École Nationale Supérieure d'Informatique - Alger

TP01: Skiplist

Réalisé par: BENDAOUD Anes **ET** BOUMISSA Mohamed Chakib

Module: Algorithmique et Structures des Données Dynamiques

Objectif

Construire la structure de données ‘skiplist’ et sa machine abstraite on optimisant l’algorithme autant que possible

Hiérarchie des fichiers

Quand on parle d’un fichier ‘fichier_x’, il est représenté en deux fichiers: ‘fichier_x.c’ pour l’implémentation des modules de ce fichier et ‘fichier_x.h’ pour les en-têtes

ImplementMain.c et main.h: pour les structures des données et les modules basics (skipAllouer, skipSuivant … + les modules de la manipulation de la ***LLC***)

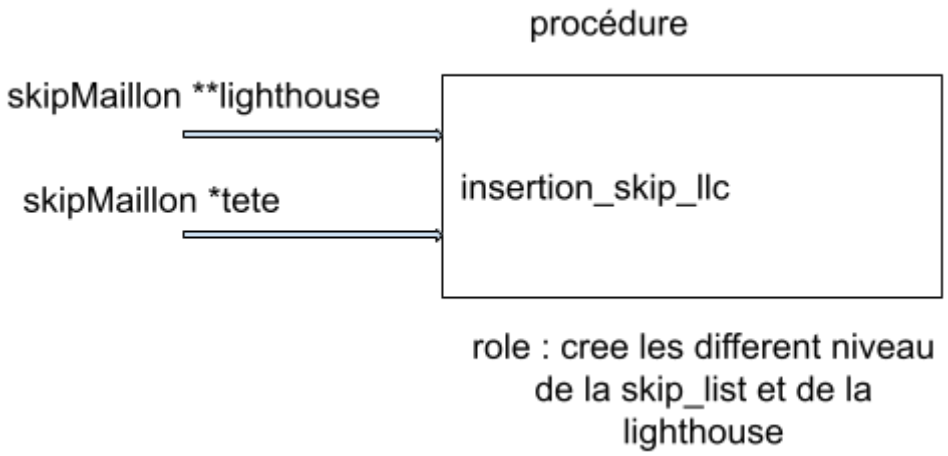
MachineAbstraite: pour les 4 modules de la machine abstraite; création, insertion, suppression et recherche

skipListRech: pour les modules utilisés dans le module de la recherche ‘skipRech’ (même chose pour le reste des fichiers)

menu: contient l’utilisation des modules de la machine abstraite + les testes de la performance pour simplifier l’interface du menu du programme ‘main’

Modules: Analyse

skipCree



analyse :

- la creation de la skip liste depuis la llc demande l'accès au different maillon précédent , d'ou vient le besoin de la creation de la liste guard , cette aura une taille équivalente au niveau max et permettera l'enregistrement des maillon précédents .
- la creation de lighthouse une autre qui permettera a la fin le mouvement au long de la skip list .
- le processus consiste à générer les niveau creation de llc et le programme sera expliquer a la démonstration .

explication de variables:

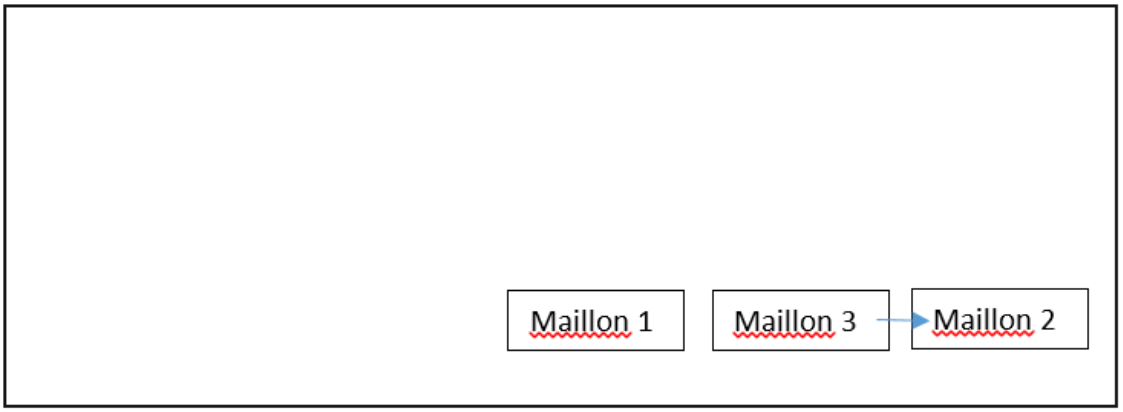
Le i , est une variable de type int qui permet de repérer le niveau actuel durant la creation des niveau .

