Master 1 STL 4I505 CPA – Examen Mai 2017

Répondez aux parties A et B sur des copies séparées, pour que nous puissions les corriger en parallèle (A=Christoph Dürr, B=Antoine Genitrini). Les documents distribués dans le cadre du cours et les notes personnelles sont autorisés.

A Programmation dynamique et structures de données

A.1 Cailloux (4 points)

Dans une chambre secrète de l'UPMC se trouvent les cailloux philosophaux. Le sol de la chambre est pavé avec $h \times w$ carrelages carrés, organisés en h lignes de l'entrée (première ligne) au fond de la chambre (dernière ligne) et en w colonnes de gauche à droite. Sur chacun des carrelages sont posés entre 1 et 100 cailloux. Vous allez traverser cette chambre de la manière suivante. D'abord vous vous posez sur un des carrelage de votre choix de la première ligne. Puis vous allez sur un carrelage de la ligne suivante, soit en allant tout droit, soit en allant en diagonale vers la gauche ou vers la droite. Quand vous avez atteint la dernière ligne, votre traversé de la chambre est terminée. Sachant que vous allez ramasser tous les cailloux des carrelages que vous traversez, déterminez le nombre maximum de cailloux que vous pourriez ramasser.

Donnez un programme dynamique qui résout ce problème en temps $O(h \cdot w)$, étant donné une matrice $M \in \{1, \ldots, 100\}^{h \times w}$. Prouvez qu'il est correct et qu'il a la complexité attendue.

entrée de la chambre

3	1	7	4	2
2	1	3	1	1
1	2	2	1	8
2	2	1	5	3
2	1	4	4	4
5	2	7	5	1

fond de la chambre

Figure 1: La réponse sur cette grille est 32, ce qui représente $7 + 1 + 8 + 5 + 4 + 7$.

A.2 Peindre des dalles (6 points)

Dans une petite ville existe une promenade qui la traverse d'est en ouest d'une longueur de 2^m mètres pour un entier positif m. Elle est composée de dalles carrées d'un mètre de côté mis bout à bout. Ces dalles sont numérotées successivement de 0 à $2^m - 1$, la dalle 0 étant la plus à l'ouest et la dalle $2^m - 1$ étant la plus à l'est. Les habitants de la ville n'arrivent pas à se mettre d'accord sur la couleur que la promenade devrait avoir. Le jour de son inauguration toutes les dalles sont grises. Puis toutes les nuits un groupe d'habitants se met à repeindre un tronçon de la promenade, avec des couleurs alternants de nuit en nuit. L'administration de la ville vous a chargé de maintenir des données sur les couleurs des dalles, de telle sorte qu'à tout moment ils peuvent vous questionner rapidement sur la couleur d'une dalle particulière. Chaque matin ils vous informent d'un intervalle $[a, b]$ de dalles qui ont été peintes dans la couleur c durant la nuit avec $0 \leq a \leq b < 2^m$ et $c \in \{1, 2\}$. Les couleurs sont codées par des entiers, 0=gris, 1=blanc, 2=noir.

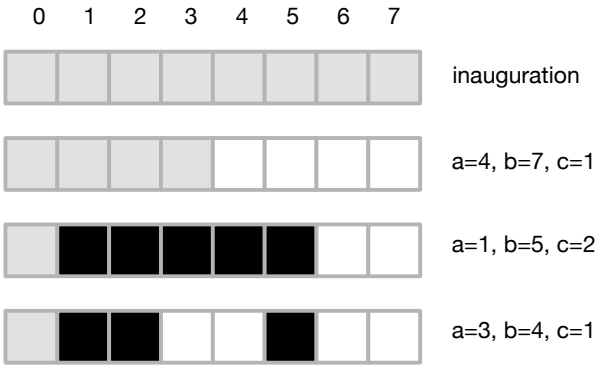

Figure 2: L'évolution des couleurs des dalles de la promenade.

Pour cela concevez une structure de données T qui implémente les opérations suivantes.

T(m) le constructeur initialise la structure de données représentant un tableau d'entiers t indicée de 0 à $2^m - 1$. Initialement t ne contient que les valeurs 0.

paint(a, b, c) a pour effet de mettre la valeur c dans toutes les entrées $t[a], t[a+1], \ldots, t[b]$. Les paramètres satisfont $0 \leq a \leq b < 2^m$ et $c \in \{1, 2\}$.

color(a) retourne la valeur de $t[a]$.

