2023/2024
PIM

EF02
Fallot Ysabel
Gutierrez Tom

PAGERANK

Résumé:

Ce rapport présente les résultats de notre travail sur plusieurs semaines, pour réaliser l'algorithme pagerank. Il contient toutes les informations sur le code réalisé pour implémenter l'algo de pagerank (architecture des programmes, choix des structures algos et détails de conception des programmes). Ensuite les résultats sont présentés sous forme d'un tableau pour pouvoir les comparer et les critiquer. Enfin, il décrit nos ressentis et notre manière de travailler pendant le projet en explicitant les difficultés que nous avons rencontrées, les qualités de nos raffinages, les améliorations possibles du projet et ce que nous avons appris lors de sa réalisation.

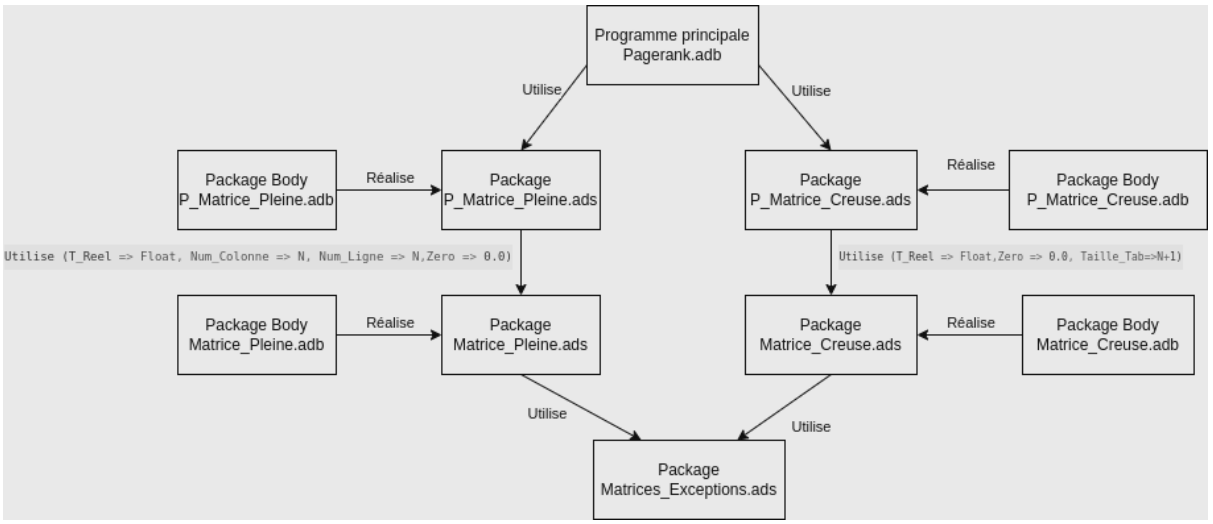
Introduction

Ce projet consiste à implémenter l'algorithme de PageRank. Il a pour objectif de classer les pages internet en fonction de leur poids. Le poids d'une page étant une manière de représenter sa popularité. Il est calculé à l'aide d'un algorithme appelé algorithme de Google, et permet de représenter quantitativement la popularité d'une page en utilisant le nombre de liens entre les différentes pages. Notre programme a aussi besoin d'une option matrice creuse qui permet de trier des matrices contenant beaucoup de valeurs en limitant les calculs et en ne stockant pas les 0. En sortie du programme, on obtient deux fichiers l'un contenant le classement de chaque page et l'autre les poids correspondants.

Plan du rapport

1- Architecture des programmes et sous programmes utilisée

Schéma d'architecture de Pagerank



2- Choix des structures algorithmiques utilisées

Dans le cas des matrices pleines, on veut stocker tous les coefficients des matrices. Il vient alors naturellement de les définir sous forme de tableau de tableau. Il est alors aussi nécessaire de stocker la taille de chaque tableau, qui correspond soit au nombre de lignes, soit au nombre de colonnes, afin de pouvoir parcourir la matrice.

