Couturier Arthur

1SN Groupe EF-08

Coquard Pierre-Jean

Enseignant: Xavier Crégut

Compte rendu de projet PIM:

Pagerank

Année 2020/2021

Sommaire

Présentation du Projet / Cahier des charges

Déroulement du Projet

I/ Architecture de pagerank:

II/ Principaux choix réalisés:

III/ Manière dont les programmes et modules ont été mis au point:

IV/ Durées moyennes d'exécution:

Perspectives

I/ Conclusions tirées de ces résultats:

II/ Difficultés rencontrées et solutions mises en oeuvre:

III/ Conclusion sur l'état d'avancement et les perspectives d'amélioration du projet:

IV/ Apports personnels tirés de ce projet:

A) Présentation du Projet / Cahier des charges

Résumé du projet:

Ce rapport a pour but de vous présenter le projet, les problématiques qu'il aborde ainsi que la mise en place de notre algorithme permettant de répondre à ces problématiques. Vous y trouverez tout d'abord une courte introduction au sujet du projet, l'architecture de notre algorithme principal de résolution: pagerank, les principaux algorithmes utiles à la manipulation de pagerank, la manière dont les différents programmes et modules ont été mis au point, les résultats moyens de durées d'exécution obtenus, les conclusions qu'il est possible d'en tirer, les difficultés rencontrées et solutions mises en œuvre afin de surmonter à ces complications, une conclusion générale sur l'avancement du projet ainsi que les perspectives d'amélioration de l'algorithme et enfin les apports personnels tirés de ce projet.

Introduction générale du projet:

Historiquement, le terme de Pagerank a été introduit en 1998 dans le but de classer la popularité de pages internet. Ce projet a pour objectif de concevoir un algorithme similaire au moyen du langage de programmation Ada à l'aide de compétences acquises en cours de programmation impérative. Cette popularité d'une page est déterminée par le nombre de pages qui référence cette première ainsi que leur propre popularité. Ainsi, le référencement d'une page populaire a plus de poids dans l'algorithme du pagerank que celui d'une page moins populaire dans l'algorithme. Ce principe de poids est retrouvé au long de notre programme car c'est précisément comme cela que la popularité des pages est calculée.

Nous utiliserons en fait une matrice de poids notée π_k à l'itération k de réalisation. Le calcul de cette matrice de poids dépendra notamment de G, une matrice dite de google qui peut être calculée de manière simple: algorithme dit par Google Naïve, ou plus fine: par Google Creuse en s'intéressant au cas des matrices creuses, c'est à dire en limitant l'espace pris par une telle matrice.

En effet, on retrouve là une des principales problématiques du sujet. Le but est en fait de limiter l'espace pris par la matrice de Google afin dans un premier temps d'avoir une matrice de taille abordable, et dans un second temps d'accélérer les calculs de l'algorithme: plus la matrice de Google est grande, plus les calculs seront longs.

B) Déroulement du Projet

I/ Architecture de pagerank:

Cet algorithme est composé de 4 grandes parties: la compréhension de ligne de commandes, le calcul de la matrice de poids par Google Naïve ou le calcul de la matrice de poids par Google Creuse et enfin l'affichage des résultats avec tri des poids.

Compréhension de lignes de commande:

Un point clé du projet est de pouvoir lancer l'algorithme par une ligne de commandes dans le shell de linux. Ces commandes permettent d'ajouter des options sur ce que l'on souhaite tirer de l'algorithme. On pourra alors décider d'un calcul par Google Naïve ou Creuse, de la valeur d'un paramètre alpha qui sera détaillé dans la suite du rapport, du nombre d'itération de calcul souhaité (obtenir π_N en sortie d'algorithme si ce nombre vaut N) ainsi que la base de données utile sur laquelle on souhaite réaliser ces calculs.

