CPTS516 Take-home Final Exam

Solving linear Diophantine constraints using labeled graphs

(For students who planned, at the beginning of this semester, to take exams at Access Center, you may text me 509-338-5089 for extended time for completing this take home final exam. Start now before it is too late.)

You may use Python, C, C++, or Java (or any other programming language that you are familiar with) to do the implementation. **All the work must be your own**; collaboration is prohibited. You can search on the Internet but mostly, you will waste your time. I design the exam so that it is googling-proof.

As we have learned this semester, NP is the class of problems that can be solved by nondeterministic algorithms in polynomial time. In particular, NP-complete problems are the hardest ones in NP. Currently, it is open whether we have efficient solutions to those problems. That is, many researchers are still trying to find (deterministic) polynomial time algorithms to solve those NP-complete problems, or at least to find practically efficient algorithms for them. Such efforts would lead to successfully cracking RSA, which is known in NP (but we do not know whether cracking RSA, i.e., factorizing a large number, is NP-complete). There is a well-known NP-complete problem, called linear Diophantine (written LD), which seeks nonnegative integer solutions to a linear constraint system Q over multiple variables $x_1, \cdots, x_k$, for some k; e.g.,

$$\begin{cases} 2x_1 + x_2 + 15 = 0 \\ 3x_1 - 4x_2 > 18 \\ x_1 + 3x_2 < 27 \\ x_1 > 15 \end{cases} \tag{1}$$

The example LD instance shown in (1) has two nonnegative integer variables x_1 and x_2 and four constraints. Notice that the range of the variables is unbounded; hence, you can not assume that they are in 32-bits unsigned ! Please stare at the example for 10 minutes and see how you would design an algorithm to find whether it has solutions and if it has, how you would come up with a solution. There is a brilliant idea which we learned this semester in solving LD: using the automata-based algorithm for Presburger arithmetic, where a linear constraint is represented as an NFA (hence a labeled graph). (Do not even try to solve LD using equation-solving skills (called Newton elimination) you learned in high school; variables in those high school equations are of real values and do not apply here.)

Formally, an LD-instance Q is given as, for some k and m,

$$\begin{cases} C_{11}x_1 + \cdots + C_{1k}x_k + C_1 \quad \#_1 \;\; 0 \\ \qquad\vdots \\ C_{m1}x_1 + \cdots + C_{mk}x_k + C_m \quad \#_m \;\; 0 \end{cases} \tag{2}$$

where all the x_j's are nonnegative integer variables (called unknowns), and the following are all the *parameters*:

- all the coefficients C_{ij}'s, which are integers (positive, negative, zero), and
- all the numbers C_i's, which are **nonnegative integers**, and, finally,
- all the $\#_i$'s, each of which is in $\{>, <, =\}$.

Excercise: What are the the values of the m, the k and the parameters for the example LD instance shown in (1) ?

Any algorithm that solves the LD problem is to take an instance Q in (2) as the algorithm's input, and to return yes (along with a solution) if the instance has a nonnegative integer solution in the unknowns $x_1, \cdots, x_k$, and to return no if otherwise. Notice that the time complexity function of the algorithm measures on the size of the input. (Hint: the size of number 4.6 billions is roughly 32.)

The remaining of the exam asks you to implement the brilliant idea in solving the LD problem: representing each linear constraint (such as $2x_1 - 3x_2 + 4 = 0$) in an instance Q using a labeled graph (i.e., a finite automaton). Hence, since there are m constraints in the instance Q, you need then implement a graph composition algorithm to convert the m graphs into one final graph. Then, you perform basic graph search on the final graph to finally solve the LD problem.

1. (70pts) Watch the class videos on Presburger arithmetic, where I presented the aforementioned brilliant ideas (which were invented decades ago by some of our brightest ancestors in algorithms) and you take notes while watching. Make sure that you fully understand the ideas. In this project, you are going to implement those ideas to solve an instance Q in three variables and with 2 equations. Being Diophantine, the variables are of nonnegative integers. I do not ask you to design the algorithm, instead, I present the algorithm's design, and you need only implement the algorithm. You need turn-in working code and prepare for a demo.

