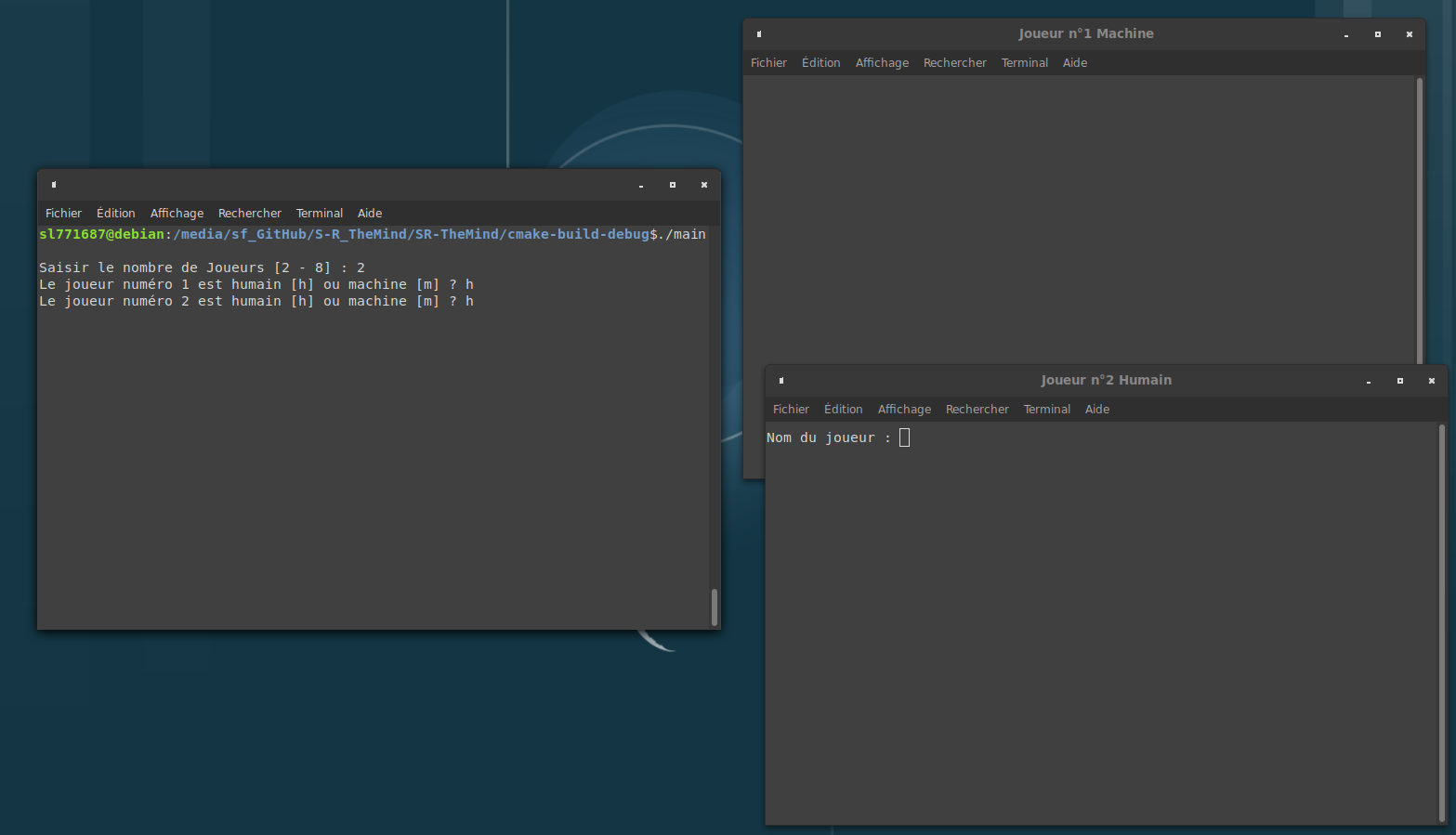


FractalSharp

LACHAUD Samuel / PAZOLA Loïs – Info S6



Une image contenant texte

Description générée automatiquement[](Installation/CDAA-R_UWP+EXE.mp4)

2/10

SOMMAIRE

Le projet S&R GraphSharp est un projet de graphe codé en C# portant sur le calcul de la dégénérescence de grands graphes réels. Ainsi à partir de deux algorithmes différents, nous allons pouvoir calculer ce nombre de dégénérescence et le comparé avec le nombre chromatique d’une multitude de graphes réels du projet KONNECT (<http://konect.cc/>). Nous proposerons également un export de la dégénérescence de chaque sommet des graphes en PDF sous formes de cercles imbriqués pour une visualisation rapide et démonstrative. Le projet est open source et trouvable à l’adresse suivante : <https://github.com/samlach2222/GraphSharp>

3/10

Pour les actions autres que la comparaison des nombres chromatiques et de la dégénérescence, plusieurs graphes différents nous sont proposés avec des tailles différentes

GraphSharp est une application console proposant par le biais du Windows Terminal, une interface basique demandant à l’utilisateur un choix parmi les différentes fonctionnalités.

