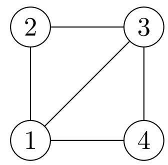
I Jeu de Shannon

Soit $G = (V, E)$ un graphe non orienté.

1. Donner une condition équivalente simple pour que G contienne un arbre couvrant. Comment trouver un arbre couvrant algorithmiquement ?

Si T_1 et T_2 sont deux arbres couvrants de G, on dit qu'ils sont disjoints s'ils n'ont aucune arête en commun.

2. Le graphe suivant possède t-il deux arbres couvrants disjoints ?



3. Soient T_1 et T_2 deux arbres couvrants de G et e_1 une arête de T_1. Montrer qu'il existe une arête e_2 de T_2 telle que $T_1 - e_1 + e_2$ (le graphe obtenu à partir de T_1 en enlevant e_1 et en ajoutant e_2) soit un arbre couvrant de G.
4. Soit T un arbre couvrant de G et e une arête de T. On contracte e dans T et G, c'est-à-dire qu'on supprime e et on identifie ses deux extrémités, pour obtenir T' et G'. Montrer que T' est un arbre couvrant de G'.

Si P est une partition de V, on note $|P|$ son cardinal et $\|P\|$ le nombre d'arêtes de G dont les deux extrémités sont dans des ensembles différents de P.

On s'intéresse maintenant au théorème suivant :

Théorème de Tutte

Soit $k \in \mathbb{N}^*$. G possède k arbres couvrants disjoints si et seulement si, pour toute partition P de V, $\|P\| \geq k(|P| - 1)$.

5. En admettant le théorème de Tutte, montrer que le problème suivant appartient à NP.

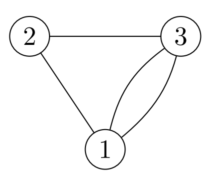
   co-PACKING-TREES

   Entrée : Un graphe $G = (V, E)$ et un entier k.
   Question : est-il faux que G possède k arbres couvrants disjoints ?

6. Montrer le théorème de Tutte pour $k = 1$.
7. Montrer le sens direct du théorème de Tutte : si G possède k arbres couvrants disjoints, alors $\|P\| \geq k(|P| - 1)$.

On considère un jeu avec un graphe $G = (V, E)$ non orienté qui peut posséder plusieurs arêtes entre deux sommets.
Deux joueurs A et B, où A commence, choisissent alternativement une arête de G non encore choisie. Si, à un moment de la partie, les arêtes choisies par B forment un arbre couvrant de G alors B gagne. Sinon, A gagne.

8. Indiquer si B a une stratégie gagnante si G est le graphe ci-dessous.



Dans la suite, on veut montrer que B possède une stratégie gagnante si et seulement si G possède deux arbres couvrants disjoints. Pour cela, on admet le théorème de Tutte.

9. Supposons qu'il existe une partition P de V telle que $\|P\| < 2(|P| - 1)$. Montrer que A a une stratégie gagnante.
10. Supposons qu'il existe deux arbres couvrants disjoints T_1 et T_2 dans G. Montrer que B a une stratégie gagnante.
    On pourra raisonner par récurrence sur $|V|$.

II Sous-graphe le plus dense

Soit $G = (V, E)$ un graphe non orienté à n sommets et p arêtes. Pour $S \subseteq V$, on définit la fonction de densité par :

$$\rho(S) = \frac{|E(S)|}{|S|}$$

où $E(S)$ est l'ensemble des arêtes de G ayant leurs deux extrémités dans S.

1. Quelles sont les valeurs minimum et maximum de $\rho(S)$, en fonction de $|S|$?
2. Quel est le lien entre $\rho(S)$ et le degré moyen des sommets dans S ?

On s'intéresse aux problèmes suivants :

DENSEST

Entrée : un graphe $G = (V, E)$.
Sortie : un ensemble $S \subseteq V$ tel que $\rho(S)$ soit maximum.

DENSEST-DEC

Entrée : un graphe $G = (V, E)$, un entier k et un réel α.
Sortie : existe t-il un ensemble $S \subseteq V$ tel que $|S| = k$ et $\rho(S) \geq \alpha$?

CLIQUE-DEC

Entrée : un graphe $G = (V, E)$ et un entier k.
Sortie : existe t-il un ensemble $S \subseteq V$ tel que $|S| = k$ et tous les sommets de S sont adjacents ($\forall u, v \in S, \{u, v\} \in E$) ?

3. En admettant que CLIQUE-DEC est NP-complet, montrer que DENSEST-DEC est NP-complet.

On propose un algorithme glouton pour DENSEST :

- Itérativement retirer un sommet de degré minimum (ainsi que tous les sommets adjacents) jusqu'à ce qu'il n'y ait plus de sommet.
- À chacune de ces itérations, calculer la valeur de ρ et conserver le maximum.

