TP1 – Le collecteur

M2 informatique – Université Paris Cité

Année 2023-2024

Corpus

On utilisera le fichier *dump* de toutes les pages Wikipédia françaises de début 2024, qui se trouve ici :
`https://dumps.wikimedia.org/frwiki/latest/frwiki-latest-pages-articles.xml.bz2`.

Attention, le fichier compressé pèse un peu plus de 5 Go… On appellera `frwiki.xml` le fichier décompressé. On trouvera un court extrait sur moodle (attention, il s'agit du fichier `frwiki.xml` tronqué aux 10.000 premières lignes sans considération pour les balises fermantes, ce n'est donc pas un fichier xml bien formé).

Traiter le fichier entier entraîne des complications que nous éviterons dans ce cours (gestion fine de la mémoire, etc.). La première étape consiste donc à choisir un sous-ensemble pertinent des pages Wikipédia, appelé « le corpus ».

Exercice 1 *Sélection et nettoyage*

1. Comprendre la structure du fichier `frwiki.xml`.
2. À partir du fichier `frwiki.xml`, choisir un sous-ensemble cohérent de pages : par exemple celles contenant tel ou tel mot, etc. Le sous-ensemble devra contenir de 200.000 à 800.000 pages.
3. Nettoyer le fichier de toutes les informations qui ne serviront pas pour le projet.

Quelques conseils

- Le choix du langage est libre. Des langages différents peuvent même être employés pour différentes parties du projet. Go semble être un bon choix si vous maîtrisez ce langage. Java est plus lourd pour coder mais convient aussi. Si vous utilisez Python, la gestion de la mémoire est plus compliquée, il faut utiliser des bibliothèques efficaces ou parser le fichier par petits bouts.
- Voici quelques éléments (non exhaustifs) de syntaxe du xml Wikipédia.
  - Un article est encadré par les balises `<page>` et `</page>`, il possède un titre et un identifiant (balises `<title>` et `<id>`), son contenu est encadré par les balises `<text>` et `</text>`.
  - À l'intérieur des balises `<text>`, les commandes `&lt;` et `&gt;` sont utilisés à la place de `<` et `>` respectivement (essentiellement pour les balises `<ref>` qui sont des liens externes).
  - Le titre d'une section est entouré par un certain nombre de signes `=` (entre 2 et 5) selon son niveau.
  - Un lien interne vers l'article Article est donné ainsi : `[[Article]]` ou `[[Article|texte de remplacement]]`.

— Les doubles accolades sont utilisées pour de la mise en page (des dates notamment), deux apostrophes sont insérées pour l'italique, trois pour le gras.
— Pour ce projet, nous pouvons nous contenter des balises `<title>` et `<text>` (et éventuellement `<id>` si cela vous arrange). Le reste peut être supprimé. On peut également supprimer sans crainte le contenu entre double accolades, les liens externes (balises `ref`), les liens internes commençant par `[[Mot_clé:titre…` (catégories, fichiers, etc.), les sections « Notes et références », « Voir aussi », « Bibliographie », « Articles connexes » et « Liens externes »…
— On mettra tout en minuscules et on supprimera les symboles spéciaux (ponctuation notamment), les espaces superflus, etc.
— On supprimera enfin les pages dont le contenu (entre balises `<text>`), après nettoyage, contient moins de 600 mots.

Dictionnaire

Exercice 2 *Dictionnaire*

1. Dresser la liste des mots sur lesquels pourront porter les requêtes de l'utilisateur. On pourra par exemple prendre les 20 000 mots les plus fréquents apparaissant dans le corpus, dont tous ceux contenus dans le titre des pages. On retiendra également la fréquence d'apparition (nombre d'occurrences) de chacun de ces mots dans le corpus.
2. Supprimer de la liste les mots « vides » (petits mots non discriminants) comme *le*, *la*, *un*, *de*, *sa*, etc. (On enregistrera pour la suite l'ensemble des mots enlevés.)
3. *(Optionnel)* Écrire une fonction permettant de trouver la « racine » des mots : par exemple, on enlèvera la conjugaison (*mangerai* → *manger* ; *buvions* → *boire*), la fin des adverbes (*longuement* → *long*), le pluriel, le féminin, etc.

   On pourra trouver des fonctions de *stemming* et *lemmatisation* déjà programmées pour s'épargner cette tâche.
4. *(Optionnel)* Remplacer chaque mot par sa racine.
5. *(Optionnel)* Écrire une fonction qui prend en entrée un mot mal orthographié et renvoie le mot de la liste « le plus proche ».

