|  |
| --- |
|  |
| Structures de données  Tableaux dynamiques 2 |
|  |
|  |
| **Lynda MEDJDOUBI**  **Sabrina DJENADI** |
|  |

**Sommaire**

**Insertion des clés dans l’ordre croissant4**

Pour alpha = 24

Pour alpha = nombre d’or 8

Pour alpha = 2013

**Insertion des clés dans l’ordre décroissant16**

Pour alpha = 216

Pour alpha = nombre d’or19

Pour alpha = 5022

**Insertion des clés aléatoires24**

Pour alpha = 224

Pour alpha = nombre d’or27

**Insertion des clés et extraction du minimum31**

Pour alpha = 231

**Conclusion34**

**Expansion et contraction dynamiques des tables**

**Expansion et contraction dynamiques des tables P = 0.5 :**

Lorsque la probabilité P est inférieure à 0.5 on ne fait que des insertions et on garde le principe que lorsqu’un élément est inséré dans le tableau complet on double la taille de la table. En parallèle si la probabilité P est supérieure à 0.5 on ne fait olus de suppressions, en gardant aussi le principe que lorsqu’on supprime un élément, le tableau est réduit en divisant par deux sa taille.

On a affiché le coût réel de l’opération insertion et suppression, on remarque que le coût réel est très perturbé il augmente jusqu’à atteindre des valeurs qui dépassent 500 surtout en effectuant cette opération avec un nombre de valeurs très grand ; prenant exemple entre 800000 et 900000 valeurs le coût réel augmente énormément et rencontre une perturbation importante. On remarque aussi qu’on n’a pas beaucoup d’allocation mémoire ce qui revient au fait qu’on fait des suppressions qui provoquent des contractions du tableau dynamique.

la probabilité d’effectuer une insertion est égale à la probabilité de faire une suppression ; donnons un exemple d’une séquence de suppression et insertion avec P = 0.5 : insertion, insertion, suppression, insertion, insertion, suppression, suppression, suppression, insertion. Ce qui explique les résultats obtenues dans ces figures.

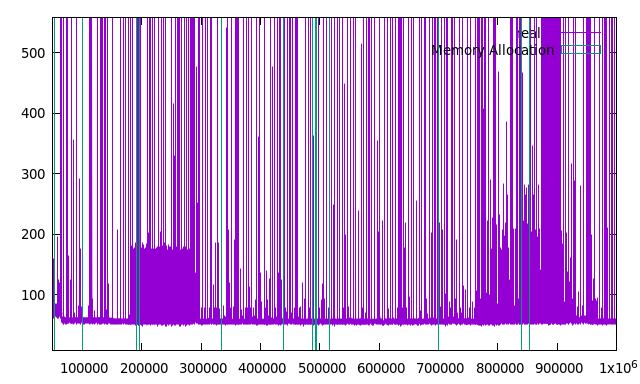


Figure  : Coût réel et allocation mémoire pour p = 0.5

Ensuite on a affiché le coût amorti, on remarque qu’il rencontre des augmentations et des démunissions au début ; pour qu’il se stabilise entre 80 et 100 et cela revient au fait qu’on fait des insertions, on alloue de la mémoire puis on copie les éléments du tableau dans notre nouveau tableau pour ensuite insérer le nouvel élément ce qui nous coute cher surtout dans le cas où notre tableau est grand. On remarque aussi qu’on a beaucoup d’allocation mémoire au début ce qui revient au fait qu’on insère beaucoup au début ensuite le tôt d’allocation mémoire diminue et cela provient des suppressions qu’on effectue en parallèle des insertions comme déjà mentionné précédemment la probabilité d’effectuer l’insertion est égale à la probabilité de faire une suppression ce qui diminue aussi le nombre d’allocations mémoire.

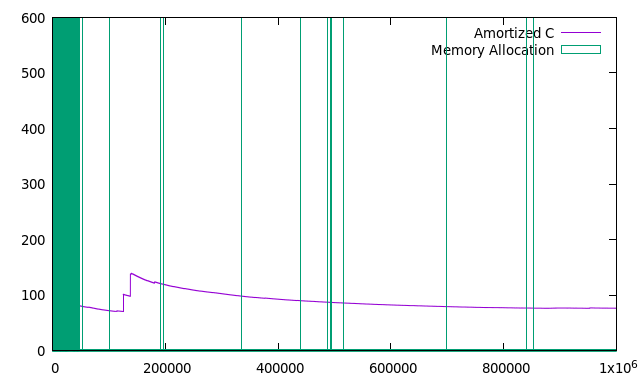


Figure  : Coût amorti et allocation mémoire pour p = 0.5

Ensuite on a affiché l’espace mémoire inutilisé amorti et réel, on remarque que la valeur de l’espace mémoire inutilisée augmente énormément dans quelques moments et diminue dans d’autres moments ; ce qui revient au fait qu’on effectue des insertions ce qui provoque l’allocation mémoire ; donc l’augmentation de la mémoire inutilisée surtout dans le cas où le tableau est très grand. Et le fait d’avoir des suppressions et de contracte la taille du tableau fait diminuer la mémoire inutilisée ce qui justifie la variation des graphes de l’espace mémoire inutilisé amorti et réel de l’opération insertion et suppression et on le remarque plus en cout réel.

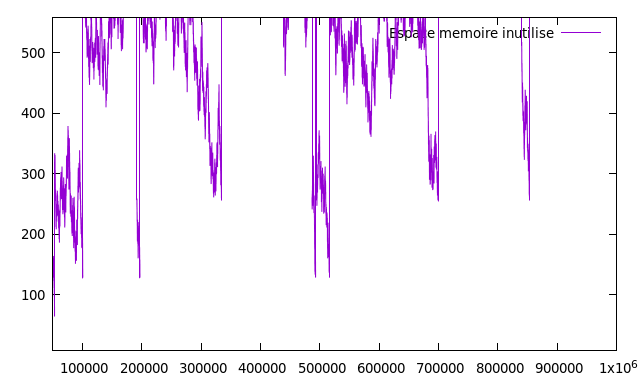


Figure  : Espace mémoire inutilisé en fonction du temps réel pour p = 0.5

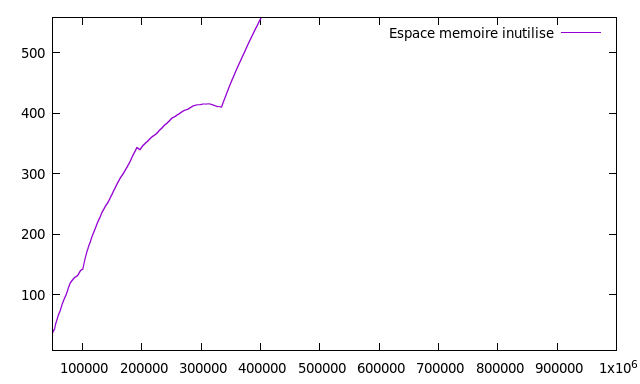


Figure  : espace mémoire inutilisé en fonction du temps amorti pour p = 0.5

Ensuite on a affiché le nombre de copies de valeurs effectuées amorti et réel de l’opération insertion et suppression, on remarque que le nombre de copies augmente au début atteignant une valeur de 160 ce revient au nombreuses allocations mémoire effectuées à ce moment là et c’est ce qu’on a mentionné précédemment. Après cette immense augmentation du nombre de copies, le graphe diminue petit à petit jusqu’à atteindre une valeur de 20. Effectivement précédemment aussi on a remarqué que le nombre d’allocation diminue donc le nombre de copies diminue automatiquement.

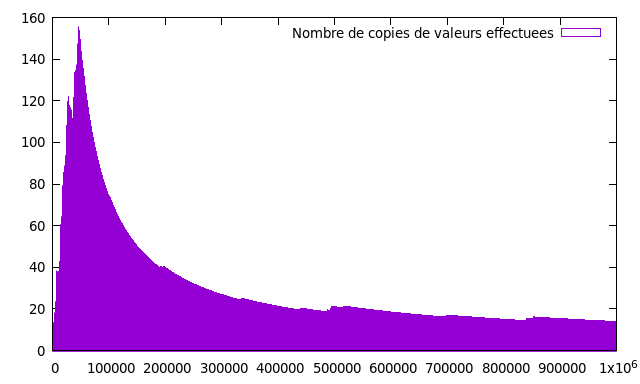


Figure  : Nombre de copies effectuées pour p=0.5

**Insertions et suppressions avec p = 0.1:**

Maintenant on va testé le cas où on fais des insertions et suppression avec p = 0.1, ce qui veut dire qu’on fais des insertions avec probabilité 0.1 et des suppression avec une probabilité égal à 0.9, tout en sachant que quand le tableau est rempli on fait automatiquement une insertion.

