Structures de données : comparaisons de structures

Table des matières

Table des figures

Liste des tableaux

1 Méthode de développement

1.1 Structure du projet

Voici comment sont organisés les fichiers du projet :

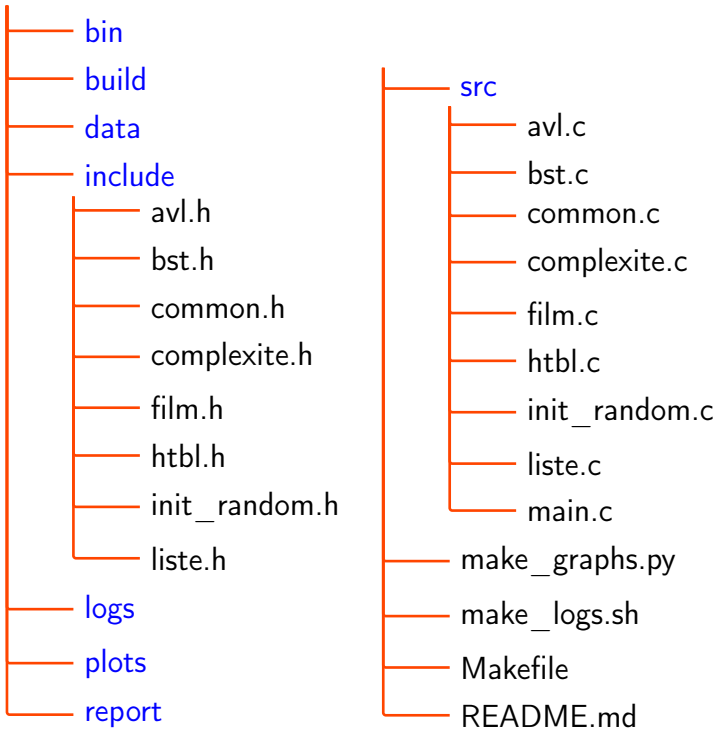

FIGURE 1 – Arborescence du projet

Et voici à quoi correspondent les fichiers :

Description	Fichiers
Implémentation des **tables de hachage**	`htbl.h`, `htbl.c`
Implémentation des **listes** (pour `htbl`)	`liste.h`, `liste.c`
Implémentation des **ABR**	`bst.h`, `bst.c`
Implémentation des **AVL**	`avl.h`, `avl.c`
Test d'une structure donnée pour un fichier test donné	`main.c`
Génération de **logs**	`make_logs.sh`
Génération des **graphes**	`make_graphs.py`

TABLE 1 – Description des fichiers principaux

1.2 Intégration

Chaque structure a été implémentée dans le TP correspondant, en utilisant le type `int`. Il a été nécessaire de faire quelques modifications afin de changer vers le type `t_film` : changement dans la définition, et dans les fonctions de recherche (utilisation de `equals` pour le test d'égalité, de `le` pour $\leqslant$).

1.3 Génération des résultats

Pour créer les graphes, il suffit de faire `make graphs` (ou `make graphs SEARCH_NB=[nombre de recherches à faire]`) : cela va compiler les sources en `C` ; lancer `make_logs.sh` sur le résultat de cette compilation ; puis lancer `make_graphs.py` sur les logs, ce qui va enregistrer les graphes dans `plots`.

2 Résultats

2.1 Description des tests

Les propriétés suivantes ont été mesurées : **temps**, **nombre d'opérations**, et **mémoire allouée**. Pour les arbres, la hauteur a aussi été mesurée.

Les opérations suivantes ont été testées : l'**insertion** et la **recherche**.

Chaque opération a été testée une fois par structure et par taille des données d'entrée.

Étant donné un fichier de test de taille n :

• **Insertion** : les compteurs sont initialisés à zéro avant la création de la structure vide. Ils sont relevés après l'insertion des n éléments dans la structure.

• **Recherche** : on définit un nombre de recherches à faire `nb_search`. On initialise les compteurs à zéro. Dans une boucle répétée `nb_search` fois, on choisit aléatoirement un film, puis on le recherche dans la structure. On relève les compteurs après la boucle, et on divise leur résultats par `nb_search`.

