



Programmation impérative

---

## Rapport de projet : PageRank

---

Groupe EF03

*Élèves :*

THEVENET Louis  
MORISSEAU Albin

*Enseignant :*

Nom Prénom

Décembre 2023

# Table des matières

1. Introduction .....	3
1.1. Paramètres de la ligne de commande .....	3
1.2. Valeurs par défaut des paramètres .....	4
1.3. Spécification des paramètres .....	4
2. Architecture de l'application .....	4
3. Choix techniques / Structure de données .....	4
3.1. Cas Matrices Pleines .....	4
3.2. Cas Matrices Creuses .....	5
3.3. Paramètre de généricité .....	5
3.4. Choix du tri .....	5
4. Performances des implantations .....	5
5. Difficultés rencontrées et solutions apportées .....	6
5.1. Représentation par matrices creuses .....	6
6. Répartition des tâches .....	6
7. Conclusion et avancement .....	6
7.1. Mise à jour le mercredi 17 Janvier .....	6
8. Annexes : .....	7
8.1. Apport personnel du projet .....	7

# 1. Introduction

Chaque jour, plus de 7 milliards de recherches sont effectuées via des moteurs de recherche soit presque 81000 par seconde. Chaque recherche peut renvoyer des millions de résultats. Ainsi, il est essentiel pour faciliter la vie des utilisateurs et minimiser les coûts, de trier ces différents résultats de la manière la plus pertinente et efficace possible.

Ce projet consiste à implanter l'algorithme PageRank qui mesure la popularité de pages Internet en utilisant les référencements qui mènent d'une page à une autre. La formule suivante traduit le mécanisme récursif du calcul du poids ( en terme de référencement) de chaque page:

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

vec :

- $P_i$  : une page internet représentée par un sommet du graphe
- $r_{k(P_j)}$  : le poids de la page  $j$  à l'itération  $k$
- $|P_j|$  : le nombre d'hyperliens de la page  $P_j$
- $P_i^{\text{IN}}$  : l'ensemble des pages qui référencent  $P_i$
- $r_{k+1}(P_i)$  : le poids de la page  $P_i$  à l'itération  $k + 1$

Le calcul est initialisé avec pour chaque sommet  $r_0(P_i) = \frac{1}{N}$  où  $N$  est le nombre de pages web à classer. On arrête le calcul lorsque les valeurs successives de deux poids est suffisamment proche, i.e  $r_{k+1}(P_i) - r_k(P_i) < \varepsilon$ , avec  $\varepsilon > 0$ . On aura alors convergé vers la valeur  $r(P_i)$  du poids de la page. En la pratique, le calcul de tous les poids des pages webs au rang  $k + 1$  est effectué par une multiplication matricielle :

$$\pi_{k+1}^T = \pi_k^T \times G$$

$$\pi_0^T = \left( \frac{1}{N}, \frac{1}{N}, \dots, \frac{1}{N} \right) \in \mathbb{R}^N$$

Avec :

- $\pi_k^T = \left( r(P_0)_k, \dots, r(P_i)_k, \dots, \frac{1}{r(P_{N-1})_k} \right)$  : est un vecteur ligne dont la composante  $i$  est le poids de la page  $P_i$  au rang  $k$ .
- $G$  : est la matrice de Google, définie comme suit:

$$G = \alpha \cdot S + \frac{1 - \alpha}{N} * e \times e^T$$

où  $e \cdot e^T$  est une matrice carrée de taille  $N$  remplie de un et la matrice  $S$  est une matrice d'adjacence pondérée qui comporte sur chaque ligne  $i$  la valeur  $1/|P_i|$ , s'il existe au moins un lien sortant de la page  $P_i$  vers la page  $P_j$ , ou  $1/N$  sur toute la ligne si le sommet  $P_i$  n'a aucun lien sortant.

On pourra choisir les différents paramètres de l'algorithme ainsi que l'implantation choisie lors de la saisie de la commande d'appel.

