Réalisé par :

LE Minh Hao

KEITA Aïssata

RAPPORT STRUCTURE DE DONNEES AVANCEE

TP2 TABLE DYNAMIQUE : EXTENSION ET CONTRACTION DE LA TABLE

Table des matières

[**2. Insertion et suppression** 3](#_Toc28602307)

[**3. Enregistrer les couts réels et amortis des opérations** 8](#_Toc28602308)

[**4. Tester des valeurs différentes de p {0.1, 0.2, 0.3, . . .}** 9](#_Toc28602309)

# **1. Nouvelle fonction potentiel**

À l’aide de la nouvelle fonction potentiel, on montre que le cout amorti d’une opération Supprimer(T, x) qui fait appel à cette stratégie est majoré par une constante.

Et la nouvelle fonction potentiele .

**Cas 1 : Pas de contraction**

**On sait que**

**Le cout amorti**

**Cas 2 : Contraction**

**Le cout amorti**

**On sait que**

**Le cout amorti**

# **2. Insertion et suppression**

Au lieu de n’effectuer que des insertions dans la table, modifier le code pour effectuer un mélange de n opérations d’insertions et de suppressions. Pour chacune des n itérations, choisir une insertion avec une probabilité p et une suppression avec une probabilité 1 − p. Si l’opération choisie est suppression alors que la table est vide on ne fait rien.

Dans TP2, on utilise la classe simple **ArrayList** et d’abord, on choisit la probabilité **p = 0.7** et modifie le code comme ci-dessous :

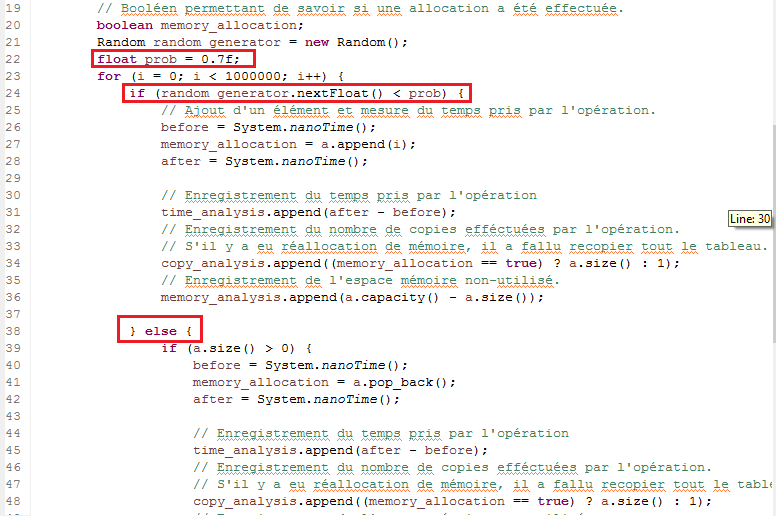


Figure 1. Le code avec p = 0.7 (70% insertions)

On va doubler la taille du tableau quand le tableau est plein et on va contracter le tableau en multipliant sa taille par 2/3 quand son facteur de remplissage tombe en dessous de 1/3 :

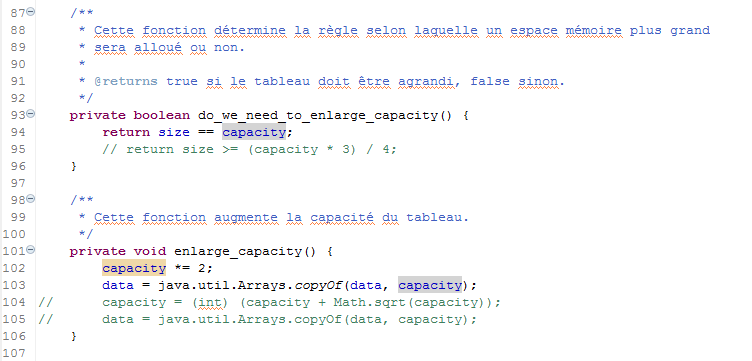


Figure 2. Doubler la taille du tableau quand le tableau est plein

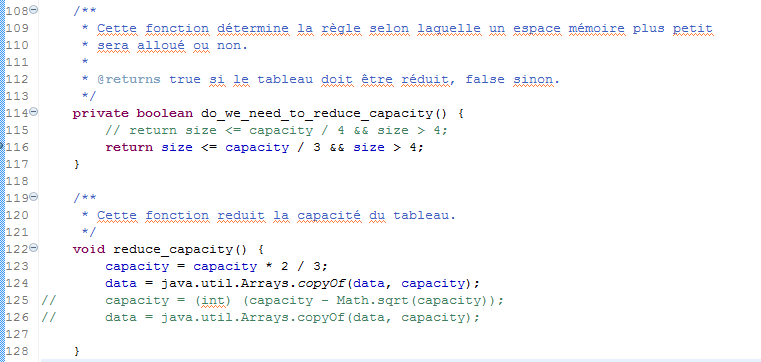


Figure 3. Contracter le tableau en multipliant sa taille par 2/3 quand son facteur de remplissage tombe en dessous de 1/3

