TD2 : Recherche exhaustive et bactrack.

Rq : les ensembles sont des multi-ensembles, des valeurs peuvent apparaître plusieurs fois.

Exercice 1.

(1) Les sous-ensembles de T peuvent être représentés par les mots à n lettres de {0,1}

Ex : $T = [-5, 2, 7, -3, 2, -6]$

le mot $\omega = 010110$ représente l'ensemble $\{2, -3, 2\}$.

Autrement dit, $T[i]$ appartient à notre sous-ensemble ssi $\omega_i = 1$.

(2) Rappel : dans le cours on a la fonction MOT-SUIVANT(ω) qui appelée successivement sur $\overbrace{0\cdots0}^{n}$ parcourt tous les mots de $\{0,1\}^n$ lexicographiquement avec MOT-SUIVANT(1…1) retourne "fin"

```
SOMME-SOUS-ENS (T, n, t)
    ω ← 0…0
    Tant que ω ≠ "fin":
        Som ← 0.
        Pour i=0 à n-1.
            Som ← Som + T[i] × ω[i]
        Si Som = t  retourner "bingo!" et ω …
        ω ← MOT-SUIVANT(ω).
    Retourner "Pas trouvé !"
```

ou :
```
S ← ϕ.
Pour i=0 à n-1
    Si ω[i] = 1 alors.
        LS.append(T[i]).
Retourner S.
```
// on fait la somme des éléments T[i] avec ω[i] = 1.

Complexité : . la boucle "Pour" prend un temps $O(n)$

. la boucle "tant que" s'exécute au pire autant de fois qu'il y a de mots de taille n, c'est à dire 2^n fois.

En tout on a $O(n.2^n)$.

(3) Exemple $t = 12$ $[3, 2, 7, 9, 1]$.

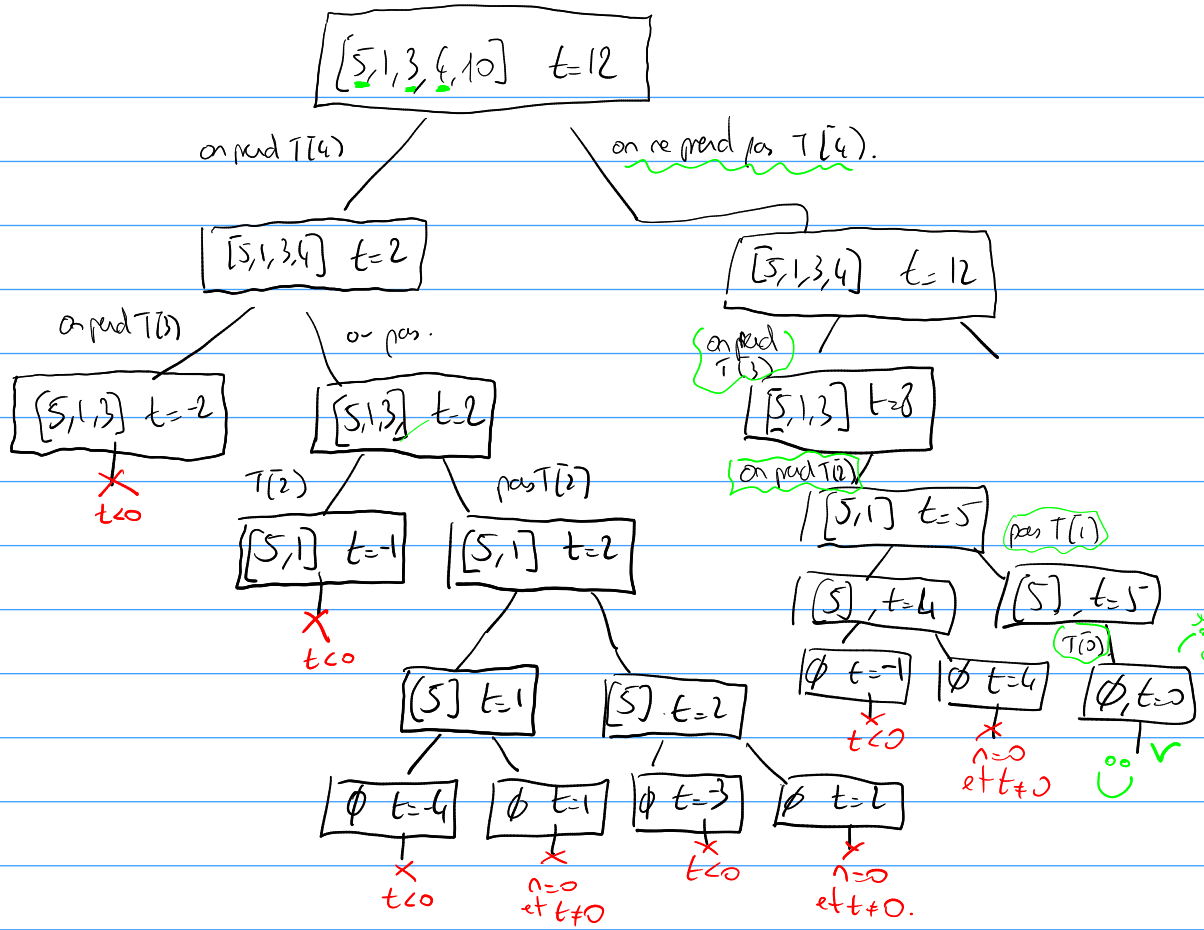
Il faut : on prend T[4]. → $t = 11$. $[3, -2, 7, 9]$

on ne prend pas T[4] → $t = 12$. $[3, 2, 7, 9]$.

ou n=0

```
SOMME-SOUS-ENS-BCTK (T, t).
    Si t=0 retourner VRAI
    Si t<0 retourner FAUX.
    retourner SOMMESOUS-ENS-BCTK (T[0:n-2], t-T[n-1]) ou
                                                    → on prend T(n-1)
              SOMME-SOUS-ENS-BCTK (T[0:n-2], t)
                                                    ← on ne prend pas T(n-1).
```

(4)



Q1: L'algo a une complexité proportionnelle au nb d'appel rec.
Si on note cette valeur $t(n)$ pour un tableau de taille n.