Le constructeur doit avoir une complexité en temps de $O(2^m)$, et les méthodes **paint** et **color** une complexité $O(m)$. Adaptez les arbres de segments pour résoudre ce problème. Donnez votre solution en peudo-code, analysez la complexité des méthodes, et expliquez pourquoi votre solution est correcte.

B Codage de Huffman Généralisé (10 points)

On se propose de généraliser le codage de Huffman binaire au cas non binaire, c'est-à-dire lorsqu'on dispose pour coder non pas de bits, mais m *caractères* avec $m > 2$.

On appelle *symboles* les lettres de l'alphabet du texte à compresser. On note n la taille de cet alphabet (nombre de symboles distincts du texte, $n \geq m$), et pour $i = 1, \ldots, n$, on note f_i la fréquence du symbole a_i.

On rappelle l'algorithme $\mathcal{H}_2$ de construction d'un arbre binaire de Huffman
ENTRÉE : fréquences $f_1, \ldots, f_n$ des symboles $a_1, \ldots, a_n$ d'un texte T.
SORTIE : arbre digital donnant un code préfixe minimal pour T.

1. Créer, pour chaque symbole a_i, $i = 1, \ldots, n$, un arbre A_i (réduit à une feuille) de poids f_i
2. Itérer le processus suivant : remplacer 2 arbres G et D de poids minimum par un nouvel arbre R ayant pour sous-arbre gauche G et pour sous-arbre droit D, et affecter comme poids à R la somme des poids de G et D.
   Arrêter lorsqu'il ne reste qu'un seul arbre et renvoyer cet arbre.

L'algorithme de Huffman $\mathcal{H}_2$ satisfait les propriétés suivantes :

(α) si un symbole a_i apparaît plus souvent dans le texte qu'un symbole a_j alors a_i possède un codage plus court (au sens large) que a_j,

(β) deux symboles a_i et a_j, de fréquences d'apparition minimales strictement, c'est-à-dire, $f_i \leq f_j$ et $\forall k \notin \{i, j\}, f_k > f_j$, ont un codage de même longueur et ces 2 codages diffèrent seulement par leur dernier caractère.

Question 1 *Montrer que si un code préfixe binaire est minimal alors la propriété* (α) *est bien vérifiée.*

Question 2 *Montrer que si un code préfixe binaire est minimal et s'il existe deux symboles a_i et a_j, de fréquences d'apparition minimales strictement, alors la propriété* (β) *est bien vérifiée.*

On peut obtenir un code de Huffman non binaire de la même manière qu'avec l'algorithme de Huffman binaire. Nous allons cependant constater que la propriété (β) pose un problème si on essaie de la remplacer respectivement par (β') : "les m symboles qui ont les fréquences d'apparition minimales strictement ont un codage de même longueur et les codages de ces m symboles diffèrent seulement par leur dernier caractère".

Question 3 *Quel problème rencontre-t-on si, dans l'étape 2 de cet algorithme, on remplace simplement "2 arbres" par "m arbres" ? Pourquoi n'a-t-on pas de problème dans le cas binaire ?*

On considère ci-dessous l'algorithme $\mathcal{H}_m$ de construction d'un arbre m-aire.

1. Créer, pour chaque symbole a_i, $i = 1, \ldots, n$, un arbre (réduit à une feuille) de poids f_i
2. Tant qu'on dispose d'au moins m arbres, itérer le processus suivant : remplacer m arbres de poids minimum, notés $A_1, \ldots, A_m$, par un nouvel arbre, noté A, ayant pour fils $A_1, \ldots, A_m$, et affecter comme poids à A, la somme des poids des A_i.
3. Renvoyer l'arbre qui a pour fils tous les arbres restants.

On veut générer un arbre de Huffman ternaire ($m = 3$) pour un texte source formé à partir de 6 symboles distincts (A,B,C,D,E,F) avec les fréquences d'apparition suivantes : $f(B) = f(C) = f(E) = 5, f(D) = 4, f(A) = 2$ et $f(F) = 1$. On utilise l'alphabet ternaire $0, 1, 2$ pour le codage.

