Rapport de projet:
Méthodes algorithmiques pour l'accès à l'information numérique

Yizhe FAN (21502157), Xiang LI (21512427)
Groupe: Fan-LI

Index

1. Architecture de logiciel

Ce projet contient 3 modules principaux : Module PageRank, Module Collecteur et Module Recherche. Il y a des répertoires et fichiers suivants:

- data: les tests dédié à pagerank et les résultats attendus.
- metaData: le répertoire ignoré qui contients un gros fichier "amazon-meta.txt", dans ce projet, nous avons testé tous les données, par contre le Module Recherche ne prend que les 20 premiers records.
- sortData: Vu que certains fichiers de test contiennent des noeuds non triés, donc il faut les trier et les mettre dans ce répertoire.
- src: tous les fichiers exécutables.
- wordsArchive: tous les dictionnaires anglais.
- wordsIgnore: tous les dictionnaires anglais qui contiennent des mots ignorés.
- config.json: le fichier pour configurer le module Recherche.
- Makefile: executer les codes dans src. Plus de détailles dans README.md

Trois Module Principaux:

➢ Module PageRank

Ce module consiste à la calculation de pagerank à partir d'un fichier qui contient les informations des références de tous les noeuds. Les fichiers doivent en format <.txt> et la premier ligne est # Nodes: [X] Edges: [X].

Dans ce module nous avons besoin des fichiers suivant:
test_pagerank.py:
Entrée: le chemin d'un fichier qui continent les noeuds tries.
Sortie: un vecteur des pageranks.
readFiletoMatrice.py: la classe MatriceCLI sert à convertir les informations des noeuds à trois tableaux C,L,I.
calculatePageRank.py: le fichier pour calculer pagerank.

➢ Module Collecteur

Le collecteur est un map dans lequel les keys sont des mots clé des titres de livre et les valeurs sont une liste d'ID de livre.
test_collecteur.py:
Entrée: les chemins d'amazon-data et des dictionnaires.
Sortie: un collecteur.
collector.py: des fonctions pour transformer les données à un collecteur.
trie.py: une structure pour stocker les mots clé des titres de livre.

➢ Module Recherche

Ce module sert à la recherche.
Entrée: une liste de mots.
Sortie: une liste de id trié par pagerank et aussi par un algorithme.

2. Utilisation de logiciel

Un fichier README.md qui indique des commandes possibles.

(1)Module PageRank:

➔ `make testpagerank` la commande va nous permettre de tester les fichiers dans le répertoire sortData.
➔ `make pagerank file=sortData/xxx.txt` le fichier doit être en format txt.

(2)Module Collecteur:

Il y a trois façon afin de construire un collecteur:
➔ `python3 src/test_collector.py [path_data] [dict1] [dict2] [...] [path_ignore]` On pourrait ajouter autant de dictionnaires que l'on veut.
➔ `make collector data=[path] dict=[path] ignore=[path]` le deuxième paramètre prend un seul dictionnnaire.
➔ `make collector data=[path] dir=[path] ignore=[path]`le deuxième paramètre prend un répertoire où il y a des dictionnaires.

(3)Module Recherche:

Avant de lancer le script, il faut configurer des paramètres: dans la racine, modifier config.json pour tester votre données.
a. pagerank_file: le fichier txt contient tous les noeuds et des références entre eux.
b. amazon_data: amazon meta-data.

c. dict_dir: le répertoire des dictionnaires
d. dict_ignore: le répertoire des dictionnaires devant être ignoré.

➔ `make search words=[word1,word2,word3,...]` attention, la liste des mots doivent séparer par virgule.

le format de résultat: (id,pagerank)
********Searching results********
(15, 1.016473164352883)
(13, 0.03996231621034525)

3. Discussion d'Algorithme de la classification

Il y a deux algorithmes possible qui s'adaptent à ce problème de classification.

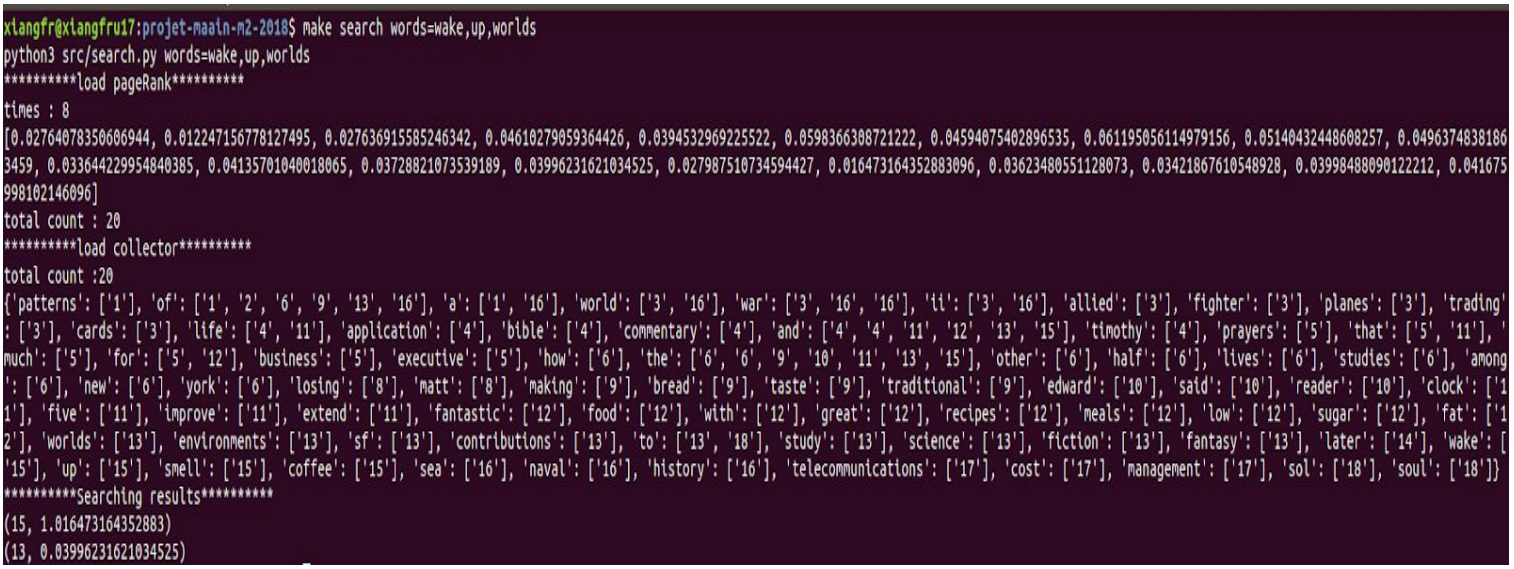
- **Classification par Union**
  On suppossons que l'occurrence est plus important que pagerank.
  Notre algorithme est basé sur deux éléments principaux:
  - pagerank : le product ayant un meilleur pagerank devrait classifier avant.
  - occurrence: si un mot apparaître plusieurs fois dans un product, celui-ci devrait classifier avant.

