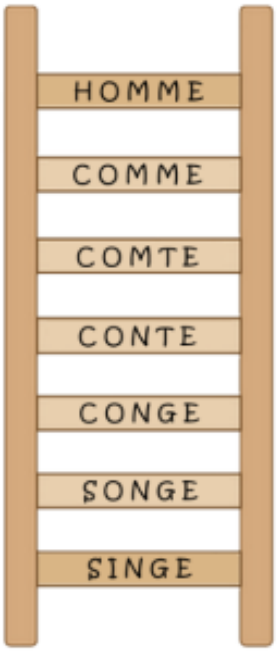
Structures abstraites 3
TP2 : Résolution de labyrinthes

---

1) Introduction : Les échelles de mots



Un doublet, ou échelle de mots (Word Ladder Puzzle en anglais) est un jeu inventé par Lewis Carroll.
La première mention de ce jeu apparaît dans son journal le 12 mars 1878.
Le jeu est publié pour la première fois le 29 mars 1879 dans le magazine britannique Vanity Fair.

Il s'agit de trouver une chaîne de mots reliant deux mots donnés, où à chaque étape les mots ne diffèrent que d'une seule lettre, sans changer la place des lettres.

Par exemple, pour relier EXOS à MATH, on peut créer une chaîne de 8 échelons :
EXOS, EROS, GROS, GRIS, GAIS, MAIS, MATS, MATH.

Nous allons écrire un programme qui recherche les chaînes les plus courtes entre deux mots...

2) Simulation : l'échelle de mots reliant les mots 'ours' et 'cage'

Nous commencerons avec une liste de 110 mots de 4 lettres :

```
DICO = ["aime", "auge", "baie", "brie", "bris", "bure", "cage", "cale",
"came", "cape", "cime", "cire", "cris", "cure", "dame", "dime", "dire",
"ducs", "dues", "duos", "dure", "durs", "fart", "fors", "gage", "gaie",
"gais", "gale", "gare", "gars", "gris", "haie", "hale", "hors", "hure",
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
"pris", "pues", "purs", "rage", "raie", "rale", "rame", "rape", "rare",
"rime", "rire", "sage", "saie", "sale", "sape", "sari", "scie", "sure",
"taie", "tale", "tape", "tare", "tari", "tige", "toge", "tore", "tors",
"tort", "trie", "tris", "troc", "truc"]
```

Sur mon site web, téléchargez echelle.py et enregistrez-le.

Question 1
Figurez-vous que vous pouvez utiliser des données directement depuis GitHub sans les télécharger ! Cela vous sera utile pour certains projets. Il faut utiliser le module requests et utiliser l'URL de la version **RAW** des fichiers GitHub.

Fichier : https://github.com/bouillotvincent/coursNSI/blob/master/dico110.dic
RAW : https://raw.githubusercontent.com/bouillotvincent/coursNSI/master/dico110.dic

a. Grâce à une recherche sur internet, rappelez comment fonctionne .split()

b. Faites fonctionner l'importation du dictionnaire correctement.

Question 2
Nous aurons besoin de la "distance" entre deux mots. Si les deux mots n'ont pas le même nombre de lettres, on renvoie -1. Sinon, on compte le nombre de lettres différentes.
Dans echelle.py, complétez la fonction `distance`.

Exemple :
- *patate et informatique ont une distance de -1*
- *aide et bide ont une distance de 1*
- *montre et mentor ont une distance de 3 (même si ce sont des anagrammes)*

Question 3
Nous allons construire le graphe qui va nous permettre de gagner au jeu de Lewis Carroll.
Le graphe sera représenté par un dictionnaire d'adjacence dont les clés (=sommets) seront les mots d'un dictionnaire "Larousse" et les valeurs (=sommets voisins) les mots de la liste qui sont à une "distance" de 1 (ils ne diffèrent que d'une lettre de la clé).

En important la classe GrapheDico via le module créé en cours, complétez la fonction `genGraphe` permettant de générer le graphe tel que décrit ci-dessus. Vous utiliserez la méthode ajouteArc.

