Devoir en temps limité n°5 – 4h

Calculatrices interdites

On veillera à présenter très clairement sa copie : il faut rédiger les réponses et encadrer les résultats. Pour le code, il doit être indenté, on ne commence pas une fonction en bas de page et on utilise de la couleur pour les commentaires.

Le code doit être commenté dès qu'il dépasse les 5 lignes.

Les fonctions en C et en Ocaml doivent avoir le type précisé. Il est donc recommandé d'utiliser des fonctions auxiliaires en Ocaml.

1 Un peu de hachage

Dans cette partie comme dans le cours, on note m la taille du tableau utilisé par notre table de hachage.

1. Qu'est-ce qu'une fonction de hachage ?

On veut désormais créer une table de hachage dont les clés sont des chaines de caractères et les valeurs sont des entiers.

On crée un type `couple` pour stocker les couples (clé, valeur).

```
struct couple{
   char* cle;
   int valeur;
};
typedef struct couple couple;
```

La table de hachage sera donc un tableau de couples, de type `couple*`.

On suppose de plus qu'on dispose d'une fonction de hachage `int h1(char* cle)` et que notre table fonctionne avec un seul hachage, sans amélioration (pas d'adressage ouvert, ...)

2. Écrire une fonction `int recherche(couple* table, char* cle)` qui recherche la cle dans la table et renvoie la valeur associée. Si on ne trouve pas la clé, on fera une erreur avec un `assert`.
3. En supposant que la fonction de hachage s'effectue en temps constant, quelle est la complexité de votre fonction `recherche` ?

2 Logique

Exercice tiré et adapté du sujet CCINP option informatique 2017

Imaginez-vous ethnologue. Vous étudiez une peuplade primitive qui présente un comportement manichéen extrême : lorsque plusieurs personnes participent à une même conversation sur un sujet donné, elles vont toutes avoir le même comportement manichéen tant que la conversation reste sur le même sujet, c'est-à-dire que toutes les affirmations seront soit des vérités, soit des mensonges. Par contre, si le sujet de la conversation change, la nature des affirmations, soit mensonge, soit vérité, peut changer, mais toutes les affirmations seront de la même nature tant que le sujet ne changera pas à nouveau.

Pour être autorisé à séjourner dans cette peuplade, vous devez respecter cette règle. Vous participez à une conversation avec trois de leurs membres que nous appellerons X, Y et Z. Ceux-ci vous indiquent comment rejoindre leur village. Si vous n'arrivez pas à le rejoindre, vous ne serez pas autorisé à y séjourner.

Le premier sujet abordé est la région dans laquelle se trouve le village :

X indique : « Le village se trouve dans la vallée » ;

Z réplique : « Non, il ne s'y trouve pas » ;

X reprend : « Ou alors dans les collines ».

Nous noterons V et C les variables propositionnelles associées à la région dans laquelle se trouve le village.

Nous noterons X_1 et Z_1 les formules propositionnelles correspondant aux affirmations de X et de Z sur le premier sujet.

Puis, le second sujet est abordé : le chemin qui permet de rejoindre le village dans la région concernée.

X dit : « Le chemin de gauche conduit au village » ;

Z répond : « Tu as raison » ;

X complète : « Le chemin de droite y conduit aussi » ;

Y affirme : « Si le chemin du milieu y conduit, alors celui de droite n'y conduit pas » ;

Z indique : « Celui du milieu n'y conduit pas ».

Nous noterons G, M, D les variables propositionnelles correspondant respectivement au fait que le chemin de gauche, du milieu et de droite, conduit au village.

Nous noterons X_2, Y_2 et Z_2 les formules propositionnelles correspondant aux affirmations de X, de Y et de Z sur le second sujet.

