PROJET PIM : IJ09
Thierry Xu
Tom Bonetto

Compression

-- nom: Compression
-- sémantique: Compresser le fichier texte à l'aide du codage Huffman
-- paramètres: fichier texte
-- pré:
-- post: fichier codé
R0 : Compresser un fichier texte en utilisant le codage de Huffman

R1 : **Comment** « Compresser un fichier texte en utilisant le codage Huffman » **?**
- Récupérer les fréquences des symboles du fichier texte
  Nb_symbole : **out**
  Tab_freq : **in out**

- Construire l'arbre de Huffman
  Arbre : **out**
  Tab_Freq :**in**
- Créer le fichier avec le texte compressé

R2: **Comment** « Récupérer les fréquences des symboles du fichier texte » **?**
- Stocker le texte dans un tableau
  Tab_text : **out**
- Parcourir le tableau du texte et compter la fréquence de chaque caractère stockée dans un autre tableau
  Tab_freq : **out**

R2: **Comment** « Construire l'arbre de Huffman » **?**
- Créer la liste triée des fréquences(type LCA)
  Tab_Freq :**in**
  Sda : **out**

  Tant Que Taille(Sda) /= 1 **Faire**
  -Construire un sous arbre avec les 2 noeuds de fréquence les plus faibles
  Sda : **in**
  Arbre : **out**

  FinTantQue;

R2: **Comment** «Créer le fichier avec le texte compressé»**?**
- Créer la liste des symboles en tant qu'octet par parcours infixe à mettre dans le fichier compressé
  Arbre: **in**
  Tab_freq : **in**
  Tab_octet : **in out**

- Créer la liste des symboles avec leur représentation binaire par parcours infixe
  Arbre : **in**
  Tab_binaire : **in out**

- Créer le fichier compressé

R3: **Comment** « Créer la liste triée des fréquences(type LCA) » **?**

```
TYPE T_Tab_integer EST TABLEAU (1..5000) de Entier

TYPE T_Tab EST ENREGISTREMENT
      Elements : T_Tab_integer;
      Nb_Elements : Integer;
FIN ENREGISTREMENT

TYPE T_Cellule;
TYPE T_LCA EST POINTEUR SUR T_Cellule;
TYPE T_Cellule EST ENREGISTREMENT
      Cle: Entier;
      Donnee: T_Arbre;
      Suivant: T_LCA;
FIN ENREGISTREMENT;

Sda : T_LCA;
Arbre : T_Arbre; -- Même arbre binaire qu'en TD
Tab_Freq : T_Tab;

Initialiser(Sda);
Initialiser(Arbre, 0); -- Initialiser et enregistrer la feuille correspond à "\$"
Enregistrer(Sda, 0, Arbre) ;  -- Fonction enregistrer du mini projet avec relation
d'ordre pour trier automatiquement

Pour i dans 1..256 Faire
      Si Tab_Freq.Elements(i) /= 0 Alors;
            Initialiser(Arbre, Tab_Freq.Elements(i));
            Enregistrer(Sda, Tab_Freq.Elements(i), Arbre);
      FinSi;
FinPour;
```

R3 : **Comment** « Construire un sous arbre avec les 2 nœuds de fréquence les plus faibles » **?**

```
Sous_arbre := Fusion(Premier(Sda), Deuxieme(Sda));
-- La fonction Premier et Deuxieme renvoie respectivement l'arbre de la première et
deuxième cellule de la Sda
Enregistrer(Sda,Frequence(Premier(Sda))+Frequence(Deuxieme(Sda)), Sous_arbre);
Supprimer_Premier(Sda); --Supprime le 1er élément d'une SDA
Supprimer_Premier(Sda);
```

