01_HF

November 7, 2021

1 Ensembles héréditairement finis (I)

Marc Lorenzi

3 novembre 2021

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

Notations. Nous utiliserons dans ce notebook les notations pas tout à fait standard suivantes.

- Pour tout ensemble fini A, $|A|$ dénote le cardinal de A.
- Étant donnés deux ensembles A et B, on note $A \subseteq B$ si A est inclus dans B. On note $A \subset B$ si A est inclus dans B *et* différent de B.
- Étant donnés un ensemble A et une propriété P, $\{x \in A : P(x)\}$ est l'ensemble des éléments de A qui vérifient P.
- Étant donnés un ensemble A et une fonction f, $\{f(x) : x \in A\}$ est l'ensemble des images par f des éléments de A.

Avertissement. Nous allons manipuler dans ce notebook des ensembles d'ensembles d'ensembles, etc. Pour des raisons de clarté et de simplicité, nous représenterons ces objets en Python par des listes de listes de listes, etc. Les fonctions qui en résulteront auront parfois une complexité (en nombre d'opérations à effectuer) qui pourrait être grandement améliorée, au prix d'un obscurcissement du code. Dans la suite, je me contenterai par-ci par-là de quelques remarques sur la complexité des fonctions écrites, sans pour autant entrer dans les détails.

1.1 1. Introduction

1.1.1 1.1 Ensembles héréditairement finis

On pose $V_0 = \emptyset$ et pour tout $n \in \mathbb{N}$, $V_{n+1} = \mathcal{P}(V_n)$, l'ensemble des parties de V_n. On définit enfin

$$V = \bigcup_{n \in \mathbb{N}} V_n$$

Les ensembles appartenant à V sont les ensembles *héréditairement finis*.

Par exemple, $\emptyset$, $\{\emptyset\}$, $\{\emptyset, \{\emptyset\}\}$, $\{\{\{\emptyset\}\}\}$, sont des ensembles héréditairement finis.

L'objet de ce notebook et des suivants est l'étude des propriétés de V et de ses éléments.

Remarque. Pour tout $n \in \mathbb{N}$, $V_n \subseteq V_n$, donc $V_n \in \mathcal{P}(V_n) = V_{n+1} \subseteq V$. Ainsi, $V_n \in V$ est un ensemble héréditairement fini.

Mise en garde. *Dans ce qui va suivre, on demande au le lecteur de faire encore plus attention que d'habitude à la différence entre $\in$ et $\subseteq$.*

1.1.2 1.2 Le cardinal de V_n

Pour tout $x \in \mathbb{R}$, définissons par récurrence sur n le réel $x^{\uparrow n}$ en posant $x^{\uparrow 0} = 0$ et pour tout $n \in \mathbb{N}$, $x^{\uparrow n+1} = x^{\left(x^{\uparrow n}\right)}$. On a ainsi $x^{\uparrow 1} = 1$, $x^{\uparrow 2} = x$, $x^{\uparrow 3} = x^x$, etc. Plus généralement, pour tout $n \geq 2$,

$$x^{\uparrow n} = x^{x^{x^{\cdots^x}}}$$

où le nombre x apparaît $n-1$ fois dans l'expression.

Proposition. Pour tout $n \in \mathbb{N}$, V_n est un ensemble fini et son cardinal est $|V_n| = 2^{\uparrow n}$.

Démonstration. C'est une récurrence facile. Il suffit d'écrire que $|V_{n+1}| = |\mathcal{P}(V_n)| = 2^{|V_n|}$.

[2]:
```
def ppow(a, n):
    p = 0
    for k in range(n):
        p = a ** p
    return p
```

Voici le cardinal des premiers ensembles V_n.

[3]:
```
for k in range(6):
    print(k, ppow(2, k))
```

```
0 0
1 1
2 2
3 4
4 16
5 65536
```

Remarquons que $|V_6| = 2^{65536} \simeq 10^{19728}$ et que personne ne verra donc jamais V_6.

Remarque. Pour tout $n \in \mathbb{N}$, $2^n \geq n+1$. De là,

$$|V_{n+1}| = 2^{|V_n|} \geq |V_n| + 1$$

d'où

$$|V_{n+1}| > |V_n|$$

La suite $(|V_n|)_{n\in\mathbb{N}}$ est ainsi strictement croissante. La croissance de $|V_n|$ est extrêmement rapide : une récurrence montre que pour tout $n \in \mathbb{N}$, $|V_n| \geq n$. De là, pour tout $n \geq 1$,

$$|V_n| = 2^{|V_{n-1}|} \geq 2^{n-1}$$

et donc, pour tout $n \geq 2$,

$$|V_n| = 2^{|V_{n-1}|} \geq 2^{2^{n-2}}$$

etc.

En corollaire, V est un ensemble infini puisqu'il contient des sous-ensembles de cardinaux non majorés. Nous verrons plus loin que l'ensemble V est *dénombrable*, en écrivant une bijection explicite de V sur $\mathbb{N}$.

Proposition. Pour tout $n \in \mathbb{N}$, $V_n \subset V_{n+1}$.

Démonstration. Montrons cette propriété par récurrence sur n.

- C'est clair pour $n = 0$, puisque $V_0 = \emptyset \subset V_1 = \{\emptyset\}$.
- Soit $n \in \mathbb{N}$. Supposons $V_n \subset V_{n+1}$. Soit $A \in V_{n+1}$. On a donc $A \in \mathcal{P}(V_n)$, c'est à dire $A \subseteq V_n$. Ainsi, pour tout $x \in A$, $x \in V_n$ et, par l'hypothèse de récurrence, $x \in V_{n+1}$. De là, $A \subseteq V_{n+1}$ et donc $A \in \mathcal{P}(V_{n+1}) = V_{n+2}$.

Enfin, $V_{n+1} \neq V_{n+2}$ car ces deux ensembles ont des cardinaux distincts. □

Corollaire. Soit $n \in \mathbb{N}$. Soit $A \in V_n$. Alors, $A \subseteq V_n$.

Démonstration. Comme $V_n \subset V_{n+1}$, on a $A \in V_{n+1} = \mathcal{P}(V_n)$ et donc $A \subseteq V_n$. □

Nous reviendrons dans le prochain notebook sur cette propriété de V_n qui s'appelle la *transitivité*.

Corollaire. Soit $A \in V$. Pour tout $x \in A$, $x \in V$.

Dit autrement, les éléments d'un ensemble héréditairement fini sont des ensembles héréditairement finis. Et donc, en réutilisant cette propriété, les éléments des éléments d'un ensemble héréditairement fini sont des ensembles héréditairement finis, etc.

1.1.3 1.3 Rang d'un ensemble héréditairement fini

Définition. Pour tout $A \in V$, le *rang* de A est le plus petit entier n tel que $A \in V_{n+1}$.

Par exemple, rg $\emptyset = 0$, rg $\{\emptyset\} = 1$, rg $\{\emptyset, \{\emptyset\}\} =$ rg $\{\{\emptyset\}\} = 2$.

Comme $V_0 \subseteq V_1 \subseteq V_2 \subseteq \ldots$, si $n =$ rg A alors

- Pour tout $k \leq n$, $A \notin V_k$.
- Pour tout $k \geq n + 1$, $A \in V_k$.

Le rang de A est donc l'unique entier naturel n tel que $A \in V_{n+1}$ et $A \notin V_n$.

Proposition. Pour tout ensemble $A \in V$ non vide, rg $A = \max\{\text{rg } x : x \in A\} + 1$.

Démonstration. Soit $A \in V$. Soit $n = \text{rg } A$. On a $A \in V_{n+1} = \mathcal{P}(V_n)$ donc $A \subseteq V_n$. Ainsi, pour tout $x \in A$, $x \in V_n$ et donc $\text{rg } x \leq n-1$. De plus, $A \notin V_n$, donc $A \not\subseteq V_{n-1}$. Il existe donc $x \in A$ tel que $x \notin V_{n-1}$, c'est à dire tel que $\text{rg } x \geq n-1$. □

Corollaire. Soient $A, B \in V$ tels que $A \in B$. On a $\text{rg } A < \text{rg } B$.

Proposition. Soit A un ensemble. On a $A \in V$ si et seulement si A est fini et $A \subseteq V$.

Démonstration. Supposons $A \in V$. Soit $n = \text{rg A}$. Alors $A \in V_{n+1} = \mathcal{P}(V_n)$ donc $A \subseteq V_n \subseteq V$. De plus, comme V_n est fini, A l'est aussi.

Inversement, supposons A fini et inclus dans V. Soit $n = \max\{\text{rg } x : x \in A\}$. Pour tout $x \in A$, $x \in V_{n+1}$ donc $A \subseteq V_{n+1}$, donc $A \in \mathcal{P}(V_{n+1}) = V_{n+2} \subseteq V$. Ainsi, $A \in V$. □

Remarque. Soit $n \geq 1$. Soient $A_1, \ldots, A_n \in V$. Supposons $A_1 \in A_2 \ldots \in A_n$. On a alors

$$\text{rg } A_1 < \ldots < \text{rg } A_n$$

et donc $A_n \notin A_1$. La relation d'appartenance ne comporte pas de « cycles ». En particulier, en prenant $n = 1$ et $n = 2$, on obtient que

- Pour tout $A \in V$, $A \notin A$.
- Pour tous $A, B \in V$, $A \in B \implies B \notin A$.

