EXAMEN — COURS D'ALGORITHMIQUE — I.S.I.A. 1995-1996

Préliminaires: Ce sujet comporte trois parties indépendantes. Une attention particulière sera consacrée à la rédaction concise et précise de vos solutions. La durée totale du sujet devrait demander une charge de travail de 8 heures. Les questions plus difficiles ont une astérique ($*$). Travaillez seul!

Présentation des 3 problèmes: Afin d'illustrer le cours d'algorithmique, on propose les problèmes suivants:

Arbre de poids minimal. Étant donné un graphe $\mathcal{G}$ valué, on cherche à construire un sous-graphe $\mathcal{T}$ qui soit un arbre recouvrant dont la somme des arêtes le définisant soit minimale.

Code de Huffman. A partir d'un tableau de fréquence d'apparition des lettres d'un alphabet, on désire batir un code binaire pour chaque lettre de façon à comprimer un texte écrit avec cet alphabet. Le code de Huffman est une façon optimale de coder un texte.

Calcul du k^{e} élément. On considére un ensemble de n éléments, on désire calculer le k^{e} plus grand.

Arbre de poids minimal (6 points)

On considére un graphe $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ non-orienté connexe de n sommets et m arêtes valué par des poids positifs. Soit $c(\cdot)$ la fonction de valuation: chaque arête $a \in \mathcal{A}$ a un poids $c(a)$. Un arbre de poids minimal est un sous-graphe $\mathcal{T}$ de $\mathcal{G}$ qui est un arbre minimisant la somme des valuations de ses arêtes. La figure 1 montre un graphe et un arbre recouvrant de poids minimal.

Question 1 *Donner un graphe simple montrant la non unicité de l'arbre recouvrant de poids minimal (*minimum spanning tree*). En supposant les valuations distinctes, montrer que l'arbre recouvrant de poids minimal est unique.*

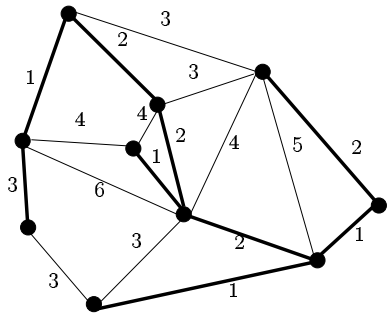


FIG. 1 - *Un graphe $\mathcal{G}$ et un arbre recouvrant de poids minimal $\mathcal{T}$ en gras.*

On se propose d'étudier deux algorithmes qui calculent un arbre recouvrant de poids minimal: l'algorithme de Prim et celui de Kruskal.

Algorithme de Prim.
Cet algorithme ressemble beaucoup à l'algorithme de Dijkstra pour les plus courts chemins: c'est un algorithme glouton. A chaque étape on choisit un minimum local selon des critères qui assurent le minimum global.

- Au départ $\mathcal{T}$ est le graphe vide et $A = \emptyset$.
- On choisit au hasard un sommet $s \in \mathcal{S}$.
- On ajoute s à A.
- On note B l'ensemble des sommets de $\mathcal{S}$ reliés à un sommet de A dans $\mathcal{G} \backslash A$ (cocycle).
- On ajoute une arête $(x, y) \in \mathcal{A}$ telle que $x \in A$ et $y \in B$ de poids minimal. (x, y) est ajouté à $\mathcal{T}$.
- On met a jour jusqu'à $A = \mathcal{S}$.

Question 2 *Donner une implantation de cet algorithme en définissant vos conventions et vos structures de données.*

Question 3 *Illustrer le fonctionnement de l'algorithme sur le graphe de la figure 1.*

Question 4 $(*)$ *Donner une preuve de la correction totale de l'algorithme de Prim. On pourra montrer le lemme suivant:*
Lemme: *Soit (x, y) une arête de poids minimal telle que $x \in A$ et $y \in B$ alors il existe un arbre recouvrant de poids minimal contenant (x, y).*

Question 5 *Étudier la complexité de l'algorithme de Prim en considérant la matrice d'adjacence et la liste d'adjacence comme structures de données pour le graphe $\mathcal{G}$.*

Algorithme de Kruskal.
L'algorithme est basé sur le fait que $\mathcal{T}$ est un graphe sans cycle. L'algorithme est itératif. A une étape de l'algorithme, on choisit une arête de poids minimal $a = (x, y)$ de $\mathcal{G}$ tel que $\mathcal{T} \cup \{a\}$ soit sans cycle. Il faut donc pouvoir détecter rapidement si l'ajout d'une arête dans un graphe créé un cycle ou pas. Autrement dit, existe-t-il déjà un chemin de x à y dans $\mathcal{T}$, ou x et y sont ils dans la même composante connexe. Initialement chaque sommet est dans une composante connexe distincte.

Question 6 *Décrire l'algorithme et illustrer son fonctionnement sur la figure 1.*

Question 7 (∗) *Prouver la correction totale de l'algorithme de Kruskal et analyser la complexité de cet algorithme.*

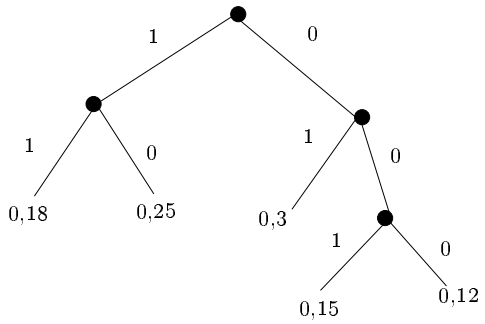
Question 8 *Comparer l'algorithme de Prim avec celui de Kruskal? Lequel est meilleur? Lequel est plus difficile à implanter?*

