Lógica Computacional

September 30, 2019

1 Trabalho 0

2. Criar um "hello world" para IPython + Numpy com os seguintes passos:

`a. Criar uma matriz de inteiros {0,1}, aleatória, A de dimensão 1024 x 1024 armazenada de forma compacta; usar Numpy.`

Para gerar a matriz de inteiros de dimensão 1024 x 1024 através do numpy, usei a função randint com os seguintes argumentos: - o primeiro argumento é o valor inteiro máximo (exclusivé) a partir do qual randomizar. Neste caso defini o valor 2, logo os valores gerados serão sempre os valores 0 ou 1.

- o segundo argumento é a dimensão da matriz. Uma vez que pretendemos gerar uma matriz de 1024 x 1024, os valores do segundo argumento são "size = (1024, 1024)".

[79]:
```
import numpy as np
```

[86]:
```
# gerar matriz com dimensao 1024 * 1024 e com valores aleatorios 0 e 1
my_array = np.random.randint(2, size = (1024, 1024))
print ("Array:\n" + str(my_array))
print ('\n')
```

```
Array:
[[0 1 0 ... 1 1 1]
 [1 0 0 ... 1 1 1]
 [0 1 0 ... 0 0 1]
 ...
 [1 0 0 ... 0 1 0]
 [1 0 0 ... 0 1 1]
 [0 0 0 ... 0 0 1]]
```

`b. Explorar serializar/deserializar A; usar pickle e JSON.`

Para serializar com o Pickle, usei as funções "dumps" e "loads" para primeiro gerar a matriz serializada e depois carregar o ficheiro para memória para poder ver a matriz.

[87]:
```
# serializacao com o pickle
dump_array = pickle.dumps(my_array) # gerar o array com o pickle
load_array = pickle.loads(dump_array) # carregar o array para memoria

print("Array with Pickle:\n" + str(loadArray))
```

```
Array with Pickle:
[[0 1 0 ... 0 1 1]
 [0 0 0 ... 0 0 1]
 [1 0 1 ... 1 1 0]
 ...
 [1 0 0 ... 0 0 1]
 [0 0 1 ... 0 0 1]
 [1 0 0 ... 1 0 0]]
```

3. Criar um "hello world" para o solver Z3 para resolver um problema de SAT aleatório gerado segundo as seguintes regras:

a. O problema tem N = 256 variáveis e M = 2 * N cláusulas.

b. Cada cláusula contém exatamente 3 literais escolhidos aleatoriamente dentro das variáveis disponíveis que, com igual probabilidade, ocorrem com ou sem conetiva de negação.

Para este exercício, começamos por gerar as M cláusulas com as 256 variáveis aleatórias. Para gerar as N variáveis de forma aleatória, usamos a função do exercício anterior para gerar valores entre 1 e 256. Após gerar as cláusulas, percorremos cada uma delas e guardamos no dicionário as que ainda não tiverem sido guardadas. Após isso, atribuimos o conectivo positivo ou negativo a cada um dos literais de forma aleatória. No fim de atribuir cada um dos conectivos aos literais de cada cláusula, esta é adicionada à lista de cláusulas. No final, usamos a função check() para verificar se o problema é ou não satisfazível, e se for imprimimos a solução.

[85]:
```
from z3 import *

n = 256; # numero de variaveis
m = n * 2; # numero de clausulas

# gerar as M clausulas com 3 literais cada
clauses = np.random.randint(1, n + 1, size = (m, 3))
randomConective = np.random.choice([-1, 1], 1)

#inicializacao do solver e do dicionario
s = Solver()
c = {}

for i in range(0, 512): # percorrer cada uma das 512 clausulas
    literal = [None] * 3
    for j in range(0, 3): # percorrer cada um dos 3 literais
```

```
        c[clauses[i][j]] = Bool(str(clauses[i][j])) # guardar no dicionario a␣
↪var actual
        if (clauses[i][j] * randomConective) < 0: # decidir se var tem valor␣
↪positivo ou negativo
            literal[j] = Not(c[clauses[i][j]]) # se negativo, guardar a negacao
        else:
            literal[j] = c[clauses[i][j]] # se positivo, guardar a afirmacao
        randomConective = np.random.choice([-1, 1], 1) # troca o valor aleatório
    s.add( Or( Or(literal[0], literal[1] ), literal[2] ) ) # adiciona a clausula␣
↪à lista

print(s.check())

if s.check() == sat:
    print(s.model())
```

```
sat
[58 = False,
 224 = False,
 71 = True,
 254 = False,
 239 = False,
 244 = False,
 89 = False,
 253 = False,
 148 = False,
 35 = True,
 192 = True,
 147 = False,
 198 = True,
 163 = True,
 91 = False,
 250 = False,
 145 = False,
 100 = False,
 10 = True,
 125 = False,
 119 = False,
 94 = False,
 54 = False,
 31 = False,
 24 = True,
 247 = False,
 144 = True,
 152 = True,
 181 = True,
 101 = False,
```

86 = False,
124 = False,
23 = False,
13 = False,
134 = False,
206 = False,
202 = False,
64 = False,
158 = True,
121 = True,
33 = False,
120 = False,
43 = True,
243 = True,
46 = True,
18 = True,
219 = False,
184 = True,
20 = True,
30 = True,
123 = False,
241 = True,
116 = False,
157 = False,
40 = True,
149 = True,
215 = False,
238 = True,
207 = True,
63 = True,
70 = False,
76 = True,
251 = False,
137 = True,
81 = True,
106 = False,
29 = False,
55 = False,
88 = False,
194 = False,
41 = False,
68 = True,
256 = False,
205 = True,
9 = False,
168 = False,
196 = True,
178 = True,

190 = True,
11 = False,
161 = True,
186 = False,
191 = True,
211 = False,
51 = True,
151 = False,
7 = False,
15 = False,
234 = False,
172 = False,
60 = True,
99 = False,
223 = True,
177 = False,
110 = True,
87 = False,
131 = False,
155 = True,
132 = True,
208 = True,
195 = True,
19 = True,
153 = False,
21 = False,
45 = True,
77 = False,
78 = False,
82 = False,
164 = False,
128 = False,
114 = False,
246 = False,
112 = False,
133 = True,
104 = False,
115 = True,
204 = True,
93 = False,
6 = False,
85 = True,
135 = False,
154 = False,
228 = False,
188 = True,
75 = False,
222 = False,

```
122 = True,
27 = False,
80 = False,
...]
```