# Pagerank

Nicolas Chenciner & David Bühler

3 mars 2013

## 1 Utilisation

**Compilation**

Un makefile est fourni :

```
$ make
```

Java 7 est nécessaire.

**Exécution :**

```
$ ./pagerank file [zap count epsilon z]
```

**Options :**

- file : nom du fichier contenant la description du graphe.
- zap : facteur zap (flottant) ; 0 par défaut.
- count : nombre maximum d'itérations de l'algorithme ; 0 par défaut.
- epsilon : distance maximale entre deux vecteurs successifs calculés par l'algorithme ; 0 par défaut.
- z : facteur influant sur le vecteur initial utilisé par l'algorithme :
  - si z est négatif, le vecteur initial est le vecteur « uniforme » dont chaque élément vaut 1/`length` (comportement par défaut).
  - si z est positif, le vecteur initial est le vecteur $v$ tel que $v[z] = 1$ et $\forall i \neq z, v[i] = 0$.

**Format de fichier**

Les graphes sont décrits par des fichiers placés dans le répertoire `examples/` tels que :

- les lignes vides ou commencant par le caractère # sont ignorées ;
- les autres lignes doivent être de la forme « i j », symbolisant un arc du sommet $i$ vers le sommet $j$.

# 2 Explication de code

## 2.1 Découpage des classes

**Vect<T>** Classe paramétrée représentant un vecteur de type T ; plusieurs constructeurs permettent d'initialiser facilement un vecteur, à partir d'une liste, d'un tableau ou d'une valeur par défaut.

**FVect** Vecteur de type float ; dispose des opérations d'addition et de multiplication par une constante, ainsi que du calcul de la norme entre deux vecteurs.

**Matrix<T>** Matrice de type T, dans le format décrit par l'énoncé, avec les méthodes permettant d'accéder aux valeurs ou de les modifier.

**FMatrix** Matrice de type float ; dispose des opérations de multiplication et de multiplication « transposée » par un `FVect`.

**Graph** Représentation d'un graphe, sous forme de hashmap. La méthode `stoch` calcule la matrice stochastique associée de manière optimale. Contient également la méthode statique `zapPagerank`.

**GraphParser** Lecture d'un fichier et construction du graphe associé.

**Test** Exécution d'un test : construction du graphe à partir d'un fichier, calcul de la matrice stochastique, exécution de l'algorithme *pagerank* avec les paramètres donnés, affichage des résultats.

**MainTest** Main, parsing des arguments, exécution du test.

## 2.2 Algorithme

# 3 Complexité

## 3.1 Multiplication d'une matrice par un vecteur

La méthode mult_naive effectue un appel à get pour chaque élément du tableau selon un parcours des lignes et des colonnes ; get étant de complexité $O(n)$ (potentiellement une opération par colonne), elle opère donc en $O(n^3)$, où $n$ est la longueur de la matrice carrée.

La méthode mult est nettement plus efficace, puisqu'elle ne considère que les éléments non nuls de la matrice en cherchant directement dans les trois tableaux de la structure : elle effectue

– deux accès au tableau L pour chaque ligne ;

– pour chaque élément non nul, un accès à I, un accès à C, une multiplication et une addition.

Soit un nombre fixe d'opérations pour chaque ligne et chaque élément non nul.

Ce calcul est donc en $O(n + m)$ où $n$ est le nombre de lignes de la matrice et $m$ le nombre d'éléments non nul.

### 3.2 Multiplication de la transposée d'une matrice par un vecteur

Bien que l'algorithme soit différent, la complexité est identique à celle de la multiplication de la matrice elle-mêe par un vecteur :

- deux accès à un tableau par ligne ;
- pour chaque élément non nul, trois accès à un tableau, deux opérations arithmétiques, une écriture dans un tableau.

Soit un temps constant pour chaque élément non nul, et un temps constant par ligne : complexité en $O(n + m)$.

## 4 Résultats et interprétation

### 4.1 Exemples

Le répertoire `examples/` contient quelques fichiers descriptifs de graphes simples et pertinents permettant de tester l'algorithme.

Notons au préalable que les résultats ne peuvent être garantis avec une précision supérieur à $10^{-8}$ ; seront donc considérés identiques des résultats ne différant que par leur huitième décimale après la virgule.

**convergence** graphe fortement connexe, pgcd des longueurs des circuits = 1 : convergence assurée quel que soit le vecteur initial (et c'est bien le résultat observé, ouf).

