Cycle ingénieur - 2ème année

Programmation fonctionnelle en Scala

Projet *Algorithmes sur des graphes simples/stricts*

2021-2022

Consignes

Consignes générales

Objectif du projet

- Implémenter certains algorithmes classiques sur des graphes simples/stricts
- Utiliser ces algorithmes sur quelques applications

Consignes générales

- Projet effectué par groupes de **4 à 5 étudiants**
- Date limite de rendu (par mail) : **03 avril 2022, 23h59**
- Soutenance : semaine du **04 au 08 avril 2022**

Nature du rendu

- *Fourni* : Projet `sbt` à compléter
- **À faire** : implémenter les parties `???`
- **Rendu** :
  - archive du répertoire `src/main/scala` uniquement
  - rapport court

Consignes spécifiques

Modification du code existant

- Modification **INTERDITE** :
  - des classes déjà implémentées
  - des valeurs déjà implémentées
  - des méthodes déjà implémentées
  - de la signature des valeurs et méthodes à implémenter
- Ajout *autorisé* de nouvelles classes, valeurs et/ou méthodes intermédiaires pour implémenter les éléments demandés

Consignes spécifiques

Tests des valeurs et méthodes à implémenter

- Des tests seront effectués sur votre code.
  Ces tests participent à la note finale du projet.
- Pour fonctionner, **ces tests supposent que les consignes précédentes ont été respectées**.
  Dans le cas contraire, les tests ne compileront pas.

Tests fournis

- Exemples de tests sur des éléments déjà implémentés
- Framework principal de test : ScalaTest
- Intégration de ScalaCheck (*property testing*)

Consignes spécifiques

Respect des règles de programmation fonctionnelle

- Utilisation des variables (`var`) **interdite**
- Utilisation des boucles (conditionnelles ou non) **interdite**
- ***Favoriser la récursivité terminale***

Quelques conseils

- *for-comprehensions* (`for ... yield`) *autorisées*.
- Utiliser le plus possible la bibliothèque des collections Scala

Rapport

Rapport de développement

- Description succincte de la conception mise en place
- Description succincte de l'implémentation mise en place
- Présentation des résultats sur les applications proposées

Étude des implémentations d'un graphe simple/strict

- Analyse comparative

Étude des algorithmes demandés

- Analyse comparative selon les implémentations
- Analyse comparative entre les deux algorithmes de coloration de sommets

Analyse de performance de la parallélisation (`.par`) *[Facultatif]*

Soutenance (semaine du 04/04 au 08/04)

Organisation

- Durée totale : *20 minutes*
  - Temps de présentation : **entre 10 et 15 min**

    Contrainte de temps à respecter absolument
  - Reste du temps consacré à nos questions
- **Tous les membres du groupe doivent présenter.**

Attendu

- Analyse comparative des implémentations de graphe
- Analyse comparative des algorithmes
- **Présenter les résultats sur les applications proposées**

Contenu du répertoire `src/main/scala` fourni

Aucune modification n'est requise dans les fichiers en *italique*.

- src/main/scala
  - undirected
    - *Edge.scala*
    - SimpleGraph.scala
    - *SimpleGraphDefaultImpl.scala*
    - SimpleGraphNeighborsImpl.scala
    - SimpleGraphMatrixImpl.scala
  - directed
    - *Arc.scala*
    - StrictGraph.scala
    - *StrictGraphDefaultImpl.scala*
    - StrictGraphSuccessorsImpl.scala
    - StrictGraphMatrixImpl.scala
  - applications

Présentation du projet

Présentation du projet

Présentation générale

Présentation générale

Contexte de travail

- Graphes non orientés *simples* (sans boucle, ni arêtes multiples)
- Graphes orientés *stricts* (sans boucle, ni arcs multiples)

Algorithmes étudiés dans ce projet

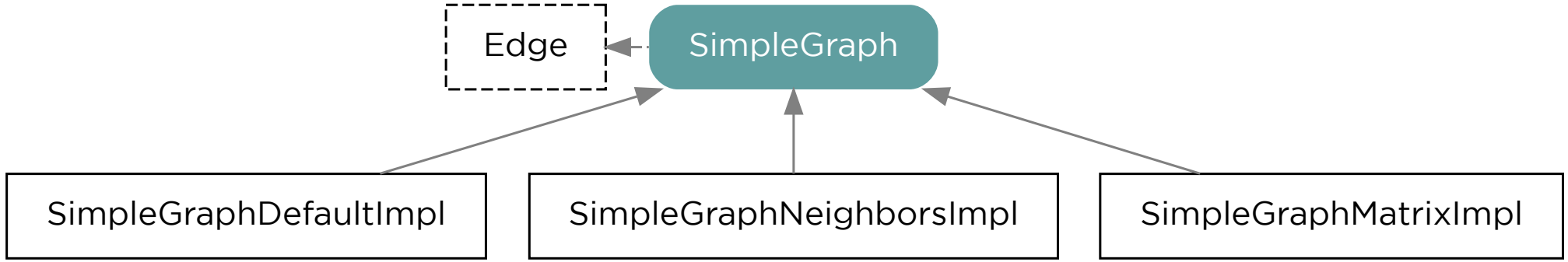
- *Coloration de sommets (graphes non orientés)*
  - Algorithme "glouton" (a.k.a Welsh-Powell)
  - Algorithme DSATUR
- *Ordonnancement (graphes orientés)*

  Tri topologique par parcours en profondeur étendu
- *Recherche de plus court chemin (graphes orientés)*

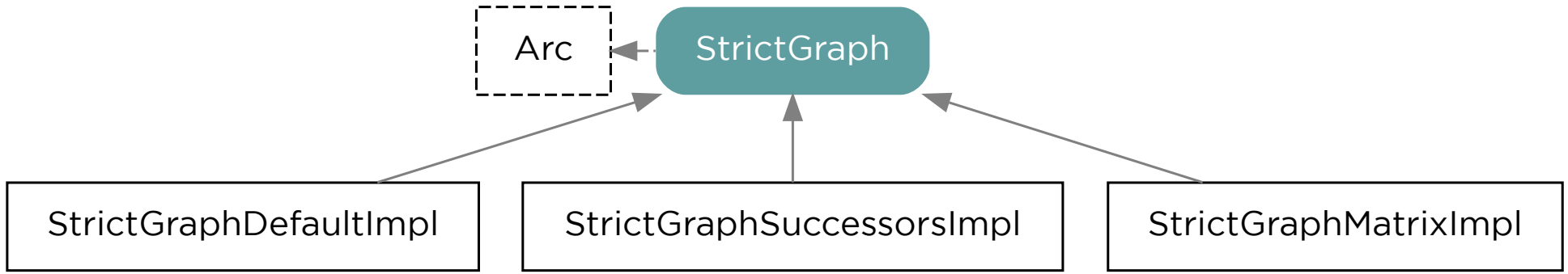
  Algorithme de Dijkstra

Hiérarchie des types du code fourni

Package `undirected`



Package `directed`



Présentation du projet

Graphes non orientés (package `undirected`)

`SimpleGraph[V]`

Trait décrivant un graphe non orienté simple

- Type des sommmets du graphe : `V`
- Type des arêtes du graphe : `Edge[V]` *(fourni)*

Fourni

- `toString`
- `toDOTString` : chaîne représentant le graphe en langage DOT

SimpleGraph[V]

À implémenter dans le trait

Méthodes générales

- `hasPath` : existence de chemin entre deux sommets
- `isConnected` : le graphe est-il connexe ?
- `isAcyclic` : le graphe est-il acyclique ?

Calcul de l'arbre couvrant minimum

- `minimumSpanningTree`

Calcul de coloration propre de sommets

- `sortedVertices` : séquence des sommets triée par degré décroissante
- `greedyColoring` : coloration par algorithme de Welsh-Powell
- `ColoringDSATUR` : coloration par algorithme DSATUR

Implémentations de `SimpleGraph[V]`

`SimpleGraphDefaultImpl`

- Implémentation la plus proche de la définition mathématique
- Stocke l'ensemble des sommets et l'ensemble des arêtes
- **Fournie** avec ses tests (`src/test/scala/undirected`)

`SimpleGraphNeighborsImpl` à compléter

- Stocke pour chaque sommet l'ensemble de ses voisins

`SimpleGraphMatrixImpl` à compléter

- Stocke la matrice d'adjacence et l'ordre dans lequel les sommets sont utilisées

Présentation du projet

Graphes orientés (package `directed`)

`StrictGraph[V]`

Trait décrivant un graphe orienté strict

- Type des sommmets du graphe : `V`
- Type des arcs du graphe : `Arc[V]` *(fourni)*

Fourni

- `toString`
- `toDOTString` : chaîne représentant le graphe en langage DOT

`StrictGraph[V]`

À implémenter dans le trait

Méthodes générales

- `inDegree` : degré intérieur d'un sommet
- `outDegree` : degré extérieur d'un sommet
- `degreeOf` : degré d'un sommet

Calcul de l'ordre topologique

- `topologicalOrder`

Calcul du plus court chemin

- `shortestPath`

Implémentations de `StrictGraph[V]`

`StrictGraphDefaultImpl`

- Implémentation la plus proche de la définition mathématique
- Stocke l'ensemble des sommets et l'ensemble des arcs
- **Fournie** avec ses tests (`src/test/scala/directed`)

`SimpleGraphSuccessorsImpl` à compléter

- Stocke pour chaque sommet l'ensemble de ses successeurs

`SimpleGraphMatrixImpl` à compléter

- Stocke la matrice d'adjacence et l'ordre dans lequel les sommets sont utilisées

Présentation du projet

Applications

Applications

- Affectations des fréquences
- Reconnexion
- *Makefile*
- Application GPS

À inclure dans le code envoyé

- Sous-répertoire `src/main/scala/applications`
- Un objet principal (i.e. avec méthode `main`) par application

Évaluation

- *Non incluses dans les tests effectués*

  ⇒ Possibilité d'utiliser des librairies extérieures

  (CSV, options de ligne de commande, ...)
- **Attendus du rapport et de la soutenance**

Affectation de fréquences

Problème posé

- Affectation de fréquences pour des antennes d'une commune
- Des antennes trop proches ne peuvent avoir la même fréquence (interférence)

Combien de fréquences sont nécessaires et quelle affectation faire ?

Données

- Positions des installations radioélectriques de plus de 5W
  (Source : Plateforme ouverte des données publiques)
  → Ficher `src/main/resources/antennes.csv`
- Distance minimale de non-interférence (paramétrable)
- Valeur de la colonne `COM_CD_INSEE` (paramétrable)
  Exemples : Cergy (95127), Pau (64445)

Reconnexion

Problème posé

- Reconnexion des antennes d'une commune en partant de zéro

Quelle longueur de cable est nécessaire et comment reconnecter ?

Données

- Positions GPS des installations radioélectriques de plus de 5W
  (Source : Plateforme ouverte des données publiques)
  → Ficher `src/main/resources/antennes.csv`
- Valeur de la colonne `COM_CD_INSEE` (paramétrable)
  Exemples : Cergy (95127), Pau (64445)

Makefile

Problème posé

- Respect des dépendances de compilation entre fichiers

Dans quel ordre compiler les fichiers pour respecter les dépendances ?

Données

- Description des dépendances
  (format de type *Makefile* simplifié)
  → Fichier `src/main/resources/Makefile`

Doit détecter les dépendances circulaires

(Inclure par vous-même un cas de dépendance circulaire)

Application GPS

Problème posé

- Recherche de l'itinéraire le plus court en temps et en distance

Données

- Positions GPS des points d'intéret
- Trajets directs possibles avec distance et temps de parcours
- Départ de l'itinéraire (paramétrable)
- Arrivée de l'itinéraire (paramétrable)

Formats d'entrée non imposés : à vous de construire l'exemple

Compléments

GraphViz/DOT

GraphViz

- Ensemble d'outils *open-source* de visualisation de graph
- Multi-plateforme
- S'appuie sur un langage de représentation : DOT
- Formats d'export variés :
  - images (JPEG, PNG, PSD, ...)
  - vectoriel (SVG)
  - document (PDF, PostScript, LaTeX/TikZ)
  - ...