Von_Neumann

September 15, 2023

1 Entiers de Von Neumann

Marc Lorenzi

14 septembre 2023

[1]:
```
import matplotlib.pyplot as plt
```

1.1 1. Introduction

On pose $\overline{0} = \emptyset$ et, pour tout $n \in \mathbb{N}^*$, $\overline{n+1} = \overline{n} \cup \{\overline{n}\}$. Les ensembles $\overline{n}$ sont les *entiers de Von Neumann*.

Nous allons représenter les ensembles en Python par la liste de leurs éléments. La fonction `VN` prend en paramètre un entier naturel n. Elle renvoie $\overline{n}$. Son code est une simple boucle `for`.

[2]:
```
def VN(n):
    m = []
    for k in range(n):
        m = m + [m]
    return m
```

Voici les premiers entiers de Von Neumann.

[3]:
```
for n in range(6):
    print(n, VN(n))
```

```
0 []
1 [[]]
2 [[], [[]]]
3 [[], [[]], [[], [[]]]]
4 [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]]]
5 [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]]]]
```

Et voici $\overline{10}$.

[4]:
```
print(VN(10))
```

```
[[], [[]], [[], [[]]], [[], [[]], [[], [[]]]], [[], [[]], [[], [[]]], [[], [[]],
[[], [[]]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]], [[], [[]], [[],
[[]]], [[], [[]], [[], [[]]]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]],
[[], [[]], [[], [[]]], [[], [[]], [[], [[]]]]], [[], [[]], [[], [[]]], [[],
[[]], [[], [[]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]]]]], [[], [[]],
[[], [[]]], [[], [[]], [[], [[]]]], [[], [[]], [[], [[]]], [[], [[]], [[],
[[]]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]], [[], [[]], [[], [[]]],
[[], [[]], [[], [[]]]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]], [[],
[[]], [[], [[]]], [[], [[]], [[], [[]]]]], [[], [[]], [[], [[]]], [[], [[]],
[[], [[]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]]]]]], [[], [[]], [[],
[[]]], [[], [[]], [[], [[]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]]],
[[], [[]], [[], [[]]], [[], [[]], [[], [[]]]], [[], [[]], [[], [[]]], [[], [[]],
[[], [[]]]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]], [[], [[]], [[],
[[]]], [[], [[]], [[], [[]]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]],
[[], [[]], [[], [[]]], [[], [[]], [[], [[]]]]]]]], [[], [[]], [[], [[]]], [[],
[[]], [[], [[]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]]], [[], [[]],
[[], [[]]], [[], [[]], [[], [[]]]], [[], [[]], [[], [[]]], [[], [[]], [[],
[[]]]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]], [[], [[]], [[], [[]]],
[[], [[]], [[], [[]]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]], [[],
[[]], [[], [[]]], [[], [[]], [[], [[]]]]]]]]], [[], [[]], [[], [[]]], [[], [[]],
[[], [[]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]]], [[], [[]], [[],
[[]]], [[], [[]], [[], [[]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]]]],
[[], [[]], [[], [[]]], [[], [[]], [[], [[]]]], [[], [[]], [[], [[]]], [[], [[]],
[[], [[]]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]], [[], [[]], [[],
[[]]], [[], [[]], [[], [[]]]]]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]],
[[], [[]], [[], [[]]], [[], [[]], [[], [[]]]]], [[], [[]], [[], [[]]], [[],
[[]], [[], [[]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]]]], [[], [[]],
[[], [[]]], [[], [[]], [[], [[]]]], [[], [[]], [[], [[]]], [[], [[]], [[],
[[]]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]], [[], [[]], [[], [[]]],
[[], [[]], [[], [[]]]]]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]], [[],
[[]], [[], [[]]], [[], [[]], [[], [[]]]]], [[], [[]], [[], [[]]], [[], [[]],
[[], [[]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]]]], [[], [[]], [[],
[[]]], [[], [[]], [[], [[]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]]],
[[], [[]], [[], [[]]], [[], [[]], [[], [[]]]], [[], [[]], [[], [[]]], [[], [[]],
[[], [[]]]]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]], [[], [[]], [[],
[[]]], [[], [[]], [[], [[]]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]],
[[], [[]], [[], [[]]], [[], [[]], [[], [[]]]]]], [[], [[]], [[], [[]]], [[],
[[]], [[], [[]]]], [[], [[]], [[], [[]]], [[], [[]], [[], [[]]]]], [[], [[]],
[[], [[]]], [[], [[]], [[], [[]]]], [[], [[]], [[], [[]]], [[], [[]], [[],
[[]]]]]]]]]]]
```

1.2 2. Une représentation plus lisible

Le symbole ∅ est le caractère unicode 2205.