4. Expliquer comment on pourrait implémenter cet algorithme en complexité temporelle $\mathrm{O}(n + p)$.

Soit S^* tel que $\rho(S^*)$ soit maximum, $v^* \in S^*$ le premier sommet de S^* retiré par l'algorithme glouton et S' l'ensemble des sommets restants juste avant de retirer v^*.

5. Montrer que $\rho(S') \geq \dfrac{\deg_{S'}(v^*)}{2}$, où $\deg_{S'}(v^*)$ est le degré de v^* dans S'.
6. Justifier que $\rho(S^*) \geq \rho(S^* \setminus \{v^*\})$.
7. En déduire que $\deg_{S^*}(v^*) \geq \rho(S^*)$.
8. En déduire que l'algorithme glouton est une 2-approximation pour DENSEST.

III Dominant

Soit $G = (V, E)$ un graphe. Un ensemble dominant de G est un sous-ensemble D de V tel que tout sommet de V est soit dans D, soit adjacent à un sommet de D.
On note $d(G)$ la taille d'un plus petit ensemble dominant de G.

1. Calculer $d(G)$ si G est un chemin à n sommets.
2. On suppose que G est connexe et contient au moins 2 sommets. Montrer que $d(G) \leq \left\lfloor \frac{n}{2} \right\rfloor$.
3. On suppose que G ne contient pas de sommet isolé (sommet de degré 0). Montrer que $d(G) \leq \left\lfloor \frac{n}{2} \right\rfloor$.

Une couverture par sommets de G est un sous-ensemble C de V tel que toute arête de G ait au moins une extrémité dans C.
On s'intéresse aux problèmes suivants :

DOMINANT

Entrée : Un graphe $G = (V, E)$ et un entier k.
Sortie : G possède-t-il un ensemble dominant de taille k ?

COUVERTURE

Entrée : Un graphe $G = (V, E)$ et un entier k.
Sortie : G possède-t-il une couverture par sommets de taille k ?

4. Soit G un graphe sans sommet isolé. Est-ce qu'une couverture par sommets est un ensemble dominant ? Et réciproquement ?
5. On admet que COUVERTURE est NP-complet. Montrer que DOMINANT est NP-complet.
6. Décrire un algorithme efficace pour résoudre DOMINANT si G est un arbre. L'implémenter en OCaml.

IV Recherche de doublon

IV.1 Doublon dans un tableau

Soit t un tableau de taille n dont les éléments sont entre 0 et $n-1$ (inclus).
On veut déterminer si t contient un doublon, c'est-à-dire un élément apparaissant plusieurs fois.

1. Donner un algorithme en complexité temporelle $\mathrm{O}(n)$ pour résoudre ce problème. Quelle est la complexité spatiale ?
2. Peut-on adapter l'algorithme précédent si les éléments de t ne sont pas entre 0 et $n-1$? pas forcément entiers ?
3. On reprend l'hypothèse où les éléments de t sont entre 0 et $n-1$.
   Décrire un algorithme en complexité $\mathrm{O}(n)$ en temps et $\mathrm{O}(1)$ en mémoire. On pourra modifier t.

IV.2 Cycle dans une liste chaînée

On considère un type `linked_list` de liste simplement chaînée impérative (chaque élément a accès à l'élément suivant `next`) :

```c
typedef struct cell {
    int elem;
    struct cell *next;
} cell;

typedef cell *linked_list;
```

Il est possible qu'une liste chaînée `l` possède un cycle, si l'on revient sur le même élément après avoir parcouru plusieurs successeurs.

4. Décrire un algorithme naïf pour tester si `l` contient un cycle. Quelle est sa complexité en temps et en espace ?

L'algorithme de Floyd est plus efficace. Il consiste à initialiser une variable `tortue` au premier élément de `l`, une variable `lievre` à la case suivante, puis, tant que c'est possible :

- Si `lievre` et `tortue` font référence à la même case, affirmer que `l` contient un cycle.
- Sinon, avancer `lievre` de deux cases et `tortue` d'une case.

5. Montrer que cet algorithme permet bien de détecter un cycle dans `l`. Quelle est l'intérêt de cet algorithme par rapport à celui de la question 4 ?
6. Écrire une fonction `bool has_cycle(linked_list l)` détectant un cycle en utilisant l'algorithme du lièvre et de la tortue.
7. Expliquer comment obtenir la longueur T du cycle ainsi que le nombre d'itérations L avant d'entrer dans le cycle.

Soit t un tableau contenant n entiers entre 0 et $n-2$ (inclus).

8. Montrer que t contient un doublon.
9. Expliquer comment utiliser l'algorithme de Floyd pour déterminer un doublon de t en complexité $\mathrm{O}(n)$ en temps et $\mathrm{O}(1)$ en mémoire, sans modifier t.

IV.3 Presque doublon

On considère un nouveau problème :

Entrée : un tableau t de n entiers et deux entiers a, b.
Sortie : un booléen indiquant s'il existe deux indices $i \neq j$ tels que $|i-j| \leq a$ et $|t[i]-t[j]| \leq b$.

