Computational Complexity
Example Solutions

Henrique Campos Navas

2023-2024

Homework 1

1.1 for addition

Our Turing Machine M will be described by a tuple (Γ, Q, δ) and it will have an input tape, an output tape and a work tape. Besides the usual symbols $0, 1, \triangleright$ and $\square$ we will add a separation character # so we can encode our pairs of numbers simply as a sequence of bits, followed by # followed by another sequence of bits, with both sequence of bits translating to natural numbers in the standard way. As such, we define Γ as $\{0, 1, \triangleright, \square, \#\}$.

Now we want to describe our states. q_s, q_h will be the starting state and the halting state respectively. For adding the numbers, it is useful to start in the units place, so first we will read the input in reverse (note that addition is commutative), and for such we will move the input tape's head to the right until we find the first $\square$ character, and for this we will use a q_{mr} state (move right). Now we will read y and write it in the work tape in reverse until we find a # character and for such we will use a new state q_{rl} (read and move left). Then we will need to move the work tape head to the beginning again so we can start adding x to y, and we will use a state q_{ml} (move leaf). Afterwards we will start adding y and x, and we will need to use two new states: q_c and q_{nc} (for when we need to carry a 1 and for when we have no carry). After adding x and y in reverse order on the work tape, we will need to write it in the correct order on the output tape. To do this, we simply need to start writing it in the correct order after we are finished with adding y and x and for that we will use our last state, q_w (write). As such, $Q = \{q_s, q_h, q_{mr}, q_{rl}, q_{ml}, q_c, q_{nc}, q_w\}$.

As described above, we will have the following transition function δ :

Input Symbol	Work Symbol	State	Move Input Head	Move Work Head	Move Output Head	Write in Work Tape	Write in Output Tape	New State
$\triangleright$	$\triangleright$	q_s	R	S	S	$\triangleright$	$\triangleright$	q_{mr}
0	$\square$	q_{mr}	R	S	S	$\triangleright$	$\triangleright$	q_{mr}
1	$\square$	q_{mr}	R	S	S	$\triangleright$	$\triangleright$	q_{mr}
#	$\square$	q_{mr}	R	S	S	$\triangleright$	$\triangleright$	q_{mr}
$\square$	$\square$	q_{mr}	L	R	S	$\triangleright$	$\triangleright$	q_{rl}
0	$\square$	q_{rl}	L	R	S	0	$\triangleright$	q_{rl}
1	$\square$	q_{rl}	L	R	S	1	$\triangleright$	q_{rl}
#	$\square$	q_{rl}	S	L	S	$\square$	$\triangleright$	q_{ml}
#	0	q_{ml}	S	L	S	0	$\triangleright$	q_{ml}
#	1	q_{ml}	S	L	S	1	$\triangleright$	q_{ml}
#	$\triangleright$	q_{ml}	L	R	S	$\triangleright$	$\triangleright$	q_{nc}
0	0	q_{nc}	L	R	S	0	$\square$	q_{nc}
0	$\square$	q_{nc}	L	R	S	0	$\square$	q_{nc}
$\triangleright$	0	q_{nc}	S	R	S	0	$\square$	q_{nc}
$\triangleright$	$\square$	q_{nc}	S	L	S	0	$\square$	q_w
0	1	q_{nc}	L	R	S	1	$\square$	q_{nc}
$\triangleright$	1	q_{nc}	S	R	S	1	$\square$	q_{nc}
1	0	q_{nc}	L	R	S	1	$\square$	q_{nc}
1	$\square$	q_{nc}	L	R	S	1	$\square$	q_{nc}
1	1	q_{nc}	L	R	S	0	$\square$	q_c
0	0	q_c	L	R	S	1	$\square$	q_{nc}
0	$\square$	q_c	L	R	S	1	$\square$	q_{nc}
$\triangleright$	0	q_c	S	R	S	1	$\square$	q_{nc}
$\triangleright$	$\square$	q_c	S	S	S	1	$\square$	q_w
0	1	q_c	L	R	S	0	$\square$	q_c
$\triangleright$	1	q_c	L	R	S	0	$\square$	q_c
1	0	q_c	L	R	S	0	$\square$	q_c
1	$\square$	q_c	L	R	S	0	$\square$	q_c
1	1	q_c	L	R	S	1	$\square$	q_c
$\triangleright$	1	q_w	S	L	S	1	1	q_w
$\triangleright$	0	q_w	S	L	S	0	0	q_w
$\triangleright$	$\triangleright$	q_w	S	S	S	$\triangleright$	$\square$	q_h

Assume every other input tape symbol, work tape symbol and state tuple goes into the the halting state immediately. Note that we need to specify some scenarios for when the numbers have different lengths in which we treat $\triangleright, \square$ as 0's (but we refrain from moving the input head).

With this, our specification for a Turing machine $M = (\Gamma, Q, \delta)$ is complete.

1.9

Let $M = (\Gamma, Q, \delta)$ be a RAM Turing Machine that computes a Boolean function in $T(n)$ time. We will construct the Turing Machine $M' = (\Gamma, Q', \delta')$ that simulates M in $\mathbf{DTIME}(T(n)^2)$, with the same number of tapes, including one work tape A that will simulate the memory tape.

First, we copy the alphabet and all of the states and transitions which do not involve q_{access}. Then, we "split" q_{access} in the following manner. We create a state $q_{\text{access begin}}$, which has the same transitions into the state as q_{access}, and a state $q_{\text{access end}}$, which has the same transitions from the state as q_{access}.