On a alors abouti à la création du type suivant :

```
type T_Tab_Elem_Ligne is array (1..Num_Colonne) of T_Reel; -- Premier tableau représentant les éléments d'une ligne donnée (le numéro de colonne varie)
type T_Tab_Ligne is array (1..Num_Ligne) of T_Tab_Elem_Ligne ; -- Deuxième tableau dont les indices correspondent aux lignes de la matrice et
                                                               -- qui contient un tableau représentant tous les éléments de la ligne

type T_Matrice_Pleine is
        record
                Matrice : T_Tab_Ligne; -- Tableau de tableau représentant la matrice
                Nb_Ligne : Integer; -- Nombre de lignes de la matrice
                Nb_Colonne : Integer; -- Nombre de colonnes de la matrice
        end record;
```

Dans le cas des matrices creuses, on veut stocker uniquement les éléments pertinents de la matrice, c'est-à-dire les coefficients non nuls. Il est alors pertinent d'utiliser des structures de données creuses en utilisant des pointeurs afin de gérer à notre convenance la mémoire de l'ordinateur.

Deux choix d'implémentation se dessinaient donc, une seule liste contenant toutes les valeurs de la matrice avec comme clé d'accès sa ligne et sa colonne, ou en s'inspirant du modèle précédent de tableau de tableau, une première liste contenant des valeurs avec pour clé leur numéro de colonne, et une deuxième liste qui contient une liste comme définie précédemment et qui a pour clé le numéro de ligne.

Contrairement au tableau, pour avoir accès à l'élément d'une liste, il faut parcourir celle-ci (l'accès ne se fait pas en temps constant), c'est pour cela que la première implémentation sous forme d'une seule liste ne semblait pas intéressante car le temps d'exécution du programme serait sûrement trop long.
Il est toujours nécessaire de connaître la taille des matrices utilisées pour être sûr que la somme et le produit matriciel ne vont pas poser problème.

Toutes ces réflexions ont mené à la définition du type suivant :

```
type T_Cellule;
type T_Colonne;
type T_Liste_Colonne is access T_Cellule ; -- Premier liste contenant les éléments d'une colonne
type T_Ptr_Colonne is access T_Colonne; -- Deuxième liste contenant les pointeurs vers la tête de chaque colonne

type T_Cellule is
        record
                Ligne : Integer; -- Clé de la valeur dans la liste contenant les éléments d'une colonne
                Valeur : T_Reel; -- Valeur d'un coefficient
                Suivant : T_Liste_Colonne; -- Pointeur vers l'élément suivant de la même colonne
        end record;

type T_Colonne is
        record
                Num_Colonne: Integer; -- Clé de la valeur dans la liste contenant les pointeurs vers la tête de chaque colonne
                Colonne_Actuelle : T_Liste_Colonne; -- pointeur vers la tête de la liste représentant la colonne Num_Colonne
                Colonne_Suivante : T_Ptr_Colonne; -- Pointeur vers la tête de la liste représentant la colonne suivante
        end record;

type T_Matrice_Creuse is
        record
                Matrice_Creuse : T_Ptr_Colonne; -- Tableau de pointeurs représentant la matrice
                Nb_Ligne : Integer; -- Nombre de lignes de la matrice
                Nb_Colonne : Integer; -- Nombre de colonnes de la matrice
        end record;
```

3- Conception des modules et des programmes

Conception des matrices :

Module pleine :

La première réflexion était autour du choix de la structure de données puis des procédures et des fonctions nécessaires pour les définir dans le fichier matrice.ads. Afin de rendre le code de l'algorithme de Pagerank plus lisible, certaines fonctions sont aussi définies sous forme de procédures.

Ensuite, il a fallu implémenter les fonctions et procédures dans le .adb et définir les exceptions utilisées. A la suite de cela, il faut écrire un fichier test pour vérifier que les fonctions sont correctes.

Module creux :

La réflexion quant à la structure de données à adopter était plus approfondie car plusieurs choix étaient possibles (cf partie 2-). Les procédures et les fonctions sont les mêmes que celles utilisées précédemment : il a fallu adapter leur code à la nouvelle structure de données utilisée. On a donc repris les fonctions et procédures de matrice.adb une par une pour les modifier ou rajouter des fonctions intermédiaires notamment dans le cas de l'insertion (l'enregistrement) et la suppression (destruction des cellules et des listes) et on les a enregistré dans matrice_creuse.adb. Les exceptions sont communes à celles de matrice pleine donc le fichier de définition des exceptions à été utilisé tel quel.

Pour le fichier de test, de même que pour l'implémentation des fonctions, celui écrit pour les matrices pleines a été repris et adapté pour correspondre aux fonctions définies pour les matrices creuses et les fonctions propres aux matrices creuses ont été ajoutées.

