Sujet D

❑ **Exercice** : *type A*

1. Rappeler le principe de l'implémentation d'un tableau associatif (ou dictionnaire) à l'aide d'une table de hachage.

   On utilise un tableau de taille M et une fonction de hachage à valeur dans $[\![0; M-1]\!]$ (par exemple en prenant le modulo M), un couple clé, valeur (c, v) est alors stocké dans la case d'indice $h(c)$ du tableau.

2. Donner la définition des *collisions* et donner brièvement (au moins) un moyen de résoudre le problème qu'elles posent.

   Une collision est l'obtention de la même valeur de *hash* pour deux clés différentes c_1 et c_2. On peut les résoudre par chainage c'est à dire qu'une case du tableau contient une liste chainée de couples clé valeur ou encore par sondage linéaire

Pour une chaine de caractères $s = c_0 \dots c_{n-1}$, on considère la fonction de hachage :

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

3. Calculer le hash de la chaine `"AB"`.

   $31 \times 65 + 66 = 2081$

4. 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.

   En notant $E = [\![65; 90]\!] \cup [\![97; 122]\!]$, on cherche $(a, b) \in E^2$ et $(a', b') \in E^2$ tels que : $(a, b) \neq (a', b')$ et $31a + b = 31a' + b'$. Cette équation se ramène à $b - b' = 31(a' - a)$. Donc $b - b'$ est un multiple non nul de 31. l'écart entre b' et b étant au maximum en valeur absolue de $122 - 65 = 57$ les seules possiblités sont :

   - $b - b' = 31$ et donc $a' - a = 1$. Ce qui donne $a' = a + 1$ (donc a' et a sont dans le même intervalle) et $b' = b - 31$ donc ils ne sont pas dans le même intervalle. On peut prendre par exemple la solution $a = 65$ donc $a' = 66$ et $b = 97$ donc $b' = 66$ c'est-à-dire les chaines `"Aa"` et `"BB"`.
   - $b - b' = -31$ et donc $a' - a = -1$. Ce qui donne $a' = a - 1$ et $b' = b + 31$, par exemple la solution $a = 66$, $a' = 65$, $b = 67$ et $b' = 98$ convient (c'est-à-dire les chaines `"BC"` et `"Ab"`)

5. En déduire une façon de construire un nombre arbitraire de chaînes de caractères de longueurs quelconques ayant la même valeur pour la fonction h.

   En concaténant plusieurs chaines de caractères de longueur deux ayant le même hash on obtient des chaine de caractère de longueur paire arbitrairement grande et ayant le même hash. Par exemple (en utilisant les exemples de longueur deux donnés à la question précédente), `"AaBC"` et `"BBAb"`. Pour la longueur impaire on rajoute le même caractère aux deux chaines.

6. Proposer un prototype pour une fonction en C qui calcule cette fonction de hachage (type de la valeur renvoyée, argument(s) et leur(s) type(s)).

   Les chaines de caractères "connaissent" leur longueur en C grâce au caractère sentinelle, on a donc pas besoin de passer la longueur de la chaine de caractère en paramètre. Par contre, cette fonction risque fort de provoquer un dépassement de capacité sur le type `int` ce qui est un comportement indéfini en C. On devrait plutôt utiliser la librairie `stdint.h` et renvoyer un `uint`. On peut donc proposer `uint hash(char* s)`

❑ **Exercice** : *type B*

Pour un entier n quelconque, le but de l'exercice est de programmer en C une recherche en *backtracking* d'une solution au problème du placement des entiers $1, \ldots, n^2$ entiers dans un carré de côté n afin de former un carré magique (c'est-à-dire un carré dont la somme des lignes, colonnes ou diagonales est la même)

1. Vérifier que dans le cas $n = 3$ le carré suivant est une solution :

2	7	6
9	5	1
4	3	8

   On trouve bien 15 en additionnant chaque ligne, colonne ou diagonale.

2. Quelle serait la somme de chacune des lignes, colonnes ou diagonales dans le cas $n = 4$?

   Les entiers de 1 à 16 de somme $8 \times 17 = 136$ sont placés dans le carré et la somme des 4 lignes est identique, donc chacune de ces sommes vaut $136/4 = 34$.

