Algorithmique

Parcours DFS et applications

Quentin DESCHAMPS

2020

Table des matières

1 Introduction

Ce rapport vise à résumer et expliquer les stratégies utilisées afin d'implémenter le parcours dfs d'un graphe et ses applications. Le projet permet de générer la librairie statique *libgraph* réalisée à partir de 6 fichiers :

1. **importGraph.c**
2. **dfsGraph.c**
3. **connexeGraph.c**
4. **cfcGraph.c**
5. **inverseGraph.c**
6. **orientForteGraph.c**

Ces fichiers sources se trouvent dans le répertoire **src**. Les headers correspondant à ces fichiers se trouvent dans le répertoire **include**. Le makefile se trouvant dans le répertoire **src** permet de créer la librairie statique *libgraph*.

Le répertoire **graphs** contient des graphes au format .txt. Les fichiers nommés *digraph* correspondent à des graphes orientés et ceux nommés *graph* à des graphes non orientés.

Le répertoire **analyzer** contient le fichier source **analyse.c** permettant d'analyser un graphe au format .txt. Le Makefile se trouvant dans le répertoire courant crée l'exécutable **graphAnalyzer** analysant ces graphes.

Le répertoire **img** contient des images .png associées aux fichiers .dot générés par le programme.

Dans ce rapport, chaque section correspond à la description d'un fichier source et chaque sous-section correspond à l'explication d'une fonction.

2 Structures

Le projet utilise deux structures qui se trouvent dans le fichier **structGraph.h** :

- **Arete**
- **Graphe**

La structure **Arete** permet de représenter une arête ou un arc d'un graphe. Elle est composée de 2 champs :

- **sommet1** (type int)
- **sommet2** (type int)

Les sommets sont de type entier. Dans le cas d'un graphe orienté, sommet1 correspond au sommet initial de l'arc et sommet2 au sommet terminal.

La structure **Graphe** permet de stocker les données d'un graphe. Elle est composée de 4 champs :

- **nbSommets** (type int) : nombre de sommets du graphe
- **nbAretes** (type int) : nombre d'arêtes ou d'arcs d'un graphe
- **oriente** (type int) : vaut 1 si le graphe est orienté, 0 sinon
- **listeAretes** (type Arete*) : liste des arêtes ou arcs du graphe

3 Importation des graphes et exportation

Les fonctions du fichier **importGraph.c** permettent d'importer les graphes sous la forme **graphe.txt** et de les exporter sous le format **graphe.dot**.

3.1 importationGraphe

La fonction **importationGraphe** prend en paramètre le nom d'un fichier .txt et l'orientation du graphe, donc 1 pour un graphe orienté et 0 sinon.

Le fichier est alors ouvert en lecture. Le programme retourne un pointeur sur une structure Graphe. Les champs de la structure sont remplis en lisant le fichier .txt pris en entrée.

3.2 affichageInformations

La fonction **affichageInformations** prend un pointeur sur une structure Graphe en paramètre. Elle permet simplement de visualiser les champs de la structure, afin de voir si les informations du fichier .txt ont été correctement transmises. Cette fonction a uniquement été utilisée pour la vérification du fonctionnement de la fonction précédente.

3.3 exportationDot

La fonction **exportationDot** prend en paramètre un pointeur sur une struture Graphe et un nom de fichier. Elle a pour but d'exporter le graphe pris en paramètre en fichier .dot. Le nom du fichier en paramètre correspond au nom du fichier .dot à créer.

Le programme créer d'abord le fichier sous le nom indiqué. Il écrit ensuite dans ce fichier :

- La première ligne, qui diffère selon que le graphe est orienté ou non
- Les sommets : obligatoire pour que tous les sommets apparaissent bien dans le cas des graphes non convexes
- Les arêtes : écrites 1 -> 2 pour les graphes orientés et 1 – 2 pour les non-orientés

3.4 liberationMemoire

La fonction **liberationMemoire** permet de libérer la mémoire occupée par une structure Graphe, dont le pointeur est pris en paramètre. On libère d'abord la liste des arêtes, puis la structure Graphe.

4 Parcours dfs

Les fonctions du fichier **dfsGraph.c** permettent d'afficher le parcours dfs d'un graphe dans le terminal et de créer un fichier .dot permettant de visualiser ce parcours.