Tri des poids:

Le tri des poids est la finalité de ce projet. En effet, le classement des pages web fait dans ce projet est à destination d'un tri qui permet de recommander en premier temps une page bien classée. Ce n'est qu'à la fin de cette étape de tri que la matrice π_N est envoyée à l'utilisateur.

II/ Principaux choix réalisés pour mener l'étude:

Plusieurs choix s'offraient à nous pour ce projet. Nous avons choisi d'avoir recours à des modules et packages bien distincts offrant chacun de la généricité afin de pouvoir les utiliser de différentes manières. Il était nécessaire de procéder ainsi car les algorithmes de Google Creuse et Google Naïve étaient bien distincts par leur architecture.

Choix d'améliorer la compréhension de l'algorithme principal:

Afin d'alléger et d'améliorer la compréhension de Pagerank, nous avons choisi de définir dans les modules Google 2 procédures: l'initialisation de la matrice de Google, puis le calcul du vecteur de poids. Ainsi, ces deux

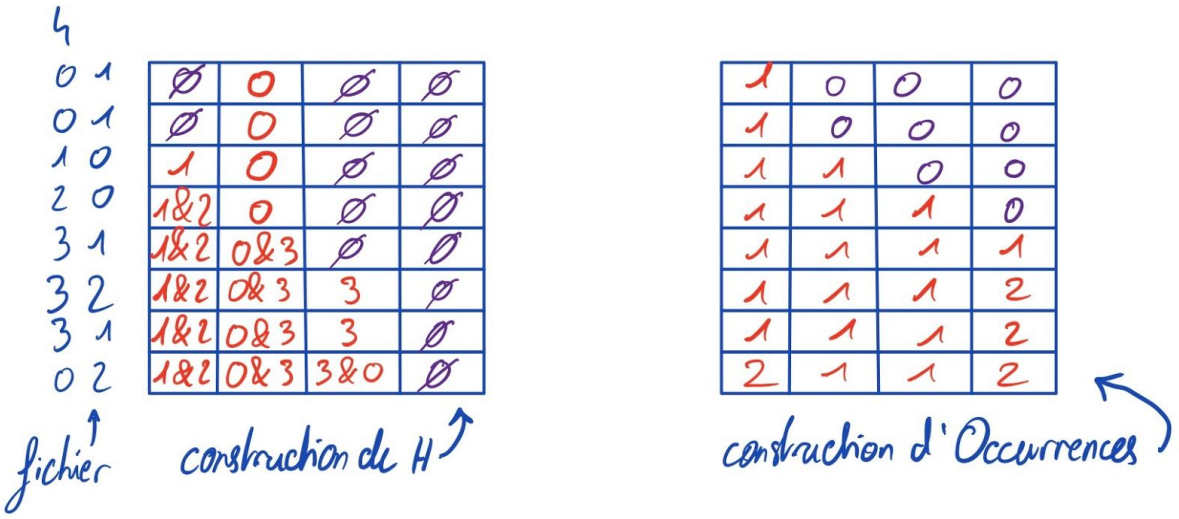
procédures sont appelées dans l'algorithme Pagerank mais sont détaillées dans les modules Google. Par exemple, là ou l'initialisation de la matrice de Google ne requiert que la manipulation de grosses matrices dans Google Naïve, l'initialisation dans Google Creuse se fait en parallèle avec une matrice auxiliaire nommée Occurrences qui est nécessaire dans l'appel de la procédure de calcul du poids.

Choix de l'algorithme de Google Naïve:

Google Naïve, étant un programme bien guidé par l'énoncé du projet, a été fait avec assez peu de choix. Néanmoins, c'est lors de sa conception que nous avons conçu le module permettant de manipuler des matrices. En effet, ce package use de volumineuses matrices pour fonctionner et les multiplie entre elles afin de calculer π_N. Cependant, le module matrice que nous avons mis au point à cette étape ne permet pas toutes les opérations imaginables sur des matrices, mais simplement celles qui nous paraissaient utiles au développement de Google Naïve. Par exemple, nous n'avons pas implanté la possibilité d'additionner deux matrices car cela nous était inutile tandis que nous avons implanté l'opération de transposée, utile notamment à la dernière étape de calcul de π_N.

