COMS 4236 Homework 1

Mengqi Zong < *mz2326@columbia.edu* >

February 15, 2012

Problem 1

a. The transition table is shown in Table-1. And here is the brief explanation:

1. If the last bit is 0 (s_0): change the bit to 1 (s_1), scan the next input.
2. If the last bit is 1 (s_1), change the bit to 0, and worktape moves back 1 bit (p_1).
   - If the bit is 0 (p_1), change it to 1 (p_0). Worktape moves to the last bit.
   - If the bit is 1 (p_1), change it to 0. Worktape moves back 1 bit (p_1).
   - If the bit is $\triangleright$, work tape move to the first bit (p_f) and flip to 1 (p'_f). Then go to the first $\sqcup$ bit and set it to 0.

b. Using amortized analysis, we can show that given any input string with length n, its k^{th} ($0 \leq m \leq \log n$) digit will be flipped $1/2^k \cdot n$ times. And the total running time for the string is

$$\begin{aligned} T(n) &= \sum_{k=0}^{\log n} \frac{1}{2^k} \cdot n \cdot k \\ &= n \sum_{k=0}^{\log n} \frac{1}{2^k} \cdot k \\ &\leq n(1 + \sum_{k=0}^{\infty} \frac{1}{2^k} \cdot k) \\ &\leq n(1+2) \\ &= O(n) \end{aligned}$$

So the time complexity of the TM is $O(n)$.

$p \in K$	$\sigma_1 \in \Sigma$	$\sigma_2 \in \Sigma$	$\delta(p1, \sigma_1, \sigma_2)$
s_0	1	0	$(s_1, 1, \rightarrow, 1, -)$
s_0	$\sqcup$	0	$(h, \sqcup, -, 0, -)$
s_1	1	1	$(p_1, 1, -, 0, \leftarrow)$
s_1	$\sqcup$	1	$(h, \sqcup, -, 1, -)$
s_1	1	0	$(p_0, 1, -, 1, \rightarrow)$
p_0	1	0	$(p_0, 1, -, 0, \rightarrow)$
p_0	1	1	$(p_0, 1, -, 1, \rightarrow)$
p_0	1	$\sqcup$	$(p'_0, 1, -, \sqcup, \leftarrow)$
p'_0	1	0	$(s_0, 1, \rightarrow, 0, -)$
p'_0	1	1	$(s_1, 1, \rightarrow, 1, -)$
p_1	1	0	$(p_0, 1, -, 1, \rightarrow)$
p_1	1	1	$(p_1, 1, -, 0, \leftarrow)$
p_1	1	$\triangleright$	$(p_f, 1, -, \sqcup, \rightarrow)$
p_f	1	0	$(p'_f, 1, -, 1, \rightarrow)$
p'_f	1	0	$(p'_f, 1, -, 0, \rightarrow)$
p'_f	1	1	$(p'_f, 1, -, 1, \rightarrow)$
p'_f	1	$\sqcup$	$(s_0, 1, \rightarrow, 0, -)$

Table 1: Problem 1-a: 2-string Turing machine for length counting

Problem 2

a. Let the TM have two heads, h_1 and h_2. First, move head h_2 to the last symbol of the input. Second, begin to compare the symbol on the two heads: If they are the same, then keep going, until both of them reach the end (h_1 reaches the right-end and h_2 reaches the left-end); If they are not the same, half and output "No". If both heads reaches the end, then half and ouput "Yes". In this case, it takes $O(n)$ to run the algorithm.

b. Suppose a multihead Turing machine has l heads. Since it is possible for a tape to have multiple heads, then we need some extra space to store the position of each head.

For every work tape, since a work tape can at most have S(n) space, then it takes $O(\log S(n))$ to store each head's position. For the input tape, since it takes n space, then it takes $O(\log n)$ to store each head's position. As a result, it takes $O(\log S(n) + \log n)$ space to store the heads' positions.

In total, the multihead Turing machine takes $O(S(n)+\log S(n)+\log n)$ space. Simplify a little, we get $O(S(n)+\log n)$.

About time complexity, let $S^{'}(n) = O(S(n)+\log n)$. Since the number of configurations of a TM M with space $S(n)$ is at most $n \cdot c^{S(n)}$ for some constant c that depends on M, replace $S(n)$ with $S^{'}(n)$ we get the time complexity of a multihead Turing machine is $O(n \cdot c^{S(n)+\log n})$.

Problem 3

a. To solve this problem, I use a i-o Turing machine with 7 work tapes. Here are the 7 work tapes:

- t_1: store input a.
- t_2: store input b.
- t_3: store $length(a)$.
- t_4: store $length(b)$.
- t_5: store the counter (will explain later).
- t_6: store carry flag (Only 1 bit, 0 means no carry and 1 means there is a carry).
- t_7: store current result with leading zeros.