Le max level est une autre variable de type int qui permet de sauvegarder le niveau maximale

:

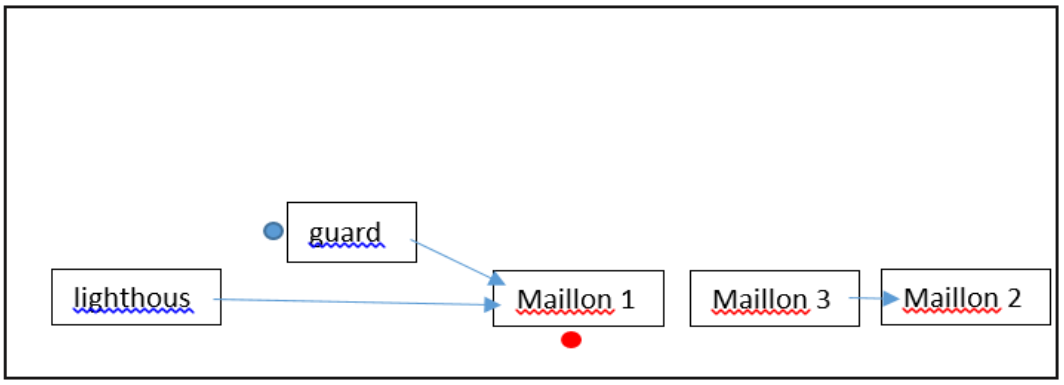
démonstration :

Afin de faire cela on aura besoin de commencer par une LLC simple de structure skip Maillon comme montrer ci dessous : (le slot (bas) de ses maillon est mis a NULL)



Apres avoir crée un maillon nommée lighthouse et insérer lui et la tète maillon 1 la fonction affectera l'adresse du premier maillons en lh(abréviation de lighthouse) et puis créera un maillon nome garde et un pointeur pt guard ce dernier nous permettra de traverser a liste Guarde et on aura une structure qui ressemble a ca :

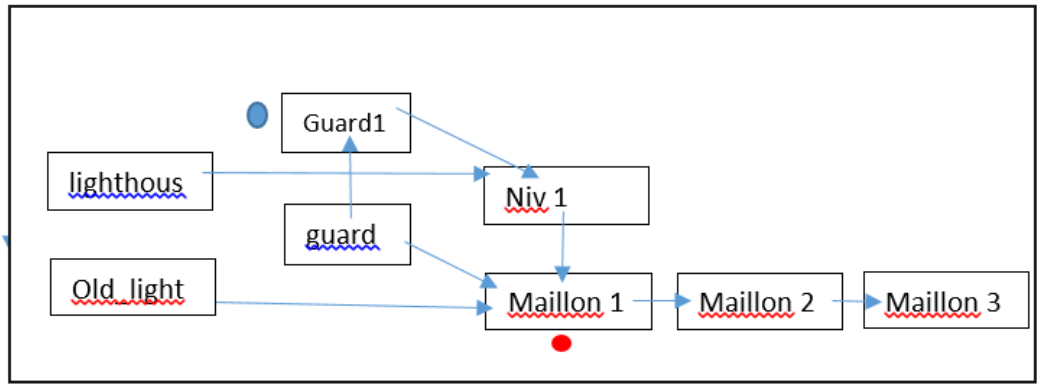
le pointeur garde est représenter comme une boulle bleu et la tête par une autre rouge .



