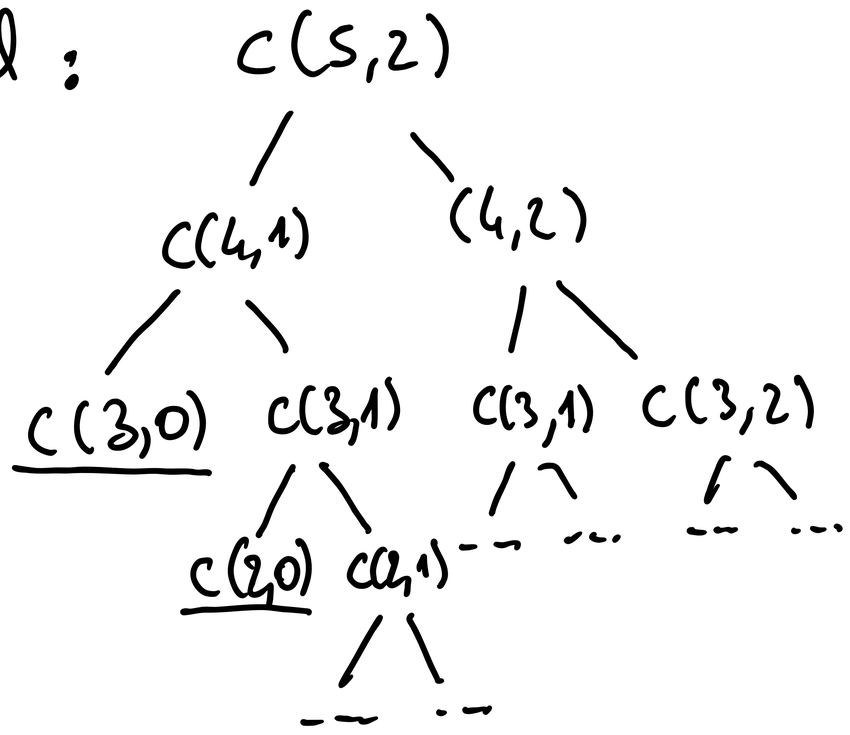
- suite TD4 -

Exo 2 :

triangle Pascal :



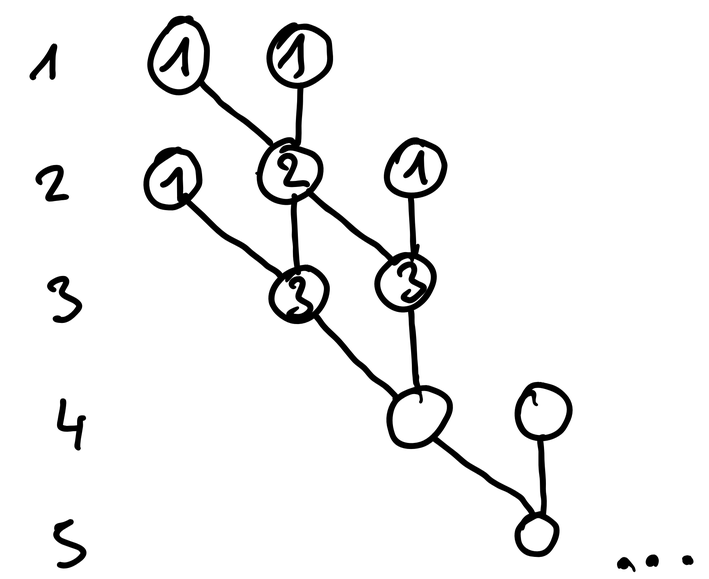

```
def C(n,p):
    if p==0 or p==n:
        return 1
    else:
        return C(n-1,p-1) + C(n-1,p)
```

$T_C(n,p)$: nbr d'appel recursif $\rightsquigarrow$ $T_C(n,p) \leq 2^n$

le p qui engendre le plus d'appel recursif : $p = \frac{n}{2}$

$\binom{n}{\frac{n}{2}} \sim \frac{2^n}{\sqrt{\pi \cdot \frac{n}{2}}}$

n \ p	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						



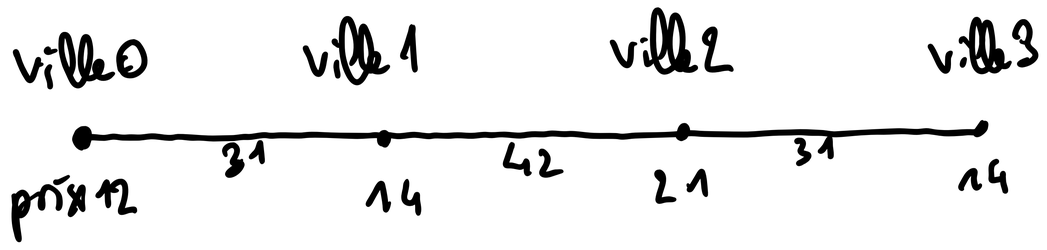
Prog dynamique $\rightsquigarrow$ C1(N,P) // dans un tableau + (si efficace, droite → gauche)

↳ O(N*P)

C1(5,3) $\rightsquigarrow$

n \ p	0	1	2	3
0	1	1	1	1
1	1	1	1	1
2	1	2	3	4
3	1	3	6	10
4	1	4	10	20
5	1	5	10	10

Exercice 3 :



charge 100% → 100 → 69 → 27 → 0

Carmoins cher

charge 0

charge 47.

