INFO-F203 – Algorithmique 2
Projet 1

Année académique 2018–2019

Ce projet a pour but de mettre en pratique des concepts vus au cours d'algorithmique 2 ainsi que vous familiariser avec certaines généralisations de notions abordées au cours. Il est découpé en deux sections.

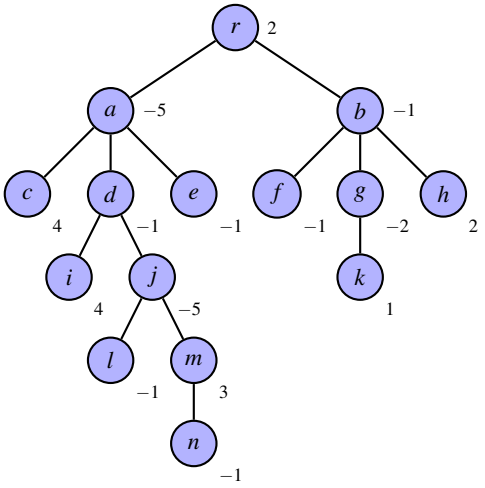
1 Sous-arbre de poids maximum

Considérons le problème suivant. Soit un arbre $T = (V, E)$ enraciné en $r \in V$, ainsi qu'une fonction de poids sur les sommets $w : V \to \mathbb{R}$, trouver un sous-arbre de poids total positif $T' = (V', E')$ de T avec racine r qui maximise

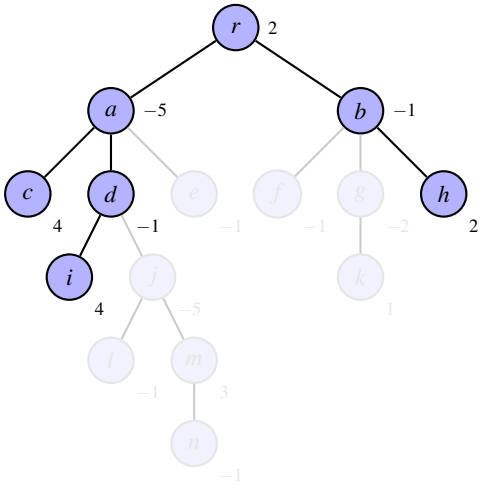
$$w(V') = \sum_{v \in V'} w(v).$$

Nous pouvons illustrer ce problème avec la figure suivante.

Arbre pondéré avec racine r



Sous-arbre avec racine r de poids maximal



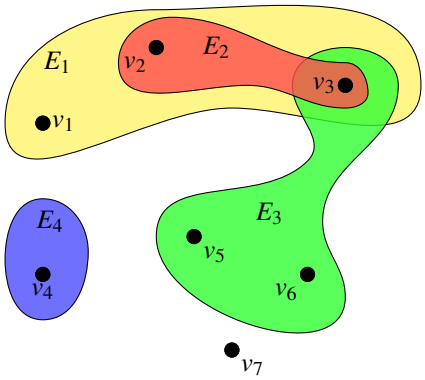
L'idée de l'algorithme est d'utiliser la programmation dynamique. Vous pouvez considérer que s'il n'existe pas de sous-arbre de poids positif contenant r, alors votre algorithme retournera l'ensemble vide.

Nous vous demandons d'écrire une fonction python `max_subtree` qui prend en paramètre un arbre enraciné en r dont les sommets sont pondérés et affichera à l'écran les sommets qui constituent un sous-arbre enraciné en r de poids maximum.

