Fermeture transitive d'un graphe orienté
Obtention du graphe minimal équivalent

Sommaire

Introduction

La plupart des projets sont découpés en tâches élémentaires et celle-ci sont confiés à un ou plusieurs réalisateurs. Une fois le découpage réalisé, il se pose le problème de les ordonnancer car ces tâches sont soumises à des contraintes diverses. Pour faciliter cette tâche, on peut représenter le tout par un graphe d'ordonnancement qui nous permettra de classer les tâches avec leurs relations d'antériorité.

Pour qu'un graphe d'ordonnancement soit complet et cohérent, nous allons devoir retirer les redondances, c'est l'objet de notre étude.

Premièrement, nous définirons les termes de graphes orientés et fermeture transitive. Ensuite, nous analyserons les solutions existantes et leur complexité temporelle et spatiale.
Nous poursuivrons sur un récapitulatif et un comparatif de ces différents algorithmes avant de finir en détaillant les choix techniques et la mise en œuvre de ces algorithmes au sein de notre application.

Théorie

Problème et solutions proposées

En ordonnancement, il est intéressant de supprimer les transitivités, dans le graphe représentant les prédécesseurs et successeurs de chaque tâche. Le graphe équivalent dont on a retiré toutes les transitivités, s'appelle le graphe minimal équivalent.
Une solution se base sur la fermeture transitive du graphe et retire pas à pas chaque transitivité trouvée lors de la construction de la fermeture.
Une seconde solution consiste à parcourir les prédécesseurs de chaque sommet et de supprimer chaque transitivité trouvée.
Nous allons détailler et mettre en pratique ces algorithmes de manière théorique puis sur des exemples à travers une application que nous avons développé.

Descriptions des solutions

Par fermeture transitive

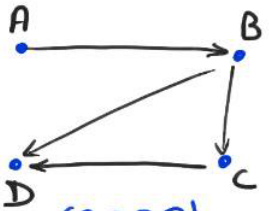
Pour obtenir la fermeture transitive, il suffit de suivre la formule suivante :

- $M^* = M \oplus M^2 \oplus \dots \oplus M^N$

(n) correspond à la puissance maximum au-delà de laquelle la matrice obtenue est invariante.
Une fois toutes les étapes pour obtenir la fermeture transitive effectuée, on crée le graphe minimal équivalent.
Pour cela, on part de notre matrice initiale et on retire les chemins de longueur supérieur à 1.

Exemple

Soit une matrice $M = \begin{pmatrix} 0&1&0&0 \\ 0&0&1&1 \\ 0&0&0&1 \\ 0&0&0&0 \end{pmatrix}$



$M^2 = \begin{pmatrix} 0&0&1&1 \\ 0&0&0&1 \\ 0&0&0&0 \\ 0&0&0&0 \end{pmatrix}$ $M^3 = \begin{pmatrix} 0&0&0&1 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \end{pmatrix}$ $M^4 = \begin{pmatrix} 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \end{pmatrix}$

Fermeture transitive : $M \oplus M^2 \oplus M^3 \oplus M^4$

$\Rightarrow \begin{pmatrix} 0&1&1&1 \\ 0&0&1&1 \\ 0&0&0&1 \\ 0&0&0&0 \end{pmatrix}$ 2 transitivités ajoutées
1 transitivité existante

Graphe minimal équivalent

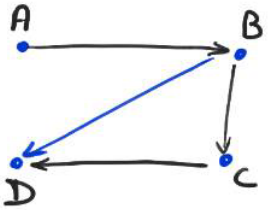
1) Comparaison $M - M^2$

$\Rightarrow \begin{pmatrix} 0&1&0&0 \\ 0&0&1&1 \\ 0&0&0&1 \\ 0&0&0&0 \end{pmatrix}$ 1 transitivité à enlever

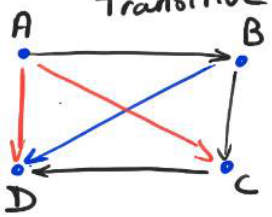
2) Comparaison à M^3

$\Rightarrow \begin{pmatrix} 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \\ 0&0&0&0 \end{pmatrix}$

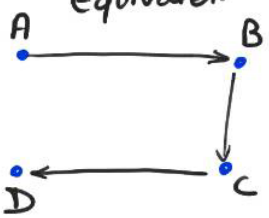
Initial



Fermeture Transitive



Minimal Equivalent



Complexité

Pour cette solution, nous avons une complexité en O(n^4).

