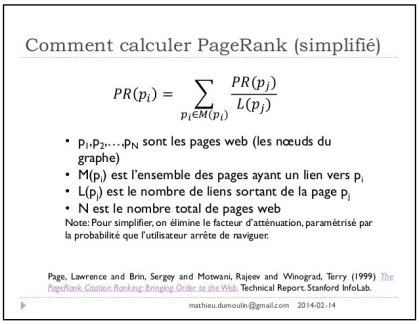
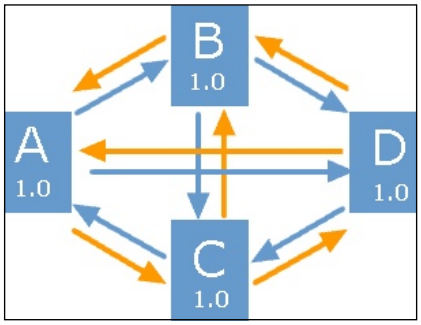
Algorithme de PageRank

Projet Algorithme et Programmation

Scotto/Pichel - 17 janvier 2017

PLAN





I - Introduction

L'objectif de ce projet est de créer les algorithmes nécessaires à la mesure de la popularité d'un réseau dirigé donné. Ces algorithmes sont très utilisés pour Internet et notamment en particulier pour les moteurs de recherche. Nous nous intéresserons plus particulièrement à l'algorithme de PageRank, algorithme de Google, qui consiste à utiliser les poids des pages Internet. Un poids représente la respectabilité d'une page aux yeux d'autres pages. Les liaisons entre les différentes pages sont assurées par ce que l'on appelle les hyperliens.

Plus une page référence d'autres pages, moins elle est populaire, et plus une page est référencée par d'autres pages, plus elle est populaire. Ces calculs sont respectivement pondérés par le poids de chaque page. Ainsi on peut faire l'analogie avec des votes. Plus une page est populaire, plus son vote compte. Et inversement. Ce schéma global implique que l'on peut calculer de manière itérative les poids de chaque page (voir image).

Ainsi, les moteurs de recherche classent les pages par ordre décroissant de leur poids, d'où l'expression de PageRank. On utilise des réseaux (ie graphes) dirigés en entrée. Les arrêtes représentent les hyperliens et les sommets les pages internet. De plus, le sujet nous indiquait quels outils utilisés et les calculs à employer. On représente tous les

poids des pages web à l'itération k+1 par une multiplication matricielle, base de notre travail.

Cette multiplication prends en compte un vecteur ligne des poids des pages à une itération précise, et une matrice adjacente qui comporte, pour chaque ligne i, la valeur 1/Pi s'il existe un lien sortant de la page Pi vers la page Pj, et 0 sinon.

Ensuite on modifie cette matrice H en remplissant les lignes vides par la valeur de 1/N avec N le nombre de ligne de la matrice. Enfin on transforme cette matrice en une matrice finale G en utilisant le coefficient alpha. Alpha est le facteur d'atténuation, et représente la probabilité pour que l'on aille sur une page non référencée (ie pas d'hyperliens vers elle) par la page où l'on est actuellement. La valeur alpha=0,85 est utilisée pour notre projet afin de garantir un bon compromis entre rapidité de calcul et précision des valeurs.

Grâce à ces indices, notre objectif est de fournir un exécutable capable de calculer, pour un réseau donné, le PageRank et le poids de chaque noeud du réseau.

II - Démarche et choix réalisés

Dans un premier temps, nous nous sommes imprégnés du sujet en faisant des recherches chacun de son côté sur le processus du PageRank. Cela nous a permis de confronter nos versions respectives de l'algorithme et ainsi de mettre en commun nos idées. Une fois que nous avions bien cerné le sujet, nous avons réfléchis aux structures que nous allions utilisées. En effet les matrices avec lesquelles on travaille sont des matrices creuses : la plupart des coefficients sont nuls. Dans le projet précédent, nous avions travaillé sur un projet de vecteur creux. Ainsi on s'est mis d'accord pour utiliser ces vecteurs creux dans les matrices afin d'éviter de manier des matrices trop lourdes. Malheureusement nous n'avions pas les mêmes définitions des vecteurs creux et quelques soucis sur nos sous programmes du projet précédent.

De ce fait nous nous sommes répartis le travail. Guillaume s'est occupé dans un premier temps de mettre en commun les sous programmes associés aux vecteurs creux, de corriger certains d'entre eux, et enfin de les optimiser, notamment pour le produit scalaire, très utile dans le produit matriciel. Pendant ce temps, Thibault codait les structures auxquelles nous avions réfléchis, et définissait les outils pour travailler avec des matrices creuses.

Ensuite Thibault s'est occupé du produit matriciel pendant que Guillaume s'occupait du sous programme de la ligne de commande et celui de la création du fichier.

Nous nous sommes aidés l'un et l'autre tout au long du projet afin de se débloquer des situations.

Cette démarche nous a permis d'être efficace et donc d'aller plutôt vite.

Nous allons maintenant présenter l 'architecture de l'application.