Total cost : 8.7804487E7

﻿Average cost : 87.804487

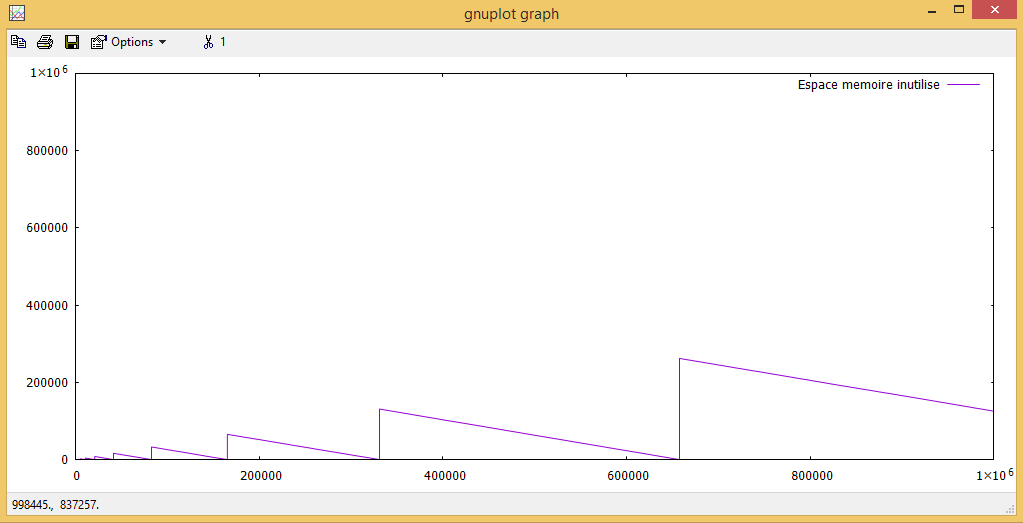


Figure 4. Espace mémoire inutilisé avec p = 0.7

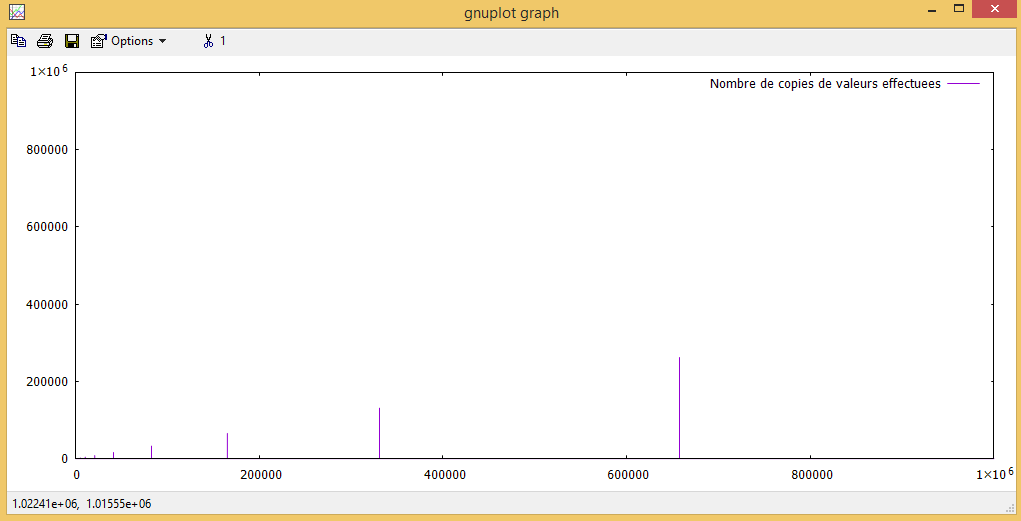


Figure 5. Nombre de copies avec p = 0.7

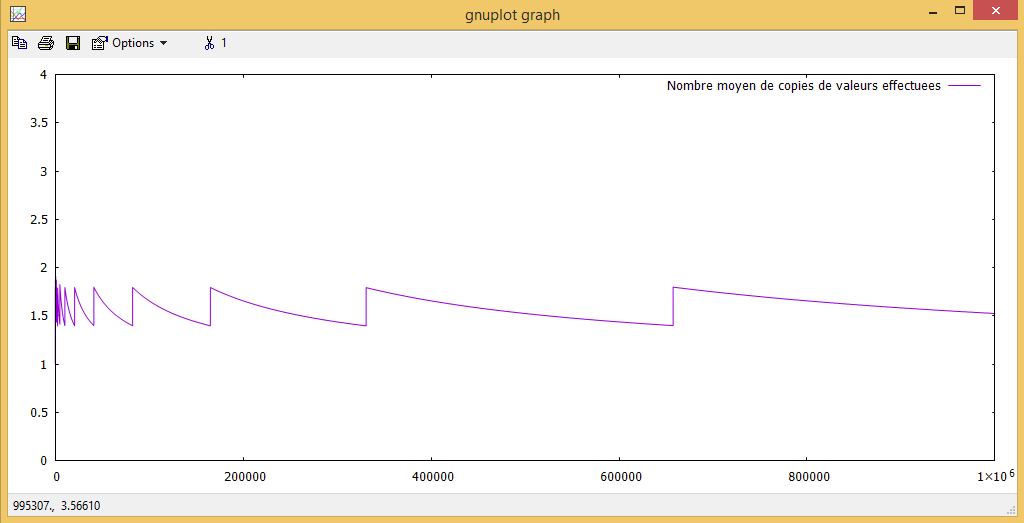


Figure 6. Nombre **moyen** de copies avec p = 0.7

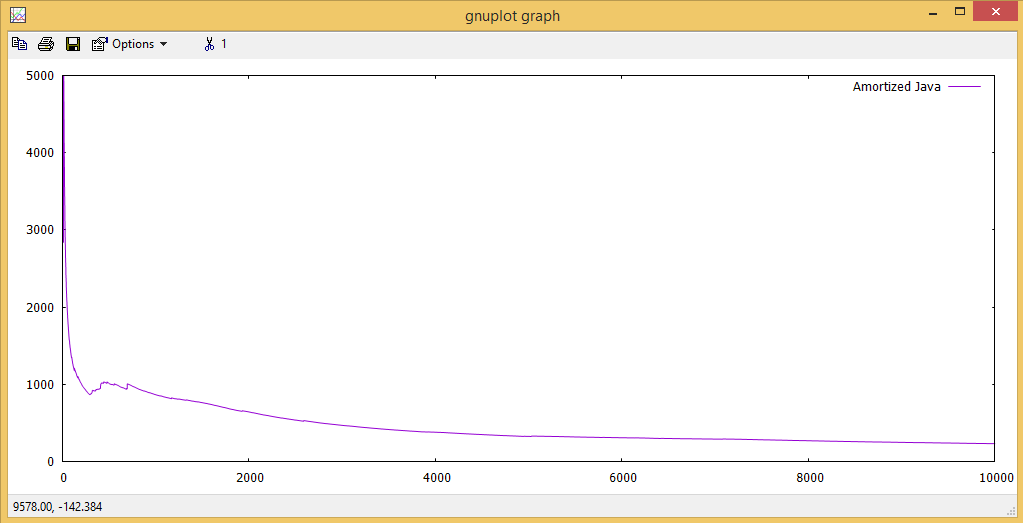
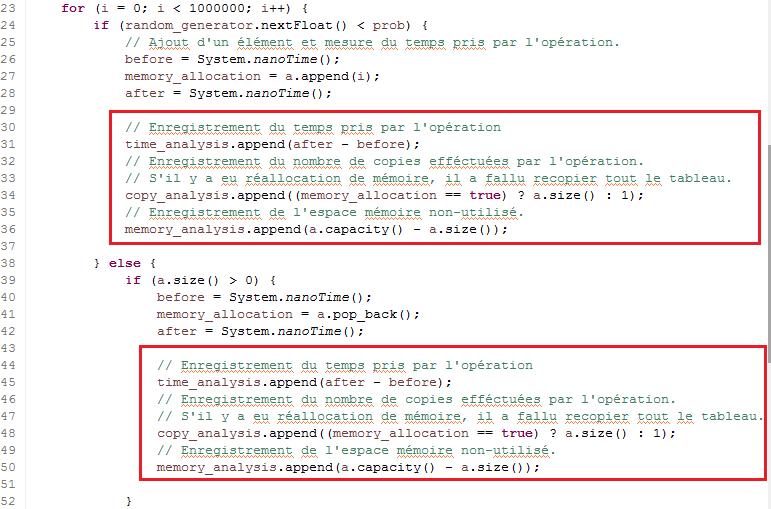


