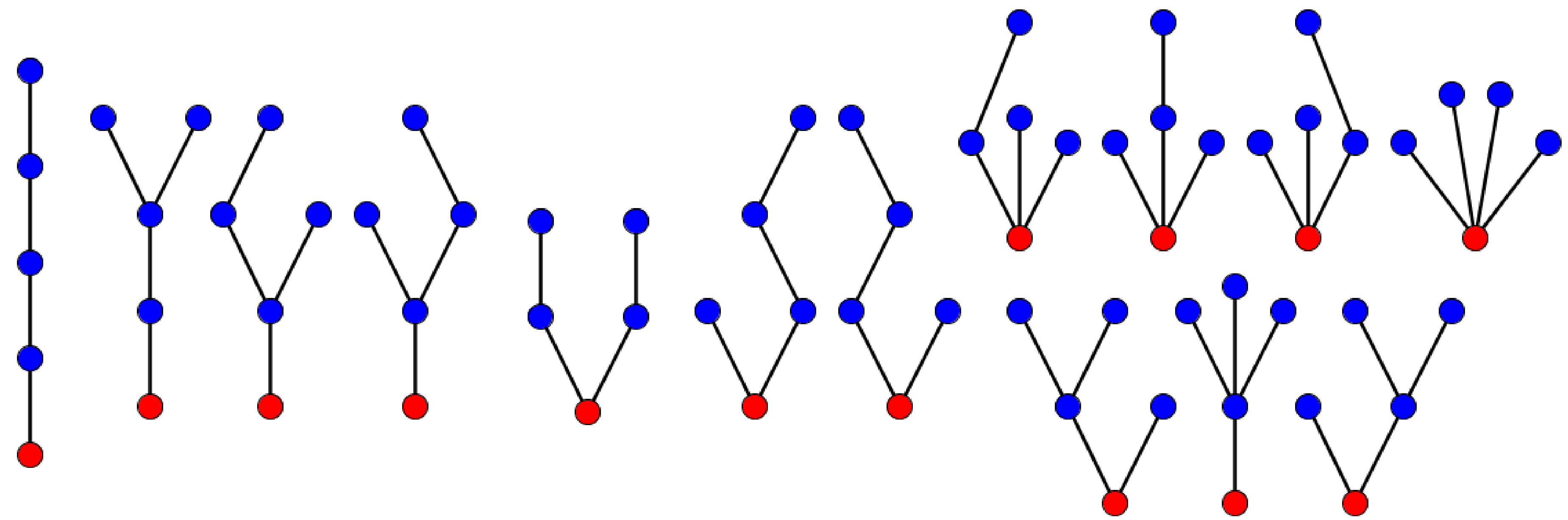
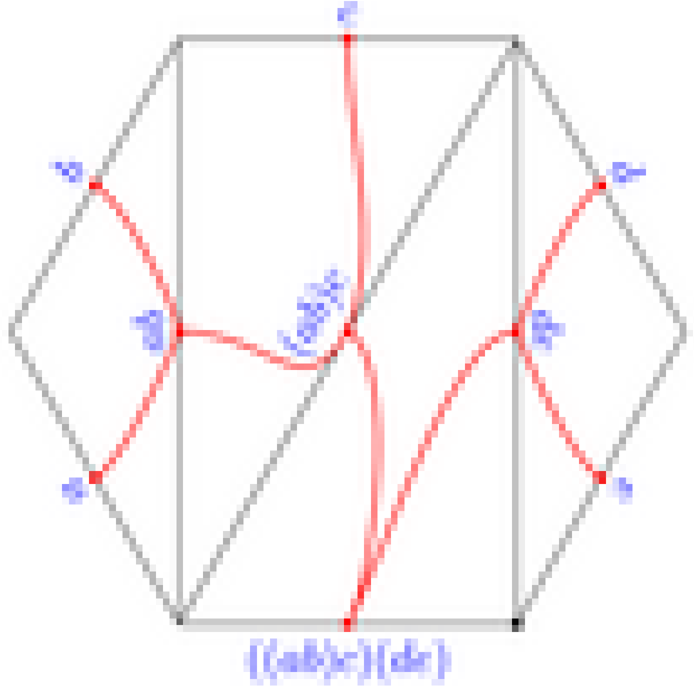
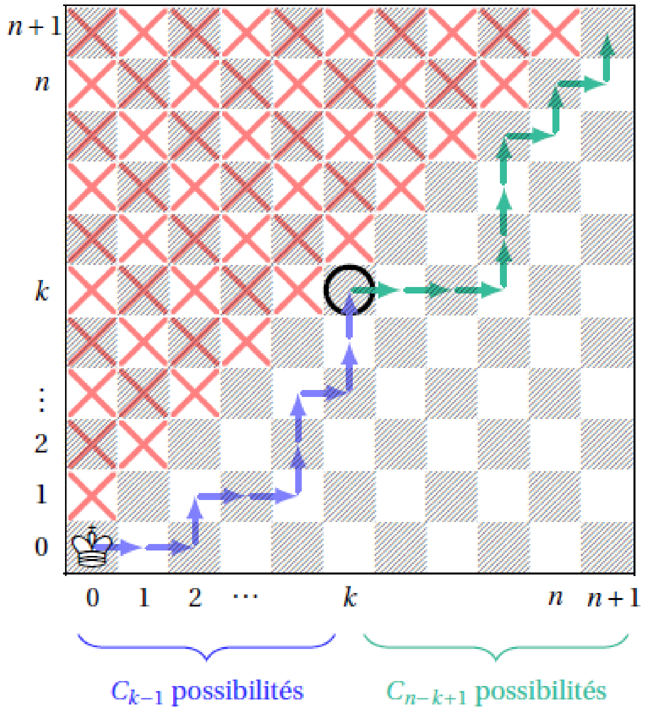
Les Nombres de Catalan





