Nom et prénom :

Haute Ecole Leonard de Vinci — Session de juin 2023

Examen de Structures de données : avancé

Christophe Damas, José Vander Meulen

Date et heure : mercredi 14 juin à 13h30

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**
- **minimax_NOM_PRENOM**
- **huffman_NOM_PRENOM**
- **competition_NOM_PRENOM**

Pour ces parties, voici les consignes principales :

1. Nous vous conseillons de créer 5 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 [6 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'adjacence. Cette classe contient également une **map** nommée **mapNomStation** qui fait le lien entre le nom des stations et l'objet **Station** correspondant.

1) Implémentez la méthode **tronconsEntrants** dans la classe **Graph.java**. Cette méthode prend en paramètre un objet **Station** et renvoie l'ensemble des tronçons entrants de cette station, c'est-à-dire les tronçons dont l'arrivée correspond à la station en paramètre. [2pt]

   Vous pouvez utiliser la classe **Main** pour tester votre méthode (le résultat apparaitre avant la barre de pontillé). Pour la station **ALMA**, la méthode devrait renvoyer les tronçons suivants :

   ```
   Troncon [ligne=1, depart=VANDERVELDE, arrivee=ALMA, duree=1]
   Troncon [ligne=N05, depart=VANDERVELDE, arrivee=ALMA, duree=2]
   Troncon [ligne=1, depart=CRAINHEM, arrivee=ALMA, duree=1]
   Troncon [ligne=N05, depart=KRAAINEM, arrivee=ALMA, duree=1]
   ```

   Il y a en effet 4 tronçons qui ont **ALMA** comme arrivée.

2) Dans la classe Graph, nous avons commencé l'implémentation de la méthode **calculerCheminMinimisantNombreTroncons**. Pour l'instant, la méthode affiche à la sortie standard les nom des stations que l'on peut atteindre dans l'ordre d'un parcours en largeur (BFS) depuis la station de départ.
   L'objectif de cette question est de compléter ce code afin que cette méthode affiche le chemin entre deux stations avec le moins de tronçons possibles [4 pt]. Pour un chemin, il faut simplement afficher les tronçons dans l'ordre en utilisant la méthode **toString()** déjà implémentée de la classe **Troncon**. Il ne faut pas afficher la durée totale du trajet, ni le nombre de tronçons. Vous pouvez supprimer le **System.out.println** du code initial mais vous devez garder les autres lignes.
   Exemple de résultat attendu pour :**g.calculerCheminMinimisantNombreTroncons("BOILEAU", "ALMA");**

   ```
   Troncon [ligne=7, depart=BOILEAU, arrivee=MONTGOMERY, duree=1]
   Troncon [ligne=1, depart=MONTGOMERY, arrivee=JOSEPH.-CHARLOTTE, duree=2]
   Troncon [ligne=1, depart=JOSEPH.-CHARLOTTE, arrivee=GRIBAUMONT, duree=1]
   Troncon [ligne=1, depart=GRIBAUMONT, arrivee=TOMBERG, duree=1]
   Troncon [ligne=1, depart=TOMBERG, arrivee=ROODEBEEK, duree=2]
   Troncon [ligne=1, depart=ROODEBEEK, arrivee=VANDERVELDE, duree=2]
   Troncon [ligne=1, depart=VANDERVELDE, arrivee=ALMA, duree=1]
   ```

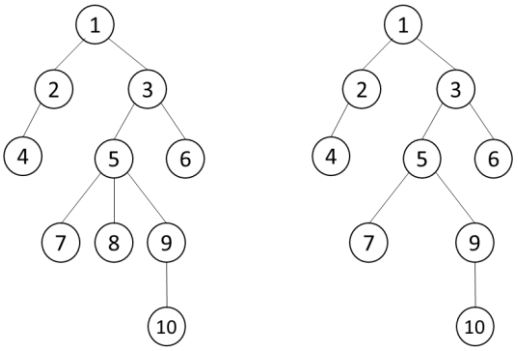
   Il est peut-être possible de trouver d'autres chemins avec 7 tronçons.

Nom et prénom :

b) Récursion [2 pts]

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

Dans **Tree**, implémentez la méthode **estArbreBinaire()** qui renverra vrai si tous les noeuds ont au maximum deux fils, faux sinon. Les deux arbres suivants sont créés dans la méthode **main()**. Le premier n'est pas binaire (5 a trois enfants) et le deuxième est binaire. Notez qu'un arbre contenant un seul nœud est un arbre binaire.



