RAPPORT TME 2-3-4

ABITBOL Yossef | SERRAF Dan

groupe 8 TME | groupe 10 TD

Introduction :

Dans ce projet, nous nous intéressons a la gestion d'une bibliothèque musicale composée de morceaux. Un morceau est repéré par son titre, le nom de l'artiste qui l'interprète et un numéro d'enregistrement.
L'objectif de ce mini-projet est d'apprendre a comparer des structures de données. Nous utiliserons dans ce projet pour implémenter une bibliothèque :

- une liste simplement chaînée,
- un arbre lexicographique,
- un tableau dynamique,
- une table de hachage.

Notre rapport va être composé de plusieurs parties en commençant par la présentation de nos différents fichiers .h et .c ainsi que du Makefile pour faciliter la compilation, ensuite dans la seconde partie nous allons aborder les différentes structures utilisées pour implémenter une bibliothèque et qui dans la partie suivante vont être testées par des Jeux d'Essais et comparées par leurs performances.

I/Le fonctionnement du Makefile

a) Les fichiers .h

Les fichiers .h ou "fichiers d'en-tête" sont principalement constitués des prototypes de fonctions définis dans les fichiers .c , dans notre projet nous avons principalement implémenter tous les fichiers d'en-tête dans un seul fichier nommé « biblio.h » regroupant aussi les prototypes des fonctions communes a tous les algorithmes.

b) Les fichiers .c

Les fichiers .c correspondent aux corps de fonctions des différents prototypes présents dans les fichiers .h. Ceci permet d'inclure tout simplement les fichiers .h qui feront référence a leurs fichiers .c et permet ainsi de signaler l'existence d'une fonction avant de pouvoir l'utiliser.

c) Le MakeFile

#include ''Biblio.h''

