Rapport du projet de première année : Page Rank

Unité d'enseignement : Programmation impérative

NOMS, Prénoms : BONGIOVANNI, Arthur
JOMA'A, Ainji

Année : 2023 – 2024

Sommaire

Ce rapport est un complément qui explique le dérouler du projet PageRank. Nous commencerons donc dans ce document par expliquer le but du projet et en quoi consiste l'algorithme réalisé. Ensuite nous établirons le plan, l'architecture du code et les différents modules choisis… Nous détaillerons cela en explicitant les choix réalisés et la manière dont le programme et les modules ont été mis au point. De plus, nous citerons les différentes difficultés rencontrées ainsi que la manière dont celles-ci ont été résolues.

1- Problème traité :

Le but du travail réalisé et de reproduire l'algorithme « PageRank » proposé par Brin&Page en 1998. Celui-ci trie les pages internet de la plus populaire à la moins populaire selon un ordre nommé PageRank. Bien que celui-ci est majoritairement utilisé pour classer les pages internet il peut aussi être exploité pour tout graphe orienté. Afin de pondérer les résultats d'une requête dans le World Wide Web, Google exploite PageRank dans son moteur de recherche. Le principe est simple, à chaque page est associé un poids. Plus le poids d'une page est élevé plus elle est proposée aux utilisateurs. L'algorithme tourne donc autour du fait de générer un vecteur avec les différents poids des pages ainsi qu'un vecteur avec le classement de ces dernières.

On note π_k le vecteur de poids à l'itération k qui contient le poids de chacune des pages et qui va être utilisé pour les classer.

On peut représenter le calcul de tous les poids des pages web au rang k+1 par la multiplication matricielle :

$$\pi_{k+1}^T = \pi_k^T . G \text{ si } k > 0$$

$$\pi_0^T = \left(\frac{1}{N}, \frac{1}{N}, \frac{1}{N}, \dots \frac{1}{N}\right)$$

Où G est la matrice de Google telle que : $G = \alpha . S + \frac{(1-\alpha)}{N} \mathrm{e}\mathrm{e}^{\mathrm{T}}$.

La matrice S est la matrice contenant les liens entre les pages et que nous allons générer en lisant le fichier txt donné en entrée.

(Cf. sujet du projet de PageRank pour plus d'explication sur la matrice de liens S).

On remarque donc que cet algorithme est composé de 3 grandes parties :

a) Obtenir les valeurs de α, k … et d'autres paramètres qui dans notre cas vont être récupérés dans la ligne de commande.
b) Créer la matrice de Google à partir d'un fichier texte contenant des liens afin de l'exploiter dans les calculs.
c) Effectuer les calculs matriciels.

A partir de cela on déduit l'architecture du programme.

2- Architecture du programme : modules et dépendances entre modules.

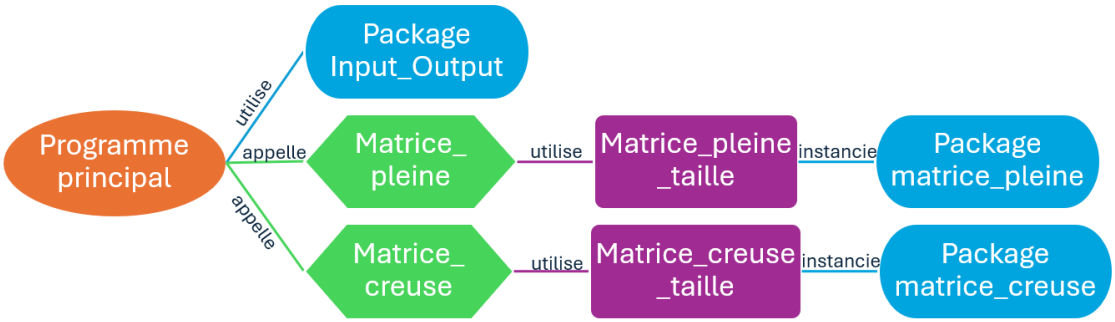

Figure 1 : Schéma montrant l'architecture du programme.

a) Programme Principal :

- Le programme principal utilise le module input_output qui contient l'analyse de la ligne de commande.
- Il appelle les procédures matrice_pleine et matrice_creuse qui utilise les module instanciés matrice_pleine_taille et matrice_creuse_taille.
- Ces modules instancient les modules génériques matrice_pleine et matrice_creuse.

b) Module Input-Output :

Ce module est consacré à l'analyse de la ligne de commande. Le programme lit donc cette dernière puis analyse les paramètres un à un.

