

DS 5

Devoir sur table

(3 heures)

Le sujet comporte deux parties totalement indépendantes.

Partie I Logique et calcul des propositions

Dans la suite, les variables propositionnelles seront notées x_1, x_2, \dots . Les connecteurs propositionnels \wedge (conjonction), \vee (disjonction), \Rightarrow (implication) et \Leftrightarrow (équivalence) seront classiquement utilisés.

De même, la négation d'une variable propositionnelle x_i (respectivement d'une formule \mathcal{F}) sera notée $\neg x_i$ (resp. $\neg \mathcal{F}$).

I. 1 - Définitions

Définition 1 (Minterme, maxterme)

Soit $(x_1 \dots x_n)$ un ensemble de n variables propositionnelles.

- On appelle minterme toute formule de la forme $y_1 \wedge y_2 \wedge \dots \wedge y_n$ où pour tout $i \in \{1, \dots, n\}$ y_i est un élément de $\{x_i, \neg x_i\}$.
- On appelle maxterme toute formule de la forme $y_1 \vee y_2 \vee \dots \vee y_n$ où pour tout $i \in \{1, \dots, n\}$ y_i est un élément de $\{x_i, \neg x_i\}$.

Les mintermes (respectivement maxtermes) $y_1 \wedge y_2 \wedge \dots \wedge y_n$ et $y'_1 \wedge y'_2 \wedge \dots \wedge y'_n$ (resp $y_1 \vee y_2 \vee \dots \vee y_n$ et $y'_1 \vee y'_2 \vee \dots \vee y'_n$) sont considérés identiques si les ensembles $\{y_i, 1 \leq i \leq n\}$ et $\{y'_i, 1 \leq i \leq n\}$ le sont.

Q1. Donner l'ensemble des mintermes et des maxtermes sur l'ensemble (x_1, x_2) .

Définition 2 (Formes normales conjonctives et disjonctives)

Soit \mathcal{F} une formule propositionnelle qui s'écrit à l'aide de n variables propositionnelles $(x_1 \dots x_n)$.

- On appelle forme normale conjonctive de \mathcal{F} toute conjonction de maxtermes logiquement équivalente à \mathcal{F} .
- On appelle forme normale disjonctive de \mathcal{F} toute disjonction de maxtermes logiquement équivalente à \mathcal{F} .

Définition 3 (Système complet)

Un ensemble de connecteurs logiques \mathcal{C} est un système complet si toute formule propositionnelle est équivalente à une formule n'utilisant que les connecteurs de \mathcal{C} .

Par définition, $\mathcal{C} = \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$ est un système complet.

I. 2 - Le connecteur de Sheffer

On définit le connecteur de Sheffer, ou d'incompatibilité, par $x_1 \diamond x_2 = \neg x_1 \vee \neg x_2$.

Q2. Construire la table de vérité du connecteur de Sheffer.

Q3. Exprimer ce connecteur en fonction de \neg et \wedge .

Q4. Vérifier que $\neg x_1 = x_1 \diamond x_1$

Q5. En déduire une expression des connecteurs \wedge , \vee et \Rightarrow en fonction du connecteur de Sheffer.

Justifier en utilisant des équivalences avec les formules propositionnelles classiques.

Q6. Donner une forme normale conjonctive de la formule $x_1 \diamond x_2$.

Q7. Donner de même une forme normale disjonctive de la formule $x_1 \diamond x_2$.

Q8. Démontrer par induction sur les formules propositionnelles que l'ensemble de connecteurs $\mathcal{C} = \{\diamond\}$ est un système complet.

Q9. Application : soit \mathcal{F} la formule propositionnelle $x_1 \vee (\neg x_2 \wedge x_3)$. Donner une forme logiquement équivalente de \mathcal{F} utilisant uniquement le connecteur de Sheffer.

Partie II Tri topologique.

Le professeur Cosinus aide régulièrement le professeur Sinus à apprendre les gestes simples de la vie quotidienne. Aujourd'hui, la leçon porte sur l'ordre dans lequel il faut mettre ses vêtements afin de s'habiller correctement pour le bal. Voici deux exemples de règles :

- Il n'est pas possible de mettre sa cravate avant sa chemise.
- Les chaussures doivent être mises après le caleçon, le pantalon et les chaussettes.