[5]:
```
empty = '\u2205'
```

[6]:
```
print(empty)
```

La fonction `chaine` prend en paramètre un entier de Von Neumann N. Elle renvoie la représentation de N sous forme de chaîne de caractères, en remplaçant toutes les listes vides [] par $\emptyset$. Son principe est simple. Si $N = []$, la fonction renvoie '$\emptyset$'. Sinon,

$$N = [N_0, \dots, N_{n-1}]$$

où les N_i sont eux-mêmes des entiers de Von Neumann. La fonction se rappelle alors récursivement et renvoie la chaîne de caractères

$$'\{\text{chaine}(N_0), \dots, \text{chaine}(N_{n-1})\}'$$

[7]:
```
def chaine(N):
    if N == []: return empty
    else:
        s = '{'
        n = len(N)
        for k in range(n):
            s += chaine(N[k])
            if k < n - 1:
                s += ','
        s += '}'
        return s
```

[8]:
```
print(chaine(VN(4)))
```

```
{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}}}
```

[9]:
```
print(chaine(VN(10)))
```

```
{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}}},{ ,{ },{ ,{ }},{
,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}}}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{
,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{
},{ ,{ }}}}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{
,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,
{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{
},{ ,{ }},{ ,{ },{ ,{ }}}}}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,
{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{
},{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}}}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,
{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{
},{ ,{ }}}}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{
},{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}}}}}},{ ,{ },{ ,{ }},{
,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,
```

```
{ },{ ,{ }},{ ,{ },{ ,{ }}}}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{
 },{ ,{ }}}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}}}},{
,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}}},{ ,{ },{ ,{ }},{ ,{
 },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}}}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{
 },{ ,{ }},{ ,{ },{ ,{ }}}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ }
,{ ,{ }}}}}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}}},{ ,
{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}}},{ ,{ },{ ,{ }},{ ,{
 },{ ,{ }}},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}}},{ ,{ },{ ,{ }},{ ,{
},{ ,{ }},{ ,{ },{ ,{ }}}}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{
},{ ,{ }}}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}}},{ ,{
 },{ ,{ }},{ ,{ },{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}}},{ ,{ },{ ,{ }},{ ,{ }
,{ ,{ }}},{ ,{ },{ ,{ }},{ ,{ },{ ,{ }}}}}}}}}}
```

1.3 3. Un peu de dénombrement

Pour tout $n \in \mathbb{N}$, notons s_n la chaîne de caractères qui représente $\overline{n}$. Notons également

- u_n = le nombre d'occurences de '{' dans s_n.
- v_n = le nombre d'occurences de '}' dans s_n.
- w_n = le nombre d'occurences de ',' dans s_n.
- x_n = le nombre d'occurences de '$\emptyset$' dans s_n.

[10]:
```
def nombre_occurences(c, s):
    n = 0
    for x in s:
        if x == c:
            n += 1
    return n
```

[11]:
```
def u(n): return nombre_occurences('{', chaine(VN(n)))
def v(n): return nombre_occurences('}', chaine(VN(n)))
def w(n): return nombre_occurences(',', chaine(VN(n)))
def x(n): return nombre_occurences(empty, chaine(VN(n)))
```

[12]:
```
n = 10
print(u(n), v(n), w(n), x(n))
print(u(n) + v(n) + w(n) + x(n), len(chaine(VN(n))))
```

```
512 512 511 512
2047 2047
```

