Rapport Page Rank

Réalisé par :

Nacere Mohammed & Ettayach Sohaib

19 Janvier

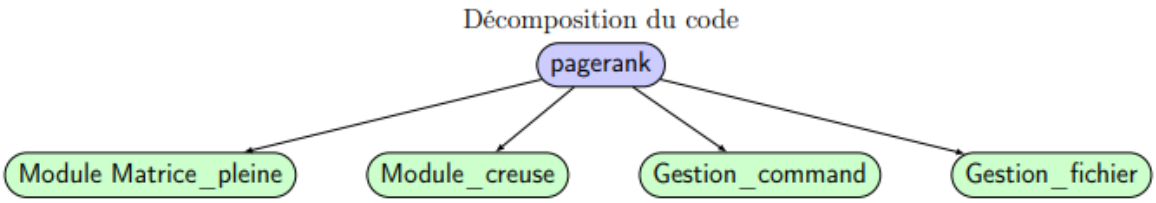
2024

Tables de matière

Introduction

Le présent rapport expose le projet de développement d'un programme en Ada visant à implémenter l'algorithme de PageRank, qui évalue la popularité des pages Internet. L'objectif principal est de fournir plusieurs implantations de cet algorithme, en comparant leurs performances. Le programme prendra en charge des arguments de ligne de commande permettant de choisir l'implantation à utiliser ainsi que les valeurs des paramètres de l'algorithme. Le cahier des charges détaille les spécifications du projet, notamment la définition du PageRank proposée par Brin Page en 1998. L'algorithme mesure la popularité des pages en fonction des liens hypertextes qui les référencent. La représentation textuelle d'un graphe orienté, la gestion des fichiers résultats, la précision des calculs, et d'autres aspects sont également abordés. Les contraintes du projet, les livrables attendus, les échéances et les critères d'évaluation sont présentés de manière exhaustive. L'algorithme sera implémenté en utilisant la matrice de Google G, et différentes précisions des calculs seront prises en compte.

Représentation générale



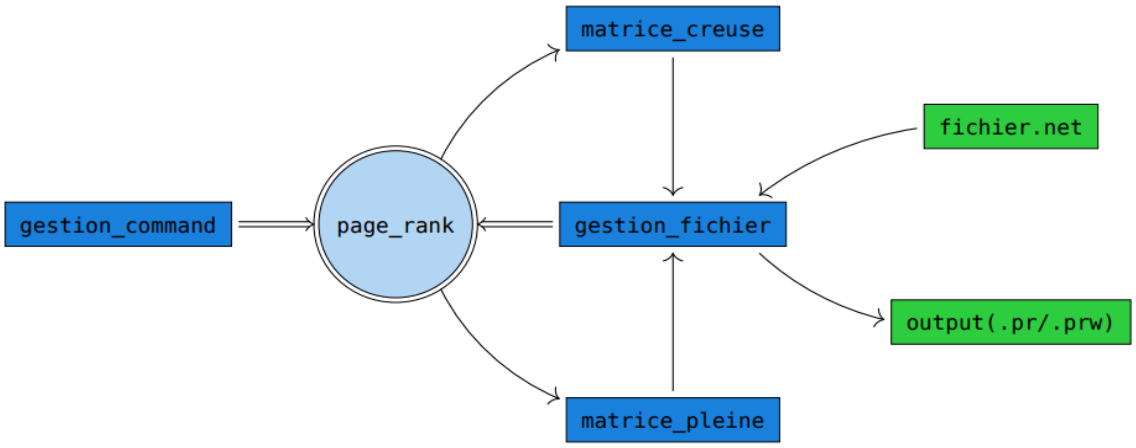
Dans le cadre du projet PageRank, nous avons intégré deux modules dédiés au calcul matriciel du vecteur poids. Le premier module, basé sur la matrice creuse, utilise la LCA (Liste de Cohérence d'Adresses) telle que discutée en cours, travaux dirigés et travaux pratiques. Le second module repose sur la matrice pleine. Ainsi que deux modules pour gestionner les fichiers et les commandes.

Modules

Notre programme se divise en cinq modules distincts, chacun étant attribué à un rôle spécifique : `**gestion_commande**`, `**gestion_fichier**`, `**matrice_creuse**`, `**matrice_pleine`,** et enfin, le traitement particulier de l'algorithme `**pagerank**`.

Chaque module comprend une partie spécification définissant ses fonctionnalités et une autre consacrée à son corps de programme. La cohésion entre ces modules est essentielle pour assurer le bon déroulement du programme. Le diagramme de dépendance entre eux illustre les liens et les relations, soulignant ainsi les interdépendances nécessaires. Ces connexions permettent à chaque module de tirer parti des fonctionnalités des autres, créant ainsi une collaboration harmonieuse dans l'exécution du programme global. Cette approche modulaire facilite la maintenance, l'extensibilité et la compréhension du code, fournissant une architecture robuste et adaptable pour notre programme.

Diagramme de dépendance



Dans ce diagramme, deux types de flèches ont été utilisés pour représenter différentes interactions. Les flèches avec des doubles traits indiquent les modules qui seront sollicités en préalable au calcul. Ces modules jouent un rôle crucial dans la préparation et la configuration avant le déclenchement du processus principal. D'autre part, les flèches simples mettent en évidence les étapes du calcul et le transfert des fichiers finals. Ces flèches symbolisent le flux du programme, du traitement initial à la production des résultats finaux.

