RAPPORT DE PROJET

Programmation Impérative - Algorithme de PageRank

Pointeau Quentin & Frenois Etan

ENSEEIHT - Janvier 2024
Groupe AB09

Table des matières

RÉSUMÉ

En 1998, Sergueï Brin et Larry Page ont proposé de mesurer la popularité des pages Internet en les triant de la plus populaire à la moins populaire selon un ordre appelé *PageRank*. *PageRank* est exploité dans le moteur de recherche Google pour pondérer le résultat d'une requête dans le *World Wide Web*. *PageRank* est calculé à partir d'une métrique de popularité basée sur une idée très simple : attribuer à chaque page une valeur, un poids 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. Il reste alors à classer l'ensemble des pages web par ordre décroissant de leur poids. Le rang d'une page web dans cet ordre est alors appelé le *PageRank*. Ainsi, une page web de rang faible (*PageRank* petit) est une page importante pour le graphe des pages web. Plus le rang d'une page est faible, plus la probabilité est importante qu'une marche aléatoire lancée sur n'importe quelle autre page, passe par cette page web. L'ensemble des pages Internet et des hyperliens peut être modélisé par un graphe orienté. Un graphe se compose d'un ensemble de nœuds (sommets) et d'arcs (liens ou arêtes) qui relient deux nœuds selon une direction, le premier est l'origine de l'arc, le second la cible.

Dans notre cas, les pages de l'*Internet* seront modélisées par un graphe, les pages web seront les nœuds du graphe et les hyperliens seront les liens orientés.

Ainsi, ce document présente le processus selon lequel nous avons implémenté l'algorithme de *PageRank*.

INTRODUCTION

L'objectif de ce projet est de fournir un programme capable de calculer, pour un graphe donné, le *PageRank* et le poids de chaque nœud du graphe. Les arguments de la ligne de commande permettront de choisir l'implantation à utiliser ainsi que les valeurs des paramètres de l'algorithme à utiliser puisque le programme ne devra faire aucune interaction avec l'utilisateur. Son comportement sera exclusivement défini au moyen des paramètres de la ligne de commande.. Cet algorithme sera implémenté selon deux approches : l'approche *matrice pleine* et l'approche *matrice creuse.*

On peut modéliser le calcul du *Pagerank* par un calcul matriciel. En effet, on peut représenter le calcul de tous les poids des pages web au rang $k + 1$ par une multiplication matricielle :

$$\pi_{k+1}^T = \pi_k^T \cdot H \qquad \text{si } k > 0$$
$$\pi_0^T = (1/N, 1/N, ..., 1/N)$$

où :

- π_k^T est un vecteur ligne dont la composante i et le poids de la page i au rang k.
- H est une matrice d'adjacence pondérée qui comporte, pour chaque ligne i, la valeur 1 / I P_i I s'il existe au moins un lien sortant de la page P_i vers la page P_j , et 0 sinon.

Pour bien répartir les poids et garantir la convergence de l'algorithme, Sergueï Brin et Larry Page ont introduit plusieurs modifications à la matrice *H*. La première transforme la matrice *H* en une matrice *S,* et la seconde transforme la matrice *S* en la matrice *G* appelée matrice de *Google*.

$$G = \alpha \cdot S + \frac{(1-\alpha)}{N}\mathbf{e}\mathbf{e}^T \quad \alpha \in [0, 1]$$

G est la somme de la matrice $\alpha \,.\, S$ et d'une matrice dont tous les termes valent $(1 - \alpha)/N$. Dans l'équation ci-dessus, *e* représente un vecteur dont tous les éléments

