S1-02 Comparaison d’approches algorithmiques

CHOFFAT Rémi - BESANÇON Marcelin - S1E

Introduction

L’objectif de la SAÉ est de comparer expérimentalement l’efficacité de trois types d’implémentation de listes triées de chaînes (représentation contiguë dans un tableau, représentation chaînée dans un tableau avec récupération des places libérées, représentation chaînée dans un tableau avec gestion de l’espace libre à l’aide d’une liste), notamment de grande taille, et ce pour les opérations d’ajout et de suppression.

Notre environnement expérimental :

- Processeur : AMD Ryzen 7 7840HS 3.80GHz, 8 coeurs
- Mémoire : 16Go
- Système d’exploitation : Windows 11 Famille, Version 23H2

Algorithmes logiques

```
Fonction adjlisT(l:liste(chaîne), c:chaîne)
  Début
    p <- tête(l)
    inséré <- Faux
    TantQue NON inséré ET NON finliste(l,p) faire
      Si val(l,p) > c alors
        Si p = tête(l) alors
          adjtlis(l,c)
        Sinon
          adjlis(l,pPre,c)
        FinSi
        inséré <- Vrai
      Sinon
        pPre <- p
        p <- suc(l,p)
      FinSi
    FinTantQue
    Si NON inséré alors
      adjlis(l,pPre)
    FinSi
  Fin
```

```
Fonction suplisT(l:liste(chaîne), c:chaîne)
  Début
    p <- tete(l)
    supprimé <- Faux
    pPre <- 0
    Si NON finliste(l,p) alors
      TantQue NON supprime ET NON finliste(l,p)
          Si val(l,p) = chaine alors
            suplis(l,p)
            supprime <- Vrai
          FinSi
          pPre <- p;
          p <- suc(l,p);
      FinTantQue
    FinSi
  Fin
```

Expériences réalisées

Question 9

Appelée avec une chaîne de caractères qui n'appartient pas à la liste, la méthode `suplisT` va effectuer un simple parcours total de la liste, à la recherche de la chaîne qui correspond, sans autre action car la chaîne ne sera pas trouvée. Ainsi, le temps d'exécution de cette méthode ne variera pas significativement. Il n'est donc pas intéressant de répéter cette opération 10 fois pour comparer les implémentations, une seule suffit.

Objectifs

L'objectif est de mesurer le temps pris en répétant 10 opérations faites sur une liste triée, en faisant varier le type d'opération (ajout ou suppression), où l'opération a lieu (en début ou fin de liste) et l'implémentation de la liste (contiguë, chaînée, ou chaînée avec liste libre).

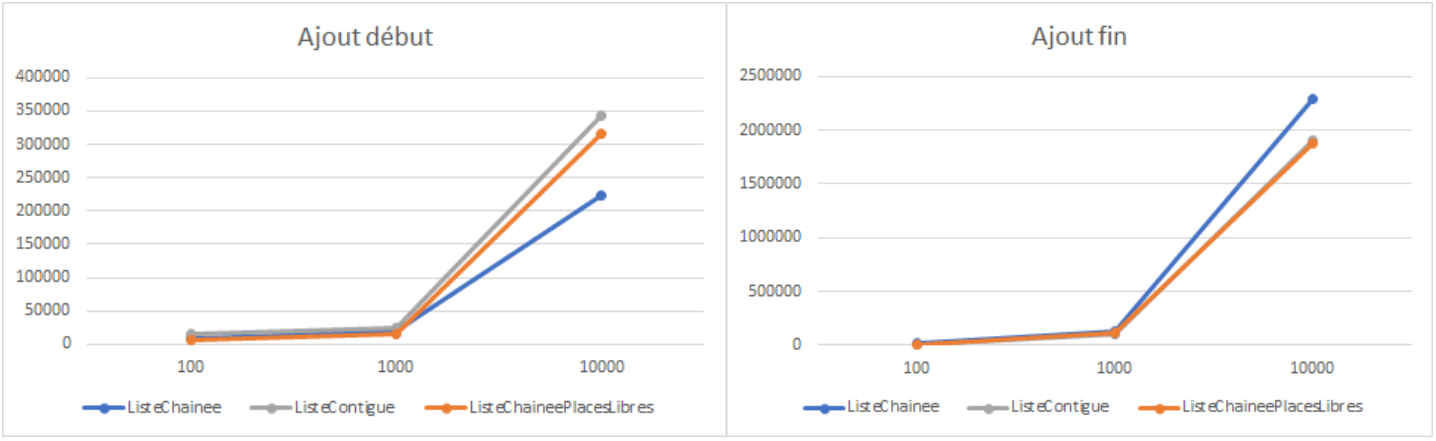
La classe `PrincipaleVariationTaille` a pour objectif d'effectuer plusieurs calculs de temps d'exécution, pour plusieurs tailles de listes (dans nos expériences, 100, 1 000, et 10 000).

