Document d'évaluation

I Choix de la structure de données

Après avoir considéré tous les paramètres de notre problème, et analysé la situation, nous avons choisi d'utiliser une structure de données statique et non dynamique.
En effet, une structure statique peut paraître contraignante à première vue puisque la mémoire est allouée dès le début et non dynamiquement, c'est à dire au fur et à mesure que l'on rajoute des données.
En plus, cela nous apporte un gain en performance par rapport à une liste chaînée, ce qui nous parait plus important au vu de la petite «perte» mémoire due à l'allocation statique du tableau.

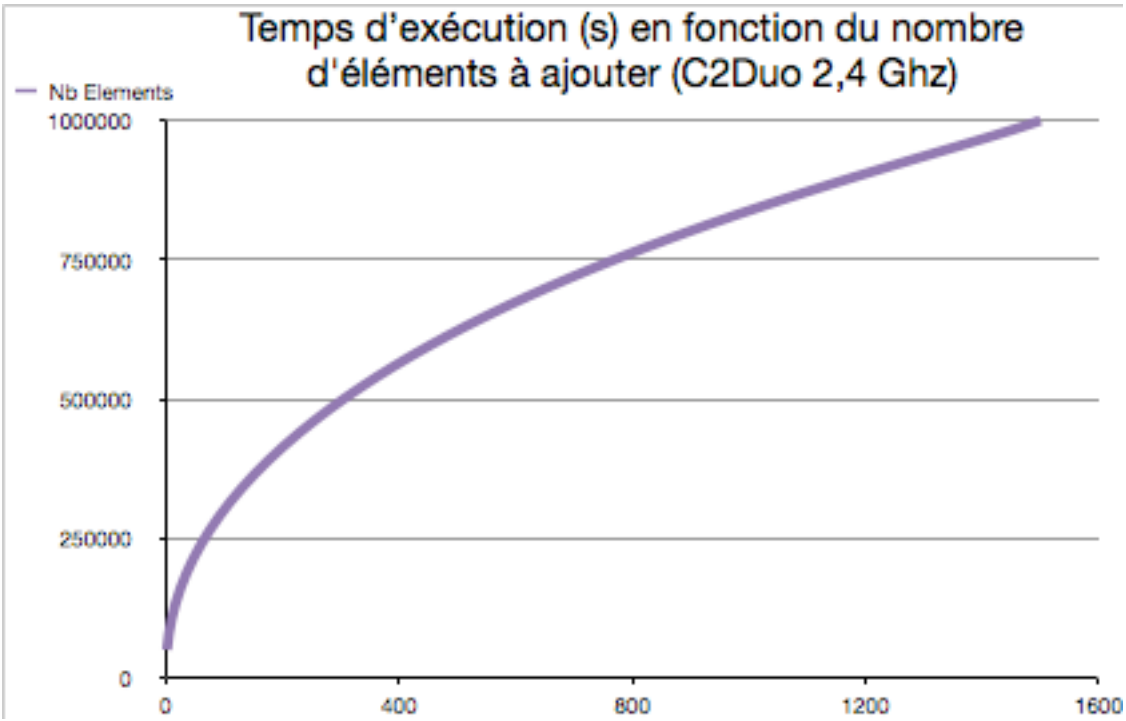
Un autre choix aurait pu être envisagé, mais il présente aussi des inconvénients :
Une liste chainée est performante lors de l'ajout d'un élément, mais est moins rapide pour accéder à un élément autre que le premier ou le dernier. Compte tenu du cahier des charges, il parait plus judicieux de privilégier la manipulation des données par rapport à l'ajout qui ne se fait qu'une seule fois.

