Devoir Non Surveillé 4

À rendre le 13 mai 2024

INFORMATIQUE MP2I

Tournez la page S.V.P.

1 Partie I. Arbres de van Emde Boas

Soit p un entier positif et $N = 2^{2^p}$. On supposera que tous les entiers manipulés restent représentables par la structure d'entier OCaml ordinaire. On considère le type de structure suivant (appelé arbre *veb* par suite) implémentant un ensemble E d'entiers positifs strictement inférieurs à N.

```
type veb = {mutable mini : int; mutable maxi : int; table : veb array}
```

Les champs mutables d'un arbre *veb* son modifiables : par exemple, pour un arbre *veb* noté `v`, `v.mini <- 1` change la valeur du champ `v.mini`.
Le codage d'un ensemble $E \subset [\![0, N-1]\!]$ par un arbre *veb* dit d'ordre $N = 2^{2^p}$ suit les règles suivantes :

- Les champs `mini` et `maxi` représentent toujours la valeur minimale et maximale de E. Ils sont mis arbitrairement à -1 si l'ensemble est vide.
- Si $N = 2$, *i.e.* si l'arbre doit coder une partie de $[\![0, 1]\!]$, le champ `table` est un tableau vide (*i.e.* `[||]`), les champs `mini` et `maxi` suffisent à coder E. **On veillera à ce que les fonctions demandées traitent correctement ce cas particulier.**
- Pour $N > 2$, on note $\widehat{E} = E \setminus \{\min(E)\}$ (où $\min(E)$ désigne le minimum de E). On décompose $\widehat{E}$ en $\sqrt{N} = 2^{2^{p-1}}$ ensembles E_k définis par

$$\forall k \in [\![0, \sqrt{N}-1]\!], E_k = \left\{ x - k\sqrt{N} \;\middle|\; x \in \widehat{E} \cap [\![k\sqrt{N}, (k+1)\sqrt{N} - 1 \right\}$$

  Le champ `table` est alors un tableau de $\sqrt{N}+1$ arbres *veb* d'ordre $\sqrt{N}$:
  - pour $k \in [\![0, \sqrt{N}-1]\!]$, l'abre *veb* stocké dans la case `table.(k)` code l'ensemble E_k ;
  - l'abre *veb* stocké dans la case `table.`$(\sqrt{N})$ code l'ensemble $R = \left\{ k \in [\![0, \sqrt{N}-1]\!] \;\middle|\; E_k \neq \emptyset \right\}$

Par exemple, considérons l'arbre *veb* `ex` codant l'ensemble $E = \{2, 13, 14, 15\}$ avec $p = 2$ *i.e.* $N = 16$. On a $\widehat{E} = \{13, 14, 15\}$ puisque $\min(E) = 2$ et donc les cinq cases du tableau `ex.table` correspondent respectivement aux ensembles

$$E_0 = \emptyset \quad E_1 = \emptyset \quad E_2 = \emptyset \quad E_3 = \left\{13 - 3\sqrt{16}, 14 - 3\sqrt{16}, 15 - 3\sqrt{16}\right\} = \{1, 2, 3\} \quad R = \{3\}$$

On obtient la structure représentée en figure 1.

☐ 1 – On veut coder l'ensemble $\{0, 2, 3, 5, 7, 13, 14\}$ par un arbre *veb* d'ordre $N = 16$, noté `ex_veb`. Indiquer quel ensemble code chaque arbre *veb* stocké dans `ex_veb.table` puis préciser `ex_veb.table.(3)`.

☐ 2 – Écrire une fonction `creer_veb` de signature `int -> veb` prenant en argument un entier p et créant un arbre *veb* d'ordre $N = 2^{2^p}$ implémentant l'ensemble vide.

☐ 3 – Résoudre en fonction de q la récurrence $C(q) = C(\sqrt{q}) + \mathcal{O}(1)$ dans le cas où $q = 2^{2^s}$.

☐ 4 – Écrire une fonction `appartient_veb` de signature `veb -> int -> bool` qui teste l'appartenance d'un entier x quelconque à un ensemble E codé par un arbre *veb*. Calculer sa complexité dans le pire cas.

☐ 5 – Écrire une fonction `successeur_veb` de signature `veb -> int -> int` qui calcule le successeur d'un entier x quelconque dans E (-1 s'il n'y en a pas). Calculer sa complexité dans le pire cas.

☐ 6 – Écrire une fonction `insertion_veb` de signature `veb -> int -> unit` qui insère un entier x dans un arbre *veb*. On suppose que $x \in [\![0, N-1]\!]$. Cette fonction devra avoir une complexité en $\mathcal{O}(\log_2(\log_2(N)))$.

