MAC0328 GRAPH ALGORITHMS: PROGRAMMING ASSIGNMENT 1

2-SATISFIABILITY

MARCEL K. DE CARLI SILVA AND HELOISA DE LAZARI BENTO

DUE DATE: 29/SEP/2024 AT 11:59PM

1. INTRODUCTION

In this programming assignment, your task is to write a program that finds a truth assignment for a 2-SAT instance (a logical formula in CNF where each clause has at most 2 literals), or determine that none exists.

This assignment will be graded out of 80 marks. This grade will summed to the assignment 0 grade to complete one official grade for assignment 1 out of 100 marks.

Your code **must** be written in `C++17` and use the BGL library.

2. DEFINITIONS

Let $n \geq 1$ be an integer, and let $x_1, \ldots, x_n$ be *boolean variables*, i.e., they can take on the logical values **true** or **false**. A *literal* is either a variable or its negation, i.e., x_j or $\neg x_j$ for some $j \in [n]$.

A *(disjunctive) clause* (or a *disjunction*) is a logical expression of the form $\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k$, where $k \geq 1$ is an integer and each one of $\ell_1, \ell_2, \ldots, \ell_k$ is a literal. The symbol '$\vee$' denotes "logical or", i.e., a disjunction. The notation is **not** meant to imply that ℓ_j is a literal corresponding to the variable x_j. Some examples of clauses are

$$\neg x_5 \vee x_7 \vee x_2$$

and

$$x_3.$$

Another notation for the disjunction $\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k$ is $\bigvee_{j=1}^{k} \ell_j$.

A formula in **conjunctive normal form** (**CNF**) is a logical expression of the form

$$C_1 \wedge C_2 \wedge \cdots \wedge C_m,$$

where each C_i is a clause. The symbol '$\wedge$' denotes "logical and", i.e., a conjunction. An example of a formula in CNF is

$$\Phi = \underbrace{(x_1 \vee x_3 \vee \neg x_2)}_{C_1} \wedge \underbrace{x_2}_{C_2} \wedge \underbrace{(\neg x_1 \vee \neg x_2)}_{C_3} \wedge \underbrace{(\neg x_3 \vee \neg x_2)}_{C_4}.$$

Another notation for the conjunction $C_1 \wedge C_2 \wedge \cdots \wedge C_m$ is $\bigwedge_{i=1}^{m} C_i$. Hence, if one considers a clause to be a set of literals (rather than a logical expression), a CNF formula is an expression of the form $\bigwedge_{i=1}^{m} \bigvee_{\ell \in C_i} \ell$.

A **truth assignment** for a CNF formula Φ is an assignment of logical values **true** or **false** to the variables of Φ, i.e., a function $t\colon \{x_1, x_2, \ldots, x_n\} \to \{\textbf{true}, \textbf{false}\}$. A truth assignment t for Φ

INSTITUTO DE MATEMÁTICA E ESTATÍSTICA, UNIVERSIDADE DE SÃO PAULO, R. DO MATÃO 1010, 05508-090, SÃO PAULO, SP
E-mail address: {mksilva,heloisa}@ime.usp.br.

is **satisfying** if all the clauses of Φ are satisfied, i.e., each clause of Φ has at least one literal with logical value equal to **true**.

The problem that your program must solve is the following. **Given** a CNF formula Φ where each clause has at most two literals, **find** a satisfying truth assignment for Φ, or **determine** that none exists.

2.1. **Auxiliary Digraph.** In determining the existence of (and finding) a satisfying truth assignment for a CNF formula Φ, where each clause has exactly 2 (potentially equal) literals, it will be helpful to consider the following **auxiliary digraph** $D_\Phi = (V, A)$. We have one vertex for each possible literal, i.e., two vertices for each variable, one for the variable itself and one for its negation. And for each clause $\ell_a \vee \ell_b$, potentially with $\ell_a = \ell_b$, we add the arcs $(\neg\ell_a, \ell_b)$ and $(\neg\ell_b, \ell_a)$; naturally, $\neg\neg x_i$ is the same as x_i. More formally, if $x_1, \ldots, x_n$ are the variables of $\Phi = C_1 \wedge C_2 \cdots \wedge C_m$ and each clause C_i has the form $C_i = \ell_a^i \vee \ell_b^i$, then

$$V = \{x_1, x_2, \ldots, x_n\} \cup \{\neg x_1, \neg x_2, \ldots, \neg x_n\}$$

and

$$A = \{(\neg\ell_a^1, \ell_b^1), (\neg\ell_a^2, \ell_b^2), \ldots, (\neg\ell_a^m, \ell_b^m)\} \cup \{(\neg\ell_b^1, \ell_a^1), (\neg\ell_b^2, \ell_a^2), \ldots, (\neg\ell_b^m, \ell_a^m)\}.$$

Thus, D_Φ has $2n$ vertices and at most $2m$ arcs.

3. Examples

In this section, we give two examples of CNF's. Consider the formula:

$$\Phi = (\neg x_1 \vee \neg x_2) \wedge (x_3 \vee \neg x_1) \wedge (x_2 \vee \neg x_3) \wedge (x_4 \vee \neg x_3).$$

If we assign the value **true** to x_2 and x_4, and **false** to x_1 and x_3, then all clauses of Φ will be satisfied.

