

# TP 2 – Arbres orientés

Gaëtan Staquet

2026

Ce TP est découpé en deux parties. La première consiste à implémenter les arbres orientés, ainsi qu'à comprendre un algorithme pour trouver un arbre couvrant de poids minimum dans un graphe orienté. La seconde se concentre sur des applications pouvant être résolues via des arbres.

Avant de vous lancer dans le TP, voici un rappel des consignes pour les TP de GRAAL.

- La première partie est la plus importante pour la suite du cours. Essayez quand même d'avancer le plus possible dans la seconde partie.
- Le langage à utiliser pendant les TP est Python.
- Vous êtes libre d'utiliser les outils que vous voulez, y inclus des LLM et autres IA génératives.
- Veuillez limiter votre usage de modules externes. En particulier, les bibliothèques de graphes (comme `NetworkX` ou `graph-tool`) sont interdites, étant donné que le but est d'implémenter les algorithmes par vous-même.
- **Il vous est demandé d'utiliser le même projet Python pendant l'ensemble des TP. Ceci vous permettra de mettre en application des bonnes pratiques à plus long terme, dans un contexte dans lequel les besoins évoluent et vous sont inconnus. Notamment, pensez à bien structurer, documenter et tester votre code en amont afin de pouvoir facilement le modifier plus tard.**
- À cette fin, créez un ou des modules pour la première partie. Le code pour la seconde partie peut ensuite se reposer sur ces modules. En d'autres termes, séparez les algorithmes génériques des applications spécifiques.

# Partie I — Arbres orientés

À la fin de cette partie, vous aurez une implémentation des arbres orientés et une version spécifique du DFS. De plus, vous prendrez connaissance d'un algorithme pour trouver un arbre couvrant de poids minimum d'un graphe pondéré orienté.

## I.1. Recherche d'une sous-suite augmentante

D'abord, exploitons les arbres orientés pour aisément trouver une sous-suite augmentante de longueur maximale dans une suite de nombre entiers. Cet exercice vient de Michel GONDRAN et Michel MINOUX. *Graphes et algorithmes (3e éd.)* Eyrolles, 1995.

Soit  $S = (x_1, \dots, x_n)$  une suite quelconque de  $n$  nombres entiers. On veut extraire de cette suite une sous-suite  $(x_{i_1}, x_{i_2}, \dots, x_{i_p})$  de longueur maximale telle que

- $i_1 < i_2 < \dots < i_p$  (afin d'avoir une sous-suite), et
- $x_{i_1} \leq x_{i_2} \leq \dots \leq x_{i_p}$  (la sous-suite doit être augmentante).

On peut trouver une telle sous-suite augmentante de longueur maximale via un arbre orienté qu'on construit récursivement comme suit.

- Fixons  $r$  la racine de l'arbre. Ce sommet est relié à  $n$  autres sommets, un pour chaque élément de la suite. Nommons-les  $x_1^1, x_2^1, \dots, x_n^1$ .
- Pour chaque  $j$ , le sommet  $x_j^1$  (on regarde le  $j^{\text{e}}$  élément de la suite au niveau 1 de notre construction) est relié à un sommet  $x_k^2$  si et seulement si  $j < k$  ( $x_k$  apparaît après  $x_j$  dans la liste) et  $x_j \leq x_k$  (on peut étendre la sous-suite augmentante avec  $x_k$ ).
- On continue ce principe avec les sommets  $x_j^2, x_j^3$ , et ainsi de suite jusqu'à ne plus pouvoir étendre l'arbre.

**Exercice I.1.1.** Construisez l'**arbre orienté** pour la suite  $(1, 3, 4, 3, 5)$ . Comment en déduire une sous-suite augmentante de longueur maximale? Existe-t-il une unique telle sous-suite?

Combien de sommets avez-vous? Pourrait-on réduire le nombre de sommets en construisant un graphe quelconque au lieu d'un arbre?

**Exercice I.1.2.** Justifiez en quelques mots que l'algorithme finit par s'arrêter et met en évidence les sous-suites augmentantes de longueur maximale d'une suite.

## I.2. Implémentation

Reprenez et étendez votre implémentation des graphes orientés du TP précédent.

**Exercice I.2.1.** Ajoutez une nouvelle classe qui implémente les arbres orientés, en vous basant sur la classe `DirectedGraph`. Que faut-il changer? En particulier, faut-il modifier les classes des sommets et des arcs?

