Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI5651 – Diseño de Algoritmos I
Intensivo Junio-Julio 2020

# Proyecto
# Resolvedor de Sudoku

La vida en Titán puede ser dura e incluso tediosa, pero nadie puede negar lo majestuoso de sus amaneceres. Cada mañana, Saturno se eleva colosal en el cielo, cubriendo gran parte del firmamento, pintando el cielo de colores y adornando el horizonte con sus enormes aros. ¿En verdad hubo una época en la que vivíamos sin esto? Sé que vinimos de un planeta lejano, uno de esos planetas rocosos del sistema solar interno, pero hace tanto tiempo de eso que ya nadie lo recuerda del todo. Lo único que conservamos de aquella época es nuestra reliquia más sagrada, grabada en piedra por los mismísimos fundadores de nuestra sociedad: **un tablero de *Sudoku***.

Por generaciones, científicos han intentado descifrar sus secretos. Los grandes pensadores de Titan piensan que al resolver el antiguo rompecabezas podremos entender mejor nuestra naturaleza, de dónde vinimos y a dónde vamos. Yo, por supuesto, creo que no es más que una fantasía. ¡No tiene solución! Y si la tuviera, dudo que podamos encontrarla algún día.

Después de todo, no es SAT.

Es de notar que de todo el conocimiento de nuestro pasado, ese antiguo problema parece ser lo único que ha perdurado. SAT, el problema de satisfacibilidad booleana, es la especialidad de los científicos en Titán y el pasatiempos favorito de todos. Quizá sea por nuestra proximidad a SATurno o por vivir en un SATélite natural, pero lo cierto es que es lo único que sabemos resolver. ¡Qué fácil sería la vida si todo fuera SAT!

Hace unos días leí una noticia sumamente intrigante: un grupo de entusiastas dice poder transformar el sagrado tablero a una instancia de SAT. ¡Eso habrá que verlo! Sería la solución que hemos esperado durante generaciones. Sé que la noticia mencionó el nombre del grupo y era un nombre largo y complicado. Se hacen llamar *"estudiantes de diseño de algoritmos I"*. La verdad es que no los había escuchado nombrar antes. ¿Qué significará ese nombre tan extraño? ¿Qué es un algoritmo? En fin, claman que pueden transformar nuestra antigua reliquia, resolverla y reportar la solución.

El gobierno central de Titán les ha dado acceso a la reliquia durante dos semanas, para que puedan realizar sus experimentos. ¿Ustedes qué creen? ¿Podrán lograrlo?

# 1 Sudoku

Dado un número entero positivo $N$, un tablero de Sudoku de orden $N$ es una matrix de $N^2$ filas y $N^2$ columnas. La matriz está dividida en $N^2$ secciones disjuntas, cada una una matriz de tamaño $N \times N$. Cada celda puede contener a lo sumo un número entre 1 y $N$.

El siguiente es un ejemplo para un tablero de Sudoku de orden 3.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | **4** | | | | | **9** |
| | **2** | | **9** | **8** | | | **3** | |
| | **9** | **3** | | | | **8** | **4** | |
| | | | | | | | | |
| **2** | | | | | **8** | | **6** | **7** |
| **1** | **6** | | **3** | | | | | **2** |
| | **4** | **9** | | | | **6** | **5** | |
| **6** | | | | | **3** | | | |
| | | | | **5** | **9** | | **1** | |

Un tablero de Sudoku de orden $N$ es una *solución válida* cuando:

- Cada celda de la matriz debe contener un número entero entre 1 y $N$ (ambos inclusive)
- Cada fila debe tener todos los números entre 1 y $N$ (ambos inclusive)
- Cada columna debe tener todos los números entre 1 y $N$ (ambos inclusive)
- Cada sección (submatrix $N \times N$) debe tener todos los números entre 1 y $N$ (ambos inclusive)

El siguiente tablero es una solución válida para el tablero de Sudoku anterior.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *8* | *1* | *6* | **4** | *3* | *5* | *2* | *7* | **9** |
| *4* | **2** | *7* | **9** | **8** | *6* | *1* | **3** | *5* |
| *5* | **9** | **3** | *2* | *1* | 7 | **8** | **4** | *6* |
| *9* | *7* | *4* | *5* | *6* | *2* | *3* | *8* | *1* |
| **2** | *3* | *5* | *1* | *9* | **8** | *4* | **6** | **7** |
| **1** | **6** | *8* | **3** | *7* | *4* | *5* | *9* | **2** |
| *7* | **4** | **9** | *8* | *2* | *1* | **6** | **5** | *3* |
| **6** | *5* | *1* | *7* | *4* | **3** | *9* | *2* | *8* |
| *3* | *8* | *2* | *6* | **5** | **9** | *7* | **1** | *4* |

Diremos que un tablero de Sudoku es *factible* si es posible llenar todas las casillas, sin modificar aquellas que ya estén llenas en su estado inicial, de tal forma que se llegue a una solución válida.

