Concours Centrale-Supélec, France

Session 2023, filière MP

Option informatique

Durée : 4 heures

Ce que traite le sujet

- Partie I : langages et automates
- Partie II : représentations classiques d'ensembles
- Partie III : représentation par arbres binaires complets
- Partie IV : arbres de van Emde Boas

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/centrale/mp/2023/info

Prepa Help ne revendique aucun droit sur le sujet, propriété de l'organisateur du concours.

Option informatique

CONCOURS CENTRALE•SUPÉLEC

MP

2023

4 heures

Calculatrice autorisée

Ce sujet comporte quatre parties.

La partie I est totalement indépendante des suivantes et étudie des transformations sur des langages.

Les autres parties s'intéressent à des structures de données représentant des ensembles d'entiers naturels E contenus dans $[\![0, N-1]\!]$, « denses » au sens où $|E|$ est du même ordre de grandeur que N. On cherche à optimiser le temps d'exécution des opérations de test d'appartenance, d'insertion ou de suppression d'un élément, de calcul du minimum, de calcul de l'élément de E immédiatement supérieur à x (qu'on appellera successeur de x dans E, même si x n'appartient pas à E) ou d'opérations ensemblistes telles que l'union.

Dans la partie II, on étudie des structures ordinaires dont on identifie le défaut. Dans la partie III, on étudie une structure simple d'arbre complet modélisant des parties de $[\![0, 2^p - 1]\!]$ et dans la partie IV, on implémente des arbres de van Emde Boas qui, sur des parties E de $[\![0, 2^{2^p} - 1]\!]$, donnent des opérations en temps quasi-constant.

I Langages et automates

Dans cette partie, on s'intéresse à des transformations sur des langages définis sur un alphabet à deux lettres $X = \{a, b\}$. On note $L_1|L_2$ la réunion de deux langages L_1 et L_2 et ε le mot vide (ou le langage réduit au mot vide, suivant le contexte). Les automates considérés sont des quadruplets (Q, I, F, Δ) où Q est l'ensemble des états de l'automate, I l'ensemble de ses états initiaux, F l'ensemble de ses états finals et $\Delta \subset Q \times X \times Q$ l'ensemble de ses transitions.

Soit L et K deux langages. On définit le langage noté $L \rhd K$ par

$$L \rhd K = \left\{ uvw \;\middle|\; (u, v, w) \in (X^*)^3, uw \in K, v \in L \right\}$$

Q 1. Déterminer les langages suivants :

$$a^* \rhd b^* \qquad (ba)^* \rhd (ab)^* \qquad a^* \rhd \left\{ a^p b a^q \;\middle|\; (p, q) \in \mathbb{N}^2, p \leqslant q \right\}$$

Q 2. Soit L, K_1 et K_2 trois langages. Exprimer à l'aide de $L \rhd K_1$, $L \rhd K_2$, K_1 et K_2 les langages suivants :

$$L \rhd (K_1|K_2) \qquad L \rhd (K_1.K_2) \qquad L \rhd (K_1^*)$$

On ne justifiera que la formule portant sur l'étoile de Kleene.

Q 3. Montrer que si L et K sont deux langages réguliers, $L \rhd K$ est régulier.

Q 4. Donner l'exemple d'un langage régulier L et d'un langage non régulier K tel que $L \rhd K$ soit régulier. On justifiera en détail que le langage K proposé n'est pas régulier.

Q 5. À l'aide d'automates reconnaissant des langages réguliers L et K et d'ε-transitions, construire un automate à ε-transitions reconnaissant $L \rhd K$. On expliquera la construction puis on formalisera l'automate correspondant. Il est inutile de justifier qu'il convient.

