Outils logiques et algorithmiques – Examen – Mai 2024

Durée 2h. Le sujet comporte 3 exercices indépendants. Notes personnelles et documents de cours autorisés. Barème approximativement proportionnel au temps nécessaire à la résolution de chaque question.

Exercice 1. Tableaux dynamiques *(50 minutes)*

Dans cet exercice on va construire une structure de données, qui permet d'accéder aux éléments par un indice à la manière de ce que l'on fait dans un tableau, dans laquelle on peut également ajouter efficacement de nouveaux éléments en tête. Ainsi, partant du tableau

`t` ⟶ | 2 | 4 | 8 | 16 | 32 |

pour lequel `t[0] = 2` et `t[3] = 16`, on pourrait ajouter l'élément `1` en tête et obtenir un tableau

`t'` ⟶ | 1 | 2 | 4 | 8 | 16 | 32 |

dans lequel `t'[0] = 1` et `t'[3] = 8`.

Représentation. Pour représenter un tel tableau, on utilise des arbres binaires dont chaque feuille porte un nombre entier, définis par les constructeurs suivants :

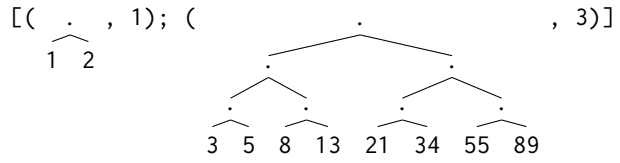
```
type arbre =
  | F of int
  | N of arbre * arbre
```

Un arbre binaire complet d'ordre 0 est une feuille. Un arbre binaire complet d'ordre $k+1$ est un nœud ayant pour fils deux arbres binaires complets d'ordre k.

Un tableau dynamique est alors représenté par une liste `[(`a_1`, `k_1`); (`a_2`, `k_2`); ...; (`a_n`, `k_n`)]` de paires d'un arbre binaire complet et d'un entier tels que pour tout i, l'arbre a_i est d'ordre k_i, et les k_i sont tous différents et rangés en ordre croissant. Les éléments du tableau sont les feuilles des arbres a_i, prises dans l'ordre. Ainsi, le tableau

`f` ⟶ | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 |

sera représenté par



Questions.

1. Donner des arbres binaires complets d'ordre 0, 1, 2 et 3.
2. Combien de feuilles possède un arbre binaire complet d'ordre k ? Démontrez votre affirmation.
3. Définir une fonction `get_a` telle que `get_a i a k` renvoie l'entier stockée dans la feuille d'indice i de l'arbre a, en supposant que a est un arbre binaire complet d'ordre k et que a contient bien au moins $i+1$ feuilles (l'indice 0 désigne la feuille la plus à gauche, l'indice 1 désigne la suivante, etc). Cette fonction doit avoir une complexité temporelle proportionnelle à k. *Votre réponse peut, au choix, prendre la forme d'un ensemble d'équations ou d'un code caml. L'expression caml* `1 lsl k` *calcule* 2^k.
4. Énumérer les invariants que doit respecter notre structure de tableau dynamique.
5. Donner une borne sur la longueur de la liste en fonction du nombre total d'éléments du tableau.
6. Voici le code d'une fonction accédant à l'élément d'indice `i` d'un tableau dynamique `t`.

```
let rec get i t = match t with
  | [] -> raise Not_found
  | (a, k) :: tl ->
     let n = 1 lsl k in
     if i < n then get_a i a k
              else get (i-n) tl
```

Donner un ordre de grandeur de sa complexité temporelle, exprimé en fonction du nombre total n d'éléments dans le tableau.