## 1.1. Paramètres de la ligne de commande

- A** Le *Dumping Factor* ( $\alpha$ ), est un paramètre de pondération, plus sa valeur approche 1, plus la convergence est lente
- K** Le nombre d'itération ( $k$ ) de l'algorithme à réaliser
- E** La précision ( $\varepsilon$ ) permet d'interrompre l'algorithme lorsque la variation du vecteur poids d'une itération à l'autre lui est inférieure
- R** Le préfixe utilisé pour les fichiers de sortie (<prefixe>.pr et <prefixe>.prw)
- C** Permet de choisir la version matrices creuses
- P** Permet de choisir la version matrices pleines

## 1.2. Valeurs par défaut des paramètres

Les paramètres du programme lorsque non spécifiés ont pour valeurs :

$$\alpha := 0.85$$

$$k := 150$$

$$\varepsilon := 0.0$$

## 1.3. Spécification des paramètres

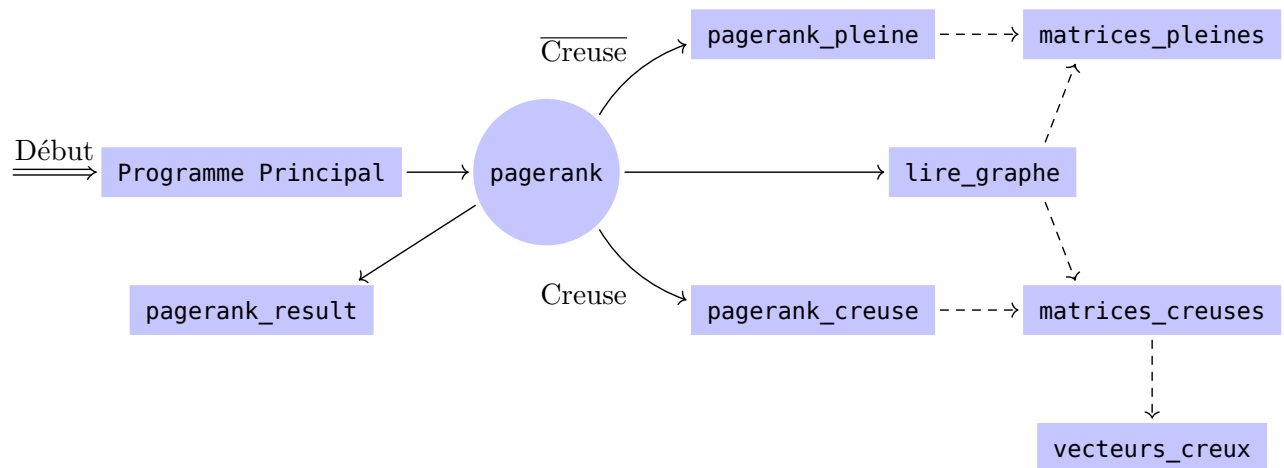
Les paramètres doivent respecter certaines conditions pour que le programme exécute l'algorithme :

$$\alpha \in [0, 1] \varepsilon \geq 0$$

Un seul algorithme choisi (Creuse ou Pleine)

Préfixe non vide

## 2. Architecture de l'application



Légende :

$\Rightarrow$  point d'entrée

$\rightarrow$  appelle

$\dashrightarrow$  utilise

## 3. Choix techniques / Structure de données

### 3.1. Cas Matrices Pleines

```

1 type T_Matrice_Pleine is array(1..N, 1..P) of Long_Float;
2

```

```

3  type T_Matrice is record
4      Mat : T_Matrice_Pleine;
5      Lignes : Integer;
6      Colonnes : Integer;
7  end record;

```

### 3.2. Cas Matrices Creuses

```

1  -- matrices_creuses.adb
2  type T_Matrice is array(1..Taille) of T_Vecteur_Creux;
3  type T_Facteurs is array(1..Taille) of Long_Float;
4
5  -- vecteurs_creux.adb
6  type T_Cellule;
7  type T_Vecteur_Creux is access T_Cellule;
8
9  type T_Cellule is
10     record
11         Indice : Integer;
12         Valeur : Long_Float;
13         Suivant : T_Vecteur_Creux;
14     end record;

```

### 3.3. Paramètre de généricité

#### 3.3.1. Pour Matrice pleine

```

1  -- matrices_pleines.ads
2  generic
3      N : Integer; -- Nombre de lignes
4      P : Integer; -- Nombre de Colonnes

```

#### 3.3.2. Pour Matrice creuse

```

1  -- matrices_creuses.ads
2  generic
3      Taille: Integer; -- Nombre de colonnes ( vecteurs creux)

```

### 3.4. Choix du tri

Nous avons choisi de trier notre vecteur résultat avec un tri par insertion. Ce n'est certes pas le plus efficace en terme de complexité mais cela était plus simple à coder. Cela fera partie des micro optimisation à réaliser plus tard.

