INFO-F103 : Projet 2

Algorithmique 1

Année académique 2019–2020

Introduction

Le codage de Huffman est une méthode de compression de données inventée par David Huffman dans les années 1950. Il propose un codage de longueur variable qui encode les caractères d'un fichier selon leur probabilité d'apparition : plus un caractère apparaît dans le contenu à compresser, plus la chaîne de bits utilisée pour l'encoder est courte.

Par exemple, pour compresser le syllabus d'algorithmique 1, nous pouvons d'abord compter la fréquence de chaque caractère. Puis, à l'aide d'un vecteur contenant ces fréquences, nous pouvons construire l'arbre de Huffman (qui sera expliqué plus loin) et, alors, assigner une chaîne de bits à chaque caractère. Ensuite, nous pouvons procéder à la compression en remplaçant chaque caractère du syllabus par son codage en bits. Le résultat de ce processus est un codage de longueur minimum en bits du syllabus.

Construction de l'arbre de Huffman

L'arbre de Huffman est un arbre binaire dont les feuilles sont associées à un symbole. Celui-ci permet de construire la chaîne de bits associée à chaque symbole en suivant le chemin pris pour atteindre la feuille lui correspondant depuis la racine de l'arbre.

Pour construire un arbre de Huffman :

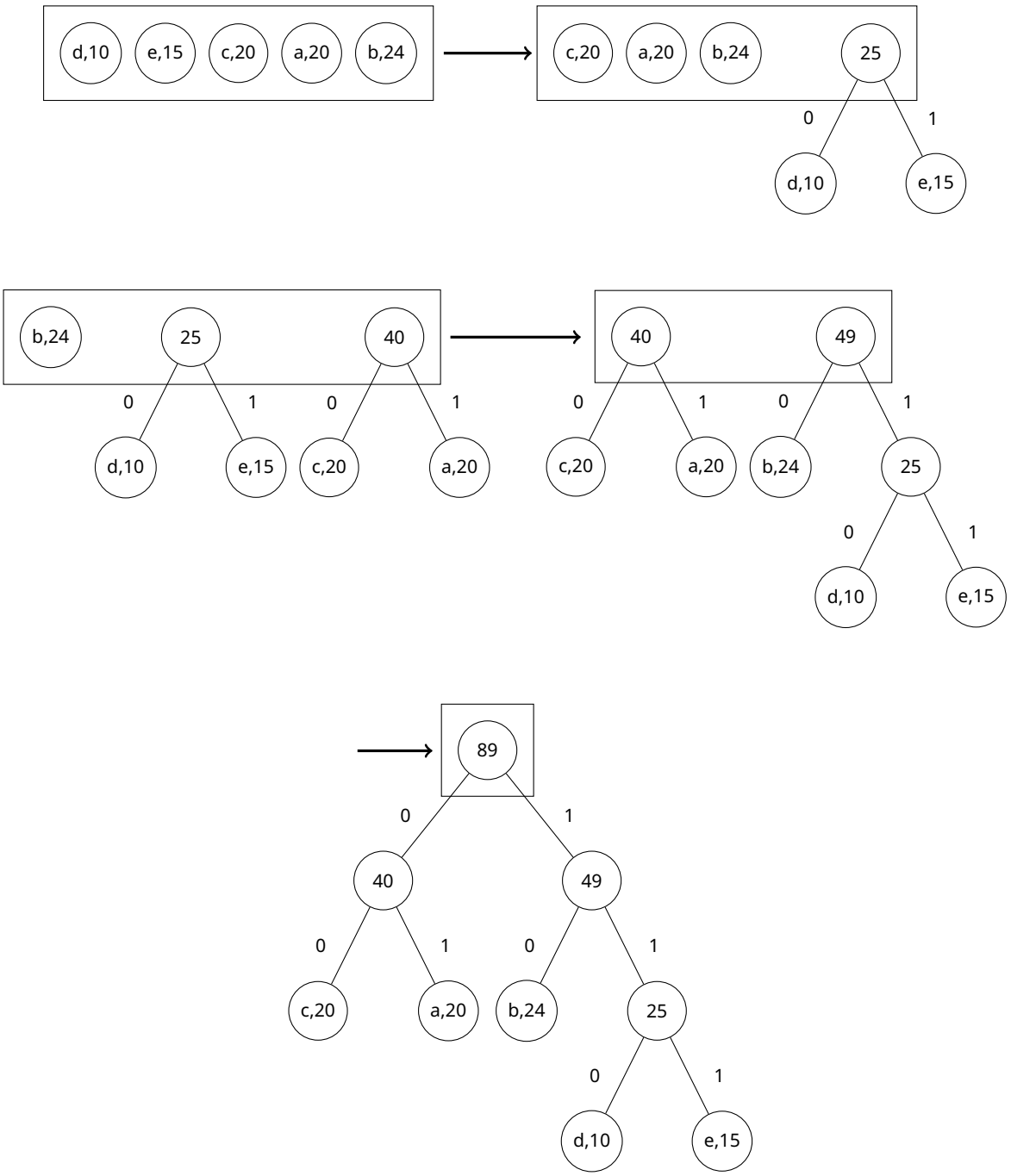
- Avant toute chose, il faut connaître la probabilité d'apparition de chaque symbole constituant le document à compresser. Dans notre cas, la fréquence d'apparition de chaque symbole est suffisante. Pour cela, il faut faire une première passe sur le document pour calculer la fréquence d'apparition de chaque symbole et stocker cette information dans une liste permettant d'associer un symbole avec la fréquence calculée.

`a,20`	`b,24`	`c,20`	`d,10`	`e,15`

- Il faut ensuite trier cette liste par ordre croissant sur base des fréquences.