Résultats

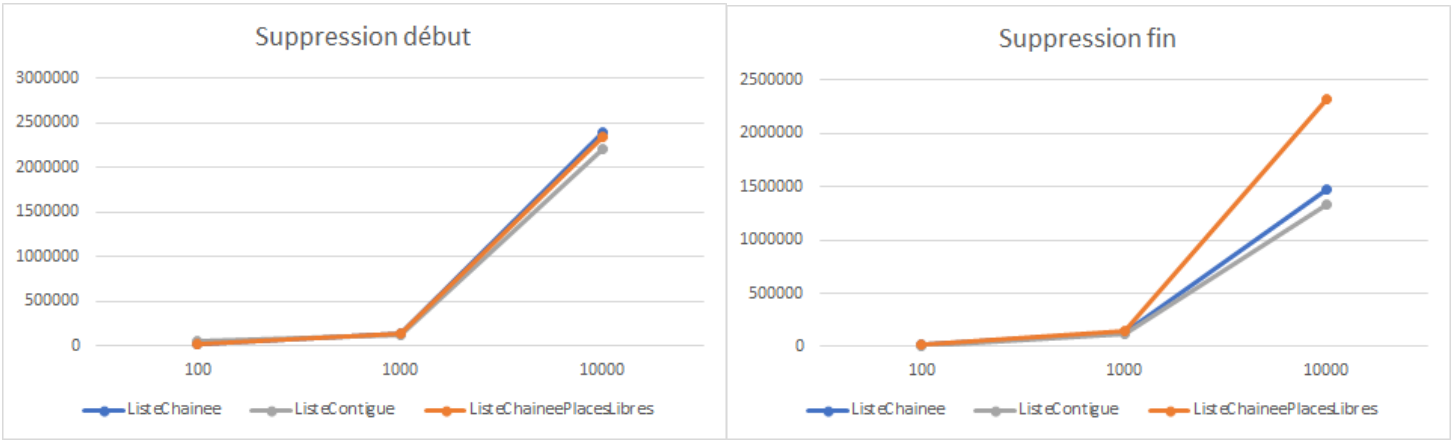
liste	operation	emplacement	duree
ListeChainee	ajout	debut	7658
ListeContigue	ajout	debut	16099
ListeChaineeP	ajout	debut	6415
ListeChainee	ajout	fin	14083
ListeContigue	ajout	fin	13121
ListeChaineeP	ajout	fin	13239
ListeChainee	suppression	debut	24864
ListeContigue	suppression	debut	62607
ListeChaineeP	suppression	debut	26304
ListeChainee	suppression	fin	15508
ListeContigue	suppression	fin	12416
ListeChaineeP	suppression	fin	14785
ListeChainee	ajout	debut	19806
ListeContigue	ajout	debut	24954
ListeChaineeP	ajout	debut	16436
ListeChainee	ajout	fin	133367
ListeContigue	ajout	fin	107846
ListeChaineeP	ajout	fin	115348
ListeChainee	suppression	debut	143492
ListeContigue	suppression	debut	123538
ListeChaineeP	suppression	debut	141812
ListeChainee	suppression	fin	133803
ListeContigue	suppression	fin	115596
ListeChaineeP	suppression	fin	151385
ListeChainee	ajout	debut	223744
ListeContigue	ajout	debut	343618
ListeChaineeP	ajout	debut	315126
ListeChainee	ajout	fin	2290693
ListeContigue	ajout	fin	1901415
ListeChaineeP	ajout	fin	1874083
ListeChainee	suppression	debut	2395147
ListeContigue	suppression	debut	2207985
ListeChaineeP	suppression	debut	2340674
ListeChainee	suppression	fin	1479804
ListeContigue	suppression	fin	1327746
ListeChaineeP	suppression	fin	2320221

Nous avons exporté les résultats de nos mesures de temps d'exécution dans un fichier *csv* (d'abord les mesures pour une liste de 100 noms, puis de 1000, puis de 10000), afin de pouvoir les exploiter et les comparer.

Nous avons représenté le résultat de cette expérience par un graphique avec, en abscisse, le nombre d'éléments de la liste et, en ordonnée, le temps d'exécution d'ajout ou de suppression de 10 éléments en début, puis à la fin de cette liste (moyenne sur 100 opérations).



La liste chaînée est plus efficace lors d'un ajout au début. Cependant, la liste contiguë et la liste chaînée avec gestion des places libres ont une meilleure efficacité pour un ajout à la fin de la liste.



Les trois implémentations ont une efficacité similaire pour une suppression en début, alors que la liste chaînée et la liste contiguë sont bien plus performantes que la liste chaînée avec gestion des places libres pour une suppression en fin de liste.

Conclusion

En conclusion de notre comparaison d'approches algorithmiques pour la gestion de listes triées de chaînes, nous avons observé des résultats significatifs. Notre étude s'est concentrée sur trois types d'implémentations : la représentation contiguë dans un tableau, la représentation chaînée dans un tableau avec récupération des places libérées, et la représentation chaînée dans un tableau avec gestion de l'espace libre à l'aide d'une liste.

L'objectif principal était de mesurer l'efficacité de ces implémentations pour les opérations d'ajout et de suppression, en tenant compte de la taille de la liste, du type d'opération (ajout ou suppression), et de l'endroit où l'opération avait lieu (début ou fin de la liste).

Nous avons synthétisé nos résultats en représentant graphiquement les temps d'exécution pour les différentes tailles de listes et types d'opérations. Ces graphiques nous ont permis de visualiser clairement les tendances et de comparer les performances des différentes approches.

Si les ajouts au début sont fréquents, la liste chaînée peut être privilégiée, tandis que pour des ajouts à la fin ou des suppressions en fin de liste, la liste contiguë ou la liste chaînée sans gestion des places libres peut être plus appropriée. Ces résultats fournissent des informations utiles pour guider le choix d'implémentation en fonction des besoins spécifiques de l'application.