Hachage
TP05

a) Nous allons nous servir de la table ASCII, qui est la table utilisée par défaut dans le langage C. Nous aurons la fonction suivante, pour forcer un int pour pouvoir implémenter notre fonction de hachage,

```
int code(char c) { //C traite les chars comme des ints
     return (int) c; //et se sert de la table en imprimant
} //quand le format char est spécifié.
```

Comme précisé dans l'énoncé, nous commenceront avec une fonction de hachage naive,

H : char[10] → int
H(mot) = code(mot) mod M

Je ne vais pas instaurer un système de gestion de collisions pour le premier cas, en cas de collisions je ne fait qu'afficher « collision » et je compte le nombre.

b) Je me sert des données fournies en pièce jointe dans le ficher .txt, j'ai effectué le calcul sur 100'000 passagers, il a fallu rencoder le texte en UTF-8 pour qu'il soit lisible par la fonction de <string.h> fscanf. En implémentant l'algorithme en C dans code blocks utilisant le compilateur GCC, j'ai le résultat suivant,

```
Il y a eu 63120, insertions et 36880 collisions pour la methode simple.

Process returned 0 (0x0)   execution time : 0.097 s
```

On a donc réussi à insérer 63.12 % des passagers dans la table via le hachage simple, ce qui n'est pas mal, voyons ce qu'on peut faire avec une gestion de collision.

c) Le hachage linéaire a généré des résultats plus intéressants, commençons par définir ce à quoi la fonction ressemble, formellement :

HL: char[10] x int → int
HL(mot, i) = (H(mot) + i) mod M

Par propriété des mods, on a que a*b mod n = ((a mod n)*(b mod n)) mod n, donc ça joue à ce niveau, on peux mettre des modulos sans trop s'inquiéter. Les résultats utilisant cette fonction furent,

```
Il y a eu 100000, insertions et 22440072 collisions pour la methode lineaire.

Process returned 0 (0x0)   execution time : 0.302 s
```

Oui vous lisez cela bien, 22'440'072, on a bien eu 22 millions de collisions, ce qui n'est pas si étonnant que cela, sur la fin la table est quasi remplie et il faut reparcourir O(100'000) valeurs à chaque fois avant de trouver une place ou insérer. Et on arrive bien à faire 100'000 insertions, car le hachage linéaire garanti de pouvoir accéder à toutes les cases, étant donné que quand on dépasse le tableau, on retourne à 0 et on se décale de une case à chaque fois. La répartition pour 100'000 personnes s'est faite en un gros bloc de 100'000 personnes, ce qui n'est pas très intéressant, j'ai essayé pour une table de 100'000 entrées mais avec 40'000 personnes voir ce que cela donne, et j'ai obtenu une répartition ressemblant,

```
 2, 2, 1, 1, 2, 2, 1, 6, 3, 1, 1, 5, 1, 1, 2, 1, 1, 2, 1, 2, 3, 3, 1, 1, 3, 2, 1, 3, 1, 1, 1, 1, 1,
1, 2, 3, 1, 1, 1, 2, 4, 1, 1, 2, 1, 3, 1, 1, 1, 3, 1, 1, 1, 3, 3, 1, 2, 3, 1, 1, 1, 2, 2, 1, 5, 4, 2
, 3, 1, 1, 3, 2, 2, 3, 4, 1, 3, 1, 2, 2, 1, 1, 2, 3, 1, 2, 2, 1, 1, 1, 1, 1, 1, 8, 1, 2, 1, 2, 1, 1,
 7, 3, 1, 1, 2, 2, 4, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 10, 1, 4, 3, 2, 1, 1, 1, 4, 1, 5, 1, 3,
 1, 1, 8, 5, 1, 3, 2, 1, 2, 1, 4, 8, 2, 1, 1, 1, 2, 4, 1, 5, 1, 1, 4, 1, 1, 3, 1, 2, 1, 2, 1, 2, 2,
1, 1, 1, 2, 1, 2, 2, 2, 1, 1, 3, 2, 1, 1, 1, 3, 2, 1, 1, 2, 1, 5, 1, 3, 1, 1, 3, 2, 1, 1, 1, 3, 7, 1, 1
, 3, 1, 1, 1, 1, 1, 24, 2, 2, 1, 4, 1, 1, 2, 3, 4, 4, 1, 2, 1, 1, 2, 9, 2, 1, 1, 1, 1, 3, 1, 2, 1, 1
, 2, 1, 1, 2, 2, 2, 1, 1, 3, 1, 3, 4, 1, 1, 3, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2, 5, 3, 2, 1, 3, 1, 2, 1,
 1, 1, 2, 3, 2, 3, 2, 1, 1, 4, 1, 4, 5, 5, 1, 1, 1, 7, 1, 2, 1, 1, 1, 2, 1, 1, 3, 1, 1, 1, 1, 2, 3,
1, 2, 1, 1, 3, 1, 3, 1, 3, 1, 1, 5, 2, 1, 3, 1, 2, 2, 1, 3, 1, 1, 1, 1, 3, 5, 2, 1, 1, 1, 2, 2, 8, 1
, 2, 1, 2, 1, 1, 7, 1, 1, 3, 3, 2, 3, 3, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 9, 2, 2, 1, 3, 1, 6, 1, 2,
 4, 1, 2, 2, 2, 2, 1, 5, 5, 1, 1, 2, 1, 4, 1, 1, 1, 2, 1, 1, 1, 1, 2, 3, 1, 5, 1, 5, 1, 2, 3, 1, 1,
1, 2, 3, 1, 2, 2, 1, 1, 1, 4, 2, 1, 1, 1, 4, 1, 2, 2, 7, 2, 1, 1, 1, 5, 2, 4, 1, 2, 2, 1, 1, 4, 1, 1
, 1, 2, 3, 3, 8, 1, 1, 1, 1, 1, 4, 3, 1, 1, 1, 4, 1, 3, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1,
 3, 3, 1, 1, 2, 3, 3, 4, 1, 1, 1, 1, 3, 1, 8, 13, 3, 3, 2, 3, 2, 6, 1, 2, 2, 1, 1, 10, 1, 2, 2, 1, 1
```