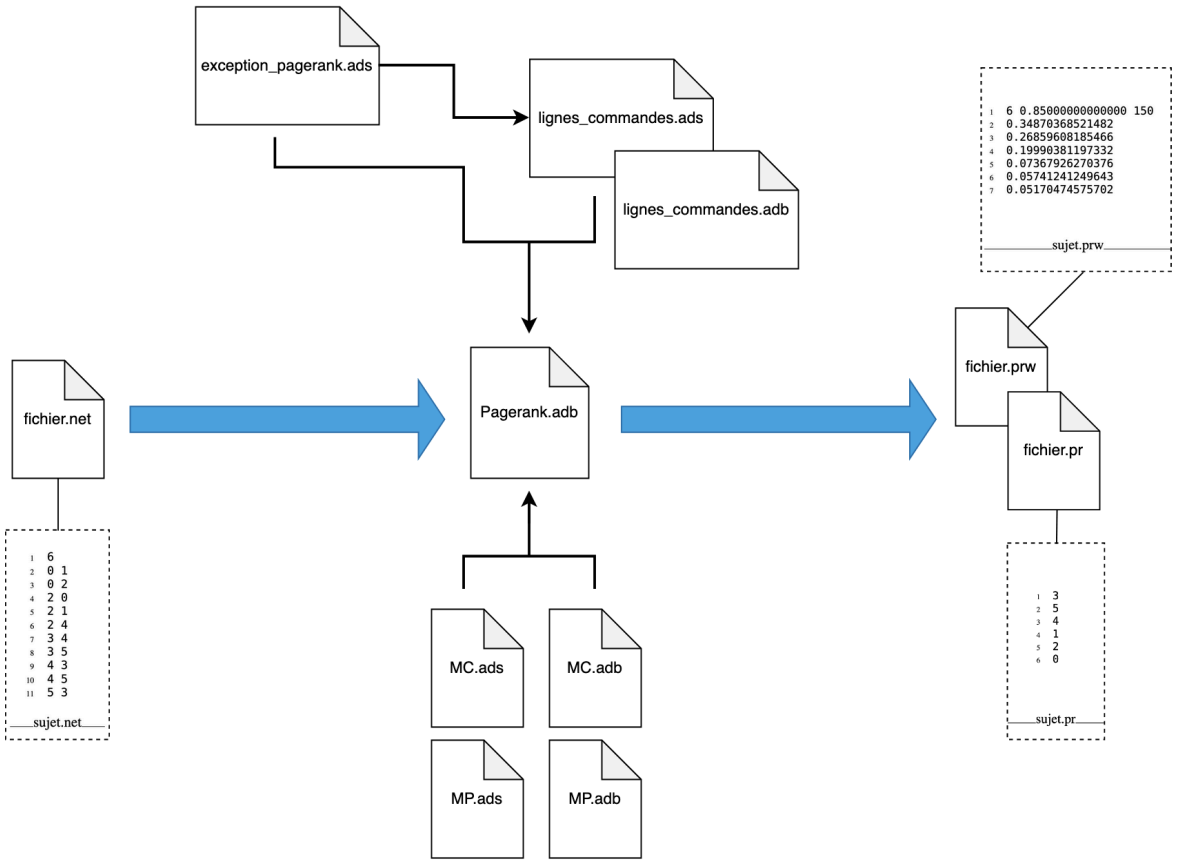
valent 1. La pondération α est appelée le *dumping factor*. Plus sa valeur est proche de 1, plus la convergence du calcul est lente. Plus la valeur est faible, plus le calcul est rapide mais moins les poids prennent en compte la topologie du graphe.

Ainsi, l'approche *matrice pleine* consiste à calculer la matrice de *Google* et l'approche *matrice creuses* consiste à considérer le fait que la matrice *H* est très creuse.

Vous trouverez dans la suite de ce document, l'architecture de notre application, les principaux choix réalisés concernant la mise au point du programme et des modules, des informations concernant la durée d'exécution de notre programme et les difficultés que nous avons rencontrées quant à la conception du programme.

ARCHITECTURE DE L'APPLICATION

Comme on peut le voir ci-dessous, l'architecture de notre application est assez simple. Elle se compose seulement de trois modules *MP* (*matrice pleine*), *MC* (*matrice creuse*), *lignes_commandes*, d'un fichier de définition d'exceptions et d'un fichier principal *Pagerank.adb* :



Le fichier *.net* décrit le graphe sur lequel nous voulons appliquer l'algorithme de *PageRank*. Une fois le calcul de *PageRank* réalisé, notre application produit deux fichiers : un fichier *.pr* décrit le classement des nœuds suivant l'algorithme PageRank et le fichier *.prw* qui décrit le poids des nœuds correspondants. Comme décrit sur le schéma ci-dessus, le fichier ***PageRank.adb*** utilise les modules *MP* et *MC* pour calculer le *PageRank* selon l'approche *matrices pleines* ou *creuses*. Le fichier *PageRank* utilise également le module *lignes_commandes* qui traite la ligne de commandes et un fichier *exceptions_pagerank.ads* qui regroupe toutes les exceptions de notre application.

