WU Sébastien

DUDEK Tomasz

Structure de données avancées

TP1

1. Si on multiplie la taille par un facteur α>=1 on obtient la fonction potentielle suivante :

Ⲫ (i) = xni - yti

Avant extension :

xni - yti = ni

xni - yni = ni

x = 1 + y

Aprés extension :

xni - yti = 0

xni - yαni = 0

x= yα

1 + y = yα

y = 1 / (α-1)

x = α / (α-1)

=> O(i) = ( α / (α-1) ) ni - (1 / (α-1))ti

α = 2 => O(i) = 2ni -ti

1. Coût amorti d’insérer-table :

Ci = ci + O(i) - O(i-1)

Cas 1 : pas d’extension

Ci = 1 + ( (α / (α-1))ni - (1 / α-1)ti ) - ( (α / α-1)ni-1 - (1 / (α-1))ti-1 )

= 1 + (α / (α-1))ni - (1 / α-1)ti - (α / (α-1))ni + (α / (α-1)) + (1 / (α-1))ti

= 1 + (α / (α-1)) = θ(α)

Cas 2 : extension

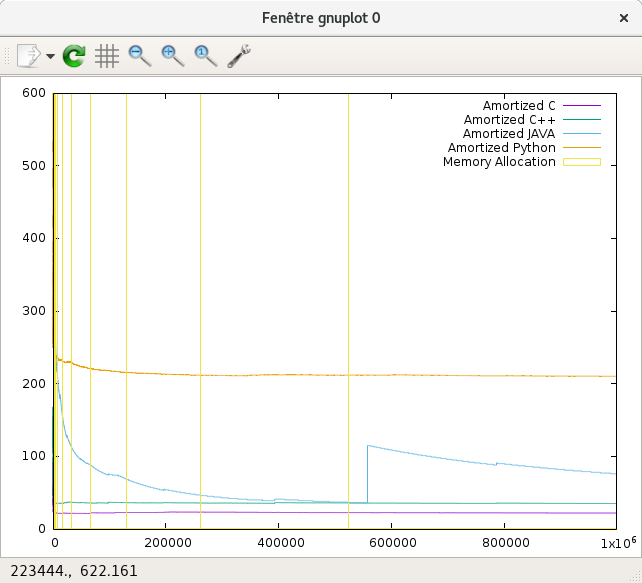
Ci = ci + θ(i) - θ(i-1) Ci = 1 + (α / (α -1))

= θ(α) Composé de n éléments de coûts θ(α)

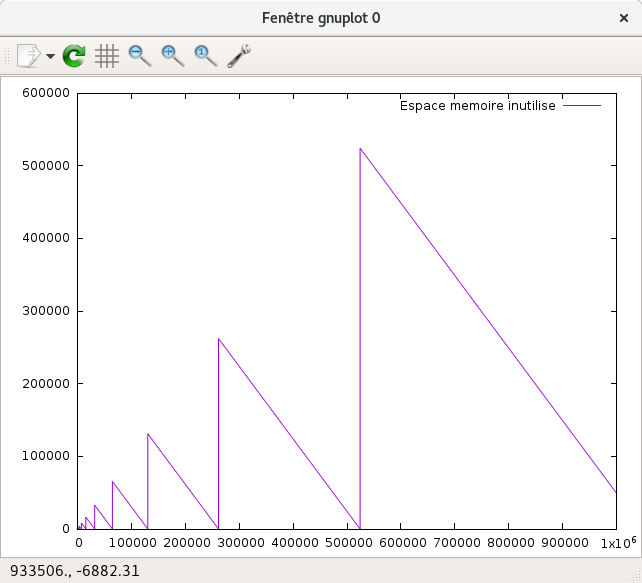
= θ(1) = n θ(α) = θ(αn) = θ(n) avec α constant

1. a)Lors de l’exécution du code le plot, dont la complexité est linéaire, met du temps à être exécutée à cause de l’écriture dûe à l’exécution des instructions.

