

Examen

- 19 Octobre 2021 -

Durée 2 heures

Deux feuilles manuscrites autorisées.

- **Exercice 1** - On veut énumérer tous les mots de $0,1$ de longueur k . Une solution possible est d'incrémenter un compteur binaire, on obtient par exemple pour $k = 3$:

000, 001, 010, 011, 100, 101, 110, 111

Toutefois le passage d'un mot au suivant peut changer plus d'un bit (par exemple passer de 011 à 100 en change trois) et l'on souhaite une solution plus économique. Frank Gray a proposé l'algorithme suivant: On part du mot 0^k et on termine lorsque le mot est 1.0^{k-1} . De plus, à chaque étape, on calcule ainsi le mot suivant: (i) si le nombre de 1 est pair on change le bit le plus à droite, (ii) si le nombre de 1 est impair on change le bit qui est à gauche de la lettre 1 qui est la plus à droite. Cet ordre sur $\{0,1\}^k$ est appelé *code de Gray*.

1. Calculer les codes de Gray pour les valeurs de k inférieures ou égales à 4.
2. On veut visualiser le code de Gray. Pour cela, on considère le graphe suivant appelé *hypercube* de dimension k : ses sommets sont les mots de $0,1$ de longueur k et deux sommets sont reliés par une arête lorsqu'ils diffèrent d'un bit. Dessiner les hypercubes de dimension 1, 2 et 3. Que représente le code de Gray pour chacun?
3. Dédurre de la question précédente une construction du code de Gray sur $k + 1$ lettres basée sur le code de Gray sur k lettres.
4. * On souhaite réhabiliter l'incrément de compteur binaire, finalement plus intuitif: montrer que le nombre moyen de changements de bits par itération (on effectue $2^k - 1$ itérations) est au plus 2.

- **Exercice 2** - Un mot sur l'alphabet $\{(,)\}$ est *bien parenthésé* s'il a autant de lettres $)$ que de lettres $($ et si tout préfixe possède au moins autant de lettres $($ que de lettres $)$.

1. Ecrire tous les mots bien parenthésés sur au plus 8 lettres.
2. Décrire le fonctionnement général d'une machine de Turing qui reconnaît si un mot en entrée sur son ruban est bien parenthésé ou pas. * On pourra aussi la présenter explicitement.
3. * Peut-on reconnaître le langage des mots bien parenthésés à l'aide d'un automate?

4. Lorsqu'un mot de longueur $2n$ est bien parenthésé, on peut indiquer chaque $)$ et chaque $($ par un nombre dans $\{1, \dots, n\}$ de sorte que pour tout nombre i il existe une unique paire $(_, _)_i$. De plus, $(_)$ est toujours situé avant $)_i$, et on ne peut croiser les parenthèses, c'est à dire $\dots (i \dots (j \dots)_i \dots)_j$ est interdit. Indicer les parenthésages sur 6 lettres.
5. Proposer un algorithme qui reçoit en entrée un mot bien parenthésé et effectue l'indigage précédent. Utiliser pour cela une pile. Ecrire le pseudo-code de votre algorithme en supposant que l'entrée est un tableau de $2n$ parenthèses et la sortie le tableau des $2n$ indices.
6. Vous avez réussi à placer un émetteur dans l'ordinateur de votre principal concurrent industriel, spécialisé dans le calcul d'arbre en profondeur. Malheureusement le signal bruité ne vous autorise qu'à détecter les appels empiler e et dépiler d de sa pile. Aujourd'hui, vous avez détecté *eedededdeededdeddd*, pouvez vous retrouver l'arbre calculé?

- **Exercice 3** - On se donne un ensemble E de n espèces animales $x_1, \dots, x_n$ et une fonction d de *dissimilarité* sur les couples d'espèces. On suppose juste que d vérifie $d(x_i, x_j) = d(x_j, x_i)$ pour tous les i, j et $d(x_i, x_i) = 0$, mais aucune autre propriété utilisable algorithmiquement. On cherche néanmoins à trouver une structure d'arbre naturelle sur cet ensemble d'espèces.

