Types et package utilisées pour la décompression et la compression:

package arbre EST nouveau arbre(T_donnee=>T_octet);

pckage lca_arbre is new lca(T_cle=>integer,T_donnee=>T_arbre)

pckage lca_integer is new lca(T_cle=>integer,T_donnee=>T_octet)

type T_octet est mod 256

type T_tab2 EST TABLEAU (1..257) de unbounded_string

I)Raffinage pour la compression de Huffamn:

R0: Compresser un texte

R1: Comment “compresser un texte”?

Pour i allant de 1 à argument_count Faire

Lire le fichier et stocker les caractères ainsi que leur fréquence. frequence:out lca_frequence.t_lca

Construire l’arbre de Huffman frequence_arbre:in out lca_arbre.T_lca arbre: out T_arbre

Réaliser la table contenant les codes de huffman des caractères du texte. Tab_code: out T_tab, in frequence_sup,arbre: in arbre

Obtenir le parcours infixe et les caractères dans l’ordre infixe de l’arbre code_parcours_infixe:out unbounded_string, tab_caractere:out T_tab,in arbre

Encoder le texte

Fin Pour

R2:Comment”Lire le fichier et stocker les caractères ainsi que leur fréquence”?

initialiser(frequence)

Ouvrir le fichier texte en entrée

Définir un stream S S:out stream_access

Tant que le fichier n’est pas lu entièrement FAIRE

octet<- T_octet'input(S)

SI donnee_presente(frequence,octet)

enregistrer(frequence,la_cle(frequence,octet)+1,octet)

SINON
enregistrer(frequence,1,octet)
FIN SI

FIN TANT QUE

– *on enregistre le symbole de fin de fréquence: associé à l'octet -1*
enregistrer(frequence,0,construction_feuille(0,-1));

fermer le fichier

R3: comment "ouvrir le fichier"?

open(file,in_file,argument(i));

R3: Comment "fermer le fichier"?

close(file);

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

Convertir la lca_integer contenant les caracteres et leur fréquence en une lca_arbre
out frequence_arbre: lca_arbre.T_lca

TANT QUE taille(frequence)>1

Obtenir l'arbre dont la fréquence associée à sa racine est minimale out arbre1

supprimer cette arbre de frequence_arbre

Obtenir l'arbre dont la fréquence associée à sa racine est minimale out arbre2

supprimer cette arbre de frequence_arbre

fusion(arbre1,arbre2) in out arbre 1

enregistrer(frequence,cle(arbre1),arbre1);

FIN TANT QUE

– *A ce stade on a une lca de taille 1 dont la donnée est l'arbre de huffman*

arbre← donnee(frequence)

RETOURNER arbre

R3: comment "Convertir la lca_integer contenant les caracteres et leur fréquence en une lca_arbre"?

– - *On place dans une T_lca.lca_arbre des feuilles contenant un caractère et sa fréquence:*

curseur ← frequence
TANT QUE not est_vide(curseur) FAIRE
enregistrer une feuille contenant l'octet de curseur et sa fréquence.
curseur ← suivante (curseur)
FIN TANT QUE

R4: Comment "enregistrer une feuille contenant la clé de curseur et sa donnée"

enregistrer(frequence_arbre,cle(curseur),construction_feuille(cle(curseur),donnee(curseur)))

R3:Comment "obtenir la clé associée à l'arbre de fréquence minimale dans fréquence_arbre"?

curseur← frequence_arbre

– *on initialise le min à la clé du premier arbre*

fonction cle_min(frequence:in T_lca);
cle_min←cle(frequence_arbre)
TANT QUE not est_vide(curseur) ALORS

SI cle(curseur)<min ALORS
cle_min ←cle(curseur)
FIN SI
curseur ← suivante(curseur)
FIN TANT QUE
RETOURNER cle_min

arbre1 ← la_donnee(frequence_arbre,cle_min(frequence_arbre))

R3: Comment"supprimer cet arbre de frequence"

surppimer(frequence,cle_min(frequence));

R2:Comment "Réaliser la table contenant les codes de huffman des caractères du texte"?

curseur←frequence

Tab_code(1)<-- code_asscie(arbre,-1) - - *on écrit le code de fin en première position* du tableau

TANT QUE est_vide(curseur) loop
indice←integer(donnee(curseur)) - - on prend comme indice l'entier associé à l'octet du caractère présent dans T_tab

tab_code(indice) ← code_associe(arbre,donnee(curseur));
curseur ← suivante(curseur)
FIN TANT QUE

R2: Comment"Obtenir le parcours infixe et les caractères dans l'ordre infixe de l'arbre "?

nb_feuille ← 0 - - nb_feuille:T_octet

code_parcours_infixe ← " "