La mise au carré d'une matrice est en O(n^3).

> Soit une matrice $M = (a_{ij})$ d'ordre n
>
> Pour cela, on part de notre matrice initiale et on retire les chemins de longueur supérieur à 1.Sa matrice $M^2 = (c_{ij})$ est donnée par la répétition de l'équation $c_{ij} = \sum_{k=1}^{n} a_{ik} \otimes a_{kj}$ qui est de complexité n.
> On va donc effectuer le calcul c_{ij}autant de fois que d'élément dans la matrice soit n^2fois.
> On obtient donc bien la complexité $O(n^2 * n) = O(n^3)$.

Par parcours des prédécesseurs

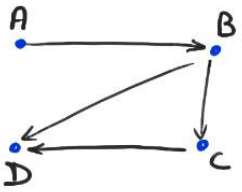
Tout d'abord, cette solution ne peut pas s'effectuer sur des graphes comprenant un cycle.

En utilisant le parcours en largeur, on va noter pour chaque sommet, ses prédécesseurs puis les prédécesseurs de ses prédécesseurs et ainsi de suite, jusqu'à ne plus en avoir.

À chaque niveau, si un prédécesseur direct du sommet est commun à un prédécesseur du niveau courant, on le supprime de la liste des prédécesseurs directs.

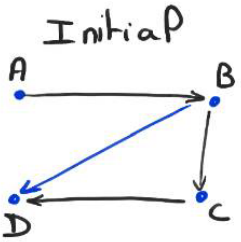
Exemple

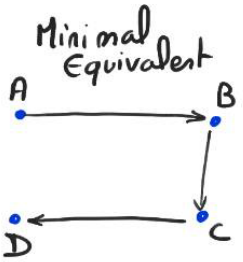
Soit une matrice $M = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$



S	P_r	P_r/P_r	$P_r/P_r/P_r$
A	∅		
B	A	∅	
C	B	A	∅
D	~~B~~C	A~~B~~	∅

On retire B des prédecesseurs de D.





Application

Langages de programmation

Nous avons deux grandes parties au sein de notre application :

- la visualisation.
- les algorithmes mathématiques.

Pour ce qui est de la visualisation, nous avons décidé de le faire au sein d'un navigateur web, dit client léger, avec les langages HTML/CSS ainsi que JavaScript. Pour la partie algorithme, nous avions choisi d'utiliser le langage Java afin d'effectuer les différents calculs mathématiques. Ce langage est typé, le développeur doit indiquer le type de chacune de ses variables, ce qui nous oblige d'avoir une grande rigueur et donc d'éliminer beaucoup d'erreurs potentielles lors de la compilation. Ceci dit, nous avons eu des difficultés à effectuer la communication entre nos deux parties que sont la visualisation et les algorithmes. Nous nous sommes rendu compte que cela affectait notre vélocité de développement, c'est pour cela que nous avons implémenté les algorithmes en JavaScript, qui s'exécutent directement au sein du navigateur web, ce qui simplifie grandement l'architecture du logiciel, un seul et même langage utilisé.