☐ 7 – Soit $M(N)$ la totalité de l'espace mémoire nécessaire pour stocker un ensemble par un arbre *veb* d'ordre N avec $N = 2^{2^p}$. Donner la relation de récurrence vérifiée par $M(N)$ puis déterminer l'ordre de grandeur de $M(N)$.

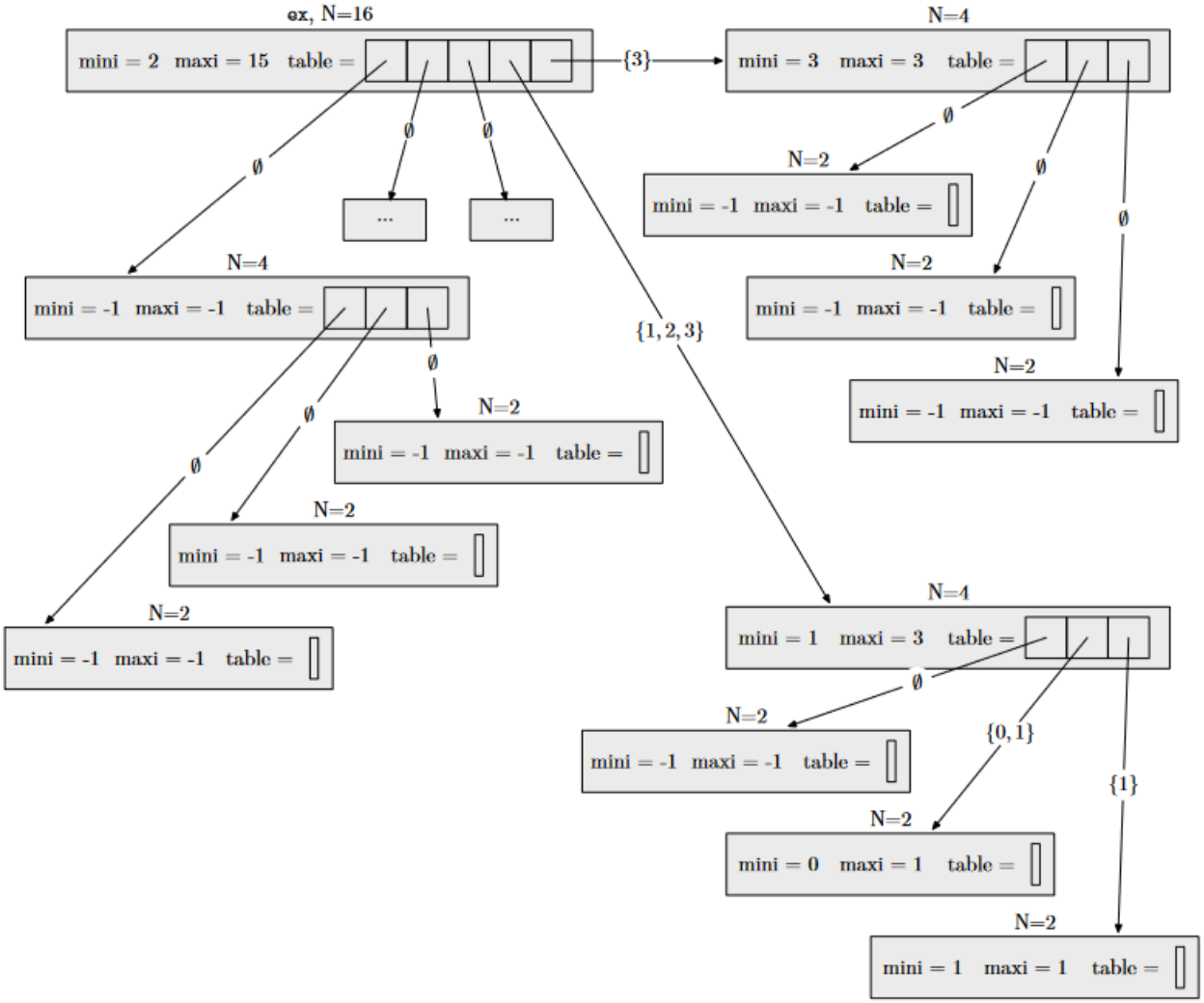

FIGURE 1 – L'arbre *veb* `ex` associé à l'ensemble $\{2, 13, 14, 15\}$ avec $p = 2$

Partie II. Diagrammes de décision binaires (OCaml) – 15 pts

On s'intéresse au diagnostique automatique d'une maladie à l'aide d'un diagramme de décision binaire. Un diagramme de décision binaire est un arbre binaire strict (ou localement complet) dans lequel les nœuds internes sont étiquetés par une question fermée, ici la présence ou non d'un symptôme, et les feuilles sont étiquetés par une décision, ici le diagnostique.

