PIM : Projet codage de Huffman

Auteur 1 : Issam Alouane
Auteur 2 : Jean-Baptiste Prevost

Raffinages Compression :

Les raffinages:

R0: Compresser un texte à l'aide du codage de Huffman

R1: Comment *"Compresser un texte à l'aide du codage de Huffman"* ?
- Calculer la fréquence d'apparition des caractères
- Construire l'arbre de Huffman
- Afficher la construction de l'arbre
- Encoder le fichier

R2: Comment *"Calculer la fréquence d'apparition des caractères"* ?
- Parcourir le texte en comptant chaque caractère au fur et à mesure
- Renvoyer la fréquence dans l'ordre croissant du nombre d'apparition des caractères

R2: Comment *"Afficher la construction de l'arbre"* ?
- **Si** Bavard **alors** Bavard : **in** Boolean
  Afficher l'arbre détaillé
  Sinon
  Afficher le code associé aux caractères
  Fin Si

R2: Comment *"Construire l'arbre de Huffman?"* ? ptr_frequence:in out, arbre_Huffman: out
- Associer les deux éléments ayant les plus petites fréquences
- i ← 2 --compteur qui indique la première case où il y a une feuille après les arbres au début
- Translater les cases à gauche
- j ← 1 --le nombre des arbres actuelles
- **Tant que** ptr_frequence(i)^.fils_droit=null **faire**

Si ptr_frequence(1)^.indice >= ptr_frequence(i)^.indice **alors**
i ← 1 Entier
associer la feuillle à un arbre “si possible”
associer ptr_frequence(i) et ptr_frequence(i+1) --si i=j alors on ne peut pas associer la fréquence à aucune arbre
Sinon
Associer ptr_frequence(j-1) et ptr_frequence(j)
Trier l’arbre (ptr_frequence(j-1)) entre les arbres
i ← i-1
Translater les cases à gauche
j ← j-1
Finsi
Fin Tant Que
associer les arbres restantes
arbre_Huffman=ptr_frequence(1)

R2: Comment *“Encoder le fichier”* ?

- Coder l’arbre sous forme de 0 et 1 et créer un tableau qui contient les valeurs des caractères en Octet
- Coder le texte sous forme de 0 et 1
- Retrouver le texte codé

R3: Comment *“Afficher l’arbre détaillé”* ? arbre: in

- **Procedure** affichage_arbre(arbre) **est**
Si arbre.gauche=null et arbre.droite=null **faire**
Ecrire (“(“)
Écrire(arbre^.indice)
Ecrire(“)”)
Écrire(“ “)
Écrire(arbre^.valeur)
Sinon
Ecrire (“(“)
Écrire(arbre^.indice)
Ecrire(“)”)
Nouvelle ligne
Écrire(“\--0--”)
affichage_arbre(arbre^.gauche)
Pour i de 1..Taille(arbre/)/2 **Faire**
Écrire(“|”)
Nouvelle ligne
Fin Pour
Écrire(“\--1--”)
affichage_arbre(arbre^.droite)
Fin Si
Fin affichage_arbre

R3: Comment *“Afficher le code associé aux caractères”* ?

- **Pour** i allant de 1..Taille(lca_code) **Faire**
  Ecrire(lca_code^.Cle)
  Ecrire("-->")
  Ecrire(lca^.Valeur)
  Nouvelle ligne
  lca_code ← lca_code^.Suivant
  Fin Pour

R3: Comment *"Parcourir le texte en comptant chaque caractère au fur et à mesure"* ?

- Initialiser (lca_frequence) lca_frequence (LCA) : **out**
- **Pour** i allant de 1..Length(Texte) **faire**
  Si non Cle_Presente(lca_frequence, Texte(i)) **alors** Texte(i):Caractere
  Enregistrer(lca_frequence, Texte(i), 1)
  Sinon
  Enregistrer(lca_frequence, Texte(i), La_Donnee(lca_frequence, Texte(i))+1)
  Fin Si
  Fin Pour

R3: Comment *"Renvoyer la fréquence dans l'ordre croissant du nombre d'apparition des caractères"* ?

