UNIVERSITÉ PARIS 13 | USPC Université Sorbonne Paris Cité

Structures de Données Avancées

Rapport TP2

Tables Dynamiques
Extension et contraction de la table

Réalisé par :
Aoudjehane Sarah
KOUACHI Abdeldjalil

1) Le coût amorti de supprimer(t,x)

On a :

$$\theta(i) = |2*nomi - taille|$$

On distingue deux cas:

Cas 1 : pas de contraction $\alpha = ni/ti > 1/3$

On a :

$$c_i\hat{} = c_i + \Phi(Di) - \Phi(Di-1)$$

$c_i\hat{} = 1 + |2*n_i - taille_i| - |2*n_{i-1} - taille_{i-1}|$

On sait que

$n_{i-1} = n_i + 1$

$t_{i-1} = ti$

$\alpha_{i-1} = n_i - 1/t_i - 1 > 1/3$

$n_i + 1/t_i > 1/3$

$t_i > 3(ni + 1)$ donc $t_i > 3n_i + 3$

$3n_i + 3 - t_i < 0$

Donc $2n_i - t_i < 0$

$ci\hat{} = 1 + (ti - 2n_i) - |2*(n_i + 1) - taille_i| = 1 + ti - 2n_i - (ti - 2(n_i + 1))$

$= 1 + ti - 2n_i - t_i + 2n_i + 2 = 3$

$ci\hat{} = 3$

cas 2 : contraction

$c_i\hat{} = n_{i-1} - 1 + |2*n_i - taille_i| - |2*n_{i-1} - taille_{i-1}|$

on sait que

$n_{i-1} = n_i + 1$

$t_{i-1} = 3/2ti$

$\alpha_{i-1}=n_i-1/t_i-1=1/3$

Donc $t_{i-1}=3n_{i-1}$

$3/2t_i=3(n_i+1)$ donc $t_i=2(n_i+1)$

$ci\hat{}=n_i+(ti-2ni) - |2(ni+1)-3/2t_i|$

$=n_i+2ni+2-2ni-|2n_i+2-3/2*2(n_i+1)|$

$=n_i+2-|2n_i+2-3n_i-3|$

$=n_i+2-|-n_i-1|$

$=n_i+2-(n_i+1)$

$=n_i+2-n_i-1=1$

$ci\hat{}=1$

2)

La modification du Code afin de pouvoir

- Effectuer des opérations d'insertion : en choisissant une probabilité P.
- Effectuer des opérations de suppression avec une probabilité " 1 - P " (Lors de la suppression si la table est vide on ne fait rien).

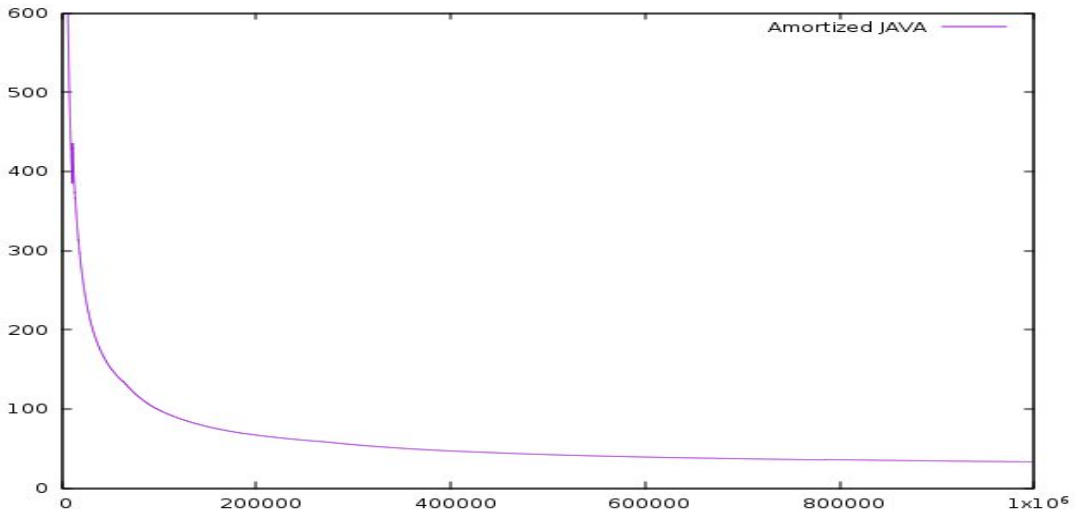
```
double p = 0.9;

      for( i = 0; i < 1000000 ; i++ )
      {
          //if(random_generator.nextBoolean()) {
          if( random_generator.nextDouble() <= p )
              {
                      // l'opération choisie est l'insertion
                      // ajouter un élement
                      //compter le nombre que l'opération a pris
                      before = System.nanoTime();
                      memory_allocation = a.append(i);
                      after = System.nanoTime();

                       // compter le temps que l'opération a pris
                      time_analysis.append(after - before);
                          // enregistrer le nombre de copie que l'opération a pris.

                      copy_analysis.append( (memory_allocation == true)? a.size(): 1);
                              // Enregistrer l'espace mémoire non-utilisé.
                      memory_analysis.append( a.capacity() - a.size() );
              }
          else
              { // le choix de la supression
                      if ( a.size() > 0 )
                      {
                              before = System.nanoTime();
                              memory_allocation = a.pop_back();
                              after = System.nanoTime();

                               // compter le temps que l'opération a pris
                              time_analysis.append(after - before);
                               // enregistrer le nombre de copie que l'opération a pris.

                              copy_analysis.append( (memory_allocation == true)? a.size(): 1);
                              // Enregistrer l'espace mémoire non-utilisé.
                              memory_analysis.append( a.capacity() - a.size() );
                      }

           }


      }
```

3) Les coûts réels et amortis des opérations, ainsi que l'espace mémoire non-utilisé

