CMSC 303 Introduction to Theory of Computation, VCU
Spring 2017, Course Mini-Project
Due: Tuesday, May 2, 2017 in class
Steven Hernandez

Total marks: 26 marks + 3 marks bonus for typing your solutions in LaTeX.

One of the goals of this course is to train you to be able to expand your education independently, given the tools you've learned here. For this reason, your mini-project will consist of the following task. First, please independently read Sections 8.1 and 8.2 on Space Complexity. This will introduce you to the complexity class PSPACE and Savitch's theorem. Next, answer the following questions.

1. [2 marks] In words, which class of problems does PSPACE characterize?

   PSPACE is meant to characterize all problems which can be completed on a TM using only $O(f(n))$ cells of the TM where $f(n)$ is a function that can of polynomial time. Savitchs theorem shows that NP $\subseteq$ PSPACE.

2. [4 marks] Show that PSPACE is closed under the concatenation and star operations.

   First for concatenation:

   Suppose we have two TMs which run in $O(f(n)) \subseteq$ PSPACE, labeled A and B for input(s) n. We construct a second TM C which simulates both A and B. Our goal is to run C such that $C \subseteq$ PSPACE. We can input into C the input for A and B separated by a delimiter $d \notin \Sigma$. For this example we will simply use a comma ,

   $$\begin{aligned} C = \text{"On input } & m \\ & \text{Split } m \text{ on , to retrieve inputs } a \text{ and } b \\ & \text{Run } A \text{ on } a \text{ in PSPACE} \\ & \text{Reset the tape by clearing the content.} \\ & \text{Return pointer back to the first cell} \\ & \text{Run } B \text{ on } b \text{ in PSPACE overwriting anything written by } A\text{"} \end{aligned} \tag{1}$$

   Because we move the pointer back to the first cell and overwrite any values, we are back to a clear TM, thus if B ran in PSPACE normally, it would run in PSPACE this time as well, because there are no remnants from A. The amount of space used would be the greater of the space used by A and B.

   Because of this, showing * is trivial because we can run as many TMs (that are $\subseteq$ PSPACE) on this machine as we wanted. Because each machine runs overtop of the same cells as the previous machine, completely unaware of the past.

3. [4 marks] Prove that NP $\subseteq$ PSPACE by giving a polynomial-space algorithm for simulating an NP machine (use Theorem 7.20 for this, which says a language is in NP iff it is decided by a non-deterministic polynomial time Turing machine).

Theorem 7.20 shows us that a language in NP must be decidable by nondeterministic polynomial time TM. This means that if the time the TM takes is $O(f(n))$, where $f(n)$ takes polynomial time, then there is no way for the TM to use more than polynomial space. If a polynomial time TM spends every step moving left and writing a character, the furthest it could reach would be equal to the amount of time it spent, which itself is polynomial.

So what we can do is we can create a TM S that models a Breadth First Search of the NTM.

$S =$ "On input n
Starting from the first configuration
If the configuration is accepting, accept (2)
Search each configuration as a BFS, erasing the tape and returning to the first cell after each.
accept once an accepting configuration is reached.

---

4. [2 marks] Why is Savitch's theorem surprising, given our study of P versus NP?

The book made a great statement for Example 8.3 "Space appears to be more powerful than time because space can be reused, whereas time cannot."

But what this means is, we can model the hard problems on physical computers, because computers are limited when it comes to memory size, but in essence, could run forever.

But what else this could mean is, there may be more problems. What sort of problems might require exponential space to compute? This implies a whole new set of problems.

---

5. [14 marks] This question studies the proof of Savitch's theorem. You may assume the TM M in the proof can compute function $f(n)$ in the proof in $O(f(n))$ space.

   (a) [2 marks] In words, give a high-level overview of how the proof of Savitch's theorem works.

   The proof shows that a function that runs in NSPACE in polynomial time can be run SPACE in polynomial time as well. To simulate this we begin with a starting configuration c_1 and we want to reach an accepting configuration c_2 within t steps where t is a polynomial time for input n.
   We create a TM which recursively checks whether we can get from a configuration a to some configuration b in t steps by continuously dividing the problem in half. The TM would create a new configuration m which is expected to be between a and b, and then run itself no a to m in $t/2$ time as well as m to b in $t/2$ time. (All in PSPACE of course)

   (b) [6 marks] The CANYIELD procedure in the proof has 6 steps: Describe in a sentence or two the purpose of each step.

   1: If $t = 1$ this means we are simply checking the configuration is itself or the configuration c_1 can reach c_2 in t steps which is 1 step.
   2: If $t > 1$ we must split and try all configurations between c_1 and c_2.
   3: Simply check if the first half can be accepted

4: and that the second half is accepted

5: If 3 and 4 accept, the TM accepts. However, if they don't we try a different configuration c_m, jumping back to 2 until we've tried all configurations using $f(n)$ space.

6: If the loop does not result in an accept, reject.

(c) [2 marks] Why does the machine M in the proof correctly simulate the NTM N?

M correctly simulates NTM N because it is running similarly to the way normal NTM would run. It guesses configurations c_m for example, tries out their guess, then tries a new configuration if the machine does not accept. In general, it is searching for a path down an NTM N's branch.

(d) [4 marks] Why does M use $O(f^2(n))$ space?

Because we are recursively trying different values, we are storing configuration for each time we've recursed. These configurations take up $O(f(n))$ space each. In our loop from steps 2 to 5, if we do not reach an accepting state, we simply overwrite this failed branch, so that we don't end up exponentially growing in size. Our recursions only reach the depth of $logt$ where $t = 2^{o(f(n))}$, the maximum time the NTM should use per branch. Thus, $logt = O(f(n))$ which we multiply with the size of each configuration equals: $O(f^2(n))$ space.