2.2 Comparaison des trois structures

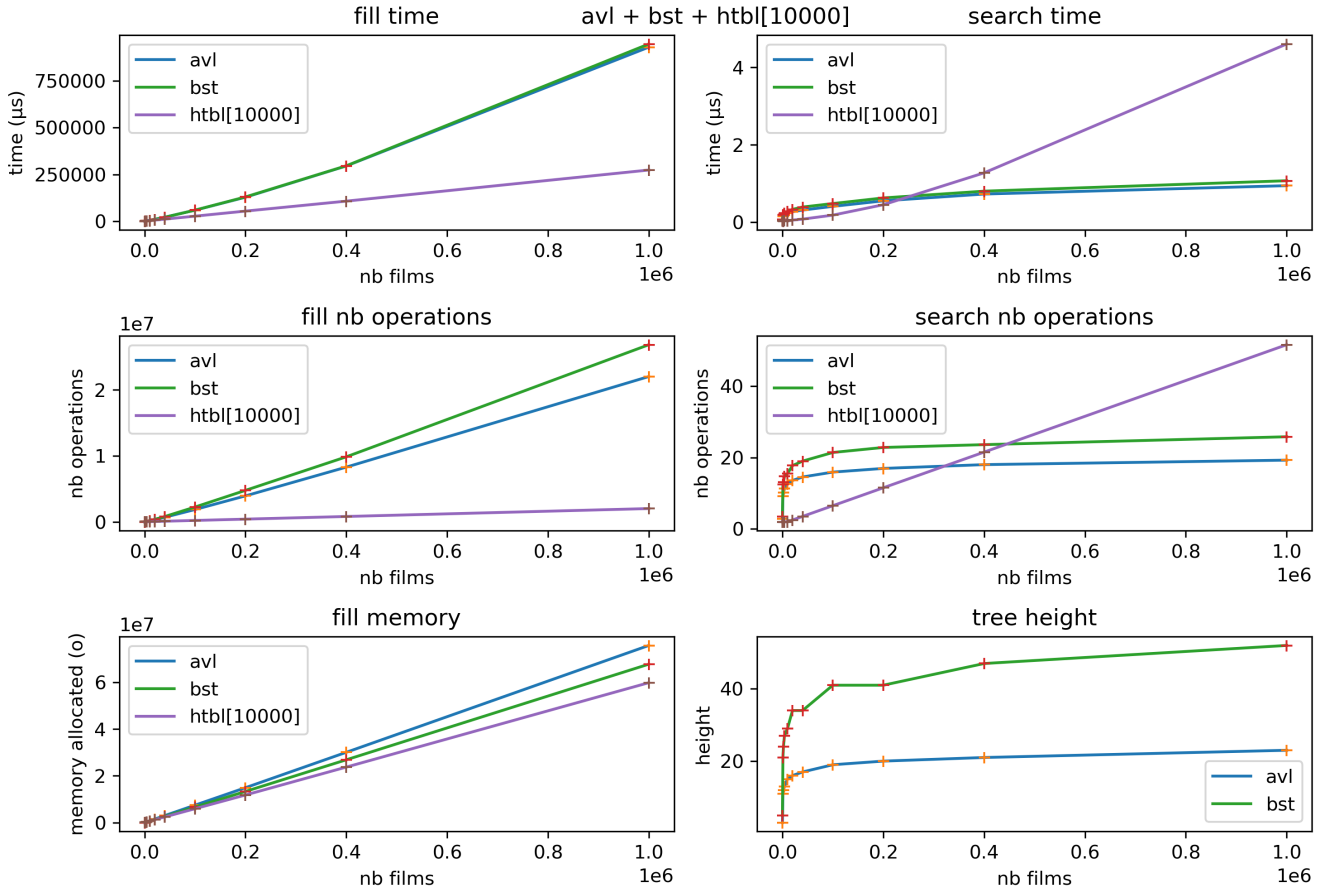

FIGURE 2 – Résultats des mesures pour les trois structures, avec `nb_search=500 000`.

On remarque que les arbres (AVL et ABR=BST) ont des comportements similaires et se distinguent de la table de hachage (htbl).

• **Insertion** (*fill*) : comme on ne regarde pas l'insertion d'un élément dans une structure de taille n, mais l'insertion des n éléments dans la structure, c'est linéaire pour toutes les structures.

On remarque que c'est plus rapide pour les tables de hachage : en effet, il y a moins d'opérations à faire lors de l'ajout d'un élément.