Gestion de projet

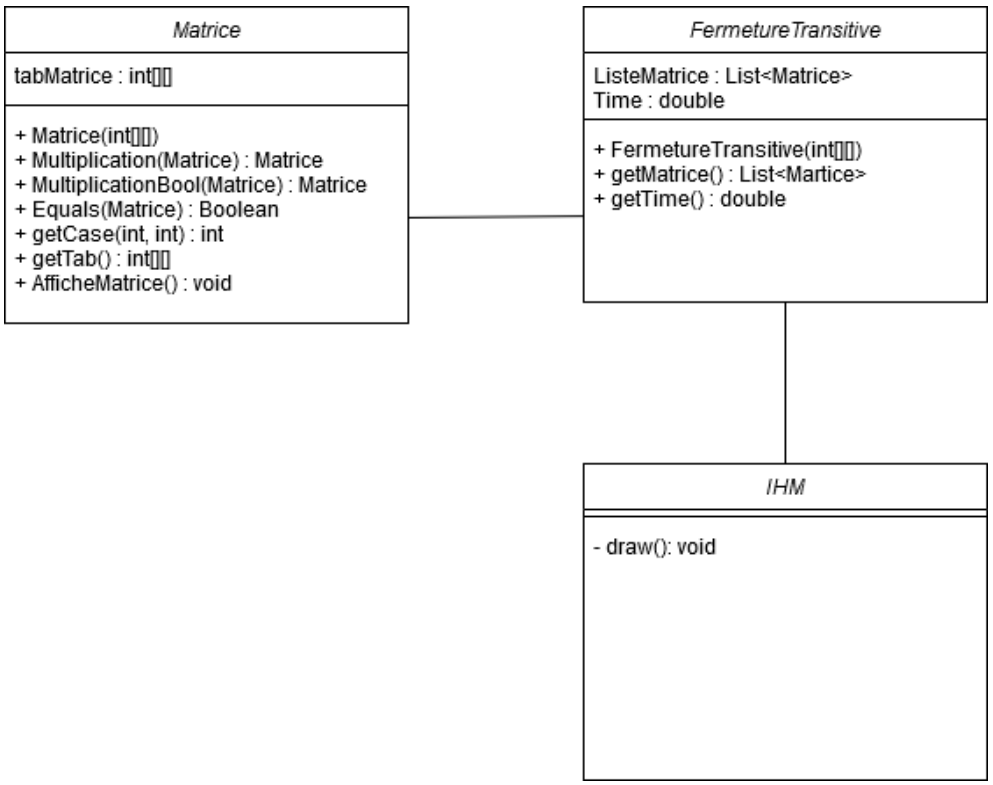
Nous avons utilisé l'outil Git afin de gérer les versions de notre logiciel et le service Github (https://github.com) afin d'héberger le projet ainsi que le piloter. Ces composants, nous ont permit de travailler sans se marcher dessus, nous savons qui fait quoi à tout moment et nous gardons tout l'historique de chaque modification.