## 1.1 Representación de entrada y salida

Para la ejecución de este proyecto nos concentraremos en tableros de sudoku de orden máximo 3.

Cada instancia será representada por una linea de texto con el siguiente formato

- Un entero $N$, tal que $1 \leq N \leq 3$
- Un espacio en blanco
- Una cadena $T$ de $N^2$ números, donde $0 \leq T_i \leq N$
  - Si $T_i = 0$ la casilla está vacía
  - Si $T_i \neq 0$ la casilla tiene el dígito $T_i$

Por ejemplo, la siguiente cadena representa a nuestro ejemplo de tablero de Sudoku de orden 3.

```
3 000400009020980030093000840000000000200080671603000020490006506000030000000059010
```

La entrada de su programa será un archivo con tantas líneas como instancias se quieran resolver. Dada esta entrada, su salida debe ser otro archivo con una línea por cada línea de la entrada, conteniendo el tablero de sudoku resuelto (una solución válida) para la instancia descrita en dicha línea.

**Extensión sugerida** Nótese que podemos permitir tableros de orden hasta 6 (con $6^2 = 36$ símbolos) si adoptamos la siguiente convención:

- Las posiciones vacías son representadas con $d = 0$.
- Para $1 \leq d \leq 9$, se representa $d$ con el mismo dígito $d$.
  Esto es: $1 \to 1, 2 \to 2, \cdots, 9 \to 9$
- Para $10 \leq d \leq 35$, se representa $d$ con la $(d - 10)$–ésima letra del alfabeto.
  Esto es: $10 \to A, 11 \to B, \cdots, 35 \to Z$
- Para $d = 36$ se representa $d$ con el símbolo punto (“.”).
  Esto es: $36 \to .$

Esta extensión es opcional, ya que sólo se realizarán experimentos con tableros de hasta orden 3.

# 2 SAT

Dada una fórmula en lógica proposicional $\phi$, el *Problema de Satisfacibilidad Booleana* o SAT consta de intentar hallar una asignación de verdad para las variables en $\phi$, que hagan que dicha fórmula sea satisfecha (evalúe a *true*).

Por ejemplo:

- $\phi = p \wedge (\neg q \vee \neg p) \wedge (q \vee p)$
  Es satisfecha cuando $p \equiv true$ y $q \equiv false$.
- $\phi = p \wedge (\neg q \vee \neg p) \wedge (q \vee \neg p)$
  No existe una asignación de valores a $p$ y $q$ que satisfaga $\phi$.

## 2.1 Representación de entrada

Una entrada para SAT será un archivo que contiene primeramente un preámbulo y luego la información de cláusulas para la instancia.

### 2.1.1 Preámbulo

Establece la *metadata* necesaria para una instancia del problema.

- Si la línea empieza con el caracter `c`, el resto de la línea se considera un comentario.
- Si la línea empieza con el caracter `p`, el resto de la línea establece el *encabezado* de la instancia y tiene la siguiente forma: `p` ⟨*formato*⟩ ⟨*variables*⟩ ⟨*cláusulas*⟩
  - ⟨*formato*⟩ es el formato de la fórmula, que, en nuestro caso, debe ser siempre "`cnf`" (sin las comillas).
  - ⟨*variables*⟩ es la cantidad de variables que tiene la fórmula.
  - ⟨*cláusulas*⟩ es la cantidad de cláusulas que tiene la fórmula.

### 2.1.2 Cláusulas

Una vez establecida la cantidad de variables $V$ y cláusulas $C$, esta sección contendrá la información de dichas cláusulas.

Se supondrá que las variables están representadas por números del 1 al $V$. La negación de cada variable estará representada por el mismo valor, pero en negativo.

Cada cláusula estará separada de la siguiente por el valor 0 (cero). No es necesario que cada cláusula esté en una única linea. El único discriminador para separar cláusulas es el valor 0.