PRINCIPAUX CHOIX RÉALISÉS

Pour l'approche *matrice pleine*, nous avons défini trois structures de données qui sont définies comme suit :

- Un type ***T_Matrice*** qui est un tableau à double entrées de flottants qui nous permet de représenter des matrices qui nous est utile notamment pour les matrices *H, G, S* et ee^T :

```
type T_Matrice is array (0..TAILLE-1, 0..TAILLE-1) of Float;
```

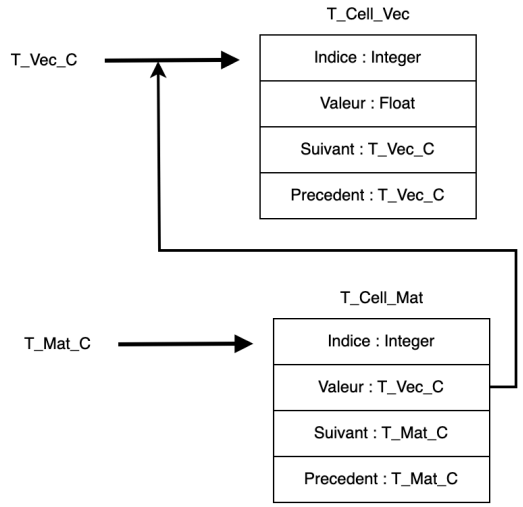
- Un type ***T_Vecteur_Float_MP*** qui est un tableau de flottants représentant un vecteur qui nous sert pour le classement des poids des pages :

```
type T_Vecteur_Float_MP is array (0..TAILLE-1) of Float;
```

- Un type ***T_Vecteur_Integer_MP*** qui est un tableau d'entiers représentant un vecteur qui nous sert pour le classement des pages :

```
type T_Vecteur_Integer_MP is array (0..TAILLE-1) of Integer;
```

Pour l'approche *matrice creuse*, nous avons utilisé des listes doublement chaînées définies comme suit :




```
type T_Cell_Vec;
type T_Cell_Mat;

type T_Vec_C is access T_Cell_Vec;
type T_Mat_C is access T_Cell_Mat;

type T_Cell_Vec is record
    Indice : Integer;
    Valeur : Float;
    Suivant : T_Vec_C;
    Precedent : T_Vec_C;
end record;

type T_Cell_Mat is record
    Indice : Integer;
    Valeur : T_Vec_C;
    Suivant : T_Mat_C;
    Precedent : T_Mat_C;
end record;
```

Nous avons également défini deux types ***T_Vecteur_Float_MC*** et ***T_Vecteur_Integer_MC*** qui sont respectivement un tableau de flottants et un tableau d'entiers qui nous servent pour les classement des poids des pages et pour le classement des pages comme pour l'approche avec *matrices pleines*.

Le choix le plus déterminant que nous avons fait est de structurer notre matrice creuse en faisant en sorte que les colonnes de la matrice soient chacune stockées dans une liste chaînée. Cela permet d'optimiser le calcul matriciel car cela réduit le nombre d'accès aux données stockées au vu de la structure du produit matriciel.

Un autre point d'optimisation important a été de séparer le calcul matriciel en deux parties, une partie commune et une partie variable. La partie commune du calcul se fait donc en amont, ce qui permet de réduire drastiquement le nombre d'opérations de calcul et d'accès aux données. La calcul est découpé comme suit :

$$\pi \cdot G = \pi \cdot \alpha \cdot S + \pi \cdot \frac{1-\alpha}{N} e e^T$$

Le membre de gauche de la somme correspond à la partie variable et le membre de droite à la partie commune.

En ce qui concerne l'algorithme, nous avons suivi le même processus selon les deux approches, *matrice pleine et matrice creuse.* Nous avons décidé d'implémenter dans un premier temps le traitement de la ligne de commandes. Ensuite nous récupérons le

nombre de nœuds du fichier source pour pouvoir remplir la matrice d'adjacence et construire les matrices ee^T, *G et S* dans le cas *matrice pleine* et seulement calculer explicitement la matrice d'adjacence dans le cas *matrice creuse.* Ensuite, nous réalisons l'itération matriciel du calcul du *Pagerank* puis nous classons les pages par ordre de poids décroissant. En dernier lieu, nous écrivons les fichiers résultats (`.pr` *et* `.prw`).