Fichiers .c

- Biblio_liste.c
- Biblio_tabdyn.c
- Biblio_arbrelex.c
- Biblio_hachage.c

Est inclus dans

Fichiers .h

- Biblio_liste.h
- Biblio_tabdyn.h
- Biblio_arbrelex.h
- Biblio_hachage.h

Est inclus dans

Biblio.h

Est inclus dans

Fonction principale

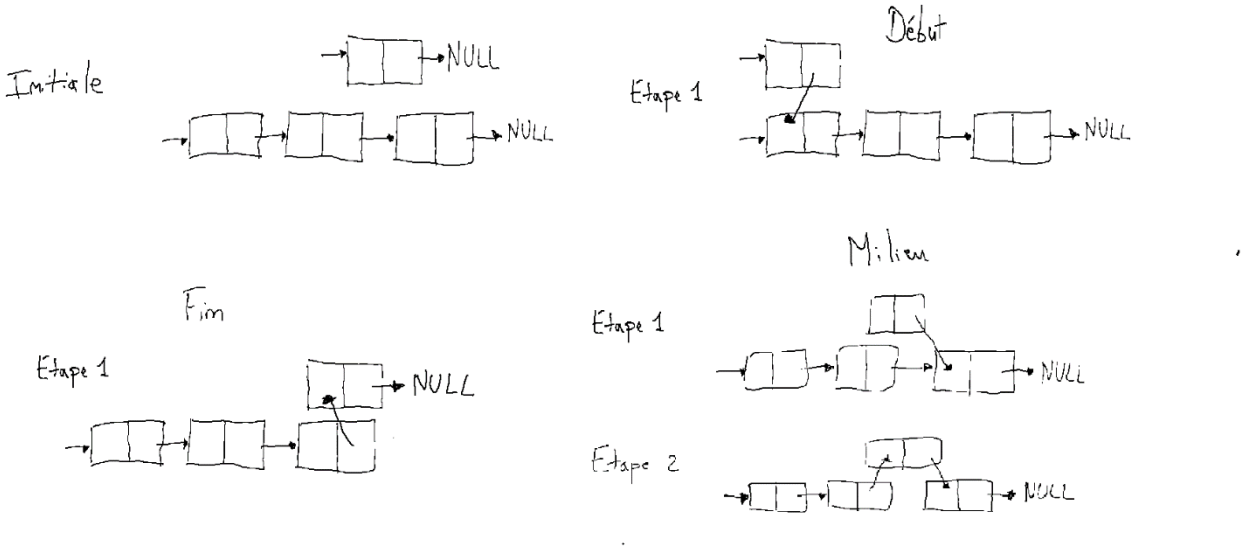
Main.c

Pour éviter d'avoir à recopier les commandes de compilation, on utilise un Make qui permet aussi de ne recompiler un fichier source que si c'est nécessaire, ce qui peut faire gagner beaucoup de temps. Tout au long du rapport on mettra les commandes nécessaires a la compilation.

II/ Les différentes structures utilisées

a) Liste Chainée:

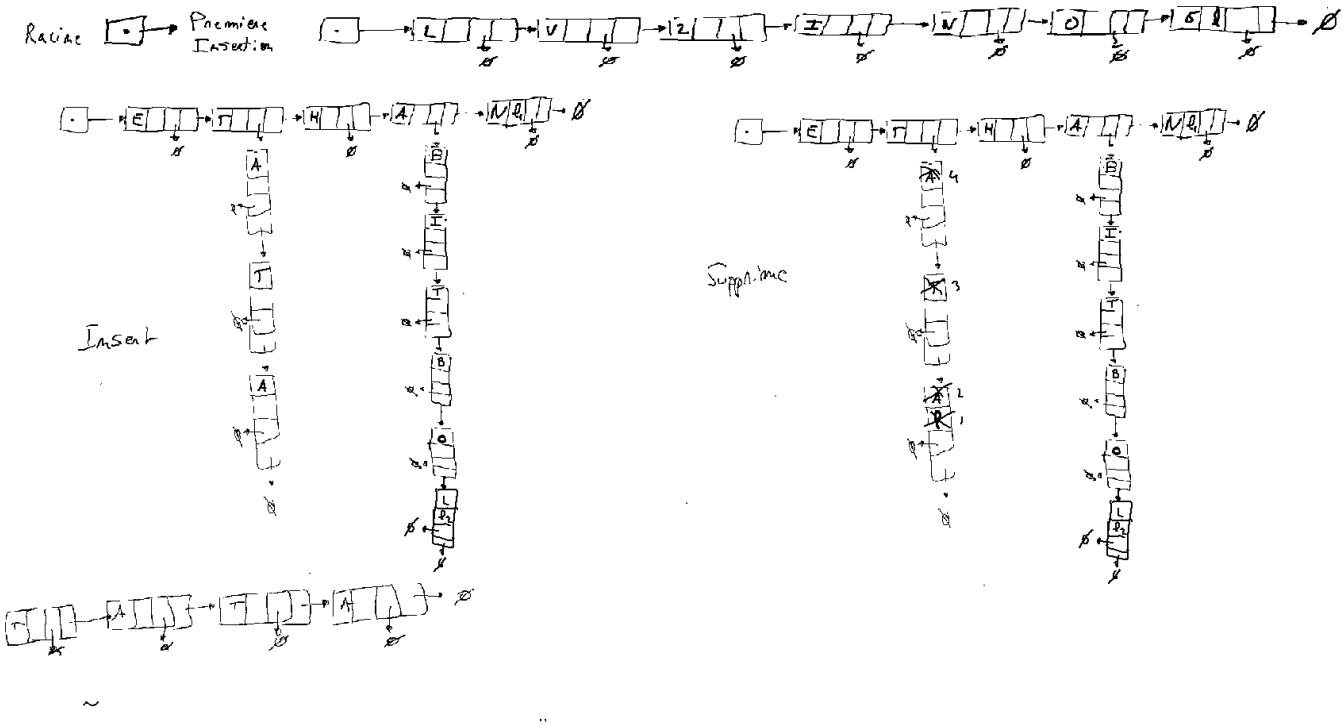
La première structure utilisé dans notre programme est la liste chaînée.



