Université Catholique de Louvain-la-Neuve

Projet SINF1250
« Ranking de réseaux sociaux et pages web: PageRank »

Henrard Jonathan 6033-09-00
Outters Charline 7637-09-00

9 mai 2016

1. Introduction

Pour mettre en pratique le chapitre traitant des graphes du cours de *Mathématiques pour l'informatique*. Nous avions comme mission d'implémenter un algorithme très important du web : l'algorithme du PageRank. Il s'agit de l'algorithme phare originaire de Google qui permet d'octroyer un score d'importance à chaque noeud d'un graphe.

2. Algorithme de PageRank

2.1. Origine et principe de l'algorithme

Avec la formation du Web, la quantité des informations disponibles est devenue immense à tel point qu'une branche spécifique de recherche d'information consacrée au Web a vu le jour. Mais la quantité des informations n'est pas la seule raison qui a impliqué la création de cette nouvelle branche de recherche. En plus de contenir un nombre incalculable de pages, ce contenu est dynamique ! Comprenez que contrairement à des livres se trouvant dans une bibliothèque, les pages Web sont constamment en changement. Il est donc impossible d'exécuter des recherches de la même manière dans les deux cas. Une autre différence essentielle entre les informations statiques telles que les livres et les informations dynamiques, est que les premières sont généralement organisées par des spécialistes alors que sur le Web, n'importe quel quidam peut ajouter du contenu. Mais le gros avantage du Web, et le principe fondamental sur lequel va s'appuyer l'algorithme du PageRank, c'est que les pages sont liées entre elles à l'aide d'hyperliens.

Le principe est assez simple : un lien vers une page peut être interprété comme une recommandation de cette page. La suite est assez logique : une page qui est souvent recommandée est plus intéressante (et donc plus populaire) qu'une autre. Mais ce n'est pas tout ! Une recommandation faite par une page populaire a plus de valeur qu'une recommandation faite par une page inconnue. La quantité de recommandations effectuées par une page joue aussi : si une page renvoie vers des centaines d'autres pages, chacun des liens vaudra moins que le lien d'une page vers une unique autre page.

2.2. Approche formelle

Pour commencer, un score $score_i$ est associé à chaque page web i proportionnellement au score moyen pondéré des pages ayant un lien vers i. p_{ij} est le poids du lien entre les pages i et j. $p_{ij} = 1$ si le lien existe et $p_{ij} = 0$ si le lien n'existe pas.

$$score_i \propto \sum_{j=1} \frac{p_{ji} score_j}{p_j}$$
$$p_j = \sum_{i=1} p_{ji}$$
où p_j est le degré sortant de la page j

Le problème ici, c'est la dépendance entre le score d'une page et celui des autres. Par conséquent, il faut utiliser ces équations de manière récursive jusqu'à ce que les résultats se stabilisent (qu'il y ait une convergence) et normaliser le vecteur X.

3. Usage

Pour lancer le programme il faut exécuter la commande suivante :

```
./main [teleportation] [fin]
```

Où la valeur par défaut de `teleportation` est à 1
Où `fin` est la matrice d'entrée. La valeur par défaut étant le fichier "matrix.csv" dans le répertoire courant.

4. Procédure Java

Notre méthode `main()` fonctionne d'une manière très simple. La première étape consiste à extraire les options de la ligne de commande utilisée pour exécuter le programme. Ensuite, il faut générer la matrice pour qu'elle puisse être utilisée dans notre classe `PageRank`. Pour ce faire, la méthode `main()` fait appel à la méthode `load()` qui va charger le fichier. Une fois le fichier chargé, chacune des lignes qui le compose est transposée dans une liste de strings dont chaque entrée est une ligne du fichier. Une fois cette transposition réalisée, on fait appel à la méthode `convertToMatrix()` qui, comme son nom l'indique, va convertir la liste en une matrice.

La particularité ici est que toutes les strings comprises dans la liste doivent être transformées en un tableau de doubles à deux dimensions. Et ce, car la librairie utilisée postérieument l'exige. Par ailleurs cette méthode, profite de parcourir la matrice (représentée dans ce cas, sous forme de liste de String) pour calculer le degré de chaque ligne. Les résultats sont stockés dans les variables de classe correspondantes. `matrix` pour la matrice d'entrée, `degrees` pour les degrés de chaque ligne (représentés sous forme de matrice ligne).