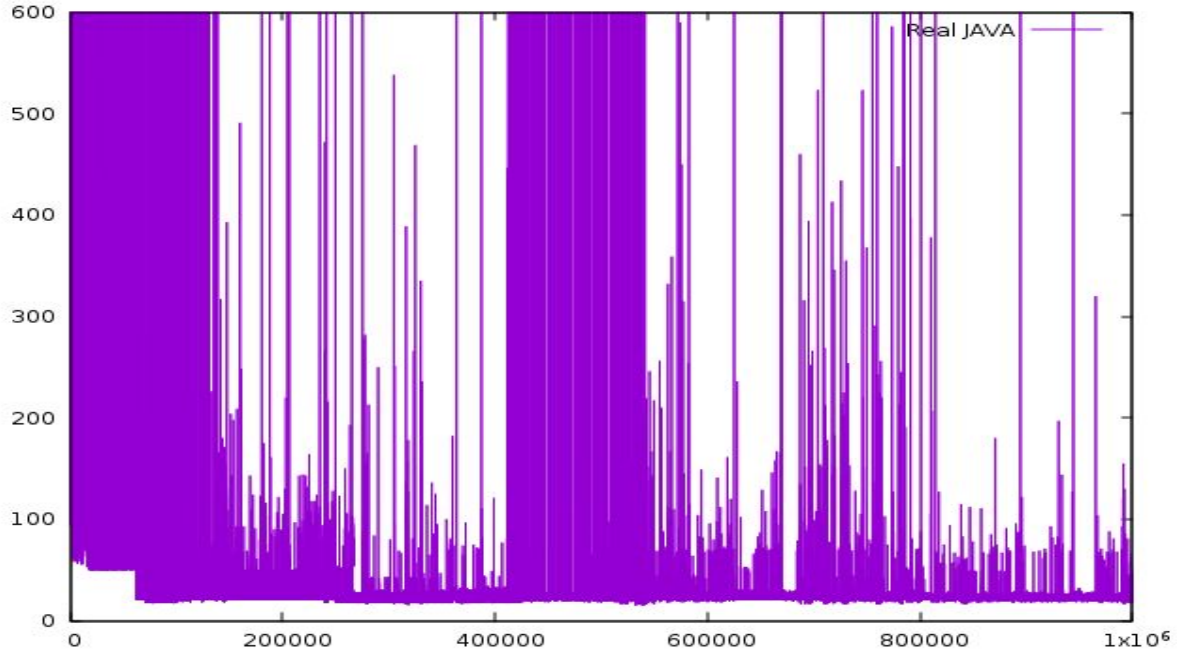
Résultat de compilation :

Total cost : 3.3046553E7
Average cost : 33.046553
Variance :7.186326816726926E12
Standard deviation :2680732.5149531285

