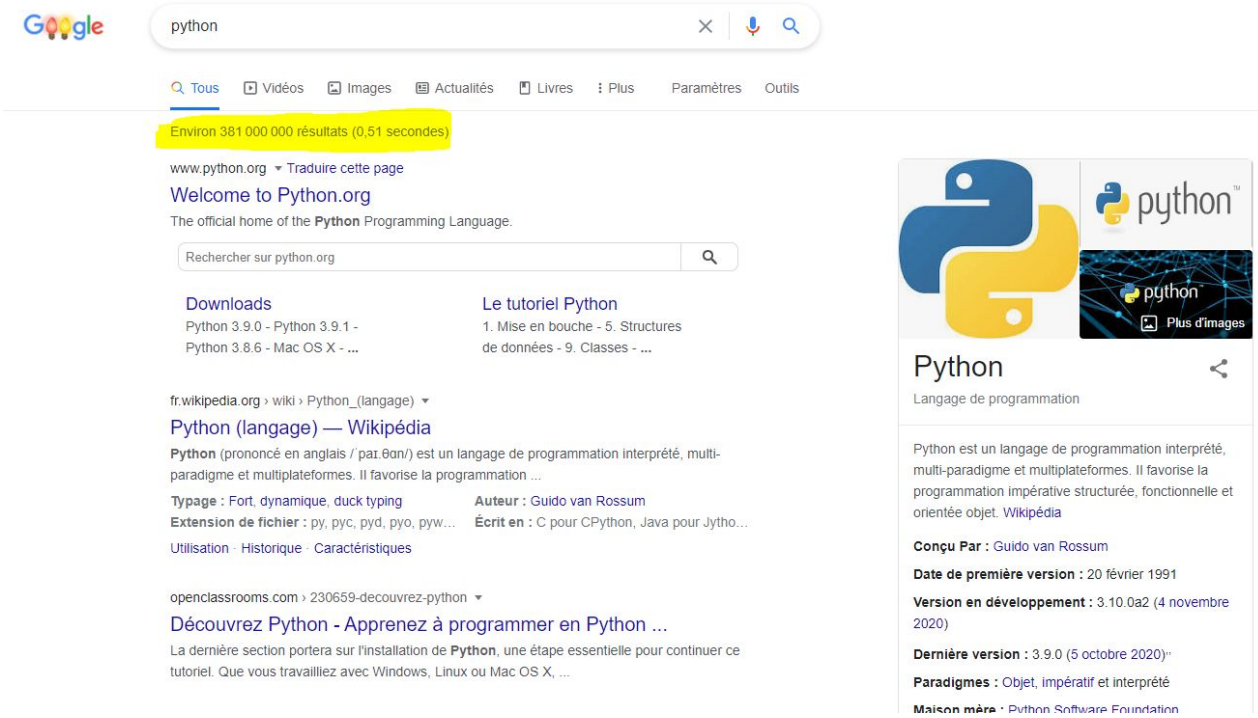
Département Sciences du Numérique

Rapport de projet en Programmation Impérative : Calcul efficace du PageRank (classement de noeuds dans un réseau quelconque)



Rapport remis le 16/01/2021

Encadré par Neeraj Kumar Sing, *Associate Professor INPT-ENSEEIHT/IRIT*

Réalisé par KIRUPANANTHAN Nadesan et HEURTEBISE Tom, *Elèves en première année à l'ENSEEIHT, filière Sciences du Numérique*

Résumé

Ce rapport synthétise un projet mené sur la création d'un algorithme de Pagerank permettant de classer la pertinence de nœuds dans un réseau donné. Par le biais de calculs matriciels successifs ainsi que d'un algorithme de tri, il nous a été possible d'attribuer à chaque nœud un poids symbolisant sa pertinence . Une fois ce calcul effectué il nous a suffit de classer ces nœuds par ordre décroissant de leur poids pour obtenir le nœud le plus pertinent en premier et ainsi de suite .