Question 4 *Dérouler l'algorithme $\mathcal{H}_3$ sur l'exemple précédent en donnant les arbres ternaires obtenus à chaque étape avec leur poids.*

Question 5 *Calculer la taille du texte compressé (nombre de caractères $0, 1$ ou 2) en utilisant l'arbre de Huffman ternaire ainsi construit.*

Question 6 *Prouver, pour tout $m > 2$, que les propriétés (α), (β') sont satisfaites par le code l'algorithme $\mathcal{H}_m$, lorsqu'il y a m symboles (parmi les n) dont les fréquences d'apparition sont minimales strictement.*

Question 7 *Montrer qu'un arbre contenant p nœuds internes, tous exactement de degré m, contient exactement $p \times (m-1) + 1$ feuilles.*

Question 8 *S'il existe $p \in \mathbb{N}$ tel que $n = p \times (m-1) + 1$, que peut-on dire de l'arbre construit par l'algorithme $\mathcal{H}_m$?*

Question 9 *Supposons qu'il n'existe pas $p \in \mathbb{N}$ tel que $n = p \times (m-1) + 1$. Soit $q \in \mathbb{N}$ tel que $q \times (m-1) + 1 < n < (q+1) \times (m-1) + 1$. Comment calculer q en fonction de n et de m ? Que représente q dans l'arbre construit par $\mathcal{H}_m$?*
Soit m' le degré de la racine de l'arbre construit par $\mathcal{H}_m$. Calculer m' en fonction de m, n et q. On admet que $2 \leq m' < m$.

On propose un autre algorithme de construction de l'arbre. Lorsque tous les nœuds internes ne peuvent pas être de degré m, on admet que plutôt que ce soit la racine dont le nombre de fils est m', on peut construire un arbre dont le nœud interne le plus profond est de degré m' et tous les autres de degré m.
On considère ci-dessous l'algorithme $\mathcal{H}'_m$ de construction d'un arbre m-aire.

1. Créer, pour chaque symbole a_i, $i = 1, \ldots, n$, un arbre (réduit à une feuille) de poids f_i

2. S'il n'existe pas $p \in \mathbb{N}$ tel que $n = p \times (m-1) + 1$, calculer m'. Remplacer m' arbres de poids minimum, notés $A_1, \ldots, A_{m'}$, par un nouvel arbre, noté A, ayant pour fils $A_1, \ldots, A_{m'}$, et affecter comme poids à A, la somme des poids des A_i.

3. Itérer le processus suivant : Remplacer m arbres, notés $A_1, \ldots, A_m$, de poids minimum par un nouvel arbre A ayant pour fils $A_1, \ldots, A_m$, et affecter comme poids à A la somme des poids des A_i.
   Arrêter lorsqu'il ne reste qu'un seul arbre et renvoyer cet arbre.

Question 10 *Dans l'exemple précédent, où $n = 6$ et $m = 3$, vérifier qu'il n'existe pas $p \in \mathbb{N}$ tel que $n = p \times (m-1) + 1$ et vérifier que m' vaut 2.*

Question 11 *Dérouler chaque étape de l'algorithme $\mathcal{H}'_3$ sur l'exemple ci-dessus en donnant les arbres obtenus à chaque étape avec leur poids. Les arêtes des arbres sont étiquetées de gauche à droite respectivement par $0, 1$ et 2.*

Question 12 *Calculer la taille du texte compressé (nombre de caractères $0, 1$ ou 2) en utilisant l'arbre de Huffman ternaire H ainsi construit.*

Question 13 *Quel est le résultat de la compression à l'aide de l'arbre H du texte suivant :* `BCADFFEA`

Question 14 *Examiner si les propriétés (α) et (β') sont satisfaites par le code obtenu avec l'algorithme $\mathcal{H}'_m$.*

Question 15 *Montrer qu'un code préfixe m-aire vérifiant les propriétés (α) et (β') n'est pas forcément minimal.*

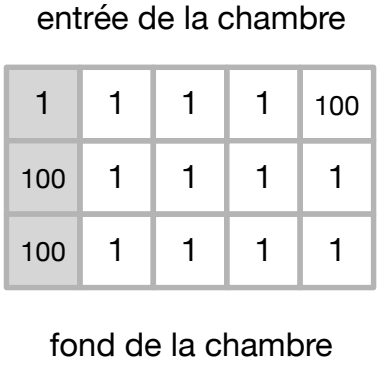

Figure 3: La solution optimale ne débute pas par la case maximale en première ligne.

C Éléments de correction

C.1 Cailloux

Un programme dynamique était demandé, pas un algorithme glouton. D'ailleurs partir de la case maximale de la première ligne ne mène pas vers la solution optimale comme vous pouvez le voir en figure 3.