Notre classe `PageRank` possède deux variables d'instance, une variable de classe, et trois méthodes. Les trois variables sont les suivantes :

precision : une constante de classe donc, qui donne la précision (nombre de décimales) du résultat souhaité. Dans ce cas ci, `100` vérifie qu'entre deux itérations, les deux chiffres après la virgule sont les mêmes pour chaque résultat.

matrix : qui est la matrice contenant nos scores.

size : qui représente la taille de la matrice. Il n'y en a qu'une comme les matrices d'ajacence sont carrées.

La première des méthodes est `PageRank()` :

```
1  public PageRank(double[][] m, double[][] d, double prob) {
2     this.matrix = new Matrix(m);
3     this.size = matrix.getRowDimension();
4
5     Matrix p = transitionMatrix(new Matrix(d), prob);
6     Matrix result = powerMethod(p);
7     result.print(0, 5);
8  }
```

Le constructeur prend en arguments deux matrices, la matrice des scores et celle des degrés, ainsi qu'une probabilité. Elle initialise les variables d'instance, en utilisant au besoin la librairie. Puis, elle appelle les deux méthodes pour calculer le PageRank et enfin imprime les résultats sur la sortie standard (ligne 7).

La seconde méthode, `transitionMatrix()`,s'occupe de calculer la matrice de transition qui est la somme de la matrice de probabilité de saut et la matrice de *leap probabilities*. Voici le code de cette méthode :

```
1  private Matrix transitionMatrix(Matrix degrees, double probability){
2     Matrix leapProbabilities = new Matrix(size,size,(1-probability)/size);
3     Matrix linkProbabilities = new Matrix(size,size)
4     for (int i = 0; i < size; i++) {
5        for (int j = 0; j < size; j++) {
6           double linkValue = matrix.get(i, j)*(probability/degrees.get(i, 0));
7           linkProbabilities.set(i, j, linkValue);
8        }
9     }
10     return leapProbabilities.plus(linkProbabilities);
11 }
```

La première étape est de créer la matrice de*leap probabilities* (ligne 2). La nouvelle matrice aura la même taille que la matrice passée à la classe `PageRank` et les valeurs seront égales à

$$\frac{1-probability}{size}.$$

Ensuite, on crée la matrice de probabilité de liens (ligne 3). A nouveau, sa taille sera la même que la matrice de base. Les valeurs la composant seront attribuées au sein de la double boucle qui suit (lignes 4 à 7). Il s'agit de la valeur des liens que l'on obtient par la formule suivante :

$$matrix.get(i,j) * \frac{probability}{degrees.get(i,0)}.$$

Enfin, la méthode retournera la matrice qui sera le résultat de la somme de la matrice de *leap probabilities* et celle de probabilité des liens.

La troisième et dernière méthode est celle qui permet de déterminer les scores. Nous avons choisi la `powerMethod`. Ce choix a été effectué par simplicité. Au lieu de résoudre des équations linéaires, nous multiplions un vecteur (representé dans le programme sous forme de matrice d'une ligne) par la matrice de transition. Ces opérations sont supportées par la librairie (méthode `times()`), ce qui représente moins de lignes de code, plus de clarté et un risque d'erreur moindre.

```
1   private Matrix powerMethod(Matrix transition){
2      Matrix rank = new Matrix(1,size);
3      rank.set(0, 0, 1);
4      double previousRank [][] = new double[1][size];
5      boolean accurate = false;
6      int nbIteration = 0;
7      while (!accurate) {
8         nbIteration++;
9         rank = rank.times(transition);
10        int n = 0;
11        double result [][] = rank.getArray();
12        for (int i = 0; i < result[0].length; i++) {
13           if((int)(result[0][i]*PRECISION) == (int)(previousRank[0][i]*PRECISION)){
14                 n++;
15              }
16        }
17        previousRank = rank.getArray();
18        if (n == result[0].length)
19           accurate = true; // = break
20     }
21     System.out.println("Nombre d'itérations pour la power-method: " + nbIteration);
22     return rank;
23  }
```

Nous avons une matrice ligne `rank`, dont la toute première colonne sera égale à 1 et le reste à 0. On multiplie cette matrice avec la matrice de transition passée en paramètre de la méthode. Le résultat de cette opération écrase l'ancienne matrice ligne `rank`. Cette opération est répétée plusieurs fois.
Afin de détérminer le nombre de répétition à faire, nous avons engloblé cette opération dans une boucle qui vérifie si entre l'itération précédente et l'itération courante les N [1] chiffres après la virgule sont identiques pour chaque valeur de la matrice ligne. Dès que toutes les valeurs de la matrice ligne ne varient plus, le nombre d'itérations effectuées est

1. Où N = log_{10}(`PRECISION`))

imprimé sur la sortie standard et la matrice ligne `rank` est retournée.