Réaliser le parcours infixe et remplir code_parcours_infixe et tab_caractere

doubler le caractere de fin in out tab_caractere

R3: Comment"Réaliser le parcours infixe et remplir code_parcours_infixe et tab_caractere"?

fonction recursive infixe(arbre:in T_arbre):

SI not est_une_feuille(arbre) ALORS
code_parcours_infixe ← code_parcours_infixe & " "
infixe(gauche(arbre))
infixe(droite(arbre))
SINON
nb_feuille←nb_feuille+1

SI la feuille est celle du code de fin '/$' ALORS
Ecrire à la position nb_feuille du tableau le code binaire associé à nb_feuille
SINON
Ecrire à la position nb_feuille du tableau le code binaire associé à la feuille parcourue
FIN SI
code_parcours_infixe ← code_parcours_infixe & "1"

R4:Comment "Ecrire à la position nb_feuille du tableau le code binaire associé à la feuille parcourue"?

tab_caractere(integer(nb_feuille)) ← code binaire de nb_feuille

R4:Comment “Ecrire à la position nb_feuille du tableau le code binaire associé à la feuille parcourue”?

tab_caractere(integer(nb_feuille)) ← code binaire de donnee(arbre)

R2: Comment “Encoder le texte “?

Concatener les caracteres en parcours infixe de l’abre, le parcours infixe de l’arbre et le code de huffman du texte. out chaine : Unbounded_string, in tab_caractere , in code_parcours_infixe, in tab_codage

Créer un fichier .hff et écrire le texte compressé en octet

R3: Comment “Concatener les caracteres en parcours infixe de l’abre, le parcours infixe de l’arbre et le code de huffman du texte”?

```
chaine ← “ “
Pour i allant de 1 à length(tab_caractere) FAIRE
        chaine ← chaine & tab_caractere(i)
FIN POUR
chaine ← chaine & code_parcours_infixe  - - on concatène le parcours infixe

Ouvrir le fichier texte
s ← stream(file)

TANT QUE le fichier n’est pas lu jusqu’au bout FAIRE
        octet ← T_octet’input(s)
        Indice ←valeur de l’octet lu +1
        chaine:= ← chaine & tab_codage(indice)  - - on concatène le code de chaque caractères du texte

FIN TANT QUE
chaine ← chaine&tab_codage(1)    - - ajout du caractère de fin ‘/$’
```

R3: Comment “Créer un fichier .hff et écrire le texte compressé “

```
Créer un fichier .hff
définir un stream

POUR i allant de 1 à longueur(chaine) FAIRE
        Ecrire octets par 8 bits
        stocker le les bits qui ne forment pas un octet
FIN POUR
```

écrire le dernier octet incomplet en complétant avec des zéro

		Evaluation (I/P/A/+)
Forme (D-21)	Respect de la syntaxe Ri : Comment "... une action complexe ..." ? des actions combinées avec des structures de contrôle Rj : ...	A
	Verbes à l'infinitif pour les actions complexes	A
	Noms ou équivalent pour expressions complexes	A
	Tous les Ri sont écrits contre la marge et espacés	A
	Les flots de données sont définis	A
	Une seule décision ou répétition par raffinage	A
	Pas trop d'actions dans un raffinage (moins de 5 ou 6)	A
Fond (D21-D22)	Le vocabulaire est précis	+
	Le raffinage d'une action décrit complètement cette action	A
	Le raffinage d'une action ne décrit que cette action	A
	Les flots de données sont cohérents	A
	Pas de structure de contrôle déguisée	A
	Qualité	A

I)Raffinage pour la décompression de Huffamn:

Types et package utilisées pour la décompression et la compression:

type T_octet est mod 256

type T_tab2 EST TABLEAU (1..257) de unbounded_string
type T_tab EST TABLEAU (1..257) de unbounded_string

R0: décompresser un texte selon Huffman

R1:Comment “décompresser un texte selon Huffman”?

POUR chaque fichier en entrée FAIRE

Décomposer l’entête de l’entrée out texte_infixe:in unbounded_string,out indice_dollard:integer, out texte_code:unbounded_string,out texte_caractere:unbounded_string

Reconstruire la table de Huffman out table_huffman:T_tab 2

Décoder le texte out tab_element:T_tab , out nb_element: integer

FIN POUR

R2:Comment “ Décomposer l’entête de l’entrée “?