4.1 dfsAffiche et afficheParcoursDfs

Les fonctions **dfsAffiche** et **afficheParcoursDfs** permettent d'afficher dans le terminal un parcours dfs du graphe dont le pointeur est pris en paramètre. La fonction dfsAffiche est récursive et effectue le parcours dfs.

La liste nonMarque permet de marquer les sommets déjà visités par le parcours. La taille de cette liste correspond au nombre de sommets. Les sommets étant numérotés à partir de 1, l'indice i de la liste correspond au sommet i + 1.

4.2 exportationDotDfs, dfsCouleur et creeParcoursDfsCouleur

La fonction **creeParcoursDfsCouleur** crée un fichier .dot où on peut visualiser le parcours dfs d'un graphe dont le pointeur est pris en paramètre. L'argument nomFichier correspond au nom du fichier .dot à créer.

Cette fonction fait appel à **dfsCouleur**, qui effectue le parcours dfs sur le graphe. La liste color est une liste de 0 et de 1 qui permet de renseigner quelles arêtes sont parcourues pendant le parcours dfs. À chaque arête correspond une case de la liste color. Les indices des arêtes sont les mêmes que pour la liste listeAretes. Les arêtes parcourues sont notées 1 dans la liste color. De la même façon, la liste step permet de connaître le numéro de l'étape du dfs à laquelle l'arête est parcourue. La première étape est numérotée 1. Les arêtes non parcourues par le dfs ont comme numéro d'étape 0 dans cette liste.

Une fois le parcours dfs terminé, le programme effectue l'exportation en .dot en appelant la fonction **exportationDotDfs**. Celle-ci fonctionne comme **exportationDot**, sauf qu'elle rajoute une couleur bleue sur les arêtes parcourues par le dfs, ainsi que le numéro des étapes des arêtes parcourues. Pour cela, elle utilise les listes color et step qui renseignent quelles sont les arêtes à colorer et leur numéro d'étape. Les arêtes non parcourues par le dfs sont colorées en gris.

5 Connexité

Les fonctions du fichier **connexeGraph.c** permettent d'étudier la connexité des graphes. Un parcours dfs est alors utilisé.

5.1 dfsConnexe et estConnexe

La fonction **estConnexe** renvoie 1 si le graphe dont le pointeur est pris en argument est connexe, 0 sinon.

Pour étudier la connexité, le programme appelle la fonction **dfsConnexe** qui effectue un parcours dfs du graphe de façon récursive. Ce parcours ne tient pas compte de l'orientation du graphe. En appelant une fois cette fonction et en comptant le nombre de sommets parcourus à la fin, on peut déterminer si le graphe est connexe. En effet, si le compteur est égal au nombre de sommets du graphe, alors il est connexe. Si le compteur est inférieur, alors il n'est pas connexe.

5.2 afficheEstConnexe

La fonction **afficheEstConnexe** permet juste d'afficher dans le terminal si le graphe dont le pointeur est pris en paramètre est connexe ou non, en appelant la fonction **estConnexe**.

6 Inversion

Le fichier **inverseGraph.c** contient une fonction permettant d'obtenir l'inverse d'un graphe.

6.1 inverseGraphe

La fonction **inverseGraphe** retourne l'inverse du graphe dont le pointeur est pris en paramètre. Elle crée une nouvelle structure Graphe et, en parcourant la liste des arêtes du graphe en paramètre, remplit sa liste listeAretes en inversant les sommets notés sommet1 et sommet2 des arêtes.

7 Composantes fortement connexes

Le fichier **cfcGraph.c** permet de déterminer les composantes fortement connexes d'un graphe et de créer un fichier .dot permettant de les visualiser. La complexité des algorithmes utilisés est polynomiale.

7.1 dfsPostOrdre, dfsClasse, exportationDotCFC et CFC

La fonction **CFC** permet de montrer les composantes fortement connexes d'un graphe dont le pointeur est pris en argument.

Le programme va tout d'abord déterminer le post-ordre du parcours dfs du graphe. Pour cela, il fait appel à la fonction **dfsPostOrdre** qui permet de déterminer cet ordre, en effectuant de manière récursive le parcours dfs. Le post-ordre est aussi affiché dans le terminal.