1											
1	1										
1	2	2									
1	3	5	5								
1	4	9	14	14							
1	5	14	28	42	42						
1	6	20	48	90	132	132					
1	7	27	75	165	297	429	429				
1	8	35	110	275	572	1001	1430	1430			
1	9	44	154	429	1001	2002	3432	4862	4862		
1	10	54	208	637	1638	3640	7072	11934	16796	16796	
1	11	65	273	910	2548	6188	13260	25194	41990	58786	58786

Exemple de calcul
48 = 20 + 28



Les Nombres de Catalan

C’est quoi ?

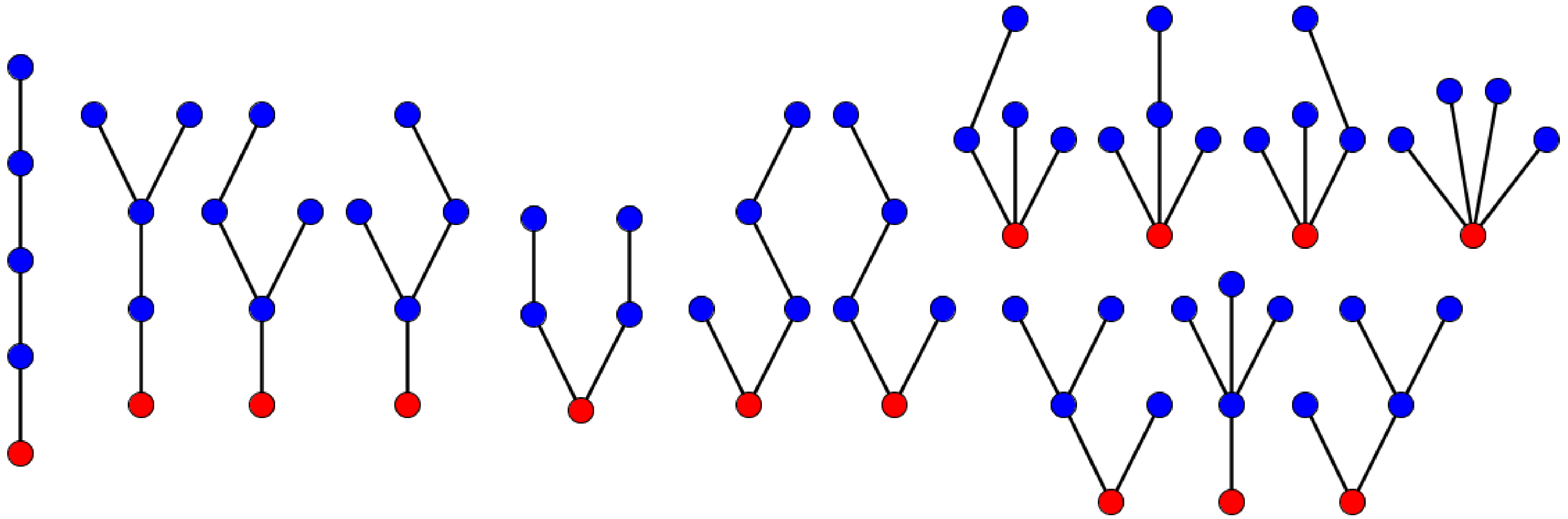
Qui l’a découvert ?

