# INFO-F-302 Informatique Fondamentale
# Projet: Logique Propositionnelle et Utilisation de l'Outil MiniSat

**Attention :** lire tout le sujet avant de commencer!
L'objectif de ce projet est de modéliser le problème *Light-Up*[1][2] en logique propositionelle, et de le résoudre avec l'outil MiniSat.

## 1 L'Outil MiniSat

L'outil MiniSat est un programme qui décide le problème SAT. Si la formule est satisfaisable, une interprétation qui la satisfait est retournée.

`http://minisat.se/`

MiniSat prend en entrée une formule en FNC de la logique propositionnelle. Vous avez déjà un exemple du code nécessaire pour utiliser MiniSat, disponible sur l'ancienne page web du cours:

`http://di.ulb.ac.be/info-f-302/`

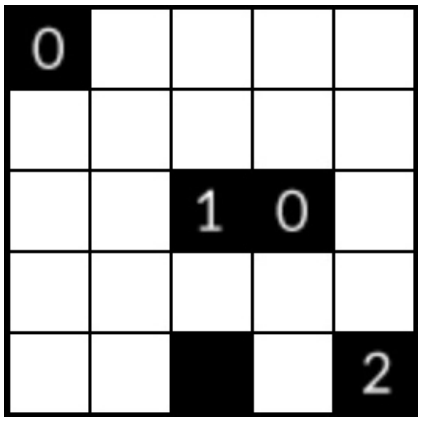
## 2 Le problème Light-Up

Etant donnée une grille $m \times n$ où chaque case est soit un mur soit un espace vide, le problème consiste à placer des ampoules dans des cases vides afin d'éclairer toutes les cases vides. Chaque ampoule placée éclaire toutes les cases qui partagent sa ligne et sa colonne pourvu qu'un mur n'interrompe pas le faisceau.

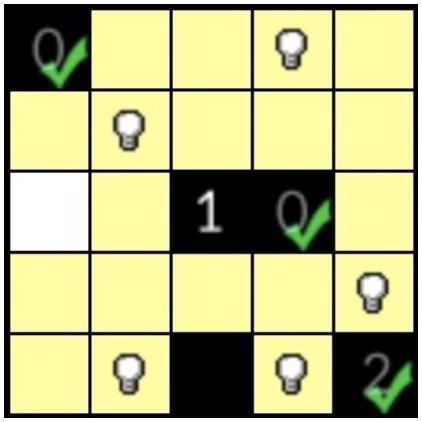
De plus, le placement d'ampoules est soumit aux conditions suivantes:

- Deux ampoules ne peuvent pas s'éclairer mutuellement.
- Chaque mur est doté d'une capacité entière qui limite le nombre d'ampoules pouvant être placées en adjacence à ce mur. Par exemple, si la capacité du mur vaut 1, une seule ampoule doit être placée à coté de ce mur (en excluant les diagonales). Si la capacité vaut 0, aucune ampoule peut être placée à coté du mur. Si la capacité vaut $-1$, aucune limite de capacité est placée sur ce mur (il peut donc y avoir de 0 à 4 ampoules adjacentes au mur).

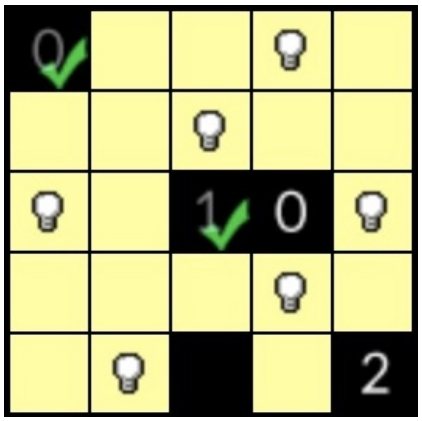
[1] `https://www.brainbashers.com/lightuphelp.asp`
[2] `https://www.brainbashers.com/lightup.asp`



