

Homework 10: Fake

*Due: Never**Vinod Vaikuntanathan*

Readings: Sipser, Sections 8.1-8.4

Problem 1: (Sipser exercise 8.1) Show that for any function $f : N \rightarrow N$, where $f(n) \geq n$, the space complexity class $\text{SPACE}(f(n))$ is the same whether you define the class by using the single-tape TM model or the two tape read-only TM model.

Solution 1: First, we can simulate a $\text{SPACE}(f(n))$, for $f(n) \geq n$, single-tape TM on a $\text{SPACE}(f(n))$ two tape read-only TM as follows. Step one: scan the read-only tape, copying its contents to the work tape. Step two: simulate the remainder of the computation, treating the read/write work-tape as the input tape of a “single-tape” TM. We can copy the contents of the counter, using only $\log n$ space to keep track of our position in the input. We can write all of the input on our work-tape, since $f(n) \geq n$.

Secondly, we can simulate a $\text{SPACE}(f(n))$, for $f(n) \geq n$, two tape read-only TM on a $\text{SPACE}(f(n))$ single-tape TM by refraining from writing over the first n input symbols, using the remainder of the tape for our work area. Since this only adds n amount of space used, and we were already using at least n space, this increases our space by at most a constant.

Problem 2: The japanese game *go-moku* is played by two players, “X” and “O”, on a 19×19 grid. Players take turns placing markers, and the first player to achieve 5 of his markers consecutively in a row, column or diagonal, is the winner. Consider this game generalized to an $n \times n$ board. Let

$$GM = \{ \langle P \rangle \mid P \text{ is a position in generalized go-moku, where player “X” has a winning strategy} \}.$$

By a *position*, we mean a board with markers placed on it, such as may occur in the middle of a play of the game. Show that $GM \in \text{PSPACE}$.

Solution 2: (borrowed from Spring 1999) First let’s assume that P is written as a grid of X,O and empty, so the length of the input is $O(n^2)$. Let’s define a recursive algorithm to solve $GM(P)$, which accepts if there is a winning strategy for player X starting at position P:

GM(P)

(1) For all spaces i in position P without markers on them, (potential X moves)

1. Put an X marker on space i , thus changing the position to P’. If there are now 5 X’s in a row, **accept**, this is obviously a good move. If the board is now full, and no one has won, **reject**.
2. Otherwise, for all spaces j in position P’ without markers on them, (potential O moves)
 - (a) Put an O on space j , thus changing the position to P”. If there are now 5 O’s in a row, or the board is full and no one has won, loop to the next i (goto step (1)); putting an X on i is obviously a bad move.
 - (b) Otherwise, run **GM(P’)**. If it accepts, loop to the next j (goto step (b)). If it rejects, loop to the next i (goto step (1)).
3. If all j cause **GM(P’)** to accept, i is a good X move, since it covers all possible O moves, so **accept**.

(2) If no i in step (1) causes an accept, reject, there are no good moves from this position, so **reject**.

We can safely loop through all moves i, j at each step, since we can just reuse the space. Our recursion is only $O(n^2)$ deep, since there are at most n^2 moves during a game, and we just need to store configurations P', P'' on the stack at each level, which takes $O(n^2)$ space. So, the total space needed is $O(n^4)$, which is polynomial in the input length, since we assumed the input had length $O(n^2)$.

Why not just a 19x19 board? Well, that would mean there were 19^2 spaces on the board, and 3^{19^2} possible configurations. This is a big big number, but asymptotically, it's tiny. So, we just search through all possible move sequences and see if X can win. In the world of complexity theory, any expression without a variable is just a small constant, even if it's bigger than the number of atoms in the universe, which that number most certainly is. This is why we like to generalize things to variable lengths, so we can fit them into our complexity classes.