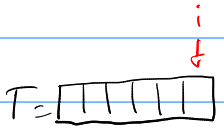
$t(n) \leq 2.t(n-1)$ $\rightsquigarrow$ $t(n) \leq 2^n$ on a une complexité en $O(2^n)$

-Exercice 2-

(1) Le compteur $\overbrace{0,0,\dots,0}^{n}$ et finit en $\overbrace{k-1,\dots,k-1}^{n}$. Il y a k^n éléments à parcourir, dans l'ordre lexicographique.

Par ex si $k=4$ et $n=5$: $10233 \rightarrow 10300 \rightarrow 10301 \rightarrow 10302 \dots$

On adapte l'algo de l'addition :




```
Suivant(T).
    i ← n-1        // n = taille de T.
    Tant Que (T[i] = k-1 et i ≥ 0).
        | T[i] ← 0
        | i ← i-1
    Si i = -1 retourner "fin" // T ne contenant que k-1 comme valeur.
    T[i] ← T[i] + 1.
```

Complexité : La boucle Tant Que fait au + n tours $\rightarrow O(n)$

② Ex: $n=5$ et $h=4$ $\quad 01233 \to 01333 \to 02222 \to 02223 \to 02233$

```
Suivant (T).
    i ← n-1        // n = taille de T.
    tantQue (T[i] = h-1 et i ≥ 0).      // on détermine le + grande suffixe
        i ← i-1                          // qui contient que des "h-1"
    Si i = -1 retourner "fin"  // T ne contenait que h-1 comme valeur.
    T[i] ← T[i]+1.
    Pour j = i+1 à n-1
        T[j] ← T[i]
```

Pour compter le nbre de n-uplets croissants sur $\{0,\dots,h-1\}$, on va les encoder de la façon suivante : on écrit $0,1,\dots,h-1$ et on intercale n "traits" avant les nombres qu'on veut choisir dans le n-uplet :

$h=5$, $n=6$ $\quad \| 0\ 1\ |\ 2\ \|\ 3\ |\ 4$ donne le n-uplet : $0\,0\,2\,3\,3\,4$.

inversement $1\,2\,2\,4\,4\,4$ donne le code $0\,|\,1\,\|\,2\,3\,\|\,|\,4$

On admet qu'on a une bijection entre les n-uplets croissants et ces codes.

Un code contient exactement $k+n$ symboles (les h lettres de 0 à $h-1$ + n barres).

On veut placer les n barres $\rightsquigarrow$ il y a $\binom{n+h-1}{n}$ possibilités.

Donc $\binom{n+h-1}{n}$ n-uplets croissants sur $\{0,\dots,h-1\}$

↳ sauf à la fin du code qui finit par $h-1$.

Il y a $n+h-1$ emplacements possibles pour les barres.

Exo 3.

Exemple de cryptarithme :

$$\begin{array}{r} \text{OASIS} \\ +\,\text{SOLEIL} \\ \hline \text{MIRAGE} \end{array}$$

il faut attribuer une valeur dans $\{0,\dots,9\}$ à chaque lettre pour que l'addition soit correcte.

Dans l'exemple apparaissent A, E, G, I, L, M, O, R, S. Pour avoir 10 lettres, il faut en rajouter une : par ex : X.

① À chaque lettre, on associe un entier entre 0 et 9.

mots du cryptarithme.

```
Dico (a,b,c)
    i ← 0  // valeur à attribuer.
    dico ← ∅  // dictionnaire
    Pour l ∈ {a,b,c}  // on parcourt les lettres de a, b et c.
        Si l ∉ dico
            dico[l] ← i
            i ← i+1.
    Tant que i < 10   // s'il y a moins de 10 lettres dans a, b et c.
        choisir z ∉ d.
        dico[z] ← i
        i ← i+1.
```

(Sur l'exemple on aurait : (A,0), (E,1), (G,2), (I,3), (L,4), (M,5), (O,6), (R,7), (S,8), (Z,9)).

par ex... t

(3)

```
Valeur (m, dico, π)
    v ← 0;
    l ← longueur de m.
    Pour i = m-1 à 0
        v ← v + π(dico(m[i])) × 10^(m-1-i)
    Retourner v.
```

unité.

OASIS.

(4)

```
CRYPTARITHME (a,b,c)
    dic ← Dico(a,b,c)
    π ← [0,1,2,...,9].
    Tant Que π ≠ "Fin"
        Si Valeur(a, dico, π) + Valeur(b, dico, π) = Valeur(c, dico, π)
            Retourner "bingo!" et dic et π...
        π ← PERMUT_SUIVANTE(π).
    Retourner "Pas bingo".
```