Codage de Huffman (6 points)

On considére un alphabet Σ de n lettres $a_1, ..., a_n$. A chaque lettre a_i est associée la probabilité p_i d'apparition de la lettre dans le langage ($\sum_{i=1}^{n} p_i = 1$). On désire construire un arbre binaire où les lettres occupent les feuilles de telle sorte que la distance des lettres à la racine, pondérée par la probabilité d'apparition des lettres, soit minimisée. Par exemple on considére l'alphabet suivant: $\Sigma := [A, B, C, D, E]$. Les probabilités d'apparition des lettes dans un texte donné sont:

A	$0,15$
B	$0,3$
C	$0,25$
D	$0,12$
E	$0,18$

L'arbre optimal sera:



La distance pondérée est

$$2 \times 0,18 + 2 \times 0,25 + 2 \times 0,3 + 3 \times 0,15 + 3 \times 0,12 = 2,27.$$

On associe à chaque feuille de l'arbre ainsi créé un mot binaire défini depuis la racine par son cheminement. On étiquette le fils gauche d'un nœud par 1 et le fils droit par 0. Ainsi, nous obtenons les codes suivants:

A	001
B	01
C	10
D	000
E	11

Question 1 *Montrer comment utiliser le code de Huffman pour coder/décoder le texte suivant: "ADCBAA".*

On construit l'arbre binaire grâce à un tas. Informellement, on souhaite que les lettres ayant une petite probabilité se trouvent en profondeur tandis que celles qui arrivent fréquemment soient peu profondes.
Au départ, chaque lettre constitue un arbre dont la racine est la probabilité d'apparition de cette lettre. A une étape donnée, on choisit deux arbres dont les probabilités de la racine sont les plus petites et on créé un arbre ayant comme sous-arbres directs ces deux arbres. On associe à la racine la somme des probabilités des racines des deux sous-arbres.

Question 2 *Illustrer cet algorithme sur l'exemple donné ci-dessus.*

Question 3 *Écrire l'algorithme correspondant. On pourra choisir un tas comme structure de données des arbres. Quelle est la complexité de cet algorithme?*

Question 4 (∗) *Montrer que le code de Huffman est le code optimal, c'est-à-dire celui qui minimise la longueur du texte codé (et non pas compressé!)?*

Calcul de k^{e} plus grand élément (8 points)

Soit $\mathcal{S}$ une liste non triée de n éléments. Les éléments sont totalement ordonnables. On désire calculer le k^{e} plus grand élément de $\mathcal{S}$. Dans un premier temps, on supposera que les éléments de $\mathcal{S}$ distincts.

Question 1 *Donner un algorithme qui calcule le 2^e plus grand élément et étudier sa complexité dans le meilleur des cas, en moyenne et dans le cas le pire. Généraliser sous forme d'automate votre algorithme.*

Question 2 *En prenant exemple sur le tri rapide (ou tri de Hoare), donner un algorithme récursif permettant de calculer le k^e plus grand élément. Soit le tableau suivant d'entiers:*

$$T := [0, 2, 8, 4, 1, 9, 3, 7, 5, 6],$$

donner les appels récursifs de votre algorithme sur T.

On propose la méthodologie suivante (appelé bisection-décimation):

- Grouper les élements de $\mathcal{S}$ en p paquets $\mathcal{S}_1, ..., \mathcal{S}_p$ de tailles identiques ($\lfloor \frac{n}{p} \rfloor \leq \mathcal{S}_i \leq \lceil \frac{n}{p} \rceil$).
- Pour chaque paquet $\mathcal{S}_i$ calculer la médiane m_i (le $\lfloor \frac{n}{2p} \rfloor^{\text{e}}$ élément) en triant le paquet $\mathcal{S}_i$.
- On calcule la médiane M des médianes récursivement.

Question 3 (*) *Classer les éléments de $\mathcal{S}$ en fonction de M. Analyser la taille des paquets ainsi engendrés. En déduire un algorithme récursif pour calculer le k^e plus grand élément.*

On considére l'équation récursive suivante:

$$c(n) = \begin{cases} A & \text{si } n = 1, \\ B + c(\alpha n) & \text{sinon} \end{cases}$$

où α est un réel dans $[0, 1)$.

Question 4 *Montrer que $c(n) = O(n)$ et donner la constante multiplicative cachée dans la notation $O()$, c'est-à-dire C tel que $c(n) \leq Cn$.*

Question 5 *Étudier la complexité de l'algorithme de sélection du k^e plus grand élément en fonction de p, le nombre de paquets. Quelle est la valeur de p qui minimise le coût dans le cas le pire de votre algorithme?*

Question 6 *Donner une borne inférieure pour la recherche du k^e élément dans une liste non triée. Comment se comportent vos algorithmes dans le cas d'éléments identiques? Que se passe-t-il si $\mathcal{S}$ est une liste triée?*

Soit $\mathcal{I} = \{i_1, ..., i_q\}$ un ensemble de q entiers dans $[1, n]$, on désire calculer les q éléments de $\mathcal{S}$ de rangs respectifs $i_1, ..., i_q$.

Question 7 $(*)$ *En partitionnant $\mathcal{I}$ en deux paquets $\mathcal{I}_1$ et $\mathcal{I}_2$ grâce à la médiane de $\mathcal{I}$, montrer comment on peut donner un algorithme de complexité $O(n \log q)$. Pour cela, on donnera l'algorithme et sa fonction récursive de complexité $c(n)$ puis on montrera qu'il existe $A \in \mathbb{R}$ tel que $c(n) \leq An \log q$.*
Comparer ce résultat numériquement avec un algorithme de tri en $O(n \log n)$.