Nom _____

Prénom _____

Code permanent _____

# Examen final

Date : 21 avril 2017

Titre du cours : Structures de données

Sigle et groupe : INF7341-40

Enseignant : Alexandre Blondin Massé

**Instructions**

1) Vous avez trois heures pour répondre à l'examen ;
2) Vous pouvez répondre aux *cinq* questions de l'examen ;
3) Les *quatre* meilleurs résultats seront comptabilisés ;
4) Vous avez droit à toute votre documentation ;
5) Il est interdit d'utiliser un ordinateur, peu importe sa taille et sa forme (téléphone portable, agenda électronique, etc.) ;
6) Il est interdit de parler et de prêter de la documentation à un autre étudiant ;
7) Au besoin, utilisez le verso comme brouillon ;
8) À moins d'avis contraire, justifiez toutes vos réponses et donnez le détail de vos calculs ;
9) Indiquez clairement vos réponses finales ;

| Question | 1 | 2 | 3 | 4 | 5 | Total |
|---|---|---|---|---|---|---|
| Sur | 25 | 25 | 25 | 25 | 25 | 100 |
| Note | | | | | | |

**Question 1**. . . . . (25 points)

Il est possible d'implémenter la structure de données File à l'aide de deux piles. Dans cette question, pour simplifier l'écriture du code, on suppose qu'une pile $p$ offre une opération $p$.DEPILER() qui retourne l'élément sur la tête de la pile *et* qui supprime cet élément de la pile (plutôt que de séparer les deux opérations comme nous l'avons fait en classe). De la même façon, on n'implémentera que l'opération DÉFILER() de la file plutôt que de diviser les opérations en deux opérations DÉFILER() et PREMIER().

Le constructeur d'une file vide $f$ construit tout simplement deux piles vides qu'on appellera $f$.GAUCHE() et $f$.DROITE(). Pour enfiler, il suffit d'utiliser le pseudocode suivant :

```
1: procedure ENFILER(f : File, e : Element)
2:     f.GAUCHE().EMPILER(e)
3: fin procedure
```

L'opération de défilement est la plus complexe :

```
1: fonction DÉFILER(f : File) : Element
2:     si f.DROITE().estVIDE() alors
3:         tant que ¬f.GAUCHE().estVIDE() faire
4:             f.DROITE().EMPILER(f.GAUCHE().DÉPILER())
5:         fin tant que
6:     fin si
7:     retourner f.DROITE().DÉPILER()
8: fin fonction
```

(a) (10 points) Illustrez l'évolution de la structure de données si on effectue les opérations suivantes :

1. On crée une file vide ;
2. On enfile les éléments $a$, $b$ et $c$ ;
3. On défile un élément ;
4. On enfile les éléments $d$ et $e$ ;
5. On défile deux éléments.
6. On défile un élément.

Autrement dit, donnez la trace de cet algorithme en dessinant, après les étapes 1, 2, 3, 4, 5 et 6 le contenu des deux piles.

(b) (15 points) Montrez que les opérations $f$.ENFILER et $f$.DEFILER ont une complexité amortie de $\Theta(1)$ à l'aide de la méthode du potentiel si la file est initialement vide. *Indice :* La fonction potentielle d'une file $f$ est $\Phi(f) = |f.\text{GAUCHE}()|$, c'est-à-dire le nombre d'éléments dans la première pile.

**Question 2**. . . . . (25 points)

Soit $T$ un arbre binaire de $n$ noeuds. La *hauteur minimale* de $T$, dénotée par $h_{\min}(T)$ est le nombre minimum de noeuds qu'on rencontre lorsqu'on commence à la racine et qu'on se dirige vers une feuille.

(a) (5 points) Est-ce vrai que $h_{\min}(T) \in \mathcal{O}(n)$ ? (Aucune justification demandée)

(a) _____

(b) (5 points) Est-ce vrai que $h_{\min}(T) \in \Omega(\log n)$ ? (Aucune justification demandée)

(b) _____

(c) (15 points) Donnez le pseudocode d'un algorithme *récursif* qui calcule la hauteur minimale de $T$.

**Question 3**. . . . . (25 points)

Nous avons vu en classe la structure d'ensembles disjoints $E$, qui offrent minimalement les opérations suivantes :

- $E$.AjouterSingleton($e$), qui ajoute un élément seul dans sa classe.
- $E$.Fusionner($e, f$), qui fusionne les classes de $e$ et de $f$.
- $E$.Representant($e$), qui retourne un représentant de la classe de $e$. Ce représentant est le même qui est retourné pour tout autre élément $f$ dans la classe de $e$.

Nous souhaitons maintenant ajouter une quatrième opération

- $E$.Afficher($e$), qui affiche tous les éléments de la classe de $e$, dans n'importe quel ordre.

