Département IMA / 3A (S6) / 2013 **Programmation Avancée**
Jonathan Aceituno, Vincent Aranéga, Nathalie Devésa, Rachid Elkouri et Laure Gonnord

TP8 Table de hachage, utilisation de bibliothèque perso

Objectifs

- Savoir utiliser une bibliothèque.
- Savoir construire une table de hachage et l'utiliser.

Contexte et préparation Dans ce TP, nous allons récupérer la bibliothèque construite au TP8 et l'utiliser pour construire et utiliser des tables de hachage. Classiquement, les tables de hachage étant utilisées pour coder des dictionnaires, nous allons réaliser un dictionnaire anglais sommaire.

1 Questions du TP (à faire impérativement)

1.1 Point sur le TP7 et récupération des fonctions fournies

Dans un répertoire `TP8`, *à la ligne de commande!* :

1. Copier votre librairie statique du TP8 (`listechaines.h` et `liblistechaines.a`). On n'a plus besoin du source. Si vous n'avez pas terminé, récupérer la librairie sur la page web du cours.
2. Copier le fichier `hash.c`, le fichier `dico.english` ainsi que le Makefile fournis dans le répertoire Polytech de Laure Gonnord à l'adresse :

   `/home/imaEns/lgonnord/PA2012/TP8/`

3. Vérifier que le Makefile fonctionne : commande `make`.
4. Lancer le programme (que fait-il ?) et noter son temps d'exécution (commande `time`).

1.2 Explications tables de hachages

Lire attentivement l'annexe.

1.3 Travail demandé

Dans le fichier fourni `hash.c` :

1. Déclarer le type `Hashtable` (tableau de taille `TABLE_SIZE de Liste`)
2. Écrire une fonction void `init_ht(Hashtable ht)` qui initialise `ht` (listes à `NULL`).

3. Écrire une fonction `void update_ht(char *word, Hashtable ht)` qui met à jour `ht` avec le mot en l'ajoutant (fonction `ajoutAlphab` fournie) à la liste d'indice `hash(word)`. La fonction de hachage est fournie dans le .c.

4. Écrire une fonction `void load_ht(FILE *fp, Hashtable ht)` analogue à `chargeFichier` qui charge les mots du fichier dans `ht`.

5. Modifier le `main` en remplaçant la liste par une `Hashtable`.

6. Comparer les temps d'exécution avec la liste (version précédente)[1]. Expérimenter avec des valeurs progressives de `TABLE_SIZE:`
   `TABLE_SIZE=1` : vous devriez retrouver des performances très comparables...
   `TABLE_SIZE=10, 20, ..., 100, 200, 300, 400...`

7. Écrire une fonction `void collisions(Hashtable ht)` qui affiche pour chaque indice (valeur de hachage) le nombre de collisions dans la table, c'est à dire la taille de la liste correspondante (utiliser la fonction `taille(Liste l)` de librairie). Sur l'exemple (avec `TABLE_SIZE=50`) :
   ```
   0 335
   1 309
   2 308
   3 318
   4 340
   5 293
   ...
   ```

2 Questions s'il vous reste du temps

1. Ranger les valeurs irprimées par la fonction précédente dans un fichier `trace.txt`. Pour cela, mettez en commentaire tout autre affichage que ceux de la fonction `collisions` et rediriger la sortie du programme vers `trace.txt`. Visualiser la répartition des collisions avec `gnuplot`, et faire de même avec différentes valeurs de `TABLE_SIZE` (10, 50, 100, 200, 400):
   ```
   bash:  ./hash dico.english > trace.txt
   bash:  gnuplot
   gnuplot> plot ``./trace.txt'' with lines
   ```
   Remarquer que la courbe n'a plus l'air de changer après `TABLE_SIZE`≈250. Remarquer que quelques points "traînent" cependant autour de 310...

2. Déterminer la plus grande valeur de hachage possible du dictionnaire. Pour cela écrire une fonction:
   `void max_hash(FILE *fp, char *max_word, int *hmax)`
   qui détermine la plus grande somme de codes ascii des mots du dictionnaire, soit `hmax` (utiliser la fonction fournie `asciis(word)`) et renvoie dans `max_word` un mot vérifiant cela (il s'avère qu'il est unique).

3. Afficher `max_word et hmax`. Fixer `TABLE_SIZE à hmax+1`, vous devez retrouver `max_word` en affichant la liste de collisions d'indice `hmax` (utiliser la fonction `afficherListe` de la bibliothèque).

[1] N'oubliez pas de comparer ce qui est comparable en mettant en commentaire notamment les traces ou affichages éventuels.