Comment on le calcule ?

C’est quoi ?

En mathématiques, les nombres de Catalan C(n) sont une suite de nombres entiers utilisés dans divers problèmes en combinatoire. Ils permettent notamment :

De dénombrer le nombre d’arbres binaires possédant exactement n nœud internes



De dénombrer le nombre de manières de parenthéser correctement un mot de longueur n

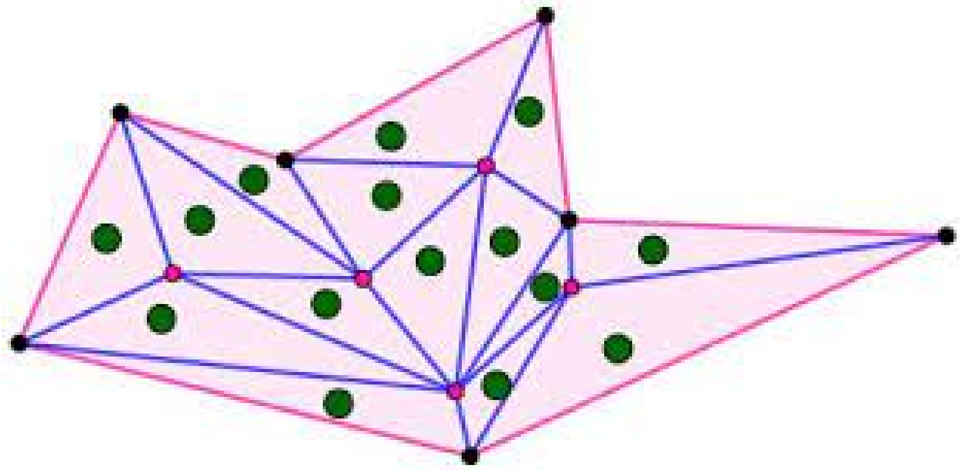
C’est qui ? C’est Quand ?

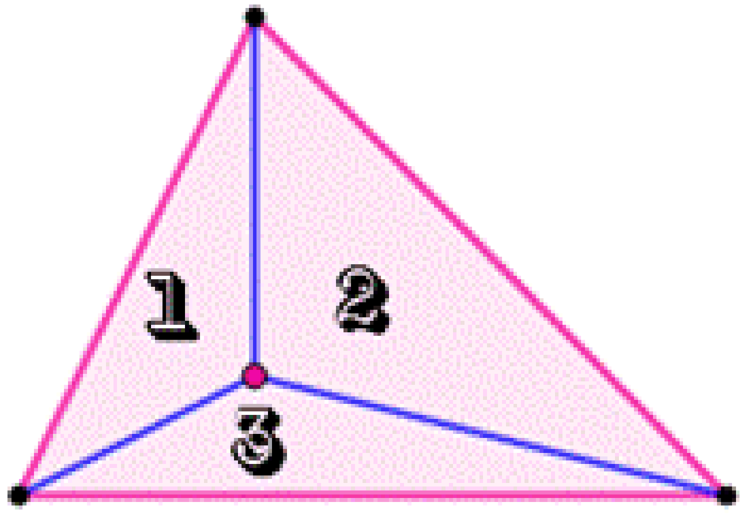
C’est qui ? C’est Quand ?

Leonhard Euler

Portrait par Johann Georg Brucker (1756).

En 1751 il pose le problème du dénombrement des triangulations d'un polygone.





Il trouve les 8 premiers nombres

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).

C’est qui ? C’est Quand ?

Eugène Charles Catalan

Eugène Charles Catalan, par Émile Delperée

Ce n'est qu'en 1838 qu'Eugène Charles Catalan pose et résout le problème du "nombre de manières d'effectuer le produit de n facteurs différents".

En 1839 Catalan fait le lien avec le nombre de triangulations, et publie un article complet sur le sujet en 1887.

Les Nombres de Catalan

C'est quoi ?

Qui l'a découvert ?

Comment on le calcule ?

Énoncé de mon projet

Implémenter plusieurs algorithmes permettant de calculer les nombres C(n), et les comparer en temps d’exécution. Écrire un programme qui affiche toutes les manières de bien parenthéser un mot de longueur n, discuter de son efficacité.

