PIM : Projet 3

Raffinages de Compresser

Déclaration des types :

type treeNodePointer est **POINTEUR** treeNode
type treeNodeArray est **TABLEAU** (1 .. 256) de treeNodePointer
type treeQueue est **ENREGISTREMENT**
    last : treeNodePointer
    storageArray : treeNodeArray
    realSize : entier
fin ENREGISTREMENT
type treeNode est **ENREGISTREMENT**
    symbol : chaîne de caractères
    occurrences : entier
    rightChild : treeNodePointer
    leftChild : treeNodePointer
    parent : treeNodePointer
    isSeen : Booléen
fin ENREGISTREMENT

Déclaration des procédures :

procedure BuildHuffmanTree (symbolsHashTable : i**n** hashMap; binaryTree : **out** treeNodePointer) **EST**

current : entier = 1
storageTree : treeQueue
newNode : treeNodePointer
storageTree.realSize = 0
storageTree.last = null
Pour k allant de 1 à 256 **Faire**
storageTree.storageArray (k) = null
Fin Pour
Pour i allant de 1 à hashTableSize **Faire**
Si symbolsHashTable.entryNodeArray (i) != null **Alors**
newNode = **Nouveau** treeNode'
(symbolsHashTable.entryNodeArray (i).key, symbolsHashTable.entryNodeArray (i).value, null, null, null, False)
storageTree.storageArray (current) = newNode
current = current + 1
Fin Si
Fin Pour
storageTree.realSize := current - 1
storageTree.last := storageTree.storageArray (storageTree.realSize)
Tant que storageTree.realSize /= 1 **Faire**
SortArray (storageTree)
CreateNode (storageTree)
Fin Tant que
binaryTree = storageTree.storageArray (1)
Fin BuildHuffmanTree

procedure SortArray (storageTree : **in out** treeQueue) **EST**

intermediateValue : treeNodePointer
Pour i allant de 1 à (storageTree.realSize - 1) **Faire**
Pour j allant de 1 à (storageTree.realSize - 1) **Faire**

```
                    Si storageTree.storageArray (j).occurrences <
storageTree.storageArray (j + 1).occurrences Alors
                        intermediateValue = storageTree.storageArray (j)
                        storageTree.storageArray (j) =
                    storageTree.storageArray (j + 1)
                        storageTree.storageArray (j + 1) = intermediateValue
                    Fin Si
                Fin Pour
            Fin Pour
        Fin SortArray
```

```
procedure CreateNode (storageTree : in out treeQueue) EST
        firstNode : CONSTANT treeNodePointer = storageTree.storageArray
(storageTree.realSize)
        secondNode : CONSTANT treeNodePointer = storageTree.storageArray
(storageTree.realSize - 1)
        newNode : treeNodePointer

        newNode := nouveau treeNode' ("--------", secondNode.occurrences +
firstNode.occurrences, secondNode, firstNode, null, False)
        firstNode.parent := newNode;
        secondNode.parent := newNode
        storageTree.last := newNode
        storageTree.realSize := storageTree.realSize - 1
        storageTree.storageArray (storageTree.realSize) := newNode
Fin CreateNode
```

```
  procedure InfixBrowsing (it : in out Entier; symbolsHashTable : in hashMap;
binaryTree : in treeNodePointer; infixTree : out Chaîne de caractères; encodedFile :
in out Fichier) EST
```

```
        current : treeNodePointer
        Si it < symbolsHashTable.size Alors
                current = binaryTree
                Si current.leftChild != null Alors
                        Si current.leftChild.isSeen and current.rightChild.isSeen Alors
                                current.isSeen := True
                                InfixBrowsing (it, symbolsHashTable, current.parent,
infixTree, encodedFile)
                        Sinon Si current.leftChild.isSeen Alors
                                InfixBrowsing (it, symbolsHashTable, current.rightChild,
infixTree, encodedFile)
                        Sinon
                                infixTree := infixTree & "0"
                                current := current.leftChild
                                InfixBrowsing (it, symbolsHashTable, current, infixTree,
                        encodedFile)
                        Fin Si
                Sinon
                        current.isSeen := Vrai
                        infixTree := infixTree & "1"
                        it := it + 1
                        PutSymbols (symbolsHashTable, it, current.symbol,
                encodedFile)
                        InfixBrowsing (it, symbolsHashTable, current.parent, infixTree,
                encodedFile)
                        Free3 (current)
                Fin Si
        Fin Si
  Fin InfixBrowsing

  procedure PutSymbols (symbolsHashTable : in hashMap; it : in Entier; symbol : in
String; encodedFile : in out File_Type) EST

        Put (encodedFile, symbol);
        Si it = symbolsHashTable.size Alors
                Put (encodedFile, symbol);
```

