

**PMR3201 - Computação para Automação**  
**Exercício Programa 1 - 2018**

**Satisfazibilidade booleana (SAT)**

**Prof. Dr. Thiago de Castro Martins**  
**Prof. Dr. Marcos de Sales Guerra Tsuzuki**  
**Prof. Dr. Newton Maruyama**

**Deadline: 10/04/2018 - 23h59min**

## 1 Introdução

O *Problema de Satisfazibilidade Booleana* (SAT) é um problema de decisão. Trata-se da verificação da existência (ou não) de um conjunto de valores booleanos que se atribuídos às proposições que constituem os elementos de uma fórmula proposicional tornam esta fórmula verdadeira. Por exemplo, seja a seguinte fórmula:

$$f_1 = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_3). \quad (1)$$

onde  $\vee$  representa o conectivo lógico de disjunção **ou**,  $\wedge$  representa o conectivo lógico de conjunção **e** e  $\neg$  representa a negação.

Será que existe um conjunto de valores booleanos que se atribuídos à  $(x_1, x_2, x_3)$  fariam com que a fórmula  $f_1$  seja verdadeira ?

Uma das maneiras de resolver este problema seria construindo uma tabela verdade como ilustrado abaixo:

$x_1$	$x_2$	$x_3$	$(x_1 \vee x_2)$	$(\neg x_1 \vee x_3)$	$f_1$
0	0	0	0	1	0
0	0	1	0	1	0
0	1	0	1	1	1
0	1	1	1	1	1
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	0	0
1	1	1	1	1	1

Tabela 1: Tabela verdade para a fórmula  $f_1$ .

Através da construção da tabela verdade sabemos que existem quatro conjuntos de valores  $(x_1, x_2, x_3)$  que tornam a fórmula  $f$  verdadeira.

Fórmulas proposicionais podem não ser verdadeiras para nenhum conjunto de valores ? Sim, por exemplo:

$$f_2 = (x_1 \wedge \neg x_1). \quad (2)$$

Pode-se verificar facilmente que a fórmula  $f_2$  não possui valores booleanos para  $x_1$  que a tornam verdadeira.

Num problema SAT, muito embora uma possível candidata à solução pode facilmente ser verificada não existe uma formulação determinística que facilmente encontre uma solução. O Problema de Satisfazibilidade Booleana se torna de difícil solução a medida que o número de variáveis  $n$  cresce.

Em 1971 Stephen A. Cook provou que o Problema SAT é *NP*-Completo (onde *NP* significa não determinístico polinomial), ou seja, não existe um algoritmo determinístico que possa resolver o problema em tempo polinomial. Esta prova passou a ser chamada de Teorema de Cook-Levin.

Atualmente, o Problema SAT desperta grande interesse devido a sua aplicação nas áreas de Inteligência Artificial e checagem de modelos de hardware e software.

Uma das ambições intelectuais contemporânea é a possibilidade da descoberta de algum algoritmo eficiente para SAT o que seria equivalente fazer com que um problema  $NP$  possa ser reduzido a  $P$ , ou seja,  $P = NP$ . Esse problema denominado  $P$  versus  $NP$  é um dos problemas que o Clay Mathematics Institute oferece um prêmio de US\$1.000.000,00 para uma eventual solução.

## 2 O Problema $k$ -SAT

O Problema  $k$ -SAT se refere a fórmulas com cláusulas disjuntivas com  $k$  literais<sup>1</sup>.

Particularmente, estamos interessados no Problema 3-SAT. Por exemplo, uma fórmula com quatro variáveis  $(x_1, x_2, x_3, x_4)$  poderia ser a seguinte:

$$f_3 = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \wedge x_2 \wedge x_4). \quad (3)$$

A fórmula  $f_3$  tem duas cláusulas (Expressão entre parênteses). Cada cláusula possui três literais que podem ser compostos apenas pelo operador de disjunção  $\vee$ . A fórmula completa é constituída por cláusulas compostas pelo operador de conjunção  $\wedge$ .

Uma fórmula genérica seria composta por  $n$  variáveis,  $m$  cláusulas e  $k$  literais.

## 3 Representação de Fórmulas proposicionais

Para as competições do *The International SAT Competition* é utilizado um formato do tipo CNF (Conjunctive Normal Form) Você deve utilizar este formato para a representação das fórmulas proposicionais.

Um exemplo de arquivo 3-SAT é apresentado a seguir:

```
c
c start with comments
c
c
p cnf 5 3
1 -5 4 0
-1 3 4 0
2 -3 -4 0
```

O arquivo pode começar com linhas de comentários. No arquivo acima as linhas iniciais começando com o caracter 'c' são as linhas de comentários.

A linha dada por:

```
p cnf 5 3
```

indica que existem 3 cláusulas e 5 variáveis.

A linha

```
1 -5 4 0
```

se refere à cláusula  $(x_1 \vee \neg x_5 \vee x_4)$ .

