Mini-Projet :

Théorie des graphes & Applications

Encadré par :

Pr. GHADI Abderrahim

Réalisé par :

BENTAJ HAMYANI Ahmed

EL METIOUI Fouad

Table des matières

Définition

Le graphe est un ensemble de points dans un plan ou dans un espace et un ensemble de segment de droite de courbe dont chacun joint deux points ou se joint à lui-même. Un graphe G = (V(G), E(G)) composé de deux ensembles finies. V(G), l'ensemble des sommets du graphe noté V, qui est un ensemble non vide d'éléments appelés sommets et E(G), l'ensemble des arêtes de graphe, souvent noté juste E, qui est un ensemble éventuellement vide d'éléments appelés arêtes. Chaque arête e dans E a un ensemble d'un ou deux sommets qui lui sont associés et qui sont appelés ses extrémités.

Pourquoi Théorie des Graphes ?

Les graphes peuvent être utilisés pour modéliser de nombreux types de relations et de processus dans les systèmes physiques, biologiques, sociaux et d'information …

Notamment tous les problèmes pratiques peuvent être représentés par des graphes. Insistant sur leur application aux systèmes du monde réel, le terme réseau est défini comme un graphe dans lequel des attributs (par exemple des noms) sont associés aux nœuds et/ou aux arêtes.

En informatique, les graphes sont utilisés pour représenter les réseaux de communication, l'organisation des données, les dispositifs de calcul, le flux de calcul, etc.

Exemples d'utilisation

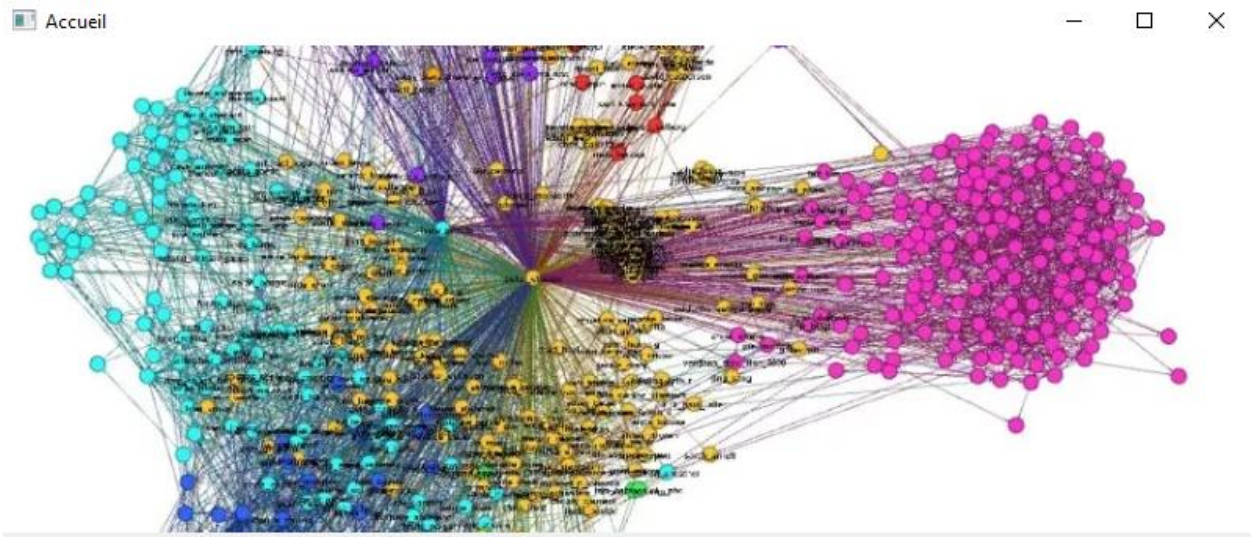
La structure des liens d'un site Web peut être représentée par un graphe orienté, dans lequel les sommets représentent des pages Web. Et les bords dirigés représentent des liens d'une page à une autre. Une approche similaire peut être adoptée pour les problèmes liés aux médias sociaux, aux voyages, à la biologie, à la conception de puces informatiques et à de nombreux autres domaines.

Introduction

Dans ce projet on a réalisé une application **Python** basé sur l'interface graphique **PyQt5,** son rôle est de tracer un graphe d'un type et ordre donné avec la bibliothèque **Networkx,** afficher ses caractéristiques et appliquer quelques algorithmes.

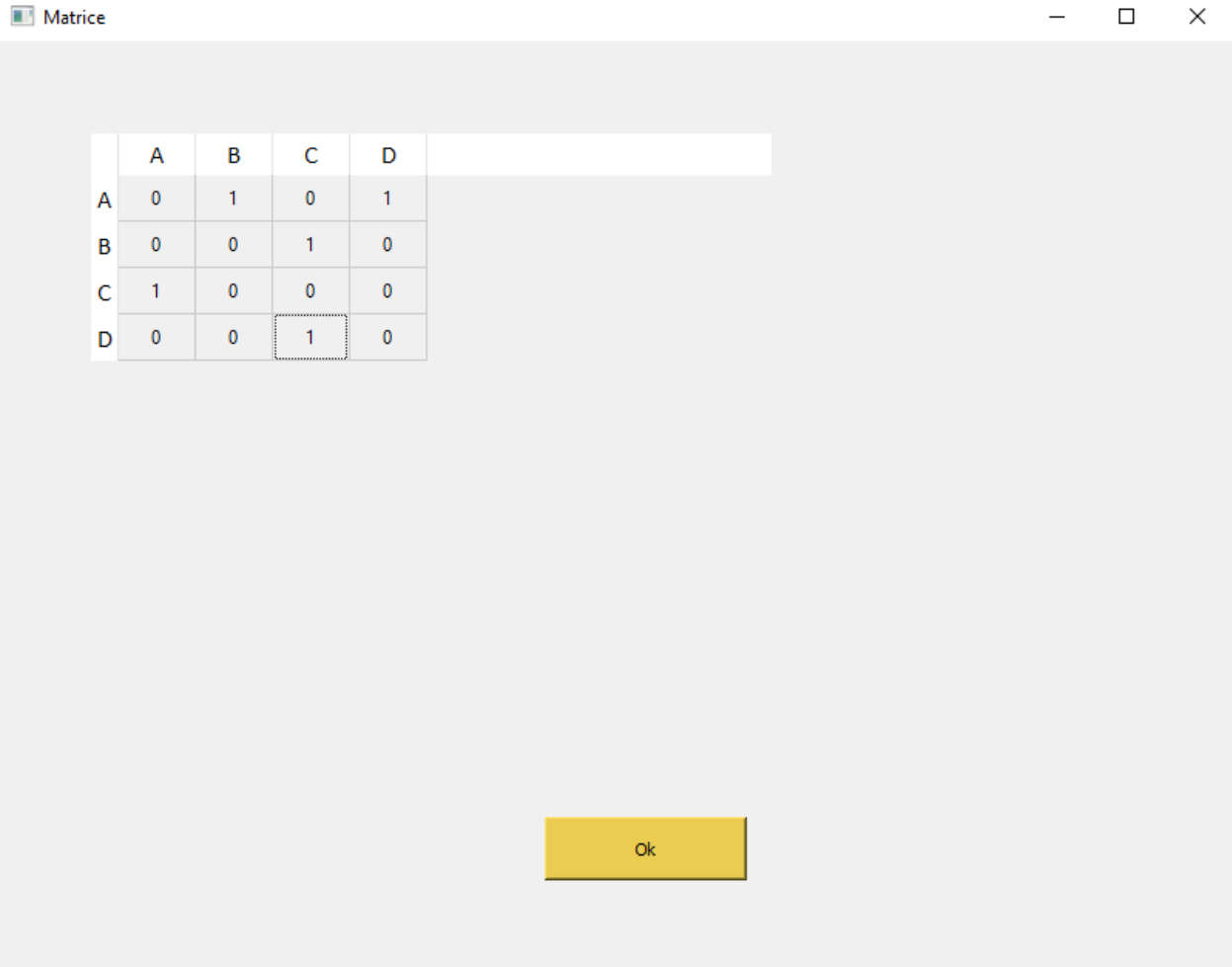
1 Première Page

Pour la première page on a demandé à l'utilisateur d'entrer le type du graphe **Non orienté et non pondéré**, **Orienté et non pondéré**, **Non orienté et pondéré** et **Orienté et pondéré**, et d'entrer l'ordre du graphe.


