Practice/Real-Life Applications of Computational Algorithms, Fall 2016

Homework 2: Counting Sudoku solutions by using decision diagrams

1. Goal

By using decision diagrams, your task is to find the number of solutions to a Sudoku puzzle. Detailed steps are listed below:

(1) Read a Sudoku puzzle from the input
(2) Encode the puzzle to a Boolean expression
(3) Use PyEDA to construct the corresponding decision diagram
(4) Output the number of solutions

2. Input / Output

Each input file contains a puzzle that

(1) has a size N*N, and
(2) is prefilled with numbers 0 to N, where 0 represents an empty square.

Each output file should contain one number, which is the number of solutions to the input Sudoku puzzle.

Sample input	Sample Output
060104050 008305600 200000001 800407006 006000300 700901004 500000002 007206900 040508070	2

3. Command Line

Your program should take these arguments:
[input file (containing a puzzle)] [output file (containing a number)]

4. Programming Language

Python 3.x

5. Hand in Your Assignment

In a single folder, include your (*i*) source code and (*ii*) a report introducing your implementation. Zip the folder and submit only one *.zip file. Name the *.zip file with your student ID (e.g., 0456456.zip). Other filenames and formats such as *.rar and *.7z are NOT accepted!

6. Q&A

For any questions regarding Homework 2, please contact 林淯晨 (miz1205@gmail.com)

7. References

PyEDA: http://pyeda.readthedocs.io/en/latest/

About using PyEDA for this homework:

(1) PyEDA has been installed in all of the CSCC servers, including:
linux1.cs.nctu.edu.tw ~ linux6.cs.nctu.edu.tw
bsd1.cs.nctu.edu.tw ~ bsd6.cs.nctu.edu.tw

(2) After you get into the Python shell, do the following so as to be able to use PyEDA:
>>> from pyeda.inter import *

(3) Constructing decision diagrams involves deep recursion; therefore, do the following to enlarge the recursion limit:
>>> import sys
>>> sys.setrecursionlimit(10000)

(4) Suppose you are going to construct a decision diagram for Boolean function f = a ⊕ b ⊕ c, the conventional way is either:
>>> a, b, c = map(bddvar, “a b c”.split())
>>> f = a^b^c
or:
>>> f = expr(“a^b^c”)
>>> f = expr2bdd(f)
Then, if you would like to assign some variable(s) to constant 0 or 1, e.g., assign variable a to 0 and variable b to 1 (as what you should do for those prefills in the puzzle), the conventional way is:
>>> f = f.restrict({a:0, b:1})
If you have to construct a decision diagram for a very complex Boolean expression, the above approach involves creating a very complex diagram first and then reducing its size (by doing “restrict”). Overall it is very inefficient and time-consuming. To save time, I suggest the following:

>>> f = expr("0^1^c")

That is, include constants in the expression. By doing so, you don't even have to do "expr2bdd" since "restrict" in no longer required. Yes, you don't even have to construct a decision diagram. In fact, the decision diagram corresponding to the Boolean expression implicitly exists after doing "expr" to describe the expression and declare/create relevant data structures in the Python/PyEDA environment.

(5) The final hint: you may want to use "satisfy_all" to find the answer to this homework!