10. Décrire un algorithme en complexité temporelle $\mathrm{O}(n)$ en utilisant une table de hachage.

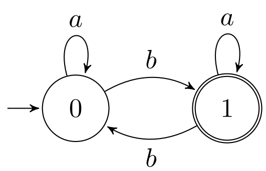
V Morphisme d'automate

Dans toute la suite, les automates sont **déterministes** et **complets**.
On fixe l'alphabet $\Sigma = \{a, b\}$.
Soient deux automates $A = (Q_A, i_A, \delta_A, F_A)$ (où Q_A est l'ensemble d'états, i_A l'état initial, δ_A la fonction de transition et F_A les états finaux) et $A' = (Q_{A'}, i_{A'}, \delta_{A'}, F_{A'})$. Une fonction $f : Q_A \longrightarrow Q_{A'}$ est un **morphisme d'automates** si :

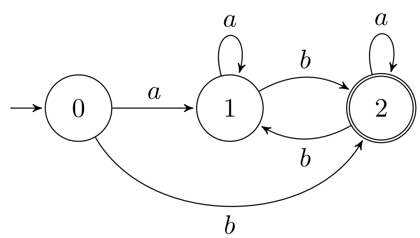
(i) $f(i_A) = i_{A'}$

(ii) $\forall q \in Q_A, \forall a \in \Sigma : \ f(\delta_A(q, a)) = \delta_{A'}(f(q), a)$

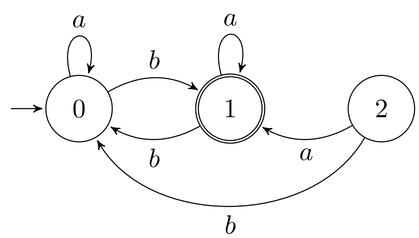
(iii) $\forall q \in Q_A : \ f(q) \in F_{A'} \iff q \in F_A$



A_1



A_2



A_3

1. Expliciter un morphisme de A_2 vers A_1.
2. Montrer qu'il n'existe pas de morphisme de A_3 vers A_1.
3. Montrer que s'il existe un morphisme f de A vers A' alors A et A' acceptent le même langage. La réciproque est-elle vraie ?

On suppose que A' est **accessible**, c'est-à-dire que tout état de A' est accessible depuis l'état initial i'.

4. On suppose qu'il existe un morphisme f de A vers A'. Montrer que f est surjective.
5. Décrire un algorithme permettant de savoir s'il existe un morphisme de A vers A' et de le calculer s'il existe. On précisera les structures de données utilisées et la complexité.

On définit l'**automate produit** $A \times A' = (Q_A \times Q_{A'}, (i_A, i_{A'}), \delta_{A \times A'}, F_A \times F_{A'})$ où $\delta_{A \times A'}((q, q'), a) = (\delta_A(q, a), \delta_{A'}(q', a))$.

6. On suppose que $A \times A'$ est accessible et $L(A) = L(A')$. Montrer qu'il existe un morphisme de $A \times A'$ vers A (et donc aussi une morphisme de $A \times A'$ vers A').

Soit $B = (Q_B, i_B, \delta_B, F_B)$ un automate accessible. On suppose qu'il existe un morphisme f de B vers A et un morphisme g de B vers A'.
On veut trouver un automate C et des morphismes f' de A vers C et g' de A' vers C.
Pour cela, on introduit la relation d'équivalence suivante sur Q_B :

$$p \equiv q \iff \exists q_0, q_1, ..., q_k \text{ états de } Q_B \text{ tels que } q_0 = p, q_k = q \text{ et } \forall i \in [\![0, k-1]\!], \ f(q_i) = f(q_{i+1}) \text{ ou } g(q_i) = g(q_{i+1})$$

7. Montrer que si $p \equiv q$ alors $\forall x \in \Sigma, \ \delta_B(p, x) \equiv \delta_B(q, x)$.
8. Montrer que si $p \equiv q$ alors p est un état final si et seulement si q est un état final.

On note $[q]$ la **classe d'équivalence** de $q \in Q_B$ pour la relation $\equiv$ et Q_C l'ensemble des classes d'équivalence de $\equiv$.

9. Décrire un algorithme pour calculer Q_C. On précisera les structures de données utilisées et la complexité.
10. Définir un automate C dont l'ensemble d'états est Q_C et tel que $h : q \longrightarrow [q]$ soit un morphisme de B vers C.
11. Définir deux morphismes f' de A vers C et g' de A' vers C tels que $f' \circ f = h = g' \circ g$.

Soit L un langage rationnel et n le plus petit nombre d'états d'un automate reconnaissant L.

12. Montrer que deux automates à n états reconnaissant L sont isomorphes, c'est à dire qu'il existe un morphisme bijectif de l'un à l'autre.