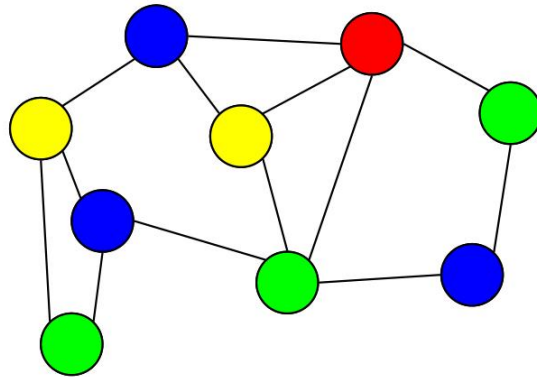


Rapport de Projet:



Théorie des Graphes

Etudiants:

Matt TAYLOR & Yoann SOCHAJ

Responsable de l'UE:

Philippe GABORIT



Université
de Limoges

Licence Informatique
2020-2021

Tables des matières:

1. Composantes fortement connexes d'un graphe	4
2. Algorithme de Dijkstra	6
a. Principe de l'algorithme de Dijkstra	
b. Structure de données choisie	
c. Cas d'usage	
3. Algorithme de Welsh-Powell	8
a. Principe de l'algorithme de Welsh-Powell	
b. Structure de données choisie	
c. Cas d'usage	
4. Conclusion	10



Pour les trois exercices nous avons décidé de définir nos **graphes** dans un **fichier texte** (.txt). Voici un exemple simple pour voir à quoi cela ressemble:

- La première ligne contient le **nom des sommets**.
- Les lignes suivantes représentent toutes les **arêtes du graphes**.

ie: 0 3 signifie qu'il existe une arête entre le sommet 0 et le sommet 3. Pour l'exercice 1 sur les composantes fortement connexes on distingue 0 3 et 3 0, dans un cas on part du sommet 0 pour aller au sommet 3 et dans l'autre cas l'inverse (**graphe orienté**).

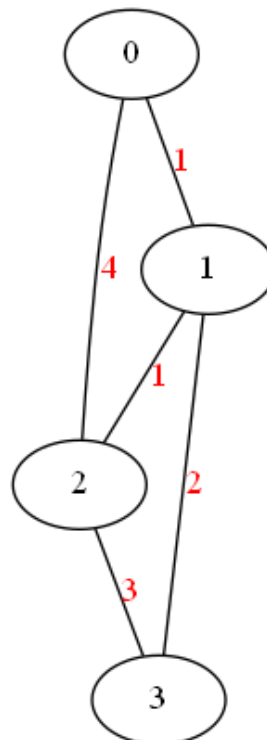
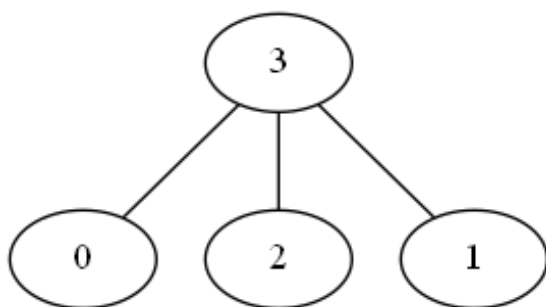
graphe1 - Notepad				
File	Edit	Format	Vi	
0	1	2	3	
0	3			
1	3			
2	3			

Attention: pour l'algorithme de **Dijkstra** il est nécessaire d'avoir le **coût des arêtes**. Ces fichiers sont représentés de la manière suivante:

*ie: la troisième ligne indique une arête entre le **sommet 1** et le **sommet 3** avec un **coût de 2**.*

graphe_cout2				
File	Edit	Forma		
0	1	2	3	
0	1	1		
1	2	3		
0	4	2		
1	1	2		
2	3	3		

Voici à quoi ressemble ces graphes:



1. Composantes fortement connexes d'un graphe

Pour déterminer les composantes fortement connexes d'un graphe il faut d'abord déterminer la matrice d'adjacence du graphe. Une fois établi, il faut calculer la matrice du graphe transitif. Pour repérer les composantes fortement connexes il suffit de regarder les colonnes identiques de cette matrice.