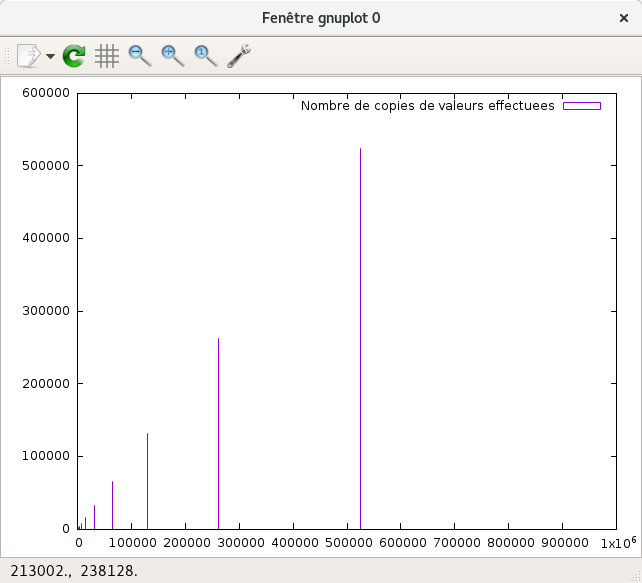
b) coût amorti augmente lorsqu’on doit allouer de la mémoire pour écrire dans le tableau (augmentation de la taille du tableau)



On remarque que le coût amorti est constant lors de l’allocation de la mémoire sauf pour le JAVA. Celui-ci diminue jusqu’à un certain coût puis augmente jusqu’à un pic et redescend au même coût. Cela forme un signal répété.

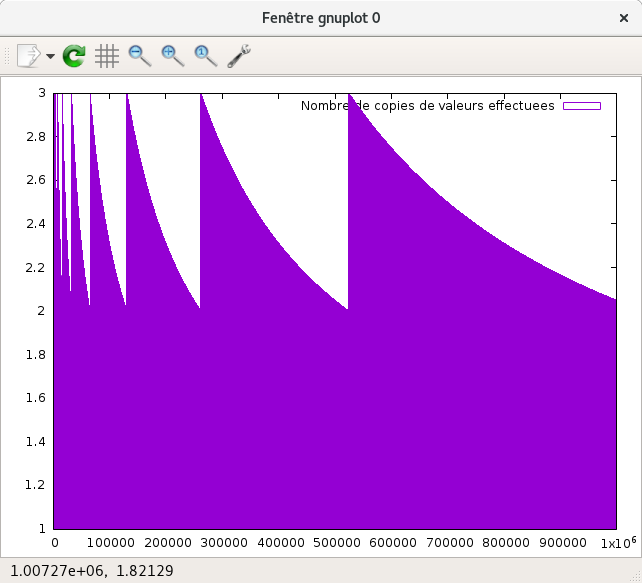


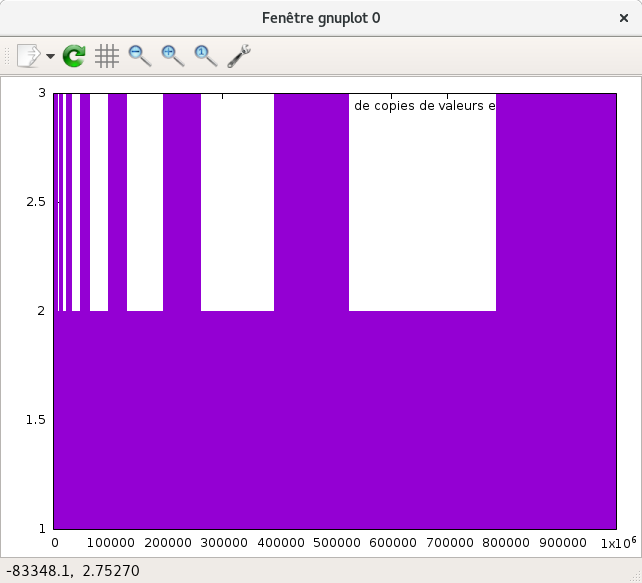
On remarque sur le schéma ci-dessous que a chaque fois qu’on allouée de la mémoire supplémentaire, la taille de la mémoire non utilisée est doublée. Ensuite elle décroît progressivement jusqu à l’allocation mémoire suivante.



