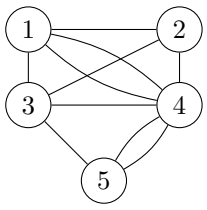
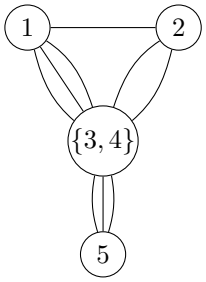
I Coupe minimum dans un graphe

Soit $G = (S, A)$ un **multigraphe**, c'est-à-dire un graphe où il peut y avoir plusieurs arêtes entre les mêmes sommets.
Une **coupe** de G est une partition de S en S_1, S_2. Sa **taille** est le nombre d'arêtes entre S_1 et S_2. Une coupe (S_1, S_2) est dite **minimum** si elle est de taille minimum parmi toutes les coupes de G.
Soient $s \neq t$ deux sommets de S. La **contraction** de $\{s, t\}$ dans G, notée $G/\{s, t\}$, est le multigraphe formé à partir de G où s et t ont été fusionnés, où les arêtes entre s et t ont disparu, et toute arête entre s et un sommet u devient une arête entre $\{s, t\}$ et u (et de même pour une arête entre t et u).

Exemple : Un multigraphe et sa contraction de $\{3, 4\}$:





On cherche à trouver une coupe minimum dans G par un algorithme randomisé.

1. Donner une coupe minimum sur le graphe de l'exemple (à gauche).
2. On suppose qu'il existe k coupes minimums. On tire aléatoirement et uniformément une coupe. Déterminer la probabilité qu'elle soit minimum en fonction de $n = |S|$ et k. Commenter.

On propose l'algorithme suivant :

Début algorithme
  Tant que $|S| > 2$ **:**
    Choisir $\{s, t\} \in A$ aléatoirement et uniformément.
    $G \leftarrow G/\{s, t\}$.
  Renvoyer la coupe (S_1, S_2), où S_1 et S_2 sont les deux sommets restants de G

3. Appliquer cet algorithme sur le graphe en exemple, en choisissant arbitrairement les arêtes à contracter.
4. En détaillant les structures de données choisies, déterminer la complexité temporelle de cet algorithme.
5. Soit C une coupe minimum G. Soit k la taille de C. Montrer que $|A| \geq \dfrac{nk}{2}$.
6. Montrer que l'algorithme donne C avec probabilité au moins $\displaystyle\prod_{i=1}^{n-2}(1 - \frac{2}{n-i+1})$, puis exprimer plus simplement cette probabilité.
7. Combien de fois peut-on répéter l'algorithme précédent pour obtenir une probabilité supérieure à $\dfrac{1}{e}$ d'obtenir une coupe minimum ?
8. Déterminer le nombre maximal de coupes minimums dans un multi-graphe d'ordre n.

1. Il s'agit ici de compter le nombre de coupes possibles, c'est-à-dire le nombre de partitions de taille 2 dans un ensemble à n éléments. Pour créer une coupe, on place le premier sommet dans V_1, et chaque autre sommet a 2 choix possibles (chacun des deux ensembles). Cela forme 2^{n-1} choix possibles, auquel il faut enlever le choix où tous les sommets se retrouvent dans V_1 (car ce n'est pas une coupe). Ainsi, la probabilité de choisir une coupe minimale si elle est unique est $\frac{1}{2^{n-1}-1}$. La probabilité cherchée est donc $\frac{k}{2^{n-1}-1}$.
2. En utilisant une représentation par matrice d'adjacence, une contraction peut être effectuée par un simple parcours de toute la matrice, soit $\mathcal{O}(n^2)$.
3. À tout moment, les sommets de G forment une partition de l'ensemble initial V. C'est effectivement un invariant de boucle :
   - Initialement, chaque sommet est un singleton et tous les sommets sont présents, donc c'est effectivement une partition ;
   - à chaque contraction, deux ensembles sont fusionnés, aucun sommet ne disparait lors de la fusion, et les autres ensembles ne sont pas modifiés. Cela reste une partition.

   À la fin de la boucle, il ne reste que deux sommets, donc une partition de taille 2. Il s'agit bien d'une coupe.
4. Supposons que la coupe minimale est unique et soit C l'ensemble des arêtes entre les deux ensembles de la coupe et $k = |C|$. Il faut calculer la probabilité qu'aucune arête de C ne soit contracté au cours de l'exécution de l'algorithme.