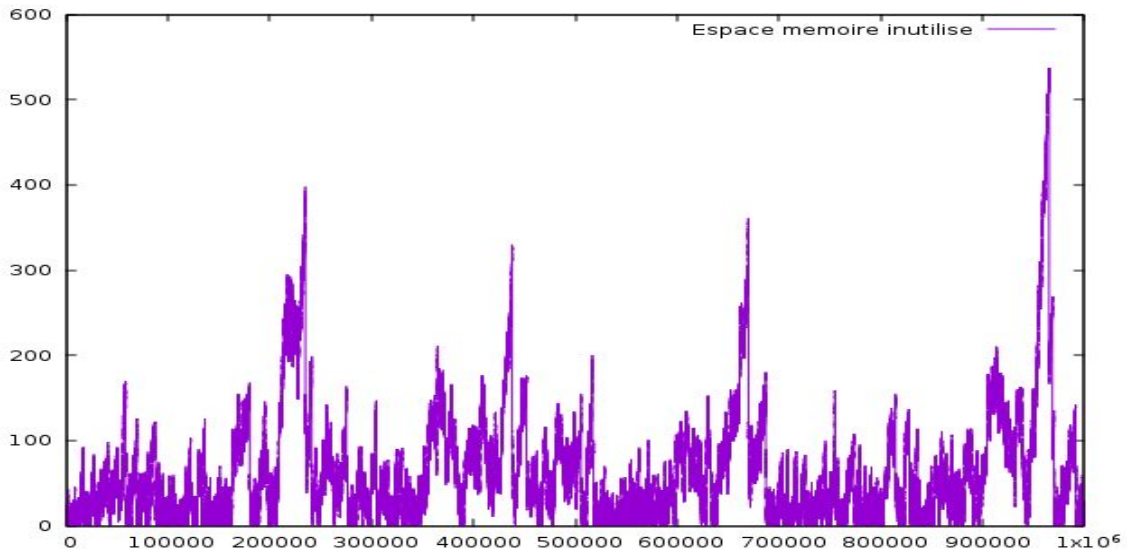
Le coût amorti



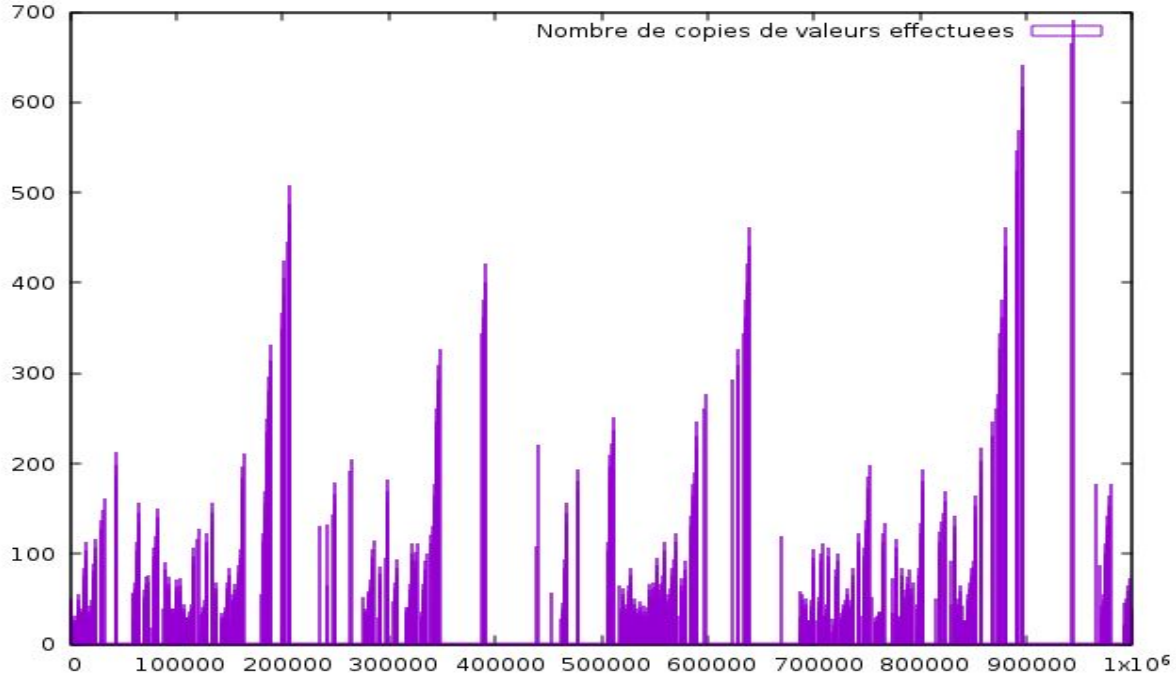
Le coût réel



L’espace inutilisé



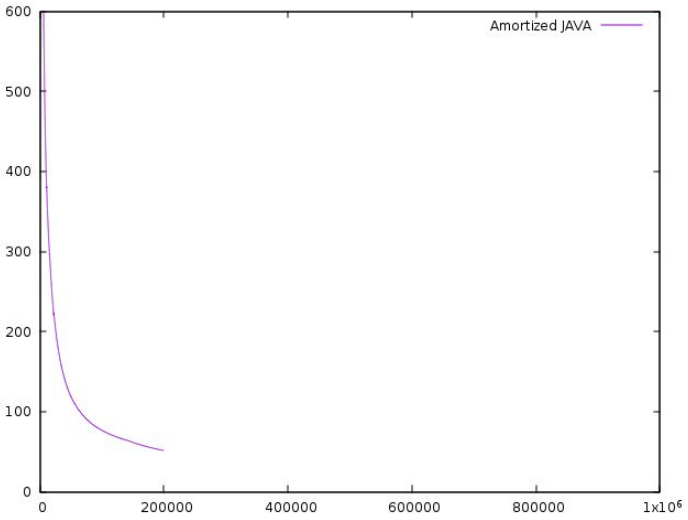
Le nombre de copies

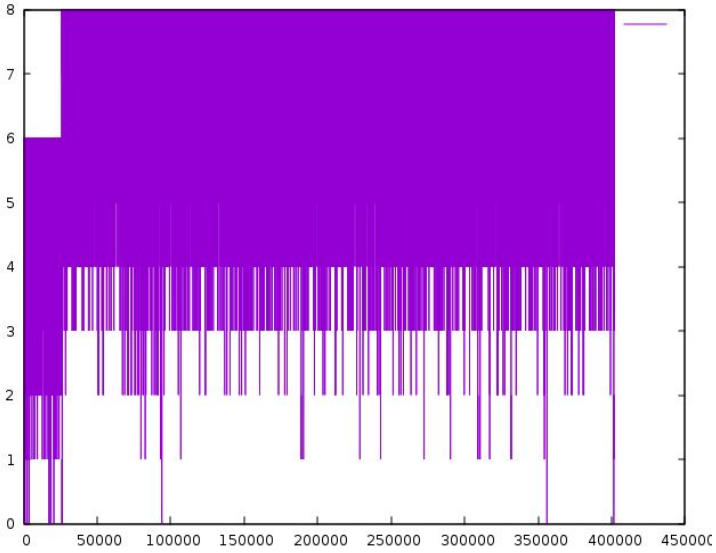


4) les coûts amortis et l'espace mémoire non-utilisé pour chacune des expériences.

p = 0.1 :

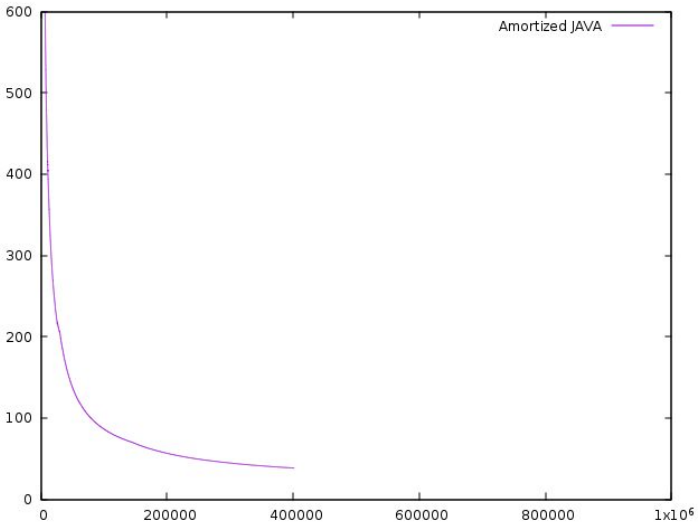
```
Total cost : 1.027868E7
Average cost : 51.441755249934936
Variance :5.846264916355746E12
Standard deviation :2417905.067688917
```

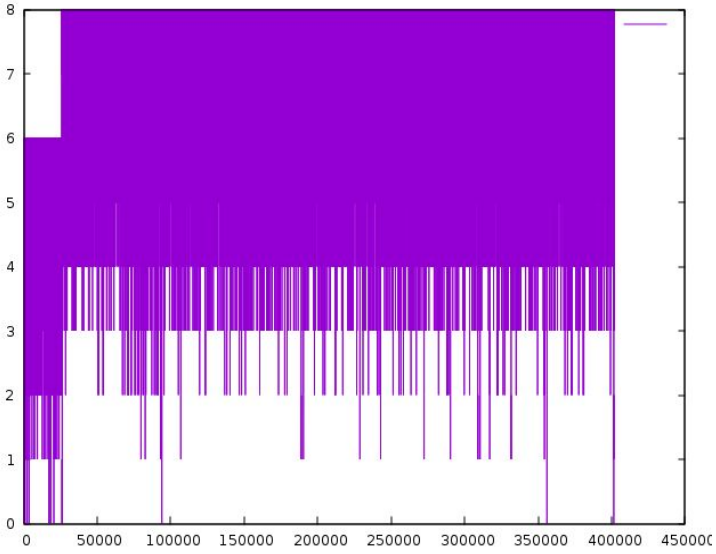




p = 0.2 :

