad temporal se mantiene igual en O(N). En la tercera variante con el uso de coeficientes binomiales la complejidad temporal se mantiene en O(N) pero la complejidad espacial se reduce hasta O(1).

7. Ejercicios

A continuación una lista de ejercicios que se pueden resolver a través de los números catalanes:

- CSES - Bracket Sequences I
- UVA - 12887 - The Soldier's Dilemma

- UVA - 991 - Safe Salutations
- SPOJ - SKYLINE - Skyline
- CodeChef - Stacking Pancakes
- CodeForce - D. How many trees?
- LighOJ - Counting Perfect BST