Accueil
Choisir le type de graphe ?
Non orienté et non pondéré
Non orienté et pondéré
Orienté et non pondéré
Orienté et pondéré
L'ordre du graphe :
4
OK

2 Remplissage de la matrice

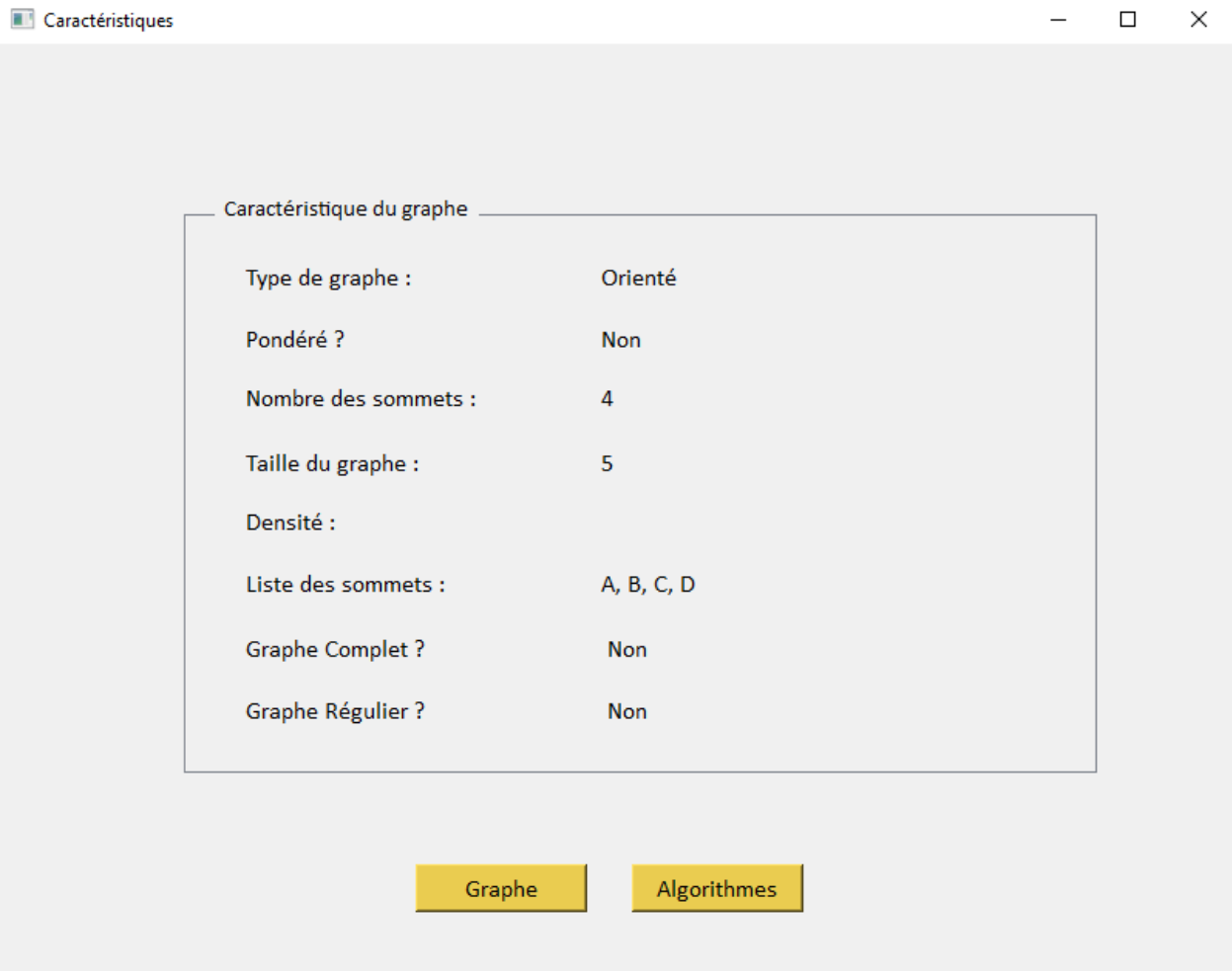
Après le saisie des informations sur le graphe et appuyer sur **OK**, une matrice s'affiche.



3 Caractéristique du graphe

Cette partie est dédiée pour afficher toutes les caractéristiques utiles pour un graphe, dont lequel on donne des informations sur le graphe afin de le comprend avant de l'utiliser par exemple pour résoudre un problème.

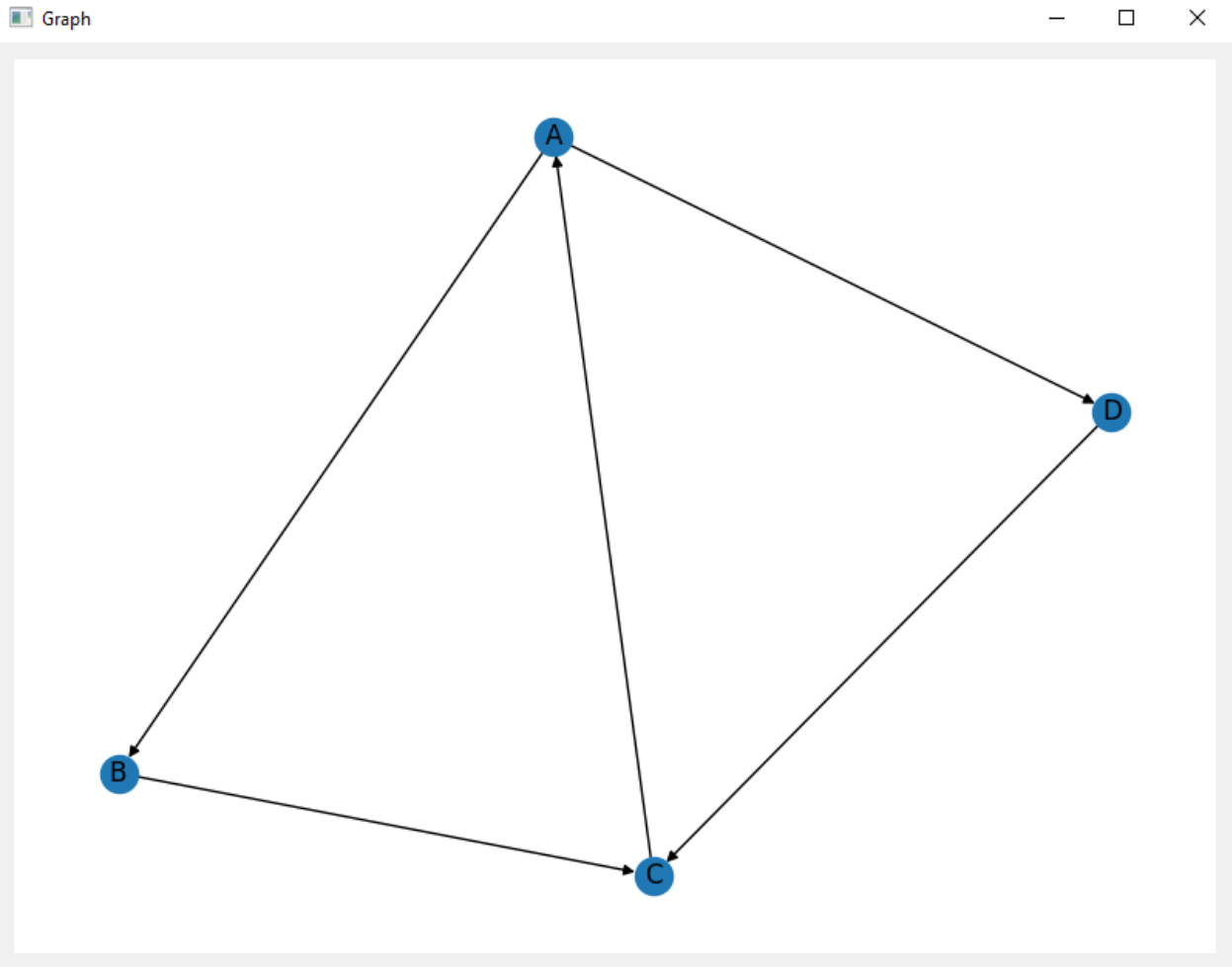
Voici le résultat de notre graphe:



Dans cette fenêtre on a mis l'accent sur plusieurs informations. Premièrement, le type de graphe pour dire si le graphe est **Oriente** ou **Non oriente**, on récupère cette valeur à travers le choix de l'utilisateur. De même pour le deuxième champ Pondéré, il possède deux possibilités **Oui/Non**. Pour le nombre de sommet et la taille de graphe on affiche successivement V (G), E(G). Concernant la densité, on a mis une condition pour qu'il soit affichable que pour le cas d'un graphe **Non oriente**, et on a utilisé cette relation **2|E| / |V|. (|V|-1)** pour la calculer. A propos du champ Graphe Complet, il y a deux possibilités soit il est complet (on affiche Oui) sinon on affiche Non, en se basant sur la relation **|E| = (|V|. (|V|-1)) /2**, de même pour le champ du Graphe Régulier, on affiche Oui si toutes les sommets on même degrés, on affiche Non sinon.