Tests

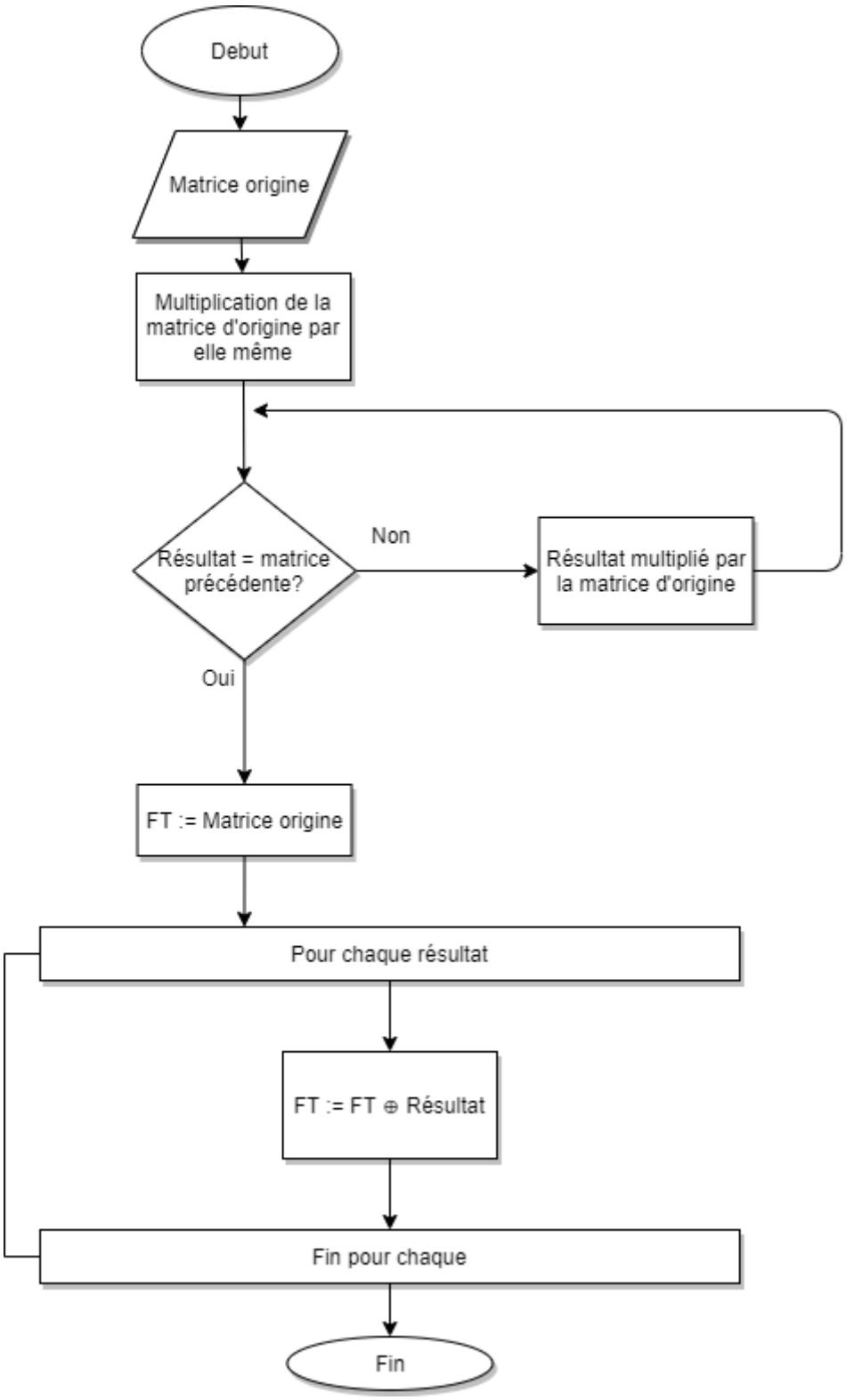
Pour tester et s'assurer du bon fonctionnement de notre logiciel, nous avons créé des jeux de données, avec des données en entrées ainsi que les résultats attendus. Ces jeux de données avec les différents scénarios de tests sont exécutés à chaque fois qu'une fonctionnalité est modifiée ou ajoutée.