Présentation de l’application

Qu'est-ce que GraphSharp

3/10

I) Introduction



II) Fonctionnement de

l'application

Affichage du graphe



4/10

Dans un premier temps, nous devons lire le fichier texte, ainsi, on détecte chaque ligne et on récupère les deux valeurs qui sont déparées par une tabulation (\t). Une fois récupéré, nous allons ranger ces valeurs de la manière suivante :

On crée un tableau composé de deux colonnes : la première étant le numéro du sommet et la deuxième une autre liste composée de tout les sommets qui ont une arrête en commun avec le sommet de la première colonne

Une fois le fichier chargé, l’affichage du graphe consiste juste à afficher le contenu du tableau nouvellement créé.

6/10

6 / 9

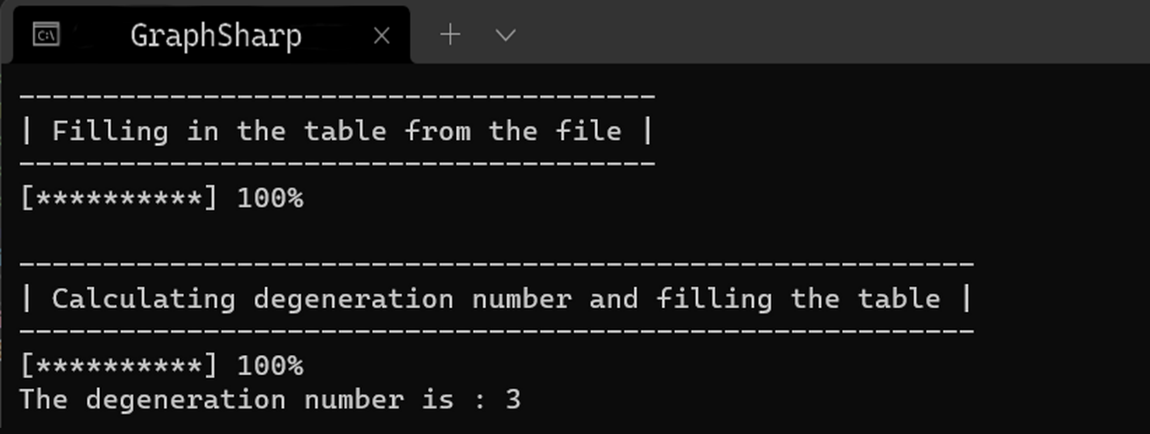


3/11

Calcul de dégénérescence



Une fois que notre graphe est chargé dans notre tableau, nous pouvons calculer la dégénérescence de celui-ci.