Comme on le voit des clusters de 24 à 30 passagers apparaissent, les autres étant généralement isolés. L'implémentation étant en pièce jointe, je vous laisse étudier ce phénomène en changeant le fichier ouvert de « passagers.txt » à « passagers40000.txt ». Le nombre de collisions étant toutefois monstrueux, voyons ce que le hachage quadratique donne.

d) Le hachage quadratique permet un espacement des valeurs lors de la détection d'une collision et du décalage qui ensuit. Une telle fonction est décrite par,

HQ: char[10] x int → int
HQ(mot, i) = (H(mot) + a1*i + a2*i²) mod M

ou a2 et a1 sont des entiers naturels, dans le cadre de ce TP, j'ai arbitrairement choisi a1 = 1 et a2 = 3. Les résultats de cette implémentation furent,

```
Il y a eu 99793, insertions et 630253 collisions pour la methode quadratique.
```

On a donc pû insérer 99.793 % des passagers, donc 207 cases sont restées vides hélas. Mais ce n'est pas mal, et le nombre de collisions est de 630'253, soit environ 36 fois moins que le hachage linéaire, ce qui est clairement mieux. On a sacrifié des cases au coût de moins de collisions. Cela à donné la répartition suivantes, ou les nombres signifient le nombre de passagers à la suite avant un trou,

```
10, 305, 287, 93, 5, 479, 200, 280, 213, 149, 97, 2051, 51, 3213, 129, 243, 1145, 47, 243, 69, 212,
94, 25, 205, 47, 1193, 81, 943, 405, 131, 543, 469, 273, 38, 192, 25, 67, 533, 657, 87, 1311, 207, 3
87, 873, 1381, 233, 207, 259, 83, 1515, 629, 845, 153, 489, 192, 176, 359, 131, 489, 185, 433, 289,
1395, 814, 908, 3299, 2421, 1687, 17, 19, 217, 839, 143, 1, 587, 435, 687, 117, 963, 13, 7, 823, 455
, 657, 55, 71, 567, 449, 403, 345, 21, 137, 403, 63, 924, 452, 437, 20, 250, 91, 809, 35, 103, 125,
25, 5, 777, 93, 1006, 632, 95, 31, 591, 257, 25, 399, 107, 375, 342, 346, 545, 677, 261, 39, 341, 10
49, 193, 599, 691, 497, 293, 269, 57, 1183, 33, 65, 625, 233, 2467, 2439, 71, 185, 489, 552, 394, 10
97, 97, 181, 97, 827, 87, 165, 49, 79, 106, 1766, 479, 627, 715, 99, 117, 961, 33, 1525, 1014, 2210,
 472, 1060, 775, 1243, 587, 1059, 409, 2017, 162, 92, 117, 1537, 84, 36, 365, 451, 1364, 34, 6, 2143
, 90, 33, 163, 1019, 79, 43, 15, 73, 67, 29, 171, 189, 201, 43, 111, 400, 774, 143, 133, 181, 579,
```

