Introduction to Algorithms **Solution Sketches for Problem Set 9**
CS 4820, Spring 2018

(1) *(10 points) Design a single-tape Turing machine to evaluate the "less than" relation on two natural numbers represented in binary. The input to the Turing machine is represented as a string over the alphabet* $\{0, 1, <, ?\}$*. The input string is always in the format* $a < b?$*, where each of* a *and* b *is a string of one or more binary digits, beginning with the digit 1. Your Turing machine should terminate in the "yes" state if the number represented by* a *in binary is strictly less than the number represented by* b *in binary, and it should terminate in the "no" state if the number represented by* a *in binary is greater than or equal to the number represented by* b *in binary. If the input violates the format requirements, any behavior is acceptable as long as your algorithm terminates.*

Example: *if the input is* $11 < 100?$ *the answer is "yes". If the input is* $0011 < 100?$ *your algorithm should terminate, but it is fine to terminate in any of the "yes", "no", or "halt" states because the first binary string does not begin with the digit 1.*

Your answer must include a description, in English (with accompanying notation as needed), of the alphabet, state set, and transition rule of your Turing machine, and a few sentences explaining how the algorithm operates and how the specified states and transitions implement those operations. You may choose to include a representation of the transition rule in tabular form (similar to those represented in Section 2 of the lecture notes) if you wish, but the tabular representation of the Turing machine is optional *whereas the human-interpretable explanation is* mandatory*.*

Your solution should include an analysis of the worst-case running time, as a function of the length of the input string. You do not need to write a proof of correctness.

Solution. If a and b represent two natural numbers in binary, then $a < b$ if and only if one of the following two conditions holds.

1. a has fewer digits than b,
2. a has the same number of digits as b, there is at least one place value in which their digits differ, and in the first such place value (reading both numbers from left to right) a has a 0 whereas b has a 1.

We will describe a Turing machine that is designed to test for these two possibilities, in two phases. For convenience, we'll enlarge the alphabet by adding two symbols $\bar{0}, \bar{1}$. These are used in the first phase when we're counting digits, to represent a 0 or 1 that has already been counted. The alphabet of the Turing machine is the set

$$\Sigma = \{\triangleright, \sqcup, <, ?, 0, 1, \bar{0}, \bar{1}\}.$$

The machine starts by moving right without overwriting the contents of the tape until it reaches the ? symbol. Phase 1 consists of a sequence of rounds, each of which begins with the read/write head situated on the ? symbol. In one such round, the machine moves left while changing the rightmost 0 or 1 in each of the strings a, b to $\bar{0}$ or $\bar{1}$, respectively. If there is a round in which one of the two strings a or b has a remaining 0 or 1 but the other string does not, then we have ascertained that one of the numbers has strictly fewer digits than the other, which means we can terminate and output "yes" if a has fewer digits, "no" if b has fewer digits. Otherwise, Phase 2 begins. Phase 2 also consists of a sequence of rounds; this time each round begins with the read/write head situated on the $\triangleright$ symbol. During one such round, the machine moves right until it finds the leftmost digit of a that is $\bar{0}$ or $\bar{1}$. It commits this digit to memory, changes it back to 0 or 1, respectively, and continues moving right until

it finds the leftmost digit of b that is $\bar{0}$ or $\bar{1}$. If this digit differs from the memorized one, we terminate and output "yes" if the memorized digit is 0, "no" if the memorized digit is 1. If the leftmost $\bar{0}$ or $\bar{1}$ in the string b is equal to the memorized digit, it changes to 0 or 1, respectively, and moves leftward back to the $\triangleright$ symbol to start the next round of phase 2.

To accomplish all of this, we use the following states.

1. s: starting state, also used for moving rightward in phase 1
2. ℓ_b: moving leftward in phase 1 through string b, searching for 0 or 1
3. $\ell_<$: moving leftward in phase 1 through string b, searching for $<$ symbol
4. ℓ_a: moving leftward in phase 1 through string a
5. ℓ_a^*: moving leftward in phase 1 through string a, having already exhausted all digits of b
6. r_a: moving rightward in phase 2 through string a, with no digit committed to memory yet
7. r_{a0}, r_{a1}: moving rightward in phase 2 through a memorizing 0 or 1
8. r_{b0}, r_{b1}: moving rightward in phase 2 through b memorizing 0 or 1
9. t: moving leftward in phase 2

In state s, the machine moves right without overwriting the contents of the tape until it reaches the ? symbol. Then it moves left and transitions into state ℓ_b.

