Première partie

Thème structures de données - Corrigé

Exercice 1 : Tas binaire

Mines-Télécom, 2024

Q. 1 Donner la définition d'un tas binaire.

Q. 2 Proposer deux implémentations.

Q. 3 Trier le tableau suivant grâce à l'algorithme de tri par tas : $[4, 12, 5, 7, 3, 8, 1, 9]$

Q. 4 Quelle est la complexité de l'algorithme ? On attend une justification.

Exercice 2 : File

Mines-Télécom, 2024

Q. 1 Rappeler le principe d'une pile et d'une file.

Q. 2 Rappeler les opérations associées aux piles et aux files.

Q. 3 Expliquer comment faire une file en utilisant deux piles.

Q. 4 Prouver la correction de la réponse proposée à la question 3.

Q. 5 Donner la complexité des opérations dans le cadre de ce modèle.

Exercice 3 : Arbres et tri

Mines-Télécom, 2023

Soit A_n un arbre enraciné à n nœuds et de hauteur h.

Q. 1 Donner le nombre minimal et maximal de nœuds en fonction de la hauteur h et le prouver.

Soit T un tableau de taille n. On suppose qu'on ait un algorithme de tri qui puisse seulement comparer deux à deux les valeurs de ce tableau. On note a_n l'arbre de décisions associé à l'exécution de algorithme sur T. Les feuilles représentant le résultat de l'algorithme et les nœuds le choix de l'échange.

Q. 2 Prouver que a_n a $n!$ feuilles.

Q. 3 Prouver que la hauteur de a_n est un $\Omega(n \log_2(n))$ c'est-à-dire qu'il existe C une constante telle que la hauteur de a_n est supérieure ou égale à $Cnlog_2(n)$.

Q. 4 Que peut-on en déduire sur la complexité pire cas d'un algorithme de tri par comparaisons ?

Q. 5 Si on suppose que T ne contient que des entiers de $[\![0, K]\!]$, donner un algorithme de tri de T en $\Theta(n + K)$.

Exercice 4 : Arbres binaires de recherche

CCINP type A, sujet 0

Dans cet exercice, on autorise les doublons dans un arbre binaire de recherche et pour le cas d'égalité on choisira le sous-arbre gauche. On ne cherchera pas à équilibrer les arbres.

Q. 1 Rappeler la définition d'un arbre binaire de recherche.

Q. 2 Insérer successivement et une à une dans un arbre binaire de recherche initialement vide toutes les lettres du mot $bacddabdbae$ en utilisant l'ordre alphabétique sur les lettres. Quelle est la hauteur de l'arbre ainsi obtenu ?

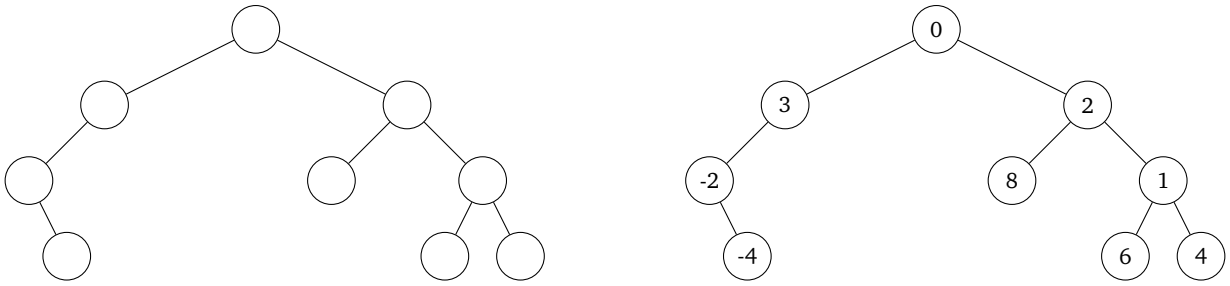
Q. 3 Montrer que le parcours en profondeur infixe d'un arbre binaire de recherche de lettres est un mot dont les lettres sont rangées dans l'ordre croissant. On pourra procéder par induction structurelle.

Q. 4 Proposer un algorithme qui permet de compter le nombre d'occurrences d'une lettre dans un arbre binaire de recherche de lettres. Quelle est sa complexité ?