On remarque d’après ce graphe que le coût amorti est très petit et cela s’explique par le fait qu’on fait beaucoup moins d’insertions que de suppressions ce qui nous fait gagné en temps d’exécution.

La suppression se fait en temps constant quand le nombre d’éléments du tableau dépasse un quart de sa taille car on ne fait que retirer le dernière élément, par contre si le nombre d’éléments est plus petit, dans ce cas là on va contracter le tableau et recopier les éléments du tableau.

Le nombre de copies aussi est très réduit pour la même raison précédente, on n’insère pas beaucoup d’élément, on n’a donc pas besoin de faire d’allocation mémoire, d’où le nombre de copies réduit.

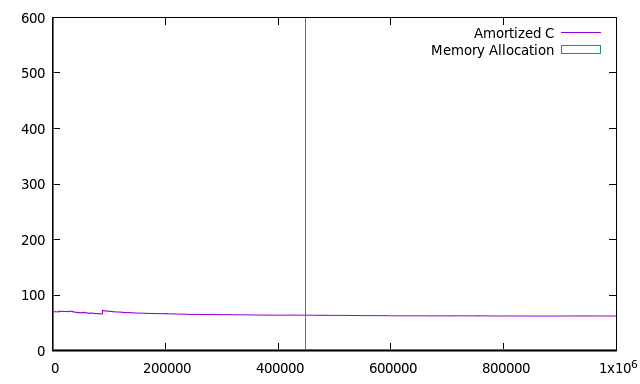


Figure  : Coût amorti et allocation mémoire pour l’insertion et suppression avec p = 0.1

On a aussi affiché le coût réel, on remarque qu’il est plus perturbé que le coût amorti car il change à chaque opération et dépend de l’état de la structure.

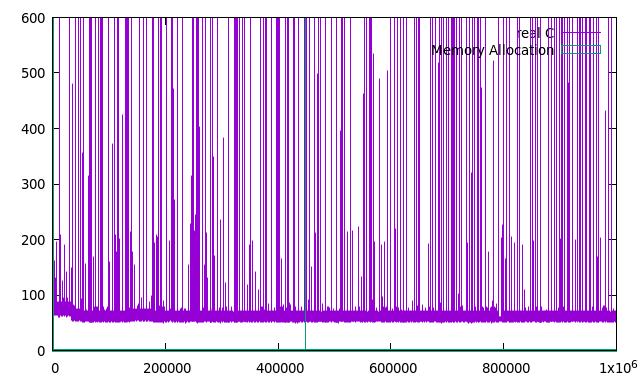


Figure  : Coût réel et allocation de l’insertion et suppression pour p = 0 .1

En ce qui concerne l’espace mémoire inutilisé on remarque que ses valeurs ne dépassent pas 10, ce qui est très peu mais assez logique car, comme dit précédemment, on ne fait pas beaucoup d’allocation mémoire donc on n’en gaspille pas.

Le fait de supprimer plus fait que le tableau est souvent contracté et ainsi on n’a pas beaucoup de cases mémoires vides.

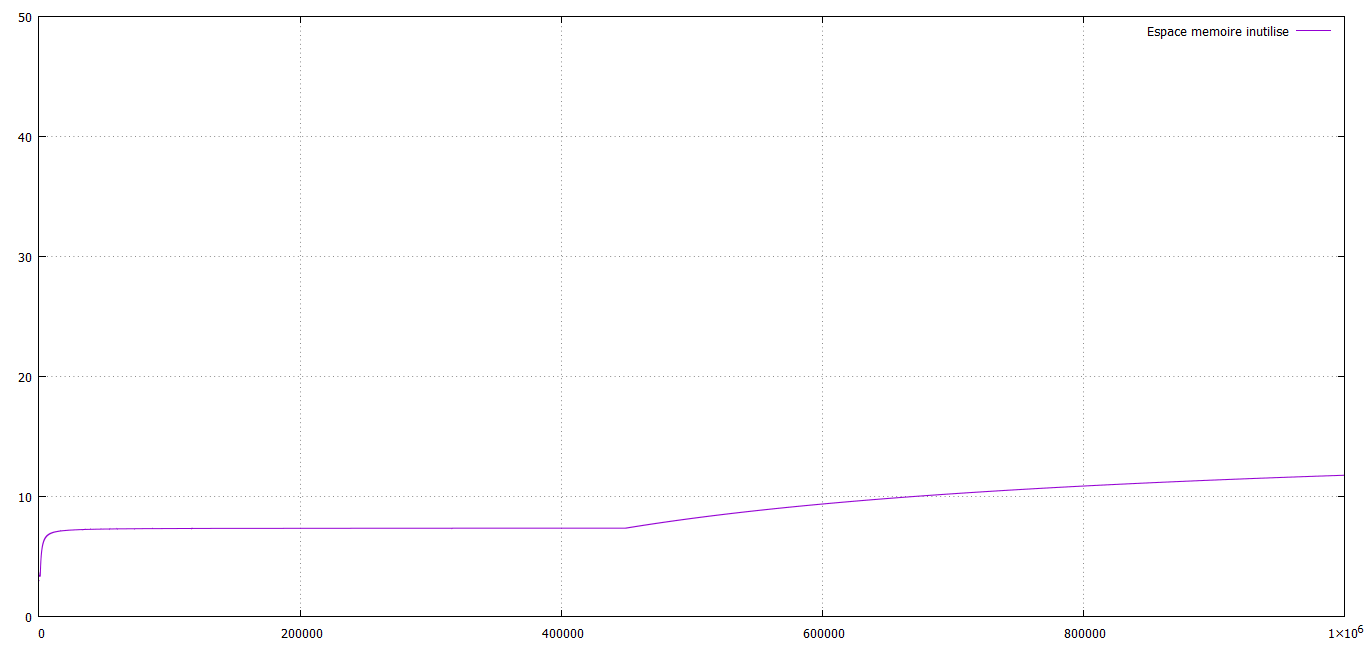
****

Figure  : Espace mémoire inutilisé avec p = 0.1

**Insertions et suppressions avec p = 0.2:**

On a refait la même expérience pour p = 0.2, ce qui veut dire qu’ici on fait des insertions avec une probabilité égal à 0.2 donc un peu plus que l’expérience précédente, ce qui se voit d’ailleurs sur ce graphe du coût amorti car celui-ci a un peu augmenté comparé au cas précédent. En effet en insérant plus le coût amorti augmente plus.

Le fait de faire plus de suppressions sur le tableau fait que celui-ci n’est pas très souvent rempli ce qui réduit le nombre d’allocation mémoire et de copies.

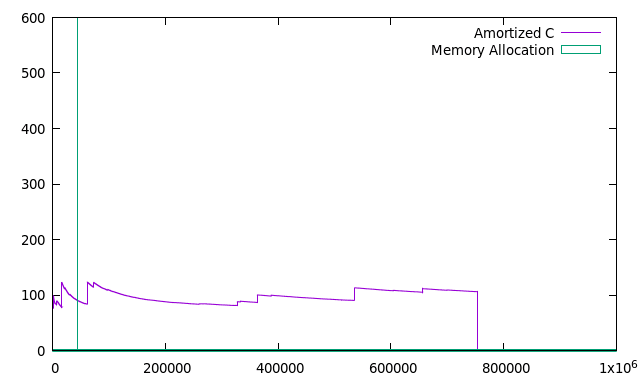


Figure  : Coût amorti et allocation mémoire pour l’insertion et suppression avec p = 0.2

On a aussi affiché le coût réel de ces insertions et suppressions et on remarque qu’il a un peu augmenté tout comme le coût amorti et cela pour les mêmes raisons.