On définit sur X^* l'application σ par :

$$\begin{cases} \sigma(\varepsilon) = \varepsilon \\ \forall u \in X^*,\ \forall x \in X, \quad \sigma(ux) = xu \end{cases}$$

On note alors pour tout langage L, $\widehat{L}$ le langage défini par :

$$\widehat{L} = \bigcup_{k=0}^{+\infty} \sigma^k(L)$$

Q 6. Pour chacun des langages L suivants, déterminer $\sigma(L)$ puis $\widehat{L}$:

$$L_1 = (ab)^* \qquad L_2 = a^*(ba|b)$$

Q 7. Soit L un langage régulier et $\mathcal{A} = (Q, I, F, \Delta)$ un automate le reconnaissant. Pour $(q, q') \in Q^2$, on note $L_{q,q'}$ l'ensemble des mots qui étiquettent un chemin de l'état q à l'état q' dans $\mathcal{A}$. Exprimer $\sigma(L)$ à l'aide de langages $L_{q,q'}$ et du langage X. On portera une attention particulière au cas du mot vide et on justifiera la formule proposée.

BY-NC-SA

Q 8. Montrer que si L est régulier, $\sigma(L)$ est régulier.

Q 9. Montrer que si L n'est pas régulier, $\sigma(L)$ ne l'est pas non plus.

Q 10. Montrer que si L est régulier, $\widehat{L}$ est régulier. Étudier la réciproque.

II Représentations classiques d'ensembles

Dans cette partie, on implémente des ensembles par des structures connues. On note $|E|$ le cardinal d'un ensemble E.

II.A – Avec une liste triée

Q 11. Dans cette question uniquement, on implémente un ensemble d'entiers positifs par la liste de ses éléments, rangés **dans l'ordre croissant**. Écrire une fonction `succ_list` de signature `int list -> int -> int` prenant en arguments une liste d'entiers distincts dans l'ordre croissant et un entier x et renvoyant le successeur de x dans la liste, c'est-à-dire le plus petit entier strictement supérieur à x de la liste (-1 si cela n'existe pas). Donner sa complexité dans le pire cas.

II.B – Avec un vecteur trié

Soit N un entier naturel strictement positif, fixé pour toute cette partie. On choisit de représenter un ensemble d'entiers E de cardinal $n \leqslant N$ par un tableau `t` de taille $N+1$ dont la case d'indice 0 indique le nombre n d'éléments de E et les cases d'indices 1 à n contiennent les éléments de E rangés dans l'ordre croissant, les autres cases étant non significatives. Par exemple, le tableau `[|3;2;5;7;9;1;14|]` représente l'ensemble à 3 éléments $\{2,5,7\}$.

Q 12. Pour une telle implémentation d'un ensemble E, décrire brièvement des méthodes permettant de réaliser chacune des opérations ci-dessous (on ne demande pas d'écrire des programmes) et donner leurs complexités dans le pire cas :

— déterminer le maximum de E ;

— tester l'appartenance d'un élément x à E ;

— ajouter un élément x dans E (on suppose la taille du tableau suffisante et que x n'appartient pas à E).

Q 13. Par une méthode dichotomique, écrire une fonction `succ_vect` de signature `int array -> int -> int` prenant en arguments un tableau `t` codant un ensemble E comme ci-dessus et un entier `x` et renvoyant le successeur de `x` dans E (-1 si cela n'existe pas).

Q 14. Calculer la complexité dans le pire cas de la fonction `succ_vect` en fonction de n.

Q 15. Écrire une fonction `union_vect` de signature `int array -> int array -> int array` prenant en arguments deux tableaux `t_1` et `t_2`, de taille N, codant deux ensembles E_1 et E_2 et renvoyant le tableau correspondant à $E_1 \cup E_2$. On supposera que $|E_1 \cup E_2| \leqslant N$.

II.C – Avec un arbre binaire de recherche

On choisit maintenant de représenter un ensemble d'entiers à l'aide d'un arbre binaire de recherche, les nœuds étant étiquetés par les éléments de E. On définit le type

```
type abr = Nil | Noeud of  int * abr * abr
```

Pour un tel arbre, pour **tout** nœud x, les étiquettes de tous les nœuds du sous-arbre gauche de x, s'il y en a, doivent être inférieures à l'étiquette de x et les étiquettes de tous les nœuds du sous-arbre droit de x, s'il y en a, doivent être supérieures à l'étiquette de x.