Q. 5 On souhaite supprimer *une* occurrence d'une lettre donnée dans un arbre binaire de recherche de lettres. Expliquer le principe de l'algorithme permettant de résoudre ce problème et le mettre en oeuvre sur l'arbre obtenu à la question **Q. 2** en supprimant successivement une occurrence des lettres e, b, b, c et d. Quelle est sa complexité en fonction de la hauteur de l'arbre ?

Exercice 5 : Minima locaux dans des arbres *CCINP type A, sujet 0*

Dans cet exercice, on considère des arbres binaires étiquetés par des entiers relatifs deux à deux distincts. Un nœud est un minimum local d'un arbre si son étiquette est plus petite que celle de son éventuel père et celles de ses éventuels fils. Considérons par exemple l'étiquetage 1b de l'arbre binaire non étiqueté 1a.



(a) Sans étiquette (b) Avec étiquettes

FIGURE 1 – Arbres considérés dans l'exercice

Q. 1 Déterminer le ou les minima locaux de l'arbre 1b.

Q. 2 Donner une définition inductive permettant de définir les arbres binaires ainsi que la définition de la hauteur d'un arbre. Quelle est la hauteur de l'arbre 1b ?

Q. 3 Montrer que tout arbre non vide possède un minimum local.

Q. 4 Proposer un algorithme permettant de trouver un minimum local d'un arbre non vide et déterminer sa complexité.

On considère un arbre binaire non étiqueté que l'on souhaite étiqueter par des entiers relatifs distincts deux à deux de manière à maximiser le nombre de minima locaux de cet arbre.

Q. 5 Proposer sans justifier un étiquetage de l'arbre 1a qui maximise le nombre de minima locaux.

Q. 6 Proposer un algorithme qui, étant donné un arbre binaire non étiqueté en entrée, permet de calculer le nombre maximal de minima locaux qu'il est possible d'obtenir pour cet arbre. Déterminer la complexité de cet algorithme.

Q. 7 Montrer que, pour un arbre de taille $n \in \mathbb{N}$, le nombre maximal de minima locaux est majoré par $\left\lfloor \frac{2n+1}{3} \right\rfloor$. *On pourra remarquer que les nœuds non minimaux couvrent l'ensemble des arêtes de l'arbre.*

Exercice 6 : B-arbre

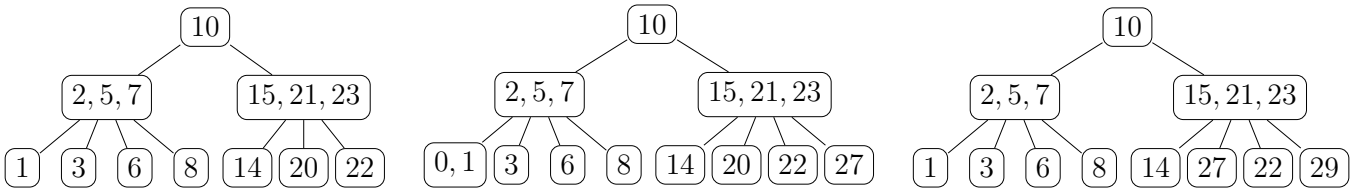
CCINP type A, 2024

Q. 1 Donner la définition d'un arbre binaire de recherche. Quelle condition doit respecter un tel arbre pour réaliser les opérations en temps logarithmique ? Donner une classe d'arbres qui vérifient cette condition. On notera $\mathcal{F}$ cette classe.

On définit un B**-arbre d'ordre** t comme étant un arbre dont les nœuds stockent des clés (au moins une) et vérifiant les conditions suivantes.

- Toutes les feuilles ont la même profondeur.
- Chaque nœud a au plus $2t-1$ clés.
- Chaque nœud qui n'est ni racine ni feuille a au moins $t-1$ clés.
- Si un nœud interne a n clés alors il a exactement $n+1$ enfants.
- Dans chaque nœud les clés sont classées par ordre croissant.
- Pour tout nœud, les clés de son i-ème sous-arbre sont toutes inférieures ou égales à sa i-ème clé (si elle existe) et toutes strictement supérieures) sa $i-1$-ème clé (si elle existe).

Q. 2 Lequel de ces trois arbres est un B-arbre d'ordre 2.



Q. 3 Soit a un B-arbre d'ordre t et de hauteur $h \geqslant 1$ qui contient n clés. Montrer que $n \geqslant 2t^{h-1} - 1$.
Indication : On pourra commencer par établir combien de nœuds internes a un arbre dont tous le nœuds internes ont t enfants et dont les feuilles sont toutes de profondeur h'.