Afin d'optimiser notre algorithme en termes d'espace mémoire et de temps d'exécution, nous avons programmé deux versions différentes du calcul matriciel. L'une est plus rapide mais nécessite une occupation mémoire plus importante (utilisation de matrices pleines) tandis que l'autre et bien plus économe en termes de place mais déploie des temps d'exécution importants.

Enfin un dernier volet de ce projet que vous retrouverez abordé dans ce rapport est la précision des calculs effectués. Nous avons tenté de maintenir des résultats cohérents selon que les calculs sont effectués sur 3,6 ou 10 digits. A propos de ce point, la lecture du rapport vous apprendra que nous n'avons pas réussi à prendre en compte de façon satisfaisante cette précision sur des réseaux de grande ampleur (voir la partie traitant du réseau brainlinks).

Sommaire

I - Introduction

Partons d'un constat simple, peu importe la recherche internet effectuée, des milliers voire des millions de résultats sont toujours associés à cette dernière. Ainsi, ce sont potentiellement un nombre de résultats pratiquement illimité qui peut s'afficher à l'écran de l'utilisateur. La question qui peut nous venir est alors "comment ces très nombreux résultats peuvent être classés pour que le moteur de recherche propose une mise en forme pertinente ?".

C'est ce que nous avons étudié durant notre projet. Cependant, le but de ce rapport n'est pas de présenter toutes les méthodes permettant de répondre à cette question mais plutôt d'en étudier une qui a été employée par Google jusqu'à récemment et qui s'appuie sur un algorithme de PageRank.

Ce rapport s'articule en plusieurs parties distinctes : un rappel succinct du cahier des charges, des éléments de conception de l'application pagerank, une explication sur les principaux algorithmes implémentés, les résultats obtenus sur les fichiers de tests fournis et enfin un bilan sur les difficultés rencontrées et les solutions adoptées lors de ce projet.

II- Objectifs du projet

Ce projet avait pour but de créer un programme qui classe les nœuds d'un réseau donné par ordre décroissant de pertinence. Cet ordre est basé sur les liens des pages entre elles et sur la qualité de chaque page (qui décroît si une page référence trop d'autres pages).

Nous avons procédé en implémentant un algorithme qui calcule le poids de chaque page en fonction du poids des autres par le biais d'une formule itérative fournie dans le sujet :

$$r_{k+1}(P_i) = \sum_{P_j \in \mathcal{P}_i^{IN}} \frac{r_k(P_j)}{|P_j|}$$

(Pj est un noeud, r_{k+1}(Pi) le poids du noeud Pi à l'itération k, Pi^{IN} est l'ensemble des noeuds référençant Pi et |Pj| le nombre d'hyperliens de Pj)

Dans le cahier des charges, deux versions étaient exigées, une utilisant des matrices pleines (donc une mémoire d'exécution importante en cas de grands réseaux) et une autre version utilisant des matrices creuses (libérant ainsi de la mémoire mais nécessitant plus de temps de calcul).

III- Conception de l'application pagerank

1) Méthode de conception

Nous avons construit l'application pagerank en nous basant sur les raffinages. De ces derniers nous avons dégagé trois axes différents.

Premièrement, la récupération des arguments se doit d'être robuste et doit assurer une précision suffisante à alpha. Ensuite nous avons identifié un TAD (Type Abstrait de Données) permettant de représenter des matrices (creuses ou pleines) et de les manipuler. Enfin nous avons réalisé un dernier module relatif au tri.

