Projet 2 L3IF — DM

à faire **seul(e)**, à rendre pour le 12/2/2013 à 23h59

Vous pouvez vous adresser aux encadrants du cours si vous avez des doutes sur tel ou tel aspect de votre implémentation (pas pour corriger vos bugs).

1 En entrée

Votre programme devra respecter le format suivant en entrée, pour la description d'un probleme SAT.

- Ligne de départ: `p cnf V C`, où `V` est le nombre de variables et `C` le nombre de clauses.
- Lignes représentant une clause: une suite d'entiers $\neq 0$ terminée par un 0 (x_3 est représenté par 3, $\overline{x_3}$ par -3).

  Exemple: `1 -9 -2 7 0` *(qui représente $x_1 \vee \overline{x_9} \vee \overline{x_2} \vee x_7$)*
- Lignes "`c xxx`" : commentaire

Si vous codez en Caml, vous pourrez partir des fichiers du TP 13 du cours "Théorie de la Programmation" du premier semestre[1].

Vous pouvez également partir de choses similaires disponibles en ligne pour C, C++[2] ou Java[3].

2 À l'intérieur

Il vous est demandé d'implémenter l'algorithme de Davis Putnam vu en cours pour résoudre une instance de SAT. Une fois saisie la formule sous forme de suite de clauses, il vous faudra comprendre comment manipuler litéraux et clauses dans votre programme, et comment remplir et manipuler les "seaux" de l'algorithme Davis Putnam.

Une question. Expliquez comment, dans le cas où la formule initiale est satisfiable, on trouve une affectation des variables témoignant de cela.

3 En sortie

- Lignes "`c xxx`" : commentaire;
- Ligne "`s SATISFIABLE`" : problème satisfiable;
- Ligne "`s UNSATISFIABLE`" : le contraire;
- Ligne "`s xxx`" : solvabilité inconnue;
- Lignes "`v N`" : assignation de la variable $|x|$ (vrai si x est positif, faux sinon) qui rend le problème satisfiable (ainsi, `v -3` signifie que x_3 est à faux, et `v 12` signifie que x_{12} est à vrai).

Votre programme devra renvoyer une affectation de variables dans le cas où la réponse est `SATISFIABLE`.

[1] `http://perso.ens-lyon.fr/jeanmarie.madiot/prog/tp13.tgz`
[2] Voir page www du cours, fichiers `flexbison.tgz` et `flexbison++.tgz`.
[3] Nous faire signe.

4 Fichiers de test

Vous trouverez à l'adresse

`http://perso.ens-lyon.fr/daniel.hirschkoff/P2/tests-dm`

des fichiers de test de base, que vous devez récupérer sur votre machine, et soumettre à votre solveur.

5 Mettre à l'épreuve votre programme

Étape 1: si vous pensez que votre programme répond en temps linéaire par rapport à la taille du problème d'entrée, relisez votre cours d'Algo 1. Lorsque vous sortez de la boucle correspondant à cette étape 1, allez à l'étape 2.
Étape 2: Écrivez un petit programme qui engendre des fichiers de tests de complexité croissante, afin de pousser votre programme dans ses retranchements. Indiquez à partir de quel moment le programme prend plus de 5 minutes pour répondre (typiquement, les tests sont engendrés à partir d'un paramètre, vous indiquerez pour quelle valeur du paramètre la barrière est franchie).

La solution la plus immédiate pour mesurer le temps d'exécution est d'utiliser la commande `time` dans une console (faire `man time` dans la console pour vous renseigner sur `time` — un exemple idiot est "`time sleep 1; echo "alligator"`").

6 Exigences pour le rendu

Vous enverrez votre DM par mail avec en attachement une archive compressée (si possible avec un nom de fichier significatif), à

`benjamin.girault@ens-lyon.fr` et `daniel.hirschkoff@ens-lyon.fr`

L'archive devra contenir:

- Un fichier README (ou Lisez-moi si vous préférez), dans lequel vous indiquerez succintement des informations essentielles (comment compiler votre programme, comment l'exécuter), des remarques sur des choix d'implémentation (litéraux, clauses, et éventuellement autres structures de données), et vous donnerez votre réponse à la question de la partie 2.
- Votre rendu ne doit pas engendrer d'erreur à la compilation!
- Il y aura également un répertoire de tests comportant les fichiers indiqués plus haut (plus éventuellement d'autres tests que vous avez ajoutés).