Computational Complexity **(Due: Flexible)**

Assignment #1 (Arora Barak Ch - 1)

Instructor: Shreesh Maharaj *Name:* Jimut Bahan Pal

Q. 1. Show that for every time-constructible T: N $\rightarrow$ N, if L $\in$ **DTIME(T(n))**, then there is an oblivious TM that decides L in time $O(T(n) \log T(n))$

Ans. 1 Let $f : \{0,1\}^* \rightarrow \{0,1\}$ be length-respecting and let $t : N \rightarrow N$ be time constructible, $t(n) \geq n$. If $f \in$ **DTIME(T(n))** then there is an oblivious Turing Machine computing f in time $O(T(n) \log T(n))$.

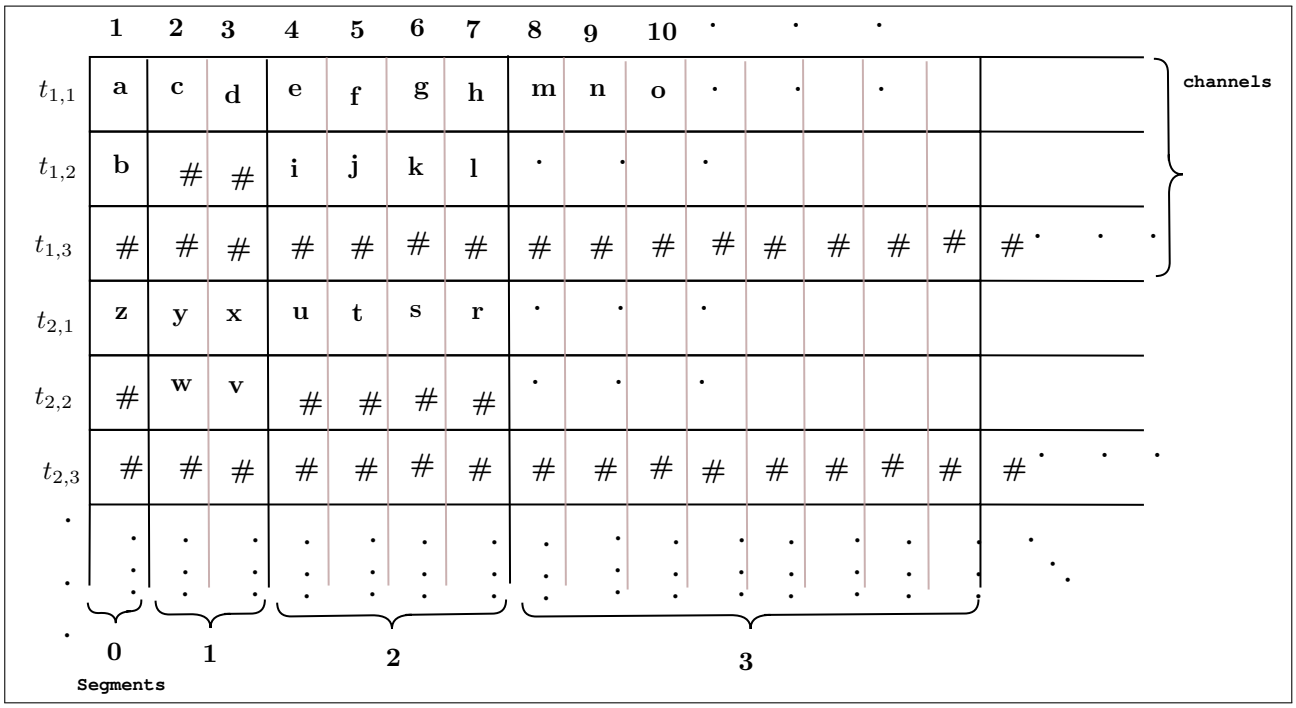

Figure 1: The structure of the Master tape of Oblivious Turing Machine M''.

Let a Turing Machine M compute a function f in time t. We simulate M by a machine M' which satisfy the property of obliviousness [1, 2, 3] on the moves of the head of input and output tapes. This is achieved by considering that M' has three more work tapes than M [4]. M' starts its execution by copying the input to an extra work tape, and then it simulates M for exactly $t(n)$ steps, where n is the length of the input. M' uses one of its work tapes instead of M's output tape, and finally copies the output from this particular work tape to output tape later. The heads on the input and output tape during the simulation move in an oblivious way. Now, let us simulate the machine M' by another machine M'' which is oblivious for all heads. Here we will use pushdown stores (one way infinite tape), where the head is on the rightmost non-blank symbol which is used in two ways:

- Move one cell to the right and print the symbol a; we denote this operation as **push(a)**
- Print a blank symbol # and move one cell left, and denote this by **pop**

The cell of every tape of two one-way infinite tapes of M'' are numbered as 1,2,3,... The first tape of M'' is called the *mastertape* and it is divided into channels. For each of the pushdown stores there will be one *channel* and for every channel there will be three *tracks*. Each cell can hold either # or one symbol of M' per track.