Fin Si

Fin PutSymbols

procedure ExploreTree (binaryTree : **in** treeNodePointer; code : **in out** Chaîne de caractères; encodedSymbols : **in out** hashMap2) **EST**

leftCode, rightCode : **Chaîne de caractères**

Si binaryTree.leftChild = null **Alors**

Register2 (encodedSymbols, To_Unbounded_String (binaryTree.All.symbol), code) – fonction de THCHAR

Sinon

leftCode = code & "0"

ExploreTree (binaryTree.leftChild, leftCode, encodedSymbols)

rightCode = code & "1"

ExploreTree (binaryTree.rightChild, rightCode, encodedSymbols)

Fin Si

Fin ExploreTree

Raffinages :

R0 : Compresser un fichier binaire à l'aide du codage de Huffman.

R1 : **Comment** " Compresser un fichier binaire à l'aide du codage de Huffman. " ?

Construire l'arbre de Huffman Texte_A_Compresser : **In Fichier**
Créer le fichier compressé avec les informations demandées
Si Mode = Bavard **Alors** Bavard **: In Chaîne de caractères**
Afficher l'arbre de Huffman et la table de Huffman
FinSi

R2 : Comment " Construire l'arbre de Huffman " ?

Construire le tableau des fréquences des symboles sous forme d'une table de hachage
Construire les noeuds de l'arbre

R2 : Comment "Créer le fichier compressé avec les informations demandées. "?

Créer (encodedFile)
Stocker dans le fichier compressé les symboles utilisés
Stocker dans le fichier compressé la structure de l'arbre
Stocker dans le fichier compressé le codage du fichier à compresser

R2 : Comment " Afficher l'arbre de Huffman et la table de Huffman. " ?

Afficher l'arbre de Huffman

Afficher la table de Huffman

R3 : Comment " Construire le tableau des fréquences des symboles sous forme d'une table de hachage ?

Initialiser la table de hachage à l'aide du module TH.
Ouvrir le document à compresser
Tant que non fin du fichier **faire** result : **chaîne de caractères** (taille 8)
Lire 8 caractères (un octet) et affecter leur valeur à result
Enregistrer le caractère dans la tableau de hachage selon sa présence dans ce dernier
Fin TantQue
Enregistrer à l'aide de la fonction register de TH le symbole marquant la fin
Fermer le document à compresser.

R3 : Comment " Construire les nœuds de l'arbre de Huffman. " ?

Construction des noeuds l'arbre de Huffman par appel récursif de la procédure BuilHuffmanTree sur une liste triée et s'arrêtant quand la taille de la liste vaut 1

R3 : Comment " Stocker dans le fichier compressé les symboles utilisés. " ?
Pour i allant de 1 à 256 **Faire**
Ajouter au fichier compressé le symbole
Fin Pour

R3 : Comment " Stocker dans le fichier compressé la structure de l'arbre par parcours infixe de ce dernier. " ?

Réaliser le parcours infixe de l'arbre
Mettre(encodedFile, infixTree)

R3 : Comment " Stocker dans le fichier compressé le codage du fichier à compresser " ?

Obtenir le code de chaque symbole par récursivité avec la procédure ExploreTree avec comme critère l'arrêt le fait d'arriver sur une feuille

Ecrire dans le fichier compressé le texte du fichier à compresser

R3 : Comment " Afficher l'arbre de Huffman. " ?

Afficher l'arbre de Huffman selon les contraintes imposées

R3 : Comment " Afficher la table de Huffman. " ?

Afficher la table de Huffman selon les contraintes imposées

R4 : Comment " Lire 8 caractères (un octet) et affecter leur valeur à result" ?

Pour i allant de 1 à 8 **Faire**
Obtenir (caractère_fichier)
result(i) = caractère_fichier
Fin Pour

R4 : Comment " Enregistrer le caractère dans la tableau de hachage selon sa présence dans ce dernier" ?

Si Result est dans la table de hachage **Alors**
Ajouter un à la valeur associé à la clé Result
Sinon
Enregistrer dans la table de hachage à l'indice correspondant au code ASCII du symbole à l'aide de la fonction register de TH la clé Result avec pour valeur 1

R4 : Comment " Ajouter au fichier compressé le symbole" ?

Si hashtable(i) /= Null **Alors**
    Put(encodedFile, hashtable(i).symbol)
Fin Si

R4 : Comment “Réaliser le parcours infixe de l’arbre.” ?

Obtenir le code du parcours infixe de l’arbre par récursivité de la procédure InfixBrowsing avec comme critère l’arrêt le test de si on a atteint toutes les feuilles de l’arbre.