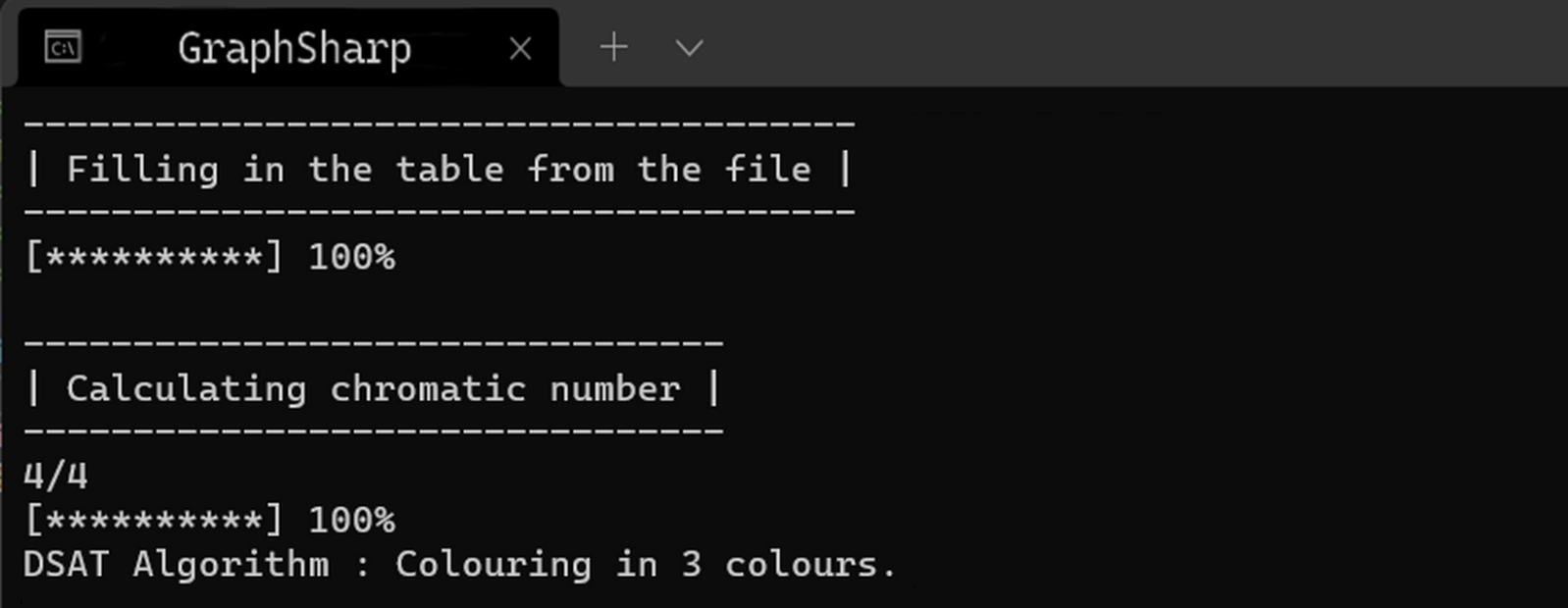


5/10

Nous stockons le graphe précédemment chargé dans une copie afin de pouvoir localement supprimer les sommets déjà traités par la k-dégénérescence. Cela nous permet de ne pas altérer le graphe chargé et ainsi ne pas avoir à le recharger en mémoire durant une autre opération. En supprimant les nœuds au fur et à mesure, on obtient finalement la dégénérescence du graphe, le k le plus grand.

Sur l’exemple du cours, nous avons donc la dégénérescence k = 3 qui est calculée :

7/10



static void Main()

{

AskUserNbProcessMpi();

InitializeForm(pixelWidth, pixelHeight);

CalculateMandelbrot(0, 0, pixelWidth, pixelHeight); // Calculate the whole Mandelbrot

