// Objectifs

- modélisation en python d'un automate sur lequel vous appliquerez les principaux points du cours.
- de charger la description d'un automate sous forme d'un fichier texte (texte brut, json, xml...) dont vous définirez le format
- d'afficher l'automate à l'écran ou de générer un fichier image
- d'afficher / lister les états accessibles et co-accessibles
- de compléter l'automate
- de déterminiser l'automate en détaillant les étapes
- de sauvegarder la description d'un automate sous forme d'un fichier texte dont le format respecte celui en lecture

// Travail à réaliser

/ Modélisation d'un automate

Créer une classe `Automate` qui permet de modéliser un automate fini. Cette classe doit contenir les attributs suivants :

- `alphabet` : l'alphabet de l'automate : un ensemble de chaînes de caractères. Vous pouvez utiliser la classe `set` de python pour représenter cet ensemble.
- `etats` : l'ensemble des états de l'automate.
- `initiaux` : l'ensemble des états initiaux de l'automate.
- `terminaux` : l'ensemble des états terminaux de l'automate.
- `transitions` : l'ensemble des transitions de l'automate. Attention à choisir une bonne structure de données pour représenter les transitions, cela pourra fortement impacter la complexité de fonctions par la suite.

La classe doit contenir les méthodes suivantes :

- Un constructeur qui prend en paramètre un alphabet.
- Une méthode `ajouter_etat(self, id, est_initial=False, est_terminal=False)` qui ajoute un état à l'automate. L'état est identifié par un identifiant unique. L'état peut être initial et/ou terminal.
- Une méthode `ajouter_transition(self, source, symbole, destination)` qui ajoute une transition à l'automate. La transition est définie par un état source, un symbole de l'alphabet et un état destination. Cette méthode peut lever une exception si la transition est invalide (état source ou destination non existante, symbole non présent dans l'alphabet).
- Vous pouvez également prévoir des méthodes simplifiant la tâche comme `symbole_transition(self, source, destination)` qui retourne le symbole de la transition entre deux états ou `destination_transition(self, source, symbole)` qui retourne l'état destination d'une transition à partir d'un état source et d'un symbole.
- Une méthode `__str__(self)` qui affiche l'automate à l'écran au format suivant :

```
Alphabet: {'d', 'a', 'c', 'b'}
Etats: {'4', '3', '2', '1'}
Etats initiaux: {'1'}
Etats terminaux: {'3'}
Transitions:
    1 --(a)--> 2
    1 --(b)--> 4
    2 --(a,b)--> 2
    2 --(d,c)--> 3
    4 --(d,c)--> 3
```

/ Génération d'un fichier image

Il est possible d'exporter l'automate sous forme d'un fichier image. Pour cela, vous pouvez utiliser la librairie `graphviz` qui permet de générer des graphes à partir d'un fichier texte au format `dot`. Pour cela, vous devez installer la librairie `graphviz` sur votre machine. Vous pouvez utiliser la commande suivante pour installer la librairie sur votre machine :

```
pip install graphviz
```

Pour générer l'image, il faut définir un fichier texte au format `dot` qui contient la description du graphe. Pour cela, vous devez ajouter une méthode `to_dot(self)` à la classe `Automate` qui retourne la description du graphe au format `dot`.

Pour cela vous pouvez commencer un graphe avec `dot = graphViz.diagraph()` et ajouter les éléments du graphe avec les méthodes `dot.node("nom du noeud", shape="...")` (shape peut être "circle", "point", "doublecircle", etc.) et `dot.edge(source, destination, label=symbole)`. Pensez également à ajouter un `dot.attr(rankdir="LR")` pour afficher le graphe de gauche à droite (question de lisibilité).

Par exemple, le programme suivant :

```
automate = Automate.Automate(['a', 'b', 'c', 'd'])
automate.ajouter_etat('1', est_initial=True)
automate.ajouter_etat('2')
automate.ajouter_etat('3', est_terminal=True)
automate.ajouter_etat('4')
automate.ajouter_transition('1', 'a', '2')
automate.ajouter_transition('1', 'b', '4')
automate.ajouter_transition('2', 'a', '2')
automate.ajouter_transition('2', 'b', '2')
automate.ajouter_transition('2', 'c', '3')
automate.ajouter_transition('2', 'd', '3')
automate.ajouter_transition('4', 'c', '3')
automate.ajouter_transition('4', 'd', '3')
print(automate.to_dot().source)
```