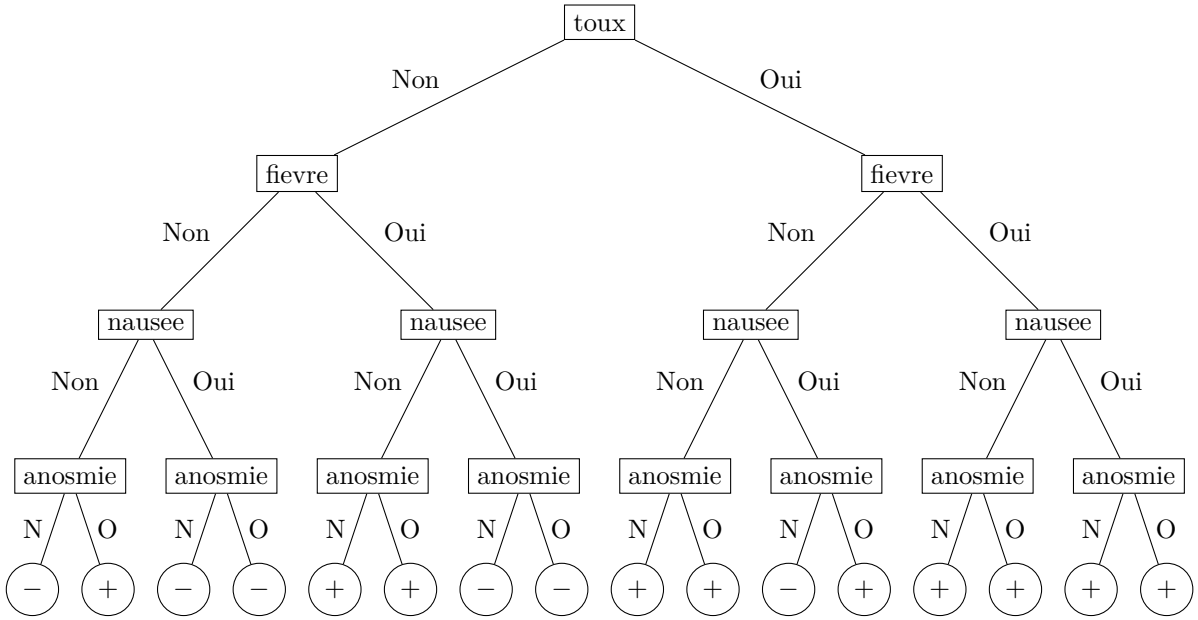

FIGURE 2 – Exemple d'arbre de décision binaire pour le diagnostique de la Covid-19.
Les diagnostiques possibles sont positif (+) ou négatif (-).

Dans un premier temps, on suppose que les symptômes à tester sont donnés dans le bon ordre, et on modélise en OCaml les diagrammes de décisions et les données des patients par les types suivants :

```
type patient = bool list

type bdd = Diagnostic of bool | Symptom of bdd * bdd
(* binary decision diagrams,
   Diagnostic true -> positif,
   Diagnostic false -> negative *)

let patient_zero = [true;false;false;true]
(* tousse, souffre d'anosmie mais pas de nausée ni de fièvre *)
```

☐ 8 – Écrire une fonction OCaml `len : 'a list -> int` qui calcule la taille d'une liste.

☐ 9 – Écrire une fonction OCaml `height : bdd -> int` qui calcule la hauteur d'un diagramme de décision binaire.

☐ 10 – Pour établir un diagnostique à l'aide d'un diagramme de décision binaire, on la parcours de la manière suivante :

- si le nœud est une feuille, alors le diagnostique est celui désigné par la feuille ;
- si le patient est touché par le symptôme à tester, alors la recherche continue dans le sous-arbre droit, sinon elle continue dans le sous-arbre gauche.

Quel est le diagnostique posé par l'arbre de la figure 2 pour le patient `patient_zero` ?

☐ 11 – Écrire une fonction OCaml `applique : bdd -> patient -> bool` qui applique un diagramme de décision binaire sur les données d'un patient pour donner un diagnostique. On utilisera une assertion pour vérifier que la liste de symptômes est assez longue pour appliquer correctement le diagramme de décision binaire.