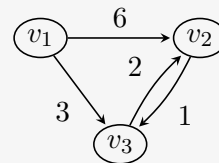
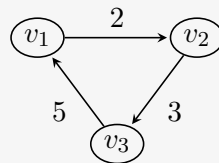
### I.3. Arbres couvrants de poids minimum

Durant le CM, vous avez vu un algorithme pour calculer un arbre couvrant de poids minimum d'un graphe pondéré non orienté. Ici, nous allons nous intéresser à la même problématique pour les **graphes pondérés orientés**.

**Exercice I.3.1.** En vous inspirant de ce qui a été fait durant le cours, proposez une définition pour un arbre couvrant de poids minimum d'un graphe orienté. Est-ce qu'on pourrait avoir un arc qui entre dans la racine ?

**Algorithme de Kruskal.** Une première approche serait de prendre l'algorithme de Kruskal (*cf.* les slides) tel quel.

**Exercice I.3.2.** Appliquez l'algorithme de Kruskal sur les deux graphes suivants.



Est-ce que les deux arbres obtenus sont bien des arbres couvrants de poids minimum ? Que pouvez-vous en déduire sur le fait d'utiliser l'algorithme de Kruskal sur des graphes orientés ? En particulier, que se passe-t-il si on supprime l'arc de  $v_1$  à  $v_2$  dans le second graphe ?

Si l'arbre retourné par l'algorithme de Kruskal sur le second graphe vous paraissait correct, faites-moi signe.

**Algorithme de Chu-Liu/Edmonds.** À partir de maintenant, nous allons supposer que les graphes orientés que nous allons considérer sont **connexes** : n'importe quel sommet est accessible depuis au moins un autre sommet.

**Définition I.3.3 (Graphe orienté connexe).** Un graphe orienté  $\mathcal{G} = (V, E)$  est dit *connexe* si et seulement si, pour tout  $u, v \in V$ , il existe une chaîne qui relie  $u$  et  $v$ . Mathématiquement, il existe  $x_1, \dots, x_n \in V$  tels que  $(x_1, \dots, x_n)$  est un chemin dans  $\mathcal{G}$  et  $(x_1 = u \wedge x_n = v) \vee (x_1 = v \wedge x_n = u)$ .

**Exercice I.3.4.** Argumentez que n'importe quel graphe **fortement connexe** est **connexe**.

Nous allons nous concentrer sur l'algorithme de Chu-Liu/Edmonds.<sup>1</sup> Cet algorithme identifie un arbre couvrant de poids minimum d'un graphe orienté pondéré  $\mathcal{G} = (V, E, p)$ . La racine  $r$  de cet arbre est fixé dans les paramètres de l'algorithme. L'explication qui suit et l'exemple sont inspirés de GONDRAN et MINOUX, *Graphes et algorithmes* (3e éd.)

1. Yoeng-Jin Chu et Tseng-Hong Liu ont proposé l'algorithme en 1965. Jack Edmonds l'a indépendamment décrit en 1967.

L'idée est de sélectionner, pour chaque sommet qui n'est pas la racine, l'arc entrant de poids minimum. Si le graphe obtenu contient un cycle  $\pi = (v_1, \dots, v_n)$ , on le **contracte** pour obtenir :

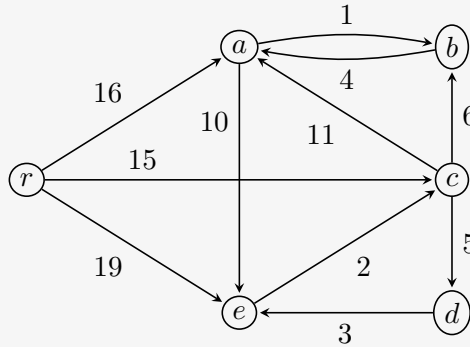
- Tous les sommets de  $\pi$  sont remplacés par un unique sommet, disons  $v_\pi$ .
- Pour chaque  $i \in \{1, \dots, n\}$  et arc  $(u, v_i) \in E$  (*i.e.*, un arc qui entre dans le cycle), on remplace cet arc par  $(u, v_\pi)$  (*i.e.*, on garde l'information que l'arc entre dans le cycle).
- On change les poids  $p^\mathcal{I}$  pour refléter les modifications des arcs. Tous les arcs  $(u, v)$  qui ne rentrent pas dans le cycle  $\pi$  (*i.e.*, tels que  $v \notin \{v_1, \dots, v_n\}$ ) gardent leur poids :  $p^\mathcal{I}((u, v)) = p((u, v))$ . Dans le cas où plusieurs poids sont possibles, on garde le poids minimum.