Notons par $A[i, j]$ le points maximum atteignable en marchant de la ligne 1 à la case en ligne i et colonne j. Le cas de base est $A[1, j] = M[1, j]$, car il y a un seul chemin possible pour atteindre un case de la première ligne. Le cas d'induction est pour $2 \leq i \leq h$ et $2 \leq j \leq w - 1$

$$A[i, j] = M[i, j] + \max\{A[i - 1, j - 1], A[i - 1, j], A[i - 1, j + 1]\}.$$

En effet il n'y a que ces trois cases à partir des quelles on peut atteindre la case (i, j) en un seul pas. Les formules d'induction pour $A[i, 1]$ et $A[i, w]$ sont similaires mais avec une valeur de moins afin de ne pas dépasser le bord de la grille. L'optimum est simplement $\max_j A[1, j]$. Il y a $O(w \cdot h)$ variables et le calcul de chacune se fait en temps constant.

C.2 Peindre un mur

Voir le billet correspondant de notre blog tryalgo.

Idée: On construit un arbre binaire avec 2^m feuilles, voir Figure 4. Chaque nœud de l'arbre est responsable d'une plage d'indices dans t, appelons range(i) cette plage. Le nœud i est étiquetté par une valeur $s[i]$. Si celle-ci est 0 la plage d'indices n'a pas été repeinte. Sinon celle-ci est de couleur $s[i]$ pour le sous-arbre. Pour la méthode **paint(a,b,c)** il suffit de décomposer l'intervalle $[a, b]$ en intervalles correspondant à des nœuds de l'arbre et de leur donner l'étiquette c. Seulement $O(m)$ nœuds sont concernés par cette mise à jour. Pour la méthode **color(a)** il suffit de suivre le chemin de la racine au nœud correspondant à $t[a]$. Si

les étiquettes sont tous 0 la réponse sera 0. Sinon elle sera la valeur de la première étiquette positive rencontrée.

Détail: Au début du traitement d'un nœud i par la méthode **paint(a,b,c)** il faut propager la couleur de i aux descendants.

```
class T:
    def __init__(self, m):
        self.N = 1 << m                      # 2 puissance m
        self.s = [0] * 2 * self.N            # 0 = gris ou pas de couleur

    def paint(self, a, b, c):
        self._paint((a,b), c, 1, (0, self.N - 1))  # initier à la racine

    def _paint(self, color_range, color, node, node_range):
        if disjoint(color_range, node_range):
            return                           # rien à faire
        elif included(node_range, color_range):
            self.s[node] = color             # colorier juste ce nœud
        else:
            left_son = 2 * node
            right_son = 2 * node + 1
            if self.s[node] != 0:            # propager couleur vers le bas
                self.s[left_son] = self.s[node]
                self.s[right_son] = self.s[node]
            left_range, right_range = split(node_range)
            self._paint(color_range, color, left_son, left_range)
            self._paint(color_range, color, right_son, right_range)

    def color(self, a):
        node = 1                             # démarrer à la racine
        node_range = (0, self.N - 1)
        while (node_range != (a, a)):        # feuille ?
            if self.s[node] != 0:
                return self.s[node]          # première couleur rencontrée
            left_range, right_range = split(node_range)
            if contains(left_range, a):      # descendre dans l'arbre
                node = 2 * node
                node_range = left_range
            else:
                node = 2 * node + 1
                node_range = right_range
        return self.s[node]
```

Ce code utilise les fonctions utiles suivantes sur les intervalles.

```
def disjoint(A, B):
    return max(A) < min(B) or max(B) < min(A)

def included(A, B):
    return min(B) <= min(A) and max(A) <= max(B)

def split(A):
```

```
    lo, hi = A
    mid = (lo + hi + 1) // 2
    return (lo, mid - 1), (mid, hi)

def contains(A, x):
    return min(A) <= x and x <= max(A)
```

C.3 Arbres de Huffman

1. cf cours, slide 17
2. a_i et a_j sont jumelés au 1er coup (donc dernier caractère distinct) et après ils appartiennent toujours au même arbre : donc les caractères suivants sont identiques
3. A la dernière étape il ne reste pas forcément m arbres. Pour 2, à chaque fois le nb d'arbres est décrémenté de 1.
4. D-A-F (pds 7) puis B-C-E (pds 15), il reste 2 arbres (D-A-F)-(B-C-E)
5. 44
6. même preuve que question 2
7. récurrence par exemple
8. Tous les nœuds internes sont de degré exactement m
9. $q = \text{quotient}(n-1, m-1)$ car $q \times (m-1) < n-1 < (q+1) \times (m-1)$.
   q est le nombre de nœuds interne de degré exactement m.
   $m' = n - q \times (m-1)$.
10. $n = 6$ n'est pas un nombre impair.
    $m' = 2$
11. A-F (pds 3), E-D-(A-F) (pds 12) et B-C-(E-D-(A-F))
12. 37
13. tout dépend de l'arbre
14. non pas 2'
15. $37 < 44$, $\mathcal{H}_m$ vérifie 1 et 2', mais n'est pas minimal.