III - Architecture de l'application

Raffinages:

R0 : Classer N pages selon l'algorithme PageRank

R1 : Calculer le vecteur pi des poids réels de chaque page et classer les N pages en fonction, en utilisant une architecture peu couteuse en mémoire

R2.1 : lire le fichier contenant les couples (page source, page cible)

R2.2 : calculer la matrice H

R2.3 : effectuer I itérations de la relation du sujet pour calculer un valeur approchée de pi

R2.4 : classer les pages du plus grand poids au plus faible

R3.1 : On lit le fichier et on enregistre les couples dans un tableau de dimension N par 2

R3.2.1 : On le tri et on supprimer les doublons

R3.2.2 : On entre les données dans la matrice creuse H

R3.3.1 : Initialiser Pi0

R3.3.2 : On calcule la colonne i de G

R3.3.3 : On calcule la coordonnée i de Pi(k+1) qui vaut colonne_i(G) scalaire Pi(k)

R3.4 : On tri les valeurs de Pi

Nous utilisons vecteur-creux.c et vecteur-creux.h qui servent respectivement à donner les sous fonctions utiles associées aux vecteur creux et leur utilité, leurs pré conditions et leurs post conditions.

Ensuite nous utilisons le programme matrice.c qui décrit les sous fonctions associées aux matrices creuses ainsi qu'aux tris.

Le programme principal est le pagerank.c qui contient l'entrée en ligne de commande dans le main.

Enfin, nous avons un programme de test appelé seatest.c qui contient les tests sur les sous fonctions de bases.

IV - Présentation des principaux algorithmes

Structures:

Structures des vecteurs creux

```
typedef struct {
  Indice indice;
  Valeur valeur;
  Element *suivant;
} Element;
```

```
typedef struct{
  int dimension;
  Element*premier;
} Vecteur;
```

Structure des matrices creuses

```
typedef struct {
  int dimension;
  Vecteur *lignes;
} Matricecreuse;
```

On commence par créer le sous-programme qui initialise les matrices creuses que l'on rempli avec des vecteurs creux. Ainsi on utilise la sous fonction initialiser vecteur creux et on alloue la mémoire associée donc dynamiquement. Une matrice est donc un tableau de vecteurs creux.

Ensuite une des fonctions importantes est celle qui trie le réseau que l'utilisateur rentre dans la ligne de commande. Le réseau est considéré comme un tableau, que l'on parcours par ligne et par colonne pour trier les valeurs. Le tri est par insertion car il a une meilleure complexité spatiale (donc moins de mémoire utilisée) par rapport aux autres tris (tri rapide/tri par sélection).

Le programme de récupération de la matrice via un fichier est aussi l'un des principal algorithme, permettant de lire la matrice sur un fichier texte en se servant des commandes de lecture sur fichier.

Le programme de ligne de commande, quand à lui, se sert des paramètres d'entrée dans la ligne de commande: argc & argv. Il demande à l'utilisateur quel est le réseau qu'il veut trié, et éventuellement les valeurs de alpha et de nombre d'itérations max, qui sont par défaut respectivement à 0,85 et 150.

Enfin la fonction la plus importante est celle qui effectue le produit matriciel, à l'aide du produit scalaire préalablement défini dans le projet vecteur creux. On multiplie le vecteur ligne des poids à l'itération k avec la matrice G pour obtenir le vecteur ligne des poids à l'itération k+1.

V - Difficultés rencontrées

On a eu quelques difficultés sur la fonction ajouter des vecteurs creux. En effet on a remarqué que l'on n'avait pas fait tous les tests. De plus il fallait que toutes nos fonctions renvoient la même chose pour uniformiser notre continuation dans PageRank.

Ensuite nous avons à nouveau remarqué que la fonction ajouter avait une mince erreur qui était passée outre nos deux tests. Nous avons pris quelques temps pour remarquer d'où provenait l'erreur.

Par la suite, nous avons obtenus des premiers résultats qui convergeaient tous vers 0 (les poids). Après un long moment, nous avons compris que l'on s'était trompé de sens dans le produit matriciel. En effet, nous avions considéré le produit de la matrice par le vecteur des poids en vecteur colonne.

Enfin, nous avons essayer de régler ce problème, ce qui a pris la fin du temps que l'on avait avant de finir le projet. Nous l'avons résolu mais les résultats sont toujours faux.

Ainsi nous essayons toujours de trouver la ou les erreurs potentielles dans les calculs et nous n'avons pour l'instant pas de piste valable.

VI - Conclusion

Ce projet demandait de prendre en facteur des paramètres que nous n'avions pas particulièrement pris en compte lors des précédents projets. Nous entendons par là la question de l'efficacité de nos programmes, c'est-à-dire de leur temps d'exécution. En effet, l'application PageRank est censée pouvoir travailler avec des matrices très grandes, et donc des calculs très longs, et un algorithme de ce type doit tourner dans une limite de

temps raisonnable. Puisque nous n'avons pas une application qui fonctionne pour l'instant, nous n'avons pas encore pu tester les temps d'exécutions. Par contre, nous avons pensé à quelques solutions futures à apporter pour tenter de rendre l'application plus rapide. En effet, nous pensons pouvoir gagner du temps lors du produit matriciel en utilisant un produit scalaire récursif. Nous avons aussi trouvé quelques potentielles solutions au problème de temps quand nous avons dû changer le produit matriciel après l'erreur de sens. En effet le fichier n'est parcouru qu'une fois, on stocke les données dans un tableau dynamique adapté qui est trié et on calcule les poids au fur et à mesure. La matrice est alors remplie en une seule fois et la mémoire libérée.