5. Librairie de calcul matriciel : Jama

Pour nous faciliter la vie avec le calcul matriciel, nous avons choisi d'utiliser la librairie JAMA. Nous avons utilisé deux constructeurs de cette classe `Matrix` et plusieurs méthodes que nous allons présenter maintenant.
Tout d'abord, présentons les deux constructeurs :

Matrix (int r, int c, double s) — `int r` est le nombre de lignes de la matrice
— `int c` est le nombre de colonnes de la matrice
— `double s` est la valeur que l'on souhaite placer partout dans la matrice

Matrix (int r, int c) — `int r` est le nombre de lignes de la matrice
— `int c` est le nombre de colonnes de la matrice

Présentons maintenant les diverses méthodes qui nous ont été utiles :

double get(int i, int j) retourne un élément de la matrice se trouvant à la ligne i et la colonne j

void set(int i, int j, double s) insère la valeur s dans la matrice à la ligne i et la colonne j

Matrix plus(Matrix b) fait la somme de deux matrices et retourne la matrice contenant les résultats

int getRowDimension() qui, comme son nom l'indique, permet de connaitre le nombre de lignes constituant la matrice

Matrix times(Matrix b) fait le produit de deux matrices et retourne la matrice contenant les résultats

double [ ][ ] getArray() permet d'accéder à une liste à deux dimensions et retourne un pointeur vers la liste à deux dimensions des éléments de la matrice

void print(int w, int d) qui imprime la matrice sur la sortie. Le premier argument est la largeur de la colonne et le deuxième, le nombre de décimales souhaité.

6. Conclusion

L'algorithme PageRank est un des algorithmes les plus connus, et le plus connnu en matière de recherche pour le Web. Son implémentation nous aura permis de voir que son fonctionnement n'est pas si complexe que ça, puisqu'il repose sur la théorie des graphes. La logique qui se cache derrière tous les calculs matriciels est plus difficile à cerner au premier abord, mais comme nous avons du diviser le processus lors de l'implémentation, nous y voyons plus clair maintenant.

Références

[1] A.N. Langville & C.D. Meyer, *Google's PageRank and Beyond*, Princeton University Press.

[2] R. Sedgewick & K. Wayne , *1.6 Case Study : PageRank*, disponible sur : `http://introcs.cs.princeton.edu/java/16pagerank/`.

A. Annexe

A.1. Matrice Bott.csv

Après 9 itérations :

1	0.04830	5	0.07253	9	0.17252
2	0.08714	6	0.03587	10	0.07212
3	0.10458	7	0.03200	11	0.13965
4	0.12976	8	0.10553		

A.2. Matrice Coleman.csv

Après 20 itérations :

1	0.00000	26	0.00000	51	0.00150
2	0.00000	27	0.00000	52	0.00000
3	0.00000	28	0.00000	53	0.00000
4	0.00000	29	0.00000	54	0.00306
5	0.00000	30	0.00000	55	0.00287
6	0.00021	31	0.00000	56	0.00000
7	0.00047	32	0.00000	57	0.00049
8	0.00047	33	0.00000	58	0.00000
9	0.00000	34	0.00000	59	0.00000
10	0.00000	35	0.00000	60	0.00000
11	0.00000	36	0.00000	61	0.00000
12	0.00021	37	0.00000	62	0.00000
13	0.00009	38	0.00084	63	0.11061
14	0.00015	39	0.00000	64	0.12552
15	0.00000	40	0.00000	65	0.00000
16	0.00000	41	0.00000	66	0.16326
17	0.00047	42	0.00000	67	0.16318
18	0.00000	43	0.00000	68	0.00000
19	0.00000	44	0.00000	69	0.16369
20	0.00083	45	0.00195	70	0.08454
21	0.00201	46	0.00000	71	0.16907
22	0.00208	47	0.00000	72	0.00000
23	0.00000	48	0.00000	73	0.00000
24	0.00000	49	0.00244		
25	0.00000	50	0.00000		