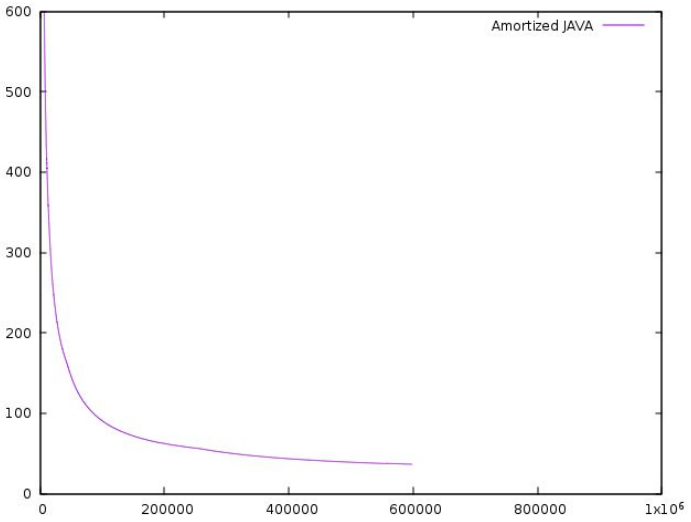
```
Total cost : 1.5407117E7
Average cost : 38.33340880365442
Variance :7.05919215951155E12
Standard deviation :2656914.029379112
```

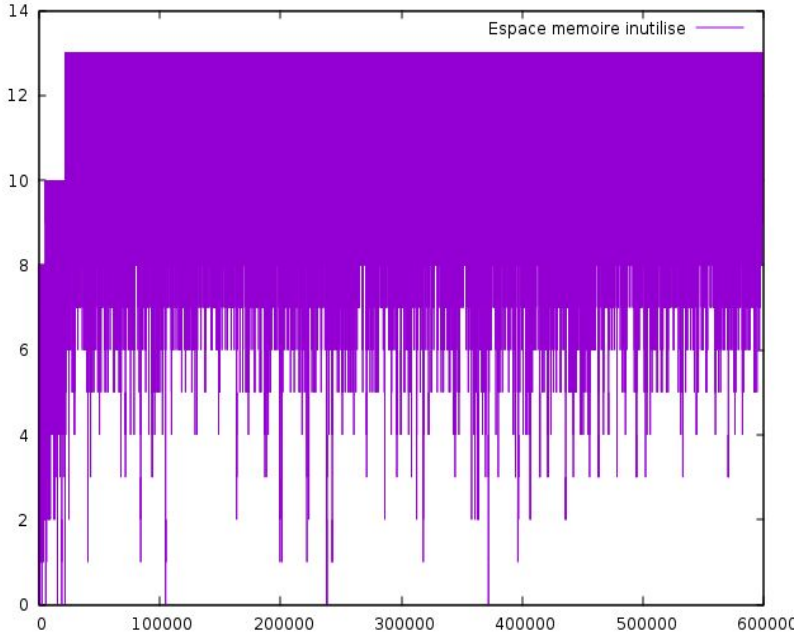




p = 0.3 :

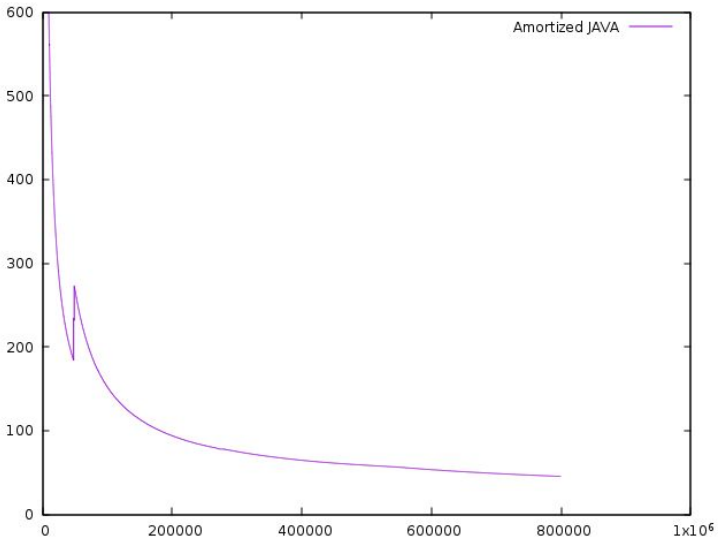
```
Total cost : 2.1817611E7
Average cost : 36.45954586016851
Variance :6.744109620713701E12
Standard deviation :2596942.359913616
```

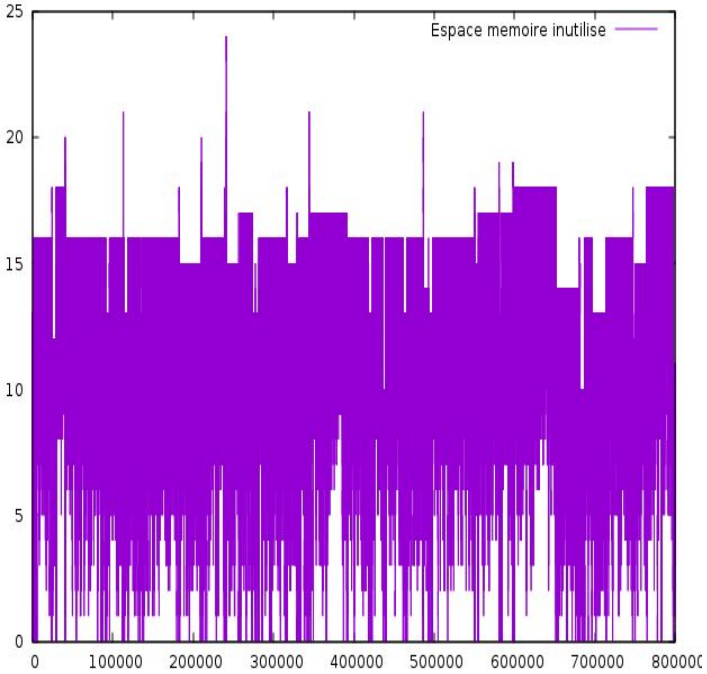




p = 0.4 :