4. Rappeler la définition de l'ensemble des formules propositionnelles.
5. Représenter le comportement manichéen des interlocuteurs dans le premier sujet abordé sous la forme d'une formule ϕ du calcul des propositions dépendant des formules propositionnelles X_1 et Z_1 .
6. Représenter les informations données par les participants sous la forme de deux formules du calcul des propositions X_1 et Z_1 dépendant des variables V et C.
7. En utilisant des équivalents sémantiques, simplifier ϕ et déterminer dans quelle région vous devez vous rendre pour rejoindre le village.
   Dans cette question, on n'utilisera ni tables de vérités, ni raisonnement mathématiques, que des équivalents sémantiques classiques et du calcul.
8. Représenter le comportement manichéen des interlocuteurs dans le second sujet abordé sous la forme d'une formule du calcul des propositions ψ dépendant des formules propositionnelles X_2 , Y_2 et Z_2 .
9. Représenter les informations données par les participants sous la forme de trois formules du calcul des propositions X_2 , Y_2 et Z_2 dépendant des variables G, M et D.
10. En faisant une table de vérité, déterminer quel chemin vous devez suivre pour rejoindre le village.
    Dans la table de vérité, on mettra les variables et formules dans l'ordre G, M, D, X_2, Y_2, Z_2, ψ.
    Dans cette question, on n'utilisera ni équivalents sémantiques, ni raisonnement mathématiques, que une (ou des si pas la place) table de vérité
11. En admettant que les trois participants aient menti, pouviez-vous prendre d'autres chemins ? Si oui, le ou lesquels ?

3 Représentations classiques d'ensembles

Exercice tiré et adapté du sujet Centrale option informatique 2023

Dans les parties qui suivent, on implémente des ensembles par des structures connues. On note $|E|$ le cardinal d'un ensemble E. On rappelle que dans un ensemble, **un élément ne peut pas apparaitre 2 fois**.

Attention : certaines parties sont en C et d'autre en Ocaml, soyez attentifs

1. Avec une liste triée (Ocaml)

12. 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 : 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 (ou -1 si cela n'existe pas).
13. Donner sa complexité dans le pire cas.

2. Avec un tableau trié (C)

Soit N un entier naturel strictement positif, fixé pour toute cette partie et qui est une variable globale. On choisit de représenter un ensemble d'entiers E de cardinal $n < N$ 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 avec les valeurs `[3,2,5,7,9,1,14]` représente l'ensemble à 3 éléments $\{2, 5, 7\}$.

14. 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).

15. Par une méthode dichotomique, écrire une fonction `int succ_tab(int* t, int x)` 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).
16. Calculer la complexité dans le pire cas de la fonction `succ_tab` en fonction de n.
17. Écrire une fonction `int* union_tab(int* t_1, int* t_2)` 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 $|E_1 \cup E_2| \leq N$.

