Pagerank Computation with Special Attention To Dangling Nodes

RAKOTOMANGA Andrianina, DOUBI Dylan, ALI AHMEDI Mycipssa

May 12, 2021

1 Pagerank

Premièrement nous avons implémenté la version classique du google Pagerank avec la méthode des puissance et le calcul du vecteur de notes pour chaque page défini comme suit :

$$\text{xG} = \alpha * x * P + [(1-\alpha) * (\tfrac{1}{N}) + \alpha * (\tfrac{1}{N}) * (x * f^t)] * e$$

Les résultats obtenus sont concluants et vérifiables avec tous les graphes.
Le vecteur d'intitialisation de base v est constitué de $\frac{1}{N}$ pour chaque valeur, N étant le nombre de pages du graphe.

Pour web1.txt : 0.161522 0.087397 0.126937 0.183909 0.093037 0.136452 0.134734 0.076012
Pour web4.txt : 0.194326 0.081095 0.158093 0.202864 0.064577 0.194757 0.048059 0.056229.

Nous avons vérifié avec d'autres élèves et leurs résultats obtenus pour le même algorithme sont similaires.
Pour utiliser la version du pagerank normal, veuillez entrer make dans un terminal, ce qui lancera le programme, puis selectionnez le fichier à traiter, la convergence (0.x1) puis la valeur d'alpha (0.85) enfin choisissez l'option 1, ce qui lancera le pagerank de google normal.

2 Lumping method of Dangling nodes

D'après la preuve [IS07] de Ipsen et Seele, regrouper les noeuds de sortie nulle dans un seul état permet de réduire la taille de la matrice pour le calcul puis de calculer leur pertinance en un seul calcul à la fin de la convergence, ce qui augmenterait la vitesse d'exécution et de convergence de l'algorithme.

Un noeud nul est un noeud sans sortie, c'est à dire qui n'a aucun lien vers un autre noeud. Par contre il est possible que d'autres noeuds non nuls pointent vers celui-ci.

Pour effectuer ce calcul il faut tout d'abord initialiser certaines conditions.

2.1 Initialisation

Avant tout, il est primordial de représenter la matrice du web de cette manière.

$$\begin{bmatrix} H11 & H12 \\ 0 & 0 \end{bmatrix}$$

H11 représente la matrice de liens entre noeuds non nuls.
H12 représente celle de liens d'un noeud non nul vers un noeud nul. H11 est de taille $k * k$ et H12 et de taille $k * (n - k)$
Cette représentation sera donc de taille $n * n$, ce qui est similaire à la représentation du pagerank classique (n = nombre de noeud du graphe).
Nous aurons besoin d'un vecteur de noeuds nuls w de taille $1 * n$ et de v, vecteur de personnalisation de même taille.

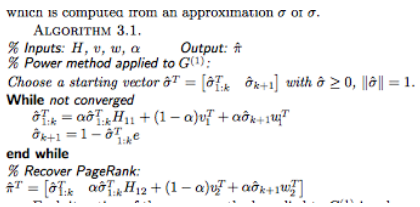

Figure 1: Algorithme du Lumping.

Nous allons également utiliser une approximation de σ qui est une distribution stationnaire de G (G etant la matrice de google). Nous allons calculer l'approximation pour trouver la valeur des notes des noeuds non nuls. Cette approximation nous aidera également pour le calcul final des notes des noeuds nuls.

2.2 Algorithme

L'algorithme se code à l'instar du pagerank avec quelques modifications.

Pour vérifier les résultats obtenus, il suffit de les comparer avec ceux du pagerank normal dans les mêmes configurations.

Pour web1.txt : 0.161522 0.087397 0.126937 0.183909 0.093037 0.136452 0.134734 0.076012

Pour web4.txt : 0.194326 0.081095 0.158093 0.202864 0.064577 0.194757 0.048059 0.056229.

Nous obtenons effectivement les mêmes résultats que pour le pagerank normal, ce qui confirme l'efficacité de cette méthode.

Pour lancer l'exécution de cet algorithme, il suffit de choisir l'option 2 lors du choix qui est proposé au moment du lancement du programme.

3 Analyse Technique

Le code est divisé en trois parties : une de lecture (read.c), une de calcul (product.c), une d'exécution (pagerank.c)

3.1 Read.c

$int\ check-url(const\ char\ *str_check)$: fonction qui permet de vérifier la validité du format de fichier choisi pour le graphe du web, renvoie 0 si le fichier est valide 1 sinon.

$void\ insert(Sommet\ *s1, Sommet\ *s2)$: Fonction qui insère un sommet s2 au niveau du sommet s1 dans sa liste de prédecesseur (liste chainée)

$void\ read-txt-(Sommet\ *tabSommet, int\ sommet, FILE\ *F, int\ *k)$: Fonction de lecture pour la méthode d'aggrégation des noeuds nuls. Cette lecture consiste à séparer les noeuds nuls et non nuls en deux parties H11 et H12 dans le même tableau de lecture, La lecture est de complexité similaire à la méthode qui sépare ses deux parties en deux tableaux (essai).