Preparation 1.

Herein, an equation is in the form of

$$C_1x_1 + C_2x_2 + C_3x_3 + C = 0 \tag{3}$$

where constants C_1, C_2, C_3 are integers (positive,negative, zero), and constant C is **nonnegative**. Hence, for instance, $3x_1 + 0x_2 - 4x_3 - 17 = 0$ is not an equation in the above form. However, equivalently, $-3x_1 - 0x_2 + 4x_3 + 17 = 0$ is in the above form.

For the equation in (3), we define

$$C_{\texttt{max}} = \max_{d,d_1,d_2,d_3 \in \{0,1\}} |C_1d_1 + C_2d_2 + C_3d_3 + d|. \tag{4}$$

(Exercise: For the aforementioned equation $-3x_1 - 0x_2 + 4x_3 + 17 = 0$, what is the value of $C_{\texttt{max}}$?) **Implement** a function that returns $C_{\texttt{max}}$ from the description of an equation in (3), and use the above Exercise to test your function.

Preparation 2.

For a constant $C \geq 0$, we use binary representation for C. For instance, if $C = 34$, then in binary, $C = 100010$. In this case, we use

$$b_6b_5b_4b_3b_2b_1$$

for it, with $b_6 = 1, b_5 = b_4 = b_3 = 0, b_2 = 1, b_1 = 0$. Herein, we define $K_C = 6$ (the number of bits needed to represent C). In particular when $C = 0$, we let $K_C = 0$.

Hence, for $1 \leq i \leq K_C$, the b_i is defined as above. However, for $i = K_C + 1$, the b_i is now defined as 0. (Exercise: Let $C = 18$. What is the value of K_C? what is the value of b_6? what is the value of b_4?)

Implement a function that returns the value K_C from a given constant $C \geq 0$.
Implement a function that returns the value b_i from a given constant $C \geq 0$ and i, noticing that the i shall be in the range of $1 \leq i \leq K_C + 1$.

Algorithm 1. Equation to labeled graph (automaton)

We now construct a finite automaton M from the description of the equation in (3).

The input alphabet of M contains exactly eight input symbols, where each symbol is in the form of (a_1, a_2, a_3) with $a_1, a_2, a_3 \in \{0, 1\}$. (Hence, an input symbol is a triple of three Boolean values.)

A state in M is a pair of values: $[carry, i]$, where

$$-C_{\texttt{max}} \leq carry \leq C_{\texttt{max}},$$

recalling that $C_{\texttt{max}}$ is defined in (4), and $1 \leq i \leq K_C + 1$.

The initial state in M is $[carry = 0, i = 1]$. The accepting state is $[carry = 0, i = K_C + 1]$.

For all states $[carry, i]$ and $[carry', i']$ and all input symbols (a_1, a_2, a_3), the following is true:

$[carry, i] \overset{(a_1,a_2,a_3)}{\longrightarrow} [carry', i']$ is a transition in M (i.e., M moves from state $[carry, i]$ to state $[carry', i']$ on reading input symbol (a_1, a_2, a_3)) if and only if the following is true:

> Let $R = C_1a_1 + C_2a_2 + C_3a_3 + b_i + carry$. Then, R is divisible by 2, and $carry' = \frac{R}{2}$. Furthermore, if $1 \leq i \leq K_C$, then $i' = i + 1$, else $i' = i$. (You shall use the functions **implemented** above to implement this step; e.g., to compute the b_i.)

Implement a function that returns the finite automaton M from the description of an equation in (3). Notice that you shall use a graph as the data structure of the automaton M. (The automata we constructed are all deterministic by definition.)