Raffinages de Décompresser

Déclaration des types :

type treeNodePointer est **POINTEUR** treeNode
type treeNodeArray est **TABLEAU** (1 .. 256) de treeNodePointer
type treeQueue est **ENREGISTREMENT**
    last : treeNodePointer
    storageArray : treeNodeArray
    realSize : entier
fin ENREGISTREMENT
type treeNode est **ENREGISTREMENT**
    symbol : chaîne de caractères
    occurrences : entier
    rightChild : treeNodePointer
    leftChild : treeNodePointer
    parent : treeNodePointer
    isSeen : Booléen
fin ENREGISTREMENT
type stringArray est **TABLEAU** (1 .. 256) de chaîne de caractères

Déclaration des procédures :

procedure ReconstructHuffmanTree (it : **in out** Integer; countInfix : **in out** Integer; encodedFile : **in** File_Type; infixTree : **in out** Unbounded_String; current : **in out** treeNodePointer; symbolsArray : **in** stringArray) **EST**

infixString : **CONSTANT** String:= To_String (infixTree)
rightChild : treeNodePointer
leftChild : treeNodePointer
toCall : treeNodePointer

Si infixString (countInfix) = '0' **Alors**
rightChild = **nouveau** treeNode' ("--------", 0, null, null, current, False);
leftChild = **nouveau** treeNode' ("--------", 0, null, null, current, False);
current.leftChild = leftChild
current.rightChild = rightChild
toCall = leftChild
Sinon
current.isSeen = True
current.symbol = symbolsArray (it)
Si current.parent.isSeen **Alors**
Tant Que current.isSeen **Faire**
Si current.parent = null **Alors**
return
FinSi
current = current.parent
Fin Tant Que
current.isSeen = True
toCall = current.rightChild
Sinon
toCall = current.parent.rightChild
current.parent.isSeen := True
Fin Si
it = it + 1
Fin Si
countInfix = countInfix + 1

```
    ReconstructHuffmanTree (it, countInfix, encodedFile, infixTree, toCall,
symbolsArray)
  Fin ReconstructHuffmanTree


Procedure ExploreTree  (root : in  treeNodePointer; encodedFile : in out File_Type;
decodedFile : in out File_Type ) is

  code : chaîne de caractères

    Si root.leftChild = null Alors
      Si root.symbol /= "11111111" Alors
        Put (decodedFile, root.symbol)
      Fin Si
      return
    Fin Si
    Get (encodedFile, code)
    Si code = "0" Alors
      ExploreTree (root.leftChild, encodedFile, decodedFile);
    Sinon
      ExploreTree (root.rightChild, encodedFile, decodedFile);
    FinSi
  Fin ExploreTree
```

Raffinages :

R0 : Décompresser un fichier binaire à l'aide du codage de Huffman.

R1 : Comment " Décompresser un fichier texte à l'aide du codage de Huffman " ?

Lire les données des symboles et du parcours infixe du fichier compressé
encoded_File **in** : fichier , symbolcode **out** : chaîne de caractères , symbolArray : **out** stringArray
Reconstruire l'arbre de Huffman à partir du parcours infixe infixTree : **in** chaîne de caractères
Tant que non Fin de encoded_File **Faire**
Décoder la séquence
Afficher le texte dans un nouveau fichier décompressé
Fin Tant Que

R2 : Comment " Lire les données des symboles et du parcours infixe du fichier compressé" ?

previouscode : chaîne de caractères = symbolcode
Tant que previouscode != symbolecode **Faire**
Récupérer le code de chaque symbole
Fin Faire
Lire le code infixe

R2: Comment " Reconstruire l'arbre de Huffman à partir du parcours infixe"

Reconstruire l'arbre à l'aide d'un appel récursif de la procédure RebuildHuffmanTree qui a pour critère d'arrêt le fait qu'on a atteint toutes les feuilles

R2: Comment " Décoder la séquence" ?

Décoder la séquence par appel récursif de la procédure ExploreTree avec comme critère d'arrêt l'atteinte de la fin du fichier compressé

R2: Comment " Afficher le texte dans un nouveau fichier décompressé ?

Créer (Decoded_File)

Ajouter chaque symbole du texte au fichier décompressé par appel récursif de la procédure ExploreTree avec comme critère d'arrêt l'atteinte de la fin du fichier compressé

R3 : Comment "Récupérer le code de chaque symbole" ?

Pour i allant de 1 à 8 **Faire**
    Obtenir le caractère du fichier
    Concaténer le caractère à symbolcode
Fin Pour
Ajouter symbolcode à symbolArray

R3: Comment "Lire le code infixe" ?

length : Integer = longueur symbolArray
Tant que length < 2
    Récupérer le caractère du fichier
    Concaténer ce caractère à infixTree
    Vérifier si le caractère correspond à une feuille
Fin Tant que

R4: Comment" Vérifier si le caractère correspond à une feuille" ?