La relation $\in$ est donc *irréflexive* et *asymétrique* sur V. Il ne lui manque que la *transitivité* pour être une relation d'ordre strict. Nous reviendrons sur ce sujet dans le notebook suivant.

Proposition. Pour tout $n \in \mathbb{N}$, $\text{rg } V_n = n+1$.

Démonstration. On a $V_n \in V_{n+1}$ et $V_n \notin V_n$. □

1.1.4 1.4 Une bijection de V sur $\mathbb{N}$

La bijection que nous allons décrire ici est due à Wilhelm Ackermann.

Soit $\varphi : V \longrightarrow \mathbb{N}$ définie pour tout $A \in V$ par

$$\varphi(A) = \sum_{x \in A} 2^{\varphi(x)}$$

Une récurrence forte sur rg A montre que cette fonction est bien définie.

Proposition. Pour tout $n \in \mathbb{N}$, φ est une bijection de V_n sur $[\![0, 2^{\uparrow n} - 1]\!]$.

Démonstration. Faisons une récurrence sur n.

- C'est clair pour $n = 0$ puisque $V_0 = \emptyset$.
- Soit $n \in \mathbb{N}$. Supposons la propriété vérifiée pour n.

Montrons tout d'abord que φ envoie V_{n+1} dans $[\![0, 2^{\uparrow n+1} - 1]\!]$. Soit $A \in V_{n+1}$. On a $\text{rg } A \leq n$. Les éléments de A sont donc de rang inférieur ou égal à $n-1$, et appartiennent ainsi à V_n. De là,

$$\varphi(A) = \sum_{x \in A} 2^{\varphi(x)} \leq \sum_{x \in V_n} 2^{\varphi(x)}$$

Par l'hypothèse de récurrence, φ est une bijection de V_n sur $[\![0, 2^{\uparrow n} - 1]\!]$. On a donc

$$\sum_{x \in V_n} 2^{\varphi(x)} = \sum_{k=0}^{2^{\uparrow n}-1} 2^k = 2^{2^{\uparrow n}} - 1 = 2^{\uparrow n+1} - 1$$

Montrons maintenant l'injectivité de φ. Soient $A, B \in V_{n+1}$. Supposons $\varphi(A) = \varphi(B)$. On a donc

$$\sum_{x \in A} 2^{\varphi(x)} = \sum_{x \in B} 2^{\varphi(x)}$$

Par les propriétés de l'écriture des entiers en base 2, les exposants des deux membres sont les mêmes :

$$\{\varphi(x) : x \in A\} = \{\varphi(x) : x \in B\}$$

Les éléments de A et B appartiennent à V_n et φ est, par l'hypothèse de récurrence, injective sur V_n. On a donc

$$\{x : x \in A\} = \{x : x \in B\}$$

Bref, $A = B$.

Pour conclure, φ envoie injectivement V_{n+1} dans $[\![0, 2^{\uparrow n+1} - 1]\!]$. Or, ces deux ensembles ont le même cardinal. Il y a donc aussi surjectivité. □

Corollaire. φ est une bijection de V sur $\mathbb{N}$.

Démonstration. Montrons l'injectivité. Soient $A, B \in V$. Supposons $\varphi(A) = \varphi(B)$. Soit $n = \max(\text{rg } A, \text{rg } B)$. On a $A, B \in V_{n+1}$. Or, φ est injective sur V_{n+1}, donc $A = B$.

Montrons la surjectivité. Soit $n \in \mathbb{N}$. Il existe un entier $r \in \mathbb{N}$ tel que $2^{\uparrow r} > n$ (par exemple, $r = n$). Comme φ est une surjection de V_r sur $[\![0, 2^{\uparrow r} - 1]\!]$, il existe $A \in V_r$ tel que $n = \varphi(A)$. □

Proposition. Pour tout $n \in \mathbb{N}$, $\varphi(V_n) = 2^{n+1} - 1$.

Démonstration. On a

$$V_n = \{x \in V : \varphi(x) \leq 2^{\uparrow n} - 1\}$$

De là,

$$\varphi(V_n) = \sum_{x \in V_n} 2^{\varphi(x)} = \sum_{k=0}^{2^{\uparrow n}-1} 2^k = 2^{2^{\uparrow n>}} - 1 = 2^{\uparrow n+1} - 1$$

1.1.5 Une convention Python

Convention. Nous représenterons un élément de V en Python par la liste de ses éléments, **ordonnés dans l'ordre croissant de leurs images par φ.**

Remarquons que par cette convention il est facile d'obtenir le rang d'un ensemble. En effet, pour tout $A \in V$ non vide, on a

$$\text{rg } A = \max\{\text{rg } x : x \in A\} + 1$$

Avec notre convention, $\text{rg } \{x_1, \ldots, x_n\} = 1 + \text{rg } x_n$. Il est donc immédiat d'écrire une fonction `rang` qui renvoie le rang d'un ensemble $A \in V$.

[4]:
```
def rang(A):
    k = 0
    while A != []:
        A = A[-1]
        k = k + 1
    return k
```

La fonction `phi` ci-dessous renvoie $\varphi(A)$ pour tout $A \in V$.

[5]:
```
def phi(A):
    n = 0
    for x in A:
        n = n + 2 ** phi(x)
    return n
```

Avant de faire des tests, décrivons la réciproque de φ.

1.1.6 1.5 La réciproque de φ

Nous noterons ψ la réciproque de φ.

Proposition. Pour tout $n \in \mathbb{N}$, $\psi(n) = \{\psi(m) : m \prec n\}$ où $m \prec n$ si et seulement si le $m$ième chiffre de n en base 2 est un 1.

Démonstration. Soit $n \in \mathbb{N}$. Soit $A = \psi(n)$. On a

$$n = \varphi(A) = \sum_{x \in A} 2^{\varphi(x)}$$

Par ailleurs,

$$n = \sum_{m \prec n} 2^m$$

Par unicité de l'écriture de n en base 2, on a donc pour tout $x \in V$, $x \in A \iff \varphi(x) \prec n$. Ainsi, pour tout $m \in \mathbb{N}$, $\psi(m) \in A \iff m \prec n$, d'où

$$A = \{\psi(m) : m \prec n\} \square$$

[6]:
```
def psi(n):
    s = []
    m = 0
    while n != 0:
        if n % 2 == 1: s.append(psi(m))
        n = n // 2
        m = m + 1
    return s
```

Nous pouvons maintenant effectuer quelques tests sur les fonctions φ et ψ. Listons d'abord les premiers ensembles héréditairement finis.

[7]:
```
for n in range(10):
    print(n, psi(n))
```

```
0 []
1 [[]]
2 [[[]]]
3 [[], [[]]]
4 [[[[]]]]
5 [[], [[[]]]]
6 [[[]], [[[]]]]
7 [[], [[]], [[[]]]]
8 [[[], [[]]]]
9 [[], [[], [[]]]]
```

Vérifions que φ et ψ sont bien réciproques l'une de l'autre.

[8]:
```
print(phi(psi(1234567891234567 89)))
```

```
1234567891234567 89
```

Comme $\varphi(V_n) = 2^{\uparrow n+1} - 1$, il est maintenant facile d'obtenir V_n.

[9]:
```
def V(n):
    return psi(ppow(2, n + 1) - 1)
```

Voici V_4, qui a 16 éléments.

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

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

V_5 a déjà $2^{16} = 65536$ éléments. Ce ne serait pas une bonne idée de demander à Python de l'afficher.

1.1.7 1.6 Une relation d'ordre sur V

Définissons une relation $\leq$ sur V en posant, pour tous $A, B \in V$,

$$A \leq B \iff \varphi(A) \leq \varphi(B)$$

On note évidemment $A < B$ si $A \leq B$ et $A \neq B$. Comme φ est bijective et que $(\mathbb{N}, \leq)$ est bien ordonné, la relation $\leq$ est elle-même un bon ordre sur V.