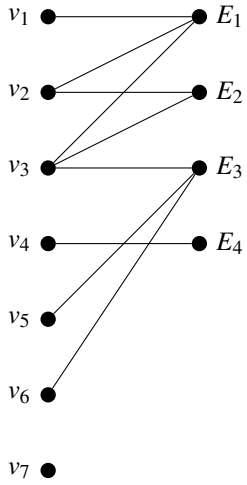
2 Les hypergraphes et hypertrees

Définition hypergraphe

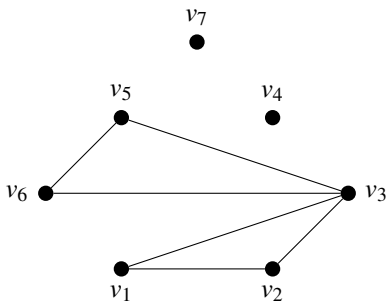
Ce projet a aussi pour but de vous familiariser avec une généralisation des graphes : les "hypergraphes". Commençons par la définition, un *hypergraphe* est un couple $(V, \mathscr{E})$ avec V un ensemble de *sommets* et $\mathscr{E} \subseteq 2^V$ un ensemble d'*hyper-arêtes*. Ci-dessous un exemple d'hypergraphe avec 7 sommets et 4 hyper-arêtes.



Un outil utile pour travailler sur les hypergraphes sont les *graphes d'incidence* : un hypergraphe $(V, \mathscr{E})$ peut être représenté par un graphe biparti $(V \cup \mathscr{E}, T)$, où $\{v, E\}$ avec $v \in V$ et $E \in \mathscr{E}$ est dans T si $v \in E$. Ci-dessous, le graphe d'incidence de l'hypergraphe exhibé plus haut.

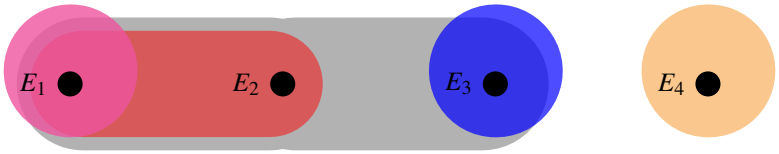


Une autre définition utilisée est le *graphe primal* d'un hypergraphe, ce graphe est défini sur les sommets de l'hypergraphe, avec des arêtes entre sommets contenus dans la même hyper-arête. Ci-dessous, le graphe primal de l'hypergraphe exhibé plus haut.



Dualité, acyclicité et hypertrees

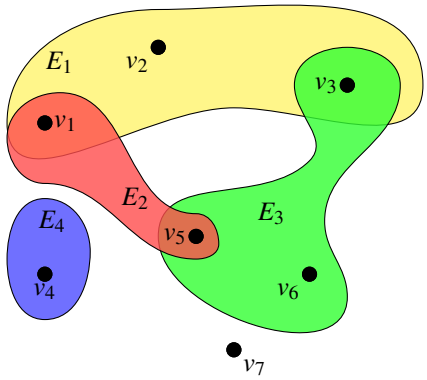
Pour un hypergraphe $H = (V, \mathscr{E})$ nous définissons son *hypergraphe dual* H^* comme l'hypergraphe basé sur H où les sommets et hyper-arêtes ont été permutés. Plus formellement, $H^* = (V^*, \mathscr{E}^*)$ où $V^* = \mathscr{E}$ et pour chaque sommet v dans V nous avons une hyper-arête dans $\mathscr{E}^*$ de la forme $E_v = \{X \subseteq \mathscr{E} : v \in X\}$. Ci-dessous, l'hypergraphe dual de l'hypergraphe exhibé plus haut.



E_{v_1} E_{v_2} E_{v_3} E_{v_4} E_{v_5} E_{v_6} E_{v_7}

Avec cette définition nous pouvons dire qu'un hypergraphe H est un *hypertree* si et seulement si son hypergraphe dual est α-acyclic. Nous donnons également la définition de ce dernier concept. Un hypergraphe est α-*acyclique* si son graphe primal est cordal et que toute clique maximale (au sens de l'inclusion) de taille deux ou plus dans le graphe primale est une hyper-arête dans l'hypergraphe. Rappelons qu'une clique est un sous-graphe induit complet et qu'un graphe est cordal si chacun de ses cycles de quatre sommets ou plus possède une corde, c'est-à-dire une arête reliant deux sommets non adjacents du cycle.