☐ 12 – On souhaite diminuer la taille des diagrammes de décision à l'aide de la remarque suivante : si un nœud possède deux sous-arbres qui sont des feuilles de même diagnostique, on peut le remplacer par une feuille de ce diagnostique, comme illustré sur la figure 3.

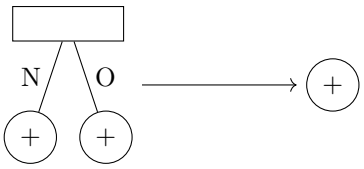

FIGURE 3 – Règle de réduction des diagrammes de décision binaire.

Donner l'arbre obtenu en appliquant cette réduction à l'arbre de la figure 2.

☐ 13 – Écrire une fonction OCaml `reduit : bdd -> bdd` qui met en œuvre cette réduction.

☐ 14 – Le fait de stocker les symptômes dans un ordre prédéfini restreint les diagrammes de décisions utilisables à ceux pour lesquels à une profondeur donnée, un seul symptôme est testé dans toutes les branches. On souhaite éviter cette restriction. Proposer une structure de données pour les données des patients ainsi que pour les diagrammes de décision binaire.

Question*. En supposant que la modification de la question précédente a été appliquée, on cherche à appliquer un diagramme de décision binaire sur des données partielles : lorsqu'une information sur un symptôme est manquante, on évalue le diagramme pour les deux possibilités : avec et sans ce symptôme, et si les deux informations concordent, on l'utilise, sinon on lève une exception. Écrire le code OCaml correspondant. On pourra se donner un module implémentant la structure de données de son choix en précisant les différentes opérations réalisables par celui-ci.

Partie III. Quelques opérations sur les arbres binaires (OCaml)

Dans cette section, on s'intéresse aux arbres binaires localement complets non vides dont les nœuds sont étiquetés par des éléments comparables, représentés en OCaml par le type suivant :

```
type 'a arbre = F of 'a | N of 'a arbre * 'a * 'a arbre;;
```

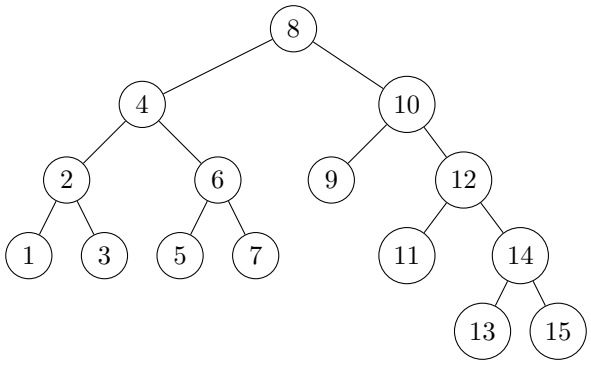

FIGURE 4 – Exemple d'arbre binaire localement complet non vide.

☐ 15 – Écrire une fonction `h` qui calcule la hauteur d'un arbre.

☐ 16 – Écrire une fonction `deepest` qui prend en entrée un arbre a et renvoie l'étiquette d'une feuille de profondeur maximale dans a. La solution proposée doit être de complexité dans le pire des cas linéaire en la taille de l'arbre.
Sur l'exemple de la figure 4, la fonction doit renvoyer 13 *ou* 15.
Indication : on pourra définir une fonction auxiliaire qui renvoie l'étiquette d'une feuille de profondeur maximale ainsi que sa profondeur.

☐ 17 – Écrire une fonction `median` qui prend en entrée un arbre a qui est un arbre binaire de recherche (trié de gauche à droite, ou encore pour un parcours infixe) et qui renvoie l'étiquette du nœud en position médiane. Quelle est sa complexité ? Comparer à la complexité de recherche de l'élément médian dans une liste chaînée ou dans un tableau trié.
Sur l'exemple de la figure 4, la fonction doit renvoyer 8.
Indication : on pourra définir en premier lieu une fonction qui renvoie l'élément en position k.

☐ 18 – Proposer une modification à la structure de données pour pouvoir trouver l'étiquette du nœud en position k avec une complexité dans le pire des cas linéaire en la hauteur de l'arbre.

☐ 19 – Écrire une fonction `highest` qui prend en entrée un arbre a et qui renvoie l'étiquette d'une feuille de profondeur minimale dans a. On cherchera une solution dont la complexité dépend uniquement de la profondeur de ce nœud.
Sur l'exemple de la figure 4, la fonction doit renvoyer 9.
Indication : On pourra faire appel aux fonctions `create`, `is_empty`, `push` et `pop` du module `Queue`.

FIN DE L'ÉPREUVE