Doit afficher :

```
digraph {
    rankdir=LR
    3 [shape=doublecircle]
    4 [shape=circle]
    __1__ [shape=point]
    1 [shape=circle]
    2 [shape=circle]
    __1__ -> 1
    1 -> 2 [label=a]
    1 -> 4 [label=b]
    2 -> 2 [label="b,a"]
    2 -> 3 [label="c,d"]
    4 -> 3 [label="c,d"]
}
```

Ce qui permettra par la suite d'obtenir l'image du graphe suivant :

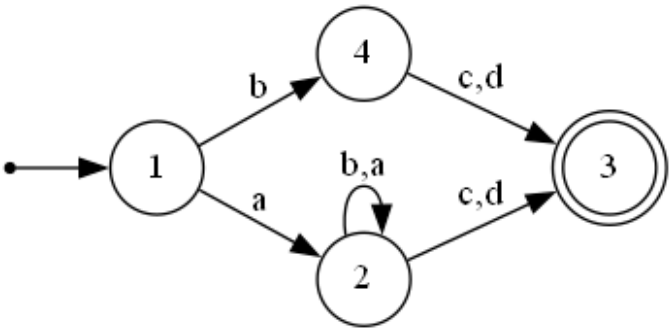

Figure 1: automate

Pour obtenir cette image, ajoutez une méthode `to_png(self, filename)` à la classe `Automate` qui génère l'image dans le fichier `filename`. Pour cela, vous pouvez utiliser la méthode `render(filename, format="png")` de l'objet

retourné par votre méthode `to_dot()`.

/ Sauvegarde et chargement d'un automate

Pour charger et sauvegarder un automate, il convient de définir un format de fichier texte qui permet de décrire un automate qui soit facile à écrire, mais aussi à lire. L'idée est de stocker les informations nécessaires pour qu'à la lecture on puisse facilement recréer un automate.

Voici un exemple de fichier texte qui décrit un automate :

```
d a b c
4
3 0 1
2 0 0
4 0 0
1 1 0
8
1 a 2
1 b 4
2 a 2
2 b 2
2 d 3
2 c 3
4 d 3
4 c 3
```

Comprenez-vous le format de ce fichier ? Il décrit exactement le graphe précédent. La première ligne contient les symboles de l'alphabet, la deuxième ligne contient le nombre d'états, les lignes suivantes contiennent les états et indiquent si l'état est initial et/ou terminal, la ligne suivante contient le nombre de transitions, les lignes suivantes contiennent les transitions (état source, symbole, état destination).

Vous avez maintenant toutes les informations pour définir les méthodes `sauvegarder(self, filename)` et `charger(self, filename)` de la classe `Automate`.

Comment écrire un fichier texte ?

Pour écrire un fichier texte, vous pouvez utiliser la méthode `write()` de l'objet `file` retourné par la fonction `open(filename, mode="w")`. Par exemple, pour écrire le fichier `test.txt` avec le contenu `Hello World !`, vous pouvez utiliser le code suivant :

```python
f = open("test.txt", mode="w")
f.write("Hello World !")
f.close()
```

Une autre manière de faire est d'utiliser le mot clé `with` qui permet de fermer automatiquement le fichier à la fin du bloc :

```python
with open("test.txt", mode="w") as f:
    f.write("Hello World !")
```

Comment lire un fichier texte ?

Pour lire un fichier texte, vous pouvez utiliser les méthodes `read()`/`readline()`/`readlines()` de l'objet `file` retourné par la fonction `open(filename, mode="r")`. Par exemple, pour lire le fichier `test.txt` :

```python
f = open("test.txt", mode="r")
content = f.read()
f.close()
```

Une autre manière de faire est d'utiliser le mot clé `with` qui permet de fermer automatiquement le fichier à la fin du bloc :

```
with open("test.txt", mode="r") as f:
    content = f.read()
```

Si vous souhaitez lire le fichier ligne par ligne, vous pouvez utiliser la méthode `readline()` :

```
with open("test.txt", mode="r") as f:
    premiere_ligne = f.readline()
    deuxieme_ligne = f.readline()
```

/ Compléter un automate

Pour compléter un automate, il faut ajouter des transitions pour tous les symboles de l'alphabet pour tous les états qui ne sont pas déjà définis. Pour cela, on rajoute généralement un état "Puit" où toutes les transitions manquantes vont.

