❑ **Exercice 1** : *un exemple de table de hachage*
On considère une table de hachage de 13 alvéoles contenant des entiers, la fonction de hachage est : $h : k \mapsto k \mod 13$.

1. Donner le schéma de la table après insertion des entiers 17, 120, 39, 144, 100, 5, 110.
2. Quel est la taux de charge de la table ?
3. Donner deux valeurs non encore insérées et qui provoquent une collision.

❑ **Exercice 2** : *fonction de hachage*

1. Une fonction de hachage peut-elle être aléatoire ? Justifier
2. Que dire du nombre de collisions si la fonction de hachage est injective ?
3. Sur l'ensemble des chaines de caractères on choisit la fonction renvoyant la longueur de la chaine comme fonction de hachage. Schématiser le contenu de la table de hachage après insertion des mots de la phrase :« *La cigale, ayant chanté tout l'été, se trouva fort dépourvue quand la bise fut venue* »
4. On suppose qu'on construit une table de hachage de taille N avec une fonction de hachage constante égale à un entier $k \in [\![0; N-1]\!]$. Quel sera le contenu de cette table après l'insertion de n éléments ?

❑ **Exercice 3** : *Adressage ouvert par sondage linéaire*
Une autre méthode de gestion des collisions consiste lorsque deux valeurs produisent le même *hash* à rechercher le premier emplacement vide, plus loin dans la table de hachage afin d'y ranger la nouvelle valeur. Par exemple, en notant a, b, c, d les couples de clés valeurs déjà insérés, alors si le contenu de la table est :

a				b	c			d	
0	1	2	3	4	5	6	7	8	9

Alors si l'insertion de e doit se faire à l'indice `4`, il sera placé en `6` (premier emplacement vide suivant)

1. Donner le schéma de cette table après insertion de f (de *hash* 9) et de g (de *hash* 8).
2. A quelle condition portant sur le taux de charge de la table cette stratégie peut-elle fonctionner ?
3. Proposer une implémentation en C de cette nouvelle stratégie d'insertion.
4. Que dire du problème de la suppression d'une entrée dans la table ?

❑ **Exercice 4** : *Suppression des doublons*
On considère un tableau t de n valeurs, on souhaite récupérer *sans doublons* la liste des valeurs présentes dans ce tableau. Le langage d'application dans cet exercice sera OCaml.

1. Proposer un algorithme naïf n'utilisant pas de table de hachage pour répondre au problème posé.
2. Donner la complexité en fonction de n de cet algorithme.
3. En donner une implémentation en OCaml sous la forme d'une fonction `unique : 'a array -> 'a list`.
4. En utilisant les tables de hachage, donner un algorithme ayant une meilleure complexité.
5. En donner une implémentation en OCaml sous la forme d'une fonction `unique_hash : 'a array -> 'a list`. On pourra utiliser le module `Hashtbl`.

❑ **Exercice 5** : *Hachage d'une chaine de caractères*

1. Une fonction de hachage élémentaire sur les chaines consiste simplement à faire la somme des codes ASCII de ses caractères. On suppose qu'on hache des chaines de longueur maximale 26.
   a) Quel est alors la valeur maximale atteinte par la fonction de hachage ?
   b) Rappeler le nombre d'entiers représentable en OCaml par le type `int`
   c) Même question pour le type `uint64_t` en C.
   d) Que dire de ces résultats ?
2. On propose une nouvelle fonction de hachage pour une chaine s de longueur n, donnant des valeurs « assez grandes » et sensible à l'ordre des caractères :

$$h(s) = \sum_{i=0}^{n-1} 31^i \times c_i$$

où c_i est le code du caractère situé à l'indice i dans la chaine s et la constante 31 choisie arbitrairement.

a) Proposez une implémentation *efficace* en C pour cette fonction.
b) Même question en OCaml
c) Si la longueur de la table de hachage est N peut-on directement utiliser $h(s) \mod N$ comme indice du seau de la chaine s ? Expliquer

❑ **Exercice 6** : *Recherche de collisions*
On reprend la fonction de hachage de l'exercice précédent : pour une chaine s de longueur n,

$$h(s) = \sum_{i=0}^{n-1} 31^i \times c_i$$

où c_i est le code du caractère situé à l'indice i dans la chaine s.

1. Montrer qu'il existe deux chaines de caractères de longueur 2, formées de lettres minuscules (code 97 à 122) ou majuscules(code 65 à 90) et produisant la même valeur pour h.
2. En déduire une façon de construire un nombre arbitraire de chaînes de caractères ayant la même valeur pour la fonction h.