Algorithmique et Programmation avancée

6: Tp Final

1 Codage de Huffman

Le codage de Huffman est un procédé très utilisé en compression de données. Il sert à encoder un texte en binaire, en utilisant pour chaque lettre un nombre de bits dépendant de la fréquence de la lettre : plus la lettre apparaît, plus le nombre de bits est petit.

Exemple:

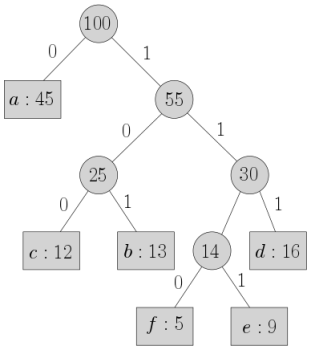
	a	b	c	d	e	f	Total
Fréquence	45	13	12	16	9	5	100
Mot de code de longueur fixe	000	001	010	011	100	101	300
Mot de code de Huffman	0	101	100	111	1101	1100	224

1.1 Code préfixe

Afin de coder et decoder de façon bijective, on utilisera un code dit “préfixe”, i.e., tel qu’aucun mot n’est préfixe d’un autre.

Dans notre exemple, la chaine “001011101” ne peut être interpretée que comme “0-0-101–1101”, ce qui donne “aabe”.

Pour construire un tel code, on utilise un arbre binaire complet (les noeuds internes ont 2 fils) dont les feuilles sont les caractères donnés. Chaque mot de code est alors donné par le chemin allant de la racine au caractère, où “0” signifie “fils gauche” et “1” signifie “fils droit”.



Arbre de Huffman

On pourra remarquer que, puisque l’arbre est binaire et complet, pour n feuilles, il dispose de $n - 1$ noeuds internes.

1.2 Construction d’un arbre de Huffman

Le codage de Huffman est un algorithme qui construit un codage préfixe optimal.

Partant d'un ensemble C de n caractères dont chaque caractère c a un attribut $c.freq$ donnant sa fréquence, l'algorithme construit du bas vers le haut l'arbre T correspondant au codage optimal. Il commence par un ensemble de n feuilles et effectue une série de $n-1$ "fusions" pour créer T. Les fusions se font entre 2 noeuds de fréquences minimales. Le résultat de la fusion de deux noeuds est un nouveau noeud qui leur est père et dont la fréquence est la somme des fréquences de ses fils. A la fin, l'algorithme renvoie la racine de l'arbre T.

Théoreme 1 (Correction de l'algorithme)**.** *Le codage de Huffman est optimal.*

1.3 Exercices

1. Représentez les étapes successives lors de l'exécution de l'algorithme, depuis l'entrée (caractères/fréquences) donnée en exemple jusqu'à l'arbe T.

2. Proposer un pseudo-code de l'algorithme dont vous préciserez le paradigme utilisé et calculerez la complexité.

   (Il peut être utile de considérer l'opération "Créer un noeud z", noeud auquel sera attribué des fils gauche et droit ainsi qu'une fréquence. Possible de faire $O(nln(n))$ avec ce que l'on a vu).

3. Coder l'algorithme de Huffman en Python3. Le code prendra en entrée une phrase, comptera la fréquence de chaque caractère, appliquera l'algorithme de Huffman, affichera la phrase codée, et la décodera. On pourra afficher divers éléments intermédiaires.

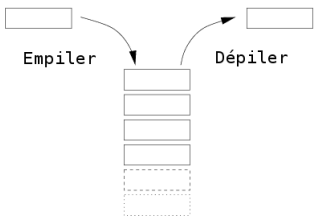
2 Enveloppe convexe par balayage de Graham

En géométrie planaire, l'enveloppe convexe d'un ensemble Q de points, notée $EC(Q)$ est le plus petit polygone convexe P tel que chaque point de Q est, soit sur le contour de P, soit à l'intérieur.

2.1 Une structure de donnée utile et simple: les piles

La pile (stack) est une structure de données fondée sur le principe " dernier arrivé, premier sorti". Elle gère un ensemble de données avec les primitives suivantes:

- **Empiler** : ajoute (push) un élément sur la pile.
- **Dépiler** : enlève (pop) un élément de la pile et le renvoie.



2.2 Le balayage de Graham

Le balayage de Graham résout le problème de l'enveloppe convexe en gérant un pile S de points candidats. On désigne par s_0 et s_1 les premier et deuxième éléments de la pile. Chaque point de l'ensemble Q est empilé une fois, puis les points qui ne sont pas des sommets de $EC(Q)$ finissent par être tous dépilés. Quand l'algorithme se termine, la pile S contient exactement les sommets de $EC(Q)$ dans l'ordre trigonométrique de leur apparition sur le contour.

L'algorithme prend en entrée un ensemble Q de plus de 3 points.

Il commence par choisir un sommet du contour p_0, par exemple le point le plus bas; trie les points par rapport à leur angle polaire relativement à p_0, puis empile les 3 premiers sommets.

Ensuite, pour chaque point p_i restant de la liste triée Q, on dépile S tant que l'angle $(s_0\vec{s}_1, s_0\vec{p}_i)$ est inférieur à π, après quoi on empile p_i sur S.

Théoreme 2 (Correction de l'algorithme)**.** *La pile S obtenue en effectuant le balayage de Graham d'un ensemble Q de points contient exactement les sommets de $EC(Q)$ pris dans l'ordre trigonométrique.*

2.3 Exercices

On supposera que les points sont en position générique (il n'existe pas de triplet de points alignés).

1. Représentez les étapes successives de l'exécution de l'algorithme pour une entrée de quelques points (disons 7) judicieusement choisis.
2. Proposer un pseudo-code de l'algorithme dont vous calculerez la complexité.
3. Coder l'algorithme du balayage de Graham en Python3. L'entrée pourra être un ensemble de points aléatoires. Vous pourrez utiliser le module matplotlib pour l'interface graphique, voire afficher une construction animée de l'enveloppe convexe.

A rendre pour Mercredi 23:
DM: Répondre aux questions 1 et 2 de chaque section.

A rendre pour Vendredi 25:
TP: Envoyer les codes Python3.

Remarques: Merci d'apporter du soin à votre code, à savoir:

Expliquer les grandes lignes de votre programme en début de code.
Sous chaque fonction, indiquer les entrées et les sorties, ainsi que le fonctionnement et le rôle de la fonction si nécessaire.
Les rôles, attributs et méthodes des classes utilisées.
Divers autres commentaires là où cela est nécessaire, ne pas surcharger non plus.
Soigner l'affichage du problème et de ses résultats.
Envoyer vos codes même s'ils sont incomplets.
Tout code "douteux" ne sera pas corrigé.