2) Architecture du projet

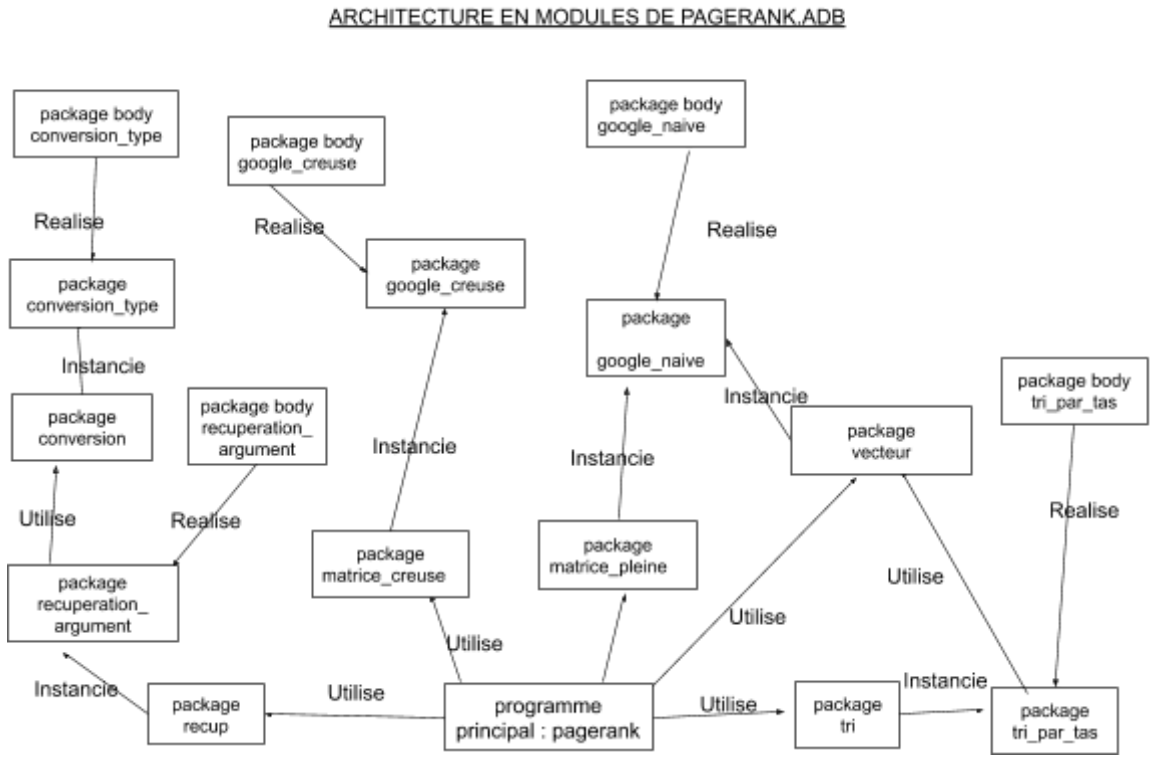


Figure 1- Architecture en modules du projet

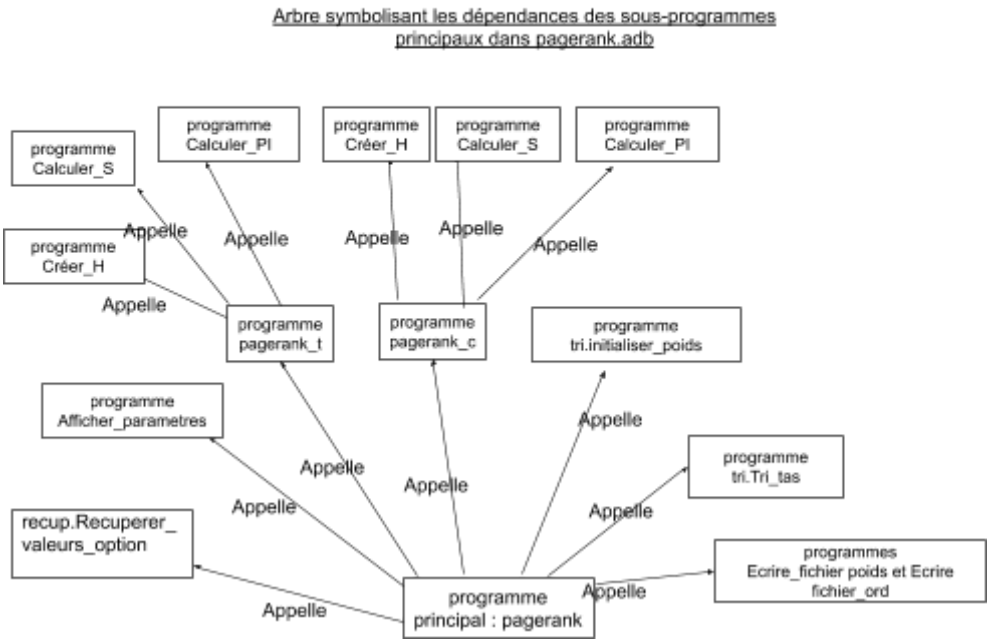


Figure 2- Architecture des sous programmes pour l'application pagerank

3) Principaux choix réalisés

Cette partie va nous permettre de lister certains choix effectués vis à vis de l'architecture du projet. Concernant le style de programmation, nous avons adopté un style défensif sur l'ensemble du projet mais, après réflexion (et puisque celui-ci n'a pas vocation à être utilisé par quelqu'un d'autre), nous aurions pu nous contenter d'une programmation par contrats.

a) Modules matriciels

Vis à vis du module Matrice_creuse, le choix d'implantation de ce type a été assez difficile puisque nous avons tour à tour comparé les performances de 2 implantations différentes : LCA (Liste Chainée Associative) de LCA (cf remarque) et TH (Table de Hachage) constituée de LCA. Afin d'optimiser le temps d'accès aux données nous avons opté pour une TH. Ce temps pourrait sûrement être amélioré par l'utilisation d'un ABR (Arbre Binaire de Recherche).

NB : Vous trouverez dans les livrables un module Google_Creuse_Lente qui implante des matrices entièrement creuse (LCA de LCA)

b) Module récupération des arguments

Pour ce module et cette récupération des arguments nous avons effectué plusieurs choix. Tout d'abord au niveau de la robustesse, sur le modèle des fonctions sur Unix nous avons choisi de lever une exception pour chaque cas d'erreur et donc de créer autant d'exceptions que de cas d'erreur. A ceci nous avons ajouté un message d'erreur générique précisant la syntaxe attendue pour l'exécution de pagerank.

c) Module de tri

Nous avons soigneusement comparé les types de tri existants pour implémenter la solution prenant le moins de mémoire possible et s'exécutant le plus rapidement possible. En raison du nombre potentiellement grand de données nous avons choisi le tri par tas de complexité spatiale théorique O(1) et de complexité temporelle moyenne nlog (n), ce qui est la meilleure combinaison possible si on compare aux autres types de tri.

