Nom et prénom :
Login examen :

Institut Paul Lambin Session de Septembre 2019

Examen de Structures de données : Avancé

Christophe Damas, José Vander Meulen

Année(s) d'études : 2ème année de baccalauréat en informatique de gestion

Date et heure : mercredi 21 août à 8h30

Durée de l'examen : 3 h ; pas de sortie durant les 60 premières minutes

Contenu

1. Questions sur machine 2
   a) Stages [8 pts] 3
   b) Graphe [5 pts] 6
   c) Récursion [3 pt] 7
   d) Programmes Etudiants [3 pts] 8
2. Question sur papier : Huffman [1 pt] 9

Total : **/20**

Nom et prénom :
Login examen :

1. Questions sur machine

Avant de commencer cette partie, suivez les étapes suivantes :

1. Dézippez le fichier compressé (**Extract here**).
2. Renommez tous les noms et prénoms des différents répertoires. Ex : stages_DAMAS_CHRISTOPHE. (Evitez les caractères spéciaux : accents, …)
3. **Utilisez la bonne version d'Eclipse (JEE) : L'eclipse normal ne contient pas le plugin XSL.**
4. Switchez votre workspace Eclipse afin qu'il soit positionné à la racine du Z.
5. Importez les différents projets dans votre workspace.

Pour cette partie, on doit avoir dans le Z :

- **stages_ NOM_PRENOM**
- **graphe_ NOM_PRENOM**
- **recursion_NOM_PRENOM**
- **programmes_etudiants_NOM_PRENOM**

Nom et prénom :
Login examen :

a) Stages [8 pts]

Dans le répertoire `stages`, vous trouverez un schéma XML valide intitulé `stages.xsd`.
Dans un document valide selon ce schéma, une asbl enregistre les inscriptions d'enfants à des stages sportifs organisés pendant l'été.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="anneeNaissance">
    <xs:restriction base="xs:string">
      <xs:pattern value="\d{4}" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="code">
    <xs:restriction base="xs:ID">
      <xs:pattern value="s\d{2}" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="semaine">
    <xs:restriction base="xs:integer">
      <xs:maxInclusive value="8" />
      <xs:minInclusive value="1" />
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="stages">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="stage" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="sport" type="xs:string" use="required" />
            <xs:attribute name="code" type="code" use="required" />
            <xs:attribute name="semaine" type="semaine" use="required" />
            <xs:attribute name="niveau" use="required">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="debutant" />
                  <xs:enumeration value="intermediaire" />
                  <xs:enumeration value="avance" />
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
          </xs:complexType>
        </xs:element>
        <xs:element name="enfant" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="inscription" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:attribute name="stage" type="xs:IDREF" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="nom" type="xs:string" use="required" />
            <xs:attribute name="prenom" type="xs:string" use="required" />
            <xs:attribute name="anneeNaissance"  type="anneeNaissance" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

A la fin de l'examen, dans votre répertoire `stages`, les trois fichiers suivants devront être présents : `stages.dtd`, `stages.xq` et `stages.xsl`.

Nom et prénom :
Login examen :

DTD [3 pts]

On vous demande d'écrire la DTD `stages.dtd` qui permet de valider les mêmes documents que `stages.xsd` (tout en étant plus permissive évidemment)

XQuery [2 pts]

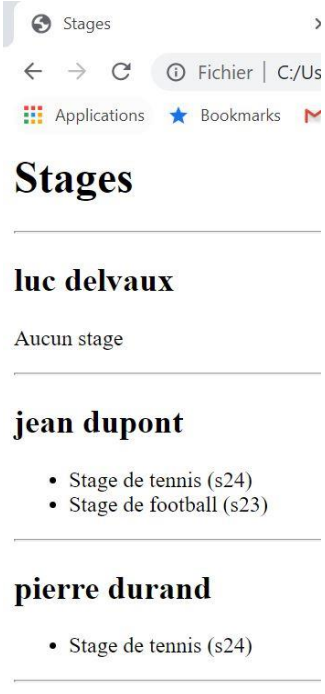
Dans un fichier `stages.xq`, écrivez la requête qui donne pour chaque stage la liste des enfants inscrits (avec l'aide de BaseX). Le résultat devrait ressembler à cela :

```
<stages>
  <stage code="s23" niveau="avance" sport="football" semaine="3">
    <enfant nom="dupont" prenom="jean"/>
  </stage>
  <stage code="s24" niveau="debutant" sport="tennis" semaine="3">
    <enfant nom="dupont" prenom="jean"/>
    <enfant nom="durand" prenom="pierre"/>
  </stage>
</stages>
```

XSL [3pts]

On vous demande d'écrire un fichier `stages.xsl` qui transforme un fichier xml valide par rapport à `stages.xsd` en html.
Le fichier généré affiche tous les enfants triés par ordre alphabétique (triés par nom et puis par prénom) avec les différents stages auxquels ils sont inscrits. Si un enfant n'est inscrit à aucun stage, il faut écrire « Aucun stage ».