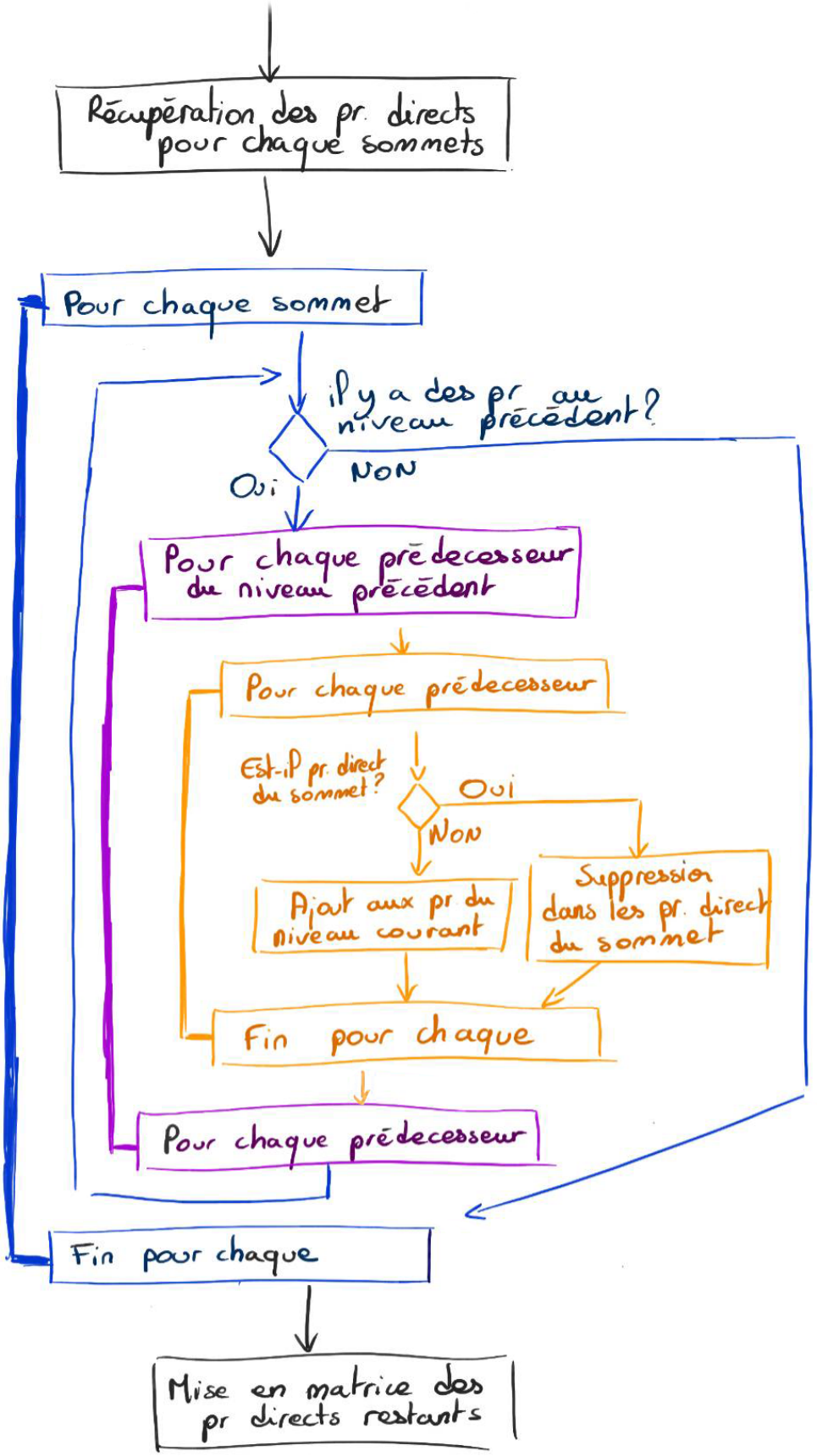
Modélisation UML de l'application

Voici le diagramme UML de notre application.



