Merci à M. Fleck pour le libre partage des sources qui inspirent ce sujet

Le *World Wide Web* (ou plus simplement « Web ») est un ensemble de pages internet, identifiées de manière unique par leurs adresses web ou URL pour *Uniform Ressource Locator*[1], et reliées entre elles par des hyperliens.
Le Web est souvent modélisé par un graphe orienté dont les sommets sont les pages web et les arcs sont les hyperliens entre page.
Le Web étant grand et sauvage, on s'intéresse souvent à des sous-graphes du Web obtenus en *naviguant* sur le Web, c'est-à-dire en le parcourant page par page en suivant les hyperliens d'une manière bien déterminée. Ce parcours du Web pour en collecter des sous-graphes est réalisé de manière automatique par des logiciels autonomes appelés *Web crawlers* ou *crawlers* en anglais, ou « collecteurs » en français.

Mise en place

Vous disposez d'un fichier `web.py` dont vous pouvez consulter le contenu mais qui vous ne **devez pas** modifier.
Vous devez compléter le fichier `crawler.py` dans Spyder pour effectuer les tests. Pour le bon fonctionnement ce ceux-ci, les deux fichiers python devront être dans un même dossier et vous devez utiliser `runfile` (par la flèche verte ou la touche F5) pour exécuter votre fichier de script (sinon vous aurez le droite à une `ImportError` qui signifie que python n'a pas trouvé le module `web.py`)

I Quelques outils

Exercice 1
Compléter la fonction `aplatir(L)` qui prend en argument une liste `L` de couples où le premier élément du couple est un objet quelconque et le second une liste (éventuellement vide) d'objets quelconques. La fonction doit renvoyer une liste où tous les objets sont mis à la suite. Un exemple valant plus qu'un long discours :

```
L = [('a', [1, 2, 3]), ('b', ['answer', 42]), (813, [])]
aplatir(L)
# retourne ['a', 1, 2, 3, 'b', 'answer', 42, 813]
```

Exercice 2
Compléter la fonction `unique(L)` qui prend en argument une liste `L` d'objets non-mutables (pouvant servir comme clé d'un dictionnaire) et renvoie un dictionnaire dont les clés sont les éléments de `L` et la valeur correspondante est l'ordre de première apparition dans `L` (à commencer par 0).

À nouveau, un petit exemple permet de clarifier les choses :

```
L = ['zz', 'xy', 'zz', 't', 't', 'xy']
unique(L)
# retourne {'xy': 1, 't': 2, 'zz': 0}
```

Remarquer que la valeur associée à la clé 't' est 2 (son ordre d'apparition) et non 3 (l'indice de première occurrence).

II Un crawler

Nous allons à présent implémenter un crawler simple en Python. Nous disposons de la fonction

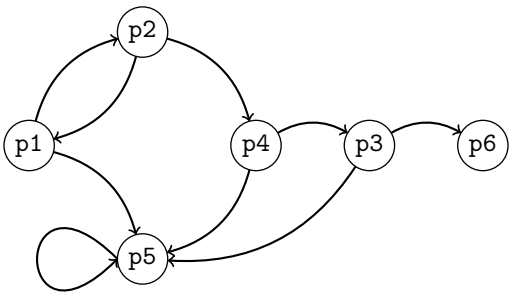
```
web.trouve_liens(p)
```

qui prend en argument une chaîne de caractère représentant une URL `p` complète et qui renvoie la liste des URL `q` telles qu'il existe un hyperlien de `p` vers `q`.
Les hyperliens retournés sont classés par ordre alphabétique.

Un exemple simple

On considère un sous-ensemble du web composé de 6 pages nommées $p1, \dots p6$ et reliées comme suit :

1. de la forme https://prepa.blaise-pascal.fr/pt/info/



Un appel à `web.trouve_liens("p1")` retournera la liste `["p2", "p5"]`.

Un *crawler* est un programme qui, à partir d'une URL, parcourt le graphe du Web en visitant progressivement les pages dont les liens sont présents dans chaque page rencontrée, en suivant une stratégie de parcours de graphe (par exemple, largeur d'abord, ou profondeur d'abord). À chaque nouvelle page, si celle-ci n'a pas déjà été visitée, tous ses hyperliens sont récupérés et ajoutés à une liste de liens à traiter. Le processus s'arrête quand une condition est atteinte (par exemple, un nombre fixé de pages ont été visitées). Le résultat renvoyé par le crawler, que l'on définira plus précisément plus loin, est appelé un *crawl*.

