Calcul efficace du PageRank

Rapport du projet

Programmation impérative

Cyprien ARNOLD
Maxime HENRY
Groupe AB – 13

Samedi 16 janvier 2021
ENSEEIHT – Sciences du Numérique
1ère année

Sommaire

I/ Introduction

Brin & Page ont proposé en 1998 de mesurer la popularité des pages Internet en les triant de la plus populaire à la moins populaire selon un ordre appelé PageRank. L'objectif de ce projet est de fournir les algorithmes nécessaires au calcul du PageRank pour un ensemble de pages Internet ou un réseau dirigé donné.

Pour calculer ce PageRank, il est nécessaire de calculer le poids associé à chaque page. Pour cela, nous effectuons le calcul matriciel d'une matrice appelée G (pour matrice de Google) avec un vecteur Pi (pour Poids) qui représente le vecteur des poids des pages. Plus nous effectuons ce calcul et plus le vecteur des poids sera précis (les mathématiques nous montrent la convergence de ce schéma).

Une fois un certain nombre de ces opérations effectuées, il faut trier les poids associés aux pages de manière décroissante. Une fois cela classé, ce tri des pages selon leur poids associé constitue le PageRank.

Nous avons implémenté cet algorithme en langage ADA. Notre implémentation permet de choisir :

- Le nombre d'itération du calcul matriciel
- Le coefficient Alpha utile au calcul de la matrice de Google
- La structure de la matrice G avec une matrice dite creuse et une matrice dite pleine

Notre programme prend en argument de manière optionnelle les trois paramètres définis ci-dessus et un fichier .net qui constitue un fichier texte rempli uniquement de nombres où sont explicités les différents liens entre les pages.

Il renvoie un fichier .p où sont écrits les poids de chaque page, de manière décroissante et un fichier .ord où sont présents les numéros des pages dans l'ordre décroissant de leur poids.

II/ Présentation de l'architecture du programme

Nous avons un fichier pagerank.adb qui constitue notre programme principal

Dans un premier temps il fait appel à :

```
begin
    -- On traite les arguments de la ligne de commande
    TraiterArguments(Alpha, estMatricePleine, nbIteration, nomFichier);
```

Cette fonction TraiterArguments est présente dans le module Arguments et permet de récupérer les arguments écrits par l'utilisateur dans la ligne de commande et de les récupérer dans des variables, utiles au programme principal.

Ensuite nous calculons la matrice Google et les poids des pages :

```
if estMatricePleine then
   -- Calcul avec la matrice pleine
   Matrice_pleineN.matrice_pleine(nbIteration,Taille_reseau, Alpha, To_String(nomFichier), pi);
```

Si l'utilisateur a demandé un calcul avec matrice pleine nous calculons les poids des pages avec la fonction matrice_pleine issue du module Matrice_pleineN qui est un module générique en paramètre de généricité le nombre de page N.

Le reste du programme ne fait plus appel à des fonctions ou à des modules car nous avons eu un problème à propos de l'appel d'un même type instancié par plusieurs modules. Nous développerons ce problème dans la partie V/).

La matrice creuse fait appel au module TH créé dans le mini-projet 2.

III/ Présentation des algorithmes implantées

Matrice pleine

Le premier principal algorithme est celui de la matrice pleine.

```
G := (1..N => (1..N => 0.0));
```

G est alors représenté comme une matrice de taille N * N (avec N la taille du réseau fourni).

```
-- Lecture du fichier entier
while not end_of_File(File) loop
    -- On récupère la page source et la page qu'elle pointe
    Get(File, PageSource);
    Get(File, PagePointee);
    -- Si le couple n'a pas de valeur associée (on évite les doublbons)
    if G(PageSource + 1, PagePointee + 1) = 0.0 then
        G(PageSource + 1, PagePointee + 1) := 1.0;
        -- On incrémente le nombre de pages pointées par la page courante
        Piref(PageSource + 1) := Piref(PageSource + 1) + 1;
    else
        null;
    end if;
end loop;
close(File);
```

Si une valeur pointe vers une autre et qu'elle n'est pas déjà présente nous mettons la valeur correspondante dans la matrice 1 (la valeur est choisie arbitrairement). Le vecteur Piref correspond au nombre de fois où une même page pointe vers une autre.

```
-- Calcul de la matrice Google
for i in 1..N loop
    for j in 1..N loop
        -- La page i ne référence aucune page
        if Piref(i) = 0 then
            G(i,j) := (1.0 / float(N)) * Alpha + CoeffGoogle;
        --La page i ne référence pas la page j
        elsif G(i,j) = 0.0 then
            G(i,j) := CoeffGoogle;
        --La page i référence la page j
        else
            G(i,j) := (1.0 / Float(Piref(i))) * Alpha + CoeffGoogle;
        end if;
    end loop;
end loop;
```

Nous remplissons la matrice G à l'aide de sa définition mathématique.

```
-- Calcul du vecteur PI
for k in 1..Nb_iteration loop
    PI_ancien := PI;
    for i in 1..N loop
        somme := 0.0;
        for j in 1..N loop
            somme := somme+PI_ancien(j)* G(j,i);
        end loop;
        PI(i) := somme;
    end loop;
end loop;
```

Nous effectuons Nb_iteration fois le calcul matriciel. La deuxième boucle "for" sur i correspond à l'itération sur les colonnes du vecteur PI et la troisième boucle "for" sur les j correspond à l'itération sur les lignes de la matrice.

