Chiffrement

3h : avec documents

Année 2020-2021

Préambule

- Les contrats et tests unitaires de la plupart les fonctions vous sont donnés. Vous pouvez compléter les tests unitaires si vous en ressentez le besoin. Les contrats et tests unitaires des fonctions auxiliaires que vous auriez besoin d'ajouter doivent être fournis.
- Pensez à décommenter les tests unitaires des fonctions que vous écrivez au fur et à mesure pour tester vos fonctions.
- Les noms et types des modules et fonctions doivent être respectés.
- Pour tester dans `utop`, vos fonctions du modules Arbre : **open** Be.Arbre;;
- Le code rendu doit impérativement compiler (lorsque vous avez une partie de code qui ne compile pas, mettez-la en commentaire).

Attention

Le sujet ressemble au BE de l'an dernier mais la structure arborescente utilisée n'est pas la même :

- arbre n-aire et non ternaire ;
- données (de type quelconque) dans les branches ;
- branches non ordonnées.

Pour ceux qui ont travaillé le BE de l'an dernier, je vous conseille de vous détacher du code que vous avez fait (sinon vous risquez de l'adapter très maladroitement) même si bien sûr les principes de certains algorithmes seront similaires.

1 Sujet d'étude

1.1 Cadre général

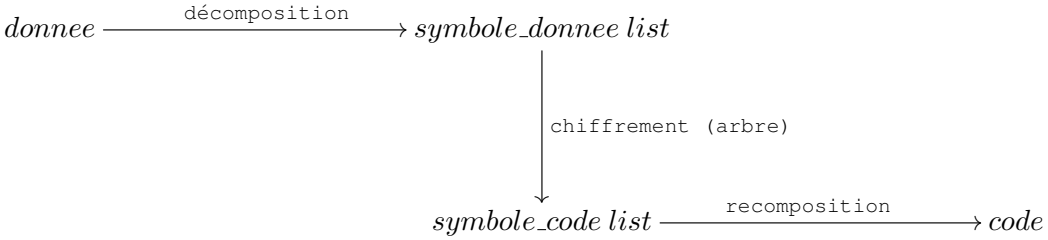
Le but du sujet est d'utiliser les arbres pour chiffrer des données. Pour cela nous manipulerons quatre types de données :

- $donnee$: le type des données à chiffrer (par exemple string)
- $symbole\_donnee$: le type des symboles de l'alphabet de $donnee$ (par exemple char)
- $code$: le type des codes (par exemple int)
- $symbole\_code$: le type des symboles de l'alphabet de $code$ (par exemple chiffre)

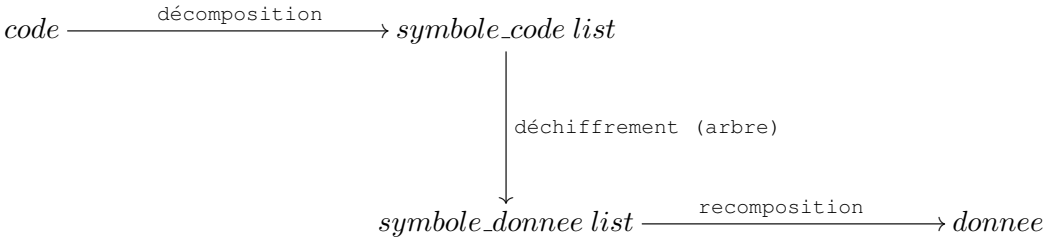
Pour chiffrer une donnée, un arbre de chiffrement contenant des données de type $symbole\_donnee$ dans les feuilles et des données de type $symbole\_code$ dans les branches est utilisé. Le chemin pour atteindre un symbole de type $symbole\_donnee$ depuis la racine donne son code ($symbole\_code\ list$). Le processus de chiffrement utilisé est le suivant :

1. la donnée à chiffrer ($donnee$) est décomposée en une liste de symboles de son alphabet ($symbole\_donnee\ list$) ;
2. les codes de chacun des symboles sont concaténés ($symbole\_code\ list$) ;
3. la liste obtenue est recomposée pour obtenir le code ($code$).