-- Mise en place d'un tri pour facilliter la création de l'arbre d'Huffman

- ptr_frequence **est Tableau** (Taille(lca_frequence)+1) de T_arbre_binaire
- Enregistrer(ptr_frequence(1), '\$', 0) -- Indique la fin de l'arbre
- Trier le tableau dans l'ordre croissant
- Vider(lca_frequence) -- Liberation de la mémoire car allocation dynamique

R3: Comment *"Associer la feuillle à un arbre "si possible""* ?

- **Tant que** I<=j **faire**
  Si ptr_frequence(i)>ptr_frequence(I)^.indice **alors**
  associer ptr_frequence(i) et ptr_frequence(I)
  trier l'arbre (frequence(I)) entre les arbres
  Translater les cases à gauche
  Arrêt
  Sinon
  Rien
  Fin Si
  I ← I+1
  Fin Tant Que

R3: Comment "associer ptr_frequence(i) et ptr_frequence(i+1)"?

- **Si** I=j **alors**
  Associer ptr_frequence(i) et ptr_frequence(i+1) --c'est de la même manière qu'on va associer les deux premiers éléments
  Trier l'arbre (ptr_frequence(i)) entre les arbres
  i ← i+1
  j ← j+1

Translater les cases à gauche
Sinon
Rien
Finsi

R3: Comment “associer les arbres restantes”?

- **Tant que** j>1 **faire**
  Associer ptr_frequence(1) et ptr_frequence(2)
  Trier l’arbre (ptr_frequence(1)) entre les arbres
  Translater les cases à gauche
  j ← j-1
  Fin Tant Que

R3: Comment *“Coder l’arbre sous forme de 0 et 1 et créer un lca qui contient les valeurs des caractères en Octet”* ?

- type T_Tab **est tableau**(1..taille(ptr_frequence)+1) d’entier
- type T_Tab_caractere **est enregistrement**
- taille: entier
- Tab: T_Tab
- Tab_caractere: T_Tab_caracter
- Tab_caracter.taille←1 -- On suppose qu’il va exister au moins un élément (\$)
- Initialiser(lca_code) lca_code (LCA) : **out** -- Procedure module lca
- Parcours_Huffman : Chaine de caractere
- Parcours_caractere : Chaine de caractere
- i←1 i : Entier **out**
- j←1 i : Entier **out**
- **Procedure** Parcours_infixe(arbre) **est**
  Si arbre^.Gauche /= null **alors**
  Parcours_infixe(arbre^.Gauche)
  Parcours_Huffman←Parcours_Huffman & “0”
  Parcours_caractere←Parcours_caractere & “0”
  Sinon
  Enregistrer(lca_code,arbre^.Valeur,Parcours_caractere)
  Parcours_caractere←Parcours_caractere(1,
  Length(Parcours_caractere)-1)
  i←i+1
  Fin Si
  Si arbre^.Valeur=’\$’ **faire**
  Tab_caractere.Tab(1)←j
  Sinon
  Tab_caractere.Tab(Tab_caractere.taille+1)←
character’Pos(arbre^.Valeur)
  j←j+1
  Fin Si
  Tab_caractere.taille←Tab_caractere.taille+1
  Si arbre^.Droite /= null **alors**
  Parcours_infixe(arbre^.Droite)
  Parcours_huffman←parcours_hufman & “1”

Parcours_caractere←Parcours_caractere & “1”
Sinon
Enregistrer(lca_code,arbre^.Valeur,Parcours_caractere)
Parcours_caractere←Parcours_caractere(1, Length(Parcours_caractere)-1)
i←i+1
Fin Si
Fin Procedure Parcours_infixe

- Tab_caractere.Tab(taille(ptr_frequence)+1)←Tab_caractere.Tab(taille(ptr_frequence)) -- On répète le dernier caractère

R3: Comment *“Coder le texte sous forme de 0 et 1”* ? texte: in, texte_code: in out

- texte_code : Chaine de caractère
- j , k : entier
- k←1
- **Pour** i dans 1..taille(texte) **faire**
  j←1
  Tant que j<Taille(Donnee(lca_code, texte(i))) **faire**
  texte_code←texte_code & Donnee(lca_code, texte(i))(j)
  k←k+1
  Si k%9=0 **faire**
  texte_code←texte_code & “.”
  Sinon
  Rien
  FinSi
  j←j+1
  Fin Tant Que
  Pour i dans 1..(9-k%9) **faire**
  texte_code←texte_code & “0”
  Fin Pour

Fin Pour

R3: Comment *“Retrouver le texte codé”* ? texte_code: **in**, texte_code_Huffman: **in out**

- i← 1 Entier
- texte_code_Huffman: Chaine de caractère
- **Tant que** i<=Taille(texte_code) **faire**
  texte_code_Huffman← texte_code_Huffman& character’Pos(Horner(texte(i, i+7))) --on applique le schéma de horner
  i← i+9

Fin Tant Que

R3: Comment *“Associer les deux éléments ayant les plus petites fréquences”* ?

