Première partie – Savoir coder avec la notion d'ensemble.

EXERCICE 1 – LES OPPOSÉS S'ATTIRENT

Écrire une fonction qui étant donné un tableau d'entiers détermine en renvoyant un booléen s'il existe deux éléments x et y dans le tableau tels que $x = -y$.

La complexité de cette fonction doit être linéaire en la taille du tableau.

EXERCICE 2 – SUITE DE VAN ECK

La suite de Van Eck (v_n) se définit comme suit. Elle commence par $v_0 = v_1 = 0$.

Pour $i \geq 1$, le terme v_{i+1} se calcule en se demandant où v_i est apparu pour la dernière fois dans la liste des termes précédents :

- Si v_i est la première occurrence de cette valeur dans la suite, alors on définit $v_{i+1} = 0$.
- Si v_i est apparu pour la dernière fois en position j avec $j < i$ (on a donc $v_i = v_j$), alors on définit v_{i+1} comme étant égal à la différence entre ces deux positions, à savoir $i - j$.

Pour clarifier, calculons les termes suivants :

- **Calcul de v_2.** Le terme précédent $v_1 = 0$, est déjà apparu en position 0 : $v_0 = 0$. Donc $v_2 = 1 - 0 = 1$.
- **Calcul de v_3.** Le terme précédent est $v_2 = 1$. C'est la première fois que 1 apparaît dans la liste. Donc $v_3 = 0$.
- **Calcul de v_4.** Le terme précédent, $v_3 = 0$, est apparu pour la dernière fois deux positions auparavant, donc $v_4 = 2$.

On obtient ainsi comme premiers termes :

$$0, 0, 1, 0, 2, 0, 2, 2, 1, 6, 0, 5, 0, 2, 6, \ldots$$

Écrire une fonction `Van_Eck` qui étant donné un entier n renvoie le n-ième terme de la suite de Van-Eck. La complexité de cette fonction doit être en $O(n)$.

EXERCICE 3 – ALGORITHME MYSTÈRE (CT 2021)

Q1. Que renvoie la fonction `mystere` suivante ?

```
Fonction mystere(tab : tableau d'entiers)
        n = taille de tab
        E = ensemble vide
        Pour i allant de 0 à n-1
        Faire   Pour j allant de 0 à i-1
                Faire   Si (tab[i] == tab[j]) et (tab[i] n'appartient pas à E)
                        Alors   Ajouter tab[i] à E
                        FinSi
                FinFaire
        FinFaire
        Renvoyer E
```

Q2. Quelle est la complexité asymptotique des lignes suivantes (qu'on retrouve dans la fonction `mystere`) ? Justifiez.

```
Si (tab[i] == tab[j]) et (tab[i] n'appartient pas à E)
        Alors   Ajouter tab[i] à E
FinSi
```

Q3. Quelle est la complexité asymptotique de la fonction `mystere` ? Justifiez.

EXERCICE 4 – INTERSECTION DE DEUX LISTES (CT 2021 DEUXIÈME SESSION)

Q1. Écrire une fonction `intersection` qui prend en paramètres deux tableaux d'entiers `tab1` et `tab2` et qui renvoie le nombre d'éléments qui sont à la fois dans `tab1` et `tab2`.

Par exemple `intersection([15,43,11,65],[11,86,15,2,15,6])` renvoie `2` car deux seuls nombres se trouvent à la fois dans les deux tableaux : `11` et `15`. Attention, les doublons ne comptent que pour une fois.

Bien sûr, votre fonction doit être la plus efficace que possible !

Q2. Quelle est la complexité asymptotique de votre fonction `intersection` ? Justifiez.

EXERCICE 5 (CHALLENGE) – PLUS LONGUE SOUS-CHAÎNE SANS RÉPÉTITION D'UN CARACTÈRE

On considère le problème suivant : on souhaite écrire une fonction qui prend en paramètre une chaîne de caractères `ch` et qui renvoie la longueur de la plus longue sous-suite de `ch` qui ne contient pas deux fois le même caractère.

Par exemple, si `ch` vaut `"babcbbdb"`, la fonction doit renvoyer `3` car `"abc"` est une sous-chaîne sans répétition. Il n'y a pas de sous-chaînes de longueur ≥ 4 qui ne contient pas deux fois la même lettre. (Par exemple, `"abcb"` contient deux fois "`b`" et `"abcd"` n'est pas considéré comme une sous-chaîne.) Autre exemple, sur l'entrée `"zzzzz"`, la fonction doit renvoyer `1`.