Durant l'implémentation, nous avons pris en compte plusieurs cas afin d'être surs de couvrir à la fois l'imprudence d'un utilisateur qui utilise notre programme ainsi que les tests élaborés qui pourraient être faits.

L'algorithme est donc développé comme suit, le programme lit la ligne de commande :

- Si toutes les options sont données correctement il stocke les valeurs dans des variables qui seront transmises au module de calcul matriciel.
- Si certaines options sont correctement données et pas d'autres des valeurs par défaut sont attribuées aux paramètres absents et les valeurs renseignées sont attribuées aux paramètres correspondants.
- Le programme renvoie une erreur accompagnée d'un message guidé si l'utilisateur :
    - Donne une option inexistante.
    - Oublie de mettre une valeur à une option qui en requiert une.
    - Met une valeur qui n'est pas dans l'intervalle prévu pour cette option (p.ex :$0 \leq \alpha \leq 1$) .
    - Oublie de mettre le nom du fichier.
    - Si le fichier n'est pas au bon format (.net ou .txt).

Les choix effectués dans ce module :

- Nous avons fait le choix de renvoyer un message d'erreur si l'utilisateur oublie de mettre une valeur à une option qui en a besoin et non de procéder directement avec la valeur par défaut. Ceci lui permet de se rendre compte que les calculs n'ont pas été effectués avec la valeur qu'il souhaitait.

c) Module Matrice pleine :

Dans ce module nous effectuons tous les calculs nécessaires au calcul des vecteurs poids et classement en considérant que les matrices sont des matrices pleines. Nous gardons donc la représentation en « tableau » de la matrice, qui contient donc toutes les valeurs dont tous les zéros.

Les différentes fonctions et procédures développées dans ce module sont :

- Initialisation des matrices/vecteurs.
- Création de la matrice de liens G.
- Produit vecteur-matrice.
- Affichage pour pouvoir effectuer les tests.
- Lecture du fichier de liens et création de la matrice S.
- Création des vecteurs poids et classement.
- Tri de valeurs pour pouvoir obtenir le classement à partir des poids.
- Créations des fichiers contenant les poids et les classements.

Au début nous avions créé des sous-programmes qui modifient et renvoient les valeurs des coordonnées d'une matrice / d'un vecteur. Mais ensuite nous nous sommes rendu compte que ceci prenait du temps de calculs ainsi que de la mémoire supplémentaires. Comme ces sous-programmes ne sont pas vraiment utiles (la matrice est juste un tableau) est la modification / renvoie des coordonnées de la matrice ne se fait que dans le module lui-même, nous avons donc décider de le retirer.

d) Module Matrice Creuse :

Ce module est quant à lui destiné à faire les calculs en utilisant des matrices (et des vecteurs) creux. Ce module reprend quasiment les mêmes sous-programmes que le module matrice pleine avec bien sûr la différence d'implémentation de ces derniers.

En revanche, nous ne pouvions pas nous autorisés dans ce module à retirer les sous-programmes qui modifient et renvoient les valeurs des coordonnées parce que le parcours d'une matrice creuse est beaucoup plus compliqué qu'une matrice pleine.

Les choix effectués dans ce module :

- Nous avons décidé de donner à chaque élément de la matrice deux pointeurs, (en plus de sa valeur et de ses coordonnées). Le premier (next_below) pointe sur la prochaine coordonnée non nulle de la même colonne et le second (next_side) vers le premier élément non nul de la colonne suivante. Ce choix nous permet alors d'avoir un parcours plus fluide de la matrice et plus adapté au calcul du produit vecteur-matrice qui est l'opération la plus répétée dans notre programme.

