Introduction à l'algorithmique **Hiver 2023**

IFT2125-6001 Évaluation (10%)

Devoir 3

Consignes générales : À faire en équipe de 2 personnes maximum. Les équipes se font sur Studium avec l'activité "Choix d'équipe pour le devoir 3". Si l'équipe d'enseignement doit vous placer en équipe sur Studium après la date limite, des points seront retirés de la note du devoir. Ne pas indiquer vos noms mais seulement vos matricules étudiants sur le devoir et dans le code. Le langage de programmation est Python 3. Les retards ne sont pas acceptés car la correction se fait en séance de TP.

À remettre :

- Un seul fichier PDF contenant toutes vos réponses écrites, vos analyses et commentaires pour le devoir
- Un fichier `controle.py`

1 Programmation dynamique - Achats de jouets (30 points)

Un magasin qui vend des jouets a publié son catalogue de jouet pour le prochain mois en notant les spéciaux sur des figurines. À chaque jour du mois, il peut y avoir soit une figurine en spécial ou aucun spécial du tout. Les prix de ces figurines en spécial sont connus d'avance car marqués dans le catalogue.

Un enfant est intéressé à acquérir le plus de figurines possibles pour sa collection. L'enfant reçoit 1$ par jour en début de journée comme allocation de ses parents. Il commence le mois à 0$. L'enfant soucieux de bien dépenser son argent, ne s'intéresse qu'aux figurines en spécial. Le problème est alors d'acquérir le nombre maximal de figurines.

Vous devez concevoir un algorithme de programmation dynamique pour résoudre ce problème et suggérer la séquence d'achats à faire à l'enfant. Si un tableau est utilisé, il faut identifier les axes, donner les conditions d'initialisation et la formule de récurrence.

Vous n'avez pas à programmer mais vous devez écrire un pseudo-code. Discutez brièvement de la complexité théorique.

Spécification :

- Programmation dynamique.
- Pas de retour en arrière, le magasin n'accepte pas les retours ou échanges pour les jouets en spécial.
- Pas d'achats à crédit ou en plusieurs versements ni d'emprunts.
- Chaque figurine est différente mais les prix peuvent être égaux.
- Certaines journées, il peut n'y avoir aucun spécial.
- Le prix de chaque figurine est inférieur ou égal au nombre de jours depuis le début du mois. Par exemple, le prix de la figurine en spécial du 3^{e} jour sera entre 1$ et 3$. Donc, il n'y a pas de figurine impossible à acheter pour l'enfant.
- À chaque jour, l'enfant reçoit 1$. Ensuite, il a le choix de ne rien acheter et passer à la journée suivante, ou d'acheter la figurine en spécial (s'il y a une figurine en spécial cette journée là et qu'il a assez d'argent pour l'acquérir).

Exemple 1 :

Journée	1	2	3	4	5	6	7
Prix de la figurine en spécial	-	2	-	3	3	-	7

La solution optimale trouvée par votre algorithme devrait être 2 figurines (achats les 2^{e} et 5^{e} jours).

Exemple 2 :

Journée	1	2	3	4	5	6	7	8	9	10
Prix de la figurine en spécial	-	-	-	4	2	3	-	-	-	-

La solution optimale trouvée par votre algorithme devrait être 2 figurines (achats les 5^{e} et 6^{e} jours).

Notez que ce problème a été demandé en entrevue technique d'une grande entreprise à un.e étudiant.e du DIRO au cours de la dernière année.

2 Retour-en-arrière - Contrôle de trafic illicite (70 points)

Mise en contexte

Un gouvernement souhaite diminuer le flot entrant de produits illicites dans sa région (fentanyl, armes à feux, produits de braconnage, ...). Les ressources étant limitées, il sera probablement impossible d'arrêter le trafic au complet. Par contre, on souhaite de diminuer ce trafic au plus possible en choisissant les interventions de façon optimale. En tant que consultant spécialisé en informatique, vous êtes engagé pour modéliser le problème, le résoudre algorithmiquement et proposer les meilleures interventions qui iront dans le sens de l'objectif.

Modélisation et approche

Le trafic de n'importe quelle marchandise, légale ou illégale, peut être modélisé comme un réseau, c'est-à-dire un graphe orienté avec des poids sur les arêtes (représentant les capacités), avec un sommet source duquel part le flot de marchandise et avec un sommet puits qui recueille le flot de marchandise. Les sommets peuvent représenter des entrepôts, des personnes, etc. Les arcs peuvent représenter des voies de transport et leur poids, la capacité maximale de transport.

Le flot de marchandise part du sommet source puis traverse les sommets intermédiaires pour arriver au sommet puits, tout en respectant les capacités sur les arcs. Le flot est conservé (flot entrant = flot sortant) pour tous les sommets sauf pour la source et le puits qui, respectivement, émet et reçoit seulement. Le flot maximal est le flot dont la quantité d'unité de marchandise est la plus grande possible. L'algorithme de Ford-Fulkerson permet de trouver le flot maximal dans un réseau. Cet algorithme ne sera pas montré en cours, donc une partie du défi de ce devoir est d'apprendre ce nouvel algorithme, de l'implémenter et de l'utiliser.