MISE AU POINT DU PROGRAMME ET DES MODULES

Comme expliqué dans la partie "Architecture de l'application", nous avons découpé notre application en trois modules : `MP` (*matrice pleine*), `MC` (*matrice creuse*), `lignes_commandes` et un fichier principal `pagerank.adb`.

Le module `lignes_commandes` regroupe une procédure `Traiter_Ligne_Commandes` qui traite la ligne de commandes et une fonction `Nombre_Noeuds` qui retourne le nombre de nœuds du graphe source.

Le module *MP* regroupe toutes les procédures nécessaires à la manipulation des matrices ainsi que les procédures et fonctions nécessaires au calcul *PageRank.* En particulier, la procédure `Remplir_Matrice_Adjacence` qui remplit la matrice d'adjacence à partir des données du fichier source, la procédure `Construire_ee_S_G` qui construit les matrices ee^T, *G et S,* la fonction `Iteration_Matriciel` qui réalise l'itération matriciel, procédure `Classer` qui classe les poids et les pages par ordres décroissant et enfin les procédures `Ecrire_Fichier_Poids` et `Ecrire_Fichier_PageRank` qui écrivent les fichiers résultats.

Le module `MC` est construit selon le même principe que le module `MP` mais avec une procédure `Calculer_Matrice_Adjacence_Creuse` qui calcule la matrice d'adjacence en considérant l'approche matrice *creuse.*

Pour finir, le fichier principal `pagerank.adb` utilise les modules `MP` et `MC` dans une procédure `Execution_Pagerank` qui calcule le classement *PageRank* selon l'approche *matrice pleine ou matrice creuse*.

Nous avons également un fichier `exceptions_pagerank.ads` qui regroupe toutes les exceptions de notre application.

Il est important de souligner que cette mise au point du programme et des modules

n'était pas l'idée première. En effet, nous voulions au départ créer un module pour les matrices pleines avec un module d'exécution du *PageRank* pour les matrices pleines, un module pour les matrices creuses avec un module d'exécution du *PageRank* pour les matrices creuses, un module qui traite les entrées et sorties de fichiers et un fichiers principales *PageRank.* Ce qui nous fait un total de cinq modules et un fichier principal. Nous nous sommes rendu compte, en discutant avec notre professeur, qu'autant de modules n'étaient pas nécessaires, et ajouté de la complexité à notre application. C'est pour cela que nous avons découpé notre application en trois modules avec un fichier principal facilitant ainsi l'architecture de celle-ci.

DURÉE D'EXÉCUTION DU PROGRAMME

La durée d'exécution du programme *PageRank* dépend de la taille du graphe représenté dans le fichier `.net`, c'est-à-dire du nombre de nœuds du graphe, mais aussi de l'approche considérée. En effet, s'il a été décidé d'implémenter *PageRank* avec des matrices creuses, c'est avant tout pour réduire nettement le temps de calcul des poids et l'espace de stockage utilisé. Ainsi, les durées d'exécutions avec l'approche *matrice pleine* sont bien plus importantes qu'avec l'approche *matrice creuse*. Voici ci-dessous le tableau récapitulatif des temps d'exécution de notre *PageRank* sur les exemples de graphes fournis, à savoir `sujet.net`, `worm.net`, `brainlinks.net` et `linux26.net`.

Les temps d'exécution ont été obtenus avec la commande time et les paramètres de la ligne de commandes par défaut avec un CPU Ryzen 5 5500U à 2.1 Ghz.

Par exemple : `./pagerank -P exemples/sujet/sujet.net`

Graphes	**sujet.net (6 noeuds)**	**worm.net (279 noeuds)**	**brainlinks.net (10000 noeuds)**	**linux26.net (285510 noeuds)**
Temps d'exécution avec l'approche ***matrice pleine***	0,00s user 0,00s system 80% cpu 0,002 total	0,03s user 0,01s system 92% cpu 0,042 total	53,87s user 0,41s system 99% cpu 54,289 total	/
Temps d'exécution avec l'approche ***matrice creuse***	0,00s user 0,00s system 77% cpu 0,002 total	0,01s user 0,00s system 93% cpu 0,006 total	48,33s user 0,04s system 99% cpu 48,376 total	1566,56s user 0,15s system 99% cpu 26:06,97 total

DIFFICULTÉS ET SOLUTIONS

Des difficultés ont été rencontrées lors de l'instanciation de modules génériques. En effet, le but était d'utiliser les types `T_Vecteur_Float` et `T_Vecteur_Integer` définies dans le module *Matrices Pleines* dans le module *Matrices Creuses*. Cependant, cela posait problème dans la reconnaissance des types dans le programme principal *pagerank.adb*.

