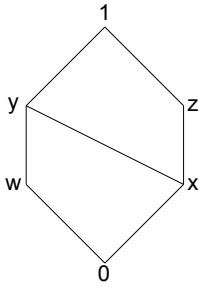
Licence L2
Mathématiques pour l'informatique
Partiel (1h30)

Exercice 1

Soit une relation d'ordre $\leq$ sur un ensemble E définie par le diagramme de Hasse suivant :



1) L'ensemble ordonné $(E, \leq)$ est-il un treillis? Justifiez votre réponse.

Correction : Un ensemble ordonné $(E, \leq)$ est un treillis si et seulement si $\forall a, b \in E$ il existe $a \wedge b$ et $a \vee b$.

Parmi les couples pouvant posés problèmes :

$z \vee y = 1$ et $z \wedge y = x$

$w \vee z = 1$ et $w \wedge z = 0$

$w \vee x = y$ et $w \wedge x = 0$

Les autres couples sont comparables et donc les bornes inférieur et supérieur sont respectivement le minimum et le maximum.

2) S'il s'agit d'un treillis, indiquer s'il est distributif, modulaire et/ou complémenté. Justifiez votre réponse.

Correction : premièrement, un treillis est distributif si on ne peut pas extraire ni de sous-treillis de type K ni de sous-treillis de type N.

Dans notre cas, on ne peut extraire de sous-treillis de type type N, car, à partir de n'importe quel élément, il n'existe pas 3 autres éléments non comparable entre eux.

Ensuite, concernant les éventuelles sous-treillis de type K, on peut essayer avec les deux seules possibilités suivantes :

- On tente de construire un sous-treillis T_1 en sélectionnant les éléments 0, w, z, y et 1. Ce n'est pas un sous-treillis car $y \wedge z = x \notin T_1$.
- On tente de construire un sous-treillis T_2 en sélectionnant les éléments 0, w, z, x et 1. Ce n'est pas un sous-treillis car $w \vee x = y \notin T_2$.

On ne peut donc pas construire de sous-treillis de type K non plus. Donc le treillis est distributif.

Deuxièmement, on sait qu'un treillis distributif est également modulaire. Puisque l'on vient de démontrer que le treillis est distributif, alors il est aussi modulaire.

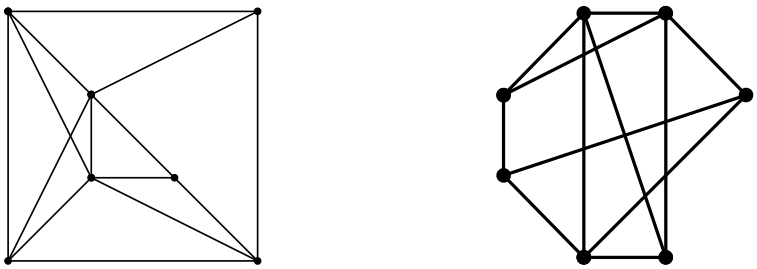
Enfin, on souhaite montrer si le treillis est complémenté (c'est à dire que tout ses éléments possèdent un complément). Essayons de trouver le complément de x :

- $x \vee z = z \neq 1$. Donc z n'est pas le complément de x.
- $x \vee w = y \neq 1$. Donc x n'est pas le complément de x.
- $x \vee y = y \neq 1$. Donc y n'est pas le complément de x.

Ainsi, x ne possède pas de complément ce qui implique que le treillis n'est pas complémenté.

Exercice 2

Soient deux graphes G_1 et G_2 définis par les représentations de la figure 1.



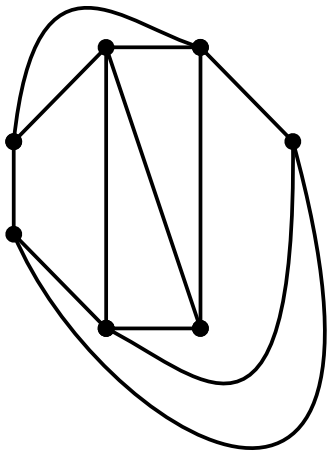
(a) G_1 (b) G_2

Figure 1

1) Parmi G_1 et G_2, indiquez celui qui est planaire et fournissez une représentation planaire lorsque cela est possible.

G_1 n'est pas un graphe planaire car on ne peut le représenter sans croiser au moins deux de ses arêtes.

G_2 est bien un graphe planaire car on peut le représenter tel que ci-dessous :



2) Pour le graphe planaire de la question précédente, indiquez son nombre de face, son nombre d'arêtes frontières et vérifier le théorème d'Euler.

Nombre de faces de G_2 : 7

Nombre d'arêtes frontières de G_2 : 12

Formule d'Euler : $f = m-n+c+1$ tel que f est le nombre de face, m est le nombre d'arêtes, n est le nombre de sommets et c est le nombre de composantes connexes. Donc, pour G_2, $f = 12-7+1+1 = 7$.

3) Donnez la matrice d'adjacence du graphe non planaire.

Tout d'abords, on nomme les sommets du graphe comme sur la figure ci-dessous.

