Table de hachage

Dans ce TP on implémente un table de hachage (ou table associative, ou dictionnaire) à l'aide d'un tableau de listes simplement chaînées.

Le but de l'exercice est d'implémenter une structure de données appelée *tableau associatif*, aussi appelé *table de hachage* (similaire aux dictionnaires de Python par exemple) afin de de compter le nombre d'apparitions de chaque mot d'un texte. On supposera qu'un mot est une suite d'au plus de 50 caractères, (lue par exemple par `scanf("%50s", mot)`).

Pour permettre un accès rapide aux mots, on utilise une méthode appelée *hachage ouvert* (ou *adressage séparé*). Elle consiste à ranger les mots dans un tableau de `N` listes chaînées. En pratique, si `mot` est un `char[]` que l'on souhaite insérer dans notre table de hachage, on s'aide d'une fonction auxiliaire `int hachage(int N, char word[])` qui renvoie une valeur comprise entre `0` et `N-1`, puis on insère `mot` dans le tableau d'indice `hachage(mot)`.

Pour plus d'efficacité, chaque liste sera maintenue triée dans l'ordre croissant des mots. L'ordre entre deux mots est fourni par la fonction `strcmp(char *, char *)` et la longueur d'un mot est fournie par la fonction `strlen(char *)`. En pratique, pour stocker un mot de longueur n, il faut prévoir d'écrire $n+1$ caractères, dont le dernier sera `'0'`, c'est-à-dire le caractère réservé aux fins de mots. Pour plus d'information, on pourra consulter `man strcmp` et `man strlen`.

La recherche d'un mot dans la table de hachage est plus efficace que dans une simple liste chaînée, car elle se ramène au calcul de la fonction de hachage puis à la recherche dans une liste (courte), plutôt qu'au parcours d'une très longue liste chaînée. Si `N` est bien choisi, on obtient alors de très bonnes performances en pratique.

Pour implémenter les listes chaînées et la table de hachage, on utilisera les types suivants :

```c
typedef struct celmot {
  char *mot;          /* mot stocké localement */
  int nombre;         /* nombre d'apparitions du mot */
  struct celmot *suivant;
} CelluleM, *ListeM;

typedef struct hashtable {
  int N;              /* nombre de liste chaînées */
  int nbMots;         /* nombre de mots distincts dans la table de hachage*/
  ListeM *tableau;
} HashTable, *Dico;
```

1. Écrire une fonction `Dico creeDico(int n)` qui crée un dictionnaire contenant `n` liste chaînées, toutes vides (et ne contient donc aucun mot).
2. Écrire une fonction `ListeM alloueCellule(char mot[], int nb)`. La nouvelle cellule est créée en utilisant la place mémoire minimale pour recopier `mot` et un nombre d'apparitions égal à `nb`.
3. Écrire une fonction `int ajouteListe(ListeM *L, char mot[], int nb)` qui ajoute `nb` occurrences du mot mot `mot` à la liste triée `*L` : on renvoie `0` si le mot était déjà présent dans la liste, et `1` sinon.
4. Écrire une fonction `int hachage(int N, char word[])` qui renvoie la somme :

$$\left(\sum_{i=0}^{n-1}(i+1)\,\texttt{word[i]}\right) \mod N$$

où n est la taille de la chaîne `word` et où chaque caractère est interprété comme un entier égal à son code ASCII.

5. Écrire une fonction `int rechercheDico(Dico dico, char mot[])` qui renvoie le nombre d'apparitions du mot `mot` dans le dictionnaire `dico`.
6. Écrire une fonction `void ajouteDico(Dico dico, char mot[], int nb)` qui insère `nb` occurrences du mot `mot` dans le dictionnaire `dico`.
7. Écrire une fonction `int nombreMots(Dico dico)` qui renvoie le nombre total de mots mémorisés dans `dico` (en comptant toutes les apparitions de chaque mot).
8. On veut écrire un programme qui lit les mots d'un fichier texte et crée un nouveau fichier texte contenant la liste de ces mots et le nombre d'apparitions de chacun d'entre eux. Le fichier de sortie contiendra un mot par ligne, les mots apparaissant en ordre quelconque.

   La référence du fichier dans lequel on lit le texte et celle du fichier dans lequel on écrit les mots du texte seront fournis comme paramètres sur la ligne de commande. Ainsi si l'exécutable a pour nom `Compte`, l'exécution de la ligne `$ Compte Examen Examen.cpt` entraînera la création du fichier `Examen.cpt`. Si le contenu du fichier `Examen` est

   ```
   Examen d'informatique tres tres facile.
   ```

   alors celui fichier `Examen.cpt` devra être (à l'ordre des lignes près) :

   ```
   tres 2
   Examen 1
   d'informatique 1
   facile. 1
   ```

   Écrire la fonction `main` permettant ce traitement.
9. Comment choisir la fonction de hachage pour n'avoir qu'une seule liste chaînée ? Comment se comporte le programme sur un texte très long dans ce cas ?
10. Tester différentes fonctions de hachage, mesurer le temps d'exécution obtenu, et comparer la taille des listes construites :
    - nombre de listes vides,
    - nombre d'éléments dans la plus longue liste,
    - nombre moyen d'éléments dans les listes non vides, etc.
11. Modifier le programme afin que la taille des tables de hachage ne soit plus fixée par une constante `N` mais varie dynamiquement au cours de l'exécution, en fonction de la charge de la table (nombre total de mots différents qu'elle contient). On utilisera le principe des tableaux dynamiques vu au S3.