Une fonction de dissimilarité f est dite *arborescente* si elle est construite comme suit:

- L'ensemble E correspond aux n feuilles d'un arbre binaire A enraciné (l'ancêtre de tous les noeuds)
- chaque noeud v de A possède une valeur $h(v)$ vérifiant que $h(u) \geq h(v)$ si u est un ancêtre de v et $h(x_i) = 0$ pour chaque feuille x_i .
- pour chaque paire de feuilles x_i, x_j , la valeur de $f(x_i, x_j)$ est égale à $h(v)$ où v est le plus proche ancêtre commun de x_i et x_j .

On dit qu'une fonction arborescente f *approche* d si $f(x_i, x_j) \leq d(x_i, x_j)$ pour tous les couples x_i, x_j .

1. Montrer qu'une fonction arborescente f vérifie que pour tout triplet x_i, x_j, x_k de E , les deux plus grandes valeurs de $\{f(x_i, x_j), f(x_i, x_k), f(x_j, x_k)\}$ sont égales.
2. ** Montrer que si une fonction f vérifie la propriété précédente, alors c'est une fonction arborescente. On pourra admettre cette question pour la suite.
3. Soit d une fonction de dissimilarité telle que $d(x_i, x_j) \leq d(x_i, x_k) < d(x_j, x_k)$ pour un certain triplet x_i, x_j, x_k . On construit d' à partir de d en posant $d'(x_j, x_k) = d'(x_k, x_j) = d(x_i, x_k)$ et d' par ailleurs égale à d sur tous les autres couples. Montrer que toute fonction arborescente f approchant d approche aussi d' .

4. Utiliser les deux questions précédentes pour calculer une fonction arborescente qui approche d donnée par la matrice suivante:

$$\begin{pmatrix} 0 & 1 & 2 & 6 \\ 1 & 0 & 3 & 5 \\ 2 & 3 & 0 & 4 \\ 6 & 5 & 4 & 0 \end{pmatrix}$$

5. On dit que f approchant d est *optimale* si pour toute fonction arborescente g approchant d , on a $g(x_i, x_j) \leq f(x_i, x_j)$ pour tous les x_i, x_j . Montrer qu'il existe toujours une fonction arborescente f optimale approchant une fonction d .
6. Montrer que le raisonnement précédent permet de construire f en temps polynomial en n . Evaluer la complexité de votre algorithme dans le pire des cas.
7. Proposer un algorithme de calcul de fonction arborescente basé sur l'algorithme de Kruskal. Quelle est sa complexité?
8. * Montrer que la solution retournée par Kruskal est optimale.

- **Exercice 4** - Indiquer une méthode de programmation dynamique qui permet de résoudre les problèmes suivants (on indiquera juste comment relier un sous-problème à ceux plus petits).

1. On dispose d'un sac à dos de capacité L litres et d'une liste de n objets, chacun ayant un volume v_i en litres et un prix p_i . On veut remplir le sac en maximisant le prix. *Indication:* on pourra considérer des sous-problèmes avec deux paramètres (i, l) l'un limitant les objets choisis et l'autre limitant le volume.
2. On a en entrée une image en noir et blanc (vue comme une matrice $n \times n$ de 0 et 1). Calculer un carré de 1 de taille maximale en temps $O(n^2)$.
3. * L'entrée est un tableau de n nombres. Calculer un sous-tableau maximal (pas forcément consécutif) dont les valeurs sont strictement croissantes.

- **Exercice 5** - * L'exercice précédent permet de résoudre le problème du sac à dos en temps polynomial en $n + L$. En réalité, la véritable taille de l'entrée de ce problème est de l'ordre de $n \log_2 L$ car les entiers peuvent se coder en binaire (on suppose ici que les volumes et les prix sont inférieurs à L). Montrer qu'il n'existe pas d'algorithme résolvant le problème du sac à dos en temps polynomial en $n \log_2 L$ à moins que $P = NP$.