Remarquons que notre convention pour représenter les ensembles héréditairement finis en Python devient : **on représente les éléments de V en Python par la liste de leurs éléments dans l'ordre strictement croissant.**

Proposition. Soient $A, B \in V$ non vides. On a

$$A \leq B \iff (\max A < \max B) \vee (\max A = \max B \wedge A \setminus \{\max A\} \leq B \setminus \{\max B\})$$

Démonstration. Notons $x = \max A$ et $y = \max B$. On a

$$\varphi(A) = 2^{\varphi(x)} + \sum_{t \in A, t \neq x} 2^{\varphi(t)}$$

et

$$\varphi(B) = 2^{\varphi(y)} + \sum_{t \in B, t \neq y} 2^{\varphi(t)}$$

- Si $\varphi(x) < \varphi(y)$, les propriétés de la représentation des entiers en base 2 montrent que $\varphi(A) < \varphi(B)$.
- Si $\varphi(x) > \varphi(y)$, on a de même $\varphi(B) < \varphi(A)$.
- Si $\varphi(x) = \varphi(y)$, on a $\varphi(A) \leq \varphi(B)$ si et seulement si

$$\sum_{t \in A, t \neq x} 2^{\varphi(t)} \leq \sum_{t \in B, t \neq y} 2^{\varphi(t)}$$

c'est à dire si et seulement si $A \setminus \{x\} \leq B \setminus \{y\}$. □

La fonction `inferieur_strict` prend en paramètres deux ensembles $A, B \in V$. Elle renvoie `True` si $A < B$ et `False` sinon.

[11]:
```
def inferieur_strict(A, B):
    if A == []: return B != []
    elif B == []: return False
    else:
        return inferieur_strict(A[-1], B[-1]) or (A[-1] == B[-1] and␣
↪inferieur_strict(A[:-1], B[:-1]))
```

[12]:
```
inferieur_strict(psi(12345), psi(12344))
```

[12]:
```
False
```

[13]:
```
inferieur_strict(psi(12344), psi(12345))
```

[13]:
```
True
```

1.2 2 Deux familles d'éléments de V

1.2.1 2.1 Les entiers de von Neumann

On définit par récurrence sur n le $n$ième *entier de von Neumann* en posant

- $\overline{0} = \emptyset$
- Pour tout $n \in \mathbb{N}$, $\overline{n+1} = \overline{n} \cup \{\overline{n}\}$.

On a donc pour tout $n \in \mathbb{N}$, $\overline{n} = \{\overline{0}, \ldots, \overline{n-1}\}$.

[14]:
```
def neumann(n):
    u = []
    for k in range(n):
        u = u + [u]
    return u
```

[15]:
```
for n in range(5):
    print(n, neumann(n))
```

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

Proposition. Pour tout $n \in \mathbb{N}$, $|\overline{n}| = \operatorname{rg} \overline{n} = n$.

Démonstration. Récurrence facile. □

[16]:
```
print(rang(neumann(20)))
```

```
20
```

Définissons par récurrence forte sur n une suite $(\nu_n)_{n\in\mathbb{N}}$ en posant pour tout $n \in \mathbb{N}$,

$$\nu_n = \sum_{k=0}^{n-1} 2^{\nu_k}$$

Remarquons que l'on a $\nu_0 = 0$ et pour tout $n \geq 1$,

$$\nu_{n+1} = 2^{\nu_n} + \nu_n$$

Proposition. Pour tout $n \in \mathbb{N}$, $\varphi(\overline{n}) = \nu_n$.

Démonstration. On fait une récurrence forte sur n. La propriété est vraie pour $n = 0$. Soit $n \geq 1$. Supposons que pour tout $k < n$, $\varphi(\overline{k}) = \nu_k$. On a alors

$$\varphi(\overline{n}) = \sum_{x \in \overline{n}} 2^{\varphi(x)} = \sum_{k=0}^{n-1} 2^{\varphi(\overline{k})} = \sum_{k=0}^{n-1} 2^{\nu_k} = \nu_n$$

[17]:
```
def nu(n):
    s = 0
    for k in range(n):
        s = 2 ** s + s
    return s
```

[18]:
```
for k in range(6):
    print(k, nu(k))
```

```
0 0
1 1
2 3
3 11
4 2059
5 661852284340449429518640674583960616149895222675773112978029474355704937244014
405492678684907989267736344943839680471439239568571402054064027405360874460838310
520368482324399959044049927980075147183260434105703798308704637800852606194444172
051991971237512107049703527278337554258761027760282673134058094295488805547820407
652775628283628842383254654485203483075749433459903099416426669267233797295981858
347350547325004154098838683614231599137708122187727119017722495531534022877597895
171217443367553504659016552051849173709742024055869412110653955407655676631932971
733672542303136122441829419995004023881954500530803855470
```

Remarquons la valeur de ν_5. Il serait absurde d'essayer de calculer ν_6.

1.2.2 2.2 Les ensembles σ_n et τ_n

On définit par récurrence sur n les ensembles σ_n et τ_n en posant

- $\sigma_0 = \tau_0 = \emptyset$
- Pour tout $n \in \mathbb{N}$, $\sigma_{n+1} = \{\sigma_n\}$.
- Pour tout $n \in \mathbb{N}$, $\tau_{n+1} = \tau_n \cup \{\sigma_n\}$.

Remarquons que pour tout $n \geq 1$, $\tau_n = \{\sigma_0, \ldots, \sigma_{n-1}\}$.

[19]:
```
def sigma(n):
    s = []
    for k in range(n): s = [s]
    return s
```

[20]:
```
def tau(n):
    return [sigma(k) for k in range(n)]
```

[21]:
```
for n in range(5):
    print(n, tau(n))
```

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

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

- Si $n \neq 0$, $|\sigma_n| = 1$.
- $\operatorname{rg} \sigma_n = n$.
- $|\tau_n| = \operatorname{rg} \tau_n = n$.

Démonstration. Récurrence sur n.

[22]:
```
for k in range(10):
    print(len(sigma(k)), rang(sigma(k)), len(tau(k)), rang(tau(k)))
```

```
0 0 0 0
1 1 1 1
1 2 2 2
1 3 3 3
1 4 4 4
1 5 5 5
1 6 6 6
1 7 7 7
1 8 8 8
1 9 9 9
```

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

$$\varphi(\sigma_n) = 2^{\uparrow n}$$

et

$$\varphi(\tau_n) = \sum_{k=1}^{n} 2^{\uparrow k}$$

Démonstration. Récurrence facile pour $\varphi(\sigma_n)$. De là, comme $\tau_n = \{\sigma_0, \dots, \sigma_{n-1}\}$,

$$\varphi(\tau_n) = \sum_{k=0}^{n-1} 2^{\varphi(\sigma_k)} = \sum_{k=0}^{n-1} 2^{2^{\uparrow k}} = \sum_{k=0}^{n-1} 2^{\uparrow k+1} = \sum_{k=1}^{n} 2^{\uparrow k}$$

[23]:
```
def phi_tau(n):
    s = 0
    for k in range(1, n + 1):
        s = s + ppow(2, k)
    return s
```

[24]:
```
for n in range(6):
    print(n, phi_tau(n))
```

```
0 0
1 1
2 3
3 7
4 23
5 65559
```

1.2.3 2.3 Afficher les éléments de V de façon plus compacte

Avouons-le, lire les éléments de V sous leur forme standard (crochets, virgules) n'est pas facile. Maintenant que nous avons mis en évidence des éléments de V particuliers ($\overline{n}$, σ_n, τ_n), nous pouvons les utiliser pour représenter de façon plus compacte les ensembles héréditairement finis. La fonction `tostr` fait le travail. Elle prend en paramètre un ensemble $A \in V$ et renvoie une représentation de A sous forme de chaîne de caractères.

- S'il existe $n \in \mathbb{N}$ tel que $A = \overline{n}$, la fonction renvoie la chaîne qui représente l'entier n.
- S'il existe $n \in \mathbb{N}$ tel que $A = \tau_n$, la fonction renvoie τn.
- S'il existe $n \in \mathbb{N}$ tel que $A = \sigma_n$, la fonction renvoie σn.
- Sinon, la fonction se rappelle récursivement sur les éléments de A.

Remarquons que $\overline{0} = \sigma_0 = \tau_0 = \emptyset$ et $\overline{1} = \sigma_1 = \tau_1 = \{\emptyset\}$ et $\overline{2} = \tau_2$. Il n'y aura donc jamais dans la chaîne renvoyée $\sigma 0$, $\sigma 1$, $\tau 0$, $\tau 1$ et $\tau 2$.

[25]:
```
def tostr(A):
    n = rang(A)
    if A == neumann(n): return str(n) # ' ' +
    elif A == sigma(n): return ' ' + str(n)
    elif A == tau(n): return ' ' + str(n)
    else:
        s = '{'
        l = len(A)
        for k in range(l):
            s = s + tostr(A[k])
            if k < l - 1: s = s + ','
        s = s + '}'
        return s
```

Les éléments de V s'écrivent maintenant de façon beaucoup plus compacte.