Ci-dessous, vous trouverez la capture d'écran ainsi que le fichier source d'un `stages.html` obtenu à partir d'un fichier xml `contenant` les informations suivantes : il y a 2 stages (tennis et football) et 3 enfants (luc delvaux n'est inscrit à aucun stage, pierre durant est inscrit au stage de tennis et jean dupont est inscrit aux deux stages).



Nom et prénom :
Login examen :

```
<html>
	<head>
		<title>Stages</title>
	</head>
	<body>
		<h1>Stages</h1>
		<hr/>
		<h2>luc delvaux</h2>
		<p>Aucun stage</p>
		<hr/>
		<h2>jean dupont</h2>
		<ul>
			<li>Stage de tennis (s24)</li>
			<li>Stage de football (s23)</li>
		</ul>
		<hr/>
		<h2>pierre durand</h2>
		<ul>
			<li>Stage de tennis (s24)</li>
		</ul>
		<hr/>
	</body>
</html>
```

(Remarque : lors de la génération, il est normal que votre fichier html ne soit pas indenté et qu'il y ait 2 différences avec le fichier ci-dessus : les balises <hr> et une balise <META>)

Nom et prénom :
Login examen :

b) Graphe [5 pts]

Dans le répertoire graphe, vous trouverez un document XML valide intitulé `flight.xml.` Ce fichier contient des informations à propos de vols aériens. Ce fichier contient un certain nombre d'aéroports (élément `airport`) identifié par un code appelé code iata. Pour chaque aéroport, on a une liste de vol partant de cet aéroport (élément `flight`). Pour chaque vol, on a le code iata de l'aéroport de destination ainsi que la compagnie (attribut `airline`) .

Ce fichier peut être vu comme un graphe dirigé. Les sommets de ce graphe sont les aéroports. Il y a un arc entre deux aéroports si un vol les relie.

Dans le répertoire `Graphe`, on vous fournit un squelette de code : la classe `Graph.java`. L'implémentation de la classe `Graph` est basée sur une matrice d'adjacence.

L'objectif de cette question est de compléter la classe `Graphe.java.` Il y a 3 attributs à remplir via un parseur DOM. **Vous ne pouvez pas rajouter d'autres attributs.**

Parseur DOM [2 pts]

A l'aide d'un parseur DOM, implémentez le constructeur de la classe Graph.java. Ce constructeur prendra le fichier xml en paramètre et remplira trois structures de données :

- `correspondanceIndiceAirport` : ce dictionnaire fait la correspondance entre les indices des lignes/colonnes de la matrice et les codes iata des différents aéroports. Pour le fichier flight.xml, cette structure devrait contenir les infos suivantes :
  `{0=FCO, 1=AMS, 2=FRA, 3=JFK, 4=DEN, 5=IST, 6=STN, 7=DME, 8=LAX, 9=LGW, 10=ORD, 11=MAD, 12=CDG, 13=EWR, 14=ATL, 15=DFW, 16=DXB, 17=IAH, 18=LHR, 19=MUC, 20=PEK, 21=BCN}`
- `correspondanceAirportIndice` : ce dictionnaire est l'inverse du dictionnaire précédent. Il va retenir la correspondance entre les codes iata des aéroports avec les indices des lignes/colonnes de la matrice. Pour le fichier flight.xml, cette structure devrait contenir les infos suivantes :
  `{ORD=10, EWR=13, LAX=8, AMS=1, CDG=12, IST=5, DEN=4, STN=6, BCN=21, JFK=3, DXB=16, MAD=11, IAH=17, FCO=0, FRA=2, DFW=15, LHR=18, PEK=20, ATL=14, MUC=19, DME=7, LGW=9}`
- `matrice` : la matrice d'adjacence. A l'indice (i,j) de cette matrice, on retient la compagnie (`airline`) s'il y a un vol entre l'aéroport correspondant à l'indice i et l'aéroport correspondant à l'indice j. Si il n'y a pas de vols entre ces deux aéroports, l'indice (i,j) contient `null`. Par exemple, l'indice (0,0) contiendra `null` car il n'y a pas de vol entre `FCO` et `FCO`. L'indice (0,1) contiendra « `Alitalia` » car Alitalia est la compagnie reliant `FCO` et `AMS`. Remarque : il y a au maximum un vol entre un aéroport source et un aéroport destination.