.

c) Minimax [3 pts]

Dans le répertoire **Minimax**, vous trouverez un projet presque complet qui implémente l'algorithme minimax pour le jeu vu au cours. Implémentez la méthode **computeMinimaxValue** de la classe **Tree**.

d) Huffman [3 pts]

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

Nom et prénom :

e) Compétition [3 pts]

Dans le projet **competition**, on vous fournit un squelette de code presque complet d'une compétition de saut en longueur.

Chaque athlète à droit à 3 sauts. Les athlètes sont classés en fonction de leur meilleur saut.

Ex : Dans la méthode main, il y a 4 athlètes (Damas, Leconte, Vander Meulen et Seront). Damas a sauté 50 cm, 250 cm et 0 cm. Leconte a sauté 450 cm, 0 cm et 0 cm. Vander Meulen a sauté 50 cm, 250 cm et 0 cm. Seront a sauté 30 cm, 300 cm et 0 cm.

Le classement devrait être le suivant (output de la méthode **classement()**) :

```
1 Leconte
2 Seront
3 Damas
4 Vander Meulen
```

Leconte est 1ère car elle a sauté à 450 cm, Seront est 2ème car il a sauté à 300 cm. Damas et Vander Meulen ont tous deux sauté à 250 cm ; ils sont donc 3ème et 4ème ; l'ordre entre eux n'a pas d'importance.

Le projet **competition** compile mais plante à l'exécution et lance l'exception suivante:

```
Exception in thread "main" java.lang.ClassCastException: class Athlete
cannot be cast to class java.lang.Comparable (Athlete is in unnamed module
of loader 'app'; java.lang.Comparable is in module java.base of loader
'bootstrap')
      at java.base/java.util.TreeMap.compare(TreeMap.java:1563)
      at java.base/java.util.TreeMap.addEntryToEmptyMap(TreeMap.java:768)
      at java.base/java.util.TreeMap.put(TreeMap.java:777)
      at java.base/java.util.TreeMap.put(TreeMap.java:534)
      at java.base/java.util.TreeSet.add(TreeSet.java:255)
      at Competition.<init>(Competition.java:9)
      at Main.main(Main.java:8)
```

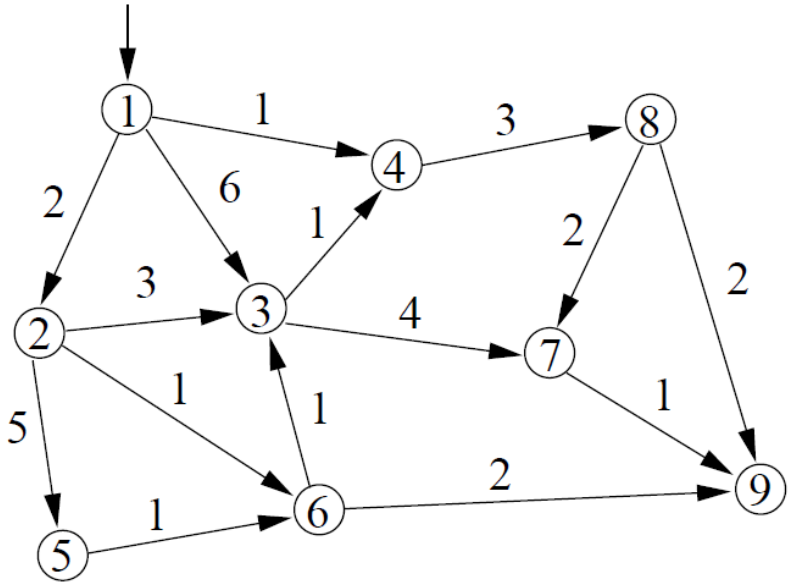
Le problème vient du fait que les **Athlete** ne sont pas comparables. Résolvez ce problème et tout devrait fonctionner. Il n'y a rien à modifier dans le code donné. Il faut juste ajouter du code pour rendre **Athlete Comparable**. Il n'y a rien d'autre à faire.

Nom et prénom :

2. Question sur papier

Dijkstra [3pts]

Appliquez l'algorithme de **Dijkstra** pour déterminer les chemins les plus courts à partir du **sommet 1**. Veuillez compléter le tableau des étiquettes provisoires et des étiquettes définitives à chaque étape de l'algorithme, comme expliqué dans les slides. Vous disposez de 10 étapes, mais il est possible que l'exécution de l'algorithme se termine en moins de 10 étapes.



Etiquettes provisoires								
1	2	3	4	5	6	7	8	9

Etiquettes définitives								
1	2	3	4	5	6	7	8	9

Nom et prénom :