Codage de Huffman – Connecteur de Sheffer

laurent.jospin.59@free.fr, http://jospin.lstl.fr

Lycée Saint-Louis, Paris

Le sujet est composé d'un exercice et d'un problème totalement indépendant.

Exercice – Connecteur de Scheffer

Le connecteur de Sheffer est défini par $a|b = \neg(a \wedge b)$.

1. Donner la table de vérité du connecteur de Sheffer

2. Montrer que toute formule logique peut s'écrire avec cet unique connecteur logique. On dit que ce connecteur est universel.

3. Montrer que $a|(b|c) \equiv a \rightarrow (b \wedge c)$.

4. Montrer que $a \rightarrow b$ implique logiquement que $(p|b) \rightarrow (a|p)$. Y a-t-il équivalence logique ?

5. Déterminer quels autres connecteurs binaires sont universels.

1 Structure de file de priorité inverse

1.1 Arbre ayant la structure de tas min

Un arbre *ayant la structure de tas min* est un arbre binaire (non nécessairement strict) tel que l'étiquette des fils (lorsqu'ils existent) soient toujours inférieure à l'étiquette de la racine et tel que les sous arbres aient également la structure de tas min.

On supposera défini le type arbre suivant :

```
type 'a arbre = Noeud of 'a * 'a arbre * 'a arbre | Vide
```

6. Représenter un arbre de hauteur 2 ayant la structure de tas min et contenant exactement les valeurs : `15, 3, 0, 8, 6, 42`.

7. Ecrire une fonction `arbre_tasmin_insere : 'a -> 'a arbre -> 'a arbre` pour insérer une valeur dans un arbre ayant la structure de tas min. On souhaite que la fonction produise un arbre relativement équilibré si on lui soumet une séquence d'insertion quelconque. Pour cela, on pourra insérer intervertir sous arbre gauche et sous arbre droit au moment de l'insertion.

8. Ecrire une fonction `arbre_tasmin_extraitmin : 'a arbre -> ('a, 'a arbre)` qui renvoie la valeur minimale stockée dans l'arbre et l'arbre privée de cette valeur minimale.

9. Lorsque l'arbre est équilibré, c'est à dire si la différence de profondeur entre les feuilles n'excède pas 1, quelle est la complexité des opérations précédentes ?

10. Ecrire une fonction `arbre_tasmin_taille : 'a arbre -> int` qui renvoie le nombre de valeurs stockées dans l'arbre ayant la propriété de tas min pris en argument.

Préciser la complexité.

1.2 File de priorité inverse

Une file de priorité inverse est une structure de données ayant pour interface :

— `filedepriorite_cree_vide : unit -> 'a filedepriorite` qui crée une structure contenant aucune valeur ;

— `filedepriorite_insere : 'a -> int -> 'a filedepriorite -> unit` qui prend en argument une valeur et sa prioirité et qui l'insère dans la structure ;

— `filedepriorite_extraitmin : 'a filedepriorite -> (int, 'a)` qui retire de la stucture le couple constitué de la priorité minimale et de la valeur associée et le renvoie ;

— `filedepriorite_taille : 'a filedepriorite -> taille` qui donne la taille de la file de priorité.

11. Pour réaliser la structure de file de priorité inverse, on réalise un arbre ayant la propriété de tas min dans laquelle on stocke les couples `(priorite, valeur)`. On souhaite de plus que la fonction `taille` ait une complexité en temps constant.

Proposer un type enregistrement `'a filedepriorite` pour stocker de façon mutable un arbre ayant la structure de tas min et contenant des couples `(int, 'a)` et sa taille. Implémenter les fonctions de l'interface en réutilisant lorsque c'est pertinent les fonctions sur les arbres ayant la propriété de tas min.

2 Lecture d'un fichier

On suppose disposer d'une fonction `epelle : unit -> char` qui énumère les lettres d'un texte. La fonction utilise un curseur qui se déplace dans le fichier à chaque lecture de caractère. La fonction déclenchera une exception `Fin_de_fichier` lorsqu'on essaye de lire dans le fichier qui a été entièrement parcouru.