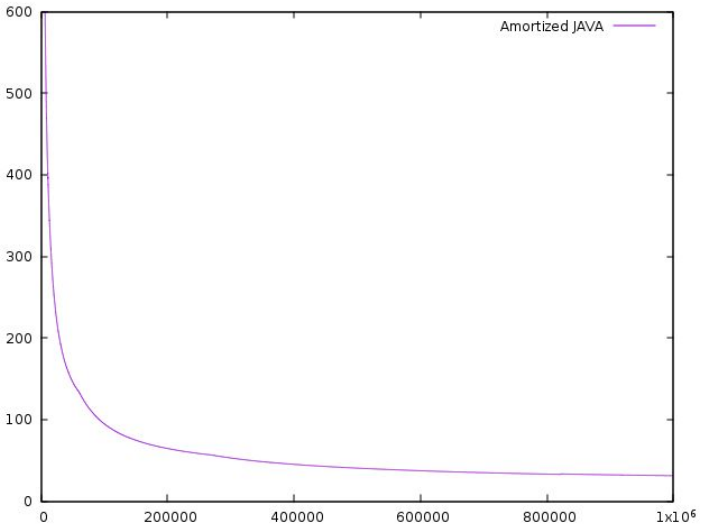
```
Total cost : 3.6215162E7
Average cost : 45.30531001128406
Variance :1.7246747413131 43E13
Standard deviation :4152920.3475544085
```

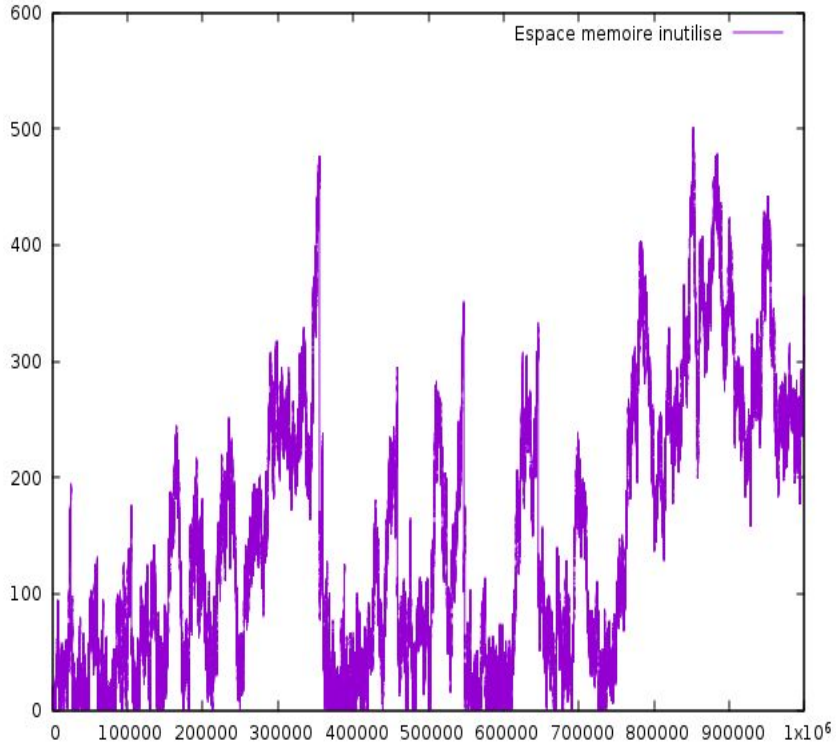




p = 0.5 :

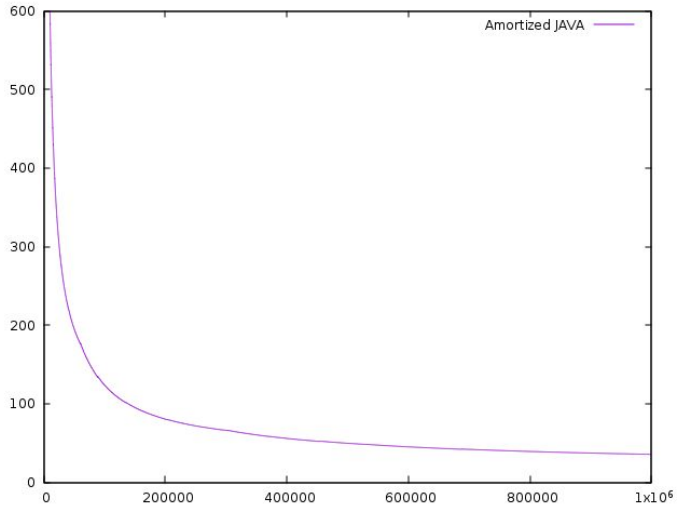
```
Total cost : 3.1138852E7
Average cost : 31.16733894779829
Variance :6.9280465429425 97E12
Standard deviation :2632118.2615799382
```

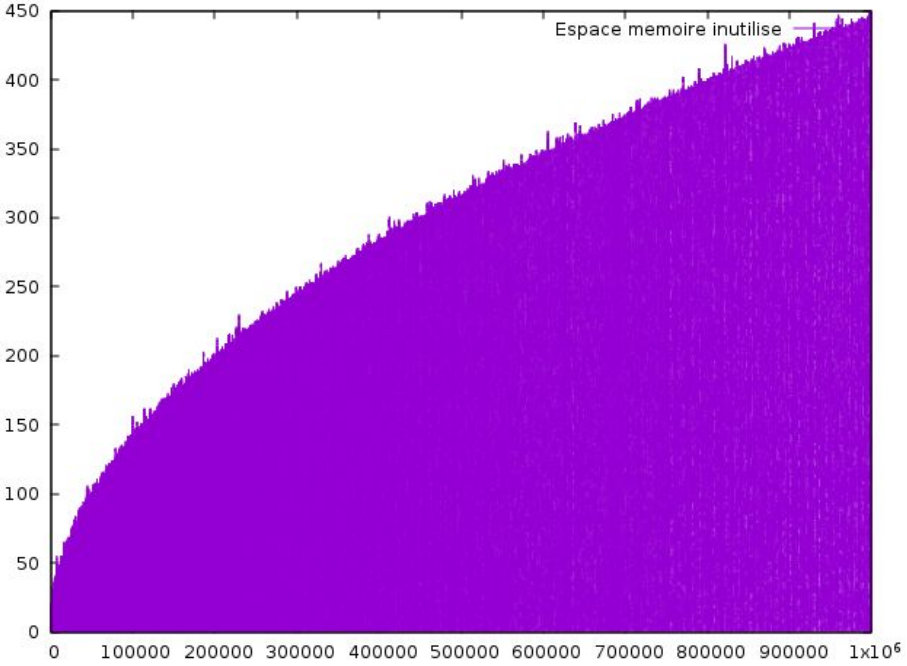




p = 0.6 :

