Projet Chaînes de Markov : *PageRank*

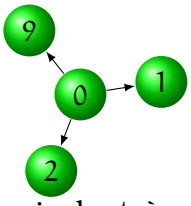
27 novembre 2014

L'une des applications célèbres du modèle des Chaînes de Markov est le *PageRank* de Google.
PageRank est l'algorithme d'analyse des liens concourant au système de classement des pages Web utilisé par le moteur de recherche Google. Il mesure quantitativement la popularité d'une page web. Le *PageRank* n'est qu'un indicateur parmi d'autres dans l'algorithme qui permet de classer les pages du Web dans les résultats de recherche de Google.
Son principe de base est d'attribuer à chaque page une valeur (ou score) proportionnelle au nombre de fois que passerait par cette page un utilisateur parcourant le graphe du Web en cliquant aléatoirement, sur un des liens apparaissant sur chaque page. Ainsi, une page a un *PageRank* d'autant plus important qu'est grande la somme des *PageRanks* des pages qui pointent vers elle (elle comprise, s'il y a des liens internes) [1].
Le but de ce projet est de simuler les calculs de *PageRank* en Java sur de petites instances : des nanoWebs.
Remarques sur le projet Le rendu de ce projet sera un ensemble de classes java et un document pdf zippés et nommé suivant les noms des auteurs.
Remarques sur le rapport Le rapport (pdf) contiendra les réponses aux questions qui ne sont pas de l'implémentation ainsi qu'un rapide tour de l'implémentation et des points particuliers soulevés par celle-ci.
Remarques sur l'implémentation Une attention particulière sera portée à la qualité et à la robustesse du code fournie. Lors de l'évaluation, le code sera testé sur d'autres exemples que ceux fournis dans l'énoncé. En particulier, le nombre de nœud ne doit pas être fixé.

Principes

On considère le Web comme une collection de N pages ($N > 10^{10}$) reliées entre elles par des liens hyper-texte (On dit qu'une page "pointe" vers ces autres pages). L'idée de base utilisée par les moteurs de recherche pour classer les pages par ordre de pertinence décroissante consiste à considérer que plus une page est la cible de liens venant d'autres pages, c'est-à-dire plus il y a de pages qui pointent vers elle, plus elle a de chances d'être fiable et intéressante pour l'utilisateur final, Il s'agit donc de quantifier cette idée, i.e. d'attribuer un score de pertinence à chaque page.
Attention : On ne tient pas compte des "auto-références" : une page qui pointe vers elle-même.
Pour chaque page i, on appelle $r_i \geq 0$ le score de pertinence de la page. *PageRank* propose comme hypothèse que plus une page est pertinente, plus de pages pointent vers elle. De plus, un lien vers la page contribuera d'autant plus à sa pertinence si il provient lui-même d'une page pertinente. Autrement dit, les scores (r_i) servent (i) à ranger les pages de la plus pertinente à la moins pertinente et (ii) à se calculer eux-mêmes récursivement : chaque page distribue son score proportionnellement au nombre de liens dont elle est l'origine vers les pages sur lesquelles elle pointe.