Q1. Ecrire de manière naïve une fonction qui teste si la sous-chaîne de `ch` commençant à l'indice `i` et finissant à l'indice `j` contient deux fois le même caractère.

Q2. En utilisant la fonction précédente, en déduire un algorithme naïf qui répond au problème.

Q3. Quelle est la complexité de l'algorithme naïf ?

Q4. Ecrire un algorithme qui marche et qui est de complexité $O(n^2)$. Indication : Faites varier une variable f de 1 à la dernière position du tableau, et essayez de décrire la sous-chaîne de longueur maximale sans répétition qui termine en f à partir de la sous-chaîne de longueur maximale sans répétition qui termine en $f-1$.

Q5. Améliorer la question 1 de sorte à que ce soit un peu moins naïf, en utilisant une table de hachage. (Si vous aviez déjà utilisé une table de hachage, bravo !)

Q6. Quelle est du coup maintenant la complexité de l'algorithme de la question 2 ?

Q7. Cramez les derniers neurones qui vous restent en essayant de trouver un algorithme qui est linéaire en n ! Si vous avez réussi à faire la question 4 de l'exercice 3 sans table de hachage, il s'agit d'améliorer cet algorithme avec une table de hachage.

Deuxième partie – Table de hachage

Cette partie peut être vue à la fois comme un TD sur feuille (en pseudo-code), ou un TP sur machine (en C). Il s'agit d'implémenter "à la main" le type abstrait "ensemble" (`set` en Python) via une table de hachage. Si vous souhaitez le faire sur machine, il est recommandé de l'écrire en C.

EXERCICE 6 – IMPLÉMENTATION DU TYPE "ENSEMBLE D'ENTIERS"

On définira deux structures :

```
Structure Noeud {
        valeur : entier
        suivant : pointeur vers Noeud
}

ListeChainee = pointeur vers Noeud
```

et

```
Structure Ensemble {
        table : tableau de ListeChainee
        taille : entier
}
```

En C, cela s'écrit :

```
struct Noeud{
        int valeur;
        struct Noeud* suivant;
};

typedef struct Noeud* ListeChainee;


struct Ensemble{
        ListeChainee* table;
        int taille;
};
```

Petite liste des opérations qu'il sera possible de faire en pseudo-code:

- Allouer de la mémoire pour un ensemble
- Allouer de la mémoire pour un noeud de liste chaînée
- Allouer un tableau de taille donnée
- Accéder et modifier à un membre d'une structure. Par exemple :
  - `Pour i allant de 0 à E.taille - 1`
  - `E.table[3] = pointeur nul`
  - `L = L.suivant`
  - ...

Q1. Ecrire une fonction `creeEnsemble` qui prend en paramètre un entier[1] `n` et qui renvoie un ~~élément de structure Ensemble de taille~~ `n`.

[1] Pourquoi impose-t-on ici une taille pour notre table de hachage, alors qu'en python ou en Java, quand on crée un ensemble, on ne précise jamais la taille ? Et bien, `python` ou `java` calcule automatiquement la taille de la table de hachage de sorte que chaque opération élémentaire associée à la notion d'ensemble se fasse en temps constant. Plus précisément, si le nombre d'éléments dans la table de hachage devient trop grand ou trop petit, on alloue une nouvelle table de hachage de taille plus adéquate, puis on recopie les valeurs de l'ancienne table de hachage vers la nouvelle. J'arrête les détails : cela reste un peu trop technique pour que ce soit réellement demandé dans le cadre de cet exercice.

Q2. Ecrire une fonction `affiche` qui prend en paramètre un ensemble E et qui affiche tous les éléments dans E.

Q3. On va considérer la fonction de hachage :

$$h : x \mapsto (15073 \times x) \text{ modulo (taille de la table de hachage)}$$

Autrement dit, quand on va insérer un entier x dans l'ensemble, on va allouer un nouveau noeud contenant x et mettre l'adresse mémoire de ce noeud à la position $h(x)$ dans la table de hachage associée.

Écrire une fonction `ajout` qui prend en paramètre un ensemble `E` et un entier `x`, et qui rajoute dans `E` l'entier x. (On rajoutera une occurrence de x même si x est dans E)

Q4. Écrire une fonction `appartient` qui prend en paramètre un ensemble `E` et un entier `x`, et qui renvoie `VRAI` si `x` appartient dans `E` ; `FAUX` sinon.

Q5. Écrire une fonction `supprime` qui prend en paramètre un ensemble `E` et un entier `x` (on supposera que E contient bien x), et qui supprime de E une occurrence de x.