R3: **Comment** «Parcourir le tableau du texte et compter la fréquence de chaque caractère stockée dans un autre tableau » **?**

```
TYPE T_Tab_character EST TABLEAU (1..5000) de Caractère
TYPE T_Tab_symbole EST ENREGISTREMENT
      Elements: T_Tab_character;
      Nb_Elements: Integer;
```

```
FIN ENREGISTREMENT;


Fonction Tab_Frequence(Tab_Texte : in T_Tab_symbole) retourne T_Tab est
      Tab_Freq: T_Tab;
      i: Entier := 1;
Debut

      Pour j dans 1..256 Faire
            Tab_Freq.Elements(j) <-- 0;
      FinPour;
TantQue i /= Tab_Texte.Nb_Elements Faire
      Pour j dans 1..256 Faire
            Si Tab_texte.Elements(i) = Character'Val(j-1) Alors
                  Tab_freq.Elements(j) := Tab_freq.Elements(j) +1;
            FinSi;
      FinPour;
      i := i+1;
FinTantQue;
Fin Tab_Frequence;
```

R3: **Comment** « Créer la liste des symboles en tant qu'octet à mettre dans le fichier compressé »

- Créer la liste des symboles en tant qu'octet par parcours infixe

```
                                                Arbre : in
                                                Tab_freq : in out
                                                Tab_octet : in out
```

- Retirer le "-1", mettre sa position en début de liste et dédoubler le dernier octet

```
                                                Tab_octet : in
                                                Tab_octet_compress : out
```

R3: **Comment** « Créer la liste des symboles avec leur représentation binaire par parcours infixe » ?

```
      Procedure Construire_Liste_binaire(Arbre: in T_Arbre; Tab_binaire: in out
T_Tab_symbole) est
      Debut
          Si non Est_Vide(Arbre_Gauche(Arbre)) Alors
              Tab_binaire.Elements(Tab_binaire.Nb_Elements+1) <-- '0';
              Tab_binaire.Nb_Elements <-- Tab_binaire.Nb_Elements + 1;
              Construire_Liste_binaire(Arbre_Gauche(Arbre), Tab_binaire);

          FinSi;

          Si non Est_Vide(Arbre_Droit(Arbre)) Alors
                  Tab_binaire.Elements(Tab_binaire.Nb_Elements+1) <-- '1';
                  Tab_binaire.Nb_Elements <-- Tab_binaire.Nb_Elements + 1;
                  Construire_Liste_binaire(Arbre_Droit(Arbre), Tab_binaire);
          Finsi;

      Fin Construire_Liste_binaire;
```

R3: **Comment** « Créer le fichier compressé »?

- Construire la table de Huffman

```
                                                    Nb_symbole : in
                                                    Tab_binaire : in
                                                    Tab_octet : in
                                                    Tab_Huff : out
```

- Remplacer les caractères par leur codage selon le codage de Huffman

```
                                                    Tab_Huff : in
                                                    Nb_symbole : in
                                                    Tab_Text : in out
```

