Problem Set 9 for CS 170

Formatting

Please use the following format for the top of the solution you turn in, with one line per item below (in the order shown below):

> <your username on cory.eecs>
> <your full name>
> `CS170, Spring 2003`
> `Homework #9`
> `Section` <your section number>
> `Partners:` <your list of partners>

(Remember to write your section number, not the name of your TA or the time of your section.) This will make it easier for us to sort and process your homeworks. Thank you!

Note

When asked for an algorithm you must give (1) a brief informal description of the algorithm, (2) a precise description using pseudo-code, (3) an informal argument for termination and correctness of the algorithm, and (4) an analysis of the running time of the algorithm. Be clear about what the input to the algorithm is, how you measure the size of the input, and what constitutes a "step" in your running-time analysis.

Problem 0. [Any questions?] (5 points)

What's the one thing you'd most like to see explained better in lecture or discussion sections? A one-line answer would be appreciated.

(Sometimes we botch the description of some concept, leaving people confused. Sometimes we omit things people would like to hear about. Sometimes the book is very confusing on some point. Here's your chance to tell us what those things were.)

Problem 1. [Card Game] (30 points)

Consider the following card game. The dealer lays out n cards face up and side-by-side in a line, so that both the player and the dealer know the face value of each card. The player and the dealer then take turns taking a card from either end (but only the ends, not anywhere in the middle) of the line of remaining cards, with the player going first. After n of these turns, all of the cards have been taken, and the player wins if the cards in her hand have a larger sum than the cards in the dealer's hand. The dealer wins if his sum is greater than or equal to the player's sum, and he always makes the best possible move he can.

Alice has found a soon-to-be-bankrupt casino that allows players to bet on this game *after* the cards have been dealt.

(a) Design an algorithm for Alice to use in determining when to bet on this game. It is given as input n and an array C where $C[i]$ contains the value of card i. It should output "Yes" if Alice can definitely win (assuming she makes the right moves) or "No" if it is possible for the dealer to win – Alice doesn't take chances.

(b) How would you modify your algorithm to output an optimal move for each possible intermediate situation? For example:

```
Cards  1..52 remain, take card 1
Cards  2..52 remain, take card 2
...
Cards  5..43 remain, take card 43
...
Cards 20..52 remain, take card 20
...
Cards 33..33 remain, take card 33
...
```

Problem 2. [Randomized 2-SAT] (35 points)

2-SAT (for 2-Satisfiability) is a (relatively easy) special case of the **NP**-complete problem SAT.

In satisfiability problems, you are given a boolean formula ϕ and you are asked to determine whether or not it has a satisfying assignment (some TRUE/FALSE setting of the variables in ϕ that makes the whole formula true) and return a satisfying assignment if it exists.

In 2-SAT, it is required that ϕ is in *2-conjunctive normal form* (2-CNF). This means that the input looks something like:

$$(\overline{x_4} \vee x_2) \wedge (x_3 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge \ldots \wedge (x_{501} \vee \overline{x_{24}})$$

That is, ϕ is a *conjunction* (and) of *clauses*, where each clause has two literals. A *literal* is an instance of a *variable* or its negation, a *disjunction* (or) of two literals makes a clause, and a conjunction of these clauses makes a 2-CNF formula. For example, in

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

x_1, x_2, and x_3 are the variables, $(\overline{x_1} \vee x_2)$ is a clause, and $\overline{x_1}$ and x_2 are literals in that clause.

A *satisfying assignment* A for a formula ϕ is a mapping $A : \{x_1, x_2, x_3\} \to \{\text{TRUE}, \text{FALSE}\}$ that makes the formula true. For example, if A is an assignment such that $A(x_1) = \text{FALSE}$ and $A(x_3) = \text{TRUE}$, it is a satisfying assignment for ϕ (notice x_2's value does not matter).

Finally, the *size* of a formula ϕ is the number of clauses. Note that a formula ϕ of size n has at most $2n$ variables.

Consider the following algorithm:

```
MC-2-SAT(φ):
1: for each variable x_i in φ do:
2:     A[i] ← FALSE
3: do k times:
4:     if φ is satisfied:
5:         return SATISFIABLE BY A
6:     randomly pick an unsatisfied clause (l_i ∨ l_j) from φ
7:     randomly pick v from {i, j}
8:     A[v] ← ¬A[v]
9: return UNSATISFIABLE
```

The value of k in line 3 will be determined in part (c) below.

(a) Can `MC-2-SAT` ever output UNSATISFIABLE when run on a formula that is in fact satisfiable?

(b) Can `MC-2-SAT` ever output SATISFIABLE when run on a formula that is in fact unsatisfiable?

(c) Show that there is a choice of k (as a function of n) so that $k = O(n^2)$ and that is sufficient to find a satisfying assignment (if one exists) in lines 3–8 with probability at least $1 - \frac{1}{2^{100}}$.

A USEFUL FACT: Suppose we have random variables X_i ($i \geq 0$) that take values in the range $\{0, \ldots, n\}$. Suppose they're defined such that $X_{i+1} = 1$ if $X_i = 0$, $X_{i+1} = n$ if $X_i = n$, and otherwise: $X_{i+1} = X_i + 1$ with probability $p_i \geq 1/2$, $X_i - 1$ with probability $1 - p_i$ (i.e., $\leq 1/2$)[1]. Let $T = \min\{k \mid X_k = n\}$. Note that T is a random variable. Then it turns out that the expected value of T is at most $(n-1)^2$. (You do not need to prove this.)

HINT: Let B be a satisfying assignment. Define X_i in terms of the difference between A and B at each iteration of the main loop.

HINT: Given only that $E[T] \leq (n-1)^2$, how big should k' be so that $\Pr[T > k'] \leq 1/2$? If those first k' steps are taken but still $X_{k'} < n$, what is the probability that $X_{2k'} < n$ (i.e., what is $\Pr[T > 2k' \mid T > k']$)?

(d) Taking into account your choice of k in part (c), what is the running time of `MC-2-SAT`(ϕ) as a function of n, the size of ϕ?

[1] Strictly speaking, this random ± 1 change must happen *independently* of the values of $X_0, \ldots, X_i$. However, you don't need to worry about independence for this problem.

Problem 3. [Deterministic 2-SAT] (30 points)

Refer to the definition of 2-SAT in Problem 2. Note that Problem 2 gives a randomized algorithm to solve 2-SAT.

For this problem, give a *deterministic* algorithm to solve 2-SAT. (*Deterministic* means that you cannot use any randomness in your algorithm.) Use the method for solving Horn formula that was given in class as a subroutine. You do not need to give a proof of correctness for your algorithm (but your algorithm had better work correctly, of course).

HINT (modified 4/11): For each variable x_i in the 2-CNF formula ϕ, introduce corresponding Horn variables x_i and X_i to represent x_i and $\overline{x_i}$. Rewrite each clause in the 2-CNF formula as an implication to get a Horn formula ψ (in terms of the x_i and X_i variables – remember, no complements in Horn formulas). If ϕ has a satisfying assignment that sets $x_i = \text{TRUE}$, what should *never* be implied when x_i is asserted true (i.e., when x_i is added by itself to the Horn formula ψ and a minimal truth assignment is deduced)? In other words, when $x_i = \text{TRUE}$ in some satisfying assignment for ϕ, what shouldn't be true of the least satisfying assignment for $\psi \wedge x_i$? And, what about $x_i = \text{FALSE}$?

Bonus Problem. [Optimal Traffic Enforcement] (0 points)

There is a $\Theta(n)$-time algorithm for the task stated in the second half of Problem 3 on the midterm. Find it.