Les interventions seront modélisées simplement comme la suppression d'arc dans le réseau. Cela peut représenter par exemple, l'installation de postes de sécurité ou de douane, l'amélioration d'équipement de détection, patrouilles fréquentes dans une zone, etc. Chaque intervention va alors couper totalement le flot dans un arc précis en réduisant la capacité de cet arc à 0.

Dans un contexte où k interventions sont possibles, il faut identifier les k arcs à retirer du réseau pour diminuer le flot maximal le plus possible. L'approche algorithmique sera celle du retour-en-arrière. Dans l'arbre de recherche du retour-en-arrière, le nœud racine représente le réseau original. Les nœuds descendants représentent les réseaux auxquels on a enlevé 1 arc. Les nœuds descendants suivants représentent les réseaux auxquels on a enlevé 2 arcs, et ainsi de suite.

Bien sûr, lorsqu'un arc est retiré du réseau à cause d'une intervention, le crime organisé

va se réajuster à la situation pour maximiser le flot sur ce nouveau réseau. Il faut donc recalculer le flot maximal pour chaque réseau modifié.

L'approche algorithmique complète est donc un premier algorithme imbriqué dans un second. Dans l'algorithme de retour-en-arrière, chaque nœud représente un réseau dont le flot maximal est résolu par l'algorithme de Ford-Fulkerson. L'algorithme de retour-en-arrière doit aller à une profondeur maximale de k qui représente le nombre d'interventions. L'algorithme va identifier le réseau avec le plus petit flot maximal possible et retourner la liste des arcs qui ont été retirés pour arriver à ce réseau.

Par exemple, s'il y a 22 arcs dans un réseau, il y aura 22 combinaisons pour enlever un arc et il y aura $\frac{22\times21}{2}$ combinaisons pour enlever 2 arcs. En général, il y aura $\frac{a!}{(a-k)!a!}$ combinaisons pour enlever k arcs parmi les a arcs du réseau, ce qui fait que plus la profondeur à explorer est grande, plus l'arbre de recherche du retour-en-arrière est grand. Il faut s'assurer d'éviter de calculer en double les combinaisons : enlever l'arc (1,2) puis l'arc (3,4) est exactement la même chose que enlever l'arc (3,4) puis l'arc (1,2). Si l'algorithme trouve une combinaison de h arcs à enlever qui met le flot maximal à 0 mais $h < k$, il peut retourner simplement ces h arcs en réponse.

À faire

Vous devez implémenter un algorithme qui fait le travail décrit ci-haut.

Implémenter l'algorithme de **Ford-Fulkerson** ou une variante de celui-ci comme l'algorithme de **Edmonds-Karp**. Vous pouvez utiliser du code déjà existant seulement pour cet algorithme mais il est obligatoire de citer la source. Assurez-vous de bien comprendre cet algorithme.

Implémenter un algorithme de retour-en-arrière qui va tester toutes les combinaisons possibles de k arcs à enlever. C'est un algorithme exact qui retourne une solution optimale. L'algorithme peut s'arrêter plus tôt s'il trouve un nombre inférieur d'arcs à enlever qui empêchent tout flot possible. Il peut y avoir plusieurs solutions optimales avec un flot maximal équivalent, il faut juste en retourner une.

Vous devez expliquer en quelques courtes lignes votre implémentation et discuter de la complexité théorique (sans preuve) et empirique (chronométrée avec le module time) en fonction des entrées. Jusqu'à quelles profondeurs (k) vous pouvez vous rendre et pour quelle tailles?

Votre code sera exécuté comme-ci (où <instance>.csv est le nom de fichier d'une instance) :

```
python3 controle.py <instance>.csv
```

Entrée et sortie

Chaque fichier d'entrée encode un réseau. La première ligne contient le nombre de sommets dans le graphe suivi du nombre d'arcs à enlever. Ensuite, chaque sommet aura sa ligne où, après son identifiant, suivent les sommets auxquels il est connecté avec un arc sortant. Les nombres entre les parenthèses représentent les capacités de ses arcs sortants. Le premier sommet est le sommet source et le dernier sommet est le sommet puits.

La sortie sera simplement une liste d'arcs. Chaque arc retiré est inscrit sur sa propre ligne : (x, y). La sortie doit être placée dans un fichier "resultat_"+(le nom du fichier d'entrée).