Pour aider son ami à comprendre les différentes règles, le professeur Cosinus décide d'utiliser une représentation sous forme de graphe orienté (voir la figure 1). Dans ce graphe, chaque arc représente une contrainte : s'il existe un arc entre le sommet A et le sommet B , cela signifie que le vêtement A doit être mis avant le vêtement B . En particulier, les deux règles énoncées ci-dessus se déduisent bien du graphe de la figure 1.

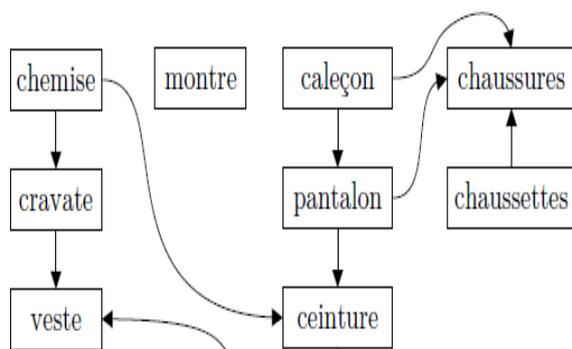


FIGURE 1

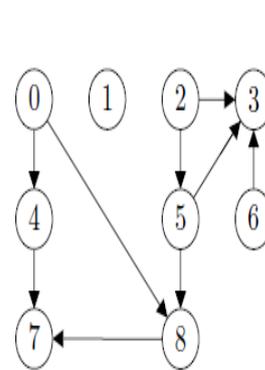


FIGURE 2

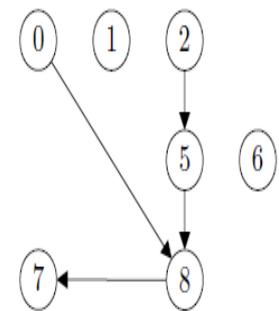


FIGURE 3

L'objectif du professeur Sinus est, à partir de la figure 1, de déterminer l'ordre dans lequel mettre ses vêtements. Voici une possibilité :

(T_1) : chaussettes, caleçon, pantalon, chaussures, montre, chemise, ceinture, cravate, veste.

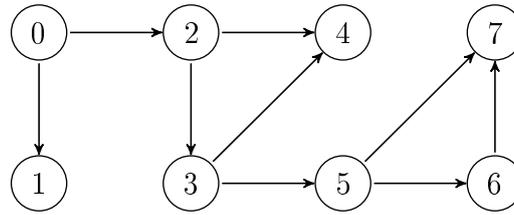
Afin de formaliser le problème, on considère un graphe orienté $G = (S, A)$ dont les sommets sont numérotés de 0 à $|S| - 1$ (la figure 2 montre une numérotation possible des sommets de la figure 1).

Dans la suite, on ne fera pas de distinction entre un "sommet u " et le "numéro du sommet u ". Notre but est de calculer un **tri topologique** du graphe, c'est à dire de déterminer un tableau (`tab`: `int array`) de taille $|S|$ contenant une et une seule fois chaque entier compris entre 0 et $|S| - 1$, et tel que si G contient l'arc (u, v) alors u apparaît strictement avant v dans `tab`. Par exemple, le tri topologique (T_1) correspond à `tab_1` :

```
let tab_1 = [16; 2; 5; 3; 1; 0; 8; 4; 7]
```

II. 1 - Un autre exemple

On considère le graphe `graphe_ex_2` suivant :



Représentation du `graphe_ex_2`

Q10. Justifier que le tableau suivant n'est pas un tri topologique pour le `graphe_ex_2` :

```
let tab_2 = [| 0;2;4;1;3;5;6;7 |]
```

Q11. Proposer un tri topologique pour le `graphe_ex_2` .

II. 2 - Représentation des graphes en OCaml

Listes de successeurs. En OCaml, un graphe orienté $G = (S, A)$ est donné par un tableau contenant les listes de successeurs des sommets. Par exemple, le graphe de la figure 2 correspond à `g_fig2` :

```
type graphe = int list array ;;
let g_fig2 = [| [4;8]; []; [3;5]; []; [7]; [3;8]; [3]; []; [7] |];;
```

Q12. Soit $(g: \text{graphe})$ un graphe avec n sommets et $(t1: \text{int array})$ un tableau de taille n contenant une et une seule fois chaque élément de $\llbracket 0, n-1 \rrbracket$.