Prenons un exemple: on remarque que les colonnes 0 et 1 sont distinctes alors que les colonnes 2, 3 et 4 sont identiques. Les composantes fortement connexes sont donc: {0}, {1} et {2, 3, 4}.

```
Transitive closure:
[
[1, 1, 1, 1, 1]
[0, 1, 1, 1, 1]
[0, 0, 1, 1, 1]
[0, 0, 1, 1, 1]
[0, 0, 1, 1, 1]
]
```

Pour résoudre cet exercice nous avons implémenté une classe **Node** (sommet), une classe **Link** (arête) avec comme attributs un sommet de départ et un sommet d'arrivée. La troisième classe est la classe **Graph** qui permet de relier les deux classes au-dessus.

Nous avons mis en place plusieurs méthodes pour simplifier les étapes du calcul des composantes fortement connexes. Tout d'abord une méthode ***adjMatrix()*** qui retourne la matrice d'adjacence du graphe, ensuite une méthode ***transitiveClosure()*** qui retourne la matrice (reach-ability matrix) du graphe transitif (image ci-dessus).

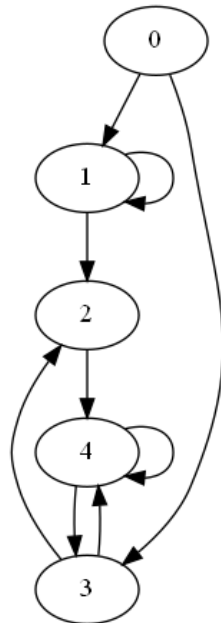


c. Cas d'usage

Voici deux cas d'usage de notre algorithme qui détecte les composantes fortement connexes d'un graphe:

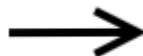
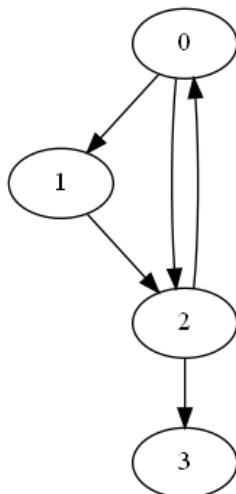
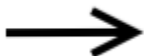
graphe4.txt est le graphe de l'exercice 2 du TD3, **graphe5.txt** est un graphe que nous avons choisi:

graphe4 -
File Edit F
0 1 2 3 4
0 1
0 3
1 1
1 2
2 4
3 2
3 4
4 3
4 4



```
E:\VMshared\UNIVERSITY\L3\S6\Graphes\Projet>ex1.py
Enter file name: (without .txt)
graphe4
File read succesfully: graphe4.txt
Number of nodes: 5
Number of links: 9
Directed matrix:
[
[0, 1, 0, 1, 0]
[0, 1, 1, 0, 0]
[0, 0, 0, 0, 1]
[0, 0, 1, 0, 1]
[0, 0, 0, 1, 1]
]
Transitive closure:
[
[1, 1, 1, 1, 1]
[0, 1, 1, 1, 1]
[0, 0, 1, 1, 1]
[0, 0, 1, 1, 1]
[0, 0, 1, 1, 1]
]
Strongly connected components:
[0] [1] [2, 3, 4]
```

graphe5
File Edit
0 1 2 3
0 1
0 2
1 2
2 0
2 3



```
E:\VMshared\UNIVERSITY\L3\S6\Graphes\Projet>ex1.py
Enter file name: (without .txt)
graphe5
File read succesfully: graphe5.txt
Number of nodes: 4
Number of links: 5
Directed matrix:
[
[0, 1, 1, 0]
[0, 0, 1, 0]
[1, 0, 0, 1]
[0, 0, 0, 0]
]
Transitive closure:
[
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[0, 0, 0, 1]
]
Strongly connected components:
[0, 1, 2] [3]
```



2. Algorithme de Dijkstra

a. Principe de l'algorithme de Dijkstra

Le principe de l'algorithme de Dijkstra est de trouver le **chemin le plus court** entre deux sommets du graphe.