4 Graphe

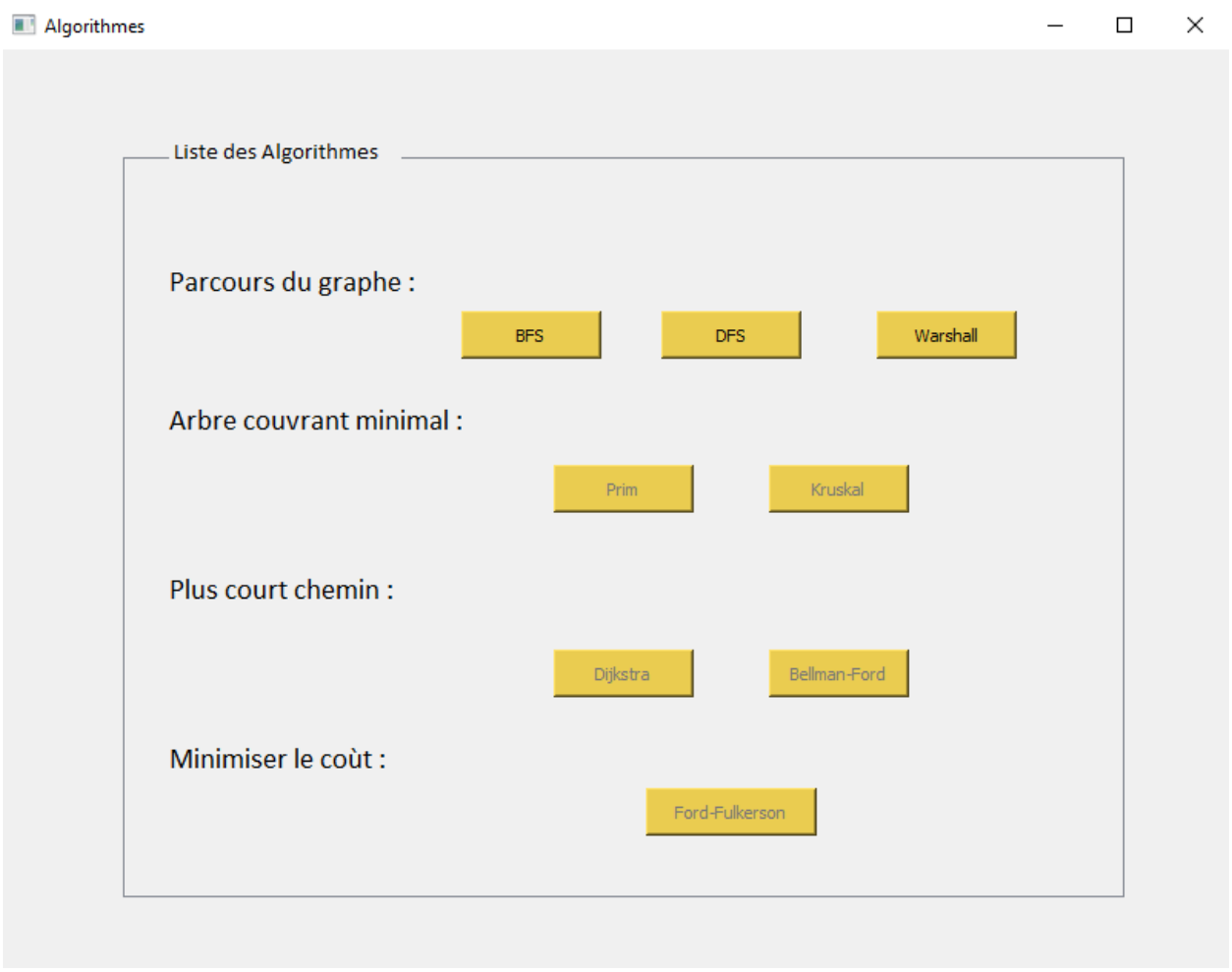
Si l'utilisateur veut voir le graphe il suffit de cliquer sur le bouton **Graphe.**



5 Algorithmes

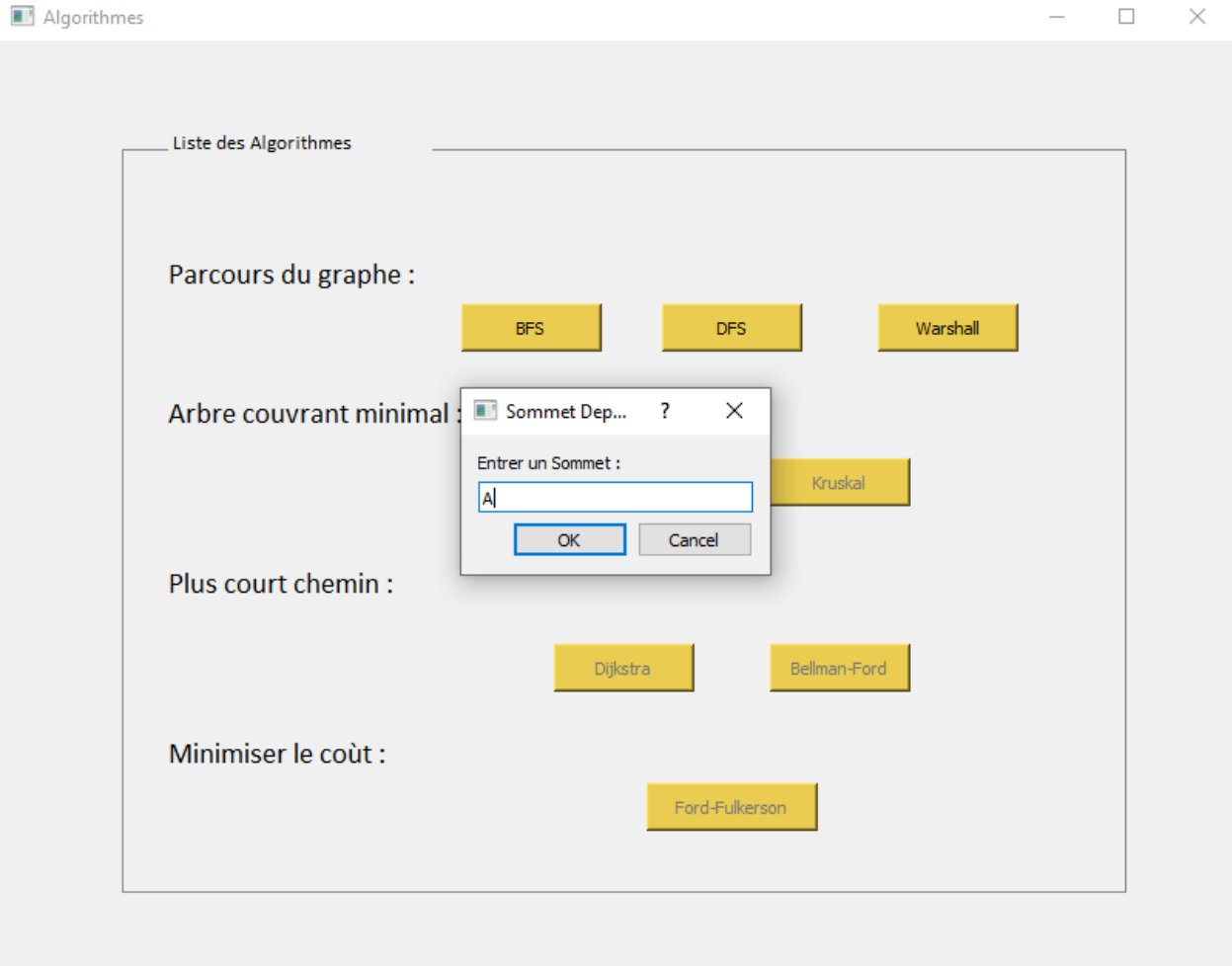
Pour la page des algorithmes il y a 8 algorithmes à la service d'utilisateur.

Pour cette graphe qui est de type **Orienté et non pondéré,** on ne peut pas appliquer quelques algorithmes pour cette raison les boutons de ces algorithmes sont désactivées.