on voit bien des agglomérations apparaître, certaines allant jusqu'à 2500 en taille, et les plus petite de l'ordre de 3 passagers.

e) Pour le double hachage, je me suis servi de deux fonctions simples suivant les consignes de l'énoncé, ma fonction à la tête suivante,

HD : char[10] x int → int
HD(nom, i) = (H(nom, i) + 3*i*H(nom, i)) mod M

Ainsi, H1 = H, et H2 = 3*i*H, on a bien que H1 doit est premier par rapport à H2, et cela évite les chevauchements. Les résultats, sur 100'000 passagers furent,

```
Il y a eu 63864, insertions et 119852 collisions pour le double hachage.
```

C'est nettement moins bon que le hachage quadratique, on n'a que réussi à insérer 63.864 % des passagers pour 120'000 collisions. On a certes divisé le nombre de collisions presque par un facteur de 6, mais on a réduit le nombre d'insertions de 35 %, est ce que cela en valait le coup ? Je pense que cela dépends de la situation. Chaque technique à ses avantages et désavantages, mais le hachage quadratique semble triompher d'un point de vue rapport insertions/collisions.

```
x...x...x.x.xx.x..xxxxx.xx.x....x.x.xxx.x.x.x..xx.xxx.xxxxxxxx..x...xxx.xx..x.x.x.x.xxx.x..xxxx.xx.x
xx.xxx.xxx..x..xx..xxx.xx....xx.x.xxxx.xxxx.xx.xxxx.x.x..x..xxx..x..x.xxxxx..xxx.x.xxxxxxxxxxx.x.xx
xxxxxxxxx.xxxxxxxxxx.x.x..x.xx.x.x.xxxxx...x.xxxx..xx.xx..xx..xx.xxxxxxxx..x.x.xx..xx.xxx.xx.xxxxx.
xx..xxx.x.xxxx.xxx.xxxxxxxxx.xx...xxxx.xx.xxx.x..xxx.x.x.xxx.x.xx..xxxxx..xx...x.xxx.xxxxxxxx.x..x
x.x.xx.xxxxxx..xx...xx.xx...xxxxxxxxxxxxxx.x..xxx..x......xxxx.x.x.xxx.xx.xxx..x.x.xxxxxxx.xx..xxxx
xx.xx.x.x.xxxx..x.xxxx..xxxxxxx.xxxxxxx.x...x.xxxxx.x.xxxx..xx..xx.xx.x.x..xxxx.x..xxx.x..xxx...x..x
xxx.x..xx.x...x.xxx.xxxx.xxxx.x.x.x.x...xxx.xx.xx.xxxx..x...x.xxx...xxxxx..xx..xxx.xxxxxx.xxx..xxxxx
x...x.x..xxxxx..x.x.xx..xxxxx.....x.x..xx.x.xxxxxxxx..x.xxxxxx.xxxx.x.xx...xx.xxx...x.xxxxxxx..xxxx.
x.xxxx.xx..xx.xx..xxx.x.xxxxx.xxx..xx...x.xxxxx.xxxxx..xxx.xxxxxxx..xx.xxx.xxxx.x.xxx..xx..xxx.xx.xx
```

