

Notation

I hate the logical operator symbols used in our textbook because they aren't on my keyboard. So, I'm using this system:

Logical Meaning	Textbook Version	My Version
not x	$\neg x$	x'
x and y	$x \wedge y$	xy
x xor y	$x \oplus y$	$x + y$
x or y	$x \vee y$	$x y$

To reduce the number of parenthesis needed in expressions, I take the following order of operators precedence:

$$\mathbf{not} > \mathbf{and} > \mathbf{xor} = \mathbf{or}.$$

Introduction

We have already seen that given a truth table for a Boolean function, we can write the Disjunctive Normal Form (DNF) of the function. However, the disjunctive normal form often is not the most efficient way to compute a Boolean function. Here is an example:

Consider the Boolean function described by the table below.

a	b	c	$f(a, b, c)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

In this case, the DNF is

$$f(a, b, c) = a'b'c | ab'c' | abc' | abc. \quad (1)$$

The expression for f given by equation 1 is logically correct, but it requires eleven operations (eight **ands** and three **ors**; we won't count the negation of individual variables as separate operations because in most digital systems you can get the negation of a variable with no extra cost). But, with a little bit of playing around, I found another expression for f as

$$f(a, b, c) = a'b'c | ac' | ab, \quad (2)$$

which requires six operations (four **ands** and two **ors**). With even more thought, I came up with

$$f(a, b, c) = b'(a + c) | ab, \quad (3)$$

which requires only four operations (one **xor**, two **ands**, and one **or**). But then, I drank some coffee, and I came up with

$$f(a, b, c) = a' + (b | c'), \quad (4)$$

which only requires two operations. This is an optimal representation for our function.

This process of logical reduction of a Boolean equation is a critical part of all computing. In circuit design, getting minimal representations for functions makes the hardware faster and allows more functionality to be placed on a single piece of silicon. In compiled code, getting minimal representations of logical expressions allows the program to run faster and take up less storage.

But, there's a problem. There is no known algorithm for efficiently finding the optimal representation for a Boolean function. (For future reference, this is an "NP Hard" problem.)

Which brings us to your assignment:

The Problem

Your team (not to exceed three people) must write a module in Sage (or Python) which will:

1. Read in a plain text file which contains a list of values for the function.
2. Find the DNF for the function.
3. Print out the number of operations required for the DNF followed by the DNF using the notation I listed above.
4. Find a "good" representation for the function.
5. Print out the number of operations required for the good representation followed by the representation using the above notation.

The better your representation for the various functions I provide for input, the better your grade. A program which can correctly provide the DNF and operation count for all the input files I feed it will receive at least a "C" grade.

A program which correctly computes all the DNF and operation counts and computes correct "good" representations (and operation counts) which are on average at least 20% below the DNF count will receive at least a "B".

A program which correctly computes all the DNF and operation counts and computes correct "good" representations (and operation counts) which are on average at least 40% below the DNF count will receive at least an "A".

Furthermore, I will buy dinner at Clementine's for the team whose program has the best over all performance!

Input/Output Format

The input file will be a plain text file (ASCII text, not Unicode) with a single character on each line. The character on each line will be either '0' or '1' and will represent the value of the Boolean function. The line

number, in binary, starting at 0, will represent the value of the inputs. For example, for the truth table given above, the input file would be:

```
0
1
0
0
1
0
1
1
```

The maximum size file I will give you will have 256 entries which corresponds to a Boolean function with eight inputs. Label your inputs from most significant to least significant bit beginning with 'a' and proceeding alphabetically.

You do not need to do any error checking on the input.

Given the input file above, your program should return something like this:

```
11 f(a,b,c) = a'b'c | ab'c' | abc' | abc
6 f(a,b,c) = a'b'c | ac' | ab
```

or it might return something like this:

```
11 f(a,b,c) = a'b'c | ab'c' | abc' | abc
2 f(a,b,c) = a' + (b | c')
```

which would be even better.

You will need to make an appointment to come demonstrate your code to me. (I find that it is often better to have teams demonstrate their code so that they can make last minute corrections.)

How not to fail this assignment

Begin working on this as soon as possible. You should expect to be completely confused when you begin this project. That is natural. Begin with small steps. Read the Python Tutorial from <http://www.python.org> and make yourself learn how to do simple things like print output and read input. Write a simple function to read the input file into a list. Make some sample input files for yourself and test out your program.

Figure out how you will deal with the representation of the Boolean function internally. Make sure you can do the "C" level assignment first. Only after you have the "C" level part working should you bother trying to figure out how to get better representations.

Honor Code

Don't use code from other teams. Within your group, you must make sure that everyone contributes to the project. Don't let one person do all the work. If you have a team member who isn't contributing, then talk with me about it.