SAT Solving

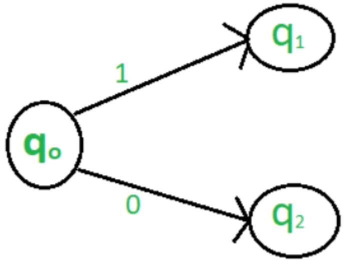
Tools Lab Assignment 5

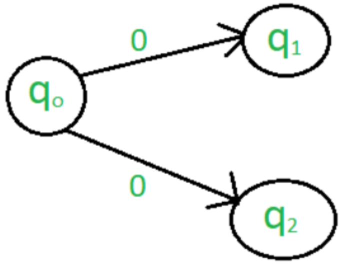
Millennium Prize problems

4	5				
3					6
	4		1		
		1		5	
2					1
				2	5

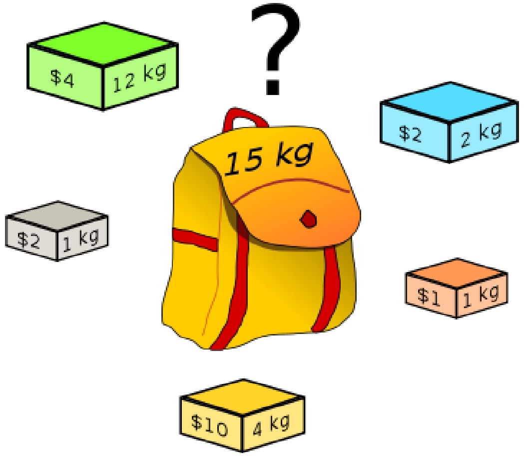
4	5	6	2	1	3
3	1	2	5	4	6
5	4	3	1	6	2
6	2	1	3	5	4
2	6	5	4	3	1
1	3	4	6	2	5

- 7 Mathematical problems (Clay Mathematics Institute)
  - \$1M to solve them or prove there is no solution
  - Only 1 has been claimed, other 6 remains far from being solved/proven
- P vs NP (Polynomial vs Non-deterministic Polynomial)
  - P = Solution in P time
    - Linear search – n
    - Binary search – log n
    - Sort – n^2
    - Merge sort – n log n





  - NP = Solution in exponential time but verification in NP time
    - Knapsack problem – 2^n
    - Sudoku – 2^n
    - Satisfiability problem (SAT) – 2^n
  - What about chess?



SAT Problem

a	b	c	Result
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

- Input : A set of clauses C with n variables
- Output : Is there an assignment of variable that satisfy all clauses?
- (a ∨ ¬b ∨ c) ∧ (¬a ∨ b ∨ ¬c) – CNF form
- (a ∧ ¬b ∧ c) ∨ (¬a ∧ b ∧ ¬c) – DNF form
- $8 = 2^3 = 2^n$
- Few SAT Solving algorithms
- Use solvers with their own algorithm

like Z3 from Microsoft Research

$$
\begin{aligned}
3x + 2y - z &= 1 \\
2x - 2y + 4z &= -2 \\
-x + \tfrac{1}{2}y - z &= 0
\end{aligned}
$$

```
#!/usr/bin/python
from z3 import *

x = Real('x')
y = Real('y')
z = Real('z')
s = Solver()
s.add(3*x + 2*y - z == 1)
s.add(2*x - 2*y + 4*z == -2)
s.add(-x + 0.5*y - z == 0)
print s.check()
print s.model()
```

```
sat
[z = -2, y = -2, x = 1]
```

Assignment 4

```python
RULE = [86 >> i & 1 for i in range(8)]
N_BYTES = 32
N = 8 * N_BYTES

def next(x):
    x = (x & 1) << N+1 | x << 1 | x >> N-1
    y = 0
    for i in range(N):
        y |= RULE[(x >> i) & 7] << i
    return y
```

- RULE=[0,1,1,0,1,0,1,0]
  - Equal number of 1s & 0s
- I'll simplify to 4 bits to show an example
  - X = 1101 = 13
  - LSB becomes 6th bit and X is shifted 1 bit to the right and MSB becomes new LSB
  - 1 | 1 1 0 1 | 1 = 111011 = 59
  - RULE is applied using truth table 3 bits at a time
  - 011 = 0
  - 101 = 0
  - 110 = 1
  - 111 = 0
  - New encoded 4 bits = 0100 = 4

i+2	i+1	i	res
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Convert Assignment 4 into a SAT problem

- Instead of manually reversing the cipher, apply a SAT solver to find the solution
- Hints:

  1) Model each bit of the input as an SAT variable

  2) Represent each bit of output as a SAT problem – Boolean expression in Disjunctive Normal Form (DNF) reflecting which bits of input (SAT variables) are used for computation of a particular bit in the output; this is specified by substitution table (i.e., RULE).

  3) You know the value of the output

  4) If you build a system of “SAT equations”, then let SAT solver to quantify the values of SAT variables with regards to known output.

More hints

- Now we're reversing the algorithm to work back to the seed
- Each bit of the next sub-key is dependent on 3 bits of previous key

next i	prev i+2, i+1, i
0	000
	011
	101
	111

next i	prev i+2, i+1, i
1	001
	010
	100
	110

- If bit of next sub-key is 0, then use the following SAT DNF form

$$(\neg a_{i+2} \wedge \neg a_{i+1} \wedge \neg a_i) \vee ((a_{i+2} \vee a_{i+1}) \wedge a_i)$$

- If bit of next sub-key is 1, then what will be the SAT DNF form?

Assignment 5

- Hand out : 12 Mar
- Hand in : 20 Mar
- Use python3 language
- *Python* file containing solution with SAT solver code
- *Write-up* with your explanation on your solution
- bash script *install.sh* to install nonstandard dependencies required for running of your solution
- Solution will be tested on contemporary Linux Ubuntu machine
- Rubrics (Solution – 4.5, Write up – 1.5)
- Reference
  - https://www.cs.cornell.edu/gomes/pdf/2008_gomes_knowledge_satisfiability.pdf
  - https://yurichev.com/writings/SAT_SMT_draft-EN.pdf