- ptr_frequence(1)← new T_cellule'(ptr_frequence(1).Indice+ptr_frequence(2).Indice, null, ptr_frequence(2) --droit, ptr_frequence(1) --gauche)

R4: Comment *“Associer ptr_frequence(i) et ptr_frequence(l)”* ?

- ptr_frequence(l)← new T_cellule'(arbre^.Indice+ptr_frequence(i).Indice, null, ptr_frequence(i) --droit, ptr_frequence(l) --gauche)

Pour k dans i..Taille(ptr_frequence)-1 **faire**
ptr_frequence(k) ← ptr_frequence(k+1)
Fin Pour

R4: Comment *"Associer ptr_frequence(1) et ptr_frequence(2)"* ?
--pour ne pas trop alléger les écritures on remplace ptr_frequence(1) par V1 et ptr_frequence(2) par V2

- V1← new T_cellule'(V1^.Indice +V2 ^.Indice, null, V1 --droit, V2 --gauche)
- Trier arbre entre les arbres

R4: Comment *"Translater les cases à gauche"* ?

- **Pour** k dans i..Taille(ptr_frequence)-1 **faire**
  ptr_frequence(k) ← ptr_frequence(k+1)
  Fin Pour

R4: Comment *"Translater les cases à gauche"* ? --dans la dernière boucle pour

- **Pour** k dans 2..j **faire**
  ptr_frequence(k) ← ptr_frequence(k+1)
  Fin Pour

R4: Comment *"Trier l'arbre entre les arbres"* ? --J'ai écris pour ptr_frequence(l) mais c'est la même procédure pour tous les cas

- k← l
- **Tant que** 1<=k et ptr_frequence(k)>ptr_frequence(k-1) **faire**
- arbre_copie ← ptr_frequence(k)
- ptr_frequence(k) ← ptr_frequence(k-1)
- ptr_frequence(k-1) ← arbre_copie
- k← k-1
- **FinTant Que**

R4: Comment *"Appliquer le schéma de horner"* ?

- valeur: Entier← integer'Value(texte(i.. i+7))
- Resultat: Entier← texte(i+7)
- **Pour** k dans 1..7 **faire**
  Resultat ← 2 * texte(i+7-k) + Resultat
  Fin pour

R4: Comment *"Trier le tableau dans l'ordre croissant"* ?

- **Pour** i allant de 2..Taille (lca_frequence) **faire** lca_frequence : **in out** lca
  Cle_test ← lca_frequence^.Cle -- Initialisation d'une clé de test pour pouvoir comparer
  Pour j allant de 1..Taille (lca_frequence) **faire**
  Si La_Donnee(lca_frequence, Cle_test) <= lca_frequence^.Valeur **alors**
  Cle_test ← lca_frequence^.Cle
  Sinon
  Rien

Fin Si
Fin Pour
Enregistrer(ptr_frequence(i),Cle_test , La_Donnee(lca_frequence, Cle_test)) -- Ici la procedure Enregistrer viens du module de l'arbre binaire
Supprimer(lca_frequence, Cle_test) -- Procedure du module lca
Fin Pour

Raffinages modules utilisés (vu en TP et/ou en TD):

R0: Faire une Liste de Donnée Associative (LCA)

R0: Faire un arbre de recherche binaire

Evaluation par les étudiants

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

Raffinages Décompression

Les raffinages:

R0: Décompresser un texte à l'aide du codage de Huffman

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

- Détecter que le fichier donné ait la bonne extension pour pouvoir le traiter
- Transformer le texte en 0 et 1
- Récupérer le code de chaque caractère
- Récupérer le texte original

R2: Comment *"Détecter que le fichier ait la bonne extension pour pouvoir le traiter"* ?