Now consider this other 2-SAT instance:

$$\Psi = (x_1 \vee x_1) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_2 \vee x_3).$$

We claim that no truth assignment is satisfying for Ψ. We can prove this by testing all possible truth assignments. In the following table, for every possible assignment we exhibit one of the clauses that is not satisfied.

x_1	x_2	x_3	unsatisfied clause
F	F	F	$(x_2 \vee x_3)$
F	F	T	$(x_1 \vee x_1)$
F	T	F	$(x_1 \vee x_1)$
F	T	T	$(x_1 \vee x_1)$
T	F	F	$(x_2 \vee x_3)$
T	F	T	$(\neg x_1 \vee \neg x_3)$
T	T	F	$(\neg x_1 \vee \neg x_2)$
T	T	T	$(\neg x_1 \vee \neg x_2)$

Table 1. Exhaustive verification that Ψ has no satisfying truth assignment.

4. Test Cases and Grading

Your program should solve each test case in $O(n+m)$ time, where n is the number of variables and m is the number of clauses.

Each test case has the following format:

- The first line has an integer $d \geq 0$ by itself. This shall be better explained in Section 5.
- The second line has two integers, n and m, such that $1 \leq n \leq 10^5$ and $1 \leq m \leq 10^5$. Here n is the number of variables, which are $x_1, x_2, \ldots, x_n$, and m is the number of clauses.
- The next group of m input lines encodes the clauses, $C_1, \ldots, C_m$. For each $i \in [m]$, the ith line in this group has two (potentially equal) integers, a and b, describing the clause C_i, with the following interpretation. If $a > 0$, then the literal 'x_a' appears in clause C_i; if $a < 0$, then the literal '$\neg x_{-a}$' appears in clause C_i. The same interpretation applies to the integer b.

If $d = 0$, the behavior should be as described in the next paragraphs. If $d > 0$, your program may safely finish, or proceed as described in Section 5.

The program prints `YES` if the formula has a satisfying truth assignment $t\colon \{x_1, x_2, \ldots, x_n\} \to \{\textbf{true}, \textbf{false}\}$. The next output line should have n integers, representing a satisfying truth assignment as follows. The value of $t(x_1), t(x_2), \ldots, t(x_n)$ should be printed in order, by printing `1` to represent the logical value **true**, and by printing `0` to represent the logical value **false**. All these tokens must be separated by a single blank space.

If no satisfying truth assignment exists, the program prints `NO` in the first line. The second line should have a variable certifying the unsatisfiability of the instance; refer to the exercise in section 2 of lecture 5. In the next line the program prints a path from that variable to its negation in the auxiliary digraph. The first integer of this line is ℓ, the length of the path, followed by $\ell + 1$ vertices. The vertex x_i is represented by the integer i and the vertex $\neg x_i$ by $-i$. The last line will contain a path from the negation of the variable to the variable itself, in the same format as described before.

There are approximately 50 test cases. Each test case has exactly one 2-SAT instance. You will have access to approximately 10 of these 50 test cases, which you may use to debug and test your code. Your grade will **not** be directly proportional to how many test cases it answers correctly; rather, we will consider all test cases as an aggregate. For example, a code that always prints `NO`, or always prints `YES`, will not earn half of the marks.

Warning: Note that even if your code passes all 10ish public test cases, it may fail to work on the remaining, "secret" test cases. If the code does not pass all public cases, then it will probably fail in most "secret" cases.

You can find the public test cases in the public repository for the course at https://gitlab.uspdigital.usp.br/mksilva/mac0328-2024-public/-/tree/master/assignments/asgt1/tests.

You should submit **exactly** one file through Moodle named `main.cpp`.

5. Fallback Grading

To avoid having an "all or nothing" marking scheme, we will allow your program to enter some debugging state. Correctly implementing this debugging functionality is **optional**. The first element in the input description from Section 4, namely the integer $d \geq 0$, describes the desired *debugging level* of your program execution; thus $d = 0$ indicates no debugging at all. When running your code on some input test case, if the program's answer is incorrect, we will run your code with a higher debugging state and assign partial credit (that is decreasing with increasing debugging levels).

5.1. **Debugging Level 1: Strong Components of Auxiliary Digraph.** If $d = 1$, then the output of your program should be a list of $2n$ integers, one for each of the literals $x_1, \ldots, x_n$ and $\neg x_1, \ldots, \neg x_n$ of the input formula Φ. Let `nscc` denote the number of strongly connected components of the auxiliary digraph $D_\Phi = (V, A)$. Your program should print the integers $s(x_1), \ldots, s(x_n), s(\neg x_1), \ldots, s(\neg x_n)$,

where $s\colon V \to [1, \texttt{nscc}]$ is any valid labeling[1] of the vertices of D_Φ into the strong components of D_Φ.

5.2. **Debugging Level 2: Digraph Construction.** If $d = 2$, then the output of your program should be a description of the digraph $D_\Phi = (V, A)$ in the following format. First print two integers, $|V|$ and $|A|$. After this the output should print $|A|$ lines, one for each arc, and each one containing two integers. An arc of the form (ℓ_1, ℓ_2), where ℓ_1 and ℓ_2 are literals, should be printed in the reverse format of the input description. Namely, if ℓ_1 is a variable x_i, print i, and if ℓ_1 is the negation of a variable, e.g., $\ell_1 = \neg x_i$, print $-i$. The same applies to the second literal ℓ_2.

You do not need to worry about the existence of *parallel* arcs, e.g., two arcs joining the same pair of vertices. That is, your program may print multiple arcs between two vertices or not, without being penalized. However, you must make sure to have $|A| \leq 2m$.

[1]That is, for each $k \in [1, \texttt{nscc}]$, the fiber $s^{-1}(k) = \{\, v \in V : s(v) = k\}$ is a strong component of D_Φ.