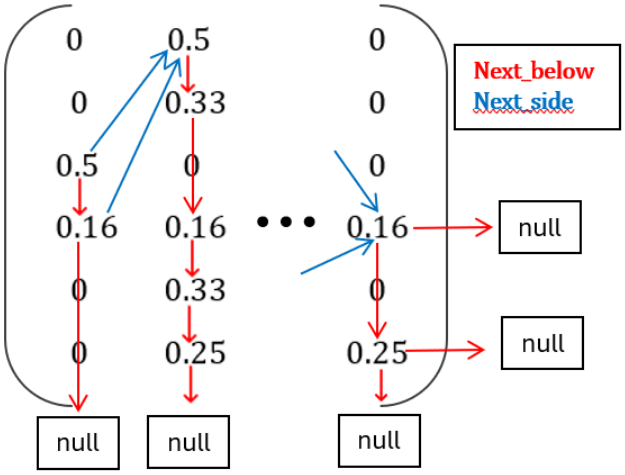

Figure 2 : Schéma illustrant la construction de matrice creuse.

```
Type T_Cell_mat;
Type ptr_Matrice is access T_Cell_mat;
Type T_Cell_mat is record
    valeur : T_reel;
    ind_L : Integer;
    ind_C : Integer;
    next_below : ptr_Matrice;
    next_side : ptr_Matrice;
    -- Invariant :
        --      TAILLE >= ind_L >= 0;
        --      TAILLE >= ind_C >= 0;
        --      next_below = Null or else next_below.all.ind_L > ind_L and next_below.all.ind_C = ind_C ;
        --      next_side = NuLL or else next_side.all.ind_C > ind_C
        --                          and for i in 0..next_side.all.ind_L -1 :
        --                              get_value(Matrice,i,next_side.all.ind_C) = 0
        --      -- cellules sont stockées dans l'ordre croissant des indices en ligne et en colones
end record;
```

Figure 3 : Implémentation de la structure de donnée matrice_creuse.

- De plus, nous avons choisis de ne pas générer la matrice G (à partir de la matrice S) comme pour le module matrice pleine. En effet, nous avons intégré cela à la procédure produit-vect-mat qui trouve la coordonnée, la multiplie par α et lui rajoute $\frac{(1-\alpha)}{N}$ avant de la multiplier par la coordonnée du vecteur V. Ceci nous fait gagner de la place en mémoire vu que nous n'avons pas besoin de stocker S et G (qui n'a aucune valeur nulle). De plus, le calcul est moins long vu que ça nous évite de faire le parcours deux fois.

3- <u>Les Résultats et les durées d'exécution du programme en fonctions des implantations</u> :

Légende : Matrice pleine Matrice Creuse Sujet du projet

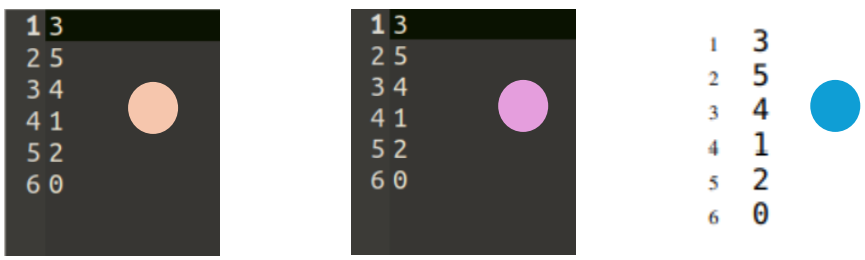

Figure 3 : Vecteur classement de l'exemple sujet.net obtenu avec matrice pleine, matrice creuse et le sujet du projet.

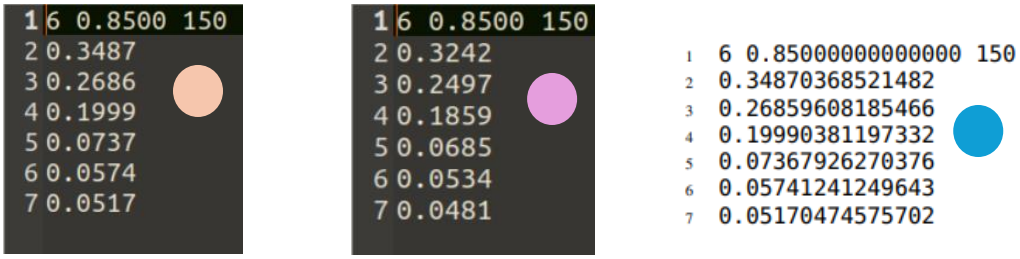

Figure 4 : Vecteur poids de l'exemple sujet.net obtenu avec matrice pleine, matrice creuse et le sujet du projet.