If we consider M' as a k-tape machine, then M'''s first work tape consists of $2k$ channels and $6k$ tracks. The tracks can be represented as $t_{1,1}, t_{1,2}, t_{1,3}, t_{2,1}, ...t_{2k,3}$ which means if M' has alphabet $\sum$ then M'' has alphabet in $\sum \cup \{\#\}^{6k}$. The master tape is divided into sequences of segments where segment number i will consist of tape cells 2^i up to $2^{i+1} - 1$ for all $i \geq 0$. A *block* is the portion of the track that lies within one segment. When we perform the simulation, it will be such that a block in segment i will consist of either 2^i symbols from $\sum$ (we call the block to be filled) or 2^i symbols from # (we call the block to be empty). The contents of the channel is the concatenation of the filled blocks, first by segment number and then by track number. Here, in Figure 1 we see that channel 1 holds the string $abcdefghi...$ and channel 2 holds the string $zyxwvuts..$

The contents of pushdown store of M' are the contents of a channel in reversed order. Here in the above figure, a and z will be the rightmost symbols where the pushdown heads will be found. Here, we will describe how to simulate a single pushdown by its corresponding channel, where all channels will be handled in parallel in the same way. In the simulation, each segment will have either 1 or 2 filled blocks, then it may be called as *clean*. The algorithm for clean can be found in Algorithm 1. The simulation's algorithm can be found from Algorithm 2. At the beginning of the simulation we can assume that track one is filled entirely with blanks, keeping all other tracks empty, which makes all segment clean. Process $simulate(k)$ [2] takes 2^k steps of M'. It is clear that during simulation, each block is either filled or empty. The following property holds by induction for each execution,

- The head of M'' doesnot move beyond segment number n.
- At the end segments 0,1,2, ..., n-1 will be clean.
- The number of filled blocks in segment n will change atmost 1.

Procedure cleans segment k, but it might possibly leave segment k+1 unclean. The time needed for $simulate(n)$ is at most twice the time for $simulate(n-1)$ plus $O(2^n)$ for the *clean* calls, which is together $O(n2^n)$. The sequence of calls to *simulate* with growing arguments will stop at the least with the call $simulate(\lceil \log t(|x|) \rceil)$. Thus the overall time is $O(t. \log t)$.

Algorithm 1: clean

Input: $k \leftarrow$ of type segment
Result: clean the k-th segment
if *k-th segment has 3 filled blocks* **then**
  concatenate blocks in track 2 and 3 to segment k+1;
end
if *segment k has no filled blocks* **then**
  divide a block from segment k+1 into two and move the results into tracks 1 and 2;
end

Algorithm 2: simulate

Input: n$\leftarrow$ of type integer
Result: simulate 2^n steps of M'
if *n = 0 then simulate 1 step of M by updating cell 1 where* **then**
  push(a) = {
  track3 = track2;
  track2 = track1;
  track1 = a;
  }
  pop() ={
  track1 = track2;
  track2 = track3;
  track3 = #;
  }
else
  simulate(n-1);
  clean(n-1);
  simulate(n-1);
  clean(n-1);
end

Q. 2. Prove that the following languages/decision problems on graphs are in **P**. (You may pick either the adjacency matrix or adjacency list representation for graphs; it will not make a difference. Can you see why?)

1. CONNECTED — The set of all connected graphs. That is, G $\in$ CONNECTED if every pair of vertices u, v in G are connected by a path.
2. TRIANGLEFREE — The set of all graphs that do not contain a triangle (i.e., a triplet u, v, w of connected distinct vertices).
3. BIPARTITE — The set of all bipartite graphs. That is, G $\in$ BIPARTITE if the vertices of G can be partitioned to two sets A, B such that all edges in G are from a vertex in A to a vertex in B (there is no edge between two members of A or two members of B)
4. TREE — The set of all trees. A graph is a tree if it is connected and contains no cycles. Equivalently, a graph G is a tree if every two distinct vertices u, v in G are connected by exactly one simple path (a path is simple if it has no repeated vertices).

References

[1] Steven Homer and Alan L. Selman. *Computability and Complexity Theory*. Springer Publishing Company, Incorporated, 2nd edition, 2011.

[2] F. C. Hennie and Richard Edwin Stearns. Two-tape simulation of multitape turing machines. *J. ACM*, 13(4):533–546, 1966.

[3] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009.

[4] Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999.