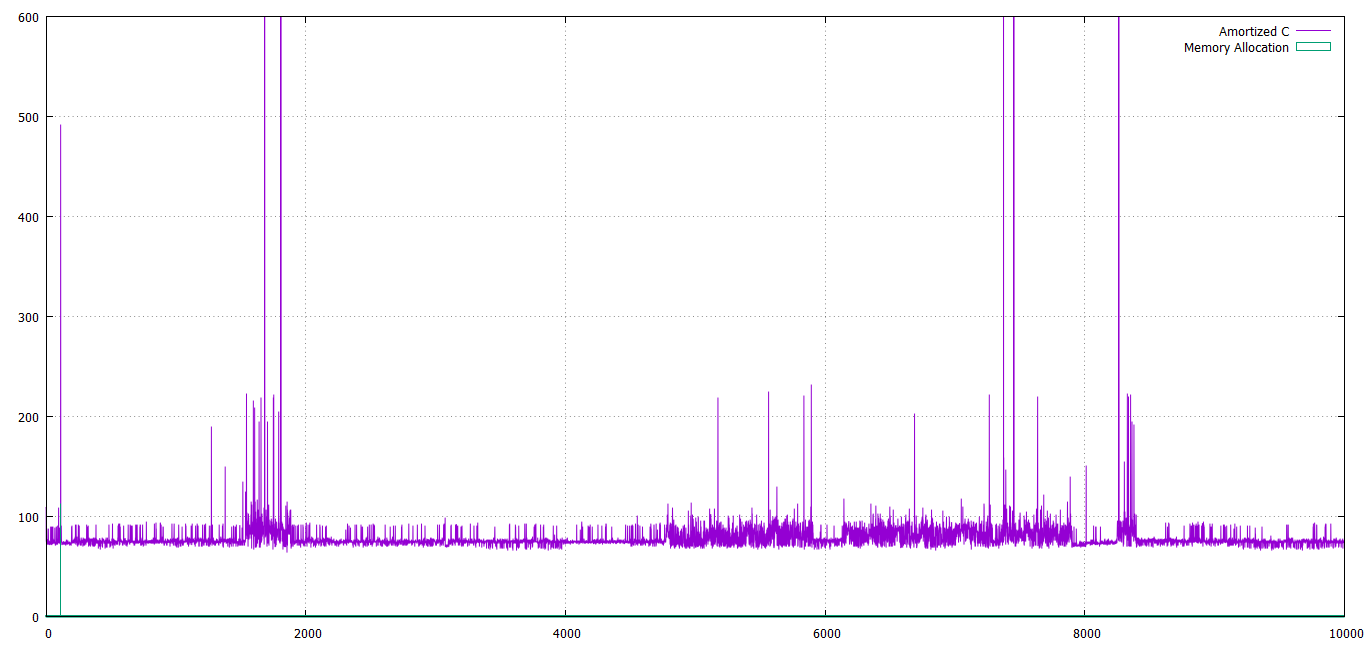


Figure  : coût réel et allocation mémoire avec p = 0.2

Passant maintenant à l’analyse de l’espace mémoire inutilisé, on remarque qu’il a aussi augmenté car on fait plus d’allocation mémoire, ainsi on en gaspille plus.

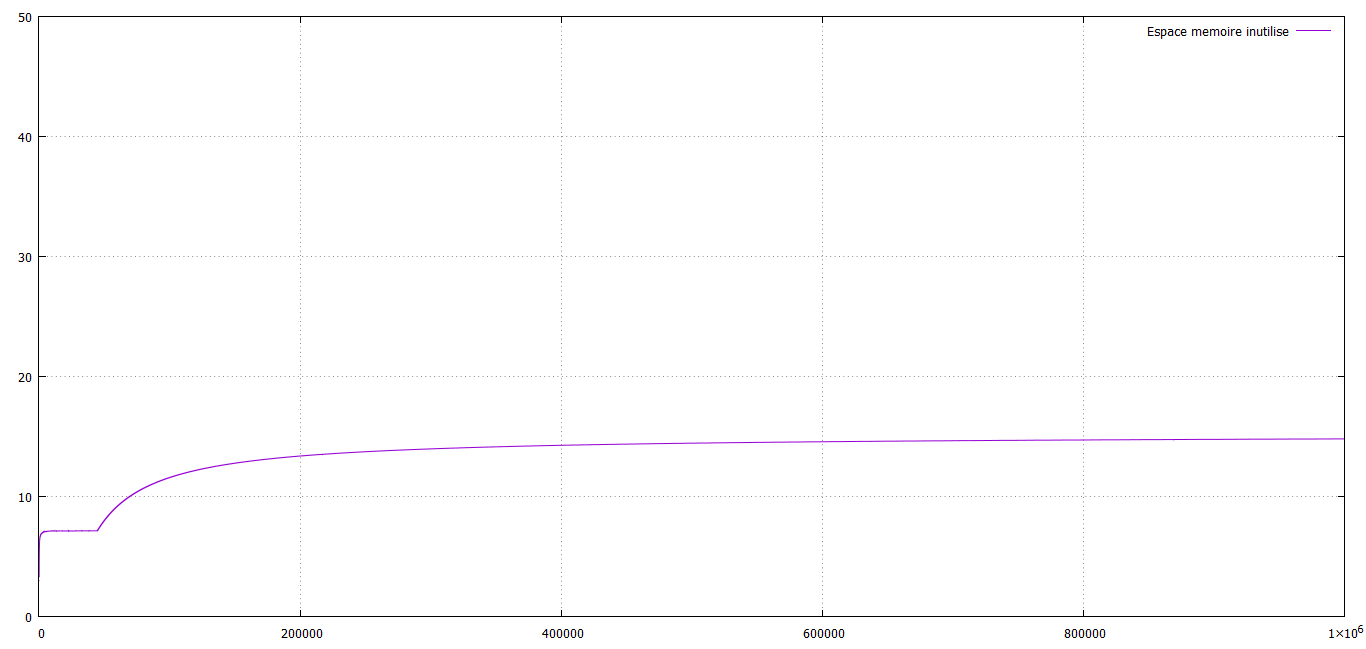


Figure  : Espace mémoire inutilisé avec p = 0.2

Faisant maintenant le lien entre l’espace mémoire inutilisé et le nombre de copies effectuées (ou l’allocation mémoire).

On remarque que l’espace mémoire inutilisé augmente quand le nombre de copies augmente aussi car c’est à ce moment là que l’on fait de l’allocation mémoire donc il nous reste des cases vides dans le tableau.

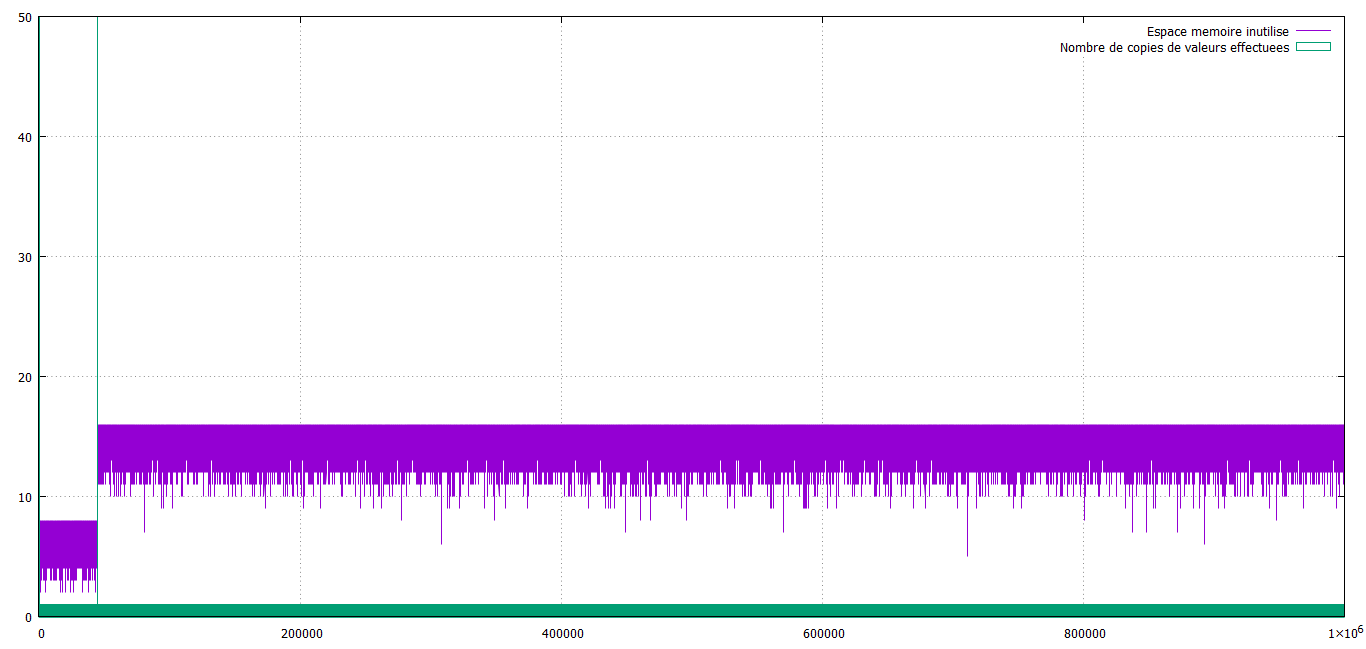


Figure  : Espace mémoire inutilisé en fonction du temps amorti et nombre de copies effectuées pour p = 0.2

**Insertions et suppressions avec p = 0.3:**

Dans cette expérience nous allons effectuer un peu plus d’insertions que les fois précédentes.