Exercice 3
En 2008, un utilisateur de Wikipédia (l'encyclopédie libre en ligne) a remarqué qu'en démarrant d'une page quelconque du Wikipédia en anglais et en cliquant systématiquement sur le premier lien du texte de la page, on finissait presque toujours (dans 97% des cas en 2016 [2]) sur l'article *Philosophy*. S'agit-il d'un parcours en largeur ou en profondeur du graphe associé au site Wikipédia ? Justifier succinctement.

Exercice 4
On décide d'utiliser d'abord une stratégie de parcours en largeur.
Compléter la fonction `web_en_largeur(u, n)` qui prend comme argument une URL `u` et un nombre entier naturel `n`. cette fonction renvoie une liste de longueur au plus n de couples (v, ℓ) où v est l'URL d'une page visitée et ℓ la liste des liens trouvés sur cette page. Les pages doivent apparaître dans l'ordre dans lequel elles sont été visitées.
Pour un parcours en largeur on visite en priorité les pages rencontrées au plus tôt dans l'exploration. Le crawler visiter au plus n pages distinctes et donc appeler au plus n fois la fonction `web.trouve_liens` (on peut se trouver dans un cas où toutes les pages accessibles ont été visitées sans atteindre n pages en tout).
On utilisera une variable de type dictionnaire pour retenir les pages déjà visitées.
Par exemple, sur le mini-graphe, `web_en_largeur("p1", 4)` pourra renvoyer le résultat

```
[("p1", ["p2", "p5"]), ("p2", ["p1", "p4"]), ("p5", ["p5"]), ("p4", ["p3", "p5"])]
```

Exercice 5
Reprendre l'exercice précédent en complétant la fonction `web_en_profondeur` qui prend les mêmes arguments, mais utilise cette fois une stratégie de parcours en profondeur, c'est à dire que l'on visite en priorité la dernière page découverte.

III Interpréter les résultats

Exercice 6
Compléter la fonction `construit_graphe(crawl)` telle que si `crawl` est le résultat renvoyé par un crawler (une liste de couples formés dune URL v et de la liste des liens récupérés sur la page v), alors `construit_graphe(crawl)` est un couple (ℓ, G) où ℓ est une liste de toutes les URL de pages contenues dans la liste `crawl` et G est la matrice dadjacence du sous-graphe partiel du Web restreint aux pages de la liste ℓ : G_{ij} est le nombre de liens découverts dans le crawl de la page dindice i dans ℓ vers la page dindice j dans ℓ.

- Rappelons que les indices commencent à 0.
- Pour construire une matrice carrée de taille n remplie de 0 on pourra utiliser

```
G = [[0]*n for k in range(n)]
```

- Pour obtenir la liste des clés d'un dictionnaire d on peut utiliser la liste par compréhension

```
[cle for cle in d]
```

- On utilisera les fonctions `aplatir` et `unique` pour obtenir toutes les pages visitées.
- On utilisera la fonction `dico_indices` sur la liste des pages uniques (la liste des sommets du graphe) pour obtenir un dictionnaire contenant les indices dans le graphe.

Donnons un exemple de valeur de retour

2. Voir https://en.wikipedia.org/wiki/Wikipedia:Getting_to_Philosophy

```
crawl = [("p1", ["p2", "p5"]), ("p2", ["p1", "p4"]), ("p5", ["p5"]), ("p4", ["p3", "p5"])]
# la valeur de retour exemple d'un parcours en largeur
construit_graphe(crawl)
```

doit retourner

```
["p1"; "p2"; "p5"; "p4"; "p3"],
[[0, 1, 1, 0, 0],
 [1, 0, 0, 1, 0],
 [0, 0, 1, 0, 0],
 [0, 0, 1, 0, 1],
 [0, 0, 0, 0, 0]]
```

En particulier :

- `p3` apparaît même sil na pas été visité dans le crawl ;
- `p6` n'apparaît pas car il na pas été découvert dans le crawl ;
- l'hyperlien de `p3` à `p5` n'apparaît pas car `p3` na pas été visité.

La suite est facultative

L'algorithme PageRank

PageRank est une manière d'affecter un score à l'ensemble des pages du Web, imaginée par Sergey Brin et Larry Page, les fondateurs du moteur de recherche Google. L'introduction de PageRank a révolutionné la technologie des moteurs de recherche sur le Web. Nous allons maintenant implémenter le calcul de PageRank.