- Mettre en octet et écrire les octet dans le fichier

R4: **Comment** « Construire la table de Huffman »?

```
      TYPE T_Char EST ENREGISTREMENT
            char: Entier;
            binaire: T_Pile; -- Défini comme dans les TD avec Capacite => 8 et T_Element
=> Character
      FIN ENREGISTREMENT;
      TYPE T_Tab_Huff_char EST TABLEAU (1..256) de Entier;

      TYPE T_Tab_Huff EST ENREGISTREMENT
            Elements: T_Tab_Huff_char;
            Nb_Elements: Entier;
      FIN ENREGISTREMENT;



      Tab_Huff: T_Tab_Huff;
      i: Integer := 1;
      cpt: Integer := 1;
      Tab_binaire_new: T_Tab_symbole;
      arret: boolean <-- False;

      Tab_binaire_new.Elements <-- Tab_binaire.Elements;
      Tab_binaire_new.Nb_Elements <-- Nb_symbole + 1;  -- Lors du calcul de Nb_symbole,
"\$" n'était pas prise en compte
      Tab_Huff.Nb_Elements <-- Nb_symbole + 1;

      Pour j in 1..Nb_symbole + 1 Faire
            Pile_Caractere.Initialiser(Tab_Huff.Elements(j).binaire);
      FinPour;

      TantQue cpt <= (Nb_symbole + 1) et i <= Tab_binaire.Nb_Elements Faire
            Si Tab_octet.Elements(cpt) = -1 Alors
            Tab_Huff.Elements(cpt).char <-- -1;
            Sinon
            Tab_Huff.Elements(cpt).char <-- Tab_octet.Elements(cpt);

            FinSi;
```

```
            Si Tab_binaire_new.Elements(i) = '0' Alors
            Pile_Caractere.Empiler(Tab_Huff.Elements(cpt).binaire, '0');
            i := i + 1;

            SinonSi Tab_binaire_new.Elements(i) = 'm' Alors
            Pile_Caractere.Empiler(Tab_Huff.Elements(cpt).binaire, '1');
            i <-- i + 1;


            SinonSi Tab_binaire_new.Elements(i) = 'x' Alors
            i <-- i + 1;

            SinonSi Tab_binaire_new.Elements(i) = '1' Alors
            cpt := cpt +1;
            Tab_binaire_new.Elements(i) <-- 'm';  -- 'm' pour garder l'ancien '1' en
mémoire
            TantQue Tab_binaire_new.Elements(i-1) /= '0' Faire
                        Tab_binaire_new.Elements(i-1) <-- 'x';     -- 'x' signifie chiffre
supprimé
                        i <-- i-1;
            FinTantQue;
            Tab_binaire_new.Elements(i-1) <-- 'x';
            i <-- 1;
            FinSi;
      FinPour;
```

R4 : **Comment** «Créer la liste des symboles en tant qu'octet par parcours infixe» **?**

```
      Procedure Construire_Liste_octet (Arbre: in T_Arbre; Tab_freq: in out T_Tab;
Tab_octet: in out T_Tab) est
      bool : boolean <-- False;
      Debut
            Si Est_Feuille(Arbre) Alors
                  -Ajouter le symbole en tant qu'octet dans une liste
                                                      Tab_octet : in out
                                                      Tab_freq : in out
            SinonSi not Est_Vide(Arbre_Gauche(Arbre)) Alors
                  Construire_Liste_octet(Arbre_Gauche(Arbre), Tab_freq, Tab_octet);
                  Construire_Liste_octet(Arbre_Droit(Arbre), Tab_freq, Tab_octet);
            FinSi;
      Fin Construire_Liste_octet;
```

R4: **Comment** « Retirer le "-1", mettre sa position en début de liste et dédoubler le dernier octet » **?**

```
      Tab_octet_compress: T_tab;
      indice: Integer <-- 1;

      Tab_octet_compress <-- Tab_octet;
      TantQue Tab_octet_compress.Elements(indice) /= -1 Faire
```

```
            indice <-- indice + 1;
      FinTantQue;

      Pour i dans 0..indice-2 Faire
            Tab_octet_compress.Elements(indice-i) <--
Tab_octet_compress.Elements(indice-i-1);
      FinPour;

      Tab_octet_compress.Elements(1) <-- indice;
      Tab_octet_compress.Elements(Tab_octet_compress.Nb_Elements+1) <--
Tab_octet_compress.Elements(Tab_octet_compress.Nb_Elements);
      Tab_octet_compress.Nb_Elements <-- Tab_octet_compress.Nb_Elements + 1;
```

R4: **Comment** « Remplacer les caractères par leur codage selon le codage de Huffman »?

```
      courant: Entier;
      pile : T_Pile;
      Tableau : T_Tab_symbole;
      cpt : Entier := 1;

      Tableau.Nb_Elements <-- 0;
      Pile_Caractere.Initialiser(pile);
      Pour i dans 1..Tab_octet_compress.Nb_Elements Faire
            courant <-- Tab_octet_compress.Elements(i);
            Pour j dans 1..8 Faire
            Si (courant mod 2) = 1 Alors
                        Empiler(pile, '1');
            Sinon
                        Empiler(pile, '0');
            FinSi;
            courant <-- courant / 2;
            FinPour;
            TantQue non Est_Vide(pile) Faire
            Tableau.Elements(cpt) <-- Pile_Caractere.Sommet(pile);
            Depiler(pile);
            Tableau.Nb_Elements <-- Tableau.Nb_Elements + 1;
            cpt <-- cpt + 1;
            FinPour;
      FinPour;
```