IV- Principaux algorithmes

1) Modules matriciels

Les principaux algorithmes dans les modules matriciels sont l'application des formules mathématiques pour la somme ou le produit des matrices. Cela se réalise facilement pardes boucles for imbriquées les unes dans les autres dans le cas des matrices pleines et par des appels récursifs dans le cas de matrices creuses.

Ces modules sont organisés de façon à initialiser les matrices avec certaines dimensions, puis les manipuler. Pour les manipuler, il faut parcourir chaque ligne et colonne de chaque matrice pour accéder aux coefficients et ainsi réaliser l'opération mathématique désirée.

2) Tri par tas

Ce module de tri par tas s'appuie sur un tri par arbre dans lequel "l'arbre" est représenté par un tableau dans lequel chaque nœud d'indice *n* a deux fils qui sont les cases 2^n et 2^{n+1}. Nous avons fait cette implémentation plutôt que l'ABR effectué en TD car le projet initial était de faire un tri en place directement sur le vecteur, ce qui s'est avéré au final impossible.

Ce module est organisé de la façon suivante. Un premier sous-programme permet de passer du vecteur PI à trier à un vecteur poids constitué des éléments (indice d'origine : poids). Ensuite vient le programme de tri par tas qui crée le tas à partir du tableau et extraie la racine (qui est l'élément le plus petit du tableau par construction).

3) Récupération d'arguments

Ce module de récupération d'arguments consiste en l'analyse de la ligne de commande et en la récupération des options y figurant. Pour réaliser cela, notre programme vérifie la validité de la ligne de commande (options mal rentrées, oubliées, etc..) ainsi que la valeur des options (l entier supérieur à 1, alpha réel compris entre 0 et 1 et existence du fichier rentré).