Plus précisément, la fonction `epelle` est renvoyée par une fonction `cree_epelle : string -> (unit -> char )` qui prend en argument l'adresse d'un fichier (sous la forme d'une chaîne de caractères) et qui renvoie la fonction `epelle`. Ainsi deux appels à `cree_epelle adresseFichier` donne deux fonctions `epelle` distinctes avec un curseur propre à chacune.

La syntaxe pour intercepter une exception est :

```
1 try
2    expression1 (* pouvant déclencher une exception *)
3 with
4    Exception_a_intercepter ->  expression2 (* à évaluer cas d'exception *)
```

Les deux expressions (ou séquence d'instructions) doivent s'évaluer en une valeur d'un même type.

12. En déduire une fonction `cree_epelle_opt` qui renvoie une fonction `epelle_opt : unit -> char option` qui renvoie elle-même soit `None` quand la fin du fichier est atteinte, soit `Some c` où c est le caractère lu.

13. Ecrire une fonction `compte_occurences : string -> int array` qui prend en argument un nom de fichier et qui renvoie un tableau de 256 cases qui à chaque indice i renvoie le nombre d'apparitions du caractères correspondant au nombre i.

3 Codage de Huffman

L'algorithme de Huffman est un algorithme de compression de données sans perte. Le principe consiste à rompre la correspondance entre un caractère et un octet en attribuant des codes plus courts aux caractères plus fréquemment utilisés et des codes plus longs aux caractères plus rares. Dans un fichier texte, la distribution de fréquences des caractères n'est pas du tout uniforme, on peut donc espérer réduire considérablement la taille d'un fichier avec cette idée.

Dans la description de l'algorithme, on appelle *collection* une structure de données contenant plusieurs valeurs sans préciser le fonctionnement de la structure.

L'algorithme de Huffman consiste à compter les occurences de tous les caractères du texte à compresser puis construire un arbre binaire strict avec l'algorithme suivant :

1. Construire une collection d'arbres qui stocke des arbres et leur poids et contenant initialement les feuilles étiquetées par chacun des caractères apparaissant dans le texte (sans doublon) ayant pour poids le nombre d'occurences du caractère correpondant.

2. Tant qu'il y a au moins deux arbres dans la collection, extraire les deux arbres de poids minimum, puis replacer dans la collection un nouvel arbre ayant pour poids la somme des poids des deux arbres extraits et ayant pour fils les deux arbres extraits.
3. Renvoyer le seul arbre de la collection

Ainsi, la collection utilisée dans cet algorithme aura idéalement une structure permettant l'insertion d'une valeur conjointement avec un poids et l'extraction de la valeur de poids minimal stockée dans la collection en temps efficace, on utilisera donc la structure de file de priorité inverse construite dans la première partie en utilisant le poids comme priorité et des arbres comme valeur.

Les noeuds internes ne sont pas étiquettés.

14. Proposer un type adapté pour représenter les arbres utilisés dans cette partie.

15. Ecrire une fonction qui prend un tableau d'occurences et qui renvoie une file de priorité inverse remplie avec les feuilles correspondant à chaque caractère d'occurence non nulle et ayant pour poids cette occurence.

Préciser le type exact de la fonction ainsi écrite.

16. Ecrire la fonction qui prend en argument la structure précédente et renvoie l'arbre de Huffman correspondant en appliquant l'algorithme de construction d'arbre décrit.

17. Pour obtenir le codage d'un caractère, on attribue à chaque branche le code 0 (branche gauche) ou 1 (branche droite). Le code d'un caractère est la liste des codages des branches qui le séparent de la racine (de la racine jusqu'à la feuille). Ecrire une fonction qui prend un arbre de Huffman et renvoie une tableau de 256 cases qui à chaque indice i renvoie le code du caractère d'indice i. Les caractères n'apparaissant pas auront un code réduit à la liste vide.

18. Le code d'un texte avec le codage de Huffman est la concaténation du code des caractères (dans le même ordre que dans le texte évidemment) de ce texte. Justifier que ce code est non ambigu c'est-à-dire que si on connaît l'arbre et qu'on dispose du codage de Huffman du texte, on peut reconstruire le texte sans ambiguïté.

19. Ecrire une fonction qui prend en argument l'arbre et un texte compressé par le codage de Huffman et qui renvoie la liste des caractères du texte d'origine.