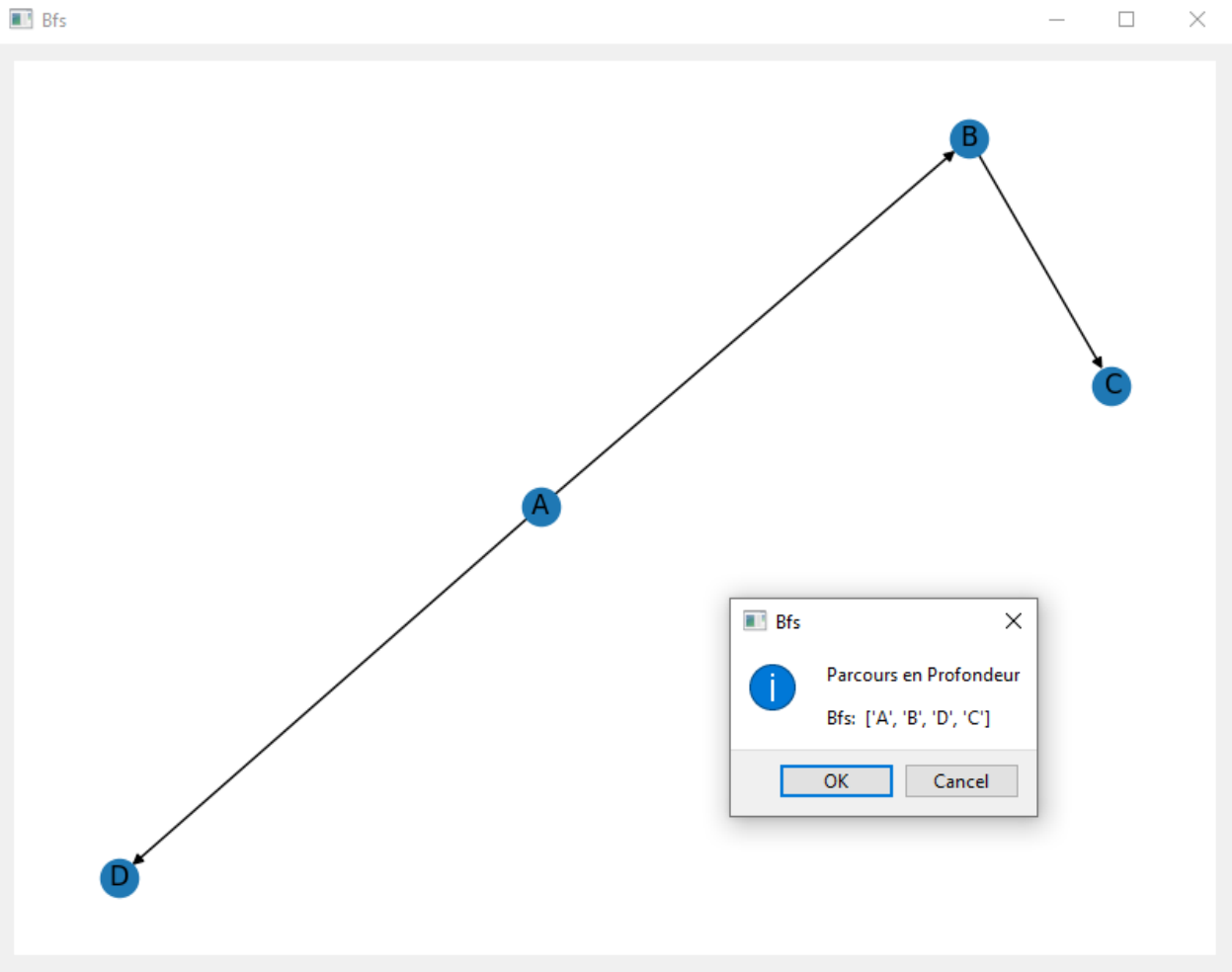

Algorithmes
Liste des Algorithmes
Parcours du graphe :
BFS
DFS
Warshall
Arbre couvrant minimal :
Prim
Kruskal
Plus court chemin :
Dijkstra
Bellman-Ford
Minimiser le coût :
Ford-Fulkerson

5.1 Breath First Search

En récupérant le sommet de départ avec une boîte de dialogue de saisie pour appliquer l'algorithme **BFS** à notre graphe.

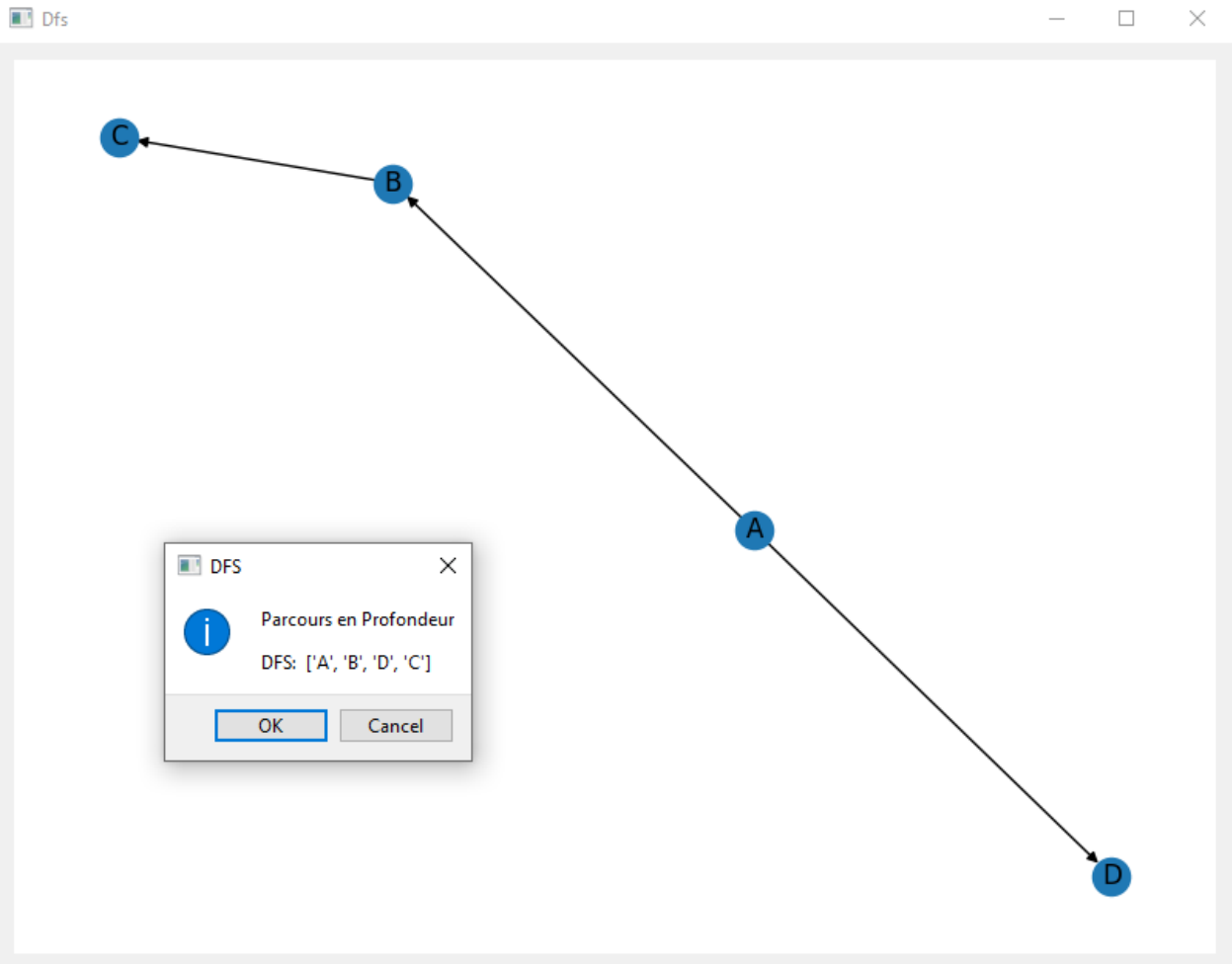


On affiche le résultat de l'algorithmes comme un graphe et une boite de message indique le parcours de notre graphe.


Bfs
B
C
A
D
Bfs
Parcours en Profondeur
Bfs: ['A', 'B', 'D', 'C']
OK
Cancel

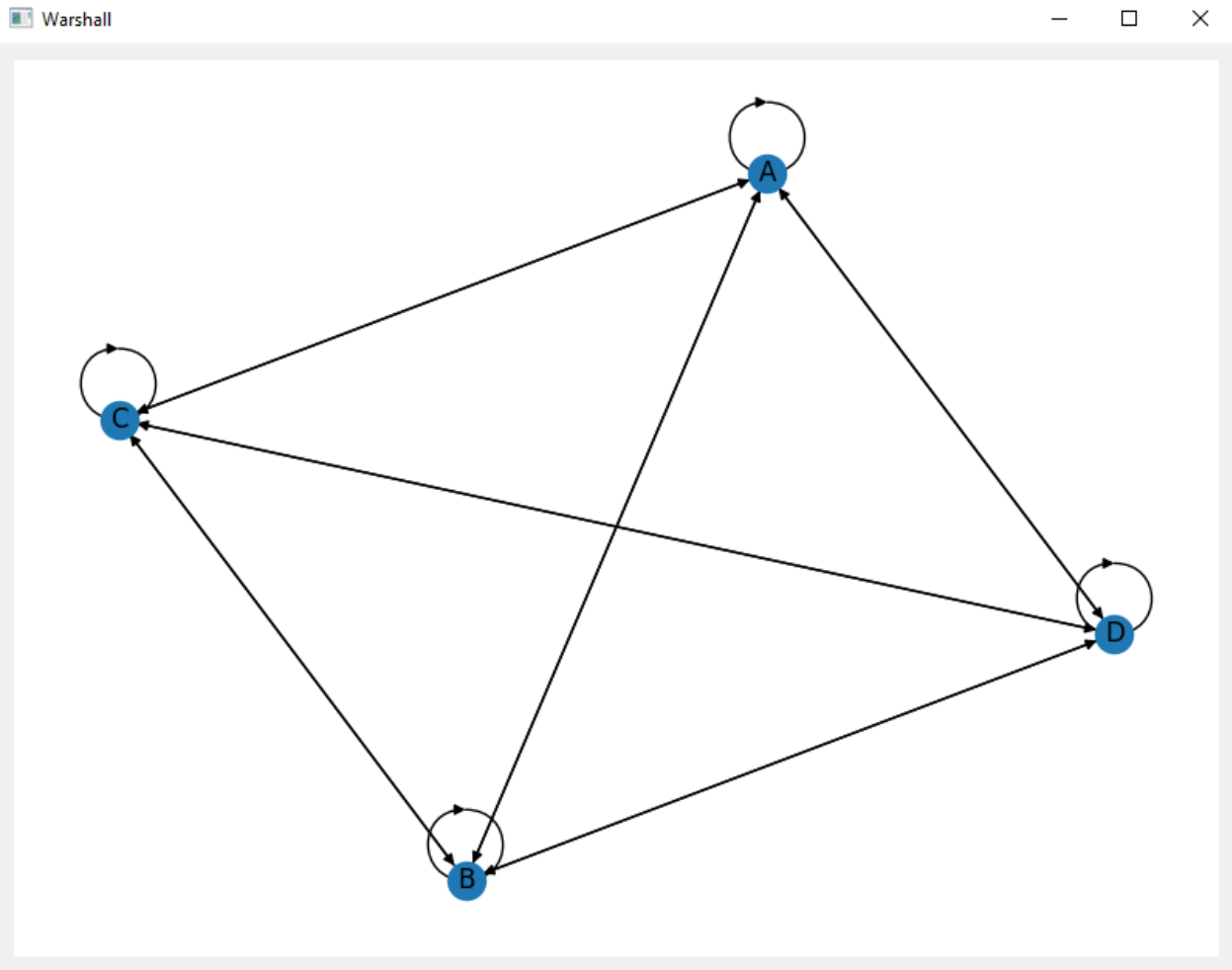
5.2 Depth First Search

La même chose pour **DFS**.



