L2 Informatique

TP Algorithmique – Chiffrement et déchiffrement de messages

Le TP consiste à implémenter un algorithme original permettant le chiffrement d'un message en clair et le déchiffrement d'un message chiffré, à partir d'une clé. Le message en clair, la clé et le message chiffré seront tous composés de nombres entiers mais représentés par des structures de données différentes.

Chaque question demande soit de définir un type, soit d'écrire un sous-programme (pour ce faire des sous-programmes annexes pourront être définis). Parallèlement, ne pas oublier d'**écrire un programme principal** qui utilisera de manière cohérente les principaux sous-programmes définis.

1 Le message en clair

Un message en clair (type `Message`) sera représenté par un enregistrement composé d'un tableau d'entiers et d'un entier représentant la longueur du message. → *Exemple figure 1.*

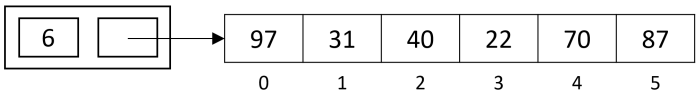

Figure 1: Variable de type `Message` représentant le message (97,31,40,22,70,87).

1. Définir le type `Message`.
2. Écrire un sous-programme qui génère aléatoirement un `Message` d'une longueur donnée en paramètre. Chaque valeur sera un nombre compris entre 0 et une borne supérieure définie comme constante.
3. Écrire un sous-programme qui affiche un `Message`.

2 La clé

Une clé sera représentée au moyen d'une liste doublement chainée circulaire. C'est-à-dire que chaque maillon de la liste doit comporter un lien vers le maillon suivant et un lien vers le maillon précédent. Le dernier et le premier maillon sont également liés. Chaque valeur de la liste représente un code permettant de chiffrer et déchiffrer un message. Une variable de type `Cle` comportera également la taille de la clé. → *Exemple figure 2.*

4. Définir le type `Cle`.
5. Écrire un sous-programme qui génère aléatoirement une `Cle` d'une longueur donnée en paramètre. Chaque valeur (code) sera un nombre entier compris entre 1 et une borne supérieure définie comme constante (une autre constante que `longueurMax` définie ci-avant).
6. Écrire un sous-programme qui affiche la liste des codes d'une `Cle`.

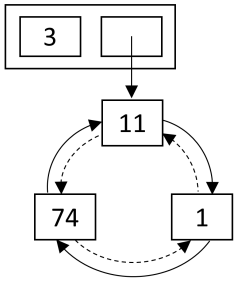

Figure 2: Représentation de la clé (11,1,74) au moyen d'une variable de type `Cle` et d'une structure de type liste doublement chainée circulaire. La taille de la clé est 3. La tête de la clé est l'adresse du maillon comportant le code 11. Les flèches en continu (resp. en pointillés) sont les liens suivant (resp. précédent).

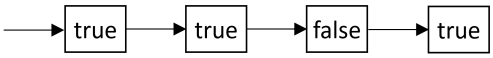

Figure 3: `Chemin` construit à partir du nombre 11 (voir question 7 pour l'explication).

3 Les chemins

Le chiffrement et le déchiffrement de messages seront effectués en utilisant des chemins. Un chemin est une liste chainée (simple) de **booléens**. → *Exemple de chemin figure 3.*

7. Définir le type `Chemin`

8. Écrire un sous-programme qui construit un `Chemin` à partir d'un nombre entré en paramètre.
   À tout nombre entier strictement positif correspond un unique chemin non vide, en reprenant le principe du système de numération binaire. Ainsi, le premier élément d'un chemin correspondant à un nombre `n` sera `false` si `n` est pair (`true` si `n` est impair), et la suite sera le chemin correspondant au nombre `n/2` (le quotient de `n` par 2). Un chemin correspond donc à la représentation binaire d'un nombre, mais dans le sens inverse.

   Par exemple, les chemins correspondant aux premiers entiers sont les suivants :

nombre `n`	chemin associé	représentation binaire de `n`[1]
1	`true`	1
2	`false` → `true`	10
3	`true` → `true`	11
4	`false` → `false` → `true`	100
5	`true` → `false` → `true`	101
6	`false` → `true` → `true`	110
7	`true` → `true` → `true`	111
8	`false` → `false` → `false` → `true`	1000
...	...	...

9. Écrire un sous-programme qui affiche un `Chemin`. Il faudra par exemple afficher `TTFT` pour le chemin de la figure 3.

10. Écrire un sous-programme qui supprime un `Chemin` (libération de l'espace mémoire).

[1] Il n'est pas demandé de convertir un nombre en binaire, mais la construction du chemin est basée sur le même principe. On peut d'ailleurs remarquer que la liste vide qui clôt chaque chemin (`nullptr`) représente les zéros non significatifs du nombre.

4 Le message chiffré

On représentera un message chiffré par un arbre binaire de nombres entiers représentant des valeurs chiffrées (c'est-à-dire des valeurs modifiées par des opérations de chiffrement). Un tel arbre sera dit de type `MessageChiff`. → *Exemple figure 4.*

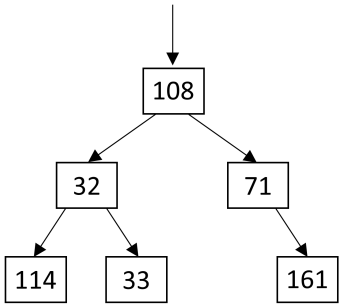

Figure 4: Exemple de message chiffré représenté au moyen d'un `MessageChiff`.