Conception du PageRank :

Pagerank peut être décomposée en deux parties ,d'abord, le traitement des arguments on récupère les paramètres choisis par l'utilisateur, puis on appelle le module correspondant aux programme décidé par l'utilisateur.
Pour matrice pleine on crée d'abord, une matrice H,à partir des données que l'on extrait du graphe, puis une matrice S qui remplit les lignes vides de H, on utilise ensuite la formule de google pour créer la matrice G. Ensuite on recherche le poids de chaque page, en multipliant le vecteur poids Pi par G un certain nombre de fois K. On utilise ensuite ces valeurs de poids pour trier les pages,et l'on inscrit les données dans des fichiers correspondants.
Pour Matrice creuse le cas est plus complexe, on doit obtenir le même vecteur poids mais sans calculer G directement, et avec une matrice creuse. Le programme commence tout d'abord de la même manière en créant H. on y multiplie ensuite le terme alpha que l'on peut retrouver dans la formule de google, et on va ensuite découper le calcul en trois étapes, tout d'abord, on multiplie H par pi, puis on ajoute à ce nouveau vecteur pi les valeurs manquantes, tels que la valeur calculé du terme constant de G ou les lignes vides, qui se répètent à chaque calcul et qui permettent donc réduisent le temps d'exécution quand calculé à côtés. Après avoir obtenu le vecteur poids, on réalise le classement et on inscrit le résultat dans les fichiers de la même manière que pour matrice pleine.

4- Résultats obtenues et commentaires

temps d'exécution	sujet	worm	brainlinks	linux26
pleine	0,004s	0,052s	STORAGE_ERROR	STORAGE_ERROR
creuse	0,003s	0,012s	7m31,718s	116m39,425s

On remarque que matrice creuse est beaucoup plus efficace que matrice pleine avec sujet et worm, et que les matrices creuses permettent aussi de trier des matrices beaucoup plus importantes.

Valgrinds

```
tom@tom-IdeaPad-Slim-3-14IAH8:~/pr3$ valgrind ./pagerank -C -R worm exemples/worm/worm.net
==98367== Memcheck, a memory error detector
==98367== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==98367== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==98367== Command: ./pagerank -C -R worm exemples/worm/worm.net
==98367==
Matrice creuse
==98367==
==98367== HEAP SUMMARY:
==98367==     in use at exit: 0 bytes in 0 blocks
==98367==   total heap usage: 4,935 allocs, 4,935 frees, 101,920 bytes allocated
==98367==
==98367== All heap blocks were freed -- no leaks are possible
==98367==
==98367== For lists of detected and suppressed errors, rerun with: -s
==98367== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

-test robustesse 1 (mauvais nombre de sommet)

```
tom@tom-IdeaPad-Slim-3-14IAH8:~/pr3$ ./pagerank -C exemples/robustesse/rob_taille_matrice
Fichier invalide l'un des sommets est supérieur ou égale au nombre de sommet il n'appartient donc pas au graphe!!
```

-test robustesse 2 (mauvais format du fichier)

```
tom@tom-IdeaPad-Slim-3-14IAH8:~/pr3$ ./pagerank -C exemples/robustesse/rob_format_fichier
Mauvais format du fichier Le fichier doit être du format:
Entier(nombre de sommet)
Entier(sommet de départ)     Entier(sommet d'arrivée)
    .
    .
    .
Entier(sommet de départ)     Entier(sommet d'arrivée)
```

-test de robustesse ligne de commande