Figure 7. Le cout amorti avec p = 0.7

# **3. Enregistrer les couts réels et amortis des opérations**

Utiliser les outils développés pour le TP1 afin d’enregistrer les coûts réels et amortis des opérations, ainsi que l’espace mémoire non-utilisé.

On utilise la classe ArrayList et la classe Analyzer de TP1 pour d’enregistrer les chiffres.



# **4. Tester des valeurs différentes de p {0.1, 0.2, 0.3, . . .}**

Utiliser gnuplot pour afficher les coûts amortis et l’espace mémoire non-utilisé pour chacune de ces expériences. Que pensez-vous de la relation entre p, le coût en temps et le gaspillage de mémoire ?

Avec p = {0.1, 0.2, 0.3, 0.4}, le nombre moyen de copies est approximativement égale à 1. Parce que le nombre des suppressions est égale au le nombre d’insertion.

Par exemple : Avec p = 0.1, on a 10% opérations d’insertion (100 mille opérations d’insertion et 100 mille opérations effectuée de suppression dans 1 million opérations).

Avec p = 0.2, on a 20% opérations d’insertion (environs 200 mille opérations d’insertion et 200 mille opérations effectuée de suppression dans 1 million opérations).

Alors, la taille du tableau est plus petite comme 8, 16, 32.