Q. 4 Donner, en français ou en pseudo-code, un algorithme pour rechercher une clé dans un B-arbre.

Q. 5 Quelle est la complexité pire cas de cet algorithme ?
La comparer à la complexité pour la classe $\mathcal{F}$ de la **Q. 1**.

Q. 6 Soit a un B-arbre d'ordre t dont la racine a exactement $2t-1$ clés. Proposer un B-arbre d'ordre t contenant les mêmes clés mais dont la racine a exactement 1 clé.

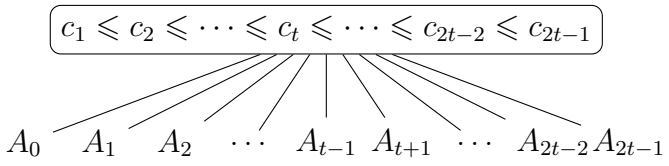


FIGURE 2 – Un B-arbre d'ordre t dont la racine a exactement $2t-1$ clés.

*NB : La dernière question n'est pas nécessairement celle du sujet original et l'indication de la **Q. 3** a été ajoutée.*

Exercice 7 : Arbres de Braun

CCINP type B

Le sujet fournit un fichier compagnon nommé `ccinp_arbre_braun.ml` fournissant le type décrit par l'énoncé ainsi qu'une partie des fonctions décrites ci-après. Il est à compléter avec les fonctions demandées.

Si t est un arbre, on note $|t|$ sa taille. Un *arbre de Braun* est un arbre binaire qui est :
- soit l'arbre vide ;
- soit de la forme $N(r, g, d)$ avec r une étiquette et g et d deux arbres de Braun vérifiant

$$|d| \leqslant |g| \leqslant |d| + 1$$

On se limite dans ce sujet sujet au cas où les étiquettes des arbres de Braun sont des entiers et on représente de tels arbres à l'aide du type suivant en OCaml.

```
type braun = E | N of int * braun * braun
```

Q. 1 Pour tout $n \in [\![1, 6]\!]$, dessiner les squelettes des arbres de Braun de taille n. Que remarque-t-on ?

On admet que la hauteur h et la taille n d'un arbre de Braun vérifient la relation $h = \Theta(\log n)$.

Q. 2 Écrire une fonction `hauteur : braun -> int` calculant la hauteur d'un arbre de Braun. On demande une complexité logarithmique en la taille de l'arbre en entrée, qu'on justifiera brièvement.

Un *tas de Braun* est un arbre de Braun vérifiant de surcroît que l'étiquette d'un nœud est toujours inférieure ou égale à celles de ses fils éventuels. Dans la suite, on implémente des fonctions sur les tas de Braun de façon à les utiliser pour implémenter une file de priorité. Le code compagnon met par ailleurs à disposition deux tas de Braun `t1` et `t2` pour les tests.

Q. 3 Écrire une fonction `minimum : braun -> int` renvoyant l'élément minimal d'un tas de Braun. Dans le cas où l'arbre est vide, on renverra `max_int`.

Q. 4 Le code compagnon fournit une fonction `inserer : braun -> int -> braun`. Justifier que `inserer a x` est un tas de Braun et qu'il contient l'élément `x` en plus de ceux présents dans `a`.

On suppose que l'on dispose d'une fonction `fusionner : braun -> braun -> braun` prenant en entrée deux tas de Braun t et t' tels que $|t'| \leqslant |t| \leqslant |t'| + 1$ et qui renvoie un tas de Braun contenant les éléments de t et de t'.

Q. 5 Écrire alors une fonction `extraire_min : braun -> int * braun` prenant en entrée un tas de Braun t et qui renvoie un couple (m, t') avec m le minimum de t et t' un tas de Braun contenant les étiquettes de t moins une occurrence de m. On lèvera une exception en cas d'arbre vide.

On cherche à présent à implémenter les fonctions nécessaires au bon fonctionnement de la fonction `fusionner` fournie dans le code compagnon.

Q. 6 Écrire une fonction `extraire_element : braun -> int * braun` prenant en entrée un tas de Braun t et renvoyant (x, t') tels que x est un élément quelconque de t et t' est un tas de Braun contenant les éléments de t sauf une occurrence de x. On pourra s'inspirer du fonctionnement de la fonction `inserer` et on lèvera une exception si l'arbre est vide.