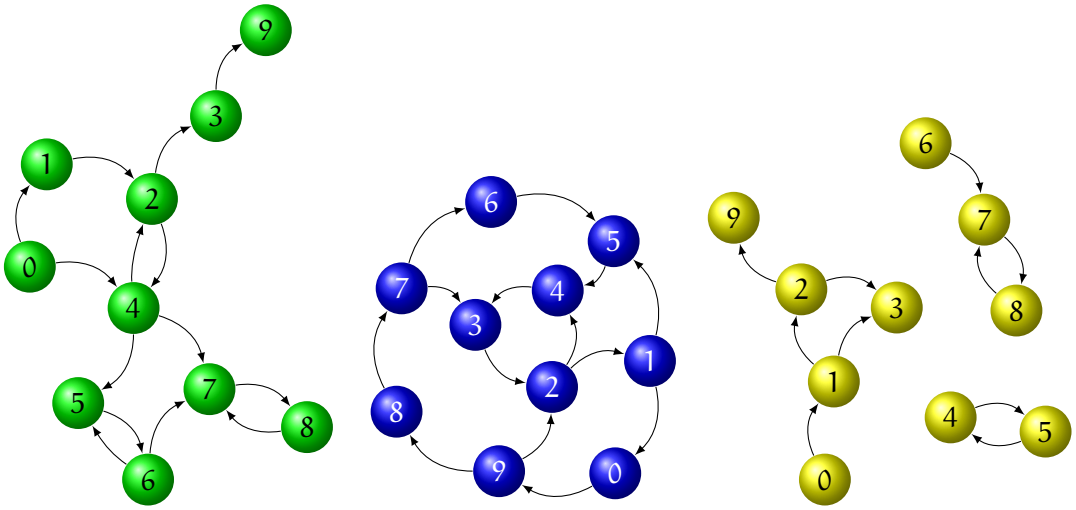
1. From `fr.wikipedia.org`



Exemple : soit le sous-graphe suivant d'un nanoWeb. Il indique que sur la page 0, il existe des liens vers les âges 1, 2 et 9.
Supposons que $r_0 = 9$ alors la contribution du nœud 0 aux scores des nœuds 1, 2, et 9 sera de $\frac{1}{3} \cdot r_0 = 3$.

On remarque alors que le principe est très proche (pour ne pas dire équivalent à une modélisation du Web sous la forme de chaîne de Markov) dans lesquelles les probabilités de transition sont uniformes (faire attention au cas d'une page qui ne pointe sur aucune autre page).

Question1 – Déterminer les matrices de transition des 3 nanoWebs suivant (dont les couleurs ne sont là que pour les séparer et faire joli) :



nanoWeb 1 *nanoWeb 2* *nanoWeb 3*

Question2 – Décrire au mieux ces 3 chaînes de Markov (types de nœud, classes d'équivalences, ensembles absorbants, périodicité, etc.)

Structures de données

semaine 1

Il s'agit maintenant d'implémenter une structure de données qui permettra de représenter correctement ces chaînes de Markov. Comme nous nous intéresserons aux problèmes de petite taille, nous nous permettons de ne pas choisir entre représentation par graphe ou matrice de transition : nous aurons les 2 représentations simultanément dans nos objets `SimpleWeb`.
On supposera que les graphes sont composés de nœuds (identifié par un entier) et d'arcs (orientés, identifiés par un entier `tail` et un entier `head` : `tail`→`head`).

Question3 – Proposer des implémentations des classes java suivantes : `Node`, `Arc`, `SimpleWeb` :

- `Arc` contiendra au moins une référence vers ses 2 `Node` ainsi que la représentation de la probabilité associé à l'arc,
- `Node` contiendra au moins son identifiant, ainsi qu'une liste d'`Arc` entrants et une liste d'`Arc` sortants du nœud,
- `SimpleWeb` contiendra au moins la liste des nœuds du graphe de transition et une représentation de la matrice de transition.

Ces classes doivent permettre de construire les 3 exemples de nanoWebs comme des classes dérivant de `SimpleWeb` :

```
1  public class NanoWeb
2          extends SimplewWeb {
3
4      public NanoWeb(int maxNode) {
5          super(maxNode);
6      }
7
8      public static NanoWeb nanoWeb1() {
9          NanoWeb n1 = new NanoWeb(10);
10         n1.addArc(0, 1);n1.addArc(0, 4);
11         n1.addArc(1, 2);
12         n1.addArc(2, 3);n1.addArc(2, 4);
13         n1.addArc(3, 9);
14         n1.addArc(4, 2);n1.addArc(4, 5);n1.addArc(4, 6);
15         n1.addArc(5, 6);
16         n1.addArc(6, 5);n1.addArc(6, 7);
17         n1.addArc(7, 8);
18         n1.addArc(8, 7);
19         return n1;
20     }
21     public static void main(String[] args) {
22         NanoWeb n=nanoWeb1();
23         n.updateProbas();
24         n.showTransitionTable();
25     }
26 }
```

Les méthodes nécessaires dans `SimpleWeb` sont donc les suivantes :

- `SimpleWeb(int max)` qui construit un CMTD de `max` nœuds (pour l'instant sans arc),
- `AddArc(int tail,int head` qui construit cet arc (sans probabilité !) et qui met à jour les différentes références nécessaires,
- `updateProbas()` qui calcule l'ensemble des probabilités de transition des arcs suivant la proposition faite par *PageRank*,

Vérifier (tests unitaires ?) que les matrices de transition que vous obtenez pour les 3 nanoWebs ci-dessus sont bien celles que vous avez calculées précédemment.