}

6/10

Calcul du nombre chromatique



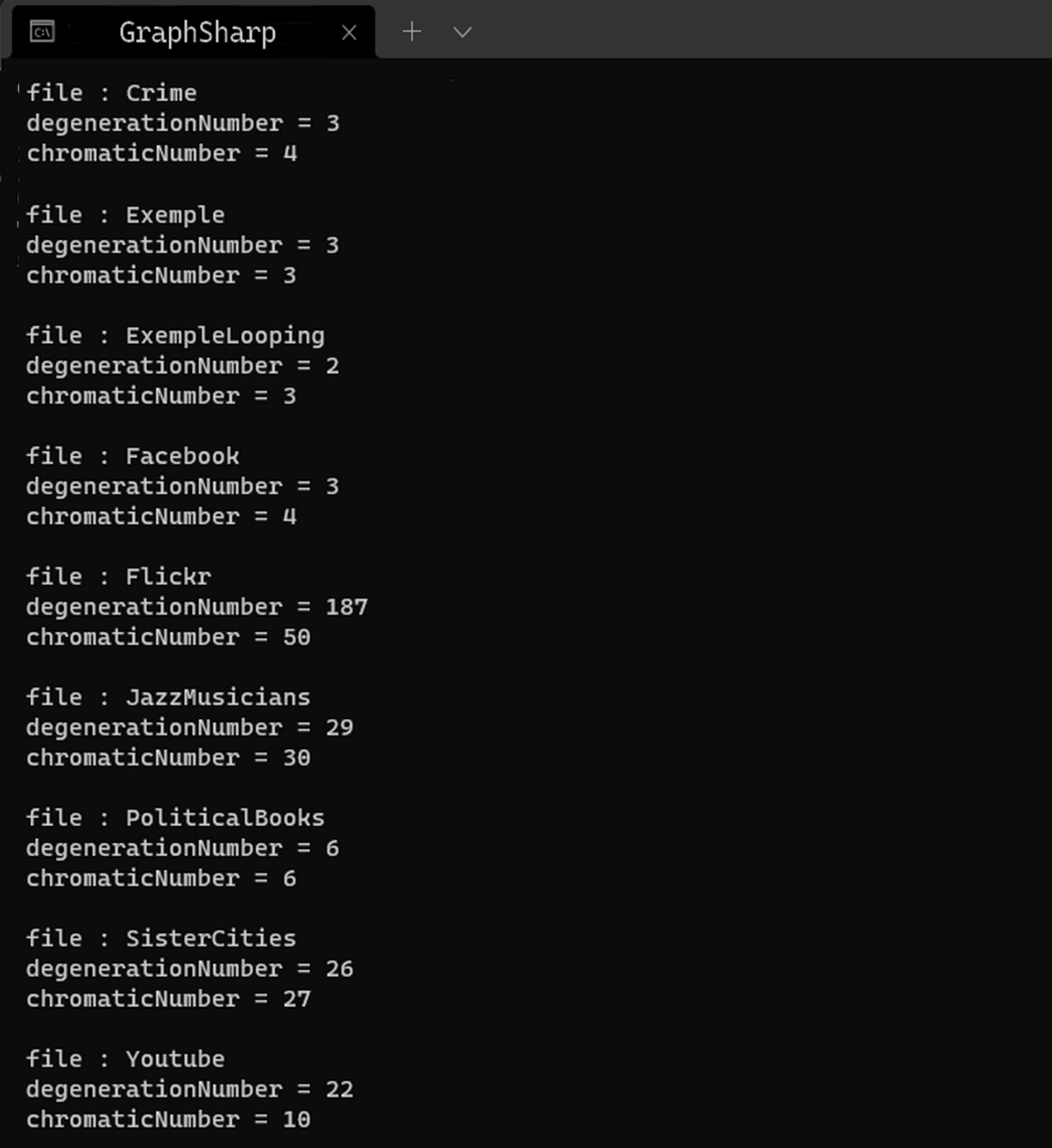
Pour aller plus loin dans le projet, 3 améliorations ont été mises en place. Nous allons tout d’abord voir la comparaison entre la dégénérescence et le nombre chromatique sur un ensemble de graphe **(point 3.b du sujet)**, puis la génération d’un fichier PDF **(point 3.c du sujet)** et enfin un meilleur algorithme pour le calcul de la dégénérescence qu’est l’algorithme de Matula & Beck **(point 3.a du sujet).**

III) Fonctionnalités

supplémentaires

Dégénérescence/nombre chromatique

Après avoir calculé le nombre chromatique et la dégénérescence d’un graphe, nous pouvons donc maintenant la comparer sur l’ensemble des graphes présents dans le programme. En saisissant l’option 5 sur le programme, ces deux valeurs vont donc être calculées sur l’ensemble des graphes (sauf Flickr car beaucoup trop lourd). Un récapitulatif sera alors montré à l’utilisateur :