Ouvrir le fichier texte en entrée

compteur ← 1

Définir un stream S S:out stream_access

indice_dollards ← integerr(T_octet’input(s));

octet ←T_octet’input(s)

tab(1) ← octet *- - le tableau sert stock aussi les caractères présents dans l’arbre, il n’est pas utile pour la suite mais permet de comparer les octets entre eux au fil de la lecture du fichier pour savoir quand on atteint un doublon d’octet (ce qui signifie qu’on passe au parcours infixe).*

texte_caractere← texte_caractere&octet_binaire(octet)

compteur ← compteur +1

octet ←T_octet’input(s)

TANT QUE octet/=tab(compteur-1) **FAIRE**

tab(compteur) ← octet
texte_caractere← texte_caractere&octet_binaire(octet)
compteur ← compteur +1
octet ← T_octet’input(s)

FIN TANT QUE

tab(compteur)<-- octet
ajouter dans un unbounded_string l'octet lu

TANT QUE le fichier n'est pas entièrement ly **FAIRE**

octet ← T_octet'input(s)
texte:=texte&octet_binaire(octet) – la fonction octet_binaire a déjà été raffinée
Fin TANT QUE

limit ← 0 - - compte le nombre de '1'
i←1
TANT QUE limit/=compteur FAIRE - - on s'arrête quand on a lu autant de 1 qu'il ya de caractères
SI element(texte,i)='1' ALORS
limit ← limit +1
FIN SI
texte_infixe ← texte_infixe & element(texte,i)
i←i+1

FIN TANT QUE

stocker le reste du texte dans texte_code

R3: Comment "stocker le reste du texte dans texte_code "?

texte_code ← unbounded_slice(texte,in,length(texte)-i)

R2: Comment "Reconstruire la table de Huffman"?

code ← null_unbouned_string
nb_feuille ← 1
texte_caractere_curseur ← texte_caractere

POUR i allant de 1 à longueur(texte_infixe) FAIRE

SI element(texte_infixe,i)='0' ALORS
append(code,to_unbouned_string("0")
SINON
SI nb_feuille:=indice_dollards ALORS
table_huffman(257)<-- code - - cas du caractere de fin
FIN SI

table_huffman(integer(binaire_to_octet(unbounded_slice(texte_caractere_curseur,1,8))<-- code

```
        texte_caractere_curseur ← unbounded_slice(texte_caractere_curseur
    ,9,length(texte_caractere_curseur )) – on retire l'octet dont on vient de trouver le
    code

        nb_feuille ← nb_feuille +1

        code ← unbounded_slice(code,length(code)-1)
        TANT QUE element (code,length(code))='1' FAIRE
            code ← unbounded_slice(code,1,length(code)-1));
        FIN TANT QUE
        replace_element(code,length(code),1))
    FIN SI
```

R2:Comment "decoder le texte"?

```
    i ← 1
    limit ← 1
    fin_texte ← false – est faux tant que l'on ne rencontre pas le code de /$
    nb_element ←1
    TANT QUE non fin_texte FAIRE

        code ← code&element(texte_code,i)

        POUR limit allant de   1 à 257 faire

          SI tablea_huffman(limit) non vide ALORS

            SI code = table_huffman(limit) ALORS

                SI code= table_huffman(257) ALORS

                    fin_texte ← true

                SINON

                tab_element(nb_element):=T_octet(limit)
                nb_element ← nb_element+1
                FIN SI
                code ← to_unbounded_string("")

            FIN SI
          FIN SI
        FIN POUR
    i← i+1

    FIN TANT QUE
```

Écrire les caractères contenues dans tab_element dans l'ordre dans un fichier texte

R3: comment” Écrire les caractères contenues dans tab_element dans l’ordre dans un fichier texte “?

créer un fichier .txt

définir un stream

POUR i allant de 1 à nb_element FAIRE
octet← tab_element(i)
character’write(s,character’val(octet))
FIN POUR
fermer le fichier

		Evaluation (I/P/A/+)
Forme (D-21)	Respect de la syntaxe Ri : Comment "... une action complexe ..." ? des actions combinées avec des structures de contrôle Rj : ...	A
	Verbes à l'infinitif pour les actions complexes	A
	Noms ou équivalent pour expressions complexes	A
	Tous les Ri sont écrits contre la marge et espacés	A
	Les flots de données sont définis	A
	Une seule décision ou répétition par raffinage	A
	Pas trop d'actions dans un raffinage (moins de 5 ou 6)	A
Fond (D21-D22)	Le vocabulaire est précis	+
	Le raffinage d'une action décrit complètement cette action	A
	Le raffinage d'une action ne décrit que cette action	A
	Les flots de données sont cohérents	A
	Pas de structure de contrôle déguisée	A
	Qualité	A

III) Principaux types et fonctions des modules utilisés:

- **module LCA**

1) type:

```
type T_cellule;
  type T_LCA is access T_cellule;
  type T_cellule is record
     cle:T_cle
donnee:T_donnee;
     suivante: T_LCA;
  end record;
```