Ajout/retrait d'éléments. Pour ajouter un élément `x` à un tableau dynamique `t`, on crée d'abord une feuille `F(x)` contenant cet élément, puis on fusionne éventuellement cette feuille avec certains arbres du tableau `t` de sorte à maintenir les invariants de la structure. Voici une fonction `add` réalisant cela à l'aide d'une fonction auxiliaire `merge`.

```
let rec merge (a, k) t = match t with
  | [] -> [(a, k)]
  | (a', k') :: tl ->
     if k < k' then
       (a, k) :: t
     else (* k=k' *)
       merge (N(a, a'), k+1) tl

let add x t = merge (F(x), 0) t
```

Questions.

7. Dessiner la structure `f'` obtenue après avoir ajouté `1` en tête du tableau `f` précédent, puis la structure `f"` obtenue après avoir ajouté `0` en tête de `f'`.
8. Le code de `merge` suggère une précondition pour cette fonction. Quelle est-elle ? Pouvez-vous justifier qu'elle est bien vérifiée dans chacun des appels à `merge` présents dans ce code ?
9. Quelle est la complexité de `add` dans le meilleur cas ? Dans le pire cas ? Avez-vous une conjecture concernant la moyenne ?
10. Proposer une fonction `remove` telle que `remove t` renvoie la structure obtenue en retirant l'élément de tête de `t`. Que dire de sa complexité ? *La structure renvoyée doit toujours respecter les invariants des tableaux dynamiques.*

Exercice 2. Mobiles de Calder *(20 minutes)*

Rappel du DM : un *mobile* est formé par un ensemble d'objets, suspendus à des barres elles-mêmes suspendues en équilibre à d'autres barres, et ainsi de suite jusqu'à un unique point de suspension auquel pend l'ensemble de la structure. On décrit les mobiles comme des arbres binaires pouvant prendre l'une des deux formes suivantes :

— soit un objet `O(`w`)` de poids w strictement positif,

— soit une barre horizontale `B(`m_1`, `m_2`)` aux extrémités de laquelle pendent deux sous-mobiles m_1 et m_2.

On les représente en caml à l'aide du type algébrique suivant :

```
type mobile =
  | O of int
  | B of mobile * mobile
```

Le *poids* d'un mobile est la somme des poids de tous ses objets. Un mobile est considéré comme *équilibré* lorsque, pour chacune de ses barres `B(`m_1`, `m_2`)`, les sous-mobiles m_1 et m_2 ont exactement le même poids. *Note : pour simplifier cette étude, on néglige le poids des barres et des fils auxquels sont suspendus les éléments.* On définit en outre la *hauteur* d'un mobile comme le nombre maximal de barres à traverser pour aller du point de suspension à un objet.

Questions.

1. On note $\#_k$ le nombre de mobiles équilibrés de poids 2^k. On affirme que ce nombre vérifie les équations suivantes :

$$\begin{aligned} \#_0 &= 1 \\ \#_{k+1} &= 1 + (\#_k)^2 \end{aligned}$$

   Justifier ces équations, et en déduire une fonction caml `count: int -> int` telle que `count` k calcule $\#_k$.
2. Montrer que pour tout k on a $\#_k \geq 2^k$.
3. Montrer que tout mobile équilibré de hauteur h a un poids supérieur ou égal à 2^h. *Rédaction soignée indispensable pour cette question.*

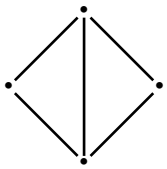
Exercice 3. Graphes réguliers et chemins hamiltoniens *(50 minutes)*

Dans cet exercice, on considère des graphes non orientés et simples (c'est-à-dire sans boucles ni arêtes multiples). Un *graphe k-régulier* est un graphe dont tous les sommets ont le même degré k. On parlera simplement de *graphe régulier* lorsque l'on ne souhaite pas préciser la valeur de k.

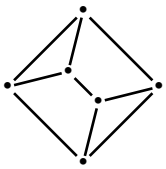
Questions.

1. Les graphes suivants sont-ils réguliers ? Pour quelle valeur de k ?

(a)



(b)



2. La clique à n sommets est-elle un graphe régulier ? Si oui, préciser la valeur de k.
3. Dessiner deux graphes 2-réguliers.
4. Combien d'arêtes a un graphe k-régulier avec n sommets ?
5. Peut-on trouver un graphe 9-régulier avec 11 sommets ?