Par exemple, on peut représenter l'ensemble $\{2,3,5,8,13\}$ par l'arbre figure 1.

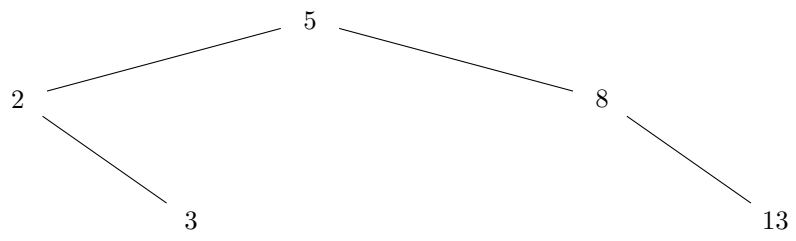


Figure 1 Un arbre binaire de recherche pour l'ensemble $\{2,3,5,8,13\}$

Q 16. Écrire une fonction `min_abr` de signature `abr -> int` prenant en argument un arbre binaire de recherche représentant un ensemble E et renvoyant son étiquette minimale (-1 si l'ensemble est vide).

Q 17. Écrire une fonction récursive `partitionne_abr` de signature `abr -> int -> (bool * abr * abr)` prenant en arguments un arbre binaire de recherche représentant un ensemble E et un entier x et renvoyant un

BY-NC-SA

triplet `(b, ag, ad)` où `b` vaut `true` si x appartient à E et `false` sinon, `ag` est un arbre binaire de recherche codant les éléments de E strictement plus petits que x et `ad` un arbre binaire de recherche codant les éléments de E strictement plus grands que x.

Q 18. Écrire une fonction `insertion_abr` de signature `abr -> int -> abr` prenant en arguments un arbre binaire de recherche représentant un ensemble E et un entier x et renvoyant un arbre binaire de recherche associé à l'ensemble $E \cup \{x\}$ et de racine étiquetée par x. Calculer sa complexité dans le pire cas en fonction de l'arbre reçu puis en fonction de E.

Q 19. Écrire une fonction `union_abr` de signature `abr -> abr -> abr` prenant en arguments deux arbres binaires de recherche représentant deux ensembles E_1 et E_2 et renvoyant un arbre binaire de recherche associé à l'ensemble $E_1 \cup E_2$. On expliquera brièvement la méthode choisie.

III Représentation par arbres binaires complets

On considère dans cette partie des arbres binaires complets dont les nœuds sont étiquetés par des booléens. On rappelle que la profondeur d'un nœud est égale à la longueur du chemin reliant la racine à ce nœud et la hauteur d'un arbre est la plus grande profondeur de ses feuilles (la racine est donc à la profondeur 0). Les nœuds seront numérotés à partir de la racine qui porte le numéro 1 dans l'ordre d'un parcours en largeur (de gauche à droite à chaque profondeur).

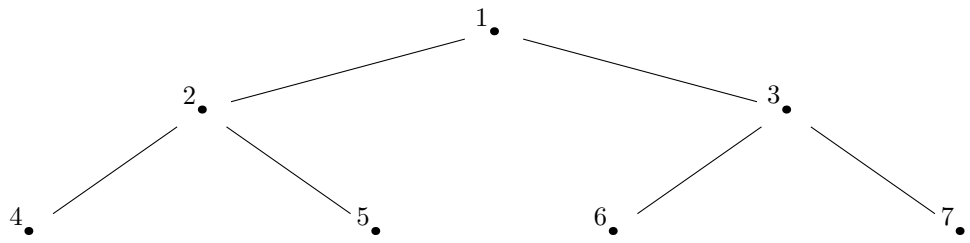


Figure 2 Numérotation des nœuds

Q 20. Dans un arbre binaire complet de hauteur $p \in \mathbb{N}$, quel numéro peut avoir un nœud à la profondeur $k \in [\![0, p]\!]$? Combien le sous-arbre dont la racine a le numéro i a-t-il de feuilles ?

