I Recherche de doublon

I.1 Doublon dans un tableau

Soit t un tableau de taille n dont les éléments sont entre 0 et $n-1$ (inclus).
On veut déterminer si t contient un doublon, c'est-à-dire un élément apparaissant plusieurs fois.

1. Donner un algorithme en complexité temporelle $\mathrm{O}(n)$ pour résoudre ce problème. Quelle est la complexité spatiale ?
2. Peut-on adapter l'algorithme précédent si les éléments de t ne sont pas entre 0 et $n-1$? pas forcément entiers ?
3. On reprend l'hypothèse où les éléments de t sont entre 0 et $n-1$.
   Décrire un algorithme en complexité $\mathrm{O}(n)$ en temps et $\mathrm{O}(1)$ en mémoire. On pourra modifier t.

I.2 Cycle dans une liste chaînée

On considère un type `linked_list` de liste simplement chaînée impérative (chaque élément a accès à l'élément suivant `next`) :

```
type 'a cell = { elem : 'a, next : 'a linked_list }
and 'a linked_list = E | C of 'a cell
```

Il est possible qu'une liste chaînée `l` possède un cycle, si l'on revient sur le même élément après avoir parcouru plusieurs successeurs.

4. Décrire un algorithme naïf pour tester si `l` contient un cycle. Quelle est sa complexité en temps et en espace ?

L'algorithme de Floyd est plus efficace. Il consiste à initialiser une variable `tortue` au premier élément de `l`, une variable `lievre` à la case suivante, puis, tant que c'est possible :

- Si `lievre` et `tortue` font référence à la même case, affirmer que `l` contient un cycle.
- Sinon, avancer `lievre` de deux cases et `tortue` d'une case.

5. Montrer que cet algorithme permet bien de détecter un cycle dans `l`. Quelle est l'intérêt de cet algorithme par rapport à celui de la question 4 ?
6. Écrire une fonction `cycle : 'a linked_list -> bool` détectant un cycle en utilisant l'algorithme du lièvre et de la tortue.
7. Expliquer comment obtenir la longueur T du cycle ainsi que le nombre d'itérations L avant d'entrer dans le cycle.

Soit t un tableau contenant n entiers entre 0 et $n-2$ (inclus).

8. Montrer que t contient un doublon.
9. Expliquer comment utiliser l'algorithme de Floyd pour trouver un doublon de t en complexité $\mathrm{O}(n)$ en temps et $\mathrm{O}(1)$ en mémoire, sans modifier t.

I.3 Presque doublon

On considère un nouveau problème :

Entrée : un tableau t de n entiers et deux entiers a, b.
Sortie : un booléen indiquant s'il existe deux indices $i \neq j$ tels que $|i-j| \leq a$ et $|t.(i) - t[j]| \leq b$.

10. Décrire un algorithme en complexité temporelle $\mathrm{O}(n)$ en utilisant une table de hachage.