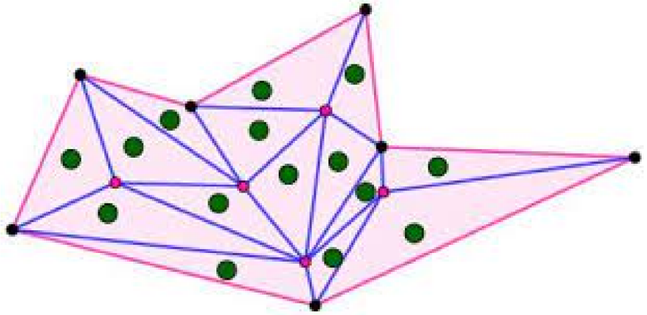
Les Nombres de Catalan

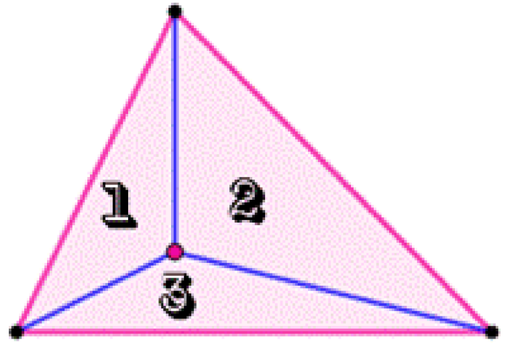
Les Nombres de Catalan

Sommaire :

Leonhard Euler

Portrait par Johann Georg Brucker (1756).





Eugène Charles Catalan

Eugène Charles Catalan, par Émile Delperée

Le nombre de Catalan d'indice n est défini par :

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

Pour $n \geqslant 2$, on peut écrire :

$$C_n = \prod_{k=2}^{n} \frac{n+k}{k} = \frac{(n+2)(n+3)\cdots 2n}{2.3\cdots n}$$

(voir Coefficient binomial central).

Les dix premiers nombres de Catalan (pour n de 0 à 9) sont :

n	0	1	2	3	4	5	6	7	8	9
C_n	1	1	2	5	14	42	132	429	1 430	4 862

(pour les 1 001 premiers, voir les liens de la suite A000108 de l'OEIS).

Énoncé de mon projet

Implémentations:

Implémentations:

- Récursive

$$C_0 = 1$$
$$C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

Implémentations:

- Récursive

$$C_0 = 1$$
$$C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

- Explicite

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

Implémentations:

- Récursive

$$C_0 = 1$$
$$C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

- Explicite

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

- Générative

$$C(x) = \sum_{n=0}^{\infty} C_n x^n = \frac{1-\sqrt{1-4x}}{2x}$$

Implémentations:

- Récursive

$$C_0 = 1$$
$$C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

- Explicite

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

- Générative

$$C(x) = \sum_{n=0}^{\infty} C_n x^n = \frac{1-\sqrt{1-4x}}{2x}$$

- Hypergéométrique

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

Implémentations:

- Récursive

$$C_0 = 1$$
$$C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

- Explicite

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

- Générative

$$C(x) = \sum_{n=0}^{\infty} C_n x^n = \frac{1-\sqrt{1-4x}}{2x}$$

- Hypergéométrique

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

- Intégrale

$$C_n = \frac{1}{n+1}\int_0^1 t^n(1-t)^n\,dt$$

Implémentations:

- Récursive

$$C_0 = 1$$
$$C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

- Explicite

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

- Générative

$$C(x) = \sum_{n=0}^{\infty} C_n x^n = \frac{1-\sqrt{1-4x}}{2x}$$

- Hypergéométrique

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

- Intégrale

$$C_n = \frac{1}{n+1}\int_0^1 t^n(1-t)^n\,dt$$

- Série infinie

$$C_n = \sum_{k=0}^{n} \frac{1}{k+1}\binom{n}{k}^2$$

Récursive:

```
unsigned long long int recursive(int N) {
    if (N <= 1)
        return 1;
    unsigned long long int somme = 0;
    int index;
    for (index = 0; index < N; index += 1)
        somme += recursive(index) * recursive(N - index - 1);
    return somme;
}
```

$$C_0 = 1$$

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

Récursive:

```c
unsigned long long int iterative(int n) {
    unsigned long long int *nombresCatalan = malloc((sizeof *nombresCatalan) * (n + 1));
    unsigned long long int resultat;
    int indexcourant, indexPP;
    nombresCatalan[0] = nombresCatalan[1] = 1;
    for (indexcourant = 2; indexcourant <= n; indexcourant += 1) {
        nombresCatalan[indexcourant] = 0;
        for (indexPP = 0; indexPP < indexcourant; indexPP += 1) {
            nombresCatalan[indexcourant] += nombresCatalan[indexPP] * nombresCatalan[indexcourant - indexPP - 1];
        }
    }
    resultat = nombresCatalan[n];
    free(nombresCatalan);
    return resultat;
}
```

$$C_0 = 1$$

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

Explicite:

```
unsigned long long int factoriel(int N) {
    if (N < 0) return 0;
    if (N == 0) return 1;
    return (unsigned long long int)(factoriel_calcule(2 * N) / ((double)factoriel_calcule(N +
    1) * factoriel_calcule(N)));
}
unsigned long long int factoriel_calcule(int N) {
    if (N <= 1) return 1;
    else return (unsigned long long int)N * factoriel_calcule(N - 1);
}
```