Implémentation Récursive

sources > C catalan.c > iterative(int)

```c
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;
}
```

Tests fonction récursive

```
Execution time test:
( 19 ) : Supérieur à 0.500000s -> arret du test.
Catalan -> recursive | overall_execution_time: 0.641319s.
```

1											
1	1										
1	2	2									
1	3	5	5								
1	4	9	14	14							
1	5	14	28	42	42						
1	6	20	48	90	132	132					
1	7	27	75	165	297	429	429				
1	8	35	110	275	572	1001	1430	1430			
1	9	44	154	429	1001	2002	3432	4862	4862		
1	10	54	208	637	1638	3640	7072	11934	16796	16796	
1	11	65	273	910	2548	6188	13260	25194	41990	58786	58786

Exemple de calcul
48 = 20 + 28

Implémentation Itérative

sources > catalan.c > iterative(int)

```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;
}
```

1											
1	1										
1	2	2									
1	3	5	5								
1	4	9	14	14							
1	5	14	28	42	42						
1	6	20	48	90	132	132					
1	7	27	75	165	297	429	429				
1	8	35	110	275	572	1001	1430	1430			
1	9	44	154	429	1001	2002	3432	4862	4862		
1	10	54	208	637	1638	3640	7072	11934	16796	16796	
1	11	65	273	910	2548	6188	13260	25194	41990	58786	58786

Exemple de calcul
48 = 20 + 28

Tests fonctions de 0 à 100000

```
Execution time test:
( 19 ) : 1.291289 supérieur à 0.500000s -> arret du test.
Catalan -> recursive | overall_execution_time: 0.648494s.


Execution time test:
Catalan -> iterative | overall_execution_time: 0.455392s.
```

1											
1	1										
1	2	2									
1	3	5	5								
1	4	9	14	14							
1	5	14	28	42	42						
1	6	20	48	90	132	132					
1	7	27	75	165	297	429	429				
1	8	35	110	275	572	1001	1430	1430			
1	9	44	154	429	1001	2002	3432	4862	4862		
1	10	54	208	637	1638	3640	7072	11934	16796	16796	
1	11	65	273	910	2548	6188	13260	25194	41990	58786	58786

Exemple de calcul
48 = 20 + 28

Implémentation Coefficient Binomial

sources > C catalan.c > binomial_coefficient(int)

```c
unsigned long long int binomial_coefficient(int N) {
    return binomial_coefficient_calcul(N * 2, N) / (N + 1);
}
unsigned long long int binomial_coefficient_calcul(int N, int k) {
    unsigned long long int result = 1;
    int temp, index;
    temp = N - k;
    if (k > temp) k = temp;
    for (index = 0; index < k; index += 1)
    {
        result *= (N - index);
        result /= (index + 1);
    }
    return result;
}
```

Zone de Test

Pour des tests de 0 à 100 000

```
Execution time test:
( 19 ) : Supérieur à 0.500000s -> arret du test.
Catalan -> recursive | overall_execution_time: 0.641319s.


Execution time test:
Catalan -> iterative | overall_execution_time: 453.358643s.


Execution time test:
Catalan -> binomial_coefficient | overall_execution_time: 0.488278s.
```

Pour des tests de 0 à 1 000 000

```
Execution time test:
( 578018 ) : Supérieur à 0.500000s -> arret du test.
Catalan -> binomial_coefficient | overall_execution_time: 1514.580444s.
```

Comment on le calcule ?

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).

1											
1	1										
1	2	2									
1	3	5	5								
1	4	9	14	14							
1	5	14	28	42	42						
1	6	20	48	90	132	132					
1	7	27	75	165	297	429	429				
1	8	35	110	275	572	1001	1430	1430			
1	9	44	154	429	1001	2002	3432	4862	4862		
1	10	54	208	637	1638	3640	7072	11934	16796	16796	
1	11	65	273	910	2548	6188	13260	25194	41990	58786	58786

Exemple de calcul
48 = 20 + 28

Énoncé de mon projet

Implémenter plusieurs algorithmes permettant de calculer les nombres C(n), et les comparer en temps d'exécution. Écrire un programme qui affiche toutes les manières de bien parenthéser un mot de longueur n, discuter de son efficacité.

Génération de parenthèses

```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);
}
```

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

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

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

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

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

Avez vous des questions ?