Choix de l'algorithme de Google Creuse:

L'algorithme de Google Creuse était l'algorithme pour lequel nous avions le plus de choix. Nous avions l'idée de n'utiliser que des structures chaînées pour concevoir la matrice de Google mais ce choix qui simplifiait l'étude lors de l'initialisation de la matrice de Google allait s'avérer peu efficace lors de l'étape de calcul du poids (qui de plus s'exécute N fois). C'est pourquoi nous avons choisis de séparer les informations en 2 matrices: la matrice H (matrice intermédiaire aux calculs) qui deviendra ensuite G (matrice de Google), ainsi que la matrice Occurrences. Vous trouverez ci-après un schéma de pensée de notre algorithme:



À la suite de l'étape d'initialisation, H est une matrice de N structures chaînées nulles (représentées sur le schéma par des ensembles vides violets), et Occurrences est une matrice remplie de N zéros. Lors de l'étape 1, 0 redirige vers 1 donc on ajoute une cellule à l'emplacement de la page 1 sur H indiquant que 0 redirige vers cette page. Plus tard, à l'étape 5, on remarque que cette cellule redirige vers une cellule indiquant que la page 3 redirige aussi vers la page 1. Dans Occurrences, on ajoute 1 à la case k si la k ième page en redirige une nouvelle. Lorsqu'il y a plusieurs fois la même redirection (une page redirige à plusieurs reprises vers la même page), on ne fait rien, comme à l'étape 2 qui est la même que l'étape 1 par exemple.

Cette méthode permet, une fois l'initialisation de G terminée, de pouvoir calculer le poids bien plus efficacement. On sait rapidement si une page de redirige vers aucune autre (présence d'un 0 dans Occurrences) et quelles pages redirige vers une même page (et donc si elle est populaire et recommandable à notre sens).

Optimiser l'étape de calcul du poids est plus important que l'optimisation de l'initialisation de la matrice G car cette étape est répétée plusieurs fois tandis que l'initialisation de G ne se réalise qu'une fois. C'est d'ailleurs un intérêt d'utiliser des matrices creuses: pouvoir calculer de manière efficace le poids et ne pas avoir à faire de très nombreuses opérations par 0 répétées d'autant plus de fois que la précision du poids souhaité est grande.

Choix de l'algorithme de tri:

L'algorithme que nous avons choisi pour le tri est souvent appelé tri rapide. Ce choix a été fait en raison de la volonté de rapidité de calcul souhaitée lors de ce projet. En effet, sa complexité est en O(N*log(N)) avec N le nombre de pages à trier, ce qui pour un algorithme à destination d'un

nombre N très grand est le plus optimal. De plus, cet algorithme ne nécessite pas de créer un second tableau pour l

III/ Manière dont les programmes et modules ont été mis au point:

Mise au point des programmes et modules:

Nous avons tout d'abord pensé la structure globale du programme Pagerank pour ensuite déduire les packages Google. Puis c'est en concevant le premier package Google Naïve que nous avons remarqué le besoin d'un module permettant de manier les matrices. Par la suite, en souhaitant développer le package Google Creuse, nous avons eu besoin de manipuler des Sda (structure de données chaînées). Nous avons donc implanté un nouveau module permettant cela.

Enfin, un dernier module a été nécessaire: le module de tri. Car, comme pour les modules Google, nous souhaitions améliorer la clarté du programme Pagerank en définissant les étapes de tri dans un module complémentaire. Cette séparation des étapes en modules complémentaires au programme Pagerank ou ses 2 packages Google nous a de plus permis de travailler simultanément sur deux modules différents via subversion. Nous procédions enfin à une vérification des deux codes.