Exercice 3 *IDF*

Le coefficient IDF (*Inverse Document Frequency*) d'un mot m dans un corpus de documents D est défini comme suit :

$$IDF(m) = \log_{10}\left(\frac{|D|}{|\{d \in D \ : \ m \in d\}|}\right)$$

(en supposant que m apparaisse dans au moins un document du corpus).

Calculer et stocker le coefficient IDF de chaque mot du dictionnaire.

Matrices creuses

Le graphe des pages visitées sera représenté par une matrice stochastique d'adjacence (cf. TP 2). C'est un graphe orienté : s'il y a un arc du sommet i vers le sommet j, alors le coefficient (i, j) de la matrice d'adjacence est $1/d_i$ (où d_i est le degré sortant de i), sinon ce coefficient est nul.

Or, dans notre cas, cette matrice comporte beaucoup de coefficients nuls. Pour éviter de stocker tous ces coefficients nuls, on adopte un format appelé « CLI » pour les matrices creuses.

Exercice 4 *Graphe*

1. Donner la matrice d'adjacence d'un graphe orienté de votre choix à 4 sommets.
2. Si on indexe n pages, quel est le nombre de sommets du graphe des pages visitées ? Quelle est la taille de la matrice d'adjacence ? Pour $n = 10^9$, peut-on stocker une telle matrice en mémoire ?
3. Si chaque page a 10 liens en moyenne, quel est le nombre de coefficients non nuls dans la matrice précédente ? Si on enlève tous les zéros, peut-on stocker les coefficients en mémoire pour $n = 10^9$?

Soit n un entier. On veut stocker des matrices réelles $n \times n$, avec ces deux contraintes :
— accès facile à la ligne i de la matrice ;
— on ne stocke pas les zéros.

Soit M une matrice $n \times n$ ayant m entrées non nulles. On code M sous forme de :
— l'entier n ;
— un tableau C de m `float` (NB : un stockage simple précision est suffisant pour nos calculs) ;
— un tableau I de m `int` (suffisant pour nos calculs) ;
— un tableau L de $(n+1)$ `int` ;

qui vérifient les règles suivantes :
— la première ligne (respectivement colonne) est la ligne (resp. colonne) numéro 0 ;
— C contient les contenus de toutes les cases non nulles de la matrice M : d'abord ceux de la première ligne de M, puis ceux de la deuxième ligne, etc. ;
— $L[i]$ est l'indice du début de la i-ème ligne de M dans C : cette ligne s'étend donc de $C[L[i]]$ inclus à $C[L[i+1]]$ exclu ;
— mais les éléments non nuls de la ligne i de la matrice M ne sont pas nécessairement consécutifs : afin de pouvoir retrouver leur place et que le codage avec ces trois tableaux soit un codage exact de la matrice, on utilise le tableau I, qui contient les indices des colonnes associées aux éléments non nuls de la ligne i ;
— si $C[k]$ est la j-ème case de la ligne i alors $I[k] = j$;
— $L[n+1] = m$ (afin de traiter la dernière ligne comme les autres) ;
— si la ligne i ne contient que des zéros, on a $L[i] = L[i+1]$.

Exemple de matrice M :

0	3	5	8
1	0	2	0
0	0	0	0
0	3	0	0

tableau L :

0	3	5	5	6

tableau C :

3	5	8	1	2	3

tableau I :

1	2	3	0	2	1

Exercice 5 *Matrices*

1. Donner la représentation CLI de la matrice suivante :

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 3 & 0 & 4 \\ 0 & 5 & 6 & 7 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

2. Définir le type `matrice` sous forme CLI et créer la matrice ci-dessus.
3. Définir le type `vecteur` et programmer le produit d'une matrice (sous forme CLI) par un vecteur.

Collecteur

Il s'agit maintenant de parcourir l'ensemble des pages du corpus. Pour chaque page visitée et chaque mot du dictionnaire, si ce mot apparaît dans la page on retiendra sa fréquence d'apparition. Pour cela, à chaque mot du dictionnaire sera associée la liste des pages qui contiennent ce mot, avec la fréquence d'apparition du mot dans cette page (voir l'exercice 8). Cette structure sera appelée *la relation mots-pages* (voir la figure 1).

