Introduction to Algorithms **Problem Set 7**
CS 482 Spring 2000 **Due April 21, 2000**

(1) *Self-avoiding walks* are a basic object of study in the area of statistical physics; they can be defined as follows. Let $\mathcal{L}$ denote the set of all points in $\mathbf{R}^2$ with integer coordinates. A *self-avoiding walk* W of length n is a sequence of points $(p_1, p_2, \ldots, p_n) \in \mathcal{L}^n$ so that

(i) $p_1 = (0, 0)$. *(The walk starts the origin.)*

(ii) No two of the points are equal. *(The walk "avoids" itself.)*

(iii) For each $i = 1, 2, \ldots, n-1$, the points p_i and p_{i+1} are at distance 1 from each other. *(The walk moves between neighboring points in $\mathcal{L}$.)*

Self-avoiding walks (in both two and three dimensions) are used in physical chemistry as a simple geometric model for the possible conformations of long-chain polymer molecules. Such molecules can be viewed as a flexible chain of beads that flop around in solution, adopting different geometric layouts, and self-avoiding walks are a simple combinatorial abstraction for these layouts.

A famous unsolved problem in this area is the following. For a natural number $n \geq 1$, let $A(n)$ denote the number of distinct self-avoiding walks of length n. Note that we view walks as *sequences* of points rather than sets; so two walks can be distinct even if they pass through the same set of points, provided that they do so in different orders. (Formally, the walks $(p_1, p_2, \ldots, p_n)$ and $(q_1, q_2, \ldots, q_n)$ are distinct if there is some i $(1 \leq i \leq n)$ for which $p_i \neq q_i$.) See the figure below for an example. In polymer models based on self-avoiding walks, $A(n)$ is directly related to the *entropy* of a chain molecule, and so it appears in theories concering the rates of certain metabolic and organic synthesis reactions.

Despite its importance, no simple formula is known for the value $A(n)$. Indeed, no algorithm is known for computing $A(n)$ that runs in time polynomial in n.

(a) Show that $A(n) \geq 2^{n-1}$ for all natural numbers $n \geq 1$.

(b) Give an algorithm that takes a number n as input, and outputs $A(n)$ as a number in binary notation, using space (i.e. memory) that is polynomial in n.

(Thus the running time of your algorithm can be exponential, as long as its space usage is polynomial. Note also that "polynomial" here means "polynomial in n," not "polynomial in $\log n$". Indeed, by part (a), we see that it will take at least $n-1$ bits to write the value of $A(n)$, so clearly $n-1$ is a lower bound on the amount of space you need for producing a correct answer.)

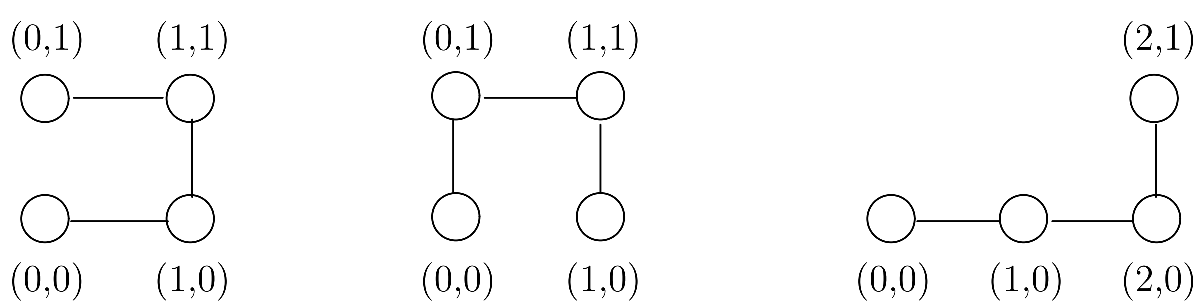

Figure 1: Three distinct self-avoiding walks of length 4. Note that although walks (a) and (b) involve the same set of points, they are considered different walks because they pass through them in a different order.

(2) *(Based on a problem proposed by Maverick Woo and Ryan Williams.)* Let's consider a special case of *Quantified 3-SAT* in which the underlying Boolean formula is *monotone* (in the sense of Problem Set 5). Specifically, let $\Phi(x_1, \ldots, x_n)$ be a Boolean formula of the form

$$C_1 \wedge C_2 \wedge \cdots \wedge C_k,$$

where each C_i is a disjunction of three terms. We say Φ is *monotone* if each term in each clause consists of a non-negated variable — i.e. each term is equal to x_i, for some i, rather than $\overline{x_i}$.

We define *Monotone QSAT* to be the decision problem

$$\exists x_1 \forall x_2 \cdots \exists x_{n-2} \forall x_{n-1} \exists x_n \Phi(x_1, \ldots, x_n)?$$

where the formula Φ is monotone.

Do one of the following two things: (a) prove that *Monotone QSAT* is PSPACE-complete; or (b) give an algorithm to solve arbitrary instances of *Monotone QSAT* that runs in time polynomial in n. (Note that in (b), the goal is polynomial *time*, not just polynomial space.)

(3) Recall that *Geography* on a graph, the game we looked at in lecture, is an abstraction of *Geography* on a set of words. The latter game is defined on a set of words $W = \{w_1, w_2, \ldots, w_n\}$ over an alphabet $\Sigma = \{a_1, a_2, \ldots, a_m\}$.

Player 1 moves first; and let's assume that Player 1 must start by choosing word w_1. Player 2 then must choose a word $w_{i_2} \in W$ which *begins* with the same letter on which w_1 ends. Player 1 then must choose a word $w_{i_3} \in W$ which begins with the same letter on which w_{i_2} ends. The game continues in this way — each player choosing a word that begins with the letter on which the previous word ends — with the additional requirement that each player must choose a word that has not yet been chosen. The first player without a valid move — i.e. all legal words have already been chosen — loses.

Suppose the set of words W has the following property: the characters in each w_i are distinct, and in *alphabetical order*. That is, each w_i has the form $a_{i_1} a_{i_2} \cdots a_{i_\ell}$, where $a_{i_1}, a_{i_2}, \ldots, a_{i_\ell} \in \Sigma$ and $i_1 < i_2 < \cdots < i_\ell$. We will also assume that each word has at least two characters.

Give a polynomial-time algorithm for this "alphabetical-order" special case of the problem that decides whether Player 1 can force a win.