Pour commencer nous avons affiché le coût amorti ainsi que les allocations mémoire, On remarque qu’ici on fait beaucoup plus de copies car on insère plus d’éléments, le tableau est souvent rempli alors on alloue de la mémoire et recopie les éléments dans le nouveau tableau.

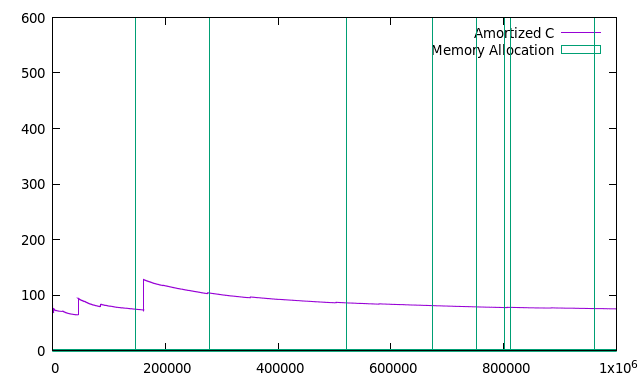


Figure  : Coût amorti et allocation mémoire pour p = 0.3

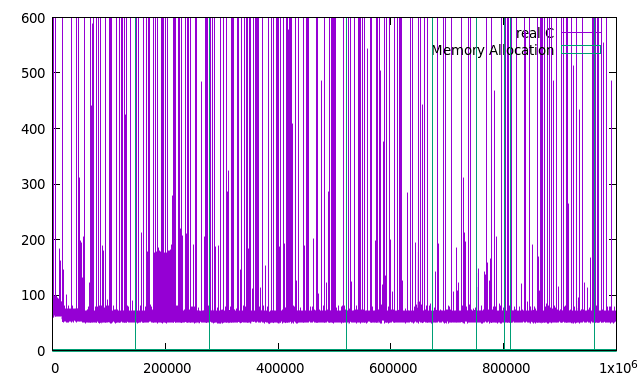
****

Figure  : coût réel et allocation mémoire pour p = 0.3

L’espace mémoire inutilisé a augmenté un peu comparé aux expériences précédentes, tout simplement par ce que on multiplie la taille du tableau par 2 à chaque réallocation ce qui nous fais perdre en mémoire mais pas autant que ca, car ne l’oublions pas, dans cette expérience, on fait aussi des suppressions ce qui veut dire que le tableau est constamment contracté ainsi la mémoire gaspillée pas très élevée.

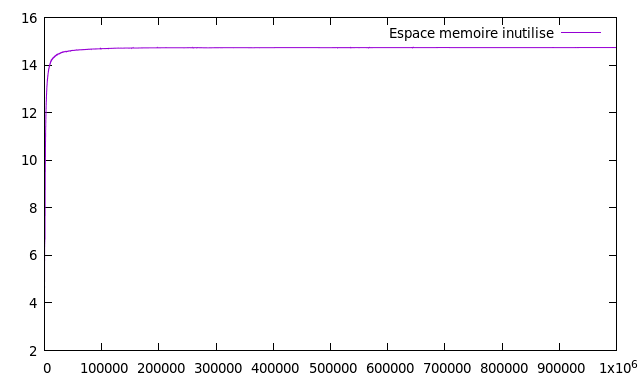


Figure  : Espace mémoire inutilisé en fonction du temps amorti pour p = 0.3

**Expansion et contraction dynamiques des tables P = 0.9 :**

Lorsque la probabilité P est inférieure ou égal à 0.9 on ne fait que des insertions et symétriquement lorsque la probabilité P est supérieure à 0.9 on ne fait que des suppressions.

On a affiché le coût réel de l’opération insertion et suppression, on remarque que le coût réel est carrément caché à cause de la couleur verte qui représente l’allocation mémoire. De ce fait on a zoomer pour bien analyser ce que la figure ci-dessous nous montre le coût réel est très perturbé il augmente jusqu’à atteindre des valeurs qui dépassent 600 ; cela est remarquable surtout au début. On remarque aussi qu’on a énormément d’allocation mémoire, une suite d’allocation consécutive ce qui revient au fait qu’on fait beaucoup d’insertion lorsque la probabilité P est inférieure à 0.9 on ne fait que des insertions et donc automatiquement le nombre d’allocation augmente énormément. La probabilité de faire une insertion est beaucoup plus grande que la probabilité d’effectuer une suppression ; donnons un exemple d’une séquence de suppression te insertion avec P = 0.9 : insertion, insertion, insertion, insertion, suppression, insertion, insertion, insertion, insertion, suppression, insertion, insertion, insertion. Ce qui explique les résultats obtenues dans ces figures

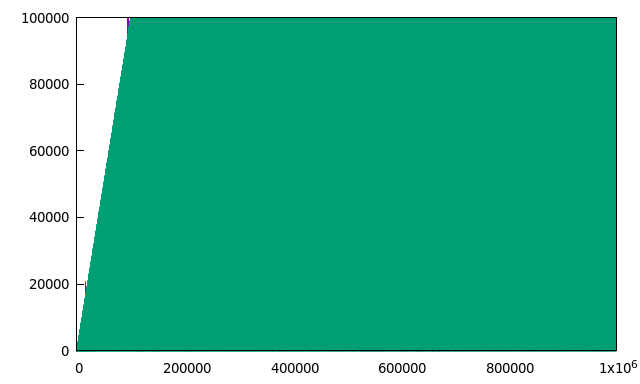


Figure  : Coût réel et allocation mémoire pour p = 0.9

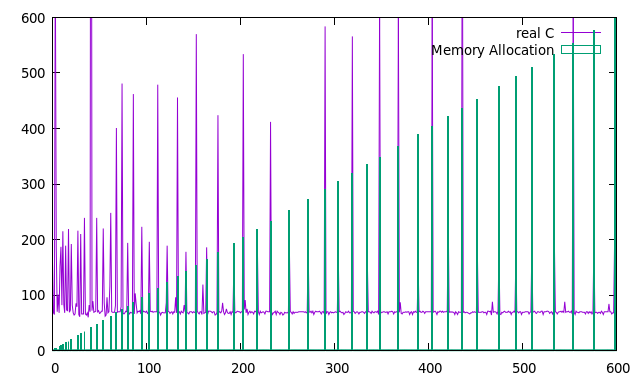


Figure  : Coût réel et allocation mémoire pour p = 0.9

Ensuite on a affiché le cout amorti de l’opération insertion et suppression, on remarque que le cout amorti est carrément caché comme notre cas précédent à cause de la couleur verte qui représente l’allocation mémoire. De ce fait on a zoomé pour bien analyser ce que la figure ci-dessous nous montre que le coût amorti rencontre des augmentations (atteint 300) et des démunissions au début ; pour qu’il se stabilise entre 100 et 120 et cela revient au fait qu’on fait beaucoup plus d’insertions que de suppressions.

On remarque aussi qu’on a beaucoup d’allocation mémoire que ce soit au début ou à la fin ; c’est une suite d’allocation qui augmente de plus en plus ce qui revient au fait qu’on insère beaucoup ce qu’on a déjà mentionné précédemment. La probabilité d’effectuer des suppressions est très faible donc c’est rare de rencontrer des contractions de la taille du tableau.