[26]:
```
for k in range(20):
    print('%3d %-35s %-15s' % (k, psi(k), tostr(psi(k))))
```

```
  0 []                                  0
  1 [[]]                                1
  2 [[[]]]                               2
  3 [[], [[]]]                          2
  4 [[[[]]]]                             3
  5 [[], [[[]]]]                        {0, 2}
  6 [[[]], [[[]]]]                      {1, 2}
  7 [[], [[]], [[[]]]]                   3
  8 [[[], [[]]]]                        {2}
  9 [[], [[], [[]]]]                    {0,2}
 10 [[[]], [[], [[]]]]                  {1,2}
 11 [[], [[]], [[], [[]]]]              3
 12 [[[[]]], [[], [[]]]]                { 2,2}
 13 [[], [[[]]], [[], [[]]]]            {0, 2,2}
 14 [[[]], [[[]]], [[], [[]]]]          {1, 2,2}
 15 [[], [[]], [[[]]], [[], [[]]]]      {0,1, 2,2}
 16 [[[[[]]]]]                          4
 17 [[], [[[[]]]]]                      {0, 3}
 18 [[[]], [[[[]]]]]                    {1, 3}
 19 [[], [[]], [[[[]]]]]                {0,1, 3}
```

Voici, avec nos nouvelles notations, l'ensemble V_4.

[27]:
```
A = V(4)
print(tostr(A))
```

```
{0,1, 2,2, 3,{0, 2},{1, 2}, 3,{2},{0,2},{1,2},3,{ 2,2},{0, 2,2},{1, 2,2},{0,1, 2
,2}}
```

1.3 3. Arbre d'un ensemble héréditairement fini

1.3.1 3.1 Introduction

Nous n'entrerons pas ici dans les détails de la notion d'*arbre*. Disons qu'un arbre possède une *racine*, qui peut être *étiquetée*, et un ensemble fini de *fils*, qui sont eux-mêmes des arbres. La *hauteur* d'un arbre T est définie récursivement comme suit.

- Si T n'a pas de fils, $h(T) = 0$.
- Sinon, $h(T) = 1 + \max\{h(T') : T' \text{ fils de } T\}$.

Soit $A \in V$. On peut associer à A un arbre $T(A)$ comme suit :

- La racine de $T(A)$ est étiquetée par A.
- Les fils de $T(A)$ sont les arbres $T(x)$, où $x \in A$.

On vérifie facilement que pour tout $A \in V$, on a $h(T(A)) = \text{rg } A$.

Proposition. Soient $A, B \in V$. On a $A = B \iff T(A) = T(B)$.

Démonstration. Le sens direct est évident. Pour la réciproque, on procède par récurrence forte sur $\max(\text{rg } A, \text{rg } B)$.

1.3.2 3.2 Tracer l'arbre

Nous ne commenterons pas les détails de la fonction `tracer0` ci-dessous. Elle prend en paramètres

- Un ensemble $A \in V$.
- Des bornes `xmin`, `xmax`, `y` et `d`.

Elle trace l'arbre de A dans le rectangle $[x_{min}, x_{max}] \times [y - d, y]$.

[28]:
```
def tracer0(A, bornes):
    xmin, xmax, y, d = bornes
    xm = (xmin + xmax) / 2
    if A == []:
        plt.plot([xm], [y], 'or')
    else:
        n = len(A)
        dx = (xmax - xmin) / n
        xm = (xmin + xmax) / 2
        for k in range(n):
            x1 = xmin + k * dx
            x2 = xmin + (k + 1) * dx
            xm2 = (x1 + x2) / 2
            plt.plot([xm, xm2], [y, y - d], color=[0.5, 0.5, 0.5])
            tracer0(A[k], (x1, x2, y - d, d))
        plt.plot([xm], [y], 'og')
```

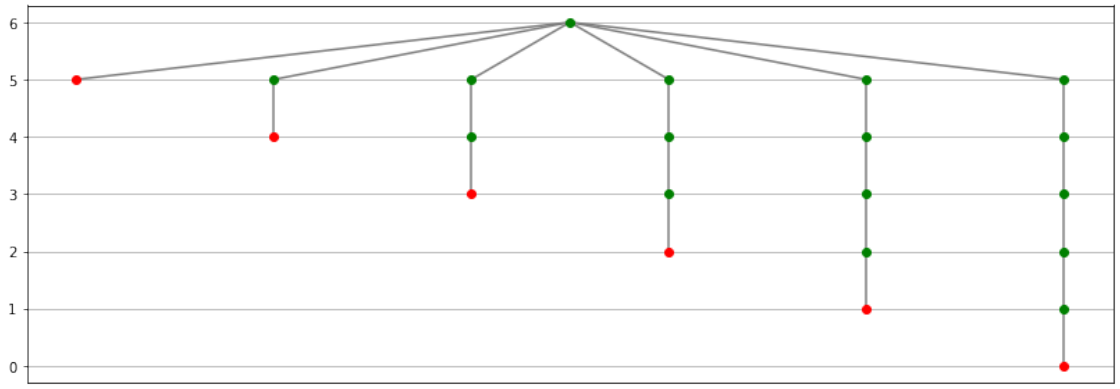
La fonction `tracer_arbre` prend en paramètre un ensemble $A \in V$. Elle trace l'arbre $\mathcal{T}(A)$. Nous n'affichons pas les étiquettes des noeuds.

[29]:
```
def tracer_arbre(A):
    plt.rcParams['figure.figsize'] = (14, 5)
    r = rang(A)
    tracer0(A, (0, 1, r, 1))
    plt.xticks([])
    plt.yticks(range(r + 1))
    plt.grid()
```

1.3.3 3.3 Quelques exemples

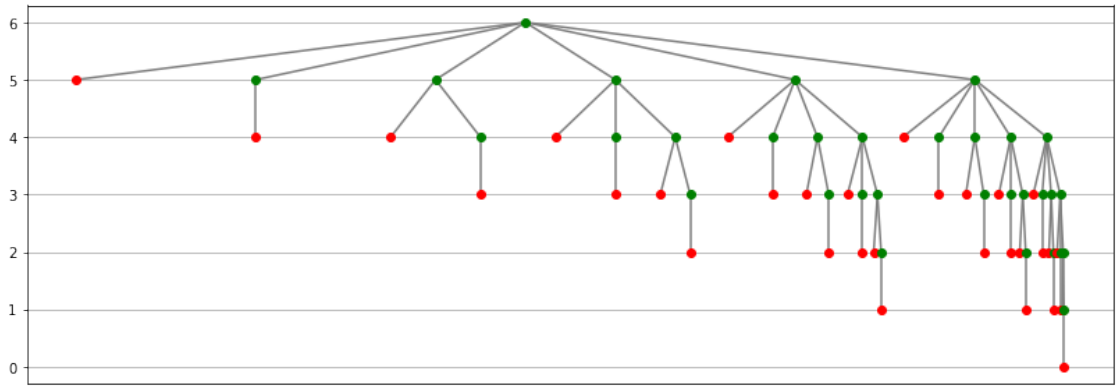
Voici l'arbre de τ_6.

[30]:
```
A = tau(6)
tracer_arbre(A)
```



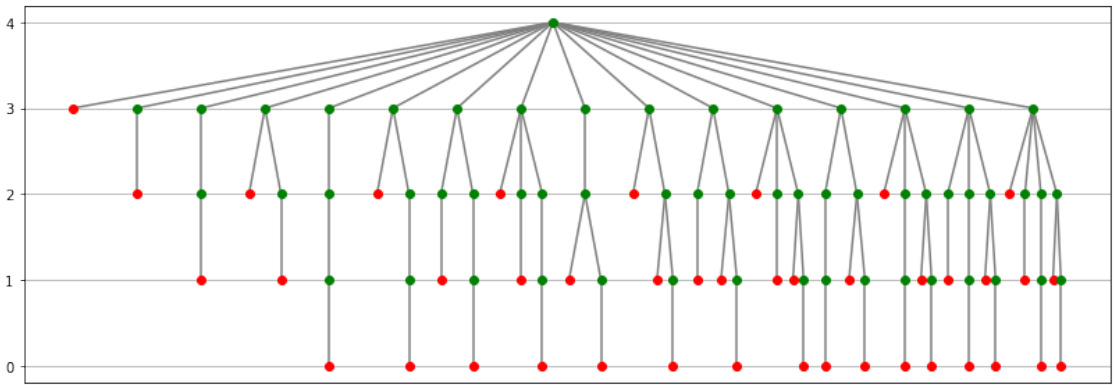
Et voici celui de $\overline{6}$.

[31]:
```
A = neumann(6)
tracer_arbre(A)
```



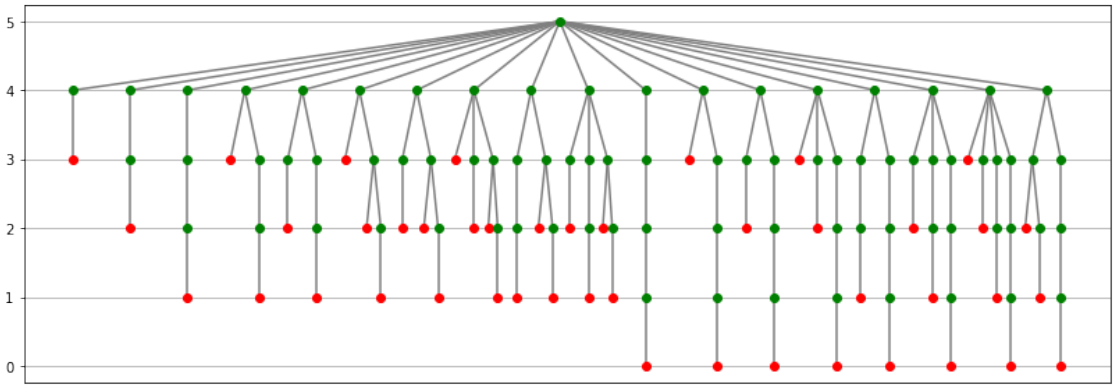
Et enfin l'arbre de V_4.

