Calcul efficace du PageRank (Projet PIM)

Sébastien Pont, Philippe Négrel-Jerzy

Equipe KL-02

Résumé :

Ce rapport présente la démarche que nous avons suivi pour réaliser l’algorithme de PageRank de façon « naïve » c’est-à-dire en utilisant d’abord un système de stockage peu optimisé en terme d’espace mémoire requis, puis selon une logique plus dynamique et adaptative, en utilisant des matrices creuses. On rappellera dans un premier temps les termes imposés par le sujet, puis on explorera l’architecture du programme principal, ainsi que les modules entrant en jeu, tout en présentant l’évolution du code et les choix réalisés pour en arriver à cette version finale. Par la suite, on abordera le thème des performances des algorithmes mis en place, ainsi que leurs temps d’exécution respectifs. Enfin, on conclura sur les potentielles améliorations qui peuvent être apportées au projet dans son état actuel, ainsi que les bénéfices que nous avons tiré de sa réalisation.

Introduction :

Dans ce projet, comme précisé plus haut, on cherche à mettre en œuvre l’algorithme de PageRank en langage Ada, qui consiste, à partir d’une liste de sites internet appartenant à un même réseau, et de connexions entre ces derniers (relations de référencement, un site référence zéro ou plus autres sites) , à donner leurs « poids » respectifs, i.e. leur popularité au sein de ce réseau. Le plan du rapport, qui constitue le fruit de nos recherches, est présenté ci-après :

1. Architecture globale de l’application
2. Programme principal
3. Module Google
4. Module Google\_naive
5. Module Google\_creuse
6. Modules Helpers et Exceptions
7. Choix de conception et stratégies mises en œuvre
8. Interface utilisateur
9. Stockage des données
10. À propos du module google
11. Organisation des fichiers source
12. Performances et intérêt de la méthode creuse
13. Temps d’exécution
14. Impact sur la mémoire
15. Obstacles rencontrés
16. Limites liées au stockage
17. *Une autre partie peut-être jsp trop*
18. Conclusion
19. Architecture globale de l’application
20. Programme principal

Dans le fichier pagerank.adb, où se situe notre programme principal, on importe les modules Ada requis, ainsi que les packages Exceptions, Helpers et Google.

with Ada.Text\_IO;           use Ada.Text\_IO;

with Ada.Strings.Unbounded; use Ada.Strings.Unbounded;

with Ada.Exceptions;        use Ada.Exceptions;

with Exceptions;            use Exceptions;

with Helpers;               use Helpers;

with Google;

Ce dernier ne sera instancié que lors de la définition de la procédure Pakerank afin de préciser les arguments génériques requis :

package Google\_Table is new Google

   (T\_Float => T\_Double, Capacite\_Max => 10\_000);

use Google\_Table;

Par la suite, on appelle la procédure Nettoyer\_Console, on lit les arguments donnés par l’utilisateur dans la console (procédure Lire\_Args), et on exécute les sous-programmes qui permettent le calcul des poids et leur tri, ainsi que leur stockage :

*--  Supprimer les lignes du terminal*

**Nettoyer\_Console**;

*--  Lire les arguments fournis par l'utilisateur*

**Lire\_Args** (Nom\_Fichier, A, Max\_Iter, P, Aide\_Demande);

   if not (Aide\_Demande) then *-- Si l'utilisateur ne demande pas de l'aide*

*--  Recuperer le nombre de pages*

      Taille := Nb\_Page (Nom\_Fichier);

*--  Informer l'utilisateur de la taille requise pour le calcul*

**Estimer\_Donnees**(Taille, Max\_Iter);

*-- Calculer la matrice G*

**Calculer\_Coefs**(Nom\_Fichier, Taille, A);

*--  Initialiser le tableau pi des poids*

**Initialiser\_Poids** (Pi, Taille);

*--  Rechercher la valeur finale du vecteur pi*

**Calculer\_Poids** (Pi, Taille, Max\_Iter, Nom\_Fichier);

*--  Trier les elements de pi par ordre de poids croissant*

**Trier\_Poids** (Pi\_Trie, Pi, Taille);

*--  Generer les fichiers de sortie*

**Creer\_Fichiers** (Pi, Pi\_Trie, Taille, A, Max\_Iter, Nom\_Fichier);

   end if;

Les exceptions sont gérées dans le bloc exception et affichent des messages d’erreurs détaillant l’origine de l’exception à l’utilisateur :

exception

   when Erreur\_Lire\_Args => *-- le message d'erreur a deja ete emis*

      null;

   when Erreur\_Lecture\_Taille =>

**Put\_Line**

        ("Une erreur est survenue lors de la lecture de la taille du reseau.");

**Put\_Line**

        ("Veuillez vérifier que la premiere ligne du fichier est bien");

**Put\_Line** ("un entier representant la taille du reseau.");

   when Erreur\_Capacite\_Max\_Depassee =>

**Put\_Line** ("Attention la taille du reseau est trop grande.");

**Put\_Line** ("Veuillez augmenter la valeur de la variable CAPACITE\_MAX");

**Put\_Line** ("dans le fichier pagerank.adb");

   when E : others =>

**Put\_Line** ("Une erreur non geree est survenue :");

**Put\_Line** (Exception\_Message (E));

end **Pagerank**;

1. Module Google

On importe encore les mêmes modules Ada, ainsi que les packages Helpers, Google\_Naive, et Google\_Creuse que l’on va instancier dans la spécification :

package Google\_Naive\_Table is new Google\_Naive

   (T\_Float => T\_Double, Capacite\_Max => 10\_000);

use Google\_Naive\_Table;

package Google\_Creuse\_Table is new Google\_Creuse

   (T\_Float => T\_Double);

use Google\_Creuse\_Table;

1. Module Google\_Naive
2. Module Google\_Creuse
3. Modules Helpers et Exceptions
4. Choix de conception et stratégies mises en œuvre
   1. Interface utilisateur
   2. Stockage des données
   3. A propos du module google
   4. Organisation des fichiers source
5. Performances et intérêt de la méthode creuse
   1. Temps d’exécution
   2. Impact sur la mémoire
6. Obstacles rencontrés
   1. Limites liées au stockage
7. Conclusion