Chaque morceau de la liste chaînée sera constituer du numéro, du titre et du nom de l'artiste ainsi qu'un pointeur sur l'élément suivant.
*Ajouter: Ajoute élément en tête de liste en créant un élément et en faisant pointer le suivant par la liste chaînée.
*Supprimer: Il faudra distinguer 3 cas de tête, milieu, fin de liste.
Mais le schéma restera le même pour tous les cas on garde en mémoire l'élément courant à supprimer appelons var1 ainsi que sont suivant appelons le var2 puis on libère la mémoire var1 en n'oubliant pas de d'abords libérer la mémoire des éléments de la structure et enfin on refait le chaînage avec la liste chaînée grâce à var2.
*Rechercher: On parcourt l'ensemble de la liste jusqu'à trouve l'élément.

b) Arbre lexicographique :

La seconde structure utilisé dans notre programme est l'arbre lexicographique.

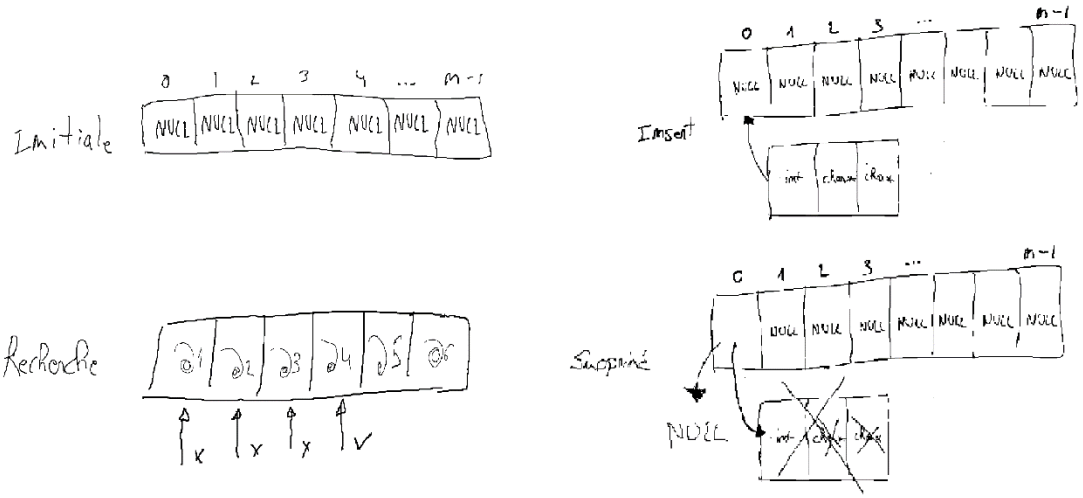


Chaque nœud de l'arbre sera constitué d'un pointeur vers un nœud suivant qui représenterons la liste des différents caractère possibles pour le caractère courant et d'un pointeur vers la liste des caractères suivants de l'artiste et enfin un pointeur vers un morceau qui correspondra à la liste chaînée des morceaux d'un même artiste qui sera stocker dans la dernière lettre de l'artiste.

*Ajouter: On parcourt l'ensemble de l'arbre si on trouve l'artiste correspondant on ajoute alors le morceau a la liste chaînée de morceau sinon on créer des nœuds pour chaque lettre de l'artiste restant.
*Supprimer: On parcourt l'ensemble de l'arbre jusqu'à trouve l'artiste correspondant puis on parcourt la liste chaînée jusqu'à trouve le morceau correspondant puis on le supprime (voir liste chaînée pour mécanisme).
*Recherche: On parcourt l'ensemble de l'arbre jusqu'à trouve l'artiste correspondant puis on parcourt la liste chaînée jusqu'à trouve le morceau correspondant.

c) Tableau Dynamique:

La troisième structure utilisé dans notre programme est le tableau dynamique.