R4 : **Comment** « Ajouter le symbole en tant qu'octet dans une liste »?

```
      Si Frequence(Arbre) = 0 Alors
            Tab_octet.Elements(Tab_octet.Nb_Elements+1) <-- -1;
            Tab_octet.Nb_Elements <-- Tab_octet.Nb_Elements + 1;
      Sinon
            Pour i dans 1..256 Faire
                  Si Tab_Freq.Elements(i) = Frequence(Arbre) and not bool Alors
                        Tab_octet.Elements(Tab_octet.Nb_Elements+1) <-- i-1;
                        Tab_octet.Nb_Elements <-- Tab_octet.Nb_Elements + 1;
                        Tab_freq.Elements(i) <-- 0;
                        bool <-- True;
```

```
            FinSi;
      FinPour;
FinSi;
```

Décompression

-- nom: Décompression
-- sémantique: Décompresser le fichier compressé à l'aide du codage Huffman
-- paramètres: fichier compressé
-- pré:
-- post: fichier original
R0: Décompresser le fichier compressé à l'aide du codage Huffman

R1: **Comment** « Décompresser le fichier compressé à l'aide du codage Huffman »?

- Récupérer la liste d'octet compressé, la binaire qui correspond au parcours infixe de l'arbre et le texte codé
  Tab_Texte: **out**
  Tab_octet_compress: **out**
  Tab_arbre: **out**
- Reconstruire la liste d'octet avec le "-1" à son indice de départ
  Tab_octet_compress: **in**
  Tab_octet: **out**
- Reconstruire l'arbre de Huffman partir des données du fichier
  Tab_Huff: **in**
  Arbre: **in out**
- Créer un tableau de caractère à partir du tableau du texte codé en remplaçant chaque code par son caractère
  Tab_Texte: **in**
  Arbre: **in**
  Text: **out**

R2: **Comment** « Créer un fichier texte à partir du codage en remplaçant chaque code par son caractère »?

```
compteur_text : Entier <-- 1;
pointeur : T_Arbre;
courant: T_Arbre;
Text : T_Tab_symbole;
arret : booléen <-- False;

Text.Nb_Elements <-- 0;
courant <-- Arbre;
TantQue compteur_text <= Tab_Text.Nb_Elements et non arret Faire
      TantQue non Est_Feuille(courant) Faire
      Si Tab_Text.Elements(compteur_text) = '0' Alors
                  courant <-- Arbre_Gauche(courant);
                  compteur_text <-- compteur_text + 1;
      Sinon
                  courant <-- Arbre_Droit(courant);
                  compteur_text <-- compteur_text + 1;
      FinSi;
      FinPour;
      Si Frequence(courant) = -1 Alors
      arret <-- True;
```

```
            Sinon
            Text.Elements(Text.Nb_Elements+1) <-- Character'Val(Frequence(courant));
            Text.Nb_Elements <-- Text.Nb_Elements + 1;
            FinSi;
            courant <-- Arbre;
      FinTantQue;
```

R2: **Comment** « Reconstruire l'arbre de Huffman à partir des données du fichier »?