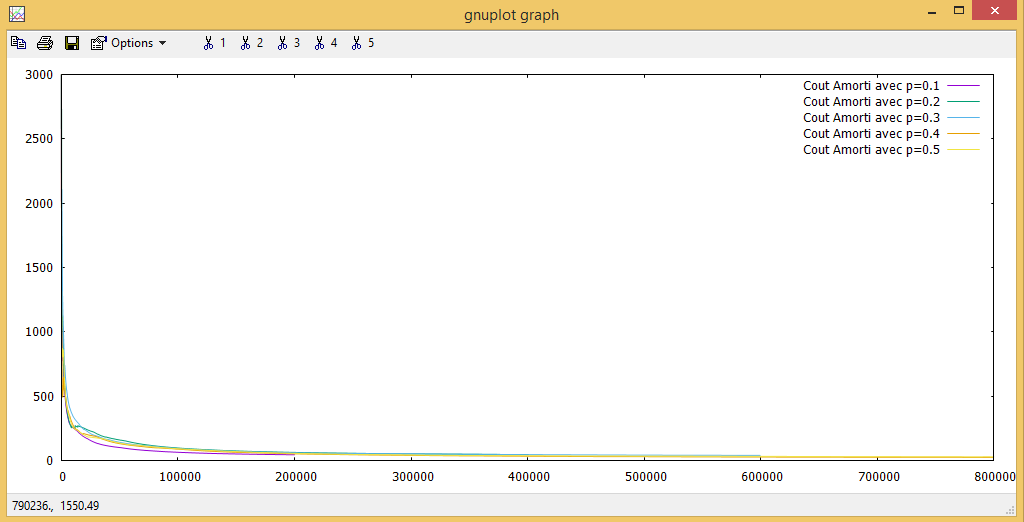


Figure 8. Cout Amorti avec p <= 0.5

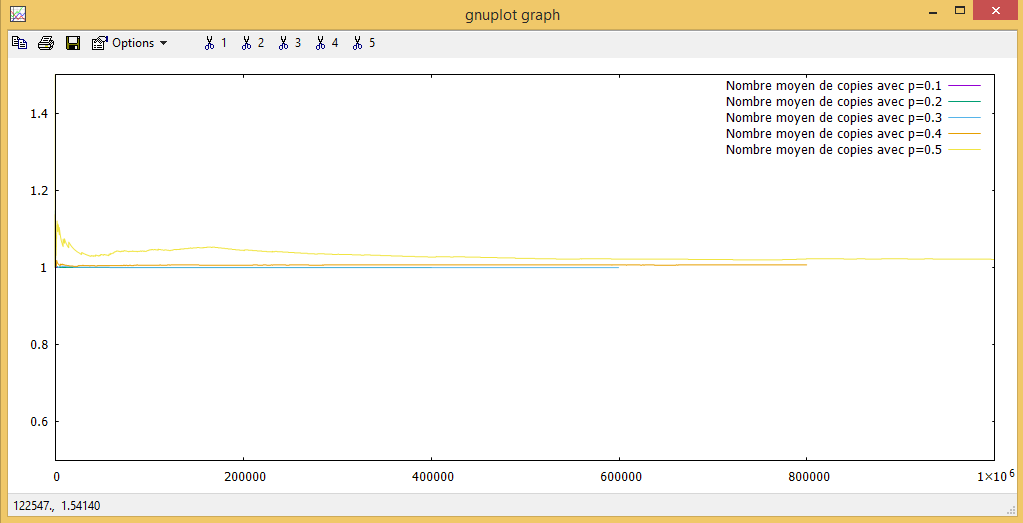


Figure 9. Nombre moyen de copies avec p <= 0.5

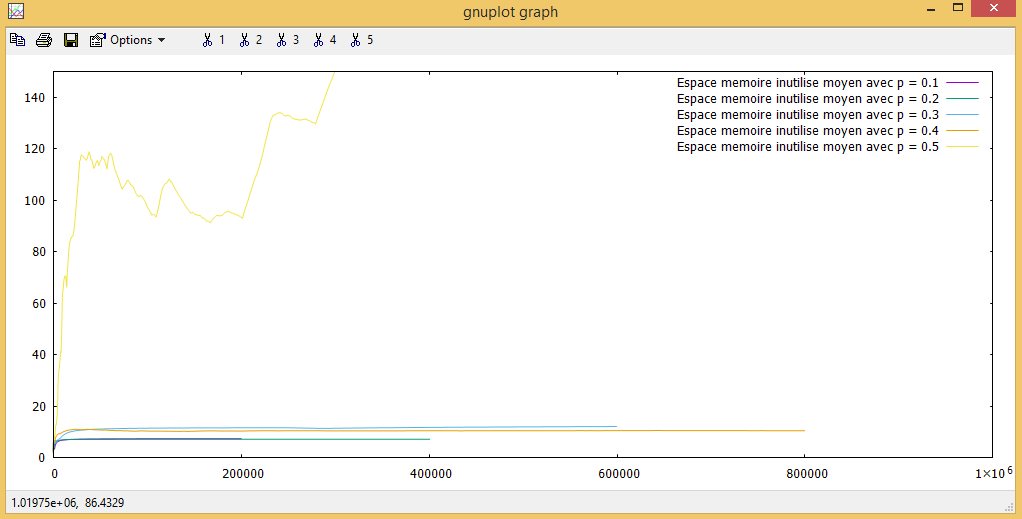


Figure 10. Espace memoire inutilise moyen avec p <= 0.5

Avec p = 0.5, l’espace mémoire inutilisé est également plus petit que le nombre d’opération (environs 400 dans 1 million d’opération).

De plus, le facteur multiplicatif est 2 et le facteur de contracter est 2/3. Alors il y a normalement des mémoires inutilisés après une contraction. Donc, la taille du tableau est augmentée au fur et à mesure avec p = 0.5