[32]:
```
A = V(4)
tracer_arbre(A)
```



Ci-dessous, un espace de libre expression .

[33]:
```
A = psi(31415926)
tracer_arbre(A)
```



1.3.4 3.4 Une propriété d'unicité

Étant donné un arbre T, les *noeuds* de T sont sa racine et les noeuds de ses fils. Un *réétiquetage* de T est un « renommage » des noeuds de T. Si T et T' sont deux arbres et que par un réétiquetage de T on obtient T', nous noterons $T \simeq T'$. Pour ne pas alourdir ce notebook, nous ne serons pas plus précis que cela.

Proposition. Soit T un arbre. Il existe un unique ensemble $A \in V$ tel que $T \simeq T(A)$.

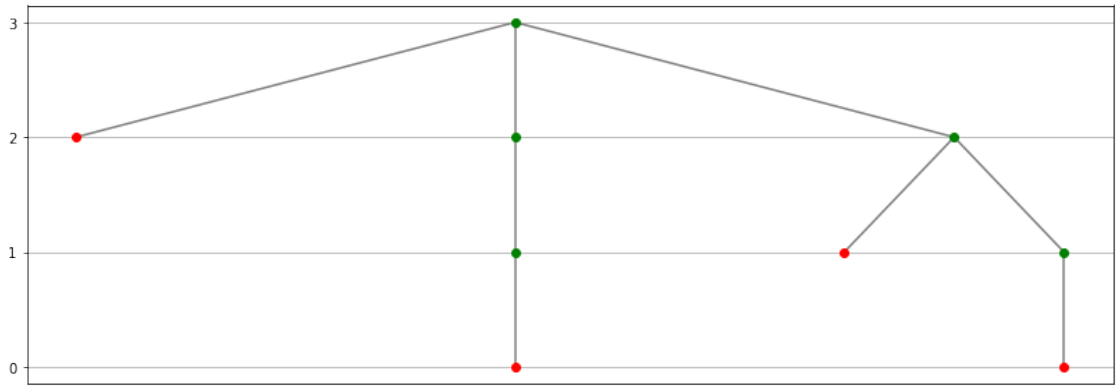
Démonstration. Voici une idée de la preuve, qui se fait par récurrence forte sur la hauteur de T.

- Si T est de hauteur 0, il possède un seul noeud (sa racine) et aucun fils. Clairement, le seul étiquetage possible est d'étiqueter sa racine par $\emptyset$.
- Soit $n \in \mathbb{N}^*$. Supposons la propriété vérifiée pour tous les arbres de hauteur strictement inférieure à n. Soit T un arbre de hauteur n. Soient x sa racine et $T_1, \ldots, T_m$ les fils de x. Les

T_i étant de hauteur strictement inférieure à n il existe un unique $x_i \in V$ tel que par $T_i \simeq T(x_i)$. Le seul étiquetage de T possible est l'étiquetage de la racine de T par $A = \{x_1, \ldots, x_m\}$. On a alors $T \simeq T(A)$. □

Ce que nous dit ce théorème, c'est que le fait de ne pas afficher les étiquettes n'amène aucune ambiguïté : on peut retrouver les valeurs des étiquettes, uniquement à partir de la « forme » de l'arbre. Prenons un exemple.

[34]:
```
A = psi(13)
tracer_arbre(A)
```



- Le fils « gauche » de la racine est clairement $T(\emptyset)$.
- Le fils du « milieu » est $T(x)$ où x a un unique élément, cet élément ayant un unique élément, qui est $\emptyset$. Ainsi, $x = \{\{\emptyset\}\} = \sigma_2$.
- Le fils « droit » de la racine a deux éléments. Celui de « gauche » est $\emptyset$, celui de droite est $\{\emptyset\}$. Ce fils droit est donc $T(\{\emptyset, \{\emptyset\}\}) = T(\overline{2})$.

Ainsi, $T = T(\{\overline{0}, \sigma_2, \overline{2}\})$. Vérifions …

[35]:
```
A = psi(13)
print(A)
print(tostr(A))
```

```
[[], [[[]]], [[], [[]]]]
{0, 2,2}
```

1.4 4. Énumérer V

1.4.1 4.1 Successeur

Pour tout $A \in V$, notons A^+ le *successeur* de A, défini par $\varphi(A^+) = \varphi(A) + 1$.

Proposition. Soit m le plus petit ensemble héréditairement fini (au sens de la relation $\leq$ définie sur V) tel que $m \notin A$. On a alors

$$A^+ = \{m\} \cup \{x \in A : x > m\}$$

Démonstration. Par définition de m, on a la réunion disjointe

$$A = A' \cup A'' = \{x \in V : x < m\} \cup \{x \in A : m < x\}$$

De là,

$$\varphi(A) = \varphi(A') + \varphi(A'') = \sum_{k \in \mathbb{N}, k < \varphi(m)} 2^k + \sum_{x \in A, x > m} 2^{\varphi(x)}$$

Remarquons que

$$1 + \sum_{k < \varphi(m)} 2^k = 2^{\varphi(m)}$$

De là,

$$\varphi(A^+) = 2^{\varphi(m)} + \sum_{x \in A, x > m} 2^{\varphi(x)} \ \square$$

La fonction `succ` prend un ensemble $A \in V$ en paramètre. Elle renvoie A^+.

[36]:
```
def succ(A):
    k = 0
    B = []
    while k < len(A) and A[k] == B:
        B = succ(B)
        k = k + 1
    return [B] + A[k:]
```

Voici un exemple d'ensemble pour lequel la fonction `succ` donne une réponse immédiate, alors que l'image par φ de cet ensemble est un entier colossalement grand.

[37]:
```
A = sigma(100)
print(tostr(A))
print(tostr(succ(A)))
print(tostr(succ(succ(A))))
print(tostr(succ(succ(succ(A)))))
```

```
 100
{0, 99}
{1, 99}
{0,1, 99}
```

1.4.2 4.2 Itérer sur les ensembles héréditairement finis

La fonction `HF0` est un itérateur. Il prend en paramètres un ensemble $A \in V$ et un entier n puis énumère n ensembles héréditairement finis à partir de A inclus.

[38]:
```
def HF0(A, n):
    for k in range(n):
        yield A
        if k < n: A = succ(A)
```

Dans l'exemple ci-dessous, on affiche 10 ensembles de V à partir de $\psi(123456)$.

[39]:
```
for A in HF0(psi(123456), 10):
    print(phi(A), tostr(A))
```

```
123456 {{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123457 {0,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123458 {1,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123459 {0,1,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123460 { 2,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123461 {0, 2,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123462 {1, 2,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123463 {0,1, 2,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123464 {2,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123465 {0,2,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
```

1.4.3 4.3 Prédécesseur

Pour tout $A \in V$ non vide, notons A^- le *prédécesseur* de A, c'est à dire l'ensemble tel que $(A^-)^+ = A$.

Proposition. Soit $x = \min A$.

- Si $x = \emptyset$, alors $A^- = A \setminus \{\emptyset\}$.
- Sinon, $A^- = \{\emptyset, \emptyset^+, \emptyset^{++}, \ldots, x^-\} \cup (A \setminus \{x\})$.

Démonstration. Soit $B = \{\emptyset, \emptyset^+, \emptyset^{++}, \ldots, x^-\} \cup (A \setminus \{x\})$. Le plus petit $m \in V$ tel que $m \notin B$ est $m = x$. On a donc

$$B^+ = \{x\} \cup \{y \in A : x < y\} = \{x\} \cup (A \setminus \{x\}) = A \ \square$$

[40]:
```
def pred(A):
    X = A[0]
    B = []
    y = []
    while inferieur_strict(y, X):
        B = B + [y[:]]
        y = succ(y)
```

```
    return B + A[1:]
```

[41]:
```
A = psi(123456)
print(tostr(A))
```

```
{{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
```

[42]:
```
A = pred(A)
print(tostr(A))
print(phi(A))
```

```
{0,1, 2,2, 3,{0, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123455
```

[43]:
```
A = sigma(123)
print(pred(succ(A)) == A)
```

```
True
```

Remarque. Contrairement à la fonction `succ`, la fonction `pred` peut renvoyer (ou plutôt ne renvoie pas) un ensemble de taille gigantesque. Par exemple, nous ne pourrions pas demander le prédécesseur de σ_{10} parce que $\sigma_{10}^- = V_9$, un ensemble de taille … $2^{<9>}$!

