Problème du remplissage d'une structure avec de l'eau

Victor Lavairye

January 2020

1 Description du problème

On considère une structure composée de plusieurs murs de hauteurs variables. On va modéliser cette structure sous la forme d'une liste d'entiers naturels, la valeur de l'entier correspond à la hauteur du mur et l'indice de celui-ci la position du mur sur la grille. Par exemple, la liste [0, 3, 0, 1, 0, 2, 4, 0, 2, 0] correspond à la structure suivante :

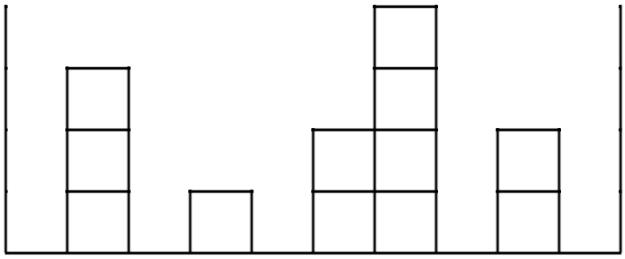

Figure 1: Exemple de figure considérée

On cherche à connaître le volume total d'eau que peut contenir n'importe quelle structure.

2 Résolution du problème

Une méthode de résolution du problème peut être de considérer des murs de hauteur infinie de part et d'autre de la structure remplis d'eau. Puis, on retire les murs et on regarde l'eau qui est resté contenue dans la structure.

L'idée dans la résolution est de diviser l'espace en plusieurs sous-espaces. Ces sous-espaces sont définis par les sous-structures dont le niveau d'eau est identiques entre les murs extérieurs.