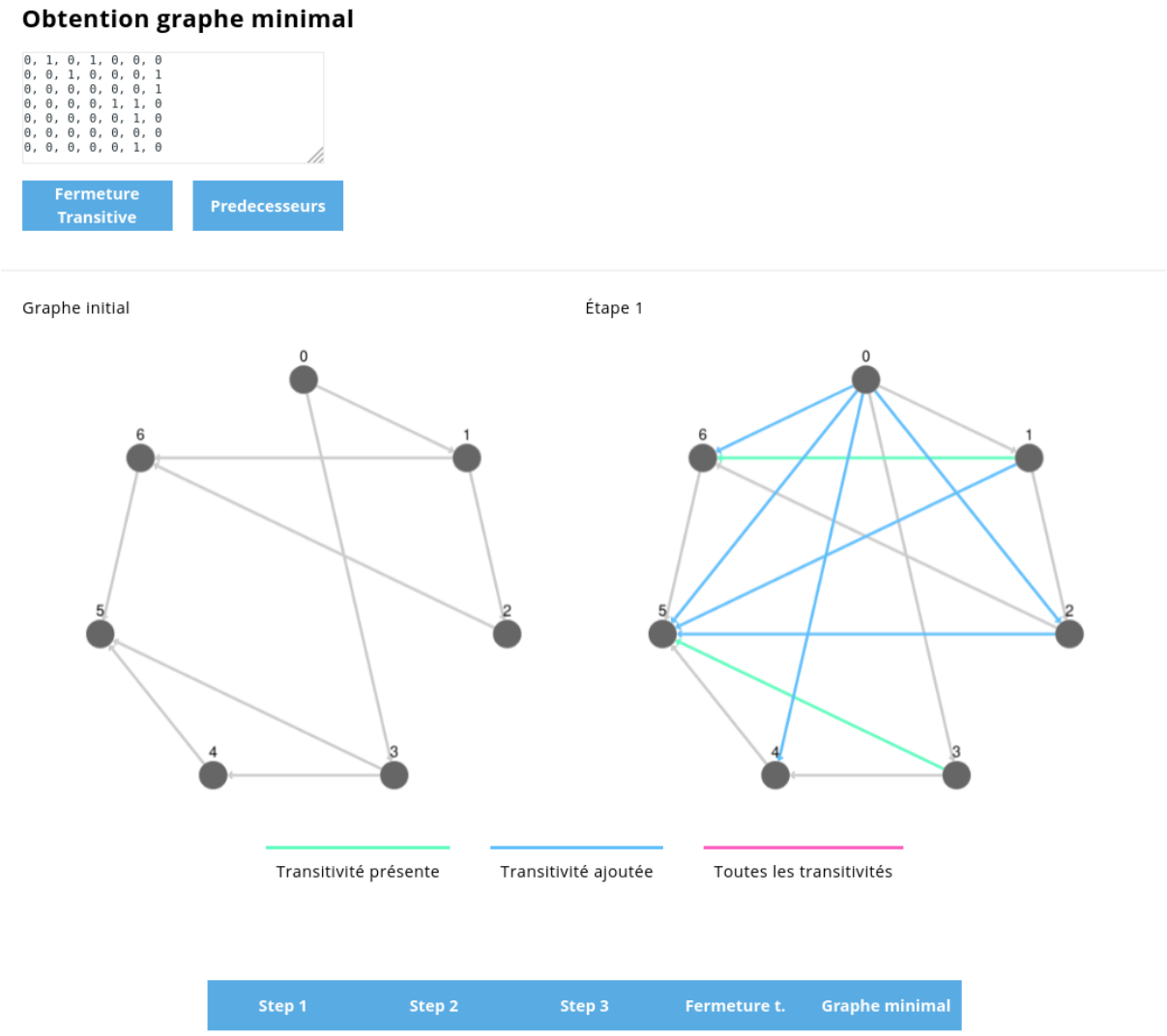
Nous y retrouvons notamment une classe "Matrice" qui encapsule nos matrices, l'avantage, c'est que celle-ci est réutilisable de projet en projet. Ensuite, nous avons une classe "FermetureTransitive" qui manipule des matrices d'adjacence afin d'obtenir la fermeture transitive d'un graphe. Pour finir, nous avons la partie "IHM" qui affiche à l'écran nos différents graphes.


Debut
Matrice origine
Multiplication de la matrice d'origine par elle même
Résultat = matrice précédente?
Non
Résultat multiplié par la matrice d'origine
Oui
FT := Matrice origine
Pour chaque résultat
FT := FT ⊕ Résultat
Fin pour chaque
Fin


Récupération des pr. directs pour chaque sommets
Pour chaque sommet
il y a des pr au niveau précédent?
Oui
NON
Pour chaque prédecesseur du niveau précédent
Pour chaque prédecesseur
Est-il pr. direct du sommet?
Oui
NON
Ajout aux pr du niveau courant
Suppression dans les pr. direct du sommet
Fin pour chaque
Pour chaque prédecesseur
Fin pour chaque
Mise en matrice des pr directs restants

Présentation de l'application

Voici le déroulement de notre application avec l'obtention de la fermeture transitive puis le graphe minimal équivalent.