La fonction `HF` est une généralisation de l'itérateur `HF0` défini un peu plus haut. Il permet d'énumérer $|n|$ éléments de V à partir de A dans l'ordre croissant ou dans l'ordre décroissant, selon que $n \geq 0$ ou $n < 0$. La remarque ci-dessus incite à la prudence quant au choix de l'ensemble A de départ.

[44]:
```
def HF(A, n):
    if n >= 0:
        for k in range(n):
            yield A
            if k < n: A = succ(A)
    else:
        for k in range(-n):
            yield A
            if k < -n: A = pred(A)
```

[45]:
```
for A in HF(psi(123456), -10):
    print(phi(A), tostr(A))
```

```
123456 {{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123455 {0,1, 2,2, 3,{0, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123454 {1, 2,2, 3,{0, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123453 {0, 2,2, 3,{0, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123452 { 2,2, 3,{0, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123451 {0,1,2, 3,{0, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123450 {1,2, 3,{0, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123449 {0,2, 3,{0, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
```

```
123448 {2, 3,{0, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123447 {0,1, 2, 3,{0, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
```

[46]:
```
for A in HF(psi(123456), 10):
    print(phi(A), tostr(A))
```

```
123456 {{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123457 {0,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123458 {1,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123459 {0,1,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123460 { 2,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123461 {0, 2,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123462 {1, 2,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123463 {0,1, 2,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123464 {2,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
123465 {0,2,{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
```

1.5 5. Opérations ensemblistes

Étant donnés deux ensemble s $A, B \in V$, a-t-on $A \cup B \in V$? Si oui, que vaut rg $A \cup B$? Que dire pour les autres opérations ensemblistes, intersection, ensemble des parties, etc ?

1.5.1 5.1 Appartenance, inclusion

Étant donnés deux ensembles $x, A \in V$, il suffit évidemment de l'instruction `x in A` pour décider si $x \in A$. Cette instruction cache un algorithme qui demande $O(|A|)$ tests d'égalité de listes. On peut faire mieux en exploitant le fait que nous représentons l'ensemble A par une *liste triée*. La fonction `appartient` ci-dessous effectue $O(\lg A)$ tests d'égalités de listes. Elle effectue une recherche dichotomique de x dans la liste triée A.

[47]:
```
def appartient(x, A):
    n = len(A)
    if n == 0: return False
    elif inferieur_strict(x, A[0]) or inferieur_strict(A[n - 1], x): return False
    else:
        i = 0
        j = n
        while j - i > 1:
            k = (i + j) // 2
            if x == A[k]: return True
            elif inferieur_strict(x, A[k]): j = k
            else: i = k
        return A[i] == x
```

[48]:
```
A = neumann(17)
x = neumann(14)
```

```
print(appartient(x, A))
```

```
True
```

Proposition. Soit $B \in V$. Soit A un ensemble tel que $A \subseteq B$. Alors, $A \in V$ et $\text{rg } A \leq \text{rg } B$.

Démonstration. Rappelons qu'un ensemble est dans V si et seulement si il est fini et inclus dans V. On a $A \subseteq B \subseteq V$, donc A est fini (car inclus dans l'ensemble fini B) et inclus dans V. Ainsi, $A \in V$.

Passons au rang de A. Si $A = \emptyset$, c'est évident. Sinon, $\text{rg } A = \max\{\text{rg } x : x \in A\} + 1 \leq \max\{\text{rg } x : x \in B\} + 1 = \text{rg } B$. □

[49]:
```
def inclus(A, B):
    for x in A:
        if not appartient(x, B): return False
    return True
```

1.5.2 5.2 Intersection

Proposition. Soient $A, B \in V$. On a $A \cap B \in V$ et $\text{rg } A \cap B \leq \min(\text{rg } A, \text{rg } B)$.

Démonstration. En effet, $A \cap B \subseteq A$ et $A \cap B \subseteq B$. □

Corollaire. Soit $n \geq 1$. Soient $A_1, \ldots, A_n$ n ensembles héréditairement finis. Alors, $\bigcap_{k=1}^{n} A_k \in V$ et $\text{rg } \bigcap_{k=1}^{n} A_k \leq \min\{\text{rg } A_k : k \in [\![1, n]\!]\}$.

Démonstration. Récurrence sur n.

Les ensembles ayant une représentation dans laquelle leurs éléments sont triés, il est facile de calculer une intersection.

On initialise la future intersection C à la liste vide.

Tant que A et B sont non vides :

- Si $\min A < \min B$, alors $\min A \notin B$. On retire $\min A$ de A.
- Si $\min A > \min B$, alors $\min B \notin A$. On retire $\min B$ de B.
- Si $\min A = \min B$, alors $\min B \in A \cap B$. On retire $\min A$ de A et $\min B$ de B, et on rajoute la valeur commune à la liste C.

[50]:
```
def intersection(A, B):
    C = []
    while A != [] and B != []:
        if inferieur_strict(A[0], B[0]):
            A = A[1:]
        elif inferieur_strict(B[0], A[0]):
            B = B[1:]
        else:
            C.append(A[0])
            A = A[1:]
            B = B[1:]
```

```
    return C
```

[51]:
```
A = psi(123456)
B = psi(1234567)
print('A            : ', tostr(A))
print('B            : ', tostr(B))
print('Intersection : ', tostr(intersection(A, B)))
```

```
A            :  {{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
B            :
{0,1, 2, 3,{0,2},{1,2},{ 2,2},{1, 2,2},{0,1, 2,2},{0, 3},{ 2, 3}}
Intersection :  {{0,2},{1, 2,2},{0,1, 2,2}}
```

1.5.3 5.3 Réunion

Proposition. Soient $A, B \in V$. On a $A \cup B \in V$ et $\text{rg } A \cup B = \max(\text{rg } A, \text{rg } B)$.

Démonstration. $A \cup B$ est fini et inclus dans V, donc $A \cup B \in V$. Passons au calcul du rang.

Si A ou B est vide le résultat est évident. Supposons donc A et B non vides.

On a $A \subseteq A \cup B$, donc $\text{rg } A \leq \text{rg } A \cup B$. de même, $\text{rg } B \leq \text{rg } A \cup B$, et donc $\max(\text{rg } A, \text{rg } B) \leq \text{rg } A \cup B$.

Pour tout $x \in A$, $\text{rg } x \leq \text{rg } A - 1$. Pour tout $x \in B$, $\text{rg } x \leq \text{rg } B - 1$. De là, pour tout $x \in A \cup B$,

$$\text{rg } x \leq \max(\text{rg } A - 1, \text{rg } B - 1) = \max(\text{rg } A, \text{rg } B) - 1$$

On en déduit que $\text{rg } A \cup B \leq \max(\text{rg } A, \text{rg } B)$. □

Corollaire. Soit $n \geq 1$. Soient $A_1, \ldots, A_n$ n ensembles héréditairement finis. Alors, $\bigcup_{k=1}^{n} A_k \in V$ et $\text{rg } \bigcup_{k=1}^{n} A_k = \max\{\text{rg } A_k : k \in [\![1, n]\!]\}$.

Démonstration. Récurrence sur n.

Voici la fonction `reunion`. Elle prend en paramètres deux ensembles $A, B \in V$ et renvoie leur réunion. Elle fonctionne comme la fonction `intersection`.

[52]:
```
def reunion(A, B):
    C = []
    while A != [] and B != []:
        if inferieur_strict(A[0], B[0]):
            C.append(A[0])
            A = A[1:]
        elif inferieur_strict(B[0], A[0]):
            C.append(B[0])
            B = B[1:]
        else:
            C.append(A[0])
            A = A[1:]
```

```
            B = B[1:]
    C = C + B[:]
    C = C + A[:]
    return C
```

[53]:
```
A = psi(123456)
B = psi(1234567)
print('A            : ', tostr(A))
print('B            : ', tostr(B))
print('Intersection : ', tostr(intersection(A, B)))
print('Réunion      : ', tostr(reunion(A, B)))
```

```
A            :  {{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
B            :
{0,1, 2, 3,{0,2},{1,2},{ 2,2},{1, 2,2},{0,1, 2,2},{0, 3},{ 2, 3}}
Intersection :  {{0,2},{1, 2,2},{0,1, 2,2}}
Réunion      :  {0,1, 2,{1, 2}, 3,{0,2},{1,2},{ 2,2},{0, 2,2},{1, 2,2},{0,1, 2,2
}, 4,{0, 3},{ 2, 3}}
```

1.5.4 5.4 Différence

Proposition. Soient $A, B \in V$. Alors, $A \setminus B \in V$ et $\operatorname{rg} A \setminus B \leq \operatorname{rg} A$.

Démonstration. En effet, $A \setminus B \subseteq A$.

De même que pour l'intersection et la réunion, la différence de deux ensembles se calcule sans difficulté.