chiffrement d'une donnée



Déchiffrement d'une donnée



Sources et vision globale du travail demandé. Entre les types $donnee$ et $symbole\_donnee$, tout comme entre les types $symbole\_code$ et $code$, il est nécessaire d'avoir des fonctions de décomposition et recomposition. C'est le sujet de l'exercice 1. Ces fonctions sont fournies sur les chaînes (`chaines.mli` et `chaines.ml`), vous devrez les écrire pour les entiers (`entiers.mli` et `entiers.ml`).
Entre les types $symbole\_donnee$ et $symbole\_code$, il est nécessaire d'avoir des fonctions de chiffrement et déchiffrement. C'est le sujet de l'exercice 2 et de l'exercice 4 (bonus).
L'exercice 3 permet de faire la "glu" entre les exercices 1 et 2 et donc d'avoir les deux chaînes complètes : $donnee \rightarrow code$ et $code \rightarrow donnee$.
A l'image de ce qui a été fait dans le TP sur les arbres lexicographiques, il y a deux modules :

— le module Arbre qui ne travaille que sur l'arbre de chiffrement (exercice 2 et 4) ;
— le module Chiffrement qui travaille sur la chaîne complète, donc en intégrant les fonctions de décomposition et recomposition (exercice 3).

1.2 Exemples

1.2.1 Un texte chiffré par un entier

Si nous souhaitons chiffrer un texte par un entier, nous pourrons choisir :
- $donnee$ = string (on veut chiffrer un texte)
- $symbole\_donnee$ = char (le texte sera découpé en une suite de caractères)
- $symbole\_code$ = int (chaque caractère sera chiffré par un entier)
- $code$ = int (la donnée sera chiffrée à l'aide d'un entier)

En utilisant l'arbre de chiffrement suivant :

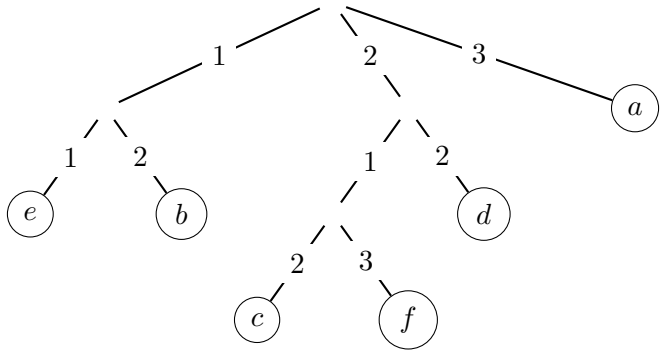


FIGURE 1 – arbre1

'a' se chiffre 3, 'b' se chiffre 12, 'c' se chiffre 212, ... Le mot "bac" se chiffre 123212.

Mais en utilisant

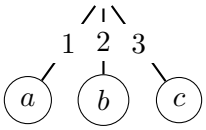


FIGURE 2 – arbre2

le mot "bac" se chiffre 213.

1.2.2 Un texte chiffré par un autre texte

Si nous souhaitons chiffrer un texte par un autre texte, nous pourrons choisir :
- $donnee$ = string
- $symbole\_donnee$ = char
- $symbole\_code$ = char
- $code$ = string

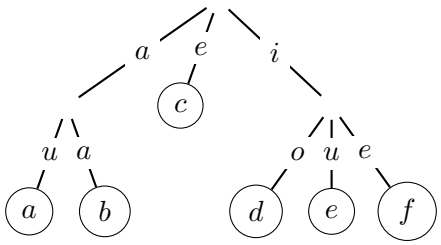


FIGURE 3 – arbre3

En utilisant l'arbre de chiffrement suivant :
'a' se chiffre "au", 'b' se chiffre "aa", 'c' se chiffre "e", ... Le mot "bac" se chiffre "aaaue".

Mais en utilisant (décalage des lettres)

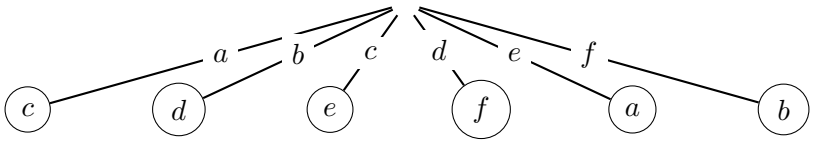


FIGURE 4 – arbre4

le mot "bac" se chiffre "fea".

2 Travail à réaliser

▷ **Exercice 1** *A réaliser dans le fichier* `entiers.ml`.
Le but de l'exercice 1 est d'écrire les fonctions de décomposition et recomposition d'un entiers vers / depuis une liste de chiffres.
Penser à bien lire les contrats des deux fonctions dans le fichier `entiers.mli`. *Les implantations des deux fonctions devront respecter ces contrats.*
Il n'est pas autorisé de passer par une conversion des entiers en chaînes de caractères et d'utiliser les fonctions de décomposition / recomposition des chaînes.