Proposition. Pour tout $n \in \mathbb{N}$,

$$\overline{n} = \{\overline{0}, \overline{1}, \dots, \overline{n-1}\}$$

Démonstration. Montrons ce résultat par récurrence sur n. C'est trivialement vrai si $n = 0$. Soit $n \in \mathbb{N}$. Supposons le résultat vrai pour n. On a alors

$$\begin{array}{rcl}\overline{n+1} & = & \overline{n} \cup \{\overline{n}\} \\ & =_{(HR)} & \{\overline{0}, \overline{1}, ..., \overline{n-1}\} \cup \{\overline{n}\} \\ & = & \{\overline{0}, \overline{1}, ..., \overline{n}\}\end{array}$$

Proposition. Pour tout $n \geq 1$,

$$u_n = v_n = x_n = 2^{n-1}$$

$$w_n = 2^{n-1} - 1$$

Démonstration. Prouvons l'égalité pour w_n par récurrence forte sur n. On laisse en exercice les trois autres suites.

- Tout d'abord, $s_1 = \{\emptyset\}$ ne contient pas de virgule, donc

$$w_1 = 0 = 2^{1-1} - 1$$

- Soit $n \in \mathbb{N}^*$. Supposons que pour tout $k \in [|1, n-1|]$, $w_k = 2^{k-1} - 1$. On a

$$s_n =' \{' + s_0 + ',' + s_1 + ',' + ... + ',' + s_{n-1} + '\}'$$

Dans l'expression ci-dessus, il y a $n-1$ virgules explicites, auxquelles il faut rajouter les virgules contenues dans s_0, s_1, ..., s_{n-1}. Celles-ci sont au nombre de w_0, w_1, ..., w_{n-1}. Comme s_0 ne contient pas de virgule,

$$\begin{array}{rcl} w_n & = & n - 1 + \displaystyle\sum_{k=1}^{n-1} w_k \\ & = & n - 1 + \displaystyle\sum_{k=1}^{n-1} (2^{k-1} - 1) \\ & = & \displaystyle\sum_{k=1}^{n-1} 2^{k-1} \\ & = & \displaystyle\sum_{k=0}^{n-2} 2^k \\ & = & 2^{n-1} - 1 \end{array}$$

Corollaire. Pour tout $n \in \mathbb{N}^*$, la longueur de S_n est $2^{n+1} - 1$.

Démonstration. Soit $n \in \mathbb{N}^*$. La longueur de s_n est

$$u_n + v_n + w_n + x_n = 4 \times 2^{n-1} - 1 = 2^{n+1} - 1$$

Par exemple, la longueur de s_{1000} est

[14]:
```
2 ** 1001 - 1
```

[14]:
```
214301721437253464189685009812000362112280962341106721488750077674070210224987224498639675763139171625518934583510629365037429057138462808719691551493971496078691355496484619708421492101247422837559083643060929499671638825347975351183310878921541258291423929553730843353208596633052487736744113361387 51
```

1.4 4. Représentation par des arbres

Pour être parfaitement rigoureux dans les quelques lignes qui suivent, il faudrait parler d'*ensembles héréditairement finis* … ce que nous ne ferons pas.

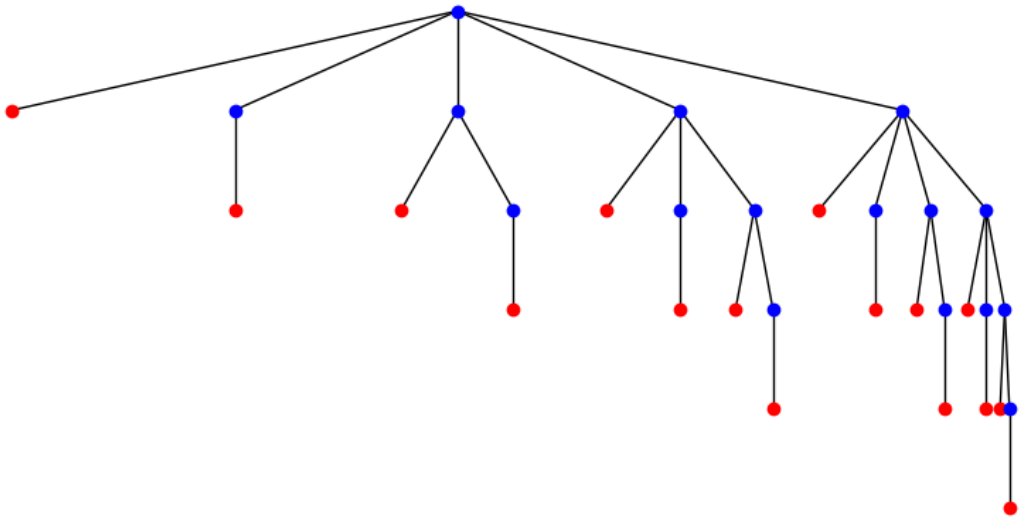
Soit E un ensemble dont les élements sont des ensembles dont les éléments etc, etc. L'*arbre associé à* E possède une racine. Si n est le cardinal de E, cette racine possède n fils, qui sont les arbres associés aux éléments de E.