3. Avec des arbres binaires de recherche (Ocaml)

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 = Vide | Noeud of abr * int * abr;;
```

18. Rappeler la propriété d'ordonnancement existant sur les étiquettes d'un ABR, sachant qu'on considèrera dans la suite qu'une étiquette ne peut apparaitre plusieurs fois dans l'ABR.
19. Dessiner un ABR pour l'ensemble de valeurs $\{2, 3, 5, 8, 13\}$.
20. Écrire une fonction `min_abr : abr -> int` prenant en argument un ABR représentant un ensemble E et renvoyant son étiquette minimale (-1 si l'ensemble est vide).
21. Écrire une fonction récursive `partitionne_abr : 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`.
22. Écrire une fonction `insere_racine_abr : 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.

    On remarquera qu'il ne s'agit pas de la méthode d'insertion vue en cours, qui ne met pas la nouvelle étiquette à la racine.
23. Écrire une fonction `union_abr : 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.

4 Représentation par des arbres binaires complets (C)

On considère dans cette partie des arbres binaires complets dont les nœuds sont étiquetés par des booléens. 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). La Figure 1 donne un exemple de cette numérotation.

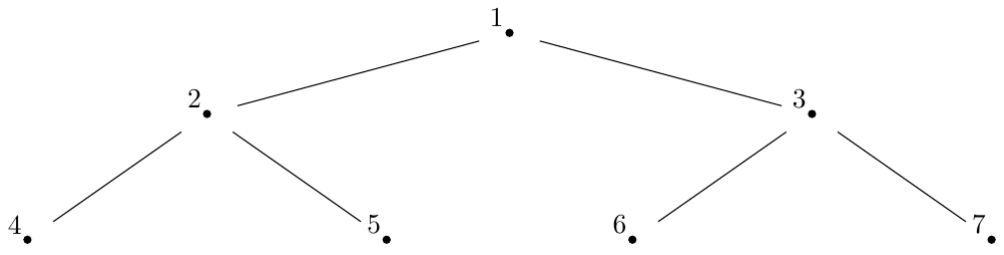

Figure 1 – Numérotation des noeuds

24. Dans un arbre binaire complet de hauteur $p \in \mathbb{N}$, quel numéro peut avoir un noeaud à la profondeur $k \in [|0, p|]$? Combien le sous-arbre dont la racine est i a-t'il de feuilles ? Justifier.
25. 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.

Par exemple, l'ensemble $\{0, 1, 7\}$ est représenté par l'arbre représenté Figure 2.

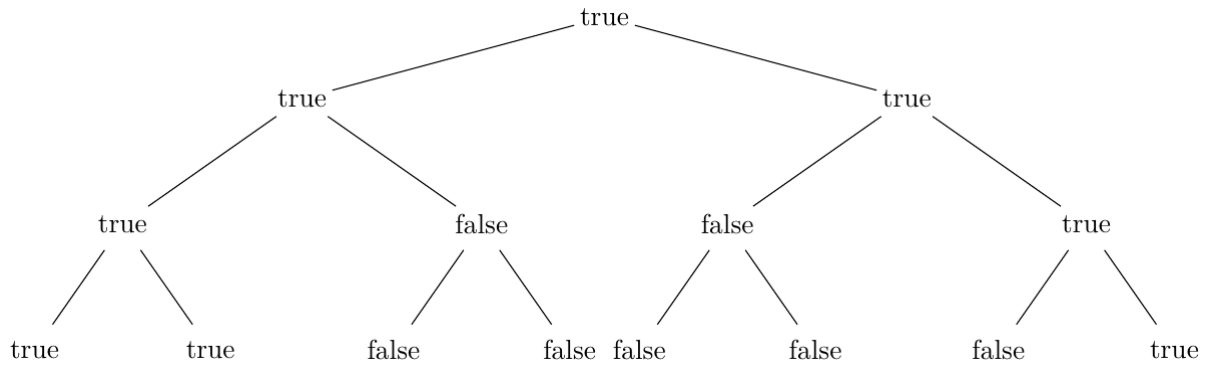

FIGURE 2 – Arbre associé à l'ensemble $\{0, 1, 7\}$ pour $p = 3$

26. Écrire une fonction `bool appartient(bool* ens, int taille, int x)` qui détermine si un entier `x` quelconque appartient ou non à un ensemble `ens` donné de taille `taille`. Calculer sa complexité.
27. Écrire une fonction `bool* fabrique(int* t, int n, int ppuis2)` qui prend en arguments un tableau d'entiers positifs distincts t de taille n et une valeur pour 2^p et renvoie l'arbre associé à l'ensemble E dont les éléments sont ceux du tableau. On supposera que tous les éléments de t appartiennent à $[|0, 2^p - 1|]$. Cette fonction devra s'exécuter en $O(2^p)$.
28. Écrire une fonction `void insere(bool* ens, int taille, int k)` 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 $O(1)$ dans le meilleur cas.
29. Écrire une fonction `void supprime(bool* ens, int taille, int k)` 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.