Si caractère = '1' **Faire**
    length = length - 1
Fin Si

Evaluation des raffinages

	Règle	Evaluation (I/P/A/+)	Justification/commentaire (optionnel)	Inacceptable (I)	Partiellement atteint (P)	Atteint (A)
Forme	**Respect de la syntaxe** **Ri : Comment "... une action complexe ..." ? des actions combinées avec des structures de controle** **Rj : ...**	A		Pas de numéro de raffinage (R0, R1, R2...) OU Pas de "Comment" OU Pas de rappel de l'action/expression complexe OU Pas de décomposition ou elle est mal indentée OU les Ri sont mentionné dans l'algo complet (non séparés)	l'action/expression est donnée ET la décomposition est donnée MAIS Erreur sur le numéro (niveau) de raffinage OU Pas de décalage pour les actions	Le bon numéro pour le raffinage ET le "Comment" est là ET l'action/complexe est donnée ET la décomposition est donnée indentée ET les actions sont décalées
	Verbe à l'infinitif pour les actions complexes	A		Pas ou peu de verbes à l'infinitif (moins de 20 %)	Moins de 70 % de verbes à l'infinitif	Tous les verbes sont à l'infinitif
	Nom ou équivalent pour expressions complexes	A		Utilisation de verbes à l'infinitif pour nommer une expression complexe	Moins de 10 % d'expressions complexes mal	Aucun verbe à l'infinitif pour nommer une expression complexe

					nommées	
	Tous les Ri sont écrits contre la marge et espacés	A		Les Ri sont décalés en fonction de la valeur de i ET il n'y a pas de ligne blanche entre eux	Les Ri sont bien contre la marge OU 1 ou 2 lignes blanches séparent les Ri	Tous les Ri sont contre la marge ET 1 ou 2 lignes blanches les séparent
	Les flots de données sont définis	A		Pas de flot de données	Le flot de données est donné pour quelques actions ou expressions complexes OU les in et out ne sont pas donnés OU le type n'est pas donné lors de la première apparition d'une donnée	Les flots de données sont donnés pour presque toutes les actions ou expressions complexes ET Les in et les out sont présents ET le type d'une donnée est défini à sa première apparition
	Une seule structure de contrôle par raffinage	A		3 ou plus structures de contrôle (conditionnelle ou répétition) dans un raffinage	2 décisions ou répétitions au plus par raffinage	au plus 1 décision ou répétition par raffinage
	Pas trop d'actions dans un raffinage (moins de 6)	A		Beaucoup trop d'actions : > 9	Trop entre 6 et 9	Moins de 6
	Bonne	A		Plus de 3	1 ou 2	Toutes les structures

	présentation des structures de contrôle			structures de contrôle sont mal présentées	structures de contrôle sont mal présentées	de contrôle sont présentées comme en cours
Fond	**Le vocabulaire est précis**	A		On ne comprend pas plusieurs noms utilisés : pas significatifs OU utilisation d'abréviations OU mélange de plusieurs langues	On ne comprend pas quelques noms.	On comprend tous les noms ET pas d'abréviations ET pas de mélange de plusieurs langues
	Le raffinage d'une action décrit complètement cette action	A		Pour au moins 2 décompositions (raffinages) : Il manque des actions OU il manque des structures de contrôle (répétition, décision) OU il y a des actions interdites (comme Recommencer, Continuer en...)	Idem mais pour 1 décomposition seulement	Toutes les décompositions respectent les trois conditions.
	Le raffinage d'une action ne décrit que cette action	A		Pour au moins 2 décompositions : Il y a des actions en trop	Idem mais pour 1 décomposition seulement	Aucune action en trop dans aucune décomposition
	Les flots de données sont cohérents	A		Au moins 2 fois, on constate : des in avant des out pour des données locales OU des out sur	Idem mais on le constate 1 fois seulement	Les 4 règles sont toujours respectées

				des données en in de l'action complexe OU pas de out sur des données en out de l'action complexe OU des actions ne sont pas dans l'ordre vis à vis du flot		
	Pas de structure de contrôle déguisée	A		Au moins 2 structures de contrôle déguisées (Faire ... si ..., Incrémenter tant que... calculer selon ...)	1 structure de contrôle déguisée	Aucune structure de contrôle déguisée
	Qualité des actions complexes	A		Au moins 2 fois, certaines actions ne sont pas explicites (i.e. on ne peut pas exécuter mentalement sans regarder le raffinage de certaines de ces actions) OU des actions pourraient être regroupées en une action complexe	Idem mais sur 1 raffinage seulement	Toujours respecté