SAE R101-C – rapport

a) Objectif (types d'implémentations) et matériel employé

Durant cette Saé, l'objectif a été de comparer empiriquement l'efficacité de trois implémentations différentes de listes triées de chaînes de caractères. A savoir :

-Liste contiguë

-Liste chaînée

-Liste chaînée avec gestion de l'espace libre

Pour cette Saé nous avons utilisé un ordinateur personnel avec un processeur AMD Ryzen dont la vitesse oscille entre 1.8 Ghz et 4.3 Ghz , la mémoire est cadencée à 3200Mhz, le système d'exploitation est Windows 10 64-bit.

b) Algorithmes logiques

```
fonction adjlisT(l InOut : Liste(chaîne), nom : chaîne)
  Début
    P←tete(l)
    si val(l,p) > nom
      alors adjtlis(l,nom)
    sinon
      precedent←p
      p←suc(l,p)
      ajout←faux
      tant que non finliste(l,p) et non ajout faire
       si val(l,p) > nom
          alors adjlis(l,précedent,nom)
                ajout←vrai
       fsi
       precedent←p
       p←suc(l,p)
       ftant
       si non ajout
```

```
            alors adjqlis(l,nom)
          fsi
        fsi
      fsi
    fin


fonction suplisT (l InOut : Liste(chaîne), c : chaîne )
    début
      p←tete(l)
      depassement←faux
      tant que non finliste(l,p) et non depassement faire
        chCourante←val(l,p)
        si chcourante >= c
          alors depassement←vrai
            si chCourante = c
              alors suplis(l,p)
            fsi
          sinon p←suc(l,p)
        fsi
      ftant
    fin
```

c)Expériences et résultats

```
long debutD = System.nanoTime();

for (int i = 0; i < ELEMENTS_DE_DEBUT.length; i++) {
    liste.adjlisT(ELEMENTS_DE_DEBUT[i]);
}
long finD = System.nanoTime();

long dureeD = finD-debutD;

return dureeD;

}
```

Nous avons mis en place un compteur. Son rôle est de mesurer le temps écoulé pendant l'exécution de la boucle for des méthodes de remplissage ou de suppression ; pour ce faire, on récupère le temps et on l'affecte à une variable de type long aux deux extrémités de l'itération. Finalement, on affecte la valeur de la différence entre la valeur finale et initiale à une variable dureeD de type long et on la retourne.

Dans le cas de l'ajout au début de 10 éléments de type String :

chainee	ajout	debut	146600.0
contigue	ajout	debut	70400.0
chaineePL	ajout	debut	160600.0

(Temps en nanosecondes)

On s'aperçoit que la liste contiguë est plus rapide avec la méthodologie de mesure présentée ci-dessus. L'ordre de grandeur pour les différentes listes chaînées est le même et est une fois supérieur à celui de la liste contiguë.

Dans le cas de l'ajout en fin de liste de 10 éléments de type String :

chainee	ajout	fin	324500.0
contigue	ajout	fin	158600.0
chaineePL	ajout	fin	149800.0

(Temps en nanosecondes)

On voit que la liste chaînée avec gestion de places libres est légèrement plus rapide que la liste contiguë cependant les trois implémentations partagent le même ordre de grandeur. La liste chaînée de base est environ deux fois plus lente que les autres.

Dans le cas de la suppression au début de 10 éléments de type String :

chainee	suppression	debut	64300.0
contigue	suppression	debut	54600.0
chaineePL	suppression	debut	69300.0

(Temps en nanosecondes)

On remarque que le temps d'exécution est en moyenne plus rapide pour la suppression que pour l'ajout. La liste contiguë est en tête et les deux implémentations chaînées sont au même niveau.

Dans le cadre de la suppression à la fin de 10 éléments de type String :

chainee	suppression	fin	21700.0
contigue	suppression	fin	21800.0
chaineePL	suppression	fin	35900.0

(Temps en nanosecondes)

Dernièrement, on voit que le liste contigüe et chaînée sont au même niveau et la chaînée avec gestion de places libres un peu plus lente en comparaison.

Question 9 :

La méthode suplisT est conçue pour éliminer un élément spécifique d'une liste. En cas d'insertion d'une chaîne qui ne fait pas partie de la liste, la fonction effectuera d'abord une recherche de cette chaîne. Si la méthode ne parvient pas à la localiser, la liste restera inchangée. Il pourrait être pertinent de répéter cette opération plusieurs fois afin de mettre à l'épreuve la fiabilité de l'algorithme logique et ainsi évaluer l'efficacité de la fonction suplisT.

Amélioration des tests :

```java
public static double fois100AjDebut(ListeTriee liste, String nom_fichier){

    double moyenne=0;

    for(int i = 0;i<100;i++){

        moyenne+=remplir_listeTempsExeDebut(liste, nom_fichier);

    }

    moyenne+=moyenne/100;
    return moyenne;
}
```

Nous avons modifié les tests simplement en faisant boucler 100 fois les méthodes précédemment utilisées, une par une.

Ajout-début

```
Chainee : 4459661.06 ns
Contigue : 4490561.0 ns
ChaineePL : 5112021.07 ns
```

La liste chaînée de base et contiguë plus ou moins équivalentes. L'implémentation chaînée avec places libres se démarque par un temps d'exécution plus lent.

Ajout-fin

```
Chainee : 1.927675092E7 ns
Contigue : 1.678215394E7 ns
ChaineePL : 1.722433901E7 ns
```

Ici, la première chose observée, relève du temps d'exécution qui est bien plus long que les autres.

10^{e7} ns revient à dire $10^{e(7-9)} = 10^{e-2}$ s.

Donc dans ce cas de figure, c'est la liste chaînée de base qui est plus lente que les autres avec ~0.02 s de temps d'exécution.

Sup-début

```
Chainee : 9667607.89 ns
Contigue : 6781246.05 ns
ChaineePL : 6533693.03 ns
```

La liste contiguë et chaînéePL sont nettement plus rapides que la liste chaînée de base.

Sup-fin

```
Chainee : 3619732.94 ns
Contigue : 3348857.0 ns
ChaineePL : 3963538.96 ns
```

Finalement, la liste contiguë reste devant ici et la liste chaînéePL est nettement plus lente.

d)Conclusion

Pour conclure cette SAE, on peut dire que dans la grande majorité des cas, la liste contiguë reste la plus efficace des trois implémentations de listes triées testées. La liste chaînée avec gestion de places libres est dans certains cas plus rapide que la liste chaînée de base.