Exemple d'exécution :

```
python3 controle.py ex1.csv
```

Exemple d'entrée avec 11 nœuds et 1 arc à enlever (`ex1.csv`) :

```
11;1
0;1(2);2(5);3(2)
1;4(1);5(2);6(3)
2;1(3);3(1);5(3);6(2)
3;4(3);6(3)
4;7(1)
5;7(1)
6;5(1);8(3);9(1)
7;8(1);10(8)
8;10(3)
9;8(1);10(2)
10
```

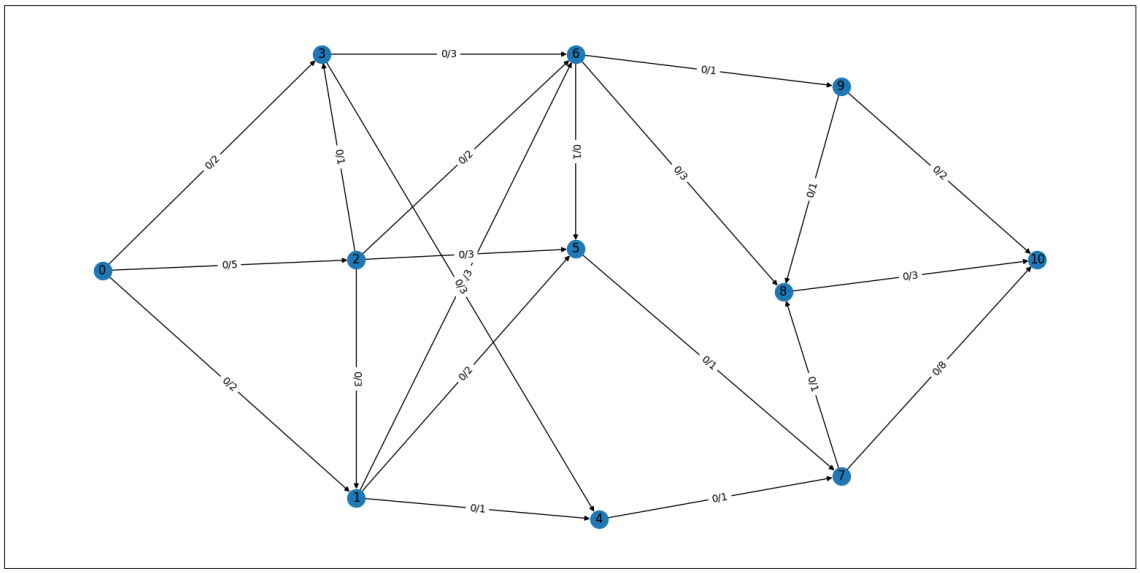
Exemple de sortie avec le seul arc à retirer (`resultat_ex1.csv`) :

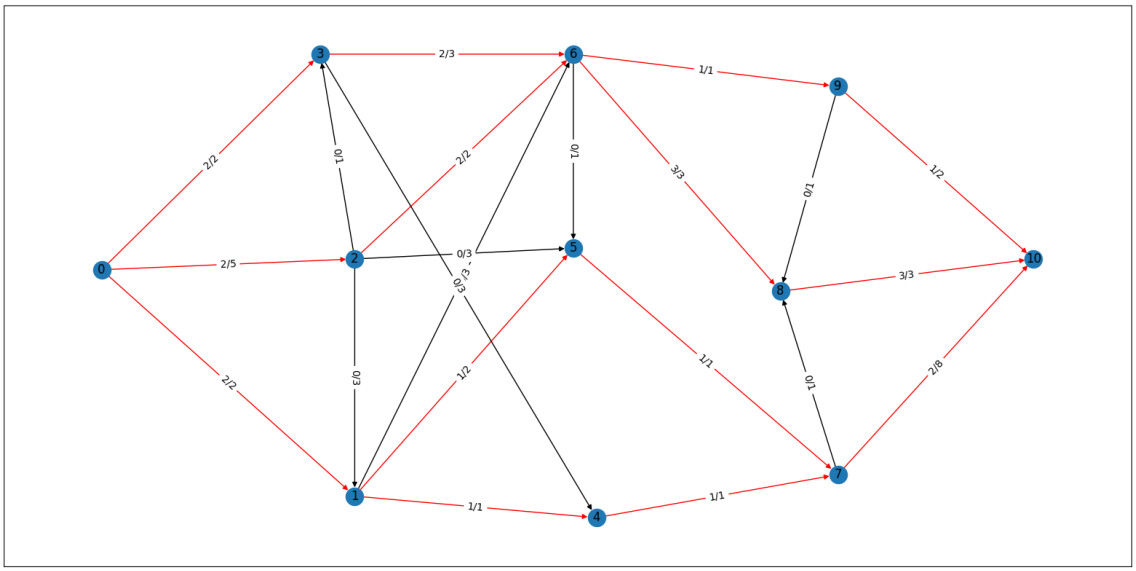
```
(6,8)
```

Exemple 1 (ex1.csv)

Pour visualiser le graphe initial :

Avec Ford-Fulkerson, on trouve un flot de 6 unités. Après avoir enlevé l'arc $(6, 8)$, on trouve un flot de 3 unités avec Ford-Fulkerson sur le graphe modifié.

