BROUILLON - SOUSTRAIRE LES PUISSANCES N^e DE (N+1) NATURELS CONSÉCUTIFS

CHRISTOPHE BAL

Document, avec son source LaTeX, disponible sur la page
https://github.com/bc-writing/drafts.

Mentions « légales »

Table des matières

Date: 3 Octobre 2019 – 21 Octobre 2019.

1. Faire la différence avec des puissances

Considérons l'algorithme suivant dont nous allons donner des cas d'application juste après.

Algorithme 1 : Version naturelle

Entrée : $n \in \mathbb{N}^*$
Sortie : ?
Actions
Choisir $(n+1)$ naturels consécutifs : $k_1 < k_2 < \cdots < k_{n+1}$.
$L \leftarrow [k_1^n, k_2^n, \ldots, k_{n+1}^n]$
Tant Que $taille(L) \neq 1$ **:**
$newL \leftarrow [\,]$
Pour i **de** *1* **à** *taille(L) - 1* **:**
Ajouter le nouvel élément $(L[i+1] - L[i])$ après la fin de la liste $newL$.
$L \leftarrow newL$
Renvoyer $L[1]$

Pour $n = 2$ avec $k_1 = 3$, $k_2 = 4$ et $k_3 = 5$, nous avons les valeurs suivantes de la liste L.

(1) $L = [3^2, 4^2, 5^2] = [9, 16, 25]$
(2) $L = [16 - 9, 25 - 16] = [7, 9]$
(3) $L = [9 - 7] = [2]$

L'algorithme renvoie donc 2 ici mais que se passe-t-il si l'on choisit d'autres naturels consécutifs ? Avec $k_1 = 10$, $k_2 = 11$ et $k_3 = 12$, nous obtenons :

(1) $L = [10^2, 11^2, 12^2] = [100, 121, 144]$
(2) $L = [121 - 100, 144 - 121] = [21, 23]$
(3) $L = [23 - 21] = [2]$

L'algorithme renvoie de nouveau 2. Que se passerait-il pour d'autres triplets de naturels consécutifs ? Pour se faire une bonne idée, il va falloir utiliser un programme. Ceci étant dit nous allons toute de suite faire l'hypothèse audacieuse que le choix des naturels consécutifs n'est pas important. Regardons alors ce que renvoie l'algorithme pour $n = 3$.

(1) $L = [0^3, 1^3, 2^3, 3^3] = [0, 1, 8, 27]$
(2) $L = [1, 7, 19]$
(3) $L = [6, 12]$
(4) $L = [6]$

L'algorithme renvoie 6 pour $n = 3$, et pour $n = 4$ ce qui suit nous donne que 24 est renvoyé. Ceci nous fait alors penser à $n!$ et donc nous amène à conjecturer, un peu rapidement c'est vrai, que l'algorithme va toujours renvoyé $n!$.

(1) $L = [0^4, 1^4, 2^4, 3^4, 4^4] = [0, 1, 16, 81, 256]$
(2) $L = [1, 15, 65, 175]$
(3) $L = [14, 50, 110]$
(4) $L = [36, 60]$
(5) $L = [24]$

Il est temps de passer aux choses un plus sérieuses via une expérimentaion informatique bien plus poussée.

2. Expérimentations et conjecture

Le code suivant [1] permet de tester notre conjecture audacieuse : aucun test raté n'est révélé.

Code Python

```
from math import factorial
from random import randint

NMAX      = 20
POWER_MAX = 6
NB_TESTS  = 10**6

print("TEST - START")

kmax = 10**POWER_MAX

for _ in range(NB_TESTS):
    n     = randint(1, NMAX)
    start = randint(0, kmax)

    L = [i**n for i in range(start, start + n + 1)]

    while len(L) != 1:
        newL = []

        for i, elt in enumerate(L[:-1]):
            newL.append(L[i+1] - elt)

        L = newL[:]

    if L[0] != factorial(n):
        print(f"    * Test failed with n = {n}")
        exit()

print("TEST - END")
```

Comme les valeurs des naturels consécutifs ne semblent pas importantes, on peut pousser l'expérimentation en travaillant avec $k_1 \in \mathbb{R}$, $k_2 = k_1 + 1$, $k_3 = k_1 + 2$, ... Ceci étant dit, il n'est pas toujours possible de travailler avec des réels en informatique : l'usage des flottants s'accompagne de son lot d'arrondis. Nous allons donc juste tester des valeurs rationnelles via le code suivant [2] qui ne révèle aucun test raté.