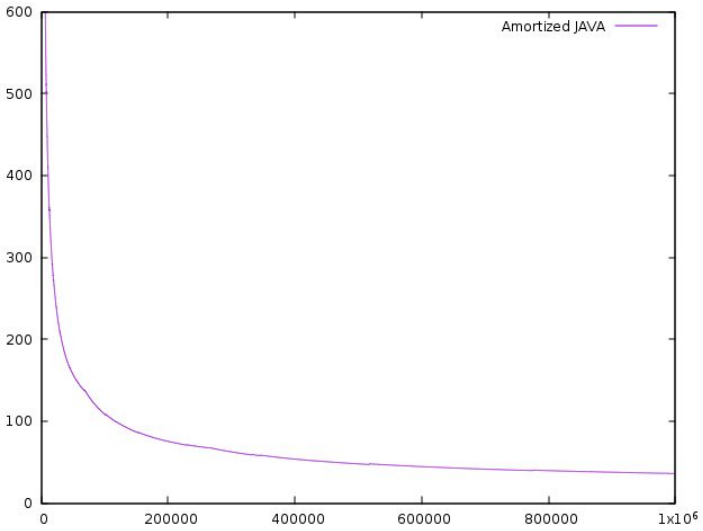
```
Total cost : 3.5536225E7
Average cost : 35.536260536260535
Variance :8.612858325176174E12
Standard deviation :2934767.167114995
```

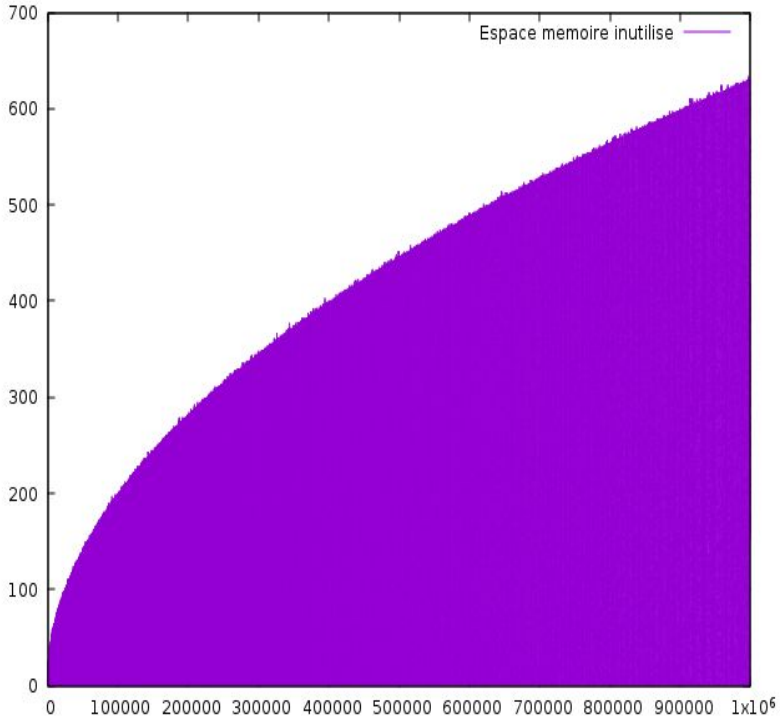




p = 0.7 :

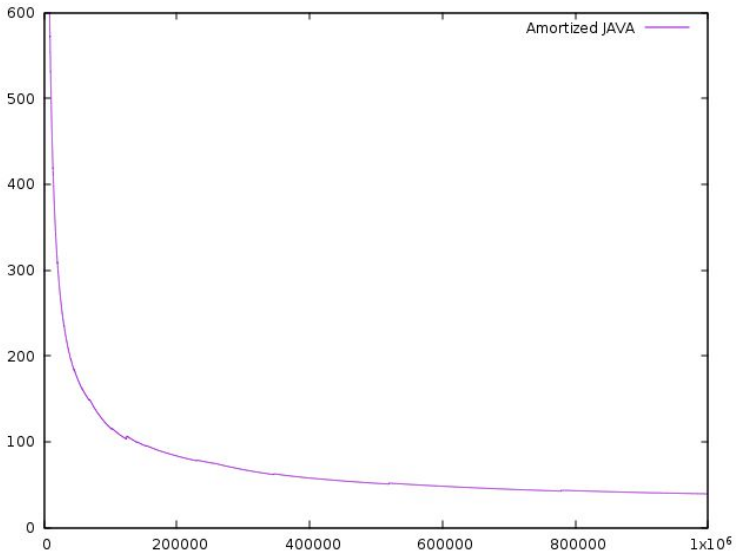
```
Total cost : 3.5981917E7
Average cost : 35.981988963977926
Variance :7.050428922740297E12
Standard deviation :2655264.379066668
```