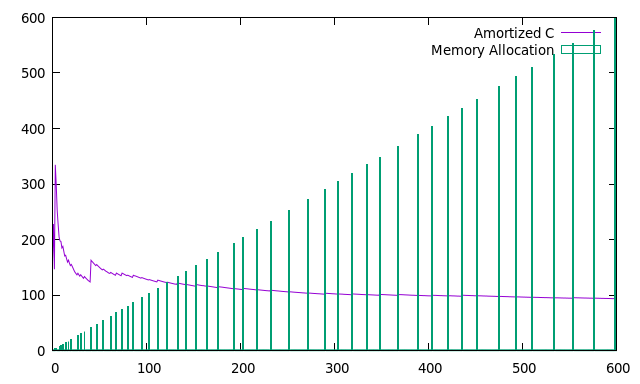


Figure  : coût amorti et allocation mémoire pour p = 0.9

Ensuite on a affiché l’espace mémoire inutilisé amorti et réel de l’opération insertion et suppression, on remarque que la valeur de l’espace mémoire inutilisée augmente énormément dans quelques moments et cela petit à petit tant qu’on insère tant qu’on gaspille du mémoire vu qu’on alloue de la mémoire à chaque allocation. On remarque son augmentation immense dans le cas où le tableau est très grand.

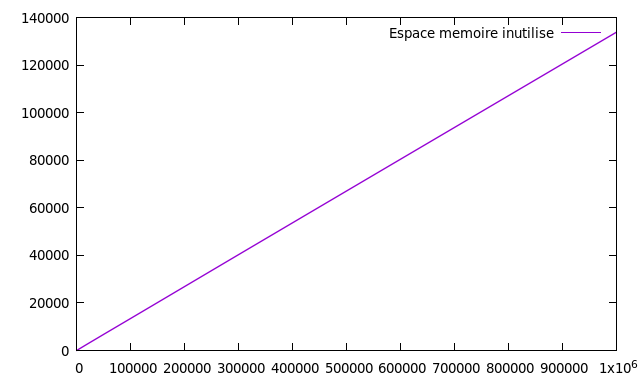


Figure  : Espace mémoire inutilisé pour p = 0.9

Ensuite on a affiché le nombre de copies de valeurs effectuées de l’opération insertion et suppression, on remarque que le nombre de copies a augmenté énormément par rapport à notre cas précédent où la probabilité était 0.5. Le nombre de copies est passé de 20 au début jusqu’à atteindre 700 et cela revient à ce qu’on a mentionné précédemment dans le graphe du coût amorti, on avait observé qu’il y avait énormément d’allocation mémoire sur le long de nos insertions. L’allocation mémoire qui a augmenté automatiquement le nombre de copies, vu qu’on copie les éléments du tableau pour les mettre dans le nouveau tableau alloué.

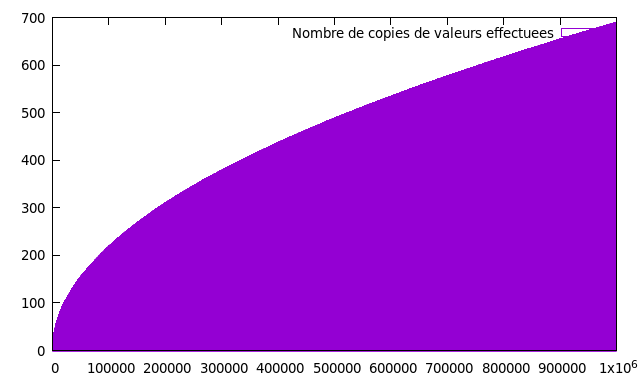


Figure  : Nombre de copies effectuées pour p = 0.9

**Expansion et contraction dynamiques des tables P = 0.7:**

Lorsque la probabilité P est inférieure à 0.7 on ne fait que des insertions et symétriquement lorsque la probabilité P est supérieure à 0.7 on ne fait que des suppressions.

On a affiché le coût réel de l’opération, on remarque que le coût réel est très perturbé, il augmente jusqu’à atteindre des valeurs qui dépassent 600 ; cela est remarquable tout au long de l’expérience. On remarque aussi qu’on a quelques allocations mémoire, qui revient au fait qu’on fait des insertions lorsque la probabilité P est inférieure à 0.7 on ne fait que des insertions et donc automatiquement le nombre d’allocation augmente mais comparé à notre précédant où la probabilité était 0.9 ici on fait moins d’insertions n’empêche qu’on a le d’allocations qui est aussi important dans ce cas et ce qu’on remarque surtout au début vu qu’on a plus d’insertion au début de l’expérience. La probabilité de faire une insertion est beaucoup plus grande que la probabilité d’effectuer une suppression ce qui justifie aussi l’augmentation du nombre de fois qu’on alloue de la mémoire

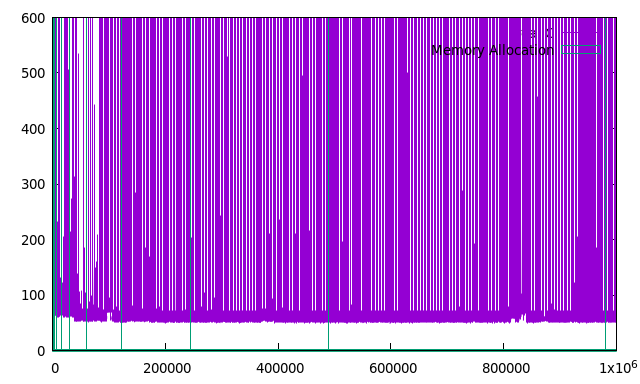


Figure  : Coût réel et allocation mémoire pour p = 0.7

### Ensuite on a affiché le coût amorti, on remarque que le coût amorti rencontre des augmentations et des démunissions, au début la valeur maximale qui est atteinte est 300. Ensuite il se stabilise entre 80 et 100 et cela revient au fait qu’on fait des insertions.

### On remarque aussi qu’on a beaucoup d’allocation mémoire au début ce qui revient au fait qu’on insère beaucoup au début ensuite le tôt d’allocation mémoire diminue et cela provient des suppressions qu’on effectue en parallèle des insertions comme déjà mentionné précédemment.

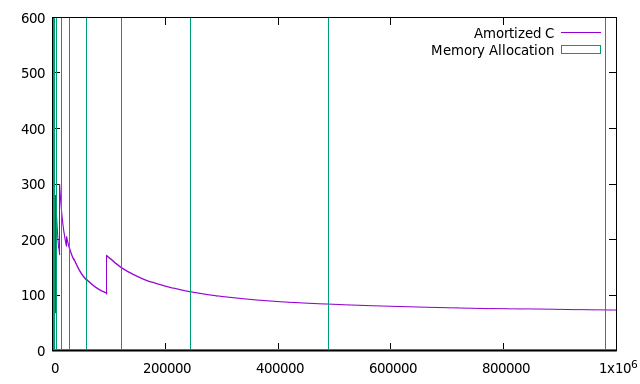


Figure  : Coût amorti et allocation mémoire pour p = 0.7

### Ensuite on a affiché l’espace mémoire inutilisé amorti et réel de l’opération insertion et suppression, on remarque que la valeur de l’espace mémoire inutilisée augmente au moment de l’allocation mémoire et diminue hors allocation mémoire. Donc si la probabilité est inférieur à 0.7 on ne fait que des insertions on a beaucoup plus de chance de faire des insertions que des suppressions ce qui augmente les chances d’allouer de la mémoire plus et de gaspiller en espace automatiquement contraient au cas ou la probabilités d’insérer et de supprimer était égaux.

### 

Figure  : Espace mémoire inutilisé en fonction du temps réel pour p = 0.7

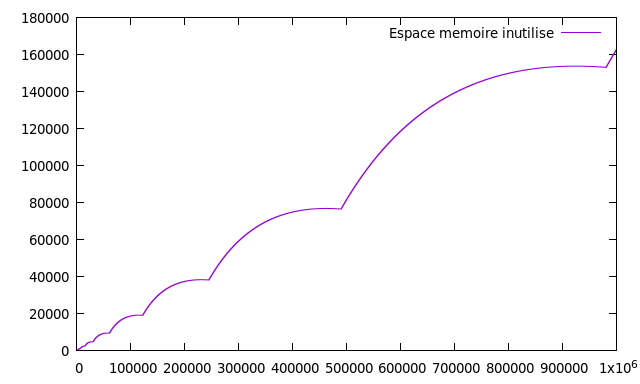


Figure  : Espace mémoire inutilisé en fonction du temps amorti pour p = 0.7

### Ensuite on a affiché le nombre de copies de valeurs effectuées amorti et réel de l’opération insertion et suppression, on remarque que le nombre de copie a augmenté mais à quand même diminuer par apport à notre cas précédent où la probabilité était 0.9. Le nombre de copies varie 100000 et 500000 dans le cout réel cela revient à ce qu’on a mentionné précédemment dans le graphe du cout amorti on avait observé qu’il y avait des allocations mémoire sur le long de nos insertions. L’allocation mémoire qui a augmenté augmente automatiquement le nombre de copies, vu qu’on copie les éléments du tableau pour les mettre dans le nouveau tableau alloué.