Au niveau de la mémoire : c'est presque la même chose pour les trois, comme on stocke les mêmes données.

• **Recherche** (*search*) : la recherche d'un élément dans une table de hachage semble être en $\mathcal{O}(n)$ (avec n le nombre d'éléments dans la table), ce qui correspond à la théorie.

Pour les arbres, cela semble être en $\mathcal{O}(\log n)$, ce qui correspond plus ou moins à la théorie (voir en détail dans 2.3).

2.3 Comparaison des structures arborescentes

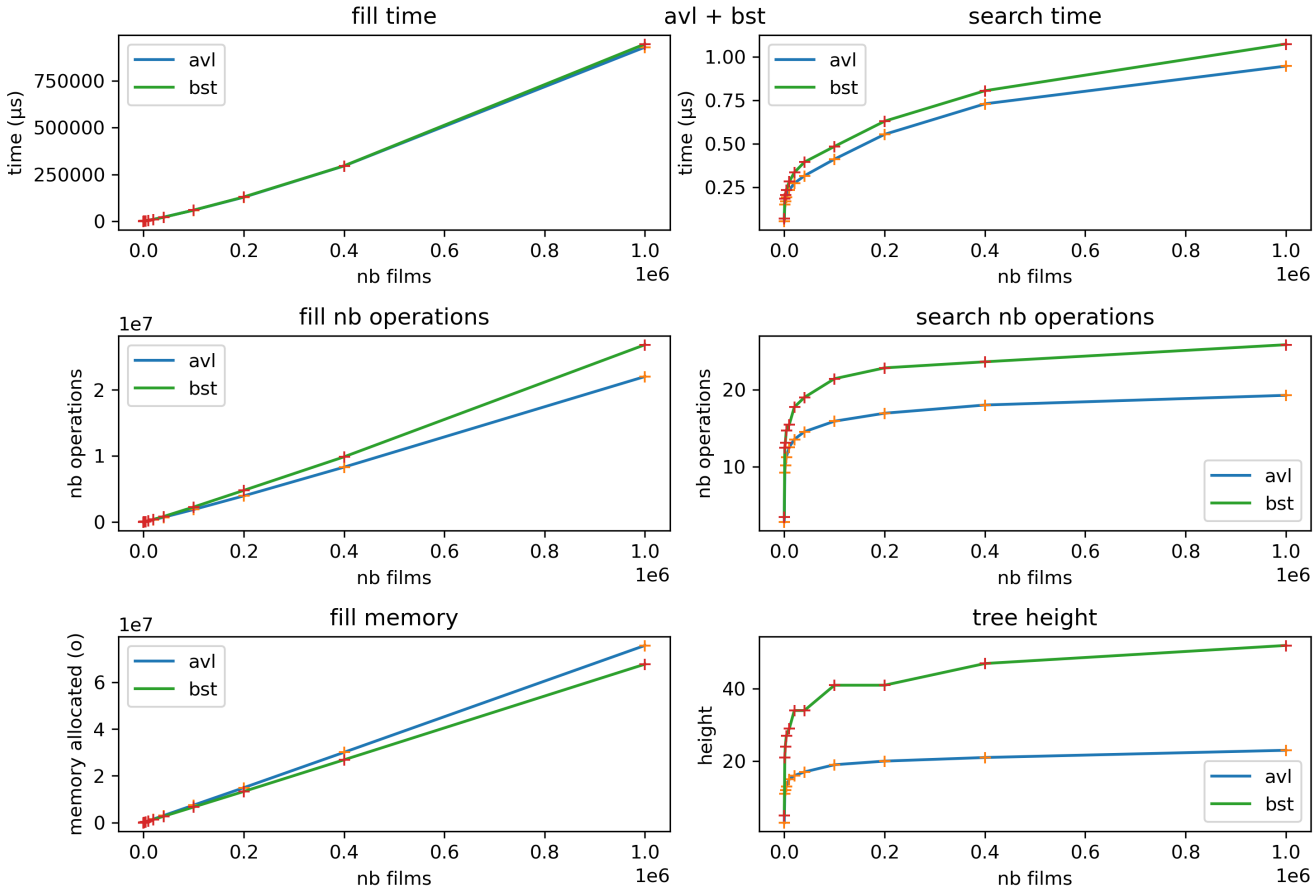

FIGURE 3 – Résultats pour les arbres (AVL et ABR), avec `nb_search=500 000`

- **Insertion** : *cf* 2.2.
- **Recherche** : la recherche est plus rapide sur les avl, ce qui corrobore donc bien la théorie.

