Dominik Machoń

Problem 1.1.

I'll use the following terms to name specific versions of TMs:

- *standard TM* – deterministic, single-tape TM,
- *modified TM* – TM modified as described in the problem statement.

Let $S = \langle Q_S, \Gamma_S, \delta_S, q_{0\,0}^S, q_A^S, q_R^S \rangle$ be a standard TM. I will construct a modified TM M satisfying $L(M) = L(S)$.

First, I will present an outline of M's behaviour:

1. M mimics the steps of S and on its tape it maintains the contents of S's tape (with possible auxiliary markings); the cell where S's head rests is additionaly marked.
2. Mimicking steps that move the head to the right and those that don't move it at all is simple because they can be performed by a modified TM in one go (as it naturally proceeds to the right).
3. Mimicking moves to the left requires finding the cell directly to the left of the head:
   1) We iterate over each cell that lies to the left of the head and in each iteration test if this cell is the one we're looking for (that is, the last one before the head).
   2) When we've found the right cell, we apply the transition, mark the cell as the new head and unmark the old one.
   3) In the end, we clean up all auxiliary flags that we used to perform the previous steps.

Now, let's proceed to the details of M's constitution. Let $M = \langle Q_M, \Gamma_M, \delta_M, q_0^M, q_A^M, q_R^M \rangle$.

- $Q_M = Q_S \times Modes \times TestResult$
  - $Modes$ consist of the following symbols:

?	searching for the head to perform next step
→	performing S's move to the right
←	performing S's move to the left
$test$	checking if a cell lies directly to the head's left
■	testing finished (the result is frozen)
$clean$	cleanin all flags used during testing

  - $TestResult$ is used during testing and consists of two symbols:

$pass$	the cell being tested is the last one so far
$fail$	the cell being tested is not the last one

- $\Gamma_M = (\Gamma_S \cup (\Gamma_S \times \{\hat{}\}) \times TestStatus$, where the first part is simply Γ_S possibly with additional marking ^ indicating the position of head; $TestStatus = \{T, NT\}$ is used during testing and shows whether the cell was already tested (T) or not (NT)
- $q_0^M = \langle q_0^S, ?, fail \rangle$, q_A^M, q_R^M are sets of those states in Q_M that contain q_A^S, q_A^S respectively

In order to limit the number of purely technical transitions I assume that:

- M's first step has number 2 (that is, it does nothing during step number 1);
- initially, there's the symbol $\langle \triangleright\ \hat{}, NT \rangle$ in the first cell of M's tape;
- blanks on M's tape have the form of $\langle \perp, NT \rangle$.

I'll divide S's steps into three categories: stationary (leaving head where it was), moves to the right and to the left and for each of these categories I'll provide a recipe for M for mimicking such steps.

Whenever I use a dot in place of some part of state or letter, I mean that I don't care for this particular part and the transition should be understood as a set of transitions for all possible substitutions of the dot (and of course this transition leaves this part unchanged).

Stationary steps

Mimic the following transition of S: $(x, p) \Longrightarrow (y, q, 0)$.

1) $(\langle x\text{^},\cdot\rangle, \langle p, ?, \cdot\rangle) \Longrightarrow (\langle y\text{^},\cdot\rangle, \langle q, ?, \cdot\rangle)$ — replace letter x with y and state p with q; searching for the next step may be started immediately (so flag ? remains).

Moves to the right

Mimic the following transition of S: $(x, p) \Longrightarrow (y, q, \rightarrow)$.

1) $(\langle x\text{^},\cdot\rangle, \langle p, ?, \cdot\rangle) \Longrightarrow (\langle x,\cdot\rangle, \langle q, \rightarrow, \cdot\rangle)$ — replace letter x with y, state p with q, remove head.
2) $(\langle x,\cdot\rangle, \langle \cdot, \rightarrow, \cdot\rangle) \Longrightarrow (\langle x\text{^},\cdot\rangle, \langle \cdot, ?, \cdot\rangle)$ — mark head in the cell to the right and set mode to ?. This step will clearly be performed on the correct cell (that is, on the previous one's right neighbour) because M with each lap increases its scope to the right before returning to ⊳.