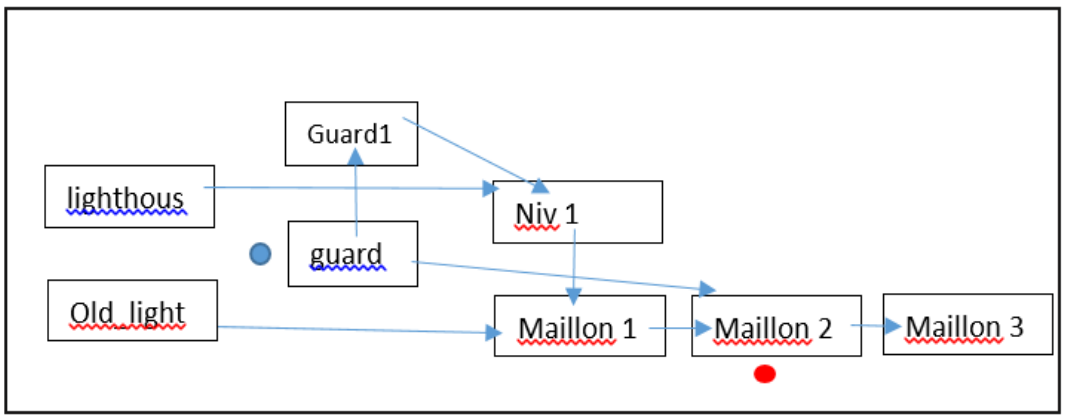
Maintenait un lancer d'une pièce sera et supposons que le résultat est pile , ce qui veut dire qu'un niveau sera ajouter au maillon 1 , mais si on regarde bien on remarque que le max level est de 0 , donc le lighthouse et guard auront toute les deux un niveau de plus ce qui sera fait par cette partie du code :

```
if( i > maxlevel){


    skipAllouer( mP: &new_guard);
    skipAllouer( mP: &new_lightouse);
    skipAffAdrBas( mP: ptr_guard, basP: new_guard);
    skipAffAdrBas( mP: new_lightouse, basP: *lighthouse);
    skipAffAdrSuivant( mP: new_lightouse, suivantP: ptr_creation);
    skipAffAdrSuivant( mP: new_guard, suivantP: ptr_creation);
    skipAffAdrBas( mP: new_guard, basP: NULL);
    *lighthouse = new_lightouse;
    ptr_guard = new_guard;
    maxlevel++;
```

Ce qui nous donne cette structure :



Maintenant une fois que tous a était crée on fera le lancer une autre fois , supposant que cette fois ca était face , donc un autre niveau ne sera pas crée dans ce cas , et tête avancera vers maillon 2



Le mouvement vers un nouvel maillon est assurer par cette partie de la partie :

```
tete = skipSuivant( mP: tete);
i = 0;
ptr_guard = guard;
```

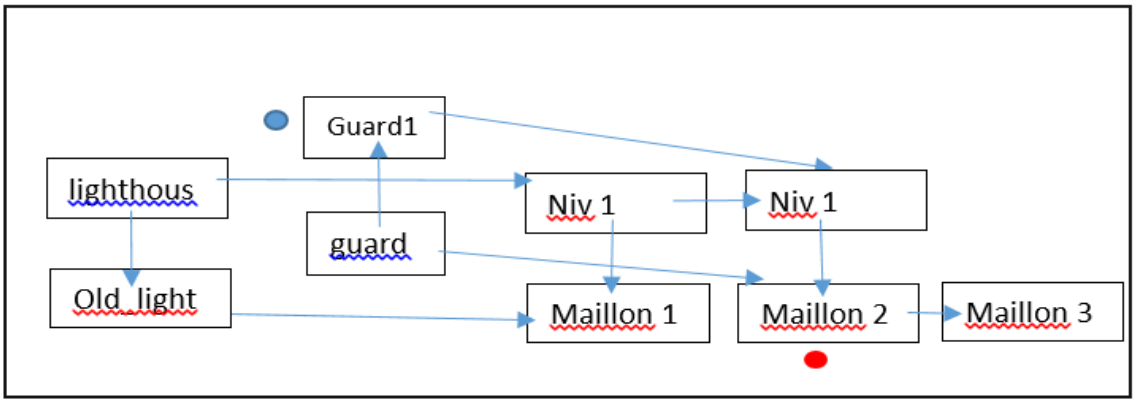
Le mouvement vers un nouvel maillon est assurer par cette partie de la partie:

```
tete = skipSuivant( mP: tete);
i = 0;
ptr_guard = guard;
```

