Codage de Huffman

Compression

// Module Arbre Huffman

TYPE P_Node EST POINTEUR VERS T_Node

TYPE T_Node EST UN ENREGISTREMENT
    Left : P_Node
    Right : P_Node
    Data : T_data
    id : T_id

TYPE element EST T_Lca DE P_Node
// voir le type lca (clé, donne, suivant)

R0 : Compresser un fichier txt

R1 : Comment “Compresser un fichier txt” ?

- Lire un fichier txt
- Compter le nombre d’occurence de chaque caractère
- Construire l’arbre de Huffman
- Enregistrer le Codage de Huffman

R2 : Comment “Lire un fichier txt” ?

Pour i allant de 1 au nombre de lignes
    ajouter à texte la ligne i

R2 : Compter le nombre d'occurrence de chaque caractère ?

- Pour i allant de 1 à length(texte)
    - Enregistrer Node avec lettre en donnée et id incrémenté de 1
    - Ajouter/remplacer ce Node dans la lca

R2 : Comment construire l’arbre de Huffman ?

Tant que taille(lca) >1 faire
- Chercher les deux plus petites clés
- Créer un noeud dans la lca avec comme fils gauche et fils droit les deux plus petites clés (freq de fg <= freq de fd)
- Supprimer les noeuds ayant ces deux plus petites clés de la lca

Fin Tant que

R2: Comment “Enregistrer le Codage de Huffman”?

- Parcourir l’arbre par la gauche jusqu'à tomber sur une feuille
- Enregistrer le caractère et son codage
- Remonter au noeud à droite précédent et refaire l’opération

R3: Comment “Chercher les deux plus petites clés”?

i1, i2: indice des deux minimums

- Rechercher le premier minimum
- Rechercher le deuxième minimum

R3: Comment “Créer un nœud avec comme fils gauche et fils droit les deux plus petites clés (freq de fg <= freq de fd)”?

- (appel de la fonction create_node) // premier node est le plus petit
- node.id = node1.id + node2.id
- node.left = node1
- node.right = node2

R3: Comment “Supprimer les nœuds ayant ces deux plus petites clés du tableau”?

- Supprimer(lca, i1)
- Supprimer(lca, i2)

R4: Comment “Rechercher le premier minimum”?

```
i_min1 = lca.clé
min(lca, i_min1, 0)

// fonction min(lca, i_min, i_interdit)
si lca = null alors
        retourner i_min
fin si
node_min= La_donnee(lca, i_min)
si lca.donnee.id  < node_min.id et lca.cle \= i_interdit faire
        min(lca.suivant, lca.cle );
fin si
min(lca.suivant, i_min );
```

R4: Comment “Rechercher le deuxième minimum”?

```
if i_min1 = 1 then i_min2 = 2 else i_min2 = 1
min(lca, i_min2, i_min1)
```

Décompresser

Structure du fichier à décompresser d'extension .hff:

Bits regroupés en octets

Ajout d'un symbole marquant la fin du fichier : -1 "\$" fréquence =0

Codage d'Huffman dans leur ordre d'apparition quand on réalise un parcours infixe sauf le symbole de fin de fichier:

- Liste de caractères : caractère de fin de liste en premier + on double le dernier caractère
- Liste de leurs valeurs (ascii) en octets : la position de -1 en premier + on double le dernier caractère

Codage de l'arbre sous la forme d'une suite de 0 et de 1 correspondant au parcours infixe de l'arbre

Remarque: On suppose qu'il y a un retour à la ligne après chacune de ces trois parties
Exemple:

R0: Décompresser un fichier .hff

R1: Comment " Décompresser un fichier .hff"?

nom : out chaine de caractère
nb_arguments: in Entier
bavard: out Booléen

```
Si nb_arguments == 2 alors
        nom <- 2ème argument
        bavard <- Faux
        Si Tester si le fichier passé en ligne de commande est ".hff" alors  - - nom : in
        String
            - Lire le fichier
        Sinon
            - Afficher un message pour signaler que le fichier n'est pas d'extension
              .hff
        FinSi
Sinon
        nom <- 3ème argument
        bavard <- Vrai
        Si Tester si le fichier passé en ligne de commande est ".hff" alors--nom : in
string
            - Lire le fichier
        Sinon
            - Afficher un message pour signaler que le fichier n'est pas d'extension
              .hff
        FinSi
FinSi
```

R2: Comment "Tester si le fichier passé en ligne de commande est ".hff""?

- Parcourir le deuxième argument de la ligne de commande "le nom du fichier" et voir si ces quatres derniers caractères sont égaux à ".hff"

R2: Comment “Lire le fichier”?
c : out caractère
nom : in chaine de caractère
new_file: out chaîne de caractère

- Ouvrir le fichier en mode lecture “nom”
- Ouvrir le fichier en mode écriture “new_file”
- Stocker la première ligne du fichier // text : out Chaîne de caractères
  Tant que c \= ‘[‘ Faire
    - Lire le caractère suivant

  Fin Tant Que
- Remplir la liste avec les caractères correspondants aux valeurs en octects – c : in out
- Lire la troisième ligne du fichier
- Construire l’arbre d’Huffman sans les fréquences
- Fermer les fichiers

R3: Comment “Ouvrir le fichier en mode lecture”?

R3: Comment “Ouvrir le fichier en mode écriture “new_file””?

R3: Comment “Stocker la première ligne du fichier”?

R3: Comment “Lire le caractère suivant”?

R3: Comment “Remplir la liste avec les caractères correspondants aux valeurs en octects”?
pos : in out Entier
pos_end : out Entier
characters : out T_Lca
octets: in out Entier

```
pos <- -1
Tant que c \= ‘]’ Faire
    Si c == ‘,’ alors
        pos <- pos +1
        Si pos == 1 Alors
            - Inserer “\$” à la liste en position nombre
        Sinon
            - Inserer le caractère correspondant à “octets” dans la liste “characters”
        FinSi
    Sinon si pos == 0 alors
        - Transformer une suite caractère en un nombre
    Sinon si pos mod 2 == 0 alors
        - Transformer une suite caractère en un nombre
    FinSi
Fin Tant que
```

R3: Comment “Ouvrir le fichier en mode lecture”?

R4: Comment “Inserer “\$” à la liste en position nombre”?

R4: Comment “Inserer le caractère correspondant à “octets” dans la liste “characters” ”?

R4: Comment “Transformer une suite caractère en un nombre”?