Ensuite, le programme crée le graphe inverse du graphe en appelant la fonction **inverseGraphe**. Le programme détermine alors les composantes fortement connexes du graphe en effectuant un dfs sur le graphe inverse, en suivant l'ordre inverse du post-ordre. Pour ce parcours dfs, le programme appelle la fonction **dfsClasse**. Les numéros de classes des sommets sont stockés dans la liste numClasse. La classe du sommet i est à la case i - 1 de la liste. Les classes sont aussi affichées dans le terminal.

Enfin, le graphe est exporté en .dot avec le fonction **exportationDotCfc** en colorant de la même couleur les sommets appartenant à la même classe et en encadrant ces différentes classes. On peut alors visualiser facilement les composantes fortement connexes du graphe.

8 Orientation forte

Le fichier **orientForteGraph.c** permet de déterminer une orientation forte, si elle existe, d'un graphe non orienté et de l'exporter au format .dot. L'algorithme utilisé pour répondre à ce problème est de complexité polynomiale.

8.1 echangeAretes

La fonction **echangeAretes** prend un pointeur sur une liste d'arêtes en paramètre et échange le contenu des arêtes aux indices i et j, qui sont en argument. Cette fonction est utile pour la fonction **estSansPont**.

8.2 estSansPont

La fonction **estSansPont** retourne 1 si le graphe dont le pointeur est pris en paramètre est sans pont et 0 sinon.

Pour déterminer cela, le programme teste la connexité des graphes partiels ayant une arête en moins. La stratégie utilisée ici est de créer une structure de graphe identique au graphe à étudier, mais en définissant un nombre d'arêtes d'une de moins que le graphe. Pour supprimer une arête, il suffit alors juste de la mettre en dernière position de la liste d'arêtes de la structure, en utilisant la fonction **echangeAretes**. Celle-ci ne sera alors pas prise en compte à

l'appel de la fonction **estConnexe**, puisque le nombre d'arêtes a été réduit de 1.

Le programme a même pour option d'afficher dans le terminal un pont du graphe s'il y en a un.

8.3 afficheEstSansPont

La fonction **afficheEstSansPont** permet d'afficher si un graphe est sans pont ou non dans le terminal. Elle fait appel à la fonction **estSansPont**.

8.4 dfsOrientationForte et orientationForte

La fonction **orientationForte** retourne un graphe fortement orienté du graphe non orienté pris en paramètre.

Tout d'abord, le programme regarde si une orientation forte du graphe en paramètre existe. Pour cela, il fait appel à la fonction **estSansPont**. En effet, un graphe non orienté est fortement orientable si et seulement si il est sans pont [1].

Dans le cas d'un graphe sans pont, le programme crée alors un graphe fortement orienté. Pour déterminer l'orientation des arêtes, il effectue alors un dfs grâce à la fonction **dfsOrientationForte**. En effet, les arêtes sont orientées dans le sens du dfs. Quand le dfs tombe sur un sommet déjà visité, il "ferme le circuit" en orientant l'arête **vers** le sommet déjà visité.

Le programme retourne alors un graphe fortement orienté sous la forme d'une structure graphe. Cette structure est alors utilisable avec les fonctions des parties précédentes, pour vérifier la forte connexité et exporter le graphe fortement orienté en .dot.

1. En effet, s'il y a un pont, il n'existe alors pas de chemin aller-retour pour aller au sommet isolé par le pont. Il est alors impossible de trouver une orientation forte dans ce cas.

9 Analyse complète

Le fichier **analyse.c** permet d'analyser entièrement un graphe à partir de toutes les fonctions précédentes.

9.1 analyseGraphe

La fonction **analyseGraphe** permet d'effectuer toutes les analyses précédemment définies pour un graphe dont le nom est pris en paramètre [2]. Nous appellerons ce fichier "graphe.txt" pour l'exemple. Voici les différentes étapes :
Si le graphe est orienté :