**Ejemplo**

Consideremos $\phi = (x_1 \vee x_3 \vee \neg x_4) \wedge x_4 \wedge (x_2 \vee \neg x_3)$

Dicha fórmula puede ser representada en un archivo como el siguiente, para ser pasado a un algoritmo que sepa resolver SAT.

```
c Ejemplo de fórmula en CNF
c
p cnf 4 3
1 3 -4 0
4 0 2
-3
```

Nótese que dicha representación no es única. Existen infinitos archivos diferentes que representan a la misma fórmula $\phi$.

## 2.2 Representación de salida

Una salida para SAT será un archivo que contiene primeramente un preámbulo y luego la información de satisfacibilidad.

### 2.2.1 Preámbulo

Establece la *metadata* necesaria para entender una solución.

- Si la línea empieza con el caracter `c`, el resto de la línea se considera un comentario.
- Si la línea empieza con el caracter `s`, el resto de la línea establece el *encabezado* de la solución y tiene la siguiente forma: `s` ⟨*formato*⟩ ⟨*solución*⟩ ⟨*variables*⟩
  - ⟨*formato*⟩ es el formato de la fórmula, que, en nuestro caso, debe ser siempre "`cnf`" (sin las comillas).
  - ⟨*solución*⟩ puede tener uno de tres valores.
    - `1`: Si la fórmula es satisfacible y se halló una solución
    - `0`: Si la fórmula es insatisfacible
    - `-1`: Si no se llegó a una decisión en el tiempo establecido
  - ⟨*variables*⟩ es la cantidad de variables que tiene la fórmula.

### 2.2.2 Solución

Una vez establecida la cantidad de variables $V$ y que la fórmula es satisfacible, esta sección contendrá la información sobre la solución.

Seguirán $V$ lineas, cada una de ellas con la forma: `v` ⟨*Literal*⟩

- `v N` establece que la variable `N` es verdadera.
- `v -N` establece que la variable `N` es falsa.

**Ejemplo**

Consideremos $\phi = (x_1 \vee x_3 \vee \neg x_4) \wedge x_4 \wedge (x_2 \vee \neg x_3)$

La fórmula $\phi$ es satisfacible, al menos, cuando $x_1 \equiv true$, $x_2 \equiv true$, $x_3 \equiv false$ y $x_4 \equiv true$.

Dicha solución puede ser representada en un archivo como el siguiente.

```
c Ejemplo de solución para fórmula en CNF
c
s cnf 1 4
v 1
v 2
v -3
v 4
```

Tanto la entrada como la salida propuestas son compatibles con el formato usado en la competencia DIMACS para resolvedores de SAT.

## 2.3 Implementación propia

Se desea que diseñe un algoritmo que permita resolver cualquier instancia de SAT, siempre y cuando la fórmula proporcionada esté en CNF (*Forma Normal Conjuntiva*).

Su programa debe ser capaz de interpretar cualquier entrada en el formato propuesto para entrada (sólo para formato `cnf`), hallar una asignación de variables que satisfaga la instancia contenida en dicho archivo (o verificar que dicha fórmula no es satisfacible) y reportar el resultado en el formato propuesto para salida.

## 2.4 ZCHAFF

ZCHAFF es un resolvedor conocido para SAT, con tiempos de ejecución muy competitivos. Usa el mismo formato de entrada aquí expuesto y reporta sus soluciones con el mismo formato de salida.

Puede hallar ZCHAFF, aquí: https://www.princeton.edu/ chaff/zchaff.html

# 3 Reducción de Sudoku a SAT

Es posible reducir cualquier instancia de Sudoku a SAT (*Problema de Satisfacibilidad Booleana*), representando el estado del tablero con variables y cláusulas de una fórmula en CNF (*Forma Normal Conjuntiva*).

## 3.1 Variables

Para representar un tablero de Sudoku de orden $N$, tendremos variables con forma $x_{i,j}^d$, que serán verdaderas si la casilla $(i, j)$ del tablero tiene el número $d$. Existirán $N^3$ variables de este tipo.

**Representación** Nótese que para llevar esta representación a la establecida en la sección anterior, debemos construir una biyección $Variables \leftrightarrow [1, N^6]$

Recordando que $0 \leq i, j < N^2$ y $1 \leq d \leq N^2$, una biyección posible sería asociar cada $x_{i,j}^d$ con el entero $i \times N^4 + j \times N^2 + d$.

## 3.2 Cláusulas

Para representar a una solución válida de Sudoku, tendremos 3 tipos de cláusulas.