3. Donner l'expression de la somme de chacune des lignes, colonnes, ou diagonales pour un entier n quelconque.

   On reprendre le raisonnement précédent, on en déduit rapidement que la somme commune est $\dfrac{n \times (n^2+1)}{2}$.

Afin d'implémenter la résolution en C, on propose de linéariser le carré en le représentant par un tableau à une seule dimension. Le carré ci-dessus est par exemple représenté par :
`int carre[9] = {2, 7, 6, 9, 5, 1, 4, 3, 8}`
Un fichier contenant le code compagnon de cet exercice est à télécharger à l'adresse :
`https://fabricenativel.github.io/cpge-info/oraux/`.
Il contient notamment une fonction `void affiche_carre(int carre[], int n)` permettant de d'afficher les valeurs contenues dans un carré.

4. Donner l'expression de l'indice d'un élément situé en ligne i, colonne j dans le tableau linéarisé (on rappelle qu'on a noté n le côté du carré).

   L'indice de l'élément situé ligne i colonne j est $i \times n + j$

5. Inversement, donner la ligne et la colonne dans le carré initial d'un élément situé à l'indice k dans le tableau linéarisé.

   L'élément d'indice k dans le tableau linéarisé est situé à la ligne $\left\lfloor \dfrac{k}{n} \right\rfloor$ et à la colonne $k \mod n$.

On représente par l'entier 0 une case non encore rempli du tableau, par exemple :
`int carre[9] = {1, 8, 4, 0, 0, 0, 0, 0, 0}` représente un carré ayant simplement la première ligne complète

6. Ecrire une fonction `bool valide_ligne(int carre[], int i, int n, int somme)` qui vérifie que la ligne `i` d'un carré en cours de construction est encore valide, c'est le cas si cette ligne contient un zéro (car elle est alors incomplète) ou si elle ne contient aucun zéro et que sa somme est égale à la variable `somme` fournie en argument.

```
bool valide_ligne(int carre[], int i, int n, int somme)
{
    int s = 0;
    for (int j = 0; j < n; j++)
    {
        if (carre[i * n + j] == 0)
        {
            return true;
        }
        s += carre[i * n + j];
    }
    return s == somme;
}
```

7. Ecrire de même la fonction `bool valide_colonne(int carre[], int j, int n, int somme)` permettant de vérifier que la colonne `j` d'un carré en cours de construction est encore valide.

```
bool valide_colonne(int carre[], int j, int n, int somme)
{
    int s = 0;
    for (int i = 0; i < n; i++)
    {
        if (carre[i * n + j] == 0)
        {
            return true;
        }
        s += carre[i * n + j];
    }
    return s == somme;
}
```

Les fonctions permettant de vérifier que les deux diagonales sont valides ainsi que la fonction de prototype `bool valide_carre(int carre[], int size, int somme)` vérifiant globalement que le carré en cours de construction est encore valide sont déjà écrites dans le fichier compagnon de cet exercice.

8. En vous aidant des commentaires, compléter le code de la fonction permettant de rechercher par backtracking un carré magique solution du problème posé et disponible dans le fichier compagnon. Cette fonction a pour signature :
   `bool resolution(int carre[], int n, int somme, int k, bool utilise[])`
   en effet, elle prend aussi en argument le tableau de booléens `utilise` de taille n^2 dont l'élément d'indice i indique si l'entier $i+1$ a déjà été placé ou non dans le carré.

```cpp
bool resolution(int carre[], int n, int somme, int k, bool utilise[])
{
    // Teste toutes les valeurs possibles pour l'élément d'indice k du carre
    // Renvoie false si aucune valeur ne convient, sinon poursuit la résolution à l'indice k+1
    if (k == n * n)
    {
        return true;
    }
    for (int v = 1; v <= n * n; v++)
    {
        if (!utilise[v - 1])
        {   // Placer v à l'indice k dans le tableau et rendre cette valeur indisponible
            utilise[v - 1] = true;
            carre[k] = v;
            // Si le carre est valide et que la résolution aboutie on renvoie vraie
            if (valide_carre(carre, n, somme) && resolution(carre, n, somme, k + 1, utilise))
            {
                return true;
            }
            // Sinon il faut faire un retour sur trace et tester les valeurs suivantes
            else
            {
                utilise[v - 1] = false;
                carre[(k / n) * n + k % n] = 0;
            }
        }
    }
    // On a testé sans succès toutes les valeurs
    return false;
}
```