Programmation dynamique :

```
def fiboNaif(n)
    if n==0 : return 0

    if n == 1 : return 1

    else : return fiboNaif(n-1) + fiboNaif(n-2)
```

// on calcul +t fois m chose mauvais.

// améliorer cet algo <=> enregistrer les calculs dans un array.

```
def fibo (n):
    T=[0,1]
    for (i in range (2,n+1):
        T.append (T[i-1]+T[i-2])
    return T[n]
```

// temp linéaire C.

// march python car ~> x2 array

↳ ~~au moins~~ dans idéal avoir la bonne taille dès le début

↳ Stratégie tableau dynamique (allocation)

↳ complexité amortie (analyse amortie)

// améliorer, stoquant que les 2 derniers termes calculés.

```
def fibo(n):
    if  n==0 : return 0
    elif  n==1 : return 1
    a=0
    b=1
    for i in range (2,n):
        a, b = b, a+b
    return a+b
```

// temp linéaire + sans utilisé mémoire

1) bottom_up
2) on stock le minimum d'info, mais suffisant pour être efficace

} programmation dynamique

"subset sum problem"

Existe-t-il un ss ensemble = x ?

ex: {2,8,3,14} {3,5,9}

V = 22 ✓ V = 6 ✗

// back-tracking (brute force)

```
BT(v,i,S){
  Si v == 0 Alors
    retourner S
  Si i=2n+1 Alors
    retourner ⊥
  essai = BT(v, i+1, S)
  Si essai = ⊥ ∧ v ≥ xi Alors
    retourner BT (v-xi, i+1, S ∪ {xi})
  Sinon retourner essai
```

$O(2^n)$

// programmation dynamique sac à dos

Autre approche:

Construire une table K[i,S] avec $0 \leq i \leq n$ et $0 \leq S \leq V$ tq:

K[i,S] = T ssi ∃ un ss-ensemble —

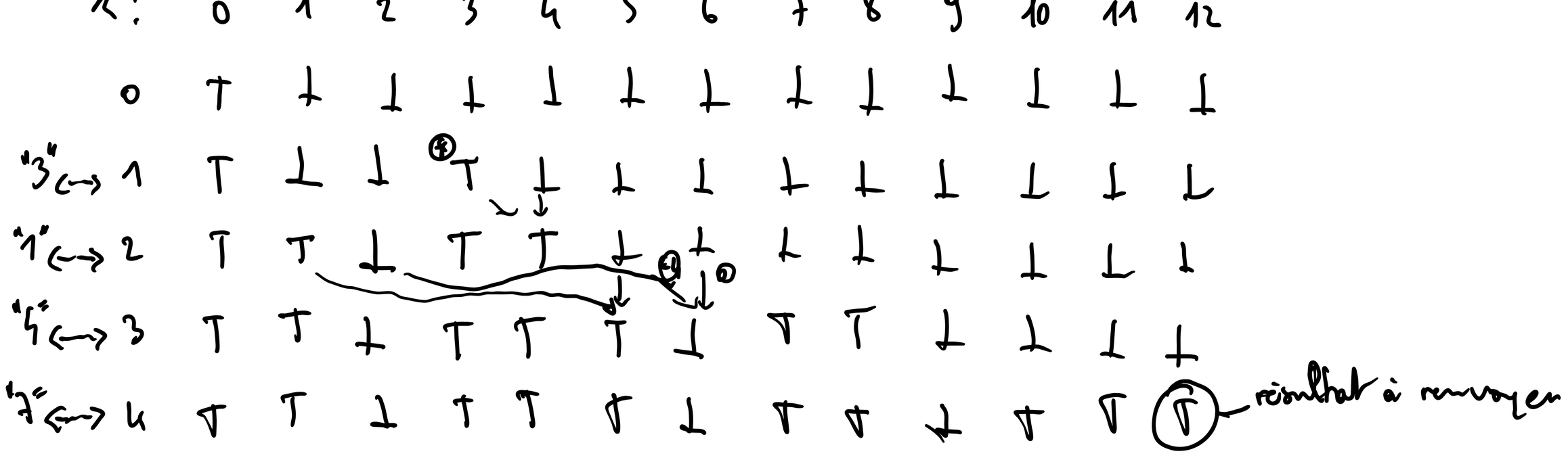
V

```
Subset Sum (V, E):
    K = tableau booleens de taille (n+1) x (V+1)
    Pour s = 1 ... V:
        K[0, s] = ⊥
    Pour i = 0 ... n
        K[i, 0] = ⊤

    Pour i = 1 à n:
        Pour s = 1 à V:
            Si x_i ≤ s Alors K[i, s] = K[i-1, s] ∨ K[i-1, s-x_i]
            Sinon K[i, s] = K[i-1, s]
    Retourner K[n, V]
```

*vraiment: K[1,3] = K[0,3] ∨ K[0,0]

K : 0 1 2 3 4 5 6 7 8 9 10 11 12



$C = O(n \cdot V)$ en temps et espace

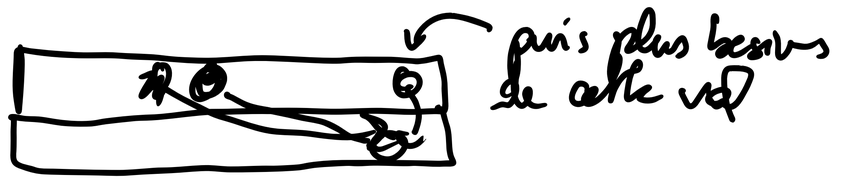
Prop : Après la $i^{ème}$ itération, $K[i,s] = \top$ ssi $\exists$ un ss-ensemble de $\{x_1, \dots, x_i\}$ de poids s.

[invariant]

Amélioration : utiliser un tableau unidimensionnel

```
SubSetSum(V, E):
    K = tableau booléens de taille V+1
    Pour s = 1 à V
        K[s] = ⊥
    K[0] = ⊤
    Pour i = 1 à n :
        Pour s = V à 1 :
            Si x_i ≤ S Alors  K[S] = K[s] ∨ K[s - x_i]
    Retourner K[V]
```

// on fait la maj de gauche à droite



Ex :

0	1	2	3	4	5	6	7	8	9	10	11	12
⊤	⊥	⊥	✱	⊥	⊥	✱	⊥	⊥	✱	⊥	⊥	✱
			⊤			⊤			⊤			⊤

// pas complexité pas prouver correction
↓
juste faire attention à la façon de parcourir

Prop : —

Construire une solution à partir de K[i, s]

```
s = V
res = ∅
Pour i = n à 1
    Si s ≥ x_i ∧ K[i-1, s-x_i] Alors | res = res ∪ {x_i}
                                    | s = s - x_i
                                    | Si s = 0 Alors return res
return ⊥
```

Subsets (V, E):

- K = tableau booléens de taille $V+1$
- Sol = tableau d'ens de taille $V+1$
- Pour $s = 1 \ldots V$

  $K[s] = \bot$, $sol[s] = \text{undef}$
- $K[0] = \top$, $sol[0] = \emptyset$
- Pour $i = 1$ à n :

  Pour $s = V$ à a_i :

  Si $K[s - x_i]$ Alors $sol[s] = sol[s - x_i] \cup \{x_i\}$

  $K[s] = \top$
- retourner $K[n, V]$

~

DM: Le compte est bon.