**Completitud** Una solución para un tablero de Sudoku no debe tener posiciones vacías, por lo que en cada posición del tablero debe haber alguno de los números disponibles

$$\bigvee_{1 \leq d \leq N^2} x_{i,j}^d$$

Existirán $N^2$ cláusulas de este tipo.

**Unicidad** Una solución para un tablero de Sudoku no debe tener dos números en una misma posición del tablero. Esto es, si $d$ y $d'$ son números tal que $1 \leq d \leq N^2 \wedge d \neq d'$

$$\neg x_{i,j}^d \vee \neg x_{i,j}^{d'}$$

Existirán $\binom{N^2}{2} \times N^4 = \frac{N^8 - N^6}{2}$ cláusulas de este tipo.

**Validez** Una solución para un tablero de Sudoku no debe tener el mismo valor dos o más veces en una misma fila, columna o sección.

Sea $S$ un conjunto de índices representando una fila, columna o sección del tablero (nótese que $|S| = N^2$), $d$ un número tal que $1 \leq d \leq N^2$ y $s, s' \in S$ con $s \neq s'$.

$$\neg x_s^d \vee \neg x_{s'}^d$$

Existirán $N^2$ filas, $N^2$ columnas y $N^2$ secciones. Para cada una de estas, habrá $\binom{N^2}{2} \times N^2$ cláusulas de este tipo. Finalmente, en total existirán $3 \times N^2 \times \binom{N^2}{2} \times N^2 = \frac{3 \times (N^8 - N^6)}{2}$ cláusulas de este tipo.

# 4 Entrega

Todos los proyectos deben estar en un repositorio privado en Github, cuyos colaboradores sean los miembro del equipo y los profesores del curso.

- Usuario de Github de Ricardo Monascal: ThisChair
- Usuario de Github de Carlos Infante: rmonascal

La entrega será el día Domingo, 18 de Junio (semana 3), a las 11:59pm VET. A la hora de la entrega, se hará `pull` del repositorio y se corregirá lo que exista ahí hasta ese punto.

## 4.1 Programas

Su entrega debe incluir los siguientes programas:

- Traductor de instancias de Sudoku a instancias de SAT
- Resolvedor propio de SAT
- Traductor de soluciones de SAT a soluciones de Sudoku

Además, debe incluir al menos dos *scripts* que permita orquestrar los programas implementados.

- Tomar una entrada de Sudoku, traducirla a una entrada de SAT, **aplicarle el resolvedor propio de SAT**, traducir la solución a una solución de Sudoku y reportar.
- Tomar una entrada de Sudoku, traducirla a una entrada de SAT, **aplicarle el resolvedor ZCHAFF de SAT**, traducir la solución a una solución de Sudoku y reportar.

Es importante que incluya en sus *scripts* alguna forma de detener el cómputo y reportar que no se pudo hallar una solución, si pasan más de $T$ unidades de tiempo. El valor de $T$ queda de su parte, pero, al realizar experimentos, debe ser igual para ambos scripts (de tal forma que estén en igualdad de condiciones).

## 4.2 Informe

Su proyecto debe incluir un manual de uso y una explicación detallada sobre la implementación: las estructuras de datos y algoritmos utilizados, así como un análisis de recursos para las componentes más importantes (en notación asintótica).

Se recomienda que esta sección del informe se ubique en un archivo `README.md` (escrito en el lenguaje *Markdown*), de tal forma que se muestre correctamente en la página inicial de su repositorio en Github.

### 4.2.1 Tiempos de ejecución

Debe ejecutar sus soluciones sobre todas las instancias incluidas en el archivo `InstanciasSudoku.txt` y reportar lo siguiente:

- Tiempo de ejecución del resolvedor de sudoku (en milisegundos), usando el resolvedor propio de SAT.
- Tiempo de ejecución del resolvedor de sudoku (en milisegundos), usando el resolvedor ZCHAFF de SAT.

Para esto es necesario crear gráficos comparativos, de tal forma que se facilite el análisis de los resultados presentados.

### 4.2.2 Instancias resueltas

Debe elaborar un informe tal que, para las instancias que hayan sido resueltas por cualquiera de los dos métodos se incluya una representación legible (en forma de matriz) del tablero de Sudoku que se pasó por entrada y la solución reportada por el algoritmo.

El análisis de tiempo de ejecución y el reporte de instancias resueltas puede ir en un mismo informe. Aunque no es un requerimento, se recomienda crear un programa que genere automáticamente estos informes a partir de la información de ejecución.