   Dans un premier temps, il est clair que le degré minimal d'un sommet de G est au moins k, sinon isoler ce sommet formerait une coupe avec moins d'arêtes traversantes que C. On en déduit que $|E| \geq \frac{nk}{2}$. Par ailleurs, la probabilité de tirer une arête de C lors de la première contraction est $\frac{k}{|E|} \leq \frac{2}{n}$. On en déduit que la probabilité d'éviter une arête de C est $\geq 1-\frac{2}{n}$. Après contraction, on obtient un graphe d'ordre $n-1$. Ainsi, la probabilité p_n d'obtenir une coupe minimale dans un graphe d'ordre n vérifie $p_n \geq \left(1-\frac{2}{n}\right) p_{n-1}$, soit $p_n \geq \prod_{i=0}^{n-3} \frac{n-i-2}{n-i} = \frac{2}{n(n-1)}$ (ce qui est bien mieux que le tirage au hasard d'une coupe).

   Pour améliorer la probabilité, on peut répéter l'algorithme plusieurs fois et garder la coupe minimale parmi les résultats obtenus. Par exemple, pour obtenir une probabilité au moins $\frac{n-1}{n}$, on peut répéter l'opération $\binom{n}{2} \ln n$ fois. En effet, la probabilité de ne jamais obtenir de coupe minimale est au plus :

$$\left(1-\frac{2}{n(n-1)}\right)^{\frac{n(n-1)}{2}\ln n} \leq \left(\frac{1}{e}\right)^{\ln n} = \frac{1}{n}$$

5. Pour montrer la borne sur le nombre de coupes minimales, comme l'algorithme de Karger a une probabilité au moins $\frac{2}{n(n-1)}$ de renvoyer une coupe minimale en particulier, il ne peut pas y avoir plus de $\frac{n(n-1)}{2}$ coupes minimales. De plus, cette borne est atteinte pour les cycles de taille n, donc la borne est optimale.

II Algorithme de Freivalds

On cherche à résoudre le problème suivant :

> Entrée : trois $n \times n$ matrices A, B et C de $\mathcal{M}_n(\mathbb{Q})$
> Sortie : est-ce que $AB = C$?

1. Quelle complexité en fonction de n peut-on obtenir en utilisant un algorithme naïf de multiplication de matrices pour résoudre ce problème ?

On propose l'algorithme suivant :

Début algorithme
Choisir aléatoirement et uniformément un vecteur colonne $X \in \mathcal{M}_{n,1}(\{0,1\})$.
Renvoyer $ABX = CX$.

2. Quelle est la complexité temporelle de l'algorithme précédent ?
3. Quelle est la probabilité de faux négatif ?

On pose $D = AB - C$. On pose de plus $Y = DX$. On cherche à déterminer la probabilité que $Y = 0$ sachant que $D \neq 0$. On suppose pour la question suivante que $D \neq 0$ et donc qu'il existe i, j tels que $d_{ij} \neq 0$.

4. En distinguant selon que $y_i - x_j d_{ij} = 0$ ou non, montrer que $\mathbb{P}(y_i = 0) \leq \dfrac{1}{2}$.
5. En déduire que la probabilité de faux positif est inférieure ou égale à $\dfrac{1}{2}$.
6. Quelle serait la complexité d'un algorithme ayant une probabilité de faux positif inférieure ou égale à $\varepsilon > 0$, en fonction de n et de ε ?

1. L'algorithme naïf de multiplication de matrices a une complexité en $\Theta(n^3)$, ce qui correspondrait au temps nécessaire pour résoudre le problème (car le test d'égalité est en $\mathrm{O}(n^2)$.
2. Il faut bien s'y prendre pour faire le calcul : on calcule $A(BX)$ et non $(AB)X$. Le calcul de BX, $A(BX)$ et CX se fait en $\mathrm{O}(n^2)$. La vérification d'égalité se fait en $\mathrm{O}(n)$, de même que le choix aléatoire de X. La complexité totale est donc en $\mathrm{O}(n^2)$.
3. L'algorithme ne peut pas renvoyer FAUX lorsque $AB = C$, car alors on aurait $ABX = CX$. La probabilité de faux négatif est donc nulle.
4. Si $y_i - x_j d_{ij} = 0$, alors $\mathbb{P}(y_i = 0) = \mathbb{P}(x_j = 0) = \dfrac{1}{2}$ (de par le choix aléatoire de X). Sinon, $\mathbb{P}(y_i = 0) \leq \mathbb{P}(xx_j = 1) = \dfrac{1}{2}$.
5. Un positif se produit lorsque $DX = 0$ alors que $D \neq 0$. Or la probabilité que chaque coefficient de DX soit nul est $\leq \dfrac{1}{2}$. On en déduit que la probabilité que DX soit nul est inférieure ou égale à $\leq \dfrac{1}{2}$ (attention, on ne peut pas majorer par $\dfrac{1}{2^n}$ car les événements ne sont pas indépendants).
6. On sait qu'en répétant l'algorithme k fois, la probabilité d'erreur est inférieure à $\dfrac{1}{2^k}$. On cherche donc k tel que $\dfrac{1}{2^k} \leq \varepsilon$, soit $k \geq -\log_2 \varepsilon$. La complexité totale est donc en $\mathrm{O}(n^2 \log(1/\varepsilon))$.