Question 4
Après avoir vérifié que le module GViz était importé, représentez graphiquement le graphe correspondant à d110, le dictionnaire de 110 mots de 4 lettres. Organisez les sommets suivant le mode "Grille" et vérifiez que votre programme a bien fait ce qui était attendu.

Question 5
Il est temps de parcourir le graphe ! Nous connaissons deux algorithmes : le parcours en profondeur et le parcours en largeur. Nous voulons construire un chemin dans le graphe afin de trouver une réponse au jeu de l'Échelle de mots…
a. Quelle est donc la méthode de GrapheDico qui pourrait nous être utile ?
b. Quel type de parcours est utilisé dans cette méthode ?
c. Implémentez votre solution et affichez le résultat pour les mots "ours" et "cage".

Vous avez du trouver :
```
['ours', 'purs', 'pers', 'pere', 'pore', 'tore', 'tors', 'mors', 'mars',
 'mare', 'rare', 'rire', 'rime', 'lime', 'lame', 'pame', 'pape', 'tape',
 'tale', 'sale', 'saie', 'gaie', 'gage', 'cage']
```

3) Amélioration des résultats

Catastrophe, notre programme est nul. Avec mes petits neurones, j'ai trouvé :
```
['ours', 'durs', 'dure', 'mure', 'mare', 'mage', 'cage']
```

Question 6
a. Expliquez pourquoi le parcours en profondeur ne donne pas systématiquement la chaine de mots la plus courte.
b. Quel serait l'avantage du parcours en largeur dans notre cas ? Pourquoi peut-on dire que le parcours en largeur nous fournit le chemin de longueur minimal ?

Question 7
Tournons nous donc vers le parcours en largeur. Nous allons devoir retourner dans grapheDico.py et créer une méthode construireCheminMinimal(self, u, v) qui trouve le chemin minimal entre deux sommets u et v.

Cette méthode ressemblera à construireChemin et fera appel à la méthode parcoursLargeur. Il faudra modifier parcoursLargeur afin que cette méthode renvoie un dictionnaire (appelé vus) qui associe à chaque sommet le sommet qui a permis de l'atteindre (pour la première fois)[1].

a. Modifiez parcoursLargeur dans grapheDico.

b. Créez une méthode construireCheminMinimal(self, u, v)

c. Implémentez votre nouvelle solution et affichez le résultat pour les mots "ours" et "cage".

4) Si on jouait pour de vrai ?

On a trouvé mieux que ce que j'ai trouvé à la main :
`['ours', 'murs', 'mars', 'mare', 'mage', 'cage']`

Tout à l'air de bien se passer. Essayons de jouer avec n'importe quel mot.

Question 8
Trouvons un nouveau dictionnaire. Celui à cette adresse est à peu près complet : https://github.com/hbenbel/French-Dictionary

a. Essayez de trouver l'échelle de mots entre "poulet" et "renard".

b. Essayez de trouver l'échelle de mots entre "régentes" et "réputées". En attendant que votre programme termine, répondez à la question **b.**