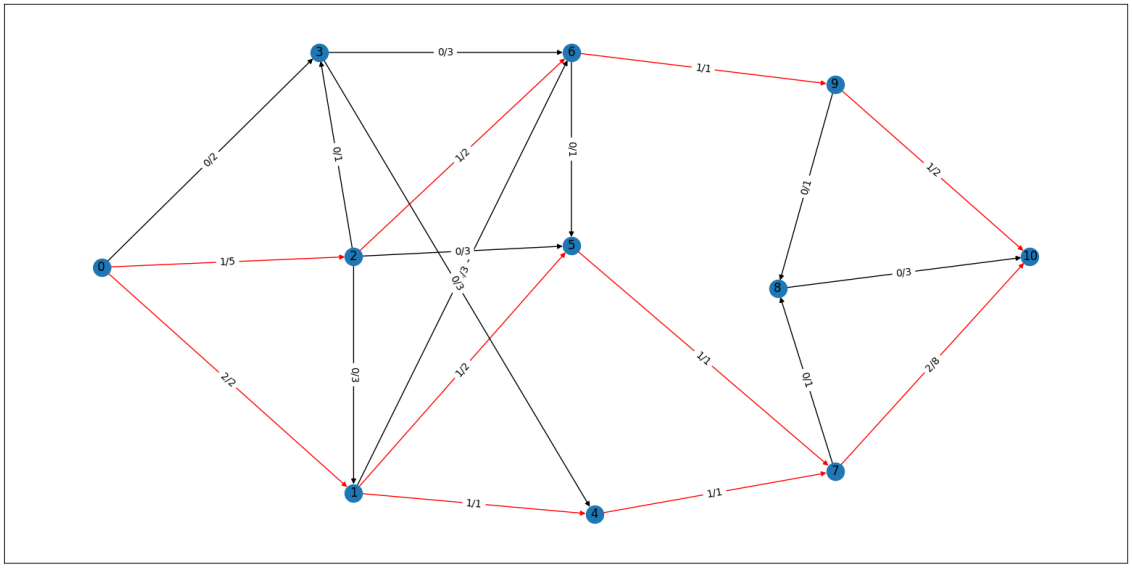




Bonus 10%

Nous allons accorder un bonus maximal de 10% (pour le devoir) aux équipes qui améliorent leur programme de contrôle de trafic illicite. Voici des avenues possibles d'améliorations :

- Faire une sortie graphique dans un fichier .png ou .jpg dans le même (flot maximal original et flot maximal avec réseau modifié). Pas de pop-up display.
- Ordonnancement des arcs à tester dans l'algorithme de retour en arrière pour que des bonnes solutions soient trouvées rapidement. C'est utile si on décidait par exemple de couper la recherche exhaustive du retour-en-arrière pour en faire une



heuristique.

- Trouver un calcul de borne supérieure valide et transformer l'algorithme de retour-en-arrière en algorithme Branch-and-Bound.
- etc.

Il faut bien sûr expliquer brièvement votre amélioration et (pour les améliorations algorithmiques, non graphiques) discuter de la différence de performance entre votre variante et la version originale.