Q12.(a) Écrire une fonction `inverse` qui prend en entrée `t1` et renvoie un tableau $(t2: \text{int array})$ de taille n tel que `t2.(i)` est l'indice de i dans `t1`. Par exemple :

```
inverse([|6; 2; 5; 3; 1; 0; 8; 4; 7|]);;
- : int array = [|5; 4; 1; 3; 7; 2; 0; 8; 6|]
```

Q12.(b) Écrire une fonction `est_tri_topo` qui prend en entrée `g` et `t_1`, et renvoie un booléen qui indique si `t_1` est un tri topologique de `g`. Le temps d'exécution devra être en $O(n + m)$ où n et m sont les nombres de sommets et d'arcs. (On pourra écrire une fonction auxiliaire pour gérer la relation d'ordre d'un sommet et la liste de ces successeurs.)

Q12.(c) Justifier la complexité de la fonction `est_tri_topo`.

Sous-graphes induits. On dit que G est un sous-graphe induit de G_0 si G peut être obtenu en supprimant certains sommets dans G_0 .

Formellement, $G = (S, A)$ est un sous-graphe induit de $G_0 = (S_0, A_0)$ si et seulement si :

$$S \subset S_0 \quad \text{et} \quad A = \{(u, v) \in A_0 : u \in S, v \in S\}$$

Par exemple, le graphe de la figure 3 est un sous-graphe induit de la figure 2, il est obtenu en supprimant les sommets numérotés 3 et 4.

En OCaml, un sous-graphe induit G est représenté par une variable (`g: graphe_induit`) telle que `g.g0` correspond au graphe initial G_0 et `g.s.(i)` vaut `true` si et seulement le sommet numéro i est un sommet de G . Par exemple, pour le graphe de la figure 3, on obtient `g_fig3` :

```
type graphe_induit = {g0: graphe ; s : bool array};;
```

```
let g_fig3 = { g0 = g_fig2 ;
s = [| true ; true ; true ; false ; false ; true ; true ; true ; true |] } ;;
```

Q13. Écrire une fonction `suppr_sommets: graphe_induit -> int list -> unit` qui prend en entrée un graphe G ainsi qu'une liste l_i , et supprime de G tous les sommets appartenant à l_i . On pourra supposer sans le vérifier que l_i ne contient que des sommets de G .

Q14.

Q14.(a) Écrire une fonction `deg_entrant` qui prend en entrée un graphe (`g: graphe_induit`) avec n sommets et renvoie un tableau (`deg: int array`) de taille n tels que `deg.(i)` est le degré entrant du sommet i dans G . Si i n'est pas un sommet de G , `deg.(i)` est fixé à -1 .

Q14.(b) Quelle est la complexité de votre fonction ? Justifier.

II. 3 - Calcul d'un tri topologique

Pour rappel, un **circuit** est un chemin de longueur strictement positive, dont les arcs sont distincts deux à deux et dont les sommets de départ et d'arrivée sont identiques.

Q15. Montrer que s'il existe un circuit dans un graphe G alors il n'existe pas de tri topologique pour G .

Dans la suite de cette partie, tous les graphes considérés ne contiennent pas de circuit.

II. 3-1 - Première méthode

Le professeur Sinus propose une solution pour obtenir un tri topologique à partir du graphe de la figure 1. Il remarque tout d'abord que les sommets étiquetés par "caleçon", "chaussettes", "montre" et "chemise" ont tous les quatre un degré entrant égal à 0. On peut donc les placer au début du tri topologique (dans un ordre arbitraire). En supprimant ces quatre sommets du graphe, les sommets "pantalon" et "cravate" ont pour degré entrant 0 et peuvent donc être placés à la suite dans le tri topologique. On continue ainsi jusqu'à ce que tous les sommets aient été supprimés du graphe. Avec cette procédure on obtient :

(T2) : caleçon, chaussettes, montre, chemise, pantalon, cravate, chaussures, ceinture, veste.