À l'inverse, le poids d'un arc  $(u, v)$  qui entre dans  $\pi$  (*i.e.*, tel que  $u \notin \{v_1, \dots, v_n\}$  et  $v \in \{v_1, \dots, v_n\}$ ) doit être modifié pour représenter l'existence du cycle caché dans le graphe contracté. Plus précisément, vu que le sommet  $v_\pi$  représente un cycle et qu'on cherche un arbre couvrant de poids minimum, on prend en compte le fait qu'à terme, on ne gardera pas l'ensemble des arcs de  $\pi$ . Ainsi, on prend l'arc entrant  $(u', v)$  qu'on a préalablement sélectionné pour  $v$  et on change le poids de  $(u, v)$  :  $p^\mathcal{I}((u, v)) = p((u, v)) - p((u', v))$ . À nouveau, si plusieurs poids sont possibles, on garde le poids minimum.

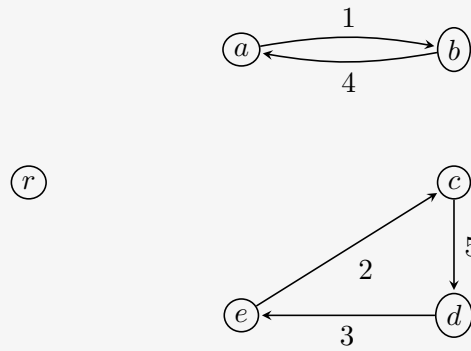
On recommence ensuite avec le graphe contracté : pour chaque sommet qui n'est pas la racine, on prend l'arc entrant de poids minimum et, si le graphe résultant contient un cycle, on le contracte à nouveau. On répète ceci jusqu'à obtenir un graphe qui ne contient pas de cycle.

Il nous reste alors à extraire un arbre couvrant de poids minimum de ce dernier graphe contracté. Par soucis de clarté, cette partie est expliquée dans l'exemple suivant.

*Exemple I.3.5.* Soit  $\mathcal{G} = (V, E, p)$  le graphe orienté pondéré suivant :



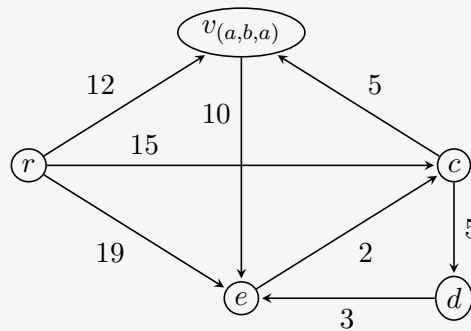
On veut un arbre couvrant de poids minimum de racine  $r$ . On commence donc par choisir, pour chaque sommet différent de  $r$ , un arc entrant de poids minimum. On obtient le graphe suivant.



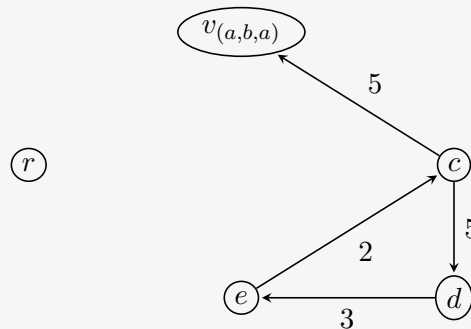
Vu que ce graphe contient (au moins) un cycle, on le contracte. Disons qu'on travaille sur le cycle  $(a, b, a)$ . On remplace donc les sommets  $a$  et  $b$  par  $v_{(a,b,a)}$ . Il nous reste à déterminer les poids des nouveaux arcs :

- L'arc  $(r, v_{(a,b,a)})$  a comme poids  $16 - 4 = 12$  (le poids de l'arc  $(r, a)$  moins celui de  $(b, a)$ ).
- Vu qu'on a  $(c, a)$  et  $(c, b)$  dans le graphe de départ, il existe donc deux poids possibles pour l'arc  $(c, v_{(a,b,a)})$  :  $6 - 1 = 5$  et  $11 - 4 = 7$ . On garde le minimum, donc 5.

