

Corrigé du TD sur les tables de hachage

Question 1.

Ici il n'y a pas besoin d'écrire un programme auxiliaire calculant les puissances : on utilise bien entendu le principe de l'algorithme de Horner !

```
let conv s = let n = string_length s - 1 in
  let res = ref (int_of_char s.[n])
  in for i = (n-1) downto 0 do
    res := 256*(!res)+(int_of_char s.[i])
  done; !res;;
```

Question 2.

On peut simplement utiliser le programme suivant :

```
let hash m s = (conv s) mod m;;
```

Cependant, pour une meilleure efficacité, il est préférable de reprendre le programme `conv` et d'effectuer le calcul modulo M à chaque étape :

```
let hash m s = let n = string_length s - 1 in
  let res = ref (int_of_char s.[n])
  in for i = (n-1) downto 0 do
    res := (256*(!res)+(int_of_char s.[i])) mod m
  done; !res;;
```

Question 3.

On crée une table de hachage dont la fonction de hachage sera la fonction `h` passée en argument, et la table de données un vecteur de longueur `m` initialisé sur la liste vide.

```
let creer h m = let vecteur = make_vect m [] in
  {hash=h; donnees=vecteur };;
```

Question 4.

Le programme `insere` est un programme auxiliaire qui insère l'élément `a` dans la liste `l` s'il ne s'y trouve pas, et ne fait rien s'il y est déjà.

```
let rec insere l a = match l with
[]->[a];
|x::reste when x=a->x::reste;
|x::reste->x::(insere reste a);;

let ajoute m k e = let i = (m.hash k) in let l=m.donnees.(i) in
m.donnees.(i)<- (insere l (k,e));;
```

Question 5.

```
let rec trouve_liste k l = match l with
[]->failwith "non trouve";
|(x1,x2)::reste when x1=k->x2
|x::reste->trouve_liste k reste;;

let trouve m k = let i = (m.hash k) in trouve_liste k
(m.donnees.(i));;
```

Question 6.

Il ne faut pas confondre la taille avec la longueur M du vecteur de données. La taille est le nombre total d'éléments dans le tableau, c'est-à-dire la somme des longueurs des listes qui y sont rangées. Une nouvelle table a donc une taille 0.

```
let dyn_cree h m = let vecteur = make_vect m [] in {hash=h; taille=0; donnees=vecteur };;
```

Question 7.

Cette question était la plus difficile. Comme on veut une fonction de type `unit()`, il ne faut pas créer de nouvelle table de hachage, mais seulement modifier l'ancienne. On crée ainsi un vecteur `nouv_donnees` de longueur $2w+1$ ($w=M$) dans lequel on va ranger tous les éléments de la table de hachage. Pour cela, on parcourt l'ancien vecteur de données avec une boucle `for`, et on insère la liste présente dans chaque case du vecteur en utilisant la fonction auxiliaire `ajoute_liste`, qui prend une liste `l` en argument et ajoute ses éléments dans la nouvelle table de hachage un à un, en calculant leur image par la nouvelle fonction de hachage.

Il ne reste plus qu'à remplacer l'ancien vecteur `m.donnees` par le nouveau.

```
let dyn_rearrange table = let old_donnees=table.donnees in
let w=vect_length old_donnees in
let nouv_donnees = make_vect (2*w+1) [] in
table.donnees<-nouv_donnees;
  let rec ajoute_liste l = match l with
    |[]->()
    |(k,e)::reste->ajoute_liste reste; let n=table.hash (2*w+1) k in
      table.donnees.(n)<-insere table.donnees.(n) (k,e)
  in for i=0 to (w-1) do
ajoute_liste old_donnees.(i)
done;;
```

Question 8.

```
let dyn_ajoute m k e = if m.taille >(2*vect_length m.donnees)
  then dyn_rearrange m;
let i=(m.hash (vect_length m.donnees) k) in
m.donnees.(i)<-insere m.donnees.(i) (k,e);
m.taille <-m.taille +1;;

let dyn_trouve m k = let i = (m.hash (vect_length m.donnees) k) in trouve_liste k
(m.donnees.(i));;
```

Question 9.

On se place d'abord dans le pire des cas : c'est le cas où la fonction de hachage est mal adaptée au problème, et tous les éléments sont dans la même case du vecteur. Dans ce cas, l'insertion et la recherche sont les mêmes que dans une liste non triée, et se font en $O(\text{taille})$

Le meilleur des cas est le cas où les éléments sont bien répartis. Si l'on adopte le principe de redimensionnement décrit, cela veut dire qu'il y a deux éléments par liste, et donc la recherche se fait en $O(1)$. L'insertion se fera en $O(1)$ s'il n'y a pas besoin de redimensionner le tableau, mais en $O(\text{taille})$ s'il le faut, car il faut recalculer la valeur de la fonction de hachage sur chacun des éléments.

Il s'agit donc de trouver un bon compromis entre fréquence des redimensionnements et efficacité de la recherche, en fonction de si en pratique on pense avoir plus souvent à insérer des éléments, ou à faire des recherches.