La fonction ci-dessous permet de dessiner l'arbre en question. Je ne détaillerai pas son code.

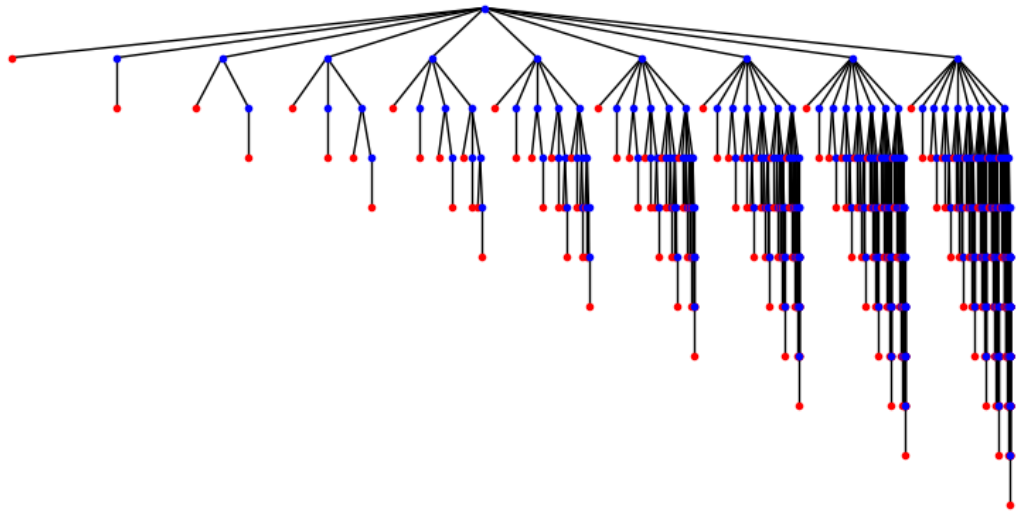
[15]:
```
def dessiner_arbre(E, xmin, xmax, y, taille_noeuds=6):
    m = (xmin + xmax) / 2
    n = len(E)
    if n > 0:
        d = (xmax - xmin) / n
        for k in range(n):
            plt.plot([m, xmin + (k + 1/2) * d], [y, y - 1], 'k', lw=1)
            dessiner_arbre(E[k], xmin + k * d, xmin + (k + 1) * d, y - 1,␣
 ↪taille_noeuds)
    if n == 0: plt.plot([m], [y], 'or', ms=taille_noeuds)
    else: plt.plot([m], [y], 'ob', ms=taille_noeuds)
    plt.axis('off')
```

[16]:
```
plt.rcParams['figure.figsize'] = (10, 5)
```

[17]:
```
dessiner_arbre(VN(5), 0, 1, 0)
```



[18]:
```
dessiner_arbre(VN(10), 0, 1, 0, taille_noeuds=3)
```



La fonction `dessiner_arbre` permet de dessiner l'arbre associé à des ensembles autres que des entiers de Von Neumann. Par exemple, étant donné un ensemble E, nous pouvons nous intéresser à l'ensemble

$$\mathcal{P}^n(E) = \mathcal{P}(\mathcal{P}(\ldots\mathcal{P}(E)\ldots))$$

où l'expression ci-dessus comporte n lettres $\mathcal{P}$. Cet ensemble peut être défini par récurrence sur n

en posant

- $\mathcal{P}^0(E) = E$
- Pour tout $n \in \mathbb{N}$, $\mathcal{P}^{n+1}(E) = \mathcal{P}(\mathcal{P}^n(E))$

La fonction `parties` prend en paramètre un ensemble E donné par la liste de ses éléments. Elle renvoie l'ensemble des parties de E. L'idée de la fonction est suivante.