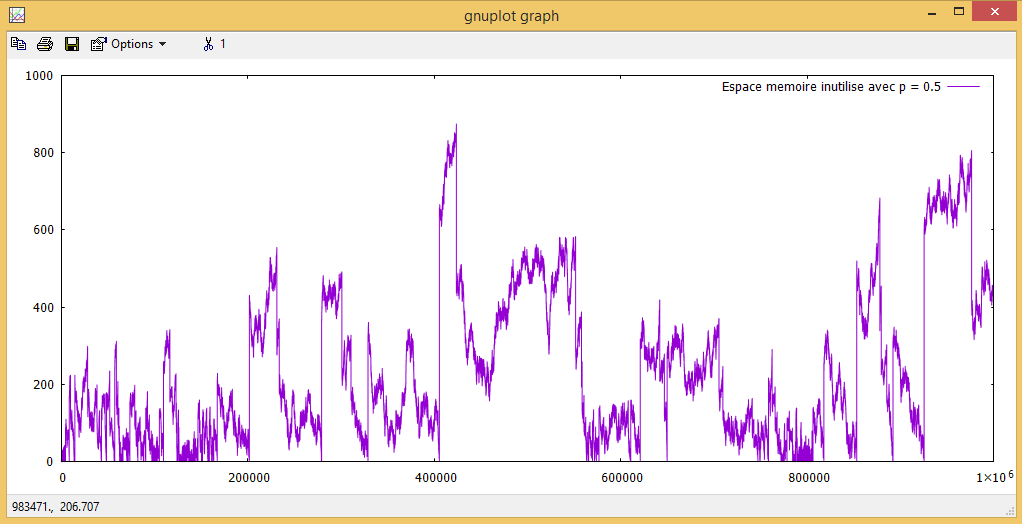


Figure 11. Espace mémoire inutilisé avec p = 0.5

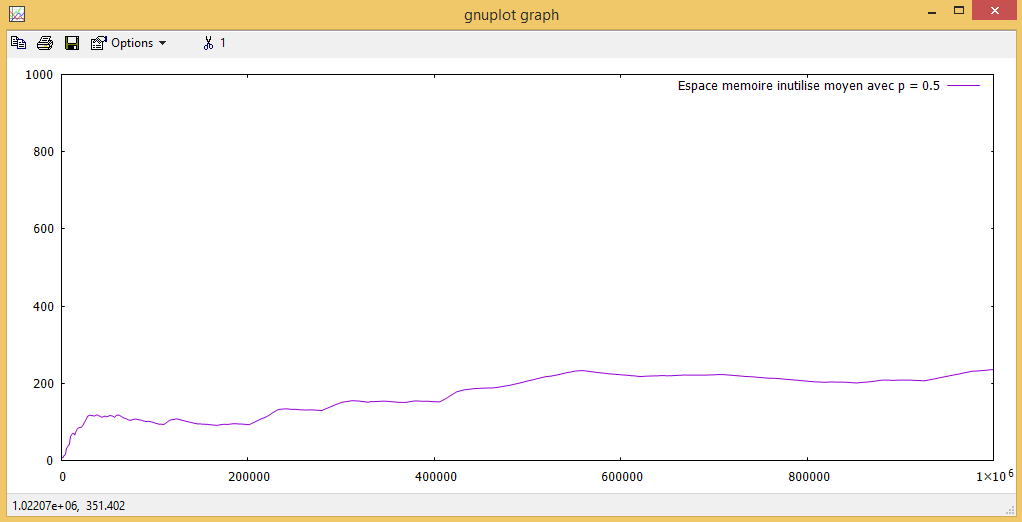
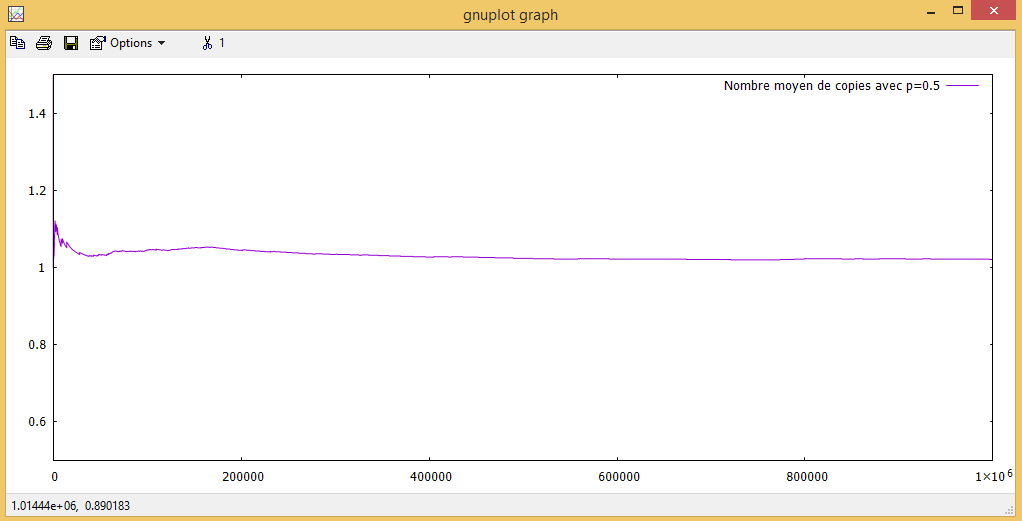
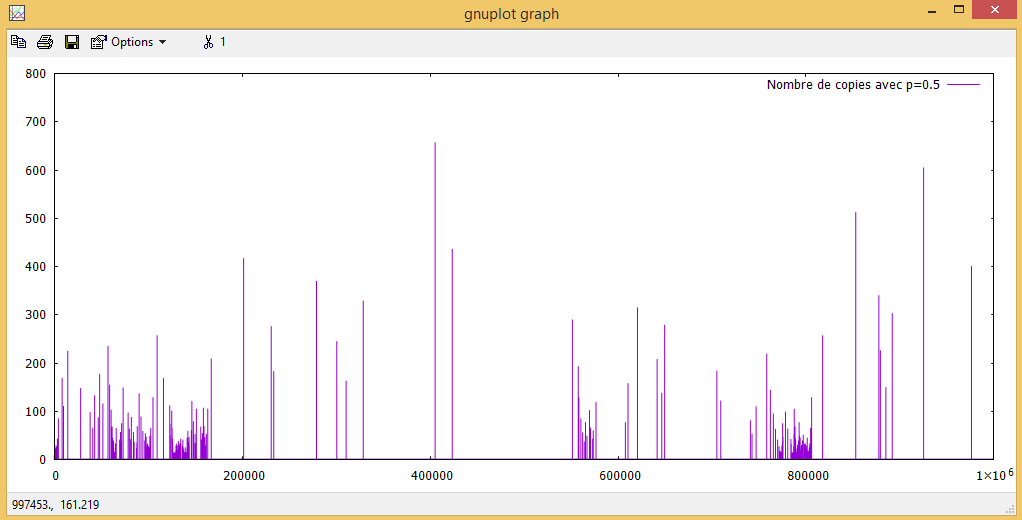


Figure 12. Espace mémoire inutilisé **moyen** avec p = 0.5

Avec p = 0.5, le nombre moyen de copies est supérieur à 1 et le plus grand nombre de copies égale à 700 (dans la taille 1400)





Avec p >= 0.6, le nombre d’insertion est plus grand que le nombre de suppression. Donc, le cout amorti ou le nombre de copies sont augmentés avec la probabilité. On peut clairement le voir dans la graphie Nombre moyen de copies.