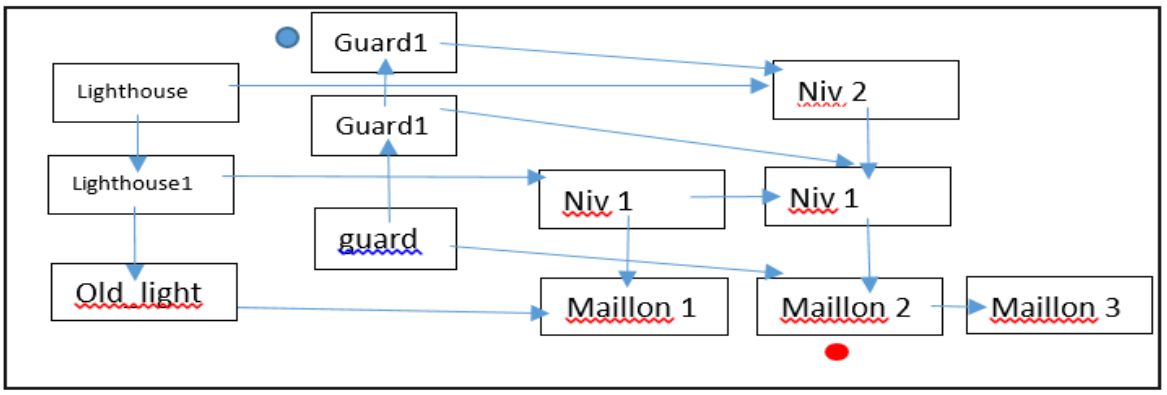
Apres supposons que le résultat a était pile donc un niveau sera ajouter a maillons , accès au niv 1 est impossible directement , et c'est ici que la liste Guardi interviennent car elle pointe toujours vers niv 1 du maillon 1 , cette fonctionnalité est assurer par cette partie :

```
} else{

    ptr_guard = bas( mP: ptr_guard);

    skipAffAdrSuivant( mP: skipSuivant( mP: ptr_guard), suivantP: ptr_creation);

    skipAffAdrSuivant( mP: ptr_guard, suivantP: ptr_creation);

}
```

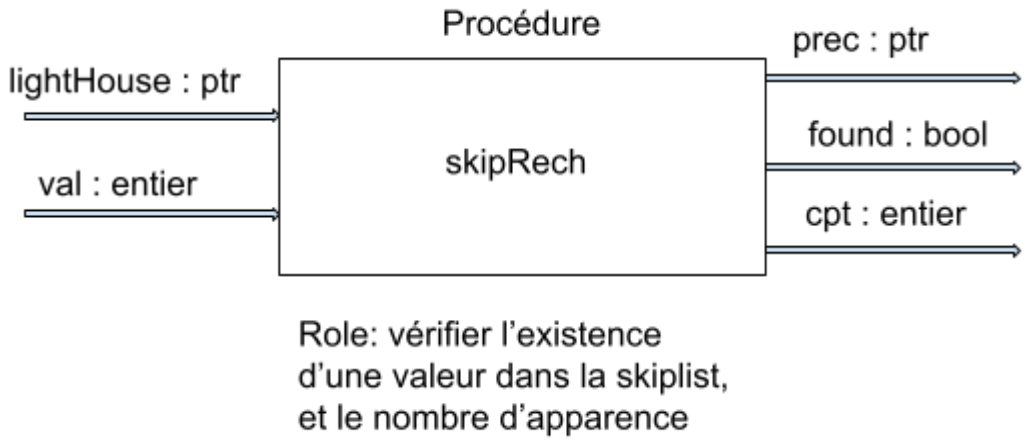
Ce qui visuellement nous donne le résultât :



