EA4 – Éléments d'algorithmique
TP n° 10 : tables de hachage

Vous téléchargerez sur Moodle les fichiers `tp10_ex*.py` à compléter. Les exercices 1, 2 et 3 sont à déposer sur Moodle sous forme d'une archive avant le dimanche 15 avril 16h00.

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`, retourne 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`, retourne le nombre d'éléments distincts dans `E`.

On utilise la structure de données (à peu près celle vue en td) pour représenter une table de hachage : `[cles, h, taille, taux, nbCles]`, où `cles` est un tableau de clés où chaque case peut prendre les valeurs `None` (case vide), `(None,None)` (case où un couple a été supprimé), `(h[0](cle),cle)` où `cle` est une clé (case contenant une clé et son premier haché), `h` est une fonction de hachage, `taille` est la taille du tableau de clés, `taux` est le taux 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 un réel `taux` compris entre 0 et 1, retourne une table de hachage `[cles, h, taille, taux, 0]` où `cles` est un tableau de `taille` cases `None` avec `taille` égale à 2^p.
2. 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 `tp10_ex2.py`) de la façon suivante :

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

   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` :
   - si `flag` vaut `False`, retourne `None` si la clé `cle` ne se trouve pas dans le tableau de clés de la table de hachage. Sinon elle retourne l'indice de la clé dans le tableau de clés de la table. La constante `MARQUE` est égale à `(None,None)` et marque donc une case où une clé a été supprimée.

- si `flag` vaut `True`, retourne l'indice où doit être insérée la clé si celle-ci ne se trouve pas dans la table. Sinon elle retourne 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.

5. Écrire la fonction `supprimer` qui, étant donnée une table de hachage et une clé, supprime la clé de la table. Une case où un couple (haché, clé) est supprimé prend la valeur `MARQUE`.

6. Afin de faire des tests avec différentes fonctions de hachage, définir les fonctions nécessaires qui implémentent les fonctions de hachage suivantes (t représente la taille de la table) :

- $h(k,i) = \{kA\} \times t + i \mod t$,
- $h(k,i) = (k + i(2k+1)) \mod t$,
- $h(k,i) = (\{kA\} \times t + i(2k+1)) \mod t$,

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 consécutives sur différents intervalles, les trois suivants correspondent au hachage sur des listes formées de valeurs uniformément distribuées. Que constatez-vous ?

Jouez avec le paramètre `taux` 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. Observez de nouveau les résultats sur les courbes et jouez de nouveau avec le paramètre `taux`. Pour cela, il faudra décommenter les deux derniers appels à `courbes`. Que constatez-vous ?

Exercice 4 :

Le but de cet exercice est de créer et manipuler une structure de données pour stocker un ensemble de mots.
Le fichier `mots.txt` sur Moodle contient une liste de 5000 mots.

1. Écrire la fonction `hash_mot(w)` qui, étant donné un mot $w = w_0 \dots w_n$ retourne le calcul de $h(w) = \sum_{i=0}^{n} c_i \ 31^{n-i} \mod 32$, où c_i est égal au code ascii de w_i.

2. En utilisant les fonctions de l'exercice 2, écrire les fonctions :

a. `creer_ens_mots(taille=0)` qui crée un ensemble de mots,

b. `ajouter(S,w)` qui ajoute le mot `w` à l'ensemble de mots `S`,

c. `retirer(S,x)` qui supprime le mot `w` de l'ensemble de mots `S`,

d. `dans_ens(S,x)` qui retourne vrai si le mot `w` est dans l'ensemble de mots `S`.