- Récupérer les quatre derniers caractères du nom du fichier donné
- Comparer les derniers caractères du nom du fichier a l'extension attendu

R2: Comment "Transformer le texte en 0 et 1*"* ? texte: in, texte_original: in out

- texte_original: Chaine de caractère
- i← 1 Entier
- **Pour** i dans 1..Length(texte) **faire**
  texte_original ← texte_original & Binaire(Character'Val(texte(i, i+7))) & "." --on transforme le nombre en binaire
  Fin Pour

R2: Comment *"Récupérer le code de chaque caractère"* ? Parcours_Huffman: in, lca_code: in out

- Code_caractere: Chaine de caractère
- Initialiser(lca_code) -- Procedure du module lca
- j: Entier --un paramètre qui décrit de combien de pas on doit retourner quand on avance, càd ce paramètre nous indique qu'il reste un fils droit pour le parcourir
- i← 1 Entier
- **Tant que** i<=Length(Parcours_Huffman) **faire**
  Code_caractere ← Code_caractere & Parcours_Huffman(i)
  Si Parcours_Huffman(i+1)='1' et Parcours_Huffman(i)='0' **faire**
  j← 1
  Sinon
  Rien
  Fin Si
  Si Parcours_Huffman(i+1)='1' **faire**
  Si i=Tab_caractere(1) **faire**
  Enregistrer(lca_code, Code_caractere, "\$")
  Sinon
  Enregistrer(lca_code, Code_caractere, character'Val(Tab_caractere(j+1)))
  Fin Si
  Code_caractere← Code_caractere(1,Length(Code_caractere)-j)

```
            j← j+1
        Sinon
            Rien
        Fin Si
        i← i+1
    Fin Tant Que
```

R2: Comment *"Récuperer le texte original"* ?

- indice ← 1
- **Tant Que** Texte_original(-1) /= "\$" **faire** Texte_original : **in out** Chaine de caractere
  Code_bin ← Code_bin & Texte_encode(indice) Code_bin : **in out** Chaine de caractère
  Pour i allant de 1..Taille(lca_code) **faire**
  Si Cle_presente(lca_code, Code_bin) **faire**
  Texte_original ← Texte_original & La_Donnee(lca_code,Code_bin)
  Code_bin ← ""
  Sinon
  Rien
  Fin Si
  indice ← indice +1
  Fin Pour
  Fin Tant Que
- Générer le fichier décompressé

R3: Comment *"Génerer le fichier décompressé"* ?

- Créer(nom_decompressé.txt) -- Création du fichier
- Modifier(nom_decompressé.txt, Texte_original) -- Ecriture du résultat
- Ecrire("La décompression est achevée")

R3: Comment *"Récuperer les quatre derniers caractères du nom du fichier donné"* ?

- **Pour** i allant de 1..Longueur_nom **faire** Longueur_nom : **in** Entier
  Si i >= Longueur_nom-4 **alors**
  Extension ← Extension & Nom(i) Extension: **out** Chaîne de caractère
  Sinon
  Rien
  Fin Si
  Fin Pour

R3: Comment *"Comparer les quatre derniers caractères du nom du fichier a l'extension attendu"* ?

- **Si** Extension /= ".hff" **alors**
  Levée Exception_extension_incompatible -- Évite les erreus qui peuvent survenir plus tard dans le programme
  Sinon
  Rien

Fin Si

R3: Comment “Transformer les nombre en binaire”? N: **in out**

- Binaire: unbounded_string
- N: entier
- **Tant que** N /=1 **faire**
  Si N%2=1 **faire**
  Binaire ← Binaire & “1”
  Sinon
  Binaire ← Binaire & “0”
  Fin Si
  N ← N//2 --le quotient de la division entière de N par 2
  Fin Tant que

Evaluation par les étudiants

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

Répartition des rôles:

- Issam Alouane
    - Construction de l'arbre d'Huffman
    - Encodage du fichier
    - Affichage de l'arbre
    - Récupération des codes binaires associé aux caractères encodé

- Jean-Baptiste Prevost
    - Calcul fréquence d'apparition des caractères
    - Affichage de l'arbre
    - Encodage du fichier
    - Vérification de l'extension
    - Décodage du fichier a partir du code binaire asocié récupéré
    - Création du fichier décodé