De plus, des éléments en vert ont été identifiés pour représenter les fichiers qui, bien qu'ils ne soient pas des modules à proprement parler, occupent une place significative en tant qu'entrées et sorties du programme. Cette coloration distinctive facilite la compréhension du mode de manipulation des fichiers au sein de notre programme. Ces fichiers verticaux sont essentiels pour la communication des données entre les différents modules et sont cruciaux pour l'orchestration globale du processus.

Cette conception permet ainsi une visualisation claire du flux d'exécution, de la préparation initiale à travers les modules déclenchés en préface, jusqu'au traitement central et à la restitution des résultats par le biais des fichiers verts identifiés. Cela renforce la compréhension du fonctionnement global de notre programme et de la façon dont il manipule les données tout au long du processus.

Choix réalisés

Nous avons dû prendre des décisions cruciales quant à la répartition de notre projet, notamment le nombre de modules et la manière de structurer le code au sein de ces modules

Nous avons ainsi décomposé notre module PageRank initial en cinq modules distincts, à savoir « **gestion_commande** », « **gestion_fichier** » ,« **matrice_pleine** », et « **matrice_creuse** ». Cette nouvelle organisation a grandement facilité notre compréhension du projet, aussi bien dans sa phase de raffinement que dans la rédaction du code. Cette répartition nous a permis de mieux nous orienter et de coder de manière plus fluide l'ensemble du projet.

En choisissant cette répartition entre ces cinq modules, notre objectif était de simplifier les liens entre eux tout en renforçant leur aspect pratique et efficace pour le codage. Cette démarche a été essentielle pour garantir une structure claire et faciliter le développement harmonieux du projet.

Mise au point des modules

Les modules ont tous été minutieusement planifiés, notamment grâce à une première étape de raffinage. Ce processus nous a été bénéfique pour acquérir une compréhension approfondie de la manière dont chaque module devrait être codé, garantissant ainsi un fonctionnement sans heurts. Nous avons initié cette démarche en se concentrant sur le module « **gestion_commande** ».

Dans ce module, nous avons élaboré un algorithme qui analyse la ligne de commande. Pour chaque paramètre détecté, l'algorithme exécute une action spécifique, accompagnée d'une vérification. Ces actions et vérifications sont étroitement liées à l'option entrée par l'utilisateur. Cette approche nous a permis de mettre en œuvre une gestion précise des commandes, alignée sur les fonctionnalités attendues, tout en anticipant les éventuels problèmes de fonctionnement des modules.

Parallèlement, nous avons élaboré le module "**matrice_pleine**", dédié au calcul, à l'initialisation et à l'implémentation des matrices, ainsi qu'aux calculs associés. Dans nos structures de matrices, où les colonnes et les lignes sont représentées par des tableaux distincts, nous avons ainsi pu garantir la cohérence de nos développements. En nous appuyant sur ces réflexions, nous avons codé en toute confiance, assurés de la conformité de notre implémentation avec les fondements mathématiques des calculs matriciels requis.

Pour le développement du module "**matrice_creuse**", une réflexion approfondie a été nécessaire. Nous avons exploré diverses options et avons finalement opté, comme mentionné précédemment, pour l'utilisation de tables de hachage. Cette décision s'est avérée particulièrement

judicieuse, car les tables de hachage offrent une solution pratique pour stocker les données et y accéder rapidement.

Grille d'évaluation du code

		Eval. étudiant	Justif./Comm.	Eval. enseignant
Forme	Respect de la syntaxe Ri : Comment « … une action complexe … » ? des actions combinées avec des structures de controle Rj : …	TB		
	Verbe à l'infinitif pour les actions complexes	TB		
	Nom ou équivalent pour expressions complexes	TB		
	Tous les Ri sont écrits contre la marge et espacés	TB		
	Les flots de données sont définis	B		
	Pas trop d'actions dans un raffinage (moins de 6)	TB		
	Bonne présentation des structures de contrôle	TB		
Fond	Le vocabulaire est précis	TB		
	Le raffinage d'une action décrit complètement cette action	TB		
	Les flots de données sont cohérents	TB		
	Pas de structure de contrôle déguisée	TB		
	Qualité des actions complexes	TB		

Problèmes rencontrés

Le problème de type que nous rencontrons concerne la définition du type `**T_Vecteur**`, initialement défini dans le module `**Matrice_Pleine**`. Il est impératif d'utiliser ce même type **`T_Vecteur`** dans le module `**Matrice_Creuse**`, notamment dans la fonction `**Multiplication_V_M_C`.** Cependant, l'utilisation de `T_Vecteur` instancié soit depuis le module `**Matrice_Pleine**`, soit redéfini avec le même nom dans le module `**Matrice_Creuse**` génère des dysfonctionnements dans le programme PageRank.

En effet, le programme PageRank doit pouvoir lire le type `T**_Vecteur**` instancié dans `**Matrice_Creuse**` ou défini à nouveau dans `**Matrice_Pleine**`. Cette dualité pose problème et peut entraîner des conflits. La solution envisagée consiste à éviter de redéfinir le type `**T_Vecteur**` dans `**Matrice_Creuse**` et de ne pas instancier le module `**Matrice_Pleine**` dans `**Matrice_Creuse**`.

Pour résoudre ce problème de type, il est proposé de définir la fonction `**Multiplication_M_V_C`** directement dans le programme PageRank. Ainsi, on contourne le problème du type en utilisant le type `**T_Vecteur**` instancié par `**Matrice_Pleine**` en toute tranquillité, tout en maintenant l'utilisation du type `**T_Matrice_Creuse**` dans `**Matrice_Creuse**`.

Cette approche permet d'éviter les conflits potentiels et assure une cohérence dans l'utilisation des types entre les modules, garantissant ainsi le bon fonctionnement du programme PageRank.