Supposons que le résultats de la lance est pile une autre fois dans ce cas le i=2 ce qui veut dire que ça dépasse le max_level = 1 , donc l'ajout de ce niveau ajoutera un nouvel élément a Gard et a lighthouse , comme mention dans la création du niv_1 de maillon 1 , ainsi de suite j'usqua la fin



skipRech

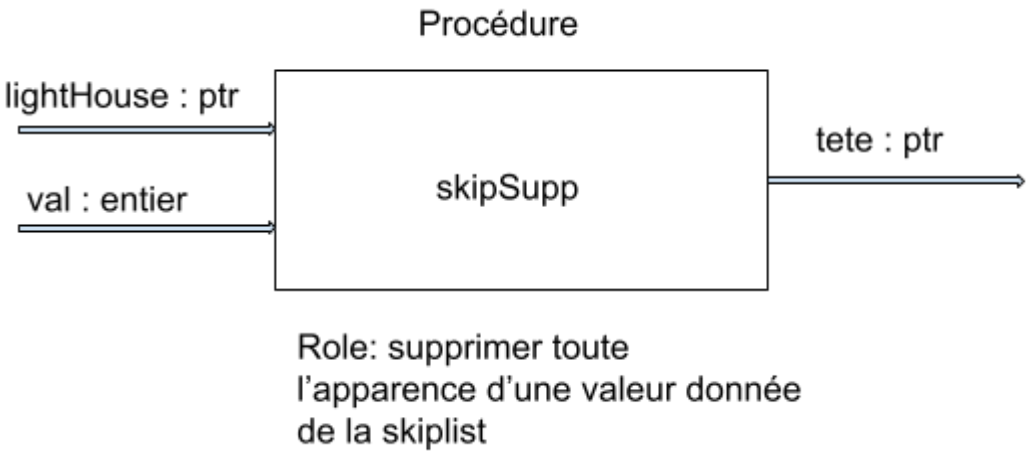


prec: ptr vers le maillon qui précède le maillon de la valeur que nous cherchons

Analyse:

- On parcourt la skiplist par le maillon qui précède le maillon que nous cherchons, ça veut dire, quand on trouvera le maillon, on serait pointé sur le maillon qui le précède, on l'appelle 'preced'
- On continue la recherche lorsque preced != 'nil' et l'élément n'est pas trouvé.
- On fait des mouvements dans la skiplist (bas ou droit) après des conditions vérifiés
- Si l'élément est trouvé, on appelle le module skipValRepeat qui va calculer le nombre d'apparences de la valeur.

skipSupp

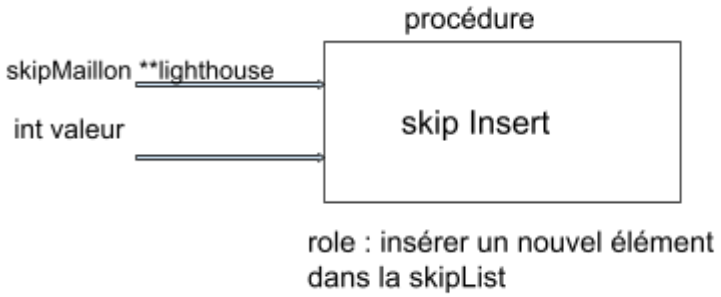


tête: ptr vers la nouvelle tête dans le cas de la suppression du premier élément

Analyse:

- On appelle le module de la recherche 'skipRech' et on récupère le ptr vers le maillon qui précède le maillon que nous voulons supprimer, soit 'prec'
- On parcours tous les niveaux au-dessous de 'prec' avec les niveaux du maillon qui suive la dernière apparence de la valeur et on fait le chaînage par le module 'suppNiveaux'