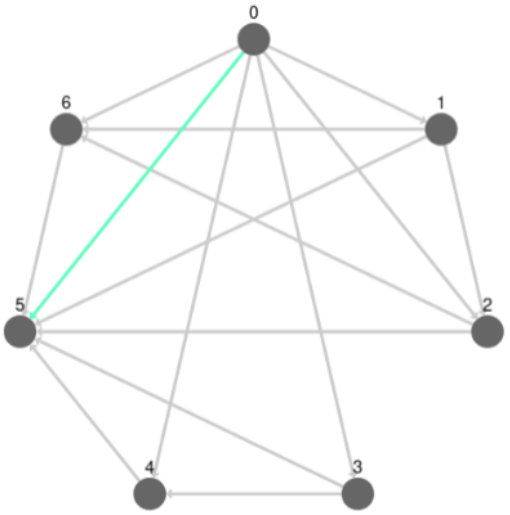


Obtention graphe minimal

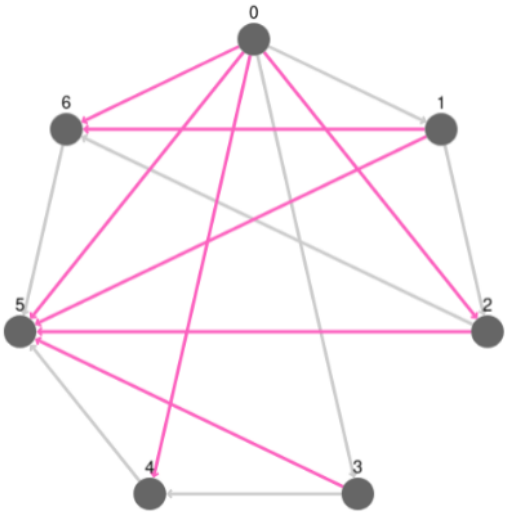
```
0, 1, 0, 1, 0, 0, 0
0, 0, 1, 0, 0, 0, 1
0, 0, 0, 0, 0, 0, 1
0, 0, 0, 0, 1, 1, 0
0, 0, 0, 0, 0, 1, 0
0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 1, 0
```