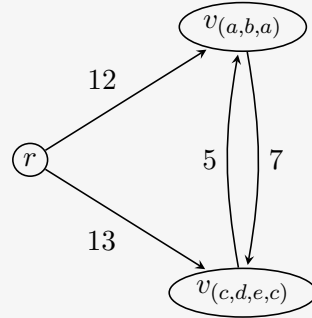
Les autres arcs gardent leur poids original, ce qui nous donne le graphe  $\mathcal{G}^1$  :



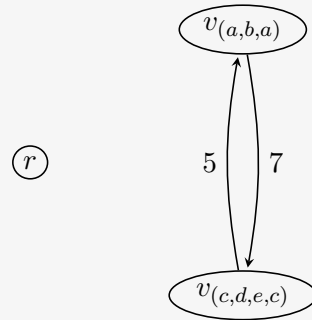
On recommence le processus. D'abord, on sélectionne, pour chaque sommet, un arc de poids minimum entrant dans ce sommet.



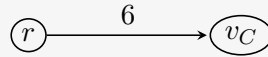
À nouveau, on a un cycle qu'on contracte pour obtenir  $\mathcal{G}^2$  :



On recommence sur ce graphe. D'abord, le choix des arcs entrants :



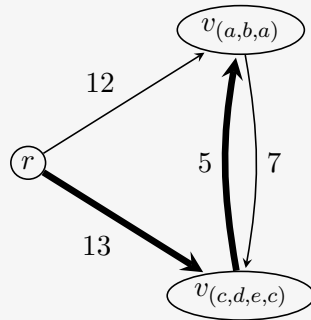
On a encore une fois un cycle qu'on contracte en  $\mathcal{G}^3$  :



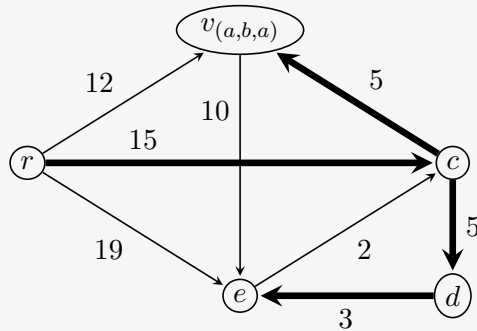
Notez que le poids de  $(r, v_C)$  est le minimum entre  $12 - 5 = 7$  (l'arc  $(r, v(a,b,a))$  moins  $(v(c,d,e,c), v(a,b,a))$ ) et  $13 - 7 = 6$  ( $(r, v(c,d,e,c))$  moins  $(v(a,b,a), v(c,d,e,c))$ ).

Clairement, le graphe obtenu en choisissant un arc pour  $v_C$  est le même et ne contient pas de cycle. On a donc fini les itérations. Il reste à construire un arbre couvrant de poids minimum de  $\mathcal{G}$  depuis  $\mathcal{G}^1, \mathcal{G}^2$  et  $\mathcal{G}^3$ . Pour ce faire, on procède dans l'ordre inverse. D'abord, on trouve un arbre couvrant de poids minimum de  $\mathcal{G}^3$  (ce qui se fait facilement).

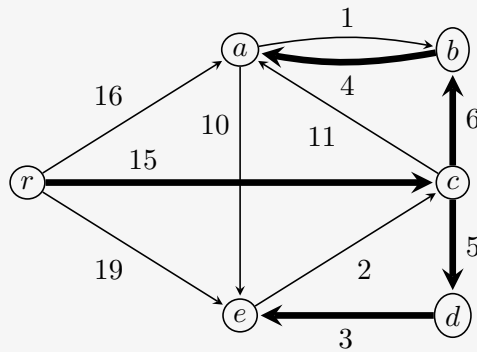
Le poids de l'arc  $(r, v_C)$  de  $\mathcal{G}^3$  vient de l'arc  $(r, v(c,d,e,c))$  de  $\mathcal{G}^2$ . Donc, un arbre couvrant de poids minimum de  $\mathcal{G}^2$  contient nécessairement l'arc  $(r, v(c,d,e,c))$ . On rajoute aussi les arcs du cycle  $(v(a,b,a), v(c,d,e,c), v(a,b,a))$ , sauf l'arc  $(v(a,b,a), v(c,d,e,c))$  dont la valeur a été retranchée pour donner l'arc de  $\mathcal{G}^3$ . Visuellement, l'arbre couvrant de poids minimum de  $\mathcal{G}^2$  est formé par les arcs en gras de cette figure :



