2-SAT and Biconnected Components

bhavit.sharma804

April 17, 2019

1 2-sat

1.1 Introduction

Consider a boolean function

$$f = (x_1 \vee y_1) \wedge (x_2 \wedge y_2) \cdots (x_n \vee y_n)$$

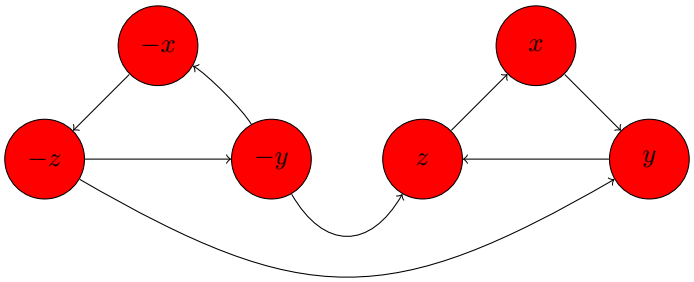
Is f satisfiable? To make $(a \vee b) = 1$, we need to consider two cases:

- if $a = 0$, then b must be either 1/0. Hence $-a \implies b$
- if $b = 0$, then a must be either 1/0. Hence $-b \implies a$

Hence, for every clause $a \vee b$, we will add two clauses $(-a \implies b)$ and $-b \implies a$ in the graph. For example, if our clause

$$\phi = (-x \vee y) \wedge (-y \vee z) \wedge (x \vee -z) \wedge (z \vee y)$$

Hence, our 2-CNF graph will look like this.



1.2 Solution

Let's say we have n clauses in our boolean function. The total number of vertices and edges in graph are $\mathcal{O}(n)$. Some observations which might help us.

Observation 1.1. *If there exists a path from node i to j, then there exists a path from node $-j$ to $-i$.*

Proof. This is because whenever we insert $a \implies b$ to the set, we also insert $-b \implies -a$ in the set. Hence, if there exists a path $(a, a_1, a_2, \cdots, b)$ then, the path $(-b, \cdots, -a_2, -a_1, -a)$. □

Observation 1.2. *If there exists a path from x to $-x$ and path from $-x$ to x, then boolean function is not satisfiable.*

Proof. Let us assign the value of `TRUE` to x. Since, a path from $-x$ exists, we will get the value of $-x$ to be `TRUE` which is a contradiction. If we assign the value of x to be `FALSE`, we again get a contradiction, since path from $-x$ to x exists. □

Observation 1.3. *As it turns out, Observation 1.2 is both Necessary and Sufficient.*

Proof. To begin the proof of the *sufficient* condition, we will present a constructive algorithm and then analyse its correctness. □

Algorithm

- Create the directed graph and decompose it into Strongly Connected Components(SCC).
- Topological sort the SCCs and create an array `comp[i]` denotes the position of node's SCC in top-sort of the SCCs.
- If `comp[i] < comp[-i]`, then we assign i = `FALSE` else `TRUE` otherwise.
- If `comp[i] == comp[-i]`, then it's not satisfiable.

Now, let us consider the case when x = `TRUE` i.e. `comp[-i] < comp[i]`, which means that x and $-x$ **doesn't lie** in the same component.

Theorem 1.1. *There doesn't exists a node y such that $-x$ has a path to y and $-y$, both.*

Proof. We'll prove this by contradiction. If $y, -y$ are reachable from $-x$, then x is reachable from $-y, y$ (By Observation 1.1). Hence, x is reachable from $-x$, and $-x$ is reachable from x, which means that they lie in the same component (a contradiction). □

We have constructed an algorithm which find the correct value of the constraints, under the assumption that x and $-x$ lie in different components. Hence, we have proved that the Observation 1.2 is both Necessary and Sufficient.

1.3 Conversion of Two-variable Boolean Function

Till now we've seen the clauses to be boolean function of LOGICAL OR, but the clauses can be any function, like XOR, NAND, NOR etc. For example our boolean function might look like this.

$$f = (x_1 \text{xor} x_2) \wedge (-x_1 \vee x_3) \wedge (-x_3 \text{nand} x_1)$$

To convert any boolean(2 variables) function into an equivalent 2-CNF form, we'll perform the *Products of Sum* transformation to the truth table of the boolean function. For example, the truth table of XOR is:

0	1	1
0	0	0
1	1	0
1	0	1

So, its POS is $(a + -b)(-a + b) = (a \vee -b) \wedge (-a \vee b)$.

1.4 Implementation

Link to Code. The code is implemented in such a way that you just have to provide the truth table of the boolean function to class, and it'll handle all of the stuff. Important thing to keep in mind is, it's 1 based indexing. To get the value of actual assignment, just look at the `std::vector comp[i]`

1.5 Problems

TODO