Q16. Appliquer l'algorithme précédent sur le graphe `graphe_ex_2` pour déterminer un tri topologique.

Q17.

Q17.(a) En généralisant cette idée, écrire une fonction (`tri_topo1: graphe -> int array`) qui renvoie un tri topologique du graphe donné en entrée. On attend ici que vous utilisiez le type `graphe_induit` de la partie 1.

Q17.(b) Justifier qu'à chaque étape de la fonction `tri_topo1`, il existe au moins un sommet dont le degré entrant est 0. On rappelle qu'on a supposé que le graphe ne contient pas de circuit.

Q17.(c) Quelle est la complexité de votre fonction ?

II. 3-2 - Deuxième méthode

Le professeur Cosinus pense que la solution de la partie 2.1 n'est pas optimale en terme de complexité. Il pense pouvoir utiliser un parcours en profondeur pour calculer le tri topologique. Voici pour commencer un pseudo-code décrivant le parcours en profondeur à partir d'un sommet s :

Algorithme 1 : – PP(s) – Parcours en profondeur à partir du sommet s

‡ Début de traitement du sommet s

```

pour chaque sommet  $t$  voisin de  $s$  faire
    si  $t$  a déjà été rencontré lors d'un parcours précédent alors
        ne rien faire
    sinon
        appeler récursivement PP( $t$ )
    fin si
fin pour

```

‡ Fin de traitement du sommet s

Étant donné qu'un appel à la procédure PP n'atteint pas forcément tous les sommets du graphe, il est parfois nécessaire d'effectuer plusieurs parcours en profondeur. La procédure principale que nous utiliserons est la suivante :

Algorithme 2 : Procédure principale

```

pour chaque sommet  $s$  du graphe faire
    si  $s$  a déjà été rencontré lors d'un parcours précédent alors
        ne rien faire
    sinon
        appeler PP( $s$ )
    fin si
fin pour

```

Appliquons l'algorithme 2 avec le graphe $G = (S, A)$ de la figure 2. Voici les différents évènements qui se produisent : début de traitement du sommet 0, début de traitement du sommet 4, début de traitement du sommet 7, fin de traitement du sommet 7, fin de traitement du sommet 4, début de traitement du sommet 8, fin de traitement du sommet 8, fin de traitement du sommet 0. La procédure principale effectue ensuite d'autres appels à la fonction PP, par exemple à partir du sommet 1, puis à partir du sommet 2 et enfin à partir du sommet 3. Ces évènements peuvent maintenant être numérotés de 1 à $2|S|$ en suivant l'ordre dans lequel ils se produisent. On obtient :

Sommet s	0	1	2	3	4	5	6	7	8
deb(s)	1	9	11	12	2	14	17	3	6
fin(s)	8	10	16	13	5	15	18	4	7

où deb(s) et fin(s) sont les instants de début et de fin de traitement du sommet s . On appelle **ordre préfixe** (resp. **ordre postfixe**) la liste des sommets triés par ordre croissant de début (resp. fin) de traitement. On obtient :

Ordre préfixe : (0,4,7,8,1,2,3,5,6) , Ordre postfixe : (7,4,8,0,1,3,5,2,6).

Q18. Appliquer l'algorithme au graphe `graphe_ex_2` en choisissant les voisins dans l'ordre croissant. On donnera les deux ordres définis précédemment.

Q19. Supposons avoir exécuté l'algorithme 2 sur un graphe $G = (S, A)$ ne contenant pas de circuit.

Q19.(a) Montrer qu'il n'existe pas d'arc $(u, v) \in A$ tel que $\text{deb}(v) < \text{deb}(u)$ et $\text{fin}(v) > \text{fin}(u)$.

Q19.(b) Montrer que l'ordre postfixe lu à l'envers est un tri topologique.

Q19.(c) En déduire par cette méthode un tri topologique du graphe `graphe_ex_2`.

Q20.

Q20.(a) Écrire une fonction (`tri_topo2: graphe -> int array`) qui renvoie le tri topologique obtenu avec la méthode donnée ci-dessus.

Q20.(b) Quelle est la complexité de votre fonction ?

FIN