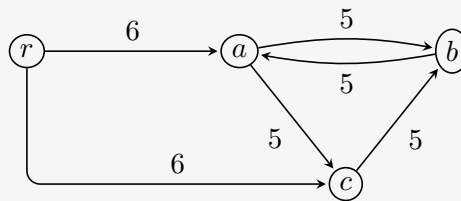
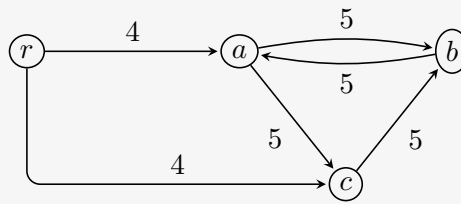
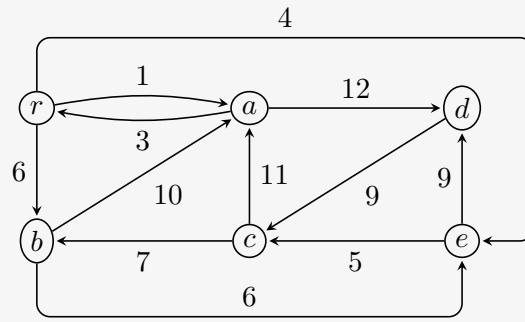
Le graphe  $\mathcal{G}^2$  a été obtenu en contractant le cycle  $(c, d, e, c)$  de  $\mathcal{G}^1$ . On « déplie » donc l'arc  $(r, v_{(c,d,e,c)})$  (celui qui entre dans le cycle).



Il nous reste à déplier l'arc  $(c, v_{(a,b,a)})$  (l'arc qui entre dans le cycle contracté pour  $\mathcal{G}^1$ ) pour obtenir un arc couvrant de poids minimum de  $\mathcal{G}$  :



**Exercice I.3.6.** Appliquez l'algorithme de Chu-Liu/Edmonds sur les graphes suivants pour trouver un arbre couvrant de poids minimal de racine  $r$ .



**Exercice I.3.7.** Est-ce que l'affirmation suivante est vraie ou fausse. Argumentez.  
Quel que soit le graphe orienté  $\mathcal{G}$  connexe, il existe toujours un unique arbre de poids minimal.

**Exercice I.3.8.** Faites une analyse descendante de ce problème :

- décrivez les structures de données nécessaires ;
- listez les grandes étapes de l'algorithme, à haut niveau ;
- donnez les fonctions secondaires pertinentes et une courte description de chacune.

Autrement dit, expliquez en quelques paragraphes comment vous implémenteriez l'algorithme. Vous n'avez pas à aller dans les détails ; par exemple, vous pouvez dire « on mémorise que le poids de l'arc  $(u, v)$  est dû à celui de l'arc  $(u', v')$ , » sans aller plus loin.

**Exercice I.3.9.** Quelle est la complexité de l'algorithme ?

Vu qu'implémenter l'algorithme de Chu-Liu/Edmonds n'est pas trivial, il ne vous est pas demandé de le faire.



# Partie II — Applications

Cette partie se concentre sur des applications des arbres orientés. Comme pour le TP 1, vous êtes libre de choisir sur quelle(s) application(s) vous vous concentrez et dans quel ordre. Il n'est pas demandé de tout réaliser.

## II.1. Recherche d'une sous-suite augmentante

Maintenant que vous avez une implémentation des arbres orientés, vous pouvez implémenter l'algorithme décrit dans la Section I.1 et vérifier vos réponses.

**Exercice II.1.1.** Vérifiez vos réponses à l'Exercice I.1.1 via votre implémentation.

## II.2. Transport d'organes

Étant donné l'urgence de transporter des organes d'un hôpital à un autre, il est important de trouver la meilleure assignation possible des organes qui minimise le coût de transport. De plus, il faut également minimiser le temps de trajet afin de maximiser le taux de transplantations réussies. Vu le manque de moyens, il n'est pas toujours possible de déplacer des organes d'un hôpital à un autre. Pour certains, il existe seulement une connexion dans un sens. Autrement dit, nous allons manipuler des graphes **orientés** munis de **deux pondérations** : une pour le coût et une pour le temps.