Étant donnée une partie du Web (où l'ensemble des pages est indexé entre 0 et $n-1$), la matrice de surf aléatoire dans cette partie du Web est la matrice M de taille $n \times n$ définie comme suit :

- S'il n'y a aucun lien depuis une page Web d'indice i, alors pour tout j, on pose $\mathrm{M}_{ij} = 1/n$.
- Sinon, s'il y a k_i liens depuis la page Web d'indice i, alors pour tout j, on pose

$$\mathrm{M}_{ij} = (1-d) \times \frac{\mathrm{G}_{ij}}{k_i} + \frac{d}{n},$$

où G_{ij} est le nombre de liens depuis la page d'indice i vers la page d'indice j et d est un nombre réel fixé appartenant à $[0,1]$ (on prend souvent $d = 0{,}15$).

Cette matrice peut être vue comme décrivant la marche aléatoire d'un surfeur sur le Web. À chaque fois que celui-ci visite une page Web :

- Si cette page ne comporte aucun lien, il visite une page Web arbitraire, choisie aléatoirement de façon uniforme.
- Si cette page comporte au moins un lien, il visite avec une probabilité égale à $1-d$ un des liens sortants de cette page, et avec une probabilité égale à d une page Web arbitraire, choisie aléatoirement de façon uniforme.

Exercice 7
Compléter la fonction `surf_aleatoire(d, G)` telle que si d est un nombre entre 0 et 1, et si G est la matrice d'adjacence d'un sous-graphe partiel du Web, alors `surf_aleatoire(d, G)` renvoie la matrice M de surf aléatoire dans ce sous-graphe.

Exercice 8
Compléter la fonction `multiplie(v, M)` une fonction prenant en argument un vecteur ligne v de taille n et une matrice M de taille $n \times n$ et renvoyant le vecteur ligne w de taille n résultant du produit de v par la matrice M : $w = v\mathrm{M}$. En d'autres termes,

$$\forall j \in [\![0, n-1]\!] \qquad w_j = \sum_{i=0}^{n-1} v_i \, \mathrm{M}_{ij}$$

Le PageRank des pages d'un sous-graphe du Web à n pages se calcule par des multiplications successives d'un vecteur ligne par la matrice de surf aléatoire M de ce sous-graphe. Plus précisément, soit ε un nombre réel strictement positif (par exemple, $\varepsilon = 10^{-4}$) et soit $v^{(0)}$ le vecteur ligne de taille n dont toutes les composantes valent $1/n$. On pose pour un entier naturel p arbitraire $v^{(p)} = v^{(0)}\,\mathrm{M}^p$. L'algorithme de PageRank calcule la suite des $v^{(p)}$ pour $p = 0, 1, \ldots$ jusqu'à ce que la norme de la différence soit inférieure à ε (c'est à dire $\|v^{(p+1)} - v^{(p)}\| \leqslant \varepsilon$) et renvoie alors le vecteur $v^{(p+1)}$, considéré comme le vecteur des scores de PageRank. On peut montrer que l'algorithme termine dès lors que d est strictement positif.

Exercice 9
Compléter la fonction `distance(v1, v2)` qui renvoie la distance entre les vecteurs lignes `v1` et `v2`, calculée comme norme de la différence.

On utilisera, pour un vecteur $a = (a_1, \ldots, a_n)$, $\|a\| = \sqrt{\sum_{k=1}^{n} a_i^2}$.

Exercice 10
Compléter la fonction `pagerank(epsilon, M)` prenant en argument un nombre $\varepsilon > 0$ et une matrice M de surf aléatoire d'un sous-graphe du Web et renvoyant le vecteur des scores de PageRank pour ε et M. La fonction `pagerank` devra faire appel à la fonction multiplie précédemment codée.

Exercice 11
Compléter la fonction `calcule_pagerank(d, epsilon, crawl)` telle que l'appel renvoie une liste de couples (u, s), un couple pour chaque URL découverte dans le crawl `crawl`, où u est lURL de cette page et s son score de PageRank. Ici, d et ε sont les deux paramètres nécessaires au calcul de la matrice de surf aléatoire et du PageRank respectivement. On pourra faire appel à l'ensemble des fonctions précédentes.