Projet algèbre linéaire

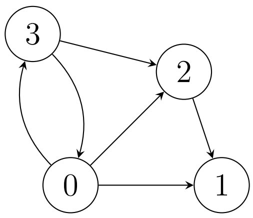
Le but de ce projet est de programmer une version simplifiée de l'algorithme *page rank* utilisé par les moteurs de recherche pour déterminer l'ordre d'affichage des résultats dans son moteur de recherche.

Vous rendrez une copie par binôme de 5 pages maximum contenant les réponses aux questions posées dans le sujet, ainsi que votre code en python au plus tard pour le :

Vendredi 10 janvier 2020

Présentation du problème

En informatique, un **graphe** est un ensemble de **sommets** reliés par des **arcs** ou **arêtes** :



L'ensemble des pages du web peut être vu comme un graphe G dans lequel
- les sommets représentent les pages
- un arc entre une page i et une page j signifie qu'un lien de la page i pointe vers la page j.

Lors d'une recherche de mot clés sur internet, un moteur de recherche sélectionne les pages contenant ces mot clés (on obtient alors un **sous-graphe** du graphe G). Puis il cherche à déterminer l'ordre d'importance des pages sélectionnées.

Pour cela il part de deux principes :
- plus une page possède de liens pointant vers elle, plus elle est importante.
- plus une page est importante, plus les liens qui en parte auront un poids important.

Il va alors calculer un **score** $R = [r_0, r_1, \ldots, r_{n-1}]$ pour chacune des n pages. Ce score devra vérifier les conditions :
- $\forall k \in [\![0, n-1]\!]\ r_k \in [0,1]$
- $\displaystyle\sum_{k=0}^{n-1} r_k = 1$
- Le score d'une page est égal à la somme des contributions des pages pointant vers elle.
- La contribution de la page k possédant p liens sortant est égale à $\dfrac{r_k}{p}$

Dans notre exemple (en supposant qu'il correspond déjà au sous-graphes des pages sélectionnées), cela se traduit par :
- la contribution de la page 0 aux pages 1, 2 et 3 est $\dfrac{r_0}{3}$ (car elle pointe vers 3 pages)
- la page n'a pas de contribution
- la contribution de la page 2 à la page 1 est $\dfrac{r_2}{1}$
- la contribution de la page 3 aux pages 0 et 2 est $\dfrac{r_3}{2}$

Qu'on peut résumer par la matrice :

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 0 & \frac{1}{3}r_0 & \frac{1}{3}r_0 & \frac{1}{3}r_0 \\ 0 & 0 & 0 & 0 \\ 0 & 1r_2 & 0 & 0 \\ \frac{1}{2}r_3 & 0 & \frac{1}{2}r_3 & 0 \end{pmatrix} \end{array}$$

On a donc comme conditions à vérifier pour le score :

$$\begin{cases} r_0 &= \frac{1}{2}r_3 \\ r_1 &= \frac{1}{3}r_0 + \frac{1}{1}r_2 \\ r_2 &= \frac{1}{3}r_0 + \frac{1}{2}r_3 \\ r_3 &= \frac{1}{3}r_0 \\ 1 &= r_0 + r_1 + r_2 + r_3 \end{cases}$$

et $r_0, r_1, r_2, r_3 \in [0,1]^4$

Qu'on peut en partie traduire par :

$$\begin{pmatrix} r_0 & r_1 & r_2 & r_3 \end{pmatrix} \times \begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{pmatrix} = \begin{pmatrix} r_0 & r_1 & r_2 & r_3 \end{pmatrix}$$

Lors de certaines recherches, des pages ne pointent vers aucune autre (comme la page 1 dans notre exemple).

Dans ces situation, notre modélisation est problématique : la contribution de ces pages n'existe pas et le calcul du score sera biaisé.

Pour palier à ce problème, on modifie le calcul des contributions des pages. On pose n le nombre de pages sélectionnées :

- Si la page i n'a pas de lien sortant, sa contribution pour toutes les autres pages est $\dfrac{r_i}{n}$
- Sinon, sa contribution vaut :
  — $0.2\dfrac{r_i}{n}$ pour les pages vers lesquelles elles ne pointent pas.
  — $0.2\dfrac{r_i}{n} + 0.8\dfrac{r_i}{p}$ pour les pages vers lesquelles elles pointent.

Dans notre exemple, $n = 4$. Le calcul des contributions devient alors :

$$P = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 0.2/4 & 0.2/4 + 0.8/3 & 0.2/4 + 0.8/3 & 0.2/4 + 0.8/3 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 0.2/4 & 0.2/4 + 0.8/1 & 0.2/4 & 0.2/4 \\ 0.2/4 + 0.8/2 & 0.2/4 & 0.2/4 + 0.8/2 & 0.2/4 \end{pmatrix} \end{array}$$

Le choix des valeurs 0.2 *et* 0.8 *est arbitraire, on aurait pu prendre n'importe quel couple* $(\lambda, 1-\lambda)$ *avec* $\lambda \in]0,1[$.