Matrice creuse

Le second principal algorithme est celui de la matrice creuse.

```
type T_LCA is access T_Cellule;
type T_Cellule is record
    donnee: T_Donnee;
    cle: T_Cle;
    Suivant:T_LCA;
end record;
```

```
type T_TH is array (1 .. Capacite) of T_LCA;
```

G est définie comme un T_Th qui est un tableau de pointeur (voir mini projet 2).

Ici Capacite est le nombre de pages, T_donnee est un Float et T_Cle est une Unbounded_String.

```
-- Lecture du fichier entier
while not end_of_File(File) loop
    -- On récupère la page source et la page qu'elle pointe
    Get(File,PageSource);
    Get(File,PagePointee);

    -- On génère la clé en fonction de ces valeurs
    cle := trim(To_Unbounded_String(Integer'Image(PageSource+1)), Both) & '_' & tr

    -- On enregistre le couple clé-valeur (valeur provisoire égale à 1.0)
    -- Si la clé est déjà présente (doublons) la valeur est écrasée
    Enregistrer(G,cle,1.0);
end loop;
close(File);
```

Pour chaque paire de page, nous créons une clé du type PageSource+1 & "_" & PagePointee+1 (avec & pour signifier la concaténation des chaînes). La fonction de hachage va retourner l'entier PageSource+1. La fonction "enregistrer" issue du module T_Th va enregistrer la clé dans G[PageSource+1] et aura pour valeur 1,0. Si nous incrémentons le numéro de la page c'est car ce numéro peut être égal à 0, or en ADA, les tableaux sont définis de 1 à N.

Par exemple, si la page 1 pointe vers la page 2, nous aurons comme clé 2_3. Et cette clé sera enregistrée dans le G[2] avec comme valeur 1,0.

```
-- Calcul de la matrice Google
for k in 1..nbIteration loop
    -- On stocke le vecteur PI pour faire les calculs avec les valeurs au rang i-1 (et non avec les valeurs calculée
    PI_ancien := PI;
    -- Calcul matriciel
    for i in 1..N loop
        somme := 0.0;
        for j in 1..N loop
            -- La page j-1 référence au moins une page
            if TH_Integer_Float.th_lca.Taille(G(j)) /= 0 then
                -- On créé une clé
                cle_verif := trim(To_Unbounded_String(Integer'Image(j)), Both) & '_' & trim(To_Unbounded_String(Inte
                -- On regarde si la page j-1 référence la page i-1
                if Cle_Presente(G,cle_verif) then
                    valeur := (1.0 / float(TH_Integer_Float.th_lca.Taille(G(j)))) * Alpha + CoeffGoogle;
                else
                    valeur := CoeffGoogle;
                end if;
            -- La page j-1 ne référence aucune page
            else
                valeur := (1.0 / float(N)) * Alpha + CoeffGoogle;
            end if;
            somme := somme + PI_ancien(j) * valeur;
        end loop;
        PI(i) := somme;
    end loop;
end loop;
```

Le principe du calcul matriciel est le même qu'avec la matrice pleine mais nous devons ici calculer chaque fois la valeur du coefficient de la matrice.

Si la page j ne référence aucune page sa valeur est donnée par une formule.

Si la page j référence au moins une page, nous devons vérifier si elle référence la ième page ou non. Pour cela nous créons une nouvelle clé et nous regardons si elle est présente dans G. Si oui, la page j référence bien la page i sinon ce n'est pas le cas.

Nous affectons ensuite la formule correspondante selon les différents cas.

IV/ Étude des résultats

Matrice pleine

La matrice pleine nous permet de calculer le poids des pages issues des fichiers exemple-sujet.net et worm.net. En revanche, pour les fichiers brainlinks.net et Linux26.net il n'est pas possible d'effectuer le calcul à cause de leur trop grande taille. Lors de la création de la matrice, l'exception STORAGE_ERROR est levée car le programme tente de créer une matrice 10000*10000 (taille pour brainlinks.net) ce qui est évidemment trop lourd.

```
mhenry2@aragorn:~/1A/PIM/tp/projet/src/obj$ ./pagerank -P brainlinks.net

raised STORAGE_ERROR : stack overflow or erroneous memory access
```

Cependant, pour les fichiers peu volumineux, le programme est très performant :

- exemple-sujet.net : 0,02s (alpha = 0,9 et 150 itérations)
- worm.net : 0.17s (alpha = 0,9 et 150 itérations)

Matrice creuse

La matrice creuse a l'avantage de pouvoir calculer les poids des 4 fichiers fournis et n'est pas limitée par la taille des tableaux. Malheureusement, nous n'avons pas réussi à la rendre performante :

- exemple-sujet.net : 0,015s (alpha = 0,9 et 150 itérations)
- worm.net : 10,88s (alpha = 0,9 et 150 itérations)
- brainlinks.net : 5m21s (alpha = 0,9 et 1 itération) 28m43s (alpha = 0,9 et 5 itérations)
- Linux.net : 1253m34s = 20h53m34s (alpha = 0,9 et 1 itération)

Pour le calcul du réseau Linux.net, la connexion avec l'ordinateur de l'école a cessé de fonctionner pendant la nuit. Ainsi il est possible que le programme se soit arrêté à ce moment-là et qu'il ait repris seulement le lendemain matin. Le temps exact est donc incertain mais il reste très conséquent

La matrice creuse respecte la gestion de l'espace mémoire (vérifié avec valkyrie).