Q 21. Dans un arbre binaire complet de hauteur $p \in \mathbb{N}$, pour le nœud numéro i, donner, en les justifiant, les numéros de son fils gauche, de son fils droit et de son père.

Informatiquement, un tel arbre complet de hauteur p sera implémenté par un tableau de booléens de taille 2^{p+1}, la case d'indice $i \in [\![1, 2^{p+1} - 1]\!]$ contenant l'étiquette du nœud numéroté i. La case d'indice 0 n'est pas utilisée. On notera que dans tous les programmes de cette partie, on peut avoir accès à la valeur 2^p via la taille du tableau.

Soit $p \in \mathbb{N}$. On choisit de coder un ensemble $E \subset [\![0, 2^p - 1]\!]$ par un arbre binaire complet de hauteur p selon les règles suivantes :

— chaque feuille numérotée i est étiquetée par `true` si et seulement si $i - 2^p \in E$;

— chaque nœud interne est étiqueté par le « ou logique » des étiquettes de ses deux fils.

```
let ensemble = bool array ;;
```

Par exemple, l'ensemble $\{0, 1, 7\}$ est représenté par l'arbre représenté figure 3.

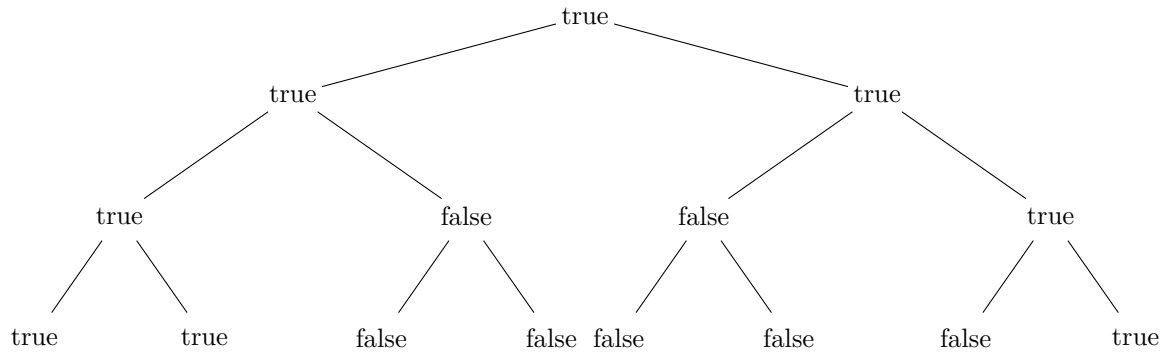


Figure 3 Arbre associé à l'ensemble $\{0, 1, 7\}$ pour $p = 3$

Q 22. Écrire une fonction `appartient` de signature `ensemble -> int -> bool` qui détermine si un entier quelconque appartient ou non à un ensemble donné. Calculer sa complexité.

BY-NC-SA

Q 23. Écrire une fonction `fabrique` de signature `int list -> int -> ensemble` qui prend en arguments une liste d'entiers positifs distincts ℓ et une valeur pour 2^p et renvoie l'arbre associé à l'ensemble E dont les éléments sont ceux de la liste. On supposera que tous les éléments de ℓ appartiennent à $[\![0, 2^p-1]\!]$. Cette fonction devra s'exécuter en $\mathcal{O}(2^p)$.

Q 24. Écrire une fonction `insere` de signature `ensemble -> int -> unit` qui ajoute un entier k à un ensemble E. On suppose k compatible avec la valeur de p associée à E. Cette fonction devra s'exécuter en $\mathcal{O}(1)$ dans le meilleur cas.

Q 25. Écrire une fonction `supprime` de signature `ensemble -> int -> unit` qui retire un entier k d'un ensemble E. On suppose k compatible avec la valeur de p associée à E. Calculer la complexité de cette fonction dans le pire cas.

