« L'obélisque autour de Hanoï »

Un artiste matheux a créé un obélisque qu'il a appelé « ***Hanoyen*** » constitué de *n* blocs de révolution en béton armé. Chacun de ses blocs est de même poids et l'épaisseur de béton notée ***e*** en centimètres est choisie constante. La ***figure_1*** donne une représentation blocs assemblés et désassemblés pour $n = 5$ et $e = 28$. On note p_k le bloc *n°k* avec $k \in \{0\,;\,1\,;\,2\,;\,3\,;\,\ldots\,;\,n-1\}$, p_0 est la flèche et p_{n-1} le socle se sont tous deux des cylindres pleins, les blocs restants sont des cylindres percés pour permettre l'emboîtement du bloc suivant comme cela est présenté dans la ***figure_2***.

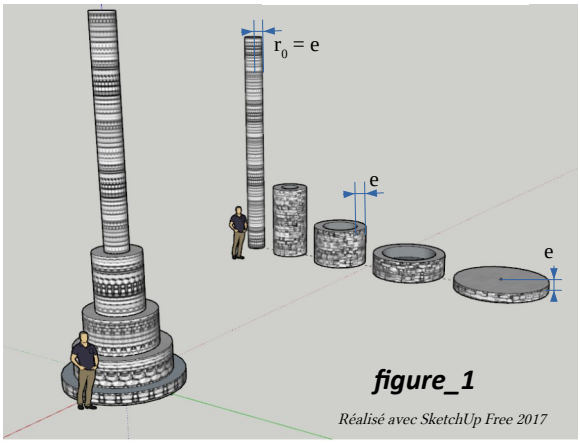


figure_1

Réalisé avec SketchUp Free 2017

La conception de l'obélisque

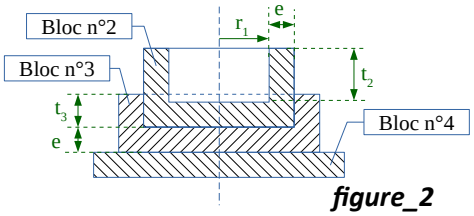


figure_2

On note r_0 le rayon du cylindre de la flèche et t_0 sa hauteur et on pose $r_0 = e$. Pour les calculs on considérera que le trou cylindrique présent dans le bloc n°*k* permettant d'accueillir le bloc n°*k-1* admet pour rayon r_{k-1} (même si en pratique il faudrait considérer l'existence d'un jeu facilitant l'emboîtement), on note t_k la profondeur de ce trou et r_k le rayon externe du bloc n°k.

1. En respectant les contraintes de l'énoncé justifier que r_k le rayon du bloc n°*k* est l'un des termes d'une suite arithmétique dont on précisera le premier terme et la raison.
2. Exprimer V_0 le volume du bloc n°0 en fonction de t_0 et de *e* .
3. Montrer que pour tout $k \in \{1\,;\,2\,;\,3\,;\,\ldots\,;\,n-1\}$ l'on peut écrire : $V_k = \pi\, e^2\left((2k+1)\cdot t_k + e\cdot(k+1)^2\right)$.

La densité massique volumique des blocs étant considérée constante (et vaut 2,5 tonnes par mètre cube de béton armé), l'égalité des masses se traduit alors par une égalité de volumes.

4. Déduire des questions précédentes l'expression de t_k en fonction de t_0 et de *e*.
5. Rappeler dans l'énoncé la phrase qui permet d'affirmer que $t_{n-1} = 0$. En déduire que $t_0 = e\,n^2$.
6. Compléter le programme Python suivant afin qu'il réponde aux objectifs annoncés.

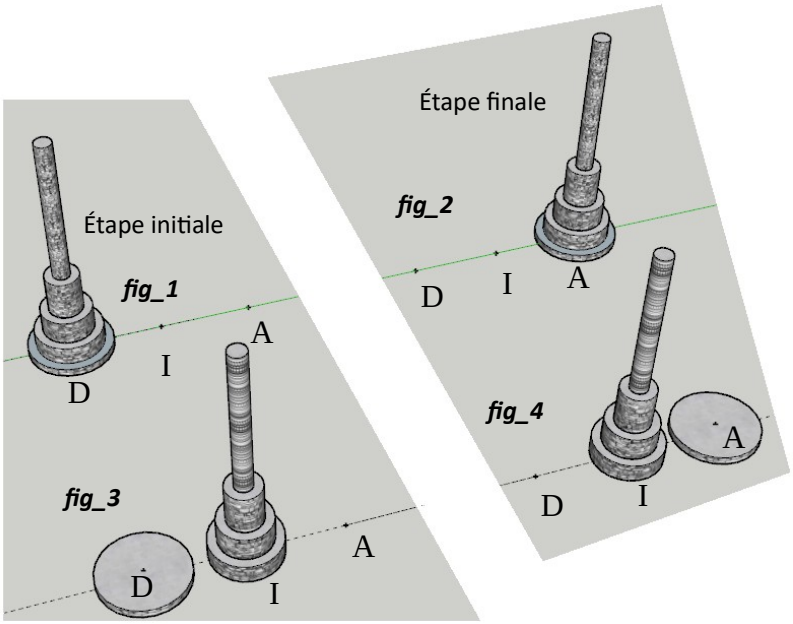
```
from math import pi

def Hanoy(e, n):
    """Renvoie la hauteur totale en mètre de l'obélisque Hanoyen (H),
    le volume en m3 de chaque bloc (V), son poids estimé en Tonne (M),
    il renvoie également deux listes: l'une contenant les rayons
    externes des blocs (Lr) et l'autre la profondeur des trous (Lt)"""
    t0 = e * n**2
    Lr = [(k + 1)*e for k in range(n)]
    Lt = [ t0 if k==0 else (t0 - e*(k + 1)**2)/(2*k + 1) for k in range(n) ]
    V = ...
    M = ...
    H = ...
    return H, V, M, Lr, Lt
```