```
tom@tom-IdeaPad-Slim-3-14IAH8:~/pr3$ time ./pagerank -C -R brainlinks huihio  exemples/brainlinks/brainlinks.net
Erreur dans l'entrée des arguments!
huihio n'est pas un argument
Les arguments possibles sont:
- -A <valeur de alpha> avec <valeur de alpha> dans [0,1]
- -K <valeur> avec <valeur> l'indice k du vecteur poids, un entier positif
- -E <valeur> avec <valeur> l'indice se précision, un réel positif
- -R  <prefixe> avec <prefixe> le prefixe du fivhier d'entrée
- -P pour calculer avec matrice pleine
- -C pour calculer avec matrice creuse
```

On peut aussi voir ici que le programme est robuste.

5- Problèmes majeurs rencontrés

Dans le cas des matrices pleines

Deux problèmes majeurs ont été rencontrés lors de la conception, l'implémentation et l'utilisation du module des matrices pleines :

1- L' écriture du fichier de test des matrices.

En effet, les difficultés venaient du fait de réussir à écrire un programme de test qui vérifiait tous les cas pour chaque fonction : à la fois ceux d'un fonctionnement habituel dans un cas simple, ceux d'un fonctionnement habituel dans un cas complexe ou extrême et ceux dans le cas de fonctionnement inhabituel pour tester la robustesse du programme.

2- Problème de stockage/RAM dans le cas des matrices pleines

La création d'un tableau pour de grosses matrices prend énormément d'espaces dans la RAM. La matrice étant pleine, elle a beaucoup de 0 inutile car non utilisée dans le calcul du poids et cela prend donc énormément de place sans raison, de plus lorsqu'on atteint une certaine quantité de sommet dans le graphe, la mémoire allouée au programme n'est plus suffisante pour contenir la matrice. Matrice creuse mène donc à une erreur de RAM importante résolu par l'existence de matrice creuse.

Dans le cas des matrices creuses

Quatre problèmes majeurs ont été rencontrés lors de la conception, l'implémentation et l'utilisation du module des matrices creuses :

1- Gestion de la mémoire avec des "listes imbriquées".

Ainsi, lors de l'enregistrement d'une nouvelle valeur, si toutes les valeurs de sa colonne son nulle, il devient nécessaire de créer une nouvelle colonne en mémoire puis de créer la cellule qui contient la valeur dans la colonne nouvellement créée. De la même manière, lors de la suppression d'un élément, soit parce qu'il devient nul, soit lors de la suppression de la matrice, il faut également supprimer la colonne en mémoire si celle-ci est vide.
Pour cela, il a été nécessaire d'implémenter des fonctions intermédiaires (pour enregistrer et pour détruire) pour chaque type de liste afin de découper le problème en sous problème et d'aboutir à un programme fonctionnel.

2- Comment calculer la somme sans parcourir toute la matrice

Afin de réduire le temps d'exécution de l'algorithme, il n'est pas intéressant de parcourir toute la matrice. De plus, la structure de données choisie fait que seules les valeurs non nulles sont stockées. Ainsi, il devient pertinent de juste parcourir ces valeurs.

Pour cela, pour une somme de deux matrices A et B, on parcourt tous les coefficients non nuls de A qu'on additionne avec ceux correspondant dans B, puis on fait la même chose pour B pour être sûr que tous les coefficients non nuls de l'addition aient été pris en compte et aient été enregistrés dans la matrice résultat.

3- Comment calculer le produit sans parcourir toute la matrice

Comme précédemment, l'intérêt des matrices creuses est de pouvoir manipuler des matrices de grandes tailles, donc parcourir toutes les matrices pour en faire le produit n'est pas du tout optimale en termes de complexité temporelle.
Au vu du choix de représentation des matrices creuses (parcours des colonnes facilité), le plus pertinent est de parcourir toutes la matrice de droite (B) du produit matriciel, et pour chaque colonne de cette matrice, parcourir toutes les lignes de la matrice de gauche (A) pour multiplier les coefficients nons nuls de B avec les coefs correspondant de A. Ainsi toutes les valeurs nulles de B sont ignorées, ce qui ne change pas le résultat du produit et réduit le temps d'exécution de l'algorithme.

4 - Ecriture du fichier de test des matrices

Les difficultés rencontrées sont les mêmes que dans le cas des matrices pleines, sauf qu'en plus dans le cas des matrices creuses, il faut tenter d'anticiper ce que peuvent faire les pointeurs pour être sûr que le système soit robuste et fonctionnel.

Dans le cas de Pagerank

1-Optimiser le temps que le programme mets à tourner et calculer G sans calculer G

Cela a été un gros obstacle dans mon avancé, on avait au début un programme vraiment très lent, et je n'avais pas saisi tout la subtilité du calcul du poids pour une matrice creuse j'ai donc mis un temps assez important à trouver des moyens de réaliser tous les calculs de G en dehors du produit des matrices afin de rendre le programme plus rapide et à conserver les avantages de la matrice creuse.

