TP1 – Le collecteur

M2 informatique – Université de Paris

Année 2020-2021

Corpus

On utilisera le fichier *dump* de toutes les pages Wikipédia françaises de début 2021, qui se trouve ici :
`https://dumps.wikimedia.org/frwiki/latest/frwiki-latest-pages-articles.xml.bz2`.

Attention, le fichier compressé pèse environ 5 Go… On appellera `frwiki.xml` le fichier décompressé. On trouvera un court extrait sur moodle.

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.

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 10 000 mots les plus fréquents apparaissant dans le corpus, dont tous ceux contenus dans le titre des pages. On retiendra également la fréquence d'apparition 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. Supprimer les accents, les majuscules et les redondances.
4. Trier cette liste par ordre alphabétique.

5. *(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.
6. *(Optionnel)* Pour chaque mot de la liste, s'il n'est pas sa propre racine, pointer vers la racine.
7. *(Optionnel)* Écrire une fonction qui prend en entrée un mot mal orthographié et renvoie le mot de la liste « le plus proche ».

Matrices creuses

Le graphe des pages visitées sera représenté par sa matrice d'adjacence. 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 3 *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 4 *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. Cette structure sera appelée *la relation mots-pages* (voir la figure 1).

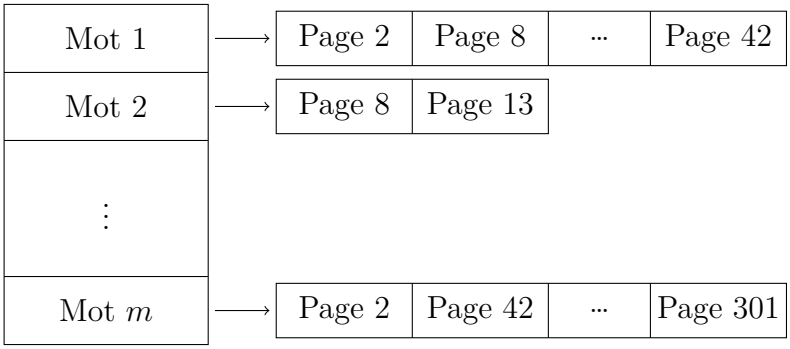

FIGURE 1 – La relation mots-pages.

Exercice 5 *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 200 mots *différents* du dictionnaire, combien d'éléments va avoir la relation mots-pages ?
   Ce nombre dépend-il de m ? Est-il raisonnable de vouloir indexer toutes les pages du corpus ?

Exercice 6 *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.
   Attention à bien ignorer les balises, à enlever les accents, les majuscules, etc. 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 Wikipédia. 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 ?