**Rapport de comparaison (chargement bulk—chargement insertion standard) :**

Apres réalisation d’un TP qui consiste à faire un chargement initial d’un B+arbre (à partir d’un fichier TOF/LOF) par deux méthodes, on va établir une comparaison entre les performances de chacune :

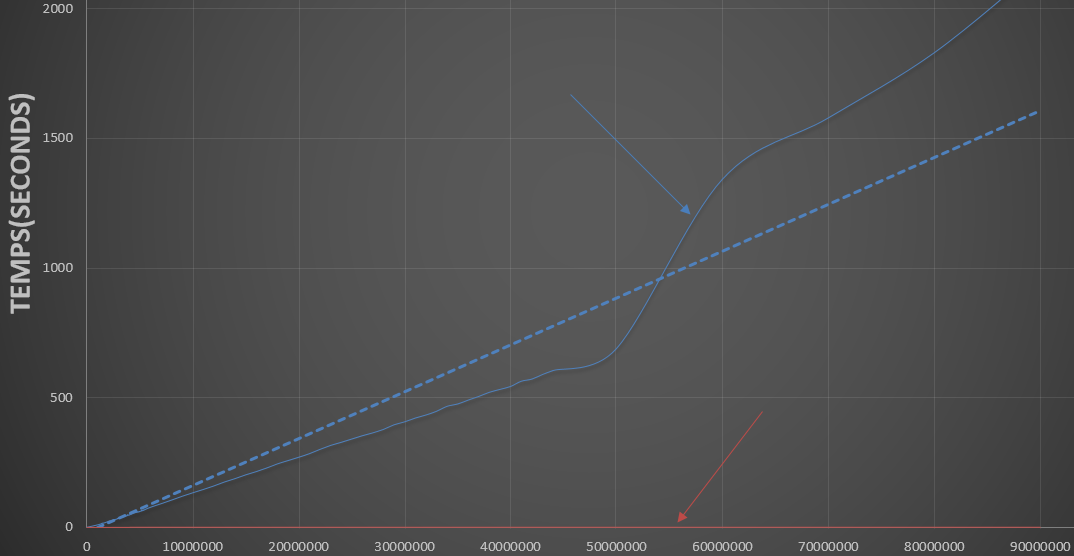
Les deux méthodes concernées sont :

1. Bulk loading.
2. Insertion standard.

Il s’agit dans ce TP de B+arbre d’ordre 127 pour l’index et 253 valeurs peuvent être insérées dans les feuilles. Dans notre étude, on prend en considération les grandeurs suivantes :

**Temps d’exécution :**

La méthode « bulk loading » dépasse de loin « l’insertion standard » en ce qui s’agit de performance de temps, on parle de millièmes secondes pour la première méthode quelle que soit le nombre de valeurs à insérer, alors que pour la deuxième méthode, l’initialisation prend 0.9s pour 100000 valeurs ce qui est 450 fois plus grands que celui nécessaire pour initialiser l’arbre par la première méthode(2ms) ; pour 1000000 valeurs, elle prend 9.91s 900 fois plus grands que celui nécessaire pour initialiser l’arbre par la première méthode(11ms) ; elle prend aussi près de 30min pour 100000000 valeurs, 1241 fois plus grands que celui nécessaire pour initialiser l’arbre par la première méthode(1.45s) .Cette grande différence est due d’une part à la recherche faite ‘inutilement’ par l’algorithme d’insertion standard vue que les valeurs à insérer sont ordonnées ( ceci provoque des lectures et des écritures successives du même bloc, et souvent , un parcours de la même branche, ce qui est couteux en terme de temps).