Une fois la matrice des contributions calculée, il reste à déterminer un vecteur $R = [r_0, r_1, r_2, r_3] \in [0,1]^4$ vérifiant $R \times P = R$ et $r_0 + r_1 + r_2 + r_3 = 1$.

Les pages seront ensuite triées et affichées par score décroissant.

Modélisation

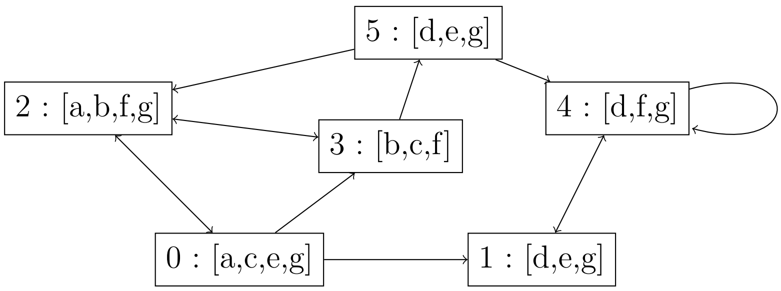
Pour simplifier le problème, nous allons considérer que les n pages web que nous allons indexer sont données sous la forme suivante :

- chaque page web est représentée par un entier compris entre 0 et $n-1$
- un tableau `mots_cles` à n cases contient les listes des mot-clés de chaque page web *`mots_cles[i]` contient la liste des mots clés présents dans la page n°i*
- une matrice `mat_adjacence` carrée de taille n représentant les liens entre les pages web :

$$\texttt{mat\_adjacence[i][j]} = \begin{cases} 1 & \text{si la page } \texttt{i} \text{ possède un lien vers la page } \texttt{j} \\ 0 & \text{sinon} \end{cases}$$

De plus, chaque mot-clé est un composé d'un seul caractère ('a','b','c', etc.).

Exemple le graphe suivant représentent les mots-clés et les liens entre 6 pages web.



Dans notre cadre, cette situation est modélisée par :

```
mot_cles = [[a,c,e,g],[d,e,g],[b,f,g],[a,b,c,f],[d,f,g],[d,e,g]]
mat_adjacence = [[0., 1., 1., 1., 0., 0.],
                 [0., 0., 0., 0., 1., 0.],
                 [1., 0., 0., 1., 0., 0.],
                 [0., 0., 1., 0., 0., 1.],
                 [0., 1., 0., 0., 1., 0.],
                 [0., 0., 1., 0., 1., 0.]])
```

Énoncé du problème

La recherche par mot-clé s'effectue en plusieurs étapes :

1. Sélection des pages contenant au moins un des mots clés : par exemple si on recherche les mots clés 'a' ou 'b' dans la situation précédente, on obtient les pages 0, 2 et 3.

2. Calcul de la sous-matrice M correspondante : si les pages sélectionnées sont 0, 2 et 3, on ne garde dans la matrice de adjacence que les lignes et colonnes correspondantes :

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \longrightarrow M = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

3. Transformation de la matrice d'adjacence M en une matrice de transition P avec :

$$P_{ij} = \begin{cases} 0.8 \times \frac{M_{ij}}{\sum_{j=1}^{m} M_{ij}} + 0.2 \times \frac{1}{m} & \text{si } \sum_{j=1}^{m} M_{ij} \neq 0 \\ \frac{1}{m} & \text{sinon} \end{cases}$$

où m est la taille de M.
Dans notre exemple, on obtient :

$$P = \begin{pmatrix} 0 + 0.2 \times \frac{1}{3} & 0.8 \times \frac{1}{2} + 0.2 \times \frac{1}{3} & 0.8 \times \frac{1}{2} + 0.2 \times \frac{1}{3} \\ 0.8 \times \frac{1}{2} + 0.2 \times \frac{1}{3} & 0 + 0.2 \times \frac{1}{3} & 0.8 \times \frac{1}{2} + 0.2 \times \frac{1}{3} \\ 0 + 0.2 \times \frac{1}{3} & 0.8 \times \frac{1}{1} + 0.2 \times \frac{1}{3} & 0 + 0.2 \times \frac{1}{3} \end{pmatrix} = \begin{pmatrix} \frac{2}{30} & \frac{14}{30} & \frac{14}{30} \\ \frac{14}{30} & \frac{2}{30} & \frac{14}{30} \\ \frac{2}{30} & \frac{26}{30} & \frac{2}{30} \end{pmatrix}$$

4. Calcul d'un score (*rank*) $R = [r_{i_1}, r_{i_2}, \ldots]$, pour chaque page sélectionnée vérifiant $R \times P = R$ et $r_{i_1} + r_{i_2} + \cdots = 1$. *$R$ est un vecteur ligne.*

Question Vérifiez que le vecteur $R = [\frac{5}{21}, \frac{9}{21}, \frac{7}{21}]$ convient pour notre situation.