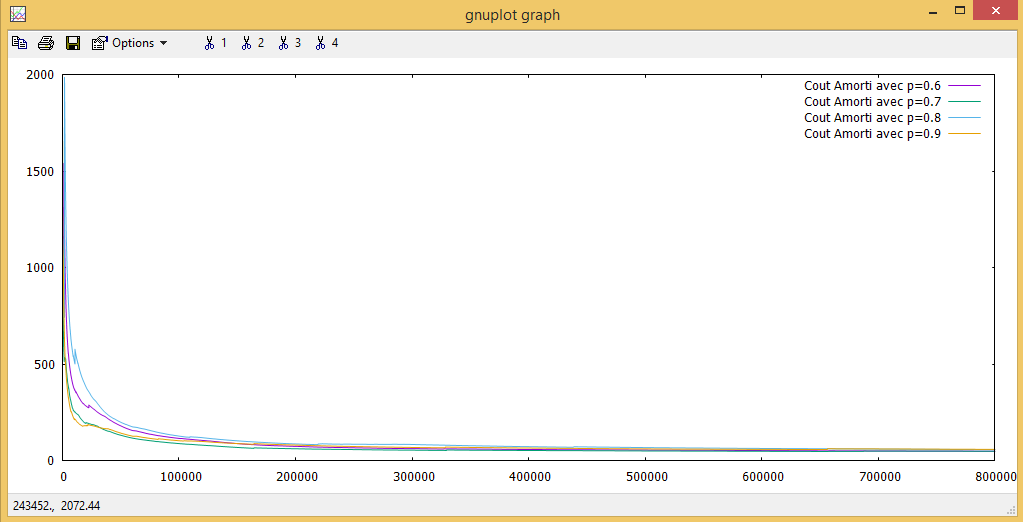


Figure 13. Cout amorti avec p >= 0.6

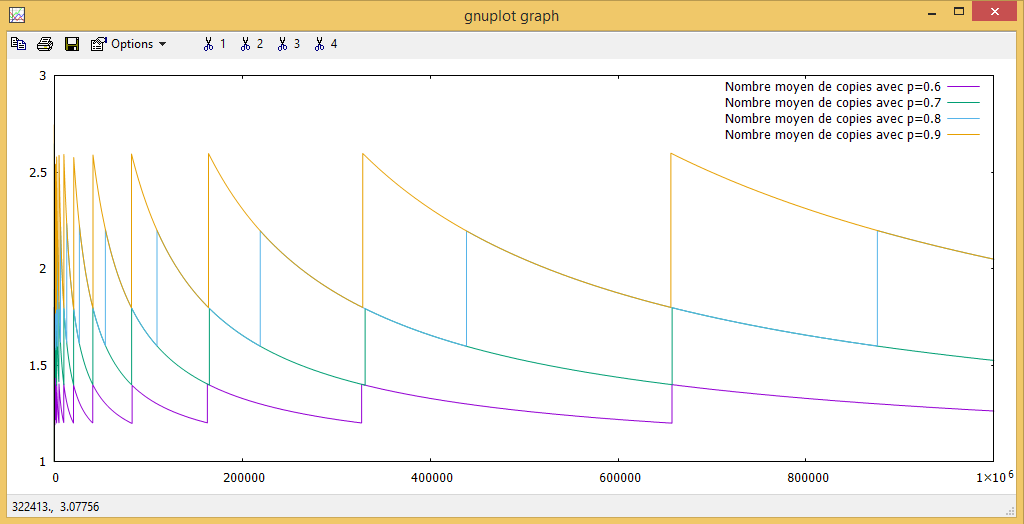


Figure 14. Nombre moyen de copies avec p >= 0.6

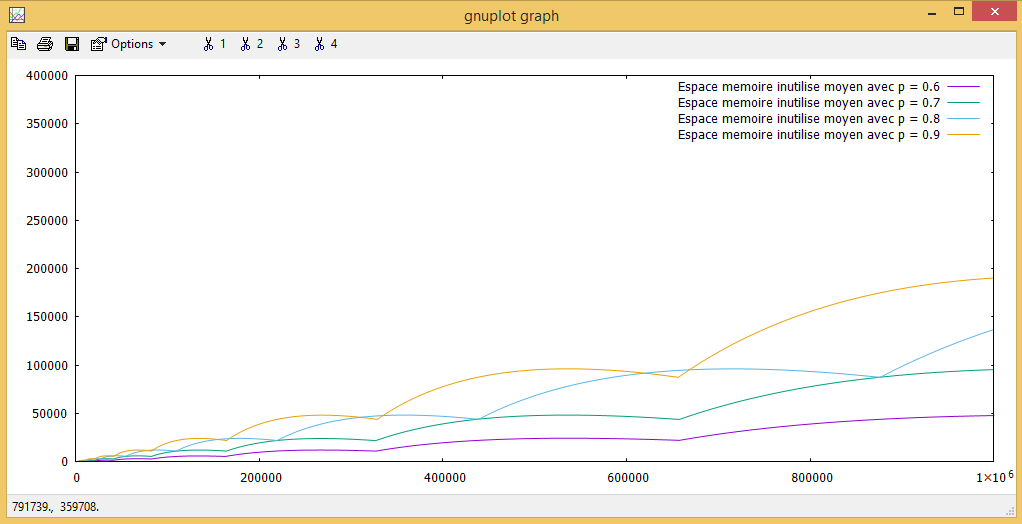


Figure 15. Espace mémoire inutilisé **moyen** avec p >= 0.6

**Dans cette stratégie, plus la probabilité p est élevée, plus le cout amorti est élevé et plus l’espace mémoire inutilisé est élevé.**

**Si on peut prédire la probabilité p (ou le nombre de suppression inefficace si le tableau est vide) dans le futur, on peut choisir la stratégie suffisante pour tenir la meilleure performance et éviter le gaspillage de mémoire.**

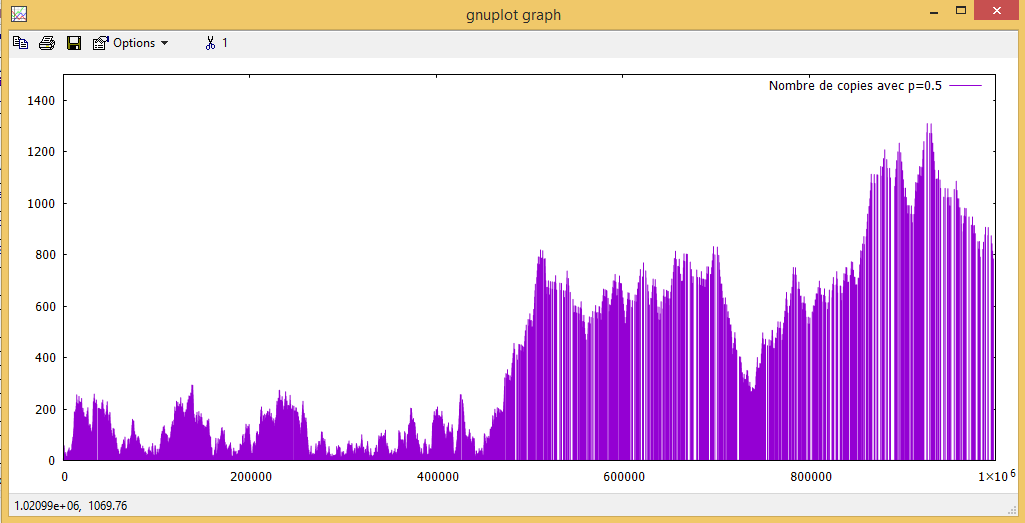
5. Choisir p = 0.5. Modifier la stratégie de redimensionnement de la table pour utiliser lors d’une extension, comme dans la dernière question du TP1 ; et lors d’une contraction. On déclenche une extension au moment de l’insertion quand le facteur de remplissage de la table, càd quand la table est pleine. Quand est-ce qu’il faut contracter la table ? Que pensez-vous de l’efficacité de cette stratégie ?