L'algorithme de Dijkstra se déroule en plusieurs étapes:

1. Créer une liste ***distance[]***, une liste ***smallestTree[]*** et une liste ***path[]*** tous les trois de taille du nombre de sommets. Initialiser tous les indices de ***distance*** à + l'infini à part l'indice du sommet de départ. Initialiser tous les indices de ***smallestTree*** à **False**. Initialiser tous les indices de ***path*** à -1.
2. Tant que ***smallestTree*** n'est pas rempli de True:
Trouver le sommet (adjacent) qui se trouve à la distance minimale du sommet de départ. "Marquer" ce sommet (S) en mettant son indice à True dans ***smallestTree***.
3. Parcourir tous les sommets (k) et vérifier que le sommet (S) et (k) sont adjacent (c'est-à-dire la valeur dans la matrice d'adjacence > 0), si la **distance** de l'indice du sommet courant (k) est supérieure à la **somme** de la **distance** du sommet marqué précédemment (S) et le **coût** de l'arête entre le sommet courant (k) et le sommet (S):
 - mettre à jour la **distance**
 - affecter à l'indice (k) de **path** le sommet (S).
4. Retourner la liste des distances et le chemin (path).

b. Structure de données choisie

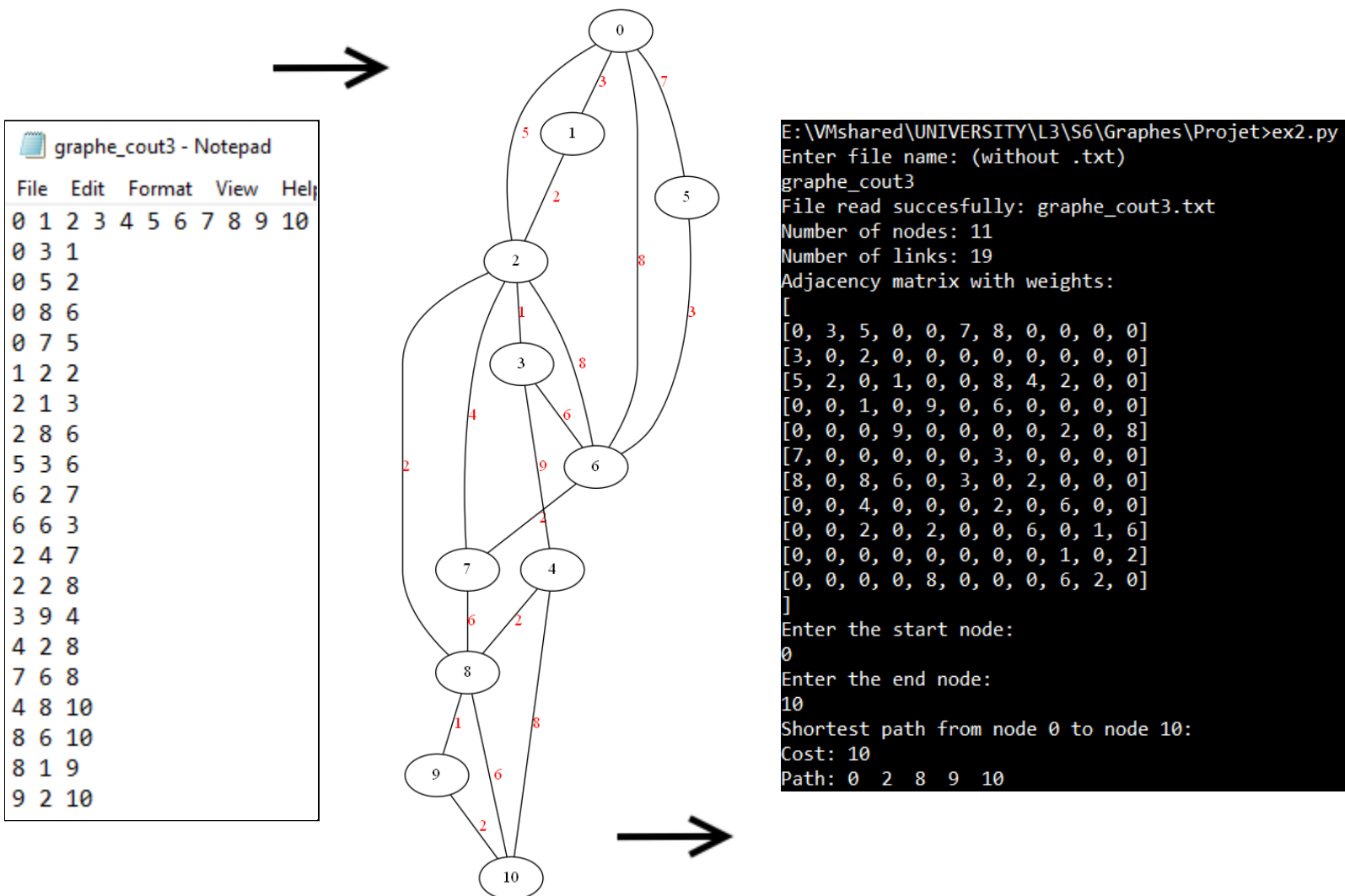
La structure de données utilisée pour cet algorithme est similaire à celle du premier exercice avec quelques modifications. La classe **Link** (arête) possède maintenant un attribut **weight** (coût de l'arête). Nous avons aussi implémenté une méthode ***showPath()*** qui permet de visualiser le chemin le plus court entre les deux sommets passés en paramètre de l'algorithme de Dijkstra.