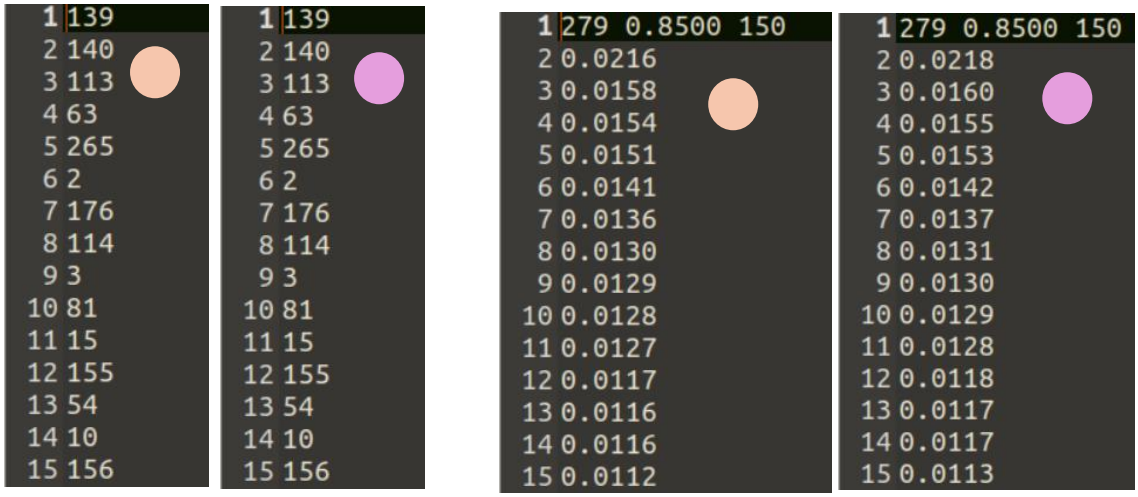

Figure 5 : Vecteur poids de l'exemple worm.net obtenu avec matrice pleine et matrice creuse.

On remarque sur les photos ci-dessus que pour chacun des exemples sujet.net et worm.net, le classement est le même pour les 2 modules (nous n'avons reporté que les 15 premières valeurs pour l'exemple worm mais vous pourrez vérifier que ceci est également valable pour le reste). Or le fichier poids calculé par matrice creuse contient des valeurs légèrement différentes que celles attendues. Comme la différence est minime, celle-ci ne change pas le classement.

Nous pensons que ceci peut être dû à une différence de précision ou une différence du nombre d'itérations.

Exemple	Durée d'exécution avec matrice pleine	Durée d'exécution avec matrice
Sujet	0,001 s	0,004s
Worm	0,046 s	0,277 s
Brain links		
Linux26		

Tableau 1 : Comparaison des durées d'exécution du programme en fonction de l'implantation.

Conclusion : On remarque que le temps de calcul pour matrice pleine est moins grand que celui pour matrice_creuse. Cela veut dire que notre parcours n'est pas assez optimisé et qu'il faudrait trouver un moyen de le rendre plus rapide.

Or pour des modules beaucoup plus grands comme brainlinks et linux26, le programme avec matrice pleine renvoyait directement l'erreur « stack overflow or erroneous memory access » dû au nombre de calcul à réaliser. Cependant, avec des matrices creuses, le programme ne nous renvoie plus directement un message d'erreur, au contraire il continue de tourner. Ces derniers ont bel et bien besoin d'un temps de calcul très élevé mais ils sont bien lancés.

Cela veut dire que nous avons pu gérer le problème de mémoire en revanche ça nous a créé un problème de temps. En effet, après 672 min le graphe de linux26 n'avait pas encore fini de tourner.

4- Les difficultés rencontrées et les solutions adoptées :

a) Nous avions au départ fait le choix de mettre l'analyse de la ligne de commande et la lecture du fichier dans un seul module. Or par la suite, lorsque nous avons voulu tester le programme PageRank, nous nous sommes rendu compte que nous avions besoin d'utiliser un paramètre (la taille, qu'on ne découvre que pendant le déroulement du programme) pour instancier le module matrice_pleine. Comme les instanciations de modules ne peuvent se faire que dans la partie déclarations du programme, ceci n'a donc pas été possible.

