|  |  |
| --- | --- |
| **n** |  |
| 0 | 1 |
| 1 | 1 |
| 2 | 2 |
| 3 | 5 |
| 4 | 14 |
| 5 | 42 |
| 6 | 132 |
| 7 | 429 |
| 8 | 1.430 |
| 9 | 4.862 |
| 10 | 16.796 |
| 11 | 58.786 |
| 12 | 208.012 |
| 13 | 742.900 |
| 14 | 2.674.440 |
| 15 | 9.694.845 |
| 16 | 35.357.670 |
| 17 | 129.644.790 |
| 18 | 477.638.700 |
| 19 | 1.767.263.190 |
| 20 | 6.564.120.420 |
| 21 | 24.466.267.020 |
| 22 | 91.482.563.640 |
| 23 | 343.059.613.650 |
| 24 | 1.289.904.147.324 |
| 25 | 4.861.946.401.452 |

**Números de Catalán**

El n-ésimo número de Catalán se obtiene, aplicando coeficientes binomiales, a partir de la siguiente fórmula:

**Propiedades**

Una expresión alternativa para Cn es

Los números de Catalán satisfacen la siguiente relación de recurrencia:

Y también satisfacen:

que puede ser una forma más eficiente de calcularlos.

La expresión en forma de recursión sería:

|  |
| --- |
|  |

**Aplicaciones en combinatoria**

Aquí se muestran algunos ejemplos, con ilustraciones para el caso C3 = 5.

Cn 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 XYXXYY XYXYXY XXYYXY XXYXYY

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

((())) ()(()) ()()() (())() (()())

Cn 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 (a(bc))d (ab)(cd) a((bc)d) a(b(cd))

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

Cn es el número de caminos monótonos que se pueden trazar a través de las líneas de una malla de n × n celdas cuadradas, de forma que nunca se cruce la diagonal. Un camino monótono es aquél 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”.

Cn es el número de formas distintas de cortar un polígono convexo de n + 2 lados en triángulos conectando vértices con líneas rectas sin que ninguna se corte.

**Sucesión de Fibonacci**

**Forma matricial**

**Propiedades de la sucesión**

Cualquier número natural se puede escribir mediante la suma de un número limitado de términos de la sucesión de Fibonacci, cada uno de ellos distinto a los demás.

Tan sólo un término de cada tres es par, uno de cada cuatro es múltiplo de 3, uno de cada cinco es múltiplo de 5.

Cada número de Fibonacci es el promedio del término que se encuentra dos posiciones antes y el término que se encuentra una posición después. Es decir

Lo anterior también puede expresarse así: calcular el siguiente número a uno dado es 2 veces éste número menos el número 2 posiciones más atrás.

La suma de los n primeros números es igual al número que ocupa la posición n+2 menos uno. Es decir

Otras identidades interesantes incluyen las siguientes:

Si , entonces , para cualquier

El máximo común divisor de dos números de Fibonacci es otro número de Fibonacci. Más específicamente

Si , tal que a es un número primo, entonces p también es un número primo, excepto, , 3 es un número primo, pero 4 no lo es.

La suma de diez números Fibonacci consecutivos es siempre 11 veces superior al séptimo número de la serie.

El último dígito de cada número se repite periódicamente cada 60 números. Los dos últimos, cada 300; a partir de ahí, se repiten cada números.

**// Convex Hull**

**const** **int** MAX = 100;

**struct** pto{

**int** x, y;

} man[MAX];

**int** n, lim, s, pt[100];

**int** sol[100];

**bool** comp(**const** **int** &i, **const** **int** &j){

if(man[i].x!=man[j].x) **return** man[i].x<man[j].x;

**return** man[i].y<man[j].y;

}

**int** cross(**int** p1, **int** p2, **int** p3){

**return** (man[p2].x-man[p1].x)\*(man[p3].y-man[p1].y)-

(man[p3].x-man[p1].x)\*(man[p2].y-man[p1].y);

}

**int** main(){

**scanf**("%d", &n);

**for**(int i=0; i<n; i++){

scanf("%d%d", &man[i].x, &man[i].y);

pt[i]=i;

}

**sort**(pt, pt+n, comp);

lim=1;

**for**(**int** i=0; i<n; sol[s++]=pt[i++])

**while**(s>lim && cross(sol[s-2], sol[s-1], pt[i])<0)

s--;

lim=s;

**for**(**int** i=n-2; i>=0; sol[s++]=pt[i--])

**while**(s>lim && cross(sol[s-2], sol[s-1], pt[i])<0)

s--;

s--;

**for** ( **int** i=0; i<=s; i++)

printf("%d ", sol[i]);

**return** 0;

}