Ainsi un algorithme de completion en pseudo-code pourrait être :

```
ajouter un etat puit
pour chaque etat de l'automate
    pour chaque symbole de l'alphabet
        si il n'existe pas de transition de l'etat avec le symbole
            ajouter une transition de l'etat avec le symbole vers l'etat puit
```

Vous devez maintenant compléter la méthode `completer(self, symbole_puit)` de la classe `Automate` pour qu'elle complète l'automate.

Si vous l'appliquez à l'automate précédent, vous obtiendrez l'automate suivant :

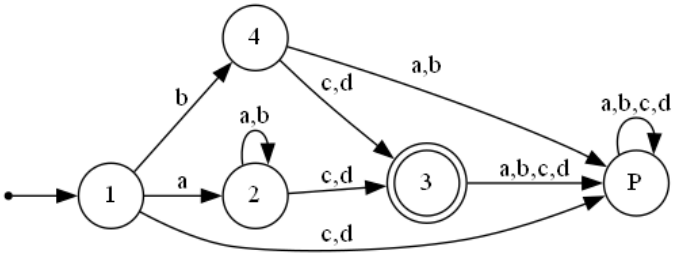

Figure 2: automate

/ Déterminisation d'un automate

Un automate est dit déterministe si pour chaque état et pour chaque symbole, il n'y a qu'une seule transition possible. Il doit également posséder un seul état initial.

Ajoutez une méthode `est_deterministe(self)` à la classe `Automate` qui retourne `True` si l'automate est déterministe, `False` sinon.

Ajoutez une méthode `determiniser(self)` à la classe `Automate` qui retourne une version déterministe de l'automate.

Par exemple, l'automate du cours :

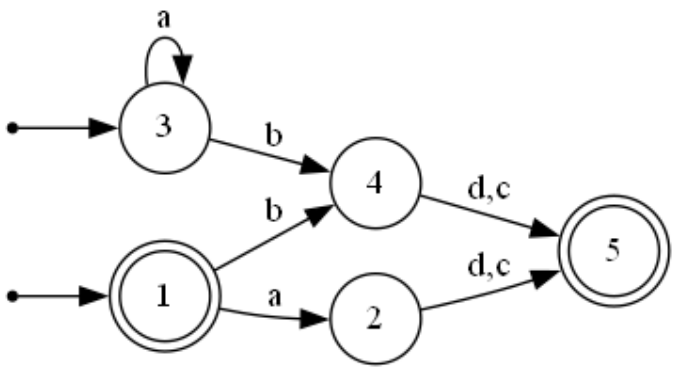

Figure 3: L'exemple du cours - Non-deterministe

devient après déterminisation :

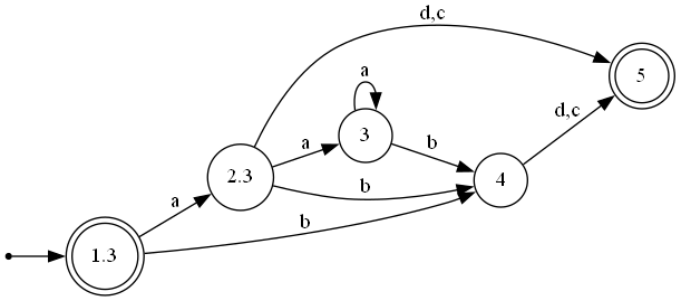

Figure 4: L'exemple du cours - Deterministe

/ Vérification d'un mot

Ajoutez une méthode `accepte_mot(self, mot)` à la classe `Automate` qui retourne `True` si le mot est accepté par l'automate, `False` sinon.

Par exemple, l'automate précédent accepte le mot `abc`, mais pas le mot `aa`.

/ Application

Créez un automate qui vérifie si une adresse mail est une adresse mail de l'université. Le format d'une adresse mail de l'université est le suivant : `prenom.nom[numero]@lacatholille.fr` où `prenom` et `nom` sont des chaines de caractères composées de lettres minuscules et du symbole "-". `numero` est un nombre qui peut être présent en cas de doublon. Le numéro commence par un chiffre non nul et peut être suivi de plusieurs chiffres.

Par exemple, les adresses suivantes sont des adresses mail de l'université :

- `lucien.mousin@lacatholille.fr`
- `jean.dupont2@lacatholille.fr`

Les adresses suivantes ne sont pas des adresses mail de l'université :

- `123.truc@lacatholille.fr`
- `dark.sasuke@gmail.com`