## 4. Performances des implantations

Fichier graphe Test	Implant. Pleine (s)	Implant. Creuse (s)
sujet.net	0.0006219	0.0006138
worms.net	0.0308927	0.0020846
brainlinks.net	109.855867000	5.2631
linux26.net	Stack overflow	54.170379000

On observe que l'implantation en matrice creuse est plus efficace en terme de complexité de calcul que celle en matrice pleine et ce d'autant plus si le graphe possède de nombreux sommets comme cela est le cas en conditions réelles. En effet, en prenant en compte l'aspect creuse de la matrice, on se débarrasse de calculs inutiles.

## 5. Difficultés rencontrées et solutions apportées

### 5.1. Représentation par matrices creuses

Nous avons initialement choisi de réaliser un vecteurs creux de vecteurs creux mais les performances n'étaient pas très bonnes alors que l'implémentation était compliquée.

Nous avons ensuite choisi d'utiliser un tableau de vecteurs creux afin que l'opération d'accès aux têtes de colonnes se fasse en temps constant (les vecteurs représentent les colonnes).

## 6. Répartition des tâches

Modules	Spécifier	Programmer	Tester	Relire
programme_principal	A	A	A	L
lire_graphe	A	A	A	L
pagerank	A	A	A	L
pagerank_creuse	L	L	L	A
pagerank_pleine	L	L	L	A
matrices_creuses	L	L	L	A
matrices_pleines	L	L	L	A
vecteurs_creux	A	A	A	L
pagerank_result	A	A	A	L

## 7. Conclusion et avancement

Nous avons réussi à implanter de manière robuste les deux versions de l'algorithme.

Nous avons pour perspective d'améliorer la précision de nos calculs de poids en introduisant un paramètre générique de précision dans nos différents programmes.

Nous aurions également voulu tester une autre idée si le temps nous l'avait permis. Dans la version matrices creuses, avec l'utilisation du vecteur **Facteurs**, nous avons remarqué qu'il devenait inutile de stocker des valeurs dans la matrice : il suffit de *marquer* les cases. Ainsi, il serait possible d'utiliser une matrice de booléens. On stockerait alors seulement le maillon et son indice (8 bits) au lieu de 64 bits pour des **Long\_Float**. Cela permettrait de réduire la taille de la matrice de 8 fois. On pourrait également utiliser des vecteurs de bits et diviser la taille utilisée en mémoire vive par 64

### 7.1. Mise à jour le mercredi 17 Janvier

Nous avons finalement implémenté notre idée.

Nous avons ainsi défini  $\tilde{H} \in \mathcal{M}_n(\{0, 1\})$  et  $F \in \mathbb{R}^n$  tels que

$$\forall (i, j) \in \llbracket 1, n \rrbracket^2, \tilde{H}_{i,j} \in \{0, 1\}$$

$$\forall i \in \llbracket 1, n \rrbracket, F_i = \max \left( \sum_{k=1}^n \mathbb{1}_{\tilde{H}_{i,k}=1}, \frac{1}{n} \right)$$

Lors des itérations, il suffira de vérifier que le maillon  $\tilde{H}_{i,j}$  existe (valeur  $\neq 0$ ) pour savoir que la valeur à utiliser est  $\frac{1}{F(i)}$ .

## 8. Annexes :

Raffinages complets, Grilles d'auto évaluation : Raffinages projet Pagerank groupe EF03

### 8.1. Apport personnel du projet

#### Albin Morisseau

Ce projet, m'a permis de mieux appréhender la mise en place d'un projet plus conséquent, notamment l'importance de l'architecture des différents modules lors de sa conception. J'ai aussi appris à travailler en groupe et à coopérer malgré les méthodes et outils de travail différents. Ce projet a été enrichissant car j'ai beaucoup appris en travaillant avec Louis.

#### Louis Thevenet

Dans ce projet, j'ai pu m'améliorer sur la gestion de différents modules et leurs liens. Il m'a aussi permis de travailler en équipe et de mieux comprendre les enjeux de la communication et de la répartition des tâches.