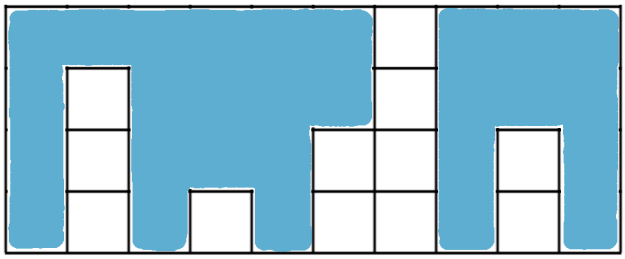

Figure 2: Structure avec murs de hauteur infinie

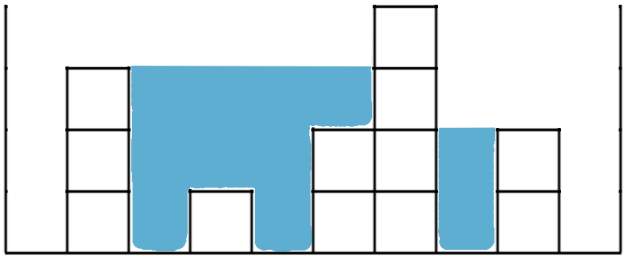

Figure 3: Structure après suppréssion des murs

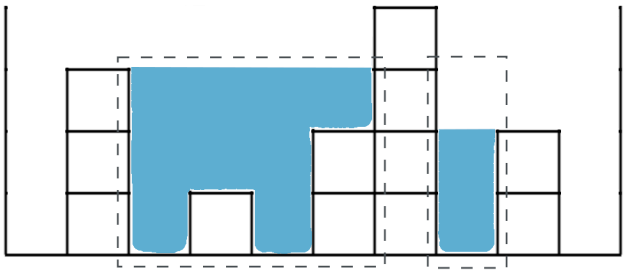

Figure 4: Découpage en sous-structures

3 Algorithme

Sous pyhton, on implemente cette méthode avec l'algorithme ci-dessous. On commencera par définir la fonction get_index_list(l, element) qui sera réutilisée ensuite.

```
def get_index_list(l, element):
index_list = []
for i in range(len(l)):
    if l[i] == element:
        index_list.append(i)
return index_list

def get_volume(construction_list):
construction_list_length  = len(construction_list)
if construction_list_length <= 2:
```

```
    volume = 0
    return volume
if construction_list_length == 3:
    volume = max(0, min(construction_list[0], construction_list[2])
                - construction_list[2])
    return volume
else:
    reverse_sorted_list = sorted(construction_list, reverse=True)
    highest_wall = reverse_sorted_list[0]
    highest_wall_index = construction_list.index(highest_wall)
    second_highest_wall = reverse_sorted_list[1]
    second_highest_wall_index_list = get_index_list(construction_list,
                second_highest_wall)
    second_highest_wall_index_distance_to_highest_wall_index =
                [abs(second_highest_wall_index_list[i]
                - highest_wall_index) for i in range(
                len(second_highest_wall_index_list))]
        second_highest_wall_index = second_highest_wall_index_list[
                second_highest_wall_index_distance_to_highest_wall_index.
                index(max(second_highest_wall_index_distance_to_highest
                    _wall_index))]
    max_index = max(highest_wall_index, second_highest_wall_index)
    min_index = min(highest_wall_index, second_highest_wall_index)

    #Construction of subset lists from construction_list
    left_construction_list = construction_list[0: min_index + 1]
    middle_construction_list = construction_list[min_index: max_index + 1]
    right_construction_list = construction_list[max_index: len(construction_list)]

    #Computation of the water volume in the middle_construction subset
    if max_index + 1 - min_index > 2:
        volume = second_highest_wall * (max_index - 1 - min_index)
                - sum(middle_construction_list[1 : max_index - 1])
        return volume + get_volume(left_construction_list)
                + get_volume(right_construction_list)
    else:
        return get_volume(left_construction_list)
                + get_volume(right_construction_list)
```

4 Résultats

4.1 Test de l'algorithme pour différentes structures

4.1.1 Exemples standards

Pour les exemples qui vont suivre, on laissera au lecteur le soin de vérifier la validité des résultats.
remarque : Tout les temps d'éxécution de la fonction a été obtenu sur la même machine avec le module time de python, en prenant la moyenne des temps obtenu après 100 simulations.

- structure_1 = [0, 3, 0, 1, 0, 2, 4, 0, 2, 0]

taille de la liste	temps d'éxécution (s)	volume
10	0.000000	11

- structure_2 = [0, 3, 0, 1, 0, 2, 4, 0, 2, 0, 0, 3, 0, 1, 0, 2, 4, 0, 2, 0]

taille de la liste	temps d'éxécution (s)	volume
20	0.000000	35

- structure_2 = [0, 3, 0, 1, 0, 2, 4, 0, 2, 0, 0, 3, 0, 1, 0, 2, 4, 0, 2, 0, 0, 3, 0, 1, 0, 2, 4, 0, 2, 0]

taille de la liste	temps d'éxécution (s)	volume
30	0.000000	63

4.1.2 Structures pathologiques

- structure_pathologique_1 = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

taille de la liste	temps d'éxécution (s)	volume
10	0.000000	0

- structure_pathologique_2 = [1, 2, 3, 4, 5, 5, 4, 3, 2, 1]

taille de la liste	temps d'éxécution (s)	volume
10	0.000000	0

- structure_pathologique_3 = [0, 0, 0, 0, 4, 4, 0, 0, 0, 0]

taille de la liste	temps d'éxécution (s)	volume
10	0.000000	0

4.2 Influence des dimensions de la structures sur la vitesse de convergence

On va maintenant étudier l'influence des dimensions de la structures sur la vitesse de convergence de l'algorithme. Les listes considérées étant potentiellement très longues, on générera au hasard (*i.e* en tirant selon une loi uniforme des entiers entre 0 et h) des listes de longueurs N.

4.2.1 Formalisme de la méthode de test

- **Génération d'une liste aléatoire** : On génère une liste aléatoire de longueur N et de hauteur maximale h en simulant N fois une variable aléatoire suivant une loi uniforme $\mathcal{U}_{[0,h]}$.
  L'implémentation sous python est la suivante :

```
l = [rd.randint(0, h) for j in range(0,N)]
```

- **Condition de test** : Pour chaque valeur de N ou h testée, on fait tourner 10000 fois la fonction pour random_list(N, h).

```
h = 10
N = 100
total_time = 0
total_volume = 0
for i in range(0,10000):
    l = [rd.randint(0, h) for j in range(0,N)]
    time_1 = time.time()
    volume = get_volume(l)
    time_2 = time.time() - time_1
    total_time += time_2
    total_volume += volume
print('total_time = %f' %total_time, 'total_volume = %d' %total_volume)
print('average_time =%f' %(total_time/1000),
            'average_volume =%f' %(total_volume/1000))
```

4.2.2 Synthèse des résultats