Fermeture Transitive | Predecesseurs

Étape 3

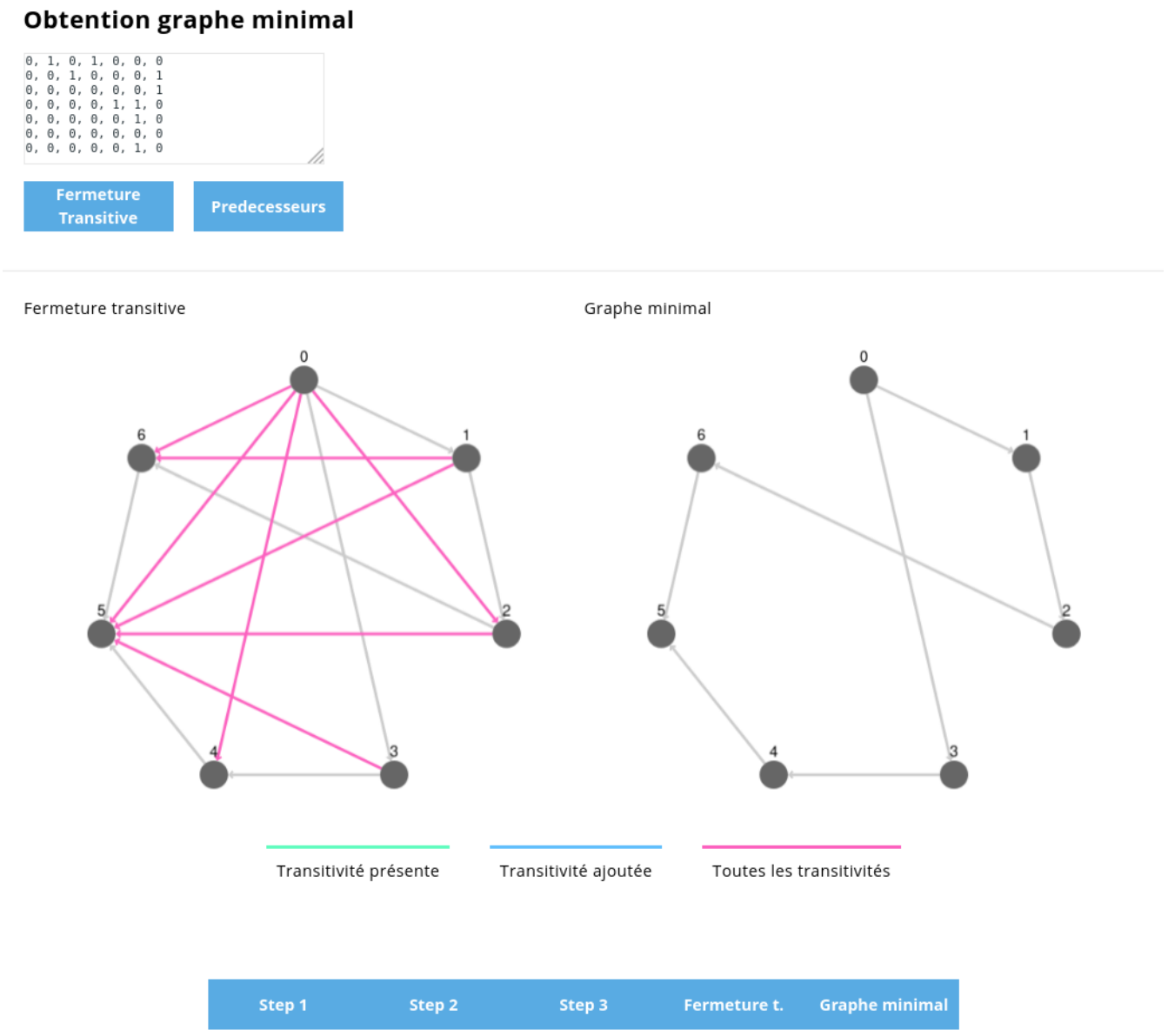


Fermeture transitive



Transitivité présente | Transitivité ajoutée | Toutes les transitivités

Step 1 | Step 2 | Step 3 | Fermeture t. | Graphe minimal



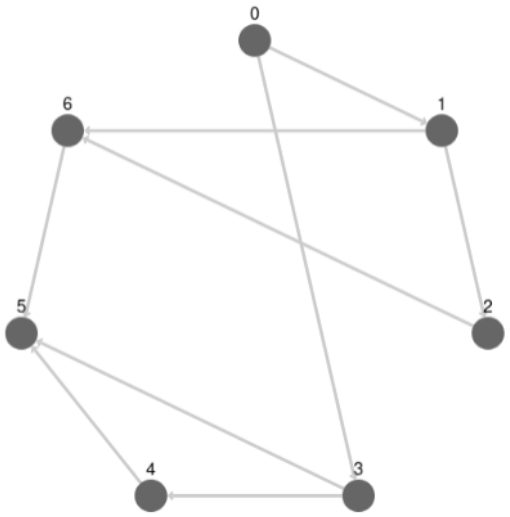
Et avec l'algorithme des prédécesseurs, où nous obtenons également le graphe minimal équivalent.

Obtention graphe minimal

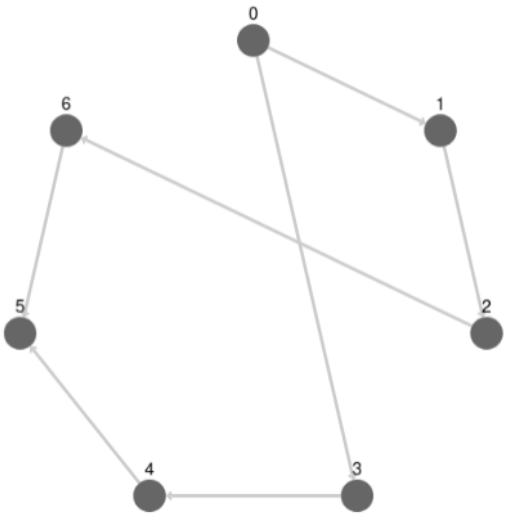
```
0, 1, 0, 1, 0, 0, 0
0, 0, 1, 0, 0, 0, 1
0, 0, 0, 0, 0, 0, 1
0, 0, 0, 0, 1, 1, 0
0, 0, 0, 0, 0, 1, 0
0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 1, 0
```

Fermeture Transitive | Predecesseurs

Graphe initial



Graphe minimal



Transitivité présente | Transitivité ajoutée | Toutes les transitivités

Conclusion

Lors de ce projet, nous avons mis en œuvre différents algorithmes afin de créer une fermeture transitive d'un graphe orienté. Ces différents algorithmes ont été implémentés au sein d'un logiciel à but éducatif montrant visuellement les différentes étapes de ces algorithmes et nous a permit de créer un comparatif de ceux-ci.