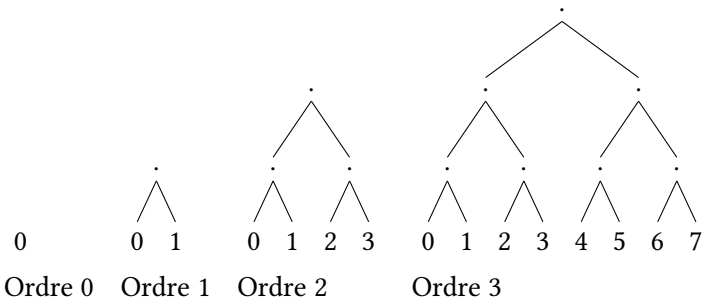
Dans la suite de l'exercice, on considère un graphe simple G qui est k-régulier et a $n = 2k$ sommets. On va montrer que ce graphe admet nécessairement un cycle hamiltonien, c'est-à-dire un cycle passant exactement une fois par chaque sommet du graphe avant de revenir à son point de départ. Dans toutes les questions qui viennent, par *chemin* on sous-entend toujours *chemin élémentaire*, c'est-à-dire chemin ne passant pas deux fois par le même sommet (à l'exception possible du point de départ/arrivée d'un cycle, vu exactement deux fois).

Questions.

6. Sachant que le graphe simple G est k-régulier, quelle est la taille minimale d'une composante connexe ? En déduire que G est nécessairement connexe.
7. On suppose que G contient un cycle élémentaire $\rho : s_1 \longrightarrow s_2 \longrightarrow \ldots \longrightarrow s_d \longrightarrow s_1$ de longueur d, et une arête a dont l'une des extrémités est un sommet s_i de ρ et l'autre extrémité *n'est pas* un sommet de ρ. Construire un chemin élémentaire de longueur $d + 1$.
8. On suppose que G contient un chemin élémentaire $\rho : s_1 \longrightarrow \ldots \longrightarrow s_d$ de longueur $d - 1$, et qu'il existe un entier i ($2 \leq i \leq d - 2$) tel que G contient deux arêtes $a : s_1 \longrightarrow s_{i+1}$ et $b : s_d \longrightarrow s_i$. Construire un cycle élémentaire de longueur d.
9. Soit $\rho : s_1 \longrightarrow s_2 \longrightarrow \ldots \longrightarrow s_d$ un chemin élémentaire de longueur maximale dans G. Montrer que tous les sommets adjacents à s_1 sont dans l'ensemble $\{s_2, \ldots, s_d\}$ et que tous les sommets adjacents à s_d sont dans l'ensemble $\{s_1, \ldots, s_{d-1}\}$.
10. En déduire que si $\rho : s_1 \longrightarrow \ldots \longrightarrow s_d$ est un chemin élémentaire de longueur maximale dans G, alors il existe un i tel que G contient une arête $s_1 \longrightarrow s_{i+1}$ et une arête $s_d \longrightarrow s_i$.
11. Combiner les questions précédentes pour démontrer que tout graphe G qui est k-régulier et a $2k$ sommets contient nécessairement un cycle hamiltonien.

Correction exercice 1.

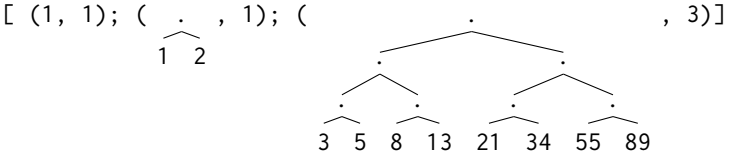
1.