On observe plusieurs pics de nombre de copies de valeurs effectuées. Ce pic semble doubler à chaque fois. Cela correspond au moment où le tableau est rempli et doit doubler en taille.

c)En C, on observe que le coût amorti est linéaire et que le nombre de copies varient entre 2 et 3 d’après le graphe ci-dessous. De plus, on constate le nombre de fois où les copies sont effectuées mais le temps fluctue à chaque opération.



d) Le temps par copie effectuées n’est pas le même en fonction du langage mais le nombre de copies et le temps où est effectué les copies sont toujours les mêmes.

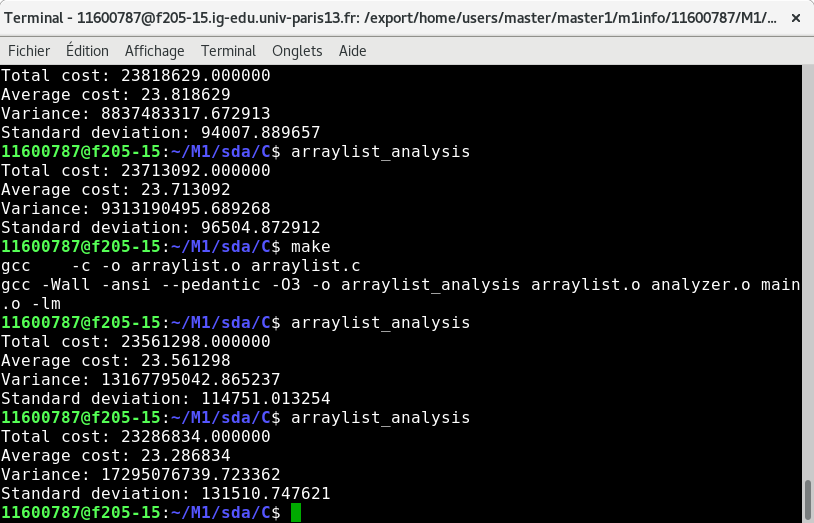
e)Certains langages sont plus rapides que d’autres dans cette expérience car ils sont compilés de différente manières :

* JAVA passe par une machine virtuelle;
* C/ C++ sont compilés en langage machine;
* Python exécute chaque instruction une par une.

f) Chaque pic d’allocation mémoire correspond à une copie effectué. Cela peut poser problème si par exemple nous venons à supprimer et insérer lorsque le tableau est rempli. Cela reviendrait à supprimer la moitié du tableau puis augmenter la taille du tableau à chaque instruction.

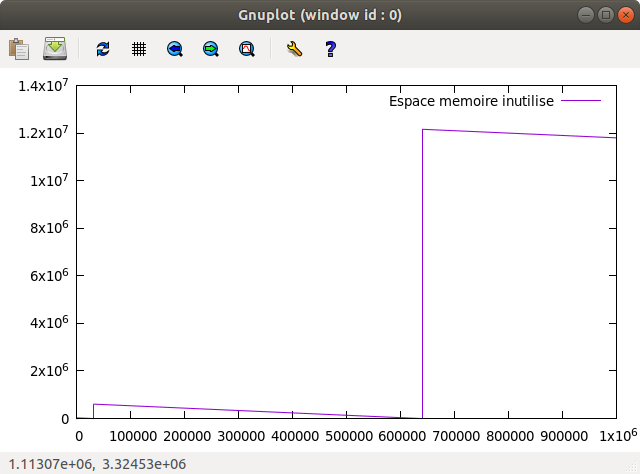
4) Lorsqu’on est à :

* 75% on a une moyenne à 23.6 - 23.8
* 100% on a une moyenne à 23,2 - 23,5

on observe une légère baisse lorsqu’on augmente le tableau à 100% remplit.

5)Si on modifie le facteur α nous obtenons un rapport entre le coût en temps et le coût en espace qui est multiplié par α.

Ici nous avons multiplié α par 10 ( 2-> 20)



6)On observe que le nombre de copies effectuées diminue si on change 2 par 2 + sqrt(2)