Q. 7 Écrire une fonction `remplacer_min : braun -> int -> braun` prenant en entrée un tas de Braun t et un entier x et renvoyant un tas de Braun contenant les éléments de t, moins une occurrence du minimum de t, plus une occurrence de x.

Q. 8 Expliquer brièvement la correction de la fonction `fusionner` fournie par l'énoncé.

Q. 9 Quelles sont les complexités de `minimum`, `inserer` et `extraire_min`, c'est-à-dire des opérations de base sur une file de priorité implémentée avec un tas de Braun ?

Exercice 8 : Liste chainée de mots

CCINP type B, 2025

Cet énoncé est accompagné d'un code compagnon `liste_chainee.c` fournissant les définitions de type de cet énoncé ainsi que des valeurs pour tester les fonctions demandées. En dehors des appels déjà présents dans ce fichier pour la création d'instance, on ne fera aucun appel à la librairie `string.h`.
Dans cet exercice on manipule des listes chainées de mots implémentées en C à l'aide des types suivants.

```c
struct maillon{
  int n; // donne la capacité de mot
  char *mot;
  struct maillon *suivant;
};
typedef struct maillon *liste;
```

Q. 1 Écrire une fonction `void affiche_simple (liste l)` qui affiche les mots de la liste `l` avec un retour à la ligne après chaque mot.

Q. 2 Écrire une fonction `void affiche_reverse (liste l)` qui affiche les mots de la liste `l` dans l'ordre inverse de la liste avec un retour à la ligne après chaque mot.

Q. 3 Écrire une fonction `void ajoute_char (liste l, char c)` qui ajoute le caractère `c` à la fin de chacun des mots de la liste `l`. On considère que l'espace alloué pour chaque mot est suffisant pour l'ajout de ce caractère.

Q. 4 Quelle est la complexité pire cas de la fonction `ajoute_char`. Quelle serait la complexité si on décidait d'ajouter le caractère en début de chaque mot plutôt qu'à la fin.

La question suivante peut différer du sujet original, elle a été reconstituée du peu d'éléments rapportés. De plus l'exercice contenait une question supplémentaire.

Q. 5 Écrire une fonction `bool est_sous_mot(char* u, int nu, char* v, int nv)` qui prend en paramètres un mot `u` de taille `nu` et un mot `v` de taille `nv` et qui teste si `u` est un sous-mot de `v`.

Exercice 9 : File cyclique en OCaml

CCINP type B, 2023

Consignes : Ce sujet est à traiter en OCaml en complétant le fichier compagnon qui vous est fourni, nommé `ccinp_2023_file_cyclique.ml`. Outre des définitions de types et de valeurs, ce fichier contient une fonction d'affichage.

Le but de cet exercice est d'implémenter en OCaml une structure de file cyclique. Pour se représenter le fonctionnement d'une telle file on peut imaginer une pioche de cartes face cachée : celle sur le dessus du paquet est la tête. L'opération de défilement consiste à retourner cette carte, ainsi on découvre sa valeur, puis à la remettre sous le paquet. Si la pioche contient 5 cartes, en répétant cette opération 5 fois on est revenu à la situation de départ. Idem pour une file contenant 5 éléments : en effectuant 5 défilements la file est revenue à son état initial.

Les opérations. Les opérations de la file cyclique sont les mêmes que celles de la file, à ceci près que l'opération de défilement ne supprime pas l'élément en tête de la structure : elle le laisse en place pour le retrouver "au prochain tour". On dispose d'une opération spécifique pour supprimer l'élément en tête. L'opération d'ajout consiste à ajouter un élément après la queue (et donc juste avant la tête).

Les figures ci-dessous illustrent ces trois opérations, l'élément pointé étant la queue de la file.