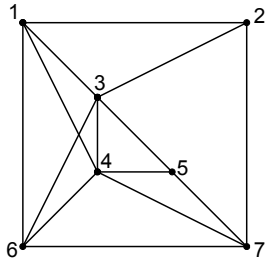


Figure 2: G_1 avec ses sommets nommés

Ensuite, on construit M_{G_1}, la matrice d'adjacence du graphe G_1 tel que :

$$M_{G_1} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

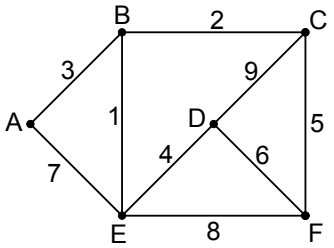
Exercice 3

L'algorithme de Kruskal (voir ci-dessous l'algorithme 1) a été proposé en 1956 par Joseph Kruskal. Il permet de rechercher un arbre couvrant de poids minimal à l'instar des algorithmes de Prim et de Sollin.

Algorithm 1 Algorithme de Kruskal

Entrée: G : le graphe à recouvrir
S : L'ensemble des sommets de G
A : l'ensemble des arêtes de G
Sortie: T : un arbre ou une forêt couvrant(e) de poids minimal du graphe G.
Début
▶ T est l'arbre ou la forêt couvrant(e) à retourner
▶ à l'initialisation, T contient autant de composante connexe que de sommets
$T \leftarrow (S, \emptyset)$
▶ A_t contient la liste des arêtes triés par ordre croissant de leur poids associé
▶ Ainsi, $A_t[0]$ correspond à l'arête de poids le plus faible
▶ et $A_t[\text{card}(S) - 1]$ correspond à l'arête de poids le plus fort
$A_t \leftarrow \text{trieCroissant}(A)$
Pour i de 1 à $\text{card}(S) - 1$ **faire**
    $a \leftarrow A_t[0]$
    $A_t \leftarrow A_t \setminus a$
    Si a connecte deux composantes connexes différentes **alors**
        $T \leftarrow T \cup a$
    Fin Si
Fin Pour
Retourner T
Fin

Soit le graphe G_3 défini par la représentation ci-dessous :



1) Appliquer l'algorithme de Kruskal à G_3 en détaillant pour chaque itération k :

- La liste des arêtes présentes dans T_k.
- Le poids $p(T_k)$ de T_k.

Correction :

Soit $G_3 = (S, A)$. Soient T_k la forêt couvrante à l'itération k et $P(T_k)$ le poids de l'arbre à l'itération K.

Initialisation :

$T = (S, \emptyset)$

$A_t = \{\{BE\}, \{BC\}, \{BA\}, \{DE\}, \{CF\}, \{DF\}, \{AE\}, \{EF\}\}$

$p(T_0) = 0$

Itération 1 :

$T_1 = (S, \{\{BE\}\})$

$P(T_1) = 1$

Itération 2 :

$T_2 = (S, \{\{BE\}, \{BC\}\})$

$P(T_2) = 3$

Itération 3 :

$T_3 = (S, \{\{BE\}, \{BC\}, \{BA\}\})$

$P(T_3) = 6$

Itération 4 :

$T_4 = (S, \{\{BE\}, \{BC\}, \{BA\}, \{DE\}\})$

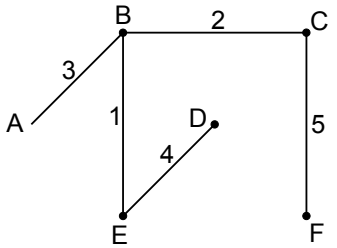
$P(T_4) = 10$

Itération 5 :

$T_5 = (S, \{\{BE\}, \{BC\}, \{BA\}, \{DE\}, \{CF\}\})$

$P(T_5) = 15$

2) Représenter graphiquement l'arbre couvrant de poids minimal obtenu à la question 1).



Exercice 4

Après avoir spécifié la représentation informatique d'un graphe que vous allez utiliser, vous écrirez un algorithme qui retourne vrai si le graphe est connexe et faux sinon. Vous pourrez vous inspirez des

algorithmes vu en cours.

Correction : On utilise la programmation orientée objet afin de représenter un graphe. La classe "Sommet" contient une variable "connexions" qui correspond à l'ensemble des sommets connectés au sommet courant par une arête. La classe "Graphe" possède la variable "sommets" correspondant à l'ensemble des sommets du graphe.

Algorithm 2 estConnexe(G) de type booléen

Entrée: G : le graphe à tester
Sortie: Vrai si G est connexe, faux sinon.
Début

```
1:  ▶ M représente l'ensemble des sommets marqués
2:  M ← ∅
3:  ▶ N représente l'ensemble des sommets non marqués
4:  N ← G.sommets
5:  ▶ C représente l'ensemble des sommets à explorer
6:  C ← { un sommet quelconque de N }
7:  Tant que C ≠ ∅ faire
8:      s_c ← C[0]
9:      M ← M ∪ s_c
10:     N ← N \ s_c
11:     C ← C \ s_c
12:     Pour chaque s_v dans s_c.connexions faire
13:         Si s_v ∉ M et s_v ∉ C alors
14:             C ← C ∪ s_v
15:         Fin Si
16:     Fin Pour
17: Fin Tant que
18: Si N ≠ ∅ alors
19:     ▶ On n'a pas pu marquer tous les sommets
20:     ▶ donc le graphe n'est pas connexe
21:     Retourner faux
22: Fin Si
23: Retourner vrai
```

Fin

Il existe bien-sûr bien d'autre manière de mettre en évidence la connexité. Par exemple, l'algorithme de Prim s'arête s'il n'arrive pas à construire un arbre couvrant par manque de connexité...