1. Importation du graphe : appel de **importationGraphe**
2. Création du fichier **graphe.dot** du graphe : appel de **exportationDot**
3. Création du graphe inverse : appel de **inverseGraphe**
4. Création du fichier **graphe-inv.dot** du graphe inverse : appel de **exportationDot** sur le graphe inverse
5. Affichage du parcours dfs dans le terminal : appel de **afficheParcoursDfs**
6. Création du fichier **graphe-dfs.dot** montrant le parcours dfs du graphe : appel de **creeParcoursDfsCouleur**
7. Affiche si le graphe est connexe ou non dans le terminal : appel de **afficheEstConnexe**
8. Création du fichier **graphe-cfc.dot** montrant les composantes fortement connexes du graphe : appel de **CFC**
9. Libération de la mémoire occupée par le graphe et son graphe inverse : appel de **liberationMemoire**

Si le graphe n'est pas orienté :

1. Affiche si le graphe est sans pont ou non dans le terminal [3] : appel de **afficheEstSansPont**
2. Détermination d'une orientation forte : appel de **orientationForte**
3. **Si une orientation forte existe**, se repporter au cas où le graphe est orienté.

2. Attention, cette fonction prend le nom du graphe sans l'extension .txt
3. Affichage d'un pont dans le cas où le graphe n'est pas sans pont

10 Exemples

10.1 Graphe orienté

Voici l'exécution de la fonction **analyseGraphe** sur le fichier **digraph-1.txt** :

Affichage en console :

```
========== Analyse du fichier : digraph-1.txt ==========
Importation du graphe du fichier digraph-1.txt réussie !
Le fichier digraph-1.dot a été créé.
Le fichier digraph-1-inv.dot a été créé.
Parcours en profondeur : 1 -> 3 -> 4 -> 2 -> 5 -> 6 -> 7 -> 8 -> 9
-> 10
Le fichier digraph-1-dfs.dot montrant le parcours dfs a été créé.
Le graphe est connexe.
Détermination des classes fortement connexes :
Post-ordre : 2 -> 7 -> 6 -> 5 -> 4 -> 10 -> 9 -> 8 -> 3 -> 1
Numéro de classe des sommets :
Sommet 1 : 1
Sommet 2 : 1
Sommet 3 : 1
Sommet 4 : 1
Sommet 5 : 5
Sommet 6 : 5
Sommet 7 : 5
Sommet 8 : 2
Sommet 9 : 3
Sommet 10 : 4
Le fichier digraph-1-cfc.dot montrant les composantes fortement connexes
a été créé.
========== Fin analyse fichier : digraph-1.txt ==========
```

Contenu de digraph-1.dot :

```
digraph mon_graphe {
        1;
        2;
        3;
        4;
        5;
        6;
        7;
        8;
        9;
        10;
        1 -> 3;
        2 -> 1;
        2 -> 3;
        3 -> 4;
        3 -> 8;
        4 -> 2;
        4 -> 5;
        4 -> 6;
        5 -> 6;
        6 -> 7;
        7 -> 5;
        8 -> 9;
        8 -> 10;
        9 -> 10;
}
```

Contenu de digraph-1-inv.dot :

```
digraph mon_graphe {
        1;
        2;
        3;
        4;
        5;
        6;
        7;
        8;
        9;
        10;
        3 -> 1;
        1 -> 2;
        3 -> 2;
        4 -> 3;
        8 -> 3;
        2 -> 4;
        5 -> 4;
        6 -> 4;
        6 -> 5;
        7 -> 6;
        5 -> 7;
        9 -> 8;
        10 -> 8;
        10 -> 9;
}
```