### 

Figure  : Nombre de copies effectuées pour p = 0.7

**Conclusion :**

### Après avoir fait plusieurs expériences sur expansion et contraction dynamiques des tables on déduit que dans le cas où la probabilité de faire plus d’insertion est plus grande que la probabilité de faire des suppressions on alloue de la mémoire surtout dans le cas où la probabilité est inférieure à 0.9 on a plus d’insertions, séquence le nombre d’allocations mémoire augmente énormément et c’est ce qu’il nous fait gaspiller de l’espace mémoire on alloue sans pour autant utilisé tout cet espace et de là aussi on a le nombre de copies qui augmente . Contrairement au cas où la probabilité de faire des insertions est inférieur à la probabilité de faire des suppressions on ne gaspille pas du mémoire vu qu’on faisant des suppressions en contracte la taille du table en la divisant par deux ce qui nous fait gagner en espace mémoire et vu qu’on ne fait pas beaucoup d’insertions on a le nombre de copies qui très petit.

### Aussi on a remarqué dans ce cas où la probabilité de faire des insertions est plus grande que la probabilité de faire des suppressions le cout amorti augmente énormément et cela revient au fait qu’en faisant de l’insertion ça nous coute plus cher ; on fait des copies lors de l’allocation mémoire puis on insère. Contrairement au cas où la probabilité de faire des insertions est inférieur à la probabilité de faire des suppressions le cout amorti est plus petit car le cout de supprimer une valeur dans le tableau nous coute pas du tout cher O (1).

**Question 5 :**

### On a modifié la stratégie de redimensionnement de la table pour utiliser taillei+1 = taillei + q taillei lors d’une extension, comme dans la dernière question du TP1 ; et taillei+1 = taillei − q taillei lors d’une contraction. On déclenche une extension au moment de l’insertion quand le facteur de remplissage de la table.

### Il faut contracter la table quand le facteur de remplissage de la table alpha = nomi/taillei = ½ vu qu’on a changé notre stratégie taillei+1 = taillei − sqrt taillei

**Expansion et contraction dynamiques des tables P = 0.5 :**

### Lorsque la probabilité P est inférieure à 0.5 on ne fait que des insertions et symétriquement lorsque la probabilité P est supérieure à 0.5 on ne fait que des suppressions

### On a affiché le cout réel de l’opération insertion et suppression, on remarque que le cout réel est carrément caché derrière la couleur verte qui représente l’allocation mémoire ; on a zoomé pour mieux voir les variations du cout réel on remarque qu’il est très perturbé il augmente jusqu’à atteindre des valeurs qui dépassent 600 surtout en effectuant cette opération avec un nombre de valeurs très grand. On remarque aussi qu’on a beaucoup d’allocation mémoire qui se suivent Il n’y a pas beaucoup d’écart entre elle ce qui revient au fait qu’on fait qu’on ne double pas la taille du tableau cette fois ci mais on la multiplie par un alpha qui est plus petit qui deux et qui vaut : 1+ 1/ sqrt(TAILLE). On parallèle on effectue des suppressions qui provoquent des contractions du tableau dynamique.la probabilité d’effectuer l’insertion est égale à la probabilité de faire une suppression ce qui explique pourquoi y a des moments où on n’a pas d’allocation mémoire.

### 

Figure  : Allocation mémoire pour p = 0.5 après modification de la stratégie

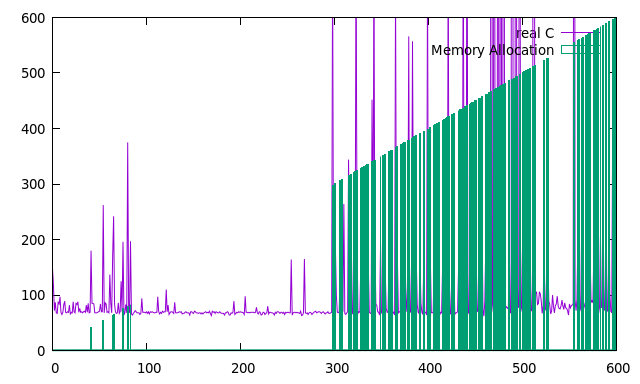


Figure  : Coût amorti et allocation mémoire pour p = 0.5 après modification de la stratégie

### Ensuite on a affiché le cout amorti, on remarque que le coût amorti rencontre des diminutions partant d’une valeur plus grande 600 il décent jusqu’à 480 pour ce stabilisé dans cet intervalle, comparé au cas où le facteur de remplissage était 2 le coût amorti ici a augmenté énormément il est passé de 100 à une valeur qui dépasse 500, un grand écart. On remarque aussi qu’on a beaucoup d’allocation mémoire au début ce qui revient au fait qu’on insère beaucoup au début ensuite le tôt d’allocation mémoire diminue et cela provient des suppressions qu’on effectue en parallèle des insertions comme déjà mentionné précédemment la probabilité d’effectuer l’insertion est égale à la probabilité de faire une suppression ce qui diminue aussi le nombre d’allocations mémoire

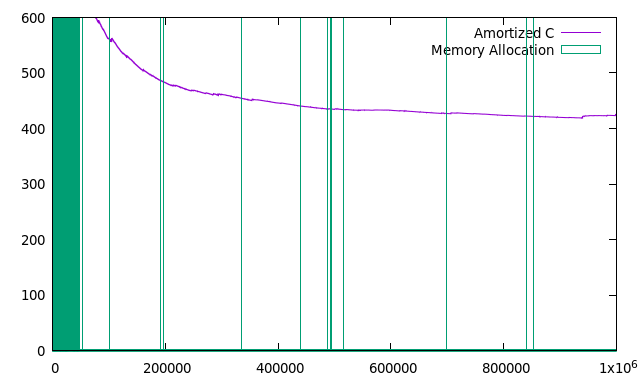


Figure  : coût amorti et allocation mémoire pour p = 0.5 près modification de la stratégie

### Ensuite on a affiché l’espace mémoire inutilisé amorti et réel de l’opération insertion et suppression, on remarque que la valeur de l’espace mémoire inutilisée augmente énormément il est passé de 0 a la valeur maximale 70000000000 ce qu’on n’a jamais vu dans les expériences précédentes tant qu’on insère tant qu’on gaspille de la mémoire vu qu’on alloue de la mémoire dans ce cas mille fois plus que les cas précédent et cela est dû au fait d’avoir adopté cette nouvelle stratégie qui a changé le facteur de remplissage. On remarque son augmentation immense dans le cas où le tableau est très grand.

### 

Figure  : Espace mémoire inutilisé pour p = 0.5 après modification de la stratégie

### Ensuite on a affiché le nombre de copies de valeurs effectuées amorti et réel de l’opération insertion et suppression, on remarque que le nombre de copie a augmenté énormément petit à petit par apport à notre cas précédent où le facteur de remplissage était 2. Le nombre de copies est passé de 20 au début jusqu’à atteindre 1000000 en cout réel et 250000 en cout amortie cela revient à ce qu’on a mentionné précédemment dans le graphe du cout amorti on avait observé qu’il y avait énormément d’allocation mémoire sur le long de nos insertions. L’allocation mémoire qui a augmenté augmente automatiquement le nombre de copies, vu qu’on copie les éléments du tableau pour les mettre dans le nouveau tableau alloué.