Temps=10-5.nbrVal [ms]

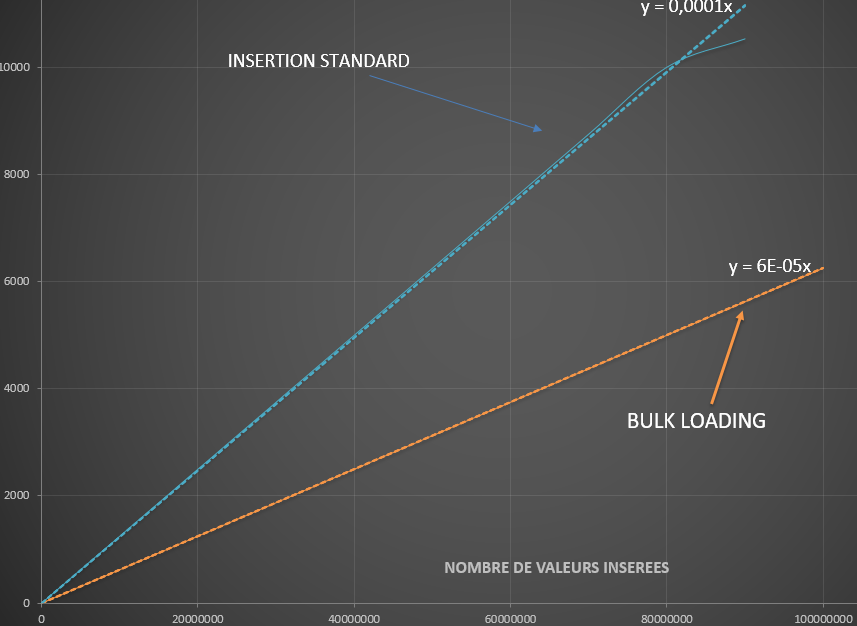
Mais malgré l’exploitation de de la nature des valeurs à insérer, le bulk loading reste plus performent en temps d’exécution à part que la différence est moins flagrante comme on peut le constater dans la représentation graphique ci-dessous

**Espace occupé par le B+arbre :**

Le fichier B+arbre a naturellement une plus grande taille que le fichier original vue les bloques en plus de l’index et le taux de chargement des blocs du fichier qui est différent. Pour celui créer par la première méthode, il y a moins de feuilles que celui créer par la deuxième méthode à cause des éclatements causés par cette dernière, par conséquent :

Nombre de bloques en plus que le fichier original :

Il s’agit des blocs nécessaires pour construire l’index. On remarque que la première méthode requis plus de bloques que la deuxième méthode, ça se justifie par le fait qu’il y a plus de feuilles dans le B+arbre construit par la deuxième méthode que par la première. Par exemple : pour 1000000, il y a respectivement : 62 et 124 de blocs en plus pour la première et la deuxième méthode puisqu’il y a 3938 et 7877 feuilles. Et pour 100000000 valeurs, il y a respectivement : 6247 et 10526 de blocs en plus pour la première et la deuxième méthode puisqu’il y a 393701 et 709385 feuilles.