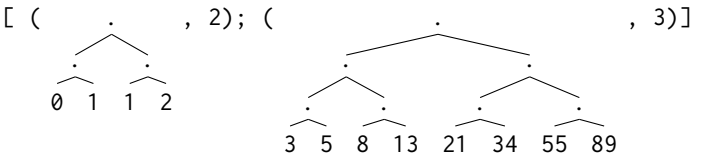
2. Un arbre binaire complet d'ordre k possède 2^k feuilles. Démonstration par récurrence sur k :
   - Par définition, un arbre binaire complet d'ordre 0 possède 1 feuille, c'est-à-dire 2^0 feuille.
   - Soit k tel que tout arbre binaire complet d'ordre k possède 2^k feuilles. Soit a un arbre binaire complet d'ordre $k+1$. Par définition, a a la forme $a = \texttt{N}(a_1,\ a_2)$ avec a_1 et a_2 deux arbres binaires complets d'ordre k. Par hypothèse de récurrence, a_1 et a_2 ont chacun 2^k feuilles. Donc a a $2^k + 2^k = 2 \times 2^k = 2^{k+1}$ feuilles.
3. Un arbre d'ordre k contient 2^k éléments. Si l'arbre n'est pas une feuille, les 2^{k-1} premiers éléments sont dans le sous-arbre gauche, et les 2^{k-1} suivants dans le sous-arbre droit.

```
1  let rec get_a i a k = match a with
2    | F(x) -> x (* par hypothèse, i = k = 0 *)
3    | N(a1, a2) ->
4       let n = 1 lsl (k-1) in
5       if i < n then get_a  i    a1 (k-1)
6                else get_a (i-n) a2 (k-1)
```

4. L'énoncé évoque deux invariants :
   - Dans toute paire $(a_i,\ k_i)$, l'arbre a_i est un arbre binaire complet d'ordre k_i (d'où au passage $k_i \geq 0$).
   - Les paires $(a_i,\ k_i)$ sont classées par ordres k_i croissants, avec tous les k_i différents.
5. Longueur logarithmique au maximum. En effet, comme tous les k_i sont différents, une liste de longueur n contient au minimum un dernier arbre d'ordre n, qui contient à lui seul 2^n éléments.
6. Ordre de grandeur au maximum logarithmique en le nombre d'éléments du tableau. En effet, cette fonction implique un parcours partiel de la liste (de longueur logarithmique), suivi d'un appel à `get_a` (de complexité proportionnelle à l'ordre k de l'arbre exploré, lui-même borné par $\log(n)$).
7. Après ajout de `1`, on a en tête un nouvel arbre d'ordre 0.



   Ajouter encore `0` apporte un nouvel arbre d'ordre 0, qui forme un arbre d'ordre 1 en fusionnant avec l'arbre d'ordre 0 déjà présent. L'arbre d'ordre 1 nouvellement formé fusionne lui-même avec l'arbre d'ordre 1 déjà présent, produisant un arbre d'ordre 2.



8. Le commentaire ligne 6 suggère que si $k \geq k'$ alors $k = k'$. Autrement dit, on suppose que lors d'un appel de la forme `merge (a, k) [(a1, k1); (a2, k2); ...; (an, kn)]` avec $n > 0$, on a $k \leq k_1$. Le code respecte bien cette précondition, si le tableau `t` donné en argument satisfait bien les invariants détaillés à la question 4 :

— L'appel initial (ligne 9) prend $k = 0$. Comme on suppose que les k_i sont positifs ou nuls, c'est vrai en particulier pour k_1.

— L'appel récursif (ligne 7) a la forme `merge (`a'`, `$k+1$`) [(`a_2`, `k_2`); ...; (`a_n`, `k_n`)]` avec l'hypothèse $k = k_1$. Comme on suppose que les k_i sont en ordre strictement croissant, on a en particulier $k_1 < k_2$, et donc $k + 1 = k_1 + 1 \leq k_2$.

Par ailleurs, ce code ne produit un tableau bien formé que si a est bien un arbre binaire complet d'ordre k. À nouveau, cette propriété est directement assurée ligne 9, et préservée lors de l'appel récursif ligne 7.