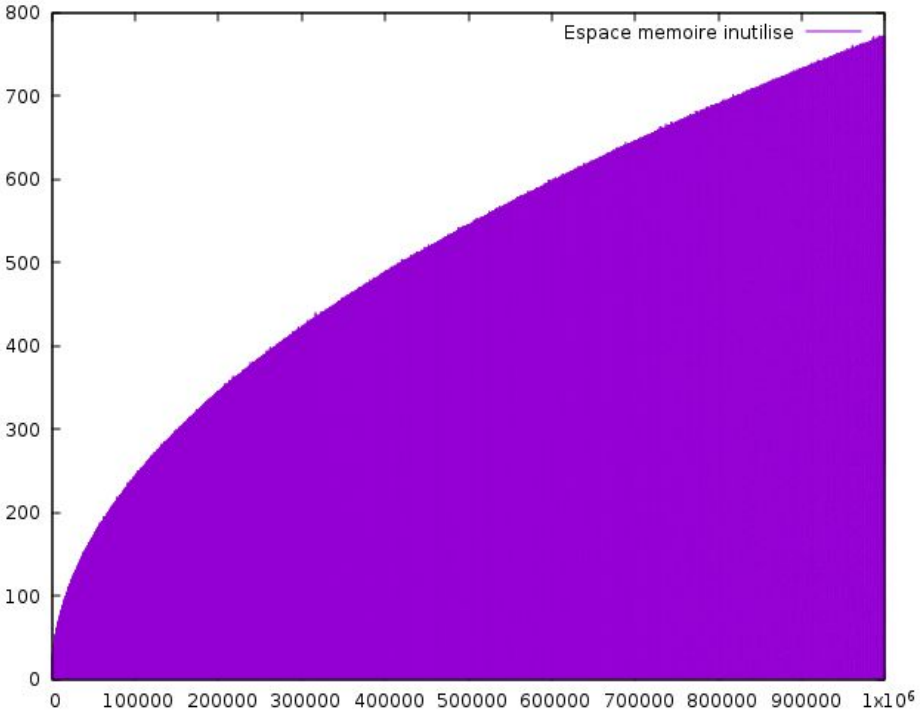




p = 0.8 :

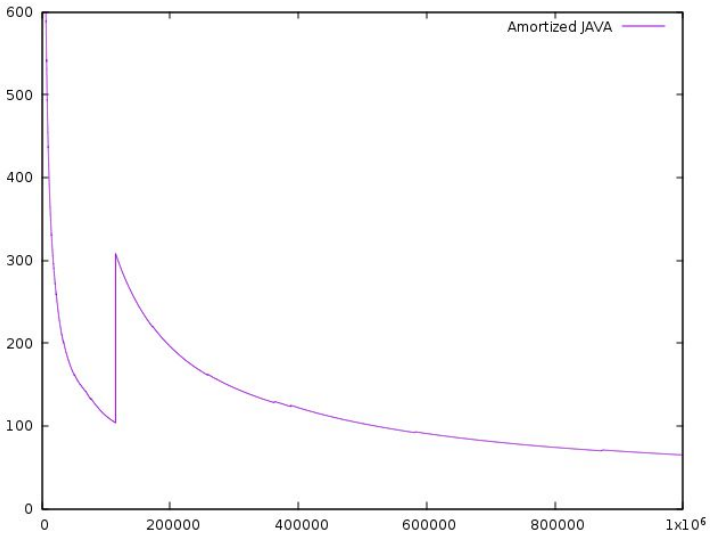
```
Total cost : 3.9207772E7
Average cost : 39.207772
Variance :9.18497555862675E12
Standard deviation :3030672.459806033
```

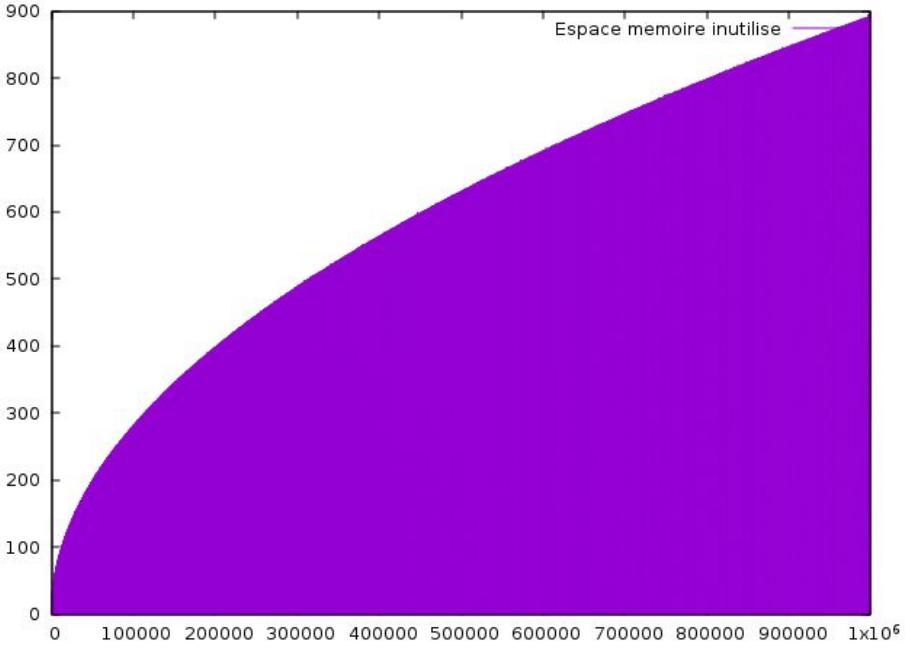




p = 0.9 :

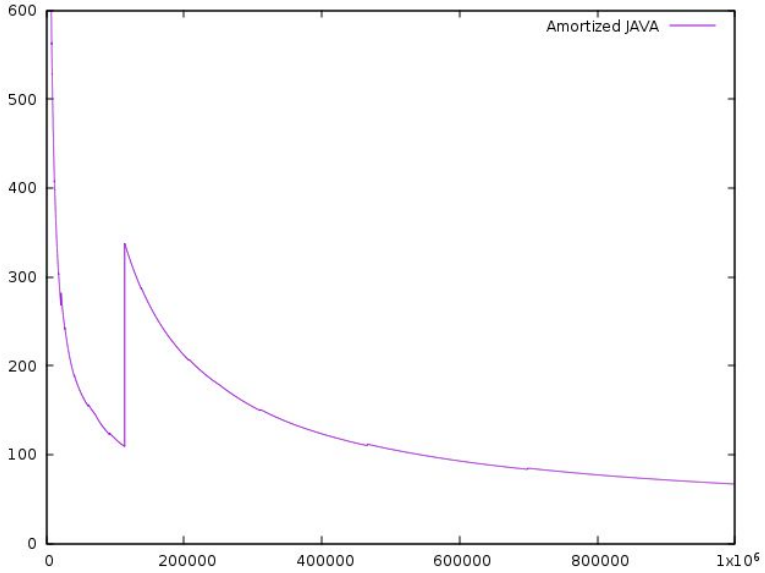
```
Total cost : 6.4874189E7
Average cost : 64.87425387425388
Variance :5.6351510501808 44E14
Standard deviation :2.3738473097865507E7
```