Remarques sur la robustesse Par exemple, deux `addArc` sur les même nœuds devraient lever une exception ; rajouter un arc doit invalider les probabilités et la matrice de transition, etc.

Simulations

semaine 1

On voudrait simuler le déplacement d'un internaute dans notre `SimpleWeb`. Le principe est simple : on place l'internaute dans un nœud initial, puis on utilise les distributions de probabilités de transition pour décider à chaque pas de temps où il se rend. L'idée est alors de garder le nombre de fois où l'internaute passe dans chaque nœud, ce qui permettrait de construire une distribution de probabilités (si convergence).

Question4 – Quels sont les différents comportements que vous pouvez attendre de la part de cette simulation ? En particulier, que pouvez-vous dire des problèmes que rencontrerait une telle simulation sur les 3 exemples de nanoWebs.

Question5 – Implémenter un nouvelle classe `Internaute` qui, à partir d'un `SimpleWeb`, va lancer une navigation. Cette navigation peut prendre fin pour deux raisons différentes : soit on a atteint un nombre limite de navigation soit on a atteint une convergence pour l'estimation de la distribution de probabilité. En notant π_t et π_{t+1} les estimations aux temps t et $t+1$, on peut proposer comme seuil :

$$\epsilon = \max_i |\pi_t(i) - \pi_{t+1}(i)|$$

Question6 – Garder certaines valeurs de ϵ (une toutes les 100 itérations par exemple) afin de produire des courbes indiquant les convergences au cours du temps pour les 3 nanoWebs.

Il sera bien évidemment nécessaire d'enrichir au moins la classe `SimpleWeb`. Le code d'utilisation attendu ressemblera à celui-ci :

```
1 SimpleWeb w;
2 // ../.. creation du microWeb
3
4 Internaute bob(w);  // bob se ballade dans w
5
6 bob.goTo(3); // bob se place dans le noeud 3
7 bob.trace("epsilons.txt"); // bob conserve les valeurs de epsilon dans ce fichier
8 bob.walk(10000,0.01); // bob se ballade 10000 fois ou jusque espilon<0.01
9 bob.showFrequencies(); // bob affiche la frequence de sa presence dans chaque noeud durant sa promenade
```

Vecteurs-matrices

semaine 2

Question7 – La simulation utilisant la classe `Internaute` n'est pas complètement satisfaisante. Pour quelles raisons à votre avis ?
On se propose donc d'utiliser la modélisation matricielle du problème qui nous permet d'écrire que (P étant la matrice de transition)

$$\pi_{t+1} = \pi_t \cdot P$$

Question8 – Écrire dans `SimpleWeb` une méthode implémentant ce calcul. Puis écrire une nouvelle méthode convergente qui, à chaque pas de temps, calcule π_t à partir d'un π_0, d'un nombre d'itérations limite, et toujours d'un seuil ϵ. Afficher les courbes d'ϵ également. Commenter les résultats

Puissances de matrices

semaine 2

Dans une troisième étape, il s'agit maintenant de calculer les puissances de la matrice P.
Question9 – Écrire une méthode dans `SimpleWeb` permettant de vérifier la convergence de la suite des puissances de P en définissant un nouveau type de seuil ϵ pour des matrices. Afficher les courbes d'ϵ également. Commenter les résultats

Génération de Webs

semaine 3

Question10 – Proposer un générateur de `SimpleWeb` ergodique, comparer les temps de calcul des différents algorithmes d'estimation de la distribution stationnaire.