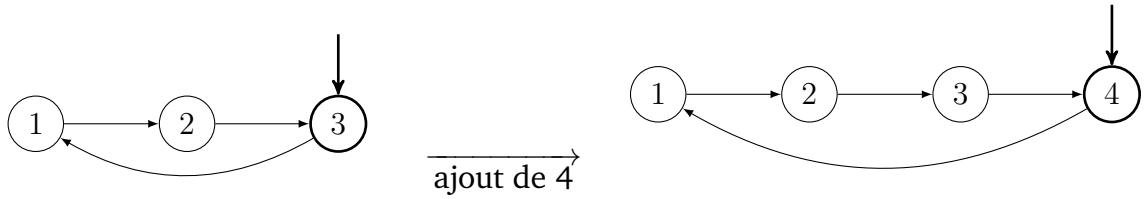


FIGURE 3 – Transformation d'une file cyclique par ajout

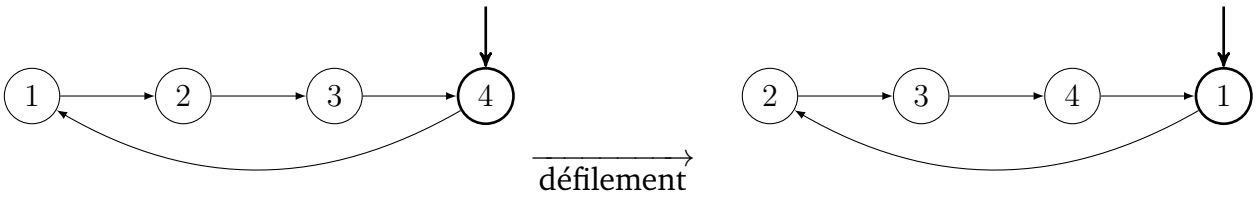


FIGURE 4 – Transformation d'une file cyclique par défilement

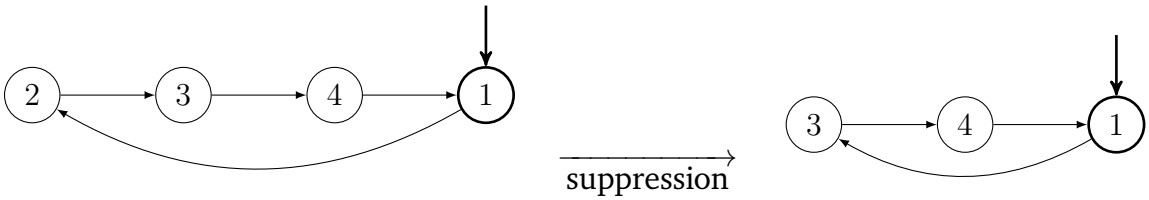


FIGURE 5 – Transformation d'une file cyclique par suppression de sa tête

Implémentation par maillon. On implémente ici la structure de file cyclique à l'aide de maillons. Chaque maillon correspond à un élément de la file, il contient non seulement cet élément mais aussi le maillon qui le suit dans la file. Ce maillon suivant est mutable pour permettre l'ajout et la suppression. On introduit donc le type suivant, où `'a` désigne le type des éléments.

```
type 'a maillon = {
  elem : 'a ;
  mutable suiv : 'a maillon
}
```

On peut alors décrire une file cyclique en indiquant le maillon correspondant à l'élément en queue, à moins que la file ne soit vide. On utilise un type option pour gérer cette disjonction de cas. De plus, afin de pouvoir modifier une file, on utilise une référence.

```
type 'a file = (('a maillon) option) ref
```

Q. 1 À la manière des figures ci-dessus dessiner les files `f1` et `f2` définies dans le fichier compagnon.

Q. 2 Compléter la fonction `cree_file_vide : unit -> 'a file` qui crée une file vide.

Q. 3 Compléter la fonction `est_file_vide : 'a file -> bool` qui teste si une file est vide.

Q. 4 Compléter la fonction `tete : 'a file -> 'a` qui calcule l'élément en tête d'une file non vide.

Q. 5 Compléter la fonction `defile : 'a file -> 'a` qui réalise le défilement pour une file cyclique.

Q. 6 Compléter la fonction `cree_singleton : 'a -> 'a file` qui crée une file réduite à un maillon contenant la valeur passée en argument et qui point vers lui même.

Q. 7 Compléter la fonction `ajoute : 'a file -> 'a -> unit` qui ajoute un élément dans une file. Préciser sa complexité.

Q. 8 Expliquer ce qui se passe lors du test d'égalité `f3 = (cree_singleton 1)`.

Q. 9 Compléter la fonction `est_singleton : 'a -> 'a file` qui teste si une file contient exactement un élément à l'aide du test d'égalité physique que permet l'opérateur `==`.

Q. 10 Compléter la fonction `supprime : 'a file -> 'a` qui supprime l'élément en tête d'une file non vide. On peut commencer par traiter le cas d'une file qui n'est pas un singleton.