2-Optimiser avec valgrind

L'optimisation avec valgrind à aussi été un gros challenge, le programme étant assez conséquent, l'oubli de la libération d'une partie de la mémoire peut être très problématique et augmenter exponentiellement avec la taille des graphes. Aussi faire attention à libérer les pointeurs aux bon moments avant de ne plus y avoir accès est très important et peut se révéler crucial pour garder une mémoire propre.

6- Grille d'évaluation du code et du raffinages

Evaluation des raffinages de FY (remplie par TG):

		Evaluation Etudiant (I/P/A/+)	**Justification / commentaire**	**Evaluation Enseignant (I/P/A/+)**
Forme (D-21)	Respect de la syntaxe Ri : Comment "... une action complexe ..." ? des actions combinées avec des structures de controle Rj : ...	+		
	Verbe à l'infinitif pour les actions complexes	+		
	Nom ou équivalent pour expressions complexes	+		
	Tous les Ri sont écrits contre la marge et espacés	+		
	Les flots de données sont définis	+		
	Une seule décision ou répétition par raffinage	+		
	Pas trop d'actions dans un raffinage (moins de 6)	+		
	Bonne présentation des structures de contrôle	+		
Fond (D21-D22)	Le vocabulaire est précis	+		
	Le raffinage d'une action décrit complètement cette action	+		
	Le raffinage d'une action ne décrit que cette action	+		
	Les flots de données sont cohérents	+		
	Pas de structure de contrôle déguisée	+		
	Qualité des actions complexes	+		

Evaluation des raffinages de TG (remplie par YF)

		Evaluation Etudiant (I/P/A/+)	**Justification / commentaire**	**Evaluation Enseignant (I/P/A/+)**
Forme (D-21)	Respect de la syntaxe Ri : Comment "... une action complexe ..." ? des actions combinées avec des structures de controle Rj : ...	+		
	Verbe à l'infinitif pour les actions complexes	+		

	Nom ou équivalent pour expressions complexes	+		
	Tous les Ri sont écrits contre la marge et espacés	+		
	Les flots de données sont définis	+		
	Une seule décision ou répétition par raffinage	+		
	Pas trop d'actions dans un raffinage (moins de 6)	+		
	Bonne présentation des structures de contrôle	+		
Fond (D21-D22)	Le vocabulaire est précis	+		
	Le raffinage d'une action décrit complètement cette action	+		
	Le raffinage d'une action ne décrit que cette action	+		
	Les flots de données sont cohérents	+		
	Pas de structure de contrôle déguisée	+		
	Qualité des actions complexes	+		

7- Tableau de répartition du travail

	spécifier	programmer	tester	relire
Pagerank	TG	TG	TG	YF
Matrice_Pleine	YF	YF	YF	TG
P_Matrice_Pleine	TG	TG	TG	YF
Test_Matrice_Pleine	YF	YF	YF	TG
Matrice_Creuse	YF	YF	YF	TG
P_Matrice_Creuse	TG	TG	TG	YF
Test_Matrice_Creuse	YF	YF	YF	TG
Matrice_Exceptions	YF	YF	YF	TG

8 - Conclusion

Ce projet a donc consisté à définir, implémenter puis utiliser les matrices pleines et les matrices creuses pour ensuite implémenter l'algorithme de Pagerank. Cet algorithme a été ensuite lancé avec des graphes de tailles variables sur les deux modules pour pouvoir comparer leurs performances. La précision des valeurs des coefficients des matrices et des calculs a commencé à être codée avec la définition des T_Reel ayant une précision quelconque, mais cette partie n’a malheureusement pas été terminée.

Afin de gagner du temps lors du parcours des listes, il serait pertinent d'enregistrer les colonnes et les valeurs par ordre croissant de clé (ordre croissant sur les numéros de colonnes pour les pointeurs vers les colonnes et ordre croissant de numéro de ligne pour les éléments des colonnes).

Enfin, une autre implémentation des matrices creuses aurait pu être possible : un tableau contenant des listes : chaque case du tableau représente les valeurs de la colonne de l'indice de la case. Cette implémentation serait plus gourmande pour la mémoire, mais la complexité temporelle de l'algorithme serait réduite car l'accès à une case du tableau se fait en temps constant.

9 - Annexes

YF : Ce projet m'a permis de m'améliorer sur la gestion des pointeurs, notamment avec les matrices creuses où il faut être capable de gérer des listes de listes. De plus, avec l'exemple linux26, j'ai vu l'importance d'avoir un code le plus optimisé possible pour réduire le temps d'exécution qui peut être très long. Enfin, bien qu'important, ce projet m'a montré que l'écriture de fichier de tests est fastidieuse et ennuyeuse.

TG : Ce projet à été très intéressant notamment pour maîtriser et ressentir le travail de groupe pour un projet informatique assez important, mais cela m'a aussi permis d'ancrer les connaissances que j'ai acquises pendant les cours de PIM notamment sur la maîtrise des exceptions. Cela m'a aussi permis de me rendre compte de l'impact que pouvait avoir certaines formes de rigueur comme la libération de pointeur pour le programme pouvant rendre un programme inutilisable si mal géré.

Raffinages:
https://docs.google.com/document/d/1anTZItkvXCZZSOvCMuXgcSacSP8-GC0SSRACt2xjPR0/edit?usp=sharing