En donnant le pseudocode des quatre opérations, montrez qu'il est possible d'étendre la structure de données d'ensembles disjoints en préservant la même complexité *spatiale* et les mêmes *complexités amorties* pour les opérations AjouterSingleton, Fusionner et Representant, ainsi qu'une complexité de $\Theta(|e|)$ pour l'opération Afficher, où $|e|$ est le nombre d'éléments dans la même classe que $e$.

*Indice 1 :* Il suffit d'ajouter un attribut à chaque noeud qui retient la liste de ses enfants et qui doit être mise à jour au fur et à mesure. *Indice 2 :* Afin de conserver les mêmes complexités, il n'est pas nécessaire que l'attribut *enfants* correspondent à la réalité, mais il peut faire abstraction de la compression de chemins.

**Question 4**. ..... (25 points)

Nous avons vu en classe qu'il existait plusieurs représentations d'un ensemble partiellement ordonné (*poset*). En particulier, une de ces représentations est la représentation matricielle. Supposons donc qu'on ait un poset de $n$ éléments $P = \{x_1, x_2, \ldots, x_n\}$ représenté par une matrice booléenne $n \times n$ et qu'on a $x_i \leq x_j$ si et seulement si $P[i, j] = \textit{vrai}$, pour $i, j \in \{1, 2, \ldots, n\}$. Autrement dit, on peut utiliser la notation matricielle $P[i, j]$ pour accéder à l'information en temps $\mathcal{O}(1)$.

Proposez une implémentation des opérations suivantes à l'aide de la représentation matricielle. Dans chacun des cas, indiquez la complexité au pire cas à l'aide de la notation $\mathcal{O}$ (la complexité doit être polynomiale) :

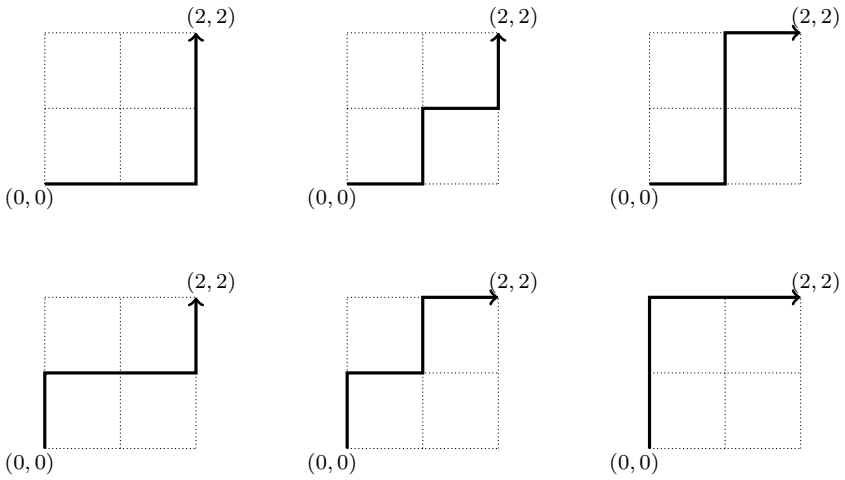
(a) (5 points) GEQ$(i, j)$ qui retourne vrai si et seulement si $x_i \geq x_j$.

(b) (10 points) COUVRE$(i, j)$ qui retourne vrai si et seulement si $x_i$ recouvre $x_j$, c'est-à-dire que $x_i \geq x_j$ et il n'existe aucun $z \in P$ tel que $x_i \geq z \geq x_j$.

(c) (10 points) JOIN$(i, j)$ qui retourne le supremum de $x_i$ et $x_j$ s'il existe, ou qui retourne *rien* sinon.

**Question 5**. ..... (25 points)

Soient $m, n \geq 0$ deux entiers. Dans cette question, on s'intéresse à la génération de tous les chemins qu'on peut dessiner sur une grille de dimensions $m \times n$ du point en bas à gauche $(0, 0)$ jusqu'au point en haut à droite $(m, n)$ en utilisant seulement des déplacements vers la droite et vers le haut. Par exemple, si $m = n = 2$, alors il existe 6 chemins :




Pour simplifier l'écriture, on représente un chemin par un mot sur l'alphabet $\{d, h\}$, qui dénote les déplacements *droite* et *haut* respectivement.

(a) (10 points) Donnez le pseudocode d'un générateur *récursif*

**générateur** CHEMINS($m, n$ : naturels) : mots

qui génère tous les mots (sans répétition, et sans en oublier) sur l'alphabet $\{d, h\}$ correspondant à tous les chemins possibles de $(0, 0)$ à $(m, n)$ n'utilisant que des déplacements *droite* et *haut*. *Indice :* Tout chemin de $(0, 0)$ à $(m, n)$ passe soit par $(m - 1, n)$ ou par $(m, n - 1)$.

(b) (15 points) Analysez la complexité d'initialisation, de délai et spatiale du générateur proposé en (a).