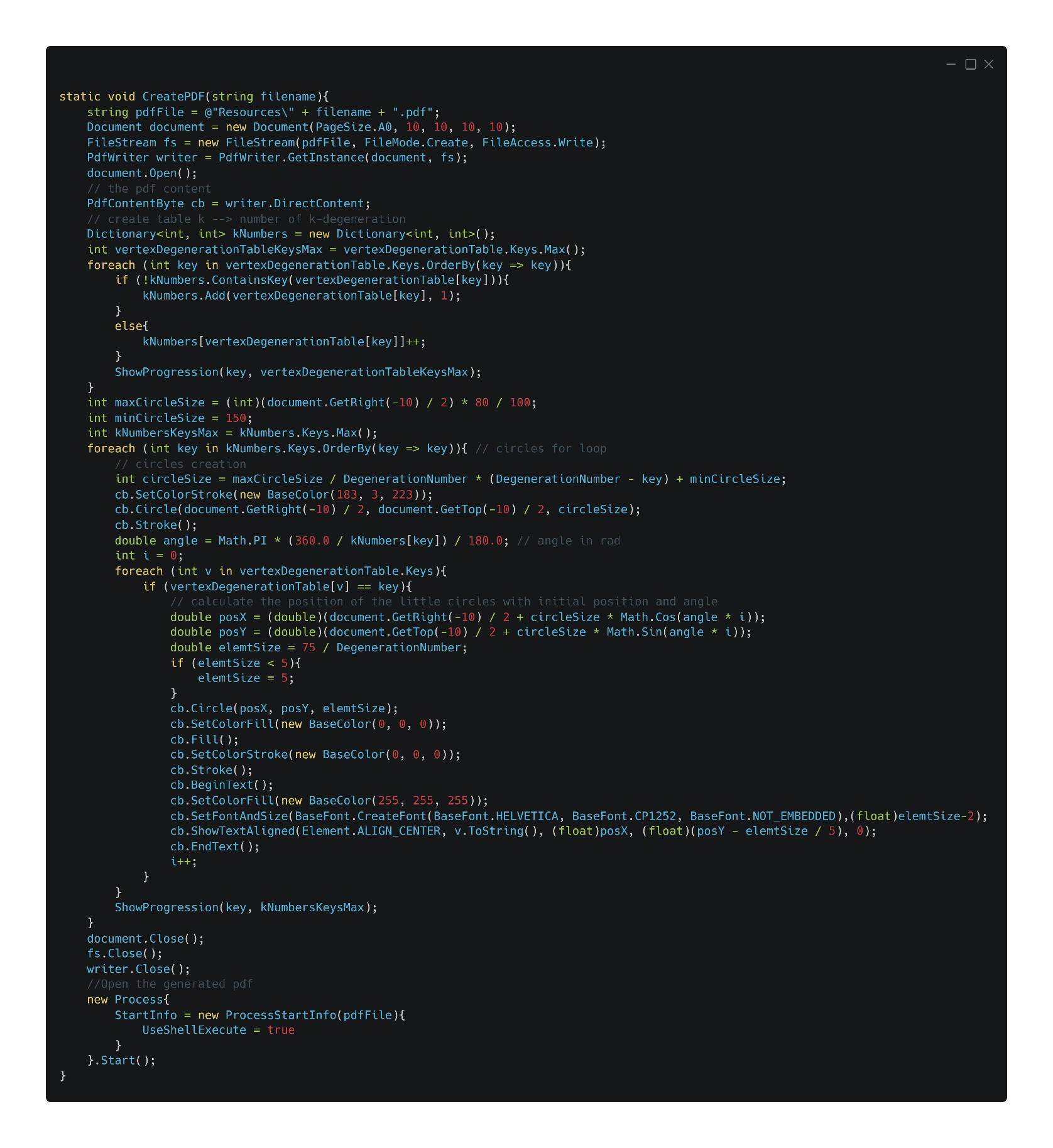
7/10



Génération d’un fichier PDF

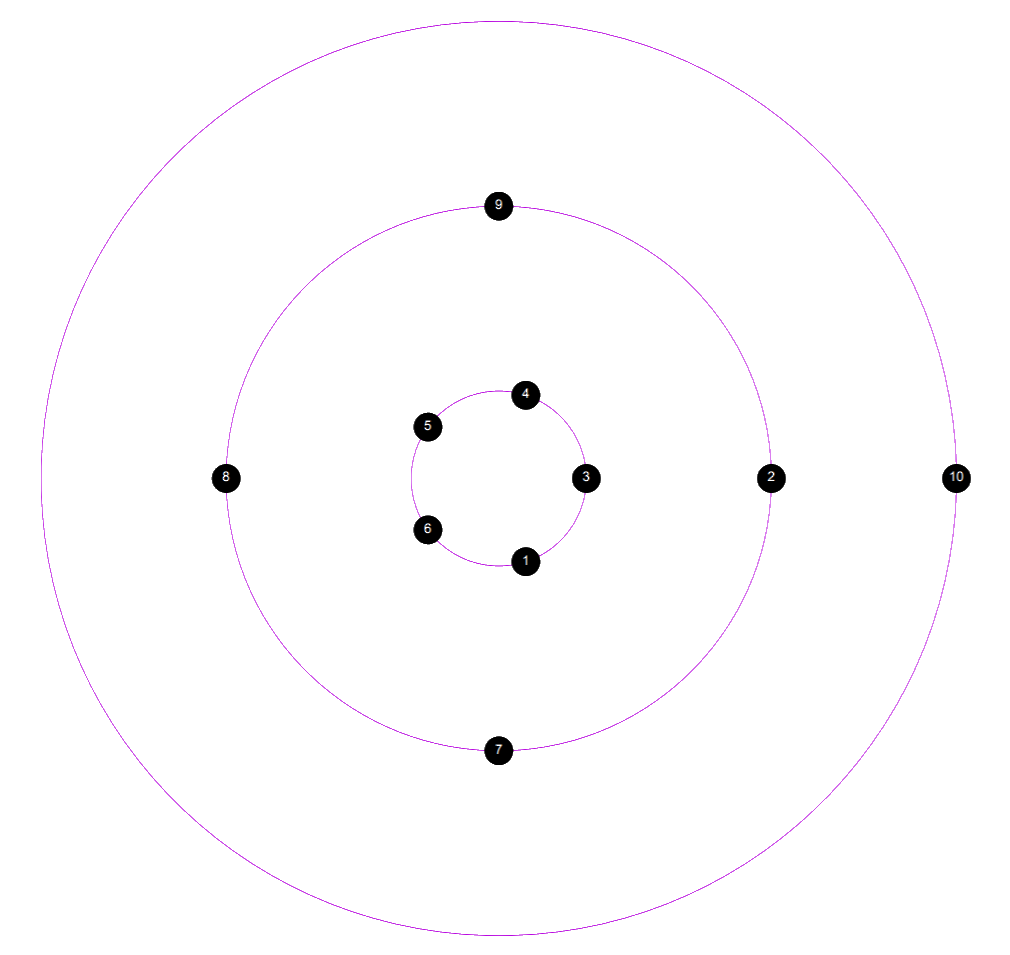
A partir de ce rendu, nous pouvons donc déduire que l’affirmation « on peux colorier un graphe k dégénéré en au plus k + 1 couleurs » est vrai dans notre cas d’observation. En effet la valeur **chromaticNumber** est toujours inférieure ou égale à la valeur **degenerationNumber** + 1.