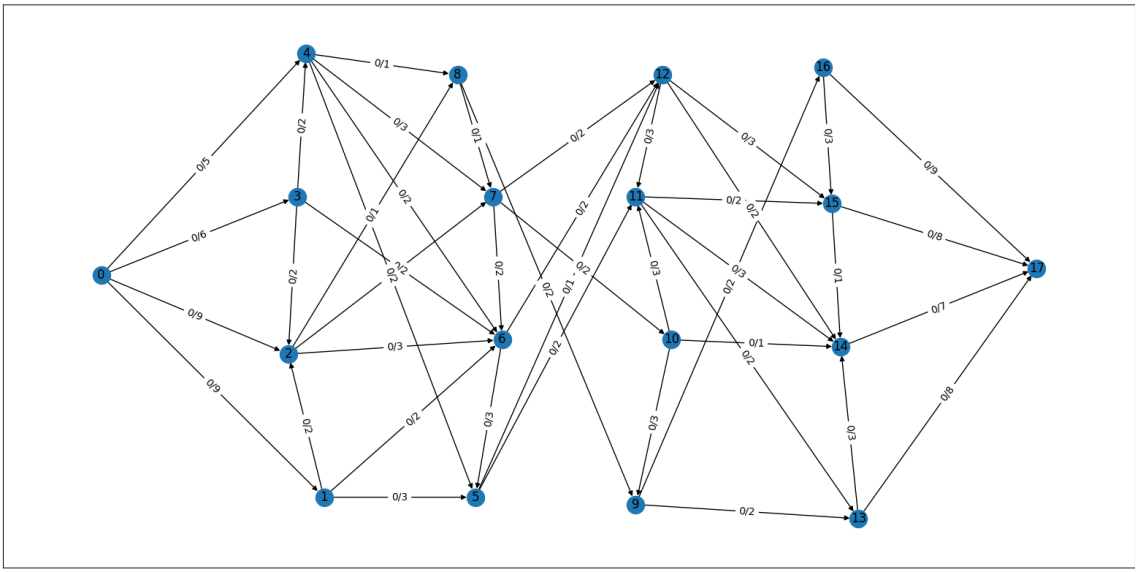
Annexe

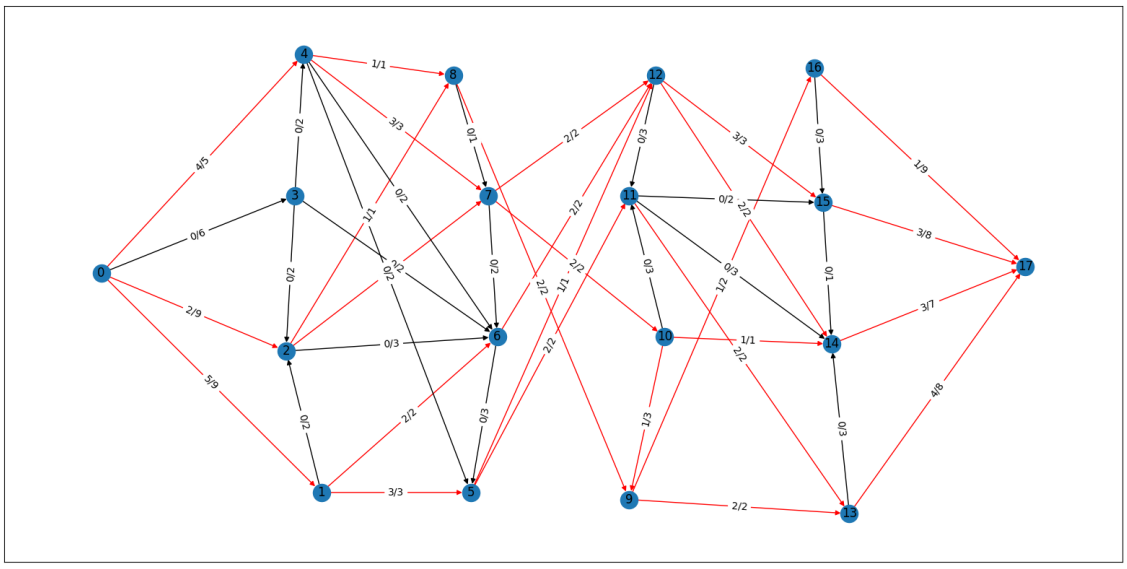
L'annexe suivante montre l'importance d'utiliser une recherche exhaustive pour ce problème avec l'algorithme de retour-en-arrière qui garantira une réponse optimale. L'heuristique vorace de prendre les arcs de plus grand flot ou de plus grande capacité ne donne pas de solution optimale dans tous les cas.

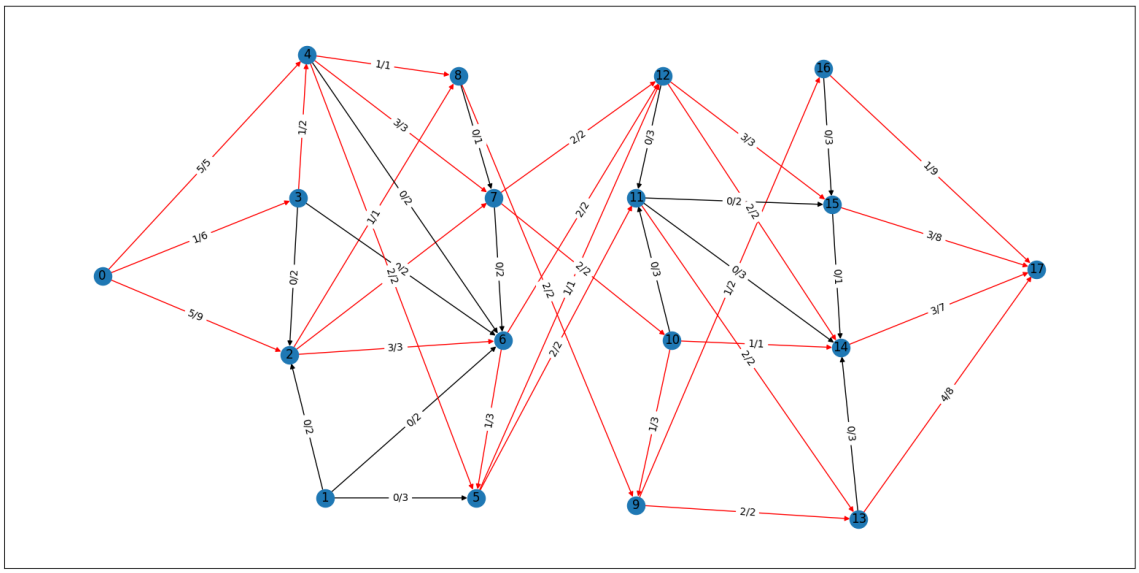
Exemple 2 (ex2.csv)