Code Python

```
from math import factorial
from random import randint
```

1. Ce fichier `diff-n-cons-int-to-the-power-of-n/exploring-int-version.py` est disponible dans le sous-dossier sur le lieu de téléchargement de ce document.

2. Ce fichier `diff-n-cons-int-to-the-power-of-n/exploring-frac-version.py` est disponible dans le sous-dossier sur le lieu de téléchargement de ce document.

```
from sympy import S

NMAX      = 20
POWER_MAX = 6
NB_TESTS  = 10**3

print("TEST - START")

kmax = 10**POWER_MAX

for _ in range(NB_TESTS):
    n     = randint(1, NMAX)
    start = S(randint(0, kmax)) / S(randint(1, kmax))

    L = [(start + i)**n for i in range(n + 1)]

    while len(L) != 1:
        newL = []

        for i, elt in enumerate(L[:-1]):
            newL.append(L[i+1] - elt)

        L = newL[:]

    if L[0] != factorial(n):
        print(f"    * Test failed with n = {n}")
        exit()

print("TEST - END")
```

Il devient donc normal de penser que le phénomène est purement polynomial. Nous allons explorer ceci dans la section suivante.

3. Une preuve polynomiale

Voici une légère modification de l'algorithme 1 où l'on manipule des polynômes dans $\mathbb{R}[X]$. Notons au passage que nous passons d'un algorithme a priori indéterministe, car on fait un choix de naturels consécutifs, à un autre complètement déterministe.

Algorithme 2 : Version polynomiale

Entrée : $n \in \mathbb{N}^*$
Sortie : ?
Actions
- $L \leftarrow [X^n, (X+1)^n, \ldots, (X+n)^n]$
- **Tant Que** $taille(L) \neq 1$ **:**
  - $newL \leftarrow [\,]$
  - **Pour** i **de** *1* **à** *taille(L) - 1* **:**
    - Ajouter le nouvel élément $(L[i+1] - L[i])$ après la fin de la liste $newL$.
  - $L \leftarrow newL$
- **Renvoyer** $L[1]$

Il est aisée de traduire cet algorithme en Python. Le code suivant ne révèle aucun test raté.

Code Python

```
from math import factorial
from random import randint

from sympy import poly
from sympy.abc import x

NMAX     = 20
NB_TESTS = 10**4

print("TEST - START")

for _ in range(NB_TESTS):
    n = randint(1, NMAX)
    L = [poly((x + i)**n) for i in range(n + 1)]

    while len(L) != 1:
        newL = []

        for i, elt in enumerate(L[:-1]):
            newL.append(L[i+1] - elt)

        L = newL[:]

    result = L[0]
    result = result.expand()

    if result.degree() != 0 \
    or L[0] != factorial(n):
```

```
        print(f"    * Test failed with n = {n}")
        exit()

print("TEST - END")
```

Il est clair que si l'on prouve que l'algorithme 2 renvoie $n!$, il en sera de même pour 1. Ceci découle directement de la validité de l'algorithme ci-dessous où $\mathbb{R}_n\big[X\big]$ désigne l'ensemble des polynômes réels de degré $n \in \mathbb{N}^*$.

Algorithme 3 : Version polynomiale élargie

Entrée : $P \in \mathbb{R}_n\big[X\big]$ de coefficient dominant a_n
Sortie : $n!\,a_n$
Actions
- $L \leftarrow [\,P(X), P(X+1), \ldots, P(X+n)\,]$
- **Tant Que** *taille*$(L) \neq 1$ **:**
  - $newL \leftarrow [\,]$
  - **Pour** i **de** *1* **à** *taille(L) - 1* **:**
    - Ajouter le nouvel élément $(L[i+1] - L[i])$ après la fin de la liste $newL$.
  - $L \leftarrow newL$
- **Renvoyer** $L[1]$

Pourquoi cet algorithme est-il valide ? Pour la suite, nous posons $P(x) = \sum_{k=0}^{n} a_k X^k$.

(1) Il est immédiat que $\forall k \in \mathbb{N}^*$, $(X+1)^k - X^k = kX^{k-1} + R(X)$ où $\deg R < k-1$ avec la convention $\deg 0 = -\infty$.