On peut également supposer aux vues de ce rendu que l’écart entre la dégénérescence et le nombre chromatique est plutôt faible, mais que quelques exceptions sont tout de même présentes comme pour le fichier **Youtube.**



La dégénérescence de notre graphe est calculable, mais difficilement visualisable, en effet, on calcule la k-dégénérescence maximale, mais également la dégénérescence de chaque sommet du graphe. Nous pouvons donc visualiser ce graphe avec des cercles imbriqués, un cercle pour chaque k-dégénérescence. Le tout est mis dans un fichier PDF et chaque sommet est placé sur le cercle lui correspondant.

8/10



Le programme ouvre donc le fichier PDF généré, voici celui correspondant à l’exemple. Ainsi le cercle extérieur représente les sommets de dégénérescence 1 puis à l’intérieur le 2 et enfin le 3 :

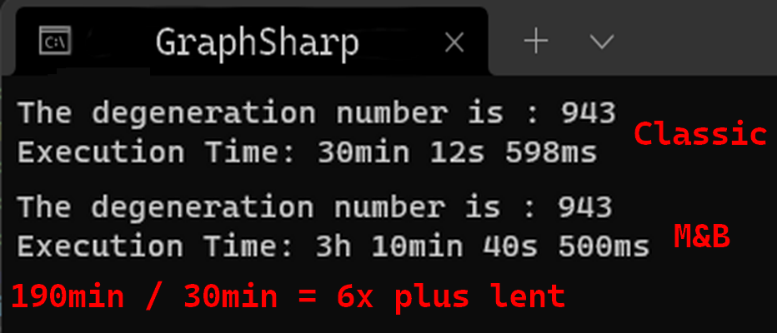
Algorithme de Matula & Beck

La dernière amélioration de notre programme est l’ajout de l’algorithme de Matula & Beck. En effet il permet également de calculer la dégénérescence d’un graphe, mais de manière beaucoup plus rapide.

La mise en place de l’algorithme de Matula & Beck s’est fait à l’aide de la thèse suivante : <https://schulzchristian.github.io/thesis/thesis_huebner.pdf>.

9/10



Une image contenant texte

Description générée automatiquement

Pour montrer cela, nous allons comparer la vitesse de calcul de la dégénérescence du graphe **Flickr**.   
Ce graphe est composé de plus de **8 500 000 arrêtes** et est stocké dans un fichier pesant près de **100Mo**. Le calcul est pourtant très rapide mais la différence est saisissante

En effectuant des essais sur des plus grands graphes, nous avons trouvé quelque chose d’étrange. Sur le graphe **UK-Domains** composé de **261 787 000 arrêtes** et est stocké dans un fichier de **4Go**. **Matula & Beck semble plus lent pour l’exécution de très grands fichiers !**

10/10