Nom et prénom :

Haute Ecole Leonard de Vinci — Session de juin 2024

Examen de Structures de données : avancé

Christophe Damas, José Vander Meulen

Date et heure : vendredi 10 juin à 11h00

Durée : 2h ; pas de sortie durant les 60 premières minutes

Contenu

Total : /20

Nom et prénom :

1. Questions sur machine

Dans votre archive d'examen, vous devez avoir les répertoires suivants :

- **graphe_NOM_PRENOM**
- **huffman_NOM_PRENOM**
- **recursion_NOM_PRENOM**
- **inscription_NOM_PRENOM**

Pour ces parties, voici les consignes principales :

1. Nous vous conseillons de créer 4 nouveaux projets et d'y copier-coller les différents fichiers
2. A la fin de l'examen, vérifiez bien que vos productions apparaissent bien sur le disque U :
3. A la fin de l'examen, il faut que vos noms apparaissent dans les différents répertoires. Ex : graphe_DAMAS_CHRISTOPHE.

Nom et prénom :

a) Graphe [4 pts]

Dans le répertoire **graphe**, on vous fournit une implémentation partielle du projet de cette année. Il comprend deux fichiers, `cities.txt` et `roads.txt`, qui contiennent des informations sur des villes et des routes les reliant. Vous disposez également d'une classe **`Graph.java`** qui contient un constructeur permettant de lire les deux fichiers texte et de construire le graphe en utilisant une **liste d'arcs**. Cette classe contient également une map nommée **`mapVilleNom`** qui fait le lien entre le nom des ville et l'objet Ville correspondant.

Nous vous demandons d'implémenter la méthode **`bfs(String nomSource)`** dans la classe `Graph`.

Cette méthode affiche à la sortie standard les noms des différentes villes qu'il est possible d'atteindre dans l'ordre d'un parcours en largeur (BFS) depuis la ville de depart.

Voici le **début** du résultat attendu de la méthode main fournie :

```
Brussels Paris Metz Amsterdam Ghent Leuven Le Havre Orleans Calais
Lyon Le Mans Reims Chaumont Luxembourg Saint-Avold Lausanne Geneva
Saarbrucken Utrecht Amersfoort Groningen Bruges Liege Hasselt Amiens
Bordeaux Tours Montargis Toulouse …
```

b) Huffman [3 pts]

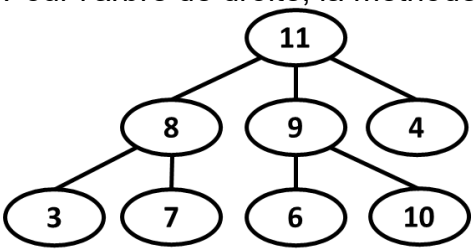
Dans le répertoire **Huffman**, vous trouverez un projet presque complet qui implémente l'algorithme de **Huffman**. Implémentez la méthode **buildTree()** de la classe **Huffman**.

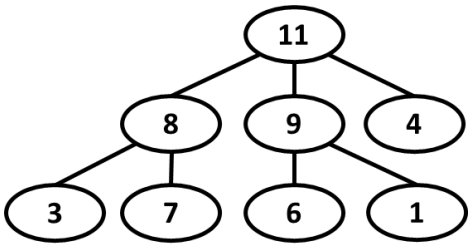
Nom et prénom :

c) Récursion [5 pts]

Dans le projet **recursion**, on vous fournit la classe **Tree** qui implémente des méthodes basiques sur les arbres

1) Dans la classe **Tree**, implémentez la méthode récursive **parentPlusGrandQueSesEnfants** qui renvoie true si la valeur de chaque nœud est strictement supérieure à la valeur de tous ses enfants, false sinon. Ci-dessous, vous trouverez les deux arbres implémentés dans la classe **Main**. Pour l'arbre de gauche, la méthode renvoie false car 10 est plus grand que 9. Pour l'arbre de droite, la méthode renvoie true.





2) Dans **Tree**, implémentez la méthode récursive **nbNoeudsParNiveau()** qui renvoie une **HashMap** contenant le nombre de nœuds pour chaque niveau de l'arbre. (Le niveau de la racine est considéré comme le niveau 0.) Pour les arbres ci-dessous, la méthode renvoie le même résultat : `{0=1, 1=3, 2=4}`. En effet, il y a un nœud au niveau 0 (la racine), 3 nœuds au niveau 1 (les nœuds 8, 9 et 4) et 4 nœuds au niveau 2.

Nom et prénom :

d) Inscription Oral [6 pts]

Un enseignant souhaite développer un programme permettant aux étudiants de s'inscrire à des examens oraux se déroulant sur une seule journée.

Le programme doit proposer 3 fonctionnalités :

1. Inscription d'un étudiant à une plage horaire. Plusieurs étudiants peuvent choisir la même plage horaire, mais un étudiant ne peut pas être inscrit à deux plages horaires différentes ou deux fois à la même plage horaire.
2. Affichage de l'horaire des examens dans l'ordre chronologique.
3. Échange des plages horaires entre deux étudiants.

Dans le répertoire **`inscription`**, nous fournissons un squelette de code.
La classe **`Etudiant`** gère les informations sur un étudiant (matricule unique, nom et prénom). Cette méthode contient une méthode **`toString()`** à réutiliser. La classe **`PlageHoraire`** gère les plages horaires (heure et minutes). Elle contient également une méthode **`toString()`** à réutiliser.
Vous devez compléter la classe **`InscriptionOral`** (et seulement cette classe) en implémentant les méthodes **`inscrireEtudiant(Etudiant e, PlageHoraire p), afficherHoraire() et echanger(Etudiant e1, Etudiant e2).`**
Vous devez ajouter un/des attribut(s) dans **`InscriptionOral`** qui devra (devront) être initialisé(s) dans le constructeur.

Vous devez garantir une efficacité maximale pour les différentes méthodes à implémenter.

Dans les cadres ci-dessous, donnez les complexités de vos trois méthodes :

inscrireEtudiant(Etudiant e, PlageHoraire p):

afficherHoraire():

echanger(Etudiant e1, Etudiant e2):

Une méthode main est fournie. L'output attendu est le suivant (l'ordre des étudiants au sein d'une plage n'est pas important):

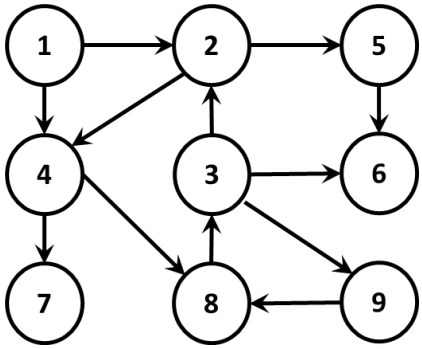
```
9h : Damas Christophe/Vander Meulen José/Cambron Isabelle/
9h30 : Seront Gregory/Lehmann Brigitte/
10h : Ferneeuw Stephanie/
----------------
9h : Vander Meulen José/Cambron Isabelle/Ferneeuw Stephanie/
9h30 : Seront Gregory/Lehmann Brigitte/
10h : Damas Christophe/

Exception in thread "main" java.lang.RuntimeException: L'étudiant est déjà inscrit
      at InscriptionsOral.inscrireEtudiant(InscriptionsOral.java:18)
      at Main.main(Main.java:26)
```

Nom et prénom :

2. Question sur papier

DFS [2pts]

Considérons le graphe suivant :



Pour ce graphe, dessinez l'arbre obtenu lors d'un parcours en profondeur (DFS) commençant au sommet 1 (les arcs sont essayés dans l'ordre croissant de leur destination).