BFS [3pts]

Implémentez la méthode `bfs()` de la classe `Graphe` qui prend le code iata d'un aéroport de départ en paramètre. Cette méthode affiche à la sortie standard les codes iata des différents aéroports qu'il peut atteindre dans l'ordre d'un parcours en largeur (BFS) depuis l'aéroport de départ.

Par exemple, une des sorties possibles de la méthode `g.bfs("JFK")` est :

```
JFK FCO AMS FRA DEN IST LAX ORD MAD CDG ATL DFW DXB LHR MUC PEK BCN DME LGW EWR STN IAH
```

Nom et prénom :
Login examen :

c) Récursion [3 pt]

Dans le répertoire `recursion`, vous trouverez la classe `Tree.java`.

strictementPositif [1 pt]

Dans la classe `Tree`, implémentez la méthode récursive `strictementPositif()` qui renvoie vrai si tous les entiers contenus dans l'arbre sont strictement positifs.
Une classe `Main` est fournie. L'appel de la méthode sur l'arbre `r4` renverra `false` car l'arbre contient l'entier -2.

toMap [2 pts]

Dans la classe `Tree`, implémentez la méthode récursive `toMap()` qui renvoie une map dont les clés sont les entiers présents dans l'arbre et les valeurs sont le nombre de fois qu'apparaissent ces entiers dans l'arbre.
Pour l'arbre `r4`, l'appel de cette méthode devrait renvoyer :

```
{1=1, -2=1, 3=2, 4=3, 5=1, 9=1}
```

En effet, 1 et -2 apparaissent une seule fois dans l'arbre, 3 est présent deux fois, …

Nom et prénom :
Login examen :

d) Programmes Etudiants [3 pts]

Dans le répertoire `programmes_etudiants`, nous fournissons un squelette de code qui a pour but de gérer les validations d'unités d'enseignement par les étudiants. Ce programme calcule pour chaque étudiant le nombre d'ects déjà validé et peut afficher la liste des étudiants triés par nombre d'ects validés. Chaque étudiant est identifié par son numéro de registre national. Une unité d'enseignement est identifiée par son nom.

L'objectif de cette question est de compléter les deux méthodes manquantes de la classe `ProgrammesEtudiants` qui enregistre les programmes de tous les étudiants :

- `valider(Etudiant e, UniteEnseignement ue)` : Enregistre la validation de l'unité d'enseignement par l'étudiant et met à jour le nombre d'ects validé par l'étudiant. Si l'unité d'enseignement a déjà été validée par l'étudiant, la méthode lance une RuntimeException avec le message 'ue déjà validée'
- `afficherEtudiantsTriesParEcts()` : affiche la liste des étudiants triée par nombre d'ects validés

Vous devez garantir une efficacité maximale pour ces méthodes.
Vous pouvez ajouter/modifier des attributs dans la classe `ProgrammesEtudiants` et vous pouvez également compléter le constructeur de cette classe. Vous pouvez également rajouter des méthodes/attributs dans `Etudiant` et `UniteEnseignement` si nécessaire.

Dans les cadres ci-dessous, donnez les complexités de vos deux méthodes en fonction de n (nombre d'étudiants) et m (nombre d'unité d'enseignement) :

`valider(Etudiant e, UniteEnseignement ue):`

`afficherEtudiantsTriesParEcts() :`

Une méthode main est fournie. L'output attendu est le suivant :

```
Alain Delcourt 10 ects
Pol Durant 8 ects
Jean Michel 0 ects
Exception in thread "main" java.lang.RuntimeException: ue déjà validée
	at ProgrammesEtudiants.valider(ProgrammeEtudiant.java:41)
	at ProgrammesEtudiants.main(ProgrammeEtudiant.java:78)
```

Nom et prénom :
Login examen :

2. Question sur papier : Huffman [1 pt]

Supposons qu'un texte à compresser soit formé des lettres suivantes avec comme fréquence :

A	B	C	D	E	F
12	7	3	5	8	1

Construisez sur papier l'arbre de Huffman pour ces fréquences.
Codez ensuite le texte : FACE
Décodez enfin la suite de bits : 00111000100001