

La sélection du $(k + 1)^{\text{e}}$ plus petit élément d'une liste d'entiers L , non nécessairement triée, consiste à trouver le $(k + 1)^{\text{e}}$ élément de la liste obtenue en triant L dans l'ordre croissant.

Par exemple, si $L = [9; 1; 2; 4; 7; 8]$ le 3^e plus petit élément de L est 4. On pourra remarquer que si la liste L est triée dans l'ordre croissant, le $(k + 1)^{\text{e}}$ plus petit élément est l'élément de rang k dans L .

On présente un algorithme permettant de résoudre ce problème de sélection avec une complexité temporelle linéaire dans le pire cas. Celui-ci est basé sur le principe de "diviser pour régner" et sur le choix d'un bon pivot pour partager la liste en deux sous-listes.

Dans cette partie, les fonctions demandées sont à écrire en OCaml et ne doivent faire intervenir aucun trait impératif du langage (références, tableaux ou autres champs mutables ou exception par exemple).

Étant donné un réel a , on note $\lfloor a \rfloor$ le plus grand entier inférieur ou égal à a .

I.1 - Fonctions utiles

Dans cette section, on écrit des fonctions auxiliaires qui sont utiles pour la fonction principale.

Q1. Écrire une fonction récursive de signature :

`longueur : 'a list -> int`

et telle que `longueur l` est la longueur de la liste `l`.

Q2. Écrire une fonction récursive de signature :

`insertion : 'a list -> 'a -> 'a list`

et telle que `insertion l a` est la liste triée dans l'ordre croissant obtenue en ajoutant l'élément a dans la liste croissante `l`.

Q3. En déduire une fonction récursive de signature :

`tri_insertion : 'a list -> 'a list`

et telle que `tri_insertion l` est la liste obtenue en triant `l` dans l'ordre croissant.

Q4. Écrire une fonction récursive de signature :

`selection_n : 'a list -> int -> 'a`

et telle que `selection_n l n` est l'élément de rang `n` de la liste `l`.

Par exemple, `selection_n [4;2;6;4;1;15] 3` est égal à 4.

Q5. Écrire une fonction récursive de signature :

`paquets_de_cinq : 'a list -> 'a list list`

et telle que `paquets_de_cinq l` est une liste de listes obtenue en regroupant les éléments de la liste `l` par paquets de cinq sauf éventuellement le dernier paquet qui est non vide et qui contient au plus cinq éléments. Par exemple :

- `paquets_de_cinq []` est égal à `[]`,
- `paquets_de_cinq [2;1;2;1;3]` est égal à `[[2;1;2;1;3]]`,
- `paquets_de_cinq [3;4;2;1;5;6;3]` est égal à `[[3;4;2;1;5]; [6;3]]`.

Q6. Écrire une fonction récursive de signature :

`medians : 'a list list -> 'a list`

et telle que `medians l` est la liste `m` obtenue en prenant dans chaque liste l_k apparaissant dans la liste de listes `l` l'élément médian de l_k . On convient que pour une liste A dont les éléments sont exactement $a_0 \leq a_1 \leq \dots \leq a_{n-1}$, l'élément médian désigne $a_{\lfloor \frac{n}{2} \rfloor}$.

Dans le cas où la liste L n'est pas triée, l'élément médian désigne l'élément médian de la liste obtenue en triant L par ordre croissant. Par exemple :

`medians [[3;1;5;3;2]; [4;3;1]; [1;3]; [5;1;2;4]]` est égal à `[3;3;1;2]`.

Q7. Écrire une fonction de signature :

`partage : 'a -> 'a list -> 'a list * 'a list * int * int`

telle que `partage p l` est un quadruplet `l1,l2,n1,n2` où `l1` est la liste des éléments de `l` plus petit que `p`, `l2` est la liste des éléments de `l` strictement plus grand que `p`, `n1` et `n2` sont respectivement les longueurs de `l1` et `l2`.

1.2 - La fonction de sélection et sa complexité

On détaille la fonction de sélection :

Q8. Écrire une fonction récursive de signature :

`selection : 'a list -> int -> 'a`

telle que `selection l k` est le $(k+1)^{\text{e}}$ plus petit élément de la liste `l`. L'écriture de la fonction sera une traduction en OCaml de l'Algorithme 1 présenté en page 4.

On cherche à déterminer la complexité en nombre de comparaisons de la fonction `selection`. Pour tout $n \in \mathbb{N}$, on note $T(n)$ le nombre maximum de comparaisons entre éléments lors d'une sélection d'un élément quelconque dans des listes L **sans répétition** de taille n .

En analysant l'Algorithme 1, il est possible de démontrer que :

$$\forall n \geq 55, T(n) \leq T\left(\left\lfloor \frac{n+4}{5} \right\rfloor\right) + T\left(\left\lfloor \frac{8n}{11} \right\rfloor\right) + 4n. \quad (I)$$

Q9. En admettant la proposition (I), montrer que pour tout entier n supérieur à 1, on a :

$$T(n) \leq (200 + T(55))n.$$

Pour l'initialisation, on pourra remarquer que T est une fonction croissante.