- Si $E = \emptyset$, $\mathcal{P}(E) = \{\emptyset\}$.
- Sinon, soit $a \in E$. Soit $E_1 = E \setminus \{a\}$.

Si $\mathcal{P}' = \mathcal{P}(E_1)$, on a alors

$$\mathcal{P}(E) = \mathcal{P}' \cup \{\{a\} \cup X : X \in \mathcal{P}'\}$$

[21]:
```
def parties(E):
    if E == []: return [[]]
    else:
        a = E[0]
        E1 = E[1:]
        P1 = parties(E1)
        P2 = [[a] + X for X in P1]
        return P1 + P2
```

[22]:
```
print(parties([1, 2, 3]))
```

```
[[], [3], [2], [2, 3], [1], [1, 3], [1, 2], [1, 2, 3]]
```

La fonction `Pn` prend en paramètres un ensemble E et un entier naturel n. Elle renvoie $\mathcal{P}^n(E)$.

[23]:
```
def Pn(E, n):
    P = E
    for k in range(n):
        P = parties(P)
    return P
```

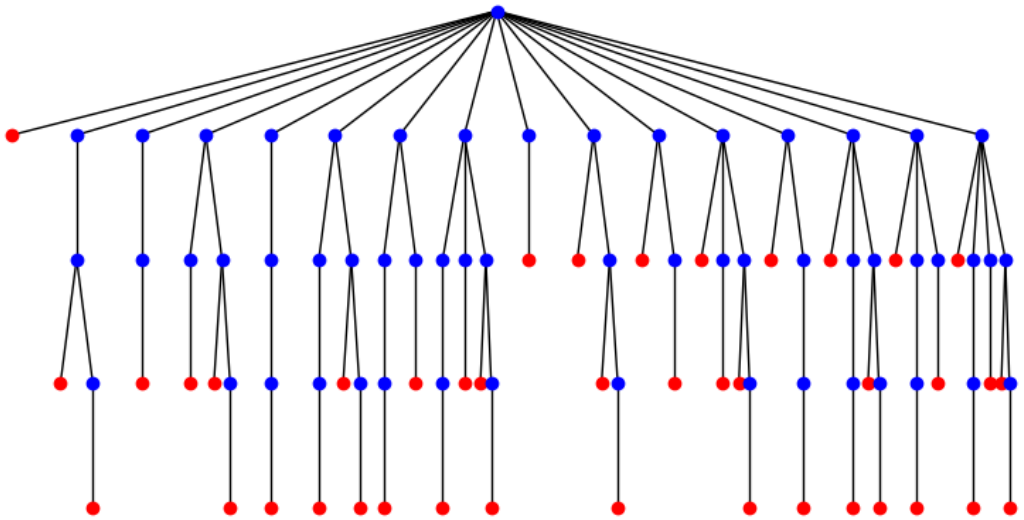
Voici la chaîne de caractères qui représente P_4.

[24]:
```
print(chaine(Pn([], 4)))
```

```
{ ,{{ ,{ }}},{{ }},{{ },{ ,{ }}},{{{ }}},{{{ }},{ ,{ }}},{{{ }},{ }},{{{ }},{ },
{ ,{ }}},{ },{ ,{ ,{ }}},{ ,{ }},{ ,{ },{ ,{ }}},{ ,{{ }}},{ ,{{ }},{ ,{ }}},{ ,
{{ }},{ }},{ ,{{ }},{ },{ ,{ }}}}
```

Et voici l'arbre correspondant.

[25]:
```
dessiner_arbre(Pn([], 4), 0, 1, 0)
```



Essaierons-nous d'afficher P_5 ? Juste pour voir, combien y a-t-il d'occurences du caractère $\emptyset$ dans P_5 ?

[27]:
```
nombre_occurences(empty, chaine(Pn([], 5)))
```

[27]: 1343489

L'arbre associé à P_5 possède donc plus d'un million de points rouges …