Lab08: Random-3SAT y SAT-solver

1. **Formato de las fórmulas**

   Recuerda la representación de las fórmulas booleanas en CNF.

   - `num_variables` contiene el número de variables que pueden aparecer en la fórmula. Las variables siempre están numeradas de 1 a `num_variables`.
   - La fórmula Booleana está representada como una lista de listas (cláusulas). Cada lista interna correponde con una única cláusula de manera que cada literal x_i de la cláusula se representa con un i y cada literal $\neg x_i$ se representa con un $-i$.
   - Para facilitar el proceso, las asignaciones `A` serán listas con `len(A) = num_variables + 1`. Una asignación posible para las variables de φ podría ser `A=[-,1,0,1,0]`, de forma que `A`(x_1)`=1, A`(x_2)`=0, A`(x_3)`=1, A`(x_4)`=0`.

2. **Implementación de un 3SAT-solver aleatorio**

   Dispones del fichero `lab08_random_3SAT.py`, donde tienes que implementar el algoritmo aleatorio que decide si una 3CNF es satisfactible o no. La cabecera de tu función es `random_3SAT(num_variables, clauses)`. Esta función debe elegir una asignación aleatoria para las variables de la 3CNF y lo puede hacer de la siguiente forma:

   ```
   assignment = [None] + [choice([0,1]) for n in range(num_variables)]
   ```

   La función `choice` se ha importado del paquete `random` y elige al azar un elemento de la lista que se le pasa como parámetro.

   Si tu función decide que la fórmula es satisfactible debe delvolver la asignación que la hace satisfactible, si no debe devolver `"UNSATISFIABLE"`.

   Fíjate que en `lab08_random_3SAT.py` se llama muchas veces a la función `random_3SAT` para minimizar el número de respuestas incorrectas que se pudieran dar.

   Por último hay tres tests para descomentar. Los dos primeros tests contienen asserts. El segundo usa un juego de pruebas que se encuentra en la carpeta `instances`. Cada 3CNF está en un fichero de texto en formato DIMACS (éstandar para la mayoría de los SAT-solvers). Hemos creado la función

   `list_minisat2list_our_sat()`

   que se encuentra en el fichero `tools.py`, para leer de estos ficheros y traducir del formato DIMACS al nuestro. El tercer test contiene una única 3CNF muy especial que se usará en el siguiente ejercicio.

3. **Implementación de un SAT-solver correcto**

   Implementa la función **solve_SAT_rec**(num_variables, φ, asig) en el fichero `Lab08_SAT_solver.py`, que se aplica a la fórmula φ, que es una CNF, y a una asignación a medio construir. Esta función, tras realizar un pre-processing, va eligiendo valores para los literales sin asignar, hasta que consigue encontrar una asignación que hace cierta a φ. En ese caso, devuelve la asignación. Si no consigue encontrar una asignación, devuelve `"UNSATISFIABLE"`.

   IMPORTANTE: usa el pre-procesing que realizaste en el laboratorio 7, pero elimina de él la función `remove_trivial_clauses` para no repetirla cada vez que llames al pre-procesing.

   Para probar dispones de cuatro tests. Los dos primeros contienen asserts. El segundo test usa un juego de pruebas que se encuentra en la carpeta `instances`. El tercero llama al generador de fórmulas aleatorias para probar con CNFs cualesquiera. Las llamadas se hacen con muchas cláusulas y muchas variables. El último test se realiza con una CNF muy especial.

4. **Comprobación del tiempo de ejecución de tu SAT-solver**

   Los dos últimos tests llevan un cálculo del tiempo que tarda tu SAT_solver y del tiempo que tarda Pysat. Verás que tu SAT-solver es más lento que Pysat, sobre todo en el útimo test. La razón es que Pysat está mucho más optimizado que tu SAT-solver.

   Para conseguir puntos prácticos es condición necesaria que tu SAT-solver sea correcto y pase todos los tests. Además se tendrá en cuenta su tiempo de ejecución.

   Ten en cuenta que en último test puedes tardar hasta 10 minutos mientras que Pysat lo resuelve instantaneamente.