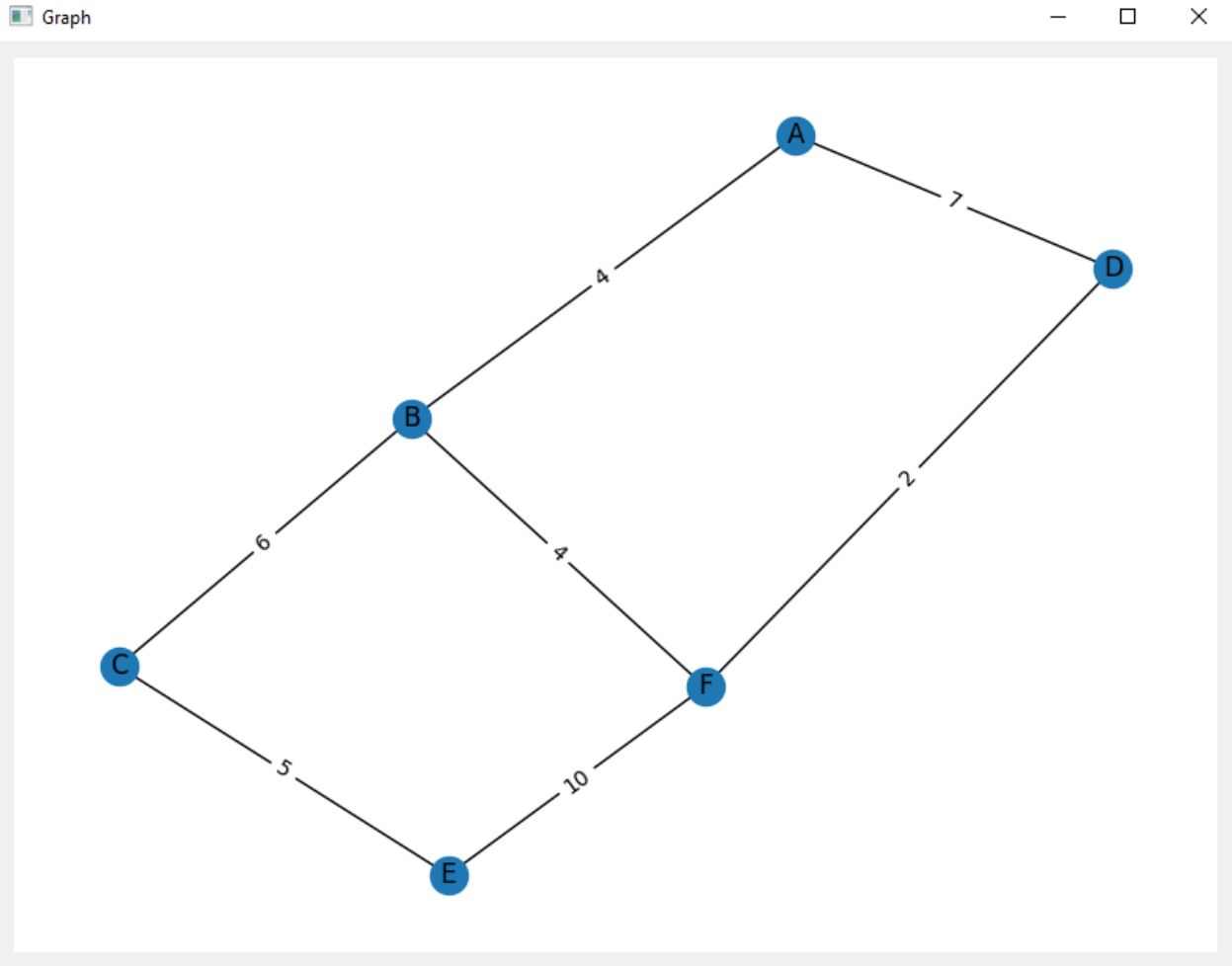
5.3 Warshall

Le résultat de l'algorithme de **Warshall** à notre graphe est le suivant.

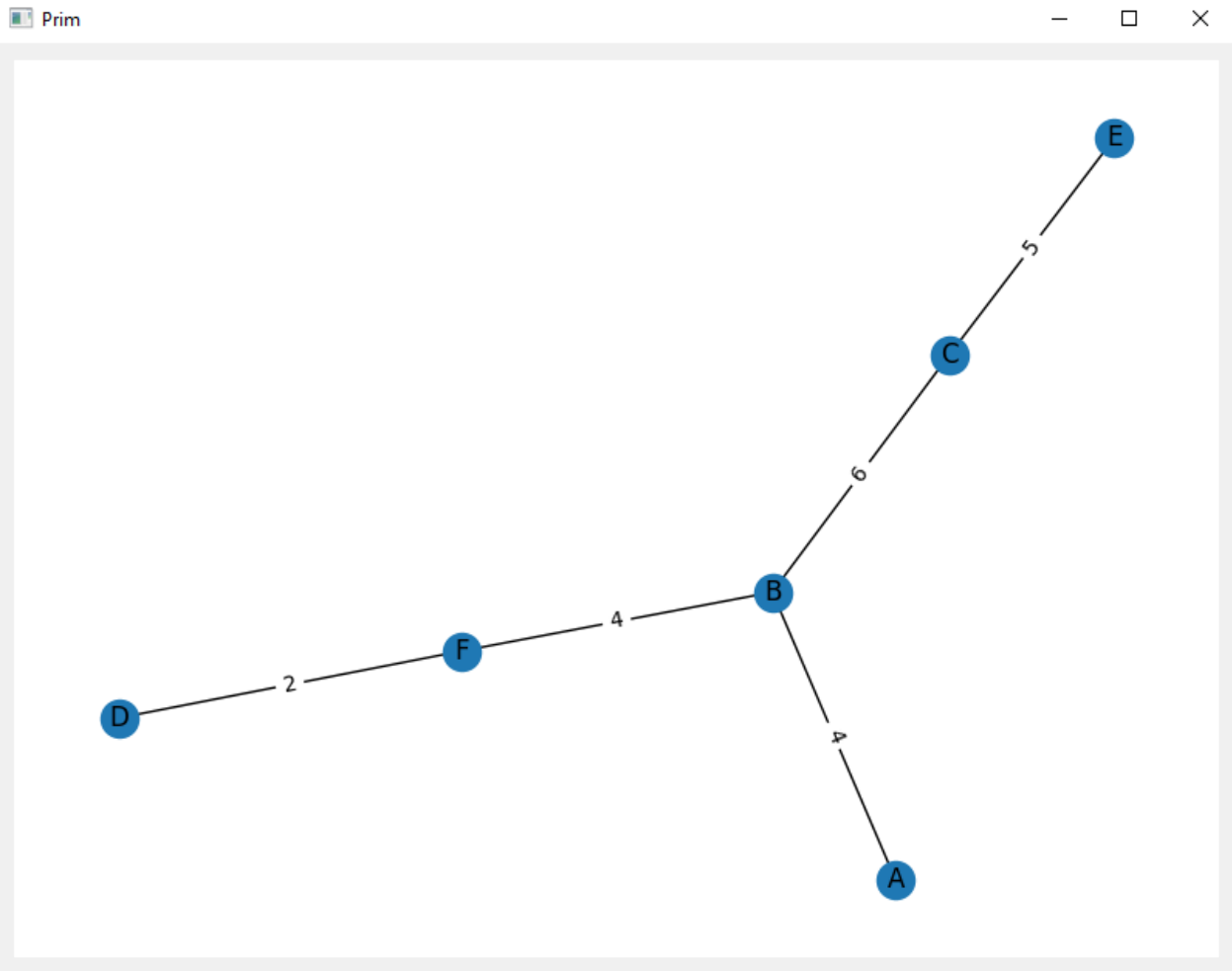


5.4 Prim

Pour appliquer l'algorithmes **Prim,** il faut que le graphe soit pondéré c'est pour cela on a changé le graphe par celui-ci.