Q 26. Écrire une fonction `minlocal` de signature `ensemble -> int -> int` qui cherche l'élément de E minimal parmi ceux codés dans le sous-arbre de racine numérotée i dans l'arbre associé à E. Si un tel élément n'existe pas, cette fonction devra renvoyer -1. Calculer la complexité de cette fonction en fonction de p et i.

On veut maintenant écrire une fonction qui calcule le successeur d'un entier x dans un ensemble E pour une telle structure. On propose l'algorithme suivant, pour $x \in [\![0, 2^p-1]\!]$:

— on part de la case numéro i codant l'entier x dans E ;

— tant que i n'est pas le nœud le plus à droite à sa profondeur et que la case $i+1$ vaut `false`, on remplace i par son père ;

— on renvoie l'élément minimum du sous-arbre de racine $i+1$ ou -1 si i était totalement à droite.

Q 27. Prouver l'algorithme décrit ci-dessus.

Q 28. Écrire une fonction `successeur` de signature `ensemble -> int -> int` prenant en arguments l'ensemble E et un entier x positif et renvoyant son successeur dans E (-1 si cela n'existe pas).

Q 29. Montrer que si $x \in E$ admet bien un successeur dans E, il existe une constante $K > 0$ indépendante de E et p telle que la complexité de `successeur e x` soit majorée par $K\big(\log_2(\texttt{successeur}(x) - x) + 2\big)$. On admet que le même type de justification montre que si x est le maximum de E, la complexité de `successeur e x` est majorée par $K\big(\log_2(2^p - x) + 2\big)$.

Q 30. **En utilisant la fonction `successeur`**, écrire une fonction `cardinal` de signature `ensemble -> int` prenant en argument un ensemble et renvoyant son cardinal.

Q 31. Déterminer la complexité de la fonction `cardinal` en fonction de p et $n = |E|$. On rappelle que la fonction $\log_2$ est concave.

Q 32. Quels sont les intérêts et inconvénients d'une telle structure ? Dans quels cas peut-elle s'avérer plus intéressante que des structures connues ?

IV 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* sont 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` suffisant à coder E. **On veillera à ce que les fonctions demandées traitent correctement ce cas particulier.**

— Pour $N > 2$, on note $\widehat{E} = E\backslash\{\min(E)\}$ (où $\min(E)$ désigne le minimum de E). On décompose $\widehat{E}$ en $\sqrt{N} = 2^{2^{p-1}}$ ensembles E_k définis par

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

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'arbre *veb* stocké dans la case `table.(k)` code l'ensemble E_k ;
- l'arbre *veb* stocké dans la case `table.`$(\sqrt{N})$ code l'ensemble $R = \left\{ k \in [\![0, \sqrt{N}-1]\!] \;\middle|\; E_k \neq \emptyset \right\}$.

BY-NC-SA

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

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

On obtient la structure représentée figure 4.