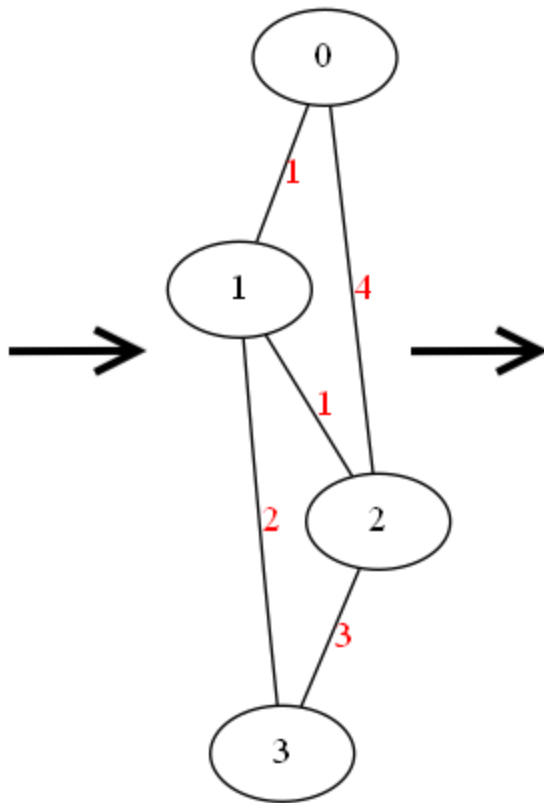
c. Cas d'usage

Dans notre dossier nous avons 3 graphes sous forme de fichier texte pour tester l'algorithme de Dijkstra (arêtes avec un coût). Nous allons vous montrer l'exécution de seulement 2 graphes ici:

graphe_cout3.txt est le graphe de l'exercice 1 du TD5 (calcul des plus courts chemins), **graphe_cout2.txt** est un graphe que nous avons choisi:



graphe_cout2
File Edit Format
0 1 2 3
0 1 1
1 2 3
0 4 2
1 1 2
2 3 3



```

E:\VMshared\UNIVERSITY\L3\S6\Graphes\Projet>ex2.py
Enter file name: (without .txt)
graphe_cout2
File read succesfully: graphe_cout2.txt
Number of nodes: 4
Number of links: 5
Adjacency matrix with weights:
[
[0, 1, 4, 0]
[1, 0, 1, 2]
[4, 1, 0, 3]
[0, 2, 3, 0]
]
Enter the start node:
0
Enter the end node:
3
Shortest path from node 0 to node 3:
Cost: 3
Path: 0 1 3
  
```



3. Algorithme de Welsh-Powell

a. Principe de l'algorithme de Welsh-Powell

Le principe de l'algorithme de Welsh-Powell est de colorier les sommets d'un graphe. Si deux sommets sont adjacents (reliés par une arête), ils sont colorés d'une différente couleur.

L'algorithme de Welsh-Powell se déroule en plusieurs étapes:

1. Déterminer le **degré** de chaque sommet.
2. **Ranger** les sommets par ordre de **degré décroissant** (dans certains cas plusieurs possibilités).
3. Attribuer au premier sommet (A) de la liste une **couleur**.
4. Suivre la liste en attribuant la **même couleur** au premier sommet (B) qui ne soit **pas adjacent** à (A).
5. Suivre (si possible) la liste jusqu'au prochain sommet (C) qui ne soit adjacent ni à (A) ni à (B).
6. Continuer jusqu'à ce que la liste soit finie.
7. **Répéter** les opérations 3 à 7.
8. Continuer jusqu'à avoir **colorié tous les sommets**.

b. Structure de données choisie

Nous avons utilisé les mêmes classes que l'exercice 1 en apportant des modifications. Une classe **Node** (sommet) avec en plus un attribut **degré** ainsi qu'un attribut **couleur**, une classe **Link** (arête), une classe **Graph** en attributs le nombre de sommets, le nombre d'arêtes, une liste d'arêtes et une liste de sommets. De plus, nous avons implémenté une classe **Color** avec trois attributs (r, g, et b).