V- Résultats obtenus

1) Matrice Pleine

Nom du test	Temps d'exécution moyen	Procédure/fonction prenant le plus de temps	Résultats conformes ? Sinon, combien de digits identiques ?
exemple_sujet.net			
précision 3	0.00 s	get_coefficient	OUI
précision 6	0.00 s	idem	OUI
précision 10	0.00 s	idem	OUI
worm.net			

précision 3	0.02 s	idem	NON(4)
précision 6	0.11 s	idem	OUI
précision 10	0.09 s	matrice_pleine_get_coeff	NON (0)
brainlinks.net			
précision 3	244.29 s	idem	NON(0)
précision 6	247.09 s	idem	NON(0)
précision 10	530.53 s	idem	NOn(0)
linux.net			
précision 3*	*Estimé > 20h*	/	

Figure 3- Résultats obtenus avec la version naïve

*trop long pour être calculé nous avons fait 250*temps pour brainlinks (car taille linux = 250 * taille brainlinks)

2) Matrice Creuse

Nom du test	Temps d'exécution moyen	Procédure/fonction prenant le plus de temps	Résultats conformes ? Sinon, combien de digits identiques ?
exemple_sujet.net			
précision 3	0.00 s	matrice_creuse__get_coefficient	Oui
précision 6	0.00 s	idem	oui
précision 10	0.00 s	idem	oui
worm.net			
précision 3	0.48 s	idem	Non (4)
précision 6	0.46 s	idem	Oui
précision 10	0.49 s	idem	Non (0)
brainlinks.net			

précision 3	2880.44s (~48min)	idem	Non (0)
précision 6	2811.61s (~46 min)	idem	Non(0)
précision 10	2770.23 (~46min)	idem	Non (0)
linux.net			
précision 3 *	*Estimé à 200h*	/	/

Figure 4- Résultats obtenus avec la version creuse

3) Conclusion de l'étude

En termes de précisions des résultats et de qualité, comme attendu il n'y aucune différence entre la version creuse et pleine et la précision des calculs ne semblent pas impacter de manière croissante les temps d'exécution.

Sur des réseaux de taille importante, le temps pour la version creuse explose comparé à celui pour la version pleine. Cela était attendu puisque d'une part le calcul PI*fois intègre beaucoup de tests conditionnels ralentissant énormément l'algorithme et d'autre part l'utilisation de LCA entraîne des temps d'exécution importants. Toutefois, l'avantage de la version creuse est qu'elle est très économe en termes d'occupation mémoire. En guise de conclusion nous dirons que l'emploi d'une version ou d'une autre est à adapter au contexte, aux contraintes et aux ressources disponibles (Stack notamment).

VII- Difficultés rencontrées et solutions adoptées

Nous avons eu beaucoup de difficultés vis à vis de la précision des calculs. En effet, les résultats que nous obtenons sur des réseaux de grandes tailles sont faux. Nous avons cherché l'origine du problème en dégageant plusieurs causes possibles : un éventuel problème d'arrondi effectué par ADA sur les 11 ou 12 digits qui se répercutent au fil des calculs ou alors une erreur de raisonnement sur un des algorithmes de calcul.

Nous avons aussi noté un dysfonctionnement de valgrind dont vous retrouverez le détail ci-dessous :