[54]:
```
def difference(A, B):
    C = []
    while A != [] and B != []:
        if inferieur_strict(A[0], B[0]):
            C.append(A[0])
            A = A[1:]
        elif inferieur_strict(B[0], A[0]):
            B = B[1:]
        else:
            A = A[1:]
            B = B[1:]
    if B == []:
        C = C + A[:]
    return C
```

[55]:
```
A = psi(123456)
B = psi(1234567)
print('A            : ', tostr(A))
print('B            : ', tostr(B))
print('Intersection : ', tostr(intersection(A, B)))
```

```
print('Réunion      : ', tostr(reunion(A, B)))
print('Différence   : ', tostr(difference(A, B)))
```

```
A            :  {{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
B            :
{0,1, 2, 3,{0,2},{1,2},{ 2,2},{1, 2,2},{0,1, 2,2},{0, 3},{ 2, 3}}
Intersection :  {{0,2},{1, 2,2},{0,1, 2,2}}
Réunion      :  {0,1, 2,{1, 2}, 3,{0,2},{1,2},{ 2,2},{0, 2,2},{1, 2,2},{0,1, 2,2
}, 4,{0, 3},{ 2, 3}}
Différence   :  {{1, 2},{0, 2,2}, 4}
```

1.5.5 5.5 Parties

Proposition. Soit $A \in V$. On a $\mathcal{P}(A) \in V$ et $\text{rg } \mathcal{P}(A) = \text{rg } A + 1$.

Démonstration. Pour tout $X \in \mathcal{P}(A)$, $X \subseteq A$ et donc $\text{rg } X \leq \text{rg } A$. On en déduit que

$$\text{rg } \mathcal{P}(A) = \max\{\text{rg } X, X \in \mathcal{P}(A)\} + 1 \leq \text{rg } A + 1$$

De plus, $A \in \mathcal{P}(A)$, donc $\text{rg } \mathcal{P}(A) \geq \text{rg } A + 1$. □

La fonction `parties` ci-dessous fait appel à la fonction `reunion` pour garantir que les ensembles renvoyés sont bien triés dans l'ordre croissant de leurs éléments. Elle fonctionne comme suit.

Soit $A \in V$.

- Si $A = \emptyset$, alors $\mathcal{P}(A) = \{\emptyset\}$.
- Sinon, soit $x = \min A$. Soit $A' = A \setminus \{x\}$. Soit $P = \mathcal{P}(A')$. On a

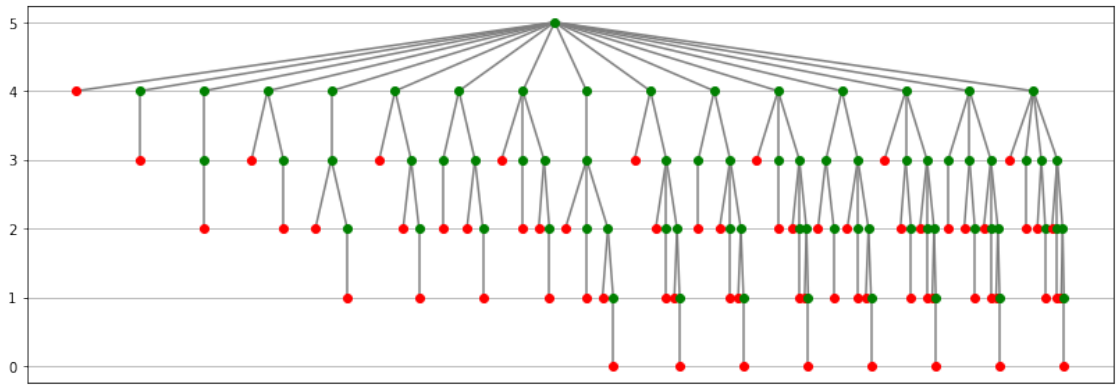
$$\mathcal{P}(A) = P \cup \{y \cup \{x\} : y \in P\}$$

[56]:
```
def parties(A):
    if A == []: return [[]]
    else:
        P = parties(A[1:])
        P1 = [reunion([A[0]], X) for X in P]
        return reunion(P, P1)
```

[57]:
```
A = parties(neumann(4))
print(tostr(A))
print(len(A), rang(A))
```

```
{0,1, 2,2,{2},{0,2},{1,2},3,{3},{0,3},{1,3},{0,1,3},{2,3},{0,2,3},{1,2,3},4}
16 5
```

[58]:
```
tracer_arbre(parties(neumann(4)))
```



1.5.6 5.6 Couples

Nous utilisons ici une définition de couple due à von Neumann.

Définition. Étant donnés deux ensembles $A, B \in V$, le *couple* (A, B) est

$$(A, B) = \{\{A\}, \{A, B\}\}$$

Proposition. Soient $A, B, C, D \in V$. On a $(A, B) = (C, D) \iff A = C \wedge B = D$.

Démonstration. Laissée en exercice. Considérer deux cas, $A = B$ et $A \neq B$.

Proposition. Soient $A, B \in V$. On a $(A, B) \in V$ et $\text{rg}\ (A, B) = \max(\text{rg}\ A, \text{rg}\ B) + 2$.

Démonstration. Soient $m = \text{rg}\ A$ et $n = \text{rg}\ B$. On a $\text{rg}\ \{A\} = m+1$ et $\text{rg}\ \{A, B\} = \max(m, n)+1$. De là,

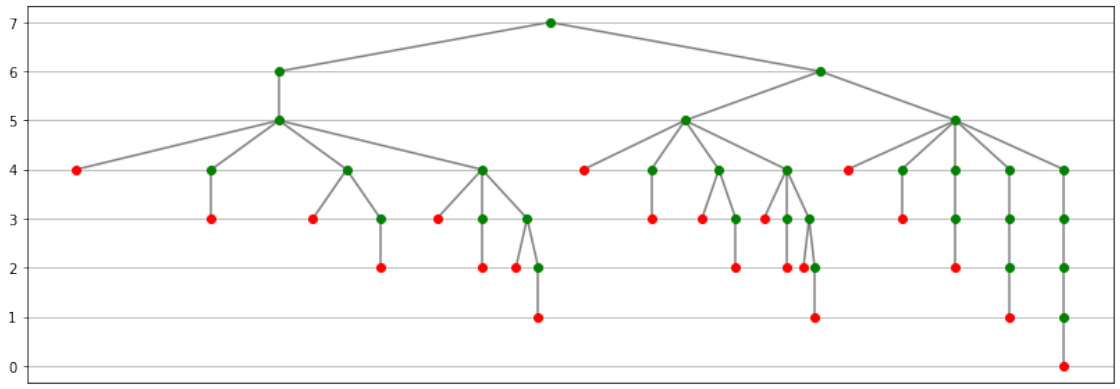
$$\text{rg}\ (A, B) = \max(m + 1, \max(m, n) + 1) + 1 = \max(m, n) + 2\ \square$$

La fonction `couple` renvoie le couple de von Neumann (A, B).

[59]:
```
def couple(A, B):
    if A == B:
        return [[A]]
    elif inferieur_strict(A, B):
        return [[A], [A, B]]
    else:
        return [[A], [B, A]]
```

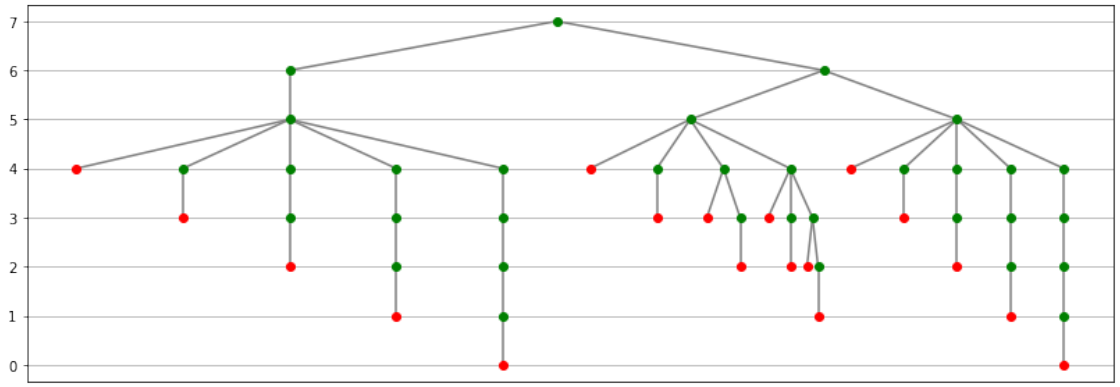
[60]:
```
C = couple(neumann(4), tau(5))
print(tostr(C))
tracer_arbre(C)
```

```
{{4},{4, 5}}
```



[61]:
```
C = couple(tau(5), neumann(4))
print(tostr(C))
tracer_arbre(C)
```

```
{{ 5},{4, 5}}
```



Comment récupérer les ensembles A et B lorsqu'on possède le couple $C = (A, B)$?

- Tout d'abord, $\{A\} = \min C$ et donc $A = \min \min C$.
- Si $\min C = \max C$, alors $B = A$.
- Snon, $\max C \setminus \min C = \{A, B\} \setminus \{A\} = \{B\}$ et donc $B = \min(\max C \setminus \min C)$.