1. *Définir l'exception* ZeroException.
2. *Écrire la fonction* decompose, *dont le contrat est donné dans le fichier* `entiers.mli` *et donner son jeu de test. Par exemple* decompose 248 = [2;4;8].
   Aide : Vous avez le droit d'utiliser une fonction auxiliaire.

```
# 248 mod 10 ;;
- : int = 8
# 248 / 10 ;;
- : int = 24
```

3. *Écrire la fonction* recompose*, dont le contrat est donné dans le fichier* `entiers.mli` *et donner son jeu de test. Pour toute liste* l *et entier* e *:* decompose e = l $\Leftrightarrow$ recompose l = e.
   Aide : Vous avez le droit d'utiliser List.rev.

▷ **Exercice 2** *A réaliser dans le fichier* `arbre.ml`.

1. *Définissez le type* arbre_chiffrement *, sachant que le code OCaml des quatre arbres du sujet vous est donné (*arbre1*,* arbre2*,* arbre3 *et* arbre4*). Pensez à décommenter* arbre1*,* arbre2*,* arbre3 *et* arbre4 *une fois votre type* arbre_chiffrement *défini.*
2. *Écrire la fonction* get_branche *qui cherche dans un arbre de chiffrement le sous-arbre associé à un symbole de l'alphabet des codes.*
   Aide : Vous pouvez utiliser la fonction List.assoc_opt *donc le contrat vous est rappelé en annexe.*
3. *Écrire la fonction* dechiffrer *qui prend un code (sous forme de liste de symboles de l'alphabet des codes) et un arbre de chiffrement et renvoie la donnée associée (sous forme de liste de symboles de l'alphabet des données).*
4. *Écrire une fonction* arbre_to_liste *qui renvoie, pour un arbre de chiffrement donné, la liste qui associe à un symbole de l'alphabet des données son code (liste de symboles de l'alphabet des codes).*
5. *Écrire une fonction* chiffrer *qui prend une donnée (sous forme de liste de symboles de l'alphabet des données) et un arbre de chiffrement et renvoie le code associé (sous forme de liste de symboles de l'alphabet des codes).*

▷ **Exercice 3** *A réaliser dans le fichier* `chiffrement.ml`.
Le squelette du module Chiffrement *vous est donné.*

1. *Compléter le module* Chiffrement *en spécifiant le type* chiffrement*. Il doit inclure un arbre de chiffrement et les fonctions de décomposition et recomposition des données et des codes.*
2. *Définir les variables* c_int_1*,* c_int_2*,* c_texte_3 *et* c_texte_4*, qui serviront pour les tests et qui correspondent aux exemples développés à l'aide des arbres des figures 1 à 4.*
3. *Écrire la fonction* dechiffrer *qui prend un code et un arbre de chiffrement et renvoie la donnée associée.*
4. *Écrire une fonction* chiffrer *qui prend une donnée et un arbre de chiffrement et renvoie le code associé.*

▷ **Exercice 4** ***BONUS***

— *Définir un itérateur* fold *sur la structure d'arbre. Donner son type. Les tests unitaires ne sont pas demandés (il sera validé avec les fonctions suivantes).*
Aide : Un fold*, en plus de la structure sur laquelle il itère, prend en paramètre une fonction par constructeur du type (fonction de même arité que le constructeur).*
— *Écrire une fonction* nb_symboles *permettant de calculer le nombre de symboles de l'alphabet des données dans l'arbre de chiffrement, à l'aide de l'itérateur précédent.*
— *Écrire une fonction* symboles *permettant de construire la liste des symboles de l'alphabet des données présents dans l'arbre de chiffrement, à l'aide de l'itérateur précédent.*
— *Écrire une fonction* arbre_to_liste_2 *, de même spécification que* arbre_to_liste *, à l'aide de l'itérateur précédent.*

Annexe

List.assoc_opt

```
(* assoc_opt a l returns the value associated with key a in the list of pairs l.
That is, assoc_opt a [ ...; (a,b); ...] = Some b if (a,b) is the leftmost binding of a in list l.
Returns None if there is no value associated with a in the list l.*)

val assoc_opt : 'a -> ('a * 'b) list -> 'b option
```