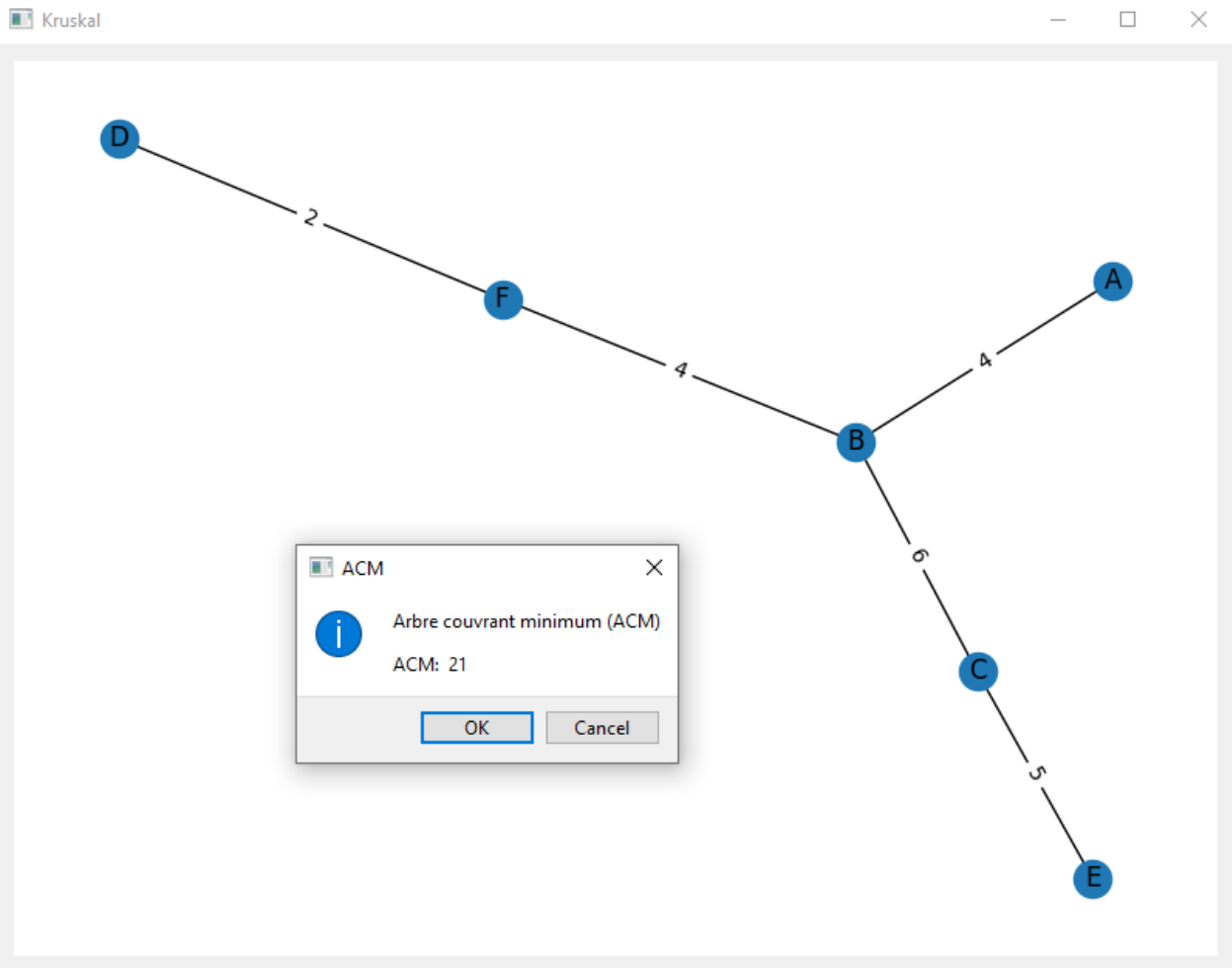


Le résultat de l’algorithme sur le nouveau graphe :



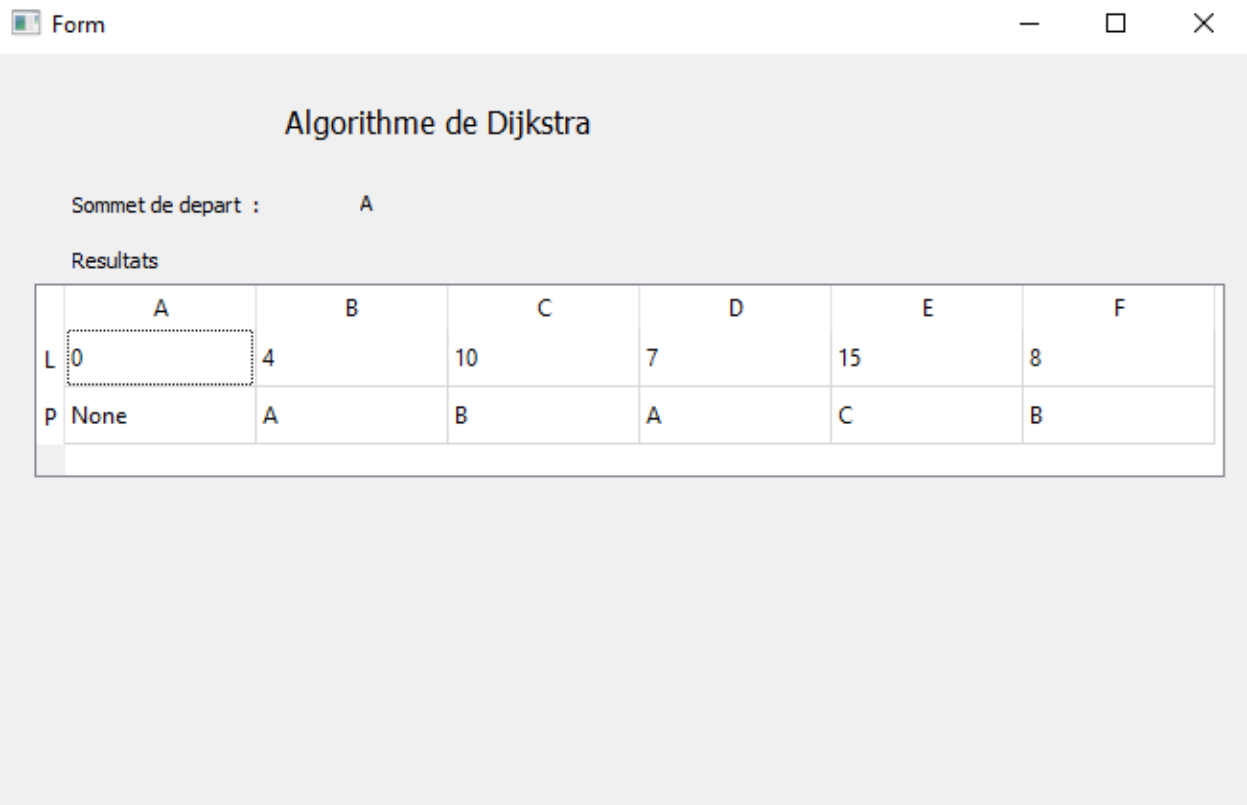
5.5 Kruskal

L’algorithme de **Kruskal** est un algorithme de recherche d'arbre recouvrant de poids minimum, dans un graphe non-orienté et pondéré. Il nous permet d’extraire un arbre avec un coût total minimum qu’on l’appelle ACM



5.6 Dijikstra

L'algorithme de **Dijkstra** sert à résoudre le problème du plus court chemin, pour cela en commence par un nœud et on cherche le plus court chemin vers les autres sommets en termes du coût.



	A	B	C	D	E	F
L	0	4	10	7	15	8
P	None	A	B	A	C	B

Ce tableau permet de Comprendre à la fois le cout pour aller vers tous les sommets du graphe et le chemin qu'il faut traverser en se basant sur les prédécesseurs.