charge 0

// pas optimale

128.2

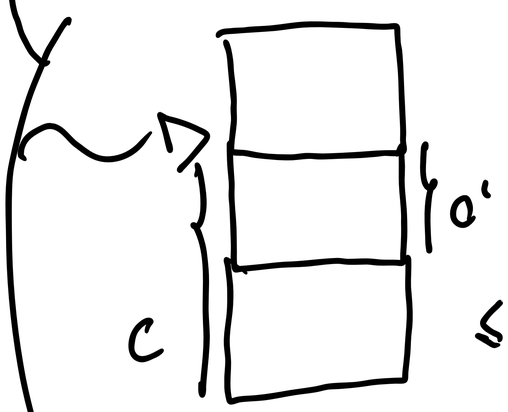
+ opti ▷ si pas ville < cher ensuite
↳ charger ⊕ .

~▷ 125.2

$$cout(i,c) = \begin{cases} C * prix[i] * 0.1 * & i = 0 \\ \min(cout[i-1], c - c' + dist[i-1] + prix[c']) & \\ c' \text{ tq } \max(0, c - 100 + dist[i-1]) \leq c' \leq c & \end{cases}$$

quel est la charge dans la station $i-1$? On l'appelle c_{i-1}

↳ $c_{i-1} - dist[i-1] + c' = c$ $\quad c_{i-1} = c - c' + dist[i-1]$



$\leq 100 - dist[i-1]$ // (max restant après changement ville)

$c - c' \leq 100 - dist[i-1]$

$c' \geq c - 100 + dist[i-1]$

```
def cout ( prix, dist, i, c):
    if i==0:
        return prix[i]*c*0.1
    else:
        mini= float("inf")
        for change in range (max (0,c +dist [i-1]-100 ),c+1):
            res = cout (prix, dist, i-1 ,c+dist[i-1 ]-change)+ prix [i]* change *0.1
            if res <mini :
                mini =res
        return mini
```

// version non efficace (recc + for + (pas dns ouvert))

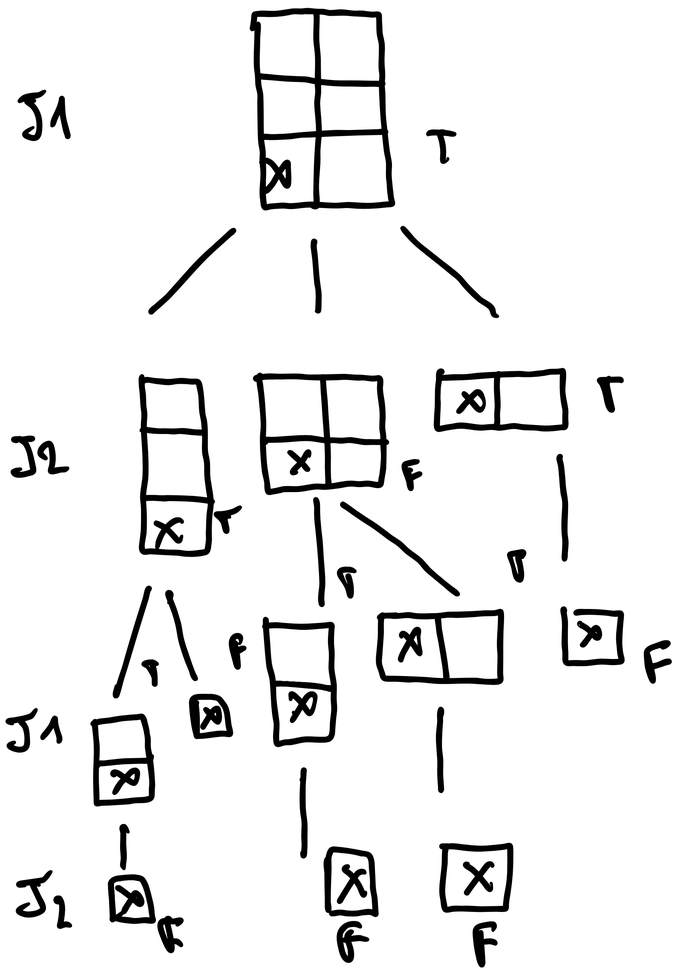
{ dynamique version

```
def coutdyn( prix , dist, i, c):
    cout = [[ 0 for j in range(101)] for k in range (i+1)]
    for c1 in range(101):
        cout[0][c1] = prix[0]* c1* 0.1

    for i1 in range (1,i+1):
        mini =float ("inf") ← for c1 in range(0,101) : mini =float("inf")
        for change in range ( max (0, c1+dist [i1-1]-100 ), c1+1):
            changeavant = c1 + dist[i1-1]- change
            res = cout[i1-1][changeavant] + prix[i1] * change * 0.1
            if res <mini : mini = res
        cout[i1][c1] mini
    return cout[i][c]
```

// Bonus QS

Exercice 4 :



// T / F => perdant / gagnant joueur actuel.

jeu (H, L, h, l)

jeu (H, L, h, l) = { False si H = L = h = l = 1

// TODO

// IDÉE : exprimer fait qnd on gagne (qnd 1 ligne remplie de true) => T ∧ ...