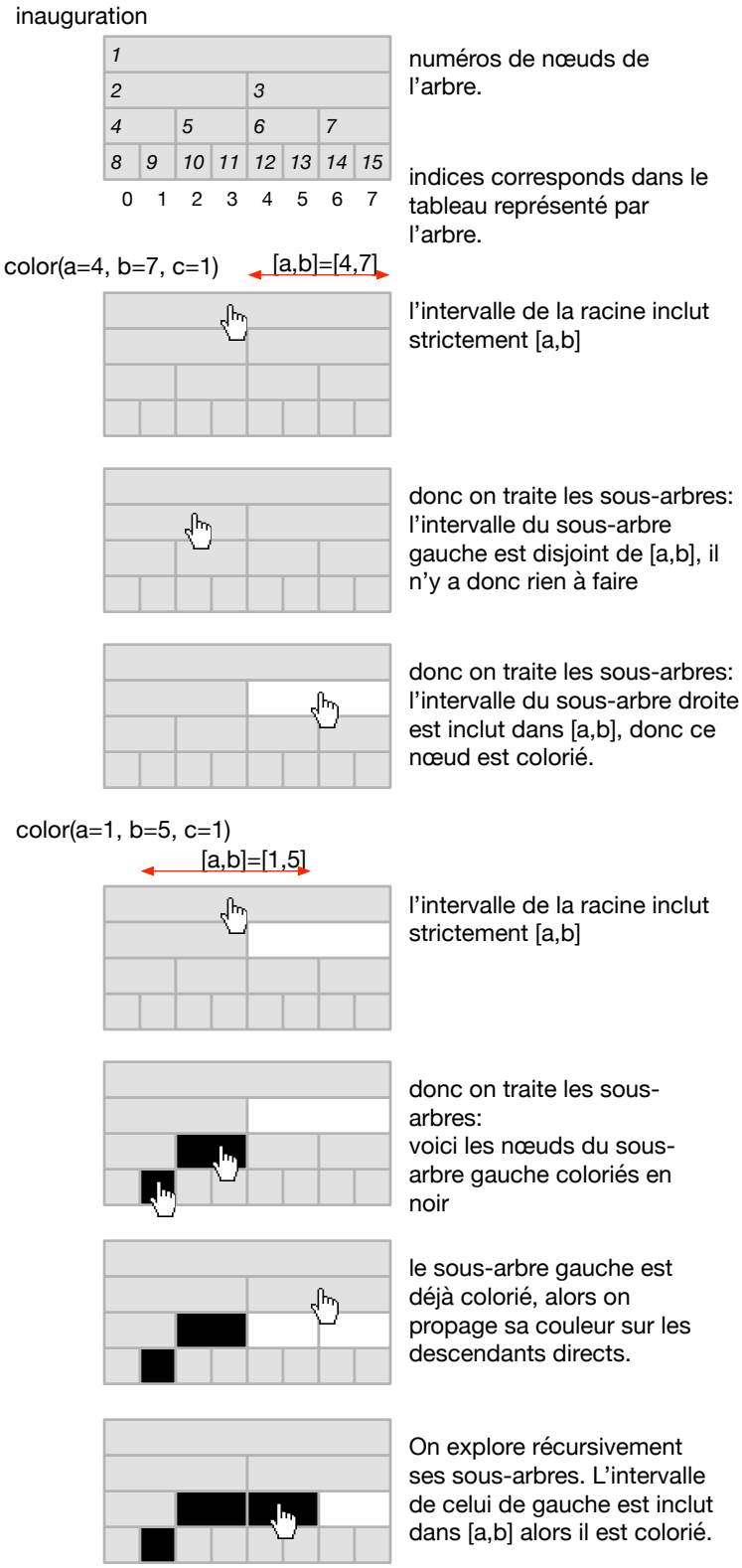

Figure 4: Exemples de manipulation de l'arbre T.