Avec Ford-Fulkerson, on trouve un flot de 11 unités. Dans cet exemple, le fait de retirer l'arc $(0, 1)$ qui a le plus grand flot trouvé par Ford-Fulkerson (il existe plusieurs possibilités de flot maximal ici), ne change rien au flot maximal. Après avoir enlevé cet arc, le trafic s'est réajusté et le flot est toujours de 11 unités. On peut remarquer qu'un goulot se trouve au milieu du réseau (c'est la "coupe minimale" du graphe). À

cet endroit, tous les arcs ont des flots égaux à la capacité maximale. Alors n'importe quel arc retiré dans cet endroit diminuerait nécessairement le flot.





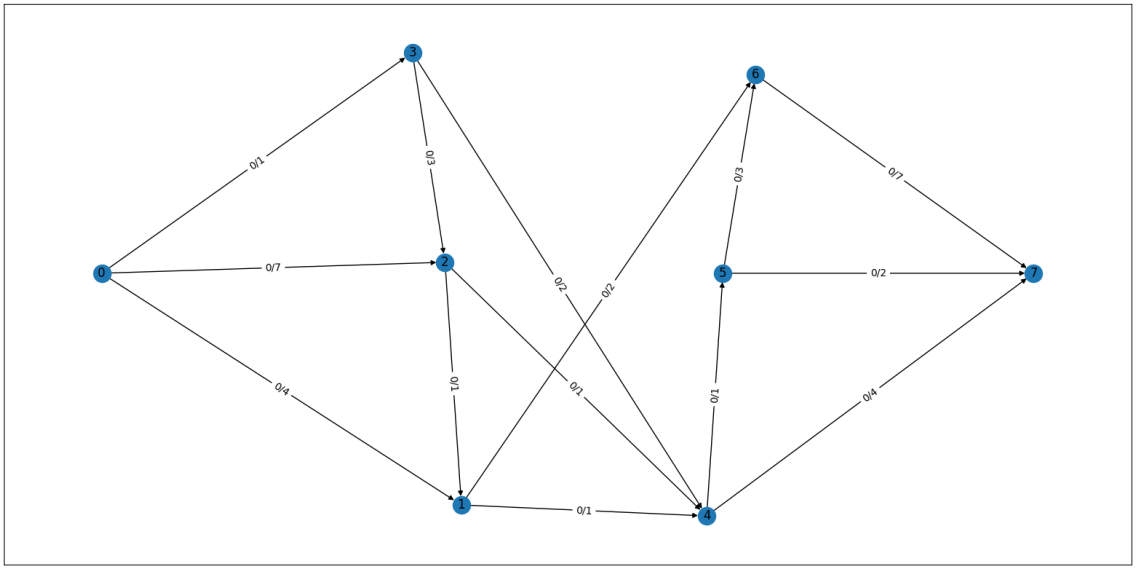


Exemple 3 (ex3.csv)

Avec Ford-Fulkerson, on trouve un flot de 5 unités. Dans cet exemple, le fait de retirer les arcs $(0, 1)$ et $(4, 7)$ ayant le plus grand flot trouvé par Ford-Fulkerson donne une solution qui n'est pas optimale. Après avoir enlevé ces arcs, il y a un flot restant de 2 unités. Par contre, on voit bien qu'en enlevant les arcs $(0, 1)$ et $(0, 2)$, nous aurions un flot maximal de 1 unité passant par l'arc $(0, 3)$.

Exemple de sortie avec les 2 arcs à retirer pour avoir la solution optimale pour l'exemple 3 (`resultat_ex3.csv`) :

```
(0,1)
(0,2)
```