[62]:
```
def composantes(C):
    A = C[0][0]
    if len(C) == 1:
        return (A, A)
    else:
        B = difference(C[1], C[0])[0]
        return (A, B)
```

[63]:
```
C = couple(psi(123456789), psi(987654321))
A, B = composantes(C)
print(A == psi(123456789), B == psi(987654321))
```

```
True True
```

La fonction `est_couple` prend en paramètre un ensemble $A \in V$. Elle renvoie `True` si A est un couple et `False` sinon.

[64]:
```
def est_couple(A):
    if len(A) == 1 and len(A[0]) == 1: return True
    elif len(A) == 2 and len(A[0]) == 1 and len(A[1]) == 2 and inclus(A[0],␣
 ↪A[1]): return True
    else: return False
```

Parmi les 2^{16} premiers ensembles de V (c'est à dire parmi les éléments de V_4), lesquels sont des couples ?

[65]:
```
for A in HF([], 65536):
    if est_couple(A): print(phi(A), tostr(A))
```

```
2  2
4  3
10 {1,2}
12 { 2,2}
16  4
34 {1,{0, 2}}
48 { 3,{0, 2}}
68 { 2,{1, 2}}
80 { 3,{1, 2}}
256 {{2}}
514 {1,{0,2}}
768 {{2},{0,2}}
1028 { 2,{1,2}}
1280 {{2},{1,2}}
4112 { 3,{ 2,2}}
4352 {{2},{ 2,2}}
```

Profitons de ce que nous avons un nouveau type d'ensemble dans V, les couples, pour réécrire la fonction `tostr`. Notre nouvelle fonction, `tostring`, détecte si l'ensemble A est un couple (x, y). Elle renvoie dans ce cas une représentation correcte pour l'ensemble, sous la forme $< x, y >$ (les crochets se distinguent plus facilement des accolades que les parenthèses).

Remarquons que

- $\overline{n}$ et τ_n ne sont jamais des couples, comme il est facile de le vérifier.
- Enn revanche, pour tout $n \geq 2$, σ_n est un couple. En effet,

$$\sigma_n = \{\{\sigma_{n-2}\}\} = (\sigma_{n-2}, \sigma_{n-2})$$

[66]:
```
def tostring(A):
    n = rang(A)
    if A == neumann(n): return str(n) # ' ' +
    elif A == sigma(n): return ' ' + str(n)
    elif A == tau(n): return ' ' + str(n)
    elif est_couple(A):
        x, y = composantes(A)
        return '<' + tostring(x) + ',' + tostring(y) + '>'
    else:
        s = '{'
        l = len(A)
        for k in range(l):
            s = s + tostring(A[k])
            if k < l - 1: s = s + ','
        s = s + '}'
        return s
```

[67]:
```
A = couple(psi(123), psi(321))
print(tostring(A))
```

```
<{0,1,2, 3,{0, 2},{1, 2}},{0,{1, 2},{2}}>
```

Quels sont les éléments de V_5 qui sont des couples ? Il y en a 16.

[68]:
```
for A in HF([], 65536):
    if est_couple(A): print(phi(A), tostring(A))
```

```
2  2
4  3
10 <0,1>
12 <1,0>
16  4
34 <0, 2>
48 < 2,0>
68 <1, 2>
80 < 2,1>
256 <2,2>
514 <0,2>
768 <2,0>
1028 <1,2>
1280 <2,1>
4112 < 2,2>
4352 <2, 2>
```

Peut-on définir la notion de triplet, de quadruplet, etc ? Oui, bien sûr.

Définition. On définit par récurrence sur n le *n-uplet* $(A_1, \ldots, A_n)$ en posant pour tout $n \geq 2$,

$$(A_1, \ldots, A_{n+1}) = (A_1, (A_2, \ldots, A_{n+1}))$$

Le lecteur consciencieux montrera que deux n-uplets sont égaux si et seulement si leurs composantes sont égales. On peut sans difficulté écrire des fonctions manipulant des n-uplets. Par exemple,

[69]:
```
def triplet(A, B, C):
    return couple(A, couple(B, C))
```

[70]:
```
def est_triplet(A):
    return est_couple(A) and est_couple(composantes(A)[1])
```

[71]:
```
def composantes3(T):
    A, U = composantes(T)
    B, C = composantes(U)
    return (A, B, C)
```

Quels sont les éléments de V_5 qui sont des triplets ? Il y en a 4.

[72]:
```
for A in HF([], 65536):
    if est_triplet(A): print(phi(A), tostring(A))
```

```
16  4
34 <0, 2>
68 <1, 2>
4352 <2, 2>
```

1.5.7 5.7 Produit cartésien

Définition. Le *produit cartésien* des ensembles A et B est $A \times B = \{(x, y) : x \in A, y \in B\}$.

Proposition. Soient $A, B \in V$. On a $A \times B \in V$. Si A et B sont non vides, alors rg $A \times B = \max(\text{rg } A, \text{rg } B) + 2$.

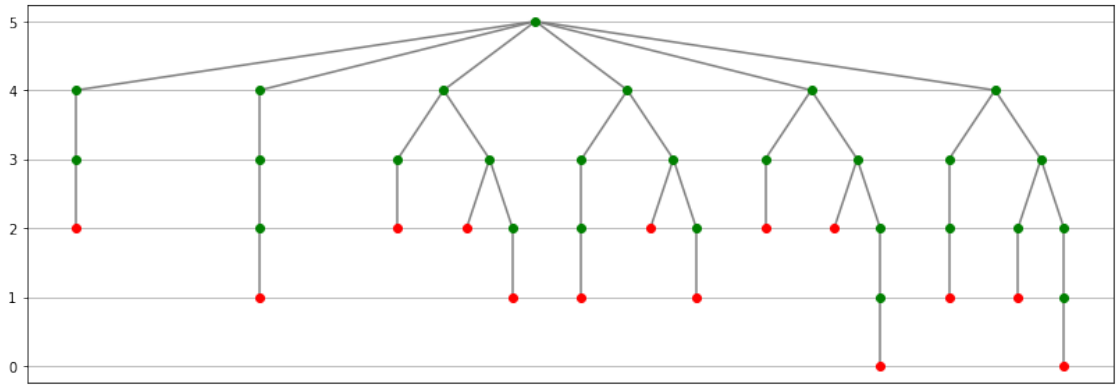
Démonstration. Soient m et n les rangs respectifs de A et B. Pour tout $x \in A$, rg $x \leq m - 1$ et pour tout $y \in B$, rg $y \leq n - 1$. De là, rg $(x, y) \leq \max(m - 1, n - 1) + 2 = \max(m, n) + 1$. On en déduit que rg $A \times B \leq \max(m, n) + 2$.

De plus, il existe $x \in A$ de rang $m - 1$ et $y \in B$ de rang $n - 1$. On a alors rg $(x, y) = \max(m, n) + 1$ donc rg $A \times B \geq \max(m, n) + 2$. □

[73]:
```
def produit(A, B):
    C = []
    for x in A:
        for y in B:
            C = reunion(C, [couple(x, y)])
    return C
```

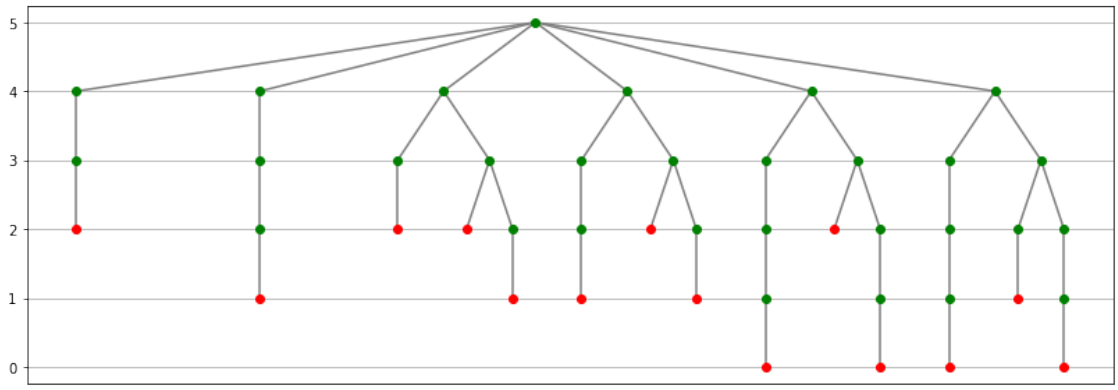
[74]:
```
A = produit(tau(2), tau(3))
print(tostring(A))
tracer_arbre(A)
```

{ 2, 3,<0,1>,<1,0>,<0, 2>,<1, 2>}



[75]:
```
A = produit(tau(3), tau(2))
print(tostring(A))
tracer_arbre(A)
```

{ 2, 3,<0,1>,<1,0>,< 2,0>,< 2,1>}



[ ]: