Projet : Réseau de distribution d'électricité (partie 2)

I Entrées-Sorties

Lors de la première phase du projet, vous avez dû construire à la main votre réseau électrique (générateurs, maisons et connexions). Cela peut s'avérer fastidieux surtout avec un grand nombre de générateurs et/ou maisons. La première modification majeure va consister à simplifier cette partie là. Nous souhaitons maintenant que le programme puisse lire dans un fichier texte la description du réseau. Un tel fichier devra respecter le format suivant :

```
generateur(gen1,60).
generateur(gen2,45).
generateur(gen3,20).
generateur(gen4,42).
maison(maison1,NORMAL).
maison(maison2,BASSE).
maison(maison3,NORMAL).
maison(maison4,FORTE).
maison(maison5,FORTE).
maison(maison6,NORMAL).
connexion(gen1,maison1).
connexion(gen2,maison2).
connexion(gen3,maison3).
connexion(gen4,maison4).
connexion(gen2,maison5).
connexion(gen3,maison6).
```

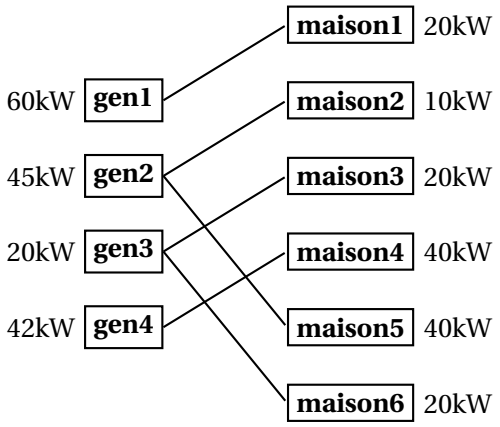

FIGURE 1 – Le réseau électrique correspondant au fichier texte

Pour être valide, le fichier doit impérativement respecter l'ordre de définitions des éléments : d'abord définir les générateurs, puis les maisons, et enfin les connexions. Les générateurs et les maisons peuvent avoir un nom quelconque mais, pour simplifier les choses, nous supposerons qu'ils seront uniquement constitués de caractères alpha-numériques (des lettres minuscules ou majuscules et des chiffres). Le format correspondant aux générateurs et aux maisons devront être les mêmes que celui de l'exemple (d'abord le nom puis la capacité maximale ou la demande électrique). Pour les connexions, vous avez l'obligation de mettre le mot clé « connexion » au début de la ligne mais il est possible de mettre soit la maison puis le générateur, ou l'inverse, ensuite dans le contenu. Par exemple, « connexion(gen1,maison1) » est équivalent à « connexion(maison1,gen1) » et signifie qu'il y a une connexion entre « maison1 » et « gen1 ».
Pour illustrer cela, le fichier précédemment décrit correspond au réseau électrique illustré sur la Figure 1.

II Automatisation de la recherche de solution

Rechercher une meilleure solution manuellement s'avère également être un exercice fastidieux. L'objectif est donc d'utiliser un algorithme permettant d'automatiser la recherche d'une meilleure solution (i.e., trouver une instance avec un coût plus petit si c'est possible). Nous présentons ici un algorithme naïf pour une résolution approximative du problème.
Intuitivement, à partir d'une instance donnée, on essaye d'améliorer le résultat localement en échangeant les connexions des maisons. Au bout de k (tentatives d')échanges, on s'arrête (autrement, l'algorithme bouclerait indéfiniment).

La fonction `changeConnection(m,g)` permet de remplacer la connexion actuelle de m vers le générateur g passé en paramètre.
La fonction `cout(S)` retourne le coût de S passée en paramètre (voir la phase 1 pour le détail des calculs).

Algorithm 1 Un algorithme approximatif (naïf)

Input: Un réseau électrique $\mathtt{S} = \langle \mathtt{M}, \mathtt{G}, \mathtt{C} \rangle$, un entier positif λ, un entier positif k
Ensure: Le réseau S avec un C potentiellement différent donnant un meilleur coût

```
i ← 0
while i < k do
    Choisir au hasard une maison m ∈ M
    Choisir au hasard un générateur g ∈ G
    S' ← changeConnection(m,g)
    if cout(S) > cout(S') then
        S ← S'
    end if
    i ← i + 1
end while
return S
```

Bien entendu, cet algorithme ne garantit pas de trouver une solution optimale, il permet seulement de s'en approcher (plus ou moins bien). C'est pour cela que l'on parle d'algorithme approximatif. Cet algorithme peut vous servir de base de travail pour proposer un algorithme plus efficace.

III Tâches à réaliser

Pour la seconde étape du projet, vous devez développer un programme qui permet à un utilisateur de configurer le réseau électrique à partir d'un fichier texte (voir Section I). Le programme prend en entrée **sur la ligne de commande** le nom d'un fichier (ou plutôt le chemin vers le fichier texte) dans lequel le réseau électrique a été défini et λ la sévérité de la pénalisation. Rappelons que les paramètres de la ligne de commande sont utilisables via le paramètre `String[] args` de la méthode `main` (voir TP1). Si vous ne passez pas d'arguments en ligne de commande, le programme effectuera le travail que vous avez fait lors de la partie 1 du projet (i.e., construction manuelle du réseau et recherche manuelle d'une meilleure solution).
Avant de passer à la suite, il vous est demandé de vérifier que les informations stockées dans le fichier texte respecte bien le format et ne comporte pas d'erreurs. Voici une liste non exhaustive des erreurs à vérifier :

- le fichier respecte bien la syntaxe définie dans la section I (e.g., l'ordre est bien respecté, pas d'oubli de point à la fin de chaque ligne, ne pas avoir d'éléments autres que generateur, maison, connexion) ;
- avoir le bon nombre de paramètres pour chaque élément ;
- les arguments passés en paramètre de connexion doivent avoir été définis ;
- ...

De même, il faut vérifier que le réseau respecte les contraintes imposées dans la partie 1 du projet (e.g., une maison doit être connectée à un unique générateur).
Si vous constatez une erreur, avant d'arrêter le programme, il faudra alors la signaler à l'utilisateur et expliquer l'origine du problème accompagné de la ligne du fichier à laquelle l'erreur est survenue.

Une fois que le fichier est considéré comme correct, celui-ci est lu et un menu à trois options est proposé à l'utilisateur :

1) résolution automatique ;

2) sauvegarder la solution actuelle ;

3) fin.

Dans le cas où l'option 1 est retenue, un algorithme de votre choix (possiblement inspiré de l'algorithme décrit précédemment, mais ce n'est pas obligatoire) propose une solution ainsi que le coût de celle-ci. **Plus votre algorithme sera efficace, plus vous aurez de points concernant cette partie.** Après cela, on revient au menu.
Quand l'option 2 est sélectionnée, le programme demande à l'utilisateur le nom d'un fichier (différent de celui décrivant le réseau), et y enregistre la solution actuelle. Le contenu du fichier devra respecter le même format que celui de la section I.
Enfin, si l'option 3 est sélectionnée, le programme s'arrête.

Il est nécessaire de gérer toutes les erreurs. Par exemple, quand le menu propose trois options, il faut indiquer à l'utilisateur que sa réponse est incorrecte s'il essaye de taper 3. Il est également demandé de gérer les exceptions, donc (par exemple) si l'utilisateur tape 1.5 ou « abc » au lieu d'un nombre entier, il faut gérer l'exception qui est levée.

Bonus (optionnel) : Un groupe a la possibilité d'obtenir des points bonus si :

- le code est convenablement couvert par des tests unitaires ;
- une interface graphique (JavaFX) du programme est proposée ;

Veuillez noter que ces tâches sont optionnels et les points seront appliqués uniquement si l'ensemble des fonctionnalités demandées a été correctement implémenté (par exemple si la résolution automatique retourne un score qui ne correspond pas à l'affectation faite alors il n'y aura aucun point bonus même si une interface graphique et des tests unitaires ont été implémentés).

IV Remise du projet

La deuxième partie du projet doit être réalisée par les mêmes binômes/trinômes que la première partie. Votre code source, correctement documenté, sera à remettre sur Moodle au plus tard le 21 décembre 2024 à 23h59, sous forme d'une archive jar ou zip (un seul dépôt par binôme/trinôme). Avant d'exporter votre projet sous forme d'archive, pensez à le renommer en utilisant vos noms !
En plus du code source, vous devrez fournir un fichier README qui précisera :

- la classe qui doit être utilisée pour exécuter votre programme (c'est-à-dire où se trouve la méthode main),
- si vous avez implémenté un algorithme de résolution automatique plus efficace que celui proposé dans ce sujet et, si oui, expliquer celui-ci ;
- les fonctionnalités vous avez correctement implémentées, et lesquelles sont manquantes ou présentent des problèmes.

Le non-respect de certaines consignes pourra être pénalisé par un malus dans la note du projet.