Between these two states, M' will simulate the memory access function of a RAM TM in at most $\mathbf{DTIME}(T(n))$ operations, requiring some extra states to do it. Regardless of whether we have an R or a W, we can first read the number i to the left of this symbol, in some manner which depends on the way numbers are encoded, and move the tape A head to the corresponding place (for simplicity's sake, assume we first move it all the way to the left so we do not have to worry about its current position).

Example: If i is encoded in binary, with a symbol different from 0 or 1 to its left, we can, for example, do the following: First we create four new symbols, $0'$, $0''$, $1'$ and $1''$. Then, whenever we read a bit, we will want to move the corresponding power of 2 in the memory tape if it is a 1. We can do this in a recursive-inductive manner. If there are no bits to its right, then we move only a single step, otherwise, we change 0 to $0'$ or 1 to $1'$ and move to the right and perform the same corresponding amount of movements, then we go back to the original bit and we will see an $0'$ and a $1'$, which we will change to either $0''$ or $1''$ and which will tell us to perform the previous steps again, thus doing twice the amount of steps as before, when we find a $0''$ or a $1''$ we know we can finally move to the left. If the original bit is a 1, we actually move A's head to the right, if it is 0 we do not, but we still perform some "phantom" movements so the recursion still works as intended.

Now, we need to move the work tape head back to the symbol, which we can do easily by simply going right until we find an R or a W (and fixing alterations done above if necessary). After reading this symbol, we either write the symbol $A[i]$ in the work tape, to the left of R, or we change the symbol $A[i]$ to the one to the left of W, with both these operations requiring only a couple of new auxiliary states.

This M' will run in the same manner as M, except when it finds q_{access}, which happens at most $T(n)$ times. Each time it finds q_{access}, it will need to simulate memory access, which takes at most the size of the used memory tape[1], which in $T(n)$ operations is at most $T(n)$, times some constant. Thus, M' simulates M in $\mathbf{DTIME}(T(n)^2)$.

1.14

We will assume that our graph is received as an adjacency matrix in the following format: #, followed by n bits which indicate if the first node is connect to other nodes, followed by #, followed by n bits which indicate if the second node is connect to other nodes, ..., #, followed by n bits which indicate if the n-th node is connect to other nodes, followed by #.

We also note that these examples are not optimal.

(a)

We do a **BFS** algorithm. On a work tape, we mark the first space with an A (accessible) and the following $n-1$ spaces with an N (not yet accessible). For each A on this tape, we go to the associated segment of the adjacency matrix and for each 1 we change N on the work tape to an A. Afterwards, we change this A to a V (visited). At some point, we will no longer have any A's. At this point, we see if we have any N. If so, we output a 0. Otherwise, we output a 1. This algorithm clearly takes polynomial time, as each node is visited at most once and each step takes $O(n^2)$ time[2].

Note: some of these steps, such as moving to the correct segment of the input tape, an extra work tape focused on performing this operation can make things rather simple, by helping us

[1]We assume we do not have huge gaps in the memory tape, otherwise by having a RAM TM that writes a representation for an exponentially big address and uses that address without using the ones before, this solution would not suffice. We would need to order the addresses in some manner or change our encodings.

[2]Changing the N's to A' can be done in linear time, the most time consuming part is moving the input header to the correct segment.

count how much we have to move.

(b)

A simple way of doing this is to check every triplet. For doing so, we will have 3 work tapes, which used space n, and check every triplet by first iterating across every space in the first tape, only afterwards moving the second head one space to the right and resetting the first one, and only after moving the second head n times do we move the third one. We only stop once every head is at the last position. We can use these tapes to simply count to which segment do we need to go and which position of the segment do we want to check. If at any point we find a triangle, we output 0, if we check every triplet with no issues we output a 1. Each check takes $O(n^2)$ time, and we perform n^3 checks, so this problem is in $\mathbf{DTIME}(n^5)$ and therefore in $\mathbf{P}$.

3SUM is in P

We check every triplet in the same manner as the previous exercise. In another tape, we add the numbers together using a similar algorithm to the one in exercise **1.1**[3]. If the the sum is 0 at any point, we halt and output 1, if we check every triplet without any success, we output 0. We need to check n^3 triplets and each check takes $O(n\log(n))$ to arrive at the numbers in the input tape and $O(\log(n))$ to perform the sums, for a total of $O(n^4\log(n))$ time.

3SUM in time $O(n^2\log(n))$ by a RAM TM

For each number x_i, we write 1 on the memory tape A in $A[x_i]$ (we can perform the trick of adding n^3 to every number so nothing is negative[4]). This can be done in by simply copying the input in time $O(n\log n)$ and then writing $W, 1$ after each number. Now, for each pair (i, j), we check if $A[-x_i - x_j]$ is 1. If it is, then we output 1, and if we check every pair without success we output 0. In order to do this in $O(n^2\log(n))$ we need to be careful, as each check needs to be done in $O(\log(n))$ time. First, we write x_i in another tape and copy it to an extra tape. Then, we for each candidate x_j we check them by order so we never need to go back, and we can start with x_i. We add it to x_i in $O(\log(n))$ and if we don't get a successful 0 sum with can simply copy back the x_i from the extra tape instead of needing to go look for it in the input tape. Thus, we perform $O(n\log(n))$ to set up each x_i, as we need to go there in time $O(n\log(n))$, after finishing the tests with the previous x_{i-1}, copy it twice in time $O(\log(n))$ and then perform $O(\log(n))$ steps to add the numbers together and so we can check the memory tape and then we move the head to the next number, also in time $O(\log(n))$. Thus, this algorithm runs in time $(n^2\log(n))$ as we hoped.

[3]Here we have negative numbers which we did not account for. A simple way of overcoming this in this case is by adding n^3 to every number and then instead of checking if the sum is 0 we check if it is $3n^3$.

[4]See footnote 3.