9. Meilleur cas : coût constant, avec ajout d'un unique arbre d'ordre 0 s'il n'y en a pas déjà un en tête. Pire cas : coût proportionnel à la longueur n de la liste, c'est-à-dire au pire logarithmique en le nombre d'éléments, si la liste contient des arbres de tous les ordres de 0 à $n - 1$ (ils sont alors tous fusionnés avec le nouvel élément pour former un unique arbre d'ordre n). En moyenne, coût constant (le meilleur cas arrive une fois sur 2, et une fusion de deux arbres d'ordre k n'arrive qu'une fois sur 2^{k+1}).

10. On retire la feuille la plus à gauche du premier arbre de la liste. Si cet arbre était d'ordre k, on va éclater les parties restantes en k arbres d'ordres 0 à $k - 1$, qui vont le remplacer en tête de la liste. Dans le code ci-dessous, cette décomposition est faite en plusieurs fois : on retire directement une feuille (supposée d'ordre 0), et dans le cas d'un nœud (supposé d'ordre non nul) on sépare les deux sous-arbres avant de poursuivre avec le sous-arbre gauche.

```
1  let rec remove = function
2    | []                                -> raise Not_found
3    | (F(_), k)       :: tl (* k=0 *) -> tl
4    | (N(a1, a2), k) :: tl (* k>0 *) -> remove ((a1, k-1) :: (a2, k-1) :: tl)
```

Complexité proportionnelle à l'ordre du premier arbre de la liste, donc dans le meilleur des cas constant, dans le pire des cas logarithmique en le nombre d'éléments, et en moyenne constant (comme pour l'ajout à la question précédente).

Correction exercice 2.

1. Il y a un seul mobile de poids 1, qui est `O(1)`. Un mobile de poids 2^{k+1} peut prendre deux formes :
   — soit le seul objet `O(`2^{k+1}`)` (compte pour 1),
   — soit une barre `O(`m_1`, `m_2`)` avec deux sous-mobiles m_1 et m_2 vérifiant `weight(`m_1`) + weight(`m_2`) =` 2^{k+1}. Si en outre le mobile est équilibré, alors `weight(`m_1`) = weight(`m_2`) =` $\frac{2^{k+1}}{2} = 2^k$. On a donc $\#_k$ possibilités pour chacun de ces deux sous-mobiles, c'est-à-dire $(\#_k)^2$ au total pour ce cas.

```
1  let rec count k =
2    if k = 0 then
3      1
4    else
5      let ck = count (k-1) in
6      1 + ck*ck
```

2. Démonstration par récurrence sur k.
   — Cas de base : $\#_0 = 1 \geq 1 = 2^0$.
   — Hérédité : soit $k \geq 0$ tel que $\#_k \geq 2^k$. Alors $\#_{k+1} = 1 + (\#_k)^2 \geq 1 + (2^k)^2 = 1 + 2^{2k}$. On conclut en raisonnant par cas sur k :
      — Si $k = 0$, alors $1 + 2^{2k} = 1 + 2^0 = 1 + 1 = 2 = 2^1 = 2^{k+1}$.
      — Si $k > 0$, alors $2^{2k} \geq 2^{k+1}$.

3. Démonstration par récurrence structurelle sur un mobile m.
   — Cas de base : tout mobile `O(`w`)` a une hauteur $h = 0$ et un poids $w \geq 1 = 2^0$.
   — Cas récursif : soient deux mobiles équilibrés m_1 de hauteur h_1 et de poids $w_1 \geq 2^{h_1}$ et m_2 et hauteur h_2 et de poids $w_2 \geq 2^{h_2}$. Supposons que `B(`m_1`, `m_2`)` est équilibré. Son poids est $w = w_1 + w_2$, avec par hypothèses de récurrence $w_1 \geq 2^{h_1}$ et $w_2 \geq 2^{h_2}$. Par équilibre, on a en outre $w_1 = w_2$. Donc $w = 2w_1 \geq 2 \times 2^{h_1}$ et $w = 2w_2 \geq 2 \times 2^{h_2}$. Autrement dit, $w \geq 2 \times 2^{\max(h_1,h_2)} = 2^{\max(h_1,h_2)+1}$. Par ailleurs, la hauteur h de `B(`m_1`, `m_2`)` est par définition $h = 1 + \max(h_1, h_2)$. D'où $w \geq 2^h$.

