Projet 2 L3IF, DM — l'algorithme de Davis&Putnam

à faire **seul(e)**, à rendre pour le **8 février** 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).
Attention: les encadrants ne sont pas nécessairement disponibles durant le week-end.

1 Format

En entrée. Votre programme devra respecter **scrupuleusement** le format suivant en entrée, pour la description d'un problème SAT. Le format est organisé par lignes.

- Ligne d'*en-tête*: `p cnf V C`, où `V` est l'indice maximum utilisé pour les 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` *représente* $x_1 \vee \overline{x_9} \vee \overline{x_2} \vee x_7$
- Lignes "`c xxx`" : commentaire.

Une ligne de commentaire peut apparaître n'importe où, y compris avant la ligne d'en-tête. Elle peut être vide. Les clauses apparaissent toutes après l'en-tête.

Si le `C` ou le `V` qui est annoncé dans l'en-tête du fichier ne correspond pas à ce que contient effectivement le fichier, affichez un message d'avertissement du style *"Le fichier comporte n clauses, alors que C clauses étaient annoncées."*, mais continuez quand même (à l'image de ce que fait `minisat`).

L'exécutable correspondant à votre programme devra être nommé `resol`, et il faudra qu'il puisse être appelé de la manière suivante pour lancer le programme sur le fichier `ex2.cnf`:
`./resol ex2.cnf` (pas de `resol < ex2.cnf`, ou de `./resol -f ex2.cnf`, ou autres).

Un fichier d'entrée du solveur aura pour suffixe `.cnf`

Il vous est vivement *conseillé de vous appuyer sur des outils comme lex/yacc (flex/bison, ocamllex/ocamlyacc) pour la partie "saisie"*[1]*. Certes, pour ce qui est nécessaire dans le cas présent, c'est un peu écraser une fourmi avec un marteau piqueur, et on peut s'en sortir avec de simples automates programmés à la main. Mais il faudra faire du lex/yacc plus tard dans le semestre, donc, si vous n'êtes pas trop à l'aise avec, autant se faire la main tout de suite.*

En sortie.

- Ligne "`s SATISFIABLE`" : problème satisfiable;
- Ligne "`s UNSATISFIABLE`" : le contraire;
- Ligne "`s ???`" : solvabilité inconnue;
- Votre programme devra renvoyer une affectation des variables dans le cas où la réponse est `SATISFIABLE`.

  Pour cela, après la ligne où est affiché `SATISFIABLE`, afficher la suite des valeurs associées aux variables, terminées par un zéro (exemple: `-1 2 3 4 -5 0`)[2].

[1] Voir la page www du cours pour un lien vers des exemples en C/C++/Caml, contactez-nous si vous codez en Java et souhaitez récupérer un exemple.

[2] Vous pourrez constater que `minisat` considère que si 5 est l'index maximal des variables, toutes les variables de 1 à 5 existent, même s'il y en a qui ne sont pas mentionnées dans les clauses.

2 À l'intérieur

Il vous est demandé d'implémenter l'algorithme de Davis Putnam vu en cours pour résoudre une instance de SAT. Il vous faudra pour cela comprendre comment manipuler litéraux et clauses dans votre programme, et comment remplir et manipuler les *seaux* de l'algorithme Davis Putnam.

Cet algorithme n'est pas très efficace, il vous est néanmoins demandé de l'implémenter "tel quel". Vous pouvez réfléchir si vous le souhaitez à des améliorations, que vous pourrez offrir en tant qu'options du programme — mais la version "monument historique" devra être fournie.

Première question. Expliquez comment, dans le cas où la formule initiale est satisfiable, on trouve une affectation des variables témoignant de cela. Vous pouvez rédiger vos explications dans le fichier README (voir plus bas).

Passage obligé : modularité. Le code de votre implémentation devra être réparti en au moins deux fichiers, outre la partie "saisie" décrite ci-dessus: un fichier (au moins) contiendra l'algorithme et ses structures de données, l'autre fichier jouera le rôle de fichier principal: appel à la fonction réalisant la saisie des données, appel à l'algorithme, affichage de la sortie.

3 Fichiers de test et corrigé

Vous trouverez à l'adresse

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

des fichiers de test, que vous devez récupérer sur votre machine, et soumettre à votre solveur. Il vous est conseillé de fabriquer vous-même vos fichiers de tests, plus petits, pour debugger votre solveur.

Sur les machines élèves, vous pouvez lancer le programme `minisat`, qui implémente un solveur SAT, avec le même format de fichiers en entrée (pas en sortie). Tout au long du semestre, ce programme pourra vous servir de référence pour tester la correction de vos réponses.

L'utilisation en est simple: `minisat foo.cnf` ou alors `minisat foo.cnf resultats.txt` pour récupérer une affectation des variables.

4 Mettre à l'épreuve votre programme

Une boucle pour commencer: 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, jusqu'à arrêter de penser cela.

Seconde question. Décrivez une suite de tests de complexité croissante, afin de pousser votre programme dans ses retranchements.

Écrivez un petit programme qui engendre ces tests, et indiquez à partir de quel moment le programme prend plus de 2 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"`).

5 Exigences pour le rendu

Vous enverrez votre DM par mail avec en attachement une archive compressée (portant un nom de fichier plus significatif que `dm2`), à
`julien.bensmail@ens-lyon.fr` et `daniel.hirschkoff@ens-lyon.fr` et `vincent.lanore@ens-lyon.fr`

- Votre rendu ne doit pas engendrer d'erreur à la compilation!

  La **référence** pour la compilation et l'exécution de vos programmes est la configuration offerte sur les salles machines de l'ENS (testez que tout tourne sur les machines à l'ENS avant de nous envoyer votre rendu).

  Il ne faut pas, en particulier, que votre programme s'appuie sur des outils/librairies/etc. qui ne sont pas installés sur les machines des salles libre-service.

- 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 aux questions posées aux parties 2 et 4.

- Il y aura également un répertoire de tests comportant les fichiers de tests que vous avez utilisés pour répondre à la partie 4.