

Logique et Algorithmie - 2014

Voilà une petite série d'exercices pour vous entraîner sur les notions de logiques et d'algorithmie que nous avons vu lors des deux premiers cours. À titre indicatif : ce document a été rédigé à l'aide du langage de formatage \TeX inventé par Donald Knuth lorsqu'il rédigeait *The Art Of Computer Programming* en 1977 !

Logique :

1. Écrivez les tables de vérité des propositions suivantes :

Hint: Pour être sûr de ne pas en oublier : il y a 2^n lignes lorsqu'il y a n variables booléennes différentes. Par exemple si les variables sont P , Q et R alors il y aura $2^3 = 8$ lignes dans notre table de vérité.

Hint: On rappelle également que \wedge signifie **et**, que \vee signifie **ou** et que \neg signifie **non**.

$$(P \wedge Q) \vee R$$

$$(P \vee Q) \wedge R$$

$$(P \vee Q) \wedge \neg P$$

2. Parmi les propositions suivantes, lesquelles sont satisfaisables ? Donnez une solution si c'est le cas ou justifiez (à l'aide de mots ou d'une table de vérité) dans le cas contraire.

Hint: On rappelle qu'une proposition est satisfaisable si on peut trouver une configuration des variables (ie : une ligne dans la table de vérité) telle que la proposition est vraie. Il peut y en avoir plusieurs.

$$(P \wedge \neg P) \wedge Q$$

$$(P \wedge \neg P) \vee Q$$

$$(P \wedge \neg Q) \vee Q$$

$$(P \wedge \neg Q) \vee (\neg P \wedge Q)$$

Hint: Pour les proposition suivante, n'essayez pas d'écrire la table de vérité, il vous faudrait écrire jusqu'à $2^{26} = 67108864$ lignes !

$$A \wedge \neg B \wedge C \wedge \neg D \cdots \wedge \neg X \wedge Y \wedge \neg Z$$

$$A \wedge \neg(B \vee C \cdots \vee Z)$$

$$A \wedge \neg(A \vee B \vee C \cdots \vee Z)$$

3. Simplifiez les conditions suivantes :

Hint: Choisissez astucieusement vos variables booléennes (P, Q, etc.) pour traduire en langage logique

$$(x > 0 \text{ or } (x \leq 0 \text{ and } y > 100))$$

$$\text{not } (x \neq 0 \text{ or } y \neq 0)$$

$$\text{not } (x \neq 0 \text{ or } y \leq 0)$$

Algorithmie : On se propose d'étudier le code suivant qui calcule le n ième nombre de la suite de Fibonacci:

```
Entrée : n
Début :
  a <- 0
  b <- 1
  tant que n > 0
    t <- a+b
    a <- b
    b <- t
  Retour : b
```

4. Écrivez pas à pas l'exécution de cette fonction pour $n = 8$

step	1	2	3	...
n				
a				
b				

5. Que se passe-t-il si $n = 0$?

6. Montrez que cette fonction est un algorithme

7. Écrivez cette fonction en PHP. Testez là avec des valeurs de plus en plus grande de n . Comptez le nombre d'itération effectuées. Tracez le graphe de sa complexité.

Hint: N'oubliez pas que le graphe de la complexité se trace en fonction de la *taille* de n et non pas de sa valeur !