Nous avons fixé la taille des matrices à 10 000 x 10 000 pour exécuter la version pleine. La stack size par défaut étant insuffisante, il faut l'augmenter (et le programme fonctionne alors sans souci)

```
theurteb@kenobi:~/Annee_1/PIM/Projet/IJ-03/src$ ulimit -s unlimited
theurteb@kenobi:~/Annee_1/PIM/Projet/IJ-03/src$ ./pagerank exemple_sujet.net
alpha :8.5000002384E-01
iteration : 150
mode creux
nom du fichier : exemple_sujet.net
theurteb@kenobi:~/Annee_1/PIM/Projet/IJ-03/src$ valgrind ./pagerank -P exemple_sujet.net
==22109== Memcheck, a memory error detector
==22109== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==22109== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==22109== Command: ./pagerank -P exemple_sujet.net
==22109==
alpha :8.5000002384E-01
iteration : 150
mode naif
nom du fichier : exemple_sujet.net
==22109== Warning: client switching stacks?  SP change: 0x1ffefe32d0 --> 0x1fe726ae60
==22109==          to suppress, use: --max-stackframe=400000112 or greater
==22109== Invalid write of size 4
==22109==    at 0x11110F: pagerank__pagerank_t.5742 (pagerank.adb:86)
==22109==  Address 0x1fe726ae7c is on thread 1's stack
==22109==
==22109== Warning: client switching stacks?  SP change: 0x53bd6b0 --> 0x1ffefe32d8
==22109==          to suppress, use: --max-stackframe=137334250536 or greater

raised STORAGE_ERROR : stack overflow or erroneous memory access
==22109==
==22109== HEAP SUMMARY:
==22109==     in use at exit: 704 bytes in 1 blocks
==22109==   total heap usage: 3 allocs, 2 frees, 784 bytes allocated
==22109==
==22109== LEAK SUMMARY:
==22109==    definitely lost: 0 bytes in 0 blocks
==22109==    indirectly lost: 0 bytes in 0 blocks
==22109==      possibly lost: 0 bytes in 0 blocks
==22109==    still reachable: 704 bytes in 1 blocks
==22109==         suppressed: 0 bytes in 0 blocks
==22109== Rerun with --leak-check=full to see details of leaked memory
==22109==
==22109== For counts of detected and suppressed errors, rerun with: -v
==22109== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Cependant, en utilisant Valgrind il semble que l'augmentation de la stack size ne soit pas prise en compte et que celui-ci lève une erreur qui ne devrait pas l'être (l'exception est d'ailleurs considérée par Valgrind comme une fuite de mémoire)

Figure 5- Capture d'écran d'un problème identifié avec Valgrind

VIII— Conclusion

Pour conclure, nous avons traité de nombreux aspects du cahier des charges dont : la récupération robuste des arguments, l'implémentation d'une version naïve des matrices et d'une version creuse.

Cependant il est à noter que les résultats que nous obtenons sur des réseaux sont faussés par un problème de précision dont nous n'avons pas réussi à identifier la cause exacte.

XIX— Bilans personnels

1) Tom

Ce projet a été pour moi l'occasion de vérifier que j'avais bien compris tous les concepts présentés pendant le cours de PIM. Après avoir terminé le projet et bien qu'il ne soit pas totalement abouti, il me semble que je dispose bien des bases en programmation impérative (et en langage ADA).

De plus, j'ai beaucoup appris sur la manière dont on peut déboguer un programme et chercher des informations sur un langage méconnu du grand public.

Enfin je dirais que j'ai eu plaisir à travailler en binôme même si nous avons eu quelques fois des difficultés pour communiquer sur certaines idées complexes (nous avons souvent utilisé le partage d'écran ainsi que Paint.net). Je pense que contrairement à la plupart des groupes nous avons beaucoup travaillé à deux en partage d'écrans sur ma propre session et cela a été très enrichissant du point de vue de la réflexion.

2) Nadesan

Durant ce projet, j'ai pu me mettre à jour sur les concepts vus en cours de PIM et comprendre les différentes notions que je n'avais pas saisies en TP. J'ai également pu constater mon niveau sur le langage ADA.

Comme mon binôme, j'ai également appris comment déboguer et chercher des informations sur un langage peu répandu.

Je suis également chanceux d'avoir pu travailler avec ce binôme car il m'a aidé à comprendre énormément de concepts que je n'avais pas saisi en ADA et PIM. Le fait qu'on est travaillé en partage d'écran nous a permis de confronter nos points de vue et j'ai pu ainsi voir une façon différente de réfléchir en programmant.