On constate bien un clustering apparaître, avec une répartition ayant la tête suivante,

```
 5, 6, 3, 1, 7, 1, 1, 2, 6, 1, 1, 8, 4, 1, 10, 2, 1, 3, 2, 4, 1, 1, 1, 2, 4, 1, 1, 5, 1, 1, 2, 3, 3,
 3, 1, 2, 4, 2, 1, 2, 7, 1, 3, 1, 2, 3, 2, 4, 1, 3, 2, 2, 4, 2, 2, 6, 5, 1, 3, 2, 1, 2, 1, 3, 3, 3,
3, 2, 1, 4, 2, 3, 1, 4, 3, 1, 5, 1, 4, 2, 3, 2, 6, 1, 4, 1, 2, 3, 2, 1, 4, 1, 1, 7, 1, 2, 3, 1, 1, 3
, 3, 1, 1, 2, 1, 1, 4, 3, 7, 2, 6, 1, 1, 1, 2, 2, 1, 2, 1, 1, 2, 1, 2, 2, 2, 3, 1, 5, 1, 1, 1, 3, 3,
 5, 2, 7, 2, 9, 2, 1, 1, 1, 3, 1, 1, 1, 5, 3, 1, 1, 1, 5, 1, 1, 1, 5, 3, 1, 4, 3, 2, 2, 2, 8, 1, 1,
1, 1, 2, 1, 2, 3, 3, 1, 1, 1, 1, 6, 1, 1, 1, 4, 2, 4, 5, 2, 2, 3, 1, 1, 9, 1, 6, 2, 1, 1, 3, 2, 3, 2
, 2, 4, 3, 3, 3, 4, 3, 2, 2, 3, 1, 3, 2, 3, 1, 7, 2, 3, 5, 1, 1, 4, 1, 5, 6, 1, 1, 3, 3, 6, 2, 3, 2,
 9, 4, 5, 2, 3, 7, 1, 1, 3, 4, 3, 1, 5, 2, 2, 1, 3, 1, 1, 1, 1, 4, 2, 2, 2, 1, 2, 1, 2, 1, 2, 1, 1,
1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 8, 1, 3, 5, 2, 6, 1, 2, 1, 3, 4, 9, 1, 1, 1, 1, 1, 1, 1, 2, 8, 3, 5, 2
, 1, 1, 2, 7, 1, 4, 3, 1, 5, 4, 4, 1, 5, 1, 2, 2, 3, 1, 8, 1, 3, 1, 3, 1, 2, 1, 1, 3, 1, 1, 1, 1, 1,
 2, 1, 4, 2, 2, 3, 6, 7, 2, 3, 3, 7, 1, 1, 1, 6, 4, 1, 2, 2, 2, 2, 9, 2, 3, 2, 1, 2, 1, 6, 2, 1, 1, 5, 3,
1, 5, 6, 8, 5, 2, 1, 3, 1, 2, 1, 1, 5, 1, 3, 2, 6, 7, 1, 3, 1, 1, 1, 4, 1, 1, 3, 3, 2, 7, 2, 2, 1, 2
```

Ou les clusters les plus grand font une douzaine de passagers.

Mais de nouveau, ces résultats sont pour « seulement » 100'000 valeurs, les méthodes de double hachage sont très importantes pour des tables ayant des millions de valeurs, ou le hachage linéaire et quadratique génèrent tellement de collisions qu'il est préférable de ne pas remplir totalement la table plutôt que d'attendre que l'algorithme linéaire ou quadratique finisse.

En pratique pour améliorer la chose, on se sert de **fonctions de hachage universelles** qui réduisent les collisions encore plus, mais cela est en dehors du cadre de ce cours et abordé plus en détail en cryptographie et cybersécurité avec le protocole Solana.

LES IMPLÉMENTATION EN PIÈCE JOINTE SONT ENTIÈREMENT MON OEUVRE EXCEPTÉ LA LIGNE DU FSCANF QUI PROVIENT DE TUTORIALSPOINT[1]

1 https://www.tutorialspoint.com/c_standard_library/c_function_fscanf.htm