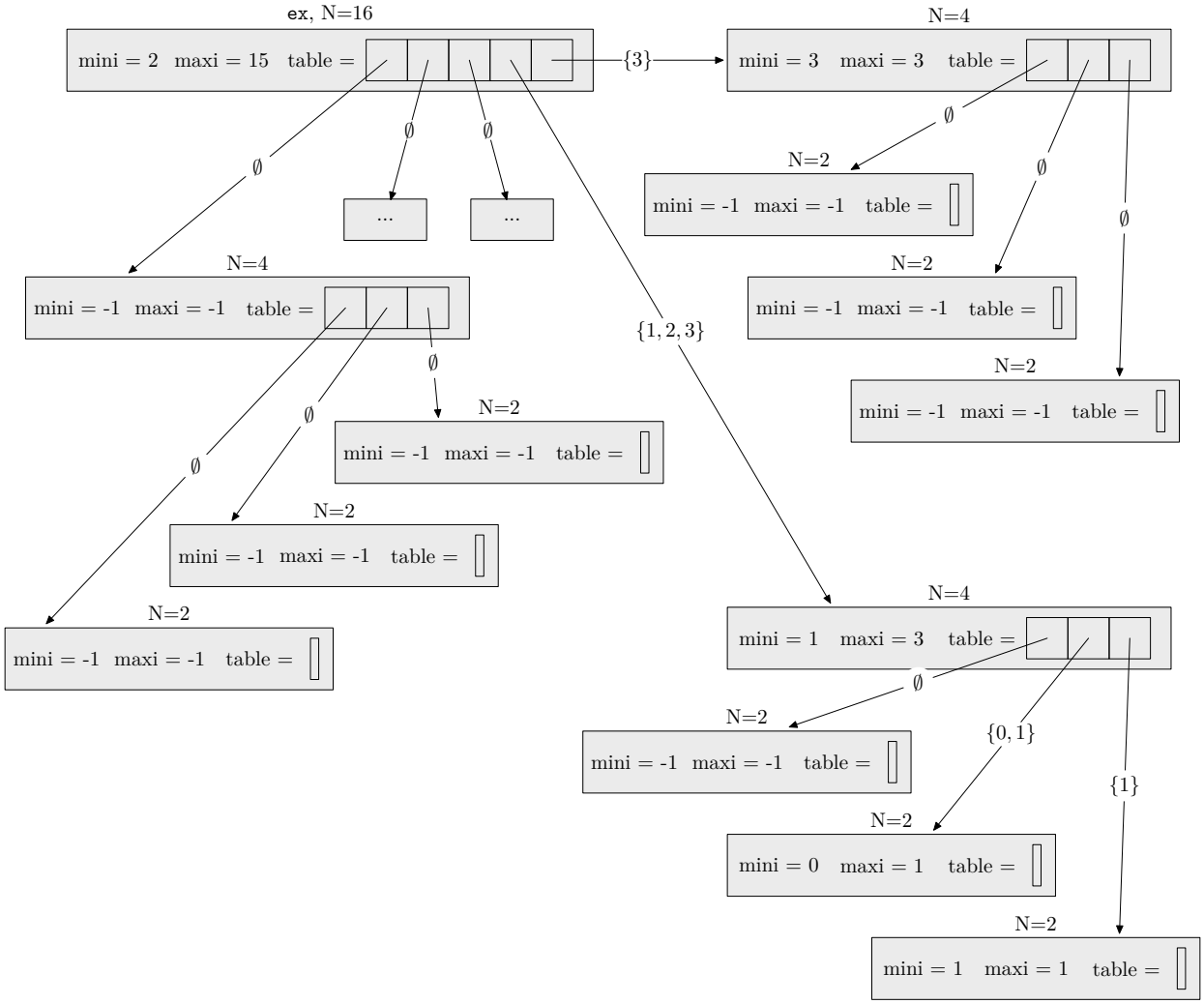


Figure 4 L'arbre *veb* `ex` associé à l'ensemble $\{2, 13, 14, 15\}$ avec $p = 2$

Q 33. 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)`.

Q 34. 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.

Q 35. Résoudre en fonction de q la récurrence $C(q) = C(\sqrt{q}) + \mathcal{O}(1)$ dans le cas où $q = 2^{2^s}$.

Q 36. 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.

Q 37. 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.

Q 38. 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 $\mathcal{O}(\log_2(\log_2(N))$.

Q 39. 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 $M(N)$.

• • • FIN • • •

BY-NC-SA

Travailler les chapitres de ce sujet

Prepa Help suit le programme de MP chapitre par chapitre : cours détaillé, méthodes, pièges, exercices et problèmes corrigés pas à pas, tirés des concours. L'essai gratuit donne accès aux chapitres offerts de ta filière, sans carte bancaire.

Chapitres liés à ce sujet

- Langage OCaml : expressions, types, filtrage, récursivité
- Complexité des algorithmes itératifs et récursifs
- Récursivité
- Tri par fusion et tri rapide
- Logique, ensembles et applications
- Dérivabilité et convexité

Ouvrir l'essai gratuit

Scanne le code ou clique sur le bouton.

prepahelp.com/#essai

Une question sur le concours ou sur ta filière : **écrire sur WhatsApp**

Les autres années et épreuves : prepahelp.com/annales/centrale/mp