30. Écrire une fonction `int minlocal(bool* ens, int taille, int i)` 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 en 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.

31. Prouver l'algorithme décrit ci-dessus.
32. Écrire une fonction `int successeur(bool* ens, int taille, int x)` prenant en arguments l'ensemble E et un entier x positif et renvoyant son successeur dans E (-1 si cela n'existe pas).
33. 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(ens, taille, x)` soit majorée par $K(log_2(\texttt{successeur(ens, ptaille,x)} - x) + 2)$. On admet que le même type de justification montre que si x est le maximum de E, la complexité de `successeur(ens, taille, x)` est majorée par $K(log_2(2^p - x) + 2)$.
34. **En utilisant la fonction** `successeur`, écrire une fonction `int cardinal(bool* ens, int taille)` prenant en argument un ensemble et renvoyant son cardinal.
35. Déterminer la complexité de la fonction `cardinal` en fonction de p et $n = |E|$. On rappelle que la fonction log_2 est concave.
36. Quels sont les intérêts et inconvénients d'une telle structure ? Dans quels cas peut-elle s'avérer plus intéressante que des structures connues ?

5 Arbres de van Emde Boas (Ocaml)

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 la 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 $\widehat{E} = E \setminus \{min(E)\}$ (où $min(E)$ désigne le minimum de E). On décompose $\widehat{E}$ en $\sqrt{N} = 2^{2^{p-1}}$ ensembles E_k définis par

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

  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 $R = \{k \in [|0, \sqrt{N}-1|] | E_k \neq 0\}$.

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 en Figure 3.

37. 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)`.
38. Écrire une fonction `creer_veb : int -> veb` prenant en argument un entier p et créant un arbre *veb* d'ordre $N = 2^{2^p}$ implémentant l'ensemble vide.
39. Résoudre en fonction de q la récurrence $C(q) = C(\sqrt{q}) + O(1)$ dans le cas où q = 2^{2^s}.
40. Écrire une fonction `appartient_veb : 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.
41. Écrire une fonction `successeur_veb : 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.
42. Écrire une fonction `insertion_veb : 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 $O(log_2(log_2(N)))$.
43. 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 $M(N)$.

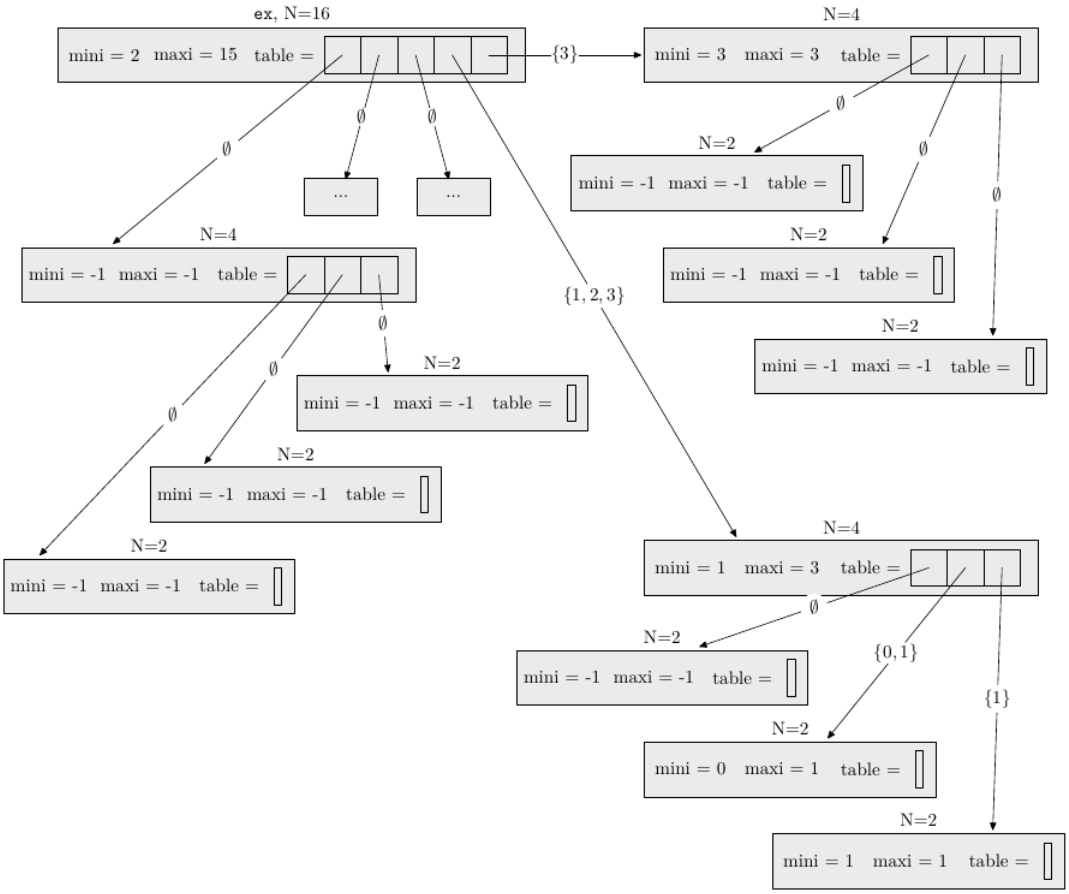

FIGURE 3 – L'arbre *veb* **ex** associé à l'ensemble $\{2, 13, 14, 15\}$ avec $p = 2$