Les résultats obtenus correspondent aux résultats attendus, quel que soit le type de la matrice utilisée.

Pour conclure cette partie, la matrice pleine ne permet pas de calculer des réseaux immenses mais est assez performante tandis que la matrice creuse n'est pas limitée par la taille du réseau mais demande énormément de temps.

V/ Difficultés rencontrées et résultats retenus

Nous avons rencontré une énorme difficulté qui nous a bloqué un moment : l'utilisation de plusieurs modules pour construire notre programme principal.

Tout d'abord, en ADA, les tableaux (et matrices) doivent être déclarés en début de programme avec leur taille. Cependant, étant donné que cette taille est récupérée en lisant le fichier texte fourni, nous ne pouvions pas déclarer nos tableaux à ce moment-là. Nous avons alors utilisé un bloc declare qui permet de déclarer des variables au cours d'un programme et de les utiliser uniquement dans ce bloc.

Initialement, nous voulions séparer notre code en plusieurs modules afin d'avoir un code clair et structuré. Ainsi, pour le calcul de la matrice pleine, nous avons utilisé un package générique (avec la taille N de la matrice). Nous avons essayé de faire de même pour la partie de tri et d'écriture des résultats qui avaient également besoin de la taille de la matrice (et l'utilisation de ces tableaux). Le problème est que lorsque nous déclarions un tableau avec l'un des package, nous avions une erreur indiquant que le tableau reçu (pour les autres algorithmes) n'était pas du type attendu. Pourtant, il s'agissait du même type mais pas déclaré par le même package.

Nous avons alors essayé de créer un module générique tableau qui nous permettrait d'utiliser un tableau provenant d'un unique package. Cette fois-ci l'erreur était que le tableau fourni en argument de notre fonction ne provenait pas de la même instanciation du module que celui demandé par notre fonction. En effet, nous avions déclaré une instance d'un tableau dans le fichier de la fonction (resultats.adb par exemple) et une dans le pagerank.adb.

Après ce nouvel échec, nous avons donc décidé de nous concentrer sur le fonctionnement de notre programme et avons écrit nos fonctions directement dans le fichier principal.

Enfin, la dernière difficulté que nous avons rencontrée aura été le tri des tableaux. En effet, il fallait trier le tableau mais également conserver les indices d'origine associés à chaque poids. Il n'était donc pas possible de mettre en place les algorithmes de tri standard que nous connaissons. Pour remédier à ce problème, nous faisons une copie de notre vecteur poids, puis nous récupérons l'indice de la valeur maximale du vecteur et mettons sa valeur à 0 (afin d'avoir une nouvelle valeur maximale). Une fois tous les indices récupérés, nous écrivons directement les résultats en utilisant la liste des indices, ordonnés par le poids des pages.

VI/ Apport personnel du projet

Cyprien ARNOLD

Ce projet est mon premier vrai projet informatique et le plus complet. Cela m'a permis de constater la nécessité de vérifier son programme dès qu'une partie est finie, cela m'a aussi poussé à aller voir des ressources (notamment pour les Unbounded_String) sur Internet car le cours ne peut tout présenter. Je me suis aussi aperçu que ce qui prenait le plus de temps n'était pas de chercher et d'écrire l'algorithme du programme mais de le déboguer, activité pénible à mon goût.

Maxime HENRY

Pour ma part, ce n'était pas mon premier gros projet mais c'était le premier en ADA. Bien que ce langage soit très robuste et très utilisé dans les domaines sensibles, je préfère manipuler d'autres langages plus permissifs. Le problème concernant l'utilisation des tableaux dans plusieurs modules (partie V/) m'a énormément frustré et nous avons passé plusieurs heures pour essayer de trouver une solution.

Mis à part ce point concernant le langage ADA, ce sujet était très intéressant notamment sur la compréhension du fonctionnement de ce type d'algorithme. Je suis satisfait de notre implémentation de la matrice pleine mais pas celle de la matrice creuse car elle peut et doit être plus performante.

VII/ Conclusion

Nous avons ainsi réussi à créer deux solutions au problème du calcul de PageRank. Ces deux solutions ont des cas différents d'application : la matrice pleine utile pour les petits réseaux et la matrice creuse pour les plus grands (malgré sa relative lenteur). Si nous avions plus de temps nous implémenterions en plus la possibilité de choisir le nombre de chiffres après la virgule. Nous essaierions aussi d'optimiser notre calcul avec la matrice creuse notamment en essayant d'éviter de recalculer à chaque fois la variable "valeur".