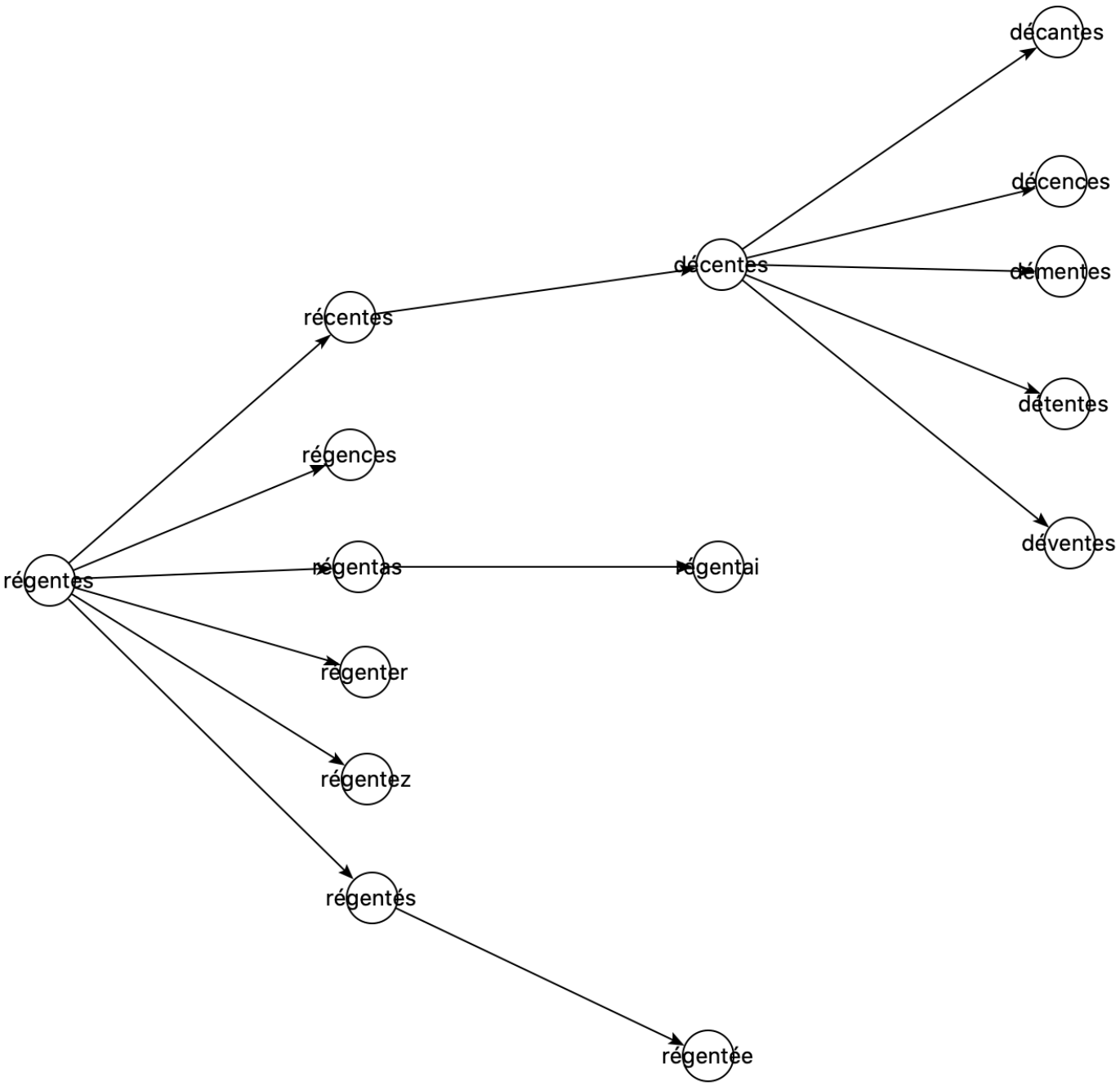
c. Quelle est la complexité de la fonction genGraphe ? Combien d'opérations va-t-il effectuer sachant que le dictionnaire fait 336532 mots ?

d. Proposez quelques idées pour améliorer la situation. Retestez votre programme.

Question 9
Vous pouvez arrêter votre code, il ne finira pas avant 10 jours.
La réponse était :
```
['régentes', 'récentes', 'décentes', 'démentes', 'démenées', 'déminées',
 'débinées', 'débitées', 'débutées', 'députées', 'réputées']
```

Pour trouver cette solution, j'ai adopté un algorithme différent : l'arbre a été construit au fur et à mesure en partant de "régentes". Pouvez-vous proposer un algorithme permettant de réaliser cela ?

[1] voir page 16 du cours.


régentes
récentes
régences
régentas
régenter
régentez
régentés
régentée
régentai
décentes
décantes
décences
démentes
détentes
déventes