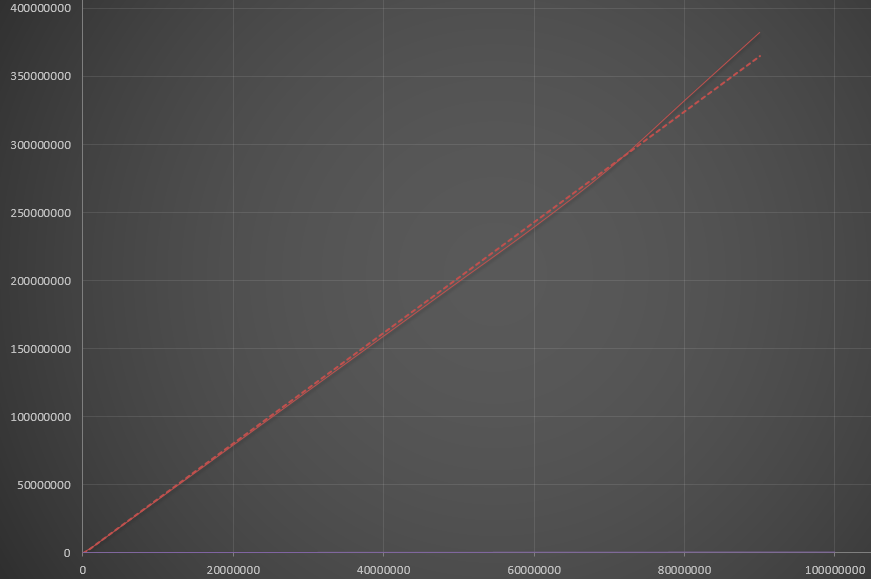
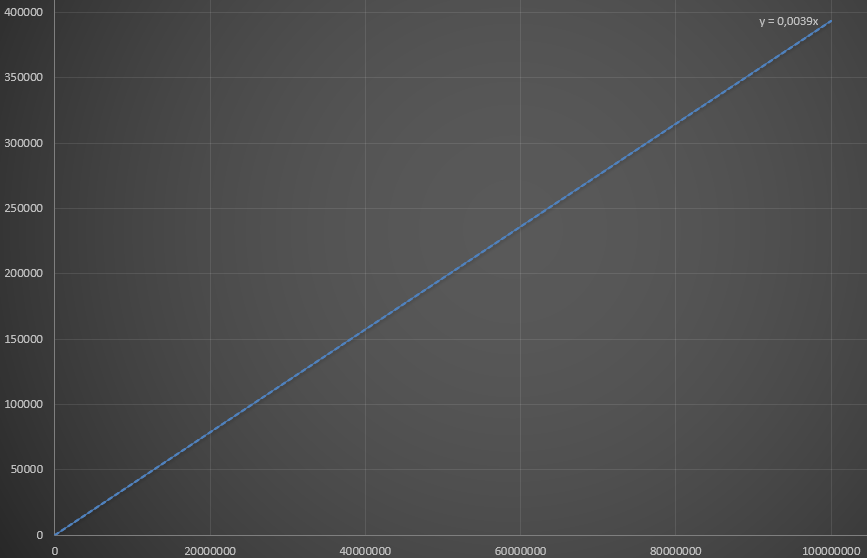
Nombre de valeurs en plus que le fichier original :

On remarque que le taux de chargement de l’index des deux arbres est à 50%, ce qui nous amène à estimer le nombre de valeurs en plus par rapport au fichier d’origine : (nombre de bloques présents dans l’index) \*127 /2. Il y a donc plus de valeurs en plus dans le B+arbre construit par la deuxième méthode que par la première puisqu’ils ont le même taux de chargement (50%) et il y a plus de blocs pour la 2ieme méthode comme cité précédemment. Par exemple : pour 1000000, il y a respectivement 3815 et 7876 valeurs en plus pour la première et la deuxième méthode ; et pour 100000000 valeurs, il y a respectivement 396684 et 668401 valeurs en plus pour la première et la deuxième méthode.

**Les accès disque :**

Le nombre d’accès disque effectues lors de l’utilisation de la méthode bulk loading est minime par rapport à celui effectues lors de l’utilisation de la méthode d’insertion standard. Même constations pour le temps d’exécution, il faut prendre en considération la recherche effectuer par le 2ieme algorithme, mais ceci ne fait que rapprocher le nombre d’accès effectues par les deux méthodes tout en gardant le bulk loading en avant puisqu’il reste toujours plus optimisé en ce qui est nombre d’accès.

En lecture :

La recherche faite par l’algorithme d’insertion standard provoque des lectures du même bloc, et plus encore, une lecture de la même branche plusieurs fois de suite ce qui fait exploser le nombre de lecture par rapport à celui de l’autre méthode

bulk

On peut constater la réduction de la différence de nombre de lecture entre les deux méthodes après avoir tirer profit de l’ordre des valeurs à insérer lors de l’utilisation de l’insertion standard :

En écriture :

Pour les même raisons, l’algorithme d’insertion standard effectue plus d’écritures (une écriture pour chaque valeur insérée plus les écritures des nœuds éclatés), il y a analogie par rapport au résultats statistiques et graphes précèdent. De même si on retarde l’écriture pour l’insertion standard.