If we write the configuration down as n , followed by a list of marker locations, we run the risk of having the input be too small; for example, let's say that no markers had been placed, so we want to know who has a winning strategy from the beginning. Our input has length $\log n$, since it takes that much space to write down the number n . Our algorithm might still take $O(n^4)$ space, but now that's exponential in our input length. So, we need to impose an appropriate encoding to ensure that we stay within PSPACE in all cases.

Problem 3: The proof of Savitch's theorem, in Section 8.2, describes in general how one can simulate any $f(n)$ -space-bounded nondeterministic Turing machine N with an $f^2(n)$ -space-bounded deterministic Turing machine M. The key is a recursive computation of the CANYIELD relation, which reuses space.

Give a good upper bound on the *running time* of M on input w .

Solution 3: Answering this "tightly" is beyond the scope of this class. For those that are interested, we can set up a recurrence. Let $T(t)$ be the maximum running time of CANYIELD on input t (we ignore c_1 and c_2). Then $T(t) = 2c^{p(n)}T(t/2) + O(1)$. Here c is the (constant) number of possible symbols in a configuration and $p(n)$ is a polynomial bounding the length of the longest configuration. The constant term $O(1)$ comes from step 1 of CANYIELD where we perform some base case checks.

Solving for this recurrence (using the Master method), gives us $T(t) = O(t^{\log_2 2c^{p(n)}}) = O(t^{p(n)})$. The value of t is set to be $2^{O(f(n))}$ in Savitch's proof, substituting this in we have an upperbound on the running time of $O(2^{O(f(n))p(n)})$. Obviously this is huge amount of time; but it still takes only polynomial space.

Problem 4: (Sipser 8.20) An undirected graph is *bipartite* if its nodes may be divided into two sets so that all edges go from a node in one set to a node in the other set. Show that a graph is bipartite if and only if it does not contain a cycle that has an odd number of nodes. Let

$$\text{BIPARTITE} = \{ \langle G \rangle \mid G \text{ is a bipartite graph} \}.$$

Show that $\text{BIPARTITE} \in NL$.

Solution 4: Suppose a graph is bipartite with partitions V_1 and V_2 . Then, any cycle in G has to alternate between the partitions, and make a whole number of such alternations (because the cycle has to reach the start vertex). Thus, any cycle will be of even length. Conversely, if a graph does not contain an odd-length cycle, then one can arrange the vertices into two partitions V_1 and V_2 as follows:

- Start from an arbitrary vertex $v \in V$ and add it to the first partition V_1 .
- Add all the neighbours of v to the second partition.

- Recurse this procedure on the neighbours of v .

Note that all the nodes that are at an even distance from v are put in V_1 , and all the nodes at an odd distance from v are put in V_2 , by this procedure. This procedure does not try to assign the same vertex u to both V_1 and V_2 . If that happens, it means there is a path of odd length from v to u , and another one of even length from v to u . This implies the existence of an odd cycle in G , a contradiction.

We will show that $\text{BIPARTITE} \in \text{coNL}$. Then, by invoking the Immerman-Szelepcsényi theorem (Section 8.6 in the old edition of Sipser) that says $\text{NL} = \text{coNL}$, we prove that $\text{BIPARTITE} \in \text{NL}$.

The coNL algorithm for BIPARTITE does the following:

- Guess a node v . Set $\text{temp} \leftarrow v$. Set a bit $b \leftarrow 0$.
- For $i = 1$ to $|V|$, do:
 - Guess a neighbour temp' of temp .
 - Check if $\text{temp}' = v$.
 - If it is, and if $b = 0$, then stop and accept.
 - Else, set $b \leftarrow \bar{b}$, $\text{temp} \leftarrow \text{temp}'$ and move to the next iteration of the for loop.
- If you came out of the for loop, reject.

If there is an odd cycle in the graph, there is a sequence of guesses such that this procedure accepts. On the other hand, if there are only even cycles in G , then, whenever $\text{temp}' = v$, the bit b will be set to 1, and there will not be any accepting path. Moreover, the total space we need to keep track is the equivalent of storing $\text{temp}, \text{temp}', v$ and b , which is $O(\log n)$.