Nom et prénom :

Haute Ecole Leonard de Vinci — Session d'août 2023

Examen de Structures de données : avancé

Christophe Damas, José Vander Meulen

Date et heure : vendredi 18 aout à 8h30

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**
- **recursion_NOM_PRENOM**
- **huffman_NOM_PRENOM**
- **courses_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 projet **graphe**, on vous fournit une implémentation partielle du projet de cette année (stib). Il comprend deux fichiers, `lignes.txt` et `tronçons.txt`, qui contiennent des informations sur les lignes et les tronçons du réseau de la STIB. 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 **`mapNomStation`** qui fait le lien entre le nom des stations et l'objet Station correspondant.

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

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

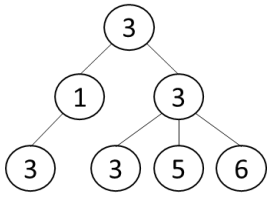
Voici le début du résultat attendu de la méthode main fournie :

```
ALMA VANDERVELDE KRAAINEM CRAINHEM KLAKKEDELLE ROODEBEEK MOUNIER
REINE ASTRID STOCKEL HYMANS WOLUWE SHOPPING CHARMILLE TOMBERG
AUDITOIRES-UCL WEZEMBEEK-SHOP. ESCRIME GERSIS AMITIE BOULEAUX
MOONENS VOOT PEUPLIERS MELARD MONTALD GRIBAUMONT CLINIQUES-UCL RING
0 VOISINS AVIATION VAL DES SEIGNEURS EGLANTIERS PADDOCK …
```

b) 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 **nbNoeudEgalA(int x)** qui compte le nombre de fois qu'une valeur apparait dans l'arbre. Sur l'arbre ci-dessous, l'appel de **nbNoeudEgalA(3)** renvoie 4.



2) Dans **Tree**, implémentez la méthode **toSet()** qui renverra un HashSet contenant les entiers présents dans l'arbre, sans doublons. Pour l'arbre ci-dessus, le résultat attendu est le suivant : **`[1, 3, 5, 6]`**

Nom et prénom :

c) Huffman [4 pts]

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

d) Courses [4 pts]

Un club cycliste souhaite créer un programme simple pour gérer des courses de type « contre-la-montre », où les participants s'élancent individuellement dans des épreuves chronométrées pour compléter un parcours spécifique le plus rapidement possible. Chaque coureur part à son propre moment et le but est d'achever la course le plus rapidement possible.

Le programme doit proposer 3 fonctionnalités :

- Permettre d'encoder le départ d'un coureur,
- Permettre d'encoder l'arrivée d'un coureur,
- Permettre d'afficher le classement provisoire des coureurs déjà arrivés. Les coureurs sont classés par temps. Si deux coureurs ont le même temps, alors l'ordre entre eux n'a pas d'importance.

Dans le répertoire courses, nous fournissons un squelette de code.
La classe **`Coureur`** gère les informations sur le coureur (id unique et son nom) ainsi que son temps de départ et d'arrivée. Les méthodes **`setDepart`** et **`setArrivee`** doivent être utilisées pour encoder les temps de départ et d'arrivée. La méthode **`obtenirTemps()`** permet de calculer le temps global d'un coureur s'il a déjà terminé le parcours. Si possible, cette classe ne devrait pas être modifiée.
Vous devez compléter la classe **`Course`** en implémentant les méthodes **`encoderDepart()`**, **`encoderArrivee()`** et **`afficherClassementProvisoire()`**.
Vous aurez besoin de rajouter un/des attribut(s) dans **`Course`** qui devra (devront) être initialisé(s) dans le constructeur.

La méthode `afficherClassementProvisoire()` est appelée à chaque arrivée d'un coureur. Vous devez garantir une efficacité maximale pour cette méthode.

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

encoderDepart():

encoderArrivee():

afficherClassementProvisoire():

Nom et prénom :

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

```
Coureur [id=3, nom=marc, Temps: 75]
Coureur [id=2, nom=pol, Temps: 90]
Coureur [id=1, nom=jean, Temps: 165]
----------------------------
Coureur [id=3, nom=marc, Temps: 75]
Coureur [id=4, nom=luc, Temps: 80]
Coureur [id=2, nom=pol, Temps: 90]
Coureur [id=5, nom=jean, Temps: 165]
Coureur [id=1, nom=jim, Temps: 165]
----------------------------
```

2. Question sur papier

Minimax [3pts]

Soit le jeu des 4 nombres. Il a les mêmes règles que le jeu des 10 nombres présenté dans les transparents du cours théorique sur l'algorithme Minimax.
Supposons que l'on commence avec les 4 nombres suivants : 1 3 4 2
Sur la feuille A3 du questionnaire d'examen, nous avons dessiné l'arbre du jeu.
Répondez aux deux questions suivantes sur la feuille A3 présentant l'arbre du jeu :

1) Pour chaque noeud de l'arbre, calculez sa valeur Minimax.
2) Est-ce que le joueur qui commence est assuré de gagner en jouant de manière optimale ? Justifiez