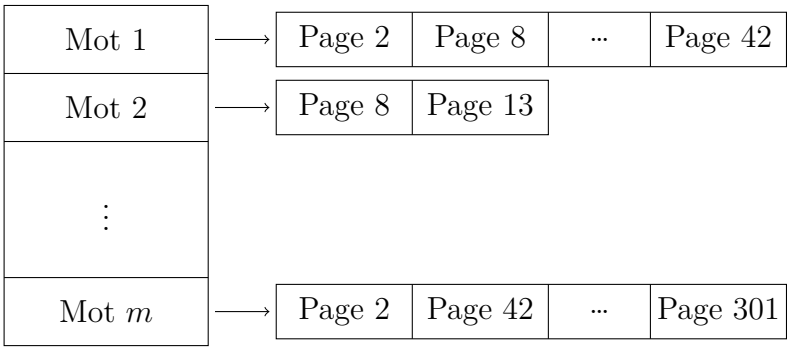


FIGURE 1 – La relation mots-pages.

Exercice 6 *Taille des structures*

1. Dans notre cas, quels sont les sommets du graphe ? Quels sont les arcs ?

   Donner le nombre de sommets et d'arêtes de votre graphe. Calculer la taille du codage CLI de la matrice qui correspond.

2. S'il y a m mots dans le dictionnaire, n pages visitées, et si chaque page contient en moyenne 500 mots *différents* du dictionnaire, combien d'éléments la relation mots-pages va-t-elle contenir ?

   Ce nombre dépend-il de m ? Est-il raisonnable de vouloir indexer toutes les pages du corpus ?

Exercice 7 *Exploration*

1. Il n'est pas commode de toujours manipuler les adresses des pages visitées : mieux vaut associer à chaque page un numéro. De même pour les mots du dictionnaire.

   Choisir une structure de donnée efficace pour stocker la relation mots-pages.

2. Écrire une fonction qui liste tous les liens internes contenus dans le texte d'une page Wikipédia.

3. Écrire une fonction qui, à partir d'une page Wikpédia, remplit grâce aux mots présents dans le texte la relation mots-pages pour cette page.

   Si un mot n'est pas présent dans notre dictionnaire, on l'ignore.

   (Optionnel) On utilisera les fonctions programmées à l'exercice 2 pour considérer seulement les racines des mots et pour corriger les éventuelles fautes d'orthographe.

4. Programmer enfin le parcours du fichier pour explorer toutes les pages du corpus. On remplira au fur et à mesure la représentation CLI de la matrice d'adjacence du graphe, ainsi que la relation mots-pages.

   Une fois le parcours effectué, bien penser à enregistrer le résultat sur le disque (graphe et relation) pour pouvoir le réutiliser plus tard.

5. Comment auriez-vous procédé si le fichier `frwiki.xml` n'était pas disponible, c'est-à-dire si vous deviez parcourir les vraies pages internet ?

Exercice 8 *TF*

Si m est un mot et d une page du corpus, le coefficient TF (*Term Frequency*) de ce mot dans cette page est défini par

$$TF(m,d) = \begin{cases} 0 & \text{si } m \notin d \\ 1 + \log_{10}(\#occ(m,d)) & \text{sinon,} \end{cases}$$

où $\#occ(m,d)$ est le nombre d'occurrences de m dans d.

1. Pour chaque mot m du dictionnaire et chaque page d contenant m, calculer le coefficient $TF(m,d)$.
2. Pour chaque page d, calculer
$$N_d = \sqrt{\sum_m TF(m,d)^2}$$
la norme du vecteur associé à d (la somme est prise sur l'ensemble des mots m contenus dans d).
3. Pour chaque mot m du dictionnaire et chaque page d de la relation mots-pages, stocker la valeur $TF(m,d)/N_d$ (coefficient TF normalisé).
4. *(Optionnel)* De la liste des pages associées à un mot m, supprimer les pages d dont la valeur
$$IDF(m) \times TF(m,d)/N_d$$
est trop faible (expliquer la démarche et faire des tests pour bien choisir le seuil).

TP2 – Le pagerank

M2 informatique – Université Paris Cité

Année 2023-2024

Matrices initiales

Soit G un graphe orienté dont les sommets sont numérotés de 0 à $n-1$. La matrice d'adjacence de G est la matrice A^0 carrée de taille n telle que :

— $A^0_{i,j} = 0$ si (i,j) n'est pas un arc de G ;

— et si (i,j) est un arc alors $A^0_{i,j} = 1/d_i$ (où d_i désigne le degré sortant du sommet i).

Rappel : cette matrice creuse est représentée sous forme « CLI ».

On ne travaillera pas directement avec A^0, car elle peut contenir des lignes de zéros correspondant aux sommets sans arc sortant. Nous considérerons donc la matrice A stochastique équiprobable associée à G, carrée de taille n, telle que :

— si le sommet i n'a pas de voisins sortants, alors $A_{i,j} = 1/n$ pour tout j ;

— sinon :

· $A_{i,j} = 0$ si (i,j) n'est pas un arc de G,

· et si (i,j) est un arc alors $A_{i,j} = 1/d_i$ (où d_i désigne le degré sortant du sommet i).