Correction exercice 3.

1. Le graphe (a) n'est pas régulier : il a des sommets de degré 2 et des sommets de degré 3. Le graphe (b) est 3-régulier.
2. La clique à k sommets est un graphe $(k-1)$-régulier.
3. Un triangle, un carré.
4. Dans un graphe k régulier à n sommets, la somme des degrés est kn. Le nombre d'arêtes est donc $kn/2$.
5. Non. Un tel graphe aurait $(9 \times 11)/2$ arêtes, ce qui n'est pas un nombre entier.
6. Prenons une composante connexe est un sommet s dans cette composante. Comme G est k-régulier, le sommet s a k voisins (qui sont bien différents car G est simple). La composante de s contient donc au moins $k+1$ sommets (s et ses voisins). Alors, si G n'était pas connexe on aurait au moins deux composantes connexes, et chacune aurait au moins $k+1$ sommets. C'est impossible car G n'a que $2k$ sommets.
7. Notons s l'extrémité de a qui n'est pas dans ρ. On propose le chemin

$$\rho' : s_i \longrightarrow s_{i+1} \longrightarrow \dots s_d \longrightarrow s_1 \longrightarrow \dots \longrightarrow s_i \xrightarrow{a} s$$

8. Du chemin ρ, on extrait un chemin allant de s_1 à s_i et un chemin allant de s_{i+1} à s_d. Ce deuxième va être emprunté à l'envers pour former le cycle.

$$s_1 \longrightarrow \dots \longrightarrow s_i \xrightarrow{b} s_d \longrightarrow s_{d-1} \longrightarrow \dots \longrightarrow s_{i+1} \xrightarrow{a} s_1$$

9. Si ce n'était pas le cas, on pourrait étendre le chemin ρ avec le voisin de s_1 ou de s_d qui n'est pas dans l'ensemble mentionné, et on contredirait ainsi la maximalité de ρ.
10. Avec la question précédente, on sait qu'il y a k indices i dans $\{s_2, \dots, s_d\}$ tels qu'il existe une arête $s_1 \longrightarrow s_i$. Autrement dit il existe k indices i dans $\{s_1, \dots, s_{d-1}\}$ tels qu'il existe une arête $s_1 \longrightarrow s_{i+1}$. Il existe de même k indices i dans $\{s_1, \dots, s_{d-1}\}$ tels qu'il existe une arête $s_i \longrightarrow s_d$. Or $\{s_1, \dots, s_{d-1}\}$ contient au maximum $2k-1$ sommets. Par principe des tiroirs nos deux ensembles d'indices ont donc un élément commun : il existe au moins un i tel qu'on a des arêtes $s_1 \longrightarrow s_{i+1}$ et $s_i \longrightarrow s_d$. On conclut alors avec la question 8.
11. Supposons que qu'un tel graphe n'ait pas de cycle hamiltonien. Considérons alors un chemin élémentaire $\rho : s_1 \longrightarrow \dots \longrightarrow s_d$ de longueur maximale. Par la question 10 on construit un cycle ρ'. Si ce cycle passe par tous les sommets alors on a un cycle hamiltonien : contradiction. Sinon, il existe au moins un sommet s hors du cycle. Comme G est connexe (question 6) il existe un chemin allant de s_1 à s. En prenant la première arête de ce chemin qui n'est pas dans ρ' on obtient une arête a qui a une extrémité dans ρ' et une extrémité hors de ρ'. Avec la question 7 on construit alors un chemin élémentaire plus long que ρ : contradiction.