Projet: Listes chaînées sous stéroïdes

Consignes

Ce TP est un mini-projet évalué, et est à rendre au plus tard le 29/03/2026 à 23h59. Les rendus sont à envoyer dans une archive nom_prenom.zip à l'adresse `fpirot@lisn.fr`. Celle-ci contiendra les fichiers bobliste.py et bobliste2.py, ainsi qu'un document pdf de réponse aux questions. Ce dernier devra, pour chaque fonction implantée, expliquer son fonctionnement et justifier sa complexité, en plus de répondre aux questions explicites du sujet.

Un squelette de code est disponible à l'url , contenant des fonctions utilitaires pour les tests dans le fichier utilities.py, et l'implémentation des listes chaînées dans le fichier liste.py, dont vous pourrez vous inspirer pour réaliser ce TP. Les fichiers à remplir sont bobliste.py et bobliste2.py. Des tests sont déjà fournis dans un bloc dédié ; vous pouvez en ajouter **dans ce même bloc**. **Ne renommez pas les fonctions** : elles seront importées par un programme d'évaluation automatique de votre code qui déterminera la moitié de votre note. L'autre moitié découlera de l'évaluation de votre rapport.

Les retards entraîneront un malus de 2 points par tranche de 12h entamée (autrement dit, rendez vos projets à l'heure !).

Bob est un étudiant qui suit un cours d'algorithmique, et qui aime beaucoup la structure de liste chaînée. Il trouve formidable le fait de pouvoir insérer un élément en tête de liste en temps constant, mais il est frustré par le coût $\Theta(i)$ nécessaire pour accéder au i-ème élément d'une liste. Il décide donc d'enrichir la structure de liste afin d'améliorer cette dernière complexité.

Bob a une idée pour cela. Afin d'accélérer l'accès au i-ème élément d'une liste principale `lst`, il propose de construire une liste secondaire plus courte (et donc plus rapide à parcourir), contenant des pointeurs vers des cases de la liste `lst` de plus en plus espacées à mesure qu'on s'approche du début de la liste. Concrètement, pour chaque case `c` en position $n - i^2$ dans `lst` (pour tout entier $i \geq 1$), il crée une case jumelle `c'` dans la liste secondaire, dont la valeur est un pointeur vers `c`. Pour éviter les confusions, nous appellerons *case* les cases de la liste principale, et *bobcase* les cases de la liste secondaire. Bob renseigne la position de la première case à l'aide d'un pointeur `head` et de la première bobcase à l'aide d'un pointeur `bobhead` dans une structure qu'il baptise `Bobliste`. Il enregistre également la taille de la bobliste dans un champ `n` dédié, qui correspond au nombre de cases (dans la liste principale). Voici la définition de cette structure, et une illustration en est donnée dans la Figure 1.

```
structure Bobliste:
    n: entier           // nombre de cases dans la liste principale
    head: Cell          // pointeur vers la première case de liste principale
    bobhead: Cell       // pointeur vers la première case de la liste secondaire

structure Cell:
    val: valeur         // valeur dans la case
    next: Cell          // pointeur vers la case suivante
```

On crée une bobliste vide `lst` avec l'instruction `lst = Bobliste()`, et une case `c` de valeur v avec l'instruction `c = Cell(v)`. On a également accès à la fonction `isqrt(n)` qui prend en entrée un entier `n` et renvoie $\lfloor \sqrt{n} \rfloor$ en temps $O(1)$ (en supposant n de taille bornée).

Dans la suite, `lst` désigne systématiquement une bobliste de taille n.

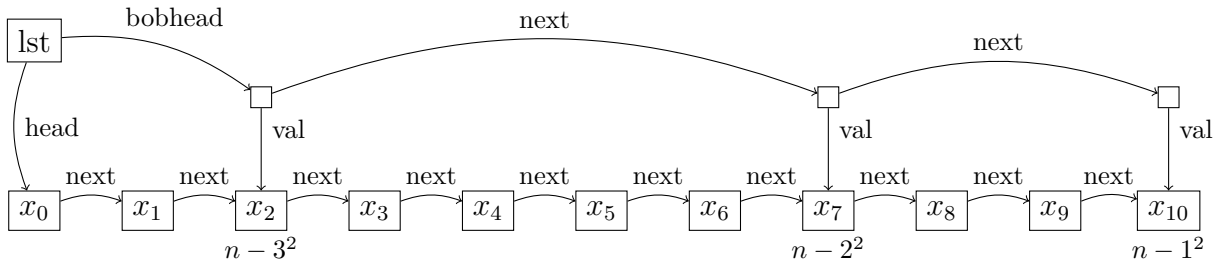

FIGURE 1 – Exemple d'une structure de bobliste de taille $n = 11$.

1. Dans une bobliste de taille n, quel est l'indice de la case pointée par la première bobcase ? À l'aide de cette observation, implanter une fonction `getElement(lst,`i`)` prenant en entrée une bobliste `lst` et une position $i \in \{0, \dots, n-1\}$, et qui renvoie la valeur de la i-ème `case` de `lst` en suivant le moins de pointeurs `next` possible. Justifier que sa complexité est $O(\sqrt{n})$ dans le pire cas.
2. Implanter une fonction `add(lst,`x`)` de complexité $O(1)$ qui ajoute une nouvelle `case` de valeur `x` au début de la bobliste `lst`. Attention à bien créer une nouvelle `bobcase` lorsque le nombre d'éléments l'impose. Puis implanter l'opération inverse `pop(lst)` qui supprime la première case et renvoie sa valeur si elle existe, ou `None` autrement.
3. Selon vous, pourquoi Bob a-t-il choisi d'avoir un espacement non constant entre les cases pointées par des bobcases ? Et pourquoi les cases pointées sont-elles plus rapprochées à la fin et pas au début de la liste principale ?
4. Dessiner la bobliste obtenue à partir de celle de de la Figure 1 si on ajoute une nouvelle case de valeur $x_{4.5}$ entre celles de valeur x_4 et x_5. Puis dessiner celle obtenue en ajoutant une case supplémentaire de valeur $x_{7.5}$ entre celles de valeur x_7 et x_8.
5. On suppose maintenant que la liste principale de la bobliste `lst` est triée. Implanter une fonction `insertSorted(lst,x)` de complexité $O(\sqrt{n})$ qui insère une nouvelle `case` de valeur `x` dans la bobliste `lst`, de sorte que cette dernière reste triée. Attention à bien mettre à jour les bobcases affectées par cette insertion (aidez-vous des dessins de la question précédente).
6. Bob souhaite maintenant pouvoir supprimer efficacement des cases de sa bobliste. Il se rend compte qu'il n'y parviendra pas avec sa structure actuelle, et qu'il lui faudra pour cela utiliser une liste doublement chaînée pour la liste principale (la liste secondaire peut, elle, rester simplement chaînée). Il ne renseigne cependant pas de pointeur vers la fin de la liste, convaincu qu'il n'en aura pas besoin. La structure correspondante est la suivante.

```
structure Bobliste:
    n: entier           // nombre de cases dans la liste principale
    head: Cell2         // pointeur vers la première case de liste principale
    bobhead: Cell       // pointeur vers la première case de la liste secondaire

structure Cell2:
    val: valeur         // valeur dans la case
    next: Cell2         // pointeur vers la case suivante
    prev: Cell2         // pointeur vers la case précédente
```

Compléter le fichier bobliste2.py avec les implantations correspondantes des précédentes fonctions, ainsi qu'une fonction `remove(lst,x)` qui supprime la première case dont la valeur est `x` dans la bobliste `lst`, en temps $O(\sqrt{n})$.