skipInsert



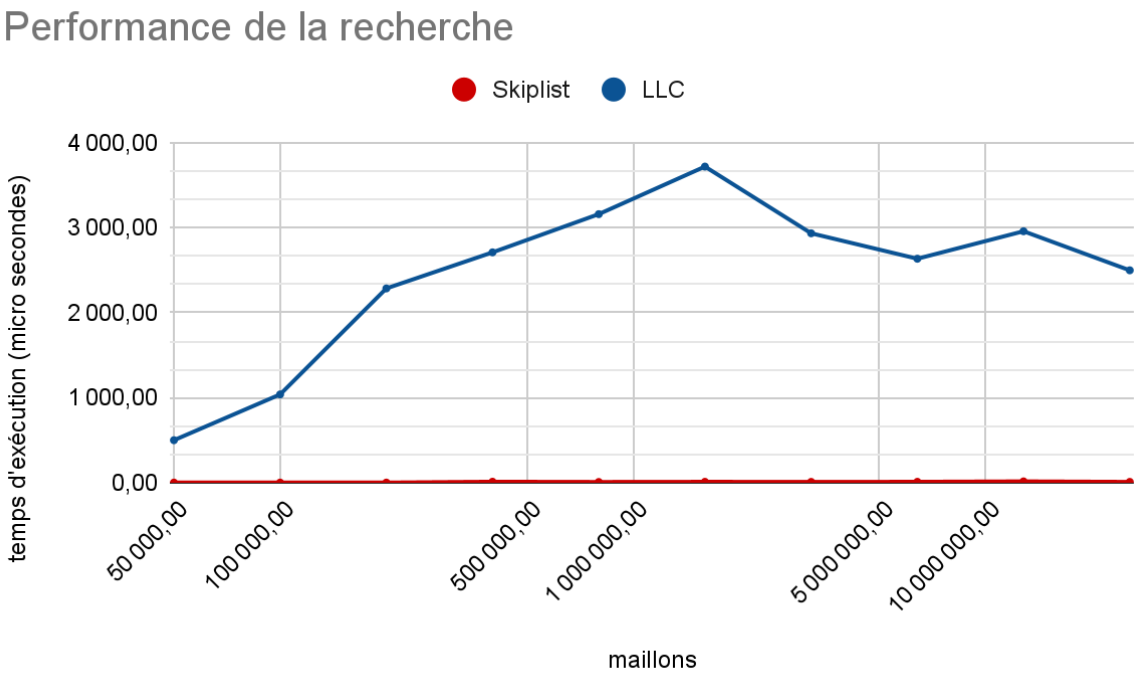
role : insérer un nouvel élément
dans la skipList

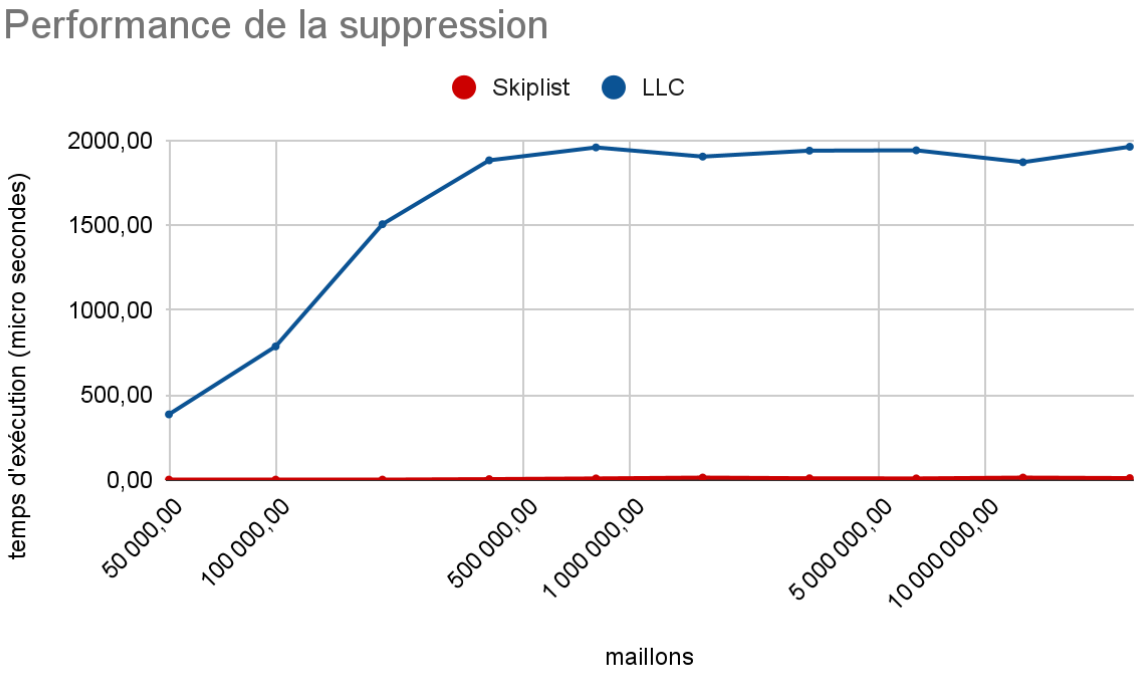
analyse :

