Algorithmique – Programmation Objet – Python
TD n°6
Listes chaînées

Licence Informatique 2ème année
Université de Nice-Sophia Antipolis

1 Listes Simplement chaînées

On supposera que les attributs suivants sont définis (L représente la liste entière et x est un élément de la liste) :

- x.DONNÉE : contient une référence à la donnée associée à l'élément x.
- x.SUIVANT : contient une référence à l'élément suivant l'élément x.
- L.PREMIER : contient une référence au premier élément de la liste, NIL si la liste est vide.

Écrivez les méthodes suivantes, en mettant éventuellement à jour les attributs mentionnés ci-dessus :

1. *L.estVide*() renvoie vrai si la liste est vide et faux sinon.
2. *L.supprimerPremier*() supprime le premier élément de L.
3. *L.vider*() supprime tous les éléments de L.
4. *L.ajouterEnTête*(x) ajoute x au début de L.
5. *L.ajouterAprès*(x, y) ajoute x dans L après y.
6. *L.supprimerSuivant*(x) supprime le suivant de x dans L.
7. *L.numÉléments*() renvoie le nombre d'éléments de L. Quelle est la complexité de cette méthode ?

1.1 Comptage des éléments

On veut améliorer la complexité de la méthode *L.numÉléments*(). Pour ce faire, on introduit un nouveau attribut L.TAILLE, qui contient le nombre d'éléments de L. Spécifiez comment les méthodes précédentes doivent être modifiées suite à l'introduction de cet attribut.

1.2 Ajouts

De quel attribut devrait-on disposer pour pouvoir ajouter un élément à la fin de la liste en temps constant ? Quelles sont les modifications à apporter aux méthodes précédentes dans ce cas ? Attention ! Toutes les méthodes peuvent être impactées.
Donnez un algorithme qui insère un élement dans une liste triée. Essayez de trouver une solution qui n'a pas besoin de mémoriser dans une variable temporaire l'élément précédent (aide : la donnée associée à un élément peut être modifiée).

1.3 Fusion de listes triées

On suppose que les deux listes L_1 et L_2 sont triées par ordre croissant et que l'on dispose de la fonction booléenne *donnéePlusGrande*(x, y) qui est vraie si la donnée associée à x est plus grande que la donnée associée à y. Le but est d'arriver à fusionner les deux listes de façon à obtenir une seule liste triée.

- Considérez les deux listes dont les données sont les lettres suivantes : L_1 : a, c, g, j, k et L_2 : b, h, m, n. Représentez graphiquement les listes.
- Proposez une méthode pour insérer les éléments de L_2 dans L_1 pour que L_1 soit triée à la fin. Donnez deux versions, une qui vide la liste L_2 et une autre qui garde intacte la liste L_2.

2 Listes doublement chaînées

Comme vu dans le cours, une liste doublement chaînée est une liste qui, en plus de permettre l'accès au suivant d'un élément, permet l'accès au précédent d'un élément. Pour représenter cette nouvelle classe de liste, on introduit l'attribut x.PRÉCÉDENT, qui contient une référence à l'élément précédant l'élément x.

— Quel est l'intérêt de ce type de liste par rapport aux listes simplement chaînées ?

— Écrivez les méthodes suivantes en mettant éventuellement à jour les attributs précédents :

1. L.*supprimerPremier*() supprime le premier élément de L.
2. L.*vider*() supprime tous les éléments de L.
3. L.*ajouterAprès*(x, y) ajoute x dans L après y.
4. L.*supprimer*(x) supprime x dans L.

Ajout en Fin

On veut maintenant gérer le dernier élément de la liste. On pourrait introduire un attribut contenant le dernier élément, mais, comme vous l'avez montré, cela complique les méthodes.
Pour résoudre ce problème, on peut travailler avec des listes circulaires : on connaît le premier et le précédent du premier est le dernier élément de la liste.
Afin de résoudre les problèmes de l'existence d'une donnée dans la liste, on introduit un élément fictif que l'on appelle *sentinelle*. Le premier élément réel de la liste devient donc le suivant de la sentinelle et le dernier élément de la liste est le précédent de la sentinelle. On peut donc supprimer l'attribut PREMIER. On le remplace par l'attribut SENTINELLE.

— Faites un dessin qui montre cela.

— Écrivez les méthodes suivantes en mettant éventuellement à jour les attributs précédents :

1. L.*premier*() renvoie le premier élément de L.
2. L.*dernier*() renvoie le dernier élément de L.
3. L.*estVide*() renvoie vrai si la liste est vide et faux sinon.
4. L.*supprimerPremier*() supprime le premier élément de L.
5. L.*vider*() supprime tous les éléments de L.
6. L.*ajouterAprès*(x, y) ajoute x dans L après y.
7. L.*supprimer*(x) supprime x dans L.

3 Opérations sur les listes

3.1 Listes circulaires avec sentinelle

On considère que L_1 et L_2 sont deux listes doublement chaînées circulaires avec sentinelle.

— Faites un dessin qui montre comment vous allez ajouter L_2 à la fin de L_1.

— Écrivez ensuite la méthode L_1.*ajouterEnFin*(L_2).

— Faites la même chose pour ajouter L_2 au début de L_1. On nommera la méthode L_1.*ajouterAuDébut*(L_2).

— Écrivez une méthode qui supprime de L_1 tous les éléments dont la donnée associée est impaire (on utilisera la fonction *donnéeEstPaire*(x)) et qui ajoute tous ceux de L_2 dont la donnée associée est paire.

3.2 Déplacement de sous-listes

Soit L une liste doublement chaînée contenant les éléments $[x_1, \ldots, x_k, \ldots x_n]$.
Écrire une méthode L.*déplacerEnFin*(x_k), pour déplacer la sous-liste constituée par les élements $[x_1, \ldots, x_k]$ en fin de liste et ainsi obtenir $[x_k, \ldots, x_n, x_1, ...x_{k_1}]$. Cette opération doit être effectuée en temps constant.