On note que la convergence est néanmoins légèrement plus rapide à partir du vecteur initial uniforme [0.25 0.25 0.25 0.25].

**divergent** graphe fortement connexe, mais dont le pgdc des longueurs des circuits vaut 3 : la convergence n'est plus assurée.

Ainsi, si pagerank converge rapidement à partir du vecteur initial équiprobable, ce n'est plus le cas si l'on part d'un vecteur de type [1 0 0 ...].

En revanche, avec un facteur zap non nul fixé, pagerank converge vers un même résultat quel que soit le vecteur initial, mais encore une fois beaucoup plus rapidement à partir du vecteur « uniforme ».

Si ce résultat diffère de celui obtenu avec un facteur zap nul et un vecteur initial « uniforme », le classement des nœuds reste inchangé.

Plus le facteur zap est élevé, plus la convergence est rapide, mais moins les résultats sont pertinents : les nœuds du graphes tendent alors à obtenir des valeurs proches. Il s'agit donc de trouver un équilibre entre rapidité de la convergence et pertinence du résultat.

**circular** graphe circulaire, donc similaire au précédent : on observe alors les mêmes résultats, de manière encore plus flagrante.

**petersen** graphe symétrique cubique. Convergence assurée.

**clique** convergence assurée (clique implique fortement connexe et pgcd des longueurs des circuits = 1).

Sur une clique, quel que soient le facteur zap et le vecteur initial, le résultat final sera nécessairement équiprobable (1/`size` pour chaque nœuds). Le résultat est immédiat à partir d'un vecteur initial équiprobable, et peut être beaucoup plus long à propager sinon.

### 4.2 Vecteur initial

Le vecteur initial, dont la somme des éléments vaut 1, représente la probabilité de se trouver sur une page au début de l'algorithme. Le vecteur « équiprobable » (1/`size` à chaque nœuds) offre ainsi une même importance à chaque page, alors qu'un vecteur [1 0 0 ...] parcourt le graphe à partir d'un seul nœud.

Sur la plupart des graphes, pagerank converge nettement plus rapidement à partir du vecteur « équiprobable », ce qui semble parfaitement logique : à partir d'un vecteur [1 0 0 ...], il faut le temps de propager les valeurs à tous les nœuds du graphe, les premières itérations de l'algorithme ne concerneront que les nœuds autour du premier ; et plus le graphe est grand, plus ce temps sera long. À partir du vecteur équiprobable, les premières itérations transmettent de l'information pour tous nœuds du graphe.

Qui plus est, le vecteur équiprobable est généralement plus proche du résultat (peu de nœuds finissent avec une valeur nulle).

Évidemment, il est également possible de construire un graphe où pagerank soit plus rapide à partir d'un autre vecteur (`examples/node1`), mais ce type de graphe n'a que peu d'intérêt et n'a aucune chance d'être rencontré lors d'une analyse du web.

Sur les graphes dont le résultat dépend du vecteur initial, le résultat est plus cohérent à partir du vecteur équiprobable. Sur les graphes circulaires, par exemple, il attribue à chaque nœud une valeur égale, alors qu'un autre vecteur ne convergerait pas.

En particulier, sur un graphe non connexe, il permet d'attribuer une valeur non nulle à chaque composante, alors qu'un vecteur [1 0 0 ...], ne s'occuperait que de l'une d'elles.

C'est pour toutes ses raisons que notre algorithme utilise le vecteur équiprobable par défaut.

### 4.3 Facteur zap

## 5 Performance

Sur le fichier `web-Stanford.txt`, disponible à l'adresse `http://snap.stanford.edu/data/web-Stanford.html`, décrivant un graphe de près de 300 000 nœuds et de plus de 2 millions d'arcs, notre programme construit le graphe et calcule la matrice stochastique associée en quelques secondes et opère une centaine d'itérations de l'algorithme de pagerank en 10 à 20 secondes sur nos ordinateurs portables, ce qui nous semble parfaitement honorable.

Sur ce graphe, pagerank ne parvient pas à trouver un point fixe (en tout cas, pas en moins de 1000 itérations).

On note néanmoins que le résultat converge sensiblement plus rapidement avec un vecteur initial « uniforme » et un facteur zap positif, et qu'on obtient assez rapidement des valeurs permettant de distinguer certains nœuds (par exemple, avec un facteur zap nul, certains nœuds sont très vites écartées avec une valeur nulle).