Algorithms
Computer Science 140 & Mathematics 168
Spring 2012
Homework 10b
Due Tuesday, April 3

1. **[15 Points] Hitting Set!** First, the gratuitous story: The Computer Science Department at The Western Institute of Technology (T.W.I.T.) has 5 faculty members, Dr. I. "Theo" Rize, Dr. I. Dutu, Dr. Sam Heer, Dr. Mia Thue, and Dr. Juan More. Each professor serves on a number of committees. The Polynomial Time Algorithms Committee consists of Professors Rize, Dutu, and More. The NP-Completeness Committee consists of Professors Dutu, Heer, and Thue. Finally, the Other Stuff Committee consists of Professors Thue and More.

   The Dean would like to convene a meeting in which the number of computer scientists invited is minimized (she can't stand their bad CS jokes!) but each committee is represented by at least one of its members. For example, inviting Professors Thue and More would be a solution in this case.

   The Dean has formulated this problem as the following general problem (known to computer scientists as the HITTING SET Problem): We are given a set S (the set of professors) and a collection, C, of subsets of S (the set of committees).

   A Hitting Set for C is a subset $S' \subseteq S$ such that S' contains at least one element from each subset in C. The HITTING SET optimization problem is to find the smallest Hitting Set.

   (a) Formulate the HITTING SET decision problem corresponding to the optimization problem described above.

   (b) Prove that the HITTING SET decision problem is NP-complete. While it's theoretically possible to reduce from any NP-complete problem, to make your life easier, try to reduce from one that has a structure that is as similar as possible to that of HITTING SET. This is good advice in general when working on a NP-completeness proof!

2. **[20 Points] The Partition and Bin Packing Problems!**

   (a) Let $S = \{a_1, \ldots a_n\}$ and let $f : S \to \mathbb{Z}_+$ (that is, f maps elements of S to the positive integers). A *partition* of S is a pair of sets A, B such that $S = A \cup B$ and $A \cap B = \emptyset$. The Partition Problem asks if there exists a partition of S into sets A, B such that

   $$\sum_{x \in A} f(x) = \sum_{x \in B} f(x)$$

   Prove that the Partition Problem is NP-complete using a reduction from a problem that we have seen in class or on a previous homework assignment. (Notice

that while this definition of the problem is seemingly awkward, it permits us to have multiple items with the same value, that is a "multiset" of numbers.)

(b) The Bin Packing Problem is the following: We are given a collection of objects $S = \{a_1, \ldots, a_n\}$, a function $s : S \to \mathbb{Z}_+$ indicating the "size" of each object, a bin capacity C such that C is greater than or equal to the size of the largest object, and a target t. The question is whether or not it is possible to pack all of the objects in S into t or fewer bins, each of capacity C. Prove that Bin Packing is NP-complete. (This problem comes up in *many* practical applications. One application is that of packing "stuff" into the least number of shipping containers. Another application is that of packing data into the least number of "blocks" - where a block is the smallest unit that a file system is willing to store.)

3. **[30 Points] 2SAT!** 2-Satisfiability, often called 2SAT, is a classic problem in logic that arises in applications in Artificial Intelligence and elsewhere. The problem goes like this: We are given a collection of n boolean variables $x_1, \ldots, x_n$. (That is each variable can take the value TRUE or FALSE.) In addition, we are given clauses where each clause comprises the disjunctions (logical OR) of exactly two literals, where a literal is a variable or its negation. We wish to find an assignment of TRUE or FALSE to each variable such that all of the clauses are satisfied. For example, here is an instance of 2SAT:

$$(x_1 \vee x_2) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3})$$

Note that $\vee$ is the logical OR, $\wedge$ is the logical AND and $\overline{x}$ denotes the logical negation of variable x. Saying that we want to satisfy each of the clauses is exactly the same thing as saying we want to satisfy the conjunction (AND) of all of the clauses. In this example, we can satisfy the expression by making x_1 be TRUE, x_2 be FALSE, and x_3 be TRUE.

As we'll see very soon, the evil twin of this problem, 3SAT, in which each clause contains the disjunction of exactly 3 literals is NP-complete: There is no known polynomial-time algorithm for that problem and there is strong evidence that suggests that no polynomial-time algorithm exists.

Amazingly, 2SAT is solvable in polynomial time and you'll show that here! In particular, in this problem, we will see that 2SAT can be solved using graph algorithms instead.

(a) Consider the following instance of 2SAT:

$$(x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$$

Show that this instance is satisfiable by giving a satisfying assignment for the variables.

(b) Now consider the following instance of 2SAT:

$$(x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2})$$

Explain briefly why this instance of 2SAT is not satisfiable.