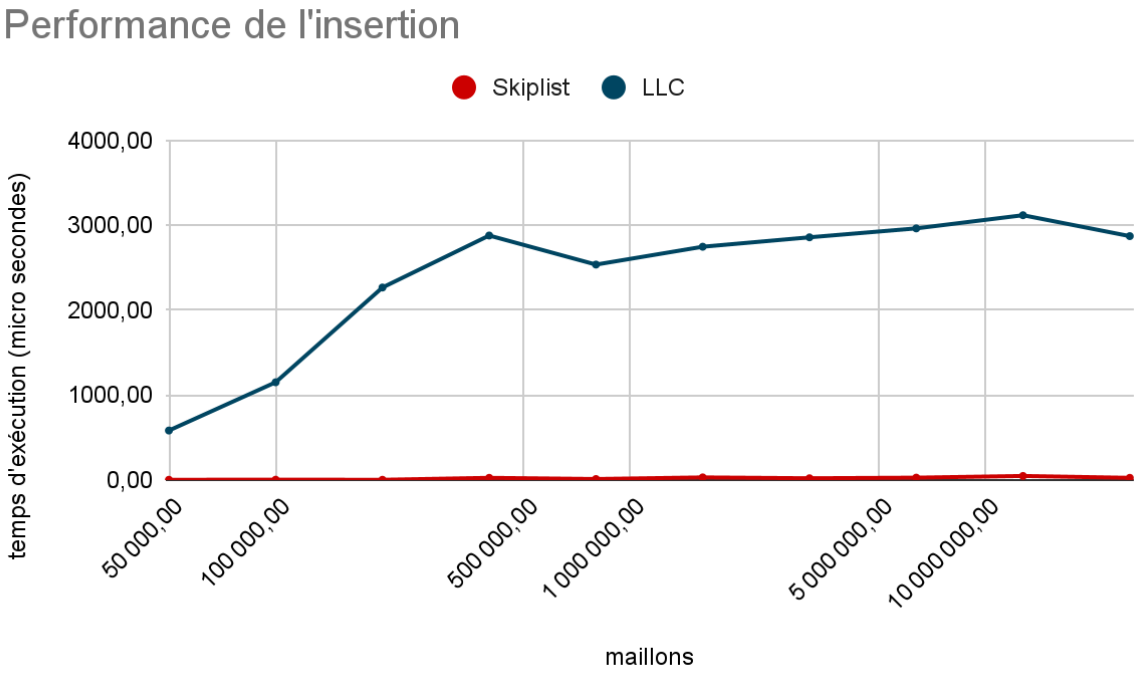
- commencer de lighthouse
- parcourire la liste en enregistrant les maillon précédent et récupération de l'emplacement
- insertion du nouvelle maillon et création des niveau

Expérience avec la structure

Ces tests sont appliqués en utilisant la valeur ‘12 345 678’ dans tous les cas.







Conclusion

En général, l'optimisation de la complexité des algorithmes de recherche, d'insertion et de suppression sur les listes a connu un grand succès grâce à l'utilisation de la skiplist, qui est très utile lorsqu'on manipule de grandes listes linéaires chaînées (avec des milliers de maillons).