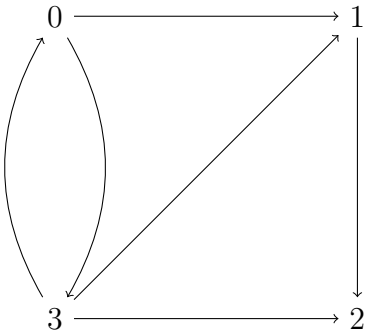
On trouvera un exemple à la figure 1. Par rapport à A^0, on a remplacé les lignes de coefficients nuls par des lignes de coefficients $1/n$. Pour représenter A, on utilisera simplement la représentation CLI de A^0 (la ligne i est vide ssi $L[i] = L[i+1]$).

Produit matrice-vecteur

Exercice 1 *Transposée*

Si M est une matrice creuse (donnée sous forme CLI) et V un vecteur, nous voulons faire le calcul non pas de $P = MV$ mais de $P = ({}^tM)V$. Or transposer une matrice est coûteux. Nous allons évaluer $({}^tM)V$ **sans calculer explicitement** la transposée de M :

$$P[i] = \sum_{j=0}^{j=n-1} M_{ji} V[j].$$



$$\begin{pmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \end{pmatrix}$$

FIGURE 1 – Un exemple de graphe orienté avec sa matrice A associée.

Calculer $P[i]$ revient à parcourir une colonne de M, peu efficace en représentation CLI. Mais si on parcourt la **ligne** i on peut faire $P[j]$ += $M_{ij}V[i]$ en ayant initialisé P au vecteur nul. On fait m fois de suite ces incréments.

1. Programmer cela, en parcourant donc la matrice M ligne par ligne. Le calcul doit **impérativement** se faire en $O(n+m)$ et en une seule passe des tableaux L, C et I.
2. Modifier votre code pour prendre en compte la liste des lignes « vides » remplacées par des coefficients $1/n$ (matrice A plutôt que A^0).

Pagerank

Soit J la matrice carrée de taille n dont tous les coefficients sont 1. Pour le *pagerank*, nous allons utiliser la matrice

$$A_G = (1-\varepsilon)A + (\varepsilon/n)J,$$

où $\varepsilon \in]0,1[$ est un réel bien choisi (de l'ordre de $1/7$). Cela permet d'obtenir une matrice stochastique dont les coefficients sont tous strictement positifs, et garantit ainsi l'existence et l'unicité du vecteur que l'on cherche (théorème de Perron-Frobenius).

Attention, la matrice A_G n'est plus creuse. Pour représenter A_G, on se servira de la matrice creuse A^0 (plus précisément de la matrice A dont la représentation est la même, cf. ci-dessus) et du réel ε.

Exercice 2
Donner la matrice A_G pour le graphe de la figure 1.

Rappelons maintenant l'algorithme de pagerank.

Données : une matrice A_G, une distribution initiale de probabilités Π_0 et un entier k.
Résultat : le vecteur Π de pagerank avec une certaine précision.
début
  $\Pi \leftarrow \Pi_0$;
  pour *i de 1 à k* **faire**
    $\Pi \leftarrow ({}^tA_G)\Pi$
  fin pour
  renvoyer Π
fin

Exercice 3 *Pagerank*

1. Programmer l'algorithme du pagerank en partant de la distribution Π_0 uniforme sur les n sommets.

   Attention à la représentation de la matrice A_G. La ligne $\Pi \leftarrow ({}^tA_G)\Pi$ de l'algorithme doit être calculée grâce à

   $$\Pi \leftarrow (1-\varepsilon)({}^tA)\Pi + (\varepsilon/n)\mathbb{1}$$

   où $\mathbb{1}$ est le vecteur dont tous les coefficients sont égaux à 1.
2. Tester votre algorithme sur des graphes bien choisis.

Application

Exercice 4 *Poids des pages*

1. Choisir (en expliquant votre choix) la valeur de ε et le nombre d'itérations k, et faire tourner l'algorithme de pagerank (exercice 3) sur le graphe des pages collectées au TP 1.
2. Vérifier sur quelques pages arbitraires que le résultat semble cohérent.
3. On a ainsi obtenu le pagerank de chaque page. Enregistrer le résultat sur le disque.

TP3 – La recherche

M2 informatique – Université Paris Cité

Année 2023-2024

Traitement de la requête

Exercice 1 *Requête*

La requête de l'utilisateur consiste en un ensemble de mots.