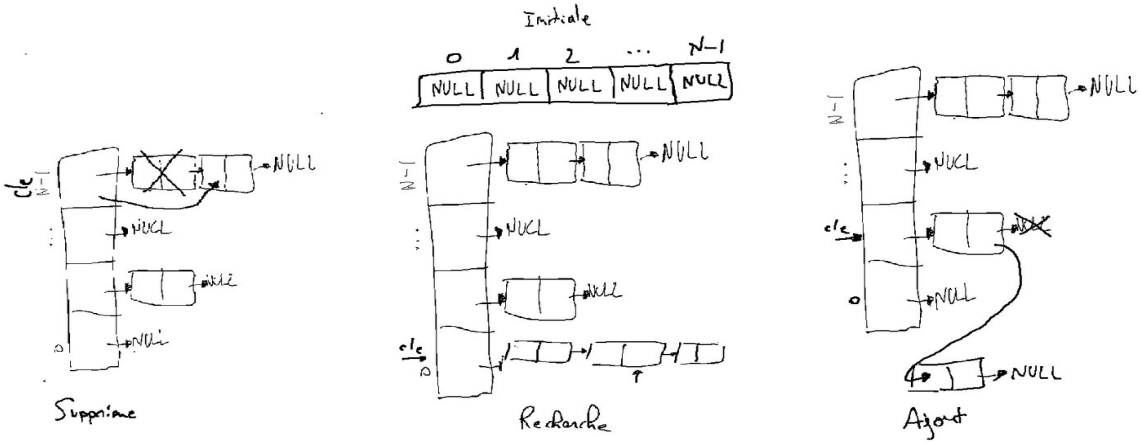


Chaque case du tableau dynamique sera un morceau qui sera constituer du numéro, du titre et du nom de l'artiste.

*Ajouter: On parcourt l'ensemble du tableau jusqu'à tomber sur une case null on insert alors l'élément dans la case. Toutefois si après le parcourt de tous le tableau on ne trouve pas de case libre il faudra ré-allouer un tableau plus grand.
*Supprimer: Il faudra parcourt l'ensemble du tableau jusqu'à trouve l'élément.
Puis on libère la mémoire de la case en n'oubliant de pas de d'abord libère la mémoire des éléments de la structure.
*Rechercher: On parcourt l'ensemble du tableau jusqu'à trouve l'élément.

d)Tableau de hachage:

La quatrième structure utilisé dans notre programme est le tableau de hachage.



Chaque case du tableau dynamique sera un morceau qui sera constituer du numéro, du titre et du nom de l'artiste ainsi qu'une clé lui permettant de connaitre directement l'index dans le tableau et enfin un pointeur vers l'élément suivant (pour les collisions).

Pour toutes les fonctions il faudra d'abord calculer l'index dans lesquels il faudra cherche le morceau en fonction d'une première fonction de hachage, puis on utilisera une deuxième fonction de hachage pour renvoyer une clé qui est inferieur a la taille du tableau.

*Rechercher: Une fois la clé trouve on sait que le morceau pourra se trouver uniquement dans la liste chaînée de cette case. Et on cherche alors dans liste chainer si l'élément est présent.
*Ajouter: Une fois la clé trouve on ajoute à la liste chaînée l'élément.
En effet plusieurs morceaux peuvent avoir une même clé, dans ce cas pour arranger la collision, on créer une liste chaînée, les chaînant les uns à la suite des autres.
*Supprimer: Une fois la clé trouve on supprime l'élément de la liste chaînée.