In state ℓ_b the machine moves left without overwriting the contents of the tape until it reaches 0, 1, or $<$. If it reaches $<$ before 0 or 1, it means that all the digits of b have been "exhausted" (changed to $\bar{0}$ or $\bar{1}$ in previous rounds); the machine transitions to ℓ_a^* and moves left. Otherwise it changes 0 to $\bar{0}$ or 1 to $\bar{1}$, transitions to state $\ell_<$, and moves left.

In state $\ell_<$ the machine moves left until it encounters $<$, then transitions to ℓ_a and moves left.

In state ℓ_a the machine moves left until it encounters 0, 1, or $\triangleright$. If it reaches $\triangleright$ before 0 or 1, it means that b has more digits than a, so the machine transitions to "yes". Otherwise, it changes 0 to $\bar{0}$ or 1 to $\bar{1}$, transitions to state s, and moves right, which begins a process of moving rightward to end the round at the ? symbol.

In state ℓ_a^* the machine moves left until it encounters 0, 1, or $\triangleright$. If it reaches 0 or 1 it means that a has more digits than b, so the machine transtions to "no". Otherwise it reaches $\triangleright$, which means that a and b have the same number of digits, and all of those digits have been changed to $\bar{0}$ or $\bar{1}$: the machine moves right and transitons to state r_a, indicating the start of phase 2.

In state r_a the machine moves right until it encounters $\bar{0}$, $\bar{1}$, or $<$. If it reaches $<$ it means that all the digits of a and b have already been compared, and they were all equal, i.e. $a = b$, so the machine transitions to "no". Otherwise it changes $\bar{0}$ to 0, moves right, and transitions to r_{a0}, or it changes $\bar{1}$ to 1, moves right, and transitions to r_{a1}. In state r_{a0} or r_{a1} the machine moves right until it encounters $<$, then transitions from r_{a0} to r_{b0} or r_{a1} to r_{b1}, and moves right.

In state r_{b0} or r_{b1} the machine moves right until it encounters $\bar{0}$ or $\bar{1}$. If it reaches $\bar{0}$ in state r_{b1} it transitions to "no", if it reaches $\bar{1}$ in state r_{b0} it transitions to "yes", otherwise it changes $\bar{0}$ to 0 or $\bar{1}$ to 1, moves left, and transitions to t.

In state t the machine moves left without overwriting the contents of the tape until it encounters $\triangleright$ to end one round of phase 2.

The running time is $O(n^2)$: both phases consist of $O(n)$ rounds, each of which involves making one back-and-forth pass over the string in $O(n)$ time.

(2) *(10 points) Let us call a Turing machine M "speedy" if it has the following property: for every input string x, the computation of M on input x terminates, and it does so after at most $|x|+100$ steps.*

Design an algorithm to decide whether a Turing machine is speedy, given a description of the Turing machine. In more detail, the input to your algorithm consists only *of the description of a machine M (in particular the input to your algorithm does not specify any particular input string for M) and its output should be "yes" if M is speedy and "no" if M is not speedy.*

In your solution you should describe the algorithm — an ordinary algorithm description is fine, no need to explain how to implement the algorithm as a Turing machine. Prove that your algorithm always terminates, and prove that it always outputs the correct answer. However, you do not need to analyze your algorithm's running time, other than proving that it always terminates.

Solution. In a computation of a Turing machine, define a *pioneering transition* to be one that visits a location on the tape that was previously never visited. Since Turing machines start at the left edge of the tape and move only one step left or right at a time, the location of a pioneering transition is always exactly one step to the right of the location of the previous pioneering transition. If a Turing machine M is speedy, then after the pioneering transition at location k it never visits a location numbered lower than $k-100$. That is because if M visits location $k-101$ at some time t after the pioneering transition at k, and we let K denote the highest-numbered location vsited before time t, and we define y to be the string consisting of the first $\min\{K, |x|\}$ symbols of x, then the first t steps of the computation of M on input y are identical to the first t steps on input x. Since M needs to move from location 0 to location K and then back to location $k-101 \leq K-101$ during those t steps, we must have $t \geq K+101 > |y|+100$, so M is not speedy.