La solution retenue a été de recopier certaines procédures du module *Matrices Pleines* dans le module *Matrices Creuses*. Ces procédures là sont donc "en doublon". C'est le cas par exemple des procédures `Ecrire_Fichier_Poids` et `Ecrire_Fichier_PageRank`.

GRILLE D'ÉVALUATION DES RAFFINAGES ET DU CODE

- Évaluation du code

		Consigne : Mettre O (oui) ou N (non) dans la colonne Etudiant suivant que la règle a été respectée ou non. Une justification peut être ajoutée dans la colonne "commentaire".	
Commentaire	**Etudiant (O/N)**	**Règle**	**Enseignant (O/N)**
	O	Le programme ne doit pas contenir d'erreurs de compilation.	
	O	Le programme doit compiler sans messages d'avertissement.	
	O	Le code doit être bien indenté.	
	O	Les règles de programmation du cours doivent être respectées : toujours un Sinon pour un Si, pas de sortie au milieu d'une répétition...	
	N	Pas de code redondant.	
	O	On doit utiliser les structures de contrôle adaptées (Si/Selon/TantQue/Répéter/Pour)	
	O	Utiliser des constantes nommées plutôt que des constantes littérales.	
	O	Les raffinages doivent être respectés dans le programme.	

	O	Les actions complexes doivent apparaître sous forme de commentaires placés AVANT les instructions correspondantes, avec la même indentation	
	O	Une ligne blanche doit séparer les principales actions complexes	
	O	Le rôle des variables doit être explicité à leur déclaration (commentaire).	

- Évaluation des raffinages

		Evaluation Etudiant (I/P/A/+)	**Justification / commentaire**	**Evaluation Enseignant (I/P/A/+)**
Forme (D-21)	Respect de la syntaxe Ri : Comment "... une action complexe ..." ? des actions combinées avec des structures de controle Rj : ...	A+		
	Verbe à l'infinitif pour les actions complexes	A+		
	Nom ou équivalent pour expressions complexes	A+		
	Tous les Ri sont écrits contre la marge et espacés	A+		
	Les flots de données sont définis	A+		
	Une seule décision ou répétition par raffinage	A+		
	Pas trop d'actions dans un raffinage (moins de 6)	A+		
	Bonne présentation des structures de contrôle	A+		
Fond (D21-D22)	Le vocabulaire est précis	A+		
	Le raffinage d'une action décrit complètement cette action	A+		
	Le raffinage d'une action ne décrit que cette action	A+		
	Les flots de données sont cohérents	A+		
	Pas de structure de contrôle déguisée	A+		
	Qualité des actions complexes	A+		

ACTIVITÉS

	Spécifier	Programmer	Tester	Relire
lignes_commandes	Quentin	Quentin	Quentin	Etan
MP	Etan	Etan	Etan	Quentin
MC	Etan & Quentin	Quentin	Quentin	Etan
Pagerank.adb	/	Quentin	Quentin	Etan
exceptions_pagerank.ads	/	Quentin	Quentin & Etan	Etan

CONCLUSION

En définitive, nous avons réussi à fournir une application qui fonctionne et compile sans "*warning",* selon les deux approches *matrice pleine et matrice creuse.* En revanche, nous avons manqué de temps pour traiter la partie sur la précision des calculs. En outre, afin d'améliorer notre application nous pouvons implémenter la partie sur la précisions des calculs et essayer d'optimiser les procédures et fonctions afin d'obtenir de meilleurs résultats sur la durée d'exécution du programme en fonction des approches *matrice pleine et matrice creuse*.

ANNEXES

En tant que premier vrai projet de programmation en équipe, celui-ci nous a permis de comprendre et de mettre en place l'organisation nécessaire pour terminer ce projet dans les temps. L'organisation du code ainsi que la répartition du travail sont des aspects que nous n'avions pas encore rencontrés dans la programmation. La persévérance et l'entraide ont été des aspects primordiaux dans ce projet tout comme la bonne répartitions des différentes tâches. Sans eux, il aurait été impossible de rendre une version fonctionnelle du *PageRank* selon les deux approches *matrice pleine* et *matrice creuse*.