Here is the algorithm:

1. Count the length of a and b, store them respectively on t_3 and t_4. Also check the input format, if the format is wrong, then half and print # on the output.
2. Store a on t_1 and store b on t_2. If $length(a) \neq length(b)$, then put leading zeros to the respective tape which stores the input with shorter length.
3. Initialize t_5 the counter to be $max(length(a), length(b))$.
4. Initialize t_6 the carry flag to be 0.

5. Since the output can take at most $max(leng(a), length(b)) + 1$ space, initialize t_7 with $max(leng(a), length(b)) + 1$ zeros.

6. With each head of t_1, t_2 and t_7 point at their last bit (the least significant bit), now let's start the addition: according to the different input of current bit of a, b and the carry flag t_6, set the bit of t_7 and the also the carry flag t_6. Then counter t_5 decrement by 1, and the heads of t_1, t_2, and t_7 move backwards by one bit.

7. Repeat the former step until counter t_5 reaches 0. And if the carry flag is 1, then t_7 move backwards by 1 bit, and set the current bit to 1.

8. Print the output from t_7 to the output tape. Depends on different requirements, it is possible to eliminate the leading zeros in the output. Note that we have to consider the condition when output is 0, but that's not a problem.

b. Compare with a, we only need 6 tapes at this time. And instead of storing the exact input a and b, we just need to store the position of the current digit of input a and b. Here are the 6 tapes:

- t_1: store the current digit position of input a.
- t_2: store the current digit position of input b.
- t_3: store $length(a)$.
- t_4: store $length(b)$.
- t_5: store the counter.
- t_6: store carry flag (1 bit). And 0 means no carry, 1 means there is a carry.

Here is the algorithm:

1. Count the length of a and b, store them respectively on t_3 and t_4. Also check the input format, if the format is wrong, then half and print # on the output.

2. Store the position of the least significant bit of a on t_1 and store the position of the least significant bit of b on t_2.

3. Initialize t_5 the counter to be $max(length(a), length(b))$.

4. Initialize t_6 the carry flag to be 0.

5. Now let's start the addition from the least significant bit: according to the different input of current bit of a, b and the carry flag t_6, set the bit of the output and the also the carry flag t_6. Then counter t_5 decrement by 1, length t_3 and $t4$ decrement by 1, and so does the head position t_1 and t_2. If one of t_3 and t_4 reaches 0, then the output is only depends on the not-yet-end input and the carry flag. Also, each step the head of the output move forward by 1 bit.

6. Repeat the former step until counter t_5 reaches 0. And if the carry flag is 1, then output move forward 1 bit, and set the current bit to 1.

Since every work tape takes at most $O(\log n)$ space, this Turing machine has a space complexity $O(\log n)$.

c. Compared with part b, we still use a 6-tape Turing machine. Here are the 6 tapes.

- t_1: store the current digit position of input a.
- t_2: store the current digit position of input b.
- t_3: store $length(a)$.
- t_4: store $length(b)$.
- t_5: store the counter.
- t_6: store carry flag (1 bit). And 0 means no carry, 1 means there is a carry.

As to the algorithm, the key of calculating the sum from the most significant bit to the least significant bit is to get the correct carry flag. Here is the algorithm:

1. Count the length of a and b, store them respectively on t_3 and t_4. Also check the input format, if the format is wrong, then half and print # on the output.

2. Store the position of the most significant bit of a on t_1 and store the position of the most significant bit of b on t_2.

3. Initialize t_5 the counter to be $max(length(a), length(b)) + 1$. Here, the counter means the bit the addition is currently dealing with.

4. Initialize t_6 the carry flag to be 0.

5. Now let's start the addition from the most significant bit:

(a) Since counter t_5 is the bit the addition that is now dealing with, then we need to check t_3 and t_4 to see if input a or b has such a bit. If t_3 or t_4 is less than t_5, it means the input does not have such a bit, or its ${t_5}^{th}$ bit is 0.

(b) With two input's current bits, we still need to know the carry flag. Here is the basic idea of how to compute the carry flag:

Look at the bits one step away from the current bit of both inputs (If they don't have such bit, then this bit count as 0). If both bits are 1, then carry flag is 1. If both bits are 0, then carry flag is 0. But if one bit is 1 and other bit is 0, then we need to continue looking, that is, repeat this procedure whether both bits are 0 or 1, or reach the least significant bit of both input. If we reach both inputs' least significant bit and still 1 bit is 0 and the other bit 1, then the carry flag is 0.