Algorithme 1 - Sélection du $(k + 1)^e$ plus petit élément

```
1 SELECTION L k :
  /* L est une liste, k est un entier positif */
2 début
3    $n \leftarrow \text{LONGUEUR } L$ 
4   si  $n \leq 5$  alors
5      $M \leftarrow (\text{TRI\_INSERTION } L)$ 
6     retourner l'élément de rang k de M
7   fin
8   sinon
9      $L\_Cinq \leftarrow \text{PAQUETS\_DE\_CINQ } L$ 
10     $M \leftarrow \text{MEDIANS } L\_Cinq$ 
11     $\text{pivot} \leftarrow \text{SELECTION } M \ ((n + 4) // 5) // 2$ 
12    /* L'opérateur // désigne le quotient d'entiers. Le rang  $((n + 4) // 5) // 2$ 
13       correspond au rang du médian de la liste M */
14     $L_1, L_2, n_1, n_2 \leftarrow \text{PARTAGE pivot } L$ 
15    si  $k < n_1$  alors
16      retourner SELECTION  $L_1$  k
17    fin
18    sinon
19      retourner SELECTION  $L_2$   $(k - n_1)$ 
20    fin
21  fin
```

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 $n \leq N$ 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 $|E_1 \cup E_2| \leq N$.

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.

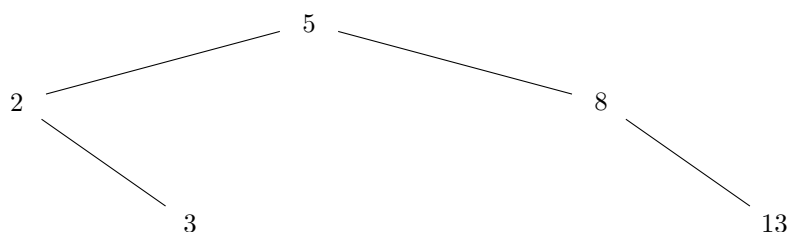


Figure 1 Un arbre binaire de recherche pour l'ensemble $\{2, 3, 5, 8, 13\}$

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

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).

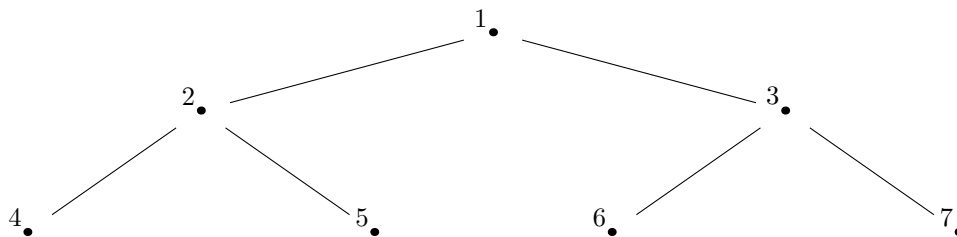


Figure 2 Numérotation des nœuds

Q 20. Dans un arbre binaire complet de hauteur $p \in \mathbb{N}$, quel numéro peut avoir un nœud à la profondeur $k \in \llbracket 0, p \rrbracket$? 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 $i \in \llbracket 1, 2^{p+1} - 1 \rrbracket$ 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 $E \subset \llbracket 0, 2^p - 1 \rrbracket$ 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.

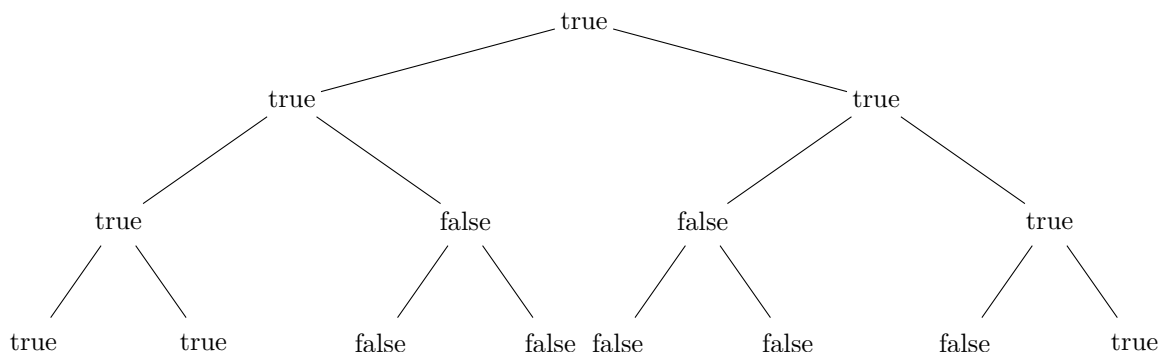


Figure 3 Arbre associé à l'ensemble $\{0, 1, 7\}$ pour $p = 3$

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.