- Initialiser l'Arbre

  Arbre: **out**

- Inverser les piles de la table de Huffman

  Tab_Huff: **in out**

- Reconstruire l'Arbre

  Tab_Huff: **in**
  Arbre: **in out**
  compteur: **in out**

R3: **Comment** « Initialiser l'Arbre » **?**

```
      compteur: Entier; <-- 1;
      Initialiser_Arbre(Arbre);
      Enregistrer_Arbre(Arbre, 0); --Le '0' signifie qu'il s'agit d'un noeud
```

R3: **Comment** « Inverser les piles de la table de Huffman » **?**

```
      Pour i dans 1..Nb_Symbole Faire
            Inverser(Tab_Huff.Elements(i).binaire);
      Fin Pour;
```

R3: **Comment** « Reconstruire l'Arbre » **?**

- Construire le sous_arbre gauche

  Tab_Huff: **in**
  Arbre: **in out**
  compteur: **in out**

- Construire le sous_arbre droit

  Tab_Huff: **in**
  Arbre: **in out**
  compteur: **in out**

R3: **Comment** « Construire le sous-arbre gauche » **?**

```
      existence : booléen <-- False; -- variable d'existence de sous-arbre
      arret : booléen <-- False;     -- variable d'arrêt si la feuille est créee
      Tab_Huff_copie : T_Tab_Huff;
      Arbre_gauche : T_Arbre;

      Tab_Huff_copie <-- Tab_Huff;

      Pour i dans 1..Tab_Huff_copie.Nb_Elements Faire
            Si non Est_Vide(Tab_Huff_copie.Elements(i).binaire) Alors
            Si Sommet(Tab_Huff_copie.Elements(i).binaire) = '0'  Alors
                  Depiler(Tab_Huff_copie.Elements(i).binaire);
                        existence <-- True;
```

```
            FinSi;
      FinSi;
FinPour;

Si existence et non arret Alors
      Enregistrer_Gauche(Arbre, 0);
FinSi;

Si Est_Vide(Tab_Huff_copie.Elements(compteur).binaire) Alors
      Si Tab_Huff_copie.Elements(compteur).char = -1 Alors
      Enregistrer_Gauche(Arbre, -1);
            arret <-- True;
            compteur <-- compteur + 1;
      Sinon
            Enregistrer_Gauche(Arbre, Tab_Huff_copie.Elements(compteur).char);
            arret <-- True;
            compteur <-- compteur + 1;
      FinSi;
FinSi;

Si non arret Alors
      Arbre_gauche := Arbre_Gauche(Arbre);
   -  Construire le sous-arbre gauche
                                                      Tab_Huff : in
                                                      Arbre_gauche : in out
                                                      compteur : in out
FinSi;
```

R3: **Comment** « Construire le sous-arbre droit»?

```
      existence : booléen := False;        -- variable d'existence de sous-arbre
      arret : booléen <-- False;     -- variable d'arret si la feuille est créée
      Tab_Huff_copie : T_Tab_Huff;
      Arbre_droit: T_Arbre;

      Tab_Huff_copie <-- Tab_Huff;

      Pour i dans 1..Tab_Huff_copie.Nb_Elements Faire
            Si non Est_Vide(Tab_Huff_copie.Elements(i).binaire) Alors
            Si Sommet(Tab_Huff_copie.Elements(i).binaire) = '1' Alors
                        Depiler(Tab_Huff_copie.Elements(i).binaire);
                        existence <-- True;
            FinSi;
            FinSi;
      FinPour;

      Si existence Alors
            Enregistrer_Droit(Arbre, 0);
      FinSi;

      Si Est_Vide(Tab_Huff_copie.Elements(compteur).binaire) Alors
            Si Tab_Huff_copie.Elements(compteur).char = -1 Alors
            Enregistrer_Droit(Arbre, -1);
```

```
      arret <-- True;
      compteur <-- compteur + 1;
      Sinon
      Enregistrer_Droit(Arbre, Tab_Huff_copie.Elements(compteur).char);
      arret <-- True;
      compteur <-- compteur + 1;
      FinSi;
FinSi;

Si non arret Alors
   -  Construire le sous-arbre droit
                                                    Tab_Huff : in
                                                    Arbre_droit : in out
                                                    compteur : in out
FinSi;
```