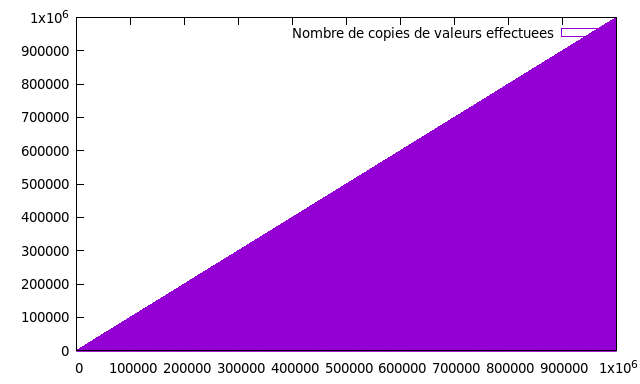


Figure  : Nombre de copies effectuées pour p = 0.5 après modification de la stratégie

### Expansion et contraction dynamiques des tables P = 0.9 :

### Lorsque la probabilité P est inférieure à 0.9 on ne fait que des insertions et symétriquement lorsque la probabilité P est supérieure à 0.9 on ne fait que des suppressions

### On a affiché le coût réel de l’opération insertion et suppression, on remarque que le coût réel est carrément caché derrière la couleur verte qui représente l’allocation mémoire comme notre précédent ; on a zoomé pour mieux voir les variations du coût réel on remarque qu’il est très perturbé il augmente jusqu’à atteindre des valeurs qui dépassent 600. On remarque aussi qu’on a beaucoup d’allocation mémoire beaucoup plus que le cas précédent vu que là la probabilité d’insertions et supérieur à la probabilité de suppressions donc on a davantage d’insertions ce qui provoque énormément d’allocations mémoire qui se suivent, Il n’y a pas beaucoup d’écart entre elle ce qui revient et le facteur de remplissage qu’on a changé a joué un rôle important dans l’augmentation des allocations mémoire.

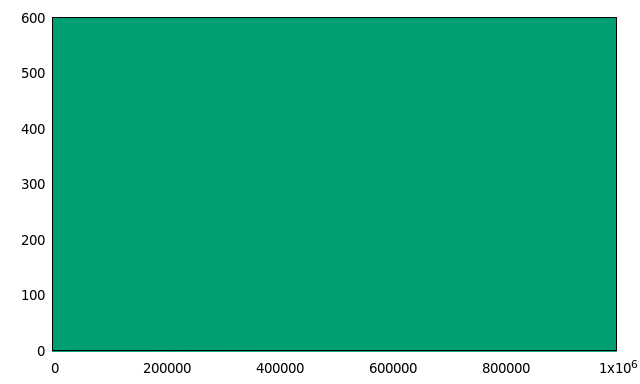


Figure  : allocation mémoire pour p = 0.9 après modification de la stratégie

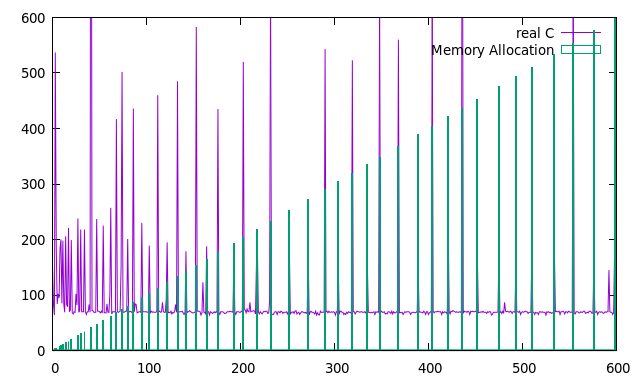


Figure  : Coût réel et allocation mémoire pour p = 0.9 après modification de la stratégie

### Ensuite on a affiché le coût amorti, on remarque que le coût amorti est carrément caché comme notre cas précédent à cause de la couleur verte qui représente l’allocation mémoire. La probabilité d’effectuer des suppressions est très faible donc c’est rare de rencontrer des contractions de la taille du tableau. Dans ce cas le facteur de remplissage n’a pas changé la valeur du coup amorti énormément vu que c’est presque la même valeur qu’on a eue dans le cas où le facteur de remplissage était égal à 2 et que la probabilité d’insertion était inférieure à 0.9.

### 

Figure  : allocation mémoire pour p = 0.9 après modification de la stratégie

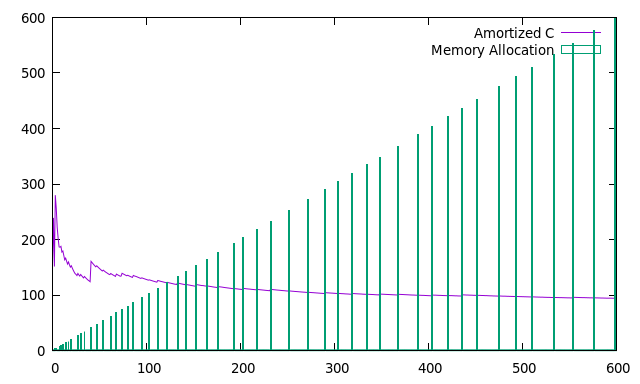


Figure  : coût amorti et allocation mémoire pour p = 0.9 après modification de la stratégie

### Ensuite on a affiché l’espace mémoire inutilisé, on remarque que la valeur de l’espace mémoire inutilisée augmente énormément il est passé de 0 a la valeur maximale 300000 ce qui plus grand dans le cas où le facteur de remplissage était égal à 2. Tant qu’on insère tant qu’on gaspille du mémoire vu qu’on alloue de la mémoire dans ce cas mille fois plus que les cas précédents et cela est pareil pour le réel et l’amorti

### 

Figure  : Espace mémoire inutilisé pour p = 0.9 après modification de la stratégie

### Ensuite on a affiché le nombre de copies de valeurs effectuées, on remarque que le nombre de copie a augmenté énormément par apport à notre cas précédent où le facteur de remplissage était 2. Le nombre de copies est passé de 20 au début jusqu’à atteindre 1000000 en cout réel et cela revient à la même raison qu’on a mentionné précédemment où les probabilités d’insertion et de suppression était égaux car même ici dans le graphe du cout amorti on avait observé qu’il y avait énormément d’allocation mémoire sur le long de nos insertions.

### 

Figure  : Nombre de copies effectuées pour p = 0.9 après modification de la stratégie

**Insertions et suppressions avec après modification de la stratégie de redimensionnement et avec p = 0.1:**

On remarque ici que le coût amorti est légèrement plus élevé que le cas où on multiplie par 2, le nombre de copies aussi a augmenté un peu car ici on multiplie par un nombre plus petit que 2 ce qui fait que le tableau se rempli plus rapidement que dans le cas précédent.

Mais en le comparant au deux cas précédents où p= 0.9 ou p = 0.5 après modification de la stratégie de redimensionnement on remarque que le nombre de copies a énormément baissé voire incomparable et cela car la probabilité d’insertions est très petite comparée à celle de la suppression.