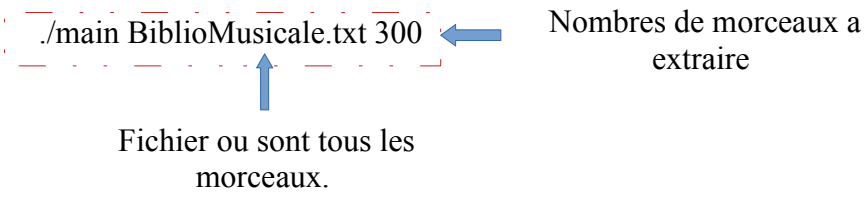
Remarque: pour recherche, ajoute, supprime dans liste chaînée voir plus haut dans liste chaînée.

III/ Jeux D'essais

a) Le main

Comme dis plus haut, notre main inclut le fichier Biblio.h qui fais référence a toutes les fonctions définis plus haut. De plus afin d'exécuter le main en fonction de chaque structure on dois entrée l'une des commandes suivante :
- make liste,
- make arbre,
- make tableau,
- make hachage, respectivement.

Puis,



si tous se passe bien, on voit afficher le menu suivant:

```
Menu:
0 - Sortie
1 - Affichage
2 - Recherche les morceaux uniques
3 - Recherche les morceaux a l'aide du titre du
morceau
4 - Recherche le morceau a l'aide du numéro
5 - Recherche les morceaux d'un artiste
6 - Ajoute un morceau dans la bibliothèque
7 - Supprimer un morceaux
Votre choix :
```

selon le chiffre qu'on entre , d'autres instructions apparaissent :

>>> 0 // on sort du main, le jeu d'essai s'arrête
>>> Au revoir ! //si aucune erreur, on voit affiché se message.

>>> 1 // on appelle la fonction Biblio *affiche(Biblio *B); et on voit afficher le nombre de morceaux demandé
>>> 0 Stuck On A Feeling Prince Royce Featuring Snoop Dogg
...
>>>299 ...
>>>Menu: ... // le menu réapparaît

>>>2 //On appelle la fonction Biblio *uniques(Biblio *B); et on voit afficher les morceaux qui n'ont pas de doublons parmi les n morceaux entrées
>>> 0 ... 288 //ici avec 300 morceaux on obtiens 288 morceaux unique
>>>Menu: ...

>>>3 //On appelle ici la fonction void recherche_morceau_avec_titre(Biblio *B, char *titre);
>>> saisir le titre du morceau: //on nous demande de saisir le titre
>>> Snowbird //1er lettre en majuscule et pas d'espace a la fin.
>>> 3 Snowbird Anne Murray //on nous affiche le morceau
>>>Menu: ...

>>> 4 //On appelle ici la fonction void recherche_morceau_avec_numero(Biblio *B, int numéro);
>>> saisir le numéro: // on nous demande de saisir le numéro
>>> 3
>>> 3 Snowbird Anne Murray
>>>Menu: ...

>>> 5 //On appelle ici la fonction void recherche_morceau_avec_artiste(Biblio *B, char * artiste);

>>> saisir le nom de l'artiste: // on nous demande de saisir le numéro

>>> Anne Murray //chaque 1er lettre en maj et pas d'espace a la fin

>>> 171 You Needed Me Anne Murray

>>> 3 Snowbird Anne Murray //on nous affiche les morceaux de cette artiste parmi les n morceaux entrées,ici 300.

>>>Menu: …

>>> 6 //on appelle ici la fonction biblio *insere(Biblio B,int numero, char *titre, char *artiste);

>>> saisir le numéro: //on nous demande de saisir le numéro

>>> 300

>>> saisir le titre: //on nous demande de saisir le titre

>>> ethan

>>>saisir l'artiste: // on nous demande de saisir l'artiste

>>> dan

>>>Menu:…

>>> 1 // pour vérifier que tous se passe bien on affiche la biblio implémenter du nouveau morceau

---

>>> 7 //on appelle ici la fonction biblio *suprime() ;

>>> saisir le numéro: //on nous demande de saisir le numéro

>>> 3

>>> saisir le titre: //on nous demande de saisir le titre

>>> Snowbird

>>>saisir l'artiste: // on nous demande de saisir l'artiste

>>> Anne Murray

>>> Menu:…

>>> 1 // pour vérifier que tous se passe bien on affiche la biblio sans le morceau.

remarquez que ici on fais l'exemple avec un morceau en milieu de bibliothèque, or la fonction est valide que le morceau sois au debut, au milieu ou en fin de liste,tableau, … sIl arrive qu'on sois des fois confronté a certaines erreurs comme Segmentation fault ou core dumped.