(2) Le point précédent donne sans effort $P(X+1) - P(X) = na_nX^{n-1} + S(X)$ où le polynôme S vérifie $\deg S < n-1$. Tout est dit comme nous allons le voir.

(3) A la 1re itération de la boucle, la liste $[\,P(X), P(X+1), \ldots, P(X+n)\,]$ est transformée en $[\,P(X+1) - P(X), P(X+2) - P(X+1), \ldots, P(X+n) - P(X+n-1)\,]$ soit $[\,Q(X), Q(X+1), \ldots, Q(X+n-1)\,]$ en posant $Q(X) = P(X+1) - P(X)$, un polynôme qui a pour coefficient dominant na_n et pour degré $\deg Q = n-1$.

(4) Il est alors facile de faire une récurrence pour prouver que la boucle se finit en fournissant ce qui est annoncé.

Joli, efficace et éclairant sur le pourquoi du comment. Il se trouve que l'algorithme 1 cache aussi une formule combinatoire très intéressante comme nous allons le voir dans les sections à venir. L'exposé qui suit reprend la démarche proposée sur le site The Math Less Traveled [3]

3. Chercher les articles *« A combinatorial proof »* publiés courant septembre 2019.

4. Une très jolie identité binomiale

De l'algorithme 1, nous allons juste garder les opérations sur les listes. Ceci nous donne l'algorithme suivant dont nous allons chercher à obtenir une formule explicite de la sortie S_n en fonction de a_0, a_1, ... , a_n.

Algorithme 4

Entrée : $[a_0, a_1, \ldots, a_n]$ une liste de naturels
Sortie : ?
Actions
  $L \leftarrow [a_0, a_1, \ldots, a_n]$
  Tant Que *taille*$(L) \neq 1$ **:**
    $newL \leftarrow [\,]$
    Pour i **de** *1* **à** *taille(L) - 1* **:**
      Ajouter le nouvel élément $(L[i+1] - L[i])$ après la fin de la liste $newL$.
    $L \leftarrow newL$
  Renvoyer $L[1]$

Pour $n = 0$ et $n = 1$, nous avons directement que $S_0 = a_0$ et $S_1 = -a_0 + a_1$ respectivement.

Pour $n = 2$, nous avons les deux étapes suivantes.

(1) $[a_1 - a_0, a_2 - a_1]$
(2) $[a_2 - a_1 - a_1 + a_0]$ d'où $S_2 = a_0 - 2a_1 + a_2$

Pour $n = 3$, nous avons les étapes suivantes où le triangle de Pascal pointe son nez.

(1) $[a_1 - a_0, a_2 - a_1, a_3 - a_2]$
(2) Le cas précédent donne alors $S_3 = (a_1 - a_0) - 2(a_2 - a_1) + a_3 - a_2$ soit après simplification $S_3 = -a_0 + 3a_1 - 3a_2 + a_3$.

Plus généralement, nous allons supposer que $S_n = \sum_{i=0}^{n} c_{i,n} a_k$ où les coefficients $c_{i,n}$ sont des entiers relatifs indépendants des valeurs des a_k. En ajoutant un élément a_{n+1} après la fin de la liste $[a_0, a_1, \ldots, a_n]$, nous devons avoir :

$$\begin{aligned}
\sum_{i=0}^{n+1} c_{i,n+1} a_k &= S_{n+1} \\
&= \sum_{i=0}^{n} c_{i,n}(a_{i+1} - a_k) \\
&= \sum_{i=1}^{n+1} c_{i-1,n} a_k - \sum_{i=0}^{n} c_{i,n} a_k \\
&= -c_{0,n} a_0 + \sum_{i=1}^{n} (c_{i-1,n} - c_{i,n}) a_k + c_{n,n} a_n
\end{aligned}$$

La suite $(c_{i,n})_{n \in \mathbb{N}, i \in [\![0;n]\!]}$ doit donc vérifier les conditions suivantes.

(1) $c_{0,0} = 1$
(2) $c_{0,1} = -1$ et $c_{1,1} = 1$

(3) $c_{0,2} = 1$, $c_{1,2} = -2$ et $c_{2,2} = 1$

(4) $\forall n \in \mathbb{N}$, $c_{0,n+1} = -c_{0,n}$ et $c_{n+1,n+1} = c_{n,n}$.

(5) $\forall n \in \mathbb{N}$, $\forall i \in [\![1\,;n]\!]$, $c_{i,n+1} = c_{i-1,n} - c_{i,n}$.