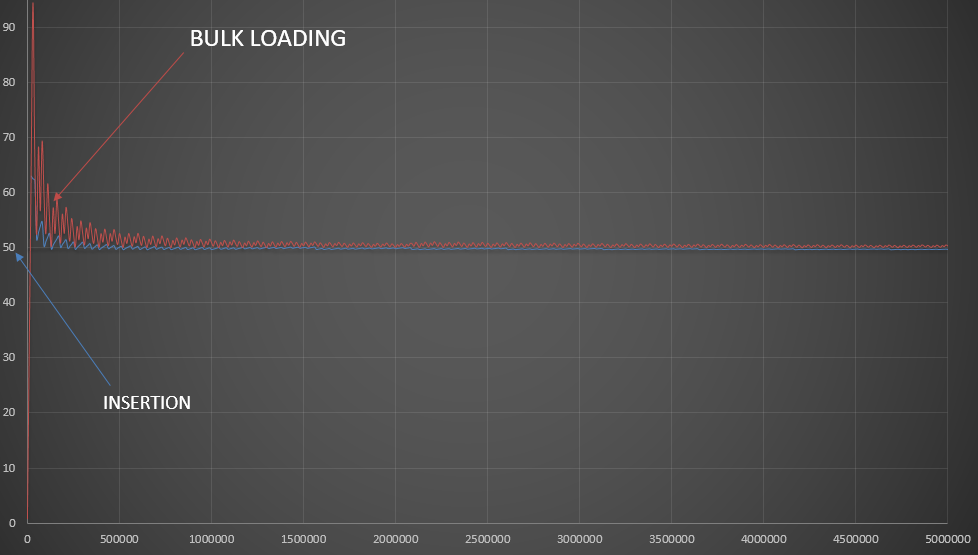
**Taux de chargement :**

Il y a toujours dominance de nombre de feuilles sur le nombre de nœuds en index comme constaté dans les graphes, ce qui explique la tendance de la moyenne de chargement du des nœuds de l’arbre à la moyenne de chargement des feuilles de ce même arbre : 100% pour bulk loading ; 50% insertion standard (à cause des éclatements des feuilles). On peut constater que pour les petits nombres de valeurs à insérer le taux de chargement de l’arbre construit par la méthode d’insertion standard est plus élevée, c’est par ce que le nombre d’éclatements est minime.



Concernant l’index :

Pour un nombre de valeurs à insérer inférieur à 500000, le taux de chargement est entre 100% et 70% (le nombre d’éclatements est minime car il y a de place pour les valeurs), après le taux tend vers 50% pour les deux méthodes



Concernant le fichier (liste de feuille) :

Bulk loading manipule les blocs alors que la méthode d’insertion standard manipule les valeurs, ce qui fait que le taux de chargement des feuilles dans l’arbre construit par la 1iere méthode est de 100% alors l’autre est de 50%

**Profondeur de l’arbre :**

Pour un petits nombre de valeur à insérer, les arbre des deux méthodes partage la même profondeur( puisque le taux de chargement est rapproché entre les deux), mais quand passe à la hauteur 3, les feuilles des arbre de la 1iere méthode peuvent supporter 254\*1273 alors que s feuilles des arbre de la 2ieme méthode peuvent supporter la moitié 254\*1273,cequi explique que les arbres de la 2ieme méthode passe à la hauteur 4 avant ceux de l’autre méthode, et même constations pour le deuxième saut ( de 4 à 5) des arbres de la 2ieme méthode.



**Conclusion :**

Globalement, la méthode bulk loading est plus performante (temps, esthétique de l’arbre). Reste à vérifier ces performances pour des valeurs aléatoires, puisque la puissance de bulk loading est de tirer profit de l’ordre des valeurs à insérer lors de l’initialisation. Et pour les insertions ultérieures dans l’arbre, la méthode 2 nous fait éviter les éclatements ou en moins les retarder, alors que la 1iere méthode nous assure un très court chemin vers la position où il faut insérer (profondeur minimale) avec un nombre limité d’éclatements puisque l’index est chargé à 50%.