Algorithm 2. Cartesian product of two labeled graphs (i.e., two automata)

Now we are given a system of 2 equations over three variables x_1, x_2, x_3, where we use $E_1(x_1, x_2, x_3)$ and $E_2(x_1, x_2, x_3)$ to indicate the two equations and recall that the equations are in the form of (3). Suppose that, using the Algorithm 1 presented earlier, we obtain finite automata M_1 and M_2 from the two equations respectively. Then, we need construct a finite automaton $\mathcal{M}$ that is the Cartesian product of M_1 and M_2. (For instance, `https://www.youtube.com/watch?v=QSpErcGyXPM` which also works for NFA's)

Implement a function that returns the $\mathcal{M}$ from the M_1 and the M_2 (which are computed using the function implemented in Algorithm 1 from descriptions of the two equations). Again, the resulting automaton $\mathcal{M}$ uses a graph as its data structure.

Final step.

If there is no path from the initial to the accepting in $\mathcal{M}$, then the equation system does not have solutions, else we can find a solution by using DFS on the graph of the $\mathcal{M}$ (from the initial to the accepting while collecting the sequence of input symbols on the path). Notice that you shall reverse the sequence and then convert it into digit before you output the solution. I will sketch a little more detail of this step. Suppose that the following sequence of input symbols is collected on a path from the initial to the accepting:

$(1, 0, 1), (0, 1, 1), (1, 0, 0), (1, 1, 0)$

then, what is the solution to the equation system? it is 1011, 0101, 1100 (how did I do it? I got 1011 by picking the first bit from each input symbol in the sequence!). After reversing them, I have 1101, 1010, 0011. Then I convert them into digits: 13, 10, 3. Hence, the solution is actually $x_1 = 13, x_2 = 10, x_3 = 3$.

Please also **implement** this final step and use the following two to test your code:

T1. The following equation system does not have nonnegative integer solutions:

$3x_1 - 2x_2 + x_3 + 5 = 0$

$6x_1 - 4x_2 + 2x_3 + 9 = 0$

T2. The following equation system does have nonnegative integer solutions (and manually verify that the solution found after you run your code indeed satisfies the following constraints):

$3x_1 - 2x_2 - x_3 + 3 = 0$
$6x_1 - 4x_2 + x_3 + 3 = 0$

You shall turn in source code (with comment) and screenshot of your running results on the two test cases. I may, during the demo, give you an instance for your code to solve.

2. (20pts) Tell me, in detail, how to generalize the algorithm that you implemented (for two constraints and three variables) to a general LD instance Q in (2).

3. (70pts) Write an essay (at least 3 pages including proper literature search) to

- Define one or more similarity metrics between two general LD instances in (2) and justify your definition(s).
- Describe an application that your metrics can be used.

Guidelines: Your writeup must be in your own words. If you use others' ideas (e.g., from the Internet), you must cite properly with proper citations, and still you write the ideas in your own words! If you copy and paste text from the Internet or other sources, no matter whether you cite the source or not, you will automatically receive 0.

Guidelines: Your solutions must be prepared in Latex and uploaded as a PDF. You use 12pt font, single space, and with margins 1.3inch on four sides, in Latex. I use the same format to prepare this document, and my latex source for the format control is as follows (you are welcome to copy and paste my format):

```
\documentclass{article}
\usepackage[letterpaper, margin=1.3in]{geometry}
\usepackage{amsmath,systeme}
\usepackage{hyperref}
\begin{document}
your write-up is here.......
\end{document}
```

Grading criteria: Probs 1: if your code is perfectly working, you get full score. Otherwise, partial credit may be given. Prob 2: correctness and readability. If you write code in place of English, you receive 0. Prob 3: it will be graded on correctness, logical clarity, depth of thinking, and quality of writing. Please use complete sentences and thread them logically. For instance, below is such a bad way to write: **My essay – It is raining today. The vaccine works. WSU is a good school.**

I will not accept any hand-written solutions! Please type.