Il est aisé de voir que $c_{i,n} = (-1)^{i+n}\binom{n}{i}$ convient. Nous avons ainsi justifié que l'algorithme 4 renvoie $\sum_{i=0}^{n}(-1)^{i+n}\binom{n}{i}a_k$. Ce résultat appliqué à la liste $[k^n, (k+1)^n, \ldots, (k+n)^n]$ et la preuve de la section précédente nous donnent la très jolie formule suivante.

$$\forall k \in \mathbb{N}, \sum_{i=0}^{n}(-1)^{i+n}\binom{n}{i}(k+i)^n = n!$$

La section finale suivante propose une preuve purement combinatoire de cette identité.

Remarque 4.1. *Il est maintenant facile de prouver les identités suivantes.*

$$\forall d \in \mathbb{N}^*, \forall k \in \mathbb{N}, \sum_{i=0}^{n}(-1)^{i+n}\binom{n}{i}(k+id)^n = d^n n!$$

5. Une preuve combinatoire

Notre dernière mission va consister à prouver la formule ci-après via de simples arguments de combinatoire.

$$\forall k \in \mathbb{N}, \sum_{i=0}^{n} (-1)^{i+n} \binom{n}{i} (k+i)^n = n!$$

Commençons par noter que $n!$ est le nombre de permutations sur $[[n]] \overset{\text{déf}}{=} [\![1\,;n]\!]$ et $(k+i)^n$ celui des applications de $[[n]]$ sur $[[k+i]]$. Nous noterons $\mathscr{A}_{n\to k+i}$ l'ensemble des applications de $[[n]]$ sur $[[k+i]]$.

Comme $[[n]] \subseteq [[k+n]]$, nous allons nous concentrer sur $\mathscr{A}_{n\to k+n}$. Dans ce contexte, pour compter les permutations de $[[n]]$, nous allons très classiquement étudier les éléments de $\mathscr{A}_{n\to k+n}$ qui ne sont pas des permutations de $[[n]]$. Nous noterons $\mathscr{P}_n$ l'ensemble des permutations de $[[n]]$ sur $[[n]] \subset [[k+n]]$.

Soit donc $a \in \mathscr{A}_{n\to k+n}$. Pour que a ne soit pas une permutation sur $[[n]]$ il faut et il suffit que $\mathcal{I}_a$ l'image de a ne soit pas $[[n]]$. Pour $i \in [\![1\,;n]\!]$, notons alors $\mathscr{M}_i$ l'ensemble des applications a telles que $\mathcal{I}_a$ ne contienne pas l'élément i de sorte que $\mathscr{P}_n = \mathscr{M}_0 - \overset{n}{\underset{i=1}{\cup}} \mathscr{M}_i$. Le principe d'inclusion-exclusion, le PIE pour les intimes, nous permet alors de faire les calculs suivants.

$$\begin{aligned} n! &= \operatorname{card} \mathscr{P}_n \\ &= \operatorname{card} \mathscr{A}_{n\to k+n} + \sum_{i=1}^{n} (-1)^i \sum_{j_1<\cdots<j_i} \operatorname{card} \left(\mathscr{M}_{j_1} \cap \cdots \cap \mathscr{M}_{j_i}\right) \end{aligned}$$

Or $\operatorname{card} \left(\mathscr{M}_{j_1} \cap \cdots \cap \mathscr{M}_{j_i}\right) = \binom{n}{i}(k+n-i)^n$ car nous avons $\binom{n}{i}$ choix possible pour j_1 , … , j_i et il reste alors $(k+n-i)$ valeurs possibles pour les images des éléments de $[[n]]$. Il est clair que $\operatorname{card} \mathscr{A}_{n\to k+n} = (-1)^0\binom{n}{0}(k+n)^n$ de sorte que nous obtenons :

$$n! = \sum_{i=0}^{n} (-1)^i \binom{n}{i} (k+n-i)^n$$

Pour retomber sur la formule à démontrer, il suffit de faire le changement d'indices $j = n - i$ pour tomber sur l'identité suivante, notre objectif, en notant que $n - j \equiv j + n \bmod 2$ et en utilisant $\binom{n}{i} = \binom{n}{n-i}$.

$$n! = \sum_{j=0}^{n} (-1)^{j+n} \binom{n}{j} (k+j)^n \text{ avec } k \in \mathbb{N} \text{ quelconque}$$