Considérons le graphe des hôpitaux de Nantes<sup>2</sup> donné dans la Figure 1. Les couples sur les arcs indiquent les deux pondérations : la première est le coût financier, la seconde le temps de transport.

La ville de Nantes désire faire des travaux sur plusieurs rues en même temps. Il y a deux intérêts en conflit :

- La ville veut faire autant de travaux que possible en parallèle.
- Pour des raisons évidentes de santé publique, il faut planifier ces travaux de manière à ne pas bloquer les transports d'organes.

En particulier, on veut s'assurer que les organes partant de l'**Hôtel-Dieu** peuvent atteindre n'importe quel autre hôpital.

Votre tâche est de trouver un sous-ensemble des arcs qui permet d'atteindre n'importe quel hôpital depuis l'Hôtel-Dieu. Malgré vos tentatives pour faire entendre raison à la ville, vous devez **d'abord minimiser le coût financier**. En cas d'égalité entre plusieurs possibilités, une qui minimise le temps est à privilégier.

**Exercice II.2.1.** Donnez un tel sous-ensemble des arcs de la Figure 1. Si on essayait de minimiser le temps de trajet, quel serait l'impact financier ?

---

2. Par facilité pour cet exercice, on considère que n'importe quel site du CHU de Nantes peut effectuer une transplantation d'organes.

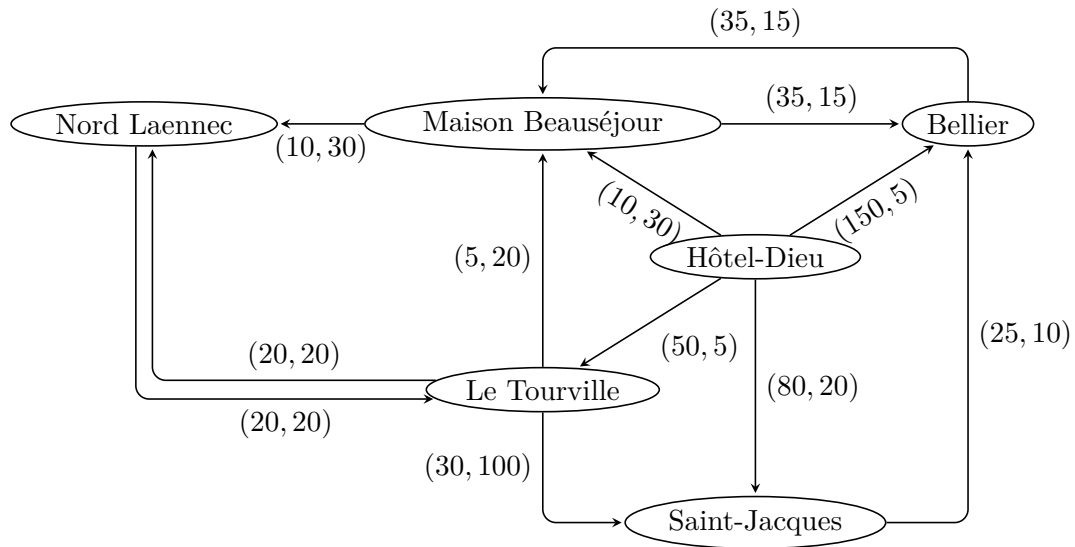


FIGURE 1 – Graphe des hôpitaux de Nantes, avec les deux pondérations pour le transport d’organes. La première valeur est le coût financier et la seconde le temps de trajet. Ces valeurs ne reflètent pas la réalité.

### II.3. Arbres rouge-noir

Comme rappelé en fin de CM, il est important d’équilibrer les arbres binaires de recherche pour garantir de bonnes performances. Il existe plusieurs façons d’automatiser cet équilibrage. Vous avez déjà vu les arbres AVL. Le but de cette partie est de découvrir, en autonomie (mais n’hésitez pas à me poser des questions !), les arbres rouge-noir.

**Exercice II.3.1.** En vous aidant de ressources disponibles en ligne, expliquez les principes derrière les arbres rouge-noir. Implémentez les arbres rouge-noir.

### Références

GONDRAN, Michel et Michel MINOUX. *Graphes et algorithmes (3e éd.)* Eyrolles, 1995.