Departamento de
Informática e Ingeniería
de Sistemas

Universidad Zaragoza

Prácticas de Algoritmia para problemas difíciles
Especialidad en Computación, grado en Ingeniería Informática

Curso 2013-2014

Universidad de Zaragoza
Escuela de Ingeniería y Arquitectura
Departamento de Informática e Ingeniería de Sistemas
Area de Lenguajes y Sistemas Informáticos

15 de octubre de 2013

Organización general de las prácticas.

Se formarán equipos de dos personas. Si una de las dos abandona la asignatura, la otra deberá terminar en solitario.

Se dispone hasta el 31 de octubre para indicar al profesor (mediante un mensaje de correo electrónico) los integrantes de cada equipo (apellidos, nombre, NIP, para ambos miembros del equipo). Transcurrido ese plazo, el profesor formará los equipos restantes con las personas sin asignación conocida y se comunicarán tales equipos.

Las prácticas se realizarán en el computador `hendrix`.

El lenguaje para la implementación puede ser Java o cualquier otro elegido por el alumno (incluyendo en la documentación todos los detalles necesarios sobre el mismo: versión, compilador, etc).

La entrega de la práctica X (X = 1, 2, ...) se realizará en `hendrix` mediante la ejecución de `someter apd_13 practicaX.tar`, siendo la fecha límite una semana antes de la fecha del examen correspondiente a la convocatoria que se presenten los miembros del grupo. Además, habrá que concertar una cita con uno de los profesores para explicar las prácticas: descripción general del programa, elaboración, funcionamiento, demostración, etc.

El fichero `practicaX.tar` contendrá **un directorio denominado practicaX** con los ficheros de texto incluyendo:

- Descripción general del programa: cómo está organizado, qué se puede y qué no se puede hacer (tiene que llamarse **LEEME**). Contendrá en sus primeras líneas la lista de integrantes del grupo, con el siguiente formato:

  ```
  Apellido1 Apellido2, Nombre [tab] correo@electronico [tab] login en
  hendrix

  Apellido1 Apellido2, Nombre [tab] correo@electronico [tab] login en
  hendrix
  ```

  Donde [tab] representa el carácter tabulador.

- Listados del código debidamente comentados y dispuestos para ser compilados y utilizados.
- Un programa para el shell `ejecutarX.sh` que automatice la compilación y ejecución de algunos casos de prueba para los programas entregados. Deberá funcionar en hendrix.
- Los ficheros auxiliares de entrada necesarios para ejecutar las pruebas del punto anterior.

En la calificación se tendrán en cuenta los siguientes aspectos: documentación, funcionamiento e implementación.

El diseño ha de ser modular, basado en el uso de tipos abstractos de datos, con todas las funciones correctamente especificadas.

Las reglas generales de tratamiento de casos de plagio de la asignatura se aplicarán, en particular, a todas las prácticas.

Práctica 1

Vamos a desarrollar un "SAT solver" o resolvedor de SAT que dada una fórmula booleana en CNF diga si es o no satisfacible.

Notación: Un *literal* es una variable booleana afirmada o negada (p. ej. x, $\neg y$, etc). Una *cláusula* es una disyunción de literales (p. ej. $(x \vee \neg y)$). Una *fórmula en CNF* es una conjunción de cláusulas (p. ej. $(x \vee \neg y \vee z) \wedge (\neg z \vee y) \wedge (\neg x)$). Una fórmula es *satisfacible* si existe una asignación de las variables que hace cierta la fórmula (p. ej. $(x \vee \neg y \vee z) \wedge (\neg z \vee y) \wedge (\neg x)$ es satisfacible haciendo falsas las tres variables x, y, z).

Se trata del problema SAT visto en clase, y es importante recordar que se trata de un problema NP-completo y que la programación de "SAT solvers" es un reto en el que trabajan miles de informáticos de primer nivel. Por tanto hay que tener en cuenta lo siguiente

- Cualquier fragmento de código o idea para el que se utilicen fuentes externas debe ser identificado y dichas fuentes citadas.
- En la práctica hay que hacer como mínimo lo que se especifica a continuación, pero la tarea completa no está limitada, por lo tanto los más ambiciosos pueden necesitar fijarse un límite personal.

1.1. ¿Qué hay que hacer?

Hacer un programa que resuelva SAT en todos los casos, separando al menos los siguientes:

1. Las fórmulas en las que todas las cláusulas tengan 1 ó 2 literales (es decir, 2-SAT permitiendo cláusulas de un solo literal).
2. Las fórmulas en las que todas las cláusulas son de Horn (Horn-SAT), es decir, cada cláusula tiene como máximo un literal afirmado (y el resto negados). Por ejemplo cláusulas como $(\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4 \vee y)$.

En los casos 1. y 2. anteriores el programa debe funcionar eficientemente (en tiempo polinómico). Los métodos a utilizar son:

1. Cualquiera de los algoritmos conocidos para 2-SAT, por ejemplo el de Krom que se basa en a) simplificar las cláusulas de un solo literal y asignar las variables que sólo aparecen

afirmadas (o sólo negadas) en la fórmula, y b) combinar cláusulas de la forma $(a \vee b)$ y $(\neg b \vee \neg c)$ en una sola $(a \vee \neg c)$.

2. El algoritmo de "Unit propagation" para Horn-SAT. Dicho algoritmo consiste en simplificar empezando por una cláusula de un solo literal l: se asigna la variable para hacer l cierto y se simplifican el resto de las cláusulas que contienen l ó $\neg l$ (si una cláusula contiene l se elimina la cláusula, si contiene $\neg l$ se borra $\neg l$ de dicha cláusula). Si no existen cláusulas de un solo literal, entonces la fórmula de Horn es satisfacible haciendo falsas todas las variables.

En el **resto de los casos**, el programa debe realizar al menos un backtracking inteligente (es decir, con alguna poda o heurística mínima) siendo recomendable (no obligatorio) implementar al menos uno de los dos siguientes algoritmos:

- DPLL (Davis–Putnam–Logemann–Loveland) consistente en backtracking,
- WalkSAT (heurística consistente en asignar las variables en orden aleatorio, si no funciona se cambia el valor de una variable (elegida aleatoriamente o vorazmente para minimizar el número de cláusulas falsas)).

Pueden utilizarse otros algoritmos que mejoren DPLL y WalkSAT o cualquier otra idea propia o ajena.

El programa se llamará `SATsolver` y tendrá al menos las siguientes opciones:

1. Leer la fórmula en CNF de un fichero de texto que utilice '+' como disyunción, '$*$' como conjunción, '$-$' como negación y nombres de variables que contengan como mucho letras, números y _, comenzando siempre por letra. La fórmula puede ocupar varias líneas.
   Ejemplo de entrada:
   $$(y + z + -t2) * (-y + -z)$$
   $$*(y + r + a\_5)$$
2. Introducir la fórmula por medio de un menú explicativo.
3. Decir si la fórmula es o no satisfacible.
4. Decir si se trata de una fórmula 2-SAT o de Horn.

1.2. Entrega

Deberá entregarse una semana antes de la fecha del examen.