Complément aux Structures de Données Dynamiques
Listes chaînées

Manuel Munier

Version 18 septembre 2024 (14:24)

Objectifs

Ceci est la suite du troisième TP qui était consacré aux structures de données dynamiques (SDD), représentées en Python via le code OO vu en cours. Cette seconde partie vous confrontera à des algorithmes "plus poussés" sur les listes chaînées ☺

Consignes de programmation

Voici quelques conseils pour prendre de bonnes habitudes en programmation avec les SDD :

- Faites des schémas sur papier pour "visualiser" vos SDD.
- Pour les algorithmes, tracer sur papier les différentes étapes avant de coder. Pensez notamment :
  - les références sont à sens unique (on ne peut pas revenir en arrière sans référence explicite)
  - si vous n'avez pas stocké la référence d'un nœud, celui-ci existera toujours en mémoire mais vous ne pourrez plus y accéder !
- Penser à commenter vos codes sources : rôle d'une classe, fonctionnalité des méthodes, etc.
- Penser "modularité"...

Travail à réaliser

Vous allez donc reprendre comme base de travail le code Python du sujet 3 sur les listes chaînées (classe `Node`, classe `LinkedList`, etc.). Pour mémoire, la classe `LinkedList` nous permet de représenter également des listes vides, i.e. des listes qui n'ont même pas de premier nœud.

Voici dans les grandes lignes les principales méthodes qu'il vous est demandé de développer sur la classe `LinkedList` dans cette seconde partie du troisième TP :

1. Une méthode `addInTail` qui ajoute une valeur en queue de liste (en créant bien évidemment un nouveau nœud)

2. Une méthode `addSorted` qui, sur une liste supposée triée par ordre croissant, ajoute une nouvelle valeur passée en paramètre. L'insertion doit être réalisée de façon à ce que la liste reste triée.

3. Une méthode `remove` qui, sur une liste supposée triée par ordre croissant, supprime tous les nœuds dont la valeur correspond à celle passée en paramètre.

4. Une méthode `merge` qui, sur une liste supposée triée par ordre croissant, prend en paramètre une seconde liste chaînées (également triée) et ajoute, "à la bonne place", tous les nœuds de cette liste

passée en paramètre. Bien évidemment, pas question ici de faire des appels à répétition à la méthode `addSorted` écrite précédemment ! Imaginez un algorithme "plus efficace"...

5. Une méthode `head` qui retourne une liste contenant les N premières valeurs de la liste (N passé en paramètre ; créer de nouveaux nœuds).

6. Une méthode `tail` qui retourne une liste contenant les N dernières valeurs de la liste (N passé en paramètre ; créer de nouveaux nœuds).

7. Plus, bien évidemment, des programmes de test en tout genre pour instancier des objets à partir des classes écrites ci-dessus, invoquer des méthodes sur ces objets, etc. Pensez également à bien tester vos méthodes **dans tous les cas de figure** : premier nœud de la liste, dernier nœud de la liste, liste vide, etc. Posez vous aussi la question de savoir si vous partagez les nœuds entre plusieurs listes (mêmes références) ou si vous créez des copies...

Document rédigé en LaTeX sous Linux.