L'inconvénient majeur de la liste chainée se trouve pour la méthode FindDMP, nerf de la classe. En effet, pour cette méthode, nous avons besoin de chercher à partir d'une position n et non à partir de la première position. Un tableau est donc plus adapté à cette méthode la.
Autre raison, on sait que la limite maximale pour le tableau est fixée à 1 million d'éléments. Or 1 million d'éléments ne font qu'environ 8 mb (une structure Mesure ne prend que 4bits (sur un processeur 32bits) et un char* ne prenant en moyenne que 6 bits). Cela n'est pas énorme et le gain en mémoire n'est donc pas important si on utilise une liste chainée, surtout sachant qu'un ordinateur actuel possède environ 4Gb de ram. Nous avons plutôt préféré privilégier les performances aux ressources mémoires dans ce cas la.

La seule approche qui aurait accéléré findDMP est l'utilisation d'une table de hashage, ce qui d'après les objectifs de ce TP (il doit rester simple et nous aider à prendre en main l'IDE Eclipse et la programmation OO en C++) n'était pas approprié pour ce cas là.

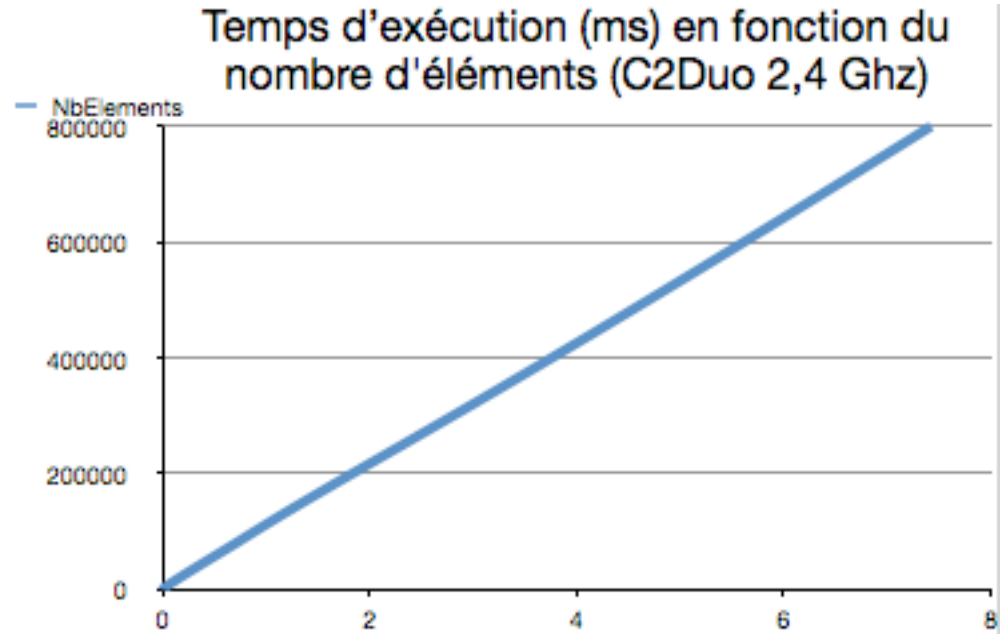
II Performances

Certains méthodes prenaient beaucoup plus de temps d'exécution que les autres. Il était intéressant d'observer leur évolution dans le temps en fonction du nombre d'éléments de la cartographie.
Le premier graphe représentent le temps d'exécution de la méthode Add en fonction du nombre d'éléments à ajouter :



On remarque que le graphe à une évolution en *log(n)* : plus on ajoute d'éléments et plus la méthode met de temps pour rajouter un point. Ceci est tout à fait normal, car à chaque fois il faut vérifier point par point si il n'existe pas déjà avant de l'ajouter, et plus on a de points plus la méthode met de temps pour parcourir tous les points.

Le graphe suivant représente le temps d'exécution du test fonctionnel 6 (Test qui sert à évaluer les performances de la méthode FindDMP en fonction du nombre d'éléments de la cartographie) :



On a un temps d'exécution proportionnel au nombre d'éléments. Ce graphe correspond au pire des cas, c'est à dire le cas où l'on recherche du début (index 0), l'élément que l'on recherche a une matière unique et c'est aussi le dernier ajouté.
Un moyen d'aller plus rapidement aurait été l'utilisation d'une table de hachage par exemple.