- 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 $\llbracket 0, 2^p - 1 \rrbracket$. 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 $x \in \llbracket 0, 2^p - 1 \rrbracket$:
- 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 $K(\log_2(\text{successeur}(x) - x) + 2)$. 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 $K(\log_2(2^p - x) + 2)$.
- 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 $E \subset \llbracket 0, N - 1 \rrbracket$ 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 $\llbracket 0, 1 \rrbracket$, 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 $\widehat{E} = E \setminus \{\min(E)\}$ (où $\min(E)$ désigne le minimum de E). On décompose \widehat{E} en $\sqrt{N} = 2^{2^{p-1}}$ ensembles E_k définis par

$$\forall k \in \llbracket 0, \sqrt{N} - 1 \rrbracket, \quad E_k = \left\{ x - k\sqrt{N} \mid x \in \widehat{E} \cap \llbracket k\sqrt{N}, (k+1)\sqrt{N} - 1 \rrbracket \right\}$$

le champ `table` est alors un tableau de $\sqrt{N} + 1$ arbres *veb* d'ordre \sqrt{N} :

- pour $k \in \llbracket 0, \sqrt{N} - 1 \rrbracket$, 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 $R = \left\{ k \in \llbracket 0, \sqrt{N} - 1 \rrbracket \mid E_k \neq \emptyset \right\}$.

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

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

On obtient la structure représentée figure 4.

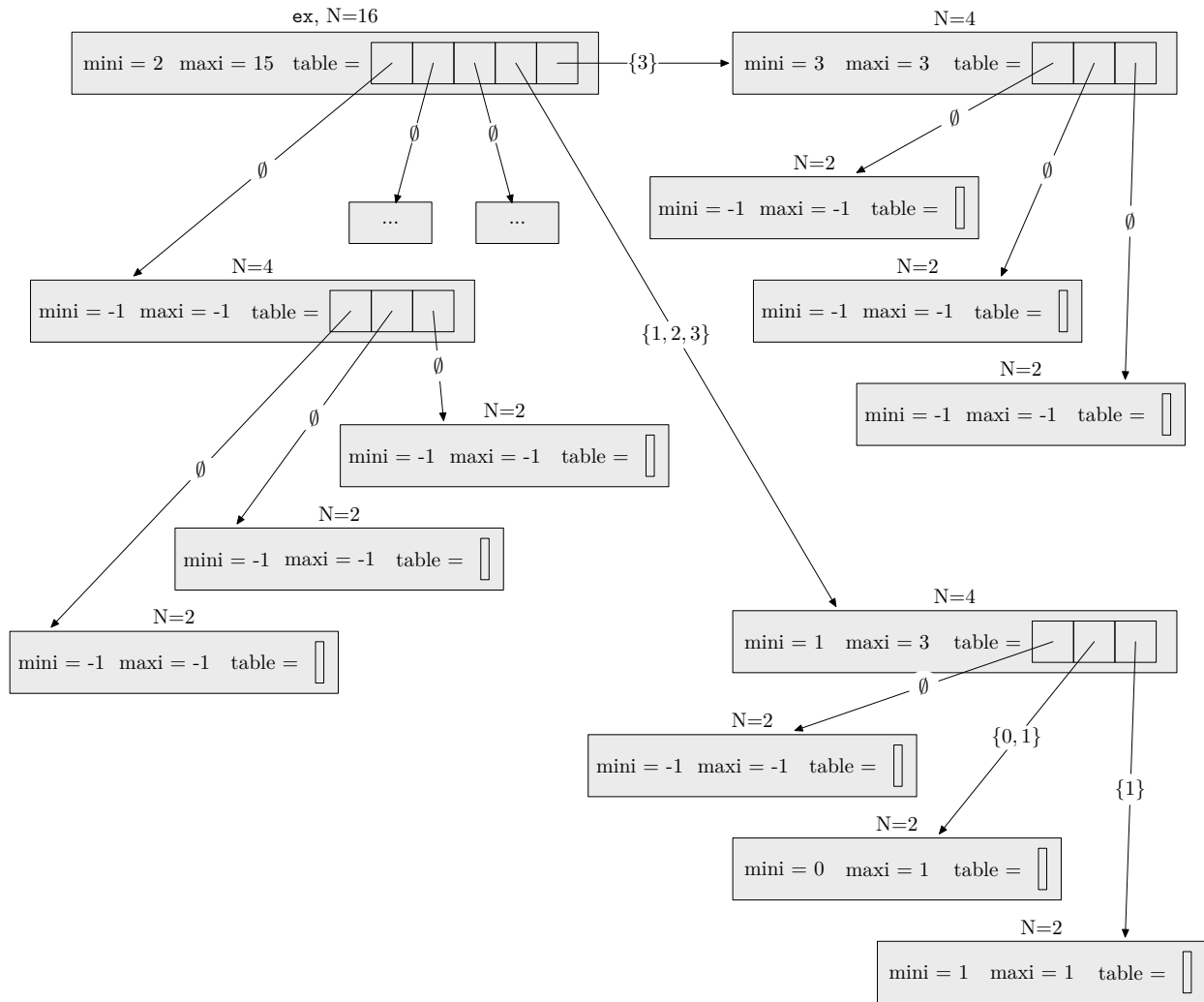


Figure 4 L'arbre *veb* **ex** associé à l'ensemble $\{2, 13, 14, 15\}$ avec $p = 2$

- 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 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 $x \in \llbracket 0, N-1 \rrbracket$. 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 • • •