Introduction aux Structures de Données Dynamiques
Listes chaînées

Manuel Munier

Version 10 septembre 2024 (14:18)

Objectifs

Ce troisième TP sera pour vous l'occasion de vous confronter aux structures de données dynamiques (SDD). Nous commencerons "calmement" avec les listes simplement chaînées : chaque nœud (ou cellule) contient 1 valeur et 1 suivant ("lien" vers le nœud suivant). Pour la représentation d'une telle SDD en Python nous réutiliserons le code OO vu en cours.

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 vu en cours pour représenter 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 principaux points à développer dans ce troisième TP :

1. Saisir le code vu en cours et écrire quelques programmes de test pour (bien) comprendre comment on crée une liste chaînée et comment fonctionnent les méthodes de base : algos itératifs (ex : méthode `printList`) et algos récursifs (ex : méthode `printListRec`).

2. Réfléchir (sur papier) puis implémenter (en Python) les méthodes suivantes (méthodes définies sur la classe `LinkedList`, mais vous aurez "peut-être" besoin de définir d'autres méthodes sur la classe `Node` également...) :
   (a) une méthode `printListRecRev` qui permet d'afficher, de manière récursive, les différentes valeurs de la liste mais dans leur ordre inverse
   (b) une méthode `countNodes` qui retourne le nombre de nœuds dans la liste

(c) une méthode `addInHead` qui ajoute une valeur en tête de la liste (en créant bien évidemment un nouveau nœud)

(d) une méthode `reverseList` qui construit une nouvelle liste (i.e. pas de nœuds en commun) contenant les mêmes valeurs, mais dans l'ordre inverse

3. 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.

Document rédigé en LaTeX sous Linux.