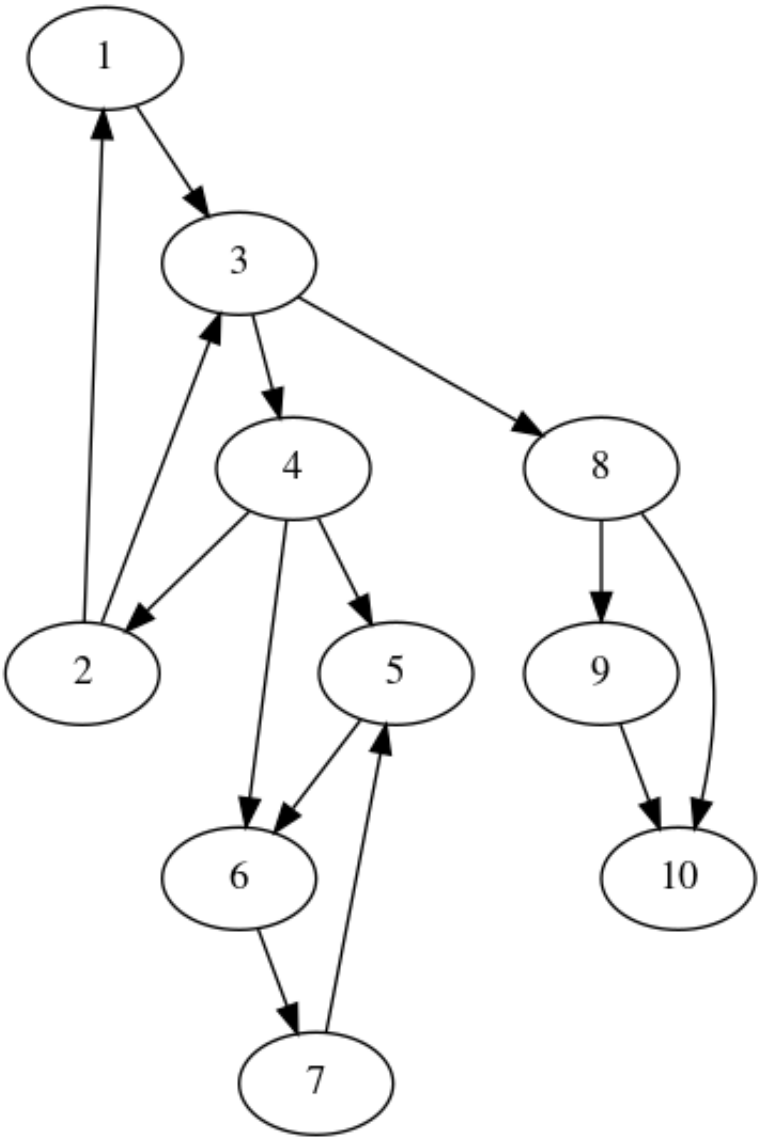

FIGURE 1 – Fichier digraph-1.dot

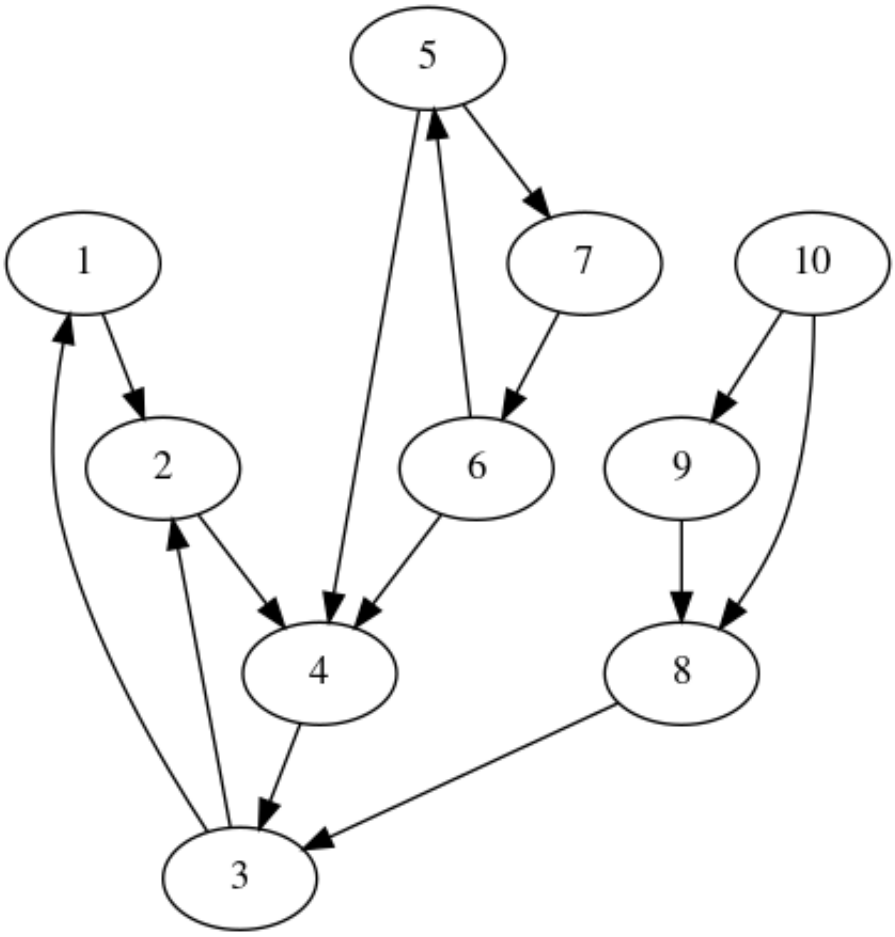

FIGURE 2 – Fichier digraph-1-inv.dot

Contenu de digraph-1-dfs.dot :

```
digraph mon_graphe {
        label="Parcours dfs";
        1;
        2;
        3;
        4;
        5;
        6;
        7;
        8;
        9;
        10;
        1 -> 3 [color=blue, penwidth=3.0, label="Step 1"];
        2 -> 1 [color=gray];
        2 -> 3 [color=gray];
        3 -> 4 [color=blue, penwidth=3.0, label="Step 2"];
        3 -> 8 [color=blue, penwidth=3.0, label="Step 7"];
        4 -> 2 [color=blue, penwidth=3.0, label="Step 3"];
        4 -> 5 [color=blue, penwidth=3.0, label="Step 4"];
        4 -> 6 [color=gray];
        5 -> 6 [color=blue, penwidth=3.0, label="Step 5"];
        6 -> 7 [color=blue, penwidth=3.0, label="Step 6"];
        7 -> 5 [color=gray];
        8 -> 9 [color=blue, penwidth=3.0, label="Step 8"];
        8 -> 10 [color=gray];
        9 -> 10 [color=blue, penwidth=3.0, label="Step 9"];
}
```