Nous avons également implémenté une visualisation du graphe colorié dans notre programme avec **pydot**. Cette partie du code est commentée par défaut. Si vous voulez l'utiliser il faut avoir la librairie **pydot** sur votre machine (**pip install pydot**). Vous trouverez ci-joint dans le dossier une **vidéo** qui montre l'exécution de cette partie (la création du fichier **.png** avec le graphe colorié à partir du fichier **.txt**).



c. Cas d'usage

Dans notre dossier nous avons 6 graphes sous forme de fichier texte. Nous allons vous montrer l'exécution de seulement 2 graphes ici:

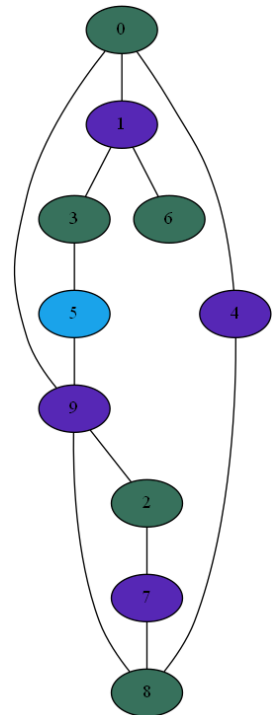
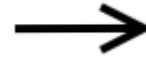
graphe6.txt est le graphe de l'exercice 1 du TD6 (exercice sur les enclos du zoo), **graphe5.txt** est un graphe que nous avons choisi:


 graphe6 - Notepad

File	Edit	Format	View						
0	1	2	3	4	5	6	7	8	9
0	1								
1	3								
1	6								
3	5								
5	9								
0	9								
9	2								
2	7								
7	8								
9	8								
0	4								
4	8								

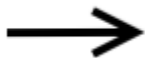


```
E:\VMshared\UNIVERSITY\L3\S6\Graphes\Projet>ex3.py
Enter file name: (without .txt)
graphe6
File read succesfully: graphe6.txt
Number of nodes: 10
Number of links: 12
Node: 9 Color: 158 152 220
Node: 0 Color: 255 167 171
Node: 1 Color: 158 152 220
Node: 8 Color: 255 167 171
Node: 2 Color: 255 167 171
Node: 3 Color: 255 167 171
Node: 4 Color: 158 152 220
Node: 5 Color: 218 141 197
Node: 7 Color: 158 152 220
Node: 6 Color: 255 167 171
The chromatic number of this graph is: 3
```

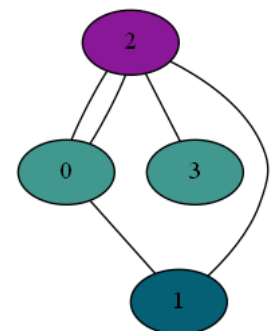


 graphe5

	File	Edit	
0	1	2	3
0	1		
0	2		
1	2		
2	0		
2	3		



```
E:\VMshared\UNIVERSITY\L3\S6\Graphes\Projet>ex3.py
Enter file name: (without .txt)
graphe5
File read succesfully: graphe5.txt
Number of nodes: 4
Number of links: 5
Node: 2 Color: 126 187 153
Node: 0 Color: 236 243 161
Node: 1 Color: 207 172 214
Node: 3 Color: 236 243 161
The chromatic number of this graph is: 3
```



4. Conclusion

La théorie des graphes est un domaine qui connaît beaucoup d'applications dans la vie réelle; l'algorithme de **Dijkstra**: la recherche du chemin le plus court est utilisé par Google Maps par exemple, l'algorithme de **Welsh-Powell** est utilisé pour le coloriage de carte pour permettre une meilleure visualisation des frontières d'un pays ou d'un département, etc.

Pour conclure nous avons vraiment adoré faire ce projet. Cela nous a permis de mieux comprendre ces algorithmes et leur usage ainsi que de découvrir de nouveaux outils de visualisation de graphe comme **pydot** et **graphviz**.