Pour faire la séparation, on lit le fichier du graphe du web de manière itérative, puis lorsqu'on rencontre un noeud nul, on le permute de place avec un noeud non nul plus loin dans la liste encore non traitée. Pour retrouver ce noeud, on effectue une boucle (while) supplémentaire pour retrouver le bon index où insérer le noeud traité.

C'est une méthode peu efficace pour séparer H11 et H12, nous ne l'utiliserons pas mais la laissons à titre informatif. Elle pourrait s'avérer utile dans une autre configuration.

$int\ read-txt(double\ *f_t, Sommet\ *tabSommet, int\ sommet, FILE\ *F, int\ *liste)$: Méthode efficace de lecture du fichier du graphe du web. Elle stocke le graphe dans un tableau de sommets à liste chainée. Pour connaître les noeuds nuls, on stocke les index dans une liste. Cette méthode de

stockage a le même temps d'exécution qu'une lecture dans deux listes séparées (noeuds nuls et non nuls).

De plus, le temps de calcul pour l'aggrégation des noeuds nuls est la même pour les deux méthodes, ce qui nous conforte dans l'idée de garder cette méthode de stockage moins couteuse (un seul tableau de liste chainée au lieu de: 1 (tableau initial) + 2 (tableaux séparés) pour la méthode de séparation).

int compare($double * x1, double * x2, int\ taille,\ doubleprecision$) :
Comparaison de deux résultats (précédent et actuel) des vecteurs de notes pour la convergence.

3.2 Product.c

void **multiplication**($Sommet * Matrice,\ int\ nb\_sommet,\ double * Vecteur, double * Res$) :
- Multiplication d'une matrice de sommets du web (creuse) et d'un vecteur de notes.

double **mult_scalaire**($double * Vect\_l,\ double * Vect\_c,\ inttaille$) :
- Multiplication d'un vecteur ligne et d'un vecteur colonne, obtention d'un scalaire (nombre).

void **mult_alpha_vecteur_ligne**(*double alpha*, $double * Vecteur,\ int\ taille$) : - Multiplication d'un scalaire et d'un vecteur ligne.

void **somme_vecteur_ligne**($double * vect1,\ double * vect2, int\ taille,\ double * res$) :
- Somme de deux vecteurs ligne.

void **multiplication_xG**($Sommet * Matrice,\ int\ nb\_sommet,\ double * Vecteur,\ double * Res,\ double\ alpha,\ double * f_t,\ double * e$) :
- Multiplication de Pagerank.

void **mult_dangling**($Sommet * Matrice,\ int\ nb\_sommet,\ Vector * Vecteur,\ double * Res,\ int * liste,\ int\ on$):
- Multiplication Matrice Creuse/Vecteur de notes de la méthode d'aggrégation.

double **multiplication_dangling_version**($Vector * w1,\ Vector * w2,\ Vector * v1,\ Vector * v2,\ Vector * theta,\ double\ alpha,\ int\ n,\ int\ k,\ Sommet * H11,\ Sommet * H12,\ double * e,\ Vector * res\_w1,\ Vector * res\_v1,\ double * theta\_k\_1,\ double\ precision,\ int * liste,\ int * q$) :
- Multiplication pagerank de la méthode d'aggrégation.

4 Comparaison

Malgré le changement d'algorithme et de méthode de calcul, il y a peu de différence (non notable) entre le temps d'execution, le nombre d'itérations (convergence) et la complexité spatiale entre les deux méthodes. Notamment du fait que nous devions également implémenter la lecture de la matrice creuse du graphe du web, ce qui demande le plus de temps d'execution.

Ce non-changement s'explique notamment par le fait qu'il nous est obligé de garder la structure de base du graphe du web, en dépit de l'augmentation d'utilisation de la mémoire de manière exponentielle. En effet si nous essayons de réduire la matrice en deux matrices H11 et H12 les indices entre le vecteur de notes et les matrices seront différents, pour aligner les indices correctement, il faudrait effectuer une recherche supplémentaire linéaire en temps ce qui amenerait à une compléxité quadratique en temps de calcul.

5 Annexe

References

[IS07] Ilse Ipsen and Teresa Selee. Pagerank computation, with special attention to dangling nodes. *SIAM J. Matrix Analysis Applications*, 29:1281–1296, 01 2007.

Normal/Lumping	Tps execution (calcul)	Nombre d'itération
web1.txt	0/0	75/31
web4.txt	0/0	31/31
wb-cs-stanford.txt	0/0	77/79
Stanford.txt	16/17	90/91
StanfordBerkeleyV2	9/10	79/80
in2004v2	37/31	81/83
wikipedia	162/171	53/53
wb-edu	108/113	71/74

Table 1: Comparaison Normal/Lumping PageRank.