EA4 – Éléments d'algorithmique
TP n° 5 : tables de hachage

Vous téléchargerez sur Moodle les fichiers `tp5_ex*.py` à compléter. Le TP est à déposer sur Moodle avant vendredi 7 mai au soir.

Exercice 1 :

Le but de cet exercice est de comparer les temps d'accès à une liste Python (`list`) et à un ensemble Python (`set`).

1. Écrire la fonction `cherche` qui recherche `x` dans `I`, où `I` est un itérateur et `x` un élément.
2. Écrire la fonction `nb_elts_diff_liste` qui, étant donné une liste `L`, renvoie le nombre d'éléments distincts dans `L`. Votre algorithme doit avoir une complexité en $O(n \log n)$.
3. Écrire la fonction `nb_elts_diff_ens` qui, étant donné un ensemble `E`, renvoie le nombre d'éléments distincts dans `E`.

On utilise la structure de données vue en td pour représenter une table de hachage :

`[cles, h, taille, tmin, tmax, nbCles]`,

où `cles` est un tableau de clés dont chaque case peut prendre les valeurs :

- `None` (case vide),
- `MARQUE = (None,None)` (case où une clé a été supprimée),
- `cle` où `cle` est une clé,

`h` est une fonction de hachage, `taille` est la longueur du tableau de clés, `tmin` (resp. `tmax`) est le taux minimal (resp. maximal) de remplissage et `nbCles` est le nombre de clés du tableau.

Exercice 2 :

Vous allez écrire les fonctions usuelles d'accès aux tables de hachage avec adressage ouvert et sondage linéaire ou double hachage. Dans un premier temps, vous ne vous préoccuperez pas de redimensionnement (c'est-à-dire qu'il faudra commencer avec une table de taille suffisamment grande).

1. Écrire la fonction `creer_table` qui, étant donné un entier `p`, une fonction de hachage `h` et deux réels `tmin`, `tmax` compris entre 0 et 1, renvoie une table de hachage dont le tableau `cles` possède `taille` cases vides (`None`) où `taille` vaut 2^p.

   Une fonction de hachage est définie comme une liste de deux fonctions `[h1, h2]`. Le haché d'une clé `k` est alors égal à `(h1(k)+i*h2(k)) % t` pour un `i` donné, où `t` est la taille de la table. Pour ensuite itérer sur les positions possibles dans la table pour une clé donnée `k`, il faut invoquer la fonction `gen_hash` (définie dans `tp5_ex2_ex3.py`) de la façon suivante :

   ```
   for pos in gen_hash(h1, h2, k, t):
       ...
   ```

2. Définir les fonctions `hash1(k, t)` et `hash2(k, t)` qui implémentent la fonction de hachage $h(k, i) = k + i \mod t$.

3. Écrire la fonction `rechercher(table, cle, flag)` qui peut avoir deux comportements différents selon la valeur de `flag` :
   - quand `flag` vaut `False`, elle renvoie `None` si la clé `cle` ne se trouve pas dans le tableau de clés de la `table` ; sinon elle renvoie son indice dans le tableau de clés de la `table` ;
   - quand `flag` vaut `True`, elle renvoie l'indice où doit être insérée la clé si celle-ci ne se trouve pas dans la table ; sinon elle renvoie l'indice de la clé dans le tableau de clés de la table.
4. Écrire la fonction `inserer` qui, étant donné une table de hachage et une clé, insère la clé dans la table (sans créer de doublon).
5. Écrire la fonction `supprimer` qui, étant donné une table de hachage et une clé, supprime la clé de la table. Une case où une clé est supprimée prend la valeur `MARQUE`.
6. Afin de faire des tests avec différentes fonctions de hachage, définir les fonctions `hash3` et `hash4` en plus des fonctions `hash1` et `hash2` afin d'implémenter les fonctions de hachage suivantes (t représente la taille de la table) :
   - $h(k,i) = \{kA\} \times t + i \mod t$,
   - $h(k,i) = (k + i(2k+1)) \mod t$,
   - $h(k,i) = (\{kA\} \times t + i(2k+1)) \mod t$,

   où $\{r\}$ représente la partie fractionnaire du réel r et A est la constante définie au début du fichier `tp5_ex2_ex3.py`.
7. Dans le `main`, compléter la liste de fonctions de hachage afin de comparer les temps d'insertion et de recherche, ainsi que la taille moyenne du plus grand cluster. Il faudra pour cela décommenter les deux premiers appels à `courbes`. Les trois premiers graphiques correspondent au hachage sur des listes qui comportent des valeurs proches sur différents intervalles, les trois suivants correspondent au hachage sur des listes formées de valeurs uniformément distribuées. Que constate-t-on ?

   Jouer avec les paramètres `tmin` et `tmax`, et regarder l'évolution de vos courbes pour les deux types de listes.

Exercice 3 :

Écrire la fonction `redimensionner` qui, étant donné une table de hachage et une taille `t`, redimensionne la table à la taille `t`. Modifier ensuite, la fonction `inserer` pour qu'elle double la taille de la table si nécessaire et la fonction `supprimer` pour qu'elle divise la taille de la table si nécessaire. Observer de nouveau les résultats sur les courbes et jouer de nouveau avec les paramètres `tmin` et `tmax`. Pour cela, il faudra décommenter les deux derniers appels à `courbes`. Que constate-t-on ?

Exercice 4 :

Le but de cet exercice est de créer et manipuler une structure de données pour stocker un ensemble de mots. Il faudra essentiellement utiliser les fonctions écrites précédemment.
Le fichier `proust.txt` sur Moodle contient le roman de Marcel Proust, *À la recherche du temps perdu*. La fonction `proust()` renvoie un générateur des mots du roman et s'utilise de la façon suivante :

```
for mot in proust():
    ...
```

1. Écrire la fonction `mot_to_int(w)` qui, étant donné un mot $w = w_0 \dots w_n$, renvoie l'entier $h(w) = \sum_{i=0}^{n} c_i \ 31^{n-i} \mod 2^{32}$, où c_i est égal au code ascii de w_i (qui peut être obtenu à l'aide de la fonction `ord`).
2. À l'aide des fonctions écrites précédemment, écrire des fonctions permettant de représenter un ensemble de mots par une table de hachage :
   a. `creer_dico(taille=0)` qui crée un ensemble de mots vide,
   b. `ajouter_mot(dico, mot)` qui ajoute le mot `mot` à l'ensemble de mots `dico`,
   c. `retirer_mot(dico, mot)` qui supprime le mot `mot` de l'ensemble de mots `dico`,
   d. `dans_dico(dico, mot)` qui renvoie vrai si le mot `mot` est dans l'ensemble de mots `dico`.

   Pour éviter de multiplier inutilement les appels à `mot_to_int()`, il pourra être judicieux de stocker dans la table des *couples* `(mot, entier)` et non simplement des mots.
3. Écrire les tests pour tester vos fonctions sur l'ensemble des mots du roman de Marcel Proust.