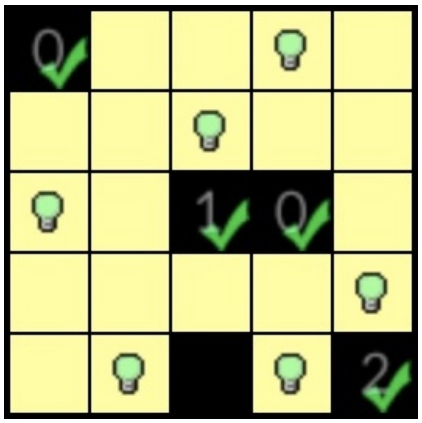

(a) Configuration vide, une case noire est un mur, le nombre présent sur un mur dénote sa capacité, i.e., le nombre d'ampoules qu'on doit placer à coté de ce mur.




(c) Solution invalide, la case en (3,1) n'est pas éclairée et les ampoules en (2,2) et (5,2) s'éclairent mutuellement.




(b) Solution invalide, deux ampoules sont adjacentes au mur en (3,4) de capacité 0.




(d) Solution valide, toutes les cases sont éclairées, toutes les capacités sont respectées.

Figure 1: Exemple de configuration et de solutions (in)valides. Notez que pour les questions de 1 à 3 vous ne devez pas gérer les murs de capacité 2.

Formellement, l'entrée du problème est la donnée de deux entiers strictement positifs $n$ et $m$, ainsi qu'une fonction (totale) $\mathcal{C} : \{1, ..., m\} \times \{1, ..., n\} \rightarrow \{-2, ..., 1\}$ qui dénote la capacité de chaque case:

- Si $\mathcal{C}(i, j) = -2$, la case est vide. I.e., il n'y a pas de mur à cet emplacement, cette case doit donc être éclairée.
- Si $\mathcal{C}(i, j) = -1$, la case contient un mur sans capacité définie. I.e., de 0 à 4 ampoules peuvent être placées dans le voisinage immédiat[3] de ce mur.
- Si $\mathcal{C}(i, j) = c_{i,j}, 0 \leq c_{i,j} \leq 1$, la case contient un mur à capacité définie. I.e., exactement $c_{i,j}$ ampoules doivent être dans le voisinage immédiat de ce mur.

Il vous est donc demandé de trouver un placement d'ampoules tel que chaque case qui n'est pas un mur est éclairée et les conditions ci-dessus sont respectées. Formellement, le but est de trouver (si elle existe), une assignation $\mu : \{1, ..., m\} \times \{1, ..., n\} \rightarrow \{1, 0\}$ tel que pour toute position $(i, j)$, $\mu(i, j) = 1$ si et seulement si il y a une ampoule à cette case, et qui respecte les contraintes du problème.

# 3 Questions

**Q1.** Pour toute instance du problème (i.e., la donnée $I = n, m, C$), construire une formule $\Phi_I$ en FNC de la logique propositionelle tel que $\Phi_I$ est satisfaisable si et seulement si il existe une assignation $\mu$ qui satisfait les contraintes du problème, autrement dit, qui est une solution au problème. *La construction de cette formule doit être expliquée dans le rapport.*

**Q2.** Implémenter à l'aide de minisat et tester sur les exemples joints à cet énoncé (en respectant le format indiqué à la section 4).

**Q3.** Modifier votre programme afin de générer toutes les solutions valides pour l'instance $I$ donnée. Cette option sera activée par le flag *'-a'*.

**Question Bonus 1** Adapter votre formule et votre programme pour gérer les capacités de 0 à 4, i.e., la fonction $\mathcal{C} : \{1, ..., m\} \times \{1, ..., n\} \rightarrow \{-2, ..., 4\}$.

**Question Bonus 2** Créer un générateur d'instances qui prend en entrée $n, m$ et $l$ et génère $l$ instances différentes de largeur $n$ et de hauteur $m$ ayant une solution unique. Cette option sera activée par le flag *'-g'*, indiquant qu'on veut générer des instances, et les flags *'-m HAUTEUR'* , *'-n LARGEUR'* et *'-l NOMBRE'* indiquent respectivement la hauteur, la largeur et le nombre d'instances à générer.

