Tables de hachage
(*hash tables*)

Principe

Les *tables de hachage* (*hash tables* en anglais) constituent une implantation souvent efficace des opérations de stockage et de recherche dans une table. L'idée est la suivante : pour chaque enregistrement à stocker/rechercher dans la table, on calcule une *clé entière*, entre 0 et $M-1$, et on utilise M listes pour stocker les enregistrements de même clé.

Le plus difficile est donc de définir une bonne *fonction de hachage* sur les clés, c'est-à-dire une fonction qui se comporte comme une fonction aléatoire (toutes les valeurs de 0 à $M-1$ doivent être equiprobables).

Réalisation

On va supposer ici que les enregistrements sont des chaînes de caractères, mais le principe se généralise quel que soit le type des clés. Une fonction de hachage possible est la suivante : soit une chaîne de caractères

$$s = a_0 a_1 \ldots a_n$$

et soit c_i le code ASCII du caractère a_i (il est donné par la fonction `int_of_char`). On choisit alors comme clé :

$$k = \sum_{i=0}^{i=n} 128^i . c_i$$

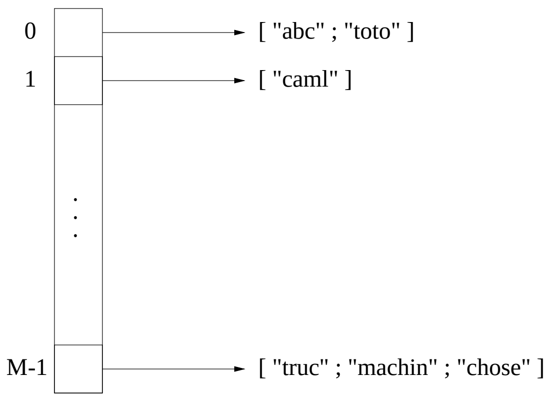
puis comme fonction de hachage :

$$h(k) = k \bmod M$$

Pourquoi est-il souhaitable de prendre M premier ?

□ Ecrire une fonction `hash` qui implante cette fonction de hachage sur les chaînes de caractères (pour une valeur de M passée en argument).

Comme nous l'avons expliqué, une table de hachage de taille M est un tableau de listes de taille M. A chaque indice i est stockée la liste des enregistrements pour lesquels la fonction de hachage vaut i.



On définit le type des tables de hachage de la façon suivante, le champ `M` contenant la taille du tableau, et le champ `donnees` contenant le tableau lui-même :

```
type 'a tableh = { M       : int ;
                   donnees : 'a list vect } ;;
```

- ☐ Ecrire une fonction `nouvelle : int -> 'a tableh` donnant une nouvelle table de hachage pour une taille donnée en argument.
- ☐ Ecrire une fonction `ajoute : string tableh -> string -> unit` ajoutant une chaîne dans une table de hachage.
- ☐ Ecrire une fonction `trouve : string tableh -> string -> bool` déterminant si une chaîne est présente ou non dans une table de hachage.

Amélioration : tables de hachage de taille dynamique

On se rend compte que la partie coûteuse de la recherche est le parcours de la liste des enregistrements correspondant à une valeur de la fonction de hachage pour déterminer si un élément s'y trouve ou non.

Une amélioration possible est d'utiliser des listes triées : l'ajout est alors un peu plus coûteux mais la recherche plus rapide.

On propose ici d'implanter une autre amélioration : les tables de hachage de taille dynamique. L'idée est d'augmenter la taille du tableau lorsque les listes deviennent trop longues : on garde ainsi des listes courtes dans lesquelles la recherche est rapide.

Pour cela, on redéfinit le type des tables de hachage de la manière suivante :

```
type 'a tableh = { mutable max     : int ;
                   mutable M       : int ;
                   mutable donnees : 'a list vect } ;;
```

où `max` est la longueur maximale des listes, `M` la taille du tableau et `donnees` le tableau lui-même. Ces trois champs sont désormais modifiables (`mutable`) car leurs valeurs doivent maintenant pouvoir être modifiées (y compris le tableau lui-même, qui doit pouvoir être substitué par un tableau plus grand).

- ☐ Ecrire une fonction `trop_long : int -> 'a list -> bool` déterminant si une liste a une longeur strictement supérieure a un entier donné.
- ☐ Ré-écrire les fonctions `nouvelle`, `ajoute` et `trouve` pour les tables de hachage de taille dynamique. On fera les choix suivants : initialement `max` vaut 3 puis, lorsque la taille d'une liste dépasse la valeur de `max`, la table de hachage est re-dimensionnée. Pour cela la valeur de M est remplacée par $2M+1$ et la valeur de `max` par $2 \times$ `max`. On prendra soin de conserver l'ordre des éléments dans les listes au moment du re-dimensionnement.