(c) Given an instance of 2SAT, let's construct a corresponding directed graph as follows: For each variable x_i that appears in the 2SAT instance, construct a *pair* of vertices, one labeled x_i and the other labeled $\overline{x_i}$. For every clause of the form $(a \vee b)$ in the 2SAT instance (where a and b are variables which may or may not be negated), place a directed edge from vertex $\overline{a}$ to vertex b and also a directed edge from vertex $\overline{b}$ to vertex a. For example, if we had a clause $(x_1 \vee \overline{x_3})$ then $a = x_1$ and $b = \overline{x_3}$. Therefore, we would place a directed edge from vertex $\overline{x_1}$ to vertex $\overline{x_3}$ as well as a directed edge from vertex x_3 to vertex x_1. In this example, you should interpret the edge from vertex $\overline{x_1}$ to vertex $\overline{x_3}$ to mean "if $\overline{x_1}$ is true (that is, x_1 is false) then $\overline{x_3}$ must be true." Similarly, the edge from vertex x_3 to the vertex x_1 is interpreted as "if x_3 is true then x_1 must be true."

Construct this directed graph for the 2SAT instance in part (a). This graph should contain 4 vertices 6 edges. Notice that there is no path from vertex x_1 to vertex $\overline{x_1}$.

(d) Notice that in the graph you constructed there is a path from vertex $\overline{x_1}$ to vertex x_1. Clearly explain why *the existence of this path* implies that a satisfying assignment for this instance of 2SAT cannot have $\overline{x_1}$ be **true** (that is, x_1 cannot be **false**). Be very clear and precise about your reasoning here.

(e) Notice that there does *not* exist a path in this graph from vertex x_1 to vertex $\overline{x_1}$. Notice also that in your satisfying assignment for this instance, x_1 was set to **true**. Clearly explain why this graph tells us that x_1 should be assigned **true** if the instance has any hope of being satisfied.

(f) What does the graph tell you about the value that should be assigned to variable x_2? Explain.

(g) Now, construct the graph for the 2SAT problem in part (b). What property does this graph have that forces you to conclude that the 2SAT instance is not satisfiable? Be precise.

(h) Next, write down a conjecture that starts as follows: "A 2SAT instance is satisfiable if and only if the corresponding directed graph has the property that blah, blah, blah." Fill in the "blah, blah, blah" with mathematically precise language.

(i) Now, prove your conjecture. This is the main part of this problem (it's also worth most of the points!) and will take a few paragraphs to prove. Note that it is an "if and only if" proof, which means that there are two things to show. In this proof, the fact that two edges were introduced for each clause will be absolutely critical. You'll need to use this fact!

(j) Now, describe an algorithm to determine whether or not a 2SAT instance is satisfiable. What is the running time of your algorithm as a function of the number of variables and clauses? Explain the running time carefully. As long as the running time is polynomial in the number of variables and clauses, everything is fine!

4. **[Optional Bonus Problem! 20 Points] Linear Generalized 3SAT!** Now, let's consider 3SAT in which we are given n boolean variables, $x_1, \ldots, x_n$, and m clauses, where each clause is the disjunction (logical OR) of exactly 3 literals (where a literal is a variable or its negation). An instance of 3SAT is the conjunction (logical AND) of m clauses and looks something like this:

$$(x_1 \vee \overline{x_2} \vee x_4) \wedge (\overline{x_5} \vee x_2 \vee x_{42}) \wedge \ldots$$

The objective is to find an assignment of **true** or **false** to each variable such that the entire expression evaluates to **true**. You may always assume that no clause appears twice. (That would be silly!)

While 2SAT can be solved in polynomial time, 3SAT is NP-complete and thus there is strong evidence that it cannot be solved by a polynomial-time algorithm!

In this problem, we consider a special version of 3SAT called "Linear Generalized 3SAT" which you will show can be solved in polynomial time. **Amazing, but true!**

"Generalized" 3SAT simply means that the clauses can contain 1, 2, or 3 literals each. So, an instance of Generalized 3SAT might look something like:

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2) \wedge (x_3 \vee \overline{x_7}) \wedge \ldots$$

Generalized 3SAT is still NP-complete: There is no known algorithm that determines whether or not any given instance of the problem is satisfiable. (This is true simply because regular 3SAT is a special case of Generalized 3SAT.)

However, consider the version of Generalized 3SAT in which the indices of the variables in each clause differ by at most 7. For example, if x_3 is in some clause, then $\overline{x_{10}}$ could be in the same clause but x_{11} could not be in that clause because 3 and 11 differ by more than 7. This version of Generalized 3SAT is called "Linear Generalized 3SAT."

The number 7 here was chosen just to be concrete. Generalized Linear 3SAT could have been defined for any fixed constant instead of 7.

Give a divide-and-conquer algorithm for solving Linear Generalized 3SAT in polynomial time. The polynomial will be extremely large, but nonetheless it should run in time polynomial in n, that is $O(n^k)$ for some fixed constant k. (What about m? Given that the polynomial is permitted to be quite large, you will be able to write the running time as a function of n without m.) Prove that your algorithm is correct and derive its running time.