IV/ Durées moyennes d'exécution:

Avec Google Naïve:	**Avec Google Creuse:**
time ./obj/pagerank -P worm.net real 0m0,075s user 0m0,075s sys 0m0,000s	time ./obj/pagerank worm.net real 0m0,040s user 0m0,034s sys 0m0,005s
Execution of ./obj/pagerank terminated by unhandled exception raised STORAGE_ERROR : stack overflow or erroneous memory access	time ./obj/pagerank brainlinks.net real 0m49,991s user 0m49,921s sys 0m0,064s

Exemple moyen de durée d'exécution

C) Perspectives

I/ Conclusions tirées de ces résultats:

On constate plusieurs différences entre les résultats de la version Google Naïve et la version Google Creuse. Tout d'abord, il est à noter que lorsque les deux réussissent à s'exécuter, les résultats dans les fichiers.ord sont les mêmes (le classement des pages est le même à la sortie des algorithmes). Cependant, on remarque que sur worm, Google Naïve est près de 2 fois plus long à s'exécuter. Cet écart se creuse davantage lorsque N grandit, en particulier car la complexité de Google Naïve est en O(N) tandis que celle de Google Creuse est en O(N*log(N)).

De plus, on remarque que l'algorithme avec Google Naïve n'arrive pas à s'exécuter par manque de mémoire pour stocker la matrice G, tandis que Google Creuse nous donne un résultat en moins d'une minute.

II/ Difficultés rencontrées et solutions mises en oeuvre:

Nous avons rencontré beaucoup de problèmes lors de l'implantation de Google Creuse. Tout d'abord, nos premières implantations, en plus de ne pas calculer ce que l'on souhaitait, ne compilaient pas à cause de nombreux problèmes de typage. C'est d'ailleurs la raison qui nous a poussé à supprimer les parties privées et limitées privées de nos codes car le typage ne se transmettait pas d'un package à l'autre.

Par la suite, nous nous sommes aperçus que les calculs ne correspondaient pas aux résultats souhaités. Nous avons alors dû fixer cela en redéfinissant des étapes clés de notre algorithme Google Creuse pour ainsi modifier notamment la partie de calcul du vecteur poids.

III/ Conclusion sur l'état d'avancement et les perspectives d'amélioration du projet:

Le projet est dans un état fonctionnel pour des réseaux de taille limitée, mais il pourrait être optimisé de plusieurs manières. Tout d'abord, l'implémentation naïve de notre programme est peu efficace en mémoire puisqu'on duplique plusieurs fois les données, notamment dans la fonction

Calculer_Vecteur_Poid. Cela cause d'un usage excessif de mémoire et empêche la bonne exécution pour des réseaux de grande taille, comme pour brainlinks. L'implémentation google creuse est plus efficace en temps et en mémoire mais pourrait être aussi améliorée en réorganisant les types de données pour créer des modules plus clairs.

IV/ Apports personnels tirés du projet:

Ce projet nous a permis de mieux appréhender tout d'abord le traitement réalisé sur les pages internet pour le référencement, fonction que l'on exploite tous de multiples fois par jour.

Par ailleurs, le projet était l'occasion de pouvoir tester concrètement les aptitudes acquises sur le langage Ada, en particulier l'utilisation de nombreux modules et package utiles à un même algorithme.

De plus, le projet nous a aussi montré l'importance de l'efficacité d'un programme par son temps d'exécution car cela était l'une des principales visées de l'exercice.

Enfin, le projet nous a permis de mieux appréhender le travail sur un même algorithme en groupe. Cela est une nouveauté car la plupart des programmes que nous réalisions avant dans notre scolarité étaient des programmes que l'on concevait seul. Nous avons donc via ce projet appris le travail de groupe à l'aide de subversion, le fait d'avancer en simultanée sur deux aspects de l'algorithme puis de mettre en commun ces parties dans le Pagerank.