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

Explicite:

```
unsigned long long int combinatoire_interpreter(int N) {
    unsigned long long int result = 1;
    int i;
    for (i = 0; i < N; ++i) {
        result *= (2 * N - i);
        result /= (i + 1);
    }
    return result / (N + 1);
}
```

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

Générative:

```cpp
unsigned long long int generative(int N) {
    unsigned long long int result = 1;
    int i;
    for (i = 1; i <= N; ++i) {
        result *= (1 - sqrt(1 - 4.0 / (i * i)));
    }
    return result / 2.0;
}
```

$$C(x) = \sum_{n=0}^{\infty} C_n x^n = \frac{1-\sqrt{1-4x}}{2x}$$

Hypergéométrique:

```cpp
// Coef Binomial utile pour la suite
unsigned long long int coef_bin(int n, int k) {
    if (k == 0 || k == n) {
        return 1;
    } else {
        return coef_bin(n - 1, k - 1) * n / k;
    }
}

// Hypergeometric
unsigned long long int hypergeometric(int n) {
    return coef_bin(2 * n, n) / (n + 1);
}
```

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

Intégral:

```cpp
// Coef Binomial utile pour la suite
unsigned long long int coef_bin(int n, int k) {
    if (k == 0 || k == n) {
        return 1;
    } else {
        return coef_bin(n - 1, k - 1) * n / k;
    }
}
// Expression intégrale
unsigned long long int integral(int n) {
    float result = 1.0;
    for (int k = 1; k <= n; ++k) {
        result *= (2.0 * k - 1) / (n + k);
    }
    return (unsigned long long int) result;
}
```

$$C_n = \frac{1}{n+1} \int_0^1 t^n (1-t)^n \, dt$$

série infini:

```c
// Coef Binomial utile pour la suite
unsigned long long int coef_bin(int n, int k) {
    if (k == 0 || k == n) {
        return 1;
    } else {
        return coef_bin(n - 1, k - 1) * n / k;
    }
}

// Série infinie
unsigned long long int infini(int n) {
    unsigned long long int result = 0;
    int k;
    for (k = 0; k <= n; k++) {
        result += coef_bin(n, k) * coef_bin(n, k) / (k + 1);
    }
    return result;
}
```

$$C_n = \sum_{k=0}^{n} \frac{1}{k+1} \binom{n}{k}^2$$

Parenthèse:

```c
void generateParenthesesRec(char current[], int left, int right, const int N) {
    if (left + right == 2 * N) {
        printf("%s\n", current);
        return;
    }
    if (left < N) {
        current[left + right] = '(';
        generateParenthesesRec(current, left + 1, right, N);
    }
    if (right < left) {
        current[left + right] = ')';
        generateParenthesesRec(current, left, right + 1, N);
    }
}
void parenthese_rec(const int N) {
    char current[2 * N + 1];  // +1 for the null terminator
    current[2 * N] = '\0';    // null terminator
    generateParenthesesRec(current, 0, 0, N);
}
```

```
Execution time test Catalan -> recursive:
index_execution_time ( 18 ) : 1.426484s > 0.50s -> arret du test.
Catalan -> recursive | 0 -> 18 ( 0.018% ) | overall_execution_time: 0.706954s.


Execution time test Catalan -> iterative:
overall_execution_time ( 5103 ): 60.012802s > 60s -> arret du test.
Catalan -> iterative | 0 -> 5103 ( 5.103% )  | overall_execution_time: 60.012802s.


Execution time test Catalan -> factoriel:
overall_execution_time ( 72199 ): 60.002823s > 60s -> arret du test.
Catalan -> factoriel | 0 -> 72199 ( 72.199% )  | overall_execution_time: 60.002823s.


Execution time test Catalan -> combinatoire_interpreter:
Catalan -> combinatoire_interpreter | 0 -> 100000 ( 100.000% )  | overall_execution_time: 46.931797s.


Execution time test Catalan -> generative:
Catalan -> generative | 0 -> 100000 ( 100.000% )  | overall_execution_time: 21.873438s.


Execution time test Catalan -> hypergeometric:
Catalan -> hypergeometric | 0 -> 100000 ( 100.000% )  | overall_execution_time: 58.210899s.


Execution time test Catalan -> infini:
overall_execution_time ( 2462 ): 60.046410s > 60s -> arret du test.
Catalan -> infini | 0 -> 2462 ( 2.462% )  | overall_execution_time: 60.046410s.


Execution time test Catalan -> integral:
Catalan -> integral | 0 -> 100000 ( 100.000% )  | overall_execution_time: 22.673565s.
```

```
Execution time test:
()
Parenthese -> recursive ( 1 ) : 0.000002s

(())
()()
Parenthese -> recursive ( 2 ) : 0.000002s

((()))
(()())
(())()
()(())
()()()
Parenthese -> recursive ( 3 ) : 0.000003s

(((())))
(((()())))
(((())()))
(((()))())
((()(())))
((()()()))
((()())())
((())(()))
((())()())
(()((())))
(()(()()))
(()(())())
(()()(()))
(()()()())
Parenthese -> recursive ( 4 ) : 0.000047s
```

```
( 13 ) : 3.062688 supérieur à 2.000000s -> arret du test.
Parenthese -> recursive | overall_execution_time: 1.167974s.
```

Avez vous des questions ?