Analyse des Couts : L

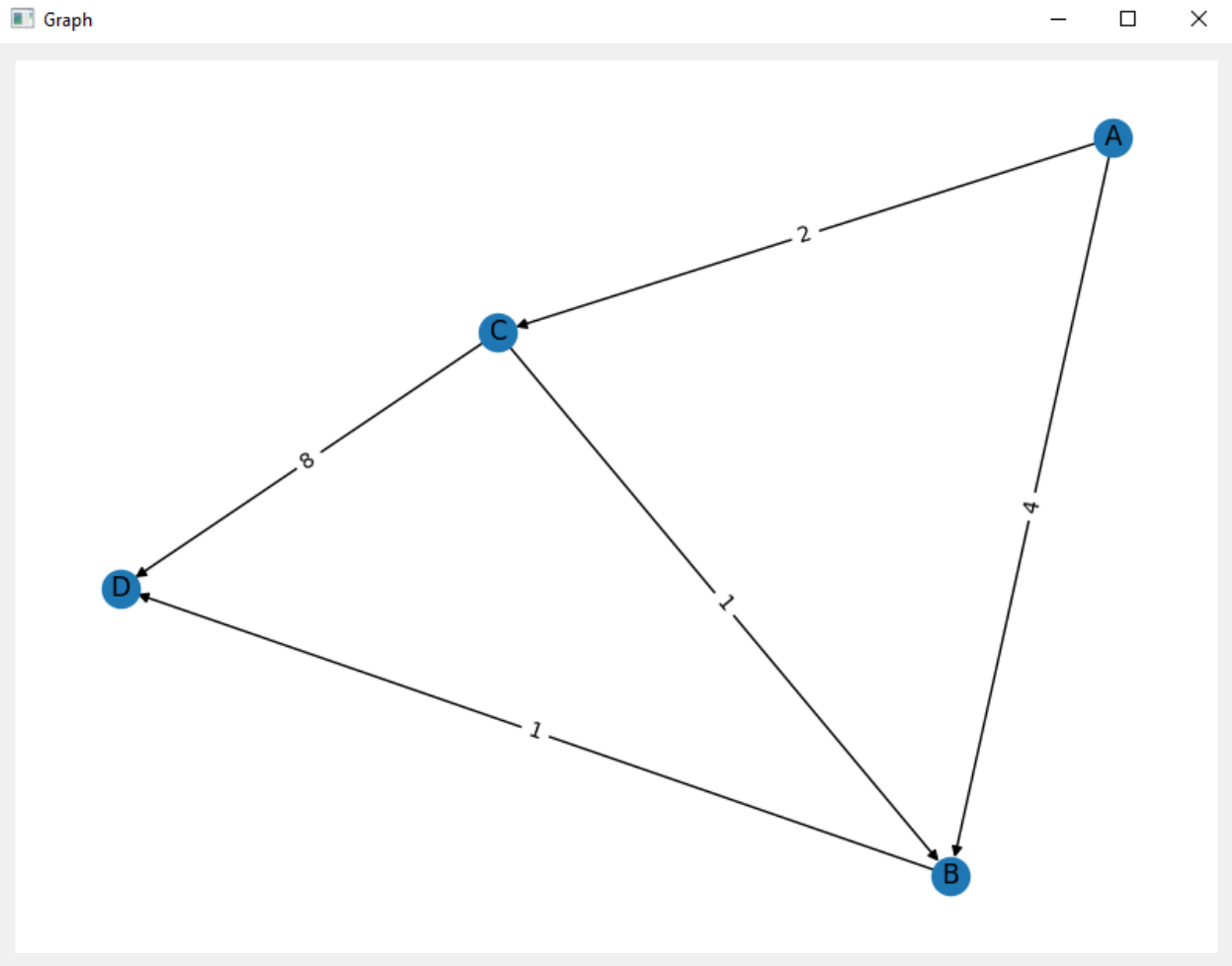
- Pour aller de A vers B le cout est : 4
- Pour aller de A vers C le cout est : 10
- Pour aller de A vers D le cout est : 7
- Pour aller de A vers E le cout est : 15
- Pour aller de A vers F le cout est : 8

Analyse des Chemins : P

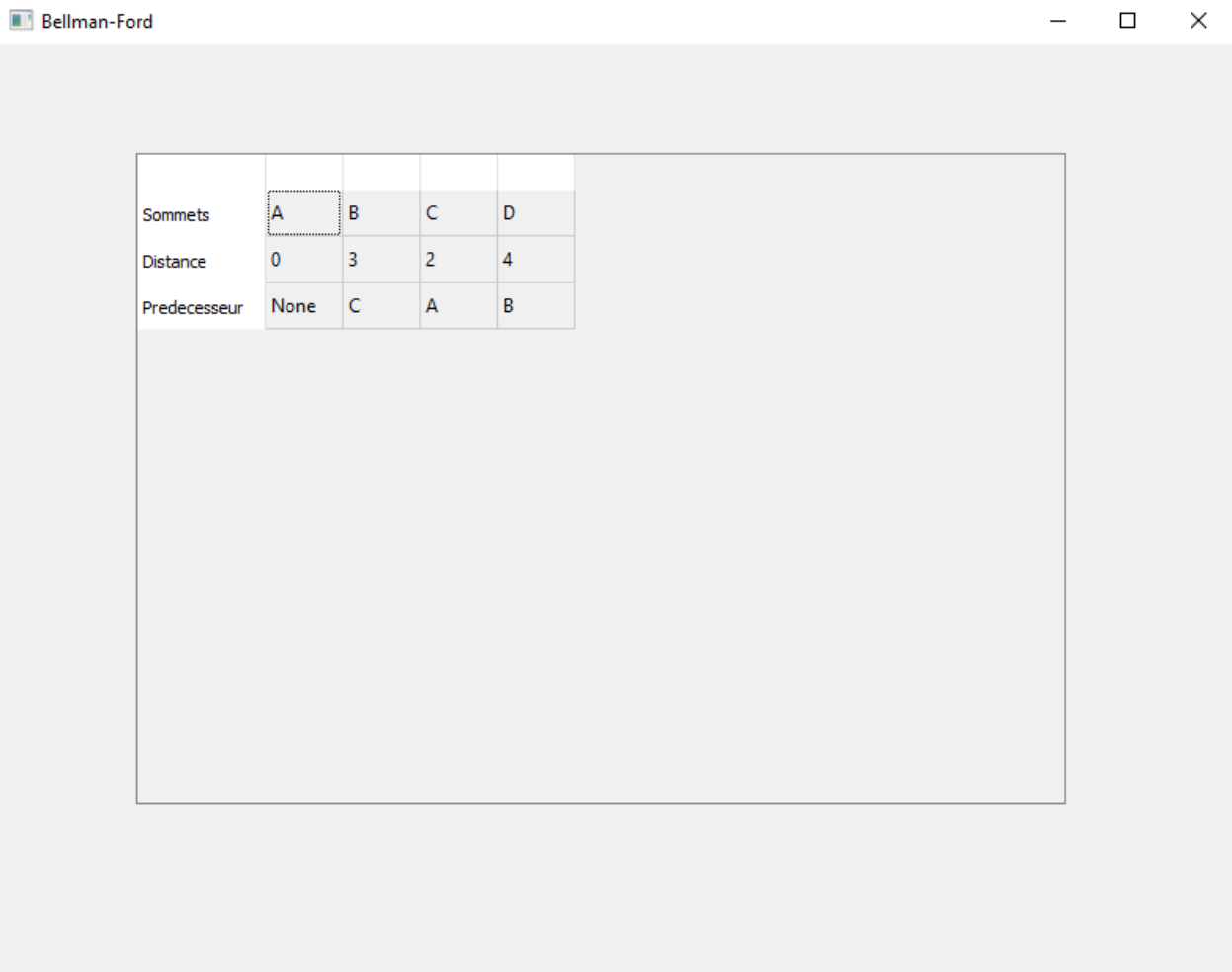
- A -> B
- A -> B->C
- A -> D
- A -> B->C->E
- A -> B->F

5.7 Bellman-Ford

L'algorithmes de **Bellman-Ford** est un algorithme qui calcule des plus courts chemins depuis un sommet source donné dans un graphe orienté pondéré.
Pour appliquer l'algorithmes on change les graphes précédents par le graphe celui-ci :



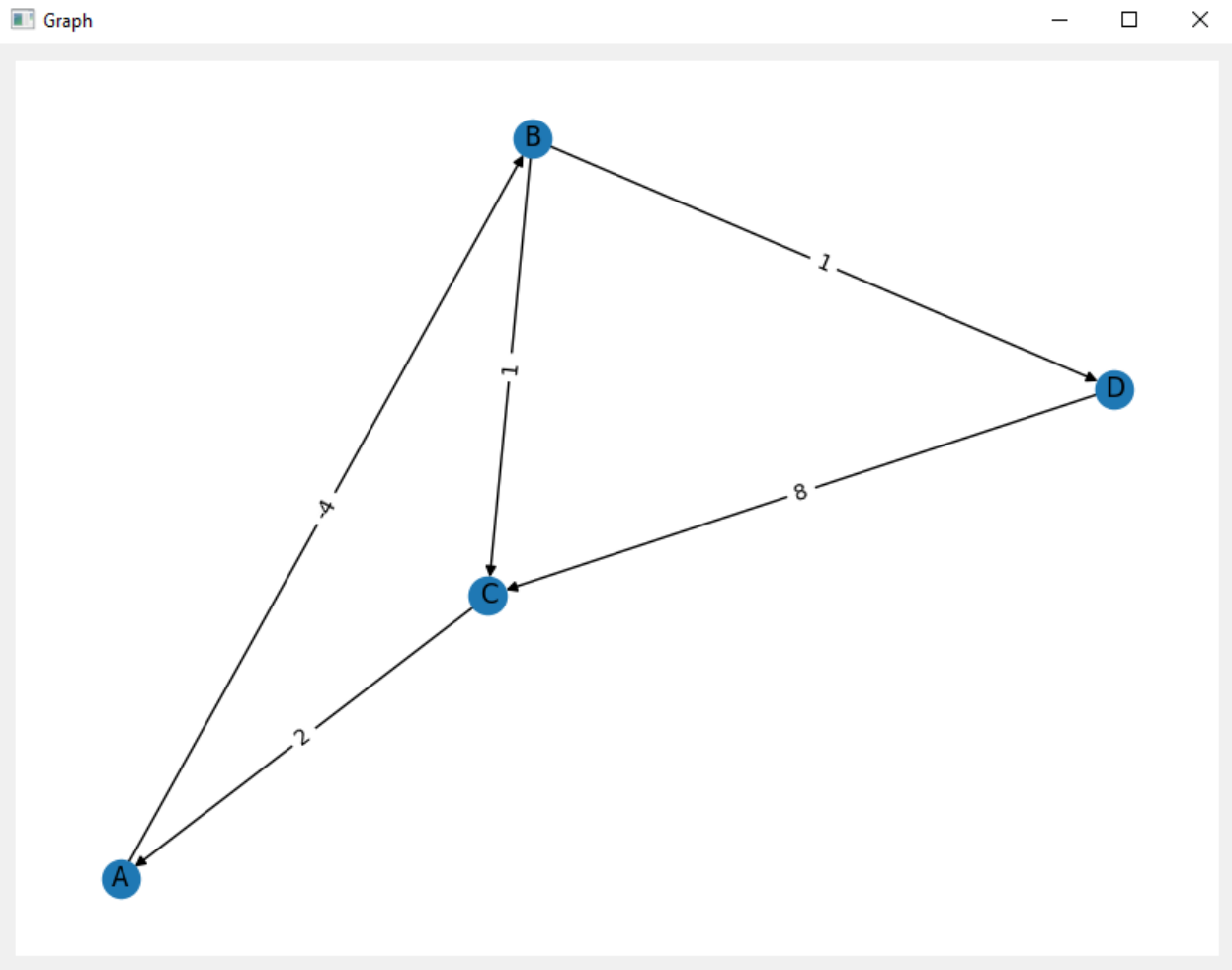
Résultat de l'algorithmes pour le sommet **A :**