   Afin de remédier à ce problème, nous avons donc décider de déplacer le sous-programme qui lit le fichier texte et de le mettre dans les modules matrices pleine et matrice creuse.

b) Après le premier test avec le module matrice_creuse, les temps de calculs étaient beaucoup plus longs que les résultats que nous avons actuellement. En effet, l'exemple worm.net avait besoin de 5,9s pour être traité alors qu'actuellement il n'en demande que 0,277s. Ce temps n'est pas parfait mais il est beaucoup plus efficace que ce que nous avions au départ.
   Afin de faire ce changement, nous avons enlevé tous les appels à la fonction get_value dans le sous-programme qui fait le produit matriciel. Cette fonction donne la valeur de la coordonnée à l'emplacement i,j de la matrice. Ceci nous faisait faire beaucoup de calculs qui sont très coûteux. A la place nous avons mis un curseur qui avance avec les itérations et qui nous permet donc d'éviter cette redondance de parcours.

5- <u>Les grilles d'évaluation du code et des raffinages</u> :

Evaluation des raffinages :

		Evaluation Etudiant (I/P/A/+)	**Justification / commentaire**	**Evaluation Enseignant (I/P/A/+)**
Forme (D-21)	Respect de la syntaxe.	+	Les raffinages commencent par comment et sont donnés sous la forme d'une question et ils sont bien indentés..	
	Verbe à l'infinitif pour les actions complexes	+	Toutes actions complexes commencent par un verbe à l'infinitif.	
	Nom ou équivalent pour expressions complexes.	+	Aucun nom utilisé.	
	Tous les Ri sont écrits contre la marge et espacés	+	Bien écrits comme il faut.	
	Les flots de données sont définis	+	On a bien les variable avec leur modes et leur types.	
	Une seule décision ou répétition par raffinage.	+	Une seule décision uniquement dans chacune des étapes.	
	Pas trop d'actions dans un raffinage (moins de 6)	+		
	Bonne présentation des structures de contrôle	+	Bonne indentation.	
Fond (D21-D22)	Le vocabulaire est précis	+	Le texte écrit décrit bien les actions faites.	
	Le raffinage d'une action décrit complètement cette action	+	Aucune action n'est incomplète et elles se finissent toutes dans l'étape de raffinage associée.	
	Le raffinage d'une action ne décrit que cette action	+		
	Les flots de données sont cohérents	+		
	Pas de structure de contrôle déguisée	+	Tout est bien écrit avec la bonne syntaxe du language algorithmique.A chaque fois il utilise bien les Si Sinon et non si ou if par exemple.	
	Qualité des actions complexes	+	Répondent bien au sujet demandé.	

Evaluation du code :

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...	
	O	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).	

6- Répartition du travail :

Module	Spécifier	Programmer	Tester	Relire
Input-Output	Arthur /Ainji	Arthur	Arthur /Ainji	Ainji/Arthur
Matrice Pleine	Ainji	Arthur/Ainji	Ainji	Arthur/Ainji
Matrice Creuse	Ainji	Arthur/Ainji	Arthur	Arthur/Ainji
Page Rank	Arthur	Arthur	Arthur/Ainji	Arthur/Ainji

7-Conclusion :

Finalement dans ce projet, nous avons pu recréer l'algorithme pageRank en l'implémentant avec le module matrice pleine et matrice creuse et en obtenant à chaque fois les bons résultats (à une petite précision près pour matrice creuse). Nous avons remarqué que les calculs de graphes très grands ne pouvaient pas être effectués par le module matrice pleine à défaut de place mémoire mais qu'il pouvait être réalisé par le module matrice creuse (même si cela prend un temps de calcul extrêmement long).

Points d'amélioration qui pourraient être abordés pour la suite :

a. Diminuer le temps de calcul :
   - Limiter le nombre de parcours effectués dans la matrice.
   - Regrouper des cas de test qui sont différenciés mais qui pourraient être les mêmes.
   - Implémenter les sous-programmes de façon récursive et non itérative.
b. Essayer de réintégrer la lecture du fichier en entrée et la création des fichiers poids et classement en sortie dans le module Input-Output plutôt que de le mettre deux fois dans deux modules différents.