De plus, si nous voulons plus d'informations sur les fonctions tel que le nombre d'allocations et le nombre de free. On remplace la ligne suivant make liste , make tableau ,… par:

```
valgrind --leak-check=yes main BiblioMusicale.txt 300
```

puis, on effectue l'une des actions du menu et a la sortie (>>> 0) on voit afficher plusieurs lignes de codes dont une qui nous intéresse.

```
==2897== total heap usage: 0 allocs, 0 frees, 0 bytes allocated
```

source : https://linoxide.com/tools/valgrind-memcheck/

b) Comparaisons des performances

- Recherche:

	Avec numéro	Avec artiste	Avec titre
Liste chaînée	0,001008	0,039701	0,023444
Arbre lexico	0,004916	0,003901	0,026381
Tableau dyn	0,005294	0,006275	0,015788
Table de hachage	0,006223	0.000254	0,012384

On peux constater que pour lorsqu'on fais une recherche sur 3000 morceaux par numéro la liste chaînée est plus rapide que les autres structures alors que lorsqu'on fais une recherche avec le nom de l'artiste c'est la table de hachage qui prédomine, cela est explique par le fait que la table de hachage a comme clé le nom de l'artiste et pour finir lorsqu'on fais une recherche avec le titre du morceau le tableau dynamique et la table de hachage sont supérieur a la liste chaînée et a l'arbre lexicographique. De plus, si on modifie la table de hachage, 2 cas seront possibles.

- on augmente la table et ainsi il y aura plus de trous mais moins de collisions,
- on diminue la table et ainsi il y aura moins de trous mais plus de collisions.

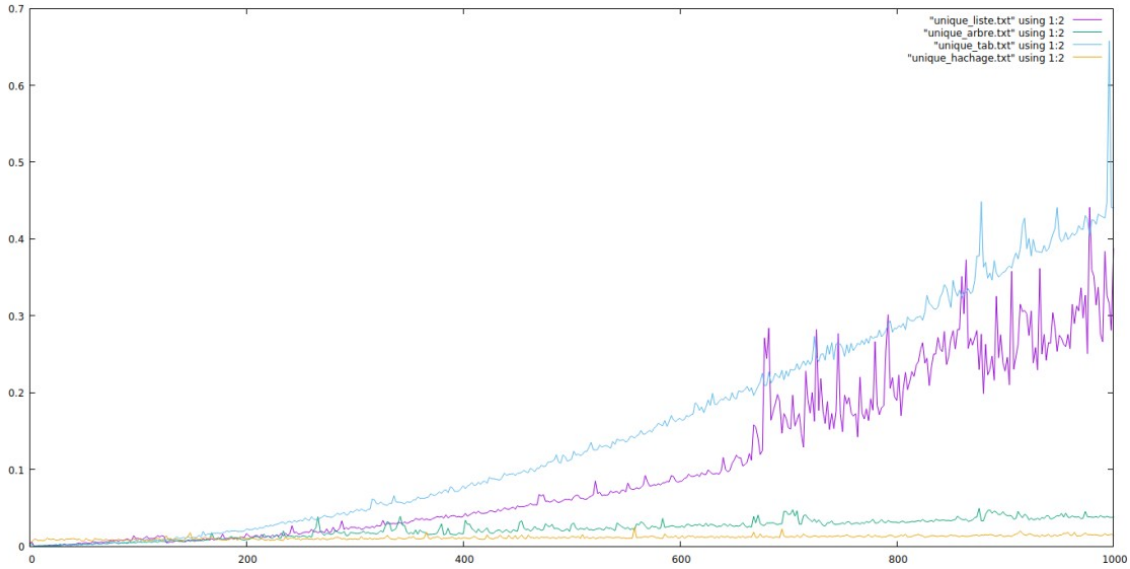
- Graphe Unique :