Par example, l'hypergraphe de la sous-section précédente est α-acyclique. En revanche, l'hypergraphe ci-dessous n'est pas α-acyclique. Effectivement, la clique maximale $\{v_1, v_3, v_5\}$ dans le graphe primal n'est pas une hyper-arête.



Nous vous demandons d'écrire une fonction python `test_hypertree` qui prend en paramètre un hypergraphe affichera à l'écran son hypergraphe dual (sous le format que vous jugez le plus simple) et si oui ou non il s'agit d'un hypertree. Vous pouvez utiliser (si cela vous aide) le concept d'ordonnancement d'élimination parfaite (voir Fulkerson et Gross) pour tester si un graphe est cordal, et l'algorithme de Bron–Kerbosch qui permet de lister toutes les cliques maximales d'un graphe.

Bonus

Si nous définissons le concept de *couverture exacte* pour un hypergraphe comme étant un ensemble d'hyper-arêtes sans intersection qui couvre tous les sommets. Nous vous demandons d'écrire une fonction python `cover_hypertree` qui prend en paramètre un hypertree et affichera si oui ou nous il existe une couverture exacte pour cet hypertree (si la réponse est oui, la fonction affichera également la couverture). Il existe un algorithme polynomial pour résoudre ce problème, mais celui-ci est loin d'être trivial. Une méthode possible est décrite dans l'article "Efficient Dominating and Edge Dominating Sets for Graphs and Hypergraphs" de Brandstädt, Leitert et Dieter.

3 Le projet

La gestion des structures de données est libre mais devra être justifiée, vous pouvez utiliser les librairies usuelles si nécessaire (e.g. `numpy`, `networkx`, ...).

Le fichier qui sera exécuté devra être nommé `algo2_2018.py`. En plus des fonctions demandées, il contiendra une fonction `main` qui testera vos fonctions `max_subtree` et `test_hypertree` sur des

exemples générés aléatoirement de tailles raisonnables (maximum 15 sommets). Le scripte sera testé avec la commande :

```
python3 algo2_2018.py
```

et devra afficher à l'écran les (hyper)graphes générés (peu importe sous quelle forme) et les résultats des fonctions. La génération aléatoire des graphes ne doit pas suivre de règle particulière, mais doit être expliquée.

Il vous est également demandé un rapport (entre 4 et 6 pages environ), expliquant comment les structures sont gérés, vos choix d'implémentation, complexité des fonctions et détails des algorithmes utilisés plus exemples éventuels. N'oubliez pas de citer vos sources si vous en utilisez. Si vous utilisez des fonctions dans les librairies usuelles il vous faudra justifier la complexité exacte de celles-ci.

Consignes

Le projet peut être réalisé par groupe de deux élèves (pas plus). Les noms et matricules devront être indiqués clairement dans le code et sur le rapport. Une version électronique du projet est à remettre sur l'Université Virtuelle sous forme de deux fichiers :

le script `algo2_2018.py` et votre rapport `rapport_algo2_2018.pdf` au format pdf.

Une unique version papier sera à remettre au secrétariat informatique. Un seul des deux élèves du groupe doit donc remettre le projet sur l'UV, par souci de simplicité, au sein d'un groupe de deux, l'élève qui ne postera pas le projet sur l'UV postera un pdf avec les noms et matricules des membres de son groupe.

Votre code doit être clair et commenté, si votre code ne s'exécute pas, votre projet ne sera pas corrigé. Le plagiat sera sanctionné comme il se doit. L'efficacité de votre code et la justification de vos choix seront prise en compte dans la correction.

La date limite est le **15 novembre à 15h**. Une séance question-réponse sera prévue pendant le semaine tampon, envoyez vos questions pertinentes à l'adresse suivante : kmerckx (at) ulb.ac.be