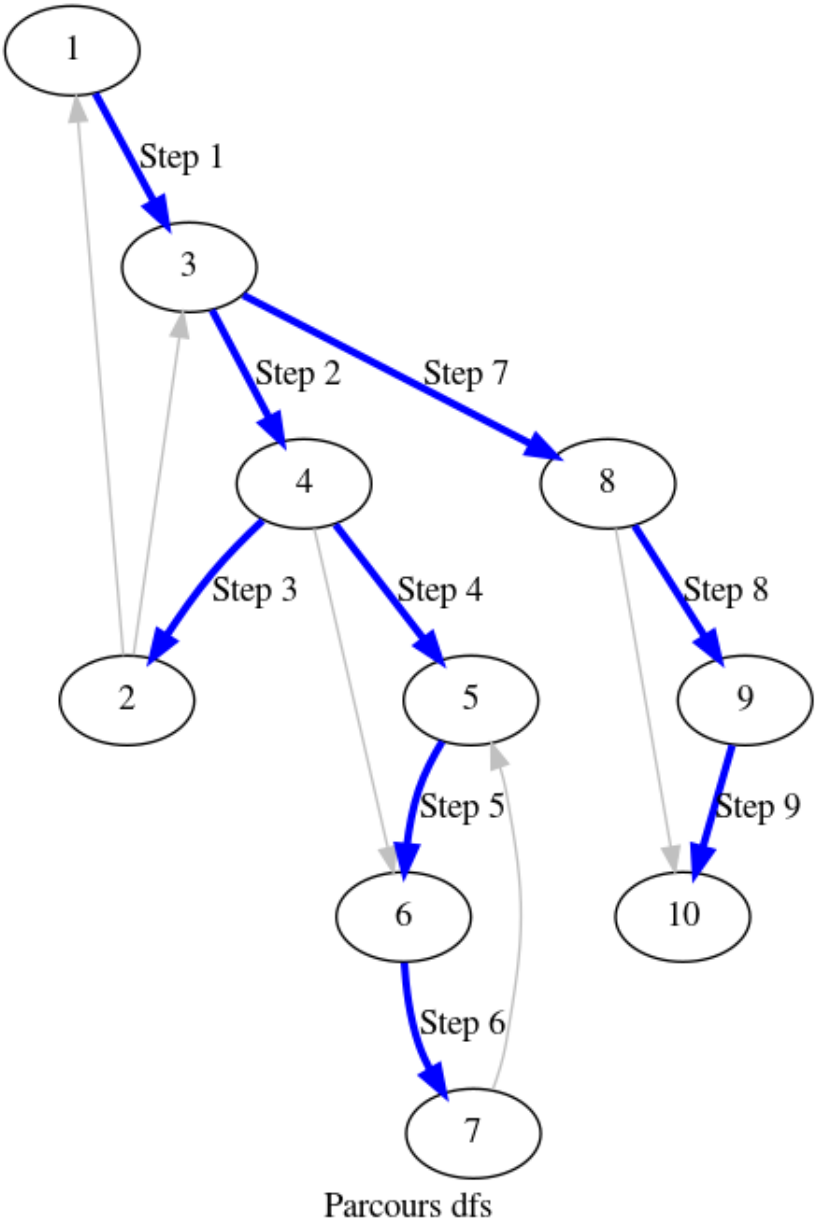

FIGURE 3 – Fichier digraph-1-dfs.dot

Contenu de digraph-1-cfc.dot :

```
digraph mon_graphe {
        label="Composantes fortement connexes";
        subgraph cluster_1 {
                label="Classe 1";
                1 [colorscheme=paired12, color=1];
                2 [colorscheme=paired12, color=1];
                3 [colorscheme=paired12, color=1];
                4 [colorscheme=paired12, color=1];
        }
        subgraph cluster_2 {
                label="Classe 2";
                8 [colorscheme=paired12, color=2];
        }
        subgraph cluster_3 {
                label="Classe 3";
                9 [colorscheme=paired12, color=3];
        }
        subgraph cluster_4 {
                label="Classe 4";
                10 [colorscheme=paired12, color=4];
        }
        subgraph cluster_5 {
                label="Classe 5";
                5 [colorscheme=paired12, color=5];
                6 [colorscheme=paired12, color=5];
                7 [colorscheme=paired12, color=5];
        }
        1 -> 3 [color=gray];
        2 -> 1 [color=gray];
        2 -> 3 [color=gray];
        3 -> 4 [color=gray];
        3 -> 8 [color=gray];
        4 -> 2 [color=gray];
        4 -> 5 [color=gray];
        4 -> 6 [color=gray];
        5 -> 6 [color=gray];
        6 -> 7 [color=gray];
        7 -> 5 [color=gray];
        8 -> 9 [color=gray];
        8 -> 10 [color=gray];
        9 -> 10 [color=gray];
}
```