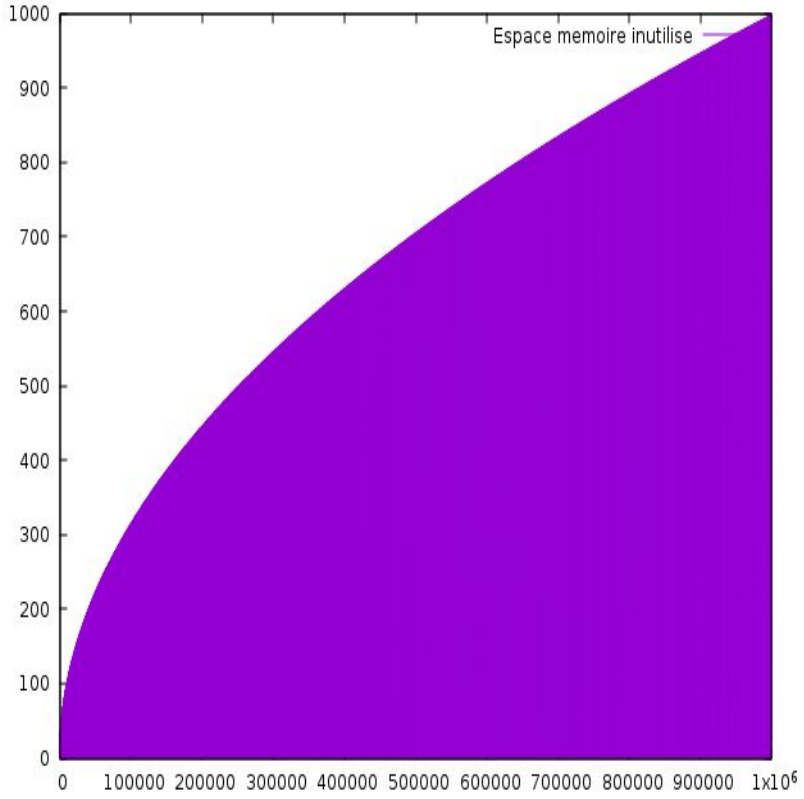




p = 1 :

```
Total cost : 6.6751064E7
Average cost : 66.751064
Variance :6.886904090829202E14
Standard deviation :2.6242911596904036E7
```





La relation entre P, le coût en temps et le gaspillage de mémoire (mémoire inutilisé) :

Nous avons déjà testé les différentes valeurs de probabilité : de 0.1 jusqu'à 1

<u>Nous avons donc remarqué :</u>

Pour p>0,5:

Le nombre d'insertion est plus grand que le nombre de supression,le coût amorti est élevé au départ puis il diminue pour rester constant.En ce qui concerne l'espace inutilisé,on remarque que cette espace augmente a chaque fois qu'on effectue des opérations sur la table

pour p<=0,5

Ici le nombre de suppression est plus grand que le nombre d'insertion,c'est le même constat que p>0,5 cad la valeur du coût amorti reste élevé au départ et diminue sauf que le nombre d'opérations est limitées (tend pas vers l'infini).Pour l'espace inutilisé on remarque que sa valeur est très petite par rapport à p>0,5

Conclusion

Plus on la probabilité p est petite ,plus on diminue de l'espace inutilisé et plus p est grande, cette espace croît de manière exponentielle

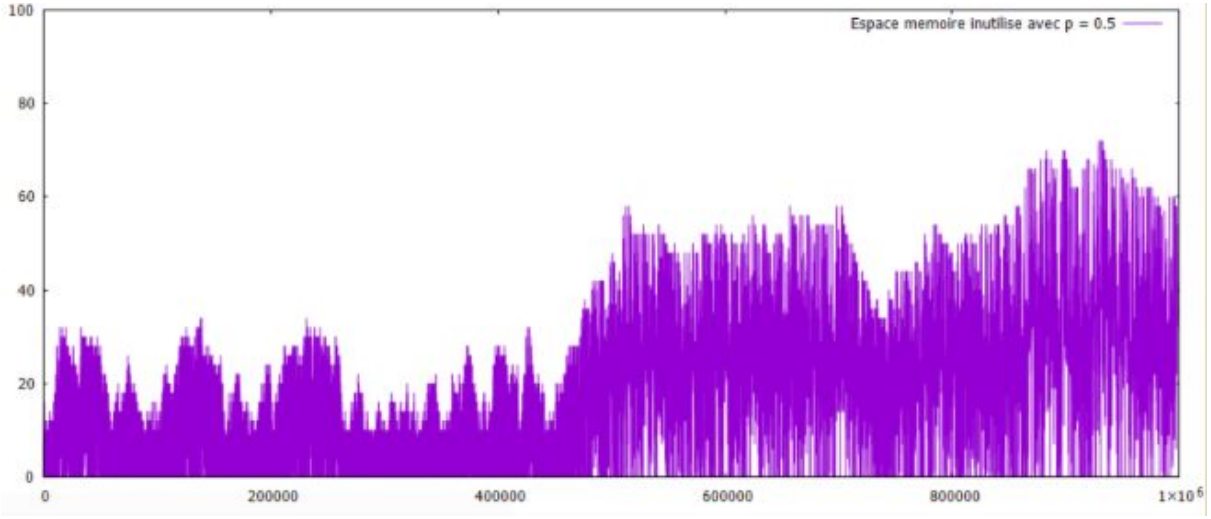
5)
On contracte la table lorsqu'on a le nombre d'éléments est inférieur ou égale à la moitié de la table.

Voici la partie du code modifiée:

```
private boolean do_we_need_to_reduce_capacity(){
    //return data.size() <= capacity/4 && data.size() >4;
    return data.size() < capacity/2;
}
```

l'efficacité de cette stratégie

l'espace inutilisé:



Avec cette stratégie on constate que l'espace inutilisé diminue par rapport au cas précédent.Donc c'est efficace en espace.