Moves to the left

Mimic the following transition of S: $(x, p) \Longrightarrow (y, q, \leftarrow)$.

1. Initialize:
   1) $\langle x\text{^},\cdot\rangle, \langle \cdot, ?, \cdot\rangle \Longrightarrow \langle x\text{^},\cdot\rangle, \langle \cdot, \leftarrow, \cdot\rangle$ — for now, leave head where it was and initialize the procedure of mimicking the step (flag ←).
2. Find the element directly to the left of the head:
   1) $\langle x, NT\rangle, \langle \cdot, \leftarrow, \cdot\rangle \Longrightarrow \langle x, NT\rangle, \langle \cdot, test, L\rangle$ — begin testing cell if it hasn't already been tested.
   2) $\langle x,\cdot\rangle, \langle \cdot, test, L\rangle \Longrightarrow \langle x,\cdot\rangle, \langle \cdot, test, NL\rangle$ — if found another cell before reaching head then the cell being tested is definitely not the last one (set flag NL).
   3) $\langle x\text{^},\cdot\rangle, \langle \cdot, test, NL\rangle \Longrightarrow \langle x\text{^},\cdot\rangle, \langle \cdot, \blacksquare, NL\rangle$ — when head reached then the testing ends (flag ■).
   4) $\langle x\text{^},\cdot\rangle, \langle \cdot, test, L\rangle \Longrightarrow \langle x,\cdot\rangle, \langle \cdot, \blacksquare, L\rangle$ — same as 3), only here the test was positive (L), so we additionaly unmark head.
   5) $\langle x, NT\rangle, \langle \cdot, \blacksquare, NL\rangle \Longrightarrow \langle x, T\rangle, \langle \cdot, \leftarrow, NL\rangle$ — return where the test started; test was negative (NL), so set the cell as tested (flag T) and continue to the next cell (with flag ←, that is: goto 2.1).
   6) $\langle x, NT\rangle, \langle p, \blacksquare, L\rangle \Longrightarrow \langle y\text{^}, NT\rangle, \langle q, clean, L\rangle$ — return where the test started; test was positive (L), so we're in the cell where head should now be. We can finally perform the transition of S that has been our goal: replace letter x with y, state p with q and mark head; remove flag L and set flag $clean$ for cleaning up flags T.
3. Clean-up:
   1) $\langle x\text{^},\cdot\rangle, \langle \cdot, clean, \cdot\rangle \Longrightarrow \langle x\text{^},\cdot\rangle, \langle \cdot, ?, \cdot\rangle$ — finish clean-up when head reached and begin searching for the next transition.
   2) $\langle \cdot, T\rangle, \langle \cdot, clean, \cdot\rangle \Longrightarrow \langle \cdot, NT\rangle, \langle \cdot, clean, \cdot\rangle$ — remove flag T and set NT.

I've shown that modified TMs are at least as powerful as stadard TMs. To complete the proof that both models of TM recognize the same languages, it remains to show that standard TMs are at least as powerful as modified TMs: We can easily simulate a modified TM using stadard TM if we simply copy its moves, add the automatic → move and keeping the moves' count so that we can return the head to the beginning of the tape whenever moves' count reaches a square of a natural number.

■

Problem 1.2.

The main idea is to iterate over each pair of nodes, compute the length of the path between them (there's exactly one such path) and check if this length is equal to k. I assume that $k \geq 0$.

My program will use $O(\log(n) + \log(\log(k)))$ space, which falls within the L class.

First, let's observe that the root is always the node number 1 because it alone doesn't have a parent.

Before we go any further, let's look at some procedures that'll be used in terms of space complexity:

1) following operations on binary numbers: $+, -, \mathrm{mod}\ 2, \mathrm{div}\ 2, >, \leq, ==, \neq$ all take $O(\log(\mathrm{n}))$ space (counting in the space for results) provided that the arguments take $O(\log(n))$ space.
2) I introduce the function $ancestor(w, k)$ for each node w and those $k \in \mathbb{N}$ for which the definition makes sense: it returns the kth ancestor of w (a parent is 1st ancestor and so on). Given our representation of tree, this function can be computed using $O(\log(n))$ space (it only needs to remember one node number at a time).