11. Définir le type `MessageChiff`.
12. Écrire un sous-programme qui ajoute un nombre (valeur chiffrée donnée en paramètre) dans un `MessageChiff` au moyen d'un chemin.
    La valeur sera nécessairement ajoutée sur une nouvelle feuille de l'arbre. Le chemin indique comment parcourir l'arbre (`false` : partir à gauche / `true` : partir à droite) jusqu'à atteindre un sous-arbre vide. Si le parcours de l'arbre s'arrête avant la fin du chemin, la valeur est ajoutée à cet emplacement libre. Si le chemin n'est pas assez long pour atteindre une feuille, alors le parcours de l'arbre se fait ensuite toujours à gauche.
    Par exemple, supposons que la valeur chiffrée 100 soit ajoutée dans le `MessageChiff` de la figure 4 au moyen du `Chemin` de la figure 3. Le parcours dans l'arbre sera le suivant : d'abord à droite (`true`), puis à droite (`true`), puis à gauche (`false`). L'emplacement étant libre dans le `MessageChiff` (le nœud dont la valeur est 161 n'a pas de fils gauche), la valeur 100 sera insérée à gauche de 161.
13. Écrire un sous-programme qui recherche et supprime une feuille dans un `MessageChiff` à partir d'un `Chemin`. Il s'agit de la feuille que l'on atteint en parcourant l'arbre selon les directions du `Chemin` (les cas particuliers évoqués dans la question précédente s'appliquent aussi ici). Le sous-programme devra retourner la valeur chiffrée de la feuille supprimée.
14. Écrire un sous-programme qui affiche les valeurs chiffrées contenues dans un `MessageChiff` selon un parcours infixe de l'arbre.
15. Écrire un sous-programme qui calcule et retourne la taille d'un `MessageChiff` (donc la taille de l'arbre).

5 Le chiffrement

Pour chiffrer un `Message`, on va chiffrer chaque valeur dans l'ordre des indices, puis ajouter la valeur chiffrée dans l'arbre. Le chiffrement d'une valeur et son insertion dans l'arbre dépend d'un code de la clé. La première valeur du message sera chiffrée au moyen du premier code de la clé, la deuxième valeur au moyen du code suivant, et ainsi de suite (sachant que lorsque tous les codes de la clé ont été utilisés, on les reconsidère en partant du début). Pour chaque valeur, le traitement est le suivant :

- On construit le `Chemin` correspondant au nombre `i+c`, où `i` est l'indice de la valeur (`v`) du message à chiffrer, et `c` est le code courant de la clé.

- Le chiffrement de `v` consiste simplement à lui ajouter le code `c`.
- On ajoute alors la valeur chiffrée (`v+c`) dans le `MessageChiff` au moyen du `Chemin`.

Chaque chemin n'étant utilisé qu'une seule fois lors du chiffrement, il faut penser à libérer la mémoire après les avoir utilisés.

Par exemple, chiffrer le message (97,31,40,22,70,87) représenté figure 1 avec la clé (11,1,74) représentée figure 2, produit le message chiffré représenté figure 3. Les valeurs chiffrées et chemins obtenus à chaque itération sont donnés ci-dessous :

indice `i`	code `c`	`i + c`	chemin associé	valeur `v`	val. chiffrée (`v+c`)
0	11	11	`true` → `true` → `false` → `true`	97	108
1	1	2	`false` → `true`	31	32
2	74	76	`false` → `false` → `true` → `true` → `false` → `false` → `true`	40	114
3	11	14	`false` → `true` → `true` → `true`	22	33
4	1	5	`true` → `false` → `true`	70	71
5	74	79	`true` → `true` → `true` → `true` → `false` → `false` → `true`	87	161

16. Écrire un sous-programme qui chiffre un `Message` à partir d'une `Cle`. Le sous-programme retournera donc un `MessageChiff`.

6 Le déchiffrement

Le déchiffrement consiste à effectuer l'opération inverse ; c'est-à-dire, retrouver le message initial à partir du message chiffré et de la clé de chiffrement. Les valeurs du message seront déchiffrées dans l'ordre inverse du chiffrement. Une fois déchiffrée, chaque valeur d'un `MessageChiff` devra être supprimée de l'arbre.
Il faut penser à bien positionner la clé au départ afin que le dernier code utilisé lors du chiffrement soit le premier utilisé lors du déchiffrement, et bien sûr de tourner la clé dans l'autre sens pour déchiffrer le message.

17. Écrire un sous-programme qui déchiffre un `MessageChiff` à partir d'une `Cle`. Le sous-programme retournera donc un `Message`.

7 Chiffrement d'une chaine de caractères

En C++, les caractères (type `char`) sont représentés comme des entiers (`int`). C'est-à-dire qu'il est possible d'affecter une valeur `char` dans une variable `int`, et réciproquement.

18. Écrire un sous-programme permettant la conversion d'un `std::string` vers un `Message`.
19. Écrire un sous-programme permettant la conversion d'un `Message` vers un `std::string`.
20. Reprendre le programme principal de manière à effectuer les traitements suivants (2 parties) :
    (a) Générer aléatoirement un `Message` et une `Cle` ; les afficher ; chiffrer ce `Message` et afficher le résultat ; déchiffrer ce dernier et afficher le résultat.
    (b) Saisir un message sous forme d'une chaine de caractères ; le chiffrer au moyen de la `Cle` précédente ; afficher le message chiffré (sous forme d'une chaine de caractères) ; déchiffrer ce dernier et afficher le résultat.