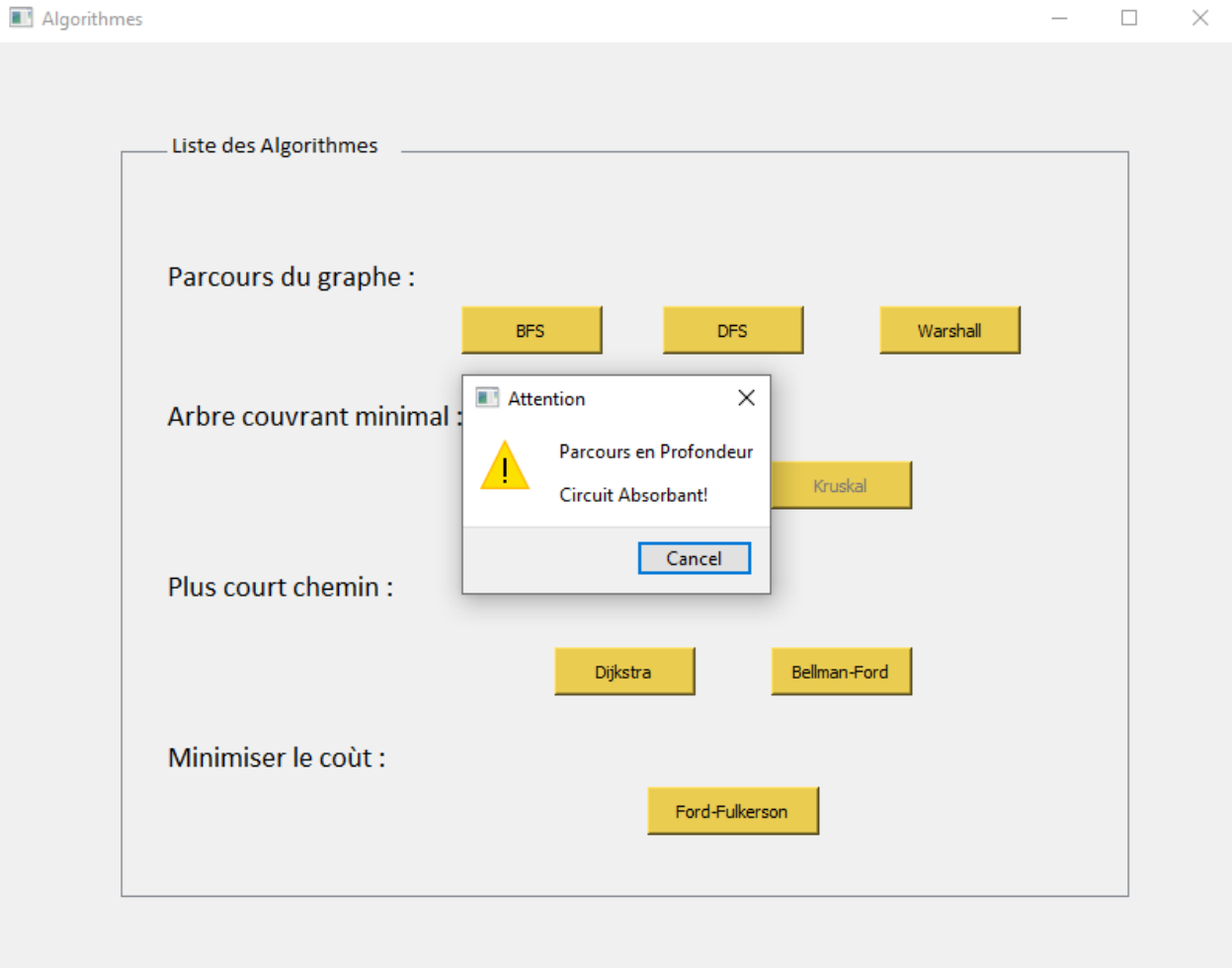
5.8 Bellman-Ford avec un circuit absorbant

Si l'algorithmes détecte un circuit absorbant une boite de message s'affiche avec le message **Attention ! Circuit Absorbant.**

Pour ce graphe qui contient un circuit absorbant :

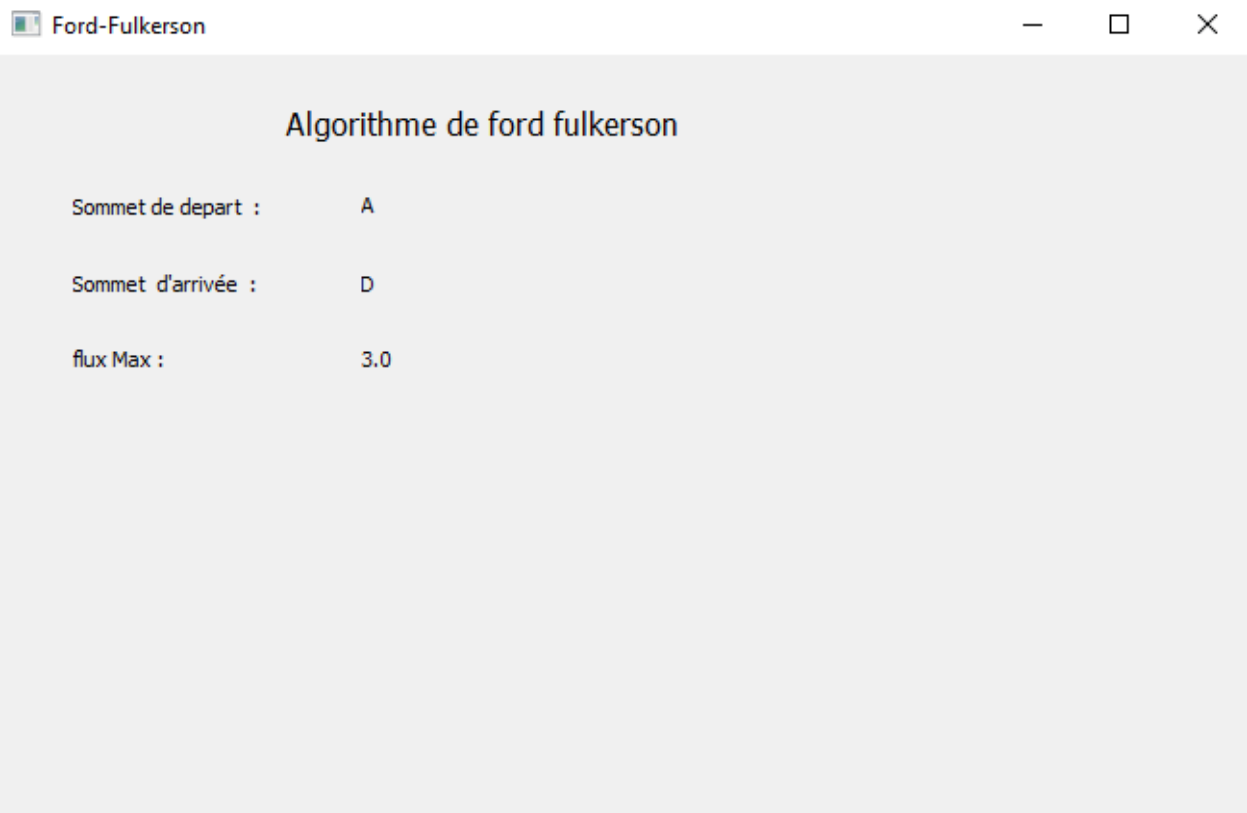


Résultat de l’algorithmes :



5.9 Ford-Fulkerson

L'algorithme de Ford-Fulkerson permet de résoudre le problème d'optimisation des coûts.
On récupère deux sommets (source et puit) depuis l'utilisateurs puis on affiche la valeur de Flux Max.



Pour le faire on a adapté notre graphe avec la bibliothèque **Networkx** pour nous pouvons utiliser la fonction maximum flow, puis on a ajouté toutes les capacités convenables pour chaque arc.

Conclusion

On souhaite avant tout remercier notre prof, Mr. GHADI Abderrahim professeur à la Faculté des Sciences et Techniques de Tanger, pour le temps qu'il a consacré à nous apporter les outils méthodologiques indispensables à la conduite de ce projet. Son exigence nous a grandement stimulé.

Le lien GitHub : https://github.com/ahmed-bentajhamyani/Graphical-interface-in-Python-Algorithms-on-graphs