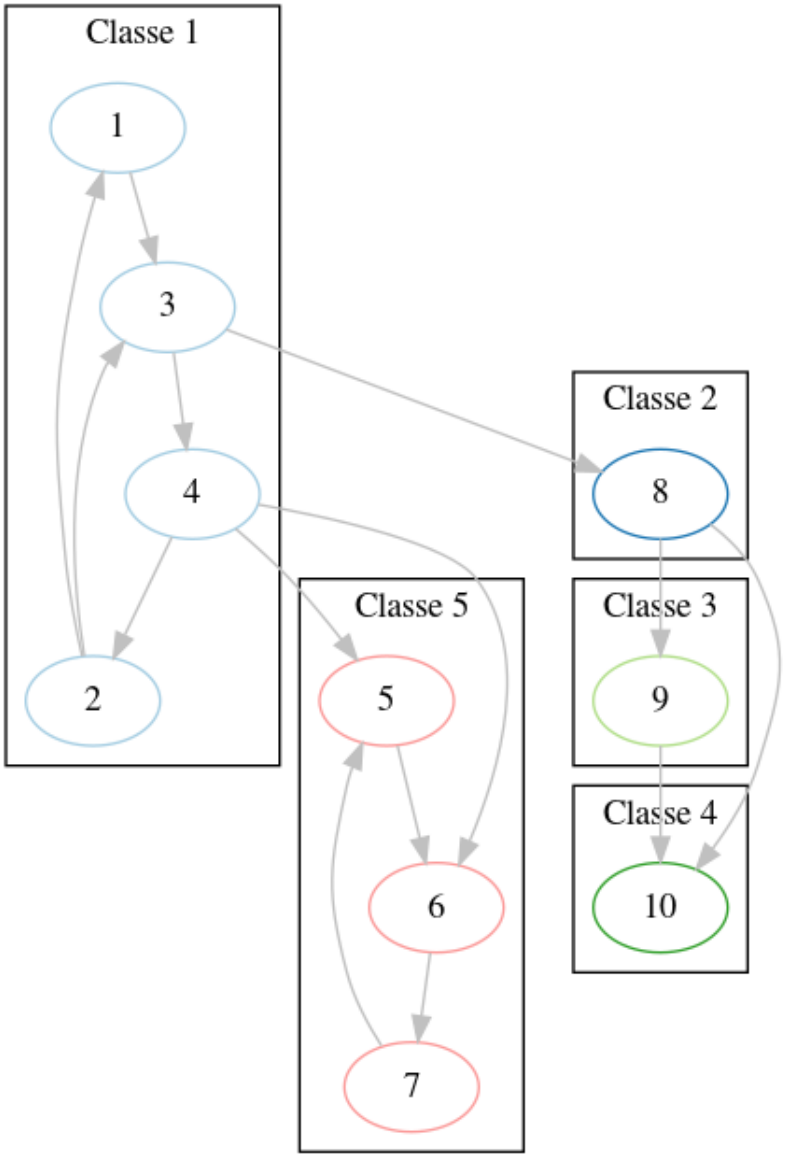

FIGURE 4 – Fichier digraph-1-cfc.dot

10.2 Graphe non orienté

Voici l'exécution de la fonction **analyseGraphe** sur le fichier **graph-k.txt** :

Affichage en console :

```
========== Analyse du fichier : graph-k.txt ==========
Importation du graphe du fichier graph-k.txt réussie !
Le fichier graph-k.dot a été créé.
Le graphe est sans pont.
Détermination d'une orientation forte...
Orientation forte déterminée !
Le fichier graph-k-ort.dot a été créé.
Parcours en profondeur : 1 -> 2 -> 3 -> 6 -> 5 -> 4 -> 7 -> 8
Le fichier graph-k-dfs.dot montrant le parcours dfs a été créé.
Le graphe est connexe.
Détermination des classes fortement connexes :
Post-ordre : 8 -> 7 -> 4 -> 5 -> 6 -> 3 -> 2 -> 1
Numéro de classe des sommets :
Sommet 1 : 1
Sommet 2 : 1
Sommet 3 : 1
Sommet 4 : 1
Sommet 5 : 1
Sommet 6 : 1
Sommet 7 : 1
Sommet 8 : 1
Le fichier graph-k-cfc.dot montrant les composantes fortement connexes
a été créé.
========== Fin analyse fichier : graph-k.txt ==========
```

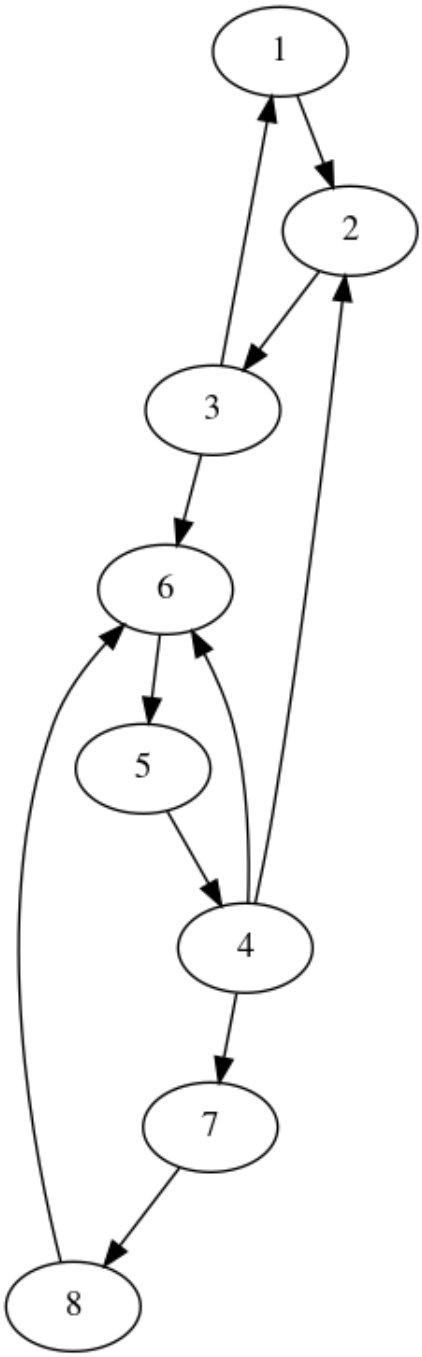

FIGURE 5 – Fichier graph-k-ort.dot