[3] I.e., en décalage d'une position soit horizontalement, soit verticalement.

# 4 Implémentation

## 4.1 Format d'entrée/sortie

Pour faciliter les tests de vos outils nous fixons un format d'entrée et de sortie. L'entrée se fera par *stdin* et la sortie sera attendue sur *stdout*. Un modèle de code à compléter est joint à l'énoncé. Vous pouvez le modifier comme vous le souhaitez, cependant assurez vous que les entrées et sorties respectent les formats spécifiés ci-dessous.

**Format d'entrée** Le format d'entrée consistera d'abord de la hauteur et de la largeur de l'instance suivie de la grille de l'instance, par exemple, l'entrée ci-dessous correspond à la Figure 1.

```
5 5
0   -2  -2  -2  -2
-2  -2  -2  -2  -2
-2  -2  1   0   -2
-2  -2  -2  -2  -2
-2  -2  -1  -2  2
```

**Format de sortie** La sortie attendue est une liste de tuples représentant la position des ampoules ainsi qu'une impression de la grille où la position des ampoules est marquée par un $X$ et la position des cases vides par un $-$. Par exemple, si on suppose que le fichier *instance1* contient une instance au format précisé ci-dessus, l'éxecution devrait (par exemple) donner le résultat suivant. S'il n'y a pas de solution, la sortie sera vide.

```
~cat exemples/instance1 | ./lightup
1 4
2 3
3 1
4 5
5 2
5 4
0       -       -       X       -
-       -       X       -       -
X       -       1       0       -
-       -       -       -       X
-       X       -1      X       -1
```

Si **plusieurs sorties** sont attendues (par exemple pour la question 3), elles seront imprimées l'une à la suite de l'autre, avec une ligne vide pour les délimiter.

```
~cat exemples/instance5 | ./lightup -a
1 1
X       -
```

```
-       -1

1 2
2 1
-       X
X       -1
```

Pour la **génération d'instances**, la sortie attendue sera une série de configurations (respectant le format d'entrée présenté au paragraphe précédent) délimités par un espace. De plus, les dimensions souhaitées ainsi que le nombre d'instances à générer seront passées en paramètre. Par exemple, si on demande de générer 3 instances de $2 \times 2$, une exécution similaire à celle ci-dessous est attendue.

```
~./lightup -g 2 2 3
2 2
-2       -2
-2       0

2 2
-2       -2
0        -2

2 2
-2       0
-2       -2
```

# 5 Instances de test

### Exemples de base

Le dossier *exemples* joint à l'énoncé contient des instances ayant des capacités de -2 à 1 sur lesquelles vous pouvez tester votre programme. Notez que les instances dont le nom contient "impossible" n'ont pas de solution.

### Exemples bonus

Le dossier *exemples-bonus* joint à l'énoncé contient des instances ayant des capacités de -2 à 4 sur lesquelles vous pouvez tester votre programme pour la question bonus 1. Notez que les instances dont le nom contient "impossible" n'ont pas de solution.

# 6 Consignes supplémentaires

Ce projet peut être réalisé en groupes de deux, dans quel cas la composition du binome devra être indiquée dans le rapport. Vous devrez remettre les fichiers

suivants sur l'UV pour le 15 mai à 23h59 au plus tard (une remise par groupe).

1. Un fichier *rapport.pdf* qui répond aux questions posées dans cet énoncé et qui documente votre implémentation.

2. Un fichier *Main.cpp* qui contiendra votre implémentation. Veillez à la propreté du code (commentaires, bonnes pratiques,...)! Pour chaque clause que vous ajoutez dans votre code, commentez à quelle clause du rapport elle correspond.

3. Un *Makefile* permettant de compiler votre programme afin de générer un fichier exécutable appelé ***lightup*** qui pourra être appelé en suivant le format des arguments présentés dans la section précédente. Vous pouvez supposer que les fichiers source de Minisat seront présents dans le même dossier. Pour toute autre librairie non standard vous devez au préalable demander l'accord.

Pour toute question, adressez vous à Axel Abels (aabels@ulb.ac.be).