Guided by this observation, we will treat the process of searching for an input x that forces M to run for more than $|x|+100$ steps as a graph search process. Each node of the graph represents a pioneering transition, and is identified with an element of $K \times \Sigma^{101} \times \{0, 1, \ldots, 101\} \times \{0, 1\}$. The four parts of a node's identifier (q, z, g, f) have the following interpretations.

- q represents the state just after the pioneering transition.
- z is a string of length 101 representing the tape contents at the location of the pioneering transition and the 100 preceding locations. If the location of the pioneering transition is $k < 100$ then the last $k+1$ symbols of the string z represent the tape contents and the first $100-k$ symbols of z consist of the blank symbol $\sqcup$ repeated $100-k$ times.
- g is a natural number representing the number of "surplus steps" in the computation of M, at the time of the pioneering transition. Specifically, if the input string has length ℓ and the pioneering transition to location k takes place at time t then $g = t - \min\{k, \ell\}$.
- f is 1 if the location of the pioneering transition is to the right of the last character of the input string, and 0 otherwise.

To figure out the set of outgoing edges from the node (q, z, g, f), we simulate the machine M starting from state q at the rightmost symbol of z, right up until the first time one of the following events happens:

1. M terminates. Then node (q, z, g, f) has no outgoing edges.

2. The simulation runs for $101 - g$ steps. Then we make an edge from (q, z, g, f) to $(q, z, 101, 1)$.

3. In the simulation, M moves to a location to the right of the rightmost symbol in z. Then, letting h denote the number of steps in the simulation preceding this final step, and letting r denote the state to which M transitions in this final step, we make edges from $(q, z, g, 0)$ to $(r, z', g + h, 0)$ for every string z' whose first 100 symbols match the last 100 symbols of the tape contents at the time of the last step of the simulation, and whose final symbol is not $\sqcup$. We also make edges from $(q, z, g, 0)$ to $(r, z', g + h + 1, 1)$ and from $(q, z, g, 1)$ to $(r, z', g + h + 1, 1)$ where z' is obtained from z by deleting the first symbol and appending $\sqcup$. For convenience (i.e., to simplify the proof of correctness) label each such edge with the last symbol of z'.

We search this graph using DFS or BFS to determine if there is a path from the starting node $(s, \sqcup \cdots \sqcup \triangleright, 0, 0)$ to a node of the form $(q, z, 101, 1)$. If so, then M is not speedy. Otherwise, M is speedy.

To prove that the algorithm is correct, first consider the case that we find a path from $(s, \sqcup \cdots \sqcup \triangleright, 0, 0)$ to a node of the form $(q, z, 101, 1)$. Then the labels of the edges of this path — with trailing $\sqcup$ symbols removed — constitute an input string x such that M runs for $|x| + 101$ or more steps when presented with input x. The proof is by induction; the induction hypothesis is that if the k^{th} vertex on the path (numbering vertices starting from zero) is $(q', z', g, 0)$ and $g < 101$, then in the computation of M on input x, the pioneering transition at location k takes place at time $k + g$. The base case is trivial, and the induction step is justified by the way in which the edge set was defined.

Conversely, suppose that M is not speedy. Then there is a string x such that M runs for more than $|x| + 100$ steps when its input is x. Consider the computation of M on input x, and for each pioneering transition define the number of surplus steps, g, to be equal to $t - \min\{k, |x|\}$ where t is the number of the time step in which the pioneering transition takes place, and k is the the location number of the pioneering transition. If we take each of the pioneering transitions whose slack parameter is less than 101, and represent it as a node of the graph using the encoding explained above, then this sequence of nodes forms a path from the starting node to a node of the form (q', z', g, f), and this node has an edge to a node of the form $(q, z, 101, 1)$.