Annexe : Tables de hachage

Une table de hachage est un compromis entre les listes chaînées efficaces sur les ajouts et les tableaux (ou listes contiguës) efficaces sur les accès. Soit un ensemble fini $\mathcal{D}$ de données (ici les mots du dictionnaire) que l'on va stocker dans un tableau `ht`. Pour ranger un élement `x` de $\mathcal{D}$, on calcule son image par une fonction de hachage `hash`, qui donne son indice (appelé "indice de hachage") dans `ht`, `x` sera donc rangé dans `ht[hash(x)]`. Il reste à trouver une fonction `hash` adéquate:

- l'idéal est de trouver une fonction bijective entre $\mathcal{D}$ et l'intervalle d'indices du tableau, ainsi chaque case du tableau contient 1 élément de $\mathcal{D}$. Mais il est difficile de trouver une telle fonction bijective, ne serait-ce que parce que $\mathcal{D}$ est rarement connu a priori.
- l'utilisation d'une fonction injective (ie qui associe à chaque `x` de $\mathcal{D}$ une case différente de $\mathcal{D}$) mène à un tableau de taille $sup\{hash(x),\ x \in \mathcal{D}\}$, c'est-à-dire à un tableau à trous, ce qui peut générer une perte importante de place.
- le principe consiste alors à prendre une fonction surjective, obtenue généralement par modulo sur une taille limitée de tableau, soit `TABLE_SIZE` $\ll card(\mathcal{D})$. On tombe alors sur un problème de "collisions" parce que plusieurs données peuvent avoir le même indice de hachage. On choisit donc une structure mixte formée d'un tableau statique de listes chaînées contenant les éléments de même indice de hachage (appelées "listes de collisions"). Si possible ces listes sont ordonnées pour optimiser les accès. L'ajout d'un élément `x` revient alors à calculer `hash(x)` de coût quasi constant et insérer `x` dans la liste `ht[hash(x)]`, en $o(n)$ n étant la taille de la liste correspondante. Toute la difficulté est de trouver un bon compromis entre la taille du tableau et la taille des listes. Remarquez que prendre `TABLE_SIZE=1` revient à faire une simple liste.

Exemple fourni : dictionnaire anglais Le dictionnaire anglais fourni, avec la fonction de hachage `int hash(char *word)` qui calcule la somme des codes ASCII des caractères du mot modulo `TABLE_SIZE` (implémentée dans `hash.c`), donne la hachtable suivante si `TABLE_SIZE=50` :

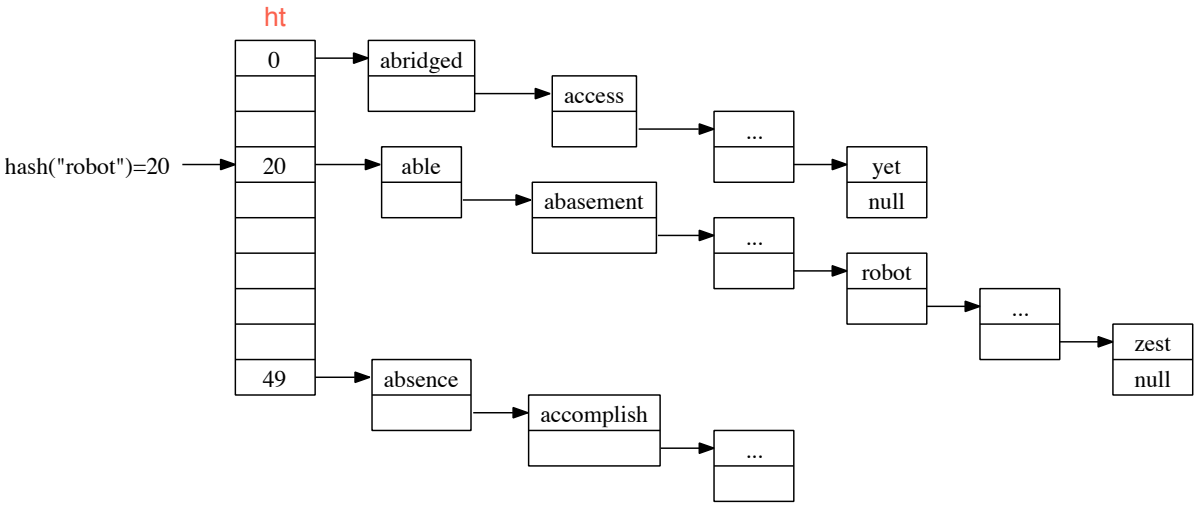

Figure 1: Table `ht` du dictionnaire anglais