TP n°22 – Hachage

1 Fonctions de hachage

Dans cette section on écrira des fonctions de hachage qui prennent en entrée m (la taille du tableau). Pour les tests vous pouvez prendre $m = 10$.

- **Q1.** Donner une fonction de hachage `int hachage_entier(int k, int m)` qui réalise le hachage des entiers vers $[|0, m-1|]$. (c'est du cours)

On va désormais considérer un hachage des chaines de caractères. Voici quelques éléments qui peuvent servir :

- On rappelle qu'en C une chaine de caractères est représenté par un tableau de caractères (donc par le type `char*`) qui se termine toujours par le caractère '\0'.
- La fonction `int strlen(char* ch)` permet de calculer la taille d'une chaine.
- La fonction `int strcmp(char* ch1, char* ch2)` permet de comparer deux chaines de caractères (résultat nul si ch1==ch2, positif si ch1>ch2 et négatif sinon).
- Pour obtenir le code ASCII d'une variable de type `char` , il suffit de le caster en `int`.
  Par exemple `char a = 'a'; int ia = (int)a;` donnera `ia=97`. Et réciproquement `int ia = 97; char a = (char)ia;` donnera `a='a'`. Faire un cast implicite fonctionne également, par exemple `char a = 'a'; int ia = a;`.

- **Q2.** Imaginer une fonction de hachage `int hachage_chaine(char* k, int m)` qui hache une chaîne de caractères. **Faites preuve d'imagination et ne copiez pas vos voisins**. Quelle est sa complexité ?
- **Q3.** Imaginer une autre fonction de hachage pour les chaines de caractères, mais cette fois sa complexité doit être constante peu importe la taille de la chaine.
- **Q4.** Utiliser la fonction `collisions` pour déterminer si vos fonctions de hachage répartissent les chaines de caractères sur les indices du tableau de manière assez uniforme.
  La fonction `collisions` crée des chaines aléatoires, teste leur haché et affiche m valeurs : la i-ème valeur est le pourcentage de chaines dont le haché est i. L'idéal est d'avoir à peu près le même pourcentage pour chaque i : distribution uniforme.
  Vous pouvez choisir 3 paramètres : `m`, `TAILLE_MAX` une borne sur la taille des chaines de caractères qu'on teste et `nb_essais` le nombre de chaines testées.

2 Implémentation d'un dictionnaire

Dans cette section on va écrire toutes les fonctions servant à implémenter un dictionnaire en C.

En Ocaml on aurait pu définir un type :

```
type ('a,'b) dico = {mutable stockage : ('a*'b) array; hash : 'a->int};;
```

En C les couples n'existent pas, donc on utilisera la structure suivante :

```
typedef struct couple{char* cle; float etiquette;} couple
```

De plus en C on ne sait pas écrire une structure de données dont un des champs est une fonction, donc on se contentera d'utiliser un tableau d'une taille m (variable globale) et une fonction de hachage, sans faire de lien entre les deux.

Remarque : le tableau est toujours de type `couple*` et de taille m, peu importe comment il est rempli. Pour indiquer qu'une case est vide, on utilisera le couple ("",0.0). (chaine vide et zéro)

- **Q5.** Écrire une fonction qui crée le dictionnaire vide.
- **Q6.** Écrire une fonction qui libère l'espace occupé par le dictionnaire.
- **Q7.** Écrire une fonction qui ajoute un couple (k, v) dans le dictionnaire.
- **Q8.** Écrire une fonction qui supprime toute référence à une clé k dans le dictionnaire. Si k ne s'y trouve pas, cette fonction ne fera rien.
- **Q9.** Écrire une fonction qui recherche la valeur associée à une clé k dans le dictionnaire.
- **Q10.** Écrire une fonction qui modifie la valeur associée à une clé k dans le dictionnaire pour une valeur v.

3 Adressage ouvert

On va améliorer l'implémentation précédente avec de l'adressage ouvert. Vous aurez besoin de plusieurs fonctions de hachage **différentes**, récupérez-en chez vos voisins. En avoir 3 serait bien.

En Ocaml on pourrait modifier le type précédemment mentionné avec un tableau de fonctions :

```
type ('a,'b) dico = {mutable stockage : ('a*'b) array; fonctions : ('a->int) array};;
```

À nouveau, comme en C on ne sait pas manipuler les fonctions comme des objets du langage (et les faire passer en argument d'une fonction, ou les mettre dans un tableau), l'implémentation de l'adressage ouvert réalisé dans cette partie sera non généralisable.

Les fonctions pour créer et supprimer un dictionnaire ne changent pas.

- **Q11.** Écrire une nouvelle fonction pour l'ajout, la suppression, la recherche et la modification, en implémentant l'adressage ouvert avec les 3 fonctions de hachage que vous avez récoltées.

Passez à la partie suivante et si vous avez terminé, reprenez votre implémentation avec un tableau redimensionnable. (comme on a intelligemment mis m en entrée des fonctions de hachage, vous n'avez pas besoin de les réécrire)

4 Le module Hashtbl de Ocaml

Ocaml propose un module pré-implémenté pour réaliser des tables de hachage.

Vous pouvez utiliser les fonctions suivantes, le type `('a,'b) t` désigne un dictionnaire avec des clés de type `'a` et des valeurs de type `'b` :

- `Hashtbl.create : int -> ('a,'b) t` permet de créer un dictionnaire vide. La fonction prend en entrée m la taille désirée pour le tableau. Cependant le module implémente des tables redimensionnables, donc ce n'est qu'une estimation et le module la changera si besoin.
- `Hashtbl.add : ('a, 'b) t -> 'a -> 'b -> unit` permet d'ajouter dans une table une clé et sa valeur.
- `Hashtbl.remove : ('a, 'b) t -> 'a -> unit` permet de retirer une valeur d'une table de hachage.
- `Hashtbl.mem : ('a, 'b) t -> 'a -> bool` vérifie si une clé est stockée dans une table de hachage.
- `Hashtbl.find : ('a, 'b) t -> 'a -> 'b` renvoie la valeur associée à la clé dans la table ou lève l'exception `Not_found` si la clé n'est pas dans la table.
- `Hashtbl.find_opt : ('a, 'b) t -> 'a -> 'b option` renvoie `Some` de la valeur associée à la clé dans la table ou `None` si la clé n'est pas dans la table.
- `Hashtbl.iter : ('a -> 'b -> unit) -> ('a, 'b) t -> unit` prend en entrée une fonction et une table de hachage et applique la fonction à toutes les valeurs stockées dans la table.

- **Q12.** Un groupe d'amis vote pour ce qu'ils vont manger à midi. En prenant en entrée la liste des votes de chaque personne, donner tous les plats proposés, avec le pourcentage de personnes qui ont voté pour eux. Par exemple, pour `l=["pizza"; "sushi"; "pizza"; "pates"; "pates"; "pizza"; "steak"; "sushi"; "pizza"; "pizza"]`, on obtiendra un dictionnaire contenant les couples (clé, valeur) suivants : ("pizza",50%), ("pates",20%), ("sushi",20%), ("steak",10 %).
- **Q13.** Écrire une fonction qui prend en entrée un dictionnaire d et renvoie un dictionnaire dans lequel les clés sont les valeurs du dictionnaire d et la valeur associées à la clé i est la liste des clés du dictionnaire d qui avaient pour valeur i.

  Par exemple pour un dictionnaire contenant les couples ("a",3), ("b",3), ("c",1), ("d",6), on renvoie un dictionnaire contenant les couples (3,["a";"b"]), (1,["c"]), (6,["d"]).