5. Tri des pages selon la valeur dans R : ici l'ordre des pages retourné est 2,3 puis 0 car $\frac{9}{21} > \frac{7}{21} > \frac{5}{21}$
La difficulté de ce processus est le calcul effectif du vecteur R.

1 Fonctions annexes `python`

Code Construisez une fonction `select_pages` prenant en paramètre une liste `key_words` représentant les mots-clés contenus dans chacune des pages, une liste `key_words_searched` représentant les mots-clés à chercher et retournant la liste des pages contenant au moins l'un des mots-clés recherchés.

Code Construisez une fonction `select_matrix` prenant en paramètre une matrice `mat` représentant les liens entre des pages web, une liste de pages `pages` et retournant une matrice ne contenant que les lignes et les colonnes dont les numéros sont présents dans `pages`.

Code Construisez une fonction `get_transition_matrix` prenant en paramètre une matrice M correspondant à la matrice d'adjacence du graphe et calculant la matrice de transition correspondante : P

Le code de la fonction `sort_pages`*, prenant en paramètre une liste de pages* `pages`*, leur rang calculé* `rank` *et retournant les pages dans l'ordre décroissant selon leur rang vous est fourni.*

2 Première méthode de calcul

On cherche à déterminer une méthode pour calculer un vecteur R vérifiant $R \times P = R$.

Question En transformant l'égalité précédente, déterminer une matrice N telle que $R \times N = \mathbf{0}$.

Ainsi ${}^tN \times {}^tR = 0$ et on peut ainsi appliquer l'algorithme du pivot de Gauss pour trouver une solution pour ${}^tR$.

Question Quelle solution immédiate convient pour R ?

Question Ajoutez une ligne à la matrice ${}^tN$ de manière à prendre en compte la contrainte que $R[0] + R[1] + \cdots = 1$.

On pourra utiliser la méthode `np.concatenate`*.*

Code Implémentez votre solution en utilisant votre méthode de Gauss et testez-la sur les exemples fournis.

Pour utiliser des fonctions présentes dans un fichier `fichier_auxiliaire.py`*, il suffit de placer ce fichier dans le répertoire contenant votre projet* `projet... .py` *et d'y ajouter au début la commande* `from fichier_auxiliaire import *`

3 Seconde méthode de calcul

Une seconde méthode de calcul pour obtenir R consiste à partir d'une solution arbitraire et de la faire évoluer jusqu'à une valeur satisfaisante.

On considère l'algorithme suivant :

Algorithme 1 : page_rank(P,epsilon)

n =taille(P);
$R0 = [1/n, 1/n, \ldots, 1/n]$;
$R1 = R0 \times P$;
tant que $max(|R1 - R0|) > epsilon$ **faire**
  $R0 = R1$;
  $R1 = R0 \times P$;
fin
retourner $R1$

Question

a. Justifier que la somme de n'importe quelle ligne de la matrice P vaut toujours 1.

b. Si toutes les sommes des lignes de la matrice P valent 1, et que la somme des coordonnées de $R0$ vaut aussi 1, montrer que la somme des coordonnées de $R1 = R0 \times P$ vaut aussi 1.

c. En déduire une preuve par récurrence que dans l'algorithme précédent, $R1$ a toujours la somme de ses coordonnées égale à 1.

Question Que signifie la condition de la boucle "Tant que" de l'algorithme ?

Code Implémentez cet algorithme et tester le sur les exemples fournis avec epsilon $= 10^{-5}$. Retrouve-t-on le même résultat qu'avec la méthode précédente ?

4 Prolongement (Bonus)

Pour déterminer quelle méthode est la plus rapide, vous pouvez effectuer des tests en utilisant le générateur aléatoire de graphe présent dans le fichier `generateur_graphe.py` qui s'utilise de la manière suivante :

`generateurGraphe(nom_fichier,n,key_words,v_m,v_M,kw_m,kw_M)` produit un graphe :

- à `n` sommets ;
- dont chaque sommet a entre `v_m` et `v_M` successeurs ;
- dont chaque sommet a entre `kw_m` et `kw_M` mots-clés, appartenant à l'ensemble `key_words` ;
- sauvegarde le graphe produit dans le fichier `nom_fichier`.

Question Expliquez selon vous quelle méthode est la meilleure en justifiant.

Vous pouvez par exemple :

a. déterminer, au maximum, combien d'opérations élémentaires entre réels (addition, soustraction, multiplication, division) sont effectuées lors d'une résolution de système de taille $n \times n$ par la méthode de Gauss.

b. déterminer, au maximum, combien d'opérations élémentaires entre réels sont effectuées lors d'une multiplication entre un vecteur et une matrice.

c. déterminer en moyenne (en faisant par exemple des simulations), combien de multiplications sont nécessaires pour que la seconde méthode rende un résultat.

d. comparer les résultats obtenus.