Let $L$ be the language of binary strings whose length is a perfect square (0, 1, 4, 9, 16, 25, 36, ...). Give a detailed verbal description of a multi-tape Turing machine that decides $L$.

**Solution.** The idea is to consider every integer $n = 0, 1, 2, 3, \ldots$, in each case computing $n^2$ and comparing the result to $|w|$. We can stop as soon as $n^2 > |w|$. In more detail, the algorithm is as follows.

<table>
<thead>
<tr>
<th>Input: string $w \in {0, 1}^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>for $n = 0, 1, 2, 3, \ldots$ do</td>
</tr>
<tr>
<td>$N \leftarrow n^2$</td>
</tr>
<tr>
<td>if $</td>
</tr>
<tr>
<td>output “$</td>
</tr>
<tr>
<td>else if $</td>
</tr>
<tr>
<td>output “$</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

We will implement this algorithm using a Turing machine with three tapes: the input tape with the string $w$ written on it, and two additional tapes. At the start of an iteration, the second tape contains $0^n$ for the current value of $n$, and the third tape is blank. The Turing machine marks the first cell of $0^n$ with a dot and then copies $0^n$ to the third tape symbol by symbol (using an additional moving mark to implement the copying). The Turing machine then advances the dot to the next symbol of $0^n$ on the second tape and again copies $0^n$ to the third tape, appending it to the third tape’s current contents. The process continues until the dot moves beyond the last symbol of $0^n$ on the second tape. At that point, the third tape contains $0^{n^2}$, which can be compared symbol by symbol with $w$ to find out which one is longer. The computation halts immediately if $|w| = n^2$ or $|w| < n^2$, with the Turing machine entering $q_{\text{accept}}$ and $q_{\text{reject}}$, respectively. If $|w| > n^2$, the Turing machine increments $n$ (by appending a 0 to the second tape), erases the third tape, and starts a new iteration.
Draw the transition diagram for a Turing machine that decides the language of palindromes over the binary alphabet.

Solution.

Let $A$ be the language whose strings are binary encodings (with leading zeroes ignored) of multiples of 2016. Thus, $A$ contains $\varepsilon, 0, 00, 1111100000, 001111100000$ but not $1, 10, 11$. Describe an algorithm that takes as input a context-free grammar $G$ and determines whether $G$ generates some string in $A$.

Solution. We first build a DFA for $A$. The DFA works by converting the input on the fly to a decimal number and checking if that number is divisible by 2016. A literal implementation of this idea produces the infinite “automaton” ($\{0, 1, 2, \ldots \}, \{0, 1\}, \delta, 0, \{0, 2016, 4032, 6048, \ldots \}$), where $\delta(q, \sigma) = 2q + \sigma$. To get an actual DFA for $A$, recall that we only care about the remainder of the decimal number modulo 2016. Therefore, we can perform all intermediate computations modulo 2016. This gives the DFA ($\{0, 1, 2, \ldots, 2015\}, \{0, 1\}, \delta, 0, \{0\}$) for $A$, where $\delta(q, \sigma) = (2q + \sigma) \mod 2016$.

To check if a given context-free grammar $G$ generates a string in $A$, we start by converting $G$ to a PDA. We then combine this PDA with the DFA for $A$ via the Cartesian-product construction, obtaining a PDA for $L(G) \cap A$. Finally, we check whether this PDA for $L(G) \cap A$ accepts any strings, by converting it to a grammar and checking whether the grammar generates any strings. For each of these steps, we gave an algorithm in class.
4 Prove that the following problems concerning finite automata with alphabet \{0, 1\} are decidable:

(3 pts) a. on input a DFA \( D \), determine whether \( D \) accepts infinitely many palindromes;
(3 pts) b. on input a DFA \( D \), determine whether \( D \) accepts a pair of distinct strings \( u \) and \( v \) such that \( u \) is a prefix of \( v \).

Solution.

a. Rephrasing, we need to determine whether \( L(D) \cap P \) is infinite, where \( P \) is the context-free language of palindromes. Construct a PDA for \( L(D) \cap P \) by applying the Cartesian-product construction to a PDA for \( P \) and the given DFA \( D \). Then, convert the resulting PDA to a grammar and use the algorithm from class to check whether the grammar accepts infinitely many strings.

b. Use any graph search algorithm to identify the set \( F' \) of accept states of \( D \) that are reachable from the start state. Output “yes” if and only if there exists a pair of states \( q', q'' \in F \) (not necessarily distinct) with a path of nonzero length from \( q' \) to \( q'' \).
For each of the problems below, determine whether it is decidable, and prove your answers:

(3 pts) a. on input a Turing machine \( M \), determine whether there is a DFA with at most 2016 states that recognizes the same language as \( M \);

(3 pts) b. on input a Turing machine \( M \) and a string \( w \), determine whether \( M \) ever moves left while computing on \( w \);

(3 pts) c. on input a Turing machine \( M \), determine if there is an integer \( c \) (perhaps very, very large) such that \( M \) terminates within \( c \) steps on every input.

**Solution.**

a. Undecidable. We showed in class that some languages such as \( 0^*1^* \) have a DFA with at most 2016 states, whereas others such as \( \{0^n1^n : n \geq 0\} \) have no DFA at all, let alone a DFA with at most 2016 states. Thus, the language property of having a DFA with at most 2016 states is nontrivial. Rice’s theorem now implies that it is undecidable whether a given Turing machine recognizes a language with this property.

b. Decidable. Just run \( M \) on \( w \) for \(|w| + |Q| + 1\) steps, where \( Q \) is the set of states of \( M \). If \( M \) has moved left at any point, output “yes”; otherwise, output “no.” To see why we can cut the simulation off after this many steps, suppose that \( M \) goes on for \(|w| + |Q| + 1\) steps without ever moving left. Then \( M \) moves through every symbol of \( w \) and then through an additional \(|Q| + 1\) blank cells. As \( M \) moves through the \(|Q| + 1\) blanks, it goes through \(|Q| + 1\) states, which by the pigeonhole principle cannot all be distinct. Once \( M \) repeats a state in its onward march through the blanks, it gets stuck in a cycle and is forced to keep moving right forever.

c. Undecidable. For the sake of contradiction, assume that this problem can be solved by some Turing machine, which we will call STOPWATCH. Then the following algorithm solves the (undecidable!) halting problem:

```
Input: Turing machine \( T \), string \( x \)

if STOPWATCH
    (Input: string \( w \)
        Run \( T \) on \( x \) (ignoring \( w \))
    ) = “yes” then
    output “\( T \) halts on \( x \)”
else
    output “\( T \) does not halt on \( x \)”
```

For any input \( T, x \), the above algorithm constructs a Turing machine (on the fly, specially tailored to \( T \) and \( x \)) whose running time is bounded on all inputs by a constant \( c \) if and only if \( T \) halts on \( x \). Then, the algorithm runs STOPWATCH on the newly constructed machine to find out the answer.