A fórmula proposicional  $f$  referente ao arquivo acima é dada por:

$$f = (x_1 \vee \neg x_5 \vee x_4) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee \neg x_4) \quad (4)$$

Detalhes deste formato podem ser encontrados em

<http://www.satcompetition.org/2009/format-benchmarks2009.html>

Uma possível maneira de representar essa fórmula proposicional na linguagem Python seria através da seguinte matriz:

---

```
formula=[[1,-5,4],[-1,3,4],[2,-3,-4]]
```

---

<sup>1</sup>Os literais da variável  $x_1$  são representados por  $x_1$  e sua negação  $\neg x_1$ .

## 4 Para você fazer:

1. Construir um programa `main()` que recebe como entrada o nome do arquivo contendo a fórmula proposicional no formato CNF e procura por soluções.
2. O usuário pode escolher entre dois algoritmos implementados em funções distintas:
  - (a) `RandomSearch()`: uma solução contendo  $n$  variáveis  $(x_1, x_2, \dots, x_n)$  é gerada escolhendo para cada variável  $x_i$ ,  $i = 1, \dots, n$  um valor booleano tal que  $P(0) = P(1) = 0.5$ .
  - (b) `BruteForce()`: uma solução é encontrada realizando uma busca exaustiva em todo espaço de soluções.
3. O programa para quando a primeira solução é encontrada. Ao parar o programa deve imprimir a solução encontrada, ou seja, o valor booleano para cada uma das  $n$  variáveis  $(x_1, x_2, \dots, x_n)$  e também o número de iterações para a obtenção dessa solução.
4. Inicialmente teste o programa, utilizando as duas opções de algoritmos, com o seguinte arquivo: `testek2n4m3.cnf`. Este arquivo define uma fórmula proposicional cuja solução é fácil de se obter.
5. Faça o mesmo agora para os seguintes arquivos:
  - `uf20-xx (01,02,03,04,05)`: arquivos com  $k = 3$ ,  $n = 20$ ,  $m = 91$ .
  - `uf50-xx (01,02,03,04,05)`: arquivos com  $k = 3$ ,  $n = 50$ ,  $m = 218$

Os arquivos teste podem ser obtidos em <http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html>.

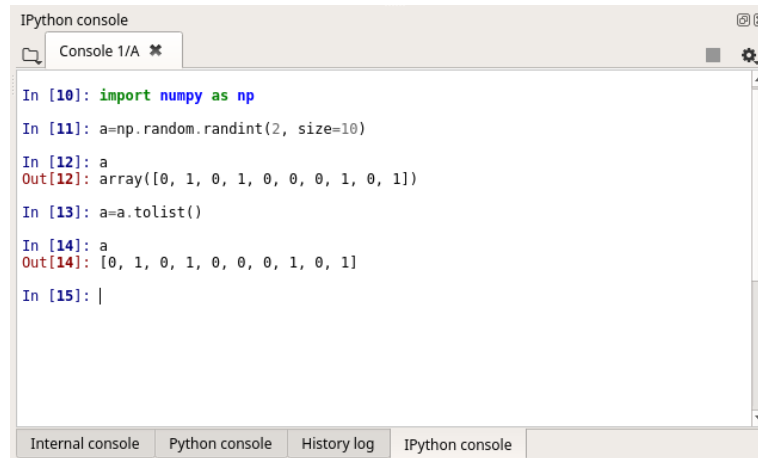
6. Escreva um relatório de no máximo uma página A4 contendo as seguintes informações:
  - (a) Uma descrição da sua solução de maneira resumida.
  - (b) Uma análise dos resultados obtidos com os arquivos teste.
  - (c) Uma breve comparação dos dois algoritmos implementados.
7. O código fonte do seu programa em conjunto com o seu relatório no formato PDF devem ser submetidos no sistema Moodle.
8. **DEADLINE: 10/04/2018 - 23h59min.**

## 5 Apêndice

### 5.1 Gerando um vetor booleano aleatório

Um vetor booleano aleatório pode ser gerado através da função `randint()` da biblioteca `numpy` como ilustrado na figurafig:boolean.

A função `tolist()` é utilizada para transformar o tipo `array` de `numpy` para o tipo `lista` convencional.



```
IPython console
Console 1/A ✖
In [10]: import numpy as np
In [11]: a=np.random.randint(2, size=10)
In [12]: a
Out[12]: array([0, 1, 0, 1, 0, 0, 0, 1, 0, 1])
In [13]: a=a.tolist()
In [14]: a
Out[14]: [0, 1, 0, 1, 0, 0, 0, 1, 0, 1]
In [15]: |
```

The screenshot shows an IPython console window with a tab labeled 'Console 1/A'. The console contains several lines of Python code and their outputs. The code generates a random boolean vector of size 10 using NumPy's `randint` function, converts it to a list, and displays it. The output of the `tolist()` method is a Python list containing the same random values as the NumPy array.

Figura 1: Geração de vetor booleano aleatório.