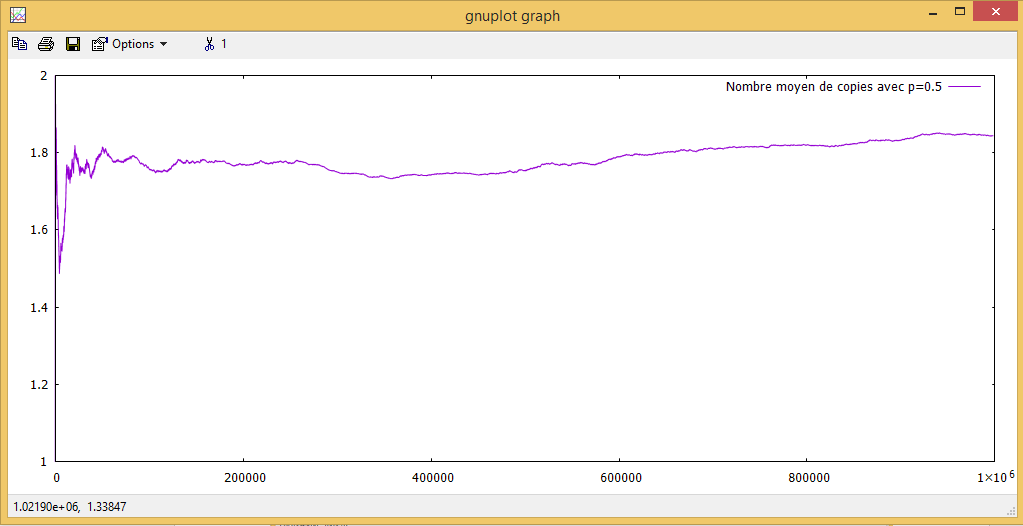
Avec ces deux facteurs de redimensionner, on peut contracter le tableau si. L’idée ici est la suivante : après une extension, le facteur de remplissage vaut. Donc, des éléments de la table devront être supprimés avant qu’une contraction n’ait lieu, puisque cette contraction ne se produit que si le facteur de remplissage descend en-dessous de. De même, après une contraction, la mémoire vide égale à (Note\*). Donc les insertions éventuelles doivent insérer d’éléments dans la table avant qu’une extension ait lieu, puisqu’elle ne se produit que lorsque le facteur de remplissage dépasse la valeur 1.

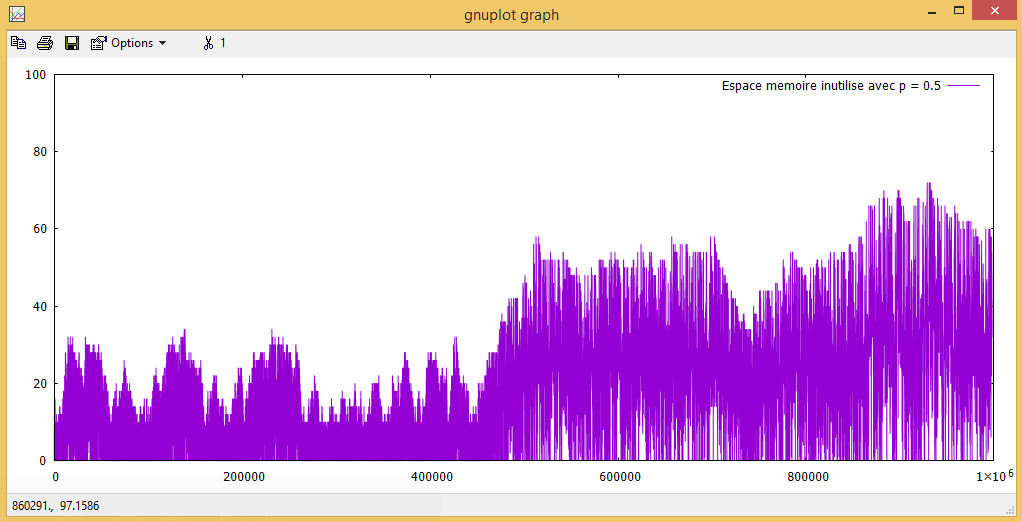
**(Note\*) : n’est pas trop différent de**

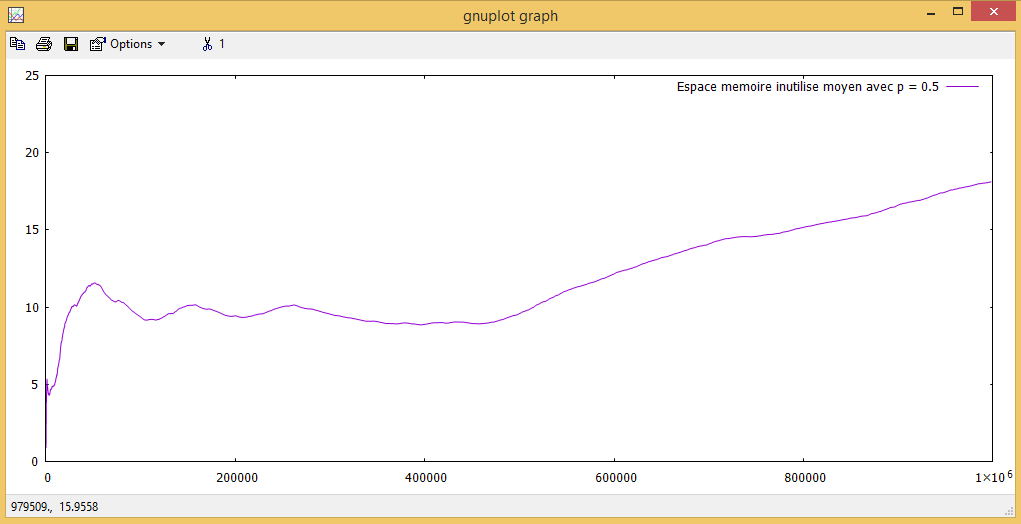
On peut comparer les résultats de cette stratégie à la stratégie précédente (même p = 0.5). L’espace mémoire inutilisé **moyen** de cette stratégie est plus petit que la stratégie précédente (18 et 225 dans Figure 12)

Mais le nombre moyen de copies de cette stratégie est plus grand que la stratégie précédente (1.8 et 1.02 avec 1 million d’opérations). De plus, le nombre moyen de copies de cette stratégie a la tendance d’augmenter avec le temps. Donc, si la taille du tableau est trop grand, le cout de copier est trop cher.