On remarque aussi que la hauteur des ABR est en $\mathcal{O}(\log n)$ (avec n le nombre d'éléments dans l'arbre) avec ces données.

Mais la hauteur des ABR est de l'ordre de deux fois plus que celle des AVL, pour un même nombre de données. Cela se traduit bien par une recherche plus rapide pour les AVL.

On peut regarder plus en détail, en faisant la différence entre les bst et les avl :

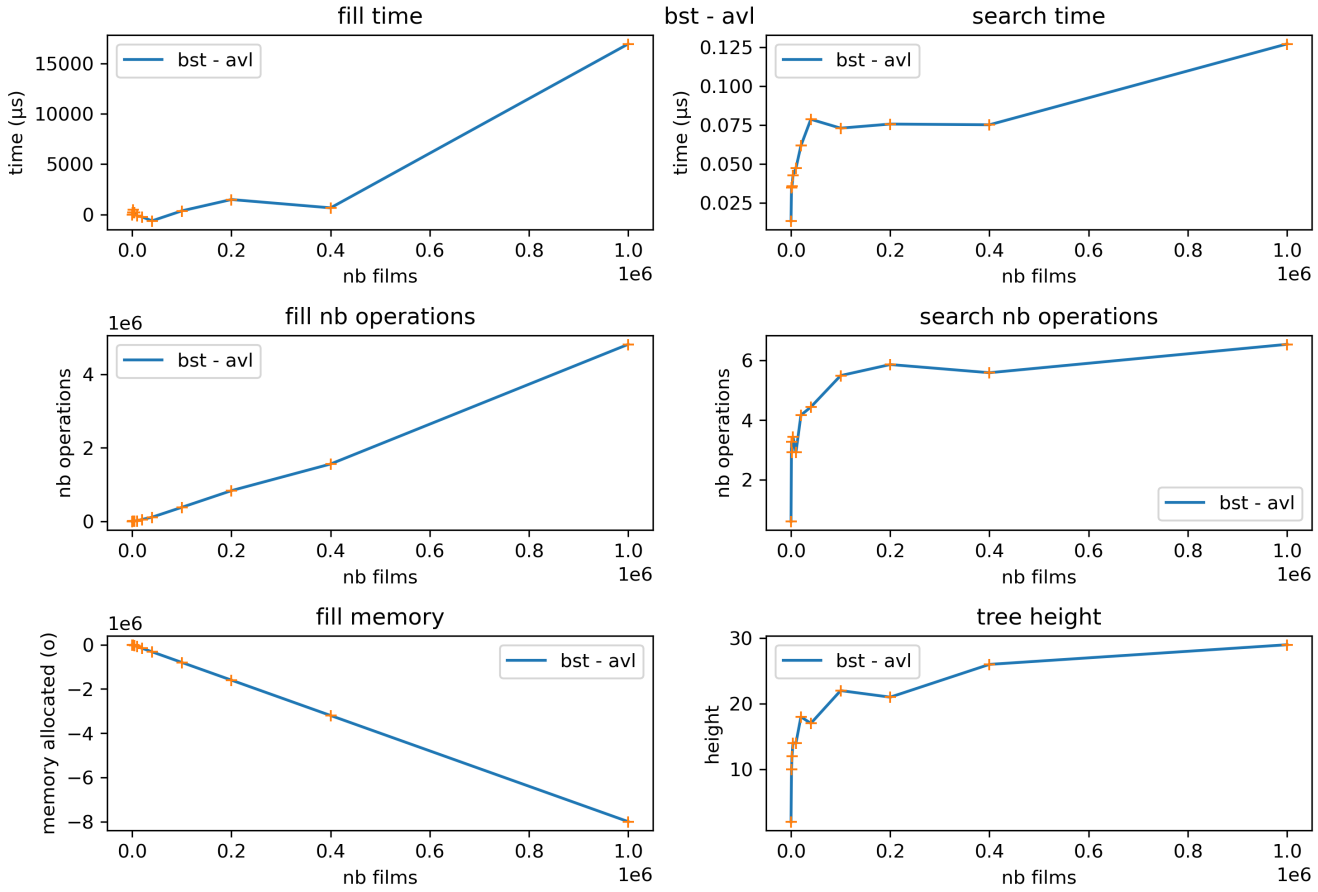

FIGURE 4 – Différence entre ABR et AVL (`nb_search=500 000`)

On remarque que la différence entre les hauteurs est même en $\mathcal{O}(\log n)$.