  Step1: Pour chaque mot dans la liste de recherche, on récupère une liste de ID.
  Step2: Faire une Union de ces listes de ID
  Step3: On va parcourir cette Union, et sauvegarder des informations dans un Map dans lequel Key est le ID, et Value est le pagerank. Si un mot est déjà présenté dans le Map, on augmente son Value de façon Value = Value + 1. (notre calcul de pagerank est entre (0,1) et on peut présumer que l'occurrence a un pagerank de 1)
  Step4: on trie le Map par le Value par ordre décroissant.

★ Avantage: On pourrait trouver tous les produits ayant la relation avec la liste de recherche des mots.

❑ Inconvénient: il est possible que le temps de recherche soit longue.

L'exemple:



- **Classification par Intersection**
  Step1: A partir d'un premier mot dans la liste de recherche, on récupère la liste de ID et on fait un intersection avec la liste de ID du mot suivant.
  Step2: On trie les IDs par pagerank.

★ Avantage: On considère le premier mot dans la liste de recherche est le plus important, le deuxième est moins que le premier et ainsi de suite. On trouve rapidement les produits.

❑ Inconvénient: dans le cas où il y a un mot n'apparaît pas dans le premier mais dans les suivants, cet algorithme ne fonctionne pas très bien.

4. Réponse des questions des TPs

TP-Collecteur

1. **Si vous considérez m mots dans le dictionnaire et n produits tel que les méta-données du produit contient en moyenne 20 mots *différents* du dictionnaire, combien d'éléments aura la relation mots-produits ?**

   m éléments

2. **Ce nombre dépend-il de m ? Est-il raisonnable de vouloir indexer toutes les produits d'Amazon dans le fichier de méta-données ? (Pour information, sur le serveur de l'UFR vous dispo- serez de 10 Go de mémoire vive au maximum.)**

   Oui, il dépend de m.
   Oui, on pense il est raisonnable de vouloir indexer toutes les produits d'Amazon. Soit dans le fichier de méta-données, soit dans une base de donnée pour sauvegarder justement des index. Car quand on a besoin de trouver le pageRank de chaque produits ça sera plus rapid et en fait on va souvent avoir besoin de savoir cet information.

3. **Quelle structure de données vous avez choisit pour la relation mot-produits ? Justifier votre choix.**

   pour mot-produits on a choisit une structure de Map (key-value).
   key: mots
   value: une liste de ID de produits

   Car le complexité d'accder Map est O(1) qui est le plus efficace.
   Et puis on va avoir une list de ID de produits qui correspondent le mot qu'on cherche. Étant donnée qu'on a déjà indexé les produits, on va trouver chaque produits efficacement.

TP-Search

Quels sont les défauts de votre moteur de recherche?

Comment les améliorer ?

1. On a pas implémenté la fonctionnalité de NLP (natural language processing). Ça nous permet de corriger les mots à rechercher mal tapé. Si l'utilisateur ne tape pas les mots strictement corrects qui sont dans le dictionnaire, il va peut-être échouer de trouver ce mot et les produits qu'ils correspondent.

Pour améliorer:

(1) on peut implémenter un Lib de NLP qui va analyser le mot que l'utilisateur tape et puis il peut donner quelque mots (on peut limiter le nombre par la configuration d'algo) qui est plus proche de ce mot comme les candidats.

(2) On va chercher aussi ces mots dans notre collecteur. Par contre on va classifier les résultats qui sont retourné par ces mots moins importants que les résultats qui sont retourné par les mots tapés par l'utilisateur exactement.

2. Notre moteur de recherche maintenant retourne que les résultats par pageRank de produit. Il faut implémenter plus de règles pour la classification selon les internes association entre les produits.

5. Conclusion

On a mise en oeuvre un simple moteur de rechercher selon chaque étape de TP. Ça nous permet de bien pratiquer les connaissances qu'on apprend au cours et de bien connaître les mécanismes de moteur de recherche.