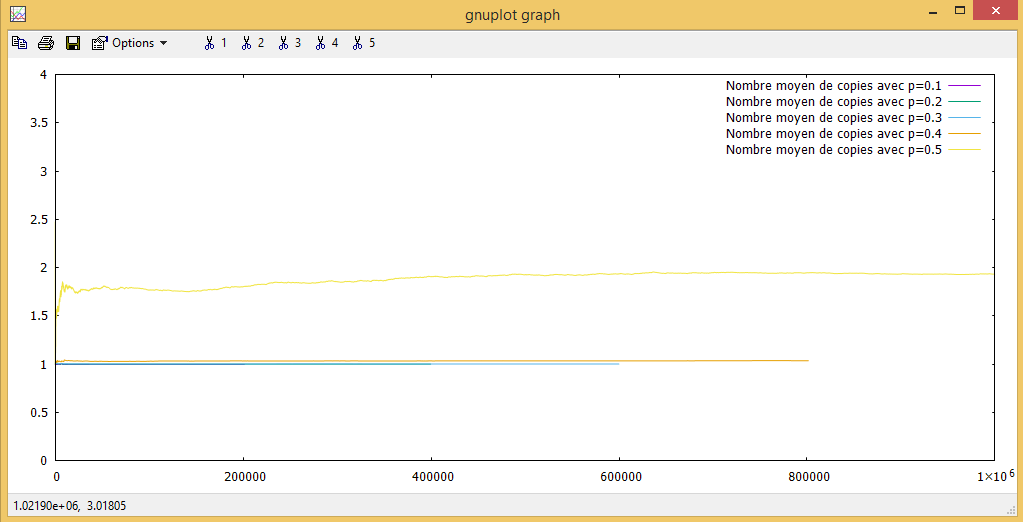




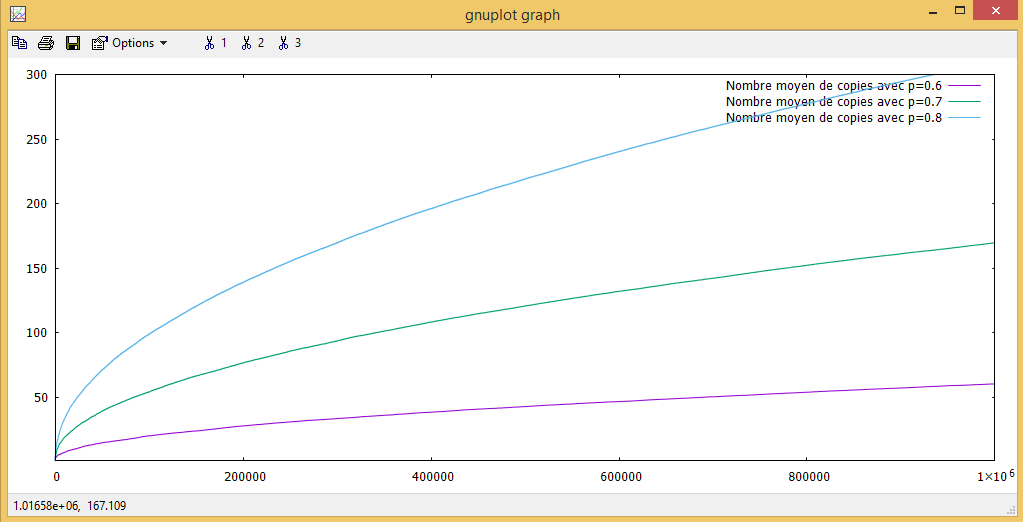


6. Tester à nouveau les différentes valeurs de p. Qu’en pensez-vous ? Quelle est la valeur de p pour laquelle cette stratégie semble mieux fonctionner ? Pourquoi ?

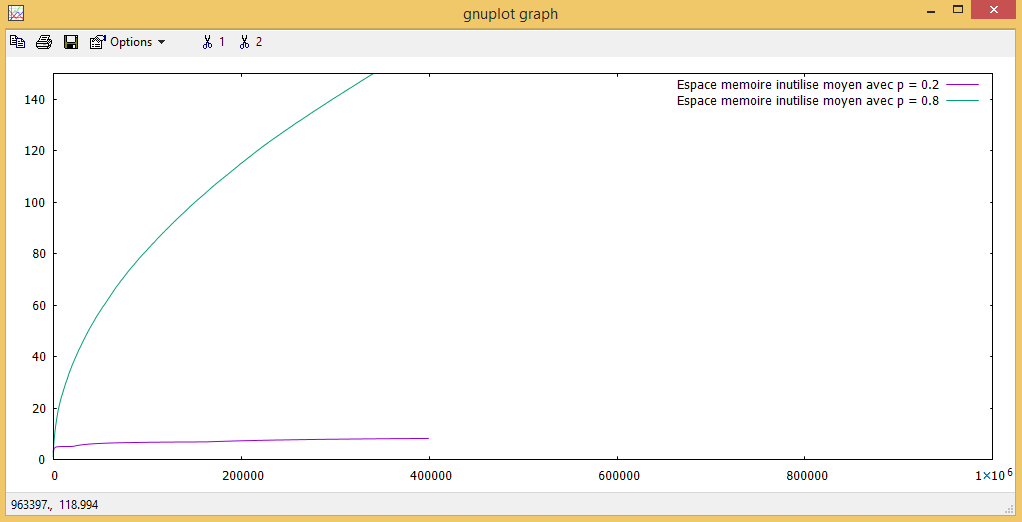
D’abord, on teste cette stratégie à nouveau les différentes valeurs de p. On pense que cette strategie est appropriée avec p < 0.5. C’est-à-dire, le nombre de suppression est plus que le nombre d’insertion. Ça rend la taille du tableau petite



Cette stratégie n’est pas appropriée avec la probabilité p > 0.5. Puisque, la grande taille rend le cout de copier trop cher avec le temps.



En plus, le facteur de multiplicatif n’est pas grand et le facteur de remplissage égal à 1, l’espace mémoire inutilisé est très grand pour éviter le gaspillage de mémoire.



Enfin, on peut clairement voir dans la graphie des couts amortis entre p = 0.2 et p = 0.8. Le cout en temps de p = 0.2 est stable avec le temps mais le cout en temps de p = 0.8 est grandi avec le temps.

