

Algorithmique Avancée

Année Académique 2021-2022

**Rendu de projet**

Soufian OUALLA

Lounes BEHLOUL

**SOMMAIRE**

1. [METHODE GLOUTONNE 4](#_bookmark2)
2. [RECURRENCE ET RECURSIVITE NAÏVE 5](#_bookmark3)
3. [PROGRAMMATION DYNAMIQUE 5](#_bookmark6)
   1. [APPROCHE TOP DOWN 5](#_bookmark7)
   2. [APPROCHE BOTTOM UP 5](#_bookmark8)
4. [APPLICATIONS 6](#_bookmark9)

# Méthode gloutonne

Concernant la méthode gloutonne, cela correspond au fichier **greedy.py**, on va vous expliquer son fonctionnement. Tout d’abord on récupère la liste de boîtes et on crée une liste avec les 3 rotations possibles d’une boîte. On se retrouve donc avec une liste 3 fois plus grande que celle d’origine. Ensuite on trie les différentes boîtes par taille d’aire en ordre décroissant afin d’avoir les boîtes avec la plus grande aire en premier (et donc celle qui sont le plus susceptibles d’être en bas de la pile). On ajoute la première boîte avec la plus grande aire à la pile. Et ensuite on parcourt toutes les boîtes et si la largeur et la profondeur sont inférieurs à ceux de la dernière boîte actuelle de la pile : on ajoute cette nouvelle boîte à la pile puis on continue ainsi.

Et donc cette solution n’est pas optimale car l’algorithme traite les cas dans l’ordre croissant d’aire mais il se peut que juste après la nouvelle boîte ajoutée, se trouve une boîte dont l’aire est plus petite et qui ne peut pas être ajouté mais qui possédait une très grande hauteur.

Le cas donné en exemple est un contre-exemple pour cet algorithme en effet au lieu de trouver 39 avec cette répartition [(7,2,5), (7,3,6), (20,5,10), (5,10,20)], on trouve 38 avec celle-là [(7,2,5), (6,3,7), (20,5,10), (5,10,20)]. Cela illustre bien le problème exposé juste avant.

# Récurrence et récursivité naïve

L’algorithme en récursive correspond au fichier **recursive.py**. Pour cet algorithme on commence de la même manière que pour le glouton, c’est-à-dire que l’on va tout d’abord créer une liste avec toutes les rotations possibles puis les trier selon leur aire par ordre décroissant. Mais cette fois-ci, nous procédons autrement, nous avons 𝐻[𝑖] la hauteur maximale de l’empilement comportant la 𝑖-ème boîte à son sommet. Puis nous allons crée une liste avec les différentes hauteurs de toutes les boîtes. Et ensuite pour chaque boîte nous avons 2 possibilités : soit on peut la placer dans l’une des boîtes si elle est plus petite que l’une de ces boîtes, soit on commence une nouvelle pile avec cette boîte. Et donc à la fin, on renvoie la plus grande hauteur 𝐻[𝑖].

La complexité de cet algorithme est O(n²) avec n le nombre total de boîtes

# Programmation dynamique

## Approche Top Down

## Approche Bottom Up

# Applications