`d,10`	`e,15`	`c,20`	`a,20`	`b,24`

- Il faut alors considérer tous les éléments de la liste comme des feuilles. L'algorithme consiste alors à connecter les arbres se trouvant dans la liste pour former un seul arbre binaire. À chaque itération, l'algorithme crée un nouveau nœud et lui assigne les deux arbres ayant les fréquences (valeur se trouvant dans la racine) les plus faibles en tant que nœuds enfants. La somme de la fréquence des deux nœuds enfants est assignée au nœud père. Cette étape est répétée jusqu'à ce qu'il n'y ait plus qu'un seul arbre.



- Nous obtenons alors l'arbre de Huffman représentant le codage des différents caractères.

Dans notre exemple le caractère 'c' sera codé par 00 ; 'a' par 01 ; 'b' par 10 ; 'd' par 110 ; 'e' par 111. Observation : 'b' est codé par une suite de bits plus courte que 'd' car il est plus fréquent (avec une plus haute probabilité d'apparition).

Compression & décompression

Une fois que l'arbre est construit, il suffit de parcourir l'arbre pour construire la suite de bits associée à chaque symbole. Le code correspond au chemin permettant d'accéder à la feuille qui lui est associée, depuis la racine, en notant 0 si il passe par la branche de gauche et 1 si on passe par la branche de droite. Le fait d'avoir généré un code en se servant d'un arbre binaire assure qu'aucun code ne peut être le préfixe d'un autre. Le préfixe de longueur *n* d'un code correspond à la suite constituée par les

n premiers bits de ce dernier. Par exemple, dans notre cas, le code de la lettre 'c' (00) ne se retrouve jamais dans le préfixe de longueur 2 des autres codes. Tout comme, le préfixe de longueur 2 de 'd' et 'e' (11) n'est pas le code d'un autre caractère.

Le codage se réalise facilement à l'aide d'une table associant à chaque symbole son code.

symbole	code
c	00
a	01
b	10
d	110
e	111

Le décodage consiste à utiliser les bits du code un par un, dans l'ordre, pour descendre dans l'arbre à droite ou à gauche. Dès qu'une feuille est atteinte, le caractère de cette feuille est ajouté à la chaîne de caractères de sortie. Le parcours de l'arbre redémarre de la racine pour la séquence suivante.

Énoncé

Pour ce projet nous vous demandons :

1. La classe **HuffmanTree** contenant une fréquence et pouvant contenir une lettre et deux nœuds enfants.
2. Implémenter une fonction **get_frequences(text)** qui retourne une liste de tuples (un par caractère) ayant comme premier élément le caractère et en second sa fréquence. Cette liste doit être triée comme expliqué ci-dessus.
3. Implémenter une fonction **build_huffman_tree(freq_list)** acceptant la liste de fréquences et retournant un objet étant une instance de la classe HuffmanTree.
4. Implémenter la fonction **compress(huffman_tree, text)** qui compresse une chaîne de caractères texte (`str`) en utilisant le codage donné par l'arbre de Huffman huffman_tree et retourne une chaîne binaire sous forme de `bytes`.
5. Implémenter la fonction **decompress(huffman_tree, compressed_text)** qui décompresse une chaîne binaire qui sera sous la forme de `bytes` en la chaîne de caractères texte (`str`) correspondante en parcourant l'arbre de Huffman huffman_tree.
6. Implémenter la fonction **print_huffman_tree(huffman_tree)** Vous devez parcourir et afficher l'arbre sur l'écran. Nous ne fixons pas un format particulier pour cet affichage, mais il faut que ce soit lisible.

Un template du code vous est fourni sous le nom `huffman_coding.py`. Il contient les classes, méthodes et fonctions nécessaires pour la résolution de ce projet. Veillez respecter le nom, la signature et la casse des classes, méthodes et fonctions déjà présentes. Vous pouvez en implémenter d'autres si vous le désirez mais celles qui sont dans le template doivent rester dans votre code. Utilisez les structures et le vocabulaire sur les arbres vus au cours théorique dans vos implémentations et commentaires. Vous ne devez pas gérer les erreurs de paramètres pour les méthodes que vous implémentez. Les variables globales, librairies python (sauf sys et math) et librairies extérieures ne sont pas autorisées.

Deux fonctions sont fournies avec le template, elles permettent de convertir des chaînes de caractères composées de "0" et de "1" en bytes et vice-versa (`convert_bin_string_to_bytes` et `convert_bytes_to_bin_string`). Par exemple, `convert_bin_string_to_bytes("00000011")` retourne `b'\x03'` et `convert_bytes_to_bin_string(b'\x03')` retourne `"00000011"`. Utilisez ces fonctions pour la compression et la décompression. N'hésitez pas à sauver le résultat de la compression dans un fichier binaire pour constater la différence en terme d'espace de stockage.

Consignes pour la remise du projet

À respecter scrupuleusement !

1. Le projet est à remettre sur l'UV uniquement.
2. Un seul fichier compressé (`.zip` ou équivalent) doit être remis sous le format `<votre_matricule>.zip`[a]. Tous les fichiers (théoriquement un seul et un README si nécessaire) doivent se trouver à la racine de cette archive donc, pas de dossier.
3. Votre code doit être ***commenté***.
4. Date limite de remise : Dimanche 29 mars à 23h59.

a. exemple : si mon matricule est 123456, je remets le fichier `123456.zip`

Les projets en retard sont considérés comme non-remis.

Pour toute question concernant ce projet, veuillez-vous adresser à Xavier Barthel [1].

1. e-mail : `<xavier.barthel@ulb.ac.be>`