1. Enlever de cet ensemble les mots redondants et les mots « vides » (voir TP 1).

   (Optionnel) Pour chaque mot, si nécessaire en corriger l'orthographe si vous avez programmé ces fonctions au TP 1. Sinon, on considérera seulement les requêtes à base de mots corrects.

   (Optionnel) Pour chaque mot, le remplacer par sa racine si vous avez programmé ces fonctions au TP 1.

2. Selon le temps dont vous disposez : programmer l'une des variantes ci-dessous pour calculer le résultat de la requête.

Rappel sur les scores

Le score d'une page d par rapport à la requête r est $s(d, r) = \alpha f(d, r) + \beta p(d)$, où $f(d, r)$ est le score obtenu par la fréquence des mots, et $p(d)$ le pagerank de d, avec α et β des coefficients à choisir de manière pertinente.
NB : il peut être judicieux de réduire la variance du pagerank en prenant $p(d)^\gamma$, pour un certain $\gamma < 1$, plutôt que $p(d)$ dans la fonction f. Faire des tests pour trouver la valeur de γ qui augmente la pertinence des résultats.

Le score de fréquence $f(d, r)$ est calculé ainsi (en reprenant les notations du TP 1) :

$$f(d, r) = \Big(\sum_{m \in r} IDF(m) \times TF(m, d) \Big) / N_d,$$

où N_d est la norme du vecteur TF associé à d calculée au TP 1.

Calcul des résultats

Exercice 2 *Version simple*

1. Donner un algorithme *efficace* qui, à partir de la relation mots-pages, énumère toutes les pages contenant *tous* les mots de la requête. On ne fera qu'un seul parcours des listes concernant les mots de la requête.
2. Calculer le score $s(d, r)$ de chacune de ces pages et les trier par score décroissant.

Exercice 3 *Cas d'une requête d'un seul mot*

(Optionnel) Lorsque la requête ne contient qu'un seul mot m, la liste des pages contenant m peut être grande et le calcul ci-dessus prendrait trop de temps.

Au prix d'un espace mémoire accru, on peut précalculer les scores $s(d, m)$ par rapport à la requête m de chaque page contenant m, et stocker une fois pour toute la liste des pages par score décroissant, permettant une réponse immédiate sur ce genre de requête.

Exercice 4 *Version optimisée WAND*

1. Dans la relation mots-pages, trier par ordre de pagerank $p(d)$ décroissant la liste des pages d associées à chaque mot.
2. Pour chaque mot m, parcourir de droite à gauche la liste des pages d associées et calculer pour chacune d'elles le maximum des valeurs $IDF(m) \times TF(m, d)/N_d$ vues jusqu'à présent : on notera $v(m, d)$ ce maximum.
3. Pour accélérer les calculs ci-dessous, on trouvera la position à laquelle avancer chaque pointeur grâce à une recherche dichotomique.
4. Programmer l'algorithme WAND vu en cours pour obtenir les k meilleures pages :
   — un tas t contient les meilleures pages vues jusqu'à présent ;
   — le score de la k-ème meilleure page est γ ;
   — chaque liste de pages pour un mot m de la requête possède un pointeur vers la page d^m en cours ;
   — on trie ces pointeurs par $p(d^m)$ décroissant : on notera $d_1, \ldots, d_n$ les pages pointées dans l'ordre, et $m_1, \ldots, m_n$ les mots correspondants ;
   — soit j minimal tel que $\beta p(d_j) + \alpha \sum_{i=1}^{j} v(m_i, d_i) \geq \gamma$ (on appelle d_j le « pivot ») ;
   — pour i de 1 à $j-1$, avancer le pointeur d_i jusqu'à la première page dont le pagerank est $\leq p(d_j)$;
   — si d_j contient suffisamment de mots de la requête, calculer son score $s(d_j)$;
   — si $s(d_j) > \gamma$, mettre à jour γ et le tas t ;
   — avancer d'un cran tous les pointeurs qui pointent vers d_j ;
   — recommencer.

Déploiement

Exercice 5 *Site*

Programmer un serveur et réaliser un site web fonctionnel où l'utilisateur entre sa requête et où s'affiche la liste des résultats de la recherche. Pour chaque résultat, afficher les valeurs correspondantes du pagerank, du score de fréquence et du score total.

Penser à écrire une fonction de conversion des liens Wikipédia donnés dans le fichier `frwiki.xml` en liens `html` pour qu'on puisse suivre les liens affichés par votre moteur de recherche.

Exercice 6 *Améliorations possibles*

Quels sont les défauts de votre moteur de recherche ? Comment les améliorer ? (Inutile de programmer ces améliorations.)