(c) With the two current bits of a and b, also the carry flag, we can compute the current bit of the output. Note that we don't care about whether this will cause a carry or not since we calculate carry flag backwards. And at the first run, $t_5 = max(length(a), length(b)) + 1$, and both inputs' current bits is 0. So the first bit of the output only depends on the carry flag we compute.

6. Repeat the former step until counter t_5 reaches 0.

About the time complexity, since both cases b and c have space complexity $O(\log n)$, so the time complexity of both cases is $O(c^{\log n})$, which is polynomial.

Problem 4

a. From $u_1v_1 \in L(M)$, we can know that given the working configuration $\delta(M, u_1v_1, |u_1|)$, M can continue processing the rest of input – v_1, and finally accept the whole input. That is, with working configuration $\delta(M, u_1v_1, |u_1|)$, whether the input u_1v_1 can be accepted or not just depends on how M process the input v_1 (since the first part of the input u_1 is totally OK with M). In other words, $\delta(M, u_1v_1, |u_1|)$ represents the process result of u_1. And given this result, whether input can be accepted or not just depends on v_1.

As a result, if $u_1v_1 \in L(M), u_2v_2 \in L(M), \delta(M, u_1v_1, |u_1|) = \delta(M, u_2v_2, |u_2|)$, then $\delta(M, u_1v_2, |u_1|) = \delta(M, u_2v_1, |u_2|) = \delta(M, u_1v_1, |u_1|) = \delta(M, u_2v_2, |u_2|)$. This is because $\delta(M, u_1v_1, |u_1|) = \delta(M, u_2v_2, |u_2|)$ means both inputs' first part u_1 and u_2 are

OK to Turing machine M. And the their first parts' processing result are the same. So based on the result, we can process either v_1 or v_2 and makes the whole input accepted by M.

As to u_1v_2, we know that u_1 is totally OK with M. And for v_2, given $\delta(M, u_1v_2, |u_1|)$, it can be processed by M and u_1v_2 can be finally accepted by M. Same to u_2v_1. Then we get $u_1v_2 \in L(M), u_2v_1 \in L(M)$.

b. For reduced crossing sequence $\delta'(M, x0^m y, |x|)$, every working condition in this sequence means that the input head has to travel at least $n/3$:

- From the last cell of x to the first cell of y. The input head has to cross the whole 0^m area, which has a length of $n/3$.
- From the first cell of y to the last cell of x. The input head also has to cross the whole 0^m area, which has a length of $n/3$.

As a result, suppose all the running time $T(n)$ is used to do the crossing, the maximum possible crossing time is

$$\begin{aligned} m &= \frac{T(n)}{n/3} \\ &= \frac{3T(n)}{n} \end{aligned}$$

Note that in the analysis I don't count the time to process the whole input sequence, since this just at most affects the maximum crossing time by a constant 3.

About the upper bound, we already know from the class that the number of configurations is $c^{S(n)}$ (without considering the input tape). Since there are $3T(n)/n$ configurations in the reduced crossing sequence, so the upper bound on the number of possible distinct reduced crossing sequence is $(c^{S(n)})^{\frac{3T(n)}{n}}$. That is, $c^{\frac{3S(n)T(n)}{n}}$.

c. Since $x_1 \neq x_2$, then one of the two conditions must happen:

1. $|x_1| \neq |x_2|$
2. Since the two palindromes are different, there must exist at least one cell in x_1 that is different from the corresponding cell in x_2.

If $|x_1| \neq |x_2$, it is obvious that they have different reduced crossing sequences. Now let's consider the condition that $|x_1| = |x_2|$. In this case, at least one cell in x_1 is different from the correspond cell in x_2. That decides that their reduced crossing sequences are different: because when TM is about to cross cell $|x_1|$, it has to store what is in the cell M is now dealing with of x_1 in order to compare with the cell in ${x_1}^R$. However, since the two different palindromes must have at least one cell is different, then the reduced crossing sequences are different.

d. Given n, there are c'^n possible palindromes. And note that every palindromes can be represented in the form of $x0^m x^R$ as mentioned in part b. And since different palindromes have different reduced crossing sequences, then the number of possible reduced sequences should be greater equal to the number of possible palindromes (note that the constant doesn't matter). As a result, we have

$$\begin{aligned} c^{\frac{3S(n)T(n)}{n}} &\geq c'^n \\ \frac{3S(n)T(n)}{n} &\geq a \cdot n \\ S(n) \cdot T(n) &\geq \frac{a}{3} \cdot n^2 \\ S(n) \cdot T(n) &= \Omega(n^2) \end{aligned}$$

Note that a is a constant. At last, we get $S(n) \cdot T(n) = \Omega(n^2)$.