2.4 Comparaison des tables de hachage

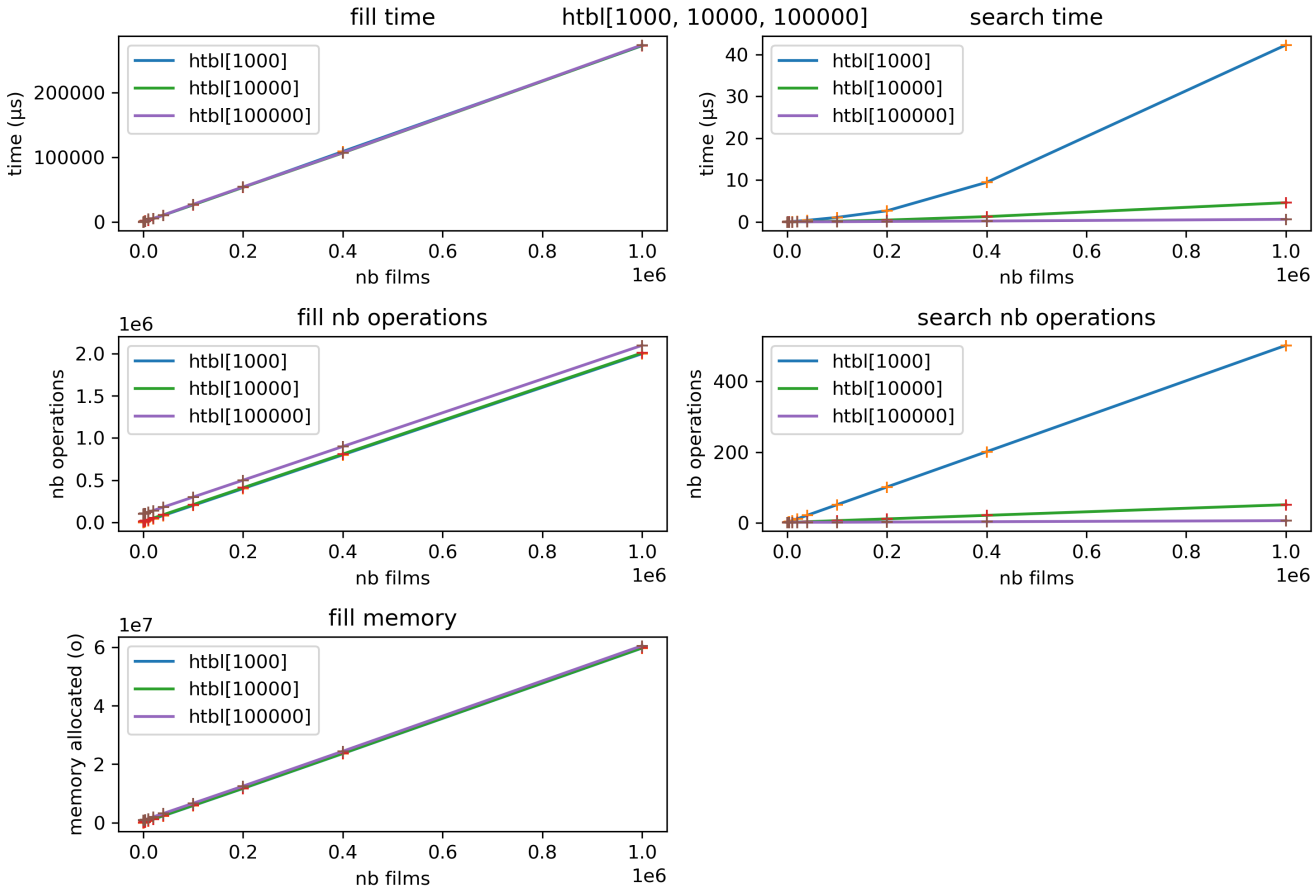

FIGURE 5 – Résultats pour les tables de hachage avec différentes tailles (`nb_search=500 000`)

Tout est linéaire en le nombre de données dans la structure, mais il semblerait qu'une taille plus grande donne de meilleurs résultats. En effet, on aura moins de collisions, mais la table prendra plus de place.