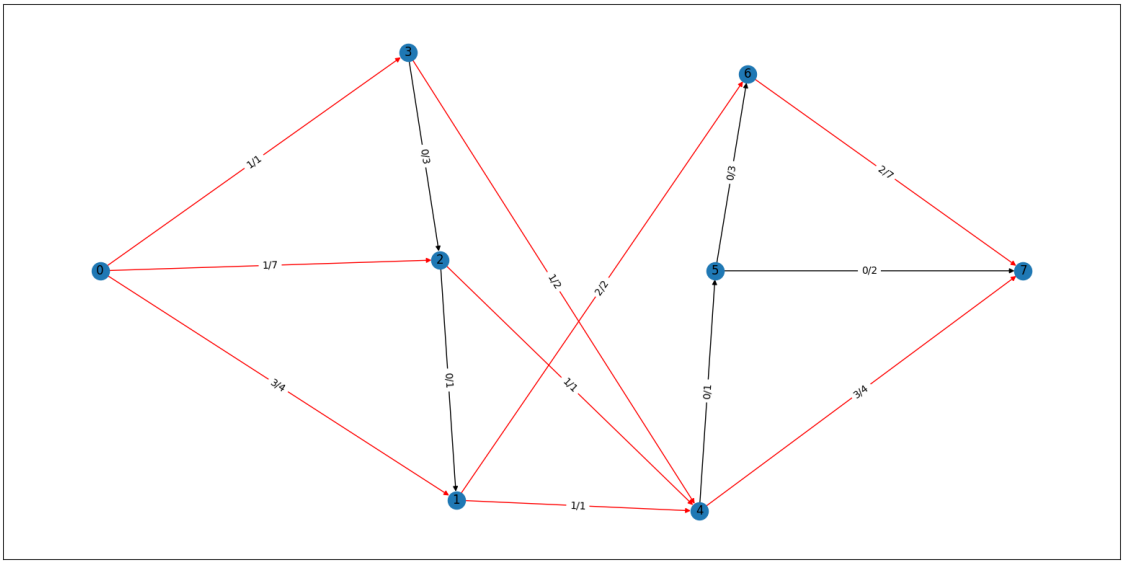
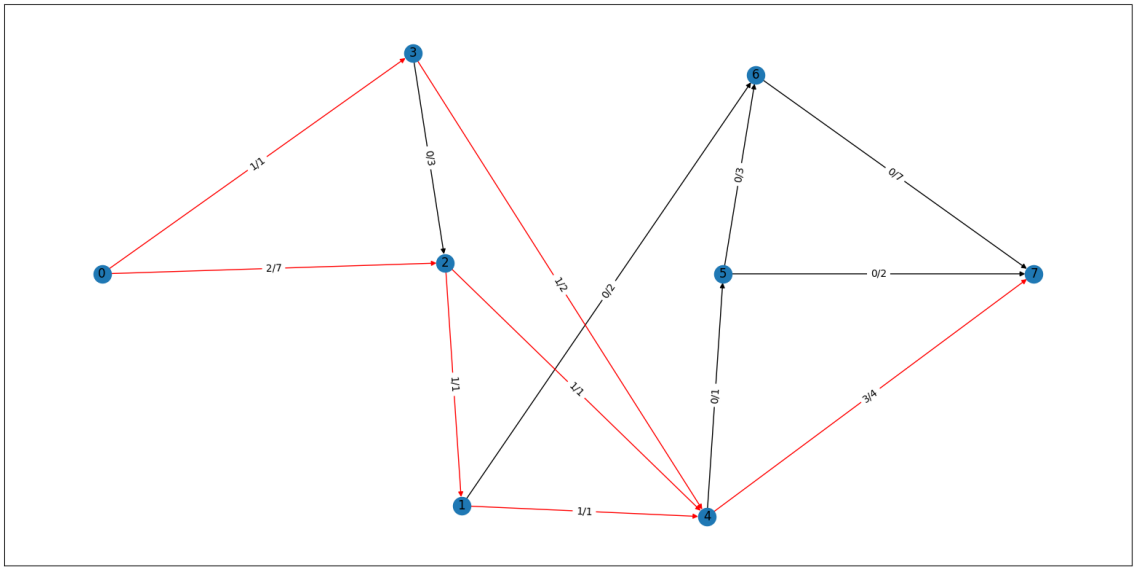
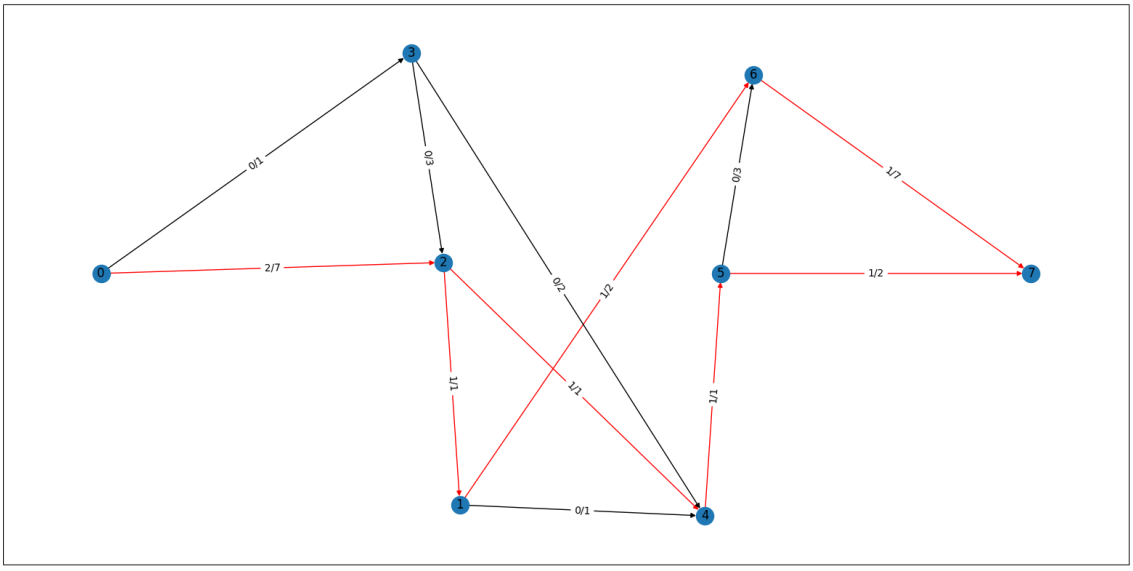
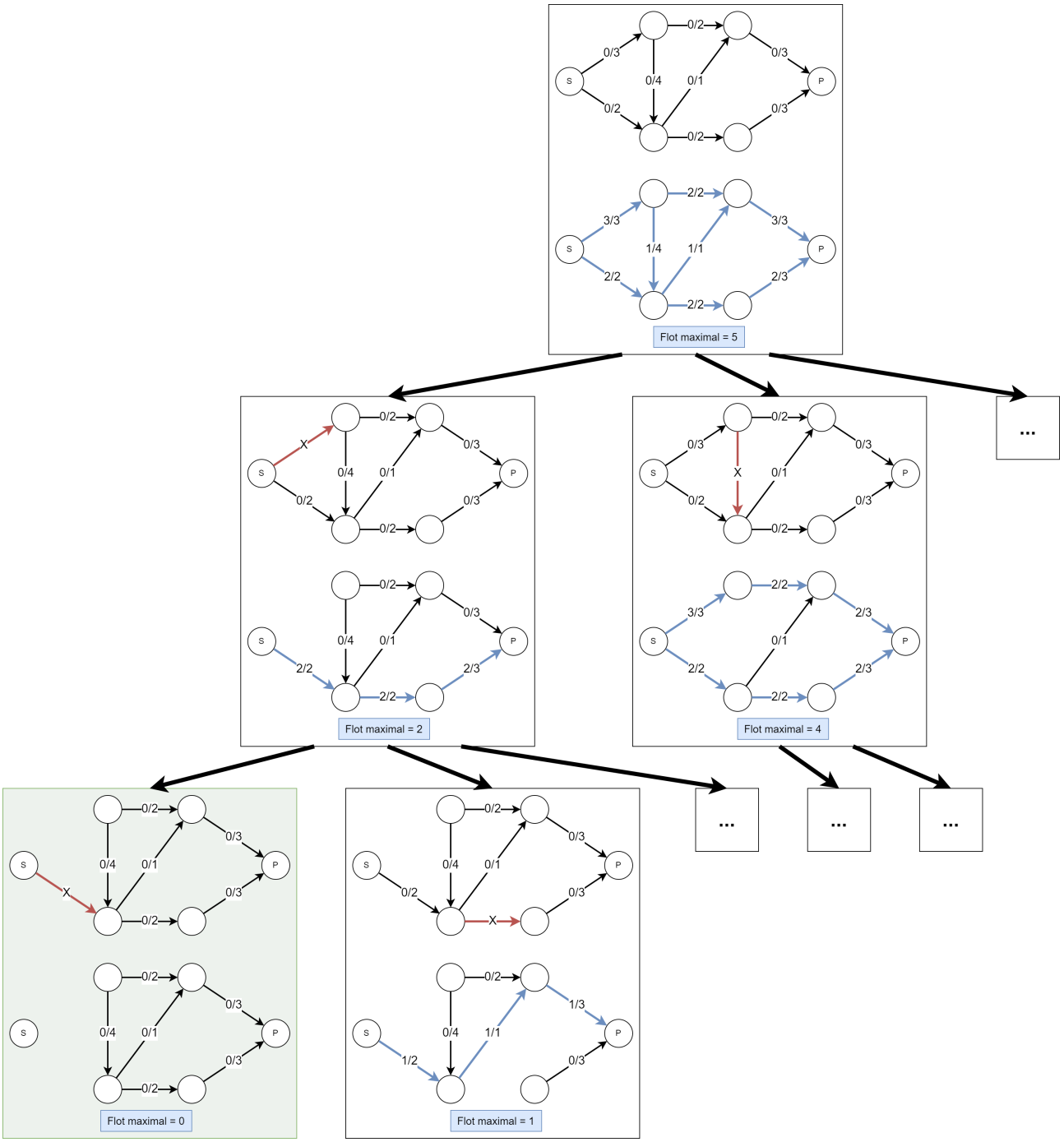


Illustration des algorithmes imbriqués de retour-en-arrière et Ford-Fulkerson


3
6
1/1
0/3
0/3
0/7
0
2/7
2
1/2
0/2
5
0/2
7
1/1
1/1
0/1
3/4
1
1/1
4


3
6
0/1
0/3
0/3
1/7
0
2/7
2
0/2
1/2
5
1/2
7
1/1
1/1
1/1
1
0/1
4


Flot maximal = 5
Flot maximal = 2
Flot maximal = 4
Flot maximal = 0
Flot maximal = 1
...