2) principales fonctions:

```
-- Initialiser une Sda.  La Sda est vide.
procedure Initialiser(Sda: out T_LCA) with
Post => Est_Vide (Sda);


-- Est-ce qu'une Sda est vide ?
function Est_Vide (Sda : T_LCA) return Boolean;

  -- Obtenir le nombre d'éléments d'une Sda.
function Taille (Sda : in T_LCA) return Integer with
Post => Taille'Result >= 0
```

```
and (Taille'Result = 0) = Est_Vide (Sda);



-- Enregistrer une Donnée associée à une Clé dans une Sda.

-- Si la clé est déjà présente dans la Sda, sa donnée est changée.

procedure Enregistrer (lca : in out T_LCA ; Cle : in T_Cle ; Donnee : in T_donnee);



-- Supprimer la Donnée associée à une Clé dans une Sda.

-- Exception : Cle_Absente_Exception si Clé n'est pas utilisée dans la Sda

procedure Supprimer (Sda : in out T_LCA ; Cle : in T_Cle) ;



-- Savoir si une donnée est présente dans une Sda.

function donnee_Presente (Sda : in T_LCA ; donnee : in T_donnee) return Boolean;



-- Obtenir la donnée associée à une Cle dans la Sda.

-- Exception : Cle_Absente_Exception si Clé n'est pas utilisée dans l'Sda

function La_Donnee (Sda : in T_LCA ; cle : in T_cle) return T_donnee;



-- Supprimer tous les éléments d'une Sda.

procedure Vider (Sda : in out T_LCA) with

        Post => Est_Vide (Sda);

   --Retourne la clé d'une sda

   function cle(sda:in T_lca) return T_cle;
```

```
   -- Retourne la donnée d'une sda
   function donnee(sda:in T_lca) return T_donnee;

   -- Retourne la sda suivante d'une sda donnée
   function suivante(sda:in T_lca) return T_lca;


-- Appliquer un traitement (Traiter) pour chaque couple d'une Sda.
generic
with procedure Traiter (Cle : in T_Cle; Donnee: in T_donnee);
   procedure Pour_Chaque (Sda : in T_LCA);


   -- la clé associée à la donnee
   function La_cle (Sda : in T_LCA ; donnee : in T_donnee) return T_cle;
```

- module arbre:

1) **les types:**

```
type T_noeud
type T_arbre is access T_neodu
type T_noeud is record
        cle:integer
        donnee:T_donnee
        gauche:T_arbre
        droit:T_arbre
```

2) les fonctions:

-Initialiser un arbre

```
procedure initialiser (Arbre: out T_arbre) with
        post=>est_vide(arbre);
```

```
-- retourne un booléen qui indique si l'arbre est vide ou non
function est_vide(arbre:in T_arbre) return boolean with
        post=>taille(arbre)=0;
```

```
--Donne la taille de l'arbre
function taille(arbre:in T_arbre) return integer;
```

```
--Indique si un arbre est réduit à une feuille
function est_une_feuille(arbre:in T_arbre) return boolean;
```

```
--La donnee est-elle présente dans l'arbre?
function donnee_presente1(arbre:in T_arbre; donnee:in T_donnee) return
boolean;
```

```
--vide l'arbre
procedure vider(arbre: in out T_arbre) with
        post=>est_vide(arbre);
```

```
-- Donne la'bre droit d'un arbre de type T_arbre
```

```
   function droit(arbre:in T_arbre) return T_arbre ;

--Donne l'arbre gauche d'un arbre de type T_arbre
   function gauche(arbre:in T_arbre) return T_arbre;

--Fusionne deux arbres selon le principe de hufman, dans l'arbred:
   procedure fusion(arbred:in out T_arbre; arbreg:in T_arbre;donnee_defaut:in
T_donnee);

--retourne le code associé à la donnée dans l'arbre de huffman
    function code_associe(arbre: in T_arbre; caractere: in T_donnee) return
Unbounded_String
          with pre=> not est_vide(arbre) and donnee_presente1(arbre,caractere);

--Donne la cle de la racine de l'arbre:
    function cle(arbre:in T_arbre) return integer with
          pre=> not est_vide(arbre);

--Donne la donnée de la racine de l'arbre:
    function donnee(arbre:in T_arbre) return T_donnee with
          pre=> not est_vide(arbre);

 --Transforme une cle et une donnée en une étiquette de donnée : donnée et de
clé:clé
    function construction_feuille(cle:integer;donnee:T_donnee) return T_arbre ;

 -- Indique si une clé est présente dans l'arbre:
    function cle_presente(arbre:in T_arbre;cle:integer) return boolean;
```

- - incrémente la fréquence de la racine d’un arbre

```
procedure incrementer_frequence(arbre:in out T_arbre);
```

p