7. Vérifier que chacun des cinq blocs de l'*Hanoyen* donné en exemple pèse un peu plus de 4,31 tonnes et que la hauteur totale de l'édifice est précisément de 8,12 mètres. Y-a-t-il d'autres couples (e,n) qui conduisent à des hauteurs de blocs s'écrivant en nombre décimal ? Si oui, donner d'autres exemples.

Le déménagement d'un obélisque *Hanoyen*

On souhaite déplacer un obélisque « *Hanoy28_5* » déjà monté pour lequel chacun des 5 blocs le constituant (où e=28cm) correspond au poids maximal autorisé pour son déplacement en toute sécurité d'un emplacement de départ (D) à un emplacement d'arrivée (A) par une grue. Malheureusement, une seule zone de stockage intermédiaire (I) n'est disponible et la surface restreinte de cette dernière ne permet au mieux qu'un empilement de blocs si bien qu'en minimisant les risques on se retrouve dans la configuration des « tours de Hanoï » dont on énonce les règles :

- On ne peut déplacer qu'un seul bloc à la fois.
- On ne dispose que d'une zone intermédiaire (notée I).
- Il est n'est pas accepté de mettre un plus large bloc par dessus un plus étroit.



Ainsi l'objectif est : partant de la configuration donnée en ***fig_1*** de parvenir à celle donnée en ***fig_2*** en un nombre minimal d'étapes.

L'idée de l'algorithme est de montrer qu'il existe une relation de récurrence. Si l'on est capable de déplacer un « sous-obélisque » de 4 blocs d'un emplacement à un autre alors l'opération globale (du déplacement des 5 blocs) devrait pouvoir s'effectuer sans difficulté. En effet, parvenir en ***fig_3*** signifie qu'on a réussi à déplacer le sous-obélisque de 4 blocs (de D à I), on déplace ensuite le bloc n°4 à sa position d'arrivée (***fig_4***) et enfin, puisqu'on a montré qu'on sait déplacer un sous-obélisque de 4 blocs, on réitère l'opération pour le positionner du point I au point A en se retrouvant à l'étape finale (***fig_2***). De façon générale pour déplacer sans difficulté un empilement de n blocs il faut être capable de savoir en déplacer un de n-1 blocs. On pressent donc bien que si l'on est capable d'en déplacer un minimum, un par exemple, de fil en aiguille (ou plutôt de bloc en bloc) on finira par en déplacer la totalité !

Soit $\left(d_n\right)_{n\in\mathbb{N}^*}$ la suite donnant le nombre de déplacements nécessaires pour déplacer un obélisque de n blocs.

En s'appuyant sur l'énoncé justifier que $\begin{cases} d_1=1 \\ d_n=2d_{n-1}+1 \end{cases}$.

1. Montrer par récurrence que l'on a $d_n=2^n-1$ en déduire la nature de la croissance de cette suite.
2. En supposant qu'il faille huit minutes pour effectuer le déplacement élémentaire d'un bloc, combien de temps faudrait-il pour déplacer l'*Hanoy28_5* ?

On donne en annexe le programme python permettant de résoudre le jeu des tours de Hanoï ainsi qu'un exemple de sa sortie.

3. Préciser comment sont codées les blocs et surligner les étapes sur la sortie du programme qui correspondent aux quatre configurations de l'énoncé (correspondants aux ***fig_1***, ***fig_2***, ***fig_3*** et ***fig_4***).

Annexe

Code python :

```
def Hanoi(n, L, d, a, i):
    """Résout le problème de la tour de Hanoï à n blocs
    L est une liste contenant les blocs présents sur chaque
    emplacement. d, a, i sont des index ayant pour valeur par défaut:
    d(départ=0), a(arrivée=2) et i(intermédiaire=1) """
    if n == 1:
        L[a].append(L[d].pop(len(L[d])-1))
        print(L)
        return
    Hanoi(n-1, L, d, i, a)
    L[a].append(L[d].pop(len(L[d])-1))# déplace le bloc de d vers a
    print(L)
    Hanoi(n-1, L, i, a, d)

n = 5 # choix du nombre de blocs
L = [list(range(n-1, -1, -1)),[],[]] # création de la liste des blocs
print(L)
Hanoi(n, L, 0, 2, 1)
```

Sortie du programme :

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