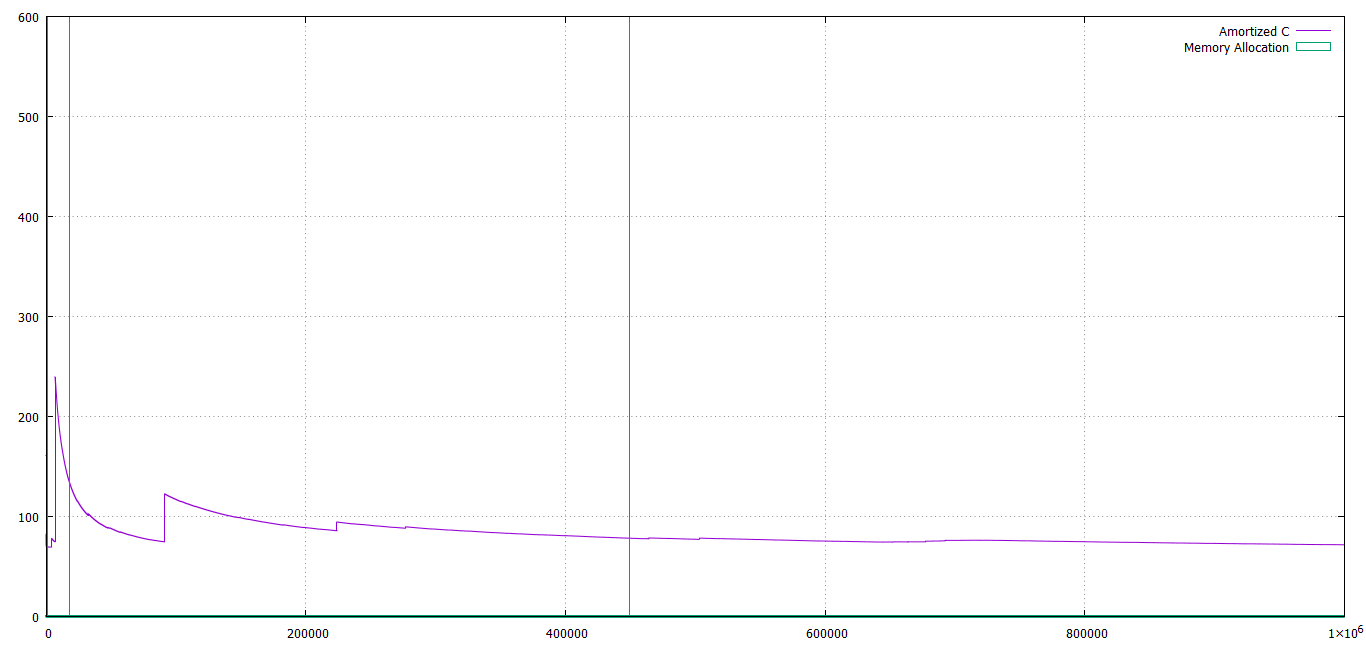


Figure  : coût amorti et allocation mémoire pour p = 0.1 après modification de la stratégie

On remarque que le coût réel aussi a baissé comparé aux expériences précédentes, néanmoins il reste perturbé comparé au coût amorti et cela est du au fait que le coût amorti dépend de la structure, donc il augmente fortement quand on fait une allocation ou une contraction du tableau.

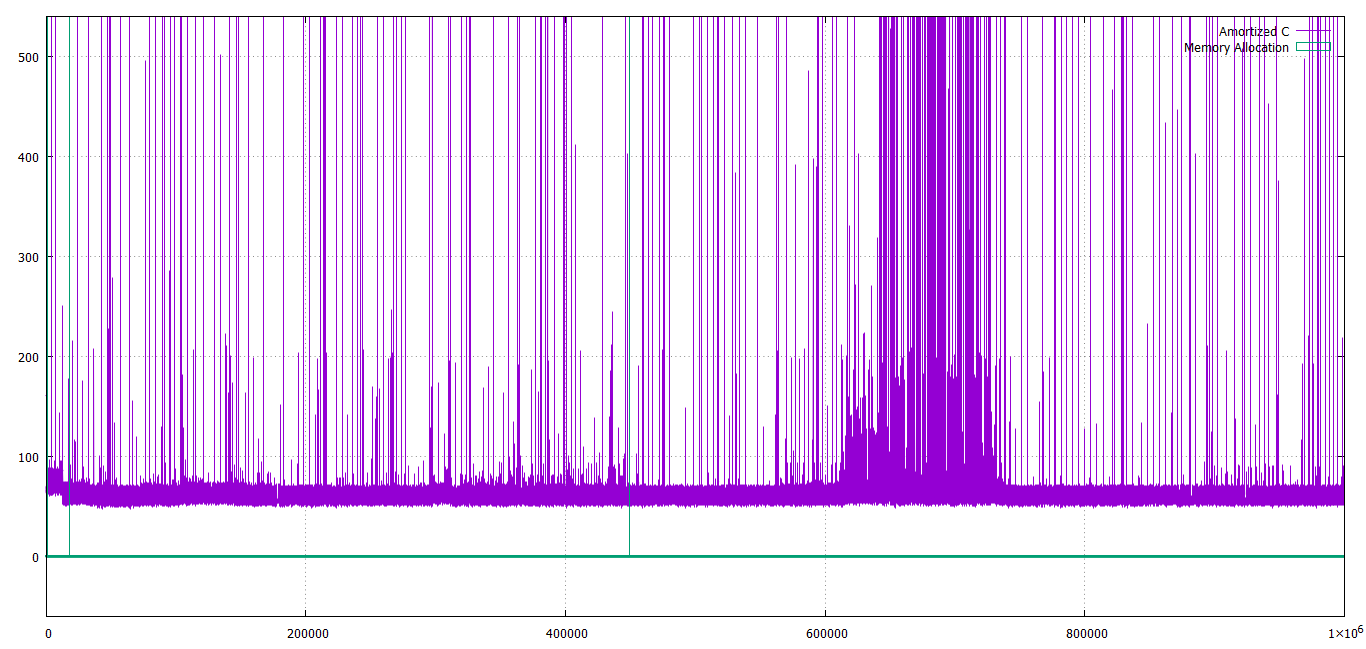


Figure  : coût réel et allocation mémoire pour p = 0.1

Pour ce qui est de l’espace mémoire inutilisé, il ne dépasse même pas 5 et est plus petit que celui des expériences précédentes. La raison de cette chute est la probabilité de suppression qui est très élevée et qui fait que le tableau est souvent contracté, ainsi il ne contient pas beaucoup de cases vides.

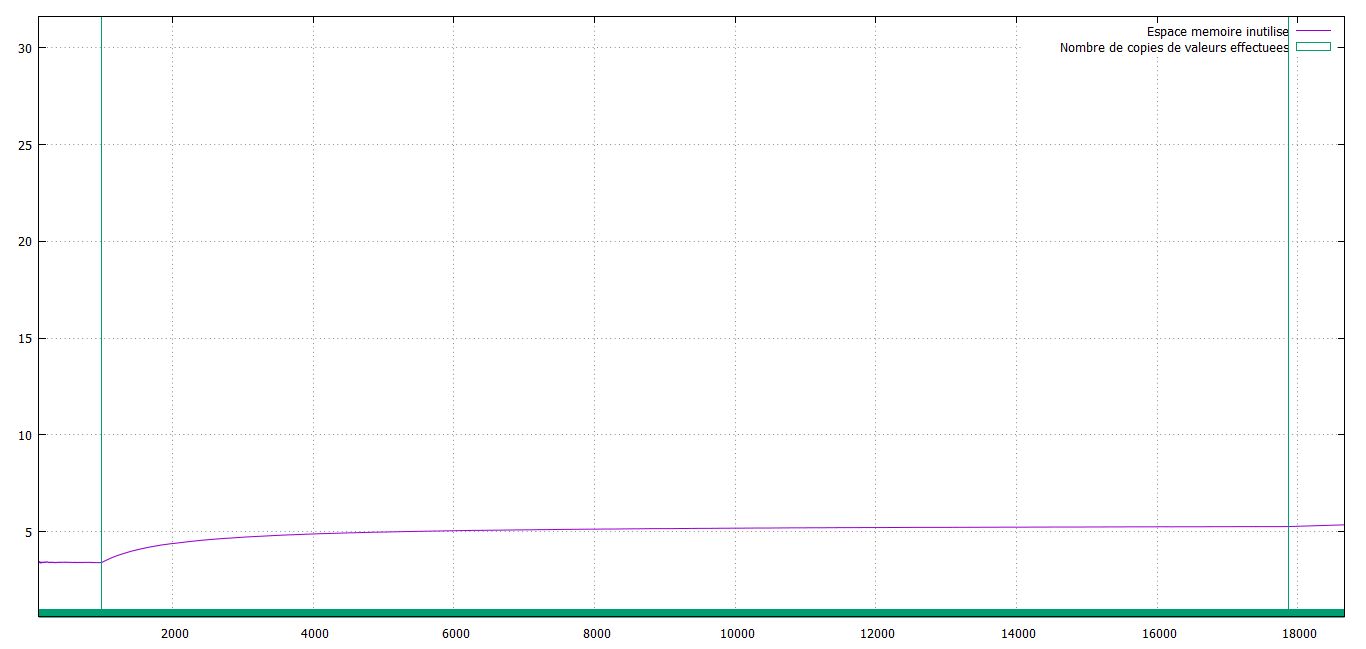


Figure  : Espace mémoire inutilisé pour p = 0.1

**Question 6 :**

La valeur de p pour laquelle cette stratégie semble mieux fonctionner est P = 0.5 par ce que pour ce cas la probabilité d’effectuer une suppression est égale à la probabilité d’effectuer une insertion ce qui équilibre nos résultats obtenus contrairement aux autre cas qui ne sont pas équilibrés parce que lorsque c’est à inférieur à 0.5 on fait plus de suppressions et lorsque c’est supérieur à 0.5 on fait plus d’insertions ce qui nous gaspille énormément de mémoire et augmente le nombre de copie .