Pour la fonction unique, grâce a la fonction void uniques_graphe(int borne_inf, int borne_sup, char *fichier); on a calculé le temps mis pour chaque structure avec 1000 morceaux en entrée avec 1 pas de 1 ainsi que le temps mis avec 150000 morceaux et un pas de 5000.

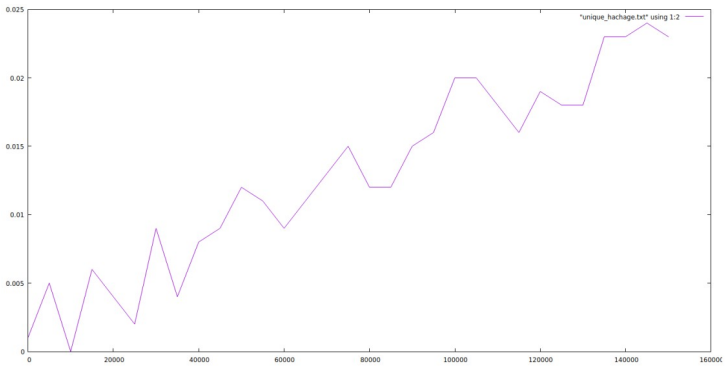
Pour ce faire, dans le menu expliqué plus haut on a un 8^{e} choix qui est « 8 – graphe », de plus pour exécuter cela on dois au préalable changer dans le main la borne_inf, la borne_sup et le pas ainsi que le fichier ou seront stockées les valeurs, ensuite nous devons entrer dans le terminal "gnuplot" et ainsi mettre:

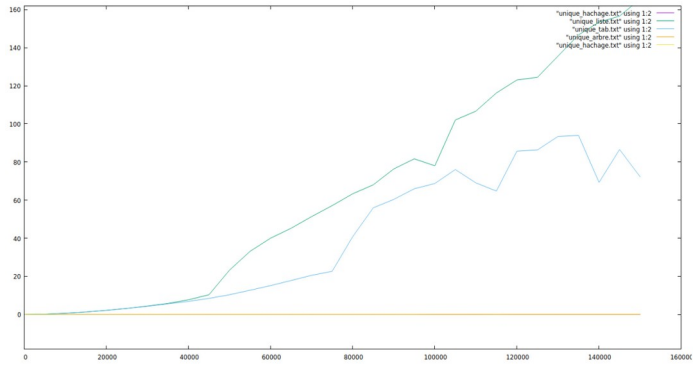
>>>gnuplot> plot "fichier" using 1:2 with lines
>>>gnuplot> replot "fichier" using 1:2 with lines //pour afficher 2 graphe dans la même fenêtre

On obtiens les graphiques suivant :



1- On peux constater que le temps mis pour la liste chaînée et l'arbre lexicographique est plus important que les 2 autres Structures avec 1000morceaux . On remarque que la table de hachage est la fonction la plus efficace .





2- Ici on remarque que avec un nombre importants de morceaux (150 000) la table de hachage est de loin la plus efficace contrairement aux 3 autres structures.

Pour conclure, ce TME nous a permis de comparer plusieurs structures face a une bibliothèque musicale, par leur rapidité et leur validité, nous pouvons constater que ces 4 structures, liste chaînée, arbre lexicographique, tableau dynamique et table de hachage, sont 4 manières différentes obtenir le résultat voulu mais que la table de hachage étais bien plus efficace que les autres par sa rapidité et son efficacité.