On peut tracer les courbes en fonction de la taille de la table au lieu du nombre de données :

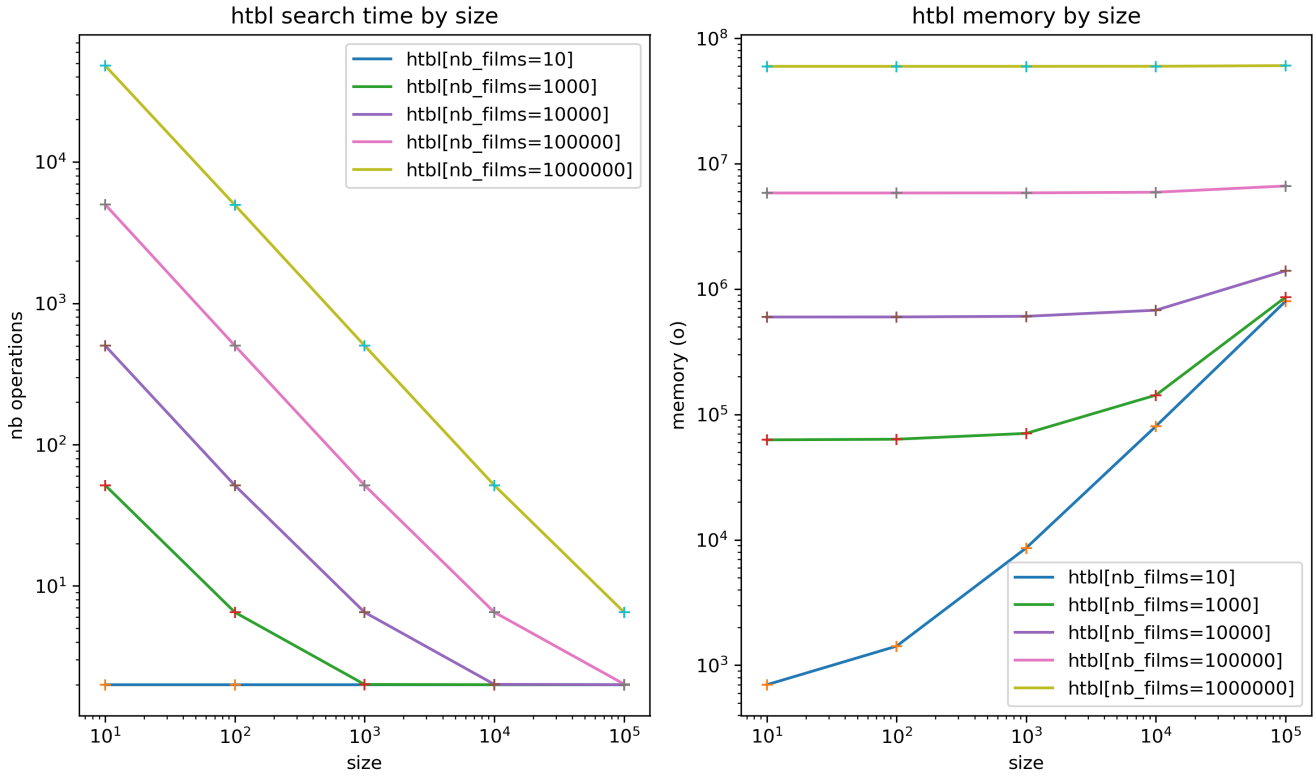

FIGURE 6 – Nombre d'opérations et mémoire des tables de hachage en fonction de la taille (échelle logarithmique)

Si on a peu de données (courbe `htbl[nb_films=10]` par exemple), alors augmenter la taille ne rend pas pas la recherche plus rapide, mais prend de la place pour rien.

Si on a beaucoup de données, (courbe `htbl[nb_films=100000]` par exemple), alors augmenter la taille de la table de hachage permet de réduire les collisions et ainsi d'augmenter la vitesse de recherche. On remarque que dans ce cas, la taille n'influe pas sur la mémoire allouée (car il n'y a jamais de collisions).