The main iteration will look similar to this:
for $u := 1$ to n do
  for $v := u$ to n do $code$
REJECT
It will then only increase numbers u, v and test equality so it falls within our space limit.

Main iteration

First, we compute $LCA(u, v)$:

1. Compute $depth_u, depth_v$, that is the depths of u and v. This can be done in $O(\log(n))$ space as it's very similar to the $ancestor$ function. These depths are evidently $\leq n$.
2. if $depth_u > depth_v$ then $swap(u, v)$ $//swap$ may need additional $O(\log(n))$ space
   $diff := depth_v - depth_u$
3. $i_u := ancestor(u, diff),\ i_v := v$ $//i_u$ and i_v start at the same depth
   while $i_u \neq i_v$ do
     $i_u := ancestor(i_u, 1)$
     $i_v := ancestor(i_v, 1)$
   $lca := i_u$

Now comes the part where we have to find the path length between u and v.

It's quite easy if k is given in unary: we may keep a variable sum of size $O(\log(k))$ and accumulate the path length successively, making sure to stop when either or sum exceeds k or bit-length of any edge on the path exceeds bit-length of k.

Let's now assume k is given in binary. In this case, we'll aim at avoiding keeping the whole sum in memory.

To achieve this, we'll perform columnar addition of the edges' lengths and after each iteration (that'll produce one bit of result) check whether the produced bit is equal to the corresponding bit in k.

Columnar addition

Columnar addition of a bit column $b_1, b_2, \dots, b_m$ for $m \in \mathbb{N}^+$ with a carry c produces bit $b' = (B + c) \bmod 2$ and carry $c' = (B + c) \operatorname{div} 2$ (integer division), where $B = b_1 + \cdots + b_m$.
It can be shown inductively that $(*)$ $c' < m$:
Inductive step: Clearly, $B \leq m$, so if $c < m$, then $c' = (B + c) \operatorname{div} 2 < m$.
Base case: $c = 0 < m$.

We can see that the number of edges on the path $u - v$ is $\leq n$, and from this and $(*)$ we conclude that during our columnar addition, the carry will always be $\leq n$. The sum of all the bits and the carry will thus be $\leq 2n$ (as there'll be $\leq n$ bits). Therefore, the carry and the sum of all bits will be represented in $O(\log(n))$ space.

Now we can perform the columnar addition:

1. We create a variable $bitnum$ that in each iteratiorn will indicate which bit to look at in this iteration. In particular, the firt iteration should look at the LSBs of all visited egdes' lengths, so we initialize $bitnum$ to 0. It will increase by 1 in each iteration, as counting goes form LSB to MSB. The variable $bitnum$ takes $O(\log(\log(k)))$ space, because iterating stops when the end of the binary k is reached.
2. $sumbits \coloneqq 0; carry \coloneqq 0$ //the sum of bits and carry; both take $O(\log(n))$ space.

One iteration:

1. $sumbits \coloneqq carry$
2. Follow the path $u - lca$ and then $v - lca$ (both upwards) and for each visited egde add the right bit (indicated by $bitnum$) of its length to $sumbits$. If the right bit doesn't exist (that is, the edge's length was shorter), go to the next edge.
3. When all edges visited:
   $newbit \coloneqq sumbits \bmod 2$
   $carry \coloneqq sumbits \operatorname{div} 2$
   if the bit in k indicated by $bitnum$ is equal to $newbit$ then
      if this was the MSB of k then
         if there aren't any unvisited bits on the path $u - v$ then ACCEPT
         else discard the nodes u, v and proceed to the next pair
      $bitnum \coloneqq bitnum + 1$
      goto next iteration
   else discard the nodes u, v and proceed to the next pair
   Note: if all the edges' lengths has already been wholly processed in previous iterations, $newbit$ will be equal to 0 in all following iterations and it will finally clash with k's MSB.

■